



Universidad Politécnica  
de Madrid



**Escuela Técnica Superior de  
Ingenieros Informáticos**

Grado en Ingeniería Informática

Trabajo Fin de Grado

**Reconocimiento de Mesas de Póker  
mediante Visión Computacional**

Autor: Álvaro Alonso Miguel

Tutor(a): Patricia Martín Chozas

Madrid, junio 2023

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

*Trabajo Fin de Grado*

*Grado en Ingeniería Informática*

*Título: Reconocimiento de Mesas de Póker mediante Visión Computacional*

*Junio 2023*

*Autor: Álvaro Alonso Miguel*

*Tutor:*

Patricia Martín Chozas

Departamento de Lingüística Aplicada a la Ciencia y a la Tecnología

ETSI Informáticos

Universidad Politécnica de Madrid

## **Dedicatoria**

Quiero dedicar este trabajo a mis abuelos Victoriano y Angelita, sin ellos nunca habría sido posible esta idea. Gracias por inculcarme el amor por las cartas en esas innumerables tardes de verano, ser siempre un apoyo incondicional y sentirlos siempre orgullosos de mí.

A mis padres Óscar y Montse; y a mi hermana Laura por aguantarme estos meses, ayudarme en todo lo que les era posible y por ser pilares fundamentales en mi vida.

A mis amigos y compañeros Rafa, Javi y Raúl por ayudarme cuando estaba atascado y darme ideas sobre qué hacer o explicarme sus puntos de vista en algunos apartados del proyecto.

Y a Marta, por confiar en mí y animarme siempre, incluso cuando ni yo mismo veía la luz en el camino que ha sido este trabajo.

# Resumen

En este trabajo se propone el diseño y desarrollo de una solución al problema de reconocimiento de mesas de póker y cálculo de probabilidad de victoria para dos jugadores a partir de imágenes mediante el uso de técnicas de visión computacional y simulaciones. Para ello, se introducen nuevos enfoques en la arquitectura de redes neuronales profundas y en la simulación de juegos.

Al comienzo de este documento se explican los fundamentos teóricos necesarios, así como, el estado de la cuestión en la actualidad. Para comprender la totalidad del trabajo, se presenta un sistema secuencial de tres módulos que obtiene imágenes de una cámara para analizarlas. Esta imagen pasa al segundo módulo que usa técnicas de preprocesamiento para eliminar el ruido de la imagen y entregar cada carta a un modelo convolucional basado en la arquitectura *VGG16* que ha sido entrenado previamente con un conjunto de datos generado y clasificado explícitamente para este trabajo. Una vez reconocidas las cartas estas se dividen en las distintas manos de los jugadores o cartas comunitarias y se realizan una simulación de Montecarlo para analizar el estado del juego, basándose de la etapa del juego en la que se encuentre. Toda esta información se muestra por pantalla al finalizar el proceso de análisis y se guarda para futuras inspecciones.

En cuanto a la arquitectura, se logra obtener una precisión de entrenamiento de aproximadamente el 99% tras 10 *epochs* superando por un amplio margen los métodos más punteros, así como una precisión realista en el apartado de la simulación. Finalmente se realiza un análisis del trabajo realizado y las posibles vías futuras que este podría tomar.

## Abstract

In this paper is purposed the design and development of a solution to the problem of poker table recognition and victory probability calculus for two players through images by the use of computer vision techniques and simulations. For this, are introduced, new focuses on neural network architectures and game simulations. At the beginning of the document is explained the needed theoretic knowledge, as well as the state of art.

To fully understand the paper, is presented a sequential system composed of three modules, which obtains images through a camera to analyze them. This image is given to the second module which uses preprocessing techniques to remove the noise from the image and input the card to a convolutional model based on the *VGG16* architecture. This model has been previously trained with a dataset generated and classified, especially for this project. Once recognised, the cards are divided into the different hands of the players or community cards and is done a Montecarlo simulation to analyze the state of the game, based on the stage of the game where the image is. All this information is displayed on the screen when the process is finished and is saved for future inspections.

Regarding the architecture, is achieved a training accuracy of approximately 99% after 10 epochs overcoming by a wide range, leading methods, as well as a realistic accuracy over the simulation side. Finally, is done a reasoning of the job done and the possible path forward.

# Tabla de contenidos

<b>1</b>	<b>Introducción</b> .....	<b>1</b>
1.1	Motivación.....	1
1.2	Estructura del documento.....	2
<b>2</b>	<b>Estado de la cuestión</b> .....	<b>3</b>
2.1	Metodología del estado de la cuestión.....	3
2.2	Póker por computador.....	3
2.3	Reconocimiento de objetos.....	5
<b>3</b>	<b>Fundamentos teóricos</b> .....	<b>6</b>
3.1	Procesamiento y extracción de características de las imágenes.....	6
3.1.1	Segmentación.....	7
3.1.1.1	Segmentación binaria.....	7
3.1.1.2	Segmentación por color.....	8
3.1.2	Detector de bordes Canny.....	9
3.2	Redes neuronales profundas.....	9
3.3	Redes neuronales convolucionales.....	10
3.3.1	Operador de convolución.....	11
3.3.2	Arquitecturas convolucionales populares.....	12
3.4	Redes neuronales recurrentes.....	13
3.4.1	Redes neuronales LSTM.....	14
3.5	Simulación Montecarlo.....	15
<b>4</b>	<b>Diseño y solución propuesta</b> .....	<b>17</b>
4.1	Obtención de datos de entrenamiento.....	17
4.2	Diseño e implementación de la arquitectura.....	17
4.3	Funcionamiento del sistema.....	19
4.4	Lenguaje y entorno de desarrollo.....	25
4.5	Selección de librerías y recursos.....	26
<b>5</b>	<b>Resultados experimentales</b> .....	<b>28</b>
5.1	Precisión del modelo.....	28
5.2	Matriz de confusión.....	29
5.3	Simulación de Montecarlo.....	30
<b>6</b>	<b>Conclusiones y vías futuras</b> .....	<b>31</b>
<b>7</b>	<b>Bibliografía</b> .....	<b>33</b>

## Índice de figuras

<b>Figura 3.1</b>	Representación de una matriz.....	6
<b>Figura 3.2</b>	Representación de un histograma .....	6
<b>Figura 3.3</b>	Función gaussiana.....	7
<b>Figura 3.4</b>	Convolución discreta.....	11
<b>Figura 3.5</b>	Evolución de las arquitecturas convolucionales hasta 2023 .....	13
<b>Figura 3.6</b>	Representación de una red neuronal recurrente.....	13
<b>Figura 3.7</b>	Representación de un bloque LSTM .....	14
<b>Figura 4.1</b>	Arquitectura VGG16 propuesta.....	18
<b>Figura 5.1</b>	Evolución de la precisión de entrenamiento y validación, así como, de los valores de la función de pérdida.....	28
<b>Figura 5.2</b>	Matriz de confusión.....	29

## Índice de ilustraciones

<b>Ilustración 3.1</b> Segmentación binaria de una carta.....	8
<b>Ilustración 3.2</b> Imagen a segmentar y segmentación en 16 grupos logrando un efecto de posterizado .....	8
<b>Ilustración 4.1</b> Pantalla del sistema.....	20
<b>Ilustración 4.2</b> Imagen tomada por el primer módulo .....	20
<b>Ilustración 4.3</b> Imagen binarizada .....	21
<b>Ilustración 4.4</b> Contornos exteriores de las cartas .....	22
<b>Ilustración 4.5</b> Carta recortada y orientada .....	22
<b>Ilustración 4.6</b> Esquina superior izquierda de la carta detectada.....	23
<b>Ilustración 4.7</b> Cartas y estado del juego reconocido .....	24
<b>Ilustración 4.8</b> Resultado final tras simulación.....	25
<b>Ilustración 5.1</b> Comparativa de la imagen de salida tras la simulación con los resultados proporcionados por la calculadora .....	30

# 1 Introducción

El proyecto se enmarca en el ámbito del póker, concretamente en la modalidad *Texas Hold'em*, y se enfoca en el desarrollo de un sistema capaz de reconocer las manos de póker de los jugadores y las cartas comunitarias, así como de calcular la probabilidad de victoria. Para abordar este reto, se propone un caso de uso con dos jugadores, ya que es la situación más habitual en esta modalidad de póker, y se diferenciará entre las distintas etapas del juego, *pre-flop*, *flop*, *turn* y *river*. Estas etapas se explican de la siguiente forma, *pre-flop* es cuando los jugadores han recibido sus cartas, pero aún no se han mostrado las cartas comunitarias. El *flop* sucede cuando se reparten las primeras tres cartas comunitarias. Para el *turn* y *river* se muestra una carta comunitaria más en cada etapa, quedando finalmente cinco cartas comunitarias y las dos cartas de cada jugador. Con las dos cartas personales el jugador debe formar la mejor jugada con las cartas comunitarias que más le convengan por cada etapa, pudiendo seleccionar cualquier combinación de cartas siempre y cuando sume un total de cinco cartas.

El póker es un juego de cartas que ha ganado popularidad en todo el mundo y es uno de los juegos de casino más jugados en línea. El póker ha evolucionado de un juego de azar a un juego de habilidad, con jugadores que utilizan estrategias y tácticas para mejorar sus posibilidades de ganar. Con el auge del póker en línea, también ha aumentado la demanda de retransmisiones de torneos de póker en vivo [1]. Estas retransmisiones son cada vez más comunes en la televisión y en plataformas en línea. En ellas, los espectadores pueden ver a los jugadores profesionales de póker competir en vivo, lo que añade emoción y drama al juego.

En la actualidad, el póker se ha convertido en un deporte profesional con jugadores altamente especializados que compiten por grandes premios en efectivo. Esto ha dado lugar a una gran demanda de retransmisiones en vivo, que permiten a los espectadores seguir a los mejores jugadores del mundo mientras compiten en torneos de alto nivel [2], por ello se necesita conocer las cartas de los jugadores y las probabilidades que tienen de ganar.

La alternativa actual consiste en el uso de barajas especiales y con un coste más elevado puesto que cuentan con una tecnología muy específica, chips RFID, para detectar de qué carta se trata en cada caso. Este método podría ser sustituido por el reconocimiento de cartas mediante visión computacional.

## 1.1 Motivación

La motivación para llevar a cabo este proyecto es el desafío que presenta, dado que el reconocimiento de objetos es un campo complejo, y en este trabajo se abordará desde el reconocimiento de las cartas individuales hasta el reconocimiento de las diferentes cartas que pertenecen los jugadores presentes en la mesa. Dado que en los torneos de póker se juega un gran número de manos, la implementación de este sistema podría reducir el coste y, de alguna forma, contribuir también al cuidado del medio ambiente ya que al estar cambiando constantemente de baraja si se usasen barajas convencionales se reduciría el gasto tanto de materiales como monetario. Esto también permitiría mayor libertad de movimiento a los jugadores ya que no precisarían de posicionar las cartas en unos lugares muy concretos para su correcta lectura,

puesto que estos deben posicionar las cartas en unos rectángulos donde se leen las cartas mediante tecnología RFID [3], que consiste en la identificación por radiofrecuencia de un objeto mediante un lector que en este caso sería donde posicionan las cartas los jugadores.

En resumen, este proyecto se enfoca en el desarrollo de un sistema de reconocimiento de manos de póker y cartas comunitarias en la modalidad Texas *Hold'em*, con el objetivo de reducir los costos asociados al uso de barajas especiales en los torneos y contribuir al desarrollo de las tecnologías presentes en el campo. Además, se trata de un proyecto desafiante y complejo que aborda el reconocimiento de objetos en varias capas, desde el reconocimiento de cartas hasta el cálculo de probabilidades de victoria.

## **1.2 Estructura del documento**

En lo relacionado con la estructura de este documento, se ha establecido una división de cinco capítulos, sin incluir el actual. En el Capítulo 2 se realiza una revisión del estado de la cuestión sobre publicaciones previas dentro del reconocimiento de cartas y el estado del póker en la actualidad. En el Capítulo 3 se desarrollan algunas explicaciones básicas sobre fundamentos teóricos necesarios para la comprensión del trabajo, pasando por técnicas de extracción de características de imágenes, redes neuronales convolucionales (CNN) y redes de neuronas recurrentes.

Posteriormente, el Capítulo 4 describe el diseño de la arquitectura propuesta para la resolución del problema planteado, listando finalmente la selección de librerías y herramientas escogidas para el desarrollo del proyecto. Más adelante, se realiza un análisis de los resultados obtenidos en el Capítulo 5 y, por último, se incluye una breve conclusión sobre el trabajo realizado y se abre la puerta a posibles vías de experimentación futuras.

## **2 Estado de la cuestión**

Este capítulo expone el estado de la cuestión relacionado con el trabajo desarrollado. En primer lugar, en la Sección 2.1 se presenta la metodología aplicada en este capítulo. En segundo lugar, en la Sección 2.2 se describe la situación actual del póker por computador y se identifican las tecnologías utilizadas para el reconocimiento de cartas. Finalmente, la Sección 2.3 expone brevemente el campo del reconocimiento de objetos.

### **2.1 Metodología del estado de la cuestión**

Este trabajo se ha guiado por la metodología Torracó, es una técnica de investigación cualitativa que se utiliza para analizar y sintetizar múltiples casos de estudio relacionados. Esta metodología fue desarrollada por el investigador Richard J. Torracó [4] y se enfoca en la identificación de patrones y temas comunes en los datos recopilados a través de múltiples casos de estudio.

La metodología Torracó se divide en varias etapas. En la primera etapa, se lleva a cabo una revisión sistemática de la literatura relacionada con el tema de investigación. Esto incluye la búsqueda y selección de documentos relevantes que se utilizarán para la síntesis.

En la segunda etapa se utiliza una técnica de análisis de contenido para extraer información de los casos de estudio seleccionados. Esto implica la identificación de temas y patrones comunes, así como la codificación y categorización de los datos.

En la tercera etapa se lleva a cabo una síntesis cualitativa de los datos recopilados. Esto implica la identificación de relaciones entre los temas y patrones comunes y la creación de una representación integrada de los resultados de los casos de estudio.

En la cuarta etapa, se realiza una evaluación de la calidad de la síntesis y se identifican las limitaciones del enfoque utilizado. Finalmente, se presenta un informe detallado que resume los hallazgos de la síntesis y se discuten las implicaciones prácticas y teóricas de los resultados.

En resumen, la metodología Torracó es una técnica de investigación cualitativa que se utiliza para sintetizar múltiples casos de estudio relacionados. Esta metodología se divide en varias etapas que incluyen una revisión sistemática de la literatura, un análisis de contenido, una síntesis cualitativa y una evaluación de la calidad de la síntesis. Al utilizar esta metodología, los investigadores pueden obtener una comprensión más profunda y completa del tema de investigación.

### **2.2 Póker por computador**

Además de las retransmisiones en vivo, también se utilizan tecnologías avanzadas para mejorar la experiencia del espectador y aumentar la precisión en el juego. Una de las tecnologías más importantes que se utilizan en las retransmisiones de póker es el reconocimiento de cartas. El reconocimiento de cartas es el proceso de utilizar tecnología para identificar las cartas que se están jugando en una mano de póker [3].

Las tecnologías de reconocimiento de cartas se utilizan para aumentar la precisión en las retransmisiones de póker. Esto se hace utilizando cámaras especiales y software avanzado para capturar imágenes de las cartas y analizarlas para determinar su valor. Esta tecnología ha sido fundamental en la mejora de la precisión en el juego y ha ayudado a reducir los errores humanos [1]. Otro enfoque para el reconocimiento de cartas es el uso de plantillas, donde se comparan las cartas que se sitúan en la mesa con fotos de cartas previamente tomadas y en función del porcentaje de similitud se predice un valor [5].

La tecnología de reconocimiento de cartas también ha dado lugar a nuevos formatos de póker en línea, como Zoom Poker, en el que los jugadores pueden jugar más manos en un período de tiempo más corto gracias a la rápida identificación de las cartas y la automatización de muchas de las funciones del juego [1]. Dichas funciones se pueden resumir brevemente en que el jugador puede retirarse de la mesa en cualquier momento y recibir automáticamente una nueva mano en otra mesa. Esto se puede hacer independientemente del momento de la partida lo que agiliza el juego y permite al jugador aumentar el tiempo de juego real lo que se traduce en más manos y mayor experiencia en el juego.

Otro avance tecnológico que ha cambiado el mundo de las retransmisiones de póker es la realidad aumentada [6]. La realidad aumentada concede a los espectadores, mediante un dispositivo ver gráficos en tiempo real sobre la mesa de juego, como las manos de los jugadores y las probabilidades de ganar. La realidad aumentada también permite a los espectadores interactuar con la mesa de juego, lo que añade una nueva dimensión al juego.

La realidad aumentada también ha permitido la creación de nuevas formas de juego, como el póker virtual en línea, [2] en el que los jugadores pueden jugar en una mesa virtual y ver a sus oponentes en tiempo real gracias a la realidad aumentada. Estas innovaciones han ayudado a expandir el mundo del póker más allá de los casinos tradicionales y han permitido a los jugadores competir con otros jugadores de todo el mundo desde la comodidad de sus propias casas.

Además de los avances en tecnología, también ha habido cambios en la forma en que se retransmite el póker en vivo. Con la pandemia del COVID-19, muchas retransmisiones de póker se han trasladado a un formato en línea para cumplir con las restricciones de distanciamiento social. Esto ha dado lugar a una mayor demanda de retransmisiones de póker en línea y ha abierto nuevas oportunidades para los jugadores y las plataformas de póker en línea, tales como PokerStars<sup>1</sup>, 888Poker<sup>2</sup> o Winamax<sup>3</sup>.

En conclusión, el póker es un juego que ha evolucionado a lo largo del tiempo gracias a la tecnología y las retransmisiones en vivo se han vuelto cada vez más comunes y precisas gracias al uso de tecnologías de reconocimiento de cartas y realidad aumentada. Además, la pandemia del COVID-19 ha acelerado la transición del póker a un formato en línea, lo que ha abierto nuevas oportunidades para los jugadores y las plataformas de póker en línea. Se espera

---

<sup>1</sup> PokerStars - <https://www.pokerstars.es/>

<sup>2</sup> 888Poker - <https://www.888poker.es/>

<sup>3</sup> Winamax - <https://www.winamax.es/>

que en el futuro sigan surgiendo nuevas tecnologías y formatos de juego que continúen impulsando la popularidad del póker en todo el mundo.

## **2.3 Reconocimiento de objetos**

El reconocimiento de objetos es una tarea esencial en el campo de la inteligencia artificial y la visión por computador. Su objetivo es identificar y clasificar objetos presentes en imágenes o videos mediante el análisis de patrones y características visuales. En la última década, el reconocimiento de objetos ha experimentado un avance significativo gracias a la aplicación de técnicas de aprendizaje profundo y grandes conjuntos de datos [7] [8].

Uno de los desafíos principales del reconocimiento de objetos es la variabilidad en la apariencia de los objetos, como su tamaño, orientación, iluminación y deformación. Para abordar estos desafíos, se han desarrollado sistemas de reconocimiento de objetos basados en redes neuronales convolucionales (CNN), [9] que son capaces de extraer características visuales de las imágenes de entrada y clasificar objetos con alta precisión.

Para abordar el reconocimiento de cartas se han desarrollado sistemas de reconocimiento que utilizan técnicas de aprendizaje profundo para identificar las cartas en una imagen y clasificarlas según su palo y valor [10]. Estos sistemas se basan en la capacidad de las redes neuronales convolucionales para aprender patrones y características visuales específicos de las cartas.

El reconocimiento de cartas es una tarea específica del reconocimiento de objetos que se enfoca en identificar y clasificar cartas en una imagen o vídeo. Esta tarea tiene una amplia variedad de aplicaciones prácticas, incluyendo la identificación de caracteres en documentos y la interpretación de juegos de cartas [11].

A pesar de los avances recientes en el reconocimiento de cartas, todavía existen desafíos importantes que deben ser superados. Uno de los desafíos es la variabilidad en la apariencia de las cartas, como diferentes diseños y colores, y las diferentes fuentes de iluminación en diferentes escenarios [7]. Otro desafío es la identificación de cartas en situaciones donde las cartas están parcialmente ocultas, inclinadas, o cuando hay distracciones en la imagen [12].

Para abordar estos desafíos, los sistemas de reconocimiento de cartas utilizan técnicas de preprocesamiento de imágenes para normalizar las imágenes de entrada y mejorar la detección de las cartas. Además, se utilizan técnicas de segmentación para separar las cartas del resto de la imagen [7].

En resumen, el reconocimiento de cartas es una aplicación importante del reconocimiento de objetos en la inteligencia artificial y la visión por computador. Los sistemas de reconocimiento de cartas basados en aprendizaje profundo y redes neuronales han demostrado un rendimiento excepcional en la detección y clasificación de cartas en una variedad de escenarios, pero aún existen desafíos importantes que deben ser superados para mejorar aún más la precisión del reconocimiento de cartas.

### 3 Fundamentos teóricos

Este capítulo está dedicado a comentar, de manera breve, todos los aspectos teóricos importantes que envuelven las redes neuronales artificiales cuya comprensión es necesaria para la correcta comprensión del trabajo, como la extracción de características, las redes recurrentes y algunas de las arquitecturas más extendidas.

#### 3.1 Procesamiento y extracción de características de las imágenes

En esta sección se explican las formas de representación de imágenes más relevantes dentro del campo del procesamiento de imágenes y la extracción de características; la segmentación, el filtrado y el operador de bordes de Canny [13]. Para comprender el mecanismo detrás de los métodos mencionados previamente, es necesario explicar la base matemática sobre la que se cimentan, en este caso, la matriz, el histograma y la función gaussiana.

Una imagen está formada por diferentes píxeles, cada uno se puede representar por un valor numérico. La matriz, representada en la Figura 3.1, es un conjunto bidimensional de números o símbolos algebraicos colocados en líneas horizontales y verticales y dispuestos en forma de rectángulo, o dicho con otras palabras permite representar la disposición de los píxeles de la imagen.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

Figura 3.1 Representación de una matriz [14]

El histograma es una representación gráfica, representada en la Figura 3.2, de una variable en forma de barras, donde la superficie de cada barra es proporcional a la frecuencia de los valores representados [15]. Esto permite representar los grupos de píxeles para generar regiones. Estas regiones dan lugar a la segmentación.

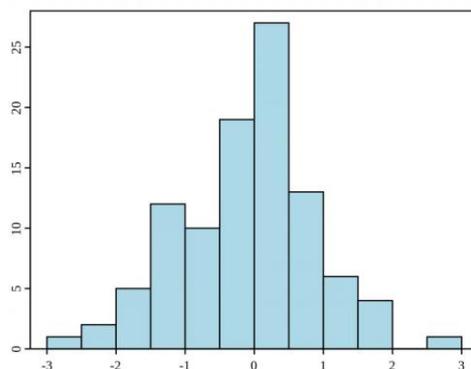
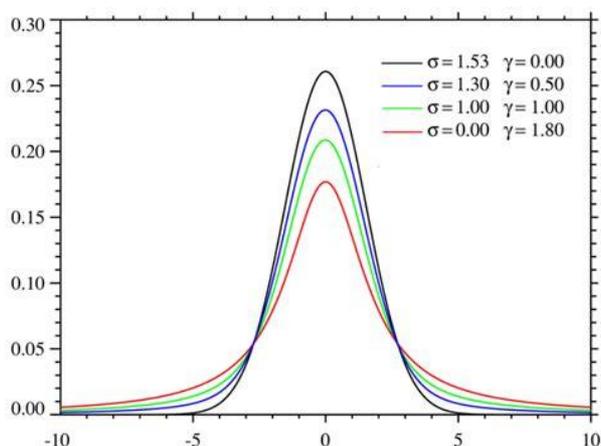


Figura 3.2 Representación de un histograma [16]

La función gaussiana, también conocida como campana de Gauss, representada en la Figura 3.3, es una función definida por la siguiente expresión.

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}}$$

Donde  $a, b$  y  $c$  son constantes reales, el parámetro  $a$  es el valor del punto más alto de la campana,  $b$  es la posición del centro de la campana y  $c$  es la desviación estándar que controla el ancho de la campana. Las funciones gaussianas se suelen utilizar en estadística y la más conocida es aquella que corresponde a una distribución normal.



**Figura 3.3** Función gaussiana [17]

### 3.1.1 Segmentación

La segmentación es uno de los problemas generales del campo de la visión computacional y se trata de un proceso de clasificación por píxel que asigna una categoría a cada píxel de la imagen analizada. El objetivo de la segmentación es localizar regiones con significado por lo tanto se usa tanto para localizar objetos como para encontrar sus bordes dentro de una imagen [18]. El resultado de la segmentación de una imagen es el conjunto de segmentos que cubren toda la imagen sin superponerse lo que nos permite representarla como un conjunto de contornos.

Dentro de la segmentación disponemos de varios algoritmos. Cada algoritmo discrimina una determinada cantidad de categorías, y a cada categoría le asigna una etiqueta o identificador. Esta etiqueta se trata de un valor entero que el algoritmo usa para expresar esa categoría. En este caso se explicará la segmentación binaria (Sección 3.1.1.1) y la segmentación por color (Sección 3.1.1.2), aunque también se cuenta con la segmentación por texturas, segmentación semántica o superpíxel.

#### 3.1.1.1 Segmentación binaria

La mínima clasificación posible es la que discrimina dos categorías, se trata de la segmentación binaria. Los segmentadores binarios son los que disponen únicamente de dos categorías denominadas generalmente frente y fondo [19]. Su uso principal es el de extraer un segmento con el objeto buscado.

La umbralización es la segmentación más básica y simple, opera cada píxel de manera independiente clasificándolos por su intensidad comparada con un umbral dado (si es más claro corresponde al frente, si es más oscuro corresponde al fondo).



**Ilustración 3.1** Segmentación binaria de una carta

### 3.1.1.2 Segmentación por color

La segmentación por color consiste en clasificar píxeles exclusivamente por su color como el propio nombre indica. La umbralización pertenece a esta categoría, y salvo esta excepción la segmentación por color no tiene particular relevancia en problemas de visión computacional, pero si lo tiene en procesamiento de imágenes [20]. Dentro de la segmentación por color el método principal es el basado en histograma.

Los métodos basados en el histograma son muy eficientes en comparación con otros métodos de segmentación de la imagen, ya que normalmente requieren solo una pasada por los píxeles. En esta técnica, un histograma se calcula a partir de todos los píxeles de la imagen, y los picos y valles se utilizan para localizar los grupos en la imagen (el color o intensidad pueden ser usados como medida).

Un refinamiento de esta técnica consiste en aplicar de forma recursiva el método de búsqueda de histograma a los clústeres de la imagen con el fin de dividirlos en grupos más pequeños hasta que no se puedan formar más agrupaciones. Una desventaja es que puede ser difícil identificar los picos y valles importantes en la imagen.



**Ilustración 3.2** Imagen a segmentar y segmentación en 16 grupos logrando un efecto de posterizado [21]

### 3.1.2 Detector de bordes Canny

La detección de bordes es una técnica que permite aislar los objetos y separarlos del fondo, una vez obtenidos los bordes se puede detectar los contornos de los objetos presentes en la imagen y es posible clasificarlos [22]. Pero no es tan sencillo, las imágenes digitales no son perfectas y el ruido de los dispositivos con los que se generan o los efectos de iluminación alteran la realidad. Es por ello por lo que se debe saber cómo corregirlo.

El objetivo es suavizar la imagen o lo que es lo mismo, eliminar los detalles (se puede asemejar al desenfoque de una cámara). En el tratamiento digital de imágenes hay diferentes métodos para eliminar el ruido, siendo el más extendido el filtro Gaussiano. Se trata de realizar una ponderación siguiendo la campana de Gauss, con esto se consigue dar más importancia a los píxeles que están más cerca del centro de los que están más alejados. Esto se hace a través de una máscara o *kernel* de convolución que se explica con detalle en el Capítulo 3.3.

Una vez suavizada la imagen ya se puede aplicar el detector de bordes de Canny. Pese a que existen múltiples algoritmos de detección de bordes se ha seleccionado Canny por su gran éxito y extensión en el campo respecto al resto [23]. Dicho detector se divide en tres pasos, encontrar el gradiente de intensidad, supresión de falsos máximos y realizar un umbral por histéresis.

En primer lugar, se debe encontrar el gradiente de intensidad, este método de detección se basa en el principio de la primera derivada matemática, mide las evoluciones y los cambios de una variable. Básicamente se centra en detectar cambios de intensidad. Por lo que se debe alisar la imagen eliminando el ruido para mejorar la calidad de los resultados, se suele usar un filtro gaussiano. Una vez la imagen ha sido alisada se calcula el gradiente de esta y se filtra de nuevo, aunque esta vez usando un filtro de Sobel [24]. Se trata de un operador diferencial que calcula la aproximación al gradiente de la intensidad de una imagen.

En segundo lugar, se debe suprimir los falsos máximos. El objetivo de esta fase es escanear la imagen para suprimir los píxeles que no forman parte de los bordes. De esta forma se obtienen bordes más finos en cuanto a los píxeles respecta y mayor precisión en el contorno.

Finalmente se debe realizar un umbral por histéresis. La umbralización nos permite segmentar una imagen en sus diferentes objetos como ya hemos explicado previamente. Por lo tanto, el último paso del algoritmo nos permitirá determinar si un píxel forma parte del contorno o no. Si su umbral es máximo se considera que forma parte del borde, si es mínimo no es parte del borde y si se encuentra entre ambos valores será parte del borde si está conectado con un píxel que ya forma parte del borde.

## 3.2 Redes neuronales profundas

Una red de neuronas artificial es una estructura de datos que, mediante una serie de algoritmos de aprendizaje, intenta reconocer relaciones y patrones subyacentes en conjuntos de datos mediante procesos de entrenamiento inspirados en el funcionamiento del cerebro.

La convención que existe sobre qué se considera como una red de neuronas artificiales profunda o *deep* es su número de capas ocultas, que debe ser mayor o igual que dos. Una red neuronal con una única capa oculta se consideraría *shallow* o poco profunda y, aunque es cierto que este tipo de redes puede aproximar una función con un error relativamente bajo, también es necesario aumentar el número de neuronas de la capa oculta para obtener precisiones más altas durante su entrenamiento, ya que solamente utiliza un único nivel de abstracción [25].

El paso de *shallow* a *deep* tiene un problema asociado, el desvanecimiento del gradiente. Este fenómeno se produce debido a que, durante cada iteración del entrenamiento cada uno de los pesos de las conexiones de una red calcula una variación proporcional a la derivada parcial de la función de error con respecto a los pesos actuales. El problema aparece cuando, por culpa de la función de activación, el gradiente resultante es tan pequeño que impide que el peso de las conexiones se actualice, pudiéndose dar el caso de que el entrenamiento de la red neuronal se detenga por completo. Una de las formas de lidiar con los efectos de este problema es el desarrollo de nuevas funciones de activación cuyas derivadas permiten tomar valores más grandes, llevando cuidado además con el posible efecto contrario que podría producirse. Este efecto es conocido como “explosión del gradiente”.

Cabe mencionar que existe una serie de optimizadores que tratan de acelerar la convergencia de este algoritmo. Esto se puede hacer teniendo en cuenta el momento del descenso del gradiente, la ineficiencia producida por las posibles superficies elongadas que puede presentar el gradiente o una combinación de ambas entre otras soluciones.

### **3.3 Redes neuronales convolucionales**

Las redes de neuronas convolucionales (CNN) nacieron en el año 1980 con el trabajo de Kunihiko Fukushima y su *necognitron* [26]. Pero no fue hasta 1989 cuando Yann LeCun entrenó por primera vez este tipo de redes de neuronas utilizando el descenso del gradiente para resolver un problema de clasificación de imágenes [27]. Es a partir de ese instante cuando empiezan a surgir nuevas arquitecturas basadas en la misma idea, con LeNet-5 [28] a la cabeza, siendo hoy en día una de las líneas de investigación más activas dentro del campo del *deep learning*.

El componente básico de este tipo de redes son las capas convolucionales, que se dividen en porciones, que dan profundidad a la capa y que están compuestas a su vez por neuronas, dentro de estas capas existen ciertas capas especiales conocidas como capas de *pooling*. Cada una de estas porciones se encarga de detectar una cierta característica, de forma que una capa convolucional representa un conjunto de características, una por porción y sus respectivas localizaciones en el espacio de la entrada. Esto permite un análisis de los patrones de entrada de bajo nivel en las capas iniciales, que a medida que profundiza es cada vez más fino y detallado. Es por este motivo que se les suele llamar también filtros o mapas de activación. Este tipo de redes son una herramienta poderosa para la resolución de problemas de clasificación con entradas de gran dimensionalidad, como son las imágenes, puesto que todas las neuronas que ocupan la misma porción de la entrada con su filtro a diferente profundidad comparten parámetros.

Las capas de *pooling* son diferentes capas de convolución en el sentido de que no utilizan pesos y simplemente agregan los valores de la matriz de la capa que les precede. El objetivo de estas capas es el de agrupar pequeñas zonas en las que los valores serían muy pocos para ser analizados de manera conjunta y mejore el rendimiento global de la red. Existen múltiples formas de agregarlos, pero las más populares son el cálculo de la media de los valores incluidos en el campo receptivo, de manera que todos ellos aportan, o bien sencillamente tomando el valor máximo.

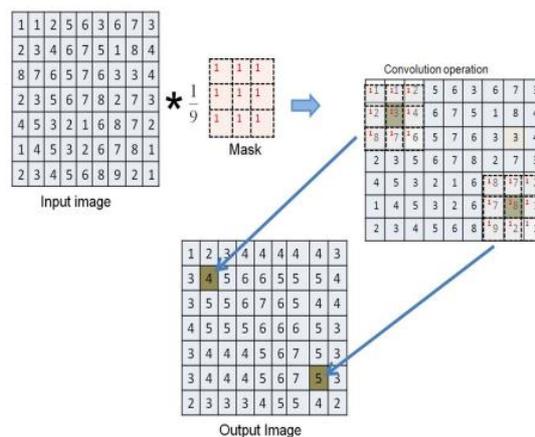
### 3.3.1 Operador de convolución

La operación de convolución es la operación básica sobre la que se asienta el funcionamiento de los modelos convolucionales. Se trata de un operador matemático que convierte dos funciones  $f$  y  $g$  en una tercera función que representa la magnitud en la que se superponen  $f$  y una versión trasladada e invertida de  $g$ . Es decir, nos da la magnitud del recorrido que hacer una función  $f$  desplazándose en otra función  $g$  [29]. Esta operación discreta se realiza sobre una imagen utilizando una máscara, como se define en la siguiente expresión, asumiendo una sola dimensión para simplificar.

$$o(r, s) = hI = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} h(i, j) I\left(r + i - \frac{m-1}{2}, s + j - \frac{n-1}{2}\right)$$

Donde  $I \in \mathbf{R}^{M \times N}$  es una imagen y  $h \in \mathbf{R}^{m \times n}$  es una máscara de convolución. Habitualmente se tiene que  $M \ll m$  y  $N \ll n$ .

Esta operación viene a expresar que el resultado de la operación sobre un píxel depende de la suma de los valores de la máscara multiplicados por los valores de los píxeles sobre los que se posiciona, centrando la máscara en el píxel sobre el que se está operando, como se muestra en la Figura 3.4.



**Figura 3.4** Convolución discreta (Fuente: UPM)

La respuesta es que simula la existencia de los píxeles ausentes como relleno o *padding* con valor de 0 normalmente. Esta modificación de la operación de convolución recibe el nombre de *half padding* y tiene como característica diferenciadora que la salida que produce,  $o(r, t)$ , es del mismo tamaño que la

entrada, o lo que es lo mismo, de tamaño  $M \times N$ . Una variación muy utilizada de este comportamiento se produce haciendo uso de *strides*, que consisten en saltarse  $s$  posiciones en la matriz de entrada al convolucionar con la máscara o *kernel*.

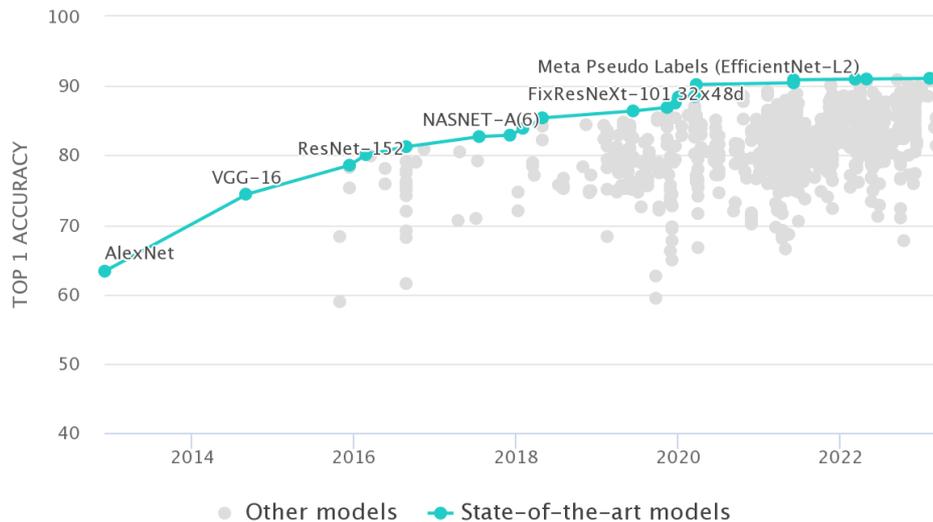
### 3.3.2 Arquitecturas convolucionales populares

Entre las arquitecturas convolucionales más destacadas se encuentra *AlexNet* [30], que fue el primer modelo convolucional entrenado exitosamente con el conjunto de datos *ImageNet* [31] en 2012, formado por imágenes etiquetadas en 1000 posibles clases, superando por un amplio margen a sus rivales de entonces. Está formado por 5 capas convolucionales con un *kernel* de tamaño  $11 \times 11$ . Esta arquitectura fue la primera en emplear capas *max-pooling*, funciones de activación *ReLU* y *dropout* como regularizador para evitar el problema del *overfitting*.

Posteriormente apareció VGG (*Visual Geometric Group*) [32], presentada en 2014, fue la primera arquitectura en demostrar que es posible aumentar, hasta cierto punto, el rendimiento de un modelo añadiendo más capas convolucionales. Además, fue la primera arquitectura con la que se planteó seriamente la normalización como un problema a abordar por las redes convolucionales.

Otra arquitectura relevante en el campo es *ResNet* [33], que surge con el objetivo de evitar el problema del gradiente desvaneciente. Esta arquitectura introdujo el concepto de red residual en el que se utiliza la técnica conocida como atajo en las conexiones que se salta el entrenamiento de algunas capas y se conecta directamente con la salida. De esta manera, es posible añadir muchas capas a la arquitectura sin perder rendimiento, pues si alguna capa perjudica el rendimiento de la arquitectura, los atajos aplican un efecto regularizador sobre la misma, evitando los problemas causados por el desvanecimiento del gradiente.

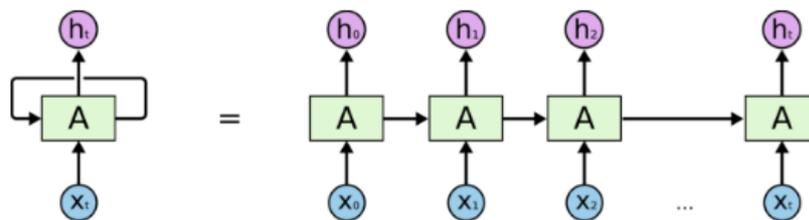
Además de las ya mencionadas, existen multitud de arquitecturas propuestas que siguen evolucionando con el paso del tiempo, mejorando de forma sustancial los resultados obtenidos respecto a la precisión lograda con el conjunto de datos *ImageNet*. La evolución de estas arquitecturas hasta 2023 puede verse representada en la figura 3.5, donde es interesante observar como la precisión ha crecido alrededor de un 30% en cuestión de 10 años, lo cual es un indicador de lo activa que es esta línea de investigación en la actualidad y de su importancia.



**Figura 3.5** Evolución de las arquitecturas convolucionales hasta 2023 [34]

### 3.4 Redes neuronales recurrentes

Las redes de neuronas clásicas presentan una limitación cuando tratan de resolver problemas en donde los datos ocultan patrones temporales o implican secuencias de algún tipo, como ocurre con el reconocimiento del habla, los indicadores económicos o los registros meteorológicos. De esta manera surgieron las redes de neuronas recurrentes [35], que dotan a las redes de neuronas con la capacidad de almacenar información proveniente de entradas anteriores. En este sentido, se podría decir que estas redes de neuronas especiales tienen una “memoria” que imita al cerebro.



**Figura 3.6** Representación de una red neuronal recurrente [36]

La primera característica propia de este tipo de redes es que las neuronas están conectadas a sí mismas como se puede ver en la Figura 3.6, donde además se muestra que equivale a desenrollar el bucle en capas individuales o pasos de tiempo diferentes dentro de la red de neuronas. La segunda característica propia de este tipo de redes es que los parámetros de cada capa de la red son compartidos, no como las redes de neuronas alimentadas hacia adelante, en las que cada neurona tiene un peso diferente en las conexiones.

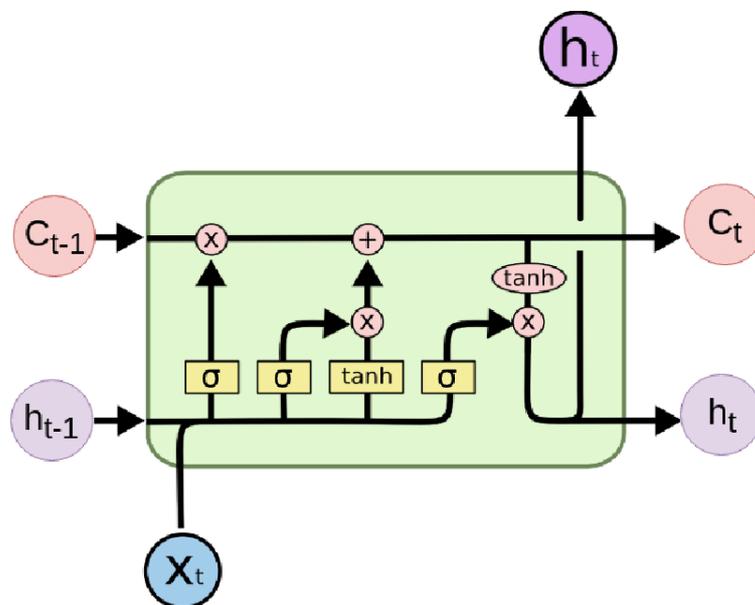
Existen diferentes tipos de redes de neuronas recurrentes. Entre ellos se encuentran las redes recurrentes bidireccionales, que no solamente trabajan con muestras de los datos anteriores, sino que además tratan con muestras futuras. También destacan las *Long Short-Term Memory (LSTM)* [37] y las *Gated Recurrent Units (GRUs)* [38] que son similares entre sí, puesto que tratan de resolver la misma limitación que presentan las redes recurrentes. El problema de las dependencias a largo plazo, que viene dado por el desvanecimiento del

gradiente en este tipo de redes. Cabe ampliar que las GRUs son una simplificación de la estructura interna de las redes LSTM.

### 3.4.1 Redes neuronales LSTM

Las redes de neuronas *Long Short-Term Memory* (LSTM) [37] son un subconjunto de los tipos de redes dentro de las redes de neuronas recurrentes que fueron introducidas por primera vez en 1997 y surgieron como una posible solución al problema que presentaban las redes de neuronas recurrentes cuando tratan de tener en cuenta las dependencias a largo plazo que se presentan en los datos.

Las redes LSTM tienen celdas en las capas ocultas de la red, que están formadas por tres estructuras que se denominan puertas, una de olvido, otra de entrada y una última de salida. Estas puertas sirven para regular y controlar el flujo de información necesario para predecir la salida en la red. Su fin es tratar de paliar el desvanecimiento del gradiente que provoca el problema de las dependencias a largo plazo.



**Figura 3.7** Representación de un bloque LSTM [39]

Representadas como un rectángulo amarillo en la Figura 3.7, contienen una función sigmoide en su interior que ayuda a mantener u olvidar la información de los datos.

Las puertas suelen ser representadas matemáticamente con la letra  $\Gamma$  y se definen como viene expresado en la siguiente ecuación, donde  $\sigma$  es la función sigmoide y  $W$  y  $b$  son coeficientes específicos de la puerta.

$$\Gamma = \sigma(Wx_{(t)} + h_{(t-1)} + b)$$

Siguiendo con la Figura 3.7, la entrada que recibe la primera puerta (la puerta de olvido) es la suma del estado oculto anterior  $h_{t-1}$ , y los datos de la entrada en el paso de tiempo actual,  $x_t$ . La salida de esta puerta es un vector cuyos valores se encuentran entre 0 y 1 que representan cuánto se quiere olvidar o mantener la información de datos anteriores. De esta manera, la red, en esta puerta, se entrena con el objetivo de que la salida sea cercana a 0 cuando una componente

de entrada se considere irrelevante y más cercana a 1 cuando ocurra lo opuesto. Esta salida se multiplica bit a bit con el estado de la celda anterior  $C_{t-1}$ , con el objetivo de reducir u olvidar la relevancia de componentes de los datos que sean prescindibles y no se necesitan a largo plazo, así como reforzar y mantener los que sí son necesarios.

La puerta de entrada involucra, además de la función sigmoide de nuevo, la función tangente hiperbólica. Como en la puerta de olvido, ambas reciben como entrada  $h_{t-1}$  y  $x_t$ . El objetivo de esta puerta es determinar qué nueva información se debe añadir al estado de la celda actual,  $C_t$ . Para ello, se combina  $h_t$  con  $x_t$  para generar un nuevo vector que actualice el estado de la celda teniendo en cuenta el estado oculto anterior. De esta manera se indica cuanto hay que actualizar cada componente en función de los nuevos datos de entrada.

El sentido de aplicar la función tangente hiperbólica para esto tiene que ver con que permite la posibilidad de obtener valores negativos, ya que puede ocurrir que la actualización sea para reducir la importancia de un componente en el estado de la celda. La función sigmoide en este caso se interpreta como una forma de comprobar si realmente es necesario recordar los nuevos datos de entrada. Por último, las salidas producidas por la función sigmoide y la función tangente hiperbólica se multiplican bit a bit, pues deben combinarse para que el proceso descrito anteriormente tenga sentido, y se suman al estado de la celda en ese paso de tiempo, alterado previamente por la puerta de olvido como ya se ha descrito.

Finalmente, la puerta de salida, que sirve para establecer el valor del nuevo estado oculto,  $h_t$ , utilizando el estado de la celda actual  $C_t$ , el valor del estado oculto anterior  $h_{t-1}$  y la entrada actual  $x_t$ . La función de activación, al igual que en la puerta de olvido, es la sigmoide. La salida que produce esta función se multiplica bit a bit con el valor del estado de la celda tras aplicarle la función tangente hiperbólica y, como resultado, se obtiene el valor del nuevo estado oculto. De forma intuitiva, este paso se puede entender como una manera de dejar pasar únicamente la información estrictamente necesaria hacia la siguiente celda.

### **3.5 Simulación Montecarlo**

El método Montecarlo o simulación Montecarlo es una técnica matemática que se utiliza para estimar los posibles resultados de un suceso incierto. Este método es una simulación de probabilidad múltiple que proporciona una serie de ventajas frente a los modelos predictivos con entradas fijas. La simulación Montecarlo fue inventada durante la Segunda Guerra Mundial por John Von Neumann y Stanislaw Ulam para mejorar la toma de decisiones en condiciones de incertidumbre [40].

A diferencia de un modelo de previsión normal, la simulación Montecarlo no se basa en un conjunto fijo de valores de entrada, sino que predice un conjunto de resultados posibles basándose en un rango estimado de valores. Esta técnica se utiliza para simular sistemas complejos y se aplica en una amplia variedad de áreas, como la ingeniería, la física, la economía y la biología.

La simulación Montecarlo utiliza una distribución normal para cualquier variable que tenga una incertidumbre inherente. En otras palabras, una simulación Montecarlo crea un modelo de resultados posibles aprovechando

una distribución normal para cualquier variable que tenga una incertidumbre inherente. A continuación, vuelve a calcular los resultados repetidamente, utilizando cada vez un conjunto diferente de números aleatorios entre los valores mínimo y máximo. En un experimento típico de Montecarlo, este ejercicio puede repetirse miles de veces para generar un gran número de resultados probables [41].

Las simulaciones Montecarlo también se utilizan para predicciones a largo plazo debido a su precisión. A medida que aumenta el número de entradas, el número de predicciones también crece, lo que permite proyectar los resultados más lejos en el tiempo con más precisión. Cuando finaliza una simulación Montecarlo, proporciona un rango de posibles resultados con la probabilidad de que se produzca cada resultado. Esto se puede utilizar para tomar decisiones informadas en situaciones en las que existe una gran incertidumbre.

En el contexto del póker, la simulación Montecarlo se utiliza para simular las cartas que aún no han sido mostradas en las etapas de *pre-flop*, *flop* y *turn*. Puesto que es imposible saber con certeza qué cartas se tratan, se puede realizar la simulación miles de veces para generar un resultado estadístico de los resultados con mayor probabilidad y de esta forma calcular la probabilidad de victoria de la mano de cada jugador. Este enfoque puede ayudar a los jugadores a tomar decisiones informadas sobre cómo jugar su mano.

## 4 Diseño y solución propuesta

En este capítulo se describen tanto el diseño como la solución propuesta para el problema mencionado al inicio del documento. Se explicarán el funcionamiento del sistema, las herramientas seleccionadas, explicando el porqué de la selección de estas y como se obtuvieron los datos de entrenamiento.

### 4.1 Obtención de datos de entrenamiento

Al comienzo de la investigación se trató de utilizar *datasets* ya disponibles con fotografías de cartas, pero estos no se adecuaban a la necesidad del sistema ya que no estaban orientados a la forma en la que se reconocen las cartas en este trabajo (Sección 4.3). Por ello se optó por realizar una serie de fotografías para que el modelo dispusiese de datos de entrenamiento.

Se realizaron dos ciclos, para el primero se hicieron cincuenta fotografías de cada carta, así como del dorso, posteriormente se procesaron estas imágenes para obtener la esquina superior izquierda de cada una ya que es lo necesario como se explica en la Sección 4.3. Una vez lograda esa transformación era necesario redimensionar y clasificar cada carta en su carpeta para que el modelo las digiriese etiquetadas y con el tamaño necesario. Con este primer ciclo se observó que el modelo necesitaba de más datos ya que con 2650 imágenes no era suficiente por lo que se optó por un segundo ciclo.

Para este segundo ciclo se hizo uso de la función del sistema para realizar las fotografías, detectar y recortar las cartas, de esta forma todas las imágenes del segundo ciclo son en gran medida similares a las que podría generar el programa en un entorno real. También se clasificaron y redimensionaron para juntarlas con las del ciclo anterior. Con el aumento a 5300 imágenes, el doble, el modelo comenzó a comportarse correctamente.

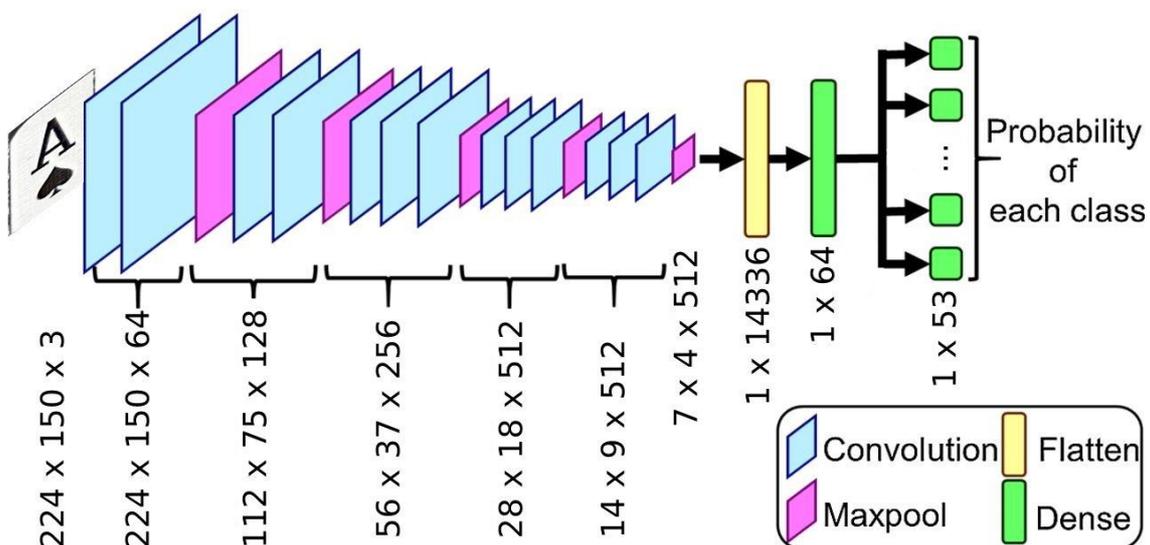
### 4.2 Diseño e implementación de la arquitectura

Para poder abordar la cuestión propuesta es necesario un buen diseño que se adapte a las necesidades del problema con el que se está trabajando, teniendo en cuenta que los datos de entrada son imágenes, las opciones disponibles son múltiples gracias a la constante investigación y avance en este campo. Hay que tener en cuenta ciertas restricciones relacionadas con la capacidad de cómputo, por lo tanto, las arquitecturas más complejas tienen que ser descartadas. También se conoce que las arquitecturas más simples, como variaciones de *LeNet-5* [28], son más veloces y aunque su capacidad de generalización es suficiente para el reconocimiento de imágenes, hay disponibles arquitecturas más sofisticadas que disponen de mejor rendimiento y eficiencia.

Durante el proceso de selección de la arquitectura que más se adecuaba al problema de reconocimiento de cartas, se experimentó con *VGG16*, *ResNet50* y *ResNet101* [42], puesto que son arquitecturas bastante populares que disponen de un equilibrio en su simplicidad y complejidad por los parámetros con los que cuentan, comparado con el resto de las arquitecturas publicadas hasta la fecha. Mientras se realizaban los experimentos con otras arquitecturas se podía apreciar como por ser más complejas no tenían por qué ser mejores, en cuanto a resolver este problema, de hecho, se manifestaba de forma clara con *ResNet101*, que no era capaz de generar entrenamientos comparables con otros

modelos propuestos, llegando a estancarse en precisiones de entre 45% y 55% aproximadamente, para el conjunto de datos seleccionado con el paso de las *epochs*. El modelo *ResNet50* se convirtió en descarte por la diferencia de tiempo de entrenamiento respecto a *VGG16*, aun teniendo ambos tiempos de entrenamiento relativamente cortos, *ResNet50* tardaba casi el doble por cada *epoch*, es por lo que se decidió descartarla.

Tras estos resultados iniciales, se llegó a la conclusión empírica que una buena arquitectura para comenzar con la resolución del problema era *VGG16*. Pese a que los resultados obtenidos con esta arquitectura sin ningún tipo de modificación eran similares a los que se pueden apreciar en el Capítulo 3.3.2 (alrededor de 75%), en cambio, modificando las dimensiones de la imagen de entrada para que esta sea rectangular y no se encuentre distorsionada aumentaba la precisión hasta un 93%. Por lo tanto, se modifica el tamaño de entrada inicial del modelo a 224x150 como se puede apreciar en la Figura 4.1 para obtener unos mejores resultados.



**Figura 4.1** Arquitectura VGG16 propuesta

También se han añadido las dos últimas capas densas, siendo la última densa con la función de activación *ReLU* con 53 neuronas, aunque sea evidente, una por cada clase necesaria para la salida; y la penúltima con función de activación *softmax* y contando con 64 neuronas. Se llegó a la conclusión de 64 neuronas después de realizar múltiples casos de prueba y error, ya que con más neuronas se dispone de más parámetros entrenables, en cambio los resultados de precisión disminuyen y aproximadamente a partir de las 256 neuronas se produce el fenómeno de *overfitting*. En cuanto al número de *epochs* que se utiliza es 10, este número ha sido obtenido haciendo uso de la función *EarlyStopping* de *Keras*, que nos permite parar el entrenamiento si no se generan mejoras entre un número definido de *epochs*, en este caso se seleccionaron tres. Con estos parámetros se obtienen unos resultados altamente positivos que están analizados en el Capítulo 5.

De esta forma, las características finales de la arquitectura pueden ser vistas descritas de forma detallada en la siguiente tabla, donde se tiene que las imágenes de entrada son de dimensiones 224x150 con 3 canales de entrada,

con las que se va a entrenar la red, que consta de 5 módulos convolucionales de 64, 128, 256, 512 y 512 filtros respectivamente. Todos los bloques hacen uso de una capa de *pooling* a la salida. Tras estos bloques se encuentra la capa densa *ReLU* de 64 unidades y finalmente, la capa densa para clasificar la entrada haciendo uso de la función *softmax* como se ha comentado previamente.

Capas de operación		Número de filtros	Tamaño del <i>kernel</i>	Tamaño de la salida
Imagen de entrada		-	-	224 x 150 x 3
Capa de convolución (2D)	<i>ReLU</i>	64	(3x3)	224 x 150 x 64
Capa de convolución (2D)	<i>ReLU</i>	64	(3x3)	224 x 150 x 64
Capa de <i>pooling</i>	Max <i>pooling</i>	-	-	112 x 75 x 64
Capa de convolución (2D)	<i>ReLU</i>	128	(3x3)	112 x 75 x 128
Capa de convolución (2D)	<i>ReLU</i>	128	(3x3)	112 x 75 x 128
Capa de <i>pooling</i>	Max <i>pooling</i>	-	-	56 x 37 x 128
Capa de convolución (2D)	<i>ReLU</i>	256	(3x3)	56 x 37 x 256
Capa de convolución (2D)	<i>ReLU</i>	256	(3x3)	56 x 37 x 256
Capa de convolución (2D)	<i>ReLU</i>	256	(3x3)	56 x 37 x 256
Capa de <i>pooling</i>	Max <i>pooling</i>	-	-	28 x 18 x 256
Capa de convolución (2D)	<i>ReLU</i>	512	(3x3)	28 x 18 x 512
Capa de convolución (2D)	<i>ReLU</i>	512	(3x3)	28 x 18 x 512
Capa de convolución (2D)	<i>ReLU</i>	512	(3x3)	28 x 18 x 512
Capa de <i>pooling</i>	Max <i>pooling</i>	-	-	14 x 9 x 512
Capa de convolución (2D)	<i>ReLU</i>	512	(3x3)	14 x 9 x 512
Capa de convolución (2D)	<i>ReLU</i>	512	(3x3)	14 x 9 x 512
Capa de convolución (2D)	<i>ReLU</i>	512	(3x3)	14 x 9 x 512
Capa de <i>pooling</i>	Max <i>pooling</i>	-	-	7 x 4 x 512
Capa densa	<i>ReLU</i>	-	-	64
Capa densa	<i>softmax</i>	-	-	53

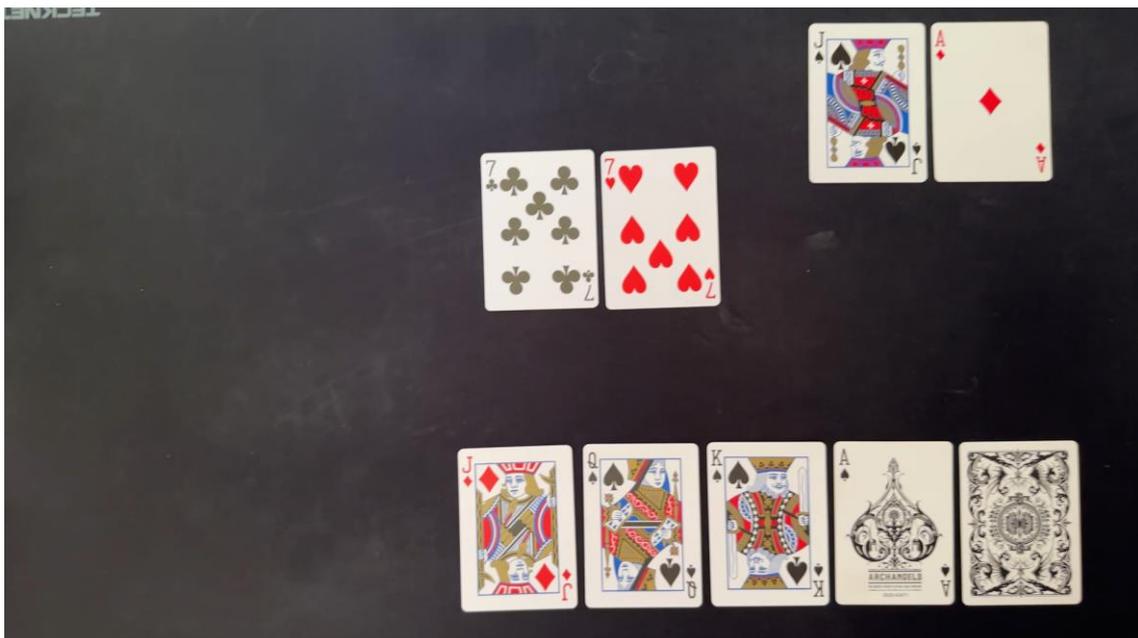
### 4.3 Funcionamiento del sistema

El sistema cuenta con un funcionamiento secuencial y está dividido en tres módulos, cada uno de ellos desarrolla una función en específico y se apoya en las salidas de otros para realizar sus funciones. Se dispone de dos formas de ejecución, siendo *python camera.py* la forma principal y la que se explicará en detalle a continuación, aunque también dispone del *flag --arg* que nos permite introducir una imagen como argumento de entrada. Esta funcionalidad sirve en el caso de que no dispongamos de una cámara en ese preciso instante o se quiera analizar una jugada ya pasada de la que disponemos una fotografía.



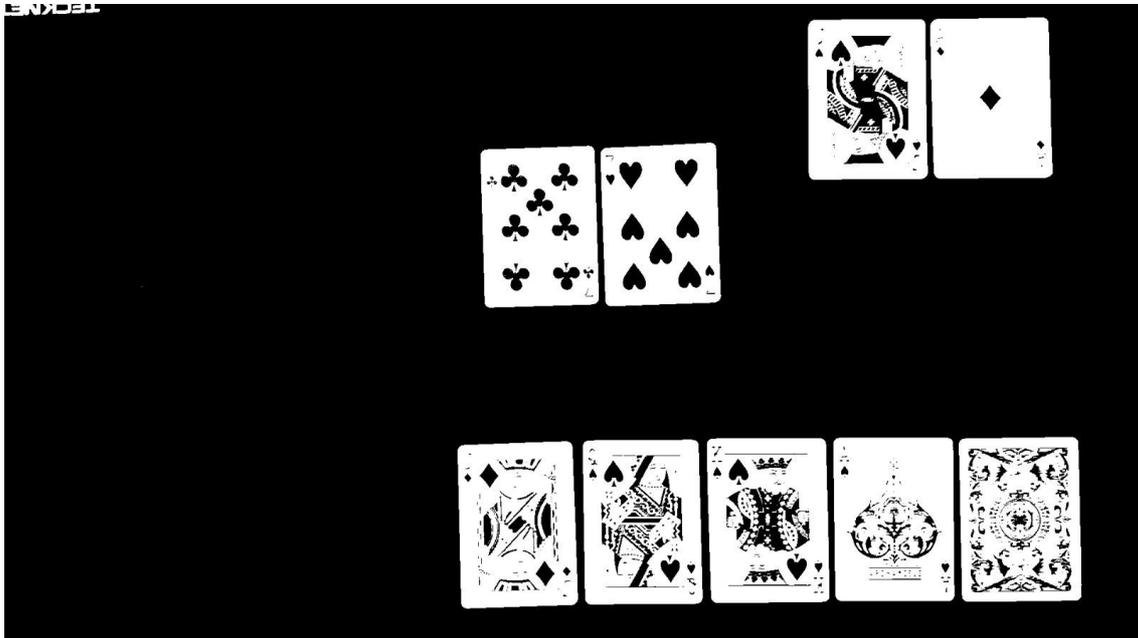
**Ilustración 4.1** Pantalla del sistema

El primer módulo es el encargado de arrancar el programa e iniciar el *frame* de tamaño 1920x1080 con la cámara, la Ilustración 4.1 es un ejemplo de lo que se puede observar. Este *frame* se encuentra en un bucle constante que comprueba si se presiona la tecla “q” para finalizar el programa o la “p” para tomar una instantánea de lo que ocurre en la cámara e iniciar la secuencia de juego. Esta imagen se guarda en el directorio de imágenes con el nombre *table\_n.png* (siendo n el número de jugada desde que se arrancó el programa), ya que en caso de querer realizar el estudio de nuevo o querer guardar la jugada el usuario podrá hacerlo con gran comodidad. Una vez pulsada la tecla “p” la imagen se pasa como parámetro al segundo módulo, el más complejo ya que es el encargado de reconocer las cartas, la Ilustración 4.2 refleja el tipo de imagen que se envía al siguiente módulo.



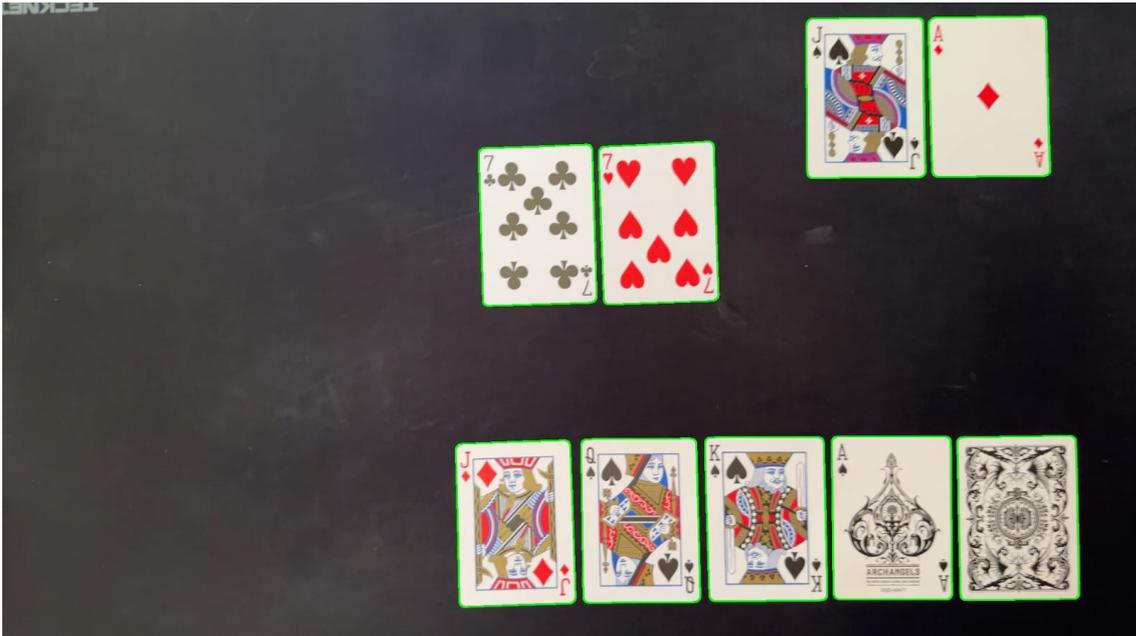
**Ilustración 4.2** Imagen tomada por el primer módulo

Una vez el segundo módulo recibe la imagen de entrada con la que trabajara esta debe ser preprocesada por lo que se le realiza una conversión a escala de grises y un *threshold* de tipo binario y otsu [43], esto se hace para binarizar la imagen, es decir, convertir todos los píxeles a negro o blanco, obteniendo el resultado de la ilustración 4.3.



**Ilustración 4.3** Imagen binarizada

Una vez procesada la imagen se obtienen los contornos y se itera sobre ellos para eliminar todos aquellos que pertenecen tanto al interior de las cartas como al exterior ya que solo nos interesan los contornos que recubren la carta para trabajar sobre estos. Esto se hace mirando el tamaño del contorno primero para descartar directamente todos aquellos que son pequeños y en caso de que se consideren suficientemente grandes se obtiene el número de esquinas del polígono y se comprueba que sea igual a cuatro.



**Ilustración 4.4** Contornos exteriores de las cartas

Con los contornos de las cartas encontrados se itera sobre ellos. En primer lugar, hay que comprobar si la carta está rotada en algún sentido ya que se necesita que esta se encuentre recta para realizar el “recorte”. Para comprobar esto se obtiene una caja con los puntos del contorno y se endereza con la función *minAreaRect* de *OpenCV*, también se comprueba que ningún parámetro supere los  $45^\circ$ , ya que en caso contrario la carta estará rotada horizontalmente y nos interesa que su disposición sea vertical. En este caso se rota  $90^\circ$  para obtener la imagen que se buscaba en un principio, como se observa en la Ilustración 4.5.



**Ilustración 4.5** Carta recortada y orientada

Tras varias aproximaciones se llegó a la conclusión de que independientemente de si la carta tuviese figura o no, lo más sencillo a la hora de reconocer era quedarse con la esquina superior izquierda de la carta ya que era limpia y precisa. De esta forma se recorta la esquina y se redimensiona a  $150 \times 224$

píxeles para que el modelo pueda ingerirla. En el caso de que se trate de un dorso no nos importa que sea la esquina o no por lo que se puede seguir usando este método y dado que el modelo esta entrenado con esta esquina la reconocerá. Se pueden observar los ejemplos en la Ilustración 4.6



**Ilustración 4.6** Esquina superior izquierda de la carta detectada

Una vez se logra la esquina, esta se envía al modelo para que la reconozca y el modelo devuelve una lista con la probabilidad de cada clase. Esta lista se decodifica siguiendo el orden de las clases y la probabilidad de la clase asociada, de esta forma se obtiene aquella clase con mayor probabilidad lo que nos indica la carta que reconoce. Las clases están nombradas de tal forma que la primera letra indica el palo de la carta en inglés, siendo D (diamantes), C (tréboles), H (corazones) y S (picas) los posibles valores; y el resto de la etiqueta el valor de la carta. En caso de detectar un dorso devuelve B de *back* y no lo añade a la lista de cartas detectadas.

Este proceso se realiza para todos los contornos y una vez se ha obtenido la lista con las cartas de la mesa se devuelve el resultado al primer módulo. Este módulo comprueba que no haya ninguna carta repetida o más cartas de las deseadas ya que el reconocimiento puede fallar, y en caso de que sea correcto separa las cartas en comunitarias y las dos manos. Esto es posible ya que las últimas dos cartas serán la primera mano siempre y las dos anteriores las de la segunda mano, es debido a como *OpenCV* ordena los contornos, por lo tanto, es importante que las jugadas siempre tengan la disposición que se muestra en la Ilustración 4.2. Una vez obtenidas las manos de los jugadores sabiendo las cartas comunitarias restantes se puede saber en qué etapa del juego se encuentra, obteniendo un estado similar al representado en la Ilustración 4.7.



**Ilustración 4.7** Cartas y estado del juego reconocido

Con las cartas separadas en los distintos grupos se pasa como parámetro tanto las dos manos como las cartas comunitarias al módulo tres, que es el encargado de realizar la simulación de Montecarlo para obtener las probabilidades de victoria o empate.

Esta simulación sigue las reglas del juego, cada jugador recibe dos cartas boca abajo llamadas “cartas de bolsillo”. Luego, se reparten cinco cartas comunitarias boca arriba en el centro de la mesa. Estas cartas comunitarias se dividen en tres rondas: el *flop* que es cuando se reparten tres cartas comunitarias; el *turn*, que es cuando se reparte la cuarta carta comunitaria; y el *river*, que es cuando se reparte la quinta y última carta comunitaria. Los jugadores pueden usar cualquiera de las cartas de la comunidad junto con sus cartas de bolsillo para formar la mejor mano de cinco cartas posible. Los jugadores pueden usar ambas cartas de bolsillo, una carta de bolsillo y cuatro cartas comunitarias, o incluso todas las cinco cartas comunitarias para formar su mano.

Por ello hay que tener en cuenta la etapa del juego en la que se encuentra para realizar las distintas combinaciones. Se ha optado por generar una baraja y en cada simulación “repartir” las cartas necesarias para completar las cinco cartas comunitarias en caso necesario. Una vez se tienen todas las cartas se generan las distintas combinaciones para quedarse con la mejor mano posible, siguiendo el orden establecido por las normas [44]. Cuando se tiene la mejor mano de cada jugador se compara y en caso de ganar se añade un punto a su cuenta particular o en caso de empate se añade al contador de empates. Si en un principio se disponía de todas las cartas comunitarias se para la simulación ya que no va a cambiar el resultado nunca, en caso contrario se realiza la misma operación hasta finalizar todas las simulaciones. Con el resultado final se divide cada cuenta entre el total de simulaciones y de esta forma se obtiene la probabilidad de victoria o empate aproximada de cada mano. Este resultado se envía de nuevo al primer módulo que muestra por pantalla el resultado final, Ilustración 4.8, y espera a que se pulse una tecla para esperar por otra jugada.



**Ilustración 4.8** Resultado final tras simulación

## 4.4 Lenguaje y entorno de desarrollo

El lenguaje de programación seleccionado para el desarrollo e implementación del sistema ha sido *Python* [45], puesto que cuenta con una gran variedad de librerías que ofrecen las cualidades necesarias para el desarrollo de aplicaciones de *deep learning* y uso de cámaras, aspectos fundamentales para el proyecto. Además, *Python* es el lenguaje utilizado mayoritariamente para la implementación de modelos de aprendizaje similares a los mencionados previamente.

En cuanto al entorno de desarrollo se ha optado por hacer uso de una máquina local para el entrenamiento del modelo ya que el número de operaciones no era tan elevado y se dispone de recursos suficientes. También se ha hecho uso de la cámara de un dispositivo *iPhone* para la toma de las fotografías y su uso en la versión final del sistema por su gran compatibilidad con la librería seleccionada para la gestión de imágenes. En resumen, el entorno de desarrollo cuenta con las siguientes características:

- 16 GB RAM
- AMD Ryzen 7 1700 Eight-Core Processor 3.20 GHz
- NVIDIA GeForce GTX 1070 8GB GDDR5
- Cámara de 12 Mpx: 26 mm, apertura de  $f/1,5$

Para contar con un control de versiones y seguridad a la hora de mantener los datos del trabajo se ha hecho uso de un repositorio público en *Github*, donde se puede encontrar todo el código del trabajo [46] y de *Google Drive* para almacenar las fotografías usadas para el entrenamiento del modelo, así como, para este documento.

Finalmente, se considera relevante el modelo de baraja que se usa durante todo el trabajo, se trata de una baraja de la marca, concretamente el modelo *Archangels* ya que para otras barajas el modelo no reconoce los dorsos y si los diseños de tanto los palos como los valores son diferentes también fallará

## 4.5 Selección de librerías y recursos

Para el manejo general de matrices y números en general se ha seleccionado el paquete *numpy* [47] por su eficiencia tanto en tiempo de ejecución como en gestión de memoria de las operaciones que computa, también por ser el módulo más extendido para las operaciones en *Python*.

Dentro de las librerías para el manejo de imágenes se ha utilizado *OpenCV* [48] por ser potente y versátil para el procesamiento de imágenes y visión por computador, con una amplia gama de funciones, documentación detallada, compatibilidad multiplataforma y un alto rendimiento.

Para el desarrollo y entrenamiento del modelo de *deep learning* se ha elegido la biblioteca *Keras* debido a su interfaz de bajo nivel, que permite la modificación de la implementación de los algoritmos y la optimización del entrenamiento, especialmente en modelos de aprendizaje profundo donde el ajuste fino de los hiperparámetros es fundamental y debe estar disponible.

*Keras* [49] está construido sobre *TensorFlow* [50], lo que agrega complejidad al manejo y carga de datos en GPU y TPU. Sin embargo, debido a que se requiere un alto rendimiento al entrenar modelos de redes neuronales que implican operaciones matriciales costosas, se considera una opción preferente. La principal alternativa a *Keras* es la librería *PyTorch* [51], que también es una opción popular en el ámbito del *deep learning*.

Tanto *Keras* como *PyTorch* son proyectos de código abierto, pero la principal diferencia entre ellos es que *PyTorch* es más adecuado para trabajar con modelos neuronales que reciben entradas de longitud variable, lo cual no es relevante para el trabajo en cuestión. En términos de curva de aprendizaje, *PyTorch* puede ser un poco más elevado que *Keras* debido a su mayor flexibilidad y capacidad para personalizar el modelo. Sin embargo, *Keras* cuenta con una documentación y comunidad mucho más grande y amplia en comparación con *PyTorch*, lo que facilita enormemente su uso y aprendizaje.

También se encuentran disponibles otras opciones como *R*, *MATLAB* o *Julia* entre otros, que podrían tenerse en cuenta, aunque implicarían el uso de otro lenguaje de programación y dificultaría la integración con el resto de los módulos del sistema.

Se utilizó la librería *Matplotlib* [52] para representar visualmente la evolución de los modelos y sus curvas de aprendizaje, entre otras figuras. *Matplotlib* es una biblioteca que permite crear visualizaciones personalizadas en 2D de forma sencilla y eficiente.

Para la creación de diagramas y figuras ilustrativas que sirven como apoyo a la explicación, se ha utilizado la herramienta *Photopea* [53], se trata de una versión web de código abierto muy similar al editor de fotografía *Photoshop*.

Finalmente, los módulos restantes importados a lo largo del código son nativos del propio lenguaje *Python* y de uso general, y no se considera relevante la mención específica de todos ellos.

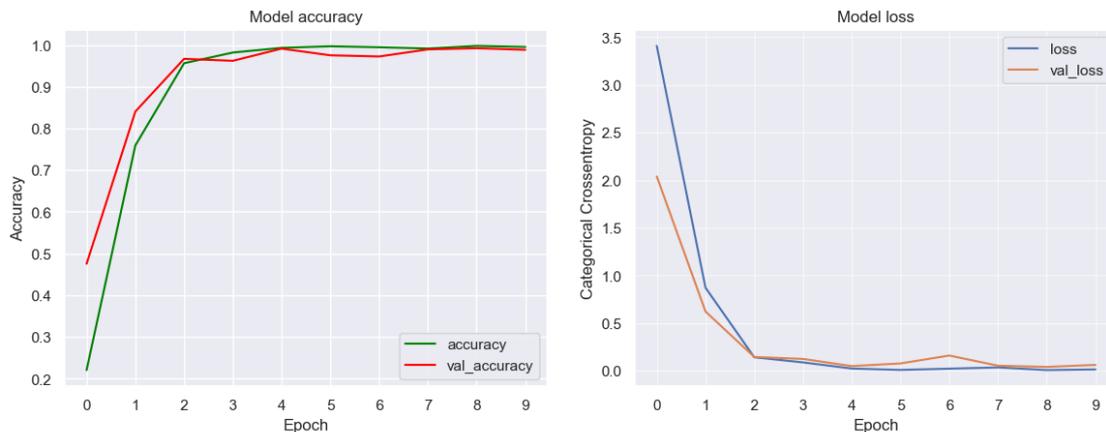
## 5 Resultados experimentales

En este capítulo se tratan los resultados obtenidos tras la realización del sistema y la resolución del problema propuesto. Se hace hincapié en la precisión del modelo y en los resultados obtenidos tras las simulaciones de Montecarlo.

### 5.1 Precisión del modelo

Como ya se ha mencionado, se ha seleccionado el modelo *VGG16*, en este caso todas las clases deben tener los mismos pesos ya que la probabilidad de que salga cualquier carta es siempre una entre el número de cartas restantes se ha optado por usar los pesos de *ImageNet*. El optimizador utilizado es una versión mejorada del *Stochastic Gradient Descent* (SGD) [54], se trata de *Adam* [55], un método basado en la estimación adaptativa de momentos de primer y segundo orden. La elección de este optimizador está basada en que es el que se usa en gran parte de los trabajos previos en esta área de investigación, ya que ayuda a que la función de pérdida realice una convergencia en los mínimos si se deja el modelo entrenando durante un tiempo prolongado, tal y como se menciona en el trabajo de *Wilson et al* [56]. La función de pérdida que se usa es la *categorical\_crossentropy* que ofrece *Keras*.

Los resultados obtenidos tras las ejecuciones iniciales con la adición de las dos capas densas y limitando el modelo a 10 *epochs* tras un análisis de este, alcanzando aproximadamente el 99.9% de precisión con el conjunto de validación, como se puede observar en la Figura 5.1.



**Figura 5.1** Evolución de la precisión de entrenamiento y validación, así como, de los valores de la función de pérdida

Para ratificar estos resultados y comprobar si se debían al azar o la capacidad y robustez del modelo, se realizaron un total de 50 ejecuciones diferentes con la misma arquitectura y configuración. Se obtuvo en prácticamente todas las ejecuciones el mismo resultado, no bajando en ningún momento del 99% de precisión. El tiempo de entrenamiento del modelo es de aproximadamente dos minutos y medio.

## 5.2 Matriz de confusión

En la Figura 5.2 se muestra la matriz de confusión generada tras realizar el entrenamiento del modelo sobre el conjunto de datos proporcionado. Pese a que es una matriz de rango 53 y puede ser complicado visualizar ciertos parámetros, en ella se pueden hacer una serie de observaciones. Pero antes es necesario explicar cómo se lee esta matriz. En los ejes se encuentran las clases, en el horizontal el valor entregado y en el vertical el valor predicho, es por ello por lo que cualquier valor fuera de la diagonal se considera un fallo.

Ahora que ya se conoce el funcionamiento de la matriz se puede realizar un análisis. La característica más llamativa es que casi todos los parámetros de la diagonal principal son 100% esto significa que el modelo es capaz de acertar todos los casos para la carta de esa celda. Pero en algunos valores falla, esto se debe a varios factores, siendo los principales la iluminación de la imagen, mismos valores pero de palos distintos aunque del mismo color y similitud entre los números, siendo el as y el cuatro los más relevantes. Esto se podría solucionar añadiendo más imágenes a la fuente de entrenamiento ya que la iluminación es el principal valor que genera interferencias en la predicción y al añadir más imágenes con iluminaciones distintas se podría mejorar, pese a que son unos resultados realmente positivos.

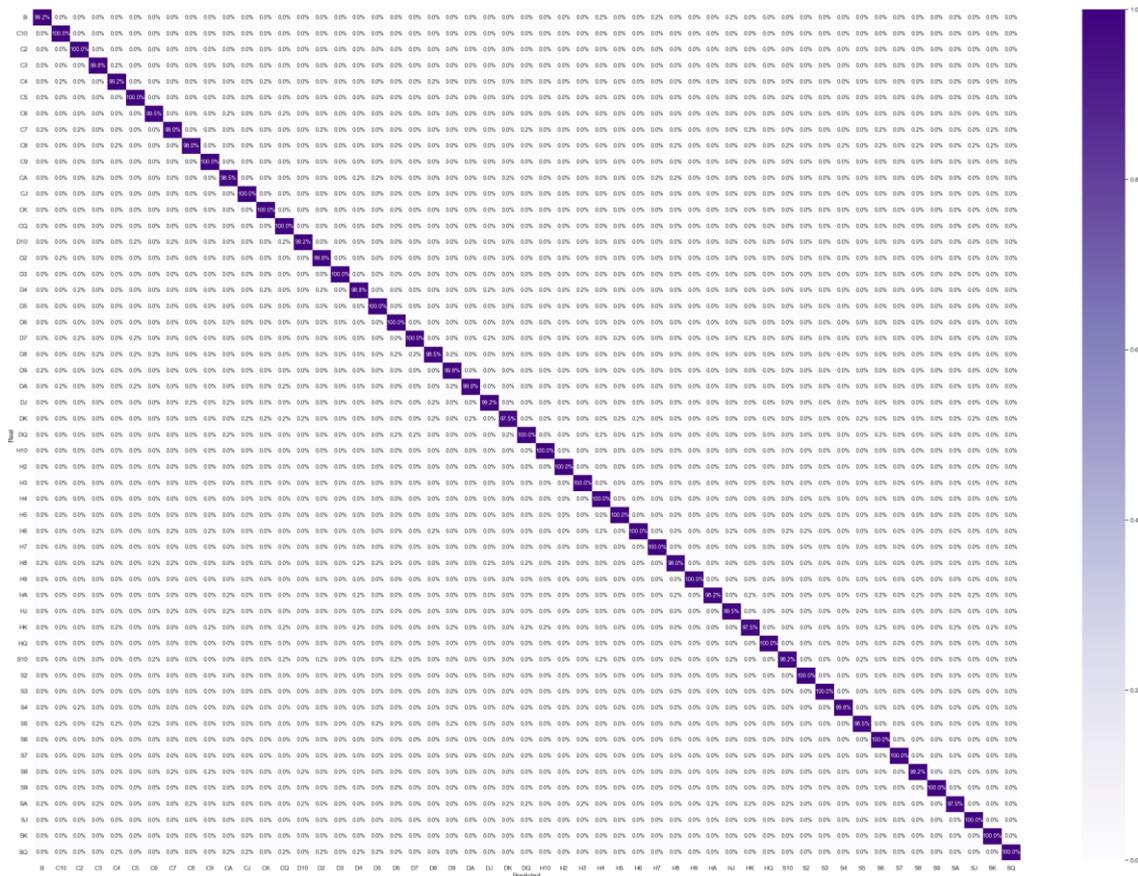


Figura 5.2 Matriz de confusión

### 5.3 Simulación de Montecarlo

Se han realizado diferentes simulaciones de Monte Carlo para obtener resultados precisos y correctos. En un principio se llevaron a cabo 1.000 simulaciones, pero después de analizar los resultados se observó que no eran consistentes. Para mejorar la precisión de las simulaciones, se decidió aumentar el número de simulaciones por diez, lo que resultó en 10.000 simulaciones por mano. Aunque la precisión mejoró considerablemente, la desviación típica aún era demasiado alta. Por lo tanto, se optó por realizar 100.000 simulaciones por mano, lo que permitió obtener resultados precisos y consistentes. Para validar los resultados obtenidos, se compararon con los resultados obtenidos por una calculadora de probabilidades *online* [57] , lo que confirmó la precisión de las simulaciones. La Ilustración 5.1 muestra que el sistema considera las mismas probabilidades que la calculadora, tal y como se puede observar en la esquina superior izquierda de la imagen y en los recuadros de la imagen derecha.



**Ilustración 5.1** Comparativa de la imagen de salida tras la simulación con los resultados proporcionados por la calculadora

## 6 Conclusiones y vías futuras

A lo largo de este trabajo se ha investigado sobre las distintas opciones que se dispone para la resolución del problema de análisis de mesas de póker a partir de imágenes utilizando técnicas de visión computacional. Se ha implementado un sistema de tres módulos que analiza la imagen entregada, la procesa y obtiene las distintas cartas mediante distintas funciones de preprocesamiento, y estas se entregan al modelo *VGG16* entrenado con las imágenes del *dataset* que se ha generado específicamente para el trabajo. Este modelo tiene una precisión de aproximadamente 99%, con ello se genera un listado y devuelve la predicción al módulo uno. Posteriormente estas predicciones pasan al tercer módulo que realiza la simulación necesaria para obtener las probabilidades de victoria de los cuales se obtienen resultados favorables.

Los resultados obtenidos de este trabajo son favorables en líneas generales, pese a que aún hay un gran margen de mejora. Para comenzar, en el primer módulo solo existe compatibilidad con cámaras de dispositivos *Apple* ya que la integración de *OpenCV* con estos es superior al resto de dispositivos. Por lo tanto, en un futuro se podría implementar una opción de arranque para seleccionar el tipo de dispositivo que se conecta al sistema, de esta forma se podría usar cualquier dispositivo con cámara y el sistema sería más compatible.

En cuanto al segundo módulo puesto que es el más importante también es en el que más mejoras se podría introducir. En primer lugar, se segmentar la imagen que cuenta con la carta recortada, de esta forma mirando los canales de color y analizándolos se podría distinguir si contiene rojo. Puesto que contamos únicamente con el color rojo o negro en las cartas esto ayudaría a descartar de antemano la mitad de las predicciones y ayudaría a descartar falsos positivos. En segundo lugar, como ya se ha mencionado se deberían incluir más imágenes en el conjunto de datos de entrenamiento ya que 5300 es un gran número, pero no es suficiente comparado con la magnitud de imágenes que se suelen manejar en estas áreas de investigación.

Introducir nuevas imágenes mejoraría las predicciones del modelo en condiciones adversas como podría ser iluminación tanto pobre como excesiva; también en el caso de que las imágenes se encuentren borrosas ya que al tomar la instantánea si no se hace uso de un trípode o similar, la imagen podría no salir nítida. Al introducir nuevas imágenes es probable que se deba modificar el número de neuronas de la capa densa con función *ReLU* o el número de *epochs* de entrenamiento, esto se debe a que los parámetros de entrada cambian. Por último, el modelo solo reconoce la baraja que se especifica en el Capítulo 4.4, es por ello que se podrían introducir nuevas imágenes de distintas barajas incluyendo estas imágenes en conjunto de datos de entrenamiento. Al tener más imágenes se debería aumentar siempre de forma balanceada ya que el modelo requiere del mismo número de imágenes para todas las clases.

El tercer módulo es el más lento de los tres, esto no afecta en gran medida ya que por la naturaleza del juego quince segundos es más que suficiente para analizar una mesa. Aunque siempre es mejor para el usuario mejorar el rendimiento y la eficiencia, es por esto, que se podrían paralelizar las simulaciones mediante el uso de concurrencia para reducir el tiempo de ejecución de este módulo.

Finalmente, se podría migrar todo el sistema a dispositivos móviles ya que aportaría una mejora significativa para los usuarios, aportando más movilidad y disponibilidad inmediata en cualquier ámbito. También sería necesario mantener la investigación en este campo y seguir realizando estudios más en profundidad sobre el tema puesto que es algo que avanza a pasos agigantados y se encuentra en constante cambio. Esto podría desembocar en un cambio en la arquitectura o en las técnicas de preprocesamiento de las imágenes.

## 7 Bibliografía

- [1] Special Guest, «How is technology influencing the online poker world?,» TMC, 29 July 2021. [En línea]. Available: <https://www.tmcnet.com/topics/articles/2021/07/29/449606-how-technology-influencing-online-poker-world.htm>.
- [2] Shawn, «Emerging Technology and the Changing Face of Poker,» DigitalConnectMag, 26 July 2022. [En línea]. Available: <https://www.digitalconnectmag.com/emerging-technology-and-the-changing-face-of-poker/>.
- [3] M. Davidson, «The Real Deal with RFID Cards and Poker,» Techspective, 5 November 2019. [En línea]. Available: <https://techspective.net/2019/11/05/the-real-deal-with-rfid-cards-and-poker/>.
- [4] R. J. Torraco, «Writing Integrative Literature Reviews: Using the Past and Present to Explore the Future,» 25 October 2016.
- [5] T. Y. K. W. a. J. Y. X. Hu, «Poker card recognition with computer vision methods,» de *2021 IEEE International Conference on Electronic Technology, Communication and Information (ICETCI)*, Changchun, China, 2021, 2021.
- [6] A. Gamba, «Arxiv,» 2 Enero 2023. [En línea]. Available: <https://arxiv.org/abs/2301.00505>.
- [7] J. Brownlee, «A Gentle Introduction to Object Recognition With Deep Learning,» 2022 May 2019. [En línea]. Available: <https://machinelearningmastery.com/object-recognition-with-deep-learning/>.
- [8] N. Kaleem, «Medium,» 14 February 2019. [En línea]. Available: <https://medium.datadriveninvestor.com/identifying-types-of-playing-cards-using-an-object-detection-classifier-fdd1bae02251>.
- [9] G. Sinha, «Deep Learning method for object detection: R-CNN explained,» Towards Data Science, 4 August 2020. [En línea]. Available: <https://towardsdatascience.com/deep-learning-method-for-object-detection-r-cnn-explained-ecdadd751d22>.
- [10] E. R. X. W. H. F. a. J. D. Q. Chen, «Poker Watcher: Playing Card Detection Based on EfficientDet and Sandglass Block,» de *2020 11th International Conference on Awareness Science and Technology (iCAST)*, Qingdao, China, 2020.
- [11] Na8, «Modelos de Detección de Objetos,» Aprende Machine Learning, 21 August 2020. [En línea]. Available:

<https://www.aprendemachinelearning.com/modelos-de-deteccion-de-objetos/>.

- [12] S. Pokhrel, «Xailient,» 10 February 2020. [En línea]. Available: <https://xailient.com/blog/6-problems-that-you-can-overcome-with-object-detection/>.
- [13] J. V. Rebaza, «Detección de bordes mediante el algoritmo de Canny,» Universidad Nacional de Trujillo , Trujillo, 2007.
- [14] «Matriz (matemática),» Wikipedia, [En línea]. Available: [https://es.wikipedia.org/wiki/Matriz\\_\(matem%C3%A1tica\)](https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1tica)).
- [15] «Histograma,» Wikipedia, 7 November 2017. [En línea]. Available: <https://es.wikipedia.org/wiki/Histograma>.
- [16] Marian, «Cómo hacer o insertar un histograma en word en pocos pasos,» Mira como se hace, 29 January 2020. [En línea]. Available: <https://miracomosehace.com/hacer-insertar-histograma-word/>.
- [17] A. M. E. Houssain, «Why do we use Gaussian function when fitting UV absorbance data?,» ResearchGate, [En línea]. Available: <https://www.researchgate.net/post/Why-do-we-use-Gaussian-function-when-fitting-UV-absorbance-data>.
- [18] P. Sharma, « A Step-by-Step Introduction to Image Segmentation Techniques,» Analytics Vidhya, 1 May 2023. [En línea]. Available: <https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/>.
- [19] P. B. H. L. A. M. S Kovács, «Seeded binary segmentation: a general methodology for fast and optimal changepoint detection,» *Biometrika*, vol. 110, n° 1, pp. 249-256, 2023.
- [20] D. Khattab, «Color Image Segmentation Based on Different Color Space Models Using Automatic GrabCut,» *Scientific World Journal*, pp. 1-10, 2014.
- [21] Wikipedia Colaborators, «Segmentacion,» Wikipedia, 2010. [En línea]. Available: [https://es.wikipedia.org/wiki/Segmentaci%C3%B3n\\_\(procesamiento\\_de\\_im%C3%A1genes\)](https://es.wikipedia.org/wiki/Segmentaci%C3%B3n_(procesamiento_de_im%C3%A1genes)).
- [22] S. Sahir, «Canny Edge Detection Step by Step in Python—Computer Vision,» Towards Data Science, 25 January 2019. [En línea]. Available: <https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123>.

- [23] Jesús, «Lo que necesitas saber sobre la detección de bordes,» Datasmarts, 30 June 2022. [En línea]. Available: <https://www.datasmarts.net/lo-que-necesitas-saber-sobre-la-deteccion-de-bordes/>.
- [24] automaticaddison, «How the Sobel Operator Works,» Automatic Addison, 17 December 2019. [En línea]. Available: <https://automaticaddison.com/how-the-sobel-operator-works/>.
- [25] Anon, «Redes neuronales profundas: qué son y cómo funcionan,» Psicologiamente, 6 November 2020. [En línea]. Available: <https://psicologiamente.com/cultura/redes-neuronales-profundas>.
- [26] K. Fukushima, «Neocognitron: A Self-organizing Neural Network Model,» Biological Cybernetics, Tokyo, Japan, 1980.
- [27] Y. Lecun, «Gradient-based learning applied to document recognition,» *Proceedings of the IEEE*, vol. 86, n° 11, pp. 2278-2324, 1998.
- [28] S. Saxena, «analyticsvidhya,» 30 March 2021. [En línea]. Available: <https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/>.
- [29] C. Urbina, «Convolución,» Blogspot, 22 August 2012. [En línea]. Available: <http://cecilia-urbina.blogspot.com/2012/08/convolucion.html>.
- [30] A. Pujara, «Medium,» Analytics Vidhya, 30 November 2020. [En línea]. Available: <https://medium.com/analytics-vidhya/concept-of-alexnet-convolutional-neural-network-6e73b4f9ee30>.
- [31] Standford Vision Lab, «IMAGENET,» 2020. [En línea]. Available: <https://www.image-net.org/>.
- [32] «Datagen,» [En línea]. Available: <https://datagen.tech/guides/computer-vision/vgg16/>.
- [33] X. Z. S. R. a. J. S. K. He, «Deep Residual Learning for Image Recognition,» 10 December 2015. [En línea]. Available: <https://doi.org/10.48550/arxiv.1512.03385..>
- [34] «Image Classification on ImageNet,» 1 May 2023. [En línea]. Available: <https://paperswithcode.com/sota/image-classification-on-imagenet>.
- [35] «¿Qué son las redes neuronales recurrentes?,» IBM, [En línea]. Available: <https://www.ibm.com/es-es/topics/recurrent-neural-networks>. [Último acceso: 1 May 2023].
- [36] alexis96, «Red neuronal recurrente,» Githubpages, [En línea]. Available: <https://alexis96.github.io/proyecto-RNN/>.
- [37] «Cómo usar redes neuronales (LSTM) en la predicción de averías en las máquinas,» blog.gft.com, 06 November 2018. [En línea]. Available:

<https://blog.gft.com/es/2018/11/06/como-usar-redes-neuronales-lstm-en-la-prediccion-de-averias-en-las-maquinas/>.

- [38] S. Kostadinov, «Understanding GRU Networks,» Medium, 10 November 2019. [En línea]. Available: <https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be>.
- [39] A. C. García, «Celda de una red neuronal recurrente LSTM,» ResearchGate, 2023. [En línea]. Available: [https://www.researchgate.net/figure/Celda-de-una-red-neuronal-recurrente-LSTM-Donde-los-valores-de-C-son-los\\_fig3\\_343450066](https://www.researchgate.net/figure/Celda-de-una-red-neuronal-recurrente-LSTM-Donde-los-valores-de-C-son-los_fig3_343450066).
- [40] «¿Qué es la simulación Monte Carlo?,» IBM, [En línea]. Available: <https://www.ibm.com/mx-es/topics/monte-carlo-simulation>. [Último acceso: 2 May 2023].
- [41] IBM, «¿Que es la simulación Montecarlo?,» IBM, [En línea]. Available: <https://www.ibm.com/es-es/topics/monte-carlo-simulation>.
- [42] Keras, «Keras Applications,» Keras, [En línea]. Available: <https://keras.io/api/applications/>.
- [43] S. Seth, «Thresholding with OpenCV,» LearnOpenCV, 5 August 2020. [En línea]. Available: <https://learnopencv.com/otsu-thresholding-with-opencv/>.
- [44] R. Corrigan, «Poker Hand Rankings & The Best Texas Hold'em Poker Hands,» Upswing Poker, 31 March 2023. [En línea]. Available: <https://upswingpoker.com/poker-hands-rankings/>.
- [45] Python, «Pyhton,» Python, [En línea]. Available: <https://www.python.org/>.
- [46] Á. A. Miguel, «PokerHandsRecognition,» Github, 2023. [En línea]. Available: <https://github.com/AlvaroAlonso-0/PokerHandsRecongition>.
- [47] Numpy, «NumPy,» [En línea]. Available: <https://numpy.org/>.
- [48] Opencv, «OpenCV,» [En línea]. Available: <https://opencv.org/>.
- [49] Keras, «Keras,» [En línea]. Available: <https://keras.io/>.
- [50] Tensorflow, «TensorFlow,» [En línea]. Available: <https://www.tensorflow.org/?hl=es-419>.
- [51] Pytorch, «PyTorch,» [En línea]. Available: <https://pytorch.org/>.
- [52] Matplotlib, «Matplotlib,» [En línea]. Available: <https://matplotlib.org/>.
- [53] Photopea, «Photopea,» [En línea]. Available: <https://www.photopea.com/>.

- [54] A. V. Srinivasan, «Stochastic Gradient Descent — Clearly Explained !!,» Medium, 7 September 2019. [En línea]. Available: <https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31>.
- [55] J. B. Diederik P. Kingma, «Adam: A Method for Stochastic Optimization,» Arxiv, 22 December 2014. [En línea]. Available: <https://arxiv.org/abs/1412.6980>.
- [56] R. R. M. S. N. S. B. R. Ashia C. Wilson, «The Marginal Value of Adaptive Gradient Methods in Machine Learning,» Arxiv, 22 May 2018. [En línea]. Available: <https://arxiv.org/abs/1705.08292>.
- [57] «Texas Hold'em Poker Odds Calculator,» CardPlayer, [En línea]. Available: <https://www.cardplayer.com/poker-tools/odds-calculator/texas-holdem>.
- [58] IBM, «What is Deep Learning?,» IBM, 2023. [En línea]. Available: <https://www.ibm.com/topics/deep-learning>.

Este documento esta firmado por



<b>Firmante</b>	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
<b>Fecha/Hora</b>	Tue May 30 16:29:26 CEST 2023
<b>Emisor del Certificado</b>	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
<b>Numero de Serie</b>	561
<b>Metodo</b>	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)