UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Reconocimiento de gestos mediante redes neuronales convolucionales, utilizando imágenes de rango capturadas con el dispositivo Leap Motion

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Marc Rodas Lorente

Tutor: José Miguel Valiente González

Curso 2020-2021

Resum

En el present treball final de grau és desenvolupa un reconeixedor de gestos estàtics realitzats amb la mà, basat en xarxes neuronals convolucionals i utilitzant *Leap Motion* com a dispositiu de presa d'imatges, a causa de la seua bona relació de compromis entre prestacions i preu. En el disseny del classificador, després d'analitzar les possibles alternatives, és proposen diferents arquitectures de xarxa d'entre les quals poder seleccionar la millor. Per això, és realitza el seu entrenament i test a partir de diferents bases de dades gestuals preparades per a tal fi.

Es compta amb un corpus de dades format per 16 classes de gestos a partir dels quals es deriven diferents variants, l'expresivitat del quals també será evaluada en els experiments. Finalment, a fi de mostrar el resultat obtingut en el treball, és presenta un demostrador del reconeixedor de gestos a partir d'imatges capturades en temps real, que alimentaran la xarxa convolucional seleccionada.

Paraules clau: Reconeixement de gestos, xarxes neuronals convolucionals, Leap Motion

Resumen

En el presente trabajo final de grado se desarrolla un reconocedor de gestos estáticos realizados con la mano, basado en redes neuronales convolucionales y empleando *Leap Motion* como dispositivo de toma de imágenes, debido a su buena relación de compromiso entre prestaciones y precio. En el diseño del clasificador, después de analizar las posibles alternativas, se proponen distintas arquitecturas de red de entre las cuales poder seleccionar la mejor. Para ello, se realiza su entrenamiento y test a partir de distintas bases de datos gestuales preparadas para tal fin.

Se cuenta con un corpus de datos formado por 16 clases de gestos a partir del cuál se derivan distintas variantes cuya expresividad se valorará también en los experimentos. Por último, a fin de mostrar el resultado obtenido en el trabajo, se presenta un demostrador del reconocedor de gestos a partir de imágenes capturadas en tiempo real, que alimentarán la red convolucional seleccionada.

Palabras clave: Reconocimiento de gestos, redes neuronales convolucionales, Leap Motion

Abstract

In this final degree project, a static hand gesture recognizer is developed, based on convolutional neural networks and using *Leap Motion* as an imaging device, due to its good compromise between performance and price. In the design of the classifier, after analyzing the possible alternatives, different network architectures are proposed from which the best one can be selected. To do this, their training and tests are carried out from different gesture databases prepared for this purpose.

There is a data corpus made up of 16 classes of gestures from which different variants are derived whose expressiveness will also be assessed in the experiments. Finally, in order to show the results obtained in the work, a gesture recognizer demonstrator is presented from images captured in real time, which will feed the selected convolutional network.

Key words: Gesture recognition, convolutional neural networks, Leap Motion

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

En este capítulo se exponen las principales motivaciones para elegir el tema del presente trabajo final de grado, los objetivos que tiene definidos y finalmente la estructura que va a seguir la memoria.

1.1 Motivación

El reconocimiento automático de gestos mediante visión por computador es un campo de investigación que cuenta con numerosas aplicaciones especialmente en lo que se refiere a la interacción persona-máquina. En ese sentido, tiene muchísimos nichos en los que trabajar y una gran cabida en el mundo contemporáneo. Un ejemplo de ello son las tecnologías para mejorar la integración de las personas con diversidad funcional auditiva ya sea para navegar en la web, comunicarse, manejar dispositivos o programar. También tiene un gran abanico de posibilidades en las aplicaciones de interacción persona-máquina para el guiado de móviles, drones, máquinas, herramientas, sistemas operativos, videojuegos, aplicaciones médicas e incluso domótica.

El presente trabajo es un primer paso para poder entender como funciona el reconocimiento de signos mediante un computador de manera que, a partir del mismo, se pueda desarrollar en un futuro un intérprete de signos completo. Así pues, el interés está en el cómo y en las herramientas necesarias para solucionar de manera robusta dicho problema. Se necesita, a nivel hardware, un dispositivo capaz de captar imágenes de los gestos y, por otra parte, disponer de una técnica de reconocimiento de formas que se adapte a la complejidad requerida.

Como dispositivo hardware se ha seleccionado *Leap Motion* debido a su buena relación de compromiso entre prestaciones y precio, cumpliendo de sobra con los requisitos tal y como se verá más adelante. En cuanto a la técnica utilizada, se ha optado por el uso de las redes neuronales convolucionales debido al atractivo que tienen y a su auge en la actualidad como una de las herramientas para el procesado y reconocimiento de imágenes. Por último, como alumno de la rama de computación, conocer más el aprendizaje profundo resulta muy satisfactorio. Indagar y conocer más esta ciencia ha sido también uno de los principales atractivos.

1.2 Objetivos

El objetivo global del presente trabajo final de grado consiste en **diseñar un reconocedor de gestos estáticos, realizados con la mano, basado en redes neuronales convolucionales, utilizando para ello imágenes obtenidas del dispositivo *Leap Motion*.** A fin de alcanzar dicho objetivo se proponen los siguientes objetivos concretos:

- Estudiar distintas aproximaciones para el reconocimiento de gestos en la literatura científico-técnica a fin de encontrar un marco de referencia útil para desarrollar el trabajo planteado.
- Conocer y comprender la arquitectura y programación de las redes neuronales convolucionales.
- Proponer distintos reconocedores de diseño propio y basados en redes pre-entrenadas y realizar una comparativa entre ellos.
- Conocer y comprender el modo de funcionamiento y programación del dispositivo de captura *Leap Motion*.
- Desarrollar un demostrador del reconocedor en tiempo real para un conjunto seleccionado de gestos.

1.3 Estructura de la memoria

La memoria actual consta de ocho capítulos junto a la bibliografía. La combinación de todos ellos sitúa el proyecto en un contexto, desarrolla una serie de algoritmos para llegar a diferentes soluciones de un reconocedor de gestos y llega a una conclusión. Los capítulos son los siguientes:

1. Introducción: expone el trabajo y sus objetivos.
2. Estado del arte: presenta el contexto del trabajo actual.
3. Materiales y métodos: enumera los elementos que se utilizan y la forma de actuar para implementar la solución final.
4. Redes neuronales convolucionales: estudia la ciencia de las redes neuronales convolucionales y las arquitecturas más punteras.
5. Procesamiento de imágenes y generación de bases de datos: describe diversas técnicas para procesar imágenes y define con ellas los diferentes conjuntos de datos que se utilizan en la experimentación posterior.
6. Diseño del clasificador de gestos: a través de los estudios y ciertos experimentos se propone la creación de diferentes diseños y selecciona una solución final.
7. Experimentos: desarrolla las pruebas con los diferentes conjuntos, analiza los resultados y se expone un demostrador en tiempo real.
8. Conclusiones y trabajos futuros: valora todas las discusiones realizadas a lo largo del proyecto, la resolución de los objetivos, expone una idea final y plantea una serie de líneas de trabajo que seguir.

CAPÍTULO 2
Estado del arte

En este capítulo se desarrolla el estado del conocimiento en los campos que son de interés para el actual trabajo. Primeramente se ha estudiado la situación actual del área de reconocimiento de gestos, dando especial importancia a las investigaciones, experimentos y avances basados en el aprendizaje profundo. Es por ello, que se ha profundizado más en las redes neuronales convolucionales, subgrupo dentro de este ámbito. Seguidamente se ha hecho lo mismo para el hardware *Leap Motion* y finalmente se expone el estado de la cuestión en todos estos ámbitos simultáneamente.

2.1 Reconocimiento de gestos

El reconocimiento de gestos es un proceso computacional cuyo objetivo es interpretar gestos humanos. Este campo se basa en extraer características de gestos, procesarlas con algoritmos matemáticos y reconocer e interpretar signos. Estos gestos pueden ser de diversos tipos: faciales, hechos con los rasgos de la cara, corporales, con los movimientos del cuerpo o diferentes poses y manuales, con símbolos producidos por las manos. Este proyecto se centra en el último tipo de reconocimiento descrito. En la figura 2.1 se pueden ver ejemplos de gestos para los tres tipos.

Figura 2.1: Ejemplos de gestos faciales, corporales y manuales. Imagen extraída de [1]

Los gestos pueden ser estáticos o dinámicos. Los estáticos son gestos que proporcionan un significado sin necesidad de alterar su forma ni posición a lo largo del tiempo, como puede ser el símbolo dos o extender el pulgar hacia arriba. En cambio, los dinámicos son gestos que a lo largo de una sucesión de movimientos transmiten un mensaje, como puede ser saludar.

Principalmente existen siete aproximaciones del reconocimiento de gestos manual según se puede extraer de [3] y [4]:

- **Basado en colores:** mediante el uso de colores asociados a ciertas zonas se pueden diferenciar distintos tipos de gestos. Esto permite crear clasificadores capaces de analizar los colores por su posición o movimientos. En [5] se utiliza un guante con colores para detectar características en manos y así poder desarrollar un clasificador de tipo *learning vector quantization*.

- **Basado en apariencia con extracción de características:** se extraen características de las imágenes en forma de vectores que son posteriormente utilizados para crear clasificadores en función de sus datos. Hay una gran variedad de técnicas capaces de hacer esto. Algunos ejemplos de ellas son *canny edge detection*, *bag of word technique* y *ORB* como se puede ver en [6].

- **Basado en movimientos:** se extrae un gesto a través de una serie de *frames*, por lo tanto este método se utiliza para modelos que detectan gestos dinámicos. Primero se captura el movimiento de una mano *frame a frame* para poder tener información de todo el gesto. Finalmente se utilizan algoritmos de reconocimiento de formas para procesar estas imágenes. En [7] se puede ver un ejemplo que utiliza sensores piroeléctricos y una cámara para detectar movimientos.

- **Basado en esqueleto:** se crean esqueletos a partir de imágenes o vídeos para tener características de mayor complejidad. Por ejemplo, [8] es una metodología basada en aprendizaje profundo que extrae características con esqueletos a partir de una secuencia de videos RGB.

- **Basado en profundidad:** este método utiliza cámaras 3D que son capaces de obtener información geométrica de los objetos que captura. Con esta nueva información se obtiene profundidad en las imágenes que puede ser muy útil a la hora de crear clasificadores. En [9], se utilizan datos capturados a través del dispositivo *Kinect*, capaz de obtener mapas de rango. Posteriormente, estos datos son clasificados con Modelos ocultos de Márkov.

- **Basado en reconocimiento de modelos 3D:** a través de modelos 3D cinemáticos se obtienen volúmenes o esqueletos de los objetos gracias a la estimación de una imagen con un modelo de mano 3D. Esto permite mucha libertad ya que se tiene un modelo 3D de las imágenes que permite observarlas en todos los ángulos. En [10] se propone una estimación 3D de la mano a partir de una imagen de rango que aproxima su estructura a través de esferas y redes neuronales.

- **Basado en aprendizaje profundo:** mediante el uso técnicas de inteligencia artificial se aprenden patrones en imágenes y en vectores de características. Posteriormente estos patrones permiten clasificar y reconocer objetos en base a ellos. Hay una gran variedad de técnicas para aprender patrones. En [11] se presenta un ejemplo de uso de *EMG*, clasificadores basados en electromiografía, y de redes neuronales convolucionales para clasificar signos.

  Este tipo de reconocimiento es el que se va a utilizar en el presente trabajo. En concreto, se utilizarán modelos de redes neuronales convolucionales, un subgrupo de

los muchos que tiene el aprendizaje profundo y que a partir de ahora, denominaremos también como *CNN*. A continuación profundizaremos más en el estado actual de este campo.

2.2 Estado del arte de las CNNs

Las redes neuronales convolucionales son especialmente utilizadas para detectar patrones presentes en imágenes, sin embargo pueden ser utilizadas en otros campos distintos. Por lo tanto, su cabida en el mundo contemporáneo es inmensa. Según [12] las nueve aplicaciones más relevantes que estas tienen son en los siguientes ámbitos:

- **Visión por computador:** su objetivo es procesar imágenes y vídeos para extraer información útil de ellos y así poder reconocer objetos. Por ejemplo, en [13] se utilizan dos CNNs para determinar si un trabajador lleva arnés de seguridad y así poder reducir el número de muertes por caídas en el trabajo.
- **Procesamiento de lenguaje natural:** busca procesar el lenguaje del ser humano para ser entendido e interpretado por los computadores. Un ejemplo de ello es [14] que utiliza vectores de máquinas soporte y redes neuronales convolucionales para clasificar la literatura médica sobre genes de susceptibilidad al cáncer.
- **Detección y segmentación de objetos:** encuentra e identifica diversos objetos en imágenes. Por ejemplo, en [15] se utiliza una *CNN* basada en regiones para detectar objetos pequeños.
- **Clasificación de imágenes:** diferencia unas imágenes de otras y las clasifica para crear sistemas que se nutren de esa diferenciación. Por ejemplo, se puede utilizar para clasificar imágenes médicas [16].
- **Reconocimiento de voz:** en el campo del reconocimiento del habla también se puede hacer uso de redes neuronales convolucionales. Distintos tipos de sonidos pueden ser identificados, por ejemplo en [17] se utiliza una red neuronal convolucional para identificar sonidos de insectos en el bosque.
- **Procesamiento de video:** se analiza el espacio y los objetos a lo largo de los *frames* de un video. Como muestra, se puede ver que en [18] se utiliza una *CNN* para eliminar el ruido de ciertos vídeos y así tener mejores imágenes.
- **Imágenes de baja resolución:** mejora la resolución de las imágenes con diferentes técnicas. Como un ejemplo de este ámbito, en [19], a partir de una secuencia de imágenes térmicas de resolución extremadamente baja se crea un método de reconocimiento de acciones.
- **Sistemas de recursos limitados:** son modelos de aprendizaje profundo que utilizan sistemas embebidos, hardware o dispositivos que no tienen muchas prestaciones. Un ejemplo de ello se puede ver en [20], donde una CNN es utilizada para detectar vehículos en condiciones lumínicas desfavorables para sistemas de transporte inteligentes.
- **Procesamiento de señal:** busca procesar datos que están representados por vectores que solo tienen una única dimensión. Un ejemplo de este tipo lo encontramos en [21] que hace uso de una *CNN* para detectar arritmia en personas expuestas a señales electrocardiográficas de larga duración.

Como se puede observar, en la actualidad, existe un gran abanico de campos de aplicación para las CNNs. En el capítulo 4 se estudiarán las características internas y funcionamiento de las CNN con detalle, por lo que el estado del arte en esta cuestiones se pospone a dicho capítulo.

2.3 Leap Motion

Como se verá en la subsección 3.1.1, *Leap Motion* es un dispositivo óptico para el seguimiento de las manos capaz de controlar sus movimientos de forma precisa. Este controlador tiene unas librerías que detectan ciertas posiciones, movimientos e incluso características de las manos que están siendo capturadas. A partir de ahora, también denominaremos este dispositivo como *Leap*. En cuanto al estado actual del dispositivo, hay una gran cantidad de proyectos que hacen uso de sus librerías para desarrollar sistemas interactivos. En [22] y [23] se pueden ver ejemplos para la creación de sistemas de realidad virtual y control de drones respectivamente.

Otras prestaciones muy interesantes de *Leap* son capturar imágenes de manos y crear ficheros con información del esqueleto de estas, todo a través de sus cámaras. El uso de estas funcionalidades se puede ver en [24], donde se extraen vectores de características a partir de imágenes y posteriormente se clasifican utilizando *hidden conditional neural field* y en [25], en donde se extraen vectores de características a partir de imágenes, y posteriormente se clasifican utilizando una CNN para reconocer gestos en movimiento.

Especialmente se van a analizar dos publicaciones: [2] y [26], debido a que son las aproximaciones que más se parecen al objetivo final del proyecto. En [2] se utiliza una base de datos con imágenes infrarrojas e información del esqueleto hecha con el dispositivo *Leap* de gestos manuales que es bastante extensa. Con estas imágenes se hace un clasificador a través de *máquinas de vectores soporte*, *histograma del gradiente* y técnicas de visión por computador. En [26] se combina *Kinect* y *Leap* para crear un clasificador de gestos a partir de imágenes de rango sacadas de *Kinect* e información del esqueleto sacada de *Leap*. El método utilizado para clasificar es a través de CNNs. A lo largo del proyecto se harán nuevas referencias y estudios de estas dos publicaciones.

2.4 Conclusiones sobre el estado del arte y planteamiento del trabajo

Los dos últimos artículos citados son los artículos de referencia para el actual proyecto. Al utilizar *Leap Motion*, una base de datos hecha con este dispositivo, va a ser de gran ayuda. Por ello, se va a utilizar la misma base de datos en el proyecto que la utilizada en [2], con el fin de poder comparar resultados con otras metodologías.

Por otra parte, [26] utiliza mapas de rango sacados del dispositivo *Kinect* ya que se argumenta que *Leap Motion* no ofrece directamente mapas de rango y que es complejo resolver la gran distorsión que presentan sus imágenes. En el capítulo 5 se discuten ambos aspectos tratando el problema de la distorsión y la generación de mapas de rango a partir de las imágenes obtenidas de *Leap*.

Una vez finalizada la investigación respecto al estado del arte, se puede concluir que el actual proyecto es un reconocedor de gestos basado en aprendizaje profundo, esqueleto y profundidad, que es aplicado al campo de la visión por computador y a la clasificación de imágenes con redes neuronales convolucionales. Por último, va a utilizar el dispositivo

Leap Motion y una base de datos obtenida mediante el mismo y que más adelante se especificará en la sección 3.3.

CAPÍTULO 3
Materiales y métodos

En este capítulo se presentan todas las herramientas y procedimientos que han sido utilizados para desarrollar la solución final. Esto incluye los elementos físicos y los programas que han sido necesarios para crear la arquitectura informática.

3.1 Hardware

3.1.1. Leap Motion

Leap Motion es un dispositivo óptico diseñado para interactuar con el movimiento de las manos en diferentes sistemas informáticos. Este dispositivo toma imágenes en escala de grises del espectro de luz infrarroja separadas en dos cámaras: una situada en la parte izquierda y otra en la parte derecha del dispositivo. La cámara puede captar imágenes gracias a la luz infrarroja emitida por el dispositivo y que viene dada por tres LEDs infrarrojos.

La distribución del conjunto mencionada se puede ver en la figura 3.1.

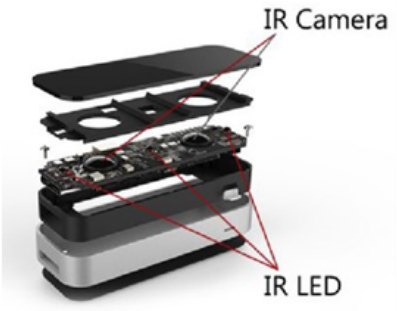

Figura 3.1: Estructura del dispositivo Leap Motion.

El dispositivo tiene un rango de actuación de 10 a 60 centímetros de profundidad, con un campo de visión de 140x120 grados. En la figura 3.2 se puede ver un esquema del alcance de *Leap Motion*. Una vez el dispositivo está en marcha y captura imágenes en ambas cámaras, el software propietario de *Leap* se encarga de extraer información del movimiento de las manos y crear esqueletos de las mismas para poder interactuar con los sistemas en base a los movimientos que se detectan y así seguirlas.

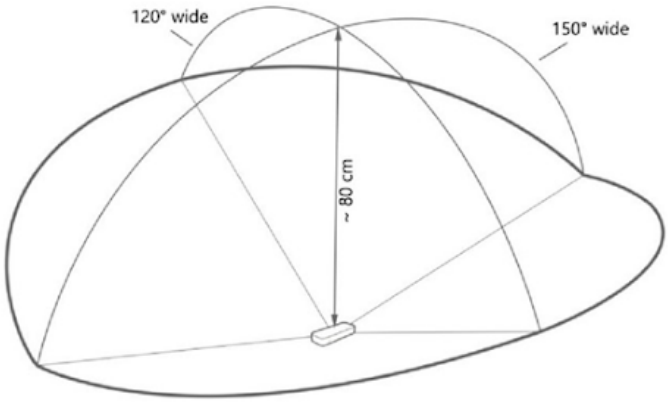

Figura 3.2: Rango de actuación del dispositivo Leap Motion.

De este proceso matemático en el que se consigue información 3D de las manos no hay información al respecto, tal y como puede verse en [27]. Presumiblemente se utilizan técnicas de stereo-visión para obtener esos datos al haber dos cámaras. Por último, *Leap Motion* ofrece la posibilidad de obtener las imágenes infrarrojas (*Raw images*) de ambas cámaras (L y R) y el fichero xml con información del esqueleto obtenido a partir de estas.

Por lo tanto, hay dos maneras de trabajar con *Leap Motion*: una con el controlador que sigue los movimientos de las manos y que está limitado a los gestos predeterminados de las librerías propias de *Leap* y la otra, trabajando directamente con las imágenes infrarrojas o esqueletos con los que se puede obtener la información que se precise, tanto 2D como 3D. En el presente proyecto se utiliza la segunda opción al ser mucho más personalizable que la primera.

3.1.2. Computador

Para la realización de los programas, entrenamiento de redes y los experimentos se ha utilizado un computador con las siguientes prestaciones:

- Una CPU AMD Ryzen 5 1600.
- Una tarjeta gráfica GeForce GTX 1060 3GB compatible con *CUDA*.
- Una memoria RAM de 8GB.

Con estas prestaciones, los experimentos tienen una duración media de unos 30 minutos, que puede ser alterada por el grueso de la base de datos o de los modelos utilizados.

3.2 Software

El software utilizado es el siguiente:

- **Sistema operativo:** *Windows 10*, es un sistema operativo sobradamente conocido que permite interactuar con el computador y los sistemas informáticos, afines a él, mediante su interfaz.

- **Entorno de programación:** *Visual Studio Code* , es un editor de código que cuenta con un gran número de extensiones para programar y utilizar todo tipo de lenguajes de programación y librerías. Permite control de versiones, monitorización de tareas y depuración de errores.
- **Lenguaje de programación:** *Python 3.8*, es un lenguaje de programación que permite construir aplicaciones web, analizar datos, crear aplicaciones y automatizar procesos.
- **TensorFlow y Keras:** *TensorFlow* es una plataforma de código abierto centrada en aprendizaje profundo. Cuenta con diversas herramientas, bibliotecas y recursos que permiten innovar y crear aplicaciones que utilizan modelos y técnicas del aprendizaje profundo de forma sencilla. *Keras* es una *API* de alto nivel que utiliza *TensorFlow* para construir y entrenar sus modelos. Dispone de información clara de todas las librerías utilizables, es configurable y no tiene muchas restricciones. Se han utilizado ambas librerías para la programación de las *CNNs*
- **OpenCV:** es una librería de código abierto que ofrece herramientas en la visión por computador, aprendizaje profundo y procesamiento de imágenes en tiempo real. Se ha utilizado para el procesamiento de imágenes.
- **MATLAB:** es una plataforma de programación y cómputo numérico para analizar datos, desarrollar algoritmos y crear modelos. Se ha utilizado junto a *OpenCV* para visualizar los procesamientos de las imágenes.
- **Leapuvc:** es una librería que permite el acceso a ciertas configuraciones procedentes de *Leap* como parámetros de la cámara, de las lentes, de la distorsión y crear mapas de rango a partir de ellas. Se ha utilizado para la creación del mapa de rango [28].
- **Leap-Motion-Python-3.8:** es una librería que con un *Wrapper* permite el manejo del dispositivo *Leap Motion* en la versión 3.8 de *Python*, se puede encontrar y descargar en [29].

3.3 Conjunto de datos original

Tal y como se ha descrito en el estado del arte, el trabajo [2] dispone de un conjunto de datos con imágenes infrarrojas y ficheros xml con información del esqueleto capturados a través del dispositivo *Leap Motion*. Los autores de este conjunto, pertenecen al grupo de investigación *Grupo de Tratamiento de Imágenes* de la universidad politécnica de Madrid, y sus nombres son: T.Mantecón, C.R.del Blanco, F.Jaureguizar y N.García.

El conjunto se referencia como `Multi-Modal Hand Gesture Dataset for Hand Gesture Recognition` y se puede descargar en [30]. Tiene 16 clases de diferentes gestos manuales estáticos y un total de 86.236 muestras con un tamaño aproximado de 416x275 *pixeles*. En la tabla de la figura 3.3 se puede ver un ejemplo de cada gesto. Siguiendo el orden de izquierda a derecha y de arriba abajo de la tabla, conforme los creadores del conjunto han denominado las clases, sus nombre son: **01_palm, 02_l, 04_fist_moved, 05_down, 06_index, 07_ok, 08_palm_m, 09_c, 11_heavy, 12_hang, 13_two, 14_three, 15_four, 16_five, 17_palm_u, 18_up**. También tiene ficheros con gestos dinámicos que no se van a utilizar ya que la propuesta es con gestos estáticos.

A la hora de mencionar las clases en el proyecto, se ha hecho una traducción y una abreviación. Siguiendo el mismo orden y en forma de tupla, estas son: **(Palma, PA), (Letra ele, LL), (Puño movido, PM), (Abajo, AB), (Dedo índice, DI), (De acuerdo, OK), (Palma de perfil, PP), (Letra ce, LC), (Cuernos, CU), (Colgar, CO), (Número dos, N2), (Número**

Figura 3.3: Tabla de clases del reconocedor de gestos.

tres, N3), (Número cuatro, N4), (Número cinco, N5), (Dorso, DO), (Arriba, AR). A partir de ahora, también denominaremos las clases con estos últimos nombres.

Para facilitar el entendimiento de los gestos se han mostrado las imágenes recortadas. No obstante, las imágenes reales tienen mucho más contenido como el brazo y el fondo. Aparte, las imágenes son diferentes para cada clase y cubren muchas variantes para hacer cada gesto. En las figuras 3.4 y 3.5, para las clases Colgar y Palma respectivamente, se puede ver de forma clara la diversidad de imágenes para cada clase.

Se dispone de imágenes infrarrojas provenientes de ambas cámaras del dispositivo *Leap* y ficheros xml de los datos esqueléticos para los gestos. El total de datos está dividido en dos partes: entrenamiento y test, con un 80 % y un 20 % de muestras respectivamente del total. A partir de ahora, se va a utilizar el término **base de datos** en castellano, como sinónimo de conjunto de datos, para nombrar los diferentes conjuntos de datos con los que se trabaja en el actual proyecto, y se va a reservar el término conjunto para agrupaciones de estos. Por último, la base de datos explicada anteriormente se va a denominar **BD-IR**. A continuación, se estudiarán las variantes que se han creado en el presente TFG a partir de BD-IR.

Figura 3.4: Muestras de la clase Colgar.

Figura 3.5: Muestras de la clase Palma.

3.4 Conjuntos de bases de datos

La base de datos escogida divide inicialmente todas sus muestras en dos partes: entrenamiento y test. Para realizar el diseño de las redes la porción de entrenamiento ha sido dividida a su vez en dos partes: una primera con el 80 % de las muestras para el entrenamiento propiamente dicho y una segunda, que contiene el 20 % de muestras para los datos de validación. La partición de test original se utilizará para evaluar el modelo ante muestras nunca vistas. Consecuentemente, se dispone entonces de tres particiones de las muestras en cada una de las bases de datos, formadas tanto por imágenes como por ficheros xml, estas son: entrenamiento (TR), validación (V) y test (TE).

El número de imágenes infrarrojas, ficheros xml y pares de imágenes derecha e izquierda no son los mismos. Es por ello, que se han dividido las posibles bases de datos extraíbles a partir de la información de la original en tres categorías o conjuntos, agrupados por número de muestras totales.

3.4.1. Conjunto de datos IR

A este conjunto pertenecen todos los grupos de imágenes derivadas sólo de cualquier imagen infrarroja. Tiene 86.236 muestras en total, 51.244 en entrenamiento, 12.813 en validación y 22.179 en test. En la tabla 3.1 se puede ver el número de muestras por clase en cada conjunto y su totalidad.

C	PA	LL	PM	AB	DI	OK	PP	LC	CU	CO	N2	N3	N4	N5	DO	AR
TR	5.1	2.8	2.8	2.4	3	3.2	5.3	3.1	3	2.9	2.8	2.6	2.6	2.6	4.7	2.4
V	1.3	0.7	0.7	0.6	0.7	0.8	1.3	0.8	0.8	0.7	0.7	0.6	0.7	0.6	1.2	0.6
TE	2	1.4	1.4	1	1.3	1.4	2.1	1.3	1.3	1.2	1.2	1.2	1.2	1.2	2	1
TO	8.4	4.9	4.9	4	5	5.4	8.7	5.2	5.1	4.8	4.7	4.4	4.5	4.4	7.9	4

Tabla 3.1: Número de muestras por partición en el conjunto de datos infrarrojo. (expresada en miles)

3.4.2. Conjunto de datos LR

Las bases de datos creadas a partir de dos imágenes de la misma muestra y que contengan algún tipo de información, es perteneciente a este conjunto. Tiene 24.819 muestras en total, 16.316 en entrenamiento, 4.078 en validación y 4.425 en test. En la tabla 3.2 se muestra el número de muestras por clase en cada conjunto y el total.

C	PA	LL	PM	AB	DI	OK	PP	LC	CU	CO	N2	N3	N4	N5	DO	AR
TR	2.3	0.7	0.7	1.1	0.7	0.7	2.3	0.8	0.8	0.6	0.7	0.7	0.7	0.6	2	1
V	0.6	0.2	0.2	0.3	0.2	0.2	0.6	0.2	0.2	0.2	0.2	0.2	0.2	0.1	0.5	0.3
TE	0.7	0.2	0.2	0.3	0.2	0.2	0.7	0.1	0.2	0.1	0.2	0.1	0.2	0.2	0.6	0.3
TO	3.6	1.1	1.1	1.7	1.1	1.1	3.6	1.1	1.2	0.9	1.1	1	1.1	0.9	3.1	1.6

Tabla 3.2: Número de muestras por partición en el conjunto de datos LR. (expresada en miles)

3.4.3. Conjunto de datos SK

De los ficheros xml con información del esqueleto de la mano para cada muestra, se pueden crear imágenes. Bases de datos provenientes de este proceso son las que pertenecen a este conjunto. Tiene 61.222 muestras en total: 34.832 en entrenamiento, 8.709 en validación y 17.681 en test. Más adelante en la tabla 3.3 se puede ver el número de muestras por clase en cada conjunto y el número total.

C	PA	LL	PM	AB	DI	OK	PP	LC	CU	CO	N2	N3	N4	N5	DO	AR
TR	2.9	2.1	2.1	1.4	2.3	2.5	2.9	2.2	2.2	2.3	2.1	1.9	1.9	2	2.6	1.4
V	0.7	0.5	0.5	0.3	0.6	0.6	0.7	0.6	0.6	0.6	0.5	0.5	0.5	0.5	0.7	0.3
TE	1.3	1.1	1.2	0.6	1.1	1.2	1.4	1.2	1.2	1.1	1.1	1.1	1	1.1	1.3	0.7
TO	4.9	3.7	3.8	2.3	4	4.3	5	4	4	4	3.7	3.5	3.4	3.6	4.6	2.4

Tabla 3.3: Número de muestras por partición en el conjunto de datos con esqueletos. (expresada en miles)

Se puede ver que los datos son bastante heterogéneos y que el número de muestras es variable, tanto entre clases como entre conjuntos. No se ha alterado el número de imágenes ni de clases para comparar adecuadamente los resultados, pese a tener cierto desbalance. Con las muestras de cada conjunto, y aplicando una serie de transformaciones a cada uno de ellos, se crearán diversas bases de datos con el objetivo de separar la red del proceso de creación de las diversas imágenes y tener independencia entre ellas y el modelo. En el capítulo 5 se exponen detalladamente todas las bases de datos creadas a partir de estos tres conjuntos y los algoritmos necesarios para dicha tarea.

3.5 Esquema de la solución

La figura 3.6 muestra un esquema general de la metodología que se va a seguir en el presente trabajo.

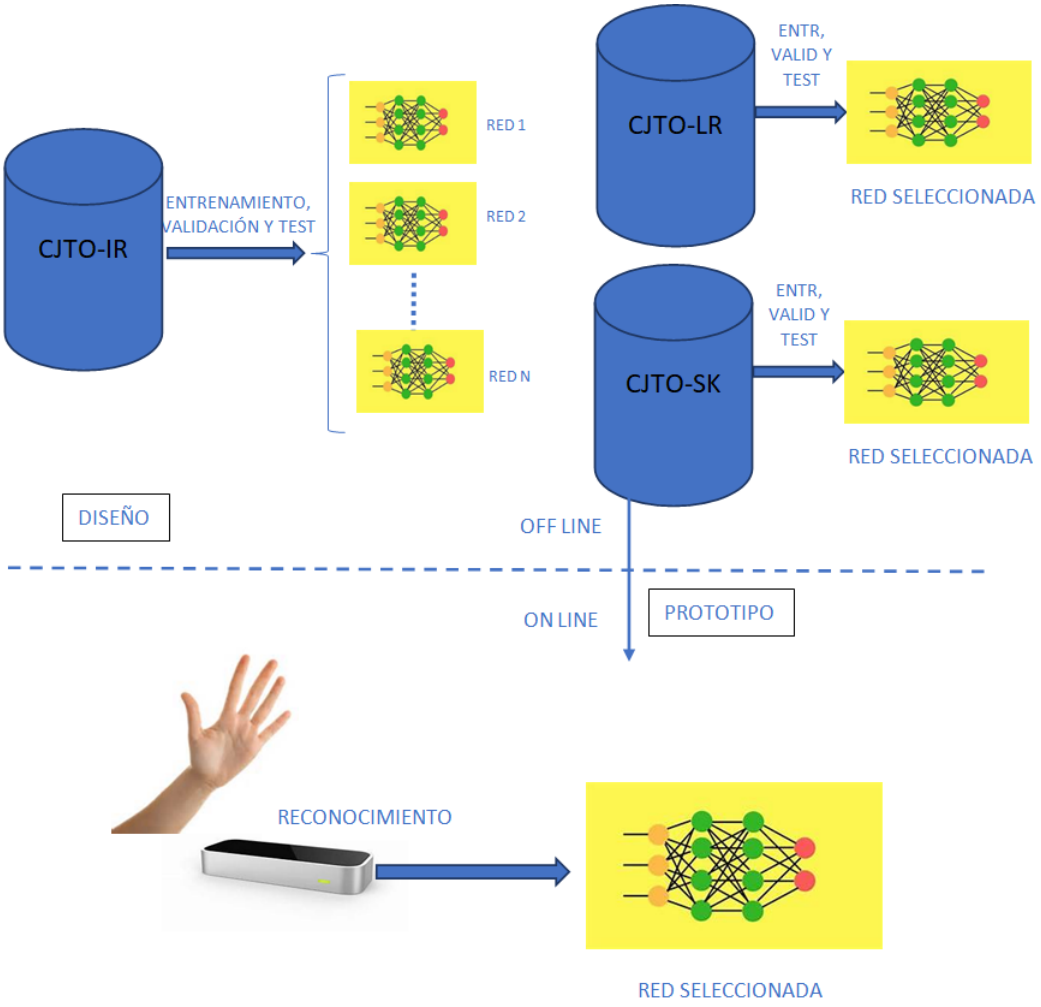

Figura 3.6: Esquema de la metodología aplicada

Se distinguen dos etapas: la superior, que se hará off-line, se dedicará al diseño de distintas arquitecturas de redes propias y pre-entrenadas. Estas redes serán entrenadas y probadas utilizando imágenes almacenadas en distintas bases de datos. Como resultado de esta etapa, se seleccionará la mejor de las arquitecturas para ser utilizada en el demostrador esquematizado de la segunda etapa, la inferior, que funcionará en tiempo real, y que será la encargada del reconocimiento de gestos on-line. La primera etapa consta a su vez de dos fases: durante la primera (arriba-izquierda) se utilizarán bases de datos con imágenes infrarrojas para seleccionar una única red de entre las planteadas y así proseguir los experimentos de la segunda (arriba-derecha).

En esta segunda fase, se utilizarán bases de datos con imágenes más elaboradas resultantes del procesamiento de la base de datos *BD-IR*. Los detalles de cada una de ellas se encuentran en el capítulo 5, dedicado al procesamiento y segmentación de imágenes. Estas bases de datos se estructuran en los tres conjuntos de datos detallados anteriormente.

CAPÍTULO 4
Redes neuronales convolucionales

Con el objetivo de implementar el clasificador de gestos correctamente, se hizo un estudio sobre las redes neuronales convolucionales: su arquitectura, parámetros y diseño. Es una parte crucial y el presente capítulo está dedicado a ello. Toda la información que aparece en este apartado está extraída de las referencias [12], [31], [32] y [33]. Si se quiere profundizar más en los temas es recomendable consultarlas.

4.1 Introducción a las redes neuronales

Una red neuronal es una arquitectura informática que se compone de una serie de estructuras conectadas llamadas neuronas o nodos, y que están organizadas por capas. Estas arquitecturas transforman unos datos de entrada en una salida. En función de la estructura y el tipo de las capas, las transformaciones que se hacen a los datos son diferentes. En una red neuronal existen tres bloques de neuronas:

- **Capa de entrada:** esta capa recoge los datos de entrada que recibe la red.
- **Capas ocultas:** estas capas tienen un número de nodos diferente en función de la transformación que se quiera conseguir, y son las que van a hacer que la red tenga un comportamiento deseado u otro.
- **Capa de salida:** esta capa tiene un nodo para cada salida deseada.

En la figura 4.1 se puede ver de forma gráfica la disposición de las capas.

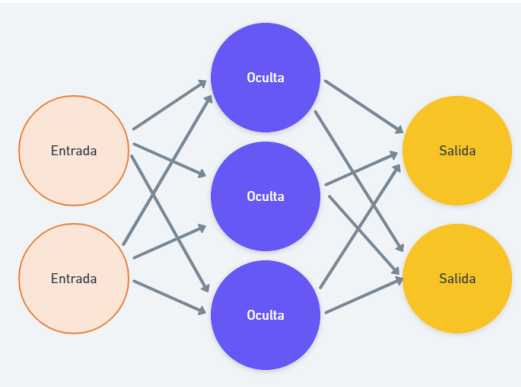

Figura 4.1: Esquema genérico de red neuronal.

Cada neurona pasa la información a otra a través de una conexión. Esta conexión tiene pesos y representa la fuerza de la conexión entre esas dos neuronas. El objetivo de

las redes neuronales es optimizar el peso de cada conexión actualizándose, para que los datos de entrada de la red tengan la salida final que se espera de ellos en base a los datos suministrados. Este proceso trata de resolver un problema de optimización al intentar actualizar los pesos, y es conocido como el entrenamiento de la red neuronal. El algoritmo que se utiliza para ello es conocido como *descenso del gradiente*. Las redes aprenden a reconocer elementos, y esto se puede utilizar en muchos campos para predecir o averiguar datos a partir de entradas nuevas.

4.1.1. Redes neuronales artificiales

En función de la estructura y los tipos de capas que se utilizan en una red, su comportamiento es diferente. Por lo tanto, hay muchos tipos de redes neuronales. Las redes más básicas son las llamadas redes neuronales artificiales, que utilizan capas de tipo *Dense*. La capa *Dense* es la capa estándar de una red neuronal y su función es conectar capas y transmitir la información modificada por cada neurona.

Esta capa puede estar en cualquiera de los tres bloques mencionados anteriormente. Recibe una entrada de otra capa y pasa una salida a todas las demás del siguiente bloque. Por lo tanto, conecta todas las neuronas de la capa con todas las neuronas de la siguiente, esa es la función de la capa *Dense*. Cuando se crea una *Dense* hay que especificar qué número de neuronas va a tener. Como se ha comentado, *Dense* coge una entrada y la transforma en una salida. Esta salida va a depender de la función de activación que tenga asociada.

4.1.2. Funciones de activación

Las funciones de activación son lo que hacen que una red neuronal tenga no linealidad y que cada capa pueda alterar su entrada en una salida nueva. Esto se consigue gracias a introducir una función no lineal, también llamada de activación, al algoritmo del *perceptrón*, también conocido como función discriminante lineal. Dada una entrada, las funciones de activación definen una salida no lineal. Una capa debe de tener una función de activación para transformar la entrada que recibe.

En base a todos los contenidos referenciados se pueden destacar cinco tipos de funciones de activación más importantes:

- **Sigmoid:** dado un número negativo lo transforma a uno muy cercano a 0. Si es positivo lo transforma a uno muy cercano a 1. Si está próximo a 0 lo transforma a uno entre 0 y 1. Esta función de activación no es muy popular, ya que tiene *el problema del desvanecimiento del gradiente* y elimina ciertos datos importantes para la red. Este es un problema a la hora de entrenar ciertas redes basadas en el descenso de gradiente, y consiste en que cada vez el valor del gradiente es menor y consecuentemente, los pesos cambian menos.

- **Tanh:** si se le pasa un número negativo, lo transforma a uno muy cercano a -1. Si es positivo, lo transforma a uno muy cercano a 1. Si está cerca de -1 lo transforma a uno entre -1 y 1. Esta función no goza de popularidad actualmente, debido también *al problema del desvanecimiento del gradiente*.

- **ReLu:** es una de las funciones más comunes, transforma la entrada entre el máximo de 0 y el dato en sí, de tal forma que no pasa los valores negativos. Esta función es la más popular, no tiene el mismo problema que las otras y aprende más rápido. El único problema es que no se puede utilizar en las capas de salida.

- **Leaky ReLu:** es una función de activación basada en ReLu. La diferencia es que sesga un poco menos, y permite pasar algunos valores negativos a otras capas, no como la ReLu. Esta función de activación se puede utilizar como forma alternativa a la ReLu si hay problemas con ella.

- **SoftMax:** dada una entrada saca probabilidades para cada salida correspondiente al modelo. Esta función de activación se suele utilizar en la última capa para clasificar o pasar el dato a otra capa. Es por ello que es muy popular.

Se recomienda utilizar para las capas de entrada y ocultas la función de activación ReLu y en las capas de salida la función de activación Softmax.

4.1.3. Otros parámetros

Una vez creada la arquitectura de la red con capas y funciones de activación, se necesitan otros parámetros para que aprenda:

- **Optimizadores:** para compilar modelos de redes se necesita seleccionar una técnica de optimización para encontrar los pesos de la forma más beneficiosa para ella.

- **Función de pérdida:** las redes neuronales utilizan *el descenso por gradiente* y es por eso, que se necesita escoger una función de pérdida. Esta función busca minimizar el error a la hora de optimizar los pesos.

- **Datos:** el aprendizaje de las redes es sobre ciertos datos. Se necesita disponer de una base de datos sobre la que entrenar la red.

- **Rondas:** es un parámetro del descenso del gradiente que indica el número de veces que la red hace el proceso de aprendizaje de pesos.

4.2 Redes neuronales convolucionales

Las redes neuronales convolucionales son muy populares en el análisis de imágenes. Son redes con ciertas características que hacen que sea posible detectar patrones. Estas nuevas características son las que hacen que sean tan efectivas en el análisis y reconocimiento de imágenes. Las CNNs tienen capas ocultas llamadas capas convolucionales, eso es lo que las diferencia del *perceptrón multicapa*. Una CNN también puede tener otro tipo de capas, como la *Dense*.

4.2.1. Capas convolucionales

Como todas ellas, una capa convolucional recibe una imagen de entrada y la transforma mediante la convolución con un conjunto de filtros creando una imagen de salida, con tantas dimensiones adicionales como filtros se han empleado. Para cada capa convolucional hay que especificar un número de filtros; estos filtros permiten detectar los patrones de las imágenes. Algunos de ellos detectan círculos, líneas, formas complejas, etc. Cada filtro detecta un patrón diferente. Las primeras capas y los primeros filtros detectan patrones simples pero conforme se profundiza en la red, las últimas detectan formas más complejas como muñecas, dedos o incluso formas mucho más difíciles como caras.

4.2.1.1. Filtros

Los filtros son pequeñas matrices de las cuales se pueden decidir las filas y columnas que tienen, así como sus coeficientes. La disposición y los valores de los coeficientes en la matriz determinan el tipo de patrones que se van a detectar en las imágenes. Los filtros se aplican de forma que se desplazan a lo largo de la imagen, alcanzando distintas submatrices del mismo tamaño que el filtro. En cada movimiento del filtro se realiza el sumatorio del producto de cada coeficiente por el píxel de la imagen que cae por debajo del mismo. Este valor acumulado será el valor del píxel central de la imagen de salida para cada submatriz. Con todo ello, se creará una nueva matriz que será la salida de aplicar ese filtro a una imagen. Todo este proceso descrito recibe el nombre de convolución.

Se pueden concatenar distintas capas convolucionales de manera que los datos de salida de una, sean usados como entrada en la siguiente. Es así como se consiguen detectar patrones más complejos al pasar de capa en capa. Por otra parte, conforme más filtros tenga una capa, más patrones detectará. Los filtros de las primeras capas serán más simples ya que detectan formas más simples.

El primer parámetro de una capa convolucional es su número de filtros, y el segundo el tamaño de esos filtros, en función del tamaño del patrón que se quiera detectar en los filtros de esa capa, tendrán un tamaño u otro.

4.2.1.2. Zero Padding

Al realizar la convolución, el filtro no puede salir de la imagen original y por lo tanto, siempre quedará una zona en la periferia de la misma que no será procesada y consecuentemente, los bordes desaparecerán. En la figura 4.2 se puede ver como un filtro reduce el tamaño de unos datos de entrada.

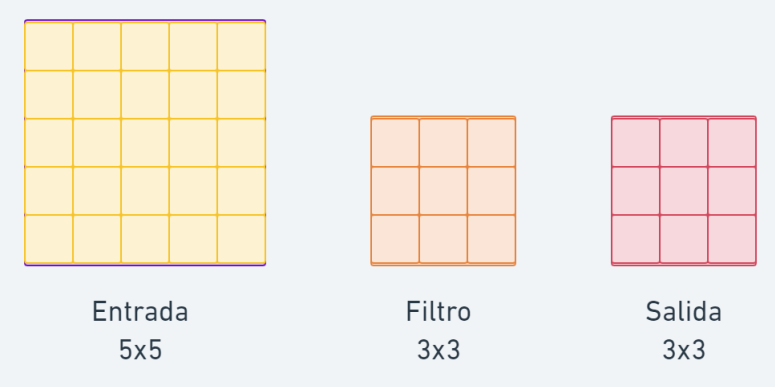

Figura 4.2: Efecto en el tamaño de una imagen al aplicarle convoluciones.

Se puede calcular como reduce un filtro convolucional a una imagen con la siguiente fórmula: si una imagen tiene un tamaño nxn y se le aplica un filtro de fxf, la salida de aplicar este filtro va a tener un tamaño de $(n-f+1)x(n-f+1)$. Para el caso de ejemplo: $(5-3+1)x(5-3+1) = 3x3$.

Esto puede ser un problema ya que conforme las capas se aplican a la imagen se va a ir haciendo más pequeña y se va a perder información útil. Por esto, se crea el *Zero Padding*, una técnica que permite preservar el tamaño de la imagen inicial. Una vez especificado el número de filtros que se quieren por capa y su tamaño, se puede especificar si se quiere *Zero Padding* o no en la capa. Por lo tanto, el cuarto parámetro de una capa convolucional es activar o no el padding.

Básicamente *Zero Padding* pone un borde de zeros alrededor de la imagen, de ahí su nombre. Esto permite tener una salida con el mismo tamaño que la imagen, ya que los bordes que se eliminan al aplicar convoluciones son los del zero padding añadidos. A veces, se necesita añadir más que un borde de zeros y esto va a depender del tamaño de los filtros y lo que eliminen del borde de la imagen.

4.2.2. Otras capas

4.2.2.1. Max Pooling

A la salida de una capa convolucional se le puede aplicar la operación *Max Pooling*, que consiste en utilizar una matriz de nxn y un desplazamiento de valor m. Lo que hace es recorrer la matriz buscando submatrices de tamaño nxn, distanciadas un número m de columnas y de filas. Para cada submatriz con el tamaño se escoge su máximo. El resultado da una matriz más pequeña que tiene los valores máximos de la matriz inicial. Con esto la dimensión se reduce mucho.

La razón para hacer *Max Pooling* es la reducción del tiempo computacional al tener matrices más pequeñas, y un mejor entrenamiento al coger los valores más altos y preservarlos, a la vez que se descartan los más bajos. *MaxPooling* también tiene *padding*. Al finalizar una capa convolucional, suele haber una capa *MaxPooling* para reducir la dimensionalidad de los datos y mejorar el cómputo.

4.2.2.2. Flatten

La capa *Flatten* se utiliza para poder pasar el vector de características a la última *Dense*, que necesita un vector en vez de una matriz. Esta capa se utiliza después de la última convolucional para poder devolver una salida a las últimas y empezar la clasificación.

4.2.2.3. Dropout

Cuando un modelo se entrena con datos muy pobres cabe la posibilidad de que se produzca sobreajuste. Esto significa que el modelo clasifica bien solo las muestras que se pasan para el conjunto de entrenamiento, es decir con el que entrena. Sin embargo ante otras muestras no clasifica tan bien. Esto se puede solucionar teniendo datos más fiables o combinando la decisión de varias redes. *Dropout* tiene el objetivo de reducir el sobreajuste de una red. Al añadir una capa *Dropout* se van descartando algunos nodos de forma aleatoria en cada ronda, en base a una frecuencia especificada. Esto permite reducir el sobreajuste, pues de algún modo al hacer este descarte aleatorio se simula que hay varias redes neuronales en un mismo modelo. En las diferentes rondas de una red con *Dropout* se dispone de unos nodos u de otros de forma aleatoria y el ajuste de los pesos es menos subjetivo debido a ello.

Esta capa no es obligatoria para las CNNs pero si puede llegar a mejorar mucho sus resultados.

4.2.3. Otros parámetros

Al igual que en el apartado 4.1.3, las redes neuronales convolucionales necesitan una serie de parámetros adicionales para ejecutar sus modelos. Los más recomendados son:

- **Optimizadores:** para la ejecución y entrenamiento de las redes neuronales convolucionales se recomienda el optimizador *Adam*, que adapta el aprendizaje de la red a la distribución de los parámetros.
- **Función de pérdida:** según el objetivo de la red, se debe de utilizar una u otra función de pérdida. En el caso del actual proyecto, al ser un problema de clasificación multiclase, se utiliza la función *Cross-Entropy*. Si las etiquetas se dan como número o como *one_hot* se usa la función *Sparse_Categorical_Crossentropy* o solo *Categorical_Crossentropy*.
- **Rondas:** no hay un valor recomendado, hasta que la red siga aprendiendo es la mejor opción.

4.2.4. Tipos de CNNs

Para entender mejor la complejidad de los diseños, se ha hecho un estudio de diferentes arquitecturas de CNNs para ver la influencia que tiene sobre la misma el añadir o quitar ciertos componentes. Para ello, resulta de gran ayuda la división que de las CNNs se realiza en [12] y que se resume a continuación.

- **CNNs basadas en explotación espacial:** las CNNs tienen muchos parámetros, entre ellos el tamaño de los filtros. Este tipo de redes se basa en alterar el tamaño de los mismos. Teniendo en cuenta que su tamaño será directamente proporcional al de los patrones que se desea extraer. Una de las arquitecturas más novedosas de este tipo de CNN es *GoogleNet*, que utiliza filtros de diferentes tamaños para captar información espacial basadas en la arquitectura *Inception* [34]. A su vez, utiliza transformaciones convolucionales multiescala.
- **CNNs basadas en profundidad:** este tipo de CNN se basa en la idea de que con el incremento de la profundidad, la red puede aproximar mejor la función objetivo. Es decir, se trata de potenciar que las redes sean lo más largas posibles y contengan un mayor número de capas. Algunas de las arquitecturas más novedosas de este tipo son: *Inception V3*, *Inception V4* [35] y Res-Net [36]. Además, este tipo de arquitectura trata de reducir el coste computacional sin afectar a la generalización de las características, lo que se consigue cambiando filtros grandes por pequeños y asimétricos.
- **CNNs basadas en múltiples rutas:** este tipo de redes utilizan multicaminos. Este concepto consiste en conectar una capa con otra, saltándose algunos pasos intermedios para permitir el flujo de información especializada a través de las capas. Esto particiona la red en varios bloques y permite el acceso de ciertas características a las capas más bajas. Un ejemplo de arquitectura de este tipo es *DenseNet* [37], y es creada para resolver el problema del *desvanecimiento de gradiente* comentado previamente.
- **CNNs basadas en multi conexiones en ancho:** este tipo de redes se basa en la importancia del ancho de una red a la hora de definir principios del aprendizaje. A más anchura, se incrementa el poder de aprendizaje de una red, a más largaría, se aprenden diversas representaciones de características. Se plantean redes más anchas, para poder aprender características más útiles al aumentar el poder de aprendizaje de la red. Una de las arquitecturas de este tipo es *ResNeXt* [38], que introduce el concepto de cardinalidad. Esto es una dimensión adicional que añade transformaciones como pueden ser funciones de separación, transformación y fusión (*Split-transform-merge*).

- **CNNs basadas en la explotación de Feature-Map:** las CNNs extraen características mediante el ajuste de los pesos asociados a una matriz. Sin embargo, también utilizan múltiples etapas, conocidas como *Feature-Maps*, encargadas de la extracción de características. Por lo tanto, no solo es importante la ingeniería de la red, sino el buen uso de los mapas de características. Una arquitectura de este tipo la podemos encontrar en *Competitive Squeeze and Excitation Networks* [39]. Esta arquitectura recalibra los mapas de características basándose en su contribución a la discriminación de clases.

- **CNNs basadas en la explotación de canal:** en ocasiones, las CNNs están limitadas por la calidad de las imágenes que se les proporcionan. Es por ello, que se plantea la mejoría de canal. Consiste en crear canales artificiales extra en las imágenes y así mejorar las representaciones de las redes, haciéndolas menos dependientes de las imágenes. Una de las arquitecturas más novedosa de este tipo es *Channel Boosted CNN* [40], que usa *Transfer Learning*, consistente en aprovechar la información aprendida en problemas anteriores.

- **CNNs basadas en la atención:** a la hora de extraer información por parte de las redes, es importante centrarse en características relevantes. Este concepto se conoce como atención. Al centrarse en ciertas partes de las imágenes, los problemas computacionales disminuyen y se mejora el reconocimiento de objetos en fondos complejos. Una de las arquitecturas más novedosas de este tipo es *Concurrent Spatial and Channel Excitation Mechanism* [41]. Esta arquitectura mezcla información espacial con mapas de características para que la red sea aplicable a tareas de segmentación.

Tras estudiar todas estas arquitecturas, el efecto y la finalidad que tienen sus capas, ya se pueden hacer decisiones de diseño para crear un propio modelo basadas en hechos probados. Todo ello, es de gran utilidad para el futuro capítulo 6 donde se diseña una arquitectura de *CNN* propia.

4.2.5. CNN básica

Organizar de forma correcta todas los conceptos anteriores es necesario para crear un buen modelo de red, existiendo muchas formas de distribuir las capas. Se ha de estudiar el problema y crear una solución de diseño ajustado a él. Un ejemplo de arquitectura básica se encuentra en la figura 4.3 que ha sido diseñada siguiendo la sintaxis de Keras.

```
model = Sequential([
    Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding = 'same', input_shape=(416,275,1)),
    MaxPool2D(pool_size=(2, 2), strides=2),
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same'),
    MaxPool2D(pool_size=(2, 2), strides=2),
    Flatten(),
    Dense(units=16, activation='softmax')
])
```

Figura 4.3: Ejemplo de distribución de la arquitectura

Cuenta con una capa de entrada que dispone de filtros de tamaño pequeño y que recoge los datos, por eso tiene un tamaño de entrada de 416x275 *pixeles*, otra capa oculta de filtros pequeños y una de salida con la función *softmax* que en base a las diferentes salidas crea una distribución de probabilidad para clasificar las 16 clases del problema, en base a esos porcentajes. El tamaño de los filtros ha sido escogido de $3x3$ ya que es el

tamaño más estándar para encontrar patrones pequeños y, para los datos, que no tienen objetos de grandes dimensiones, parece lo más apropiado.

Tiene esta serie de capas para simplificar lo más posible esta primera versión, el número de nodos por capa es menor en las primeras debido a que detectan patrones más simples y la función de activación *ReLu* al ser la recomendada. Como se puede ver, la implementación en Python de los componentes previamente estudiados es bastante intuitiva. Este diseño se utilizará como punto de partida para las distintas soluciones descritas en el capítulo 6.

CAPÍTULO 5
Procesamiento de imágenes y generación de bases de datos

Aparte de las normalizaciones típicas a las que deben someterse las imágenes para poder entrenar una red convolucional, existen un conjunto de procesos propios del problema del reconocimiento gestual. En el presente capítulo se explican los que se han realizado.

5.1 Segmentación de la mano

Las imágenes que entrega el dispositivo *Leap Motion* no solo incluyen el gesto de la mano, además del brazo, en ellas aparecen otros elementos del fondo que pueden ser captados por las cámaras. Al disponer de una lente gran angular no es raro que aparezcan en la escena la cara del usuario e incluso la parte superior del cuerpo. La figura 5.1 muestra (en rojo) los elementos indicados.

Figura 5.1: Ruido en las imágenes de Leap.

Otro aspecto a considerar es que, en general el brazo no aporta ninguna información que caracterice los gestos y por lo tanto, sería de gran ayuda poder eliminarlo de la imagen, sobre todo si se tiene en cuenta que puede aparecer algún objeto, como por ejemplo una pulsera, que podría llegar a confundir a la red (figura 5.1 en azul).

5.1.1. Umbralización. Método de Otsu

Para poder filtrar los elementos indeseados, lo más sencillo es recurrir a una umbralización de manera que se encuentre un intervalo de niveles de gris que caracteriza la mano. Después de intentar con distintos métodos de análisis del histograma sin resultados válidos, se ha optado por recurrir al algoritmo de *Otsu* que es sobradamente conocido

en la literatura [42] y que se basa en técnicas estadísticas. Concretamente, utiliza la varianza para caracterizar la dispersión de los niveles de gris de la imagen de manera que el valor umbral se establece de forma que la dispersión dentro de cada clase sea lo más pequeña posible. Por otra parte, se trata de mantener lo más alta posible la dispersión entre clases diferentes.

A continuación se muestra un primer algoritmo de segmentación:

- Calcular el umbral de Otsu.
- Obtener la imagen binaria resultante.
- Utilizar la imagen binaria como máscara para filtrar de la imagen original los píxeles pertenecientes a objetos no deseados.

La figura 5.2 muestra el resultado de aplicar el algoritmo anterior a la imagen de la figura 5.1. A la izquierda se observa el resultado de aplicar la umbralización de *Otsu* y a la derecha la imagen filtrada final. Hay que notar que la imagen resultante tiene el mismo tamaño que la original

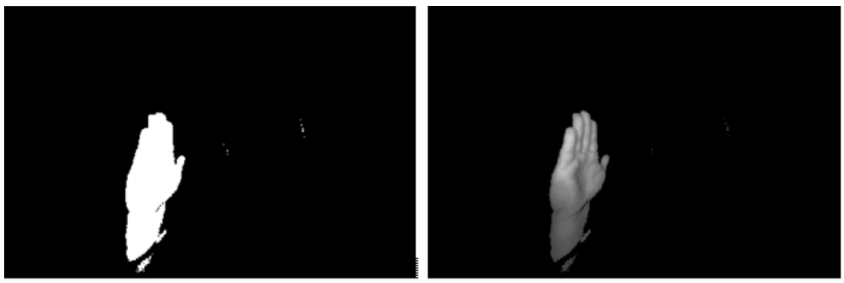

Figura 5.2: Resultado del primer algoritmo de umbralización.

Puede darse un paso más allá para quedarse con una imagen lo más centrada en la mano. Esto permitirá que la red se concentre en los detalles relevantes del gesto. Para ello se propone un segundo algoritmo:

- Calcular el umbral de Otsu.
- Obtener la imagen binaria resultante.
- Calcular el *Bounding Box* de los objetos binarios presentes.
- Utilizar el mayor de ellos como máscara de extracción de la imagen de interés.

La figura 5.3 muestra el resultado de los dos últimos pasos del segundo algoritmo.

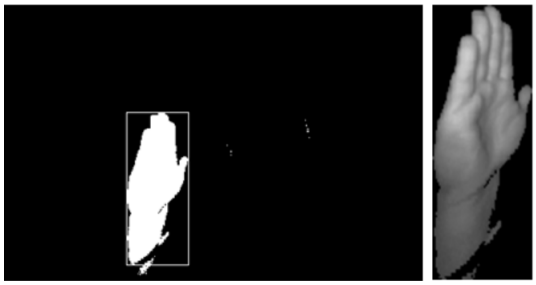

Figura 5.3: Resultado del segundo algoritmo de umbralización.

5.1.2. Recorte y escalado

A efectos prácticos, el algoritmo anterior presenta el inconveniente de que las imágenes resultantes tienen distinto tamaño que la original. Esto provocará la necesidad de realizar un escalado de las mismas con la consiguiente distorsión del gesto. Además, en ocasiones se produce el hecho de que el algoritmo de *Otsu* no de un buen resultado y se pierda parte de la mano. Para evitar estos inconvenientes, se realiza una modificación del mismo consistente en tomar un área de seguridad alrededor del *Bounding Box*. Para que el gesto de la mano entre en la imagen se puede reducir el tamaño global de ella a la cuarta parte de la original. Esto se puede garantizar tomando como referencia la esquina superior derecha del rectángulo original del *Bounding Box* para realizar a partir de ella el centrado en la imagen resultante. La figura 5.4 muestra el resultado de esta modificación.

Figura 5.4: Imágen final mejorada.

5.2 Eliminación de la distorsión de la imagen

Según puede consultarse en la página web del desarrollador de *Leap Motion* [43] las imágenes capturadas presentan una importante distorsión. Su efecto se puede observar en la figura 5.5 en la que se superpone una rejilla para remarcar su complejidad.

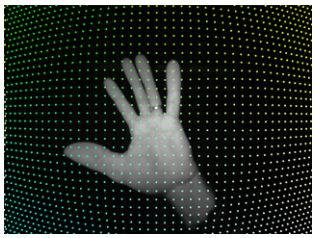

Figura 5.5: Distorsión.

Este hecho se destaca en el estado del arte del artículo de referencia [2] ya que va a provocar una alta varianza intraclase debida a los cambios de apariencia de las manos según su posición en la imagen. Por ello, muchos trabajos no utilizan las imágenes obtenidas del Leap sino únicamente el esqueleto ya procesado por el software del mismo y que ya se encuentra libre de distorsión. Un ejemplo es el trabajo [26] que combina *Leap* con imágenes capturadas mediante Kinect, el cual entregan directamente el mapa de rango libre de distorsión.

Otros trabajos combinan Leap con acelerómetros [44] o con un guante sensorizado [45]. El trabajo de referencia [2], propone una estrategia alternativa para minimizar los efectos de la distorsión basada en una estructura de clasificadores de dos capas. La primera construye una estructura de clasificador multiclase utilizando una configuración de clasificadores binarios de uno contra todos (cada uno enfocado en una clase). La segunda capa implementa cada clasificador binario anterior como un banco de *SVM* con el propósito de que cada *SVM* lidie con un subrango de apariencias visuales de una pose de

mano determinada. Una última posibilidad es corregir la distorsión como se puede ver en [46] donde se aplica un proceso de rectificación geométrico de imagen. En esta línea se trabaja a continuación.

Cuando un haz de luz entra en una de las cámaras *Leap Motion*, la lente lo curva para que alcance el sensor, que lo registra como un valor de brillo en escala de grises en una ubicación de píxel específica. Como ninguna lente es perfecta, el haz de luz no llega al sensor en el lugar ópticamente perfecto. Se requiere de un mapa de calibración que proporcione datos para corregir esta imperfección, calculando el ángulo real del haz de luz original.

La imagen 5.6 muestra una reconstrucción de la imagen con corrección de distorsión. El valor de brillo de cada píxel en la imagen se calcula a partir de un haz de luz que ingresa a la cámara desde una dirección específica. La imagen se reconstruye calculando las pendientes horizontales y verticales representadas por cada píxel y encontrando el valor de brillo real a partir de los datos de la imagen utilizando el mapa de calibración. Las partes rojas de la imagen representan áreas dentro del renderizado para las que no hay ningún valor de brillo disponible.

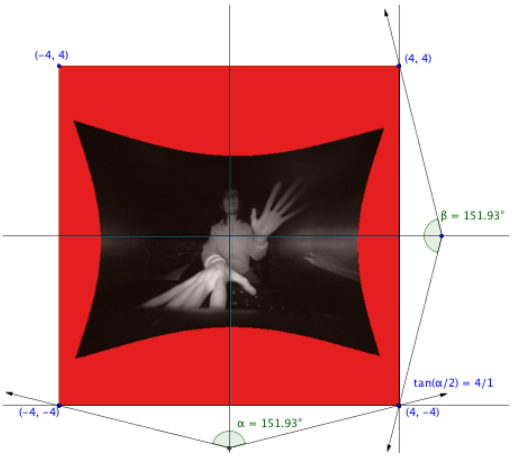

Figura 5.6: Calibracion.

5.2.1. Mapa de calibración

El mapa de calibración es una cuadrícula de puntos de 64x64 que se extrae de *Leap*. Cada punto consta de dos valores de 32 bits, uno por cámara, por lo que el tamaño del búfer es $128x64x4$. Cada punto del búfer indica dónde encontrar el valor de brillo corregido para el píxel correspondiente en la imagen sin procesar. Las coordenadas válidas se normalizan en el rango [0..1]. Los elementos individuales del mapa de calibración pueden tener un valor en el rango [-0,6..2,3], pero las coordenadas por debajo de cero o por encima de 1 no son válidas. Para convertir a coordenadas de píxeles, se multiplica por el ancho y alto de la imagen. Para los píxeles que se encuentran entre los puntos de la cuadrícula de calibración, se interpola entre los puntos de la cuadrícula más cercanos. Como se ha comentado anteriormente, no todos los puntos de la cuadrícula de calibración se asignan a un píxel válido como puede verse en la imagen 5.7. A la izquierda los valores de x y a la derecha los de y.

Se puede corregir la distorsión de dos formas, o bien se utilizan funciones de la librería de *Leap (Image_warp e Image_rectify)* o bien se accede directamente al buffer de distorsión de la imagen. La primera opción es la más sencilla pero también la más lenta, por lo que en el presente trabajo se accede directamente al buffer de distorsión y se utiliza la función cv2.remap() de *OpenCV* para hacer la interpolación bilineal.

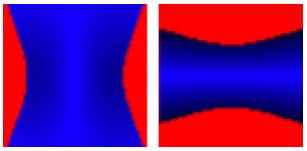

Figura 5.7: Cuadricula de calibracion.

Hay que considerar que la base de datos disponible no viene acompañada de su correspondiente mapa de calibración por lo que la única posibilidad ha sido tomar el mapa del dispositivo *Leap* propio, que se dispone para el actual proyecto, y aplicarlo a las imágenes de la base de datos. Es obvio que existirán diferencias entre las ópticas de ambos dispositivos pero se aplica el proceso a todas las imágenes con la esperanza de obtener alguna ventaja del mismo. Es cierto que, en la mayoría de imágenes, el proceso de corrección no se aprecia a simple vista pero sin embargo en otros casos es bastante visible sobre todo si la mano se encuentra sesgada respecto del centro de la cámara. Tal es el caso de la figura 5.8 que muestra respectivamente un ejemplo de imagen original y su versión corregida como se ha indicado anteriormente.

Figura 5.8: Imágen distorsionada (izquierda) y corregida tras aplicar la matriz de calibración (derecha).

El proceso de corrección de la distorsión está muy relacionado con la obtención del mapa de profundidad, que se estudiará a continuación.

5.3 Mapa de profundidad

Este apartado se refiere al proceso por el que, a partir de las dos imágenes captadas por el Leap (L y R), se infiere información de más alto nivel resultante de combinarlas. La hipótesis de partida es que la red neuronal pueda extraer características discriminantes a partir de esta nueva información.

5.3.1. Combinaciones de las imágenes L y R

Una de las primeras pruebas consiste en combinar las dos imágenes en una sola. El atractivo de esta opción es la sencillez en obtenerla y se pretende comprobar si la red neuronal es capaz de inferir alguna nueva propiedad a partir de esta sencilla combinación. Se proponen dos variantes:

- Apilarlas una al lado de otra, por ejemplo la L arriba y la R abajo como se puede ver en la figura 5.9.

- Situarlas como canales independientes en una imagen multicanal. Por simplicidad se recurre al uso de una imagen resultante RGB dejando el canal rojo a cero y situando la imagen izquierda en el canal azul y la derecha en el rojo. Un ejemplo de dicha combinación se puede ver en la figura 5.10.

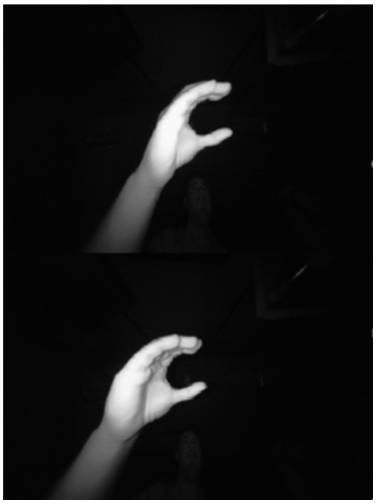

Figura 5.9: Imágen L y R en vertical.

Figura 5.10: Imágen L y R de dos canales.

5.3.2. Cálculo de la disparidad

El verdadero proceso de combinación de las imágenes L y R tiene por objeto la obtención del mapa de rango de la escena que consiste en una imagen en niveles gris (del mismo tamaño que L y R). En esta, se indica para cada píxel un valor equivalente a la distancia del mismo a la cámara, es decir, su coordenada Z. El primer paso consiste en calcular la disparidad que se define como la diferencia entre las coordenadas X de dos píxeles que se corresponden al mismo punto. Es por ello, muy importante contar con la correcta calibración de las imágenes L y R tal y como se ha estudiado en el apartado anterior 5.2. A continuación en la figura 5.11 se muestra la geometría involucrada en este proceso.

Aplicando triángulos equivalentes tenemos $Disparidad = x - x' = BfZ$.

Siendo x y x', la distancia entre el punto y el centro de la cámara en la imagen izquierda y derecha, respectivamente. Por su parte, B es la distancia entre las dos cámaras y f es la distancia focal de la cámara. Hay que notar que la profundidad de un punto en una escena es inversamente proporcional a la disparidad.

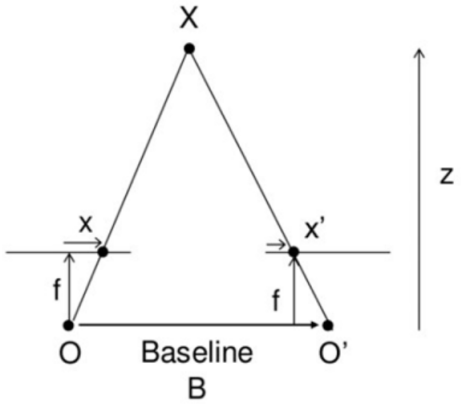

Figura 5.11: Disparidad.

La complejidad de un algoritmo de cálculo de disparidad consiste en averiguar en ambas imágenes qué píxeles pertenecen al mismo punto de la escena. La librería *OpenCV* dispone de dos funciones de cálculo de disparidad: *cv2.StereoBM_create* y *cv2.Stereo-SGBM_create*. En [28] se puede encontrar una librería que ejemplifica el uso de dichas funciones añadiendo un control para calibrar los parámetros. En el presente trabajo se ha comprobado que el primero de los algoritmos funciona mejor que el segundo. La figura 5.12 muestra un ejemplo de ejecución presente en la documentación. A la izquierda la imagen L y a la derecha el resultado.

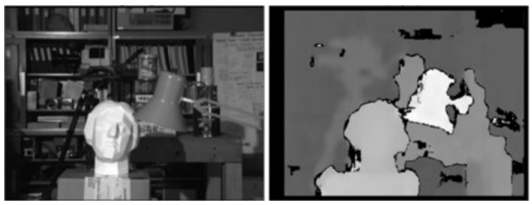

Figura 5.12: Cálculo de la disparidad.

A continuación se describen los parámetros más significativos y los valores que se han seleccionado de forma experimental para crear la base de datos con el algoritmo:

- **Umbral de textura:** filtra las áreas que no tienen una textura suficiente para poder relacionar los puntos en ambas imágenes. El umbral de textura se ha propuesto con un valor de 248.
- **Número de disparidades:** el rango de disparidad para la búsqueda. En cada pixel se encontrará la mejor disparidad posible desde 0 hasta el número de disparidad seleccionado. Se ha propuesto 32 como número para la disparidad.
- **Ratio de unicidad:** filtra los píxeles cuya disparidad encontrada no es mejor que la de cualquier otra encontrada en la búsqueda. Un ratio de unicidad de 70 se ha propuesto en la solución.
- **Tamaño del bloque:** es el tamaño de los bloques que compara el algoritmo. Debe de ser impar. Un bloque grande supone un mapa de disparidad más suave pero menos preciso y un bloque pequeño, supone un mapa más preciso pero con más probabilidad de fallo. Se ha seleccionado el valor 5 para el tamaño del bloque.

A continuación, la figura 5.13 muestra una imagen junto al entorno utilizado para calibrar los parámetros más adecuados en el cálculo del mapa de rango en tiempo real.

Para su diseño se ha combinado la forma de calcular el mapa de disparidad de [28], adaptándola en dos sentidos: el primero en lo que respecta al algoritmo de corrección de la distorsión de la imagen que se ha tomado de [43] y el segundo para el control en tiempo real del *Leap* hecho a partir del *Wrapper* [29].

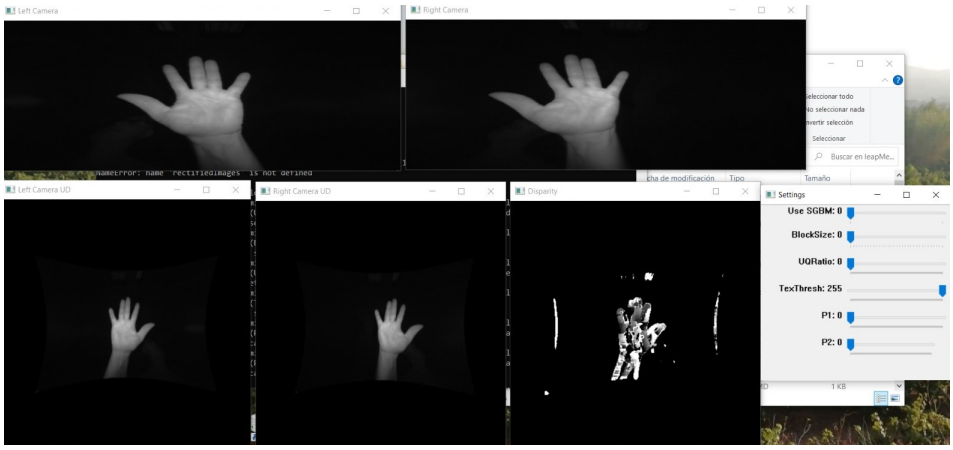

Figura 5.13: Entorno de pruebas para la creación del mapa de disparidad.

En la parte superior se observan las imágenes L y R tomadas directamente del *Leap*, mientras que en la parte inferior se muestran ambas imágenes corregidas junto con el mapa de rango resultante.

Por último, en la figura 5.14 se puede ver un ejemplo del cálculo de la disparidad.

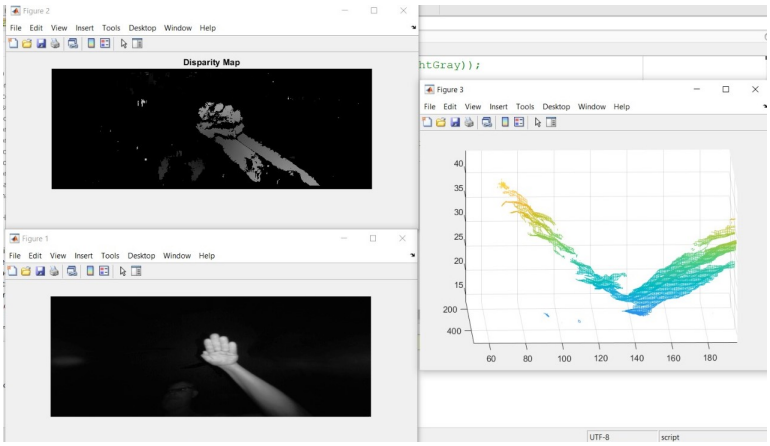

Figura 5.14: Ejemplo de cálculo del mapa de disparidad.

Abajo a la derecha se puede observar la imagen L, arriba a la izquierda el mapa de disparidad y a la derecha una visualización 3D de este último.

5.4 Procesamiento del esqueleto

Tal y como se ha comentado en el punto 3.4 se cuenta con una base de datos de esqueletos. En [47] aparece explicada toda la información necesaria para poder interpretar los ficheros xml disponibles. Como se trabaja con CNNs, se ha decidido transformar dicha información estructural a un formato de imagen. Aunque tiene mucha información, solo se va a utilizar la posición de los dedos y sus puntos clave, así como el centro de la mano para situarla. Estos puntos clave se pueden ver en la figura 5.15 separados por colores y secciones para cada dedo. La imagen esta sacada de la página oficial de *Leap Motion* [47].

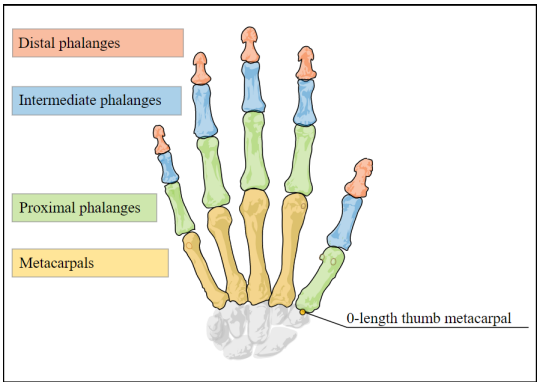

Figura 5.15: Información utilizada de la mano

A continuación se presentan varias alternativas a la hora de transformar los esqueletos en imágenes de menor a mayor complejidad.

5.4.1. Esqueleto blanco y negro

En esta primera versión se crean imágenes en blanco y negro de las manos señalizando los puntos clave. Para ello, se extrae la información solo de las posiciones de los puntos clave, recorriendo los ficheros xml y se dibujan las imágenes con los mismos valores en blanco para todos los puntos. Un ejemplo de muestra para la clase *Palma* se puede ver en la figura 5.16.

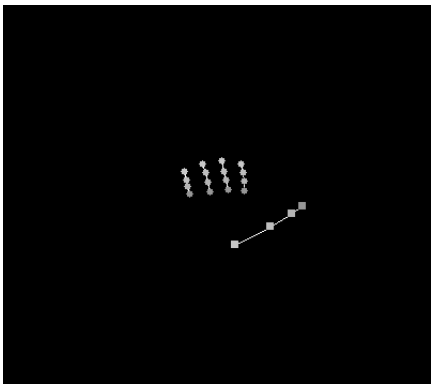

Figura 5.16: Imágen del esqueleto en blanco y negro

5.4.2. Esqueleto en escala de grises

Esta segunda versión crea mapas de rango solo de los puntos relevantes a través de la información de los ficheros. Para ello, se extrae la información X, Y y Z y se dibujan las imágenes representando en la coordenada (X, Y) un píxel cuyo nivel de gris se corresponde a la coordenada Z de cada punto. De este modo se tiene información adicional de la cercanía de los puntos a la cámara. La imagen resultante de este método se puede ver en la figura 5.17.

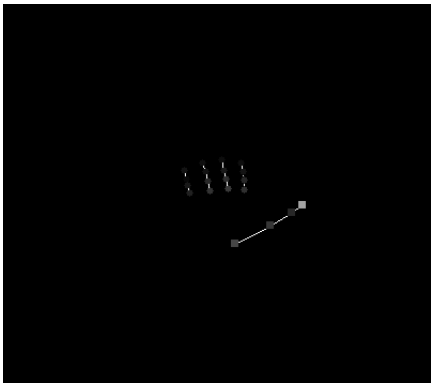

Figura 5.17: Imágen del esqueleto en escala de grises

5.4.3. Esqueleto a color

La tercera versión, crea imágenes en color a partir de la información de los ficheros. La ventaja que tiene, es que puede diferenciar cada dedo por un color diferente y proporcionando más información discriminante. El proceso es exactamente igual que en la primera versión, solo que en vez de dibujar los puntos en blanco, se dibuja cada uno con el color correspondiente al dedo al que pertenecen. En la figura 5.18 se puede comprobar la diferenciación de dedos por colores.

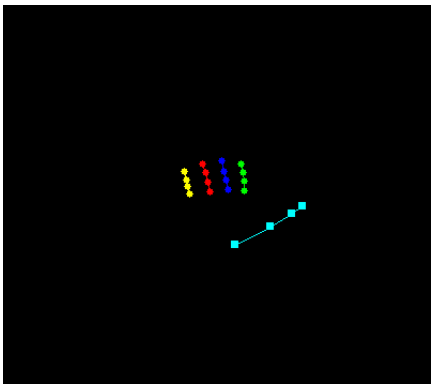

Figura 5.18: Imágen del esqueleto a color

5.4.4. Esqueleto a color con información 3D

Exactamente igual que en la segunda versión, en esta cuarta, se extrae la información 3D de los ficheros para crear imágenes a color con información adicional de la cercanía de los puntos respecto a la cámara. También se diferencian los dedos por colores y la imagen 5.19 es una muestra del uso de este método.

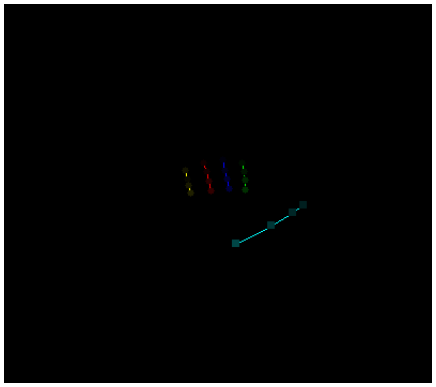

Figura 5.19: Imágen del esqueleto a color 3D

5.4.5. Mejora del esqueleto

Para añadir más información y mejorar las muestras provenientes del esqueleto, se ha propuesto diferenciar los dedos dibujando sus puntos en base a tres diferencias.

- **Dedo estirado:** para diferenciar los dedos estirados se han dibujando con círculos sus puntos clave.
- **Dedo no estirado:** los dedos no estirados se han dibujado con triángulos en sus puntos clave para diferenciarlos del resto.
- **Pulgar:** el pulgar, al ser un dedo diferente del resto siempre se ha pintado con cuadrados en sus puntos clave.

Estos cambios añaden mucha más diferenciación y variedad de características que la red podría aprovechar. En la figura 5.20 se pueden observar todas estas figuras geométricas para la clase *Índice*.

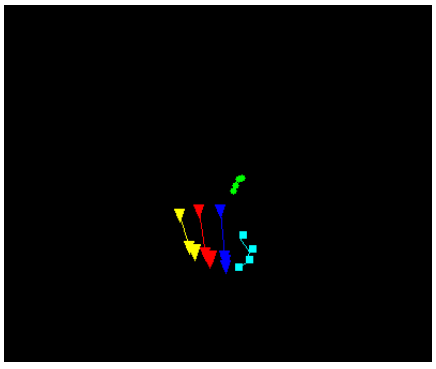

Figura 5.20: Imágen a color de la clase Índice

5.5 Bases de datos

Con todos los algoritmos de procesamiento y segmentación de imágenes desarrollados ya se pueden construir las bases de datos que se utilizarán en los experimentos de los capítulos 6 y 7. A continuación se muestra una tabla 5.1 con todas las bases de datos resultantes, una pequeña descripción de ellas, su origen y el conjunto de datos al que pertenecen tal y como se enumeraron en 3.4.

Conjunto	Base de datos	Descripción	Algoritmo que la genera
Datos IR	BD-IR	Base de datos original	3.3
	BD-Otsu	Base de datos original sin ruido de fondo	5.1.1
	BD-Crop	Base de datos original recortada	5.1.1
	BD-OtsuCrop	Base de datos original sin ruido de fondo y recortada	5.1.2
	BD-IRUD	Base de datos original con distorsión corregida	5.2.1
	BD-CropUD	Base de datos original con distorsión corregida y recortada	5.2.1, 5.1.1
Datos LR	BD-Concat	Base de datos incluyendo la unión vertical de LR	5.3.1
	BD-CH2	Base de datos con dos canales LR	5.3.1
	BD-RM	Base de datos con mapa de rango	5.3.2
Datos SK	BD-SK	Base de datos con esqueleto convertido a imagen	5.4.1
	BD-SK3D	Base de datos con esqueleto convertido a imagen 3D	5.4.2
	BD-SKC	Base de datos con esqueleto convertido a imagen en color	5.4.3
	BD-SKC3D	Base de datos con esqueleto convertido a imagen 3D en color	5.4.5

Tabla 5.1: Conjunto de bases de datos utilizadas en los experimentos

CAPÍTULO 6
Diseño del clasificador de gestos

Con las bases de datos ya creadas, el siguiente paso consiste en elegir una arquitectura para el modelo del clasificador. Se establecen dos caminos para seleccionar un modelo final: la creación de un modelo propio y el uso de redes pre-entrenadas. A continuación se desarrollan ambas aproximaciones. En todos los casos y por simplicidad se han utilizado un número de rondas igual a 10 en el entrenamiento.

6.1 Red neuronal Convolucional propia

En base a todo lo aprendido se puede abordar el diseño de un conjunto de modelos teniendo en cuenta el problema del reconocimiento de gestos manuales estáticos. Para el entrenamiento de estos modelos se han utilizado los conjuntos de datos provenientes de IR: BD-IR, BD-Otsu, BD-Crop, BD-OtsuCrop. Se han elegido éstos al considerarse los más representativos del problema. El resto de conjuntos restantes son creaciones que pueden tener ruidos y no tienen porqué funcionar bien. Así que se ha preferido no incluir ese posible mal funcionamiento en el diseño a la hora de probar las redes y escoger una.

- **Primera versión:** primero se ha creado una versión sencilla de CNN para ver qué resultados dan las bases de datos y si los datos se han recogido correctamente. De hecho la red es la misma que se mencionó anteriormente en la figura 4.3. Es muy estándar pero es una base de la que partir. En la tabla **??** se puede ver para cada base de datos el porcentaje de acierto obtenido en cada conjunto de datos.

Base de datos	Entrenamiento	Validación	Test
BD-IR	99.81 %	86.04 %	69.85 %
BD-Otsu	99.65 %	85.88 %	70.11 %
BD-Crop	99.52 %	86.20 %	70.35 %
BD-OtsuCrop	99.70 %	86.96 %	70.52 %

Tabla 6.1: Resultados con primera versión.

Como puede observarse, los resultados no son muy buenos, sobre todo al presentar datos nunca vistos por la red, en torno al 70 % de acierto. Aparte, se observa que la red está bastante sobreentrenada debido a que tiene muy pocas capas y filtros con poca variedad. Se deduce que necesita más profundidad y diversidad de filtros para aprender un mayor número de patrones. También puede deducirse de este resultado que para la red propuesta no se observan diferencias significativas entre las bases de datos.

- **Segunda versión:** en la segunda versión se ha introducido para la primera capa un filtro de tamaño $(5, 5)$ para que detecte patrones mayores. Las primeras capas detectan patrones más simples por lo tanto, es lógico que primero detecta los más grandes con un tamaño de filtro mayor. Sin embargo, al ser superficies no muy grandes, como la mano, con un tamaño $5x5$ es suficiente. Por último, se ha añadido al final una capa *Dense* para dar más profundidad a la red y así aprender más características. En la tabla 6.2 se pueden ver los resultados de esta versión.

Base de datos	Entrenamiento	Validación	Test
BD-IR	99.64 %	94.19 %	86.69 %
BD-Otsu	99.54 %	93.85 %	86.98 %
BD-Crop	99.37 %	94.26 %	87.28 %
BD-OtsuCrop	99.69 %	94.72 %	87.60 %

Tabla 6.2: Resultados con segunda versión.

Como se puede ver añadir más profundidad mejora considerablemente la red. Sin embargo, se necesita mejorar el sobreentrenamiento aún más. Al ser una red más compleja ya se empiezan a observar diferencias entre las bases de datos.

- **Tercera versión:** en la tercera versión se han introducido las mencionadas capas *Dropout* del capítulo 4. Estas capas van descartando ciertas neuronas cada iteración de forma diferente. El entrenamiento de los pesos de la red es realizado cada vez con un número de neuronas diferente, lo que reduce el sobreajuste. Se han añadido dos capas Dropout con una frecuencia de 0,5 cada una. En la tabla 6.3 aparecen los resultados de esta versión.

Base de datos	Entrenamiento	Validación	Test
BD-IR	92.80 %	93.18 %	89.79 %
BD-Otsu	92.44 %	92.91 %	89.94 %
BD-Crop	92.28 %	93.30 %	90.27 %
BD-OtsuCrop	93.72 %	94.07 %	90.68 %

Tabla 6.3: Resultados con tercera versión.

El uso de las dos capas *Dropout* ha mejorado considerablemente el sobreentrenamiento de la red. La respuesta ante el conjunto de test también ha aumentado significativamente, así como la respuesta ante las distintas bases de datos donde ya empieza a observarse el efecto del procesamiento.

- **Cuarta versión:** si se observan las arquitecturas de las redes mencionadas en el capítulo 4 se puede ver que muchas contienen capas de filtros con tamaño $1x1$. Los filtros con ese tamaño bien usados pueden:

  - Reducir la dimensionalidad.
  - Reducir el coste computacional
  - Añadir menos linealidad a la red.
  - Añadir más profundidad.
  - Añadir más precisión.

  En esta versión se han añadido dos capas convolucionales con filtros de tamaño $1x1$. En la tabla 6.4 aparecen los resultados.

Base de datos	Entrenamiento	Validación	Test
BD-IR	95.04 %	92.94 %	89.63 %
BD-Otsu	94.86 %	92.27 %	89.81 %
BD-Crop	94.43 %	93.04 %	90.06 %
BD-OtsuCrop	96.29 %	93.68 %	90.35 %

Tabla 6.4: Resultados con cuarta versión.

El uso de las dos capas con filtros de tamaño $1x1$ no ha sido muy positivo. Quitando una, debido a que la red no es tan grande y añadiendo otro tipo de capa se espera que mejoren los resultados.

- **Quinta versión:** en esta versión se ha eliminado una de las capas con filtros de tamaño $1x1$ y se ha añadido otra con filtros de $5x5$ en las capas finales de la red, con el objetivo de detectar patrones más grandes pero algo más complejos que al principio, al situarse en posiciones más profundas. En la tabla 6.5 se pueden ver los resultados de esta versión que han mejorado significativamente.

Base de datos	Entrenamiento	Validación	Test
BD-IR	97.36 %	97.54 %	94.86 %
BD-Otsu	96.93 %	97.12 %	95.55 %
BD-Crop	96.39 %	97.65 %	96.11 %
BD-OtsuCrop	98.38 %	98.49 %	96.89 %

Tabla 6.5: Resultados con quinta versión.

Con estas capas la red ya consigue resultados muy buenos y el sobreentrenamiento es pequeño, de un 2 % prácticamente. Ya se constata claramente que el procesamiento que da mejores resultados es el de *OtsuCrop* que mejora en dos puntos a la base de datos original. La estructura final en Python con todos los detalles comentados se puede ver en la figura 6.1.

```
model = Sequential([
    Conv2D(filters=32, kernel_size=(5, 5), activation='relu', padding = 'same', input_shape=(416,275,1)),
    MaxPool2D(pool_size=(2, 2), strides=2),
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same'),
    MaxPool2D(pool_size=(2, 2), strides=2),
    Dropout(0.5),
    Conv2D(filters=64, kernel_size=(1, 1), activation='relu', padding = 'same'),
    MaxPool2D(pool_size=(2, 2), strides=2),
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same'),
    MaxPool2D(pool_size=(2, 2), strides=2),
    Dropout(0.5),
    Conv2D(filters=64, kernel_size=(5, 5), activation='relu', padding = 'same'),
    MaxPool2D(pool_size=(2, 2), strides=2),
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same'),
    MaxPool2D(pool_size=(2, 2), strides=2),
    Flatten(),
    Dense(units=64, activation='relu'),
    Dense(units=16, activation='softmax')
])
```

Figura 6.1: Arquitectura final de la red propia

El número de filtros utilizado es el recomendado para la información que contienen nuestros datos, en las capas iniciales 32 y para las capas más profundas 64. Al tener nuestros datos poco ruido no son necesarias capas muy anchas, así que este tamaño está bien. Las funciones de activación también han sido las recomendadas como se menciona en el apartado 4.1.2 y el tamaño de la capa *MaxPool* es de un tamaño $2x2$ fijando $m = 2$. Se ha

implementado así para reducir la dimensionalidad de los datos a la mitad. El modelo se ha compilado con el optimizador de *Adam*, con un factor de aprendizaje de 0.0001, con una función de pérdida *categorical_crossentropy* y buscando la mayor precisión posible. En la figura 6.2 se puede ver la línea de código encargada de esto.

```
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

Figura 6.2: Parámetros para la compilación del modelo

Esta última versión ha sido nombrada como *GestNet8*, a lo largo de la presente memoria se utilizará este nombre para referirse a ella. En la figura 6.3 se puede observar la estructura y disposición de capas en un dibujo ilustrativo de *GestNet8*.

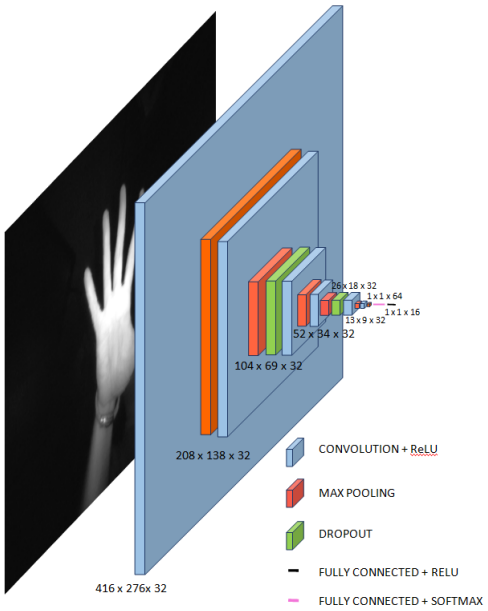

Figura 6.3: Estructura de GestNet8

6.2 Redes Neuronales Convolucionales pre-entrenadas

Otra alternativa a la hora de crear un modelo es utilizar uno previamente entrenado y adaptarlo al funcionamiento deseado. Esta opción también ha sido estudiada. Se han escogido cuatro redes, todas potentes, capaces de reconocer muchísimos objetos y detectar patrones de todos los tamaños. Las redes que han sido seleccionadas son: *InceptionV3*, *Xception*, *VG16* y *ResNet50*. Son redes de libre uso, se han escogido sólo estas cuatro para limitar la extensión del trabajo, sin embargo, existen gran cantidad de ellas. Todas ellas fueron creadas para participar en la competición conocida como *ImageNet Large Scale Visual Recognition Challenge*, que consiste en clasificar 1000 objetos de diferentes clases, provenientes de una base de datos llamada *ImageNet*.

Todas estas redes han sido adaptadas al problema, cambiando su última capa por una *Dense* con la activación *softmax*, para que clasifique las 16 clases del modelo. Por otra parte, el número de capas que se ha adaptado en el último tramo de cada red se ha escogido experimentalmente. Se ha decidido ejecutarlas solo con la BD-IR para evaluarlas ya que sus procesos computacionales son muy extensos. A continuación se presentan ordenadas de menor a mayor según los resultados obtenidos.

6.2.1. InceptionV3

Es un modelo bastante popular para reconocer imágenes, empezó como un módulo de *GoogleNet* en 2014. Las mejoras que incorpora son: suavizado de etiquetas, convoluciones factorizadas de tamaño 7x7 y la propagación de etiquetas en la parte inferior de la red.

En esta red se han entrenado las últimas 10 capas para que sus pesos se adapten al problema que se trata de resolver. El resto de capas conservan los pesos originales preentrenados. El comportamiento de *InceptionV3* en los tres conjuntos se puede ver en la tabla 6.6.

Entrenamiento	Validación	Test
91.26 %	90.62 %	87.28 %

Tabla 6.6: Resultados InceptionV3

El uso de esta red no parece adecuado para el problema del presente trabajo, encontrándose muy por debajo de *GestNet8*.

6.2.2. Xception

Xception es un modelo conocido por su profundidad, con unas 71 capas de largaría, creado en 2017. Introduce convoluciones separables en profundidad y parte de la idea de la red *Inception* y su módulo que reduce la dimensionalidad de los datos.

En esta red se han entrenado las últimas 60 capas para que sus pesos se adapten al problema. El resto de capas conservan sus pesos originales. El resultado de los experimentos con esta red puede verse en la tabla 6.7.

Entrenamiento	Validación	Test
94.19 %	92.18 %	87.42 %

Tabla 6.7: Resultados Xception

Tampoco parece que esta red haya conseguido adaptarse a los datos de los que se dispone.

6.2.3. VG16

Es una red bastante larga con una profundidad de 16 capas y unos 138 millones de parámetros. Fue creada en 2014 y mejora la red *AlexNet* cambiando filtros de grandes tamaños por múltiples filtros de tamaño 3x3 de forma consecutiva.

En el uso de esta red se ha decidido conservar todos sus pesos. Solo se ha adaptado la capa de salida. La tabla 6.8 muestra el resultado de la clasificación.

Entrenamiento	Validación	Test
96.56 %	94.72 %	90.99 %

Tabla 6.8: Resultados VG16

Como se puede observar, esta red tiene mejores resultados, aunque todavía se encuentra por debajo de *GestNet8*.

6.2.4. ResNet50

ResNet50 es una red con unas 50 capas de profundidad, muy popular en la actualidad para tareas de visión por computador y creada en 2015. Es una variante de la red *ResNet*, la propiedad que caracteriza este tipo de redes son sus conexiones, que saltan varias capas, permitiendo tener información más avanzada en las capas tempranas y viceversa.

En esta red se han entrenado las últimas 10 capas para que sus pesos aprendan de los 16 gestos. El resto de capas tienen sus pesos originales. En la tabla 6.9 se puede ver cómo clasifica esta red los tres conjuntos.

Entrenamiento	Validación	Test
99.54 %	97.66 %	94.26 %

Tabla 6.9: Resultados ResNet50

Se deduce, que al menos con los diseños seleccionados de entre las redes pre-entrenadas, la red *ResNet50* es la que mejor resultados obtiene para el problema que trata de resolver presente TFG. Para completar la comparación se realiza un último experimento con esta red y la base de datos que mejor porcentaje obtuvo en los resultados de *GestNet8*, la *BD-OtsuCrop*. La tabla 6.10 muestra los resultados.

Entrenamiento	Validación	Test
99.62 %	98.35 %	96.96 %

Tabla 6.10: Resultados de ResNet50 con datos de BD-OtsuCrop

Se puede observar que los resultados de *ResNet50* son bastante similares a los de *GestNet8* creada. A su vez, confirma que *BD-OtsuCrop* mejora significativamente la base de datos original *BD-IR*.

6.3 Discusión y elección de red

En base a los resultados, los candidatos a arquitectura para el proyecto son *GestNet8* y *ResNet50*. Es curioso observar, que una red con pocas capas, es capaz de tener los mismos resultados que una red con centenares de ellas. Esto seguramente sea debido a la naturaleza del problema. Hay que tener en cuenta que el dispositivo *Leap Motion* dispone de cámaras infrarrojas y una iluminación propia. Además, es capaz de adaptarse a las condiciones ambientales. Con todo ello, las bases de datos con las que se cuentan presentan una uniformidad, ya que aún perteneciendo a distintos sujetos, las características de reflectancia de las manos se captan de forma similar. Es presumible que la red pueda centrarse más en los contornos que en las regiones de la mano.

Se conoce como funciona la *ResNet50*, es muy profunda y detecta patrones pequeños, sin embargo, no se conoce el funcionamiento de *GestNet8*. Entenderla es necesario para saber qué sucede y eso es lo que se va a discutir a continuación.

6.3.1. Patrones

Como se explicó en el capítulo 4 los filtros, en una *CNN*, son los encargados de extraer patrones de las imágenes. Es posible visualizar los patrones que un filtro aprende en su entrenamiento para entender el funcionamiento de una red. Para ver las diferencias entre las redes, se va a estudiar primero la forma de los patrones que aprenden ambas. Cada capa aprende patrones diferentes, sin embargo con comparar un par de ellas es suficiente para sacar conclusiones sobre el detalle y la complejidad de las formas que infieren. En la figura 6.4 aparecen los patrones que aprende *GestNet8* en los primeros 25 filtros de su quinta y en la figura 6.5 los de *ResNet50* en los primeros 49 filtros de su última capa.

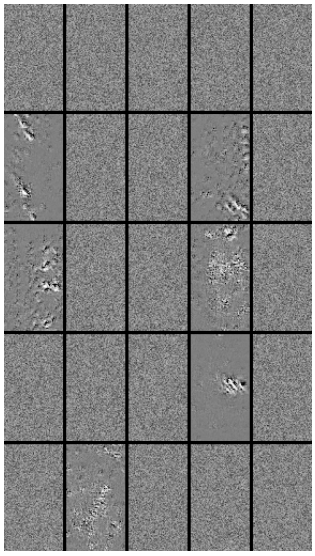

Figura 6.4: Patrones que aprende *GestNet8* en la quinta capa.

Para ser una capa bastante avanzada en la red, se puede ver que los patrones no son muy complicados ni contienen formas incomprensibles.

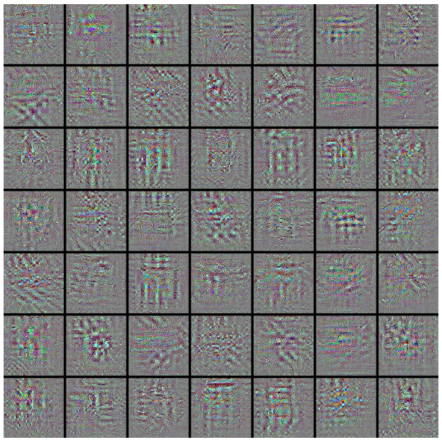

Figura 6.5: Patrones que aprende ResNet50 en su última capa.

En estos patrones si que se puede observar formas complejas y confusas en comparación a las otras.

De estos resultados se podría deducir que *GestNet8* recoge patrones mucho más sencillos y que con las formas poco texturizadas de las manos tiene suficiente para clasificar bien ya que el problema en cuestión se ajusta bien a este tipo de redes. Por otra parte, la *ResNet50* pese a detectar patrones muy complejos y formas complicadas también tiene buenos resultados. Como es evidente si se puede reconocer formas complejas, también simples. Esa es la hipótesis y a continuación se va a intentar reafirmar comprobando los mapas de características en las últimas capas, de ambas redes, para cierta muestra.

6.3.2. Feature-Maps

Dada una imagen de entrada en una red, las salidas resultantes de procesarla, en las capas convolucionales, se pueden visualizar. Estas salidas son conocidas como mapas de características y son útiles para ver los diferentes tipos de características que infiere una red dado un objeto. Contienen información relevante de la manera en la que los patrones que aprenden las redes son aplicados a una muestra y puede ayudar a encontrar soluciones en base a su análisis. Dada la muestra de la clase Palma que aparece en la figura 6.6, se han extraído los mapas de características para ambas redes en su última capa. En la figura 6.7 se pueden ver las características extraídas por *GestNet8* y en la figura 6.8 los de *ResNet50*.

Figura 6.6: Muestra de la clase Palma para analizar los mapas de características.

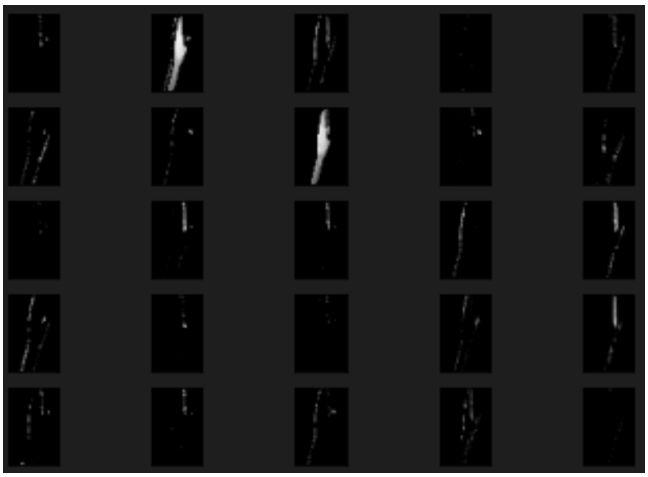

Figura 6.7: Mapas de características que aprende la propia red en la última capa.

Parece ser que las características que *GestNet8* necesita para clasificar son la forma y el color de la mano de manera muy general. Solo necesita los contornos como se puede ver y no extrae más características como texturas o fondos.

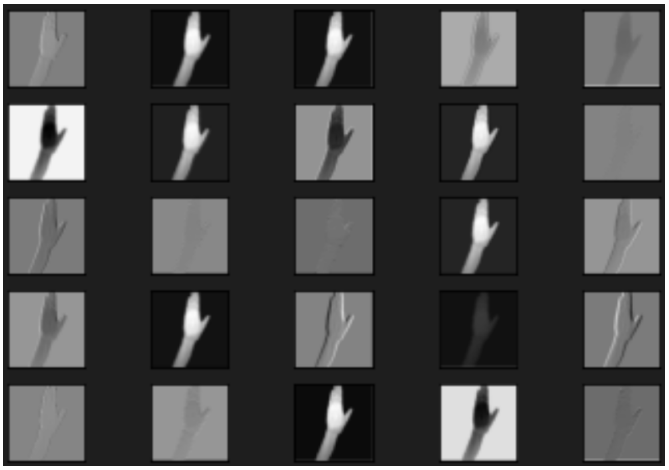

Figura 6.8: Mapas de características que aprende ResNet50 en la última capa.

Como se puede observar, por otra parte, parece ser que las características que extrae *ResNet50* son mucho más depuradas. Extrae los contornos, las formas, los bordes, sombras e incluso fondos.

Con todo ello, se podría deducir que el hecho de que una red con pocas capas tenga los mismos resultados que otra muchísimo más compleja, pudiera deberse a que las muestras y los requisitos del problema no necesiten una extracción tan compleja de patrones para diferenciar unas clases de otras. Finalmente se decide escoger como modelo para el reconocedor de gestos la red *GestNet8*. Las razones son: un tiempo computacional significativamente menor, resultados similares y el atractivo de poder utilizar una arquitectura propia.

Siguiendo la metodología explicada en el capítulo 3, con el modelo de *CNN* ya seleccionado *(GestNet8)* y con los distintos conjuntos de datos explicados en el capítulo 5, el siguiente paso es realizar una extensa experimentación para decidir la mejor configuración de la red y cuales son los mejores datos para construir un reconocedor de gestos. Esto es lo que se explica en el capítulo 7.

CAPÍTULO 7
Experimentos

En este capítulo se expondrán los experimentos restantes para todos los conjuntos de datos y se compararán tanto entre ellos, como con el proyecto de referencia [2]. Todos los experimentos se han hecho con el modelo seleccionado, *GestNet8*, del apartado anterior. El objetivo es decidir cuales son las mejores bases de datos de entre las disponibles, destacadas en el apartado 5.5.

Para ello, se ha realizado una experimentación de todas ellas y por simplicidad se ha limitado la presentación de resultados sólo a la tasa de aciertos. A fin de disponer de mayores criterios de comparación se hará uso de las matrices de confusión, solo de las bases de datos con mejores resultados para cada conjunto de datos. Independientemente de la base de datos de que se trate, se ha decidido conservar los nombres originales de las clases para facilitar la comparación de resultados.

7.1 Conjunto IR

Este conjunto de datos ya ha sido testeado para seleccionar una red. Sin embargo, para comparar con los otros conjuntos se vuelven a mostrar sus resultados en la tabla 7.1.

Base de datos	Entrenamiento	Validación	Test
BD-IR	97.36 %	97.54 %	94.86 %
BD-Otsu	96.93 %	97.12 %	95.55 %
BD-Crop	96.39 %	97.65 %	96.11 %
BD-OtsuCrop	98.38 %	98.49 %	96.89 %

Tabla 7.1: Resultados con conjuntos IR

Como se puede ver, la base de datos con mejores resultados ha sido *BD-OtsuCrop* en la que las imágenes de la mano han sido recortadas y escaladas y el ruido de fondo se ha eliminado, lo cual es consistente con la mejora de la tasa de aciertos respecto a las otras bases. Esto podría ser debido a que es la que menos ruido tiene, tanto del fondo como del brazo. Para ver mejor los resultados de las clases se va a estudiar la matriz de confusión.

Las matrices de confusión, se utilizan para comprobar la actuación o los resultados que ha tenido un modelo de clasificación. Es una matriz que tiene tantas filas y columnas como número de clases en el problema de clasificación. En base a un set de datos, el modelo hace predicciones que son apuntadas en la matriz. En la celda correspondiente a la fila i-ésima y la columna j-ésmia se muestra el porcentaje de muestras que perteneciendo a la clase i son clasificadas como pertenecientes a la clase j.

A continuación, se muestra la matriz de confusión para las predicciones de *GestNet8* con el conjunto de test proveniente de los datos de *BD-OtsuCrop* en porcentaje.

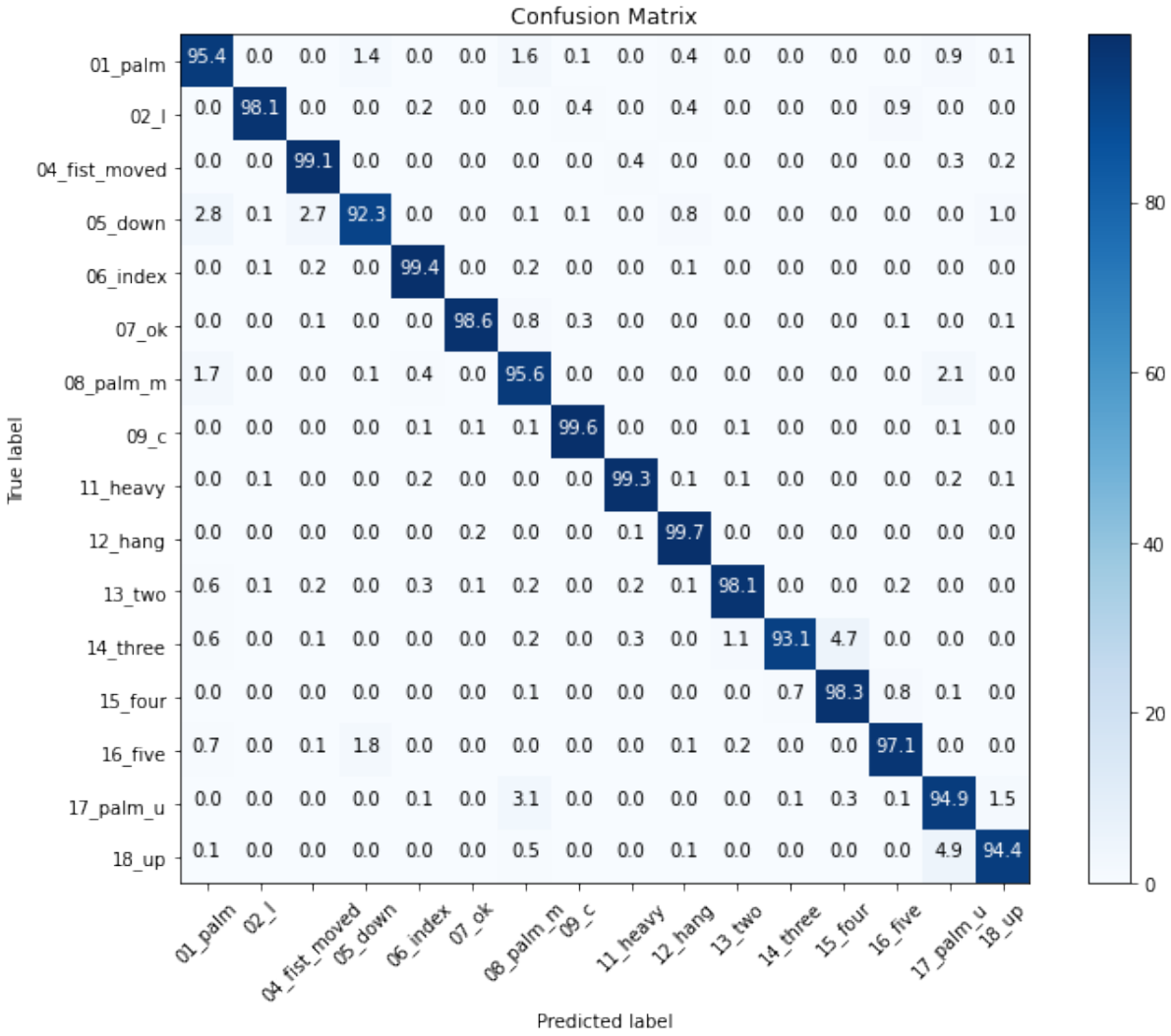

Figura 7.1: Matriz de confusion para BD-OtsuCrop

En la figura 7.1 se observan muy buenos porcentajes, con un mínimo en la clase *05_down* de 92,3 % y un máximo de 99.7 % en la clase *12_hang*. Los mayores cruces se producen entre las clases *14_three* y *15_Four* y entre las clases *18_Up* y *17_palm_u*. No es de extrañar debido a que en ambos se trata de clases que se parecen entre sí relativamente.

7.2 Conjunto LR

En la tabla 7.2 se pueden ver los resultados para los conjuntos que han sido creados a partir de las dos imágenes de una misma muestra.

Base de datos	Entrenamiento	Validación	Test
BD-Concat	96.06 %	92.05 %	90.06 %
BD-CH2	94.20 %	94.94 %	85.38 %
BD-RM	96.99 %	83.82 %	72.78 %

Tabla 7.2: Resultados con conjuntos LR

Tanto en la tabla 7.2, como en la matriz de confusión de la figura 7.2 para *BD-Concat*, los resultados no han sido muy buenos. Reducir el número de muestras a cambio de más información no parece funcionar. La clase con mejores porcentajes ha sido la clase *05_down* con un 95.5 % y la peor *07_Ok* con un 63 %. Resulta curioso como la mejor clase de este set es la peor del anterior. Con lo cual, la forma de funcionar de la red cambia según el tipo de datos que se le pasa. Las clases que más se cruzan son *06_índex* y *07_Ok* pero en general hay muchos cruces.

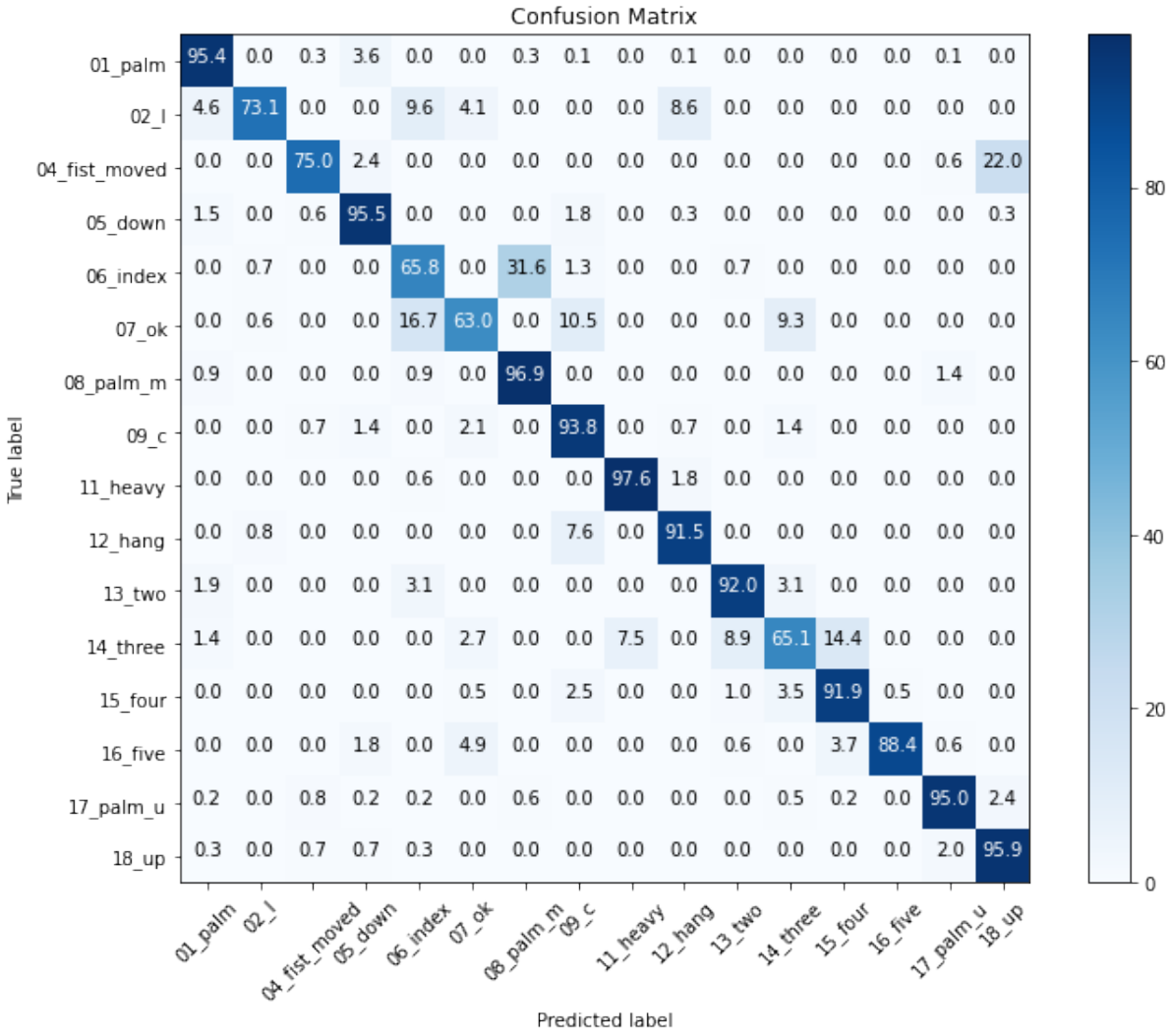

Figura 7.2: Matriz de confusión para BD-Concat

El empeoramiento de resultados respecto a las bases de datos *BD-Concat* y *BD-CH2* era previsible y se debe a que su tamaño es significativamente menor y que su creación no ha sido muy elaborada. Respecto a la base de datos *BD-RM*, en la que se tenía cierta expectativa, los malos resultados se explican además de por sus muestras, por los siguientes motivos: en primer lugar debido a la distorsión presente en las imágenes del *Leap* que no se ha podido eliminar completamente tal y como se ha comentado en apartados anteriores. En segundo lugar, es evidente que el reescalado al que está sujeta la base de datos original ha producido una pérdida de resolución en las imágenes que ha dificultado también la obtención del mapa de rango.

7.3 Conjunto SK

En la tabla 7.3 se muestran los resultados de los experimentos para cada una de las bases de datos que forman este conjunto.

Base de datos	Entrenamiento	Validación	Test
BD-SK	95.98 %	96.02 %	95.64 %
BD-SK3D	96.03 %	96.08 %	95.72 %
BD-SKC	96.20 %	96.67 %	96.05 %
BD-SKC3D	96.12 %	95.39 %	95.65 %

Tabla 7.3: Resultados con conjuntos SK

La base de datos con mejores resultados en este conjunto ha sido *BD-SKC*. Al diferenciar los dedos por colores de forma clara, no como las otras, la distinción entre clases es mejor. Discrepa en ciertas partes con la base de datos *BD-OtsuCrop* pero genéricamente tiene un comportamiento similar. De hecho las clases que más cruzan son las mismas, *14_three* y *15_Four* y *18_Up* y *17_palm_u*.

Respecto a la matriz de confusión de la figura 7.3 para *BD-SKC*, se vuelven a observar buenos resultados. La clase mejor clasificada es *02_l* con un porcentaje de acierto del 99.7 % y la peor *18_Up* con un 90 %. En cuanto a esta diferencia se puede concluir que el resultado es peor que para la BD-OtsuCrop.

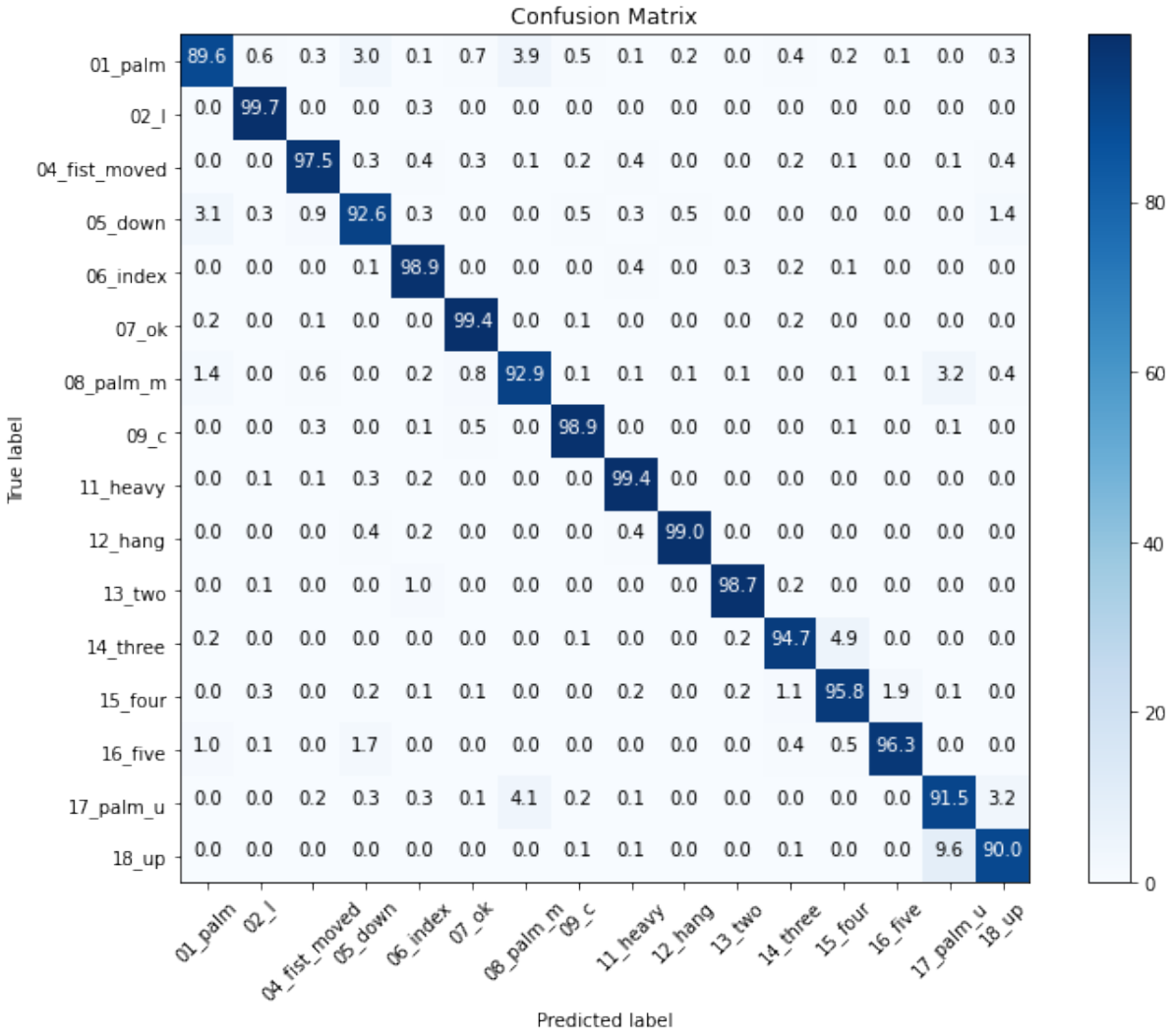

Figura 7.3: Matriz de confusión para BD-SKC.

7.4 Conjunto IR-desdistorsionado

El último conjunto para experimentar ha sido el de las imágenes infrarrojas sin distorsión. En la tabla 7.4 se pueden ver los resultados de la clasificación y entrenamiento en este conjunto.

Base de datos	Entrenamiento	Validación	Test
BD-IRUD	97.64 %	98.48 %	97.31 %
BD-CropUD	96.23 %	97.48 %	96.24 %

Tabla 7.4: Resultados con conjuntos IR-corregidos.

Como puede observarse, al contrario que con el conjunto infrarrojo, el mejor resultado ha sido para la base de datos *BD-IRUD*. Esto podría ser debido a que al corregir las imágenes, los ruidos como el brazo pasan a ser una característica más, en vez de algo en contra. Las otras, se han quedado igual que las originales y no cambia el resultado prácticamente, ya que no tienen esa información adicional que ha cambiado. A continuación, en la figura 7.4 se muestra la matriz de confusión para la *BD-IRUD*.

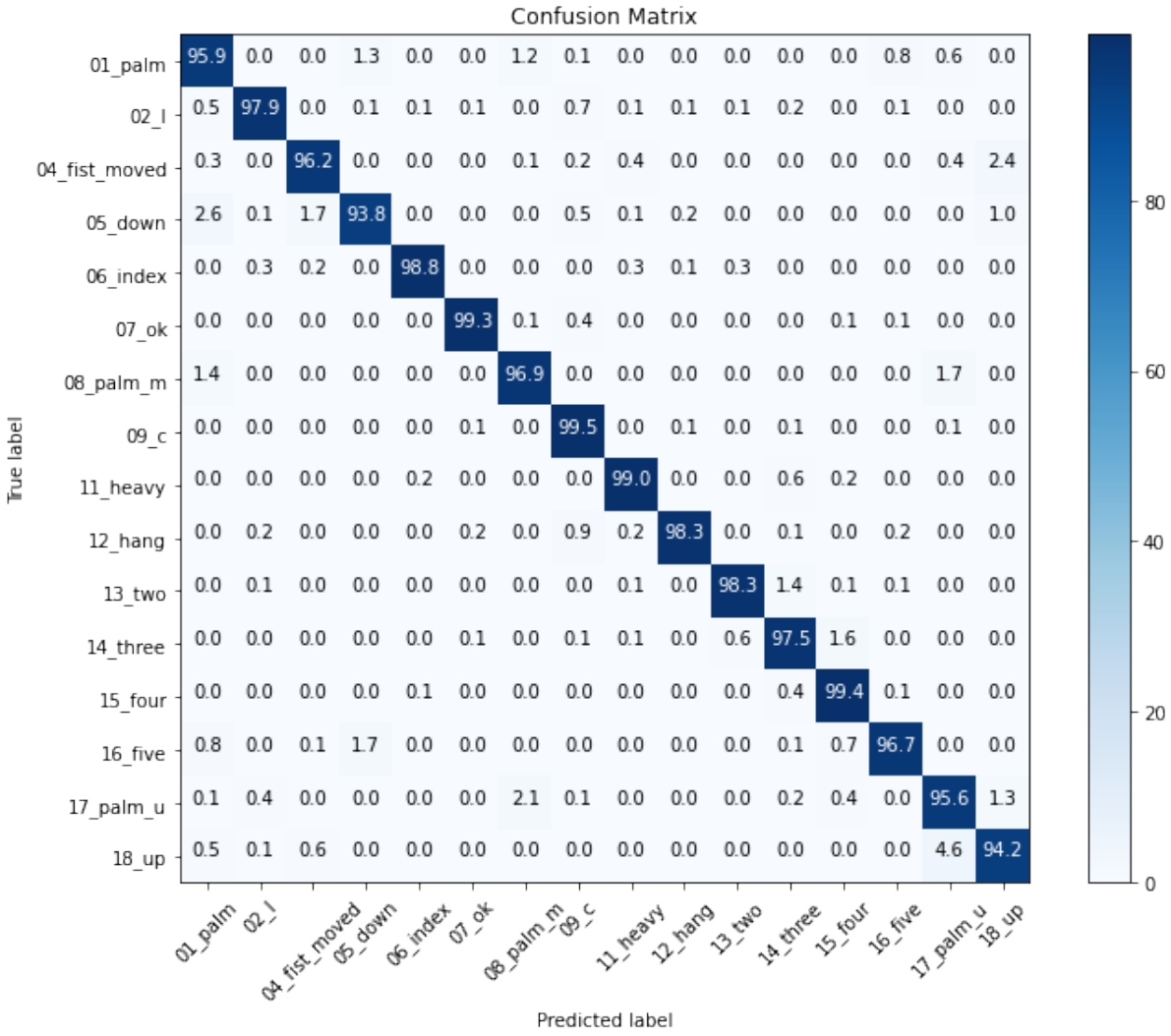

Figura 7.4: Matriz de confusion para BD-IRUD

Los datos de la matriz de confusión parecen mucho más equilibrados y mejores. La peor clasificación ha sido la de la clase *05_down* con un porcentaje de 93.8 % y la mejor *09_c* con un 99.5 %. Los cruces entre clases están mucho más repartidos y son de menor cuantía. En este caso los dos pares de clases más cruzadas son *05_down* y *01_palm* y *17_palm_u* y *18_Up*. Estas puntuaciones puede que se deban a las dimensiones y a la cantidad de información aprovechable que tiene la base de datos.

7.5 Discusión

Viendo las matrices de confusión y los resultados, se puede concluir que el conjunto que mejor clasifica para nuestra red es el de las muestras con corrección de la distorsión y que la mejor base de datos es *BD-IRUD*. Por otra parte, la peor clasificación ha sido para el conjunto LR. Puede que sea debido que estos conjuntos son los que menos muestras tienen, que las técnicas utilizadas no han sido suficientes para superar al gran número de muestras de los otros y que la distorsión de las imágenes no ha podido desaparecer del todo y ha introducido ruído en estos datos dificultando la combinación de las imágenes L y R.

Por otra parte, el conjunto de los esqueletos, que tiene un número de muestras y características razonables, ha clasificado correctamente pero no con los mejores resultados. Para realizar la comparación con otros trabajos de la literatura, no se dispone de resultados de clasificación de estas mismas clases utilizando técnicas con esqueleto. No obstante, sí que hay datos correspondientes al reconocimiento de gestos dinámicos que utilizan el esqueleto reportados en [46].

Estos datos son bastante inferiores en comparación a la clasificación de gestos dinámicos con modelos que utilizan imágenes. En concreto, un 18.6 % de diferencia respecto al mejor porcentaje alcanzado en la clase *12_hang*. Se podría inferir que la clasificación utilizando esqueletos da peores resultados y que por lo tanto, la adaptación de esqueletos a imágenes presentada en el punto 5 representa una muy buena alternativa a la mejor base de datos del conjunto IR.

También se puede concluir que las clases que más se cruzan en general son: *14_three* con *15_Four* y *18_Up* con *17_palm_u*. Para visualizar el cruce, se muestran ambos pares de clases en las figuras 7.5 y 7.6 respectivamente.

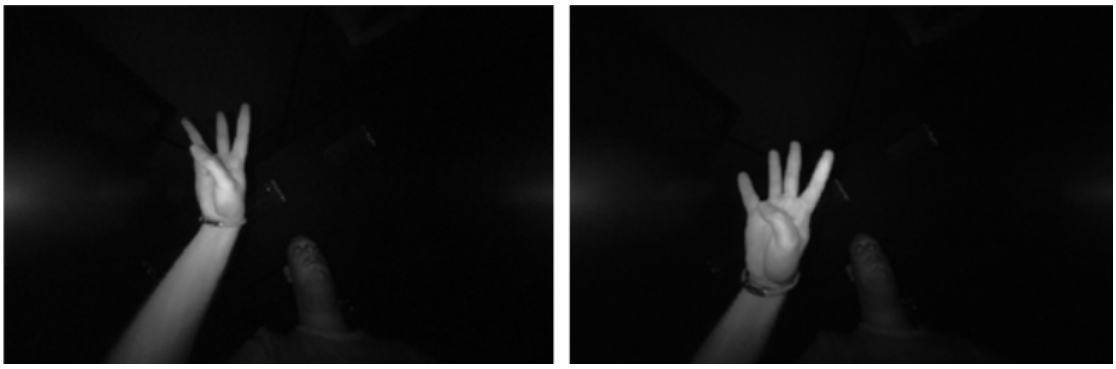

Figura 7.5: Cruce entre *14_three* y *15_Four*.

Figura 7.6: Cruce entre *18_Up* y *17_palm_u*.

Viendo la forma de ambos cruces, se puede apreciar que tienen una gran similitud entre ellos y que el error de clasificación no es debido a un incorrecto funcionamiento de la red muy severo.

Una vez seleccionada la arquitectura de red y la base de datos *BD-IRUD* como mejor solución solo resta comparar los resultados final con los del artículo de referencia [2], que se recogen en la figura 7.7.

Como se puede ver, la clasificación en su modelo también da porcentajes muy buenos. Con un porcentaje de 98.55 % para la clase más alta en *16_five* y un 92.95 % para la clase más baja en *17_palm_u*. El par de clases que más se cruzan son *17_palm_u* con *08_palm_m* y *13_two* con *06_index*. Parece que las clases que confunde son diferentes a las del mo-

	palm	L	fist m	down	index	ok	palm m	C	heavy	hang	two	three	four	five	palm u	up
palm	**95.13**	0.14	0.72	0.83	0.14	0.00	1.00	0.58	0.00	0.00	0.14	0.00	0.53	0.34	0.67	0.10
L	2.19	**95.55**	0.14	0.27	0.00	0.14	0.00	0.00	0.75	0.00	0.96	0.07	0.00	0.00	0.00	0.00
fist m	0.00	0.00	**97.29**	0.00	0.00	0.00	0.00	0.21	0.00	0.00	0.00	0.07	0.00	0.00	0.14	2.30
down	1.45	0.34	0.00	**95.09**	0.00	0.99	0.30	0.10	0.40	0.00	0.10	0.40	0.00	0.10	0.00	0.80
index	0.00	0.00	0.00	0.00	**96.22**	0.14	0.00	0.00	1.55	0.78	0.00	0.14	0.00	0.00	0.14	0.99
ok	0.26	0.56	0.26	0.06	0.45	**97.05**	0.00	0.00	0.00	0.00	0.32	0.00	0.00	0.58	0.32	0.06
palm m	1.73	0.00	0.77	0.05	0.34	0.34	**94.42**	0.24	0.43	0.05	0.00	0.00	0.05	0.00	1.59	0.00
C	0.00	0.00	0.51	0.22	0.07	0.58	0.00	**98.25**	0.00	0.22	0.00	0.00	0.00	0.00	0.00	0.15
heavy	0.15	0.37	0.00	0.00	1.19	0.00	0.07	0.07	**96.30**	0.00	0.00	0.96	0.30	0.00	0.59	0.00
hang	0.08	0.16	0.08	0.00	0.23	0.00	0.00	1.71	1.09	**95.47**	0.00	0.31	0.85	0.00	0.00	0.00
two	0.00	0.00	0.39	0.00	3.31	0.08	0.00	0.15	0.00	0.00	**95.90**	0.00	0.00	0.15	0.00	0.00
three	0.00	0.00	0.00	0.32	0.48	0.00	0.00	0.00	0.00	0.56	0.00	**98.34**	0.00	0.00	0.00	0.32
four	0.00	0.08	0.00	0.00	0.00	0.00	0.00	0.00	1.46	2.11	0.57	0.00	**95.75**	0.00	0.00	0.08
five	0.00	0.00	0.00	0.00	0.08	0.00	0.00	0.00	0.08	0.55	0.00	0.63	0.00	**98.55**	0.00	0.00
palm u	0.00	0.00	0.55	0.00	0.20	0.00	3.83	0.00	0.00	0.00	0.00	0.00	0.25	0.00	**92.95**	2.22
up	0.00	0.00	2.39	0.00	0.10	0.00	0.00	0.00	0.00	0.10	0.00	0.00	0.00	0.66	0.00	**96.75**

https://doi.org/10.1371/journal.pone.0223320.t001

Figura 7.7: Matriz de confusión del documento de referencia [2].

delo propio. Esto puede ser debido a que utiliza técnicas totalmente distintas basadas en vectores de características y *SVM*.

Los resultados son muy positivos, se puede concluir que conseguir valores igual de buenos que los de una arquitectura tan compleja como la suya a través de un sistema más sencillo y portable es un buen desenlace. También cabe destacar que los autores de esta base de datos también reconocen gestos dinámicos y no solo se centra en los estáticos, lo que se encuentra fuera del alcance del presente trabajo fin de grado.

Presumiblemente los buenos resultados de *GestNet8* sean debidos a su forma de interpretar las muestras y sin duda, al procesamiento de la imágen propuesta. Esta segunda afirmación puede ser apoyada por los resultados de la matriz de confusión con las bases de datos originales *BD-IR*. Los resultados para cada partición de datos en esta base de datos eran: 97.36 % para entrenamiento, %97.54 para validación y %94.86 en test. A continuación, en la figura 7.8 se pueden ver los resultados para su matriz de confusión.

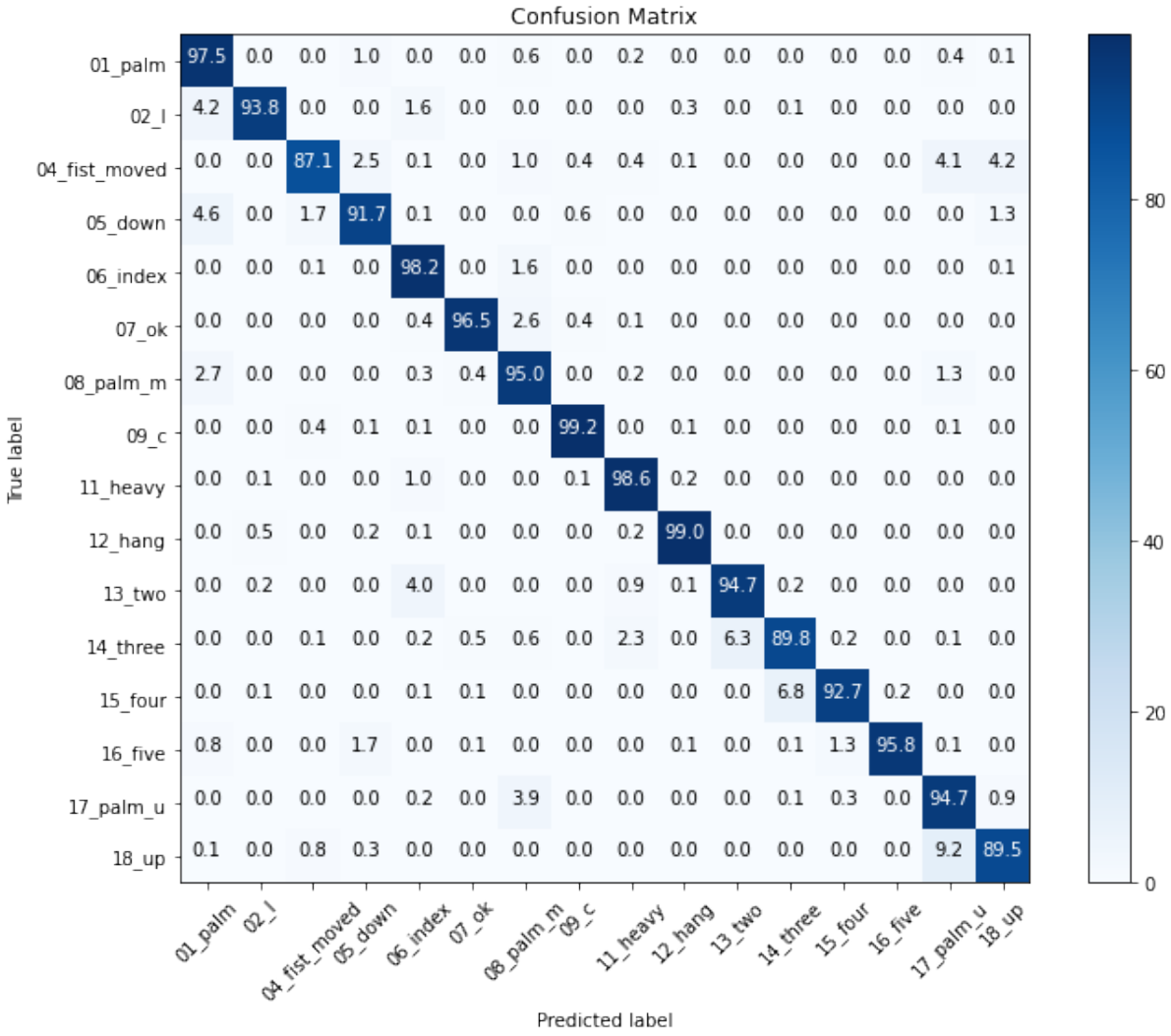

Figura 7.8: Matriz de confusión de la red con BD-IR original

Aunque los resultados son buenos, se encuentran por debajo de los mostrados anteriormente y se puede comprobar que los mejores son los obtenidos con BD-IRUD. Esto significa que el peso del algoritmo de corrección de la distorsión influye en el resultado final. Probablemente, su modelo con la base de datos BD-IRUD daría todavía mejores resultados.

7.6 Reconocedor con imágenes en tiempo real

Ya se tienen todos los elementos para crear el reconocedor de gestos en tiempo real. Se podría hacer un experimento para cada base de datos, transformando las imágenes y pasándolas a las redes correspondientes para cada una de ellas. Sin embargo, al tratarse de un demostrador de prueba para ver el funcionamiento y la capacidad de generalización del modelo en tiempo real, con imágenes tomadas de otro Leap, se va a utilizar la versión más sencilla y rápida con imágenes recogidas de *Leap Motion* directamente sin ningún tipo de procesamiento. Por lo tanto, para crearlo se va a utilizar el modelo propio de red entrenado para clasificar las muestras sin retocar (*BD-IR*).

Para desarrollar esta tarea, se ha utilizado *Python*, el controlador de *Leap* y *GestNet8*. Por ello, se ha puesto en marcha el controlador del dispositivo para tomar imágenes en

tiempo real, se ha cargado el modelo de red y la ventana en la que poner los resultados de las predicciones. En la figura 7.9 se pueden ver las líneas de código encargadas de esto.

```
def main():
    controller = Leap.Controller()
    controller.set_policy_flags(Leap.Controller.POLICY_IMAGES)
    model = tf.keras.models.load_model('IR_model.h5')
    display = np.zeros((275,416),np.uint8)
```

Figura 7.9: Puesta en marcha del modelo

A continuación, se extrae del controlador las imágenes *frame a frame*, se visualizan por pantalla y el modelo hace predicciones de cada una de ellas. Por último, las tres clases con mayor probabilidad de ser clasificadas junto a sus porcentajes en el frame actual, son mostradas en la ventana y ordenadas de mayor a menor probabilidad de reconocimiento.

La figura 7.10 es un ejemplo del funcionamiento e interfaz explicados. En la ventana de la derecha se puede ver la cámara del *Leap Motion* en tiempo real y en la de la izquierda, las clases que predice el modelo y sus porcentajes. En este caso se hace el gesto Dos y es acertado con un 95.33 %. Funciona bastante bien, aunque no se ha hecho una experimentación extensa en esas condiciones.

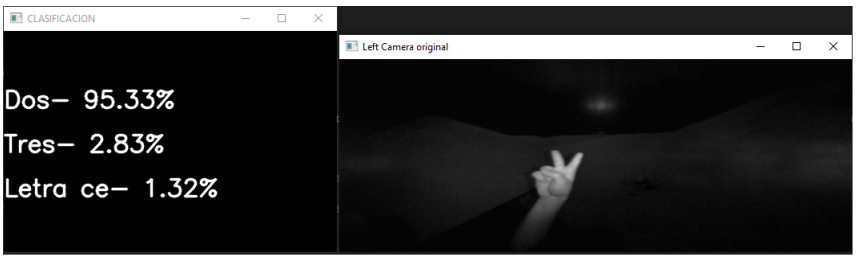

Figura 7.10: Imágen ejemplo del demostrador en tiempo real.

CAPÍTULO 8
Conclusiones y trabajos futuros

En este capítulo final se presentan las conclusiones inferidas de todos los experimentos, se analiza la relación entre el trabajo realizado con los estudios cursados y se enuncian los posibles trabajos futuros una vez terminado el actual.

8.1 Conclusiones

En el presente trabajo se ha desarrollado un reconocedor de gestos estático, formado por 16 clases diferentes y utilizando como dispositivo de adquisición de imágenes *Leap Motion*. Como técnica de clasificación se han empleado las redes neuronales convolucionales. Para ello se han diseñado distintas variantes de redes, tanto propias como pre-entrenadas, a fin de escoger entre ellas la que más se ajusta al problema.

El entrenamiento se ha llevado a cabo mediante una base de datos presente en la literatura que viene acompañada de un artículo con el que realizar comparaciones. Se han creado distintas versiones de esta base de datos a fin de explorar el conjunto de características que puede encontrar la red a partir de ellas. Por último, a fin de mostrar el comportamiento del reconocedor, se ha creado un demostrador en tiempo real del mismo. Como puede deducirse de lo anterior, todos y cada uno de los objetivos marcados en el punto 1.2 han sido alcanzados.

Los resultados obtenidos han sido: una red de profundidad 8 (*GestNet8*), sin utilizar redes pre-entrenadas, una serie de algoritmos para transformar información de Leap en diversos tipos de imágenes, especialmente la obtención de imágenes con distorsión corregida y un demostrador en tiempo real de las clases del problema.

La primera conclusión que cabe destacar es que la metodología basada en redes neuronales convolucionales se muestra muy eficaz para resolver el problema de reconocimiento de gestos, igualando en resultados como así atestiguan los experimentos a otras metodologías encontradas en la literatura. Lo que subyace es que estas redes son capaces de captar las características discriminantes de las imágenes y se adaptan satisfactoriamente ante problemas complejos.

Otra conclusión a la que se puede llegar es que en base a las características del problema, una arquitectura de red no muy extensa puede tener los mismos resultados que una más compleja. Este es el caso de la red seleccionada tras los experimentos y a partir de la cual se crea el demostrador.

Respecto al tipo de información que mejor funciona, se ha llegado a la conclusión de que las imágenes infrarrojas son más adecuadas que las basadas en información 3D,

especialmente si se consigue eliminar la distorsión presente en las mismas. Son varias las razones por las que se puede producir este comportamiento:

- Las imágenes IR tienen un mayor número de muestras. En cambio, cuando se intenta crear información 3D se pierde como mínimo la mitad de ellas.
- Para conseguir una información 3D adecuada es necesario que las imágenes L y R estén libres de distorsión. Esto no ha podido lograrse del todo ante la imposibilidad de disponer de los parámetros del dispositivo *Leap* con el que se adquirieron las imágenes utilizadas.
- Quizás la red puede aprovechar mejor la información de las imágenes *BD-IR* que la de las que tienen más características ya que introducen mayor ruido con el que lidiar en general.
- Otra posibilidad es que la potencia intrínseca de las redes neuronales convolucionales pudiera ser capaz de inferir la información resultante de modificar las bases de datos con los distintos algoritmos propuestos.

8.2 Relación del trabajo desarrollado con los estudios cursados

El presente TFG desarrollado, está bastante relacionado con los estudios cursados en la rama de computación. Al tratarse de un clasificador con redes neuronales, se han aplicado muchos de los conocimientos adquiridos en el grado, en especial, los provenientes de las asignaturas relacionadas con el aprendizaje profundo. Por lo tanto, cabe destacar las asignaturas: sistemas inteligentes, percepción y aprendizaje automático. A su vez, estas asignaturas también han necesitado de los conocimientos de otras, provenientes de cursos anteriores para poder ser cursadas en su momento.

No obstante, también ha sido necesario adquirir nuevos conocimientos. Algunos de ellos son: estudiar las CNNs, entender todas las librerías y algoritmos para transformar las imágenes y aprender a usar herramientas totalmente nuevas como TensorFlow, Keras y todas las mencionadas en el apartado 3. Por otra parte, también es cierto que sin las competencias transversales y a todo lo aprendido a lo largo del grado, todos estos nuevos conceptos no se podrían haber obtenido. Por ello, es necesario mencionar de entre las competencias transversales: comprensión e integración, análisis y resolución de problemas, creatividad, innovación y emprendimiento y sobre todo, aprendizaje permanente.

8.3 Trabajos futuros

Como se comentó en el capítulo 1, el presente proyecto final de grado es un primer paso para entender como se crean los clasificadores basados en imágenes, especialmente los gestuales. Es por eso, que hay muchas directrices que seguir a partir del trabajo desarrollado. A continuación se exponen alguna de ellas:

- **Crear un reconocedor de gestos para el lenguaje de signos:** con el dispositivo *Leap Motion* y los recursos adecuados se puede hacer cualquier base de datos lo suficientemente extensa como para crear cualquier reconocedor de gestos con buenos porcentajes. Un buen trabajo futuro podría ser crear una base de datos para el lenguaje de signos y utilizar la misma técnica para reconocer el lenguaje de signos en tiempo real.

- **Investigar los clasificadores de gestos dinámicos:** los gestos dinámicos no dejan de ser una sucesión de gestos estáticos. Adaptar todo lo aprendido a clasificar gestos dinámicos puede ser un buen objetivo.
- **Crear demostradores para el resto de bases de datos:** se ha creado un demostrador para reconocer en tiempo real la base de datos infrarroja. Se podría hacer lo mismo para el resto de bases de datos y transformar las imágenes tomadas por Leap en tiempo real a las correspondientes. No debería de ser muy complicado debido a que los algoritmos de transformación de imágenes ya están creados.
- **Utilizar otros dispositivos:** sería interesante probar otros dispositivos de captura de imágenes como una cámara infrarroja monocular, disponer de dispositivos con más profundidad o con un ángulo focal menor que no tuviera distorsión.
- **Crear portabilidad en el modelo:** las redes neuronales y sus modelos tienen una fácil portabilidad. Pasar el modelo creado a algún dispositivo portátil capaz de ejecutar el modelo como un celular o la raspberry es un escenario estudiable.

Bibliografía

[1] El silencio habla mucho. Tipos de gestos no verbales. https://elsilenciohablamucho.wordpress.com/2015/07/08/tipos-de-gestos-no-verbales/, 2015.

[2] Tomás Mantecón, Carlos R. del Blanco, Fernando Jaureguizar, and Narciso García. A real-time gesture recognition system using near-infrared imagery. *PLOS ONE*, 14(10):1–17, 10 2019.

[3] Munir Oudah, Ali Al-Naji, and Javaan Chahl. Hand gesture recognition based on computer vision: A review of techniques. *Journal of Imaging*, 6(8), 2020.

[4] Ahmed Kadem Hamed Al-Saedi and Abbas H Hassin Al-Asadi. Survey of hand gesture recognition systems. *Journal of Physics: Conference Series*, 1294:042003, sep 2019.

[5] Luigi Lamberti and Francesco Camastra. Real-time hand gesture recognition using a color glove. In Giuseppe Maino and Gian Luca Foresti, editors, *Image Analysis and Processing – ICIAP 2011*, pages 365–373, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[6] Ashish Sharma, Anmol Mittal, Savitoj Singh, and Vasudev Awatramani. Hand gesture recognition using image processing and feature extraction techniques. *Procedia Computer Science*, 173:181–190, 2020. International Conference on Smart Sustainable Intelligent Computing and Applications under ICITETM2020.

[7] Fatih Erden and A. Enis Çetin. Hand gesture based remote control system using infrared sensors and a camera. *IEEE Transactions on Consumer Electronics*, 60(4):675–680, 2014.

[8] Dimitrios Konstantinidis, Kosmas Dimitropoulos, and Petros Daras. Sign language recognition based on hand and body skeletal data. In *2018 - 3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON)*, pages 1–4, 2018.

[9] Lee Jaemin, Hironori Takimoto, Hitoshi Yamauchi, Akihiro Kanazawa, and Yasue Mitsukura. A robust gesture recognition based on depth data. In *The 19th Korea-Japan Joint Workshop on Frontiers of Computer Vision*, pages 127–132, 2013.

[10] Chengde Wan, Thomas Probst, Luc Van Gool, and Angela Yao. Self-supervised 3d hand pose estimation through training by fitting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[11] Kai Xing, Zhen Ding, Shuai Jiang, Xueyan Ma, Kai Yang, Chifu Yang, Xiang Li, and Feng Jiang. Hand gesture recognition based on deep learning method. In *2018*

IEEE Third International Conference on Data Science in Cyberspace (DSC), pages 542–546, 2018.

[12] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 53(8):5455–5516, Apr 2020.

[13] Weili Fang, Lieyun Ding, Hanbin Luo, and Peter E.D. Love. Falls from heights: A computer vision-based approach for safety harness detection. *Automation in Construction*, 91:53–61, 2018.

[14] Yujia Bao, Zhengyi Deng, Yan Wang, Heeyoon Kim, Victor Armengol, Francisco Acevedo, Nofal Ouardaoui, Cathy Wang, Giovanni Parmigiani, Regina Barzilay, Danielle Braun, and Kevin Hughes. Using machine learning and natural language processing to review and classify the medical literature on cancer susceptibility genes. *JCO Clinical Cancer Informatics*, 3:1–9, 09 2019.

[15] Chenyi Chen, Ming-Yu Liu, Oncel Tuzel, and Jianxiong Xiao. R-cnn for small object detection. In Shang-Hong Lai, Vincent Lepetit, Ko Nishino, and Yoichi Sato, editors, *Computer Vision – ACCV 2016*, pages 214–230, Cham, 2017. Springer International Publishing.

[16] Qing Li, Weidong Cai, Xiaogang Wang, Yun Zhou, David Dagan Feng, and Mei Chen. Medical image classification with convolutional neural network. In *2014 13th International Conference on Control Automation Robotics Vision (ICARCV)*, pages 844–848, 2014.

[17] Suchang Lim, Seunghyun Kim, Sungwook Park, and Doyeon Kim. Development of application for forest insect classification using cnn. In *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 1128–1131, 2018.

[18] Axel Davy, Thibaud Ehret, Jean-Michel Morel, Pablo Arias, and Gabriele Facciolo. A non-local cnn for video denoising. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 2409–2413, 2019.

[19] Takayuki Kawashima, Yasutomo Kawanishi, Ichiro Ide, Hiroshi Murase, Daisuke Deguchi, Tomoyoshi Aizawa, and Masato Kawade. Action recognition from extremely low-resolution thermal image sequence. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6, 2017.

[20] Muhammad Butt, Asad Khattak, Sarmad Shafique, Bashir Hayat, Samia Abid, Ki-Il Kim, Muhammad Ayub, Ahthasham Sajid, and Awais Adnan. Convolutional neural network based vehicle classification in adverse illuminous conditions for intelligent transportation systems. *Complexity*, 2021:1–11, 02 2021.

[21] Özal Yıldırım, PawełPławiak, Ru-San Tan, and U. Rajendra Acharya. Arrhythmia detection using deep convolutional neural network with long duration ecg signals. *Computers in Biology and Medicine*, 102:411–420, 2018.

[22] Peter Wozniak, Oliver Vauderwange, Avikarsha Mandal, Nicolas Javahiraly, and Dan Curticapean. Possible applications of the LEAP motion controller for more interactive simulated experiments in augmented or virtual reality. In G. Groot Gregory, editor, *Optics Education and Outreach IV*, volume 9946, pages 234 – 245. International Society for Optics and Photonics, SPIE, 2016.

[23] Ayanava Sarkar, Ketul Arvindbhai Patel, R.K. Ganesh Ram, and Geet Krishna Capoor. Gesture control of drone using a motion controller. In *2016 International Conference on Industrial Informatics and Computer Systems (CIICS)*, pages 1–5, 2016.

[24] Wei Lu, Zheng Tong, and Jinghui Chu. Dynamic hand gesture recognition with leap motion controller. *IEEE Signal Processing Letters*, 23(9):1188–1192, 2016.

[25] Katia Lupinetti, Andrea Ranieri, Franca Giannini, and Marina Monti. 3d dynamic hand gestures recognition using the leap motion sensor and convolutional neural networks. In Lucio Tommaso De Paolis and Patrick Bourdot, editors, *Augmented Reality, Virtual Reality, and Computer Graphics*, pages 420–439, Cham, 2020. Springer International Publishing.

[26] Pedro M. Ferreira, Jaime S. Cardoso, and Ana Rebelo. Multimodal learning for sign language recognition. In Luís A. Alexandre, José Salvador Sánchez, and João M. F. Rodrigues, editors, *Pattern Recognition and Image Analysis*, pages 313–321, Cham, 2017. Springer International Publishing.

[27] ultraleap. How hand tracking works. https://www.ultraleap.com/company/news/blog/how-hand-tracking-works/, 2020.

[28] Johnathon Selstad. Leapmotion, leapuvc. https://github.com/leapmotion/leapuvc/tree/master/Python, 2021.

[29] ano0002. Leap-motion-python-3.8. https://github.com/ano0002/Leap-Motion-Python-3.8, 2021.

[30] Tomás Mantecón, Carlos R. del Blanco, Fernando Jaureguizar, and Narciso García. Multi-modal hand gesture dataset for hand gesture recognition. https://www.gti.ssr.upm.es/data/MultiModalHandGesture_dataset, 2019.

[31] A. Rosebrock. *Deep Learning for Computer Vision with Python: Starter Bundle*. PyImageSearch, 2017.

[32] François Chollet. *Deep Learning with Python*. Manning, November 2017.

[33] DeepLizard. Keras-python deep learning neural network api. https://deeplizard.com/learn/playlist/PLZbbT5o_s2xrwRnXk_yCPtnqqo4_u2YGL, 2019.

[34] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[35] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[36] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning, 2016.

[37] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[38] Saining Xie, Ross Girshick, Piotr Dollar, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[39] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[40] Asifullah Khan, Anabia Sohail, and Amna Ali. A new channel boosted convolutional neural network using transfer learning, 2020.

[41] Abhijit Guha Roy, Nassir Navab, and Christian Wachinger. Concurrent spatial and channel 'squeeze & excitation' in fully convolutional networks. In Alejandro F. Frangi, Julia A. Schnabel, Christos Davatzikos, Carlos Alberola-López, and Gabor Fichtinger, editors, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*, pages 421–429, Cham, 2018. Springer International Publishing.

[42] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979.

[43] Ultraleap. Ultraleap camera image. https://developer-archive.leapmotion.com/documentation/python/devguide/Leap_Images.html?-proglang=python, 2014.

[44] Ping Zhang, Bei Li, Guanglong Du, and Xin Liu. A wearable-based and markerless human-manipulator interface with feedback mechanism and kalman filters. *International Journal of Advanced Robotic Systems*, 12(11):164, 2015.

[45] Godwin Ponraj and Hongliang Ren. Sensor fusion of leap motion controller and flex sensors using kalman filter for human finger tracking. *IEEE Sensors Journal*, 18(5):2042–2049, 2018.

[46] Runqing Zhang, Yue Ming, and Juanjuan Sun. Hand gesture recognition with surfbof based on gray threshold segmentation. In *2016 IEEE 13th International Conference on Signal Processing (ICSP)*, pages 118–122, 2016.

[47] Ultraleap. Ultraleap introducing the skeletal tracking model. https://developer-archive.leapmotion.com/documentation/objc/devguide/Intro_Skeleton_API.html, 2014.