



UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA DE INGENIERÍA INDUSTRIAL DE
TOLEDO

TRABAJO FIN DE GRADO N° 20-A/B-225441

CONTROL EN PLATAFORMA RASPBERRY MEDIANTE
RECONOCIMIENTO VISUAL



Autor:

SERGIO FUENTES CÁMARA

Director:

JUAN MORENO GARCÍA

ENERO DE 2020

Agradecimientos

Agradezco encarecidamente a todas aquellas personas que me han ayudado directa e indirectamente en la elaboración de mi proyecto de fin de grado, el apoyo ofrecido y la paciencia dada.

Especialmente a mi familia, sin ellos no podría haber llegado al momento en el que estoy hoy. A mi novia Laura, que ha estado junto a mi en toda esta etapa, tanto en los momentos donde los resultados eran buenos como en los que no, apoyándome y sin dudar en prestarme ayuda en cualquier momento. A mis amigos, por esos momentos de descanso para poder ver la solución al problema desde otros puntos de vista. A mis profesores por los conocimientos que he adquirido durante estos años y a mi tutor por la ayuda ofrecida en todo momento.

Gracias a todos ellos y gracias por tener fe en mí, sin vosotros no hubiese sido posible.

Resumen

En este TFG se ha realizado un diseño software para el control en plataforma Raspberry mediante técnicas de visión por computador. Para ello se ha utilizado la librería de OpenCV dedicada a la visión por ordenador.

El objetivo del diseño ha sido rastrear una mano con un guante azul para ubicar el puntero del sistema y los gestos de la mano para realizar diferentes acciones, como rotar imágenes o aplicarles zoom, o manejar pequeños juegos.

La clasificación de estos gestos se ha realizado mediante aprendizaje automático, con la utilización de redes neuronales convolucionales.

Abstract

In this final degree project, a desing software for Raspberry platform control has been made using computer vision techniques. For this, the OpenCV library, dedicated to computer vision, has been used.

The objective of the design has been to track a hand with a blue glove to locate the system pointer, and hand gestures to perform different actions, such as rotating images or zooming in on them, or handling small games.

The classification of these gestures has been done through machine learning, with the use of convolutional neural networks.

Índice general

1. Introducción	1
1.1. Presentación	2
1.2. Motivación	3
1.3. Objetivos del proyecto	4
1.4. Antecedentes	5
1.5. Organización de la memoria	7
1.5.1. Esquema	7
1.5.2. Temporización del TFG	7
2. Marco teórico	9
2.1. Conceptos	10
2.2. Procesamiento de la imagen	12
2.3. Lenguaje y Librerías	21
2.4. Redes neuronales	23
3. Desarrollo y Metodología	31
3.1. Hardware y software	32
3.1.1. Ordenador	32
3.1.2. Raspberry Pi 3 B+	33
3.2. Primeras aproximaciones	34
3.2.1. Template matching	35
3.2.2. Local binary patterns	36
3.2.3. Backprojection	41
3.3. Diseño del sistema de clasificación	44
3.3.1. Rango de valores y técnica CAMshift	44
3.3.2. Búsqueda de contornos y comparación de formas	46
3.3.2.1. Búsqueda de contornos	47
3.3.2.2. Comparación de formas	49
3.3.3. Basado en redes neuronales convolucionales	55
3.3.3.1. Generación de la base de datos	55
3.3.3.2. Estructura de una red neuronal convolucional	57
3.3.3.3. Entrenamiento de la red neuronal	60

3.3.3.4.	Resultados de validación del modelo	63
3.3.3.5.	Predicción del modelo	66
3.4.	Software y ajustes del sistema	67
3.4.1.	Diseño del software	67
3.4.2.	Ajustes del software	70
3.4.3.	Depuración de los resultados de la clasificación	71
4.	Resultados y aplicaciones	73
4.1.	Evaluación	74
4.2.	Aplicaciones	77
4.2.1.	Imágenes	78
4.2.1.1.	Rotación de imágenes	79
4.2.1.2.	Zoom en las imágenes	81
4.2.2.	Juego	82
5.	Conclusiones y trabajos futuros	83
	Referencias	85
A.	Anexo A	89

Índice de tablas

3.1. Detalles técnicos Raspberry Pi 3 Model B+	33
3.2. Métodos utilizados en las primeras aproximaciones	34
3.3. Métodos utilizados en el sistema de clasificación	44
3.4. Imágenes de la base de datos	56
3.5. Biblioteca de Símbolos del diagrama de flujo	68
4.1. Resultados utilizando comparación de formas	75
4.2. Valores de rendimiento del clasificador basado en comparaciones de las formas	75
4.3. Resultados de las predicciones del modelo neuronal	76
4.4. Valores de rendimiento del clasificador basado en las predicciones del modelo neuronal	76
A.1. Estructura de la red neuronal de Mobilenet v2	90

Índice de figuras

1.1. Técnicas de reconocimiento visual	2
1.2. Kinect	5
1.3. Leap Motion	6
1.4. Diagrama Gantt	7
2.1. Imagen con 750 píxeles de ancho x 562 píxeles de alto	10
2.2. Matriz de píxeles con diferentes valores	10
2.3. Imágenes con canales R, G y B	11
2.4. Imagen con canales RGB	11
2.5. Píxeles de una imagen con canales RGB	12
2.6. Cilindro HSV	13
2.7. Barras del modelo RGB	13
2.8. Barras del modelo HSV	14
2.9. Sustracción de color	14
2.10. Sustracción de colores	15
2.11. Núcleo del filtro	16
2.12. Matriz con intensidades	16
2.13. Filtro mediana	17
2.14. Imagen en escala de grises con su histograma de intensidades	17
2.15. Histogramas de la imagen RGB	18
2.16. Método de Backprojection	18
2.17. Conversión a LBP. Fuente Coursera (1)	19
2.18. Diferentes rangos del LBP	19
2.19. Gradientes de píxeles	20
2.20. Obtención de los bordes con el algoritmo Canny de la Figura 2.4	21
2.21. Técnica de meanshift	22
2.22. Representación gráfica de una neurona artificial simple	23
2.23. Capas de una red neuronal	24
2.24. Funciones de activación	25
2.25. Backpropagation	26
2.26. Red neuronal profunda	27
2.27. Red neuronal convolucional(2)	27

2.28. Red generativa adversaria	28
2.29. Modelos pre-entrenados	29
3.1. Técnica de Template matching en condiciones “buenas”	35
3.2. Técnica de Template matching en condiciones “malas”	36
3.3. Modelo LBP	37
3.4. Comparación de diagramas LBP	38
3.5. Diagrama non maximum suppression	39
3.6. Non maximum suppression	40
3.7. Imágenes LBP en pirámide	41
3.8. Proceso para la técnica de Backprojection	42
3.9. Modelo con defecto de características	43
3.10. Modelo con exceso de características	43
3.11. Pasos para generar candidatos por Rango de valores	45
3.12. Uso de la técnica CAMshift	46
3.13. Contornos Canny	47
3.14. Contorno de una máscara	48
3.15. Contornos con casco convexo y defectos de convexidad	49
3.16. Uso de la CAMshift en el software	49
3.17. Tabla de momentos (3)	50
3.18. Comparaciones de formas	51
3.19. Comparaciones con cv2.matchShape	52
3.20. Comparación de patrones en el vídeo	53
3.21. Diagrama de la función crear patrones	53
3.22. Directorio de los patrones y su creación desde la función crear patrones	54
3.23. Comparación del candidato con múltiples patrones de cada gesto	54
3.24. Ejemplo de estructura y composición de la base de datos	56
3.25. Proceso de la convolución en una imagen a color	57
3.26. Red neuronal convolucional básica	58
3.27. Imágenes generadas para el entrenamiento	61
3.28. Descenso del gradiente con un solo mínimo	61
3.29. Vector de salida en la red neuronal	62
3.30. Épocas de entrenamiento del modelo Mobilenet modificado	62
3.31. Gráficas de evaluación	63
3.32. Errores de entrenamiento	64
3.33. Diferentes evaluaciones	65
3.34. Escalado manteniendo las proporciones	66
3.35. Diagrama simplificado	67
3.36. Diagrama general	68
3.37. Menús	69
3.38. Diagrama submenú Ajustes	70
3.39. Diagrama del inicio del rastreo	71

4.1. Identificadores del objeto	74
4.2. Indicadores en la imagen de captura	77
4.3. Vista de la selección de aplicaciones	78
4.4. Aplicación imágenes	79
4.5. Rotando imágenes	80
4.6. Zoom en las imágenes	81
4.7. Aplicación juego	82

Capítulo 1

Introducción

El ser humano ha guiado sus actos por la información que sus ojos le han proporcionado, el sentido de la vista es un recurso muy valioso. Las máquinas y ordenadores creados por el ser humano son unos logros realmente increíbles y tras el paso del tiempo se han actualizado, modernizado y buscado nuevos métodos para obtener la mayor información posible. La información es valiosa, recabarla y saber gestionarla es uno de los hitos que se ha de alcanzar.

Dotar a las máquinas y ordenadores de la capacidad de concebir el entorno que les rodea es una gran ventaja a la hora de recoger información, ya sea para una toma de decisión inmediata o simplemente para ver su evolución. Esa información es muy valiosa ya que permite prever como puede ir evolucionando, aunque a veces su predicción sea muy compleja.

Las aplicaciones de la visión por computador son muy numerosas y comprenden ámbitos tan diversos como puede ser la agricultura, controles de calidad en industria, realidad aumentada, análisis de imágenes médicas, seguridad y vigilancia, transporte, robótica y numerosas disciplinas más (4).

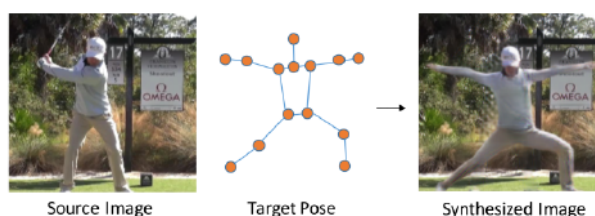
La visión artificial es la tecnología que permite detectar la luz y saber interpretarla, convierte cámaras pasivas en dispositivos inteligentes capaces de detectar eventos específicos y tomar decisiones. La visión artificial puede ser capaz de clasificar y diferenciar formas, detectar y seguir objetos o realizar un conteo de estos, al igual que otras muchas cosas más según la finalidad que se dé al sistema. El desarrollo de esta tecnología sirve para encontrar soluciones a los problemas que importan (5).

1.1. Presentación

El propósito de este estudio ha sido diseñar un sistema de reconocimiento visual, que permita recopilar información de las imágenes capturadas por una cámara, y con esta controlar un dispositivo. El uso del machine learning ha sido el que ha conseguido impulsar técnicas de reconocimiento en la visión artificial a tiempo real. Este tipo de técnicas que en 2017 fueron capaces de leer los labios, ver la posición de las personas o modificar esta posición en una imagen o vídeo.

- LipNet (6) es un software que ha alcanzado una exactitud mayor de predicción a la hora de leer los labios que los humanos más expertos. Su finalidad es ayudar a personas con discapacidad auditiva.
- OpenPose (7) es un software que estima la posición para múltiples personas a tiempo real, simplemente desde un vídeo tomado con una cámara convencional (Figura 1.1 (a)).
- Y un sintetizador de poses, que dada una imagen de una persona y un propósito deseado, produce una representación de esa persona en esa pose, conservando la apariencia de la persona y el fondo (8).
- Una mezcla de estos dos últimos es, Everybody Dance Now, que utiliza la técnica de OpenPose para saber la posición en el vídeo fuente y sintetiza la posición en el vídeo objetivo, dando un resultado bastante convincente a la hora de transferir el movimiento (Figura 1.1 (b)) (9).

(Vídeo: <https://www.youtube.com/watch?v=PCBTZh41Ris>)



(a) Sintetizador de poses



(b) Everybody Dance Now

Figura 1.1: Técnicas de reconocimiento visual

Existen muchas más aplicaciones de machine learning que fomentaron mi interés por la visión artificial, pero sobretodo el pensamiento de que este tipo de aplicaciones cada vez serán más, tendrán mayor utilidad y serán más robustas.

1.2. Motivación

Antes del boom del machine learning se utilizaban otro tipo de técnicas basadas en algoritmos lentos y propensos a errores, donde se buscaban regiones probables donde se pudiese encontrar el objeto, como bien cita *Richard Szeliski* en su libro *Computer Vision: Algorithms and Applications* del 2010. También reconoce que el uso del machine learning será un papel fundamental dentro de la visión artificial (10).

“Learning approaches play an increasingly important role in face recognition, e.g., in the work of Sivic, Everingham, and Zisserman (2009) and Guillaumin, Verbeek, and Schmid (2009).”

Asumiendo que los enfoques con aprendizaje desempeñan un papel importante en el reconocimiento facial y basándose en ejemplos de libros que hablan de ello. También Marc Leroy un profesor de Ciencias de la computación e ingeniero eléctrico de la Universidad de Stanford escribió en 2008 en uno de sus artículos lo siguiente:

“... that simple machine learning algorithms often outperform more sophisticated ones if trained on large enough databases.”

Refiriéndose a que los algoritmos simples de aprendizaje automático a menudo superan a las técnicas y algoritmos más sofisticados de reconocimiento si se entrenan con bases de datos lo suficientemente grandes (11).

Pero, ¿qué es el machine learning? Bueno, el machine learning o aprendizaje automático nace en 1950 a manos de Alan Turing con el test de Turing. Este test determina si una máquina es realmente inteligente, intentando engañar al humano a la hora de hacerle creer si es un computador o no. También en 1958 Frank Rosenblatt diseña el Perceptrón, la primera red neuronal artificial. Durante los años se fueron perfeccionando las técnicas, pero en la segunda mitad de la década de los 70 sufre el primer “AI-Winter”, primer invierno de la Inteligencia Artificial (AI). Tras el nacimiento de los sistemas expertos en 1980, sistemas informáticos que emulan el razonamiento humano basado por reglas, se vuelve a generar interés por el machine learning. A finales de los 80 llega el segundo invierno para la IA, hasta 1997, una fecha en la que la inteligencia artificial consigue superar al ser humano. Se trata de la partida de ajedrez entre Gary Kasparov y el ordenador de IBM, Deep Blue. El campeón mundial de ajedrez derrotado por la inteligencia artificial. Esto demostró que un ordenador podía dominar en un campo que hasta ese momento se creía muy ligado con la habilidad cognitiva humana.

Tras lograr este gran reto y acuñar el término “Deep Learning” (Aprendizaje Profundo) en 2006, para explicar arquitecturas de Redes Neuronales Profundas que son capaces de aprender mucho mejor modelos más planos, importantes empresas como Google, IBM, Microsoft, Appel

o Tesla comienzan a invertir grandes cantidades de dinero en proyectos que están relacionados con la inteligencia artificial y machine learning.

En 2012 Google desarrolla un proyecto llamado GoogleBrain, que desarrolla una Red Neuronal Profunda (Deep Neural Network o DNN) que utiliza toda la capacidad de la infraestructura de Google para detectar patrones en vídeos e imágenes. Más tarde se verán redes neuronales recurrentes que son mucho más potentes, estas redes serían capaces de reconocer personas con una precisión similar a la de un ser humano (2014).

Y eso no es todo, solo ha sido el principio de una serie de eventos que la visión artificial está consiguiendo gracias a la aportación de la inteligencia artificial (12).

El 4 de Octubre de 2016 en una conferencia de Google donde presentaban algunos de sus productos hardware, declararon una intención de desarrollar todos su productos ligados con la inteligencia artificial. Justo un año después en otra conferencia de Google se habló de AutoML, básicamente una inteligencia artificial que sirve para crear inteligencia artificial. Los modelos anteriormente diseñados eran muy complejos y están muy ligados a cómo se hayan configurado previamente, esta inteligencia aprenderá a modificar estas configuraciones de la forma que se consideren más efectivas, siendo capaces de resolver tareas de clasificación de imágenes y reconocimiento de objetos con una mayor precisión y un menor gasto de procesamiento que los modelos más avanzados actualmente (13).

Todos estos avances han dejado un abanico de posibilidades en la hora del diseño y distribución de aplicaciones. Desde una conducción autónoma hasta la elaboración de programas que permiten mediante gestos controlar periféricos de tu casa o aplicaciones de tu televisor y ordenador.

1.3. Objetivos del proyecto

El principal objetivo es el diseño e implementación de un sistema de visión por computador capaz de reconocer movimientos captados desde una cámara y obtener control del sistema. Este sistema funciona tanto en un ordenador con sistema operativo Windows, como en una Raspberry Pi con Raspbian.

Para abordar estos objetivos se plantean otros objetivos secundarios:

- Uso de algoritmos para la segmentación de la imagen, en este caso, la segmentación será por iluminación, saturación y color.
- Capacidad de modificar los parámetros para esa segmentación.
- Uso de algoritmos para el seguimiento del objeto, dando las coordenadas donde se encuentra el objeto.
- Uso de algoritmos para la clasificación de gestos, basado en comparación de patrones.
- Uso de redes neuronales para la clasificación de gestos, aplicando el aprendizaje automático o machine learning.

- Uso de entornos virtuales para la elaboración del proyecto con sus librerías adecuadas.

1.4. Antecedentes

En la actualidad existen varios dispositivos capaces de captar o reconocer movimientos de las manos. Estos sirven para realizar un control en un software que previamente se ha diseñado. Entre estos dispositivos están Kinect, Leap Motion y WiSee.

Kinect es un controlador de juego libre y entretenimiento. Permite a los usuarios controlar e interactuar con la consola sin necesidad de tener contacto físico con un controlador tradicional. Mediante una interfaz natural de usuario se reconocen gestos, comandos de voz y objetos e imágenes (Figura 1.2) (14).

El nuevo dispositivo de Kinect cuenta con una cámara de profundidad de 1 megapíxel, 7 micrófonos de audio en 360 grados, cámara RGB de 12 megapíxel y un sensor de orientación (15).

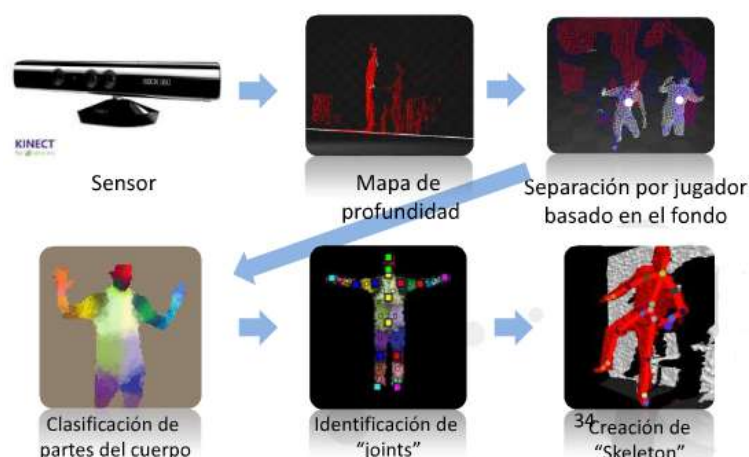


Figura 1.2: Kinect

Leap Motion es una compañía que comercializa un sensor que permite captar los movimientos de la mano y dedos como entrada, no requiere contacto y ha lanzado un software diseñado para el seguimiento manual en realidad virtual (16).

El controlador Leap Motion es un pequeño dispositivo periférico USB, usa dos cámaras IR monocromáticas y tres LEDs infrarrojos. Este dispositivo recoge la información hemisférica que está sobre él hasta una distancia de un metro (Figura 1.3) (17). En una demostración, el controlador realizaba tareas como navegar en un sitio web, usar gestos de pellizcar para hacer zoom en mapas, dibujar con alta precisión y manipular complejas visualizaciones de datos en 3D (18).

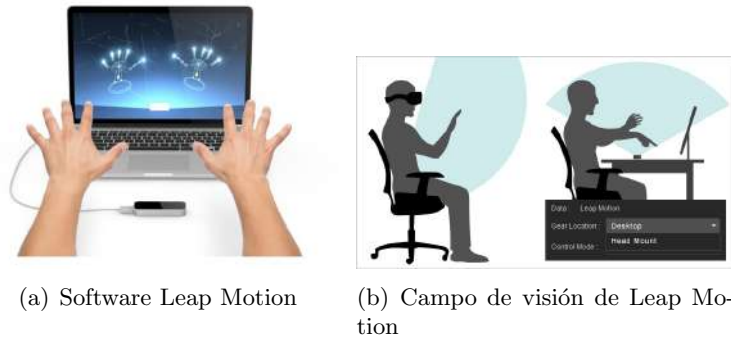


Figura 1.3: Leap Motion

WiSee es una tecnología que permite detectar gestos y movimientos a través del wifi. Lo novedoso de este sistema es que no necesita realizar los gestos hacia ningún dispositivo como en los anteriores, sino que aprovechándose del efecto Doppler, se puede realizar el movimiento hacia cualquier dirección y el dispositivo lo captura.

WiSee no necesita de ningún elemento especial, sino que simplemente con un router wifi y varios dispositivos conectados a la red inalámbrica se puede capturar los movimientos.

El uso de la inteligencia artificial en el diseño del software está dando resultados más buenos que los obtenidos por técnicas complejas sin su uso.

En estos nuevos diseños se les aplican las conocidas como redes neuronales. Actualmente existen numerosas herramientas con las que se pueden crear redes neuronales implementando funciones de librerías como Tensorflow, PyTorch, Keras, etc. Las diferencias de unas u otras herramientas es el nivel de profundidad que dan. Las herramientas más sencillas permiten menos flexibilidad en el diseño de arquitecturas, mientras que las herramientas que dan más flexibilidad sus funciones son más complejas.

Existen diferentes niveles de trabajo para la creación de redes neuronales:

- Al nivel más bajo se obtiene la programación de una red neuronal a base de funciones matemáticas con un lenguaje de programación como es Python.
- Un nivel más alto o librerías de diferenciación automática, sería con la utilización de Tensorflow (o Pytorch). Estas librerías recalculan automáticamente si existen cambios en su estructura, la creación de una capa puede darse en unas pocas líneas de código.
- Simplificando aún más la forma de crear las redes llegamos al nivel de composición de capas, donde se encuentra Keras, un módulo de Tensorflow, que en una simple línea de código es capaz de añadir una capa a la estructura.
- Aún más simple sería trabajar con los modelos, Sklearn permite en tan solo pocas líneas de código implementar una red neuronal (13).

1.5. Organización de la memoria

1.5.1. Esquema

Resume el contenido de los diferentes capítulos del documento.

1. Introducción: presentación del tema, antecedentes y motivación a realizar el TFG. Objetivos que se desean alcanzar y como se han conseguido. Temporización del proyecto.
2. Marco teórico y estado del arte: definición de los conceptos, técnicas y modelos de visión artificial que han sido utilizados en el TFG y ayudan a comprender el proyecto.
3. Desarrollo y metodología: plantea cómo se han ido alcanzando los objetivos principales y los secundarios y el porqué de las técnicas utilizadas.
4. Resultados y aplicaciones: muestra los resultados del proyecto, además de aplicaciones dentro del programa.
5. Conclusiones y trabajos futuros: muestra las conclusiones alcanzadas con el desarrollo del TFG y trabajos futuros que resultan de interés.

1.5.2. Temporización del TFG

Para la elaboración de este proyecto he seguido unas pautas conforme se indica en el diagrama de Gantt (Figura 1.4) .

- Información previa:
 - Búsqueda de un tema interesante y novedoso.
 - Búsqueda de fuentes de información relacionadas con el tema.
- Adquisición de Hardware:
 - Adquisición Raspberry Pi 3 b+.

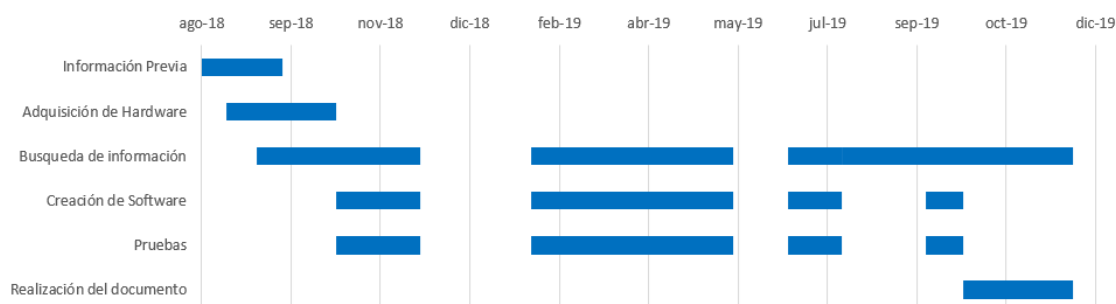


Figura 1.4: Diagrama Gantt

- Adquisición cámara Raspberry.
- Adquisición WebCam logitech.
- Pruebas:
 - Captura de vídeo con cámara en la Raspberry.
 - Grabación de vídeos con la cámara del móvil.
 - Captura de vídeo con WebCam en el ordenador.
 - Pruebas del software.
- Software:
 - Elección del sistema operativo para instalar en Raspberry.
 - Adquirir conocimientos sobre el SO elegido, su instalación y sus actualizaciones.
 - Uso y creación de entornos virtuales.
 - Instalaciones de los programas y librerías utilizadas en el proyecto, como: Python, OpenCV, PiCamera, Pyautogui, PyGame, Tensorflow, Keras...
 - Creación de redes neuronales.
 - Elaboración de una base de datos eficiente para la red neuronal.
 - Creación del software en código Python
- Realización de la documentación pertinente:
 - Redacción de la memoria.
 - Redacción de la presentación.
 - Preparación de la presentación.

Capítulo 2

Marco teórico

En este capítulo se tratarán conceptos, términos y técnicas utilizados en este TFG y que se utilizan en el campo de la visión artificial. Es una ayuda para comprender el tema de estudio y para poner en situación de algunos de los términos y herramientas que se han utilizado en la elaboración de este.

Primero se comenzará definiendo una serie de conceptos que son fundamentales en la visión por computador, como que es una imagen digital, que es el píxel y el ruido en la imagen.

También se explicarán técnicas para el procesamiento de la imagen que pueden ser por espectros de colores, por la intensidad de sus píxeles vecinos o por sus gradientes.

Por último, se mencionan las librerías utilizadas en el programa y conceptos relacionados con redes neuronales.

2.1. Conceptos

Para entender el campo de la visión artificial, primero se debe tener claro como los ordenadores trabajan con las imágenes que las cámaras u otros instrumentos les proporcionan, para ello se comienza definiendo unos términos.

La imagen y el píxel

Las imágenes dentro de la visión artificial son una representación bidimensional que se recoge en una matriz de puntos a los que se les llama píxeles. El píxel es la menor unidad homogénea dentro de la imagen. El número de filas y columnas de la matriz corresponde con el número de píxeles en el alto y ancho de la imagen (Figura 2.1) (19).



Figura 2.1: Imagen con 750 píxeles de ancho x 562 píxeles de alto

Cada píxel recoge un valor de intensidad que está entre 0 y 255, siendo 0 un píxel negro y 255 uno blanco, como muestra la Figura 2.2.

0	30	90	105	150	195	215	255
75	100	135	145	156	200	205	236
45	80	135	200	125	90	25	0
5	85	90	150	255	0	100	5

Figura 2.2: Matriz de píxeles con diferentes valores

Cuando se habla de una imagen a color, generalmente se refiere a imágenes RGB (Red, Green y Blue). Se dice que dicha imagen tiene 3 canales, cada canal corresponde a los colores rojo, verde y azul, y existen tres matrices correspondientes a la intensidad de cada color. Por ejemplo, una intensidad de 255 en el canal R de una imagen, significa que ese píxel contiene rojo y en el canal R se muestra con una intensidad alta, es decir blanco.

La combinación de estos tres canales (Figura 2.3) da el resultado de una imagen a color como se ve en la Figura 2.4.

Existen imágenes con otros tipos de canales, como el HSV (Hue, Saturation y Value) que corresponde al matiz, saturación y brillo del que se habla más adelante.



(a) Imagen canal R (b) Imagen canal G (c) Imagen canal B



(d) Matrices de píxeles R, G y B, con diferentes valores

Figura 2.3: Imágenes con canales R, G y B



Figura 2.4: Imagen con canales RGB

Cuando se tratan imágenes con tres canales, como el RGB, es incómodo tener que trabajar con las tres imágenes de cada uno de los canales, para evitar eso el valor de cada píxel es un vector con los tres valores que indican la intensidad de cada canal (Figura 2.5).

A veces aparecen píxeles que toman valores muy dispares a sus próximos, haciéndose destacar demasiado y que no corresponden con la imagen real, a esos píxeles se les llama ruido.

Ruido en la imagen

El ruido en una imagen digital es la variación aleatoria (que no se corresponde con la realidad) del brillo o el color en las imágenes digitales producido por el dispositivo de entrada (la cámara digital en este caso) (20).

Existen diferentes tipos de ruidos:



Figura 2.5: Píxeles de una imagen con canales RGB

- Ruido Gaussiano: es el que tiene una distribución normal por toda la imagen y se trata con filtros gaussianos. Estos filtros utilizan $n \times m$ píxeles como kernel (núcleo) que se encarga de equilibrar los valores de los píxeles que abarca el kernel. Siendo n y m números impares y positivos. Para un mayor kernel aumenta la sensación de desenfocado y se aprecia una imagen borrosa.
- El ruido impulsivo: hace tender a los píxeles a mínimos o máximos, es decir, a negros y blancos. Para su corrección se suelen utilizar filtros de mediana. Estos filtros utilizan un kernel de $n \times n$ píxeles y equilibra sus valores, siendo $n = 3, 5, 9$, etc. Este filtro es más rápido que el gaussiano, pero es menos efectivo en la separación de frecuencias.
- Ruidos por color: se deben a la luz incidente, la superficie del material donde se refleja y la sensibilidad de la lente de la cámara. Estos tres factores dependen en su totalidad del color que se ve reflejado en la imagen. La existencia de una luz incidente cálida, o amarilla, induce a los blancos a ser amarillentos y una luz incidente fría, o azul, induce a los blancos a ser azulados. Para su corrección se aplica el algoritmo de white-patch (1) que busca el píxel más intenso y lo rectifica a blanco. La rectificación de todos los píxeles elimina el ruido de color. Este algoritmo no puede ser utilizado si en la imagen no existe un color blanco.

2.2. Procesamiento de la imagen

El procesamiento en una imagen se realiza para descartar, resaltar o diferenciar formas de la imagen de entrada para su posterior análisis o simplemente para modificar la imagen original (21). Se utilizan técnicas de segmentación de color, histogramas, uso de filtros, etc.

Segmentación del color

Como anteriormente se ha mencionado, una imagen está compuesta por píxeles y estos píxeles tienen asociado un valor según su canal. Estos píxeles obtienen un valor entre 0 y 255, y ese valor corresponde a la intensidad de su canal. Según la imagen puede tener unos u otros canales, pero generalmente se utilizan imágenes con los canales RGB (o BGR que cambiaría el orden de los valores) para su visualización, como se vio en las Figuras 2.3, 2.4 y 2.5.

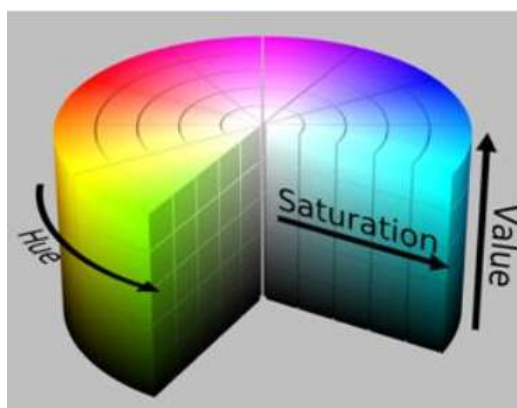


Figura 2.6: Cilindro HSV

Otro tipo de modelo de color es el HSV o HSB (Hue, Saturation, Value o Brightness) mostrado en la Figura 2.6, que diferencia sus canales en matiz, saturación y brillo o valor. Este espacio de color suele utilizarse para la búsqueda de colores por su fácil interpretación. Modificar los parámetros en el modelo de color HSV es más intuitivo que en el modelo RGB. El modelo RGB utiliza la adición de color lumínico, es decir, añade color en una imagen. Si se tiene los parámetros R, G y B para modificar mediante barras se ve como al modificar un valor, las otras barras iniciarían desde ese color para añadir o quitar luminosidad de su canal. (Figura 2.7)



(a) Valores RGB para un color negro (b) Valores RGB para un color rojo (c) Valores RGB para un color azul

Figura 2.7: Barras del modelo RGB

Sin embargo, el modelo de color HSV es bastante intuitivo al modificar sus valores con tres barras, ya que solo una de ella modifica el matiz dejando en ella todo el espectro de color, como se muestra en la Figura 2.8. Cuando se habla de segmentación del color en una imagen se habla del proceso de separar partes de la imagen según los valores de los píxeles, indiferente de los canales en los que esa imagen se vea. Es decir, separar las regiones de la imagen donde se encuentren colores deseados.

En la Figura 2.9 se ve como de la Figura 2.4 se han eliminado rangos de valores correspondientes con los verdes (a) y los azules (b). Esta práctica se consigue con el uso de máscaras. Una máscara es como una ventana que deja o no ver a través de ella según su forma. Esta máscara oculta los matices de la barra de color HSV correspondiente con los verdes (a) y los

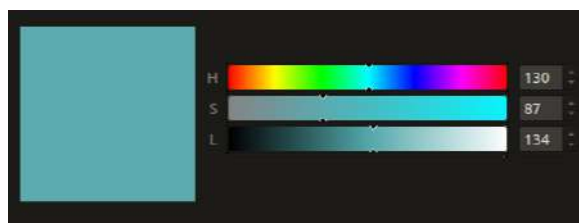


Figura 2.8: Barras del modelo HSV

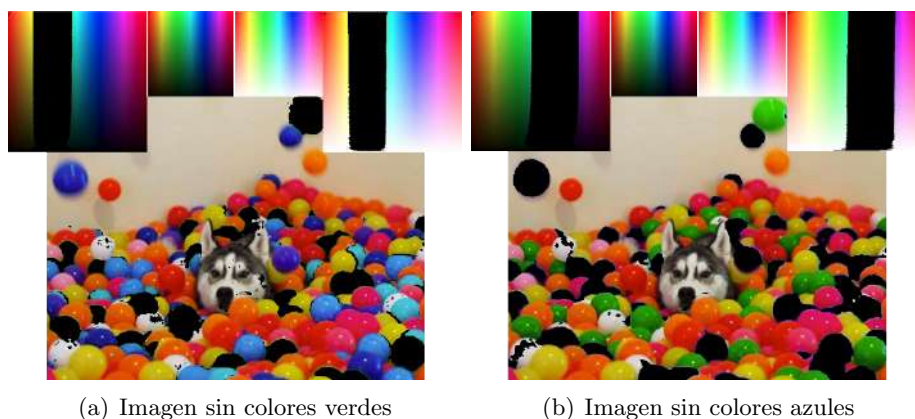


Figura 2.9: Sustracción de color

azules (b).

Otro método consiste en eliminar todos los colores salvo el deseado. En este caso se utiliza la máscara para solo dejar pasar los matices en la barra de color HSV que se desea. Las otras dos barras que corresponden a la saturación y el valor no se ha modificado, es decir, están en sus rangos máximos (Figura 2.10).

En el campo de la visión artificial la segmentación de color es un tema que bajo condiciones controladas es bastante útil y sencillo. Pero si estas condiciones no son controladas, la iluminación variante, sombras, reflejos, incluso calidad de la cámara, puede tratarse de un tema complejo y difícil de controlar.

Imagen binaria

Cuando en una imagen se utiliza una máscara, por ejemplo, para la segmentación de color, los píxeles que la máscara oculta obtienen un valor de 0 en su matriz, y los demás valores obtendrán un valor igual al de su imagen original. Entonces la función de la máscara es multiplicar los valores de la matriz de entrada por 0 o 1 según oculte o no el píxel. Estas máscaras son matrices compuestas por unos y ceros que representa la imagen en blanco y negro. Se podría decir que una imagen en blanco y negro es una máscara que anula los píxeles de las regiones negras y no afecta a las regiones blancas.

Una imagen con elementos en blanco y negro es una imagen binaria por lo tanto una

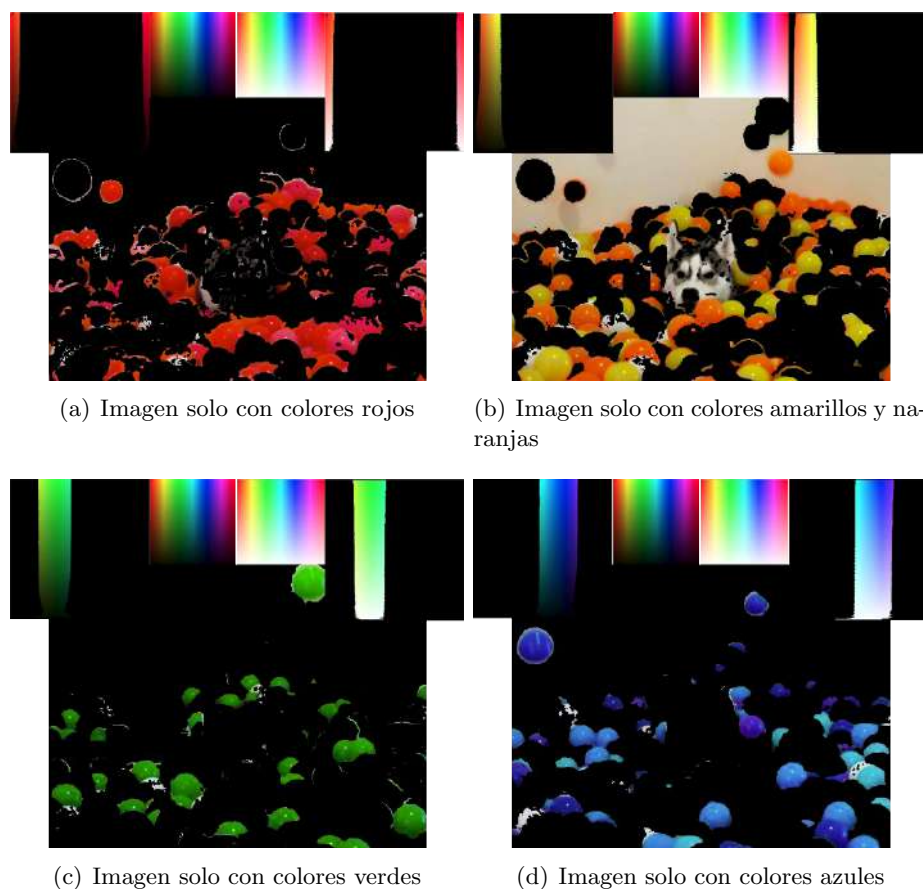


Figura 2.10: Sustracción de colores

máscara también lo es (22).

Filtros

Para tratar el ruido en las imágenes digitales se hace uso de filtros espaciales. Los filtros espaciales se implementan mediante un proceso de convolución espacial. Este proceso utiliza operaciones de procesamiento puntual (píxel por píxel) o por grupo de píxeles (sobre vecindades) en la imagen de entrada. Existen diferentes tipos de filtros, pero todos utilizan operaciones similares alrededor de su núcleo (kernel).

El núcleo o kernel en los filtros viene dado en forma de $n \times m$ y refleja el número de píxeles que abarca la operación de filtrado (Figura 2.11), a mayor tamaño del núcleo, mayor es el suavizado (equilibrio entre los valores del píxel) y más detalles se eliminan (23).

- El filtro de media o del promedio utiliza la información de la intensidad de los píxeles vecinos a un píxel central. El proceso de convolución utiliza un promedio ponderado

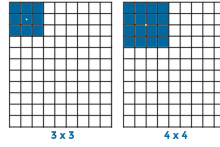


Figura 2.11: Núcleo del filtro

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

Figura 2.12: Matriz con intensidades

del píxel de entrada y de sus vecinos inmediatos para calcular el píxel de salida. De la Figura 2.12 se obtiene:

- Para un núcleo de 3x3:

$$media = \frac{16 + 26 + 16 + 26 + 41 + 26 + 16 + 26 + 16}{9} = 23,22$$

- Para un núcleo de 5x5:

$$media = \frac{1 + 4 + 7 + 4 + 1 + 4 + 16 + \dots + 16 + 4 + 1 + 4 + 7 + 4 + 1}{25} = 10,92$$

- El filtro gaussiano al igual que el filtro de media utiliza un promedio ponderado del píxel central, pero en este filtro se utilizan unos pesos según su proximidad. Veamos como se obtiene en este caso.

- Para un núcleo de 3x3:

$$gaussiano = \frac{16 + 26 + 16 + 26 + 41x2 + 26 + 16 + 26 + 16}{10} = 25$$

- Para un núcleo de 5x5:

$$gaussiano = \frac{1 + 4 + \dots + 16x2 + \dots + 41x3 + 26x2 + 7 + \dots + 4 + 1}{25} = 20,92$$

- El filtro de mediana también utiliza los píxeles vecinos para la salida del píxel central, pero no realiza ninguna operación. Se refleja el valor mediano de todos los píxeles como se muestra en la Figura 2.13.

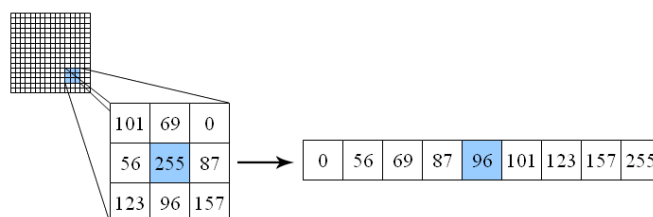
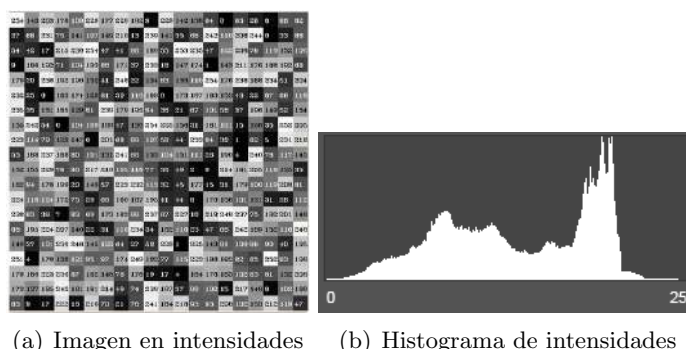


Figura 2.13: Filtro mediana

Histogramas

Un histograma es una representación gráfica de la distribución de intensidades de una imagen y cuantifica el número de píxeles para cada valor de intensidad considerado (24). En las imágenes de la Figura 2.14 se muestra un ejemplo de cómo se calcula el histograma (Figura 2.14 (a)) y el histograma de intensidades de la Figura 2.1, dando como resultado la Figura 2.14 (b). En el histograma, el eje de abscisas obtiene valores de 0 a 255, número de valores diferentes de intensidad, y en el eje de ordenadas de 0 al número de píxeles que existan en la imagen correspondientes con la intensidad del eje de abscisas.



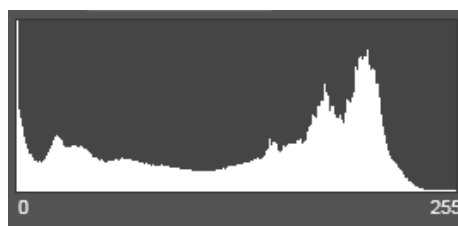
(a) Imagen en intensidades (b) Histograma de intensidades

Figura 2.14: Imagen en escala de grises con su histograma de intensidades

Para imágenes a color (con canales R, G y B) se obtiene tres histogramas correspondientes a las intensidades de cada uno de los canales y el histograma correspondiente a la imagen RGB, como se muestra en la Figura 2.15.

Con el uso de los histogramas se puede comparar similitudes de intensidad de una imagen a otras, a esto se le conoce como Backprojection (Retroproyección). El método de Backprojection utiliza una imagen modelo (descriptor) de la que se calcula su histograma, que se utiliza para buscar similitudes en la imagen y se consigue un mapa de semejanzas donde se muestran las zonas que tienen los mismos colores. En la Figura 2.16 se puede ver el modelo, que en este caso es una bola roja y algo de fondo, el mapa de similitudes se muestra en la esquina superior derecha, utilizado como máscara en la imagen original para obtener la imagen salida.

Existen otros métodos para la búsqueda de áreas iguales al modelo o descriptor, el Tem-



(a) Histograma RGB



(b) Histogramas de los canales R, G y B

Figura 2.15: Histogramas de la imagen RGB

plate matching es un método que utiliza una imagen como descriptor y crea un mapa de semejanzas. Estos ya se verán más adelante.



Figura 2.16: Método de Backprojection

Local binary patterns

El Local binary patterns (LBP) es un descriptor que diferencia texturas, común en la visión por ordenador. Un descriptor es un modelo matemático que extrae las características útiles para el reconocimiento de objetos (25). Los descriptores trabajan con las intensidades de los píxeles y sus vecindades. El cálculo del LBP en una imagen se realiza comparando si la intensidad del píxel es mayor o menor que la del píxel central. Esta comparación devuelve 1 si es mayor y 0 si es menor y realiza una conversión en binario para obtener el valor de

intensidad del píxel central. Un ejemplo para la conversión de un píxel a LBP se muestra en la Figura 2.17.



Figura 2.17: Conversión a LBP. Fuente Coursera (1)

Esta conversión se realiza en todos los píxeles de la imagen y se obtiene un valor de 0 a 255, los píxeles de bordes y esquinas se pueden ignorar al no tener con quien comparar.

Una variante sobre los LBP es el llamado LBP uniforme que reduce los valores de los píxeles al rango de 0 a 59. Esto se debe a la eliminación de rangos que pocas veces se dan. Los rangos uniformes dan a conocer si el píxel se encuentra en un borde, esquina o no, dentro de la imagen. La Figura 2.18 muestra los diferentes rangos que se pueden encontrar, siendo los no uniformes los descartados.

Búsqueda de contornos

El contorno es la línea que limita una figura o forma. Estas líneas pueden calcularse por la variación de intensidad que existen a su alrededor. Otra forma de identificar bordes y esquinas en la imagen y que se basa en la diferencia del valor de la intensidad del píxel con los píxeles vecinos es por el cálculo del gradiente. La orientación del gradiente en un píxel indica la dirección en la que existe un aumento de intensidad, y su magnitud la diferencia de intensidades que hay. (Figura 2.19)

Se puede obtener el valor del gradiente calculado la diferencia entre intensidades del los

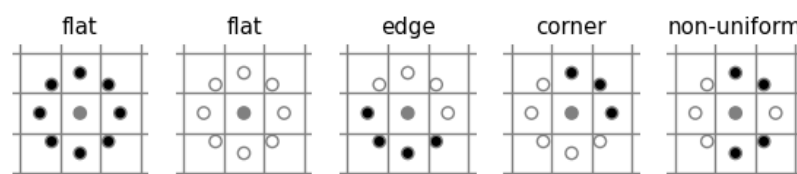


Figura 2.18: Diferentes rangos del LBP

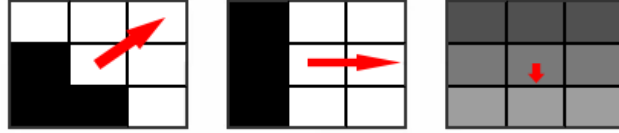


Figura 2.19: Gradientes de píxeles

píxeles. La ecuación 2.1 indica la diferencia de intensidad entre el píxel de la derecha y el de la izquierda, conociendo el gradiente en el eje x de ese píxel. Para calcular la diferencia en el eje y se utiliza la ecuación 2.2.

$$dx = I(x + 1, y) - I(x - 1, y) \quad (2.1)$$

$$dy = I(x, y + 1) - I(x, y - 1) \quad (2.2)$$

Una vez obtenidos los gradientes de los píxeles se eliminan los que no se consideran máximos, dejando así solo los píxeles que tienen mayor diferencia de intensidades que se corresponden con los bordes (1; 26).

Un de los métodos para la detección de bordes es el algoritmo de Canny, que utiliza múltiples etapas para la búsqueda de contornos. En estas se:

- Aplica un filtro gaussiano para la reducción de ruido:

$$k = \frac{1}{159} \begin{pmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{pmatrix}$$

- Encuentra el gradiente de intensidad de la imagen aplicando filtros en las direcciones x e y,

$$G_x = \begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix} \quad G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{pmatrix}$$

y magnitud y dirección con:

$$G = \sqrt{G_x^2 + G_y^2} \quad \theta = \arctan \frac{G_y}{G_x}$$

- Se eliminan los que no son máximos y se dejan los considerados como bordes que sean líneas finas.

- Se aplican dos umbrales, el superior, que es el gradiente mínimo que debe tener el píxel para considerarse borde, y el inferior, que por debajo de él no se considera borde. Si el píxel está entre los dos umbrales se considerará borde si está conectado a un píxel que se considera borde, es decir, está por encima del umbral superior.

En la Figura 2.20 se puede ver el resultado de aplicar este algoritmo a la Figura 2.4.

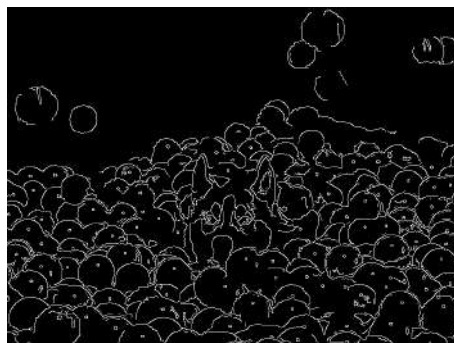


Figura 2.20: Obtención de los bordes con el algoritmo Canny de la Figura 2.4

Rastreo (CAMshift OpenCV)

Las técnicas de seguimiento de objetos en la visión por computador son muy comunes para obtener información sobre la posición de los objetos en la imagen. El desplazamiento medio o “meanshift”, es una técnica de análisis de espacio de características no paramétrica para localizar los máximos de una función de densidad (27). Es decir, busca el conjunto de píxeles más denso y converge hacia ellos de forma iterativa. Esta técnica da como resultado el centro del conjunto de píxeles más densos y una ventana que los contiene, esta ventana debe ser previamente definida.

En la Figura 2.21 se puede ver como el punto inicial $C1_o$ con su ventana $C1$, se desplaza hacia la nube de puntos más densa, dando como resultado $C1_r$ en una iteración. Al cabo de las pertinentes iteraciones la ventana $C2$ se obtiene la nube de puntos más densa (28).

Una variante de esta técnica es el CAMshift (Continuously Adaptive Mean Shift) que adapta el tamaño y la orientación de la ventana obtenida por el meanshift.

2.3. Lenguaje y Librerías

Para la elaboración del TFG se han utilizado el lenguaje de programación Python y varias de sus librerías, todo ello implementado en un entorno virtual.

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional (29).

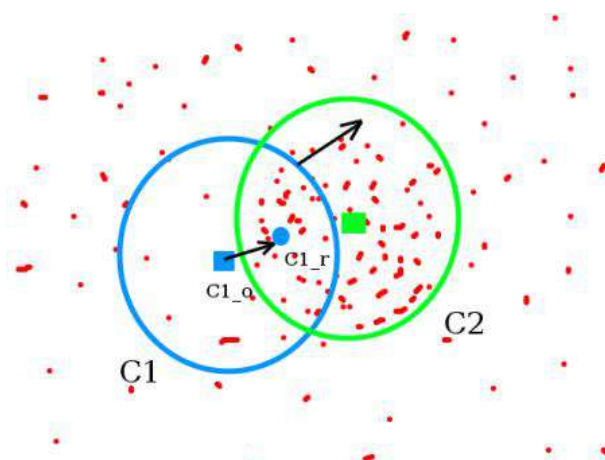


Figura 2.21: Técnica de meanshift

El uso de entornos virtuales permite tener entornos de programación totalmente separados y aislados, de manera que el intérprete y las librerías de proyectos diferentes no entren en conflicto. Esta herramienta es útil cuando en un mismo sistema se tiene más de un intérprete de Python instalado, por ejemplo Python 2.7 y Python 3.6. También se utiliza para evitar conflictos entre librerías que utilicen diferentes versiones (30).

En el campo de la visión por computador existen multitud de métodos y algoritmos. OpenCV (Open Source Computer Vision Library) es una biblioteca de software de aprendizaje por computador y visión por computador de código abierto. Se creó para proporcionar una infraestructura común para aplicaciones de visión por computador y para acelerar el uso de la percepción de máquinas en los productos comerciales.

La biblioteca tiene más de 2.500 algoritmos optimizados, que incluyen un conjunto completo de algoritmos de visión por computador y aprendizaje automático tanto clásicos como de última generación. Estos algoritmos se pueden usar desde el reconocimiento de caras o identificación de objetos hasta realidad aumentada (31).

Como es común trabajar con imágenes se utiliza Numpy, una extensión de Python, que agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel (32).

Otro conjunto de módulos del lenguaje Python que permiten la creación de videojuegos en dos dimensiones de una manera sencilla es Pygame (33).

Para el control del mouse y teclado mediante programación existen módulos como PyAutoGUI (34), o el módulo Time que está previsto de varias funciones que trabaja con los valores de tiempo.

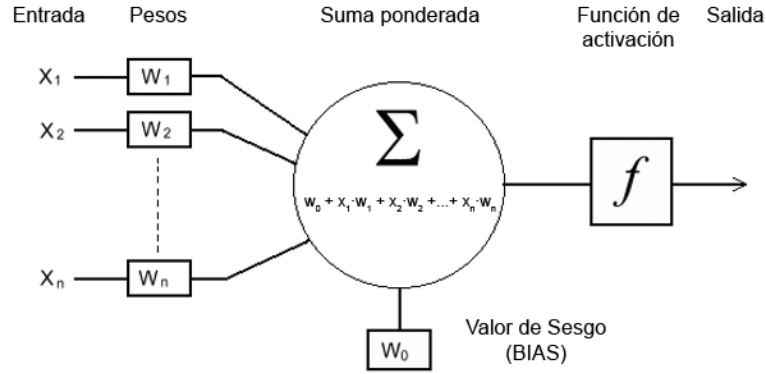


Figura 2.22: Representación gráfica de una neurona artificial simple

2.4. Redes neuronales

Las redes neuronales ya existían a nivel teórico en el siglo XX y se conocían aplicaciones en las que podrían ser útiles, aunque no se conocía como se podían entrenar de forma eficiente. Geoffrey Hinton fue el primer investigador en presentar técnicas para entrenar eficientemente las redes neuronales.

Las redes neuronales forman parte del machine learning e intentan modelar comportamientos inteligentes. Sus comportamientos y estructuras avanzadas emergen de la interacción de muchas partes más simples que trabajan conjuntamente. Estas partes simples se denominan perceptrón o neuronas. Una agrupación de neuronas forman la red neuronal artificial (o perceptrón multicapa).

Una neurona es la unidad básica de procesamiento dentro de una red. Tiene unas entradas a las que se les asignan unos pesos según la importancia de esta entrada (Figura 2.22).

Un ejemplo de asignación de pesos sería, la acción de comer es más importante que la acción de escuchar música, por lo tanto en la entrada de comer se le asignaría un peso mayor que al de escuchar música. Con estos valores la neurona realiza una suma ponderada dando como resultado una salida. Esto se puede representar como la función matemática de la ecuación 2.3.

$$f(x) = x_1 \Delta w_1 + x_2 \Delta w_2 + \dots + x_n \Delta w_n \quad (2.3)$$

Esta función con una sola entrada es similar a una recta, ecuación 2.4:

$$f(x) = w_0 + x \Delta w_1 \quad (2.4)$$

donde el término independiente sitúa a la recta en el eje vertical. Este valor también existe en la neurona y se le llama valor de sesgo (BIAS en inglés), dando una forma a la función, similar a un modelo de regresión lineal. La neurona también depende de un término llamado función de activación. Este término es importante para la implementación de las redes neuronales con múltiples capas.

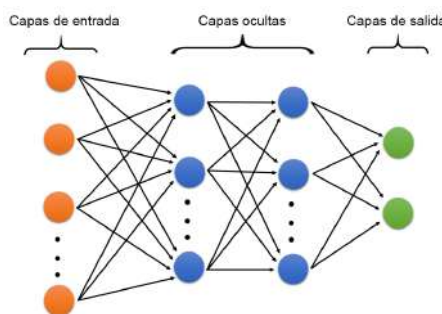


Figura 2.23: Capas de una red neuronal

Las neuronas de una red están recogidas en capas y existen tres tipos de capas en una red neuronal (Figura 2.23):

- Capa de entrada: son las neuronas que obtienen los valores de entrada y devuelven su valor a la siguiente capa. Esta puede ser una capa oculta o la capa de salida.
- Capa oculta: son las capas que se encuentran entre la capa de entrada y la de salida, reciben los valores de las neuronas que se encuentran en la capa anterior y devuelven su salida a la capa siguiente. El número de capas puede variar según el tipo de red neuronal y las neuronas que en cada capa se encuentran.
- Capa de salida: es la última capa, recibe los valores de su capa anterior y sus neuronas devuelven el resultado de la red neuronal.

Las neuronas son sumas ponderadas de las entradas en la primera capa y estas devuelven una salida que reciben otras capas realizando otra suma ponderada. Esto se podría simplificar en una sola suma ponderada, es decir, la red neuronal sería equivalente a una sola neurona, por lo que no se obtendrían los resultados esperados. Por este motivo se aplica la función de activación en cada neurona.

La función de activación es una manipulación no lineal que se aplica a la salida de la neurona, se encarga de devolver una salida a partir de un valor de entrada, normalmente devuelve valores entre los rangos $(0, 1)$ o $(-1, 1)$ (35; 13).

Existen diferentes tipos de funciones de activación en las que se busca que sus derivadas sean simples para minimizar el coste computacional.

- Función Escalonada: se define un umbral y si el valor de la salida es mayor a ese umbral la salida es 1, o 0 si el valor es menor de ese umbral. Su cambio brusco no es muy interesante a la hora de implementar por lo que no se utiliza demasiado (Figura 2.24 (a)).
- Función Sigmoide: es una función que cambia gradualmente saturando los valores grandes en 1 y los pequeños en 0. Es muy útil para la presentación de probabilidades (Figura 2.24 (b)).

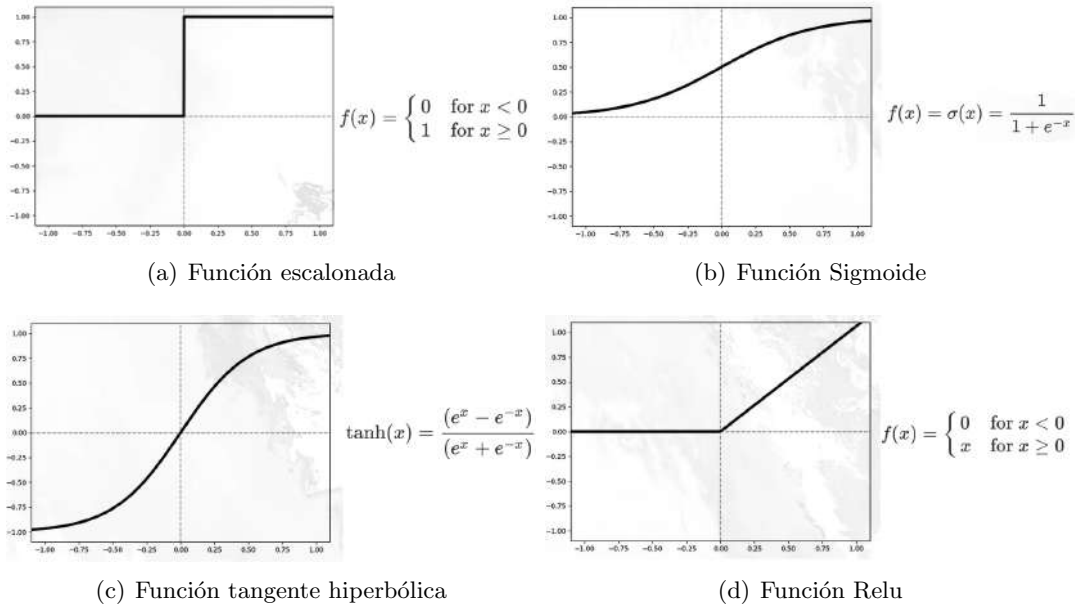


Figura 2.24: Funciones de activación

- Función tangente hiperbólica: similar a la anterior. Está centrada en cero y sus rangos están entre -1 y 1, saturando los valores pequeños en -1 (Figura 2.24 (c)).
- Función Relu o Unidad de rectificación lineal: esta función anula los valores negativos y mantiene los positivos (Figura 2.24 (d)).
- Función softmax: transforma las salidas a una representación en forma de probabilidades, de tal manera que el sumatorio de todas las probabilidades de las salidas es de 1.

Backpropagation

El método de evaluación para minimizar el error una vez obtenido el resultado de salida se le llama Backpropagation. Cuando a la salida de la red neuronal se obtiene un resultado de error, este método busca que neuronas de la red deberán modificar sus valores para reducir el error de salida. Backpropagation evalúa el peso de cada neurona en el error, desde la capa de salida a la de entrada y modifica sus parámetros para obtener el mínimo error.

En la Figura 2.25 se muestra como este método asigna los pesos del error calculado a la salida en cada neurona, detectando así a cuales de ellas deben modificar los parámetros.

Tipos de redes neuronales

Se pueden clasificar según su tipo de aprendizaje. En base a esto existen tres tipos de aprendizaje:

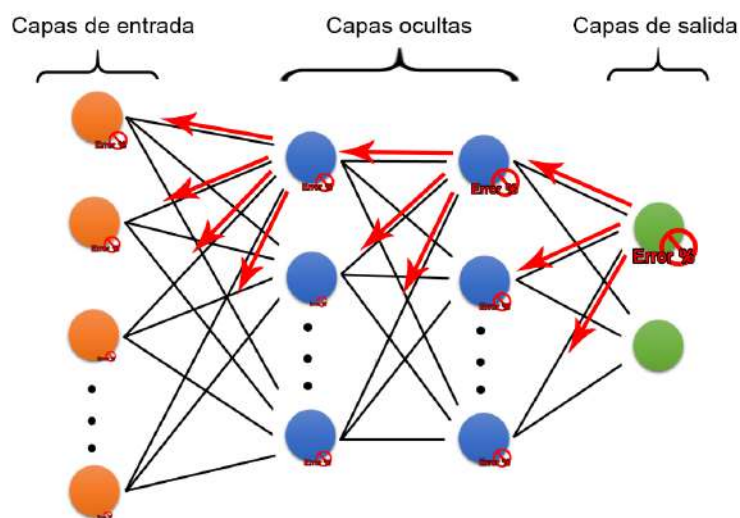


Figura 2.25: Backpropagation

- Aprendizaje supervisado: toma una gran cantidad de ejemplos con el resultado esperado y el algoritmo es capaz de dar resultados correctos incluso con ejemplos que nunca haya visto.
- Aprendizaje no supervisado: no necesita el resultado esperado, solo ejemplos, siendo capaz de clasificar las semejanzas que entre ellos existan.
- Aprendizaje reforzado: se utiliza cuando no existe un resultado claro o este es algo más abstracto. En este caso se recompensa o penaliza la resolución de acciones que deben ser resueltas o no respectivamente.

Otro tipo de clasificación se basa en el tipo de estructura, aunque existen muchos tipos diferentes que hacen difícil esta clasificación. Por lo que se detallan los tipos de redes más utilizados:

- Redes neuronales profundas (DNNs): no existe un umbral claro de cuándo pasa de ser una red neuronal superficial a una profunda, aunque si está claro que la red debe tener múltiples capas no lineales (capas ocultas) (Figura 2.26).
- Redes neuronales convolucionales (CNNs): al aumentar el número de capas en una red neuronal para hacerla más profunda, aumenta la complejidad de la red y permite modelar funciones más complicadas. Esto hace que el número de parámetros aumente exponencialmente. Estas redes reducen drásticamente la cantidad de parámetros que deben ajustarse. También, se puede decir que una red neuronal convolucional (CNN) es una red multicapa, diseñada inicialmente para reconocer patrones visuales, lo que permite mejorar ciertas propiedades respecto de otras arquitecturas. Por ello son utilizadas

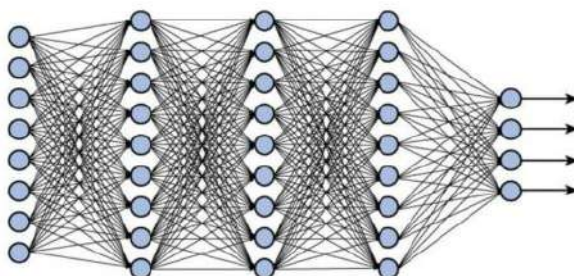


Figura 2.26: Red neuronal profunda

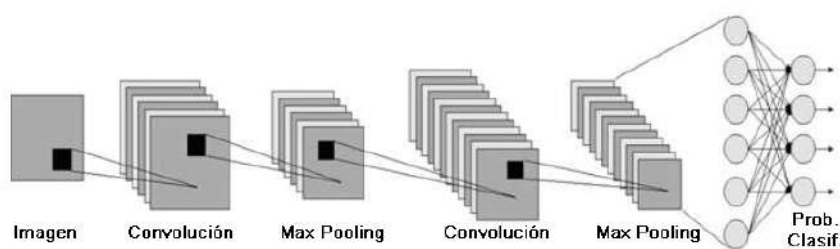
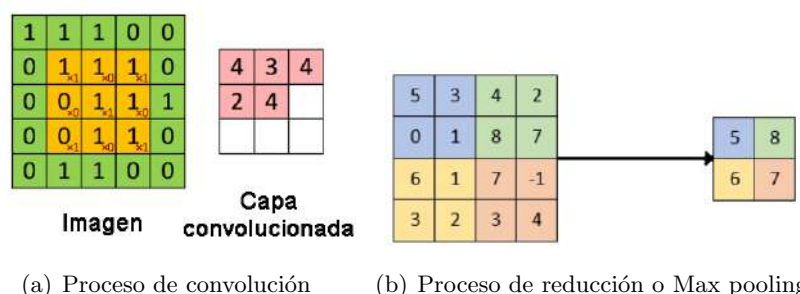


Figura 2.27: Red neuronal convolucional(2)

en visión por computador. El proceso de convolución se muestra en la Figura 2.27, que se explicará en la Sección 3.3.3.2.

- Redes generativas adversarias (GANs): se trata de una red que contiene dos redes neuronales enfrentadas, una de ellas trata de crear o generar (imágenes, textos, sonidos..) y la otra se trata de una red discriminadora. Esta red discriminadora tiene dos entradas, la salida de la red generadora y los ejemplos que debe crear la red generadora, y trata de identificar si la salida del generador se corresponde con la entrada de ejemplos. Si no corresponde el discriminador da una salida de error y la red generativa debe de volver a generar. En la Figura 2.28 se muestra su estructura (36).

Las primeras imágenes generadas por la red se trata de ruido y cambios de colores, tras cientos de intentos comienza a realizar formas y con miles o millones de intentos la red

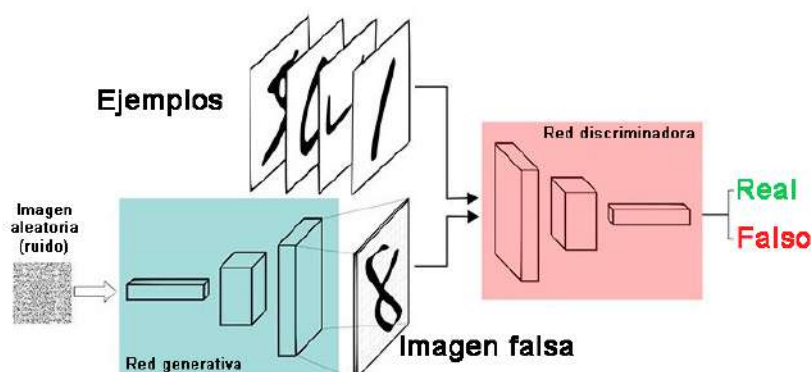


Figura 2.28: Red generativa adversaria

generadora consigue realizar una imagen que se pueda confundir con una real.

- Redes neuronales recurrentes (RNNs): se denominan recurrentes ya que repiten la misma tarea para cada elemento de una secuencia, y la salida se basa en los cálculos anteriores. Se puede decir que este tipo de redes tiene una “memoria” a corto plazo que captura la información de los pasos anteriores. También existen redes de memoria a largo plazo (LSTM) que junto con las redes neuronales convolucionales se utilizan como parte del modelo para generar descriptores de imágenes sin etiquetar.
- Auto ML: se trata de un sistema diseñado por Google. Este algoritmo crea redes neuronales con estructuras más eficientes, variando el número de capas y neuronas, reduciendo así el error de la tarea que debe realizar la red.

Modelos

Al entrenar una red neuronal se obtiene un modelo. Este modelo tiene ya los parámetros definidos para intentar solucionar el problema para el que se le ha entrenado. Los modelos funcionan mejor cuando tienen muchos parámetros, lo que da funciones con aproximaciones muy poderosas. Sin embargo, esto significa que deben tener conjuntos de datos muy grandes. Debido a eso entrenar modelos con gran cantidad de parámetros puede ser una tarea con un costo computacionalmente intenso que requiere desde días a semanas de entrenamiento.

Grande empresas como Google, OpenAI y muchas otras, se dedican a crear modelos muy eficientes donde su costo computacional es significativo. Estos modelos se les conoce como modelos pre-entrenados. Estos modelos han sido entrenados por una base de datos llamada ImageNet. Esta proporciona imágenes con sus correspondientes anotaciones indicando el contenido de las imágenes. Cuando se utiliza una red pre-entrenada no se necesita calcular sus parámetros, ya que están definidos. Ejemplos de modelos pre-entrenados se muestran en la Figura 2.29.

Otro término importante se denomina *Transfer learning* y consiste en el proceso de transferencia de un modelo a otro. Es decir, desde un modelo pre-entrenado se realiza una transferencia de su red con sus parámetros a otro modelo donde se modifican algunas de las capas. Un

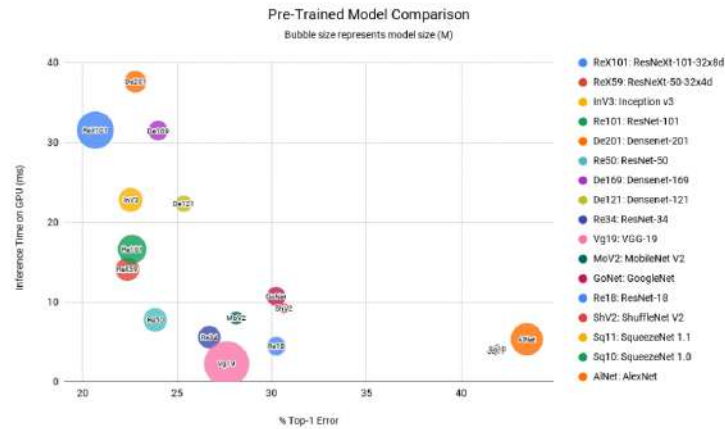


Figura 2.29: Modelos pre-entrenados

ejemplo de *Transfer learning* es desde un modelo ya entrenado como es el de Mobilenet v2, que es capaz de identificar 1000 clases diferentes de objetos, modificando su capa de salida se pueden detectar otros tipos de objetos que antes no era capaz de identificar, dejando los parámetros de las capas ocultas sin modificar para evitar el costo computacional.

Capítulo 3

Desarrollo y Metodología

En este capítulo se explica el desarrollo del software, los objetivos que deben cumplir y cómo se han abordado.

Como se vio en el apartado 1.3, el dispositivo recoge información desde una cámara. Esta información permite conocer los movimientos y gestos de la mano con la finalidad de controlar diferentes tareas, como la manipulación de imágenes, vídeos o juegos. El software diseñado debe poder ejecutarse tanto en un ordenador, como en una Raspberry.

Primero se va a hablar de los dispositivos utilizados así como el software que se ha elegido, luego de los métodos de detección que han servido para la aproximación de la decisión final. Estos no han sido elegidos por ser demasiado estrictos, necesitar demasiada potencia de cálculo o no ser lo suficiente flexibles. Se indica cuál es el método utilizado para conseguir la posición de la mano. Tras conseguir ubicación de la mano se clasifica el gesto que tiene, para ello se han utilizado dos métodos. Estos se basan en la comparación de formas y en redes neuronales convolucionales. Por último se ve el diseño del software para el ajuste de los parámetros y la depuración de la clasificación.

3.1. Hardware y software

A continuación se detallan los programas y dispositivos utilizados para la elaboración y preparación del sistema.

3.1.1. Ordenador

Para el desarrollo del software se han utilizado dos ordenadores, uno de sobremesa y un portátil, para realizar pruebas de forma más rápida que en la Raspberry, ya que en ocasiones se ha necesitado mayor velocidad de procesamiento.

Las propiedades del hardware y del sistema operativo son:

- Ordenador Sobremesa:
 - Windows 10 Home x64-bits.
 - Intel Core i7-4790 3.6GHz.
 - Disco Duro 1 TB + 2 TB.
 - Disco Duro SSD 120 GB.
 - Memoria 2x4GB DDR3 1600MHz.
 - GeForce GT 750-Ti 2GB GDDR5.
 - Pantalla Lg 22MP57VQ-P 22" LED IPS.
 - Webcam Logitech Quickcam Express.
- HP Pavilion:
 - Windows 10 Home x64-bits.
 - AMD A4-5150M APU 2.70GHz.
 - Disco Duro 930 GB.
 - Memoria 4 GB.
 - AMD Radeon HD 8350G.
 - HP TrueVision HD.

Para la creación del código del software se ha utilizado Spyder. Este es un entorno de desarrollo integrado multiplataforma de código abierto para programación científica en lenguaje Python (37). A través de Anaconda está disponible para Windows. Anaconda es una distribución de los lenguajes Python y R, utilizada en ciencia de datos y aprendizaje automático.

Anaconda utiliza un sistema de gestión de paquetes *conda*, con el que se hace más sencillo de instalar, correr y actualizar software. Con Anaconda también se puede crear fácilmente entornos virtuales e instalar paquetes mediante pip o conda, lo que resulta cómodo para instalar los paquetes que se han utilizado.

3.1.2. Raspberry Pi 3 B+

Gran parte del tiempo se ha trabajado dentro del sistema de Raspberry, ya sea para la elaboración del código del software como para las pruebas de su funcionamiento.

Los detalles técnicos de este dispositivo se ven reflejados en la Tabla 3.1.

Modelo del producto	Raspberry Pi 3 Model B+
Peso del producto	4,54 g
Dimensiones del producto	9 x 6 x 2 cm
Número de procesadores	1
Procesador	4 núcleos de 64 bits
Frecuencia procesador	1,4 GHz
Arquitectura del procesador	ARM62
Capacidad de la memoria RAM	1 GB
Salida de vídeo	Puerto HDMI
Número de puertos USB 2.0	4
Voltaje	5 voltios
Soporte	WiFi y Bluetooth

Tabla 3.1: Detalles técnicos Raspberry Pi 3 Model B+

Inicialmente la Raspberry viene sin tarjeta SD y sin sistema operativo, por lo que se ha de elegir uno para comenzar. Entre los sistemas operativos que se pueden encontrar, nos centraremos en un sistema operativo de escritorio, el utilizado será Raspbian, aunque existen múltiples opciones parecidas como Kali Linux o Ubuntu Mate. Existe la posibilidad de utilizar el software para el control de sistemas para centros multimedia, pero eso se dejará para trabajos futuros.

El primer paso es descargar el sistema operativo Raspbian y montarlo en una tarjeta SD, en mi caso he usado una tarjeta de 32 GB, ya que la primera de 16 GB se quedó pequeña. Una vez montada la imagen de Raspbian se puede poner en marcha la Raspberry y configurarla. También hay que actualizar el sistema desde el terminal utilizando los siguientes comandos:

```
...: $ sudo apt-get update
...: $ sudo apt-get dist-upgrade
```

Tras tener configurado y actualizado Raspbian, ahora se procede a realizar la instalación de OpenCV dentro de un entorno virtual, en el que se va a utilizar Python 2.7.

Para la creación de entornos virtuales en Raspbian se utiliza *virtualenv*, que se instala fácilmente con el gestor de paquetes pip. Para crear entornos virtuales se debe escribir en el terminal:

```
...: $ mkvirtualenv (nombre_entorno) -p python2
```

Dentro del entorno virtual se instala el intérprete Python2.7, también se instalarán las librerías necesarias como son NumPy, PyGame, Pyautogui y OpenCV.

A la hora de compilar OpenCV en la Raspberry se tuvo que incrementar el tamaño del paso, o *swapsize*, de 100 a 1024 MB. Esto es importante para conseguir compilar OpenCV en la Raspberry. Tras su compilación se debe volver a restablecer el *swapsize* (38).

3.2. Primeras aproximaciones

Con las herramientas anteriores se puede comenzar a realizar el desarrollo del software. En este se han utilizado diferentes métodos para la detección de objetos, basados en diferentes descriptores. Estos descriptores encuentran una conexión entre los píxeles para la extracción de las características visuales de la imagen. Estas características pueden ser propias del píxel, como es el color, o locales. Una característica local describe las propiedades de un píxel en relación a sus vecindades. Por tanto, los descriptores basados en características locales permiten encontrar bordes o texturas, entre otras, dentro de la imagen.

Los métodos de detección de objetos utilizados para las primeras aproximaciones utilizan una imagen modelo como descriptor. Con este se generan candidatos que proporcionan la ubicación del modelo en la imagen. Con los resultados obtenidos se clasifica si ese candidato se trata del objeto a detectar. Tras la clasificación se puede o no depurar el proceso para obtener mejores resultados.

En la Tabla 3.2 se muestran los diferentes métodos utilizados y su proceso para la detección de la mano. Con el método Template matching se obtiene un mapa de semejanzas que indicará posibles candidatos y su localización, aplicando un umbral a este mapa se clasifican los candidatos como válidos. Con los LBP se obtienen los candidatos según la posición de la ventana deslizante, aplicando un umbral en la comparación se obtiene los candidatos válidos. Con la técnica non maximum suppression se depuran los candidatos. Para el Backprojection se generan candidatos según las características del histograma del modelo y se clasifican si pertenecen a él.

Método	Generación de candidatos	Clasificación	Depuración
Template matching	Mapa de semejanzas	Umbral del mapa	-
LBP	Posición de la ventana deslizante	Umbral de la comparación	Non maximum suppression
Backprojection	Rango de valores del histograma	Rango de valores	-

Tabla 3.2: Métodos utilizados en las primeras aproximaciones

A continuación se detalla el proceso de los métodos y los resultados obtenidos.

3.2.1. Template matching

El modelo que utiliza este método como descriptor es una imagen. Permite encontrar áreas que se asemejan a la imagen proporcionada, creando un mapa de semejanzas. En la Figura 3.1 se pueden ver unas muestras.

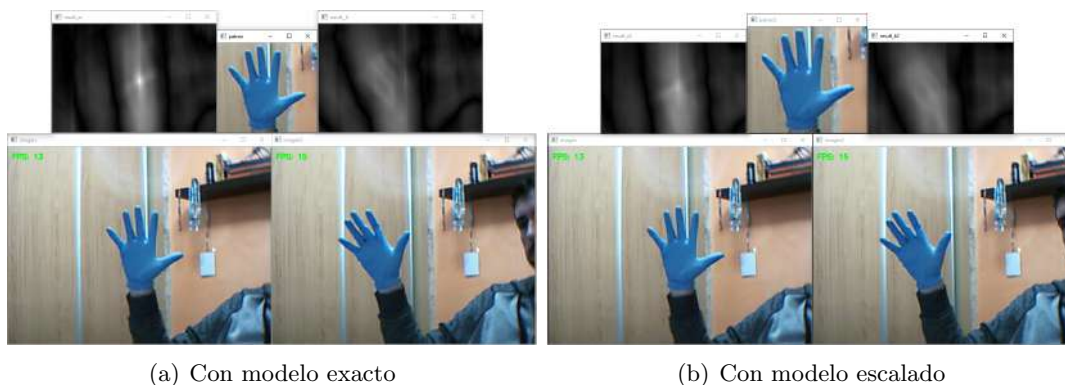


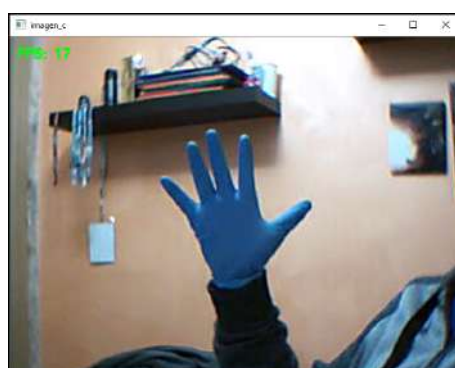
Figura 3.1: Técnica de Template matching en condiciones “buenas”

En la generación de candidatos de este método se desplaza el descriptor por todos los píxeles de la imagen de captura. Como resultado de esta técnica es un mapa de semejanzas donde los píxeles más intensos indican una mayor aproximación del modelo a la imagen entrada. Existen seis tipos de aproximaciones que se pueden implementar con las funciones de OpenCV para la búsqueda del mapa de semejanzas. Estos son basados en la diferencia, correlación cruzada y coeficiente cruzado, además de sus normalizados.

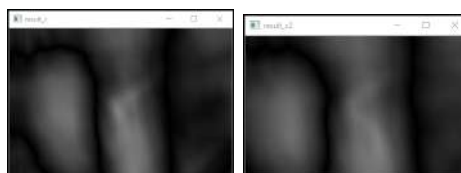
Al mapa resultante se le aplica un valor umbral donde los píxeles con mayor intensidad que el umbral son posibles ubicaciones del objeto. Con el resultado de la clasificación se determinar la localización y el gesto. Los píxeles más intensos indican la posición del objeto, y el modelo del gesto que se trata.

Cualquiera de estas aproximaciones son demasiado estrictas, en la Figura 3.1 (a) se muestra la posición donde se encuentra el modelo (el guante), pero esta comparación está lejos de la realidad, un ligero movimiento de muñeca reduce su eficacia. También en la Figura 3.1 (b), el modelo ha sido modificado con la intención de que parezca que la mano está más cerca, es decir, se ha escalado. El resultado al utilizar esta técnica se puede ver que se ve comprometido. Estas pruebas se han hecho dentro de unas condiciones no demasiado desfavorables, el fondo en la imagen es el mismo o varía ligeramente y la mano se ha modificado levemente.

En la Figura 3.2 se ha elegido una imagen con un fondo diferente y se han vuelto a utilizar los dos modelos anteriores. Para esta imagen se puede observar que los resultados de uno u otro modelo (Figuras 3.2 (b) y (c)) ya no dan información relevante de la posición de la mano, sus salidas, los mapas de semejanzas, no muestran valores que ayuden a intuir su posición. Esto se debe a que la técnica de template matching busca que todos los píxeles del modelo se encuentren en la imagen entrada. Esto es una práctica muy difícil en vídeo por el simple



(a) Nueva imagen con un fondo diferente



(b) Resultado con modelo 1 (c) Resultado con modelo escalado

Figura 3.2: Técnica de Template matching en condiciones “malas”

cambio de intensidades en el píxel por ruido.

Por lo tanto, al ser tan estricto este método, no aporta seguridad para la localización de la mano ni la clasificación del gesto.

3.2.2. Local binary patterns

Este método utiliza el descriptor LBP para la detección de la mano. Este es un buen descriptor para diferenciar texturas y clasificarlas.

Diferente del método anterior, este utiliza una imagen en escala de grises como modelo, se calcula su imagen LBP y se genera su histograma. La generación de candidatos al igual que el método anterior se realiza con una ventana deslizante. Esta ventana genera los histogramas de la imagen y los compara con el histograma del modelo. Estas comparaciones determinan zonas de la imagen donde existen candidatos.

Aplicando un umbral se clasifica como candidato o no, según la comparación. La posición de la ventana del candidato determina la posición del objeto y la imagen comparada el gesto. Como ya se vio en la Sección 2.2 los LBP trabajan con las intensidades de los píxeles vecinos y se compara con la del píxel central. Esto significa que los LBP son invariantes a cambios monotónicos del nivel de gris y a traslaciones, lo que le hace más flexible que el método anterior.

En la Figura 3.3 se muestra el cálculo de la imagen en LBP del modelo anterior y su

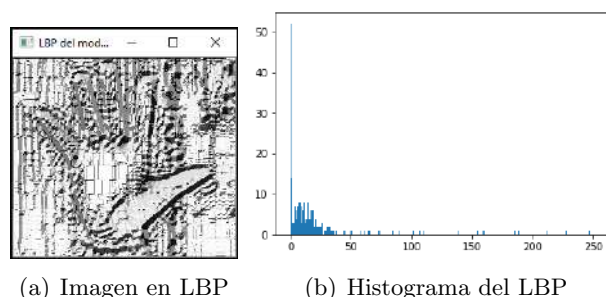


Figura 3.3: Modelo LBP

histograma. Se realiza el mismo procedimiento que en el caso anterior, pero comparando los histogramas. Para realizar comparaciones de los histogramas las dimensiones de las imágenes deben ser iguales, por ello se realiza un barrido con una ventana deslizante con las dimensiones del modelo y se comparan los histogramas de la ventana deslizante con la del modelo. Esta comparación tiene como salida un número entre 0 y 1, siendo mayor para histogramas más parecidos.

En la Figura 3.4 se muestran los ejemplos anteriormente utilizados y cuales son los resultados de las comparaciones. En el primer par de imágenes, el modelo pertenece a la Figura 3.4 (a) y se muestran las salidas de la ventana deslizante que tienen un valor superior al 0,99 en la comparación del histograma con el modelo. El resultado han sido 21 ventanas deslizantes (Figura 3.4 (b)).

Para el segundo par de imágenes, la Figura 3.4 (c) tiene una pequeña variación con el modelo y se muestran las salidas superiores al 0,985 en la comparación. El resultado han sido 4 ventanas (Figura 3.4 (d)).

Para el último par de imágenes, la Figura 3.4 (e) tiene diferencias notables con el modelo y se muestran las salidas superiores al 0,87 en la comparación. El resultado han sido 631 ventanas (Figura 3.4 (f)), de las cuales se pueden ver que algunas si pertenecen a la mano, pero otras no (son falsos positivos).

Según se reduce el umbral del resultado de la comparación, el número de candidatos, en este caso las ventanas deslizantes, aumenta. Pero aún teniendo un umbral alto como es en el último par de imágenes de la Figura 3.4 que se sitúa al 87 %, existen un número muy elevado de ventanas deslizantes que se consideran candidatos. En la primera y segunda imagen pasa lo mismo, pero en menor medida. Esto se debe a que una ventana deslizante en la posición (x,y) y otra ventana en la posición $(x+p, y)$, siendo 'p' el número de píxeles de desplazamiento de cada iteración, van a tener parte de la misma imagen en su interior. Como los LBP son invariantes a traslaciones reflejan en la comparación del histograma partes que se compararon en la ventana anterior y se compararán en la siguiente.

Para evitar esta redundancia de candidatos se utiliza una solución, llamada *Non maximum suppression (NMS)*. Esta técnica descarta los candidatos que se superpongan más de la mitad

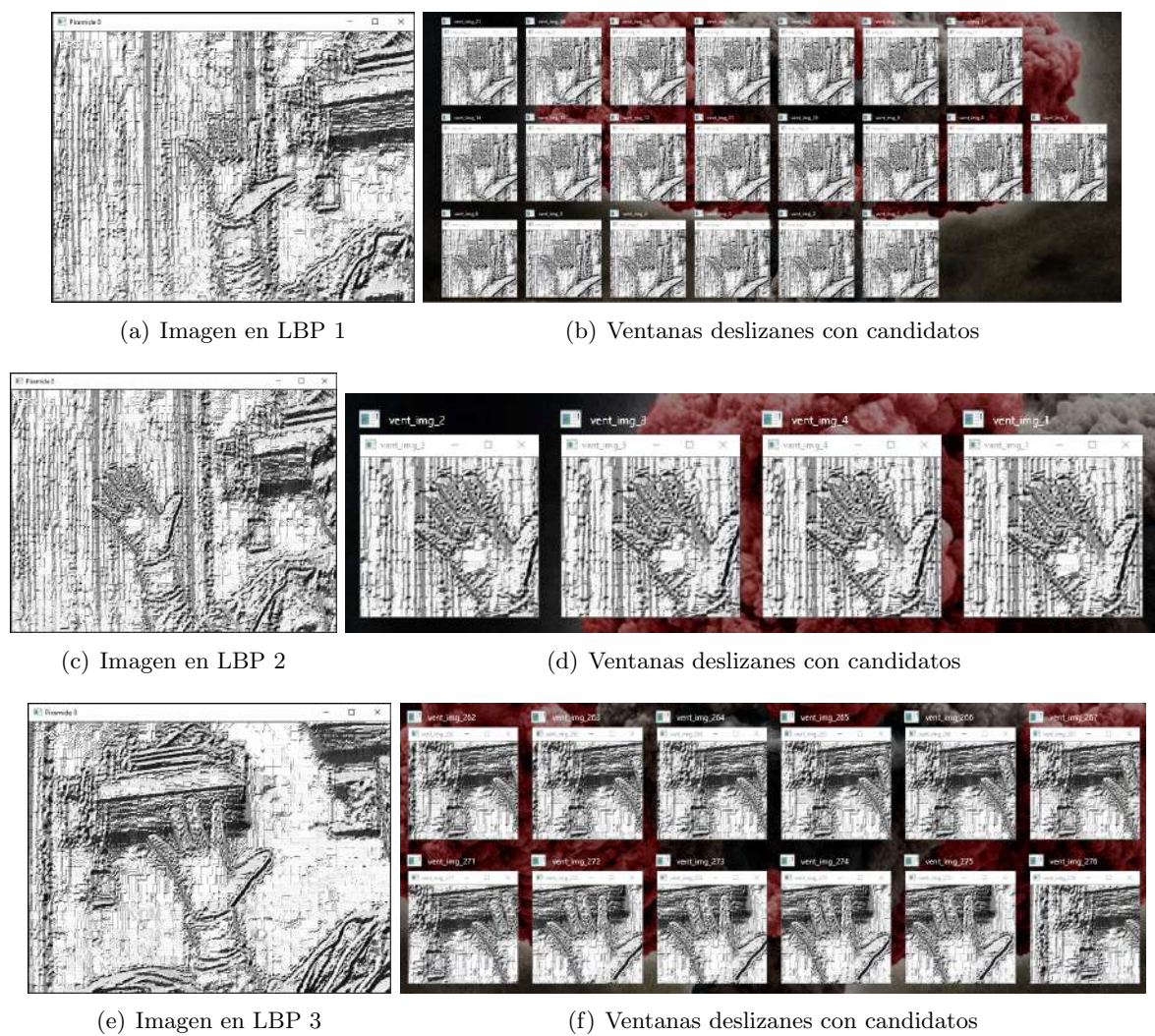


Figura 3.4: Comparación de diagramas LBP

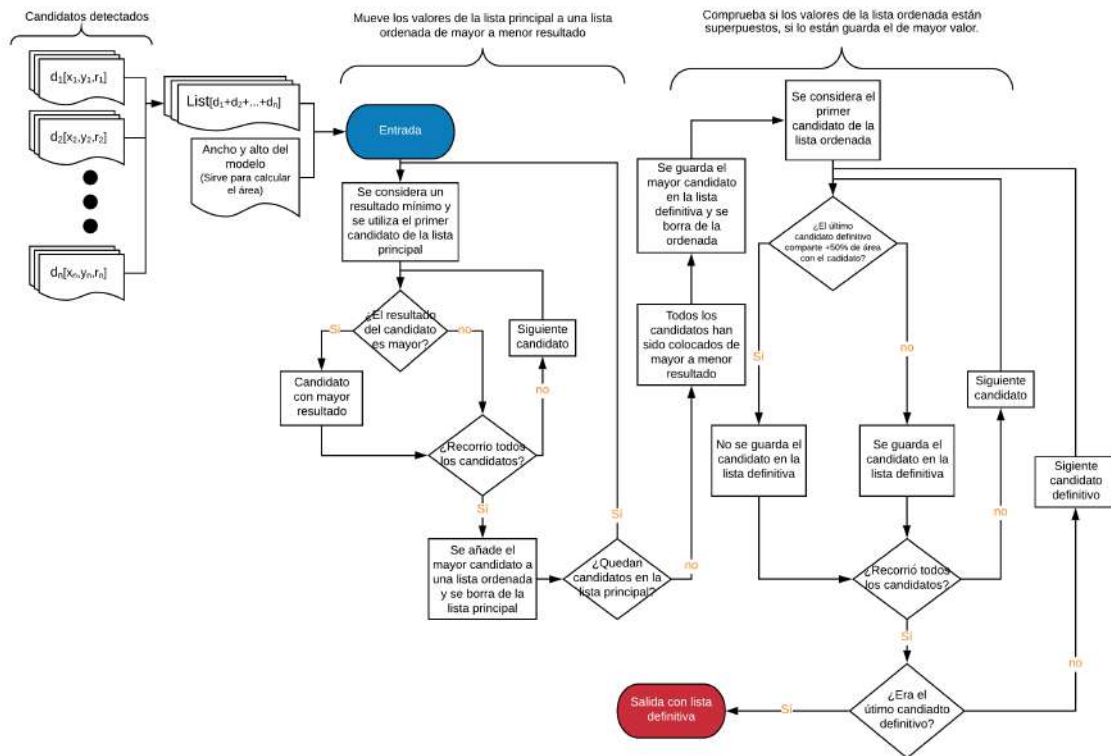


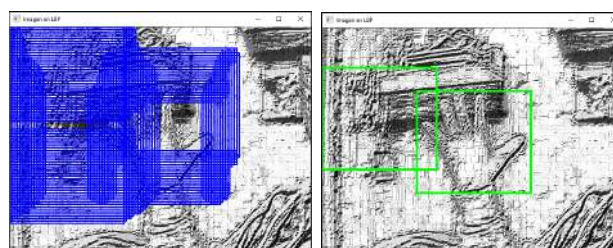
Figura 3.5: Diagrama non maximum suppression

del área, dejando los que tengan un mayor resultado en la comparación con el modelo.

En el diagrama de flujo de la Figura 3.5 se puede ver el funcionamiento de esta técnica. Recibe como entrada las dimensiones del modelo (dimensiones ventana deslizante) y todos los posibles candidatos en una lista. Cada candidato tiene la posición en la que se encuentra y el resultado de la comparación. Se ordena la lista de mayor a menor orden en base al resultado. El primer candidato con mayor resultado será el primer candidato definitivo.

El candidato definitivo se compara con las posiciones de los candidatos que están ordenados por resultado. Con la diferencia de sus posiciones y las dimensiones del modelo se puede saber la cantidad de área que tienen solapado. Si el solapamiento es del 50 % o mayor el candidato no se considera definitivo, si es menor del 50 % sí. Una vez considerado definitivo será este el que se compara con los candidatos ordenados hasta que no se añadan más definitivos.

Aplicando la técnica de NMS se obtiene a la salida solo los mejores candidatos. En la Figura 3.6 se muestran las salidas remarcadas con una comparación de histogramas superior



(a) Sin aplicar Non maximum suppression (b) Aplicando Non maximum suppression

Figura 3.6: Non maximum suppression

al 87 % sin aplicar el NMS (a) y aplicándolo (b).

El Non maximum suppression ha conseguido eliminar todos los candidatos redundantes y mostrar los que más parecido tienen al histograma del modelo. También se puede ver más claramente como ha tenido un falso positivo.

Tras estos ejemplos, la búsqueda de candidatos por LBP parece un método bastante robusto, identificando en cualquier parte la posición de la mano. Pero al trabajar con vídeo la posición de la mano no solo se desplaza en la dimensión x e y , la profundidad también es un concepto que se debe tener en cuenta.

Una solución para este problema es escalar el modelo o la imagen. Si es la imagen la que se escala se realizan diferentes escalados de menor tamaño, según la dimensión del modelo. Al agrupar estas imágenes aparecen como una pirámide (Figura 3.7). Tras crear esta, se desliza el modelo canónico por todas ellas comparando los histogramas. Para cada imagen escalada se aplica un filtro gaussiano, generando imágenes más desenfocadas.

Los modelos que estén más lejos se encontrarán en las imágenes más grande de la pirámide mientras que los más cercanos en las más pequeñas. El número de imágenes en la pirámide depende del tamaño del modelo y del paso de la ventana deslizante en x e y .

En resumen, para generar candidatos por LBP en una imagen se debe:

- Convertir la imagen en LBP.
- Crear imágenes pirámide según el tamaño del modelo.
- Deslizar una ventana por la imagen y comparar los histogramas con el modelo.
- Aplicar el NMS.

Una vez conocido el proceso, se realiza en el vídeo. Este se debe realizar en cada fotograma o captura del vídeo. La librería OpenCV no dispone de funciones para la conversión de la imagen en LBP de forma optimizada, lo que comparar cada píxel de la imagen con sus vecinos requiere un costo de procesamiento que no es despreciable. Esta conversión de cada fotograma

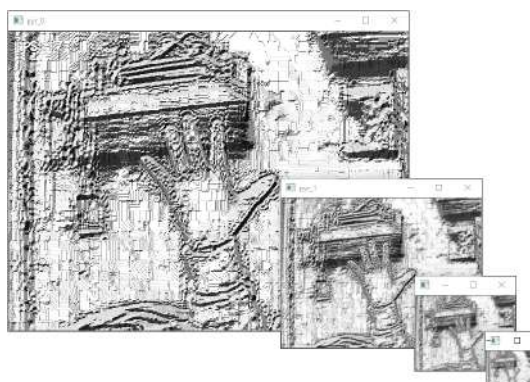


Figura 3.7: Imágenes LBP en pirámide

en LBP, su creación de pirámides, el deslizamiento de las ventanas y la aplicación del NMS, hace a este proceso algo lento a tiempo real. Cabe destacar que este proceso no solo genera los candidatos, también lo clasifica según el modelo a comparar y se depuran los candidatos con el *NMS*. Al existir diferentes gestos para clasificar, son varios los modelos que se deben comparar.

3.2.3. Backprojection

Este método utiliza como descriptor el color de un modelo. Este modelo es una imagen del que se genera un histograma y define unos rangos de valores. En este método es más apropiado hablar de extracción de características en lugar de generador de candidatos. Siendo estos rangos los que determinan la clasificación de las características de la imágenes. Un ejemplo, en el rango de valores del histograma del modelo están los colores azules, la clasificación realiza la extracción de los colores azules de la imagen pero no los verdes, ni rojos, etc.

Esta clasificación es semejante a utilizar máscaras. Utilizando esta para extraer las características que están dentro de los rangos de valores. En la Figura 3.8 se muestran los pasos a seguir para obtener la segmentación del color según un modelo.

- Paso 1: se elije un modelo que extraiga la mayor cantidad de características del guante y se evitan las que no pertenezcan a este. Si el modelo tiene colores del fondo, a la salida habrá fondo, lo que deteriora la salida obtenida en este paso (Figura 3.8 (a)).
- Paso 2: se convierte la imagen de entrada y la del modelo al modo de color HSV. Este paso puede ser prescindible, pero facilita la comprensión de los histogramas. Si este paso se obvia, los canales serían azul, verde y rojo para el histograma (Figura 3.8 (b)).
- Paso 3: se calculan los histogramas, tanto del modelo como de la imagen entrada. Estos histogramas muestran las intensidades de los píxeles en cada uno de los canales que son matiz en azul, saturación en verde y valor en rojo (HSV). Se puede ver en el histograma del modelo que hay un pico de matiz entorno al valor de 100. Ese valor es el color azul del guante, que también se puede ver en la imagen entrada (Figura 3.8 (c)).

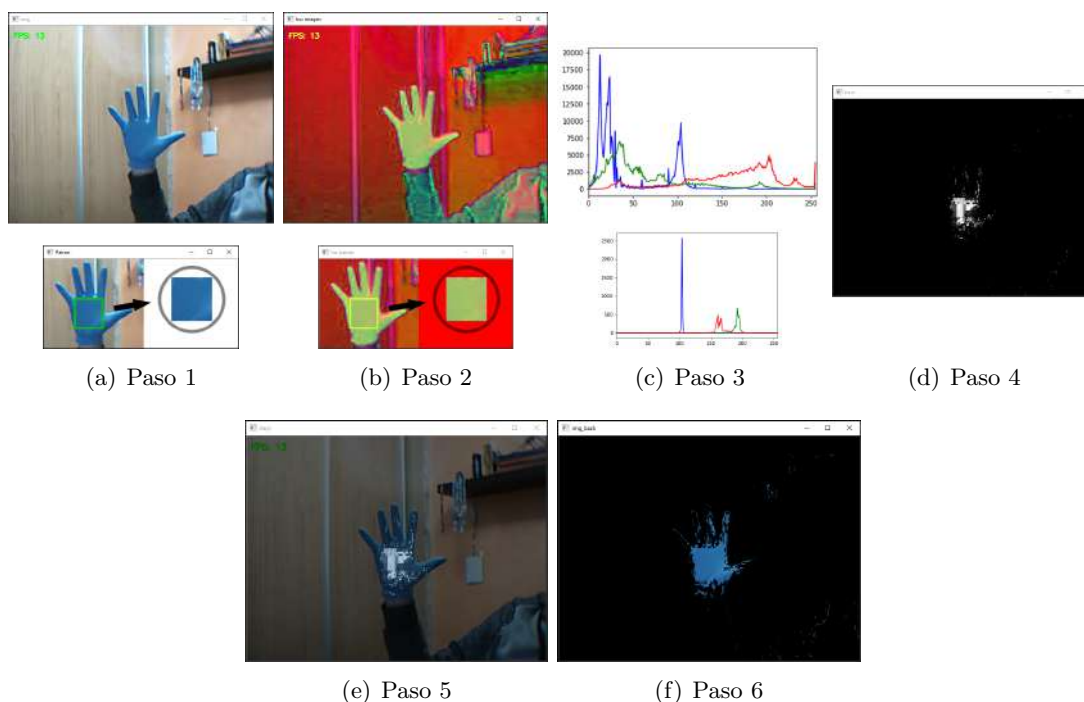
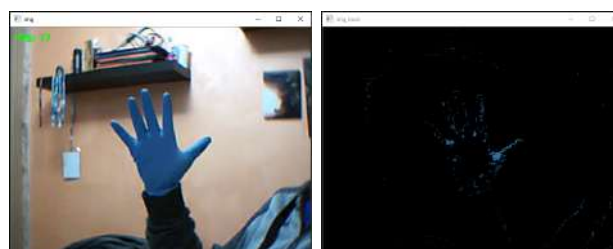


Figura 3.8: Proceso para la técnica de Backprojection

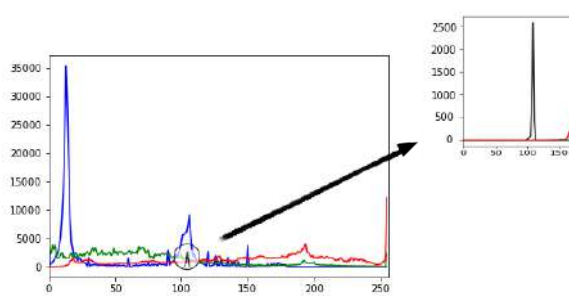
- Paso 4: uso de la función *cv2.calcBackProject* que realiza la clasificación. Da como resultado una imagen que representa la probabilidad que tiene el píxel de pertenecer al guante, basado en el histograma del modelo. Esta imagen se puede utilizar como máscara (Figura 3.8 (d)).
- Paso 5: combinación de la máscara con la imagen de entrada. Cabe destacar que esta no es una imagen binaria al tener grises, y utiliza los píxeles con intensidad mayor de 0 como 1 (Figura 3.8 (e)).
- Paso 6: imagen obtenida al mostrar solo los valores del modelo en la imagen de entrada (Figura 3.8 (f)).

Esta técnica de Backprojection es bastante flexible con un buen modelo. Si el modelo no recoge los suficientes valores o recoge valores que no se ajustan, no se obtiene una buena salida. El modelo anterior se ha utilizado en la Figura 3.9 (a), dando como salida la Figura 3.9 (b). Esta salida no es de calidad, el motivo se puede observar comparando los histogramas. Como se muestra en la Figura 3.9 (c), el histograma del modelo está remarcado con un círculo en negro dentro del histograma de la imagen de entrada. Estos colores recogidos por el modelo son una porción de los azules recogidos en la imagen de entrada, además de sus valores de saturación y valor, dando a la salida los valores comunes.



(a) Imagen entrada

(b) Imagen salida



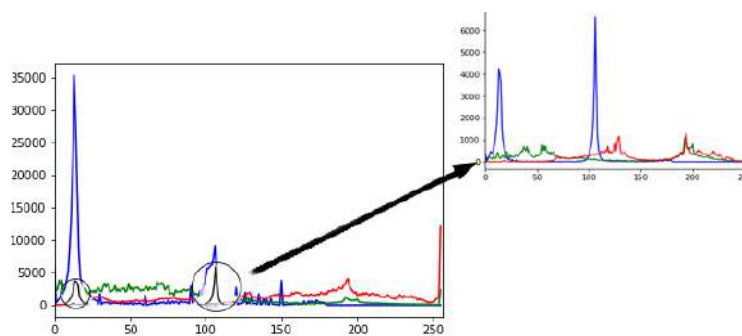
(c) Comparación de histogramas

Figura 3.9: Modelo con defecto de características



(a) Modelo

(b) Imagen salida



(c) Comparación de histogramas

Figura 3.10: Modelo con exceso de características

Otro problema en el modelo puede ser por obtener demasiados valores, dando colores del fondo u objetos cercanos (Figura 3.10). En el modelo de la Figura 3.10 (a) se extraen características en exceso, que no corresponden con el guante. Estas características son los píxeles del fondo, sombras, destellos o incluso la manga de la ropa. Al utilizar este modelo, en la salida (Figura 3.10 (b)) se observa como abarca muchas más características de las que se requieren. Al comparar los histogramas (Figura 3.10 (c)) se puede observar que este modelo presenta otro pico entre los valores 0 y 30. Estos valores son los referidos al color de la pared del fondo.

Por tanto, este método puede implementarse de forma sencilla y es flexible, pero se debe de obtener un buen modelo previamente.

3.3. Diseño del sistema de clasificación

Con los métodos utilizados anteriormente no se han conseguido obtener unos buenos resultados. Para ello se han buscado nuevos métodos para la detección de la mano y la clasificación de su gesto.

En la Tabla 3.3 se muestran los métodos utilizados en el diseño del sistema.

Método	Generación de candidatos	Clasificación	Depuración
Rango de valores definidos + CAMshift	Rango de valores definidos	Rango de valores	CAMshift
Se obtiene la posición de la mano			
Búsqueda de contornos y comparación de formas	Rango de valores definidos + CAMshift + Búsqueda de contornos	Comparación de formas	Distancia Media Pesos
Red neuronal convolucional	Mediante aprendizaje automático	Predicción del modelo	Predicciones seguras
Se obtiene el gesto de la mano			

Tabla 3.3: Métodos utilizados en el sistema de clasificación

Primero, con la combinación de rango de valores y CAMshift se encuentra la posición de la mano. Con esta ubicación se genera una ventana que la recoge.

Luego, se realizan dos métodos diferentes para la clasificación del gesto. Uno busca los contornos de la mano y los compara con patrones para conocer el gesto. El otro hace uso de redes neuronales convolucionales para predecirlo.

3.3.1. Rango de valores y técnica CAMshift

Rango de valores definidos

El método de Backprojection necesitaba un modelo para definir un rango de valores y extraer

sus características. Pero este rango de valores puede ser definido sin la necesidad de un modelo y ajustarse según interesa.

En la Figura 3.11 se muestran los pasos a seguir para la extracción de características del objeto.

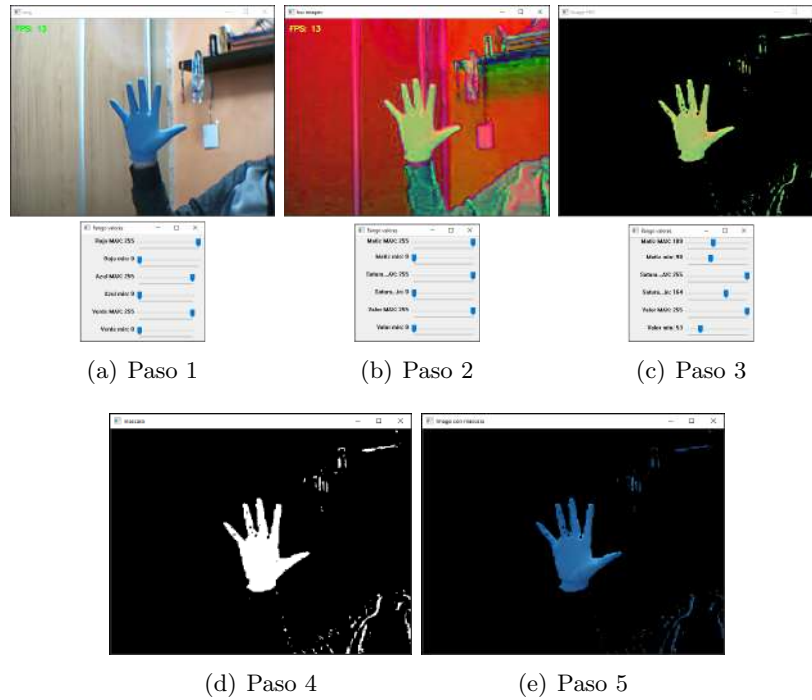


Figura 3.11: Pasos para generar candidatos por Rango de valores

- Paso 1: el descriptor en la imagen de entrada, en modelo de color RGB, abarca todo el rango de valores. Por tanto, se extraen todas las características de la imagen (Figura 3.11 (a)).
- Paso 2: se convierte la imagen de entrada al modelo de color HSV. Al modificar el modelo de color los rangos cambian, modificando ahora los canales del matiz, saturación y valor en vez de rojo, verde y azul (Figura 3.11 (b)).
- Paso 3: se ajustan los rangos de valores a las características del píxel que se desea como salida, en este caso el guante azul, intentando reducir al máximo otras partes de la imagen. Los rangos de valores se modifican con una barra de desplazamiento variando sus valores entre 0 y 255 (intensidades de los canales), se debe tener especial cuidado con los valores mínimos de saturación y valor que representan en mayor medida a los destellos y sombras. Los valores máximos de saturación y valor, en este caso, siempre deben estar a 255 (Figura 3.11 (c)).

- Paso 4: al modificar el rango de valores se realiza la clasificación, obteniendo a la salida las características que están en los rangos y generando una máscara. Esta se puede utilizar para obtener una salida de cualquiera de los modos de colores (Figura 3.11 (d)).
- Paso 5: al utilizar la máscara obtenida en la imagen de entrada, se obtiene la imagen con solo las características que se ajustan a los rangos de valores (Figura 3.11 (e)).

Este método ha sido el más eficiente para resaltar las características del guante, pero no ofrece información de su posición ni del gesto de la mano. Para obtener información relevante de la posición de la mano se ha utilizado la técnica CAMshift.

Técnica CAMshift

Esta es una técnica iterativa. Aproxima el centro de la ventana a la nube de puntos más densa. Proporcionando a esta técnica las características (o máscara) obtenida en las Secciones 3.2.3 y 3.3.1, es capaz de obtener la posición de la mano y su tamaño. Con esta información se puede contener a la mano dentro de una ventana.



Figura 3.12: Uso de la técnica CAMshift

En la Figura 3.12, se muestra el funcionamiento de esta técnica ya implementada. Tras la obtención de la máscara con rangos de valores definidos (Figura 3.12 (a)) se busca iterativamente su centro y recoge su máxima densidad en un recuadro (Figura 3.12 (b)). Dando unos márgenes al recuadro generado por la función CAMshift, se puede obtener una ventana que recoge la mano.

3.3.2. Búsqueda de contornos y comparación de formas

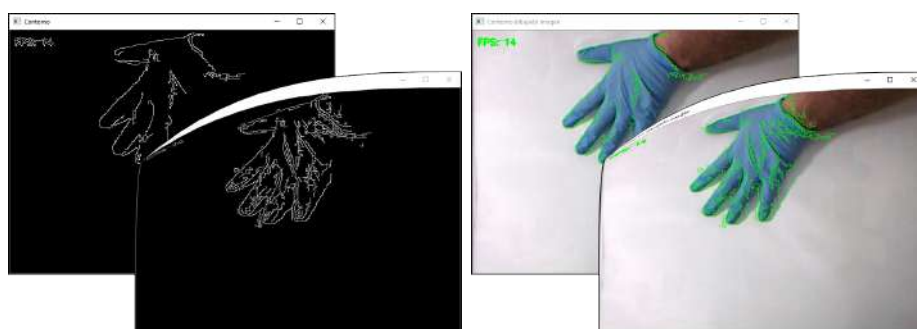
Existen algoritmos que permiten una buena aproximación para obtener los bordes de los objetos. Estos descriptores describen las regiones, contornos y formas para las imágenes 2D (39). Para conocer el gesto de la mano se puede buscar la forma que tiene y realizar una comparación con patrones, siendo estos formas ya clasificadas. Según estas comparaciones se clasifica el gesto, indicando cuándo está la mano abierta, cerrada o de perfil.

3.3.2.1. Búsqueda de contornos

Para obtener el contorno de la mano se ha utilizado el algoritmo Canny, que ya se vio en la Sección 2.2.



(a) Imagen de entrada



(b) Contornos Canny con diferentes umbrales (c) Dibujando los contornos en la imagen de entrada

Figura 3.13: Contornos Canny

Este algoritmo es muy útil para buscar contornos en una imagen, pero aún con sus dos umbrales obtener un borde limpio de ruido es muy difícil. En la Figura 3.13 (a) se muestra una imagen con un fondo neutro para mejorar la detección del borde. En las imágenes de la izquierda (Figura 3.13 (b)) se ha aplicado el algoritmo Canny de la librería OpenCV con diferentes umbrales. Y en las de la derecha (Figura 3.13 (c)) se han dibujado esos contornos. Se puede ver que la salida del contorno es bastante difusa, por lo que se debe tratar la imagen para mejorar su calidad.

Utilizando la combinación de los métodos de segmentación de color con la búsqueda de contornos, se puede obtener un contorno más definido. En la Figura 3.14 se muestra la máscara (a), su contorno (b) y el contorno resaltado en la imagen de entrada (c). Este proceso da un mejor resultado que el anterior, pero aun así se puede ver que existen partes que el rango de valores no ha considerado, dejando un contorno dentro de la mano.

La utilización de filtros gaussianos ayuda a la máscara a eludir regiones de pocos píxeles, para obtener un contorno más definido, pero esto no es suficiente. Cuando en una imagen se encuentra más de un contorno se utiliza una jerarquía para identificar los diferentes contornos. En el contorno de la Figura 3.14 (b) existen dos contornos uno el borde de la mano, y otro los píxeles de dentro que se produjo por error. Cada contorno viene identificado por 4 variables



(a) Máscara utilizando rango de valores (b) Contorno de la máscara (c) Dibujando el contorno en la imagen entrada

Figura 3.14: Contorno de una máscara

dentro de un vector que recogen información sobre él y los demás contornos.

$$[P0, P1, P2, P3]$$

[Siguiente, Anterior, Interiores, Exteriores]

Los primeros dos elementos dan información sobre los contornos encontrados a un mismo nivel del borde al que se observa, y los dos últimos sobre su entorno interior y exterior. Según este método la Figura 3.14 (b) tiene 4 contornos. El primero de ellos (contorno 0) es el borde de la mano por fuera. Con una estructura $[-1, -1, 1, -1]$ que indica que no tiene contornos al mismo nivel, tiene un contorno en su interior y ninguno fuera. El siguiente (contorno 1) es el borde de la mano por dentro. Con una estructura $[-1, -1, 2, 0]$. Tampoco tiene contornos al mismo nivel, pero indica que el contorno 0 lo está envolviendo y a su vez este envuelve al contorno 2. El contorno 2 son los píxeles de ruido por la parte exterior del borde. Con una forma de $[-1, -1, 3, 1]$. Envuelve al contorno 3 y está dentro del contorno 1. Y por último el contorno 3 con una estructura de $[-1, -1, -1, 2]$, que está dentro del contorno 2. Para identificar el contorno se puede utilizar restricciones de áreas, desechando áreas que sean más pequeñas de un cierto umbral.

Por medio de las funciones *cv2.convexhull* y *cv2.convexityDefects* de la librería OpenCV, se puede diferenciar el gesto de la mano. La función *cv2.convexhull* envuelve el contorno en un área mínimo. Y la función *cv2.convexityDefects* encuentra los puntos convexos en el contorno. Con estas dos funciones se puede diferenciar cuando los dedos están extendidos y cuando no.

En la Figura 3.15 (a) se muestra el uso de estas funciones para la Figura 3.13 (a), pero las Figuras 3.15 (b) y (c) se tratan del contorno de la Figura 3.11 (a). Al ser un fondo no homogéneo y aún aplicando la máscara obtenida por la segmentación del color, no se consigue un contorno perfecto con el que poder trabajar con estas funciones.

El uso del algoritmo Canny en la imagen binaria de la máscara, queda condicionada por la salida del descriptor de color. Ya que sin la utilización de este se obtienen contornos que no se ajustan a los bordes de la mano. Pero aún obviando los contornos que no interesan en la imagen, el contorno del borde de la mano no es perfecto. Este se corta, abarca otros objetos o desecha partes de la mano. Por tanto las funciones anteriores no resultan eficientes para

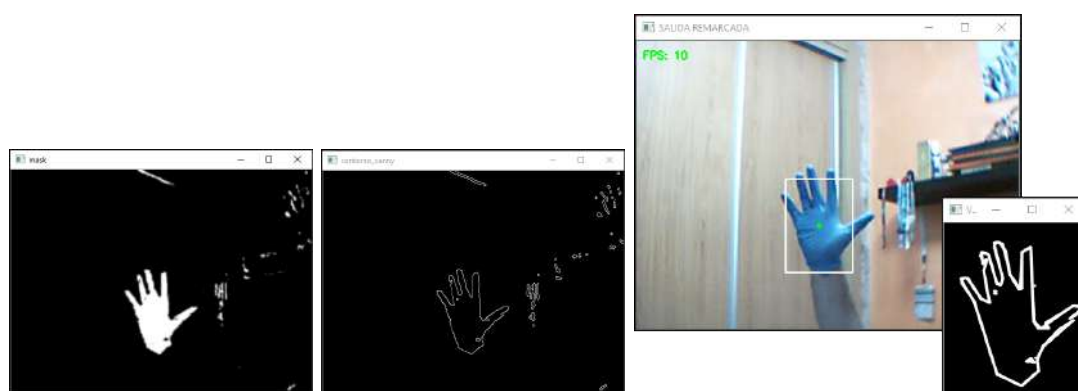


(a) Contorno de la mano en condiciones favorables (b) Contorno de la mano en condiciones normales (c) Contorno de la mano en condiciones normales

Figura 3.15: Contornos con casco convexo y defectos de convexidad

clasificar el gesto.

Con la técnica CAMshift se reduce la zona de búsqueda de contorno obteniendo en una ventana el contorno de la mano. Este ahora se puede comparar para clasificar su gesto. En la Figura 3.16 se muestra el proceso de la generación de ventanas con la forma de la mano.



(a) Máscara de entrada (b) Contorno de la máscara (c) Obtención de la ventana con la forma

Figura 3.16: Uso de la CAMshift en el software

3.3.2.2. Comparación de formas

Con el contorno de la mano en una ventana, se puede comparar con modelos o patrones previamente definidos. Según los resultados que se obtengan se clasifican en los diferentes gestos. Esta comparación se realiza con la función `cv2.matchShape`. Esta permite comparar dos formas o contornos binarios y devuelve un número que muestra la similitud entre los dos. Obteniendo un resultado que converge a 0 cuanto más parecidos son.

Este algoritmo primero calcula los momentos de la forma de la imagen. El momento depende de la intensidad de los píxeles y su ubicación en la imagen. Para obtener un mejor

resultado se hace invariante a la intensidad dando una imagen binaria, y a traslación restado el centroide. Aún así sigue siendo afectado por escalados, rotaciones, simetrías o volteos, para ello se utiliza hasta el momento de séptimo orden. La magnitud de los momentos primeros con los séptimos no son comparables debido a su magnitud. Para ello se realiza una transformación de registro dada en la ecuación 3.1.

$$H_i^7 = -sign(h_i)log|h_i| \quad (3.1)$$

Tras aplicar esta transformación se obtienen los valores de la tabla de la Figura 3.17, donde ahora si son comparables los diferentes momentos. Los valores representados por las formas S0, S1, S2, S3 y S4 son muy cercanos mientras que K0 es algo más dispar. Se observa que S3 y S4 son el reflejo vertical y su resultado cambia de signo.







id	Image	H[0]	H[1]	H[2]	H[3]	H[4]	H[5]	H[6]
K0		2.78871	6.50638	9.44249	9.84018	-19.593	-13.1205	19.6797
S0		2.67431	5.77446	9.90311	11.0016	-21.4722	-14.1102	22.0012
S1		2.67431	5.77446	9.90311	11.0016	-21.4722	-14.1102	22.0012
S2		2.65884	5.7358	9.66822	10.7427	-20.9914	-13.8694	21.3202
S3		2.66083	5.745	9.80616	10.8859	-21.2468	-13.9653	21.8214
S4		2.66083	5.745	9.80616	10.8859	-21.2468	-13.9653	-21.8214

Figura 3.17: Tabla de momentos (3)

En las diferentes imágenes de la Figura 3.18, se puede ver que resultados ofrece la función *cv2.matchShape* para distintas formas, comprobando si resultan invariantes o no. En la Figura 3.18 (a) que son dos formas idénticas se ha obtenido el resultado más bajo. Comprobando que al comparar con esta función dos contornos iguales se obtiene un resultado de 0. Para la Figura 3.18 (b) una de las imágenes ha sufrido una pequeña traslación y un corte en la forma dando como resultado un valor de $0,76 \times 10^{-3}$, incluso bastante más bajo sin ese corte. Con este resultado se puede decir que es invariante a traslaciones. En la Figura 3.18 (c), se muestra cómo afecta a esta función una rotación de 45 grados, dando un resultado de $7,6 \times 10^{-3}$. La Figura 3.18 (d) muestra un resultado de $1,1 \times 10^{-3}$ al ser una forma la simétrica de la otra. Y

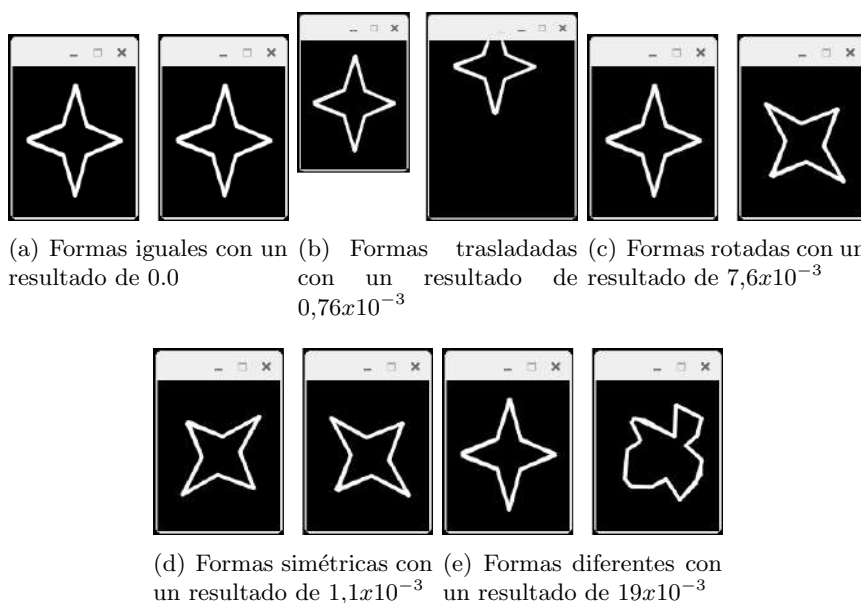


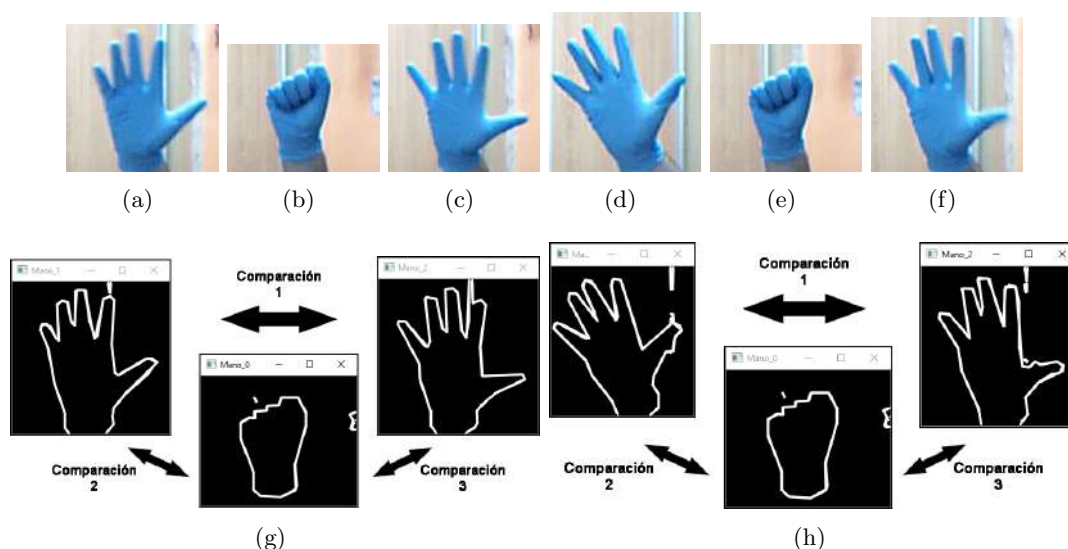
Figura 3.18: Comparaciones de formas

por ultimo la Figura 3.18 (e) que muestra dos formas diferentes con un resultado de $19x10^{-3}$.

Estas comprobaciones que se han utilizado son una pequeña prueba para ver como va a afectar cuando se comparen las formas de la mano. Por ejemplo, si en la ventana que recoge el contorno de la mano de la Figura 3.16 (c) recorta la punta de los dedos, la muñeca está algo más torcida o se ha optado por elegir la mano derecha en vez de la izquierda.

Estas comparaciones están lejos de la realidad, obteniendo formas más irregulares cuando se trata de obtener el borde de la mano. Para ver mejor el comportamiento real que se obtiene, se han comparado contornos de la mano de las siguientes Figuras 3.19 (a), (b), (c), (d), (e) y (f). Calculando las distancias entre las formas se han obtenido, según las Figuras 3.19 (g) y (h), los siguientes resultados:

- De la Figura 3.19 (g)
 - Comparación 1: $13x10^{-3}$
 - Comparación 2: $90x10^{-3}$
 - Comparación 3: $102x10^{-3}$
- De la Figura 3.19 (h)
 - Comparación 1: $4,2x10^{-3}$
 - Comparación 2: $108x10^{-3}$
 - Comparación 3: $104x10^{-3}$

Figura 3.19: Comparaciones con `cv2.matchShape`

Viendo que los valores de las pruebas dan resultados gratificantes, se ha de probar en el vídeo. Con la técnica de CAMshift se obtiene la ventana que contiene la mano y tras generar el contorno, se comparan las formas. Según los resultados de la comparación con las diferentes formas se clasifica el gesto.

Como se ve en la Figura 3.20 todos los fotogramas del vídeo se comparan con unos patrones predefinidos, pero se puede ver como del primer fotograma al último la diferencia con la forma del patrón de la mano abierta se distancia. Para corregir ese problema se ha optado por utilizar más patrones por gesto, comparando varios patrones con el candidato generado y obteniendo todos los resultados de las comparaciones.

Estos patrones pueden obtenerse mediante las mismas técnicas. Ajustando los rangos de valores al color del guante se obtiene una máscara. La técnica CAMshift recoge esa máscara en una ventana. Tras aplicar unos márgenes y buscar los contornos se obtiene la forma de la mano. Guardando esta forma en una imagen se puede comparar posteriormente en los fotogramas del vídeo.

En el diagrama de la Figura 3.21 se muestra como trabaja una función implementada para crear patrones. Al iniciar esta función se da el nombre del patrón, donde se guardarán los gestos. El programa indica los archivos ya existentes (patrones ya creados) e inicia la cámara. Se muestra la imagen de entrada y se le aplica la máscara de rango de valores. La función CAMshift obtiene la ventana con la mano, se muestra y se buscan los contornos. Al presionar la barra espaciadora se guarda una imagen de la ventana y crea la forma del contorno para guardarla también. Estas dos imágenes se guardan en el directorio del patrón. En la Figura 3.22 se muestra los directorios donde se guardan los candidatos y sus formas. Este directorio corresponde con el nombre que se le asigna al patrón.

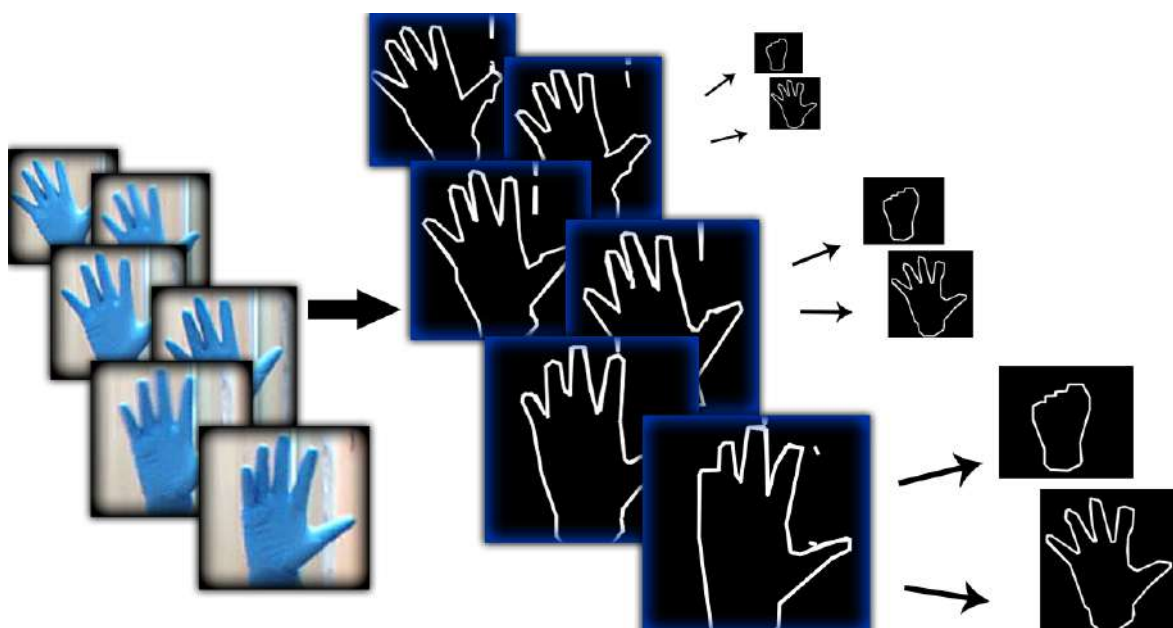


Figura 3.20: Comparación de patrones en el vídeo

Ahora se puede comparar a la forma de la mano con diferentes patrones, obteniendo un resultado más estable que no depende de un solo patrón. A mayor número de patrones mayor será la precisión de comparación. Se obtendrán tantas comparaciones como patrones se tengan de cada gesto. Es importante que exista el mismo número de patrones por gesto. Las comparaciones de cada gesto se agruparán en un vector o lista como se muestra en la Figura 3.23. Una vez se obtiene todas las comparaciones se elige la forma en que se comparan para la elección de la clasificación:

- Por distancia más próxima: el valor más bajo de la comparación clasifica esa forma como el gesto de su patrón. Este método es muy sensible y depende la clasificación a

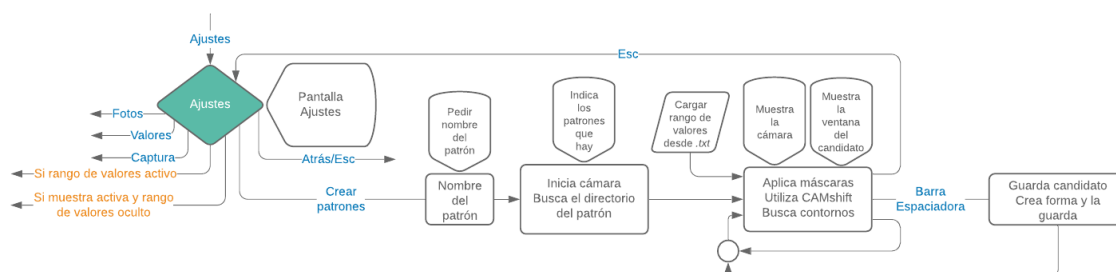


Figura 3.21: Diagrama de la función crear patrones

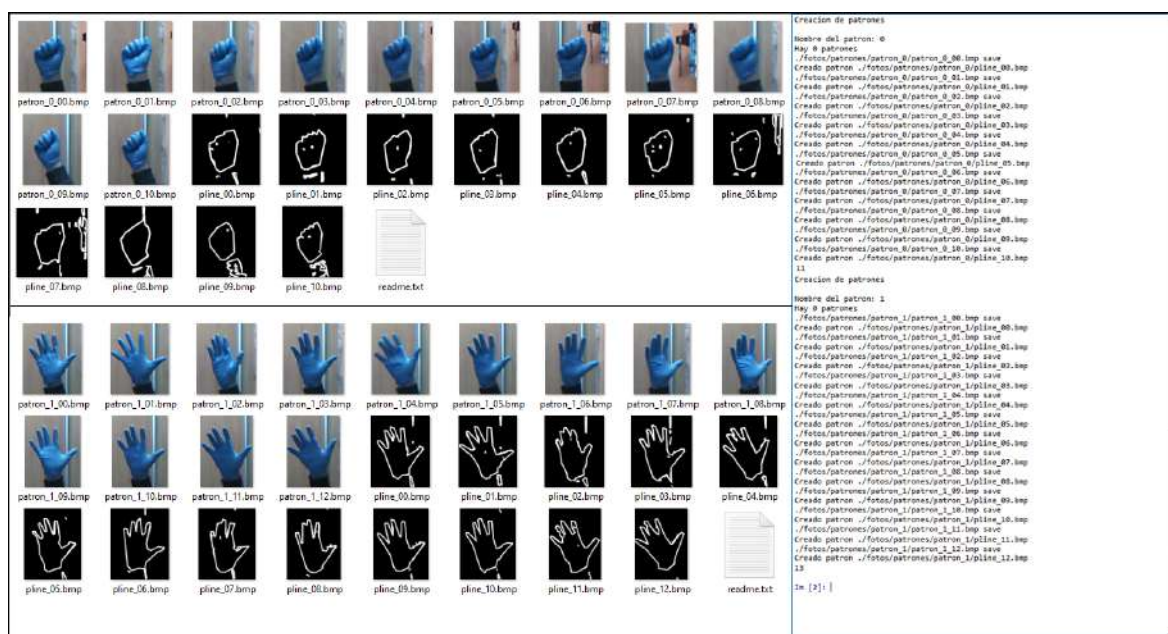


Figura 3.22: Directorio de los patrones y su creación desde la función crear patrones

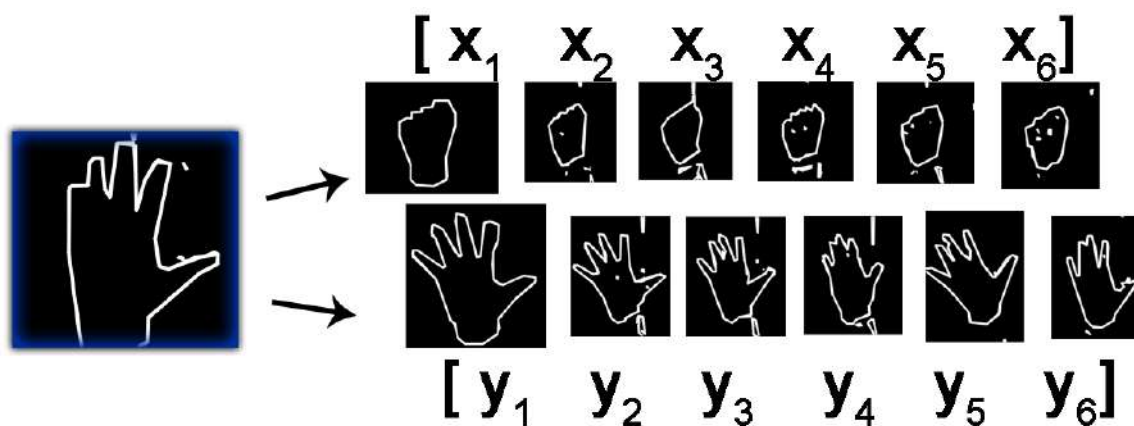


Figura 3.23: Comparación del candidato con múltiples patrones de cada gesto

una sola comparación.

- Por media: se realiza una media de las listas y se clasifica por el resultado de las medias. Este método usa todas las comparaciones para su clasificación.
- Por pesos: se utilizan unos pesos en las comparaciones y se realiza la media. Estos pesos son mayores a las comparaciones que más se asemejan y menores a las más distantes.

El método más eficiente es por pesos, este da menor importancia a resultados dispares entre los patrones del mismo gesto y más a los que se asemejan.

Por último se hará uso de redes neuronales convolucionales para clasificar la imagen según el gesto de la mano.

3.3.3. Basado en redes neuronales convolucionales

Como se comentó al principio del capítulo se utilizan técnicas de aprendizaje automático para la clasificación de los gestos. Anteriormente se vio como utilizando diferentes descriptores se conseguía resaltar el color del guante, su forma o textura. Con el uso de las redes neuronales no es necesario utilizar descriptores. Ya que la propia red neuronal se encarga de aprender cual serán los descriptores que necesita para detectar el objeto. La red neuronal procesa un gran conjunto de imágenes de las cuales ya están clasificadas. Se entrena con esas imágenes y determina cuales son las características que las diferencia. Este proceso de entrenamiento ya se habló de él en la Sección 2.4, y se denomina Backpropagation.

Como ya se comentó existen diferentes herramientas a la hora de crear una red neuronal. La red está creada a nivel de composición de capas utilizando las librerías Keras de Tensorflow. Para la creación de la red neuronal se han seguido cuatro pasos.

3.3.3.1. Generación de la base de datos

Una buena base de datos es crucial a la hora de entrenar un modelo de una red neuronal. Esta permite abastecer a la red para su entrenamiento y debe ser lo más amplia posible, abarcar el mayor rango de posibilidades y tener las características que la estructura pida. Una base de datos errónea o no lo suficiente diversa tendrá como resultado un modelo poco eficiente.

Para la creación de la base de datos del modelo definitivo se han utilizado más de 6.000 imágenes, de las cuales se han repartido en los 3 diferentes gestos y las imágenes de validación. En la Tabla 3.4 se indican los valores exactos para cada una de los gestos. El 10 % de las imágenes de la base de datos se han utilizado para la validación posterior al entrenamiento.

Este 90 % de las imágenes han sido utilizadas por la red neuronal para su entrenamiento, del que se habla más adelante. La mayor parte de las imágenes utilizadas en la base de datos han sido capturadas por la WebCam, entorno al 70 %, un 25 % por la cámara de la Raspberry y un 5 % por un teléfono móvil.

	Imágenes	Entrenamiento (90 %)	Validación (10 %)
Mano abierta	1.936	1.769	167
Mano cerrada	2.027	1.794	233
Perfil de la mano	2.059	1.865	194

Tabla 3.4: Imágenes de la base de datos

En la Figura 3.24 se muestra como deben de estar organizados los directorios y las imágenes dentro de la base de datos. Deben existir dos directorios dentro de la base de datos, uno para realizar el entrenamiento y otro para la validación. Dentro de cada uno se crean otros directorios con los diferentes tipos de clasificación, en este caso **abierta**, **cerrada** y **perfil**.

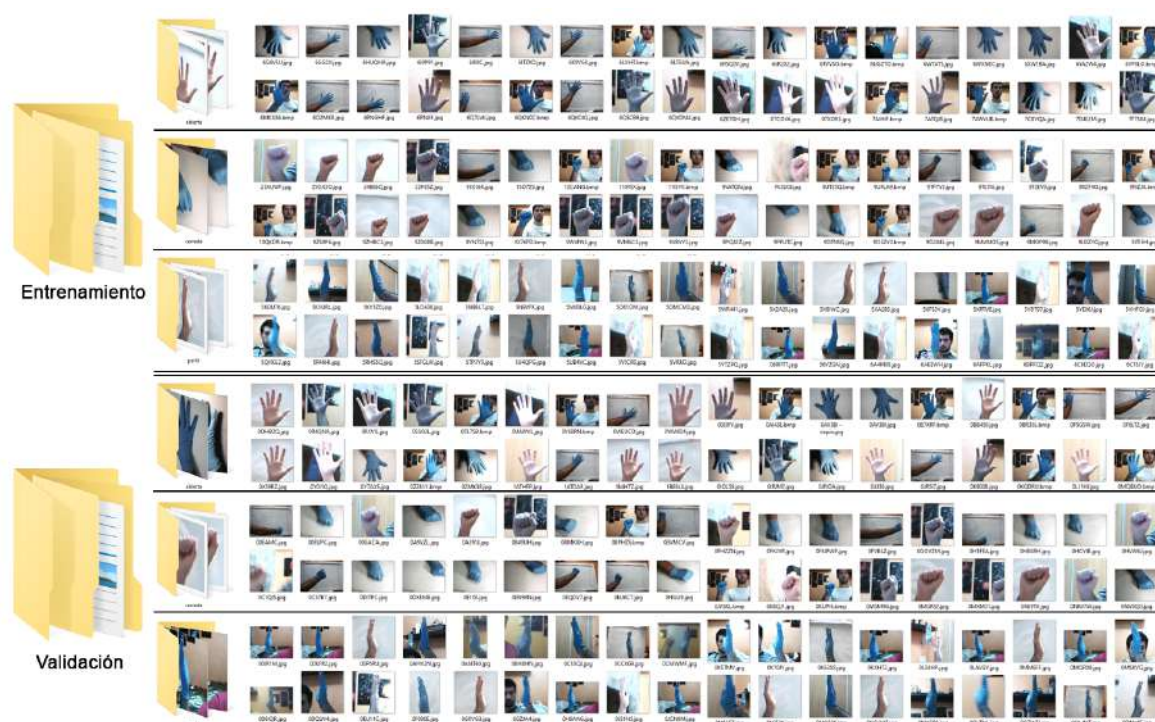


Figura 3.24: Ejemplo de estructura y composición de la base de datos

Para la creación de esta base de datos se han utilizado varios programas. Estos programas son los siguientes:

- Redimensionar imágenes: muchas de las imágenes que se han obtenido no tienen unas dimensiones aceptables. Por ejemplo, las capturadas con el teléfono móvil son de dimensiones superiores a 1000x1000 píxeles, magnitudes que no puede trabajar bien la red neuronal. Para eso se utiliza un programa en Python que redimensiona las imágenes guardando las proporciones. Si las proporciones son menores de 224x224, no es necesario.

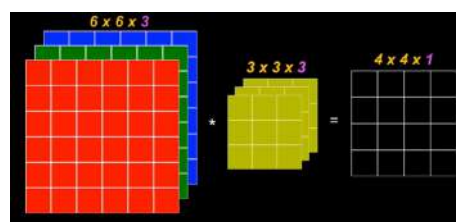
- Cambiar su extensión: para utilizar el programa de redimensionar las imágenes primero hay que convertirlas al formato *.jpg*.
- Cambiar su nombre: para conseguir una muestra aleatoria del 10 % de la base de datos, se generan nombres aleatorios de todas las imágenes de cada gesto y se selecciona el primer 10 %.

Con esto se obtiene una base de datos funcional para la red neuronal que se va a utilizar.

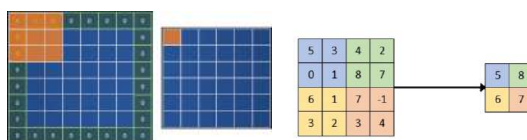
3.3.3.2. Estructura de una red neuronal convolucional

Para la creación de la red neuronal se hará uso de la librería de herramientas Keras de Tensorflow.

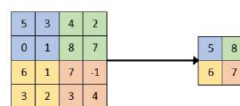
Para el tratamiento de imágenes se utilizan redes neuronales convolucionales (Sección 2.4). El proceso de convolución es la aplicación de un filtro que se desplaza píxel por píxel en la imagen, reduciendo la imagen de entrada. En la Figura 3.25 (a) se ve como una imagen con tres canales (en color) se reduce aplicando un filtro de convolución. Este proceso depende de la forma del kernel (núcleo del filtro) que se muestra en color amarillo. Cada matriz del canal realiza la convolución con cada capa del filtro y se suman para dar la matriz resultante.



(a) Proceso de convolución



(b) Padding



(c) Max pooling

Figura 3.25: Proceso de la convolución en una imagen a color

El proceso de convolución tiene asociado tres términos importantes a tener en cuenta. Uno de ellos es el *padding* que rellena los bordes de la imagen, puede ser con un valor de cero, para aplicar el filtro a los píxeles que se encuentran al borde de la imagen (Figura 3.25 (b)). Al aplicar el filtro este puede ir píxel a píxel barriendo la imagen o usar una zancada, el término *strides* marca este paso. Por último el proceso de convolución no tiene porque utilizar un solo filtro a la imagen, sino que se utilizan múltiples filtros. Estos son aprendidos por la red neuronal para la detección de bordes, esquinas, texturas, etc. Al aplicar más de un filtro, se utiliza un apilador (*steking*). Al terminar el proceso de convolución aún existen demasiadas neuronas asociadas a los píxeles, el *Subsampling* se encarga de reducir ese número de neuronas.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
conv2d (Conv2D)	(None, 224, 224, 32)	896
max_pooling2d (MaxPooling2D)	(None, 112, 112, 32)	0
conv2d_1 (Conv2D)	(None, 112, 112, 64)	8256
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 64)	0
flatten (Flatten)	(None, 200704)	0
dense (Dense)	(None, 256)	51380480
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 3)	771
Total params: 51,390,403		
Trainable params: 51,390,403		
Non-trainable params: 0		

Figura 3.26: Red neuronal convolucional básica

Esta agrupación depende del kernel y su tipo. Si se trata del *Max pooling* da como salida el píxel de mayor valor (Figura 3.25 (c)).

En la Figura 3.26 se muestra la estructura de una red neuronal convolucional con unas pocas capas que sirve para clasificar imágenes. Esta viene construida por las siguientes capas:

- Capa de entrada *input_1*: esta capa recibe la entrada de la red neuronal, en este caso los píxeles de la imagen. La capa muestra una forma de 224x224x3, siendo estas las dimensiones que tiene que tener la imagen que entre a la red neuronal, 224 píxeles de ancho, 224 píxeles de alto y tres canales de color.

```
visible = Input(shape=(ancho, alto, canales))
```

- La primera capa oculta *conv2d*, se trata de una capa convolucional de dos dimensiones (para tratar con imágenes). Se le aplica el proceso de *padding* y la dimensión del kernel es de 3x3. Como resultado al utilizar 32 filtros, se obtiene una salida de 224x224 píxeles con 32 imágenes apiladas.

```
hidden1 = Convolution2D(núm_filtros, (dim_filtro), padding="same",
input_shape=(ancho, alto, canales),
activation='relu')(visible)
```

- La segunda capa oculta *max_pooling2d*, se trata de la capa que reduce el número de neuronas, en este caso a la mitad. El tipo de agrupación utilizado ha sido el *Max pooling* que preserva el valor del píxel más alto.

```
hidden2 = MaxPooling2D(pool_size=(dim_kernel))(hidden1)
```

- La dos siguientes capas ocultas tienen la misma función, obteniendo más imágenes generadas por filtros en la de convolución y volviendo a reducir la cantidad de neuronas con el *Max pooling*.

```
hidden3 = Convolution2D(núm_filtros , (dim_filtro), padding ="same")(hidden2)
hidden4 = MaxPooling2D(pool_size=(dim_kernel))(hidden3)
```

- La capa *flatten*, se utiliza para convertir las diferentes dimensiones en una sola. Generando un capa de neuronas tradicional, compuesta por 200.704 neuronas.

```
hidden5 = Flatten()(hidden4)
```

- La capa oculta número seis se trata de una capa densa, esto significa que todas sus neuronas están conectadas a las capas colindantes.

```
hidden6 = Dense(núm_neuronas, activation='relu')(hidden5)
```

- La capa oculta número siete *dropout* es una capa que oculta información aleatoria a su siguiente capa tras cada iteración del entrenamiento. Esto refuerza a la red a predecir resultados sin tener toda la información.

```
hidden7 = Dropout(0.5)(hidden6)
```

- Por último, la capa de salida. Otra capa densa con el número de neuronas igual al número de salidas que se deban obtener. En este caso la clasificación se hará de tres gestos, por tanto su salida es de tres neuronas. El utilizar una función *softmax* da a la salida resultados entre 0 y 1.

```
output = Dense(núm_gestos , activation='softmax')(hidden7)
```

Como se comentó en la Sección 2.4, existen modelos con estructuras mucho más complejas, más eficientes para el procesamiento de imágenes y entrenadas con bases de datos muy grandes. Estos modelos han requerido un tiempo de entrenamiento elevado para obtener sus parámetros. Estas estructuras de los modelos pre-entrenados se pueden utilizar para entrenar una red, pero al ser tantos los parámetros a calcular se llevaría demasiado tiempo. Para ello se utiliza la técnica de *Transfer learning*.

Esta técnica permite utilizar estas estructuras y aplicarle pequeñas modificaciones en la capa de salida para obtener una nueva clasificación. Las capas ocultas utilizan los parámetros obtenidos en su entrenamiento previo, y al volver a entrenarlos se obtienen el resto de parámetros para la nueva clasificación.

En la Tabla A.1 del anexo A se muestra la estructura transferida que se ha utilizado para una red neuronal más eficaz que la vista anteriormente. Esta estructura se ha transferido del modelo Mobilenet_v2 y utiliza algo más de 2 millones de parámetros, a diferencia del modelo básico anterior que utilizaba más de 51 millones.

La gran diferencia de parámetros se debe a la utilización de capas densas. La capa densa en el modelo básico genera la mayor parte de esos parámetros, sin embargo el modelo transferido solo presenta una capa densa al final de la estructura. Se suele utilizar capas densas en la capa de salida para obtener el resultado con la predicción.

Sin embargo no es necesario volver a buscar esos parámetros, al final de la tabla A.1 se muestran los parámetros entrenables, es decir, los parámetros que necesitan ser ajustados. Estos parámetros son únicamente de la última capa que fue cambiada del modelo original para modificar la clasificación.

Ahora la red neuronal ya tiene una estructura creada. El siguiente paso será el entrenamiento de esta red para el cálculo de los parámetros y generación del modelo.

3.3.3.3. Entrenamiento de la red neuronal

El último paso para generar el modelo es el entrenamiento de la red neuronal. Para realizar el entrenamiento o ajuste de los parámetros se realizan tres procesos. Estos son: el procesamiento de las imágenes de la base de datos, la compilación del modelo y el ajuste de los parámetros.

En el primero proceso se abastece a la red neuronal de las imágenes de la base de datos. Este proceso diferencia el directorio de entrenamiento y el de validación, y utiliza las imágenes para generar otras nuevas. Estas nuevas imágenes que son creadas sirven para aumentar el ratio de posibilidades de encontrar los patrones y conseguir clasificar el gesto tras generar el modelo. Este proceso permite entrenar al modelo con imágenes de la base de datos rotadas, con diferentes cantidades de brillo, escaladas, sesgadas, volteadas, etc.

```
ImageDataGenerator( samplewise_center = True, rotation_range = 180,  
brightness_range = (0.2, 0.8), rescale=1./255,  
shear_range=0.9, zoom_range=0.2, horizontal_flip=True,  
vertical_flip=True)
```

Sin embargo en las imágenes de validación no se ha realizado, puesto que no es necesario. Pero si que se deben reescalar en los dos procesos las intensidades del píxel, puesto que las redes trabajan con valores entre 0 y 1.

```
test_datagen = ImageDataGenerator(rescale=1./255)
```

En la Figura 3.27 se muestran las imágenes que se han creado a partir de este proceso y con las que la red entrena. En la esquina inferior izquierda se ve la cantidad de imágenes que se han generado, casi 336 mil imágenes para el entrenamiento y más de 121 mil para

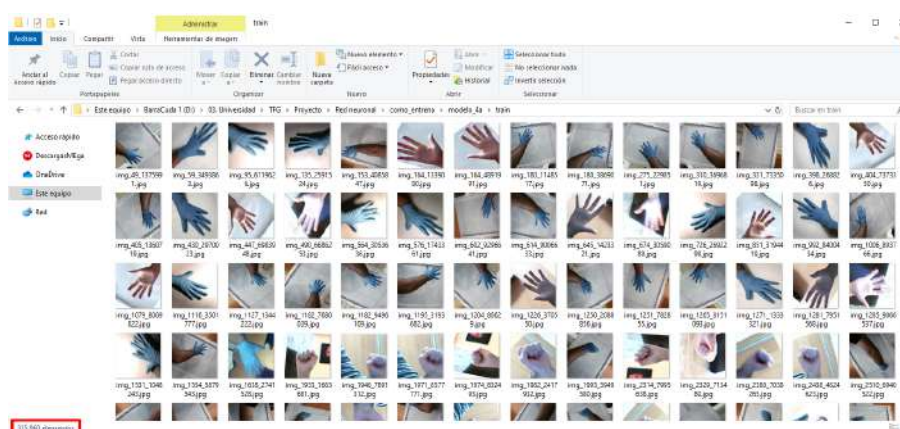


Figura 3.27: Imágenes generadas para el entrenamiento

la validación.

El proceso de compilación se encarga de generar el tipo de salida mediante la función de pérdida y optimizarla. La función de coste o pérdida es el valor del error a la salida de la red.

Existen diversas formas de optimizar el entrenamiento. Una de ellas es el optimizador *Adam*. Este es un método estocástico que intenta reducir la función de coste por el método de descenso del gradiente, siendo este la búsqueda de mínimos globales. Un método estocástico es aquel que utiliza variables aleatorias para ser menos sensible a errores o en este caso optar a una variable que le permita escapar de mínimos locales.

El ratio de aprendizaje es la variable que controla el descenso del gradiente, permitiendo acercarse al mínimo mediante iteraciones. En la Figura 3.28 se muestran las iteraciones que realiza el descenso del gradiente hasta localizar un mínimo. En este caso es global, pero normalmente no tiene por que serlo. El ratio de aprendizaje (*learning rate*) es el salto que realiza entre las iteraciones y debe de tener un valor lo suficientemente alto que evite mínimos locales y haga el proceso rápido, y no demasiado elevado para evitar divergencias.

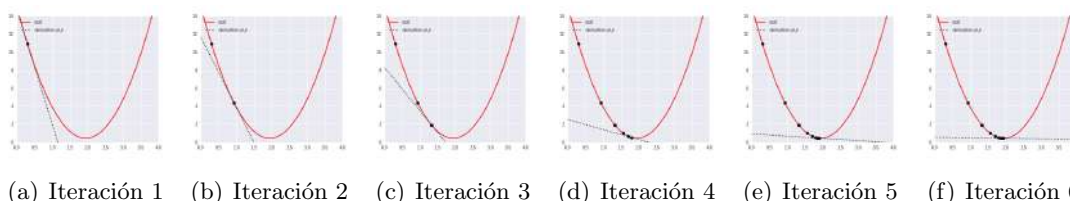


Figura 3.28: Descenso del gradiente con un solo mínimo

La función de pérdida se encarga de ofrecer la salida de la red neuronal de una forma u otra. Por ejemplo, con el error cuadrático medio de una regresión lineal o en este caso utilizando *categorical_crossentropy*. Devuelve una salida al igual que la función de activación

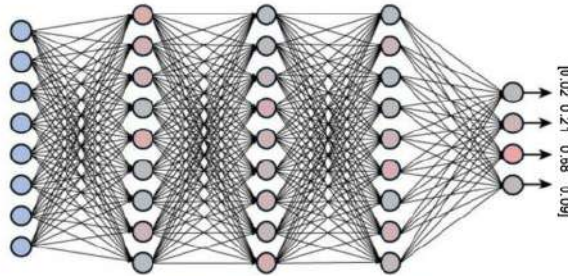


Figura 3.29: Vector de salida en la red neuronal

softmax. Obteniendo a la salida la probabilidad de seguridad en la predicción (Figura 3.29).

Finalmente el ajuste de parámetros es el que se encarga de modificar los pesos y valores de sesgo. Estos ajustes se hacen en cada época del entrenamiento. Estas épocas vienen dada por un cierto número de pasos (*steps_per_epoch*), que son las iteraciones que debe realizar el proceso para abarcar todo el número de imágenes de la base de datos. Estas imágenes son procesadas por lotes, llamados *batch_size* para el entrenamiento, y *validation_steps* para la validación. Tras terminar cada época los parámetros se ajustan gracias al algoritmo de Backpropagation.

El tiempo de entrenamiento del modelo básico fue entorno a los 4 minutos por época y el modelo pre-entrenado unos 10 minutos por época. Por lo que un mayor número de capas de la estructura de la red requiere más tiempo de procesamiento que uno que tiene menos capas, aún teniendo más parámetros que ajustar.

En la Figura 3.30 se muestran las 3 primeras y 3 últimas épocas de entrenamiento del modelo. Muestran el tiempo dedicado a cada época, los valores de la función de pérdida (loss) y acierto (acc).

```
Epoch 1/50
170/170 [=====] - 593s 3s/step - loss: 0.7329 - acc: 0.6907 - val_loss: 0.4488 - val_acc: 0.8276
Epoch 2/50
170/170 [=====] - 583s 3s/step - loss: 0.4146 - acc: 0.8633 - val_loss: 0.3261 - val_acc: 0.8810
Epoch 3/50
170/170 [=====] - 572s 3s/step - loss: 0.3263 - acc: 0.8950 - val_loss: 0.3167 - val_acc: 0.8849
```

(a) Primeras épocas del entrenamiento

```
Epoch 48/50
170/170 [=====] - 559s 3s/step - loss: 0.1378 - acc: 0.9480 - val_loss: 0.2364 - val_acc: 0.9135
Epoch 49/50
170/170 [=====] - 561s 3s/step - loss: 0.1255 - acc: 0.9543 - val_loss: 0.3278 - val_acc: 0.8976
Epoch 50/50
170/170 [=====] - 559s 3s/step - loss: 0.1553 - acc: 0.9414 - val_loss: 0.2842 - val_acc: 0.9122
```

(b) Últimas épocas del entrenamiento

Figura 3.30: Épocas de entrenamiento del modelo Mobilenet modificado

3.3.3.4. Resultados de validación del modelo

Tras el entrenamiento de la red neuronal se obtiene un modelo con los parámetros fijados. Durante el entrenamiento del modelo, el 10 % de las imágenes de la base de datos se utiliza para la validación. Esta proporciona gráficos para ver el comportamiento que ha seguido en el entrenamiento y como se va a comportar.

En la Figura 3.31 se muestran gráficas con la evolución de los valores de pérdida y acierto que ha tenido el modelo. Estas gráficas corresponden al modelo pre-entrenado de Mobilenet_v2 para la clasificación de gestos utilizado en el software.

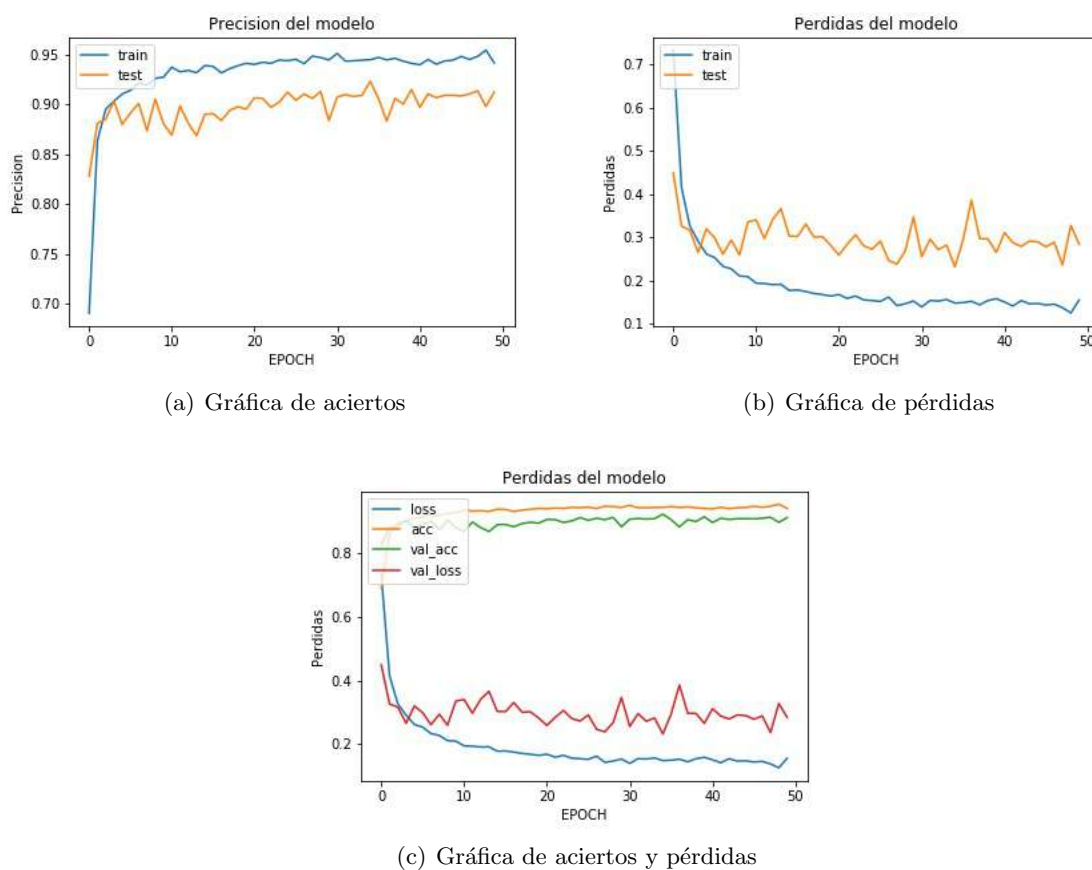


Figura 3.31: Gráficas de evaluación

Los valores de acierto en el entrenamiento están entorno al 95 % y en su validación sobre el 90 %. Para aumentar estos valores sería necesario aumentar el número de imágenes de la base de datos y su entrenamiento. En la gráfica de pérdidas se puede ver como se ha ido reduciendo según transcurrían las épocas.

En modelos anteriores a este se han presentado problemas como *overfitting* o *underfitting*.

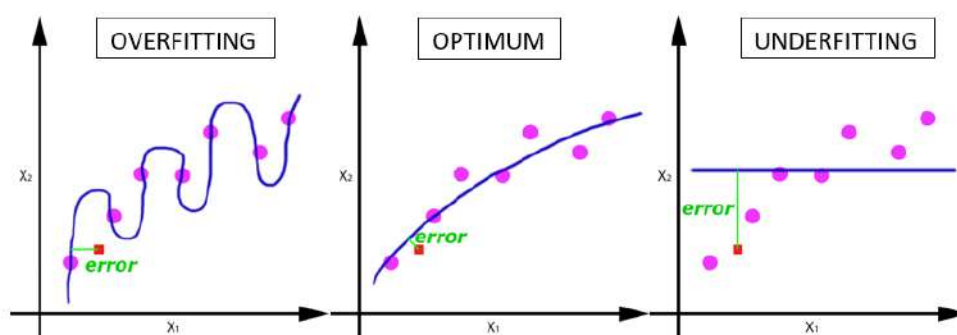


Figura 3.32: Errores de entrenamiento

Estos términos significan un tipo de error que pueden presentar los modelos (Figura 3.32).

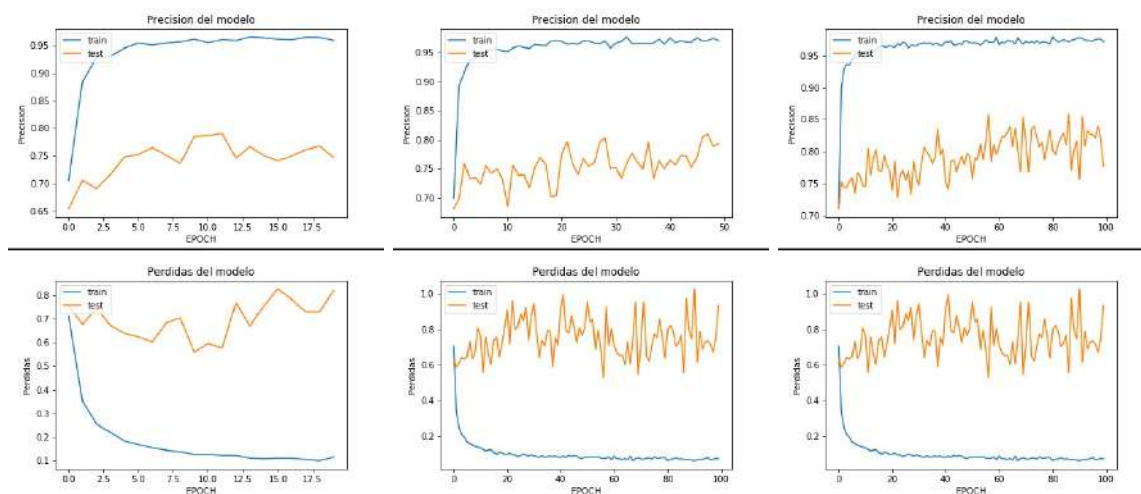
El *overfitting* o sobre ajuste, indica que el modelo no sabe generalizar lo suficiente. Eso provoca que identifique de forma correcta las imágenes de entrenamiento, pero nuevos ejemplos que no tenga entrenados no consiga clasificarlos bien. Un modelo con sobre ajuste se muestra en la Figura 3.33 (d).

El *underfitting* sería un efecto contrario al anterior, generalizando demasiado en ejemplos y clasificándolos de forma incorrecta. Este caso se corresponde por bases de datos poco elaboradas o poco tiempo de entrenamiento, mientras que el *overfitting* es por tiempos de entrenamientos largos para bases de datos pequeñas.

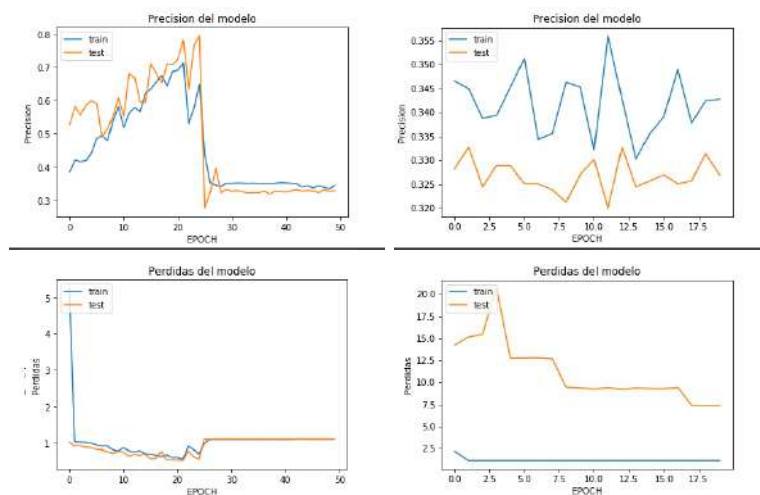
Una base de datos con amplia gama de imágenes presenta resultados mucho mejores. En la Figura 3.33 (a), (b) y (c), se muestran las gráficas de un modelo anterior al definitivo. Este modelo presenta entorno al 75-80 % de acierto, según el número de épocas de entrenamiento. Aún aumentando el número de épocas este modelo no es capaz de disminuir su pérdida ni aumentar su acierto. Para ello se aumentó el número de imágenes de la base de datos (unas 1.500 imágenes más). Estas nuevas imágenes incorporadas a la base de datos se realizaron sin el guante azul.

La razón de incorporar imágenes sin el guante viene por el hecho de la forma que aprende la red neuronal. Esta busca patrones en las imágenes y todas ellas presentan un guante azul. ¿Sería capaz de identificar una guante rojo o una mano descubierta en un nuevo ejemplo? Posiblemente tendría dificultades y su predicción no sería muy acertada, porque no se le ha entrenado para ello. Al ingresar estas nuevas imágenes la red no identifica el patrón como un “guante azul” sino que buscará su forma.

Otro de los problemas presentados durante el entrenamiento es el *learning rate* o ratio de entrenamiento. En la Figura 3.33 (e) se muestra el comportamiento de un modelo con un valor muy alto, esto dificulta la búsqueda de mínimos o incluso será imposible.



(a) Base de datos pequeña entrenada con 20 épocas (b) Base de datos pequeña entrenada con 50 épocas (c) Base de datos pequeña entrenada con 100 épocas



(d) Overfitting

(e) Learning rate elevado

Figura 3.33: Diferentes evaluaciones

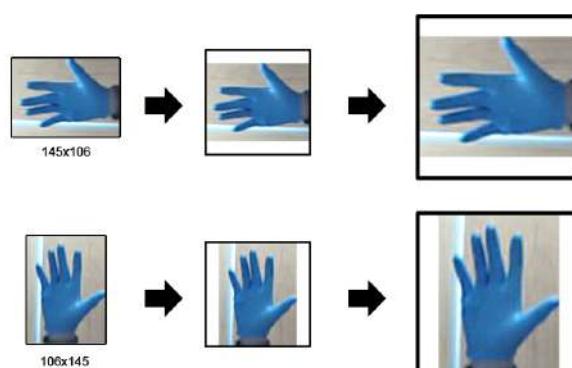


Figura 3.34: Escalado manteniendo las proporciones

3.3.3.5. Predicción del modelo

Tras los pasos anteriores se ha obtenido un modelo neuronal que se puede utilizar para la predicción de gestos. Para esta predicción se utiliza una imagen de entrada con las dimensiones 224x224 píxeles. Anteriormente se ha utilizado la función CAMshift para generar una ventana que contiene la mano. Esta puede tener cualquier dimensión y escalar la imagen supone deformarla. Para evitar esto se realiza el siguiente procedimiento (Figura 3.34):

- Se aplican unos márgenes en todos los lados de la ventana. Este margen será entorno al 15-30 % de la imagen. Esto evita cortar los dedos cuando la mano está abierta, ya que la función CAMshift recoge la nube de puntos más densa que es la palma.
- Se identifica la dimensión de la ventana, ancho y alto y se recoge el valor más alto.
- La función CAMshift devuelve el centro de la ventana. Con ese centro y la dimensión con valor más alto se obtiene una nueva ventana, con las dimensiones alto y ancho iguales.
- Finalmente la nueva ventana se escala a las dimensiones de entrada de la red neuronal (224x224) sin deformarse.

Ahora ya se tiene una entrada eficiente para la predicción del modelo, que vendrá dada en forma de vector como indicaba la Figura 3.29. Por tanto, con este vector se clasifica el gesto en uno de los tres valores dados para indicar la acción oportuna.

Para obtener solo las predicciones seguras se realiza un control de calidad. Este se trata de un umbral a partir del cual se considera como predicción segura. A la salida del modelo las tres posibles predicciones obtienen un valor que al sumarlo da 1. Con esto se puede fijar un umbral donde solo las predicciones superiores al 90 % ejecuten una acción.

Para utilizar modelos en la Raspberry Pi estos no deben ser muy pesados. El límite está entorno a los 200 MB, más allá de esa cantidad es posible que de fallos de memoria y no

consiga cargar el modelo. El modelo creado con Mobilenet_v2 pesa entorno a los 10 MB, por lo que no existe problema de peso, pero existe otro problema al tratar de mantener fluido el vídeo. Cada vez que se trata de predecir la imagen para poderla clasificar, la Raspberry tarda entorno a 1.5 - 2 segundos mientras que en el ordenador tarda 0.15 - 0.2 segundos.

3.4. Software y ajustes del sistema

Una vez visto los métodos utilizados para la detección de la mano, el proceso se puede resumir en tres pasos (Figura 3.35). Donde se:

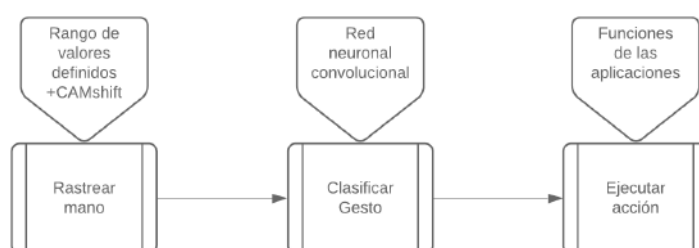


Figura 3.35: Diagrama simplificado

- Localiza la posición de la mano indicando en que lugar se encuentra y el tamaño que tiene mediante los rangos de valores definidos y la técnica CAMshift.
- Clasifica el gesto que realiza la mano con el modelo neuronal.
- Y Ejecuta la acción determinada según su posición y gesto, manipular imágenes o juegos.

Todo este proceso queda condicionado a los rangos de valores que se definen para la localización del guante. Para definir estos valores y realizar el proceso se ha creado un software, donde por medio de menús se puede calibrar estos rangos y ver como se realiza el proceso. Para ver el diseño realizado se van a utilizar diagramas de flujo, para entender mejor estos, se verán los símbolos utilizados y sus explicaciones en la Tabla 3.5.

3.4.1. Diseño del software

El software diseñado presenta dos submenús, uno de ellos para realizar ajustes y otro para iniciar el proceso de detección. Su estructura se muestra en la Figura 3.36 donde se diferencian los dos submenús, uno para realizar ajustes a la izquierda y otro para su inicio a la derecha. Dentro de este último se diferencia un gran bloque que engloba las aplicaciones. Estas son capaces de manipular imágenes y juegos mediante las acciones que se ejecutan.

Al iniciar el software se cargan todas las librerías y módulos necesarios, como se ve en el diagrama general. El módulo *conf.py* inicializa valores y variables globales que más tarde se













				
Indica el inicio del programa o diagrama.	Indica el fin del programa o salida.	Realiza un proceso.	Exportan o importan datos.	Agrupación de procesos.
				
Menú principal con opciones de selección.	Menú secundario con opciones de selección.	Conector que agrupa las líneas de flujo.	Condiciona una dirección al flujo.	Configurable por una interfaz.
				
Línea de flujo por la que se le debe de dar una acción.			Línea de flujo por la que se le debe de dar una condición.	

Tabla 3.5: Biblioteca de Símbolos del diagrama de flujo

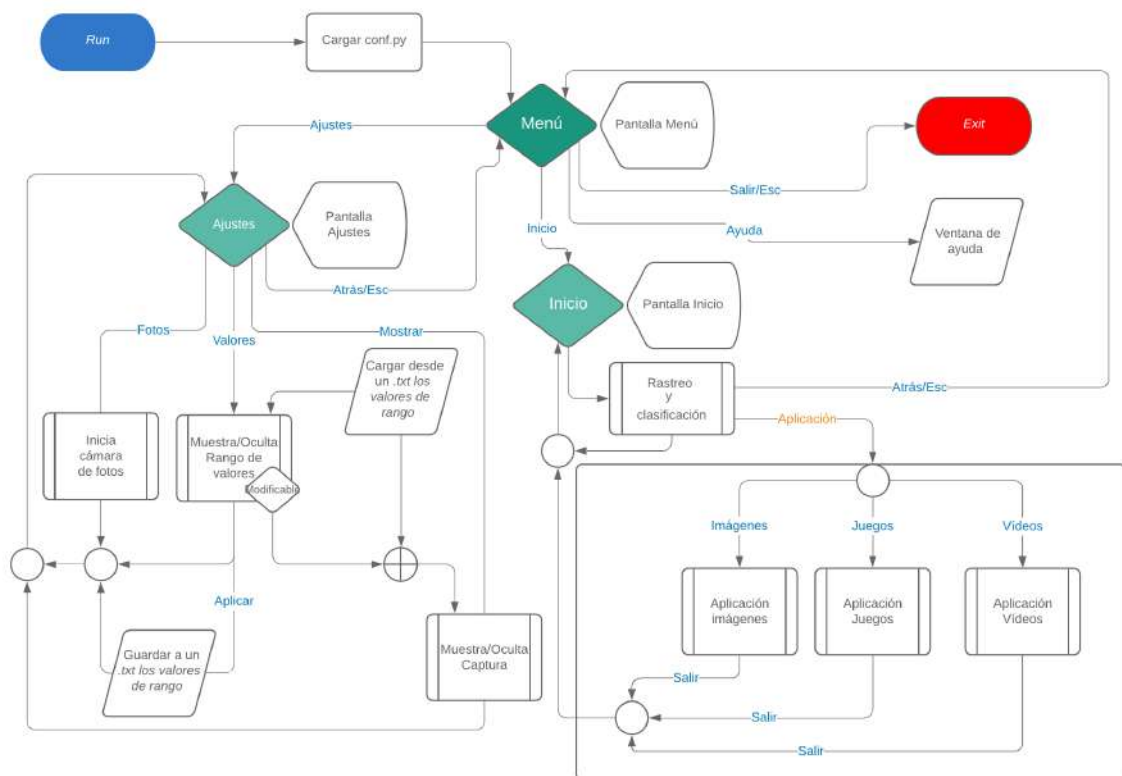


Figura 3.36: Diagrama general



Figura 3.37: Menús

utilizan en el sistema, carga las imágenes y modelos. Este módulo permite también cambiar el tipo de cámara con la que se quiere obtener la entrada, por USB la WebCam o la cámara de la Raspberry Pi.

Tras cargar e iniciar los valores de las variables se muestra el menú (Figura 3.37 (a)), que permite las opciones de iniciar, realizar ajustes o salir. Este menú es interactivo y se puede acceder a sus opciones con atajos de teclado, *i* para inicio, *h* para ayuda, *c* para ajustes y *Esc* para salir. También se puede acceder a ellos haciendo clic en las letras del título.

La opción de ajustes se realiza para obtener el mejor rendimiento a la hora de localizar la posición de la mano (Figura 3.37 (c)). En este submenú se podrán ajustar los rangos de valores con los que se va a trabajar y ver cómo afecta a la imagen mientras estos se ajustan (Figura 3.37 (d)).

Estos rangos de valores van a servir para la extracción de características del guante, obteniendo con ellas la información del objeto a seguir. Estos valores pueden guardarse para que se queden predefinidos desplazando la barra de valor *Guardar*. Estas barras son las tratadas en la Sección 3.3.1. El objeto a rastrear se trata de un guante azul, y según los gestos que se realicen con la mano se aplican una serie de acciones dentro del sistema.

Una vez configurado todo, está listo para iniciar y navegar por las aplicaciones que tenga el software (Figura 3.37 (b)). Esta navegación ya se va a realizar con la posición en la que se encuentre el guante, entrando y saliendo de las aplicaciones mediante gestos. En este caso el software diseñado solo tiene dos aplicaciones de las cuales se van a hablar en la Sección 4.2, donde se manipulan imágenes y juegos.

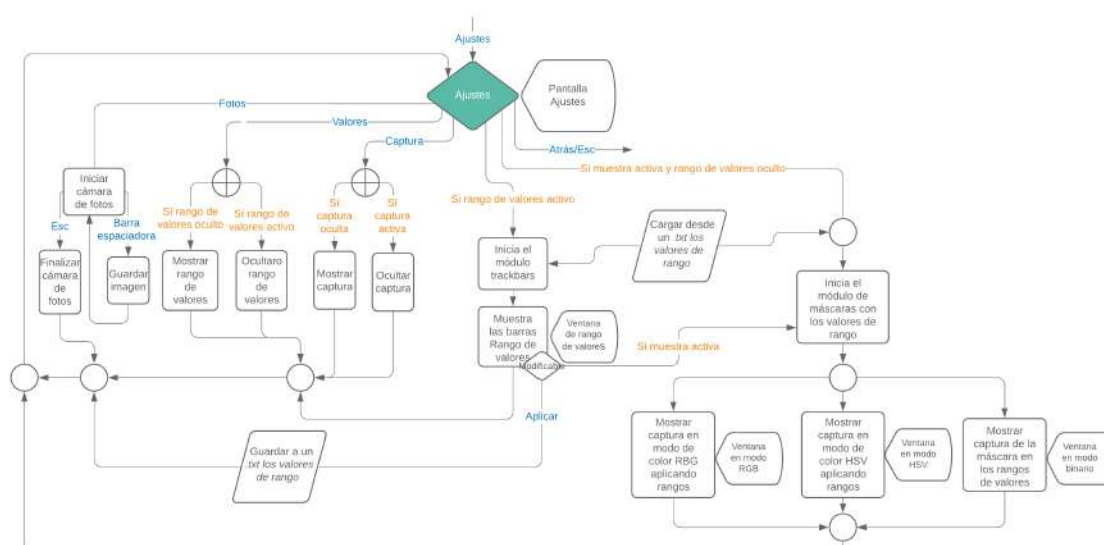


Figura 3.38: Diagrama submenú Ajustes

3.4.2. Ajustes del software

El método utilizado para la extracción de características de la imagen ha sido por la definición de rango de valores. Estos pueden ser modificados mediante barras de desplazamiento incorporadas en el submenú Ajustes. El siguiente diagrama es una expansión del diagrama general visto anteriormente (Figura 3.36), pero centrado en el submenú Ajustes (Figura 3.38).

Este diagrama indica como se ejecutan las partes del software referentes al submenú Ajustes, mostrando con letras azules cuando se trata de opciones del usuario y en naranja cuando son condiciones de variables, como se indica en la Tabla 3.5. Dentro del este submenú existen cuatro opciones:

- Opción de fotos: se inicia una pequeña aplicación presionando la tecla *f* o cliqueando sobre su título. Se muestra la captura de la cámara y existe la posibilidad de guardar imágenes presionando la barra espaciadora. Estas imágenes que captura la cámara se guardan en la carpeta `./fotos/` dentro del directorio del software. La aplicación se cierra al presionar *Esc*.
- Opción de valores: activa o desactiva la función *Rango de valores* presionando la tecla *r* o cliqueando sobre su título.
- Función *Rango de valores*: cuando está activa carga el módulo de *trackbars*. Este módulo se encarga de crear unas barras de deslizamiento para la manipulación de los máximos y mínimos del matiz, saturación y valor. También hay otras seis barras para manejar el modo de color RGB. Otra barra con valor de 0 o 1 sirve para guardar estos valores de las variables en un archivo `datos.txt` y sirve para cargar los valores siempre que se inicie la función *Rango de valores*.

- Opción captura: activa o desactiva la función *Muestra* presionando la tecla *m* o clicando sobre su título.
- Función *Muestra*: recibe los rangos de valores, bien de la función *Rango de valores* o de *datos.txt* si la función *Rango de valores* no está activa. La función *Muestra* inicia el módulo de máscaras con los valores de las variables. Este módulo aplica la técnica vista en la Sección 3.3.1, y a la salida de la función se muestra la imagen de captura en modo de color RGB, HSV y la binaria de las dos, con sus respectivos rangos de valores aplicados.
- Opción de regreso: para volver al menú principal presionando la tecla *Esc* o clicando sobre su título.

Una vez ajustados los valores más óptimos y guardados en el archivo *datos.txt* este archivo es el encargado de al iniciar el rastreo o la configuración, proporcionar los valores.

Cuando se accede al submenú Inicio se utiliza la técnica CAMshift y se predice el gesto que aparece en la ventana. Esta predicción del modelo se hace en cada fotograma que captura la cámara. Esto no es eficiente por la razón que de un fotograma a otro no puede existir gran diferencia. Además que todas las predicciones deben tener una cierta seguridad para realizar acciones en el sistema del dispositivo.

3.4.3. Depuración de los resultados de la clasificación

Para ello se realiza un control de calidad de las predicciones del modelo. El diagrama de la Figura 3.39 muestra el comportamiento del submenú Inicio, realizando un control de las predicciones y dejando la mitad de los fotogramas sin predecir.

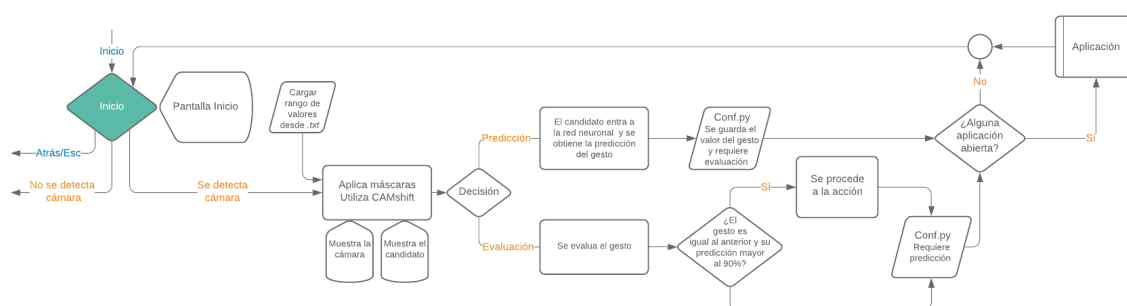


Figura 3.39: Diagrama del inicio del rastreo

Este diagrama muestra como se condiciona todo a que encuentre una cámara que capture la imagen, si no es así presentará un error. Tras obtener la imagen de entrada se aplican las máscaras generadas por los rangos de valores, vistos en la Sección 3.3.1, y la obtención de la ventana con la técnica CAMshift (Sección 3.3.1).

Utilizando la ventana como entrada del modelo se realiza la predicción del gesto. Esta predicción al tener un costo computacional considerable, no interesa hacerlo en todos los fotogramas del vídeo. Por tanto, tras predecir en un fotograma el gesto, en el siguiente se procede a evaluar la predicción. Si el resultado de la predicción es superior al 90 %, su predicción es lo suficientemente robusta para proceder a la acción. Si no lo es, no se realiza acción. Tras realizar la evaluación y la acción, si procede, se vuelve a realizar otro ciclo. Por tanto, en el inicio del rastreo se realiza la predicción en un ciclo y su evaluación en otro.

Las acciones dependen si alguna de las aplicaciones está activa. Si ninguna lo está, los gestos de mano abierta y cerrada sirven para mover el cursor y el perfil de la mano para seleccionar títulos.

Para realizar las acciones se ha utilizado la librería PyAutoGUI. Esta librería permite desplazar el cursor al centro encontrado por la función CAMshift o clicar cuando el gesto de la mano corresponda.

Capítulo 4

Resultados y aplicaciones

En este capítulo se evaluarán los dos métodos presentados en el capítulo anterior. Más concretamente, se mostrarán los resultados del método de comparación de formas y del modelo neuronal.

En la evaluación se utilizarán matrices de confusión, un sistema habitual para evaluar los métodos de visión. Con esta matriz se puede calcular el valor de exactitud del modelo, y, la precisión y sensibilidad de los diferentes gestos.

Por último se verán las aplicaciones implementadas que hacen uso del modelo neuronal, y que son capaces de manipular imágenes y juegos mediante gestos.

4.1. Evaluación

Se ha utilizado un vídeo que contiene los diferentes gestos. Cada cuatro fotogramas se ha tomado una imagen y se ha indicado el resultado de la comparación y de la predicción. Los resultados se han comparado visualmente con el gesto real de cada fotograma y se han mostrado los resultados en una matriz de confusión. Esta matriz permite conocer los valores de rendimiento del clasificador: exactitud, precisión y sensibilidad. Para poder interpretar estos términos se explicarán los siguientes conceptos: real positivo, real negativo, falso positivo y falso negativo.

El término real o falso determina si se ha clasificado correctamente o no, mientras que el término positivo o negativo depende de si se ha detectado o no. Para este clasificador, un ejemplo de clasificación correcta es la Figura 4.1 (a) donde se ha clasificado correctamente la mano sin detectar otros objetos. En la Figura 4.1 (b) existe un falso negativo al no detectar la mano y un falso positivo al detectar un objeto del fondo. En nuestro caso cuando se clasifica un gesto de forma incorrecta se obtiene una falsa detección, que genera un falso positivo para el gesto que clasifica y falso negativo para el gesto real.



(a) Real positivo y real negativo (b) Falso positivo y falso negativo

Figura 4.1: Identificadores del objeto

A continuación se detallará cómo se calcula la exactitud del modelo (Ecuación 4.1), la precisión (Ecuación 4.2) y sensibilidad (Ecuación 4.3) del clasificador de cada gesto.

- La exactitud es la proximidad del clasificador al resultado real, se obtiene utilizando la Ecuación 4.1.

$$Exactitud = \frac{Reales\ positivos + Reales\ negativos}{número\ de\ muestras} \quad (4.1)$$

- La precisión es la calidad de la respuesta positiva del clasificador, se calcula utilizando la Ecuación 4.2.

$$Precisión = \frac{Reales\ positivos}{Reales\ positivos + Falsos\ positivos} \quad (4.2)$$

- La sensibilidad es la eficiencia en la clasificación de todos los elementos (Ecuación 4.3).

$$Sensibilidad = \frac{Reales\ positivos}{Reales\ positivos + Falsos\ negativos} \quad (4.3)$$

Los resultados obtenidos para el método de comparación de formas se muestran en la Tabla 4.1. Esta representa una matriz de confusión, que es la herramienta que permite visualizar el rendimiento del clasificador. En ella se recoge la información obtenida en el vídeo representando en las filas el gesto real de la mano y en las columnas el gesto que indica el clasificador.

		Clasificación		
		Mano abierta	Mano cerrada	Perfil mano
Gesto real	Mano abierta	19	34	0
	Mano cerrada	10	32	2
	Perfil mano	17	18	6

Tabla 4.1: Resultados utilizando comparación de formas

Los resultados de la comparación han sido:

- Ha clasificado correctamente 19 veces cuando la mano está abierta y 34 como mano cerrada (falsa detección).
- Cuando la mano estaba cerrada se clasificó 10 veces como mano abierta (falsa detección) y 32 veces correctamente. Además de obtener otras dos falsas detecciones (mano de perfil).
- Cuando la mano está de perfil: se produjeron 35 falsas detecciones al confundirlos con la mano abierta (17 ocasiones) y mano cerrada (18 ocasiones). Solamente se clasificaron 6 correctamente.

Los valores de rendimiento del clasificador se muestran en la Tabla 4.2.

Exactitud		41,3 %
Precisión	Mano abierta	41,3 %
	Mano cerrada	38,1 %
	Perfil mano	75 %
Sensibilidad	Mano abierta	35,8 %
	Mano cerrada	79,5 %
	Perfil mano	14,6 %

Tabla 4.2: Valores de rendimiento del clasificador basado en comparaciones de las formas

Los resultados obtenidos por la comparación de formas han sido con 70 patrones por gesto. Es un método que no garantiza ni exactitud, ni precisión, ni tampoco sensibilidad, ya que la mayoría están por debajo del 50 %. Los únicos valores por encima del 50 % resultan ser la precisión para el perfil de la mano y la sensibilidad para la mano cerrada. Para la precisión del perfil, se debe a que detecta pocas veces el gesto de perfil, pero cuando lo hace realmente

es ese gesto. Y para la sensibilidad de la mano cerrada, se debe a que al clasificar la mayoría de los gestos como mano cerrada, tiene más acierto.

Los resultados de la predicción del modelo se han recogido en la Tabla 4.3 relativos a la evaluación del modelo neuronal.

		Clasificación		
Gesto real		Mano abierta	Mano cerrada	Perfil mano
	Mano abierta	46	0	8
	Mano cerrada	0	46	4
	Perfil mano	1	0	33

Tabla 4.3: Resultados de las predicciones del modelo neuronal

A simple vista se aprecia que sus predicciones superan las del método anterior obteniendo los siguientes resultados:

- Ha clasificado correctamente 46 veces cuando la mano está abierta y 0 como mano cerrada. Se han producido 8 falsas detecciones como mano de perfil.
- Cuando la mano estaba cerrada se clasificó 46 veces correctamente y 4 falsas detecciones clasificadas como mano de perfil.
- Cuando la mano está de perfil se clasificó 33 correctamente y una vez como mano abierta.

Los valores de rendimiento de este clasificador se muestran en la Tabla 4.4.

Exactitud		90,5 %
Precisión	Mano abierta	97,8 %
	Mano cerrada	100 %
	Perfil mano	73.3 %
Sensibilidad	Mano abierta	85.2 %
	Mano cerrada	92 %
	Perfil mano	97 %

Tabla 4.4: Valores de rendimiento del clasificador basado en las predicciones del modelo neuronal

Con esta evaluación se puede ver como los resultados obtenidos por el modelo son buenos. Además pensamos que se pueden mejorar aumentando el número de imágenes de la base de datos y su entrenamiento. Por todo ello, se utilizará el modelo de la red neuronal para la creación de aplicaciones.

En la Sección 4.2 se verán una serie de aplicaciones implementadas en el software. Estas hacen uso de imágenes que se procesan en cada ciclo, necesitando aún mayor procesamiento de datos por ciclo, lo que hace que la Raspberry Pi no las pueda procesar con fluidez.

4.2. Aplicaciones

Se han implementado una serie de aplicaciones que son controladas mediante la posición y los gestos de la mano. Estas aplicaciones son un pequeño ejemplo de la diversidad de cosas que se puede hacer. Se basan en la clasificación de los gestos según su posición para realizar determinadas acciones cómo hacer zoom sobre imágenes, rotarlas, navegar por ellas y controlar un juego. Ambas aplicaciones se manejan completamente mediante los gestos de la mano. Estos gestos son la apertura y cierre de la mano, y mostrar la mano de perfil.

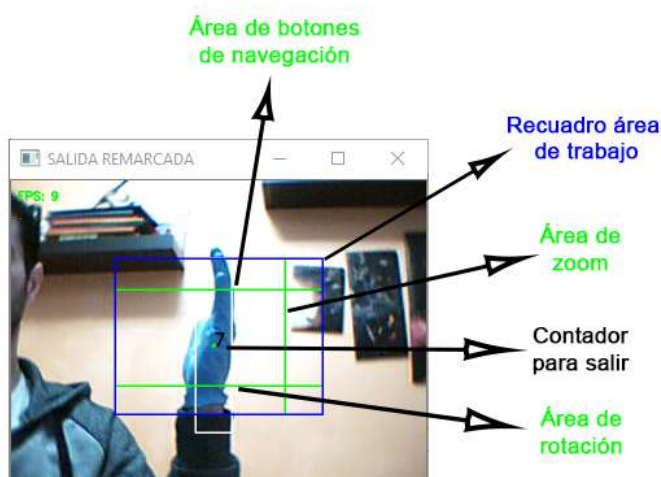


Figura 4.2: Indicadores en la imagen de captura

En la aplicación solo se han considerado las predicciones que tienen un resultado con un umbral de confianza superior al 90 %, dado un porcentaje inferior se ha comprobado que en muchas ocasiones realiza acciones erróneas.

A continuación se muestran imágenes que reflejan el comportamiento de la aplicación. En la Figura 4.2 se muestran cuales son los indicadores en la imagen de captura. En la Figura 4.3 se muestra en el menú de inicio de la aplicación, la imagen de captura en directo, la máscara de entrada a la función CAMshift y la ventana que entra al modelo para su predicción.

En las dos figuras se ve cómo la mano está abierta. El área de trabajo en verde indica que el modelo predice el gesto de mano abierta por encima del umbral. Cuando la mano se encuentra cerca del objetivo de la cámara (Figura 4.3 (a) y (b)) el centro o la posición del puntero cambia más bruscamente. Para eso se ha creado una ayuda de selección que consiste en que al acercarse el cursor a un título, su posición se fija en el centro del título (Figura 4.3 (b)).

En la Figura 4.3 (c) y (d) se muestra cómo el gesto de la mano cerrada se indica con el recuadro de color rojo y cuándo se detecta el perfil en color azul. Si la mano se encuentra abierta o cerrada en el submenú de Inicio, tan solo se mueve el cursor. El gesto de perfil se utiliza para seleccionar el título y acceder a las aplicaciones o volver al menú principal. El recuadro que cambia de color según el gesto sirve para diferenciar el área de trabajo, dentro del recuadro el cursor se mueve y selecciona por la pantalla, fuera de este, no se ejecuta

ninguna acción. Esto evita que perder el objeto del centro de la pantalla, ya sea por ruido o por un mal ajuste, haga ejecutar acciones no deseadas.



Figura 4.3: Vista de la selección de aplicaciones

4.2.1. Imágenes

Una vez se accede a la aplicación de Imágenes se cargan las imágenes existentes dentro del directorio del software. Se puede acceder a estas imágenes con los botones de navegación situados en la parte superior de la pantalla que se pulsarán cerrando la mano. Las imágenes también se pueden manipular obteniendo zoom sobre ellas o rotarlas al mover la mano cerrada por el área de trabajo. Este zoom se puede modificar su valor para obtener un zoom

mayor o menor mediante una barra de desplazamiento que se activa con el gesto de la mano (cerrar la mano y moverla en sentido vertical).



Figura 4.4: Aplicación imágenes

En la Figura 4.4 se muestran estas imágenes. Para trabajar con ellas se escalan automáticamente, manteniendo sus proporciones y se muestran. Se observa que ahora en la imagen de captura se muestran más líneas dentro del área de trabajo. Estas líneas ayudan a ver en los lugares en los que se inician las acciones. La línea horizontal de la parte superior indica el inicio de los botones de navegación. Al cerrar la mano sobre el botón de siguiente, se carga y muestra la siguiente imagen del directorio, y con el botón de anterior para volver a la imagen anterior.

4.2.1.1. Rotación de imágenes

En la Figura 4.5 se muestra cómo se realiza la acción de rotar imágenes mediante el deslizamiento de la mano cerrada por la parte inferior del área de trabajo. En la imagen de captura se muestra una línea horizontal inferior en verde para mostrar dónde se debe ejecutar el gesto. Una vez iniciado el gesto esta línea se vuelve de color rojo y las demás desaparecen. La acción de rotar imágenes puede ser tanto para un lado como para otro y se rota mientras la mano permanezca cerrada. La imagen queda rotada cuando se cambia el gesto de la mano. Se muestra un ejemplo en vídeo de esto en la dirección <https://youtu.be/au1g3bK1-EE>.



(a) Iniciando la acción de rotar

(b) Acción de rotar



(c) Imagen rotada

Figura 4.5: Rotando imágenes

4.2.1.2. Zoom en las imágenes

En la Figura 4.6 se muestra cómo se puede realizar un zoom en la imagen y modificar el valor del zoom.



Figura 4.6: Zoom en las imágenes

El valor del zoom se muestra en la esquina superior izquierda de la imagen. La línea vertical a la derecha del área de trabajo es un indicador para modificar el zoom. Esta acción es igual a la acción anterior siendo ahora en sentido vertical. El valor del zoom se obtiene con la distancia que recorra el gesto, siendo indiferente su sentido, es decir, cuanto más distancia recorra la mano cerrada en sentido vertical, mayor es el valor del zoom.

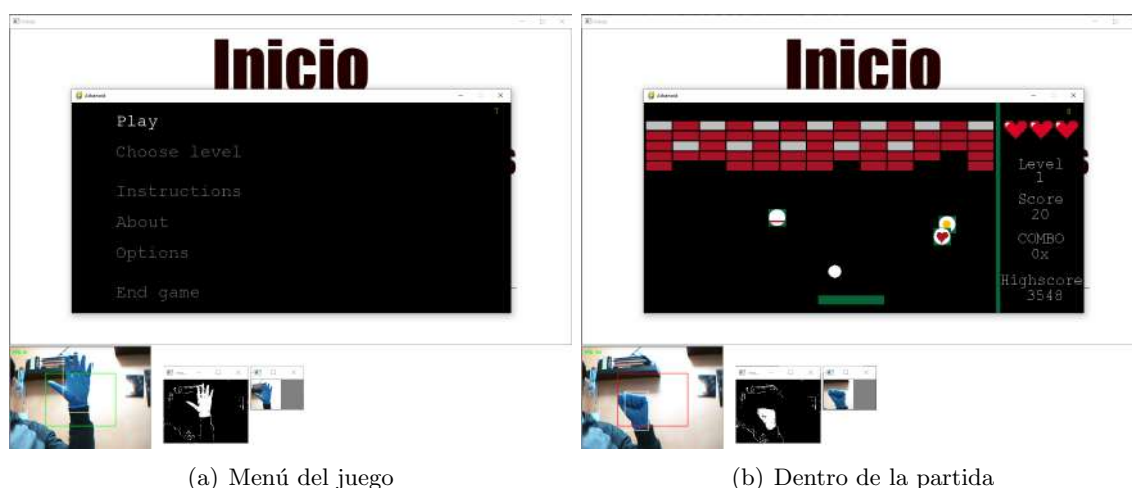
Al cerrar la mano con el cursor sobre la imagen, se abre una ventana en la que se muestra la imagen con el valor del zoom. Esta ventana se comporta como una lupa. Al mover el cursor con la mano cerrada por la imagen, se obtiene ese área de la imagen con zoom.

Para salir de la aplicación y volver al menú de inicio se utiliza el gesto del perfil de la mano. Este debe permanecer durante un breve periodo de tiempo. Cuando se realiza el gesto comienza una cuenta de 0 a 10 cerca del centro de la mano, cuando llega a 10 se vuelve al menú

de inicio. De todo esto se puede ver un vídeo en la dirección <https://youtu.be/iKx7X2ti6QA>.

4.2.2. Juego

Se ha realizado el control de un juego mediante los gestos de la mano que se ha implementado en Python utilizando la librería Pygame. Este juego ha sido descargado de un repositorio de GitHub¹ y modificado para ser controlado mediante la posición y los gestos de la mano. Se trata del clásico juego Arkanoid, donde con un bate golpea una pelota que va rompiendo ladrillos. Este juego dispone de un menú con diferentes opciones y niveles, todo ello modificado para poder acceder con gestos (Figura 4.7). Con la mano abierta se navega por las diferentes opciones y al cerrarla se acceden a ellas.



(a) Menú del juego

(b) Dentro de la partida

Figura 4.7: Aplicación juego

Una vez dentro de la partida para mover el bate basta con cerrar la mano y moverla a un lado o al otro de la pantalla. El gesto de abrir la mano se utiliza para lanzar la pelota o las balas (se obtiene de los bonus que aparecen durante la partida). Tanto para salir de la partida como del juego, solo hace falta mantener el gesto del perfil de la mano, cuando el contador llegue a 10 sale al menú de inicio.

¹<https://github.com/kolorowerowe/Arkanoid>

Capítulo 5

Conclusiones y trabajos futuros

Se han presentado varios métodos de detección de una mano para el manejo de un dispositivo. Para todos ellos se han realizado pruebas similares y estudiado los resultados. A continuación se presentarán las conclusiones de cada uno de ellos.

Template matching es un proceso demasiado estricto al utilizar una imagen como descriptor. No aporta seguridad para localizar la posición de la mano. Por ese motivo se descartó su uso.

Para el descriptor por LBP se debe de realizar un proceso que requiere un gran costo computacional, teniendo mayor costo transformar la imagen que el propio método. La conversión de una imagen a LBP comparando los valores del píxel con sus vecinos es una tarea lenta y más cuando se trata de una secuencia de imágenes. Esto repercute en las imágenes por segundo que es capaz de generar el vídeo de salida. Intentar realizar esta práctica en la Raspberry Pi a tiempo real es difícil, siendo ya complicado en el ordenador. Por esos motivos no se utiliza este descriptor.

Las pruebas realizadas con los dos métodos de segmentación de color, uno por medio de un modelo (Backprojection) y otro definiendo los rangos de valores, fueron más satisfactorias. El método de Backprojection depende de un modelo y solo se ajusta a sus características. Por tanto, definiendo los rangos de valores permite mayor flexibilidad a la hora de buscar ciertos rangos de colores.

Los descriptores que buscan los contornos de las figuras consiguen mostrar el borde de la mano, pero este borde no es perfecto. Aparecen contornos del fondo de la imagen, el borde de la mano no es continuo y a veces abarca objetos que no son de la mano (fondo, mangas, etc). Por esos motivos la utilización de funciones que buscan puntos convexos en el contorno no se utilizan. Este método podría utilizarse si existe un fondo controlado que no afecte la búsqueda del contorno de la mano.

La técnica CAMshift consigue rastrear las características resaltadas del color del guante, ofreciendo la posición y conteniéndose en una ventana.

Con esta ventana se puede comparar los contornos de la mano con patrones establecidos y clasificar su gesto. De estas comparaciones no se obtienen unos buenos resultados, siendo un clasificador poco eficiente.

También se ha utilizado un modelo neuronal pre-entrenado mediante la técnica de *transfer learning* con una red neuronal convolucional. Esta red presenta una estructura similar al de Mobilenet v2 al ser un modelo ya entrenado con esta estructura. Esta estructura está optimizada para dispositivos móviles, por eso su elección. Otras estructuras como VGG16 y VGG19 presentan modelos demasiado pesados, impidiendo cargarse en la Raspberry. Otra estructura utilizada fue NASNetMobile, pero no presenta tan buenos resultados como Mobilenet.

Respecto a la ejecución del modelo neuronal se ha observado que presenta ciertos inconvenientes en la Raspberry, ya que el tiempo de procesamiento para dar la predicción del gesto es entorno a 1,5 o 2 segundos. Esto no impide obtener las predicciones del modelo en la Raspberry, pero si afecta a obtener una salida fluida a tiempo real. La manipulación de las aplicaciones en el dispositivo Raspberry no puede producirse, debido a su poca potencia de procesamiento. Las predicciones en el ordenador tardan entorno a 0,15 o 0,2 segundos, ratio que permite manipular las imágenes en la aplicación implementada y jugar al juego Arkanoid.

Los resultados obtenidos con la utilización de redes neuronales convolucionales demuestran que las aplicaciones de machine learning dentro de la visión por computador son efectivas. Dando resultados superiores al 90 % de exactitud en el clasificador del modelo.

Para obtener unos mejores resultados a los obtenidos existen ciertas mejoras que se pueden aplicar. Una de ellas consiste en la utilización de redes neuronales con estructuras más óptimas o dedicadas a su ejecución en la Raspberry. Estas redes pueden ser recurrentes, al tratarse de la detección en vídeo, ofreciendo la posición y el gesto de la mano sin uso de la técnica CAMshift. Pero esto sigue sin solucionar el problema de manejar imágenes o aplicaciones en la Raspberry, que se debe a su potencia de procesamiento.

Conseguir mejores prestaciones en los dispositivos Raspberry se espera que sea posible en el futuro con las nuevas evoluciones. Otra posibilidad consiste en utilizar unos dispositivos nuevos adaptados a trabajar con gráficos y tensores de forma más óptima, como por ejemplo *Tinker board* de ASUS (40). Este dispositivo es similar a la Raspberry presentando unas prestaciones dedicadas a la reproducción multimedia de alta calidad, visión por ordenador, reconocimiento de gestos, etc. Con un dispositivo como este pensamos que se puede conseguir un vídeo fluido, e incluso optar al manejo de sistemas operativos de centro multimedia como *LibreELEC* de KODI.

Referencias

- [1] M. V. Antonio Lopez Peña, Ernest Valveny, “Detección de objetos,” 2019. [www.coursera.org].
- [2] P. Pastor Martín, “Usando redes neuronales convolucionales para convertir características visuales en estímulos sonoros,” 2018.
- [3] T. OpenCV, “Shape matching using hu moments (c++/python),” 2018. [Internet: descargado de <https://www.learnopencv.com/shape-matching-using-hu-moments-c-python/amp/> a 15-Diciembre-2019].
- [4] C. Pérez González, “Detección y seguimiento de objetos por colores en una plataforma raspberry pi,” 2016.
- [5] Vaelsys.com, “Soluciones de inteligencia y visión artificial.” [Vídeo promoción <https://www.youtube.com/watch?v=h38G3ZM7zc8>].
- [6] Y. M. Assael, B. Shillingford, S. Whiteson, and N. De Freitas, “Lipnet: End-to-end sentence-level lipreading,” *arXiv preprint arXiv:1611.01599*, 2016.
- [7] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, “Openpose: realtime multi-person 2d pose estimation using part affinity fields,” *arXiv preprint arXiv:1812.08008*, 2018.
- [8] G. Balakrishnan, A. Zhao, A. V. Dalca, F. Durand, and J. Guttag, “Synthesizing images of humans in unseen poses,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8340–8348, 2018.
- [9] C. Chan, S. Ginosar, T. Zhou, and A. A. Efros, “Everybody dance now,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 5933–5942, 2019.
- [10] R. Szeliski, “Computer vision algorithms and applications,” 2011. URL: <http://dx.doi.org/10.1007/978-1-84882-935-0> [Internet: descargado a 15-Diciembre-2019].
- [11] M. Levoy, “technical perspective computational photography on large collections of images,” *Communications of the ACM*, vol. 51, no. 10, pp. 86–86, 2008.

- [12] V. G. Pacheco, “Una breve historia del machine learning,” 2019. [Internet: descargado de <https://empresas.blogthinkbig.com/una-breve-historia-del-machine-learning/> a 15-Diciembre-2019].
- [13] C. S. Vega, [2017-2019]. Canal de Youtube Dot CSV. [DotCSV]. Recuperado de <https://www.youtube.com/channel/UCy5znSnfMsDwaLIROnZ7Qbg>.
- [14] D. TAKAHASHI, “Microsoft games exec details how project natal was born,” 2009. URL: <https://venturebeat.com/2009/06/02/microsoft-games-executive-describes-origins-of-project-natal-game-controls/> [Internet; descargado 11-diciembre-2019].
- [15] S. C. González, “Microsoft anuncia el regreso de kinect, muy evolucionado, por 399 dólares,” 2019.
- [16] Wikipedia, “Leap motion — wikipedia, la enciclopedia libre,” 2019. [Internet; descargado 11-Diciembre-2019].
- [17] F. Weichert, D. Bachmann, B. Rudak, and D. Fissler, “Analysis of the accuracy and robustness of the leap motion controller,” *Sensors*, vol. 13, no. 5, pp. 6380–6393, 2013.
- [18] D. Terdiman, “Leap motion: 3d hands-free motion control, unbound,” *cnet*, 2012.
- [19] Wikipedia, “Imagen digital — wikipedia, la enciclopedia libre,” 2019. [Internet; descargado 2-diciembre-2019].
- [20] Wikipedia, “Ruido en la fotografía digital — wikipedia, la enciclopedia libre,” 2019. [Internet; descargado 2-diciembre-2019].
- [21] C. M. A, “Procesamiento de imágenes en opencv,” 2017. [Internet: descargado de <http://acodigo.blogspot.com/2013/05/procesamiento-de-imagenes-en-opencv.html> a 15-Diciembre-2019].
- [22] D. L. S. Sugrañes, “Binarización de las imágenes.” [Internet: descargado de <http://www.dimages.es/Tutorial15-Diciembre-2019>].
- [23] B. Aldalur and M. Santamaría, “Realce de imágenes: filtrado espacial,” *Revista de tele-detección*, vol. 17, pp. 31–42, 2002.
- [24] T. OpenCV, “Opencv tutorials: Histogramas.” [Internet: descargado de OpenCV 2.4.13.7 documentation 8-Diciembre-2019].
- [25] Wikipedia, “Visión artificial — wikipedia, la enciclopedia libre,” 2019. [Internet; descargado 2-diciembre-2019].
- [26] T. OpenCV, “Opencv tutorials: Detector canny edge.” [Internet: descargado de OpenCV 2.4.13.7 documentation 8-Diciembre-2019].
- [27] Y. Cheng, “Mean shift, mode seeking, and clustering,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 17, no. 8, pp. 790–799, 1995.

- [28] T. OpenCV, “Opencv tutorials: Meanshift.” [Internet: descargado de OpenCV 3.4.9-pre documentation 8-Diciembre-2019].
- [29] Wikipedia, “Python — wikipedia, la enciclopedia libre,” 2019. [Internet; descargado 2-diciembre-2019].
- [30] J. J. L. Gómez, “Virtualenv y pip – instalando librerías en un entorno virtual python,” 2018. [Internet: descargado de <https://j2logo.com/virtualenv-pip-librerias-python/> a 15-Diciembre-2019].
- [31] T. OpenCV, “About opencv.” [Internet: descargado de <https://opencv.org/> 8-Diciembre-2019].
- [32] Wikipedia, “Numpy — wikipedia, la enciclopedia libre,” 2019. [Internet; descargado 2-diciembre-2019].
- [33] Wikipedia, “Pygame — wikipedia, la enciclopedia libre,” 2019. [Internet; descargado 2-diciembre-2019].
- [34] A. Sweigart, “Pyautogui.” URL: <https://pyautogui.readthedocs.io/en/latest/> [Internet: descargado 11-Diciembre-2019].
- [35] D. Calvo, “Función de activación – redes neuronales,” 2018. [Internet: descargado de <http://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/> a 15-Diciembre-2019].
- [36] M. Merino, “Conceptos de inteligencia artificial: qué son las gans o redes generativas antagónicas,” 2019.
- [37] Techworld.com, “Spyder reviwe - techworld.com,” 2014. [Internet: descargado de <http://review.techworld.com/applications/3238833/spyder-review/> a 15-Diciembre-2019].
- [38] A. Rosebrock, “Raspbian stretch: Install opencv 3 + python on your raspberry pi,” 2017. [Internet: descargado de <https://www.pyimagesearch.com/2017/09/04/raspbian-stretch-install-opencv-3-python-on-your-raspberry-pi/> a 15-Diciembre-2019].
- [39] Wikipedia, “Descriptores visuales — wikipedia, la enciclopedia libre,” 2019. [Internet; descargado 2-diciembre-2019].
- [40] A. C. Inc, “Tinker board, producto de asustek computer inc.,” 2019. [Internet; descargado 3-diciembre-2019].

Anexo A

Anexo A

Tabla A.1: Estructura de la red neuronal de Mobilenet v2

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 224, 224, 3)	0	
Conv1_pad (ZeroPadding2D)	(None, 225, 225, 3)	0	input_1
Conv1 (Conv2D)	(None, 112, 112, 32)	864	Conv1_pad
bn_Conv1 (BatchNormalization)	(None, 112, 112, 32)	128	Conv1
Conv1_relu (ReLU)	(None, 112, 112, 32)	0	bn_Conv1
expanded_conv_depthwise (DepthwiseConv2D)	(None, 112, 112, 32)	288	Conv1_relu
expanded_conv_depthwise_BN (BatchNormalization)	(None, 112, 112, 32)	128	expanded_conv_depthwise
expanded_conv_depthwise_relu (ReLU)	(None, 112, 112, 32)	0	expanded_conv_depthwise_BN
expanded_conv_project (Conv2D)	(None, 112, 112, 16)	512	expanded_conv_depthwise_relu
expanded_conv_project_BN (BatchNormalization)	(None, 112, 112, 16)	64	expanded_conv_project
block_1_expand (Conv2D)	(None, 112, 112, 96)	1536	expanded_conv_project_BN
block_1_expand_BN (BatchNormalization)	(None, 112, 112, 96)	384	block_1_expand
block_1_expand_relu (ReLU)	(None, 112, 112, 96)	0	block_1_expand_BN
block_1_pad (ZeroPadding2D)	(None, 113, 113, 96)	0	block_1_expand_relu
block_1_depthwise (DepthwiseConv2D)	(None, 56, 56, 96)	864	block_1_pad
block_1_depthwise_BN (BatchNormalization)	(None, 56, 56, 96)	384	block_1_depthwise
block_1_depthwise_relu (ReLU)	(None, 56, 56, 96)	0	block_1_depthwise_BN
block_1_project (Conv2D)	(None, 56, 56, 24)	2304	block_1_depthwise_relu
block_1_project_BN (BatchNormalization)	(None, 56, 56, 24)	96	block_1_project

Continúa en la siguiente página...

Tabla A.1 – continuación de la página anterior

Layer (type)	Output Shape	Param	Connected to
block_2_expand (Conv2D)	(None, 56, 56, 144)	3456	block_1_project_BN
block_2_expand_BN (BatchNormali	(None, 56, 56, 144)	576	block_2_expand
block_2_expand_relu (ReLU)	(None, 56, 56, 144)	0	block_2_expand_BN
block_2_depthwise (DepthwiseCon	(None, 56, 56, 144)	1296	block_2_expand_relu
block_2_depthwise_BN (BatchNorm	(None, 56, 56, 144)	576	block_2_depthwise
block_2_depthwise_relu (ReLU)	(None, 56, 56, 144)	0	block_2_depthwise_BN
block_2_project (Conv2D)	(None, 56, 56, 24)	3456	block_2_depthwise_relu
block_2_project_BN (BatchNormal	(None, 56, 56, 24)	96	block_2_project
block_2_add (Add)	(None, 56, 56, 24)	0	block_1_project_BN block_2_project_BN
block_3_expand (Conv2D)	(None, 56, 56, 144)	3456	block_2_add
block_3_expand_BN (BatchNormali	(None, 56, 56, 144)	576	block_3_expand
block_3_expand_relu (ReLU)	(None, 56, 56, 144)	0	block_3_expand_BN
block_3_pad (ZeroPad- ding2D)	(None, 57, 57, 144)	0	block_3_expand_relu
block_3_depthwise (DepthwiseCon	(None, 28, 28, 144)	1296	block_3_pad
block_3_depthwise_BN (BatchNorm	(None, 28, 28, 144)	576	block_3_depthwise
block_3_depthwise_relu (ReLU)	(None, 28, 28, 144)	0	block_3_depthwise_BN
block_3_project (Conv2D)	(None, 28, 28, 32)	4608	block_3_depthwise_relu
block_3_project_BN (BatchNormal	(None, 28, 28, 32)	128	block_3_project
block_4_expand (Conv2D)	(None, 28, 28, 192)	6144	block_3_project_BN

Continúa en la siguiente página...

Tabla A.1 – continuación de la página anterior

Layer (type)	Output Shape	Param	Connected to
block_4_expand_BN (BatchNormali	(None, 28, 28, 192)	768	block_4_expand
block_4_expand_relu (ReLU)	(None, 28, 28, 192)	0	block_4_expand_BN
block_4_depthwise (DepthwiseCon	(None, 28, 28, 192)	1728	block_4_expand_relu
block_4_depthwise_BN (BatchNorm	(None, 28, 28, 192)	768	block_4_depthwise
block_4_depthwise_relu (ReLU)	(None, 28, 28, 192)	0	block_4_depthwise_BN
block_4_project (Conv2D)	(None, 28, 28, 32)	6144	block_4_depthwise_relu
block_4_project_BN (BatchNormal	(None, 28, 28, 32)	128	block_4_project
block_4_add (Add)	(None, 28, 28, 32)	0	block_3_project_BN block_4_project_BN
block_5_expand (Conv2D)	(None, 28, 28, 192)	6144	block_4_add
block_5_expand_BN (BatchNormali	(None, 28, 28, 192)	768	block_5_expand
block_5_expand_relu (ReLU)	(None, 28, 28, 192)	0	block_5_expand_BN
block_5_depthwise (DepthwiseCon	(None, 28, 28, 192)	1728	block_5_expand_relu
block_5_depthwise_BN (BatchNorm	(None, 28, 28, 192)	768	block_5_depthwise
block_5_depthwise_relu (ReLU)	(None, 28, 28, 192)	0	block_5_depthwise_BN
block_5_project (Conv2D)	(None, 28, 28, 32)	6144	block_5_depthwise_relu
block_5_project_BN (BatchNormal	(None, 28, 28, 32)	128	block_5_project
block_5_add (Add)	(None, 28, 28, 32)	0	block_4_add block_5_project_BN
block_6_expand (Conv2D)	(None, 28, 28, 192)	6144	block_5_add

Continúa en la siguiente página...

Tabla A.1 – continuación de la página anterior

Layer (type)	Output Shape	Param	Connected to
block_6_expand_BN (BatchNormali	(None, 28, 28, 192)	768	block_6_expand
block_6_expand_relu (ReLU)	(None, 28, 28, 192)	0	block_6_expand_BN
block_6_pad (ZeroPad- ding2D)	(None, 29, 29, 192)	0	block_6_expand_relu
block_6_depthwise (DepthwiseCon	(None, 14, 14, 192)	1728	block_6_pad
block_6_depthwise_BN (BatchNorm	(None, 14, 14, 192)	768	block_6_depthwise
block_6_depthwise_relu (ReLU)	(None, 14, 14, 192)	0	block_6_depthwise_BN
block_6_project (Conv2D)	(None, 14, 14, 64)	12288	block_6_depthwise_relu
block_6_project_BN (BatchNormal	(None, 14, 14, 64)	256	block_6_project
block_7_expand (Conv2D)	(None, 14, 14, 384)	24576	block_6_project_BN
block_7_expand_BN (BatchNormali	(None, 14, 14, 384)	1536	block_7_expand
block_7_expand_relu (ReLU)	(None, 14, 14, 384)	0	block_7_expand_BN
block_7_depthwise (DepthwiseCon	(None, 14, 14, 384)	3456	block_7_expand_relu
block_7_depthwise_BN (BatchNorm	(None, 14, 14, 384)	1536	block_7_depthwise
block_7_depthwise_relu (ReLU)	(None, 14, 14, 384)	0	block_7_depthwise_BN
block_7_project (Conv2D)	(None, 14, 14, 64)	24576	block_7_depthwise_relu
block_7_project_BN (BatchNormal	(None, 14, 14, 64)	256	block_7_project
block_7_add (Add)	(None, 14, 14, 64)	0	block_6_project_BN block_7_project_BN
block_8_expand (Conv2D)	(None, 14, 14, 384)	24576	block_7_add
block_8_expand_BN (BatchNormali	(None, 14, 14, 384)	1536	block_8_expand

Continúa en la siguiente página...

Tabla A.1 – continuación de la página anterior

Layer (type)	Output Shape	Param	Connected to
block_8_expand_relu (ReLU)	(None, 14, 14, 384)	0	block_8_expand_BN
block_8_depthwise (DepthwiseCon	(None, 14, 14, 384)	3456	block_8_expand_relu
block_8_depthwise_BN (BatchNorm	(None, 14, 14, 384)	1536	block_8_depthwise
block_8_depthwise_relu (ReLU)	(None, 14, 14, 384)	0	block_8_depthwise_BN
block_8_project (Conv2D)	(None, 14, 14, 64)	24576	block_8_depthwise_relu
block_8_project_BN (BatchNormal	(None, 14, 14, 64)	256	block_8_project
block_8_add (Add)	(None, 14, 14, 64)	0	block_7_add block_8_project_BN
block_9_expand (Conv2D)	(None, 14, 14, 384)	24576	block_8_add
block_9_expand_BN (BatchNormali	(None, 14, 14, 384)	1536	block_9_expand
block_9_expand_relu (ReLU)	(None, 14, 14, 384)	0	block_9_expand_BN
block_9_depthwise (DepthwiseCon	(None, 14, 14, 384)	3456	block_9_expand_relu
block_9_depthwise_BN (BatchNorm	(None, 14, 14, 384)	1536	block_9_depthwise
block_9_depthwise_relu (ReLU)	(None, 14, 14, 384)	0	block_9_depthwise_BN
block_9_project (Conv2D)	(None, 14, 14, 64)	24576	block_9_depthwise_relu
block_9_project_BN (BatchNormal	(None, 14, 14, 64)	256	block_9_project
block_9_add (Add)	(None, 14, 14, 64)	0	block_8_add block_9_project_BN
block_10_expand (Conv2D)	(None, 14, 14, 384)	24576	block_9_add
block_10_expand_BN (BatchNormal	(None, 14, 14, 384)	1536	block_10_expand

Continúa en la siguiente página...

Tabla A.1 – continuación de la página anterior

Layer (type)	Output Shape	Param	Connected to
block_10_expand_relu (ReLU)	(None, 14, 14, 384)	0	block_10_expand_BN
block_10_depthwise (DepthwiseCo	(None, 14, 14, 384)	3456	block_10_expand_relu
block_10_depthwise_BN (BatchNor	(None, 14, 14, 384)	1536	block_10_depthwise
block_10_depthwise_relu (ReLU)	(None, 14, 14, 384)	0	block_10_depthwise_BN
block_10_project (Conv2D)	(None, 14, 14, 96)	36864	block_10_depthwise_relu
block_10_project_BN (BatchNorma	(None, 14, 14, 96)	384	block_10_project
block_11_expand (Conv2D)	(None, 14, 14, 576)	55296	block_10_project_BN
block_11_expand_BN (BatchNormal	(None, 14, 14, 576)	2304	block_11_expand
block_11_expand_relu (ReLU)	(None, 14, 14, 576)	0	block_11_expand_BN
block_11_depthwise (DepthwiseCo	(None, 14, 14, 576)	5184	block_11_expand_relu
block_11_depthwise_BN (BatchNor	(None, 14, 14, 576)	2304	block_11_depthwise
block_11_depthwise_relu (ReLU)	(None, 14, 14, 576)	0	block_11_depthwise_BN
block_11_project (Conv2D)	(None, 14, 14, 96)	55296	block_11_depthwise_relu
block_11_project_BN (BatchNorma	(None, 14, 14, 96)	384	block_11_project
block_11_add (Add)	(None, 14, 14, 96)	0	block_10_project_BN block_11_project_BN
block_12_expand (Conv2D)	(None, 14, 14, 576)	55296	block_11_add
block_12_expand_BN (BatchNormal	(None, 14, 14, 576)	2304	block_12_expand
block_12_expand_relu (ReLU)	(None, 14, 14, 576)	0	block_12_expand_BN
block_12_depthwise (DepthwiseCo	(None, 14, 14, 576)	5184	block_12_expand_relu

Continúa en la siguiente página...

Tabla A.1 – continuación de la página anterior

Layer (type)	Output Shape	Param	Connected to
block_12_depthwise_BN (BatchNor	(None, 14, 14, 576)	2304	block_12_depthwise
block_12_depthwise_relu (ReLU)	(None, 14, 14, 576)	0	block_12_depthwise_BN
block_12_project (Conv2D)	(None, 14, 14, 96)	55296	block_12_depthwise_relu
block_12_project_BN (BatchNorma	(None, 14, 14, 96)	384	block_12_project
block_12_add (Add)	(None, 14, 14, 96)	0	block_11_add block_12_project_BN
block_13_expand (Conv2D)	(None, 14, 14, 576)	55296	block_12_add
block_13_expand_BN (BatchNormal	(None, 14, 14, 576)	2304	block_13_expand
block_13_expand_relu (ReLU)	(None, 14, 14, 576)	0	block_13_expand_BN
block_13_pad (ZeroPad- ding2D)	(None, 15, 15, 576)	0	block_13_expand_relu
block_13_depthwise (DepthwiseCo	(None, 7, 7, 576)	5184	block_13_pad
block_13_depthwise_BN (BatchNor	(None, 7, 7, 576)	2304	block_13_depthwise
block_13_depthwise_relu (ReLU)	(None, 7, 7, 576)	0	block_13_depthwise_BN
block_13_project (Conv2D)	(None, 7, 7, 160)	92160	block_13_depthwise_relu
block_13_project_BN (BatchNorma	(None, 7, 7, 160)	640	block_13_project
block_14_expand (Conv2D)	(None, 7, 7, 960)	153600	block_13_project_BN
block_14_expand_BN (BatchNormal	(None, 7, 7, 960)	3840	block_14_expand
block_14_expand_relu (ReLU)	(None, 7, 7, 960)	0	block_14_expand_BN
block_14_depthwise (DepthwiseCo	(None, 7, 7, 960)	8640	block_14_expand_relu
block_14_depthwise_BN (BatchNor	(None, 7, 7, 960)	3840	block_14_depthwise

Continúa en la siguiente página...

Tabla A.1 – continuación de la página anterior

Layer (type)	Output Shape	Param	Connected to
block_14_depthwise_relu (ReLU)	(None, 7, 7, 960)	0	block_14_depthwise_BN
block_14_project (Conv2D)	(None, 7, 7, 160)	153600	block_14_depthwise_relu
block_14_project_BN (BatchNormal	(None, 7, 7, 160)	640	block_14_project
block_14_add (Add)	(None, 7, 7, 160)	0	block_13_project_BN block_14_project_BN
block_15_expand (Conv2D)	(None, 7, 7, 960)	153600	block_14_add
block_15_expand_BN (BatchNormal	(None, 7, 7, 960)	3840	block_15_expand
block_15_expand_relu (ReLU)	(None, 7, 7, 960)	0	block_15_expand_BN
block_15_depthwise (DepthwiseCo	(None, 7, 7, 960)	8640	block_15_expand_relu
block_15_depthwise_BN (BatchNor	(None, 7, 7, 960)	3840	block_15_depthwise
block_15_depthwise_relu (ReLU)	(None, 7, 7, 960)	0	block_15_depthwise_BN
block_15_project (Conv2D)	(None, 7, 7, 160)	153600	block_15_depthwise_relu
block_15_project_BN (BatchNorma	(None, 7, 7, 160)	640	block_15_project
block_15_add (Add)	(None, 7, 7, 160)	0	block_14_add block_15_project_BN
block_16_expand (Conv2D)	(None, 7, 7, 960)	153600	block_15_add
block_16_expand_BN (BatchNormal	(None, 7, 7, 960)	3840	block_16_expand
block_16_expand_relu (ReLU)	(None, 7, 7, 960)	0	block_16_expand_BN
block_16_depthwise (DepthwiseCo	(None, 7, 7, 960)	8640	block_16_expand_relu
block_16_depthwise_BN (BatchNor	(None, 7, 7, 960)	3840	block_16_depthwise

Continúa en la siguiente página...

Tabla A.1 – continuación de la página anterior

Layer (type)	Output Shape	Param	Connected to
block_16_depthwise_relu (ReLU)	(None, 7, 7, 960)	0	block_16_depthwise_BN
block_16_project (Conv2D)	(None, 7, 7, 320)	307200	block_16_depthwise_relu
block_16_project_BN (BatchNorma)	(None, 7, 7, 320)	1280	block_16_project
Conv_1 (Conv2D)	(None, 7, 7, 1280)	409600	block_16_project_BN
Conv_1_bn (BatchNormalization)	(None, 7, 7, 1280)	5120	Conv_1
out_relu (ReLU)	(None, 7, 7, 1280)	0	Conv_1_bn
global_average_pooling2d (Globala)	(None, 1280)	0	out_relu
dense (Dense)	(None, 3)	3843	global_average_pooling2d

Total params: 2,261,827

Trainable params: 3,843

Non-trainable params: 2,257,984