Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Electrónica de Comunicaciones

Trabajo Fin de Grado

Detección de personas en imágenes de profundidad mediante redes neuronales convolucionales

Autor: Roberto Martín López

Tutores: Cristina Losada Gutiérrez y David Fuentes Jiménez

2019

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Electrónica de Comunicaciones

Trabajo Fin de Grado

Detección de personas en imágenes de profundidad mediante redes neuronales convolucionales

Autor: Roberto Martín López

Tutores: Cristina Losada Gutiérrez y David Fuentes Jiménez

Tribunal:

Presidente: Juan Manuel Miguel Jiménez

Vocal 1º: Jesús Ureña Ureña

Vocal 2º: Cristina Losada Gutiérrez

Calificación: .....

Fecha: .....

Agradecimientos

Quiero agradecer este trabajo, en primer lugar, a mis tutores Cristina y David por lo fácil que es trabajar con ellos, por su disponibilidad total y por todo lo que he me han enseñado.

También quiero agradecérselo a toda mi familia, especialmente a mis padres Jesús y Susana y a mi hermana Raquel.

Merece una mención especial Sergio, porque son ya diez años trabajando juntos.

Por último a los compañeros del laboratorio y a todos los profesores que han sabido motivarme y han hecho que me alegre de haber estudiado esta carrera.

Resumen

El objetivo de este trabajo es el diseño, implementación y evaluación de un sistema de detección de personas en imágenes de profundidad, basado en Redes Neuronales Convolucionales. Se ha construido una red neuronal profunda, con una estructura compleja, basada en una configuración *encoder-decoder* y en los bloques residuales de la ResNet. Su entrenamiento se ha dividido en dos partes: en la primera se ha utilizado una base de datos que contiene un gran número de datos sintéticos, generados para esta aplicación y en la segunda se ha llevado a cabo un ajuste con un pequeño conjunto de datos reales, evitando así la necesidad de etiquetar manualmente grandes bases de datos. Tras la evaluación del sistema final, se ha obtenido una tasa de aciertos del 85 %, con una precisión del 100 % que ha permitido validar el sistema desarrollado.

Ante cualquier problema o sugerencia sobre el presente trabajo, por favor contactad con Roberto Martín López <roberto.martin@edu.uah.es>.

Palabras clave: CNNs, imágenes de profundidad, detección.

Abstract

The objective of this work is the design, implementation and evaluation of a people detection system in depth images, based on Convolutional Neural Networks. The built deep network has a complex architecture, based on *encoder-decoder* configuration and using ResNet residual layers. The training process has been divided in two parts. The first part, using a big dataset of synthetic data. The second part, using a small set of real data, avoiding to manually label big datasets. After the evaluation of the final system, the success rate obtained is 85 %, with an accuracy of 100 % that allows to validate the system.

If you have problems, suggestions or comments, please forward them to Roberto Martín López <roberto.martin@edu.uah.es>.

Keywords: CNNs, depth images, detection.

Índice general

Índice de figuras

Índice de tablas

Lista de acrónimos

BCC	*Bloque Convolucional Codificador*.
BCD	*Bloque Convolucional Decodificador*.
BIC	*Bloque Identidad Codificador*.
BID	*Bloque Identidad Decodificador*.
BP	*Bloque Principal*.
BR	*Bloque de Refuerzo de Hipótesis*.
CNN	Red Neuronal Convolucional.
CNNs	Redes Neuronales Convolucionales.
CWM	Modulación por Onda Continua.
FN	Falsos Negativos.
FOV	Campo de Visión.
FP	Falsos Positivos.
fps	*Frames* por segundo.
GEINTRA	Grupo de Ingeniería Electrónica aplicada a Espacios Inteligentes y Transporte.
MAE	*Mean Absolute Error*.
MSE	*Mean Square Error*.
SGD	*Stocastic Gradient Descent*.
TFG	Trabajo Fin de Grado.
ToF	Tiempo de Vuelo.
TP	*True Positives*.
VP	Verdaderos Positivos.

Capítulo 1

Introducción

1.1. Introducción al Trabajo Final de Grado

A lo largo de los últimos años ha cobrado importancia en el ámbito de la investigación el desarrollo de sistemas de detección de personas en imágenes, debido a su aplicación en áreas como la seguridad, la vídeo-vigilancia o el control de aforos. La principal ventaja de estos sistemas es que son no-invasivos, frente a sistemas mecánicos como barreras o tornos, que además no son lo suficientemente efectivos. Generalmente, se han desarrollado sistemas que realizan la detección sobre imágenes de color (RGB), debido a la gran información que aportan, este es el caso del trabajo descrito en [7], donde se propone un sistema que aprende modelos de apariencia de las personas, o [8], cuyo trabajo se basa en clasificación de puntos de interés. Sin embargo, el uso de imágenes de color implica una invasión de la privacidad, pues en las imágenes es posible identificar a los usuarios. Como alternativa, recientemente han surgido sistemas que llevan a cabo la detección sobre otros tipos de imágenes, como las imágenes de profundidad, donde no es posible reconocer la identidad de los usuarios, de forma que no se ve comprometida su privacidad. Ejemplos de trabajos que emplean imágenes de profundidad son [9–12]. Otro aspecto importante de los sistemas de detección es que deben ser robustos, esto implica que deben funcionar correctamente aun cuando existan cambios en el entorno, tales como variaciones en la iluminación, o frente a situaciones complicadas, como pueden ser personas parcialmente ocluidas u objetos que, por forma, color o posición, puedan asemejarse a una persona pero que no deben ser detectados.

Este Trabajo Fin de Grado (TFG) se desarrolla dentro de las líneas de trabajo del grupo de investigación Grupo de Ingeniería Electrónica aplicada a Espacios Inteligentes y Transporte (GEINTRA), que se centran en el desarrollo de sistemas de detección y conteo de personas, utilizando diversas técnicas de aprendizaje máquina para resolver las tareas descritas anteriormente. Es aquí donde se ha enmarcado el presente Trabajo Final de Grado.

Dentro del contexto descrito anteriormente, en este trabajo se ha realizado el diseño, implementación y evaluación (extracción de resultados del funcionamiento) de un sistema de detección de personas a partir de imágenes tomadas por un sensor de profundidad. En dichas imágenes, cada píxel indica la distancia de ese punto de la escena al sensor. Con está información, el sistema es capaz de llevar a cabo la detección de las personas que existan en la escena, pero no es posible conocer la identidad de éstas, debido a que las imágenes de profundidad solo permiten reconocer rasgos físicos muy generales, como altura o complexión. Otra ventaja de la utilización de información de profundidad, y que soluciona uno de los problemas expuestos al comienzo, es su poca variación frente a cambios en la iluminación, aumentándose la robustez del sistema en ese aspecto. Respecto a la ubicación de la cámara y a la perspectiva de la escena,

se ha elegido la orientación frontal elevada. Una perspectiva frontal permite utilizar un área de estudio mucho mayor, pero es frecuente la aparición en las imágenes de personas parcial o totalmente ocluidas. Este problema apenas existe si se opta por una perspectiva cenital, como por ejemplo en [13], pero el área de estudio se vería reducida. A modo de ejemplo, supongamos que se quieren tomar imágenes de una sala de aproximadamente 6×6 metros en la que caminan personas, para realizar la detección de éstas. Utilizando una posición cenital del sensor (y suponiendo un comportamiento normal de los usuarios) nos aseguramos visualizar completamente las cabezas y los hombros de cada persona, siendo muy difícil que existan oclusiones entre éstas. Sin embargo, es muy complicado que utilizando una posición cenital se pueda visualizar, en una sola imagen, una escena completa de tales dimensiones. Si se opta por una posición frontal, las dimensiones de la escena no suponen un problema y el área de estudio solamente se ve restringida por las características del sensor. Por el contrario, cuando varios usuarios caminan frente a la cámara, aparecen en las imágenes oclusiones parciales o totales. Eligiendo una posición frontal elevada del sensor, aseguramos poder visualizar una escena de grandes dimensiones, pero reduciendo el número de oclusiones frente a una posición completamente frontal.

En la Figura 1.1 se muestra un ejemplo de imagen de profundidad, donde puede observarse cómo no es posible reconocer la identidad de las personas. Sí es posible hacerlo, por el contrario, en la correspondiente imagen de color. En la Figura puede apreciarse también lo que se ha denominado perspectiva frontal elevada. Para representar la imagen de profundidad se ha utilizado una escala de colores donde cada color corresponde a un valor de distancia a la cámara. Esta correspondencia se indica en la barra de color, junto a la imagen, donde las unidades de distancia utilizadas son milímetros.

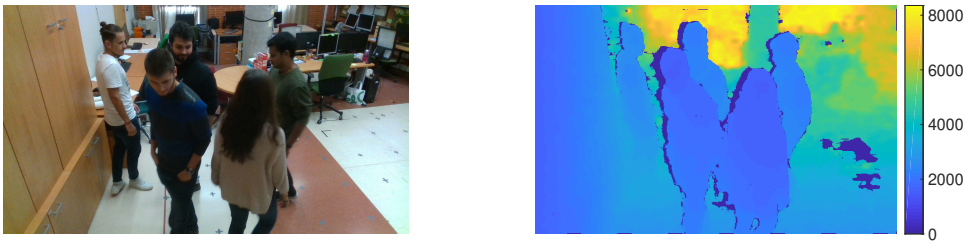


Figura 1.1: Ejemplo de imágenes obtenidas por un sensor RGB-D. A la izquierda la imagen RGB y a la derecha la correspondiente imagen de profundidad, donde los valores indicados en la barra de color corresponden a la distancia en milímetros.

En lo relativo propiamente al sistema de detección, frente a la algoritmia clásica, en este trabajo se ha optado por la utilización de redes neuronales profundas, concretamente redes neuronales convolucionales. Éstas han demostrado su eficacia en el procesado de imágenes, tanto RGB como de profundidad, y llevando a cabo tareas enfocadas a extracción de características o detección. En los últimos años han aparecido numerosos trabajos que emplean *deep learning* en tareas de detección, como por ejemplo [14–16], que utilizan tanto imágenes RGB como RGB-D. También cabe destacar los trabajos [17, 18], que al igual que este TFG, además de *deep-learning* emplean únicamente información de profundidad.

1.2. Introducción al sistema

Como se ha comentado previamente, el sistema desarrollado en este TFG realiza la detección robusta de personas en imágenes de profundidad con ubicación frontal elevada de la cámara. Para ello utiliza una red neuronal profunda, basada en capas convolucionales y siguiendo una arquitectura descrita detalladamente en los capítulos 2 y 3. El objetivo es que el sistema funcione en tiempo real, es decir, realice la

detección a una velocidad igual o superior a las 20 imágenes por segundo (20 *Frames* por segundo (fps)), velocidad suficiente para considerar una ejecución en tiempo real de la tarea de detección.

En un primer lugar, entendiendo el sistema como una *caja negra*, se recibe a la entrada una imagen de profundidad, y se obtiene a la salida el valor de las coordenadas en el plano imagen sobre las que se ubican los centroides de las cabezas de tantas personas como existan en el área de estudio (zona de la imagen sobre la que se desea llevar a cabo la detección). Si no existen personas en la escena el sistema no debe generar detección alguna. La robustez del sistema se mide en base a dos parámetros: primeramente, deben detectarse todas y cada una de las personas que existan en el área de estudio, es decir, no debe haber personas sin detectar y, por otro lado, no deben detectarse personas que no existen en el área de estudio, es decir, no deben generarse detecciones erróneas.

En segundo lugar, en lo relativo a la estructura interna del sistema de detección, se cuenta con distintas etapas:

- **Procesado de la imagen de entrada.** La imagen de entrada debe escalarse y normalizarse. La siguiente etapa está diseñada para recibir imágenes normalizadas y de unas dimensiones determinadas, por ello, esta etapa de procesado es necesaria para adaptar la imagen de entrada a la red neuronal. Además, puede incluirse opcionalmente otro procesado de la imagen de entrada enfocado a la reducción del frecuente ruido de bordes existente en las imágenes de profundidad, como puede observarse en la Figura 1.2. La parte del procesado relativa a la reducción de ruido varía en función de las características de las imágenes e idealmente no debería existir, pues la siguiente etapa debe ser capaz de actuar correctamente sobre imágenes ruidosas. Se detallarán estos aspectos en el Capítulo 3, concretamente al hablar del entrenamiento del sistema.

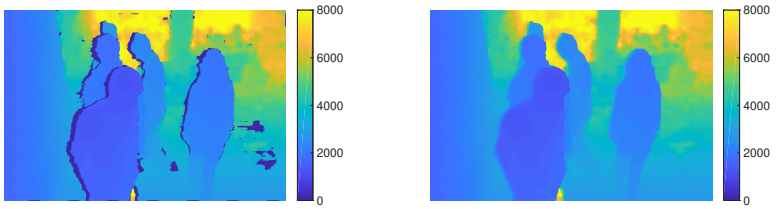


Figura 1.2: Ejemplo de imagen de profundidad antes y después de la etapa de procesado. A la izquierda la imagen tal y como la entrega la cámara, donde puede observarse el ruido de bordes (pixeles nulos), y a la derecha la imagen procesada, donde se ha eliminado el ruido de bordes.

- **Red Neuronal Convolucional**. Esta etapa implementa propiamente la detección sobre la imagen de profundidad. Internamente está formada por dos bloques claramente diferenciados, denominados **Bloque Principal** y **Bloque de Refuerzo de Hipótesis**. Cada uno de ellos constituye una arquitectura encoder-decoder basada en bloques residuales (ResNet [4]), a su vez basados en capas convolucionales. Ambos bloques entregan a su salida un mapa de probabilidad en el que cada detección se indica mediante una función gaussiana cuya media se ubica sobre las coordenadas en dos dimensiones del centroide de la cabeza de la persona detectada.

  - **Bloque Principal.** Recibe a su entrada la imagen de profundidad y entrega en su salida un mapa de probabilidad con las detecciones efectuadas. Dicho mapa de probabilidad puede contener, en ocasiones, detecciones en las que la función gaussiana no está claramente definida, o aparecen patrones que no responden propiamente a una función gaussiana. En general podría decirse que el mapa de probabilidad entregado por el Bloque Principal es, en algunas ocasiones, más ruidoso de lo deseado. De aquí surge la necesidad de construir el Bloque de Refuerzo de Hipótesis.

- **Bloque de Refuerzo de Hipótesis.** Su arquitectura es una versión ligeramente simplificada del Bloque Principal. Recibe en su entrada la salida del Bloque Principal concatenada con la imagen de profundidad. En el mapa de probabilidad entregado en su salida las gaussianas correspondientes a detecciones correctas están definidas de forma mucho más clara y desaparecen o se reducen aquellos indicios de detecciones erróneas.

La salida de la etapa de detección corresponde a la salida del bloque interno de Refuerzo de Hipótesis.

- **Procesado de la salida.** Del mapa de probabilidad entregado por la etapa anterior deben extraerse el número de personas detectadas y sus ubicaciones en la imagen (coordenadas del centroide de las cabezas). Para ello, en primer lugar, se convierte el mapa de probabilidad en una imagen binaria, que separa la zona en torno a la media de cada gaussiana del fondo, y en segundo lugar, se obtienen las coordenadas de los centroides de todos los cuerpos existentes en la imagen binaria. De forma general, la etapa de procesado obtiene las coordenadas del punto correspondiente a la media de cada gaussiana.

La Figura 1.3 muestra, reflejadas en un diagrama global, todas las etapas anteriormente descritas. La entrada corresponde a la imagen entregada por el sensor de profundidad y la salida son los pares de coordenadas en el plano imagen de todas las personas detectadas. Se muestran además ejemplos de la imagen y de los mapas de confianza en puntos internos del sistema. En el punto A se puede observar la imagen de entrada tras aplicar la reducción de ruido. En el punto B se tiene el mapa de confianza generado por el Bloque Principal de la Red Neuronal Convolucional, donde pueden observarse detecciones poco claras o erróneas. El punto C corresponde al mapa de confianza entregado por el Bloque de Refuerzo de Hipótesis, donde las detecciones se indican con una función gaussiana claramente definida.

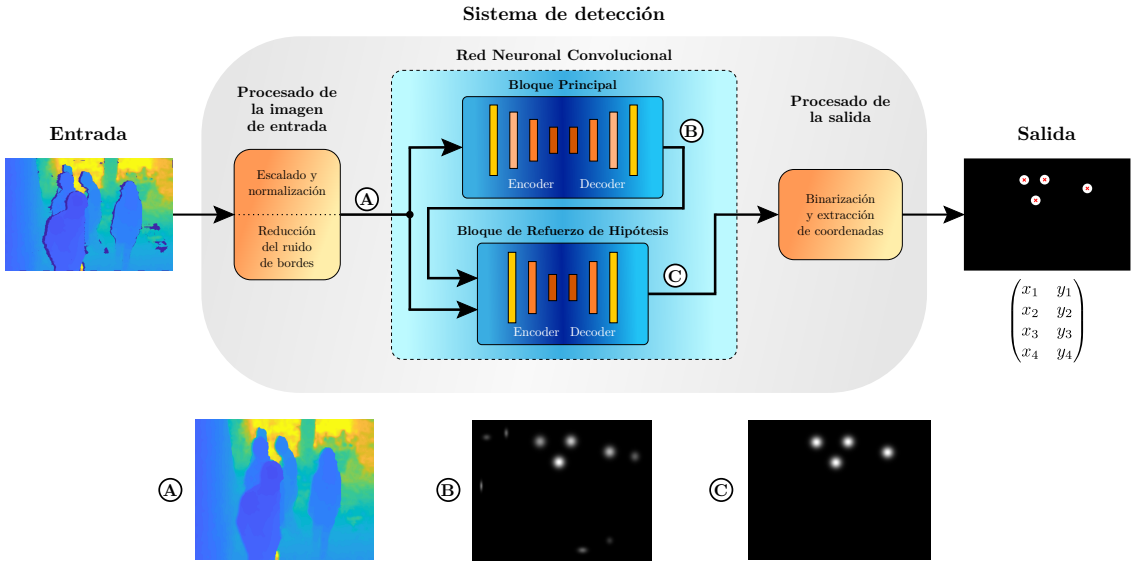


Figura 1.3: Diagrama del Sistema de Detección completo. Se muestra un ejemplo de entrada, las diferentes capas internas y la salida, además del aspecto de la imagen de profundidad y de los mapas de probabilidad en puntos internos del sistema.

1.3. Estructura del Documento

El presente documento se estructura de la forma descrita a continuación. El Capítulo 1 realiza una introducción del problema a resolver y del sistema desarrollado. El Capítulo 2 desarrolla una explicación de todos los aspectos teóricos involucrados en el diseño del sistema. El Capítulo 3 explica todos los pasos seguidos en el desarrollo del sistema hasta su implementación final. En el Capítulo 4 se detalla cómo se ha realizado la validación del sistema y la extracción de resultados, y se muestran detalladamente cuáles han sido éstos. Por último, en el Capítulo 5 se indican las principales conclusiones obtenidas tras el desarrollo del trabajo, así como algunas posibles líneas de trabajo futuras.

Capítulo 2

Estudio teórico

2.1. Introducción

En este Capítulo se realiza un estudio de los fundamentos teóricos necesarios para el diseño del sistema desarrollado. Es importante conocer todos los aspectos que aquí se tratan, de cara a construir un sistema que siga una estructura lógica, basada en la teoría, para llevar a cabo el objetivo planteado en este TFG.

En primer lugar, se dedica un apartado a las imágenes de profundidad, donde se describe su utilidad y sus principales características. También se incluye una descripción de las características y del funcionamiento de los sensores de profundidad empleados en el desarrollo de este TFG. El resto de apartados del capítulo se centran en los principales aspectos teóricos relativos a redes neuronales que intervienen en el sistema de detección. Primeramente, se resumen las ventajas y aportaciones del *Deep Learning* en tareas de detección sobre imágenes. A continuación, se describen todas las capas, elementos, arquitecturas y configuraciones utilizados en el diseño del sistema, además de otros aspectos importantes como optimizadores y funciones de coste.

2.2. Imágenes y sensores de profundidad

El sistema desarrollado en este TFG, tiene como objetivo la detección de personas en imágenes, por ello, el primer paso es decidir las características de dichas imágenes. Como se ha especificado en el Capítulo 1, se desea un sistema robusto y fiable, pero en el que además se preserve la privacidad de las personas y no exista una invasión de su intimidad. En el mercado, podemos encontrar diferentes tipos de cámaras, cada uno de los cuales ofrece imágenes con unas características determinadas. De forma general, podemos hablar de cuatro tipos diferentes de imágenes: en escala de grises, RGB, RGB-D y de profundidad.

- Las **imágenes en escala de grises** se componen de un solo canal, en el que se codifica el nivel de gris de la escena, de forma que no aportan información de color pero sí permiten identificar a las personas que aparecen en la imagen.
- Las **imágenes RGB** aportan información de color, codificada mediante la utilización de tres canales (rojo, verde y azul). Proporcionan gran cantidad de información para llevar a cabo tareas de detección, sin embargo, en ellas es posible identificar a las personas que aparecen.
- Las **imágenes RGB-D** se componen de los tres canales propios de las imágenes RGB más un canal que aporta la información de profundidad. De esta forma, las imágenes RGB-D son las que

contienen mayor información de las que aquí se mencionan, aunque al igual que en las RGB, permiten reconocer la identidad de las personas grabadas.

- Las **imágenes de profundidad** están formadas por un solo canal que codifica valores de distancia. Al no aportar información de color hacen prácticamente imposible identificar a las personas que aparecen en la escena. Sin embargo, la información que contienen es suficiente para llevar a cabo tareas de detección.

Frente a tipos de imágenes enumerados anteriormente, solo las imágenes de profundidad cumplen con el requisito de no permitir reconocer la identidad de las personas existentes en la imagen. Por ello, han sido elegidas para ser utilizadas en este trabajo.

En una imagen de profundidad, cada píxel contiene información relativa a la distancia de ese punto de la escena al sensor. Dependiendo de la tecnología utilizada se pueden clasificar los sensores de profundidad en tres tipos: luz estructurada, estéreo y tiempo de vuelo. En este documento se describen los sensores estéreo y de tiempo de vuelo, por ser los empleados en el desarrollo del TFG.

2.2.1. Cámaras estéreo

El principio de funcionamiento de las cámaras estéreo se basa en un procedimiento, denominado estereopsis [19], que consiste en estimar la profundidad o distancia de cada punto de la escena a partir del análisis de dos imágenes tomadas por dos sensores diferentes, separados entre sí una distancia conocida. Este proceso se basa en cómo el cerebro humano es capaz de percibir la profundidad de lo que se observa, gracias al análisis de las diferencias entre las imágenes proyectadas en la retina de cada ojo. De forma general, puede definirse como la obtención de información 3D a partir de dos imágenes 2D diferentes de la misma escena y de un conjunto de parámetros conocidos.

El problema del cálculo de la profundidad se subdivide en dos problemas que deben resolverse secuencialmente, correspondencia entre puntos y triangulación:

- **Correspondencia entre puntos.** Se trata de un problema de visión artificial que consiste en llevar a cabo una asociación entre los puntos o píxeles de las dos imágenes. Entendiendo que los dos sensores que incorpora una cámara estéreo se encuentran sobre el mismo eje horizontal, se denomina a las imágenes tomadas por cada uno de ellos imagen izquierda e imagen derecha. Para cada píxel de la imagen izquierda debe encontrarse el píxel de la imagen derecha, sobre el mismo eje horizontal, que representa el mismo punto de la escena. Quedan sin asociar aquellos puntos de los márgenes que no aparecen en la otra imagen.

  La Figura 2.1 muestra un ejemplo de correspondencia entre un píxel de la imagen izquierda y un píxel de la imagen derecha que representan el mismo punto de la escena (P). Loas medidas x_l y x_r indican el desplazamiento con respecto al eje central de la imagen, y su valor será necesario para calcular el valor de profundidad.

- **Triangulación.** Procedimiento por el cual, a partir de la correspondencia entre puntos realizada y conociendo la distancia entre los dos sensores (b) y la distancia focal de las lentes de ambos (f), se calcula el valor de profundidad de cada punto de la escena. El cálculo se basa en un problema de semejanza de triángulos. Como muestra la Figura 2.2, se tienen dos triángulos semejantes, uno formado por el punto de la escena y los píxeles que lo representan en cada imagen (P, p_l, p_r) y otro formado por el punto de la escena y los focos de las lentes (P, O_l, O_r). Por el principio de

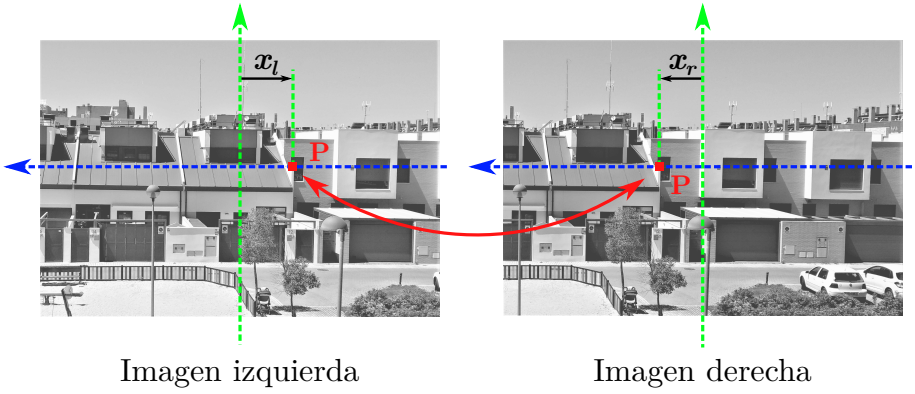


Figura 2.1: Correspondencia entre puntos

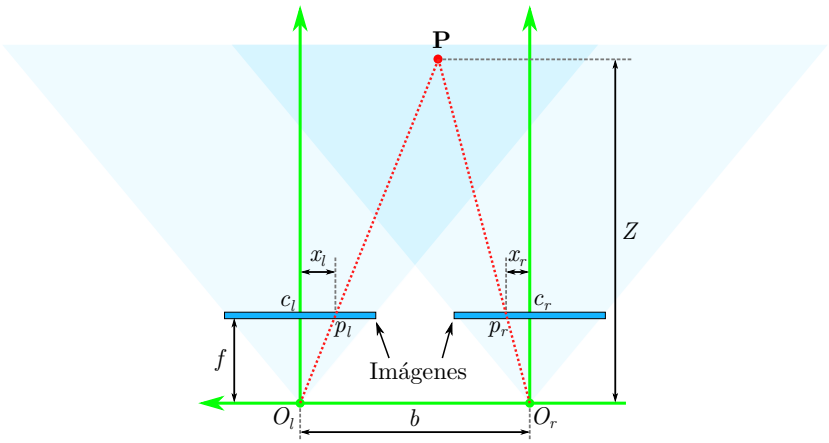


Figura 2.2: Esquema del cálculo del valor de profundidad en cámaras estéreo.

semejanza, la relación entre base y altura de ambos triángulos debe coincidir, de forma que puede despejarse el valor de profundidad (Z) tal y como indican las Ecuaciones 2.1 y 2.2.

$$\frac{b}{Z} = \frac{b + x_l - x_r}{Z - f} \tag{2.1}$$

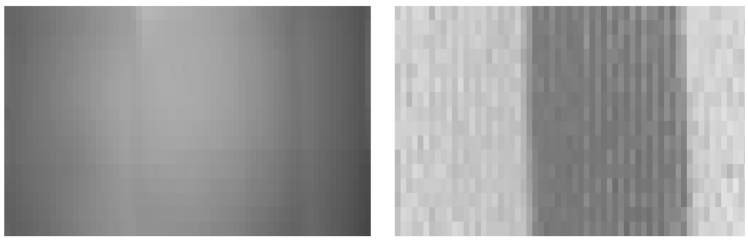
$$Z = f\ \frac{b}{||x_l - x_r||} \tag{2.2}$$

El elemento $||x_r - x_l||$ que aparece en el denominador de la Ecuación 2.2 se denomina disparidad e indica la diferencia en la ubicación de los píxeles que representan un mismo punto P entre las imágenes izquierda y derecha. Valores altos de disparidad corresponden a puntos próximos a la cámara, mientras que valores bajos corresponden a puntos alejados de la cámara.

Una característica importante de las cámaras estéreo es el Campo de Visión (FOV) resultante. Cada sensor tiene un FOV diferente y el FOV resultante es el común a ambos sensores, ya que es el único sobre el que se puede llevar a cabo el proceso de correspondencia entre puntos. En la Figura 2.2 se indican en sombreado azul claro los FOV de ambos sensores y en sombreado más oscuro el FOV resultante. Por otro lado, el error en la medida de profundidad es inversamente proporcional a b, de forma que aumentando la distancia entre los sensores se lograría una mejor estimación de Z. Esto tiene varias desventajas, la más evidente es que aumentando b se reduce el FOV resultante, y además, puede darse la circunstancia de

que algunos puntos de la escena, aun estando dentro del FOV resultante, solo sean vistos por un sensor, complicándose o imposibilitándose el cálculo de Z.

Uno de los principales problemas de la obtención de información de profundidad mediante cámaras estéreo son los fallos al hallar la correspondencia entre puntos, especialmente cuando existe carencia de texturas en las superficies de la escena, lo que dificulta relacionar correctamente los puntos. En estos casos pueden asociarse píxeles que no representan el mismo punto y estimarse valores de profundidad incorrectos. La Figura 2.3 muestra un ejemplo de imagen con unos niveles de gris demasiado homogéneos a pesar de tratarse de diferentes objetos y a distintas distancias, caso en el que una cámara estéreo podría funcionar incorrectamente por fallos de correspondencia entre puntos.



(a) (b)

Figura 2.3: (a)Imagen con unos niveles de gris muy homogéneos. (b)Imagen de profundidad correcta obtenida mediante un sensor TOF. Figura obtenida de [1].

Como ejemplo de cámara que utiliza la tecnología estéreo para obtener información de profundidad puede mencionarse la Realsense D435 de Intel, que se muestra en la Figura 2.4. Se trata de una cámara RGB-D-IR alimentada vía USB. Los dos sensores utilizados para realizar la estereopsis son idénticos y sus parámetros se resumen a continuación [20]:

- **Distancia entre los sensores (b)** → 50 mm
- **Campo de Visión (FOV)** → H: 91.2º / V: 65.5º / D: 100.6º
- **Sensor** → OV9282
- **Resolución y Relación de Aspecto** → 1280 × 800 píxeles (8:5)
- **Formato** → 10-bit RAW
- **Distancia Focal** → 1.93 mm

Es importante precisar que esta cámara no es un sensor estéreo clásico, pues tiene la opción de emitir un patrón aleatorio que haga aumentar la textura de la escena y facilite el proceso de correspondencia entre puntos, por lo que se trata de un sensor de luz estructurada. Sin embargo, su funcionamiento al margen del patrón de luz emitido se ajusta al propio de los sensores estéreo, explicado anteriormente.

2.2.2. Cámaras de tiempo de vuelo

Las cámaras Tiempo de Vuelo (ToF) funcionan enviando haces de luz infrarroja. Cuando un haz encuentra un obstáculo en su recorrido, se produce un rebote y el haz retorna hasta la cámara incidiendo en el array de celdas receptoras. El tiempo que tarda en retornar el haz reflejado, denominado *tiempo de vuelo*, permite hallar (conociendo la velocidad de la luz) el valor de la distancia a la que se ha producido

Figura 2.4: Cámara Realsense D435 de Intel [2].

el rebote, es decir, la distancia a la que se encuentra el obstáculo. Sin embargo, una medida directa del tiempo de vuelo es inviable, ya que requeriría medir con precisión tiempos del orden de decenas de picosegundos o nanosegundos. Por ello la técnica más comúnmente utilizada consiste en una medida indirecta del tiempo de vuelo a partir del desfase entre señal emitida y reflejada. Idealmente, cada celda receptora del sensor recibe un haz reflejado distinto, provocado por un rebote diferente y a distintas distancias, construyéndose así la imagen de profundidad de la escena completa. La Figura 2.5 muestra el esquema básico del principio sobre el que se basan los sensores ToF, en el que el haz emitido está modulado por una señal cuadrada. Como se observa, la señal que más tiempo tarda en retornar al sensor (mayor tiempo de vuelo) corresponde a la señal con mayor desfase con la señal emitida y proviene de una reflexión en un objeto a una distancia mayor que otra señal cuyo desfase es menor.

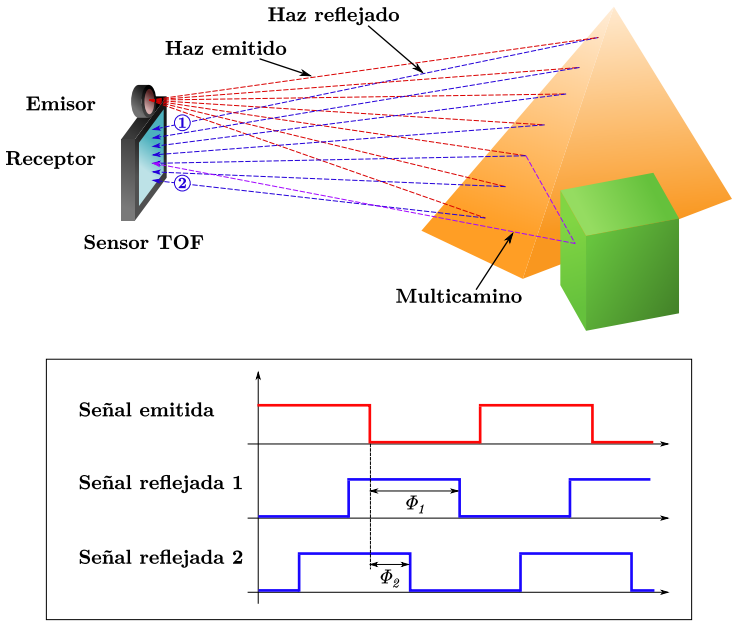


Figura 2.5: Esquema del principio de funcionamiento en el que se basa la tecnología ToF

El haz enviado por el emisor suele ser de entorno a 850 nm de longitud de onda, cercano al infrarrojo, lo que lo hace imperceptible para el ojo humano y, del mismo modo, prácticamente inmune a la iluminación del entorno donde se ubique. Existen dos técnicas de emisión de dicho haz infrarrojo, luz pulsada y Modulación por Onda Continua (CWM), y en cada una de ellas el cálculo de la distancia se realiza de manera distinta. Sin embargo, en ambos casos se utiliza para dicho cálculo la carga acumulada en unos transistores basados en tecnología CMOS y ubicados en el receptor.

En el caso de luz pulsada, el emisor envía un haz de luz infrarroja durante un período de tiempo Δt. En el receptor se sitúan dos transistores CMOS que acumulan carga cuando incide el haz reflejado en dos ventanas temporales de duración Δt, la primera de ellas en fase con la señal emitida y la segunda en contrafase. La Figura 2.6 muestra un ejemplo del funcionamiento del sistema de luz pulsada, donde

puede observarse cómo se produce la acumulación de carga en los transistores. A partir de los valores de carga acumulada Q_1 y Q_2, el cálculo del valor de la distancia se realiza como indica la Ecuación 2.3, donde c representa la velocidad de una onda electromagnética en el vacío (3×10^8 m/s) y el factor $\frac{1}{2}$ se explica entendiendo que el desfase entre señal emitida y reflejada corresponde a una distancia recorrida de ida y vuelta, siendo la distancia del objeto la mitad de este valor.

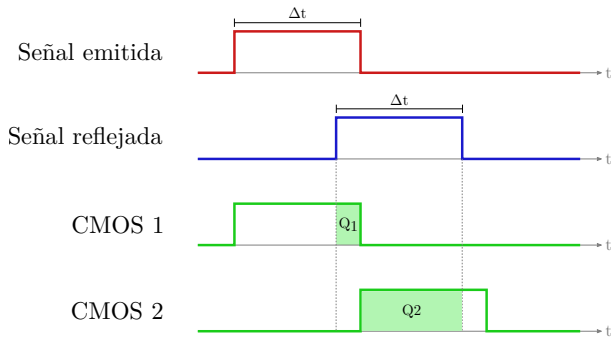


Figura 2.6: Principio de funcionamiento del sistema de luz pulsada

$$d = \frac{1}{2}\, c\, \Delta t \left(\frac{Q_2}{Q_1 + Q_2} \right) \tag{2.3}$$

En el caso de CWM el haz emitido se modula con una señal continua, que puede ser sinusoidal o cuadrada y de frecuencia f_{mod} que suele situarse en unas decenas de MHz. El primer paso es calcular el desfase existente entre el haz emitido y el reflejado, para lo cual son necesarios ahora cuatro transistores CMOS que acumulan carga cuando incide la señal recibida durante cuatro ventanas temporales desfasadas 90º entre sí, tal y como muestra la Figura 2.7. El cálculo del valor del desfase (ϕ) a partir de Q_1, Q_2, Q_3 y Q_4, se realiza según indica la ecuación 2.4. Además, para el cálculo del desfase se utiliza la carga acumulada durante un tiempo que incluye varios períodos de la señal emitida y que se denomina Período o Tiempo de Integración.

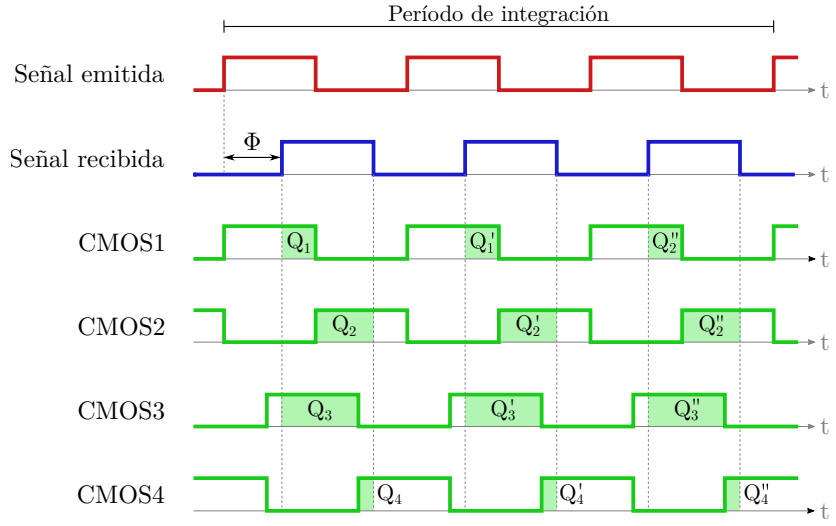


Figura 2.7: Principio de funcionamiento del sistema Modulación por Onda Continua

$$\phi = arctan\left(\frac{Q_3 - Q_4}{Q_1 - Q_1}\right) \tag{2.4}$$

A partir del valor del desfase puede obtenerse la distancia tal y como indica la Ecuación 2.5.

$$d = \frac{c}{2 \times 2\pi f_{mod}}\phi \tag{2.5}$$

Como la función *arco-tangente* es una función periódica de período 2π, es fácil deducir que cualquier rebote que dé lugar a un desfase $\phi + 2n\pi$ para $n > 0$, será interpretado por el receptor como un desfase de valor ϕ. Por lo tanto, el valor del desfase debe encontrarse en el rango $[0, 2\pi)$ para que sea correctamente medido. De esta forma, se deduce que la distancia máxima válida que puede ser capaz de medir un sensor ToF se calcula a partir de la Ecuación 2.6, resultado de sustituir ϕ por 2π en la Ecuación 2.5. Los valores de profundidad de cualquier objeto situado a una distancia de la cámara superior a la distancia máxima ($d_{máx}$), no serán correctamente medidos. La Figura 2.8 ilustra el problema de la distancia máxima en los sensores ToF.

$$d_{máx} = \frac{c}{2f_{mod}} \tag{2.6}$$

El valor de la distancia máxima puede aumentarse de forma sencilla disminuyendo la frecuencia de modulación, pero esto se traduce en una pérdida de precisión, pues la varianza de las medidas de distancia es inversamente proporcional a la frecuencia de modulación [21]. De cara a aumentar el valor de distancia máxima válida sin disminuir la frecuencia de modulación, han surgido los sistemas multifrecuencia.

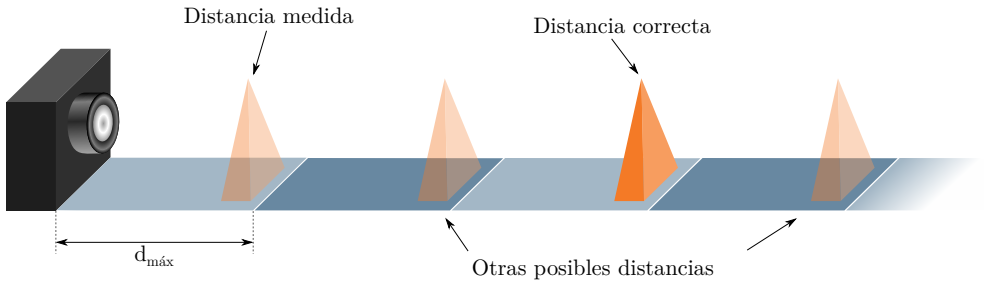


Figura 2.8: Ambigüedad en la medida de objetos a una distancia superior a $d_{máx}$

En los sistemas ToF multifrecuencia el emisor envía dos haces infrarrojos modulados con dos frecuencias distintas, $f_{mod1} > f_{mod2}$. De esta forma, objetos situados a una distancia que puede ser superior a $d_{máx1}$ y $d_{máx2}$, dan lugar a dos desfases distintos, ϕ_1 y ϕ_2, dentro del rango $[0, 2\pi)$. Sin embargo, los valores de distancia obtenidos a partir de estos desfases no serán iguales entre sí (a excepción de que la distancia real del objeto sea menor que $d_{máx1}$ y $d_{máx2}$). Por tanto, para encontrar el valor de distancia correcto deben buscarse los valores de desfase, $(\phi_1 + 2n_1\pi)$ y $(\phi_2 + 2n_2\pi)$, que dan lugar a un mismo valor de distancia. Este concepto se refleja en la ecuación 2.7, donde n_1 y n_2 son dos valores enteros en el rango $[0, \infty)$ que hacen coincidir el valor de distancia medido para las dos frecuencias de modulación. La Figura 2.9 ilustra dicho concepto, donde $n_1 = n_2 = 3$.

$$d_{correcta} = \frac{c}{2 \times 2\pi f_{mod1}}(\phi_1 + 2n_1\pi) = \frac{c}{2 \times 2\pi f_{mod2}}(\phi_2 + 2n_2\pi) \tag{2.7}$$

El sistema multifrecuencia consigue extender el rango de medidas válidas, pues, aunque existe un nuevo valor de distancia máxima, éste es mucho mayor que el propio de un sistema monofrecuencia.

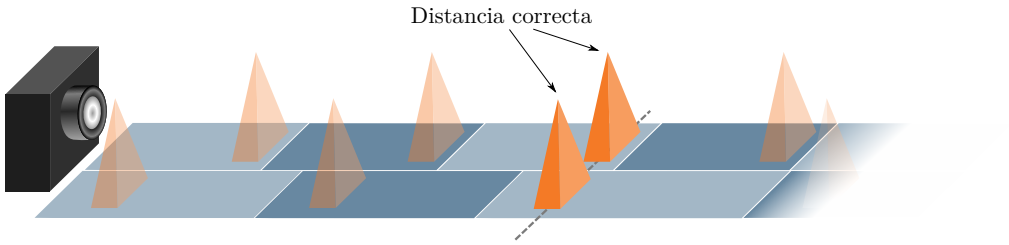


Figura 2.9: Principio de funcionamiento de los sistemas TOF multifrecuencia

En lo relativo a una comparativa entre el método de luz pulsada y CWM, aunque el primero es más simple, permitiendo hallar el valor de distancia a partir de la carga almacenada únicamente en dos transistores frente a los cuatro necesarios en CWM, éste, por la forma de cálculo del desfase, tiene menores errores de *offset* y de ganancia en la medida.

Todos los aspectos anteriormente resumidos y relativos al proceso de obtención de los valores de profundidad en las cámaras ToF se detallan en [22], [21], [23] y [24].

Respecto a los principales problemas generales de los sensores ToF están las interferencias debidas al multicamino, que consiste en que una misma celda receptora recibe más de un haz reflejado, fruto de múltiples rebotes, lo que produce una distorsión en la medida de la distancia. En [25] se proponen técnicas para poner solución a estos errores. Otro problema de la tecnología ToF es la interferencia de otras fuentes de radiación intensa y de longitud de onda similar a la utilizada por el sensor, como la luz solar, lo que dificulta el uso de estas cámaras en exteriores. Por último, es importante mencionar el problema de los denominados artefactos de movimiento (*Motion Artifacts*). Como se ha explicado anteriormente y se muestra en la Figura 2.7, cada medida de distancia se obtiene a partir de las cargas acumuladas durante un tiempo de integración. Si durante ese tiempo se mueven los objetos presentes en la escena, la medida de su distancia se ve fuertemente distorsionada. En [26] se proponen técnicas que ponen solución al problema de los artefactos de movimiento.

A pesar de la existencia de problemas y errores en la tecnología ToF, una de sus principales ventajas frente a la tecnología estéreo, es que no es necesario, en primer lugar, el empleo de dos cámaras, y en segundo lugar, llevar a cabo un proceso de correspondencia entre puntos condicionado en gran medida por las texturas de la escena.

Una cámara que basa su funcionamiento en la tecnología ToF es la Kinect II [27], desarrollada por Microsoft para ser usada en la Xbox One. Esta cámara entrega tanto imágenes de profundidad como RGB, lo cual, unido a la calidad de las imágenes y a un precio bastante competitivo, ha hecho que sea utilizada en numerosos trabajos y en la grabación de diversas bases de datos, como la introducida en [6]. La Figura 2.10 muestra el aspecto de la cámara Kinect II.

Figura 2.10: Cámara Kinect II [3]

Algunas de las principales características de la Kinect II son:

- Un ángulo de visión de 60º en vertical y 70º en horizontal
- Rango de medidas válidas de 0.5 a 4.5 metros
- Imágenes de profundidad de 512 píxeles en horizontal y 424 píxeles en vertical
- Resolución de profundidad dentro del 1 % de la distancia medida
- Error en la medida menor del 2 %

Es importante destacar que la Kinect II, al igual que otras cámaras ToF, sufre el efecto de diferentes fuentes de error propias de dicha tecnología. Estos errores se clasifican en sistemáticos y no sistemáticos, y de forma general, su efecto se traduce en un valor de profundidad erróneo.

- **Errores sistemáticos:** se producen en la conversión del haz de luz reflejado en una señal eléctrica de características adecuadas para el cálculo de la distancia. Su efecto suele intentar minimizarse mediante soluciones *hardware*. Los más comunes son el error Wiggling, que produce alteraciones en la medida debido a pequeñas vibraciones de la cámara, errores producidos por ruido de patrón fijo en píxeles concretos, ruido de disparo (*Shotnoise*), propio de los transistores MOSFET utilizados para el cálculo del desfase, errores relacionados con la temperatura del sensor, etc.

- **Errores no sistemáticos:** errores no predecibles en la medida. El principal es la aparición de *Flying Pixels*, píxeles que corresponden a una discontinuidad brusca en la escena 3D (por ejemplo entre un objeto y el fondo) y que generan una medida distorsionada de la distancia. También se consideran errores no sistemáticos las medidas realizadas fuera del rango válido, mediciones distorsionadas producidas por el problema de los artefactos de movimiento así como las interferencias debidas al multicamino o a otras fuentes de radiación, como la luz solar.

2.3. Capas y elementos más comunes de las Redes Neuronales Convolucionales

2.3.1. Introducción a las Redes Neuronales

Las Redes Neuronales artificiales, son estructuras computacionales inspiradas en el funcionamiento del sistema neuronal biólogico. En su estructura clásica, constan de mayor o menor cantidad de neuronas (en función de su complejidad), interconectadas entre sí y organizadas en capas. Cada una de estas neuronas, denominadas neuronas clásicas, recibe una serie de entradas ($\vec{x}$) y entrega un valor de salida (y), resultado de realizar una combinación lineal de las entradas utilizando un conjunto de parámetros internos denominados pesos ($\vec{w}$). Además, a su salida se aplica una operación, generalmente no lineal, denominada Función de Activación. De esta forma, el comportamiento de una neurona puede expresarse como indica la Ecuación 2.8, donde $\mathcal{F}$ representa la Función de Activación. La Figura 2.11 muestra un esquema de una posible configuración de red neuronal clásica, con tres entradas y dos salidas.

$$y = \mathcal{F}\{f(\vec{x}, \vec{w})\} \tag{2.8}$$

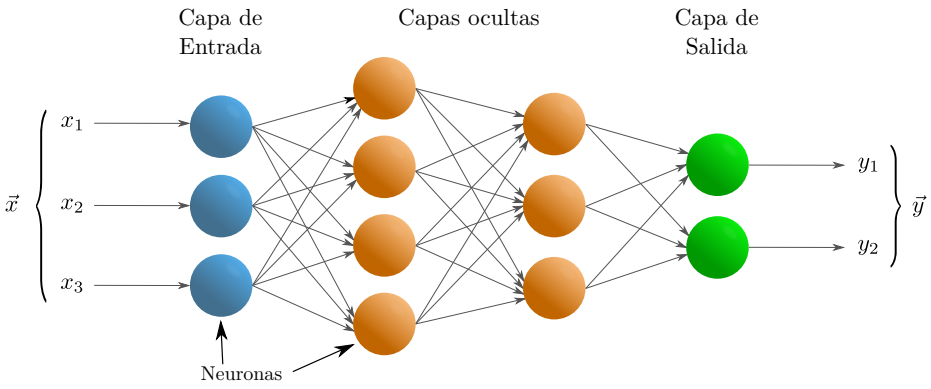


Figura 2.11: Esquema de una red neuronal clásica

La forma de definir el comportamiento de cada neurona, y por tanto el comportamiento de una red completa, consiste en ajustar el valor de sus pesos, lo cual se lleva a cabo de forma automática mediante un proceso de aprendizaje denominado entrenamiento. Éste aprendizaje puede ser supervisado, si se utilizan datos etiquetados a priori, o no supervisado, si se utilizan datos sin etiquetar, de forma que el sistema realiza agrupaciones no sujetas a un criterio establecido previamente. En [28] puede encontrarse una explicación detallada de diferentes métodos de aprendizaje no supervisado. En el caso del aprendizaje supervisado, durante el entrenamiento se entrega a la red pares entrada-salida $(\vec{x}, \widehat{y})$, formados por una posible entrada del sistema y la salida que éste debería generar para esa entrada. Los sucesivos pasos que se llevan a cabo en cada iteración del entrenamiento se enumeran a continuación:

- Se ejecuta la red para la entrada $\vec{x}$, obteniendo una salida $\vec{y}$.
- Mediante una función, denominada de coste o de pérdidas, se compara el resultado obtenido $(\vec{y})$ con el resultado correcto $(\widehat{y})$, obteniendo el error (e) cometido (dependiendo del tipo de función de coste el error se expresa de distinta forma).
- Mediante una técnica denominada *Back Propagation* se calcula el gradiente del error respecto a cada peso de la red.
- Se actualiza el valor de cada peso para que se minimice el error en la próxima iteración. Dicha actualización depende del error cometido, del valor actual de cada peso y de un parámetro denominado tasa de aprendizaje, en inglés *learning rate*.

Los diferentes pasos enumerados anteriormente son una generalización del proceso de entrenamiento, que en la práctica incluyen técnicas más elaboradas, como los optimizadores, para asegurar un mejor progreso del aprendizaje. Un de los puntos clave en este aprendizaje es no llegar al sobre-entrenamiento (*overfitting*), que consiste en que la red no aprende a realizar la tarea deseada, sino que aprende a asociar las entradas y salidas vistas durante el entrenamiento, pero funciona incorrectamente para otras entradas que no han sido procesadas.

Hasta aquí, se han descrito las redes neuronales clásicas, denominadas también Densas. A continuación se describen las Redes Neuronales Convolucionales, estructuras más complejas, enfocadas al procesamiento de imágenes y que se basan en las redes neuronales clásicas.

2.3.2. Introducción a las Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales (CNNs) son redes de neuronas que han demostrado su eficacia en tareas de segmentación y clasificación en imágenes. Su funcionamiento es similar al de las neuronas de la corteza visual primaria del cerebro humano. Una Red Neuronal Convolucional (CNN) está formada internamente por una sucesión de capas convolucionales, funciones de activación, capas de *pooling*, capas de neuronas clásicas o densas, etc, que en su conjunto dan lugar a una arquitectura que permite extraer las características deseadas de la imagen de entrada y llevar a cabo a partir de éstas, diferentes tareas como clasificación, regresión o segmentación. En función de la tarea final que debe realizar la red, se puede prescindir de algunos de los elementos enumerados. El problema planteado en este TFG podría considerarse una tarea de regresión de mapas de probabilidad, por lo que se ha optado por no utilizar capas de neuronas clásicas y por lo tanto se realizará un estudio principalmente de los aspectos, capas y elementos de las CNNs relativos a la parte puramente convolucional, enfocada a la extracción de características de las imágenes. De esta forma, a continuación se describen la Convolución Clásica, la Convolución Separable, funciones de activación, Normalización del *Batch* y operaciones de *pooling*.

2.3.3. Capa convolucional

En primer lugar, es importante destacar que la capa convolucional descrita a continuación es la 2D (bidimensional), es decir que procesa secuencialmente imágenes individuales, no grupos de imágenes (caso de la capa convolucional 3D).

La capa convolucional realiza, sobre la imagen de entrada, una operación de convolución. En otras palabras, realiza un filtrado de la imagen utilizando una máscara o *kernel*. Dicho *kernel* a aplicar es el que define el comportamiento del filtrado, de forma que diferentes *kernels* dan lugar a diferentes resultados.

Un *kernel* es una matriz cuadrada que influye directamente en el resultado del filtrado en base a dos aspectos, sus dimensiones ($n \times n$) y los valores que contiene. El primero, las dimensiones, lo define el diseñador cuando configura la arquitectura de la red, mientras que los valores que contiene, que se denominan pesos, son ajustados automáticamente durante el proceso de entrenamiento. Por otro lado, en una misma capa convolucional pueden aplicarse varios *kernels*, obteniéndose a la salida la concatenación, en una tercera dimensión, de los resultados de convolucionar la imagen de entrada con cada uno de los *kernels*.

Respecto a la operación de convolución, se trata de realizar una combinación lineal sobre cada ventana de $n \times n$ píxeles de la entrada (siendo n la dimensión del *kernel*). La Figura 2.12 resume de forma gráfica una capa convolucional, donde la entrada (x) tiene dimensiones 4×4 y el *kernel* (k) tiene dimensiones 3×3. El parámetro *bias* (b), también entrenable, es un *offset* que se aplica a todos los píxeles de salida de la convolución. El grupo de ecuaciones 2.9 muestra la forma de calcular el valor de cada píxel de salida.

Como puede deducirse a partir de la Figura 2.12, las dimensiones de la salida dependen tanto de las de la entrada como de las del *kernel*, de acuerdo a la expresión $n_y = n_x - n_k + 1$, donde n representa las dimensiones de un solo eje (u ó v) de la imagen. Por ejemplo, en una convolución de una imagen de dimensiones 10×8 con un kernel 5×5, la salida tendría dimensiones 6×4, obtenidas aplicando la expresión anterior a cada uno de los ejes.

$$\begin{aligned} y_{11} &= \sum_{j=1}^{3}\sum_{i=1}^{3} x_{i,\,j}\; k_{i,\,j} + b \qquad & y_{12} &= \sum_{j=1}^{3}\sum_{i=1}^{3} x_{i,\,j+1}\; k_{i,\,j} + b \\ y_{21} &= \sum_{j=1}^{3}\sum_{i=1}^{3} x_{i+1,\,j}\; k_{i,\,j} + b \qquad & y_{22} &= \sum_{j=1}^{3}\sum_{i=1}^{3} x_{i+1,\,j+1}\; k_{i,\,j} + b \end{aligned} \tag{2.9}$$

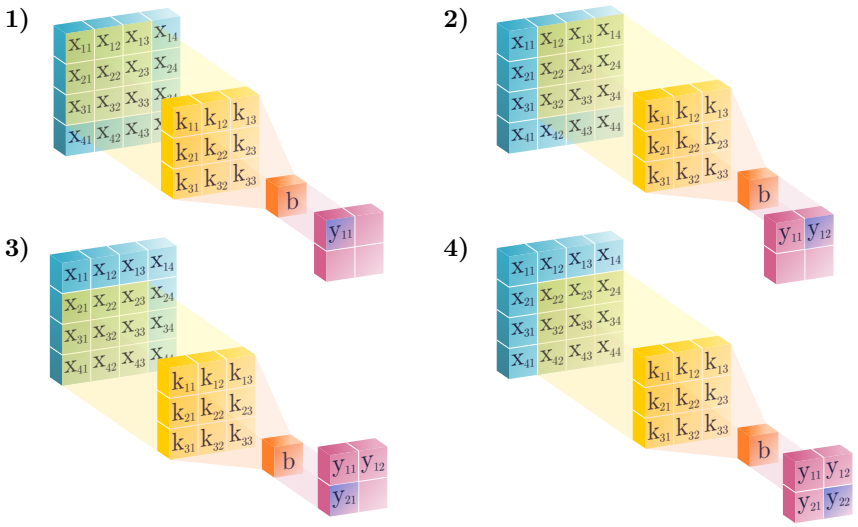


Figura 2.12: Esquema de la operación convolución sobre una imagen (x) de dimensiones 4×4, empleando un kernel (k) de dimensiones 3×3.

En ocasiones, puede resultar interesante que las dimensiones de la salida no sean muy reducidas con respecto a la entrada, o que directamente la salida sea del mismo tamaño que la entrada, pues unas dimensiones demasiado reducidas derivan en una rápida pérdida de información. Para ello se utiliza las técnicas de *padding*, como por ejemplo el *zero-padding*, que consiste en añadir filas y columnas de píxeles nulos alrededor de la imagen de entrada. De esta forma se varían las dimensiones de la entrada hasta conseguir que la salida de la convolución tenga las dimensiones deseadas. Por ejemplo, si a la entrada de la convolución de la figura 2.12 se le añadiese una tira de píxeles nulos en todos sus márgenes, sus dimensiones serían 6×6, lo que daría lugar a una salida 4×4 (mismas dimensiones que la imagen de entrada original).

En otros casos, se desea que la capa convolucional reduzca considerablemente las dimensiones, por ejemplo, en una configuración de tipo *encoder* (de la que se hablará en la sección 2.5), las capas convolucionales deben reducir sucesivamente las dimensiones de la imagen y aumentando la profundidad, como resultado de aplicar varios filtros por capa. Para conseguir una reducción de las dimensiones de la salida de cada capa se utiliza el parámetro *stride*, definido por el usuario al diseñar la arquitectura. Dicho parámetro tiene dos componentes (u y v), cuyo valor establece cuántos píxeles se desplaza la ventana sobre la que se aplica el *kernel*, en cada una de las direcciones. En la Figura 2.12 el valor del *stride* es $[1, 1]$, pues la ventana se desplaza cada vez un solo píxel, tanto en horizontal como en vertical. La Figura 2.13 muestra un esquema de una capa convolucional aplicada a una entrada de dimensiones 5×5 cuando el valor del stride es [2, 2], es decir, la ventana sobre la que se aplica el *kernel* se desplaza 2 píxeles en horizontal y 2 en vertical.

En todos los ejemplos mostrados, tanto la imagen de entrada como la salida tienen tercera dimensión de valor 1. Es importante destacar que si la entrada a una capa convolucional tiene profundidad (tercera dimensión distinta de 1), el *kernel* a utilizar tendrá la misma profundidad, de forma que tras la operación de convolución, la salida vuelve a tener profundidad 1. Esto se debe a que al convolucionar el *kernel* con cada una de las ventanas de la imagen de entrada, dichas ventanas serán tridimensionales, pero la operación de convolución sigue dando un único valor por cada ventana, aunque se extienda la operación de 2 a 3 dimensiones. La Ecuación 2.10 muestra como ejemplo, el cálculo del valor del píxel de salida y_{11} con un kernel 3×3 cuando la entrada tiene una profundidad de capa de valor 5. Por otro lado, en una misma capa convolucional pueden aplicarse varios *kernels*, lo que se denomina número de filtros aplicados. Cada uno de estos filtros aporta una capa a la salida, de forma que el valor de profundidad de

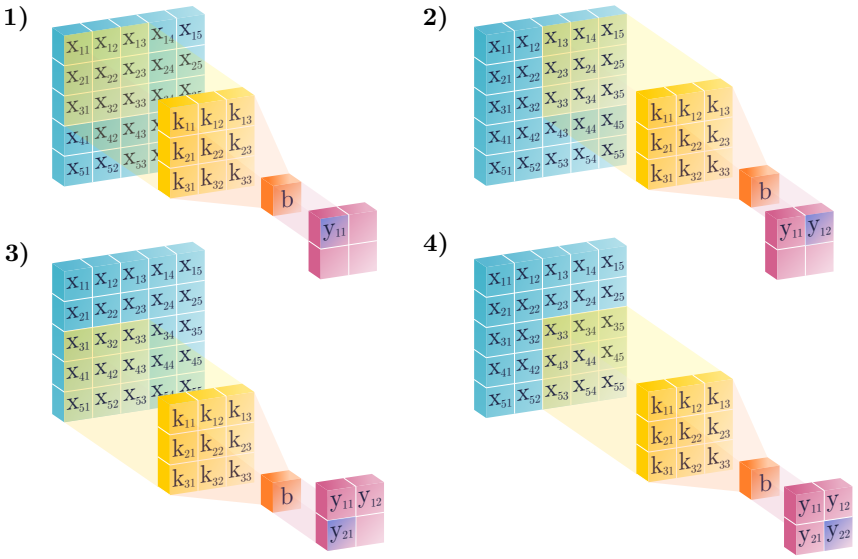


Figura 2.13: Esquema de la operación convolución sobre una imagen (x) de dimensiones 5×5, empleando un kernel (k) de dimensiones 3×3 y con *stride* [2, 2].

ésta coincide con el número de filtros aplicado. Por ejemplo, si una matriz $64 \times 64 \times 10$ se entrega como entrada a una capa convolucional de 100 filtros (manteniendo el tamaño de la imagen), la salida tiene dimensiones $64 \times 64 \times 100$.

$$y_{11} = \sum_{k=1}^{5} \sum_{j=1}^{3} \sum_{i=1}^{3} (x_{i,j,k} \; k_{i,j,k}) + b \tag{2.10}$$

Puede deducirse fácilmente que cuando en una capa convolucional se aplican numerosos filtros y por tanto la salida tiene un valor alto de profundidad, los *kernels* de la siguiente capa deberán tener esa misma profundidad, lo que se traduce en un gran número de parámetros entrenables y un aumento de la complejidad y del tamaño de la red.

En lo relativo al papel que desempeñan las capas convolucionales dentro de una red completa, su función es extraer características de la imagen de entrada. Sin embargo, hay redes formadas en su totalidad por capas convolucionales (*fully convolutional*) que además de extraer características realizan también clasificación o regresión. A continuación, la explicación se centra únicamente en la tarea de extracción de características. Dichas características extraídas por cada capa dependen de los valores del *kernel*, los cuales se ajustan durante el entrenamiento. Por ejemplo, un filtro puede extraer los píxeles que corresponden a contornos superiores de objetos, otro filtro los correspondientes a contornos inferiores, zonas cóncavas, zonas convexas, esquinas, etc. Las matrices que contienen las características extraídas, son sometidas nuevamente a operaciones de convolución que las combinan para extraer características más complejas. De esta forma con cada capa convolucional se va adquiriendo un mayor grado de abstracción.

La Figura 2.14 muestra el resultado de aplicar dos capas convolucionales a una imagen en escala de grises (un solo canal). Dichas capas han sido entrenadas (dentro de una arquitectura más compleja) para resolver un problema de clasificación. Como se puede observar, la primera capa (formada por 30 filtros) extrae características relacionadas con los distintos contornos del objeto. En cambio, la segunda capa (15 filtros) extrae características más complejas, con un grado de abstracción mayor, y que por tanto no son tan evidentes como en el primer caso. Son estas características más complejas las que permiten, a capas convolucionales posteriores y en último lugar a capas de neuronas clásicas, solucionar el problema de clasificación en este caso.

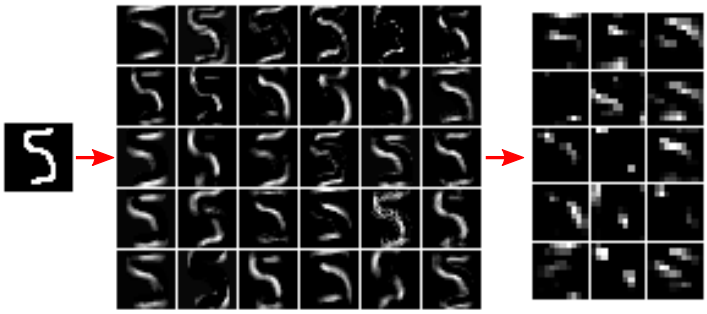


Figura 2.14: Ejemplo de extracción de características de una imagen mediante capas convolucionales.

2.3.4. Convolución Separable

Uno de los principales problemas que presentan las capas convolucionales es su alta carga computacional, ya que deben realizar numerosas multiplicaciones. En una capa convolucional clásica en la que la entrada tiene M canales, la salida tiene dimensiones $Y \times Y$, y se aplican N *kernels* de dimensiones $K \times K$, el número de multiplicaciones que se llevan a cabo en dicha capa convolucional se calcula como indica la Ecuación 2.11, entendiendo un valor de *stride* [1, 1].

$$N^{o} \text{ de } multiplicaciones = K^2 Y^2 MN \tag{2.11}$$

Con el objetivo de reducir el número de multiplicaciones y por tanto reducir la carga computacional, aumentando la velocidad y mejorando el rendimiento, se desarrolló una variante de la convolución clasica, denominada Convolución Separable, en inglés *Depthwise Separable Convolution* [29].

Vista desde fuera, una Convolución Separable se utiliza del mismo modo que una Capa Convolucional convencional. Se especifica el número de *kernels* a aplicar y las dimensiones de éstos, así como el valor del *stride*. Las dimensiones de la salida coinciden con las que tendría en el caso de utilizar una Capa Convolucional convencional. Sin embargo, internamente la Convolución Separable opera de forma totalmente diferente y además, el número de parámetros entrenables o pesos que contiene es menor, mejorando también en ese aspecto.

Su estructura interna se divide en dos etapas claramente diferenciadas. La Etapa 1, denominada en inglés *Depthwise Convolution*, convolución en profundidad, realiza una convolución individual de cada canal de la entrada con un *kernel* diferente, siendo las dimensiones de estos kernels (iguales para todos) las especificadas por el diseñador. Si la entrada tiene M canales, hay M *kernels* y cada uno se convoluciona de forma individual con cada canal de la entrada, de forma que la salida tiene M canales también. La Etapa 2, denominada *Pointwise Convolution* o convolución punto a punto, puede entenderse como una capa convolucional clásica en la que la entrada es la salida de la etapa anterior, con la particularidad de que los *kernels* (cada uno con el número de canales que tenga la salida de la Etapa 1) tienen dimensiones 1×1, siendo el número de *kernels* el especificado por el diseñador. De esta forma, si la salida de la Etapa 1 tiene M canales, se utilizan N *kernels* de dimensiones $1 \times 1 \times M$, generando una salida con las mismas dimensiones que la de la Etapa 1, pero con N canales. La Figura 2.15 muestra de forma gráfica la estructura interna de una Convolución Separable.

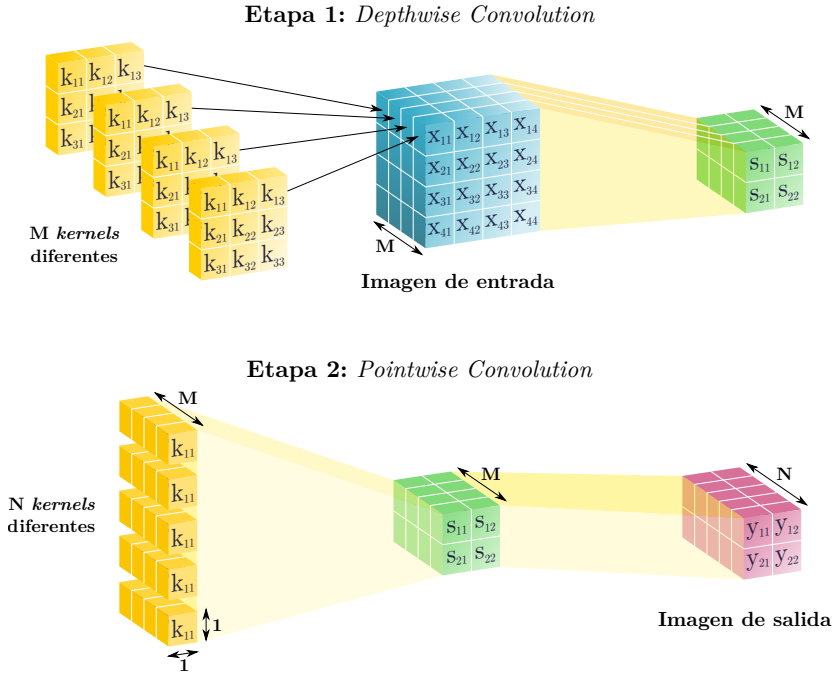


Figura 2.15: Estructura interna de una capa Convolucional Separable.

A continuación, se analiza el número de multiplicaciones que tienen lugar en una capa convolucional separable. Teniendo la entrada M canales, la salida dimensiones $Y \times Y$, y especificando N *kernels* de dimensiones $K \times K$, el número de multiplicaciones que se llevan a cabo en dicha capa se calcula como indica la Ecuación 2.12, entendiendo un valor de *stride* [1, 1].

$$N^o \text{ de multiplicaciones} = K^2Y^2M + Y^2MN = Y^2M(K^2 + N) \tag{2.12}$$

Si se compara el resultado anterior con el caso de la Capa Convolucional convencional, se obtiene la relación de reducción del número de multiplicaciones, que puede aproximarse a la relación de mejora en carga computacional. Dicha relación se muestra en la ecuación 2.13, donde se puede apreciar como solo depende de los parámetros que debe especificar el diseñador: número de filtros (N) y dimensiones de éstos (K). Como ejemplo, en una capa con 128 filtros de 7×7 elementos, una Convolución Separable es aproximadamente 35 veces más rápida que una Capa Convolucional convencional.

$$\text{Relación de mejora del rendimiento} = \frac{K^2Y^2MN}{Y^2M(K^2 + N)} = \left(\frac{1}{N} + \frac{1}{K^2}\right)^{-1} \tag{2.13}$$

Por otro lado, existe una mejora también en el número de pesos que contienen. Las Ecuaciones 2.14 muestran el cálculo del número de pesos (despreciando el parámetro *bias*) en una convolución clásica y en una separable, así como la relación entre ambas. La expresión de dicha relación coincide con la de mejora del rendimiento, de forma que, como en el ejemplo anterior, en una capa con 128 filtros de 7×7 elementos, una Convolución Separable es aproximadamente 35 veces más rápida que una Capa Convolucional convencional y tiene 35 veces menos de pesos, o dicho de otra manera, su tamaño es 35 veces menor.

$$
\begin{aligned}
&N^{o}\ pesos\ convencional = MK^2N \\
&N^{o}\ pesos\ separable = K^2M + NM = M(K^2 + N) \\
&Relación\ de\ mejora = \frac{MK^2N}{M(K^2+N)} = \left(\frac{1}{N} + \frac{1}{K^2}\right)^{-1}
\end{aligned}
\tag{2.14}
$$

2.3.5. Funciones de Activación

Una función de activación es un elemento propio de las redes neuronales que suele situarse a continuación de cada capa convolucional, en el caso de CNNs o detrás de cada neurona, si se trata de capas de *Densas*. Su misión es aplicar una no-linealidad a la salida de la capa que la precede, y de esta forma aportar mayor versatilidad al sistema. La forma en la que se realiza dicha operación no-lineal depende de la función de activación que se utilice, por ejemplo, como se verá más adelante, una Unidad Lineal Rectificada (ReLU) permite el paso de los valores positivos y convierte en nulos los valores negativos. Otras, como la función sigmoidal, acotan los valores en el rango (0, 1). Por tanto, cada función proporciona activaciones de diferentes características y se utilizan en unos casos determinados.

A continuación se detallan algunas de las funciones de activación más comunes, mostrando la expresión matemática que las define y su representación gráfica. Posteriormente se realiza una comparativa entre las funciones previamente enumeradas, explicando en qué casos conviene utilizar cada una de ellas. En [30] puede encontrarse una explicación más extensa de gran número de funciones de activación.

En primer lugar, la función **Sigmoide o Sigmoidal** es quizá la más antigua y una de las más populares. Su expresión matemática se muestra en la Ecuación 2.15. Como puede deducirse a partir de la representación gráfica, mostrada en la Figura 2.16, el comportamiento puede aproximarse a una función lineal para valores próximos a 0, mientras que satura en 1 para valores positivos altos y en 0 para valores negativos bajos, de forma que "comprime" los valores de la entrada en el intervalo (0, 1), no permitiendo activaciones negativas ni superiores a la unidad.

$$
f(x) = \frac{1}{1 + e^{-x}} \tag{2.15}
$$

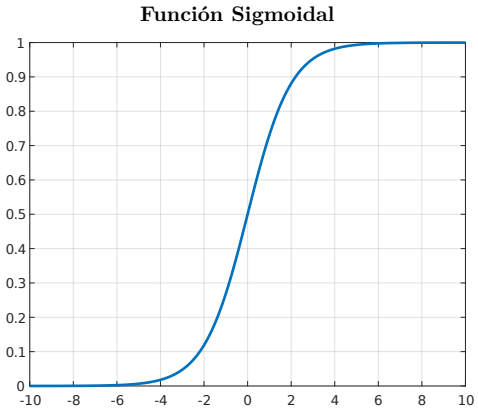


Figura 2.16: Representación gráfica de la función sigmoidal.

Otra función comúnmente utilizada como activación es la **tangente hiperbólica**. La Ecuación 2.16 muestra su expresión matemática y la Figura 2.17 su representación gráfica. La curva es similar a la de la sigmoidal pero extendida en el rango (-1, 1), permitiendo activaciones negativas. Tanto la función tangente hiperbólica como la función sigmoidal se aplican de forma individual a la salida de cada neurona en una capa de *Densas* o cada píxel de la matriz de salida en una capa Convolucional.

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.16}$$

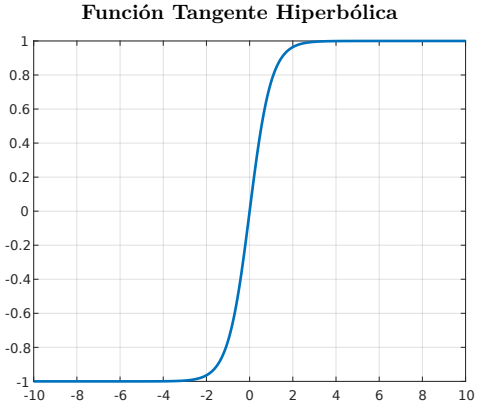


Figura 2.17: Representación gráfica de la función tangente hiperbólica.

La función ***softmax*** se utiliza como activación generalmente en capas de *Densas*, pues para su cálculo necesita un vector v formado por las salidas de una capa. Una de sus principales características es que está normalizada, es decir, la suma de las salidas de la función *softmax* para cada uno de los componentes del vector v da como resultado la unidad. La Ecuación 2.17 muestra su expresión y la Figura 2.18 su gráfica, donde puede observarse cómo para un vector v de ejemplo formado por 6 elementos, la suma de las salidas de la función es 1. Suele emplearse en problemas de clasificación multiclase.

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \tag{2.17}$$

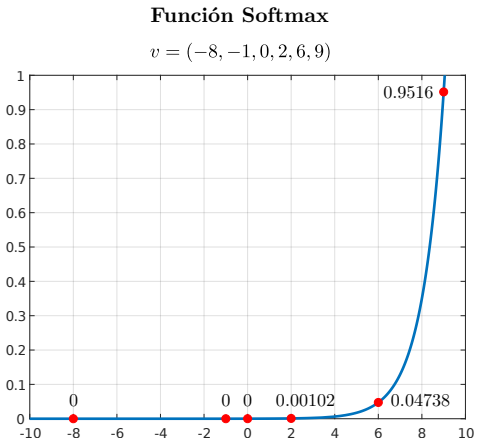


Figura 2.18: Representación gráfica de la función *softmax*.

A continuación se explican dos funciones de activación más, del grupo de las funciones rectificadas: la función ReLU (*Rectified Linear Unit*) y la *Leaky* ReLU.

La función ReLU se comporta como un rectificador lineal, es decir, no permite activaciones negativas mientras que los valores positivos a su entrada pasan tal cual a la salida. Su expresión se muestra en la Ecuación 2.18 y su representación en la Figura 2.19.

$$f(x) = máx(0, x) = \begin{cases} x, & \text{si } x \geq 0 \\ 0, & \text{si } x < 0 \end{cases} \tag{2.18}$$

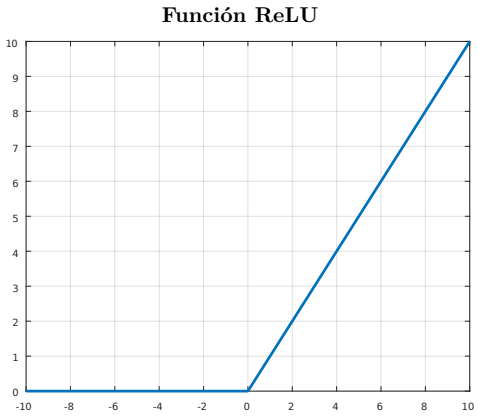


Figura 2.19: Representación gráfica de la función ReLU.

Por su parte, la función *Leaky* ReLU sí permite activaciones negativas, configurando el peso de éstas mediante el parámetro α, tal y como muestran la Ecuación 2.19 y la Figura 2.20.

$$f(x) = máx(\alpha x, x) = \begin{cases} x, & \text{si } x \geq 0 \\ \alpha x, & \text{si } x < 0 \end{cases} \tag{2.19}$$

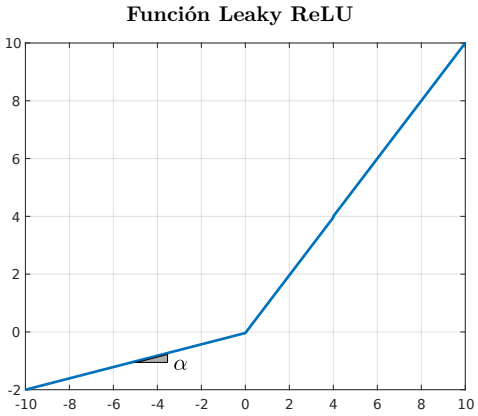


Figura 2.20: Representación gráfica de la función *Leaky* ReLU.

Respecto a la utilización de cada función de activación, de forma general pueden seguirse los siguiente puntos:

- La función ***softmax*** se utiliza en capas de *Densas* y en tareas de clasificación multiclase, generalmente en la última capa.
- La funciones **sigmoidal**, **tangente hiperbólica** y **ReLU** se utilizan tanto en capas de *Densas* como en capas convolucionales, tanto para tareas de clasificación como de regresión.
- En las capas convolucionales la función **ReLU** es la más utilizada

En [31] se analiza el uso de diferentes funciones de activación en una tarea de clasificación. Respecto a una comparativa entre ReLU y *Leaky* ReLU, en [32] se explica cómo en una tarea de clasificación

la *Leaky* ReLU obtenía mejores resultados que la ReLU, utilizando un valor de α en torno a 0.2. Una pequeña activación negativa (caso de la *leaky* ReLU) ayuda a eliminar el problema denominado *Dying ReLU*, que ocurre cuando la función ReLU entrega sucesivamente valores nulos a su salida, bloqueando el entrenamiento al obtenerse un valor de gradiente igual a cero.

2.3.6. Normalización del *batch*

La técnica denominada Normalización del *batch*, en inglés *Batch Normalization*, se emplea como una capa de regularizacion en las Redes Neuronales Profundas, tanto en CNNs como en capas de *Densas*, y su objetivo es conseguir acelerar y hacer más eficiente el proceso de entrenamiento

En [33] puede encontrarse una descripción detallada de los problemas que pretende solucionar la técnica de normalización del *batch*, así como de su funcionamiento. A continuación, se explican de forma resumida estos conceptos.

Cuando se está llevando a cabo el entrenamiento de una red, la base de datos utilizada se divide en fragmentos, cada uno de los cuales se denomina *batch*. Por ejemplo, si para entrenar una CNN se dispone de 1000 imágenes, pueden formarse 50 *batches* de 20 imágenes, 100 *batches* de 10 imágenes, etc. El tamaño elegido para el *batch* condiciona el progreso del entrenamiento, por ejemplo un tamaño demasiado grande puede hacer al modelo alcanzar el sobre-entrenamiento (*overfitting*) más rápidamente. Los datos contenidos en cada *batch* son procesados secuencialmente por la red y tras finalizar se calcula el nuevo valor de los pesos y se actualizan éstos. De esta forma, cuando se está procesando un *batch*, los valores de las distintas activaciones de una capa siguen una distribución estadística que no coincidirá con la del siguiente *batch*, pues los pesos habrán cambiado. A este cambio sucesivo en la distribución de las activaciones se le denomina en [33] *Internal Covariate Shift*, y su principal efecto es la ralentización del entrenamiento. Por tanto, manteniendo constante la distribución de las activaciones con el cambio de *batch* se debe conseguir un aumento en la velocidad a la que progresa el entrenamiento.

Para reducir el *Internal Covariate Shift*, la técnica de Normalización del batch lleva a cabo los pasos enumerados a continuación:

1. Para un *batch* completo, calcula la media (μ) y la desviación típica (σ) de todas las activaciones de la capa donde se aplique, tal y como se muestra en la ecuación 2.20, donde x representa el conjunto de valores que toma una activación concreta a lo largo de un *batch* de m elementos.

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i$$
$$\sigma = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2} \tag{2.20}$$

2. Se normaliza el valor de cada activación de acuerdo a la media y a la desviación típica calculadas, tal y como muestra la Ecuación 2.21, donde $\widehat{x}$ representa el conjunto de valores que toma una activación concreta a lo largo de un *batch*, normalizado.

$$\widehat{x} = \frac{x_i - \mu}{\sigma} \tag{2.21}$$

3. El último paso es realizar el escalado del vector normalizado ($\widehat{x}$), tal y como indica la Ecuación 2.22, donde γ y β son dos parámetros entrenables. La salida y es la salida de la capa de Normalización del *batch*.

$$y = \gamma\widehat{x} + \beta \tag{2.22}$$

Los parámetros γ y β son los dos únicos parámetros entrenables que posee la capa de Normalización del *batch*. Cuando se trabaja con una imagen o matriz de varios canales, las dimensiones de ambos parámetros son $1 \times 1 \times N$, siendo N el número de canales. De está forma, si por ejemplo se ubica una capa de Normalización del *batch* tras una capa convolucional que entrega a su salida una imagen 64×64 con 16 canales, las dimensiones de γ y de β serán $1 \times 1 \times 16$, teniendo la capa un total de 64 parámetros entrenables.

2.3.7. Operaciones de *pooling*

La capas de *pooling* son elementos de las CNNs que se insertan generalmente a continuación de una capa convolucional (y de la correspondiente función de activación). Su objetivo es realizar una selección de las características obtenidas por dicha capa, reduciendo las dimensiones de la imagen.

Si a continuación de una capa convolucional se tiene una imagen de dimensiones $u \times v$ y se aplica una operación de *pooling* de dimensiones $n \times n$, por cada ventana de $n \times n$ píxeles de la imagen de entrada, se calcula el valor de un único píxel de la salida, de forma que dicha salida tiene dimensiones $\frac{u}{n} \times \frac{v}{n}$. El valor calculado por cada ventana depende del tipo de *pooling* que se realice. Aquí se citan los dos tipos de *pooling* más comunes: *Max Pooling* y *Average Pooling*. La Figura 2.21 muestra el esquema de una operación de *pooling* 2×2 a una imagen 4×4.

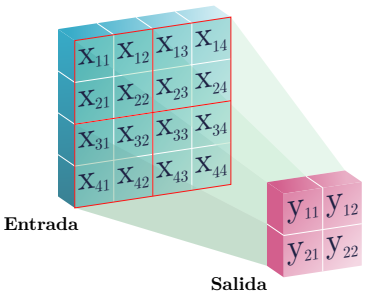


Figura 2.21: Aplicación de una operación de *pooling* 2×2 a una imagen 4×4.

La operación de ***Max Pooling*** consiste en asignar a la salida el valor más alto (máxima activación) de los píxeles de cada ventana, consiguiéndose así filtrar espurios. La Ecuación 2.23 muestra el cálculo del valor del pixel y_{11} de la Figura 2.21, realizando una operación de *Max Pooling*.

$$y_{11} = máx(x_{11}, x_{12}, x_{21}, x_{22}) \tag{2.23}$$

La operación de ***Average Pooling*** consiste en asignar a la salida el valor medio de los píxeles de cada ventana. La Ecuación 2.24 muestra el cálculo del valor del pixel y_{11} de la Figura 2.21, realizando una operación de *Average Pooling*.

$$y_{11} = \frac{x_{11} + x_{12} + x_{21} + x_{22}}{4} \tag{2.24}$$

Como se puede deducir de las ecuaciones anteriores, las dos operaciones de *pooling* descritas no contienen parámetros entrenables. Su carga computacional es muy baja, y además, al reducir las dimensiones de la imagen reducen la carga computacional de las capas posteriores.

2.4. Arquitecturas basadas en Bloques Residuales: *ResNet*

Cuando se diseñan redes neuronales profundas, se busca que obtengan un nivel alto de precisión sin que exista sobre-entrenamiento (*overfitting*). Sin embargo, se ha demostrado cómo en ocasiones, a medida que se añaden capas en una red, se alcanza un momento en el que la precisión obtenida, tanto en entrenamiento como en validación, comienza a disminuir. Como se ha explicado en el apartado 2.3.1, durante el entrenamiento de una red, se calcula el gradiente del error obtenido con respecto a todos los pesos de la red. La forma de hacerlo es calcular primeramente el gradiente en la salida y propagarlo por las sucesivas capas hacia la entrada, de forma que su valor se va reduciendo. Cuando una red tiene muchas capas el valor del gradiente propagado hasta las primeras capas tiene un valor muy pequeño, lo que se traduce en una actualización de los pesos demasiado lenta. Dicho de otra manera, las primeras capas de la red son las más lentas y difíciles de entrenar. Este problema se conoce como Desvanecimiento del Gradiente.

En [4] se propone una solución que permite construir redes más complejas (con mayor número de capas) sin que por ello se alcance más rápidamente el valor máximo de precisión y reduciendo el problema de desvanecimiento de gradiente. Dicha solución consiste en la utilización de bloques residuales. En una arquitectura clásica la entrada de cada capa es la salida de la anterior, es decir, su estructura es completamente plana. Sin embargo, en una arquitectura residual se rompe esa estructura plana, creando conexiones como la mostrada en la Figura 2.22. Una recirculación de la entrada mediante una estructura formada por dos ramas paralelas afecta a la propagación del gradiente, haciendo que éste no disminuya en exceso y llegue a las primeras capas con un valor suficientemente alto, mejorando el entrenamiento de éstas.

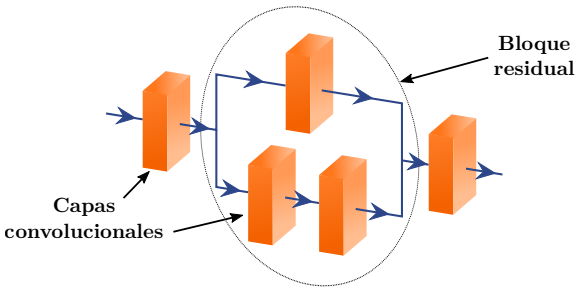


Figura 2.22: Ejemplo de bloque residual insertado en una CNN.

La estructura interna de un bloque residual puede adoptar diversas configuraciones. La solución propuesta en [4] es la utilización de lo que hemos denominado Bloques Identidad y Bloques Convolucionales.

2.4.1. Bloques Identidad

Un Bloque Identidad es un bloque residual en el cual una de las dos ramas no realiza operación alguna. De esta forma, la salida del bloque coincide con la salida de la otra rama sumada con la entrada. La Figura 2.23 es un esquema sencillo de un Bloque Identidad.

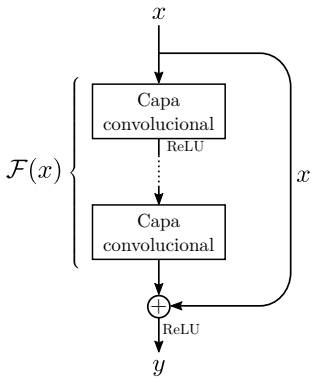


Figura 2.23: Esquema de un Bloque Identidad genérico.

Como puede observarse en la Figura 2.23 y de acuerdo con la nomenclatura utilizada en [4], se denomina x a la entrada, y a la salida y $\mathcal{F}(x)$ a la operación que realiza la rama convolucional sobre la entrada x, de forma que $y = ReLU\{\mathcal{F}(x) + x\}$. A continuación, en la Figura 2.24 se muestra la gráfica que aporta [4] donde pueden verse los resultados obtenidos utilizando los bloques identidad. La imagen de la derecha corresponde al error obtenido por dos redes planas idénticamente entrenadas, una de 18 capas (azul) y otra de 34 capas (rojo). La imagen de la izquierda muestra el error obtenido por las redes equivalente entrenadas de 18 y 34 capas, pero utilizando bloques residuales identidad de dos capas convolucionales.

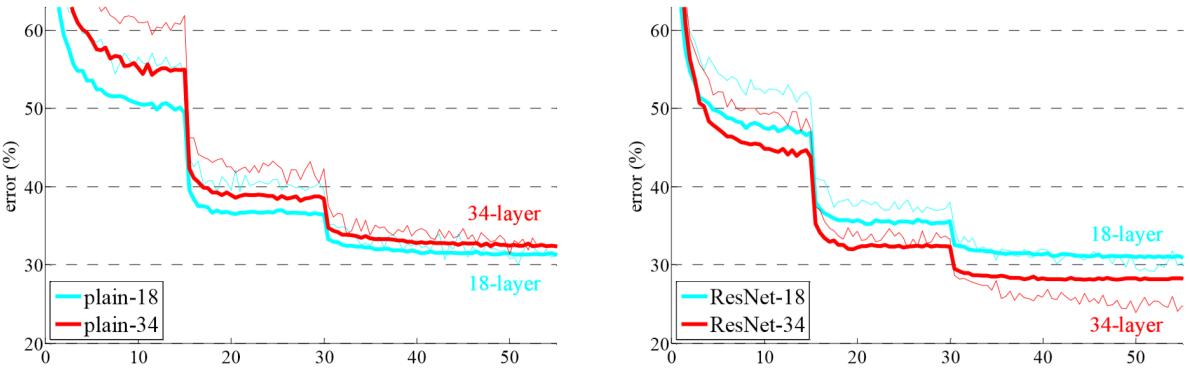


Figura 2.24: Comparativa entre la red *ResNet* y una red "plana". En trazo fino el error de entrenamiento y en trazo grueso el error de test. Imagen extraída de [4].

Como puede observarse en la Figura 2.24, en el caso de la red completamente plana, el error es mayor cuando se tienen 34 capas que cuando se tienen 18. Sin embargo, en la red que emplea bloques residuales el error es más pequeño para el caso de 34 que para el de 18, dando solución al problema inicialmente planteado. El bloques residual final utilizado en [4] fue un Bloque Identidad formado, en su rama principal, por tres capas convolucionales: la primera con 64 *kernels* 1×1, la segunda con 64 *kernels* 3×3 y la tercera con 256 *kernels* 1×1.

2.4.2. Bloques Convolucionales

Los denominados Bloques Convolucionales son otro tipo de bloques residuales con un comportamiento más parecido a la capa convolucional. La diferencia que presentan con respecto al Bloque Identidad es que en la rama derecha (de acuerdo con el esquema de la Figura 2.23) existe también una capa convolucional.

Su estructura se muestra en la Figura 2.25. La expresión del valor de la salida es $y = ReLU\{\mathcal{F}(x) + \mathcal{G}(x)\}$.

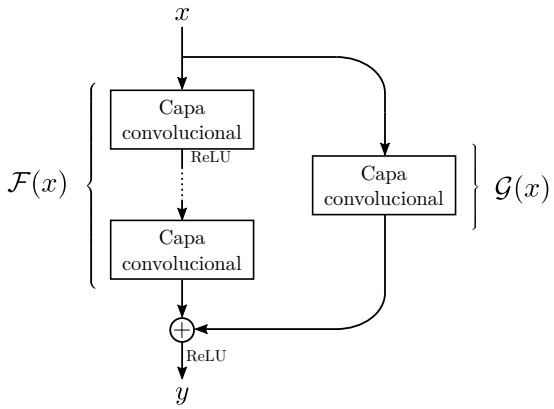


Figura 2.25: Esquema de un Bloque Convolucional.

Es importante destacar como, tanto en los Bloques Identidad como en los Bloques Convolucionales, la última función de activación se lleva a cabo tras la suma de las dos ramas, y no de forma individual en cada una de ellas.

2.5. Configuración *Encoder-Decoder*

En primer lugar, es importante definir el significado de configuración en este contexto. Una configuración es una forma de estructurar y conectar los distintos elementos, capas o bloques, para dar lugar a una red neuronal profunda completa. En concreto, para las CNNs y en este TFG, la configuración básica empleada y que se describe a continuación es el *Encoder-Decoder* (específicamente en el caso de trabajar con imágenes y CNNs).

Como su propio nombre indica, una red basada en una configuración *encoder-decoder* está formada a su vez por dos secciones bien diferenciadas: *encoder* y *decoder*. El *encoder* o codificador recibe a su entrada una imagen, que generalmente tiene un canal, en el caso de escala de grises o profundidad, o tres canales, en el caso RGB. Internamente, el *encoder* está formado por sucesivas capas convolucionales, funciones de activación, capas de *pooling*, etc, a través de las cuales la imagen va reduciendo sus dimensiones y aumentando en número de canales.

El *decoder* o decodificador recibe a su entrada la salida del *encoder* que, como se ha dicho, se trata de una matriz de dimensiones reducidas pero gran número de canales. Su estructura interna es la estructura inversa a la del *encoder*, formada por capas convolucionales a través de las cuales la imagen va aumentando sus dimensiones y reduciendo el número de filtros. De esta forma, la salida del *decoder* debe tener las mismas dimensiones que la entrada del *encoder*. Respecto a cómo las capas convolucionales generan a la salida una imagen de mayores dimensiones que su entrada, se debe al uso de capas convolucionales traspuestas. La Figura 2.26 muestra un esquema de una convolución traspuesta. En ocasiones, para una

mayor aumento de las dimnesiones de la salida se utilizan operaciones de *padding*, que consisten en intercalar nuevos valores entre los píxeles de la entrada, por ejemplo valores nulos, el valor del pixel más cercano, etc. Otra forma de aumentar las dimensiones de la imagen es el uso de operaciones de *up-sampling*.

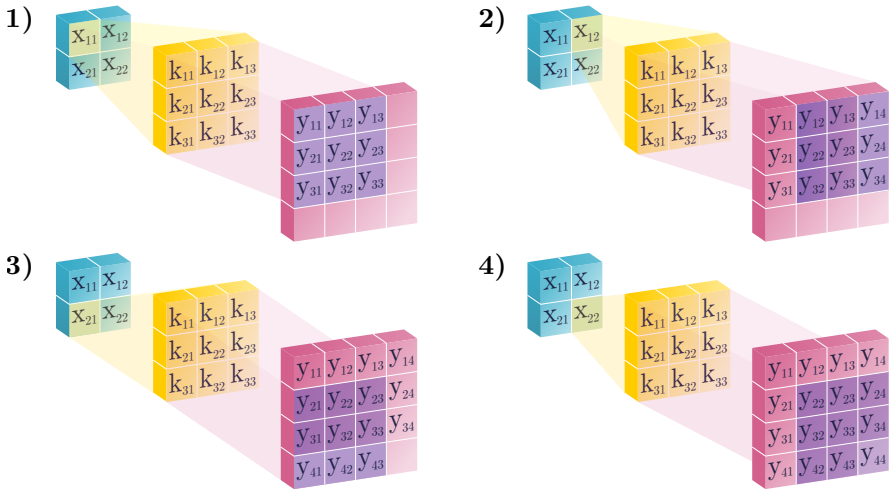


Figura 2.26: Esquema de una Capa Convolucional Traspuesta con un *kernel* 3×3, *stride* [1, 1] y sin *padding*.

En cuanto a la utilidad de la configuración *encoder-decoder*, la principal es su uso en tareas de segmentación semántica. El *encoder* extrae características de la imagen de entrada, que tras las sucesivas capas van aumentando su nivel de abstracción. Además, cada canal representa unas características en principio diferentes. Con la salida del *encoder*, es decir, en base a las características extraídas, el *decoder* realiza una nueva representación de la imagen de entrada (imagen segmentada). La Figura 2.27 muestra un ejemplo de estructura *encoder-decoder* y un par de imágenes de entrada y salida.

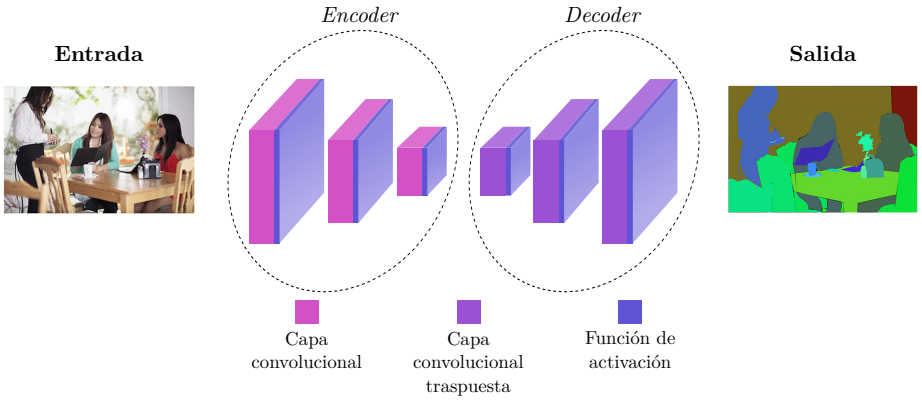


Figura 2.27: Esquema de una configuración *encoder-decoder* con imágenes de ejemplo obtenidas de [5]

.

2.6. Aspectos relativos al entrenamiento de Redes Neuronales

Como se introdujo en el apartado 2.3.1, para entrenar un sistema basado en redes neuronales, deben definirse una serie de aspectos o parámetros que influyen de forma directa en el progreso del aprendizaje. A continuación se enumeran estos parámetros:

- **Función de coste.** Compara la salida obtenida con la salida deseada, obteniendo el error. En el apartado 2.8 se describen algunas de las principales funciones utilizadas para este fin.
- **Optimizador.** Algoritmo que controla la actualización de los pesos. Se describe en el apartado 2.7.
- **Tamaño del *batch*.** Numero de entradas y salidas que se procesan antes de cada actualización de los pesos. Si el tamaño del *batch* es elevado, el progreso del aprendizaje puede derivar más fácilmente en sobre-entrenamiento. Por el contrario, un tamaño reducido de *batch* ayuda al sistema a generalizar, evitando el sobre-entrenamiento, sin embargo, puede no llegarse a una solución correcta, permaneciendo en un mínimo local de la función de pérdidas.
- **Número de épocas.** Es el número de veces que el sistema procesa el total de datos de entrenamiento. Si el número de épocas es muy alto y los datos no son muy numerosos, puede alcanzarse el sobre-entrenamiento. Sin embargo, con gran cantidad de datos y siendo éstos suficientemente variados y representativos, un número alto de épocas permite, en general, minimizar en mayor medida la función de coste, y por tanto alcanzar un mejor funcionamiento del sistema. En la práctica, al finalizar una época se procesa un pequeño set de datos no utilizados en el entrenamiento, que permite verificar el correcto progreso del aprendizaje, de cara a detectar un sobre-entrenamiento del sistema.

2.7. Optimizadores

2.7.1. Introducción

Cuando se está realizando el entrenamiento de una red neuronal, la base de datos utilizada se procesa por fragmentos o *batches*. Durante este procesado se realiza la comparación de las salidas obtenidas con las salidas deseadas (*targets*), utilizando una función de pérdidas o de coste (lo cual se describe en el siguiente punto). Cuando se completa el procesamiento de un *batch*, se aplican algoritmos como el *Stocastic Gradient Descent* (SGD) a los resultados de la función de coste. La función de estos algoritmos es modificar los pesos de la red con el objetivo de que en el siguiente *batch* los resultados de la función de coste hayan disminuido, es decir, se busca minimizar la función de coste, siendo lo ideal encontrar un mínimo absoluto. La magnitud de descenso o de actualización de los pesos depende de un parámetro ajustable denominado *learning-rate*. Cuanto mayor sea el valor del *learning-rate*, mayor será el porcentaje de cambio de los pesos tras cada *batch*.

A partir del SGD se han desarrollado diversos algoritmos, denominados optimizadores, que permiten variar de forma dinámica el valor del *learning-rate*. Dentro de los optimizadores adaptativos clásicos están *Adagrad* [34] y *RMSprop* [35], que derivaron en *Adam* [36] (entre otros), que será el optimizador adaptativo que se describirá a continuación y que se ha utilizado en este trabajo.

2.7.2. *Adam*

Adam realiza una optimización de primer orden de los gradientes de la función a minimizar, para lo cual lleva a cabo una estimación de forma adaptativa de los momentos de primer y segundo orden. Los parámetros necesarios para el funcionamiento de *Adam* son el *learning-rate* (α), los ratios de caída del primer y segundo momento (β_1 y β_2), el vector de parámetros iniciales (θ_0) y un parámetro para prevenir posibles divisiones por cero (ϵ). La secuencia de pasos que realiza el *Adam* en cada iteración se muestran en el Algoritmo 2.1.

while θ_t *no converge* **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \Delta_\theta f_t(\theta_{t-1})$
  $m_t \leftarrow \beta_1\ m_{t-1} + (1 - \beta_1)\ g_t$
  $v_t \leftarrow \beta_2\ v_{t-1} + (1 - \beta_2)\ g_t^2$
  $\widehat{m}_t \leftarrow m_t/(1 - \beta_{1,t})$
  $\widehat{v}_t \leftarrow v_t/(1 - \beta_{2,t})$
  $\theta_t \leftarrow \theta_{t-1} - \alpha\ \widehat{m}_t/(\sqrt{\widehat{v}_t} + \epsilon)$
return θ_t

Algoritmo 2.1: Secuencia de pasos del optimizador *Adam*.

Valores recomendables para inicializar *Adam* son $\alpha = 0{,}001$, $\beta_1 = 0{,}9$, $\beta_2 = 0{,}999$ y $\epsilon = 10^{-8}$. Respecto a las ventajas de este optimizador cabe destacar su sencillez de implementación, su eficiencia computacional y su buen funcionamiento cuando trabaja con numerosos datos y parámetros. También es importante destacar que *Adam* presenta algunos inconvenientes, como por ejemplo su ineficacia cuando el tamaño del *batch* es pequeño.

2.8. Funciones de coste

Como se ha dicho anteriormente, la función de coste se utiliza para comparar la salida de la red con la salida deseada durante el entrenamiento, concretamente para medir la diferencia entre ambas. De esta forma, si el resultado de la función de coste es nulo, significa que la salida obtenida es exactamente la deseada. De forma general, las funciones de coste, cuando hablamos de aprendizaje supervisado, pueden clasificarse en dos grupos, en función del tipo de problema a resolver: regresión o clasificación.

Dentro del grupo de funciones de coste enfocadas a problemas de regresión, una de las más importantes es la función *Mean Square Error* (MSE) o error cuadrático medio, cuya expresión se muestra en la Ecuación 2.25, donde y representa la salida de la red e $\widehat{y}$ la salida deseada o *ground-truth*. Por ser una función cuadrática el MSE penaliza fuertemente los errores de gran magnitud frente a errores pequeños y además, tiene únicamente en cuenta el valor absoluto del error y no su signo.

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2}{n} \tag{2.25}$$

Otra función a destacar es el *Mean Absolute Error* (MAE) o error absoluto medio, cuya expresión se muestra en la Ecuación 2.26. Al igual que el MSE no tiene en cuenta el signo del error. Por otra parte, como no es una función cuadrática, es más robusta frente a *outliers* (valores de error puntualmente altos) que el MSE.

$$MAE = \frac{\sum_{i=1}^{n}|y_i - \widehat{y}_i|}{n} \tag{2.26}$$

Dentro de las funciones de coste enfocadas a problemas de clasificación se encuentra la función *Cross Entropy* o entropía cruzada. Su expresión se muestra en la Ecuación 2.27. Una de sus principales características es que penaliza fuertemente aquellas predicciones muy seguras pero incorrectas.

$$Cross\ Entropy = -(y_i\ log(\widehat{y}_i) + (1 - y_i)\ log(1 - \widehat{y}_i)) \tag{2.27}$$

2.9. Conclusiones

Los aspectos explicados en este capítulo permiten obtener una idea global de los diferentes aspectos relativos a las CNN, así como de los distintos elementos que las componen, su utilidad, sus funciones, etc. Con estos conocimientos es posible abordar el diseño y la implementación de un sistema basado en una CNN.

Por otra parte, los aspectos relativos a imágenes y sensores de profundidad son de necesario conocimiento para poder trabajar con ellos en el diseño del sistema.

Capítulo 3

Desarrollo

3.1. Introducción

En este Capítulo se describen todos los pasos seguidos en el diseño e implementación del sistema desarrollado. En primer lugar se plantea detalladamente el problema a resolver. Posteriormente se describe la arquitectura de la Red Neuronal utilizada. A continuación, se explica cómo se han preparado y estructurado los datos para su entrenamiento. En otro apartado se describen los pasos seguidos para llevar a cabo el entrenamiento y las diferentes fases de éste. Por último se añaden las conclusiones extraídas del desarrollo del trabajo.

3.2. Planteamiento del problema

Como se introdujo en el Capítulo 1, en este TFG se lleva a cabo el diseño, implementación y evaluación de un sistema de detección de personas en imágenes de profundidad basado en una Red Neuronal Convolucional.

En primer lugar, es importante describir las características de las imágenes sobre las que se debe realizar la detección, ya que dichas características condicionan en gran medida el funcionamiento del sistema y plantean los principales problemas a resolver. Como se trata de imágenes de profundidad su rango de medidas está bastante restringido, por ejemplo, como se ha explicado en el Capítulo 2.2.2, en el caso de la cámara Kinect II el rango de medidas válidas se encuentra entre 0.5 y 4.5 metros. Esto hace que la escena sobre la que se desea llevar a cabo la detección presente valores de distancia dentro de ese rango. Por otro lado, como las imágenes de profundidad presentan un tipo de ruido muy concreto y que depende de la tecnología utilizada, el sistema debe comportarse de forma robusta frente a dicho ruido.

Otra característica importante de las imágenes utilizadas es la orientación frontal elevada de la cámara, que genera la aparición de oclusiones, parciales o totales, frente a las cuales el sistema debe comportarse de forma suficientemente robusta. Sin embargo, el problema de las oclusiones no presenta una solución concreta y claramente definida. En primer lugar, resulta evidente que ante una persona totalmente ocluida es imposible que un sistema de detección que trabaja “imagen a imagen” (como es el caso de este TFG) detecte a dicha persona, siendo necesario implementar soluciones que tengan en cuenta información temporal (sistemas de seguimiento). Partiendo de la imposibilidad de detectar una persona totalmente ocluida, se plantea la cuestión de en qué momento considerar, en la práctica, que una oclusión pasa de ser parcial a ser total. Y en línea con este problema y más en relación con el sistema planteado, a la hora de

llevar a cabo el etiquetado de las imágenes para el entrenamiento, se plantea la duda de cuándo considerar que una oclusión es de tal magnitud que no debe etiquetarse a dicha persona. De esta forma, a la hora de evaluar el sistema, si una persona, debido a encontrarse notablemente ocluida, no ha sido etiquetada pero sin embargo sí ha sido detectada, aparece un error de detección (falso positivo) que no debería considerarse como tal. En el Capítulo 5 se comentan posibles soluciones a estos problemas. La Figura 3.1 muestra un ejemplo de una imagen RGB-D tomada con la cámara estéreo en posición frontal elevada. En la imagen de profundidad puede observarse el ruido que contiene, principalmente en los bordes y en zonas de escasa textura como zonas del suelo.

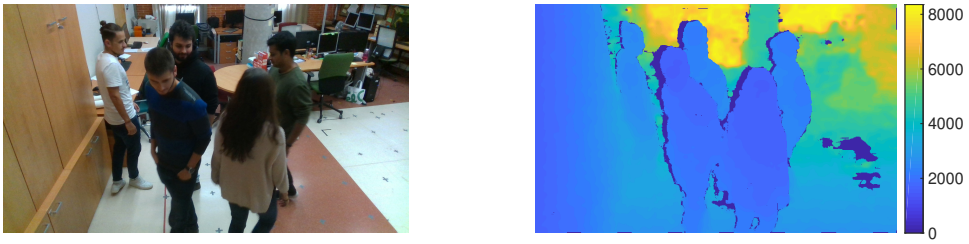


Figura 3.1: Ejemplo de imágenes obtenidas por un sensor RGB-D con posición frontal elevada de la cámara.

Respecto a la forma de realizar la detección, tal y como se ha mencionado en el párrafo anterior, el problema se ha planteado imagen a imagen, es decir, se aplica a cada imagen, de forma individual, el algoritmo de detección, obteniéndose un resultado exclusivo para la imagen en cuestión. De está forma, la detección realizada sobre una imagen determinada no influye de forma alguna sobre detecciones en las imágenes posteriores, y del mismo modo, no se ve influida por detecciones en imágenes previas. Dicho con otras palabras, no se tiene en cuenta para realizar la detección ningún tipo de información temporal. Por ello, la red neuronal que implementa el sistema de detección se basa en capas convolucionales en dos dimensiones, capaces de procesar imágenes individuales, y no en otros tipos de estructuras como capas convolucionales de tres dimensiones, capaces de procesar grupos de imágenes, o redes LSTM, que tienen en cuenta información temporal.

Por último, otro aspecto a definir antes de diseñar propiamente el sistema de detección, es la forma en la que se quiere que el sistema muestre su salida, es decir, cómo el sistema debe entregar las detecciones realizadas. Para ello se ha optado por la utilización de un mapa de probabilidad de las mismas dimensiones que la imagen de entrada, en el cual cada persona detectada se identifica con una gaussiana cuya media se ubica en el píxel central de la cabeza de la persona. La Figura 3.2 muestra un ejemplo de mapa de probabilidad en el que se han efectuado cuatro detecciones. El valor de la desviación típica de las gaussianas se ha ajustado a un valor fijo, no dependiente de la distancia a la que se encuentra cada persona, calculado tal y como muestra la Ecuación 3.1, donde D representa el diámetro medio de una cabeza, valor estimado en 15 píxeles.

$$\sigma = \frac{D}{2{,}5} = \frac{15}{2{,}5} = 6 \tag{3.1}$$

3.3. Construcción de la Red Neuronal

En este apartado se describe la arquitectura de la red construida. Como se introdujo en el Capítulo 1, ésta se subdivide en dos bloques bien diferenciados: el *Bloque Principal* (BP) y el *Bloque de Refuerzo*

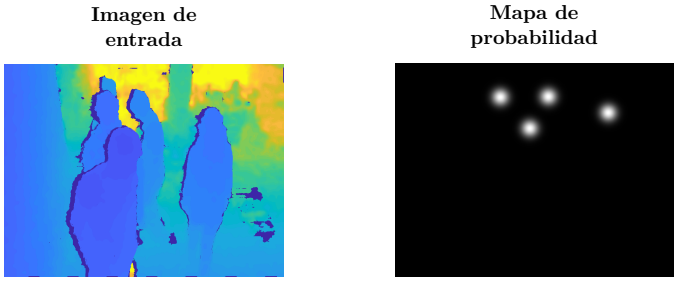


Figura 3.2: Ejemplo de una mapa de probabilidad en el que se indican cuatro detecciones

de Hipótesis (BR). La entrada del primero es la entrada del sistema, mientras que la entrada del segundo es la entrada del sistema concatenada con la salida del BP. De esta forma, el mapa de probabilidad entregado por el BP es nuevamente procesado por el BR que, a partir de éste y de la imagen de entrada, entrega un nuevo mapa de probabilidad refinado, en el que detecciones correctas de muestran de forma más clara y pequeñas detecciones incorrectas son eliminadas.

Uno de los grandes problemas de las CNNs es que a medida que se aplican filtrados a través de las capas convolucionales, éstas pierden cierta información, que se trata de recuperar a través de recirculaciones o capas residuales, por ello el uso de un nuevo bloque de refuerzo de hipótesis no es redundante ya que nos encontramos ante una situación en la cual el BP nos entrega una inicialización de la detección que permite al BR reutilizar esta información y definir mejor las fronteras de probabilidad de la detección. Este bloque es útil especialmente en los casos de oclusiones o detecciones ruidosas, aportando esa estabilidad que el bloque principal no fue capaz de conceder.

Internamente cada uno de los bloques se estructura siguiendo una configuración del tipo *encoder-decoder* y haciendo uso de sub-bloques residuales similares a los explicados en la sección 2.4. La Figura 3.3 muestra un esquema de la conexión de los bloques BP y BR.

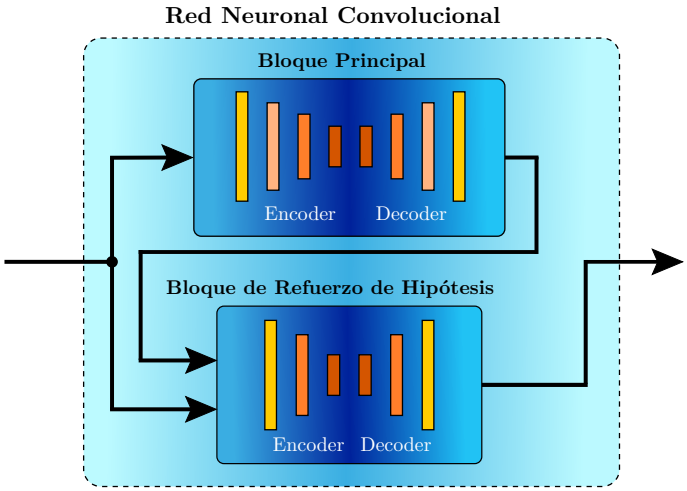


Figura 3.3: Esquema general de la Red Neuronal construida

En las Tablas 3.1 y 3.2 se detalla la arquitectura interna del BP y del BR respectivamente. Para cada capa de la red se indica el tipo de operación que realiza, el tamaño de la imagen de salida y los diferentes parámetros configurables. Como se observa en dichas tablas, la red construida utiliza internamente, tanto en el BP como en el BR, cuatro tipos de sub-bloques residuales diferentes: *Bloque Convolucional*

Codificador (BCC), *Bloque Convolucional Decodificador* (BCD), *Bloque Identidad Codificador* (BIC) y *Bloque Identidad Decodificador* (BID). Las estructuras internas de cada uno de ellos se muestran en las Figuras 3.4, 3.5 y 3.6. Los parámetros *a*, *b*, *c*, *strides* y *kernel*, se indican tanto en las tablas como en las figuras, con el objetivo de clarificar su significado. Como se observa, *a*, *b* y *c* representan el número de filtros de las capas (1), (2) y (3), y en el caso de los sub-bloques convolucionales *c* representa el número de filtros también de la capa (4), necesario para que las entradas del sumador tengan dimensiones idénticas. El valor de *strides* indicado se aplica a las capas (1) y (4), siendo siempre [1,1] para las capas (2) y (3). Del mismo modo, el tamaño de *kernel* indicado se aplica a la capa (2), utilizándose siempre un kernel 1×1 para el resto de capas. El uso de *kernels* de dimensiones 1×1, es una forma de proyectar el número de filtros a la entrada de la capa en el número de filtros deseados, obteniendo una representación simplificada de los filtros contenidos en la matriz de entrada a costa de perder una parte de la energía de dichos filtros. Además, en el caso de los sub-bloques codificadores se utilizan Convoluciones Separables, mientras que los bloques decodificadores utilizan Convoluciones Traspuestas, todas ellas bidimensionales. Esto es así porque el *encoder* debe extraer las características de la imagen, tarea en la que las Convoluciones Separables son muy eficientes, mientras que el *decoder* únicamente procesa las características extraídas por el *encoder* y configura el mapa de probabilidad. Por último, es importantes destacar que tras cada capa se aplica Normalización del *batch* y una activación ReLU, siendo la última a continuación del sumador, y no antes, ya que lo interesante es sumar las salidas de las distintas capas y tener como resultado una activación final del bloque, y no una suma de activaciones.

En resumen, la red consta de dos bloques, BP y BR, que siguen una estructura interna de tipo *encoder-decoder*. La etapa de *encoder* hace uso de sub-bloques BCC y BIC, mientras que la etapa de *decoder* utiliza sub-bloques BCD y BID. Tanto en el BP como en el BR se utiliza al comienzo del *encoder* una capa de *Max Pooling* y tras cada capa convolucional, una capa de Normalización del *batch* y una activación ReLU, excepto la última activación de cada bloque, que se trata de una función Sigmoidal. En el caso de la etapa de *decoder* se hace uso de operaciones de *UpSampling* para aumentar las dimensiones de la imagen y de operaciones de *Cropping* para ajustar éstas. En el caso del BP el tamaño de la matriz a las salida del *encoder* es de $10 \times 14 \times 1024$ y en el caso del BR es de $20 \times 27 \times 512$. Ambos bloques entregan a su salida un mapa de probabilidad de dimensiones $240 \times 320 \times 1$.

3.4. Entrenamiento

Para llevar a cabo el entrenamiento de la red, se ha optado por la utilización de datos sintéticos fotorrealistas, generados mediante un *software* de simulación. Las principales ventajas de utilizar datos sintéticos son la facilidad para crear una base de datos de gran tamaño y la posibilidad de etiquetarla de forma automática. Dicho etiquetado consiste en crear los mapas de probabilidad que debería entregar la red, lo cual es una tarea tediosa en el caso de tener que llevarla a cabo de forma manual.

Sin embargo, las imágenes sintéticas no presentan las mismas características que las imágenes tomadas con un sensor real. Por ejemplo no presentan ruido, contienen figuras, objetos y personas demasiado simples, etc. Obviamente, en función de la calidad de las imágenes sintéticas y del simulador utilizado, estos problemas pueden minimizarse y obtener datos muy similares a los que obtendría un sensor real.

Teniendo en cuenta estas cuestiones, se ha dividido el entrenamiento de la red en dos partes. En una primera parte, se ha utilizando una base de datos sintéticos, con un elevado número de imágenes, etiquetadas automáticamente. En una segunda parte se ha empleado un pequeño conjunto de datos reales y etiquetados manualmente. De esta forma se persigue el objetivo de que durante el entrenamiento con datos sintéticos la red aprenda a llevar a cabo la tarea de detección y con el entrenamiento con datos

reales perfeccione su funcionamiento, de cara a su posterior uso con datos reales. Dicho de otra manera, el entrenamiento con datos reales afina la detección, reutilizando lo aprendido con datos sintéticos y portándolo a un escenario más realista. La principal ventaja de esta distribución del entrenamiento es que solamente es necesario etiquetar manualmente una pequeña cantidad de datos reales.

Bloque Principal (BP)		
Capa	**Tamaño de salida**	**Parámetros**
Entrada	$240 \times 320 \times 1$	-
Capa Convolucional 2D	$120 \times 160 \times 64$	*kernel*=(7, 7) / *strides*=(2, 2) nº de filtros=64
Normalización del *batch*	-	
Activación	ReLU	
Max Pooling	$40 \times 53 \times 64$	*size*=(3, 3)
BCC	$40 \times 53 \times 256$	*kernel*=(3, 3) / *strides*=(1, 1) (a=64, b=64, c=256)
BIC	$40 \times 53 \times 256$	*kernel*=(3, 3) (a=64, b=64, c=256)
BCC	$20 \times 27 \times 512$	*kernel*=(3, 3) / *strides*=(2, 2) (a=128, b=128, c=512)
BIC	$20 \times 27 \times 512$	*kernel*=(3, 3) (a=128, b=128, c=512)
BCC	$10 \times 14 \times 1024$	*kernel*=(3, 3) / *strides*=(2, 2) (a=256, b=256, c=1024)
BIC	$10 \times 14 \times 1024$	*kernel*=(3, 3) (a=256, b=256, c=1024)
BIC	$10 \times 14 \times 1024$	*kernel*=(3, 3) (a=256, b=256, c=1024)
BCD	$10 \times 14 \times 256$	*kernel*=(3, 3) / *strides*=(1, 1) (a=1024, b=1024, c=256)
BID	$10 \times 14 \times 256$	*kernel*=(3, 3) (a=1024, b=1024, c=256)
BCD	$20 \times 28 \times 128$	*kernel*=(3, 3) / *strides*=(2, 2) (a=512, b=512, c=128)
BID	$20 \times 28 \times 128$	*kernel*=(3, 3) (a=512, b=512, c=128)
BCD	$40 \times 56 \times 64$	*kernel*=(3, 3) / *strides*=(2, 2) (a=256, b=256, c=64)
Cropping	$40 \times 54 \times 64$	*crop*=[(0, 0) (1, 1)]
BID	$40 \times 54 \times 64$	*kernel*=(3, 3) (a=256, b=256, c=64)
Up Sampling	$120 \times 162 \times 64$	*size*=(3, 3)
Capa Convolucional Traspuesta 2D	$240 \times 320 \times 1$	*kernel*=(3, 3) / *strides*=(1, 1) nº de filtros=1
Cropping	$240 \times 320 \times 64$	*crop*=[(0, 0) (2, 2)]
Normalización del *batch*	-	
Activación	ReLU	
Capa Convolucional Traspuesta 2D	$240 \times 320 \times 1$	*kernel*=(3, 3) / *strides*=(1, 1) nº de filtros=1
Activación	Sigmoidal	
Salida	$240 \times 320 \times 1$	-

Tabla 3.1: Arquitectura del Bloque Principal

Bloque de Refuerzo de Hipótesis (BR)		
Capa	**Tamaño de salida**	**Parámetros**
Entrada	240 × 320 × 2	-
Capa Convolucional 2D	120 × 160 × 64	*kernel*=(7, 7) / *strides*=(2, 2) nº de filtros=64
Normalización del *batch*	-	
Activación	ReLU	
Max Pooling	40 × 53 × 64	*size*=(3, 3)
BCC	40 × 53 × 256	*kernel*=(3, 3) / *strides*=(1, 1) (a=64, b=64, c=256)
BCC	20 × 27 × 512	*kernel*=(3, 3) / *strides*=(2, 2) (a=128, b=128, c=512)
BIC	20 × 27 × 512	*kernel*=(3, 3) (a=128, b=128, c=512)
BCD	40 × 54 × 128	*kernel*=(3, 3) / *strides*=(2, 2) (a=512, b=512, c=128)
BID	40 × 54 × 128	*kernel*=(3, 3) (a=512, b=512, c=128)
BCD	80 × 108 × 64	*kernel*=(3, 3) / *strides*=(2, 2) (a=256, b=256, c=64)
Up Sampling	240 × 324 × 64	*size*=(3, 3)
Cropping	240 × 320 × 64	*crop*=[(0, 0) (2, 2)]
Capa Convolucional Traspuesta 2D	240 × 320 × 1	*kernel*=(3, 3) / *strides*=(1, 1) nº de filtros=1
Normalización del *batch*	-	
Capa Convolucional Traspuesta 2D	240 × 320 × 1	*kernel*=(3, 3) / *strides*=(1, 1) nº de filtros=1
Activación	Sigmoidal	
Salida	240 × 320 × 1	-

Tabla 3.2: Arquitectura del Bloque de Refuerzo de Hipótesis

3.4.1. Preparación de las bases de datos

Como se ha explicado, para el entrenamiento completo de la red, se han utilizado dos bases de datos, la primera de ellas de datos sintéticos y la segunda de datos reales.

La base de datos sintéticos consta de 22000 imágenes que simulan haber sido tomadas con una posición frontal elevada de la cámara. El *software* de simulación uilizado ha sido Blender [37]. La escena simulada en las imágenes muestra una sala en la que caminan personas en diferentes direcciones. La perspectiva de la cámara no es fija, sino que realiza un giro de 360 grados, con respecto al eje central de la habitación, a lo largo de toda la base de datos, lo cual permite evitar un fondo constante que sería aprendido por la CNN durante el entrenamiento. El hecho de emplear diferentes fondos alrededor de toda la sala sintética, permiten que la CNN asocie el fondo a ruido y solo se centre en las personas que aparezcan en la imagen, inmunizando la red y no atándola a unas condiciones de montaje y perspectiva exclusivas. La Figura 3.7 muestra diferentes perspectivas de la sala sintética en el entorno de simulación de Blender. Las imágenes son de dimensiones 240 × 320 y codificadas en 16 bits sin signo. La Figura 3.8 muestra la apariencia de las imágenes de la base de datos en tres perspectivas diferentes.

El *sofware* Blender permite generar, además de las imágenes de profundidad sintéticas, unos mapas indicando la posición de la cabeza de cada persona. Estos mapas, mediante un *script* desarrollado en MATLAB, se transforman en los mapas de probabilidad con las características requeridas.

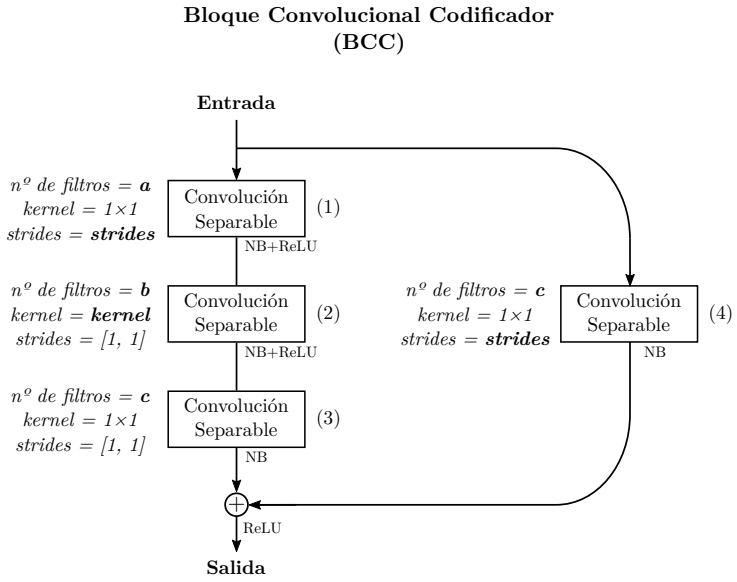


Figura 3.4: Esquema interno de un BCC

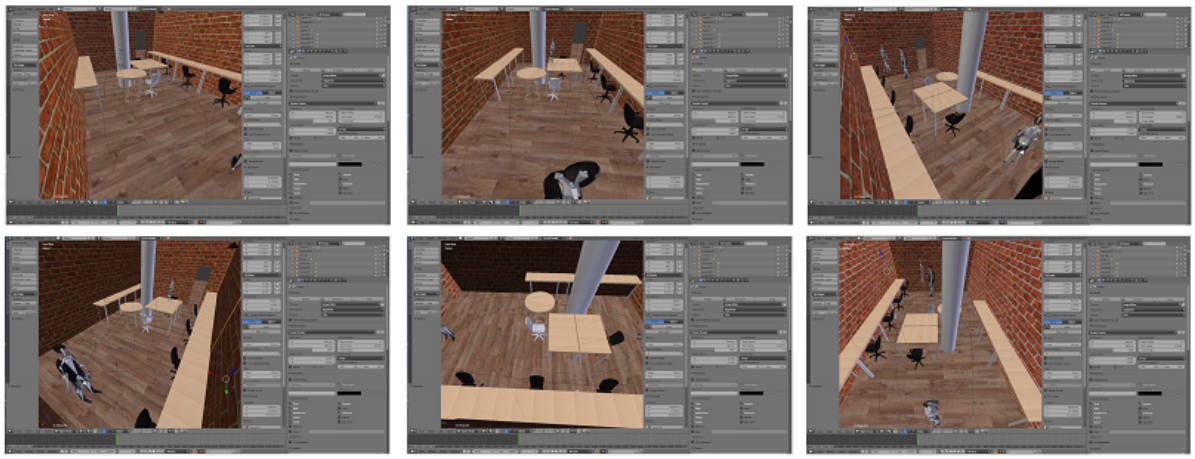


Figura 3.7: Diferentes perspectivas de la sala simulada en Blender

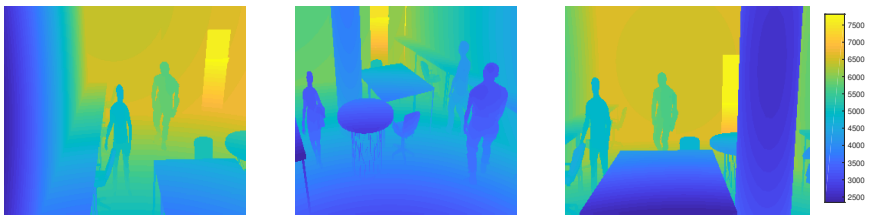


Figura 3.8: Imágenes de profundidad sintéticas

En cuanto a los datos reales para el entrenamiento, se ha utilizado una secuencia de tan solo 1700 imágenes, lo que corresponde a aproximadamente 1 minuto de vídeo a 30 fps. Los datos han sido grabados con una cámara estéreo en posición frontal elevada y fija, modelo *RealSense* D435 [20]. La escena grabada muestra una sala en la que caminan personas en diferentes direcciones. Las imágenes son de dimensiones 480×640 y codificadas en 16 bits sin signo, por lo que ha sido necesario reducir sus dimensiones a la mitad, utilizando interpolación bilineal, y conservando su relación de aspecto. La Figura 3.9 muestra

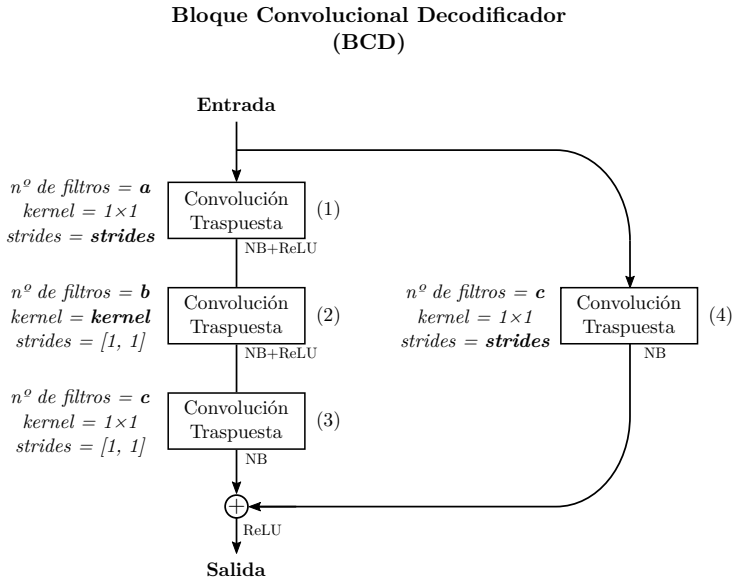


Figura 3.5: Esquema interno de un BCD

tres ejemplos de imágenes de la secuencia utilizada para el entrenamiento con datos reales. Como puede observarse, las imágenes reales presentan ruido, inexistente en las imágenes sintéticas.

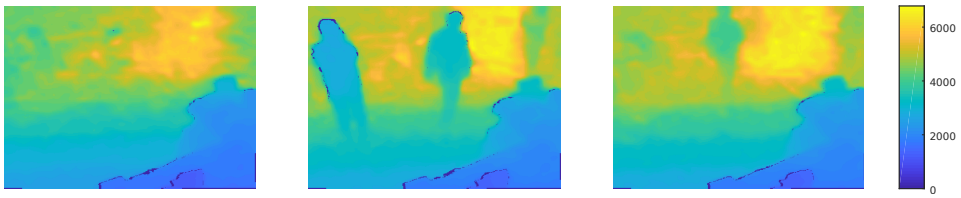


Figura 3.9: Imágenes de profundidad reales

Un aspecto importante del etiquetado de los mapas de probabilidad, tanto con datos sintéticos como con reales, es el relativo a la superposición de gaussianas. Cuando dos cabezas se encuentran muy próximas o solapadas, las gaussianas anotadas sobre cada cabeza no se suman entre sí, sino que prevalece el valor mayor de ambas. Con ello se consigue que siempre exista separación aparente entre las gaussianas, de cara a que la CNN aprenda a generar de esta forma la salida, facilitando la posterior identificación de las personas detectadas.

Cabe mencionar que inicialmente, el primer entrenamiento del sistema no se realizó utilizando el sistema de etiquetado de gaussianas anteriormente descrito, sino que las gaussinas superpuestas se sumaban entre sí. Ante los malos resultados cuando dos cabezas se encontraban muy próximas, se diseñó el nuevo sistema de etiquetado que fue perfectamente aprendido por la red, de forma que se obtuvieron mejores resultados. La Figura 3.11 muestra una comparación entre los dos sistemas de etiquetado para una misma imagen de entrada, siendo evidente el mejor funcionamiento en el segundo caso. Por ello, en adelante todo lo relativo al entrenamiento del sistema se ha llevado a cabo con el *ground-truth* de gaussianas fácilmente separables (2º caso).

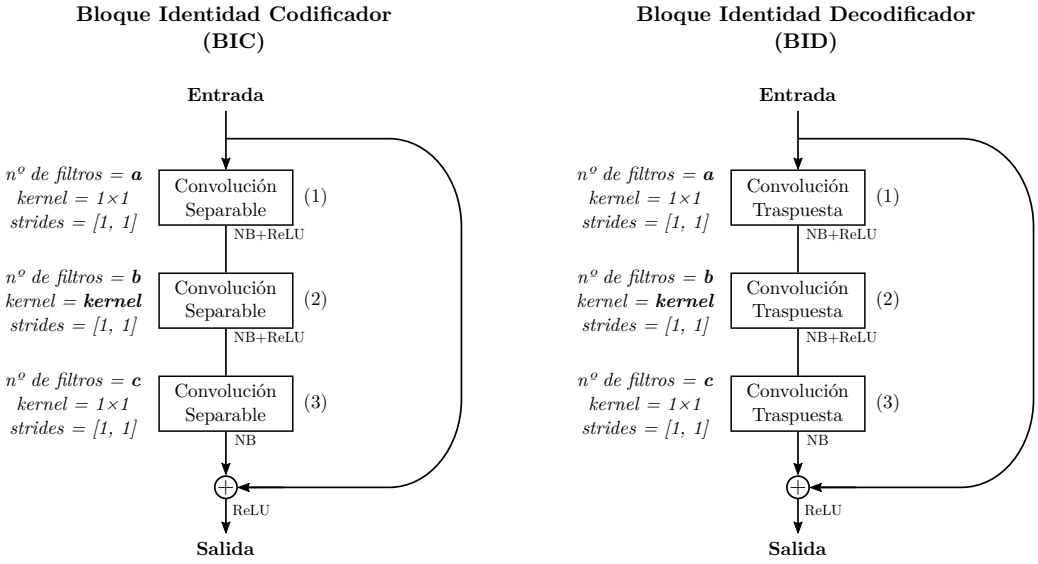


Figura 3.6: Esquemas internos de un BIC y un BID

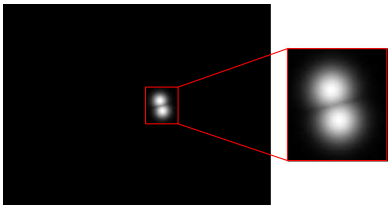


Figura 3.10: Detalle de gaussianas etiquetadas

3.4.2. Entrenamiento con datos sintéticos

Para el entrenamiento con datos sintéticos se ha dividido la base de datos de 22000 imágenes en 2 conjuntos. El primero, formado por 19800 imágenes, es el conjunto de entrenamiento (un 90 % del total de datos), es decir, aquellas imágenes que, junto con el mapa de probabilidad etiquetado, utiliza la red para ajustar los pesos. El segundo conjunto, formado por 2200 imágenes (un 10 % del total), es el conjunto de validación. Las imágenes de validación se utilizan para verificar el correcto progreso del entrenamiento, calculando al finalizar cada época el error del sistema sobre estas imágenes, que no han sido utilizadas para modificar los pesos.

Los valores de los parámetros configurables relativos al entrenamiento y al optimizador utilizado (*Adam*) se enumeran a continuación:

- Tamaño del *batch*: 15 imágenes
- *Learning Rate*: 0.001
- β_1: 0.9
- β_2: 0.999
- Función de coste: MSE

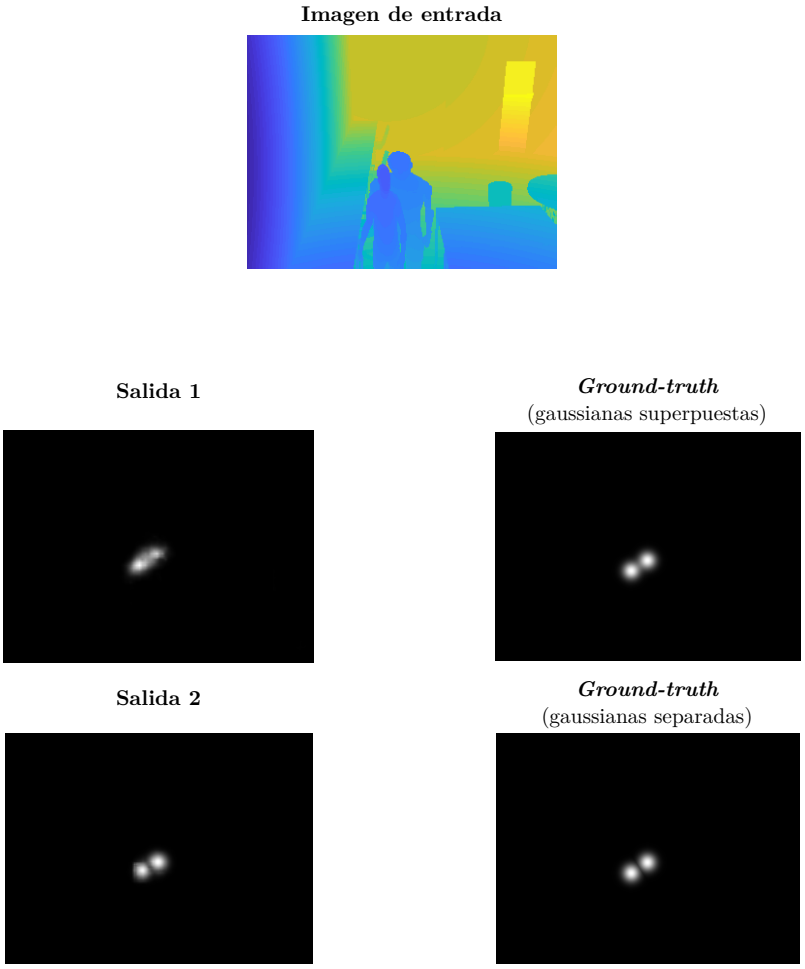


Figura 3.11: Comparación de salida cuando dos cabezas se encuentran muy próximas. La salida 1 corresponde a un primer entrenamiento con el *ground-truth* indicado a la derecha que contiene gaussianas superpuestas. La salida 2 corresponde al posterior entrenamiento tras diseñar el sistema de etiquetado que utiliza gaussianas fácilmente separables.

Además, se ha utilizado la técnica de *early stopping* para evitar sobre-entrenamiento. Este mecanismo consiste en memorizar el valor de precisión obtenido al finalizar cada época. Si éste valor supera al valor memorizado en la época anterior significa que durante la última época el sistema ha mejorado y por tanto se guarda el nuevo valor de todos los pesos. Si por el contrario, el valor de precisión obtenido es inferior al último memorizado, significa que el sistema no ha mejorado durante la última época y los valores de los pesos no son actualizados.

3.4.3. Entrenamiento con datos reales

Para el entrenamiento con datos reales se han dividido las 1700 imágenes que forman la secuencia utilizada en 2 conjuntos, entrenamiento y validación, que al igual que en el caso de datos sintéticos corresponden a un 90 % y un 10 % del total de imágenes, respectivamente.

Todos los aspectos, parámetros y técnicas utilizadas en el entrenamiento con datos sintéticos se han aplicado de igual forma en el caso del entrenamiento con datos reales.

3.5. Conclusiones

En primer lugar, en la construcción del sistema se han tenido en cuenta todos los aspectos teóricos estudiados previamente. Como resultado, se ha construido un sistema de detección complejo, basado en arquitecturas, elementos y técnicas relativas a las CNNs más recientes, como los bloques residuales, las capas de Normalización del *batch* o la utilización de un bloque de refuerzo de hipótesis. En resumen, la configuración de la red construida sigue una estructura para nada trivial.

Por otro lado, se ha propuesto una solución a uno de los principales problemas del entrenamiento de redes neuronales, que es la necesidad de disponer de numerosos datos etiquetados. En este trabajo se ha realizado un entrenamiento con datos generados y etiquetados por un simulador, solo siendo necesario etiquetar manualmente un pequeño conjunto de datos reales, utilizados para completar el entrenamiento.

En el Capítulo 4, dedicado a la validación del sistema, se describe el conjunto de datos de test utilizado para extraer resultados y se presentan éstos.

Capítulo 4

Resultados

4.1. Introducción

En este capítulo se exponen los resultados obtenido tras llevar a cabo la evaluación del sistema desarrollado, además de todos los pasos de los que ha constado dicha evaluación. Estos resultados permiten validar el sistema de detección.

En primer lugar, se describen con detenimiento todos los datos que componen el conjunto de test, utilizado para la extracción de resultados. Es muy importante conocer las características de estos datos, de cara a una correcta interpretación de los resultados obtenidos.

En segundo lugar, se enumeran las distintas métricas que se ofrecen como resultados, añadiendo una breve descripción del significado de cada una.

En tercer lugar, se dedica una sección a describir con detalle el algoritmo diseñado para obtener los resultados, es decir, cómo se han obtenido las métricas partiendo de los datos etiquetados (*ground-truth*) y de las salidas entregadas por el sistema.

El cuarto lugar, se ofrecen los resultados obtenidos. Para validar también el sistema de entrenamiento llevado a cabo y descrito en el Capítulo 3, se han extraído resultados en primer lugar del sistema entrenado únicamente con datos sintéticos y posteriormente del sistema tras completar el entrenamiento con el pequeño conjunto de datos reales.

Por último, en una sección de conclusiones se analizan los resultados experimentales obtenidos.

4.2. Descripción de los datos de test

El conjunto de datos de test está dividido en tres subconjuntos diferenciados, en función de la base de datos a la que pertenecen las imágenes. A continuación se describen estos tres subconjuntos:

- **Datos sintéticos (2200 *frames*).** Se han utilizado para extraer resultados 2200 imágenes de la base de datos sintéticos utilizada para entrenar el sistema. Sin embargo, son imágenes no utilizadas en dicho entrenamiento, es decir, que no han pasado antes por la red. Sus dimensiones son de 240×320 píxeles codificadas en 16 *bits* sin signo, y no requieren pre-procesado alguno para ser entregadas a la red. Una descripción mas extensa de las características de estas imágenes se encuentra en el apartado 3.4.1, junto con las Figuras 3.7 y 3.8. Únicamente se han extraído resultados utilizando

estos datos del sistema entrenado con datos sintéticos, y no tras completar el entrenamiento con datos reales.

- **Datos reales (base de datos propia, 1837 *frames*).** Se han utilizado también dos secuencias que forman un total de 1837 imágenes, pertenecientes a la misma base de datos que los datos reales utilizados en el entrenamiento. Esta base de datos, que utiliza el sensor *RealSense* D435 [20], se hará pública próximamente. Las imágenes tienen dimensiones de 480×640 píxeles y contienen mucho ruido, por lo que ha sido necesario ajustar su tamaño al de la entrada de la red, así como realizar un pre-procesado que elimina los píxeles nulos propios del ruido de bordes. El cambio de tamaño se ha realizando utilizando interpolación bilineal y en este caso no ha supuesto un cambio en la relación de aspecto, pues ambas dimensiones se han reducido a la mitad. Además, la escena que se muestra contiene gran cantidad de oclusiones, muchas de ellas de larga duración, pues aparecen diferentes personas formando filas o colas. Estas características, pese a haber sido utilizada en el entrenamiento, convierten a la base de datos en un escenario difícil para llevar a cabo la detección. En la Figura 4.1 pueden verse tres ejemplos de imágenes de esta base de datos.

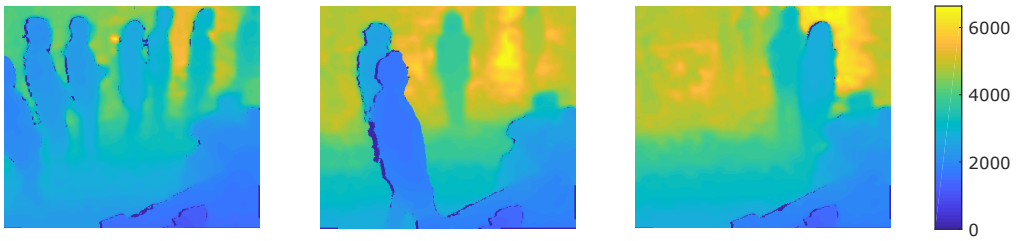


Figura 4.1: Ejemplo de imágenes utilizadas para test, pertenecientes a la base de datos del grupo.

- **Base de datos EPFL-LAB [6] (950 *frames*).** Esta base de datos se utiliza únicamente para extraer resultados, es decir, no ha sido procesada por la red durante el entrenamiento. Está formada por 950 imágenes tomadas con un sensor Kinect II [38], de dimensiones 424×512 y codificadas en 16 bits sin signo. Ha sido necesario un pre-procesado, tanto para ajustar las dimensiones de la imagen a la capa de entrada de la red, como para eliminar los píxeles nulos que contenían (fondo y márgenes). En este caso, el cambio de dimensiones supone un pequeño cambio en la relación de aspecto, pero que puede considerarse aceptable. El tipo de interpolación utilizada ha sido nuevamente la bilineal. Pueden observarse tres ejemplos de imágenes pertenecientes a esta base de datos en la Figura 4.2, antes y después del pre-procesado. Respecto a las etiquetas, la base de datos incorporaba información sobre los usuarios en formato *bounding-box*, es decir, aportaba las coordenadas de los rectángulos que encierran a cada uno de las personas que aparecen en las imágenes. Para poder extraer resultados comparando estas etiquetas con la salida de la red, ha sido necesario transformar el formato *bounding-box* en un punto (x, y) sobre la cabeza de cada persona. La ubicación de dicho punto se ha estimado de la siguiente forma: la coordenada x se ha ubicado en la mitad del ancho del rectángulo, mientras que la coordenada y se ha ubicado en un 10 % del alto del rectángulo por debajo de la parte superior de éste. Al tratarse de una estimación, en ocasiones el centroide de la cabeza ha podido ser ubicado en un píxel perteneciente al fondo, generando problemas en la extracción de resultados, como se explicará más adelante.

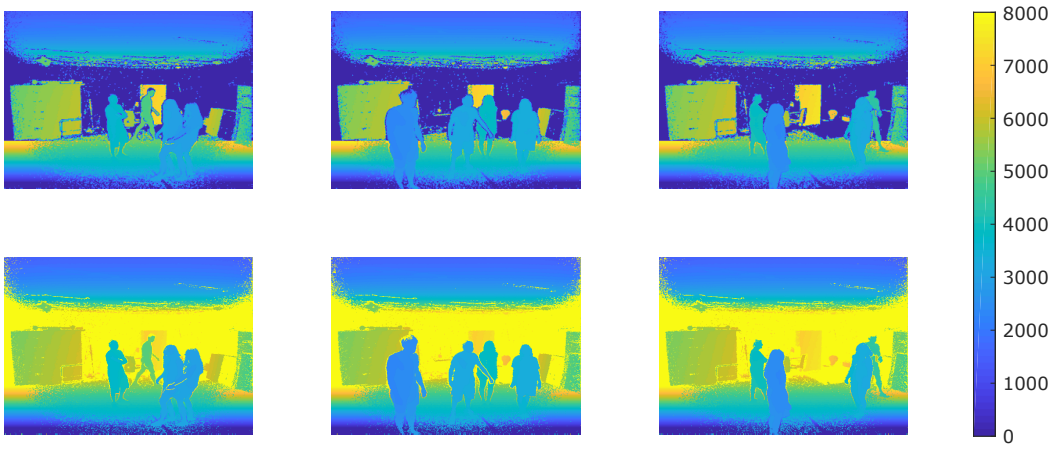


Figura 4.2: Ejemplo de imágenes utilizadas para test, pertenecientes a la base de datos EPFL-LAB [6]. En la fila superior las imágenes originales, en la fila inferior las imágenes pre-procesadas.

4.3. Métricas de calidad

Las métricas utilizadas para validar el sistema y cuyos valores se muestran en el Apartado 4.5 se describen a continuación:

- **Falsos Negativos (FN):** error en la detección que consiste en no detectar a una persona que debería ser detectada. Son aquellos puntos del *ground-truth* para los que no existe correspondencia en el mapa de probabilidad entregado por la red.
- **Falsos Positivos (FP):** error en la detección que consiste en efectuar una detección donde no existe persona. Son aquellos puntos del mapa de probabilidad entregado por la red para los cuales no existe correspondencia en el *ground-truth*.
- **Verdaderos Positivos (VP):** en inglés *True Positives* (TP). Acierto en la detección. Son aquellos puntos del mapa de probabilidad entregado por la red para los cuales existe correspondencia en el *ground-truth*.
- **Error:** suma de todos los errores de detección cometidos. Se calcula como FP + FN.
- ***Precision*:** indica cómo de preciso es el sistema, es decir, de todas las detecciones que efectúa, qué porcentaje son correctas. La Ecuación 4.1 muestra la forma en que se calcula.

$$Precision = \frac{VP}{VP + FP} \tag{4.1}$$

- ***Recall*:** indica la sensibilidad del sistema, es decir, el porcentaje de detecciones efectuadas respecto al total de detecciones que deberían haberse efectuado. Su cálculo se muestra en la Ecuación 4.2.

$$Recall = \frac{VP}{VP + FN} \tag{4.2}$$

Los parámetros FP, FN, VP y Error se indican tanto en valor absoluto como en porcentaje respecto al total de puntos etiquetados, es decir, el total de personas en el *ground-truth*. La Figura 4.3 muestra el valor de las métricas anteriormente descritas para una imagen de ejemplo.

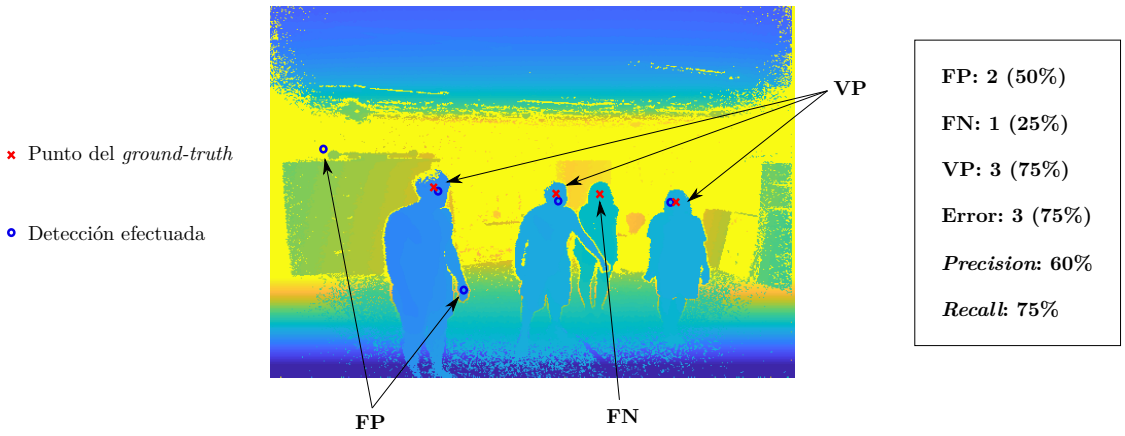


Figura 4.3: Valor de las métricas para una imagen de ejemplo, que contiene 4 puntos en el *ground-truth* y sobre la que se han efectuado 5 detecciones, cometiendo diversos errores.

4.4. Descripción del proceso de extracción de resultados

En este apartado se explica con detalle cómo se ha llevado a cabo el proceso de obtención de las métricas descritas en el apartado anterior, es decir, cuál ha sido el algoritmo seguido para comparar los puntos etiquetados del *ground-truth* con los puntos del mapa de probabilidad entregado por la red y qué consideraciones se han llevado a cabo.

En primer lugar, se parte de tener, en el *ground-truth*, las coordenadas x y y de los centroides de todas las cabezas existentes en las imágenes a evaluar. Sin embargo, en los mapas de probabilidad generados por el sistema se parte de gaussianas, a partir de las cuales se debe obtener el centroide de las detecciones, que corresponde con la ubicación de la media de cada gaussiana. Para ello, en primer lugar, se *binariza* el mapa de probabilidad con un umbral fijado de forma empírica en 0.6, para después obtener el centro de los contornos segmentados. La Figura 4.4 muestra este proceso. Un valor de umbral alto facilita la separación de gaussianas próximas.

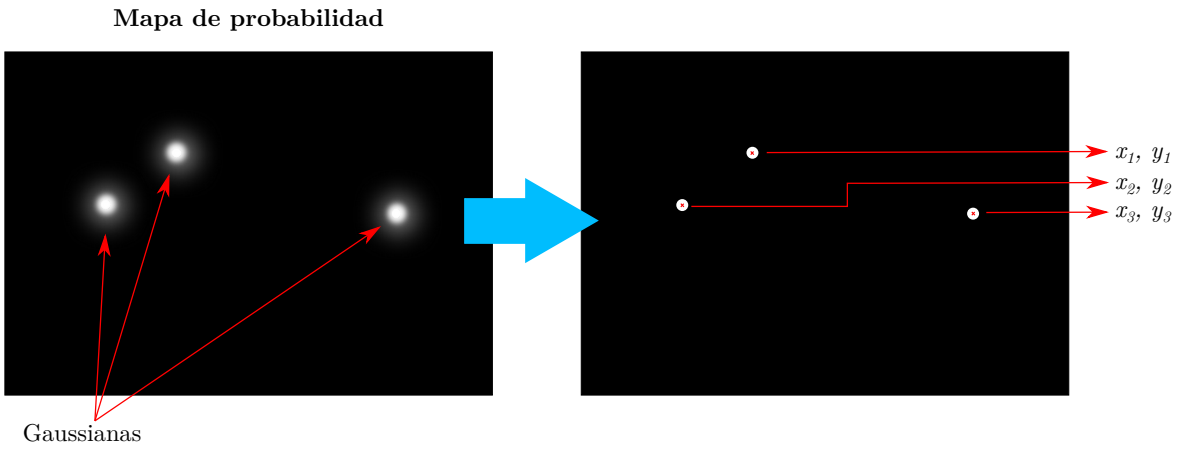


Figura 4.4: Esquema del proceso de obtención de los centroides de las detecciones a partir del mapa de probabilidad.

En segundo lugar debe determinarse el área de estudio, es decir, la zona de la imagen sobre la que se va a efectuar la validación. Este área de estudio tiene como objetivo prescindir de zonas ruidosas de las imágenes, como los márgenes. Además, una tercera dimensión del área de estudio indica el valor máximo de distancia a tener en cuenta, de esta forma, se ajusta la validación al rango válido de medida del sensor. La nomenclatura a seguir para definir el área de estudio es la siguiente: $[(p_x, p_y),\ (w, h),\ d_{máx}]$, donde p_x y p_y son las coordenadas del vértice superior izquierdo del área de estudio, w y h el ancho y el alto, respectivamente, y $d_{máx}$ el valor de distancia máxima.

Una vez se tiene, para cada imagen, un conjunto de puntos del *ground-truth* $(\widehat{x}, \widehat{y})$, y un conjunto de puntos extraídos de los mapas de probabilidad (x, y), y se han determinado las dimensiones del área de estudio, debe procederse a realizar la comparación y extraer el valor de las métricas. En los siguientes puntos se describen todos los pasos seguidos en este proceso, relativos a una imagen, pero que se extiende para todas las imágenes del conjunto de test.

- **Correspondencia entre puntos del *ground-truth* y detecciones (VP).** Este primer paso consiste en asociar cada punto del *ground-truth*, si es posible, con una detección. Cada punto del *ground-truth* se asocia con la detección más cercana, siempre que la distancia no sea mayor a un valor determinado (no confundir con el valor máximo de distancia del área de estudio, que es un valor de profundidad). Este valor se ha fijado en 37.5 píxeles, que corresponde a 2.5 veces el diámetro estimado de una cabeza (15 píxeles). Los puntos que no cumplan este requisito quedan sin asociar. Por el contrario, los puntos asociados serán aciertos, es decir, VP.
- **Cálculo de los FP y FN.** Los puntos del *ground-truth* que han quedado sin asociar en el paso anterior serán FN, y las detecciones que no han encontrado correspondencia en el *ground-truth* serán FP.
- **Filtrado del área de estudio.** Todos los valores calculados hasta aquí no han tenido en cuenta el área de estudio. Para aplicar este filtrado, en el caso de los FP y FN es inmediato, todos aquellos que se encuentren fuera del área de estudio se descartan. En el caso de los VP, es decir, asociaciones entre *ground-truth* y detecciones, el procedimiento seguido ha sido el siguiente: solo se han descartado aquellos VP en los cuales los dos puntos asociados se encuentran fuera del área de estudio, como solución a los casos límite en los que el punto del *ground-truth* se ubica fuera y la detección dentro, o viceversa.
- **Cálculo de las métricas restantes.** Una vez calculados los valores de FP, FN y VP, se calculan a partir de éstos Error, *Precision* y *Recall*, tal y como se explica en el apartado 4.3.

La Figura 4.5 muestra un ejemplo de los pasos anteriormente descritos, donde puede observarse la asociación de puntos, el cálculo de VP, FP y FN, y qué puntos se descartan tras aplicar el área de estudio.

4.5. Resultados experimentales

En este apartado se ofrece el valor numérico de todas las métricas calculadas, separadas por bases de datos e indicando para cada una número total de imágenes, el área de estudio, el número total de puntos en el *ground-truth* (dentro del área de estudio) y el valor de las métricas tanto en valor absoluto como en valor relativo.

Como se ha dicho anteriormente, el sistema se ha evaluado en primer lugar tras el entrenamiento únicamente con datos sintéticos y en segundo lugar tras el entrenamiento completo, incluyendo datos reales. De esta forma, posteriormente puede realizarse una comparativa y extraer mejores conclusiones.

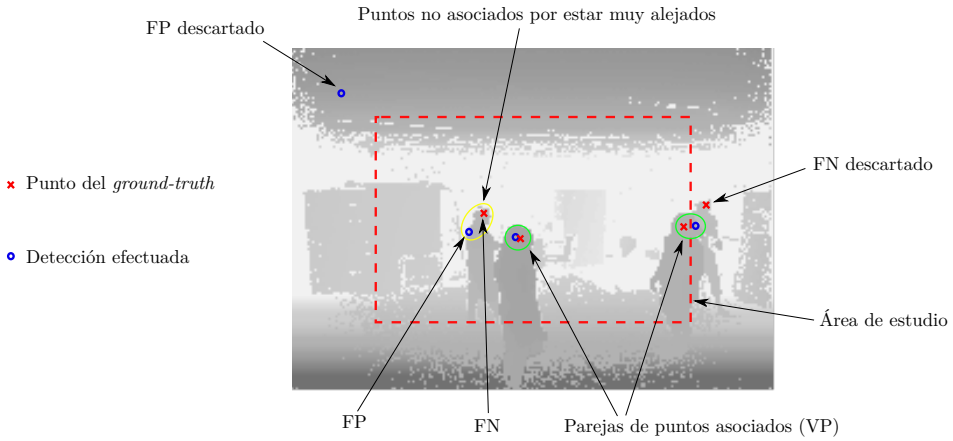


Figura 4.5: Ejemplo de extracción de las métricas para una imagen concreta.

4.5.1. Sistema entrenado únicamente con datos sintéticos

Para validar el sistema entrenado únicamente con datos sintéticos se han utilizado las tres bases de datos descritas en el apartado 4.2. Los resultados obtenidos se muestran en las Tablas 4.1, 4.2 y 4.3.

Área de Estudio	[(0, 0), (320, 240), inf]
Número de imágenes	2200
Puntos en el *ground-truth*	3176
VP	**2964 (93.32 %)**
FN	**212 (6.68 %)**
FP	**3 (0.09 %)**
Error (FN+FP)	**215 (6.77 %)**
Precision	**0.99**
Recall	**0.93**

Tabla 4.1: Resultados con datos sintéticos

Área de Estudio	[(30, 25), (260, 155), 5000 mm]
Número de imágenes	1837
Puntos en el *ground-truth*	1794
VP	**816 (45.48 %)**
FN	**978 (54.51 %)**
FP	**301 (16.78 %)**
Error (FN+FP)	**1279 (71.29 %)**
Precision	**0.73**
Recall	**0.45**

Tabla 4.2: Resultados con datos reales (base de datos propia)

4.5.2. Sistema entrenado con datos sintéticos y reales

Para validar el sistema tras completar el entrenamiento con los datos reales, se han utilizado las dos bases de datos reales descritas en el apartado 4.2, prescindiendo del conjunto de datos sintéticos. Los resultados obtenidos se muestran en las Tablas 4.4 y 4.5.

Área de Estudio	[(20, 55), (280, 150), 3500 mm]
Número de imágenes	950
Puntos en el *ground-truth*	1959
VP	**1485 (75.8 %)**
FN	**474 (24.19) %)**
FP	**3 (0.15 %)**
Error (FN+FP)	**477 (24.35 %)**
Precision	**0.99**
Recall	**0.76**

Tabla 4.3: Resultados con datos de EPFL-LAB [6]

Área de Estudio	[(30, 25), (260, 155), 5000 mm]
Número de imágenes	1837
Puntos en el *ground-truth*	1680
VP	**1421 (84.58 %)**
FN	**259 (15.41 %)**
FP	**1 (0.06 %)**
Error (FN+FP)	**260 (15.48 %)**
Precision	**0.99**
Recall	**0.85**

Tabla 4.4: Resultados con datos reales (base de datos propia)

4.5.3. Resultado de tasa de funcionamiento

Debido a la importancia del funcionamiento en tiempo real del sistema, comentada en el Capítulo 1, se indica aquí la tasa media de velocidad de funcionamiento del sistema, de valor 40 fps. El valor indicado se corresponde con la utilización del *hardware* especificado en el Capítulo 6 (procesador Intel i7-6700K multicore, 64 GB de RAM, GPU NVIDIA GeForce GTX 1080ti con 11 GB de RAM y sistema operativo Ubuntu 16.04).

Se considera que un valor de 40 fps permite al sistema ejecutarse al doble de velocidad de lo que puede considerarse tiempo real en una tarea de detección como la planteada en este trabajo.

4.6. Conclusiones

En este apartado se analiza, en base a los resultados anteriormente expuestos, el progreso del sistema desarrollado.

En primer lugar, tras entrenar únicamente con datos sintéticos, puede observarse cómo el sistema funciona de forma excelente ante datos de este tipo. No existen falsas detecciones (FP), alcanzando una precisión de prácticamente el 100 %. Sí existe un número considerable de FN, detecciones que deberían haberse efectuado y no lo han hecho. Si nos detenemos en analizar estos FN, puede comprobarse como han ocurrido siempre en situaciones de oclusiones. Además, es importante recordar que, al haberse realizado el etiquetado de forma automática, éste es completamente exhaustivo, es decir, están etiquetadas personas casi totalmente ocluidas, lo que facilita la aparición de FN. En resumen, en este punto y con datos sintéticos el sistema funciona de funciona de forma excelente salvo frente a oclusiones, donde baja su efectividad.

No ocurre lo mismo frente a datos reales. El sistema entrenado únicamente con datos sintéticos no es efectivo frente a datos reales. En función de las características de la base de datos se puede alcanzar

Área de Estudio	[(20, 55), (280, 150), 3500 mm]
Número de imágenes	950
Puntos en el *ground-truth*	1402
VP	**392 (27.96 %)**
FN	**1010 (72.04) %)**
FP	**0 (0 %)**
Error (FN+FP)	**1010 (72.04) %)**
Precision	**1**
Recall	**0.28**

Tabla 4.5: Resultados con datos de EPFL-LAB [6]

un cierto grado de efectividad, incluso llegando a un 100 % de precisión, como es el caso de los datos de EPFL-LAB. Sin embargo, con los datos reales de la base de datos propia, el número de FN supera al de VP, y los resultados son en general malos, al margen de que existan o no oclusiones.

Pasando ahora a analizar los resultados tras completar el entrenamiento, es importante recordar que los datos reales utilizados para ello han sido los de la base de datos propia y que lo que se perseguía era afinar lo aprendido con datos sintéticos para portarlo a un escenario real. Los resultados obtenidos nos demuestran, de forma general, cómo la red ha mejorado en gran medida frente a las imágenes de las misma base de datos utilizada para el entrenamiento, pero ha empeorado frente a as imágenes de EPFL-LAB. Esto nos demuestra que la red ha perdido generalización. En el entorno utilizado en el entrenamiento consigue unos resultados realmente buenos, con una precisión del 100 % (no existen FP) y una sensibilidad del 85 %. El número de FN, ahora sí, se debe prácticamente en la totalidad de los casos a situaciones de oclusiones, muy frecuentes en esta base de datos. En resumen, frente a datos reales similares a los del entrenamiento se consiguen muy buenos resultados, a pesar de haber utilizado en dicho entrenamiento un conjunto muy reducido, gracias a un previo entrenamiento con numerosos datos sintéticos.

Capítulo 5

Conclusiones y líneas futuras

En este apartado se resumen las conclusiones obtenidas y se proponen futuras líneas de investigación que se derivan del trabajo desarrollado.

5.1. Conclusiones

En este TFG se ha realizado el diseño, implementación y evaluación de un sistema de detección de personas en imágenes de profundidad basado en CNNs.

Partiendo de una serie de conocimientos teóricos adquiridos y explicados en el Capítulo 2 se ha construido una red compleja, se ha diseñado el formato de la salida y se ha llevado a cabo su entrenamiento. Para ello, ha sido necesario preparar diferentes bases de datos, en ocasiones etiquetándolas manualmente mediante *software* propio (*scripts* de MATLAB). Se ha estructurado el entrenamiento en dos partes bien diferenciadas, una primera con datos sintéticos y una segunda parte con un pequeño conjunto de datos reales. De esta forma, se ha pretendido demostrar como los datos sintéticos, con la ventaja de que pueden ser etiquetados automáticamente, permiten un primer entrenamiento en el que la red aprende a realizar la detección en un entorno favorable y poco realista. A continuación, con un pequeño set de datos reales se afina ese aprendizaje, mejorando el funcionamiento en un entorno más realista.

Los resultados obtenidos han demostrado como, en primer lugar, un sistema entrenado únicamente con datos sintéticos no funciona correctamente evaluándolo con datos reales, pese a poder conseguir resultados decentes si los datos no son excesivamente ruidosos o son similares en características a los sintéticos utilizados para entrenar. En segundo lugar, tras completar el entrenamiento afinando el aprendizaje con un pequeño set de datos reales, el sistema se comporta de forma mucho más robusta frente a datos similares a los que se han utilizado para entrenarlo (caso de la base de datos reales propia), pero empeora frente a datos reales diferentes (caso de EPFL-LAB). Esto nos demuestra como, durante ese afinado del aprendizaje, la red ha aprendido a efectuar la detección en un entorno concreto, o al menos de unas características concretas. Por tanto, es comprensible el mal funcionamiento con los datos de EPFL-LAB, ya que utilizan un sensor diferente (Kinect II), la perspectiva no es exactamente la misma y las características del ruido son distintas. A partir de las conclusiones extraídas del entrenamiento dividido en dos partes, puede entenderse que entrenando con un pequeño conjunto de imágenes de EPFL-LAB mejorarían sustancialmente los resultados frente a esa base de datos. Esta tarea no se ha llevado a cabo pues requiere un etiquetado manual de la base de datos de EPFL-LAB, ya que una estimación del centroide de las cabezas a partir del formato *bounding-box* aportado por [6] no es suficientemente preciso como para utilizarlo en un entrenamiento.

En resumen, las conclusiones extraídas se enumeran a continuación:

- El sistema entrenado únicamente con datos sintéticos puede conseguir resultados aceptables ante datos reales, siempre que no sean excesivamente ruidosos. Ante datos con demasiado ruido el funcionamiento es malo.
- Completando el entrenamiento con un pequeño set de datos reales se consiguen resultados muy buenos frente a datos similares a los utilizados en ese entrenamiento. Este es el punto más relevante, pues se ha demostrado cómo con un 7 % de datos reales en el set de entrenamiento, frente a un 93 % de datos sintéticos, el sistema es capaz de conseguir un muy buen funcionamiento ante datos reales.
- El uso de un pequeño conjunto de datos reales en el entrenamiento hace que el sistema pierda capacidad de generalización, y empeore su funcionamiento frente a datos en un entorno diferente o tomados con un sensor distinto.
- La existencia de oclusiones empeora notablemente el valor de los resultados obtenidos, en este caso con la aparición de FN. Se trata de un problema de difícil solución y evaluación en un sistema que trabaja "imagen-a-imagen" y no tiene en cuenta información temporal.

5.2. Líneas futuras

Las principales líneas de trabajo futuro, entendidas como posibles continuaciones del presente TFG, se enumeran a continuación:

- **Entrenamiento con datos de EPFL-LAB [6].** Como se ha mencionado anteriormente, puede realizarse un entrenamiento con un pequeño grupo de imágenes de EPFL-LAB (unas 100 ó 200), para comprobar cómo mejoran los resultados del sistema frente a esta base de datos. Esta tarea no se ha llevado a cabo porque requería re-etiquetar la base de datos.
- **Desarrollo de un sistema de *tracking*.** Partiendo de que el problema de las oclusiones tiene difícil solución con sistemas que trabajan "imagen-a-imagen", puede desarrollarse un sistema de seguimiento o *tracking* que complementa las detecciones del sistema aportando información temporal. Es decir, cada detección dependería del momento actual y de los anteriores. Una posible opción sería la implementación de un banco de filtros de Kalman.
- **Grabación de datos más complejos para una evaluación más exhaustiva.** Con el objetivo de comprobar más a fondo el sistema, sería interesante disponer de datos que contuviesen escenas más complejas, por ejemplo, personas con sombrero, con maletas, etc, es decir, objetos puntuales susceptibles de generar falsos positivos o falsos negativos. En caso de un empeoramiento de los resultandos puede plantearse un pequeño re-entrenamiento con este tipo de datos.

Capítulo 6

Pliego de condiciones

En este capítulo se indican las condiciones *hardware* y *software* necesarias para el desarrollo de este TFG.

6.1. Requisitos *hardware*

- Tarjeta gráfica NVIDIA GeForce GTX 1080ti con 11 GB de RAM
- Procesador Intel Core i7-6700K CPU @ 4MHz × 8
- 64 GB de memoria RAM
- 1 TB de disco duro

De los requisitos enumerados es importante la tarjeta gráfica (GPU), ya que el sistema se ejecuta en ella y de cara a un funcionamiento en tiempo real se recomienda el modelo 1080ti o 1080.

6.2. Requisitos *software*

- Sistema operativo Ubuntu 16.04
- Entorno de programación de Pycharm Community
- MATLAB
- Editor y procesador de LaTeX
- Python 3.5
- Librería OpenCV 3.1
- Librería Keras 2.1.2
- Librería TensorFlow 1.4.0
- Sistema de control de versiones CVS

Capítulo 7

Presupuesto

7.1. Costes de equipamiento

7.1.1. Equipamiento hardware utilizado:

Concepto	Cantidad	Coste unitario	Subtotal(euros)
PC Intel i7 + NVIDIA GTX1080ti	1	2000	2000
Camara RealSense D435	1	192,30	192,30
Coste Total			2192,30

7.1.2. Recursos software utilizados:

Concepto	Cantidad	Coste unitario	Subtotal(euros)
Ubuntu 16.04 LTS	1	0	0
Librerias Opencv Python 3.5.0	1	0	0
Librerias TensorFlow 1.4.0	1	0	0
Librerias Keras 2.1.2	1	0	0
MATLAB	1	2000	2000
TexStudio	1	0	0
Pycharm	1	0	0
Total			2000

7.2. Costes Mano de obra

Concepto	Cantidad	Coste unitario	Subtotal(euros)
Programacion y Desarrollo de Software	250	60 euros/hora	15000
Mecanografiado de documentación, manuales y tutoriales	60	15 euros/hora	900
Total			15900

7.3. Costes Totales

Concepto	Subtotal
Hardware	2192,30
Software	2000
Mano de obra	15900
Total	20092,30

El importe total del presupuesto asciende a la cantidad de: VEINTE MIL NOVENTA Y DOS EUROS

En Alcalá de Henares, a 16 de junio de 2019.

Roberto Martín López, Graduado en Ingeniería Electrónica de Comunicaciones

Bibliografía

[1] S. Hussmann, T. Ringbeck, and B. Hagebeuker, "A performance review of 3d tof vision systems in comparison to stereo vision systems," in *Stereo vision*. InTechOpen, 2008.

[2] "Cámara Realsense D435 de Intel," imagen disponible en https://www.intel.es.

[3] "Cámara Kinect II," imagen disponible en https://www.microsoft.com.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[5] X. Liu, Z. Deng, and Y. Yang, "Recent progress in semantic image segmentation," *Artificial Intelligence Review*, Jun. 2018. [Online]. Available: https://doi.org/10.1007/s10462-018-9641-3

[6] T. Bagautdinov, "Rgb-d pedestrian dataset," https://cvlab.epfl.ch/data/data-rgbd-pedestrian/.

[7] D. Ramanan, D. A. Forsyth, and A. Zisserman, "Tracking People by Learning Their Appearance," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 1, pp. 65–81, Nov. 2006.

[8] C. Y. Jeong, S. Choi, and S. W. Han, "A method for counting moving and stationary people by interest point classification," in *Image Processing (ICIP), 2013 20th IEEE International Conference on*, Sept 2013, pp. 4545–4548.

[9] A. Bevilacqua, L. Di Stefano, and P. Azzari, "People tracking using a time-of-flight depth sensor," in *Video and Signal Based Surveillance, 2006. AVSS '06. IEEE International Conference on*, Nov 2006, pp. 89–89.

[10] X. Zhang, J. Yan, S. Feng, Z. Lei, D. Yi, and S. Li, "Water filling: Unsupervised people counting via vertical Kinect sensor," in *Advanced Video and Signal-Based Surveillance (AVSS), 2012 IEEE Ninth International Conference on*, Sept 2012, pp. 215–220.

[11] C. Stahlschmidt, A. Gavriilidis, J. Velten, and A. Kummert, "People detection and tracking from a top-view position using a time-of-flight camera," in *Multimedia Communications, Services and Security*, ser. Communications in Computer and Information Science, A. Dziech and A. Czyazwski, Eds. Springer Berlin Heidelberg, 2013, vol. 368, pp. 213–223.

[12] C. A. Luna, C. Losada-Gutierrez, D. Fuentes-Jimenez, A. Fernandez-Rincon, M. Mazo, and J. Macias-Guarasa, "Robust people detection using depth information from an overhead time-of-flight camera," *Expert Systems with Applications*, vol. 71, pp. 240–256, 2017.

[13] D. F. Jiménez, "Diseño, implementación y evaluación de un sistema de conteo de personas basado en cámaras de tiempo de vuelo," Trabajo Fin de Grado, Universidad de Alcalá, 2016. [Online]. Available: http://www.microsoft.com/

[14] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[15] Y. Cao, C. Shen, and H. T. Shen, "Exploiting depth from single monocular images for object detection and semantic segmentation," *IEEE Transactions on Image Processing*, vol. 26, no. 2, pp. 836–846, 2017.

[16] S. Hoo-Chang, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning," *IEEE transactions on medical imaging*, vol. 35, no. 5, p. 1285, 2016.

[17] C. Wang and Y. Zhao, "Multi-layer proposal network for people counting in crowded scene," in *Intelligent Computation Technology and Automation (ICICTA), 2017 10th International Conference on*. IEEE, 2017, pp. 148–151.

[18] J. Zhao, G. Zhang, L. Tian, and Y. Q. Chen, "Real-time human detection with depth camera via a physical radius-depth detector and a cnn descriptor," in *2017 IEEE International Conference on Multimedia and Expo (ICME)*, July 2017, pp. 1536–1541.

[19] D. Vishwanath, "Toward a new theory of stereopsis," *Psychological review*, vol. 121, pp. 151–78, 04 2014.

[20] *Intel Realsense Depth Camera D-400 Series*, September 2017, rev. 0.7.

[21] L. Li, "Time-of-fligth camera-an introduction," 2014, technical white paper.

[22] M. Hansard, S. Lee, O. Choi, and R. Horaud, *Time-of-Flight Cameras: Principles, Methods and Applications*. Springer Publishing Company, Incorporated, 2012.

[23] D. Droeschel, D. Holz, and S. Behnke, "Multi-frequency phase unwrapping for time-of-flight cameras," 11 2010, pp. 1463 – 1469.

[24] H. Sarbolandi, M. Plack, and A. Kolb, "Pulse based time-of-flight range sensing," *Sensors*, vol. 18, p. 1679, 05 2018.

[25] D. Jiménez, D. Pizarro, M. Mazo, and S. Palazuelos, "Modeling and correction of multipath interference in time of flight cameras," *Image and Vision Computing*, vol. 32, no. 1, pp. 1 – 13, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0262885613001650

[26] D. Jimenez, D. Pizarro, and M. Mazo, "Single frame correction of motion artifacts in pmd-based time of flight cameras," *Image and Vision Computing*, vol. 32, no. 12, pp. 1127–1143, 2014.

[27] J. Sell and P. O'Connor, "The xbox one system on a chip and kinect sensor," *IEEE Micro*, vol. 34, no. 02, pp. 44–53, mar 2014.

[28] D. Greene, P. Cunningham, and R. Mayer, *Unsupervised Learning and Clustering*, 01 2008, pp. 51–90.

[29] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, 2017, pp. 1800–1807. [Online]. Available: https://doi.org/10.1109/CVPR.2017.195

[30] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," 11 2018.

[31] D. Pedamonti, "Comparison of non-linear activation functions for deep neural networks on mnist classification task," 04 2018.

[32] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," 05 2015.

[33] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[34] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[35] G. Hinton, "Overview of minibatch gradient descent."

[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[37] Blender Online Community, "Blender a 3d modelling package," Blender Institute, Amsterdam. [Online]. Available: http://www.blender.org

[38] J. Sell and P. O'Connor, "The Xbox one system on a chip and Kinect sensor," *Micro, IEEE*, vol. 34, no. 2, pp. 44–53, Mar 2014.

Apéndice A

Manual de usuario

A.1. Introducción

En este manual de usuario se explican diferentes aspectos relativos a la utilización del sistema de detección de personas implementado en este TFG.

A.2. Requisitos de utilización

El sistema de detección de personas se ha desarrollado sobre el sistema operativo *Linux*, concretamente utilizando la distribución *Ubuntu 16.04*. El código que implementa el sistema se ha escrito en el lenguaje interpretado *Python*. Para un correcto funcionamiento del sistema es requisito indispensable la instalación del siguiente *software*:

- ***Python* 3.5:** necesario para poder ejecutar el código, programado en este lenguaje.
- **Entorno de programación para *python*:** necesario para trabajar cómodamente con el código, modificarlo y ordenar su ejecución. Se recomienda el uso de *Pycharm Community*.
- **Librería *OpenCV* 3.0:** necesaria para tareas que requieran procesamiento de imágenes o relacionadas con visión artificial. Se utilizan para abrir las imágenes de entrada, visualizarlas, extraer el centroide de las gaussianas del mapa de probabilidad, etc.
- **Librería *TensorFlow* 1.4:** desarrollada para diferentes tareas relacionadas con el aprendizaje automático, como construcción y entrenamiento de redes neuronales. En este sistema se utiliza de forma indirecta, a través de la librería *Keras*, de más alto nivel.
- **Librería *Keras* 2.1.2:** se ejecuta sobre *TensorFlow* y permite, de una forma muy cómoda, llevar a cabo el diseño y el entrenamiento de redes neuronales profundas.

A.3. Estructura del programa

El código fuente que compone el sistema de detección se divide en dos archivos: **main.py** y **utils.py**. Ambos deben ubicarse en el mismo directorio para una correcta ejecución. Además, en el caso de querer crear de cero el sistema y entrenarlo o re-entrenar el sistema existente, éste se guarda en formato **.h5** en

el mismo directorio. Del mismo modo, si se quiere ejecutar el sistema, éste debe encontrarse en formato **.h5** y en el mismo directorio que el código. El formato **.h5** contiene todos los aspectos relativos a la arquitectura de la red: capas que la forman, conexión de éstas, pesos... A continuación se describe cada uno de los dos ficheros que contienen el código fuente del sistema:

- **main.py:** contiene las funciones que se encargan de la construcción de la red, de su entrenamiento, de cargar la red y ordenar su ejecución y de visualizar el resultado de la detección. En función del valor asignado a una serie de variables declaradas al comienzo del código se entrena desde cero el sistema, se re-entrena el sistema existente o se ejecuta el sistema entrenado. Además es necesario indicar el *path* desde el que deben cargarse las imágenes de profundidad, y en caso de querer llevar a cabo el entrenamiento es preciso también indicar el *path* del *ground-truth*.
- **utils.py:** contiene diversas funciones que son comúnmente utilizadas por **main.py**, y encargadas, por ejemplo, de crear los sub-bloques residuales, configurar el *fit-generator* utilizado para el entrenamiento, cargar y preparar los datos de entrenamiento, etc. Además contiene otras funciones relacionadas con el procesamiento de las imágenes para una correcta visualización.

A.4. Ejecución

El sistema se ejecuta siempre desde el fichero **main.py**. En función de la configuración indicada en éste, se llevarán a cabo diferentes tareas.

Si se activa la opción de entrenamiento, en primer lugar la consola nos muestra un resumen de la estructura de la red construida, y posteriormente comienza el entrenamiento. La Figura A.1 muestra un ejemplo del resumen de la arquitectura de la red que entrega el programa. Tras completar cada época la consola muestra la precisión calculada en validación e indica si se ha actualizado o no el modelo. El sistema se entrena durante tantas épocas como se le haya indicado, pero puede detenerse manualmente si se comprueba que no ha mejorado la precisión durante varias épocas consecutivas.

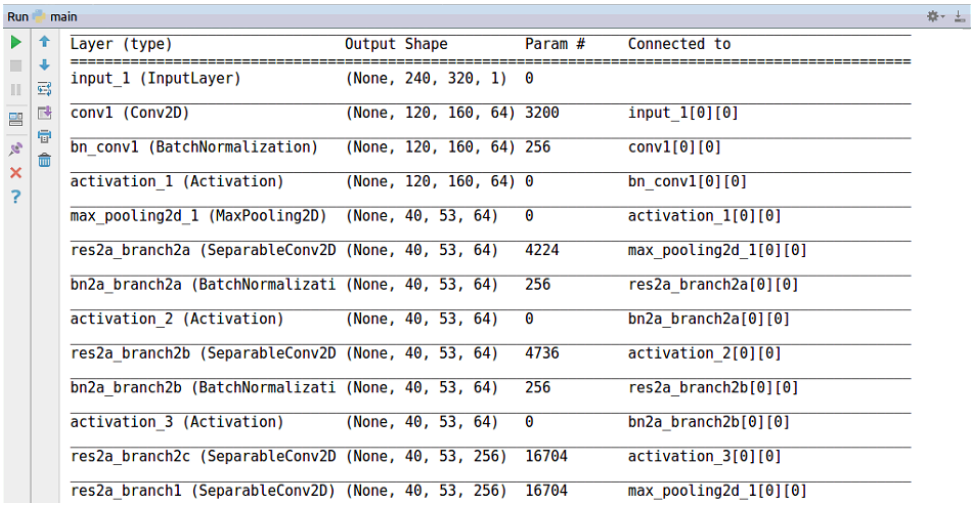


Figura A.1: Ejemplo de un fragmento del resumen de la arquitectura de la red entregado por el programa.

Si se activa la opción de re-entrenamiento, el comportamiento es similar al explicado en el párrafo anterior, con la única diferencia de que se carga la red existente (que puede estar ya pre-entrenada) en lugar de crear una red desde cero.

Desactivando las opciones de entrenamiento, el sistema simplemente se ejecuta, cargando la red guardada y entrenada, así como las imágenes de profundidad que se indiquen. Tras pasar cada imagen por la red, el programa procesa el mapa de probabilidad obtenido y obtiene las coordenadas donde se ubican los centroides de las cabezas de las personas detectadas. De esta forma, el programa muestra dos figuras, una con la imagen de profundidad e indicando, mediante un punto rojo, las personas detectadas, y otra con el mapa de probabilidad tal y como lo ha entregado la red, tal y como muestran las Figuras A.2 y A.3.

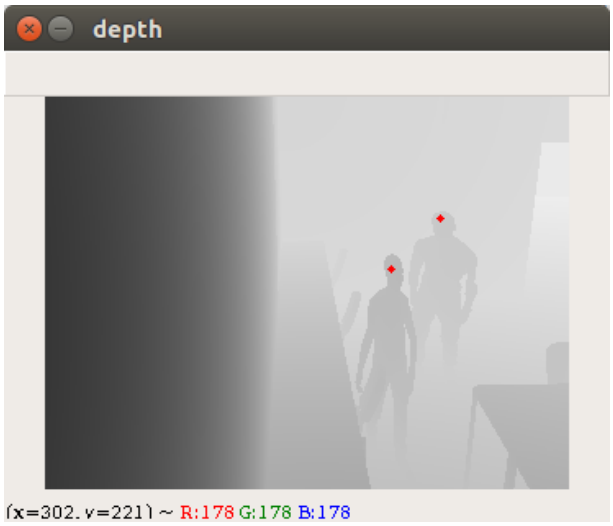


Figura A.2: Imagen de entrada con las detecciones efectuadas indicadas en rojo.

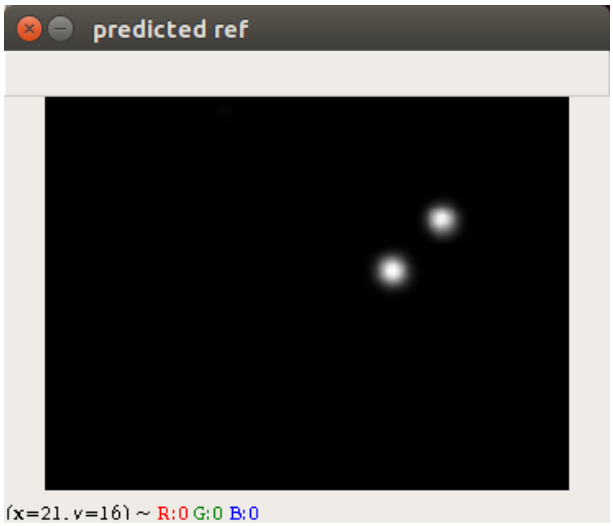


Figura A.3: Mapa de probabilidad entregado por la red.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá