



UNIVERSIDAD DE MÁLAGA



Graduado en Ingeniería Informática

Identificación de tableros de ajedrez digitales mediante
inteligencia artificial aplicada a imágenes

Identification of digital chessboards through artificial
intelligence applied to images

Realizado por
Sergio Camacho Marín

Tutorizado por
Miguel Ángel Molina Cabello
Karl-Khader Thurnhofer Hemsi

Departamento
Lenguajes y Ciencia de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, agosto de 2023

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
INFORMÁTICA GRADUADO EN INGENIERÍA INFORMÁTICA

**Identificación de tableros de ajedrez digitales mediante
inteligencia artificial aplicada a imágenes**

**Identification of digital chessboards through artificial
intelligence applied to images**

Realizado por

Sergio Camacho Marín

Tutorizado por

Miguel Ángel Molina Cabello

Karl-Khader Thurnhofer Hemsi

Departamento

Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, MAYO DE 2023

Fecha defensa: septiembre de 2023

Abstract

This work is based on the development of a program capable of reproducing a digital chessboard from a captured image, a photo, or a snapshot of the same board at a specific moment during a game. The goal is to resume the game from that point without manually arranging the pieces. The program consists of two parts:

In the first part, the program detects the chessboard using Computer Vision techniques such as the Hough transform or the Canny edge detection algorithm. Once this process is completed, an image of the chessboard is obtained. In the second part, the program segments the chessboard, aiming to extract individual squares. The intention is to process these squares using a classifier built with machine learning technologies, in order to identify the specific chess piece present in each square as seen in the image. Subsequently, each square is labeled using a notation system called "Forsyth-Edwards Notation," thus obtaining the board's ID to allow for the continuation of the game.

Finally, comprehensive tests were conducted to assess the program's performance and accuracy. These tests involved the use of various chess board images under varying conditions. The program's ability to detect the board, segment the squares, and classify the pieces in different scenarios, including variations in lighting and potential visual obstacles, were analyzed. Furthermore, the system's capacity to handle potential errors and noise in the captured images has been evaluated. The goal was to ensure the program's reliability and precision in detecting and classifying a chess board correctly.

Keywords: Computer Vision, Convolucional Neural Networks, Chess

Resumen

Este trabajo se basa en la realización de un programa capaz de reproducir un tablero digital de una partida de ajedrez a partir de una imagen capturada, una foto o un recorte del mismo tablero en un momento concreto de una partida, con el objetivo de poder reanudar la partida desde ese punto sin tener que disponer las piezas de forma manual. El programa consta de dos partes:

En una primera parte, detecta el tablero mediante técnicas de Visión por Computador como son la transformada de Hough o el algoritmo de detección de bordes Canny. Cuando se completa este proceso, se obtiene una imagen del tablero. En la segunda parte, secciona el tablero, buscando la obtención de las casillas de manera individual, con el objetivo de procesarlas mediante un clasificador, construido con tecnologías de machine learning, y obtener la pieza concreta que se visualiza en la imagen. Posteriormente se denota cada casilla con el uso de un sistema de notación denominada “Forsyth-Edwars”, obteniendo así el ID del tablero para poder continuar la partida.

Finalmente, se llevaron a cabo pruebas exhaustivas para evaluar el rendimiento y la precisión del programa. Estas pruebas involucraron la utilización de diversas imágenes de tableros de ajedrez en condiciones variables. Se analizó cómo el programa detecta el tablero, segmenta las casillas y clasifica las piezas en diferentes situaciones, incluyendo variaciones en la iluminación y posibles obstáculos visuales. Además, se evaluó la capacidad del sistema para lidiar con posibles errores y ruidos en las imágenes capturadas. El objetivo fue garantizar la fiabilidad y precisión del programa para detectar y clasificar correctamente un tablero de ajedrez.

Palabras clave: Visión por computador, Redes neuronales convolucionales, Ajedrez

Índice

1. Introducción	7
1.1. Contexto	7
1.2. Motivación	7
1.3. Objetivos	8
1.4. Estructura del documento	9
1.5. Metodología de trabajo	10
1.6. Tecnologías utilizadas	12
2. Estudio del arte	15
2.1. Visión por Computador	15
2.1.1. Transformada de Hough	15
2.2. Inteligencia Artificial	16
2.2.1. Redes Neuronales	18
2.3. Redes Neuronales Convolucionales	19
3. Requisitos del sistema	21
3.1. Requisitos Funcionales	21
3.2. Requisitos No funcionales	22
3.3. Diagramas de Secuencia	23
3.4. Patrón de Diseño	29
4. Fase de obtención de datos	31
5. Fase de Desarrollo	37
5.1. Descripción general del proceso	37
5.2. Desarrollo del algoritmo de detección de tableros	38
5.2.1. Ideas Descartadas y Motivos de No Implementación	38
5.2.2. Implementación Final del Algoritmo de Detección de Tableros	42
5.3. Desarrollo del algoritmo de seccionamiento	47
5.4. Desarrollo del clasificador	48

5.5.	Desarrollo del algoritmo tipicador de notación FEN	60
6.	Fase de Pruebas	63
6.1.	Resultados del desarrollo	63
6.1.1.	Pruebas sobre algoritmo de detección	63
6.1.2.	Resultados del programa completo	68
7.	Conclusiones y Líneas futuras	75
7.1.	Conclusiones	75
7.2.	Líneas futuras	76
Apéndice A.	Apéndice: Manual de Instalación	79
Apéndice B.	Apéndice: Manual de Usuario	81
Apéndice C.	Apéndice: Glosario	85

1 Introducción

1.1. Contexto

El ajedrez es un juego de estrategia que se juega entre dos jugadores en un tablero cuadrulado conocido como tablero de ajedrez. Cada jugador controla un conjunto de 16 piezas, que se dividen en dos tipos: las piezas mayores, como el rey, la reina, las torres, los alfiles y los caballos; y las piezas menores, los peones. El objetivo principal del juego es lograr el jaque mate al rey del oponente, lo que significa que el rey está amenazado de captura y no puede moverse a una casilla segura.

El ajedrez se juega en un entorno de turnos, donde los jugadores alternan movimientos. Cada pieza tiene restricciones específicas de movimiento, lo que exige una planificación estratégica y táctica. La reina es la pieza más versátil, ya que puede moverse en líneas rectas y diagonales a cualquier distancia. Los alfiles se mueven en diagonales, las torres en líneas rectas y los caballos tienen un movimiento peculiar en forma de "L". Los peones se mueven hacia adelante pero capturan en diagonal.

La partida puede desarrollarse en diferentes fases, como la apertura, en la que se preparan las piezas para la acción; el medio juego, donde se lucha por la posición y la iniciativa; y el final de partida, donde se buscan oportunidades de mate o ventajas decisivas.

El ajedrez no solo es un juego de entretenimiento, sino también un deporte mental y una herramienta educativa. Ha sido estudiado a fondo por jugadores, analistas y computadoras, generando una rica historia de partidas famosas y teoría ajedrecística. Además, el ajedrez promueve habilidades como la toma de decisiones bajo presión, la concentración, el cálculo estratégico y la planificación a largo plazo.

1.2. Motivación

El aprendizaje constante asociado a los juegos de estrategia, como el ajedrez, y la incesantes ganas de evolución y competitividad derivado a sistemas complejos de jerarquías para asegurar enfrentamientos equitativos. Al mismo tiempo, existen novatos que dedican su tiempo a observar partidas y analizarlas para adquirir conocimientos. Sin embargo, replicar el tablero de juego puede volverse tedioso, especialmente cuando se están enfrentando jugadores

experimentados que mueven sus piezas con rapidez, presentando un desafío a este trabajo.

La solución radica en desarrollar un programa capaz de capturar la configuración precisa del tablero en momentos cruciales. Esto es valioso no solo para principiantes que desean aprender mediante la exploración de posibilidades y evolucionar en sus movimientos, sino también para maestros que buscan realizar análisis exhaustivos en momentos clave. Estos expertos contemplan maniobras alternativas que podrían cambiar el rumbo del juego, pero a menudo las limitaciones de tiempo durante la partida les impiden explorar a fondo estas opciones.

La motivación subyacente de este proyecto es mejorar la identificación del tablero de ajedrez, lo que implica reconocer la posición de cada pieza en el tablero. Esta mejora beneficia tanto a principiantes como a maestros, ya que automatiza esta tarea y les permite enfocarse en la práctica en lugar de invertir tiempo en reproducir la disposición del tablero de juego.

1.3. Objetivos

El objetivo fundamental de este proyecto reside en abordar los desafíos inherentes al aprendizaje y la participación en juegos de estrategia, en particular el ajedrez, a través de una solución tecnológica innovadora. Se busca crear un programa efectivo que tenga la capacidad de capturar y reproducir de manera precisa y automatizada el estado de un tablero de ajedrez en momentos cruciales durante una partida.

El programa se enfocará en dos aspectos fundamentales: en primer lugar, la detección y captura de la disposición exacta de las piezas en el tablero en un momento específico, utilizando avanzadas técnicas de Visión por Computador y procesamiento de imágenes. Esta tarea es esencial para garantizar la precisión y la fidelidad en la representación del tablero.

En segundo lugar, se pretende implementar un sistema de notación sofisticado, como la "Forsyth-Edwards Notation"^[1], para identificar y registrar de manera única la configuración de piezas en el tablero. Esto permitirá no solo la reproducción precisa del tablero, sino también el análisis profundo y la revisión de diferentes estrategias y movimientos por parte de los jugadores, ya sean novatos que desean aprender o expertos que buscan perfeccionar sus habilidades.

El éxito de este proyecto se medirá por la capacidad del programa para simplificar y agilizar el proceso de captura y reproducción del tablero, aliviando a los jugadores de la tarea manual de digitalizar la posición. Al lograr esto, se espera brindar una herramienta valiosa

para la educación y la mejora continua en el ámbito del ajedrez, permitiendo a los entusiastas de todos los niveles dedicar más tiempo al análisis estratégico y al juego real, en lugar de invertir tiempo en configuraciones manuales. En última instancia, el proyecto aspira a enriquecer la experiencia de juego y a fomentar un mayor aprendizaje en un ambiente accesible y tecnológicamente avanzado.

1.4. Estructura del documento

El contenido se ha dividido en secciones y subsecciones para facilitar la comprensión del problema junto con los resultados obtenidos. La estructura es la siguiente:

- Introducción: Aquí nos sumergimos en el mundo del ajedrez, destacando su importancia como juego y su popularidad a lo largo de la historia, la motivación que surge a partir de la búsqueda de cómo la tecnología puede mejorar el estudio y la práctica del ajedrez y los objetivos que se persiguen con este trabajo. Por otro lado, se incluyen la estructura del documento, la tecnologías y metodologías utilizadas en este trabajo.
- Estudio del arte: En este apartado, se tratan dos aspectos esenciales: cómo los jugadores registran los movimientos en una partida y por qué esta notación es de gran importancia. Además, exploramos conocimientos fundamentales que tienen un impacto directo en nuestro proyecto, como la rama especializada en Visión por Computador y la Inteligencia Artificial.
- Fase de obtención de los datos: En este apartado, exploramos en detalle cómo obtuvimos y procesamos los datos que desempeñan un papel crucial en las diversas pruebas relacionadas con la detección del tablero, así como en el proceso de entrenamiento, validación y pruebas del clasificador.
- Requisitos del sistema: Este apartado aborda los requisitos esenciales para crear el sistema, incluyendo tanto los funcionales como los no funcionales. También se presentan los modelos de interacción del usuario en base a esos requisitos y se describe el patrón de diseño utilizado.
- Fase de desarrollo y resultados: En este apartado, nos adentramos en la fase central de nuestro trabajo, donde se destacan los procesos clave. Aquí se detallan los pasos para

desarrollar los algoritmos de detección de tableros, la creación de los modelos clasificadores, así como el proceso de segmentación y corrección de la notación resultante. Además, compartimos los resultados de las pruebas realizadas sobre el algoritmo de detección, junto con los resultados del desarrollo del programa.

- Conclusiones y líneas futuras: En este apartado final, se tratarán las conclusiones que hemos extraído a lo largo de todo, junto con las posibles investigaciones derivadas de este trabajo.
- Apéndices: Se incluyen dos tipos de manuales, el primero es un manual de instalación del programa y el segundo es un manual de usuario, para un correcto uso del programa. Y el tercer apéndice es un glosario de palabras claves del proceso.

1.5. Metodología de trabajo

Durante la realización del trabajo se ha tenido en cuenta las siguientes metodologías:

- Método científico[2]: Es el método principal y fundamental en el que se basa este trabajo. Se basa en la observación, la experimentación y el análisis lógico con el objetivo de obtener conocimientos confiables y verificables sobre los fenómenos naturales. Aunque existen variaciones en la forma en que se presenta el método científico, generalmente sigue una serie de pasos interconectados:
 - Identificación del problema: Reconocer y definir de manera precisa un problema actual que afecte la vida cotidiana o identificar una brecha en el conocimiento científico.
 - Investigación del problema: Reunir y examinar exhaustivamente toda la información relacionada con el problema identificado.
 - Formulación de la hipótesis: Crear una suposición que podría resolver el problema o llenar el vacío en cuestión.
 - Verificación de la hipótesis: Realizar una serie de experimentos para confirmar la validez de la hipótesis propuesta, llegando a una conclusión sobre su certeza.

- Comunicación a la comunidad científica: Informar a la comunidad científica global sobre los resultados obtenidos durante la investigación.
- Métodología SCRUM[3]: Para la realización de este trabajo se ha empleado una variación de la metodología SCRUM, para adaptarse en su totalidad al proyecto. Se ha utilizado debido a que representa un marco de trabajo ágil utilizado para la gestión de proyectos, especialmente en el desarrollo de software.
Scrum se basa en la idea de dividir un proyecto en iteraciones cortas llamadas "sprints", generalmente de 2 a 4 semanas de duración. Durante cada sprint, se establecen las siguientes fases:
 - Planificación del Sprint: Al comienzo de cada sprint, el equipo Scrum se reúne para definir y seleccionar las tareas que serán realizadas durante el sprint.
 - Reuniones Diarias: Todos los días durante el sprint, el equipo tiene una breve reunión para compartir el progreso y discutir cualquier impedimento.
 - Desarrollo del Trabajo: Durante el sprint, el equipo trabaja en las tareas seleccionadas. El trabajo se divide en pequeñas unidades.
 - Revisión del Sprint: Al final de cada sprint, el equipo realiza una reunión de revisión en la que demuestra lo que se ha completado durante el sprint. Las partes interesadas pueden proporcionar retroalimentación y ajustar las prioridades en función de lo que se ha entregado.
 - Retrospectiva del Sprint: Despues de la revisión, el equipo realiza una retrospectiva en la que analiza lo que funcionó bien y qué se puede mejorar en términos de procesos, herramientas y colaboración. Esto ayuda a iterar y mejorar continuamente el proceso de trabajo.
- Metodología de implementación: Con el objetivo de evitar problemas en el desarrollo del programa o en su defecto tener conocimiento de dónde se encuentra, se ha seguido una metodología de implementación debido a sus características de modularidad, comprensividad y capacidad de ser modificable, entre otras. Este aspecto aporta al software características como la facilidad de mantenimiento, el aislamiento de los problemas, claridad y organización al código, y la reutilización del código.

1.6. Tecnologías utilizadas

Las tecnologías utilizadas en este proyecto se componen de tres elementos: lenguaje de programación, entorno de desarrollo, y librerías y frameworks utilizados.

Lenguaje de programación

El lenguaje utilizado para este proyecto es Python[4, 5]. Es un lenguaje de programación de alto nivel e interpretado, que destaca por su legibilidad y accesibilidad universal para programadores de distintos niveles. Con origen en 1991 y diseñado por Guido van Rossum, Python es una elección destacada para diversas aplicaciones, desde el desarrollo web y científico hasta la automatización y el scripting. Su simplicidad y facilidad de uso lo hacen idóneo para una amplia variedad de tareas. Además, su capacidad para manejar volúmenes considerables de datos lo convierte en una herramienta esencial en áreas como "Big Data" "Machine Learning".

Entorno de desarrollo

Este trabajo ha sido desarrollado con Visual Studio Code[6, 7] es un editor de código fuente desarrollado por Microsoft. Se trata de un entorno de desarrollo integrado ligero y altamente personalizable que se utiliza principalmente para escribir y depurar código en una variedad de lenguajes de programación.

Por otro lado, se ha optado por emplear el lenguaje de marcado de diseño conocido como LaTeX, para la realización del documento. A diferencia de las aplicaciones de procesamiento de texto convencionales, en las cuales el énfasis recae principalmente en la apariencia visual del documento, LaTeX se orienta hacia la estructura lógica y el contenido del mismo. En este contexto, la elección de la plataforma Overleaf ha sido deliberada, ya que ofrece un enfoque en línea que facilita la creación y edición colaborativa de documentos LaTeX. Esta plataforma posibilita a los usuarios redactar y modificar documentos LaTeX directamente desde su navegador web, evitando la necesidad de instalar y configurar un entorno LaTeX en su equipo informático.

Librerías y frameworks utilizados

Dentro de este apartado se enumeran las principales librerías que se han utilizado para el desarrollo de este trabajo. Son las siguientes:

- OpenCV[8]: Es una biblioteca de código abierto ampliamente utilizada para el procesamiento de imágenes y la visión por computador. Originalmente desarrollada por Intel en

1999, ha evolucionado en un proyecto de código abierto mantenido por una comunidad activa de desarrolladores. Escrita en C++, proporciona interfaces para varios lenguajes de programación, incluyendo C++, Python y Java. Sus capacidades abarcan desde el procesamiento de imágenes hasta la detección de objetos y características faciales, el reconocimiento de patrones, y el aprendizaje automático y profundo.

- NumPy[9]: Es una biblioteca esencial en Python para realizar cálculos numéricos y manipulación de matrices y arreglos multidimensionales. Proporciona una interfaz eficiente para operaciones matemáticas y estadísticas en grandes conjuntos de datos, siendo fundamental en el análisis de datos científicos y cómputo científico en general.
- Torchvision[10]: Es una biblioteca enfocada en el aprendizaje profundo y la visión por computadora dentro del ecosistema de PyTorch. Facilita la carga y transformación de datos de imágenes, así como la construcción y entrenamiento de modelos de redes neuronales convolucionales (CNN) y la evaluación de modelos en tareas de visión por computadora.
- PyTorch[11]: Este marco de trabajo de código abierto, creado por Facebook's AI Research lab (FAIR), es reconocido por su enfoque dinámico y su sencillez en el aprendizaje profundo. Construye gráficos computacionales dinámicamente y se basa en "tensores", estructuras de datos multidimensionales. Con capacidad para aprovechar tarjetas gráficas (GPU) y una conexión fluida entre CPU y GPU, es ideal para redes neuronales y aplicaciones de inteligencia artificial.
- TensorFlow[12]: TensorFlow es un marco de trabajo versátil y poderoso que permite construir, entrenar y desplegar modelos de aprendizaje automático y profundo en diversas aplicaciones y plataformas. Lanzado en 2015, se ha convertido en uno de los marcos más populares en inteligencia artificial y análisis de datos. Proporciona herramientas para la construcción y entrenamiento de modelos de IA.

Dentro de la fase experimental del trabajo, es importante destacar que se exploró también TensorFlow[13] para inteligencia artificial, hasta que se descubrió que PyTorch[14] proporcionaba características más favorables para este proyecto.

2 Estudio del arte

2.1. Visión por Computador

La visión por computador[15] es una rama de la inteligencia artificial que se enfoca en capacitar a las computadoras para interpretar y comprender el mundo visual que nos rodea, de manera similar a como lo hacen los seres humanos. Implica el desarrollo de algoritmos y sistemas que permiten a las máquinas analizar, procesar y extraer información significativa de imágenes y videos.

La visión por computador abarca varias tareas clave, como reconocimiento de texto, realidad aumentada, conducción autónoma, segmentación semántica y clasificación de objetos, entre otras. Un apartado destacable dentro de la visión por computador es la detección de objetos que abarca una versión moderna que utiliza redes neuronales convolucionales[16] o su versión más tradicional, que utiliza modelos matemáticos proporcionando simplicidad, robustez y mayor eficiencia de los recursos.

Dentro del enfoque tradicional de detección de objetos en visión por computador, destaca una técnica conocida como la Transformada de Hough[17]. Esta técnica se utiliza para identificar formas geométricas en una imagen, como líneas, círculos o elipses. Su fundamento radica en transformar puntos en una imagen en parámetros que describen estas formas, lo que permite detectarlas incluso cuando están parcialmente ocultas o distorsionadas.

2.1.1. Transformada de Hough

La Transformada de Hough[17], una técnica utilizada en visión por computadora y procesamiento de imágenes, se basa en la idea de representar formas geométricas, como líneas, de manera probabilística en un espacio paramétrico. En contraste con la representación algebraica directa, esta técnica introduce un enfoque probabilístico para detectar incluso formas problemáticas, como líneas verticales.

La Transformada de Hough aborda el desafío de detectar líneas en una imagen a través de

los siguientes pasos probabilísticos:

- Votación Probabilística: Para cada punto en la imagen que podría pertenecer a una línea, se traza una curva en el espacio paramétrico $r - \theta$. Cada punto en esta curva representa una posible línea que pasa por el punto correspondiente en la imagen. En esta etapa, cada punto contribuye a la curva con una cierta probabilidad, ya que podría o no pertenecer a una línea.
- Acumulación Probabilística: Las curvas generadas mediante la votación se acumulan en un espacio acumulador. Cuando las curvas se cruzan o se superponen, los votos se acumulan en esas áreas. Esta acumulación de votos refleja las áreas donde hay una alta probabilidad de que existan líneas en la imagen original. Las intersecciones y áreas de alta acumulación en el espacio acumulador representan regiones altamente probables donde las líneas podrían estar presentes.
- Detección Probabilística: En esta fase, se buscan picos en el espacio acumulador. Estos picos indican las líneas más probables en la imagen original. Cada pico corresponde a una transformación inversa que representa una línea real en la imagen. La altura y ubicación de estos picos en el espacio acumulador reflejan la confianza o probabilidad de que una línea específica esté presente en la imagen.

Por último, la introducción de elementos probabilísticos en la Transformada de Hough permite manejar casos desafiantes, como líneas verticales, de manera más efectiva. En lugar de depender de una única representación matemática precisa, la técnica se basa en la probabilidad de que un punto pertenezca a una línea y acumula esta información para identificar las líneas más probables en la imagen. Esto la hace más robusta y adaptable a diversas situaciones y formas geométricas.

2.2. Inteligencia Artificial

La inteligencia artificial (IA)[18] implica recrear en sistemas computacionales ciertos procesos de inteligencia humana, como el razonamiento lógico y deductivo, la habilidad de aprender y adaptarse, así como la resolución de problemas. Este campo se basa en programar algoritmos y sistemas que puedan emular estas capacidades cognitivas, permitiendo a las máquinas

llevar a cabo tareas complejas de manera automatizada y eficiente.

Estos sistemas están diseñados para realizar tareas que normalmente requerirían la participación humana, mencionados anteriormente. La IA busca crear máquinas capaces de imitar algunas de las capacidades cognitivas de los seres humanos, lo que les permite realizar actividades complejas de manera automatizada.

Existen varias ramas dentro de la inteligencia artificial:

- Aprendizaje automático (Machine Learning): Se trata de una técnica que permite a las máquinas mejorar su rendimiento en tareas específicas a través de la experiencia y los datos. El aprendizaje automático se basa en algoritmos que permiten a los sistemas reconocer patrones y ajustar su funcionamiento sin ser programados explícitamente para cada situación. Por otro lado, requiere la intervención y el aporte humano para llevar a cabo procesos de entrenamiento y mejora. Se distinguen 3 tipos:
 - Aprendizaje Supervisado: El modelo aprende con datos que tienen etiquetas que indican las respuestas correctas. Esto le permite hacer predicciones y clasificar nuevas situaciones.
 - Aprendizaje No Supervisado: El modelo explora datos sin etiquetas para encontrar patrones ocultos, agrupando elementos similares y revelando información útil.
 - Aprendizaje por Reforzamiento: El modelo interactúa con su entorno y aprende a tomar decisiones para maximizar las recompensas a lo largo del tiempo, ajustando su comportamiento en base a las experiencias pasadas.
- Aprendizaje profundo (Deep Learning): Es una subrama del aprendizaje automático que se centra en el uso de redes neuronales artificiales profundas para abordar problemas complejos, como el reconocimiento de imágenes y el procesamiento del lenguaje natural.
- Redes neuronales artificiales[19]: Están inspiradas en la estructura y funcionamiento del cerebro humano. Utilizan capas interconectadas de nodos (neuronas artificiales) para procesar información y reconocer patrones en los datos. Aunque existen diversos tipos

de redes neuronales, el enfoque principal de este trabajo se dirige hacia el uso específico de la red neuronal convolucional.

2.2.1. Redes Neuronales

Las redes neuronales, como mencionamos anteriormente, son modelos matemáticos inspirados en la estructura y funcionamiento del cerebro humano.

Están diseñadas para procesar información y aprender patrones complejos a partir de datos. Estos datos pueden ser cualquier cosa, incluso datos complejos como lo son imágenes, texto y audio. Cada “neurona” en una red neuronal artificial realiza operaciones matemáticas en sus entradas y produce una salida que se pasa a otras neuronas.

La estructura de un modelo de redes neuronales está compuesta por una serie de capas, y estas capas se organizan en tres tipos principales:

- Capa de Entrada: Esta es la primera capa del modelo y recibe los datos iniciales que se alimentan a la red. En una red neuronal para procesar imágenes, por ejemplo, esta capa estaría compuesta por neuronas que representan los píxeles de la imagen.
- Capas Ocultas: Las capas ocultas son intermedias entre la capa de entrada y la capa de salida. Las capas ocultas procesan y transforman los datos a medida que fluyen a través de la red. Cuantas más capas ocultas tenga la red, más profunda se considera. Una mayor profundidad puede permitir aprender representaciones más complejas, pero también puede aumentar la complejidad y los recursos necesarios para entrenar la red. La elección de la profundidad debe equilibrarse según el problema y los recursos disponibles.
- Capa de Salida: La capa de salida es la última capa del modelo y produce el resultado final. Dependiendo de la tarea que se esté abordando, esta capa puede tener diferentes configuraciones. Por ejemplo, si estás haciendo clasificación, la capa de salida puede tener neuronas que representen las clases posibles.

La información fluye a través del modelo de redes neuronales en un proceso llamado propagación hacia adelante. Funcionando de la siguiente manera:

Los datos iniciales se introducen en la capa de entrada. Cada neurona en esta capa recibe una parte de los datos. Cada neurona en las capas ocultas y de salida tiene conexiones con las neuronas de la capa anterior. Cada conexión tiene un peso que determina su influencia en la salida de la neurona. Las entradas se multiplican por los pesos y se suman, esta parte referencia al “impulso nervioso” de la neurona.

Después de la suma ponderada, se aplica una función de activación. Esta función introduce no linealidades en el modelo con la finalidad de que la red capture patrones complejos en los datos. Hay diferentes tipos de funciones de activación, ejemplos más utilizados son “ReLU”, “SoftMax” y “Sigmoid”.

Por último, las salidas de las neuronas de la capa actual se convierten en entradas para las neuronas de la siguiente capa. El proceso se repite a través de todas las capas hasta llegar a la capa de salida, que produce el resultado final.

2.3. Redes Neuronales Convolucionales

Las redes neuronales convolucionales[16] son un tipo especializado de redes neuronales diseñadas principalmente para el procesamiento de imágenes y datos con estructura espacial, como señales de audio y datos en formato de cuadrícula. Su arquitectura se inspira en la organización del sistema visual del cerebro humano y ha demostrado un gran éxito en tareas de visión por computador.

Para entender esta arquitectura es necesario explicar lo que es una capa convolucional. Una capa convolucional es como una especie de “filtro” inteligente que se desliza sobre una imagen o conjunto de datos para detectar patrones importantes. Este filtro sería el kernel que se va deslizando a través de la matriz de la imagen, realizando una convolución con los píxeles de la imagen, dando un resultado que se almacena en un mapa. Y estos mapas se van pasando a las siguientes capas.

Esto sería la primera fase de la red neuronal convolucional, posteriormente, se utiliza funcio-

nes de activación[20] sobre los mapas de características producidos con el objetivo de producir valores no lineales para ayudar a la detección de patrones complejos.

Después de producirse estos cambios sobre los mapas de características, se aplican capas de Pooling. Estas capas reducen la dimensionalidad de los mapas de características al seleccionar los valores máximos o promedios en regiones locales. Esto ayuda a reducir la cantidad de parámetros y a mantener las características más importantes.

En la fase final, nos encontramos con la capa de salida, cuya función radica en interpretar los resultados finales y ofrecer una respuesta concluyente. Dentro de esta fase final tenemos dos funciones que ayudan a mejorar la red, estos son la función de pérdida y el optimizador.

La función de pérdida que calcula la diferencia entre las predicciones hechas por la red y los valores reales del conjunto de datos de entrenamiento. Mide qué tan lejos está la red de hacer predicciones precisas, dando un resultado. El optimizador, por otra parte, cuando ha sido aplicada la función de pérdida y tiene el resultado, indica cómo deben ajustarse los pesos para disminuir la pérdida.

Cabe destacar que la arquitectura de una CNN puede variar dependiendo de diversos factores, como el tipo de problema que se esté abordando (clasificación, detección de objetos, segmentación, etc.), el tamaño y la complejidad de los datos, y la cantidad de datos disponibles para el entrenamiento. Esto significa que no todas las CNNs tendrán la misma cantidad de capas, ni estarán organizadas exactamente de la misma manera.

3 Requisitos del sistema

3.1. Requisitos Funcionales

Los requisitos funcionales son especificaciones claras y detalladas que describen las acciones y funcionalidades específicas que un sistema, software o producto debe llevar a cabo. Estos requisitos definen “qué” tareas debe realizar el sistema para cumplir con los objetivos del usuario y del negocio. En el caso de este trabajo, se pueden observar las siguientes:

- Detección de Tableros:
 - El sistema debe ser capaz de identificar y localizar tableros de ajedrez en imágenes.
 - Debe detectar varios tipos de tableros y posiciones de ajedrez en diferentes orientaciones.
 - El sistema debe ser capaz de reconocer tableros con diferentes patrones y tamaños de cuadrícula.
 - El sistema tiene que detectar imágenes digitales, provenientes de dispositivos electrónicos, capturas de pantallas realizadas por un dispositivo y tableros previamente especificados por un usuario.
- Clasificación de Piezas:
 - El sistema debe ser capaz de clasificar las piezas de ajedrez presentes en cada casilla.
 - Debe reconocer y etiquetar las piezas con sus etiquetas correspondientes.
 - La clasificación debe ser confiable incluso ante variaciones en la iluminación y posición de las piezas, en su defecto el sistema se esforzará por mantener un alto nivel de precisión en estas circunstancias cambiantes.
 - El sistema debe ser capaz de tomar la disposición de las piezas en el tablero y transformarla en notación FEN.
 - La notación FEN generada debe cumplir con los estándares de notación algebraica Forsyth-Edwards.

3.2. Requisitos No funcionales

Los requisitos no funcionales son criterios que definen la calidad, el rendimiento y las características del sistema, más allá de su funcionalidad específica. Se centran en atributos como la usabilidad, la seguridad, la velocidad y la confiabilidad. Estos requisitos se refieren a “cómo” el sistema debe cumplir sus funciones y proporcionar una experiencia satisfactoria para los usuarios y las partes interesadas.

- Rendimiento:
 - El sistema debe operar en tiempo real para una experiencia fluida y en tiempo aceptable.
 - El tiempo de procesamiento de la imagen y detección de tableros debe ser lo más rápido posible.
- Precisión:
 - La detección de tableros, seccionamiento de casillas y clasificación de piezas deben ser precisos dentro de los parámetros y las capacidades del sistema.
 - La tasa de error en la clasificación de piezas debe mantenerse por debajo de un umbral establecido.
- Usabilidad:
 - La interfaz de usuario debe ser intuitiva y fácil de usar, incluso para usuarios no técnicos.
 - Se deben proporcionar indicaciones claras sobre cómo capturar y cargar imágenes de tableros para el análisis.
- Escalabilidad:
 - El sistema debe ser capaz de manejar diferentes tamaños de tablero y variaciones en la complejidad de las imágenes.
 - Debe ser escalable para abordar un aumento en la cantidad de piezas y tableros en imágenes más grandes.

- Robustez:
 - El sistema debe ser resistente a las variaciones en la iluminación, el ruido y las distorsiones en las imágenes.
 - Debe ser capaz de manejar tableros con piezas parcialmente ocultas o en ángulos irregulares.
- Disponibilidad:
 - El sistema debe estar disponible para su uso en cualquier momento, con una infraestructura que minimice los tiempos de inactividad.

3.3. Diagramas de Secuencia

Requisito	Detección de tableros
Descripción	El sistema debe ser capaz de detectar el tablero de ajedrez establecido por el usuario, en caso contrario, el sistema proporcionará al usuario una interfaz para que seleccione el tablero exactamente
Pre-Condiciones	Debe haber guardado la imagen en el dataset del proyecto y ejecutado el comando de iniciación
Post-Condiciones	Obtiene la sección del tablero
Prioridad	Alta

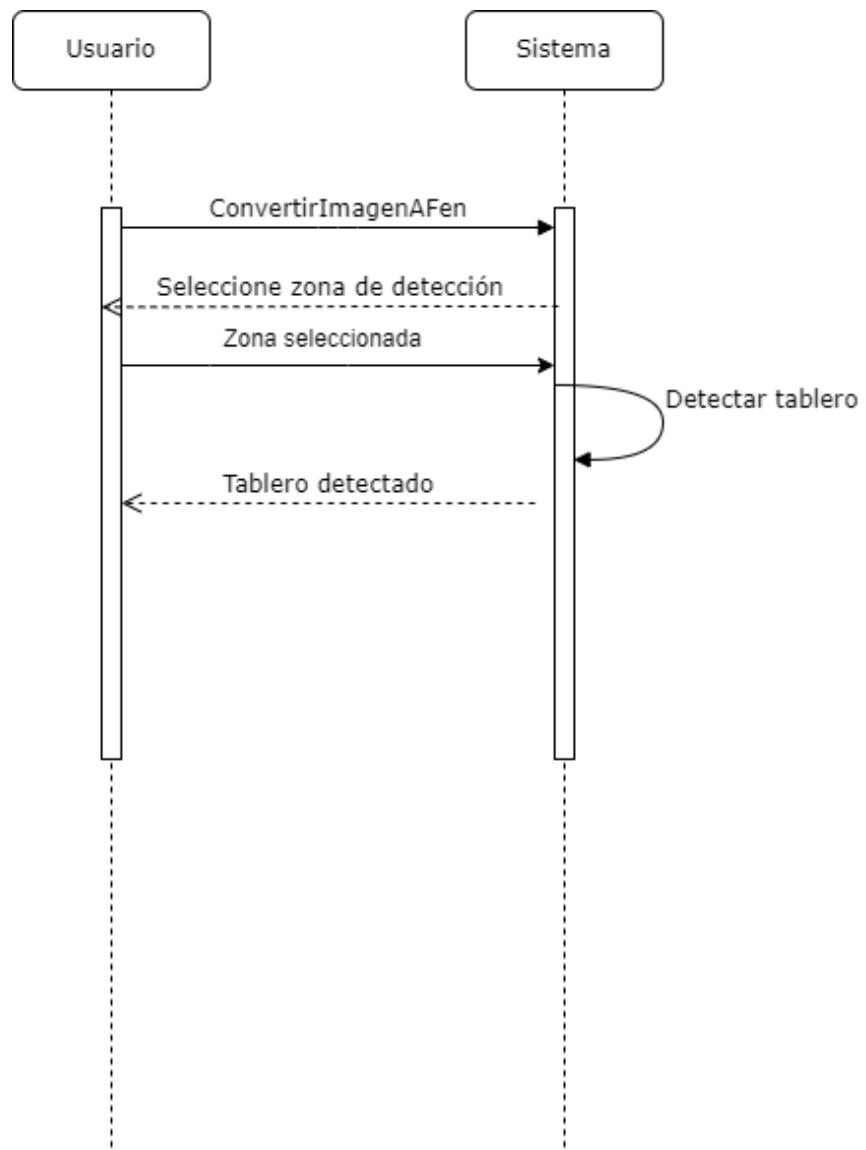


Figura 1: Secuencia de tablero detectado por el programa

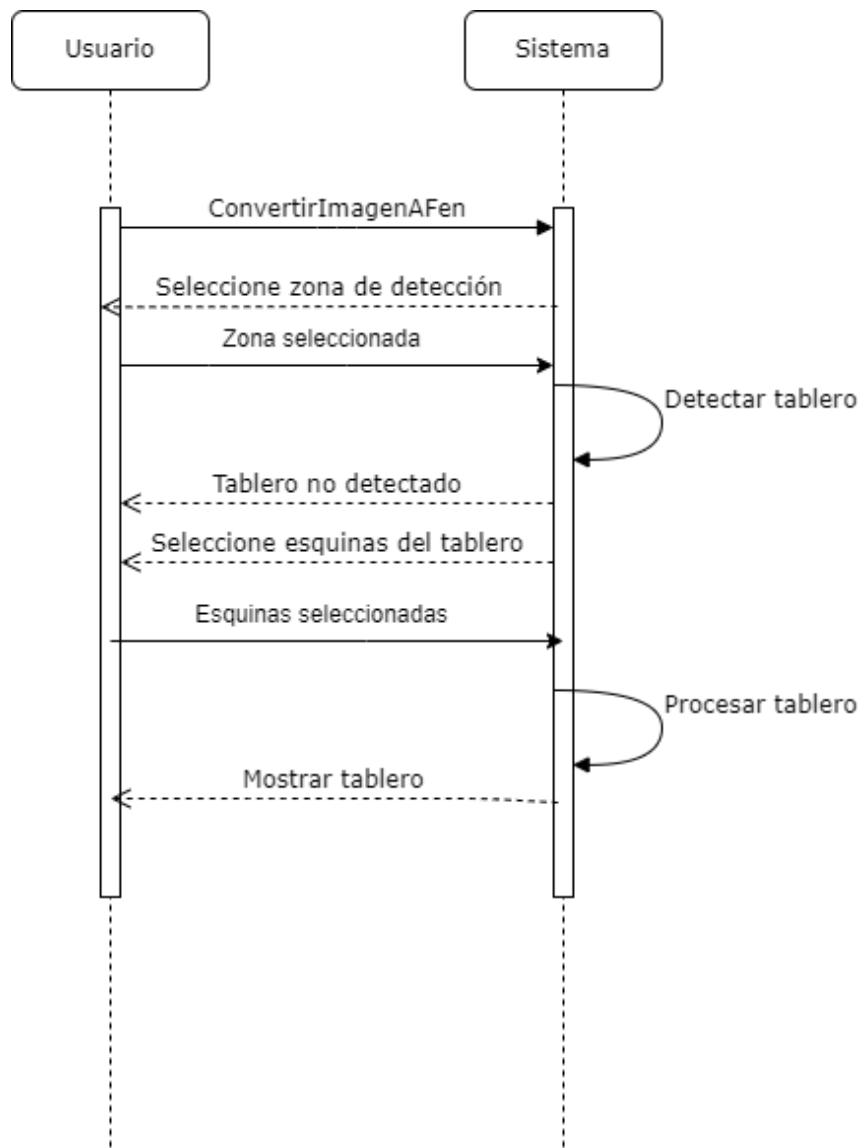


Figura 2: Secuencia de tablero no detectado por el programa

Requisito	Clasificación de piezas
Descripción	El sistema debe ser capaz mediante un tablero detectado, clasificar las piezas y devolver la notación FEN asociada a ese tablero. En caso de no poder detectar el tablero, se abrirá una interfaz para que el usuario especifique donde se encuentra el tablero, una vez realizado este proceso, clasificará las piezas del tablero y devolverá la notación FEN asociada.
Pre-Condiciones	Debe haber guardado la imagen en el dataset del proyecto y ejecutado el comando
Post-Condiciones	Obtiene la notación FEN del tablero
Prioridad	Alta

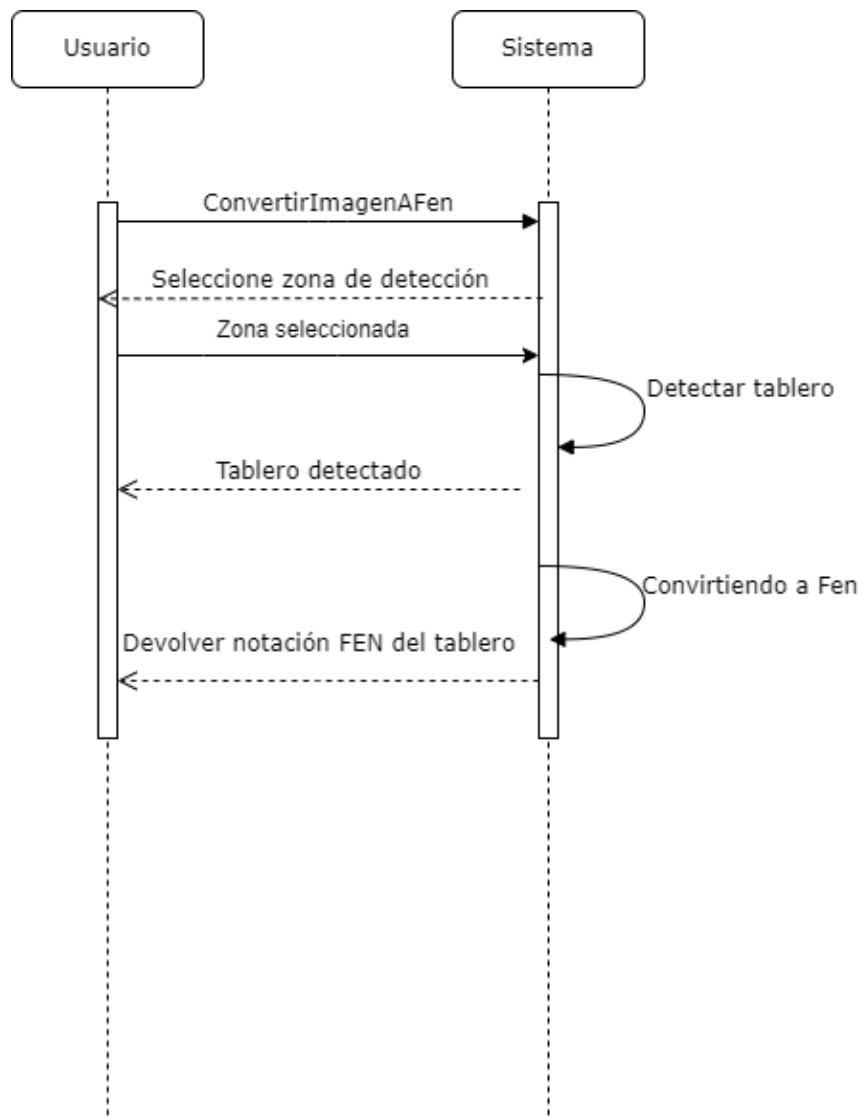


Figura 3: Secuencia de funcionamiento estándar

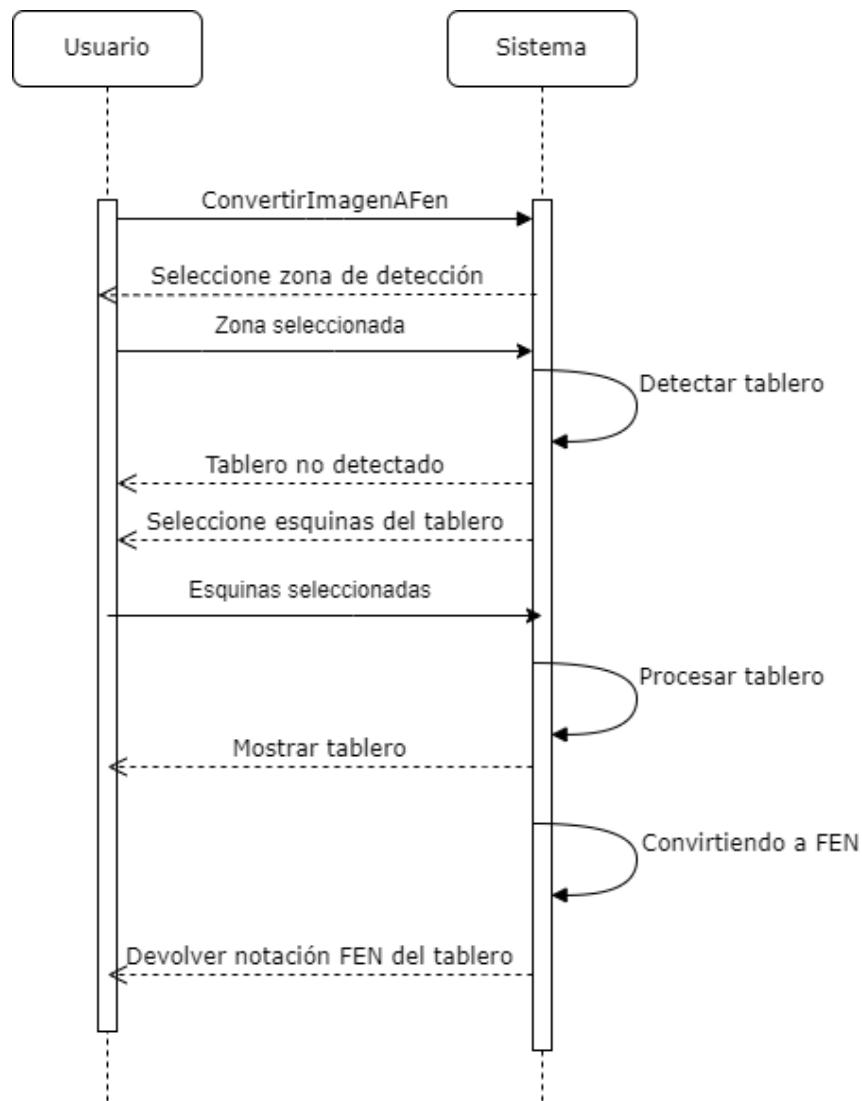


Figura 4: Secuencia de funcionamiento ante fallos producidos en detección

3.4. Patrón de Diseño

El código de este proyecto se estructura de la siguiente manera:

- El código principal, “image2Fen.py”, que se encarga de realizar el paso previo a la detección y es el archivo que incorporá todas las funciones necesarias para el desarrollo del programa. Actúa como interfaz y archivo principal de ejecución
- Funciones de detección correspondientes al archivo “detectline.py” que se encarga de proporcionar funciones que detectan líneas e intersecciones, siendo el encargado del seccionamiento del tablero.
- Funciones de predicción y encargados de tipificar la notación, este último archivo, “predecir2version.py”, se encarga de proporcionar funciones que obtenga una predicción de las casillas del tablero y proporcione una notación FEN en base a esas predicciones.

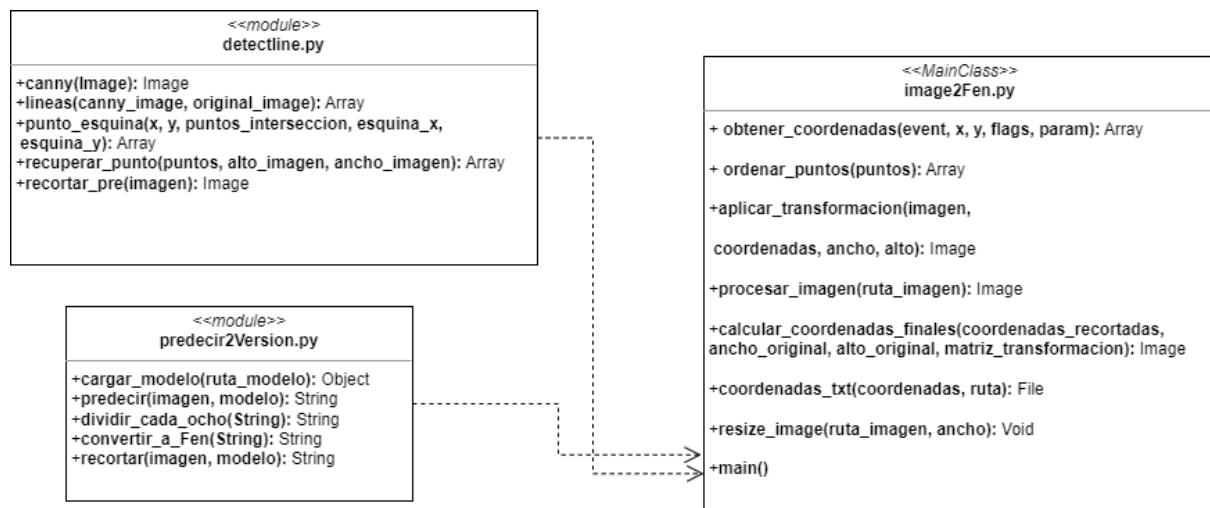


Figura 5: Esquema del patrón

El diseño adoptado se caracteriza por una estructura en la que el sistema se fragmenta en módulos funcionales individuales, cada uno de los cuales asume la responsabilidad de una parte específica del programa.

Estos módulos poseen tareas bien definidas y concretas, lo que contribuye a mantener el código de manera ordenada y modular. Además, esta organización propicia la reutilización del código y simplifica las labores de mantenimiento.

En esencia, el patrón de diseño empleado se identifica como el enfoque de “Módulos” o “Módulos Funcionales”[21], un enfoque que promueve la eficiente gestión de las responsabilidades en el sistema.

4 Fase de obtención de datos

Dentro de este trabajo, se distinguen dos tipos de conjuntos de datos esenciales: el conjunto de pruebas y el conjunto de entrenamiento.

En el conjunto de pruebas, se ha seleccionado un conjunto de imágenes de tableros de ajedrez de plataformas en línea relevantes en la actualidad, tales como:

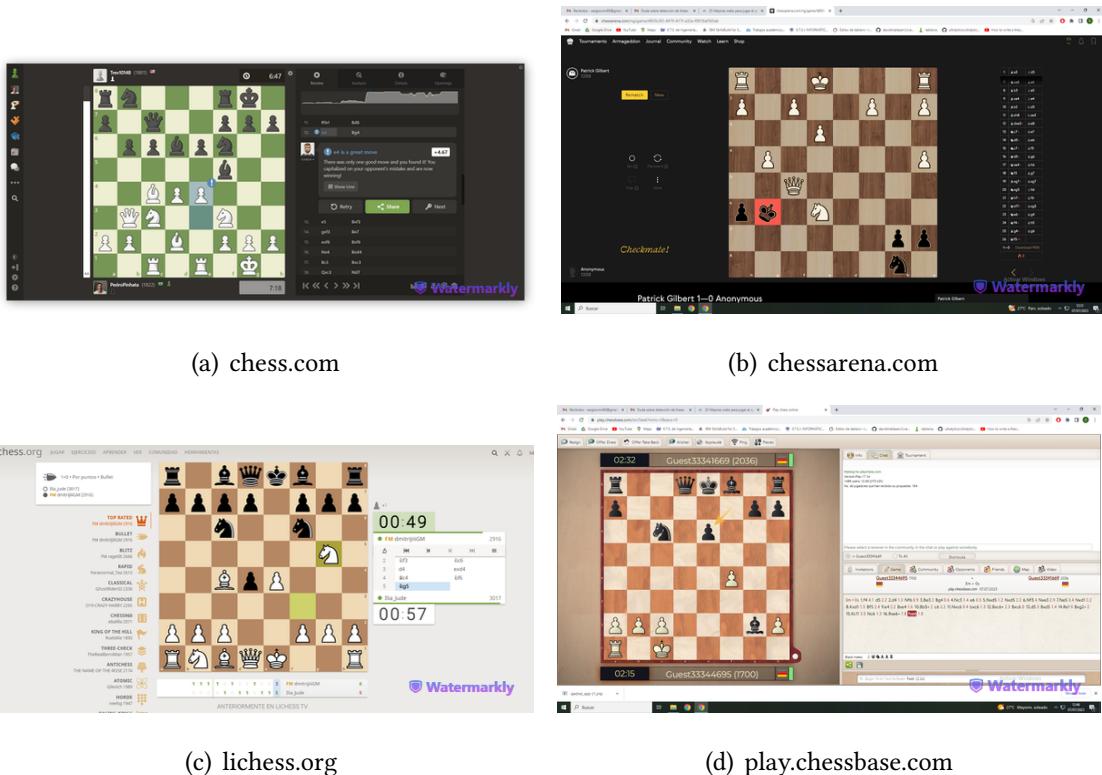


Figura 6: Plataformas online de ajedrez

De cada una de estas plataformas se han capturado diez imágenes principales, presentando distintos tableros y disposiciones únicas de piezas. Las imágenes han sido reescaladas a una altura constante de 600 píxeles, mientras que el ancho se ha ajustado dentro de un rango de [280-360] píxeles, manteniendo la proporción original. Esta adaptación busca facilitar el manejo cómodo desde la consola para el usuario final.

Luego de recopilar estas capturas principales, se han aplicado diversas transformaciones utilizando una serie de funciones. La función principal de estas transformaciones tiene como objetivo verificar el rendimiento de la aplicación bajo ciertas condiciones de iluminación, contraste, ruido gaussiano, entre otros. Estos parámetros resultantes de estas transformaciones quedan registrados durante el proceso de transformación. Entre las transformaciones aplicadas se destacan:

- Transformaciones homográficas: Son transformaciones geométricas que mapean puntos y figuras de un espacio a otro mediante una matriz homográfica. Estas transformaciones resultan en cambios de perspectiva, orientación y tamaño.



Figura 7: Transformación

- Ajustes de brillo y color: Se han aplicado aumentos y disminuciones de brillo a las imágenes, lo que afecta las tonalidades del tablero y las piezas.

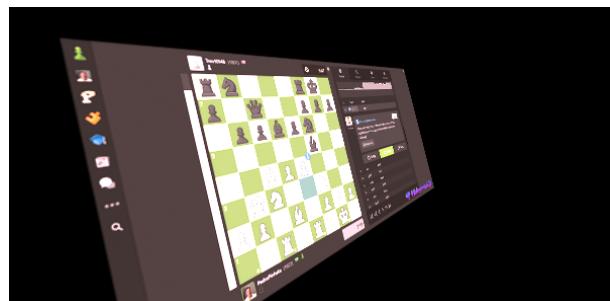


Figura 8: Transformación brillo y color

- Ruido gaussiano: Algunas imágenes han sido sometidas a ruido gaussiano para incrementar el nivel de dificultad en la detección de las líneas del tablero.



Figura 9: Transformación con ruido gaussiano

Los efectos de estas modificaciones se han incluido en el conjunto de pruebas. Junto con las imágenes, se ha agregado un archivo en formato “.txt” para guardar las coordenadas precisas del tablero, y también un archivo en formato “.xlsx” donde se registrarán todas las alteraciones de brillo, color y ruido gaussiano aplicadas a la imagen. Estos registros se utilizarán para analizar el rendimiento y la eficacia del algoritmo de detección y posteriormente la aplicación completa. Este conjunto suma en total 2000 imágenes para el uso exclusivo de pruebas.

Por otro lado, el conjunto de entrenamiento, es fundamental para la creación del modelo de inteligencia artificial que clasifica las piezas del tablero de ajedrez. Este conjunto se ha obtenido de la plataforma chess.com, que ofrece una variedad de configuraciones de tableros y piezas para familiarizar a los usuarios con el entorno del juego.

Tras capturar diversas imágenes que combinan distintas configuraciones de tablero y piezas, se ha utilizado una herramienta de recorte para generar las primeras 1000 imágenes (90 x 90) correspondientes a cada casilla del tablero almacenado. Posteriormente, se aplicaron diversas técnicas para expandir este conjunto de datos y mejorar el proceso de entrenamiento de la inteligencia artificial. Las técnicas incluyen:

- Rotaciones: Se realizaron tres tipos de rotaciones para presentar las piezas en diferentes orientaciones.



(a) Pieza recta (b) Pieza rotada a izquierda (c) Pieza rotada a derecha

Figura 10: Piezas con distintas orientaciones

- Cambios en la resolución: Se redujo y luego aumentó el tamaño de las imágenes, lo que disminuyó la resolución y difuminó las imágenes.



Figura 11: Pieza difuminada

- Variaciones de tamaño: Algunas imágenes se escalan a (120 x 120) para mayor claridad, y se varió la posición de la pieza dentro de la imagen, incluso presentando piezas con partes seccionadas para enfatizar aspectos claves de cada pieza.



(a) Pieza seccionada (b) Pieza pixelada

Figura 12: Piezas de diferente tamaño y seccionada

- Ruido gaussiano: Se agregó ruido gaussiano a algunas imágenes para dificultar ligeramente la clasificación.



Figura 13: Pieza bajo ruido gaussiano

El propósito fundamental de aplicar esta serie de transformaciones a los datos de entrada en un modelo es entrenarlo de manera que sea altamente resistente a diversas variaciones en las condiciones de los datos, como cambios en la iluminación, tamaño, rotación o presencia de ruido gaussiano en las imágenes de las piezas. Los resultados tras aplicar estas transformaciones se han incorporado al conjunto de entrenamiento, alcanzando un total de 1714 imágenes por cada clase de piezas en el tablero. Dichas imágenes se han dividido en tres subconjuntos dentro del algoritmo de entrenamiento de la inteligencia artificial y se han organizado en carpetas bajo la denominación general “data”.

5 Fase de Desarrollo

5.1. Descripción general del proceso

La evolución del desarrollo del proyecto se perfila en la siguiente fase, donde se realiza un desglose detallado de las tareas clave.

En primer lugar, en el “Desarrollo del algoritmo de detección de tableros”, se lleva a cabo una exploración exhaustiva de enfoques experimentales en el ámbito de la visión por computador. Este proceso abarca desde el uso de máscaras para resaltar áreas de interés hasta la aplicación de la transformada de Hough para identificar patrones, considerando además la opción de automatización o semiautomatización según la necesidad. Por otro lado se utilizan el conjunto de datos de prueba con el objetivo de proporcionar una evaluación el algoritmo obtenido y determinar si satisface las necesidades del proyecto.

A continuación, se profundiza en el “Desarrollo del algoritmo de seccionamiento de casillas”, donde se diseña un proceso específico para procesar el tablero capturado de manera eficaz, permitiendo que el clasificador opere de manera óptima. La atención se centra en la extracción de casillas individuales para evitar que el clasificador se vea abrumado por la complejidad del tablero en su conjunto.

En el tercer aspecto, el “Desarrollo del clasificador”, se realiza una exhaustiva experimentación con diversos modelos de clasificación para lograr un equilibrio entre eficiencia y rendimiento. Se busca el modelo más adecuado que cumpla con los requisitos y objetivos del proyecto, con ajustes de arquitectura y parámetros para obtener la mejor capacidad de clasificación en la detección y reconocimiento de piezas de ajedrez.

El cuarto aspecto, el “Desarrollo del algoritmo transformador de notación FEN” cierra el ciclo. En esta fase, se aplican los resultados obtenidos del clasificador y se adaptan a la notación FEN, esencial para representar de manera precisa y estandarizada la disposición de las piezas en el tablero. Esta etapa es crucial para el análisis y la interpretación de las partidas de ajedrez.

Por último, se visualizarán los resultados obtenidos de la fase de desarrollo mediante pruebas unitarias realizadas al conjunto de datos sobre imágenes de tableros, para validar y verificar el funcionamiento del proyecto en base a los requisitos establecidos.

5.2. Desarrollo del algoritmo de detección de tableros

Durante la fase de desarrollo se experimentó diversos enfoques a medida que avanzaba el proyecto debido a los inconvenientes que iban surgiendo. A medida que avanza en el proceso se van explicando los diferentes enfoques y sus resultados. Por otro lado se expone al final, el algoritmo de detección resultado de estos experimentos.

5.2.1. Ideas Descartadas y Motivos de No Implementación

En la etapa inicial del desarrollo, el enfoque principal consistía en lograr la detección automática de tableros de ajedrez en imágenes. Esto implicaba identificar las líneas presentes en la imagen como punto de partida. Sin embargo, a lo largo de esta fase, se enfrentaron varios desafíos que llevaron a la exploración de diferentes enfoques.

En primer lugar, se planteó la idea de detectar las líneas exteriores del tablero utilizando detección de bordes mediante el algoritmo de Canny y la Transformada de Hough. Se intentó seleccionar líneas de longitud similar y con variaciones controladas en el tamaño. A pesar de este esfuerzo, se encontraron dificultades al lidiar con tableros digitales con posibles deformaciones y otros problemas inherentes a la captura de imágenes.

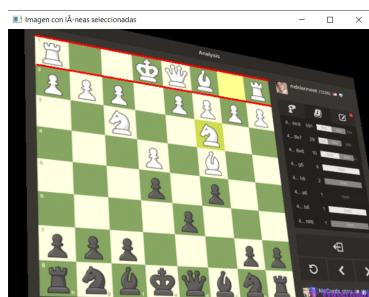


Figura 14: Primer Experimento

Aquí empezamos a ver una de las problemáticas, los tableros de ajedrez online no son perfectos en su totalidad e incluso podemos observar otro defecto, que se centra en el reescalado de la imagen, puede deformar el tablero. Otros puntos negativos se centran en la nula detección sobre fotos del mismo tablero, como se puede ver a continuación:



Figura 15: Fallos del primer experimento en fotos

Una segunda estrategia se centró en aplicar una variante de la Transformada de Hough con parámetros adicionales:

- minLineLength: este parámetro establece la longitud mínima de la línea
- maxLineGap: este parámetro indica el intervalo máximo de líneas para tratarlas como una sola.

Aunque este produjo mejoras en la detección, seguía siendo un enfoque insuficiente para manejar de manera efectiva la detección en todas las condiciones de captura y tableros.



Figura 16: Segundo experimento

Esto se debía a un mejor detección de las líneas del tablero pero no se podía obtener un filtro que simplificara el número de líneas existentes. Por otro lado, el algoritmo era impracticable en las fotos.

La tercera idea está enfocada en la forma característica del tablero de ajedrez debido a que por su forma debe ser cercana a un cuadrado perfecto, lo que facilita que una vez detectado las líneas de una imagen se puedan extraer aquellas líneas que conformen un cuadrado, teniendo como resultado el tablero recortado.

Para la realización de este proceso era necesario detectar las líneas junto con la imagen original para después detectar los contornos. En este punto, se muestra el primer problema de este desarrollo, las formas residuales presentes en la imagen donde no se han producidos las líneas provocan falsos positivos que hacen que falle el algoritmo.

Por lo que para poder aplicar la detección por contornos, es necesario crear una máscara en blanco del mismo tamaño que la imagen original para eliminar esas formas residuales. Posteriormente, dibujarle las líneas previamente detectadas y utilizar el algoritmo de detección de contornos.

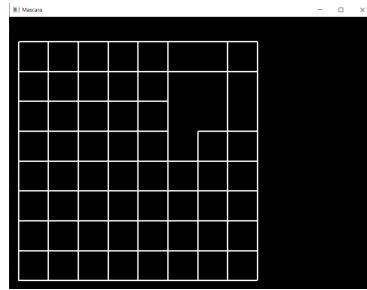
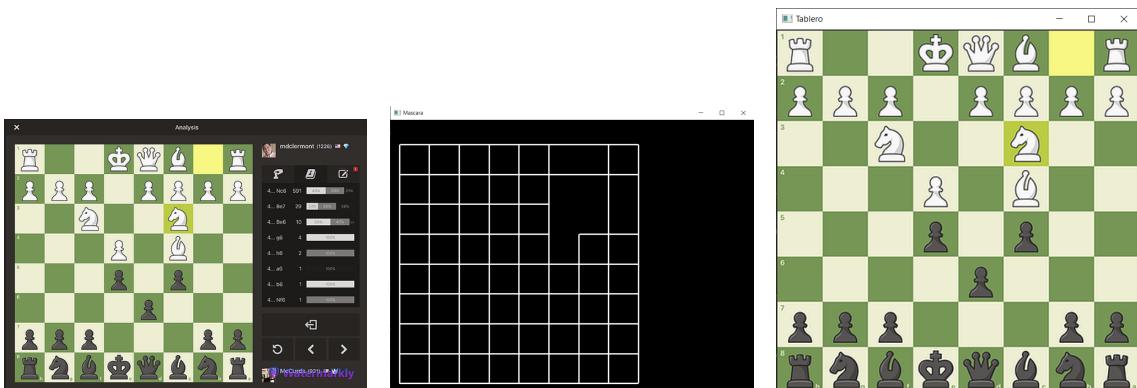


Figura 17: Máscara Tercer Experimento

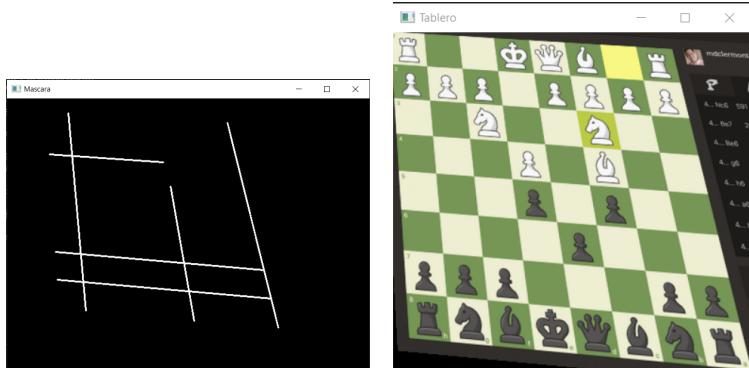
En este punto, se muestra el segundo problema del desarrollo, dentro de los contornos detectados se pueden encontrar más elementos similares al tablero, por lo que es necesario un filtrado. Este filtrado se realizaría comprobando el ancho y el alto del rectángulo delimitado por el contorno, y si este cumplía un umbral se recortaba para mostrarlo al usuario.



(a) Tercer Experimento (b) Máscara Tercer Experimento (c) Tercer Experimento Resultado

Figura 18: Resultado tercer experimento

Aunque para los casos enfocados en las capturas de tableros y tableros previamente recortados se cumplía las expectativas, para el caso de las fotos realizadas a tableros digitales, esta idea no conseguía extraer correctamente el tablero.



(a) Fallos en Máscara Tercer Experimento (b) Fallos Resultado Tercer Experimento

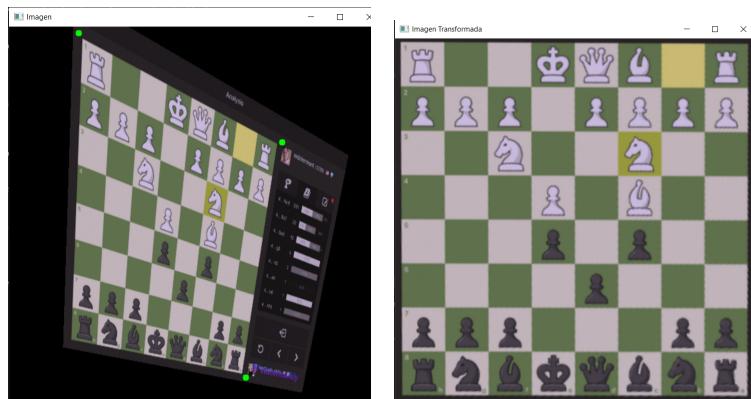
Figura 19: Fallos relacionados con el tercer experimento

A la vista de los resultados proporcionados por este último desarrollo, se empezó a enfocar el trabajo bajo la idea de un algoritmo semiautomático con la participación de un usuario. En gran parte, debido a los problemas derivados de la toma de imágenes realizadas por dispositivos electrónicos, por ejemplo el ángulo de la captura, la iluminación y el ruido existente en la imagen junto con un posible cambio de color producen problemas que no se encuentra en las capturas de imágenes o en tableros directamente recortados.

5.2.2. Implementación Final del Algoritmo de Detección de Tableros

Una vez experimentado con diversos focos de orientación con respecto a la detección, se empezó a discernir diferentes pasos en el desarrollo del algoritmo, en retrospectiva con los inconvenientes de las pruebas anteriores. El enfoque principal se encaminó hacia la centralización del tablero mediante la inclusión de cuatro puntos de referencia. Estos puntos, situados cerca de las esquinas del tablero, desempeñarían una función principal al efectuar una transformación proyectiva. Su objetivo era ajustar la perspectiva de la imagen adquirida y orientarla hacia un plano perpendicular. En otras palabras, se buscaba aplicar una transformación de proyectiva que convirtiera la región del tablero seleccionada en una imagen plana y adecuada para la posterior detección de dicho tablero.

Este proceso de transformación proyectiva, utiliza cuatro puntos delimitadores, posibilitando alinear la imagen en una perspectiva más favorable. Esto resultaba esencial para eliminar distorsiones y desviaciones que pudieran surgir debido a la captura en ángulos o condiciones variadas. Así, se lograría una representación bidimensional óptima del tablero, permitiendo que los algoritmos de detección y procesamiento actuaran con precisión en un plano que favoreciera la detección del tablero de ajedrez y sus características inherentes.



(a) Selección Zona de Interés (b) Transformación Zona de Interés

Figura 20: Transformación sobre la zona de interés

Posteriormente a la transformación perspectiva se retomarían nuevas ideas con objetivo de encontrar las esquinas del tablero. La idea general era utilizar los puntos de referencia obtenidos en la selección del usuario, obteniendo así regiones que contienen las esquinas del tableros. De la manera que se muestra a continuación:

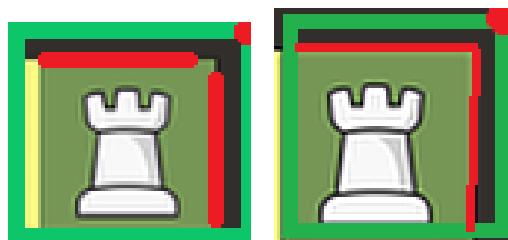
De esta concepción emergieron dos enfoques diferentes:

La primera aproximación empleó regiones de interés dentro de las cuales se buscarían las intersecciones. Estas regiones contenían las esquinas del tablero, identificadas como dos líneas discretas. La detección de la intersección de estas líneas, a su vez, engendraría el punto determinante de la esquina del tablero.

La segunda variante, en cambio, se centró en identificar líneas dentro de la región de interés y posteriormente filtrar aquellas que formaran un ángulo aproximado de 90 grados. El resultado sería la obtención del punto de intersección derivado de estas líneas.

No obstante, la primera estrategia inicialmente parecía prometedora, aunque la aplicación práctica reveló dificultades. Las líneas a menudo no se intersectaban debido a diversos factores, como ruido en la imagen, intersecciones parciales o separación entre píxeles. Por lo tanto, la eficacia de este enfoque se vio comprometida.

Por otro lado, el segundo método no resolvió los desafíos planteados por la primera ramificación. Además, la deformación ocasional de los ángulos debido a la transformación de perspectiva limitó la precisión de este enfoque. A continuación, se muestra un ejemplo práctico del resultado de aplicar estos métodos:



(a) Primera Variante (b) Segunda Variante

Figura 21: Minimap de variantes de regiones de interés

En vista de los obstáculos encontrados, se reconoció la necesidad de un replanteamiento del enfoque en la detección de las esquinas del tablero.

Los resultados derivados de estos últimos procedimientos motivaron una reconsideración de la secuencia de pasos para la detección de las esquinas del tablero. Esta revisión se fundamentó en el reconocimiento de que la reducción del espacio entre líneas conlleva un aumento de la complejidad en la detección de estas, lo que justificó una reestructuración de los procesos.

La nueva aproximación implicó una secuencia estratégica: primero, se efectuaría una transformación proyectiva para convertir la perspectiva en una vista bidimensional; a continuación, se procedería a la detección de todas las líneas presentes en la región delimitada y transformada. De las líneas identificadas, se recolectarían los puntos de intersección generados por estas, seguidamente se definirían las áreas de interés cercanas a las esquinas y se seleccionaría, entre el conjunto de puntos, aquel que se encontrara en proximidad al centro de la región acotada. Así se obtendría la esquina del tablero con un alto grado de precisión, como podemos ver en la siguiente imagen:

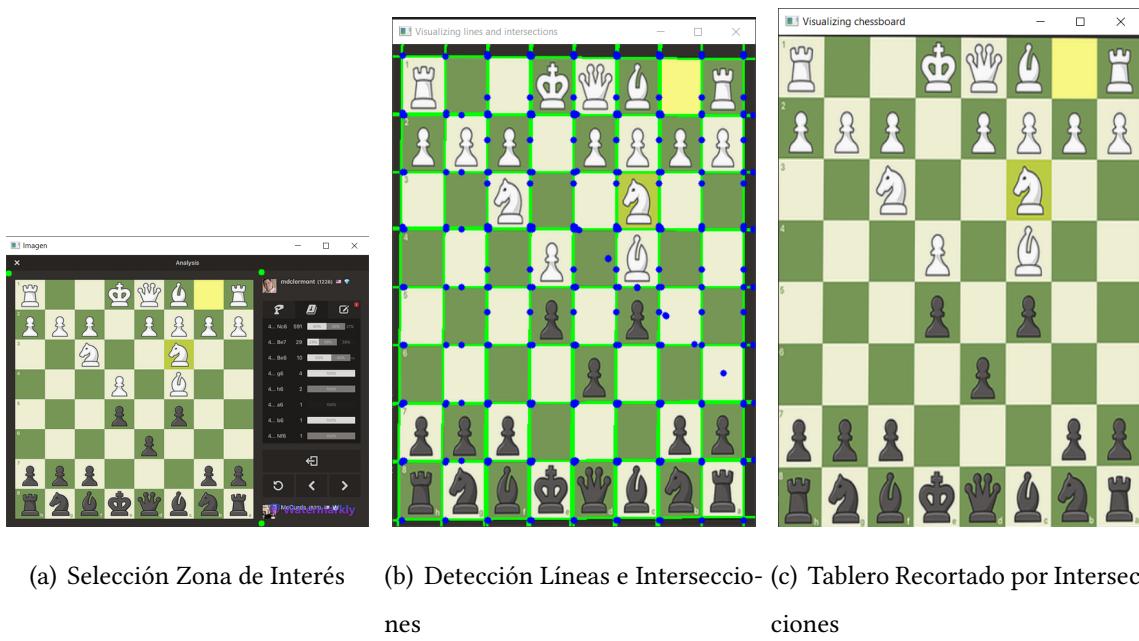


Figura 22: Resultados del proceso final

Sin embargo, surgía un inconveniente: no en todos los casos en que se detectaban líneas y sus intersecciones se obtenían puntos correspondientes a esas intersecciones. La obtención de estos puntos era esencial para definir las esquinas del tablero y, por ende, asegurar su identificación precisa. Como podemos ver en la siguiente imagen:

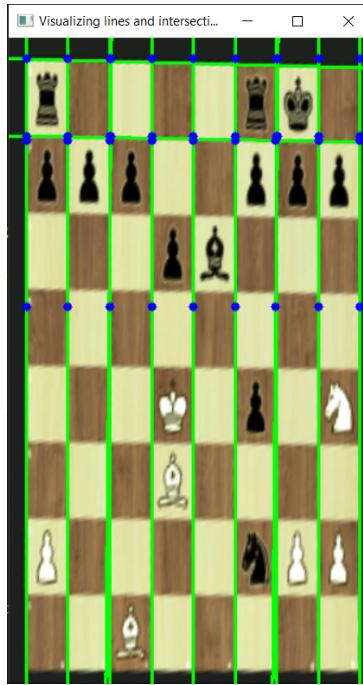


Figura 23: Ausencia de Puntos de Interés

Era primordial desarrollar una función que abordara esta situación; su finalidad radicaba en la capacidad de generar, en caso de la ausencia de un máximo de dos puntos, las ubicaciones aproximadas para los puntos restantes.

Esta función cumpliría un objetivo importante al permitir la delimitación y recorte adecuado del tablero, incluso en escenarios en los que no se contara con la totalidad de puntos necesarios.

Su objetivo sería encargarse de procesar los puntos ya obtenidos y, a través del cálculo proporcional en función de la imagen capturada, llevaría a cabo operaciones aritméticas. El propósito fundamental de estas operaciones sería la recuperación de los puntos restantes, de manera que se pudieran determinar con precisión las ubicaciones ausentes en base a la estructura general de la imagen. Facilitando de esta manera los puntos ausentes, pudiendo así extraer el tablero:

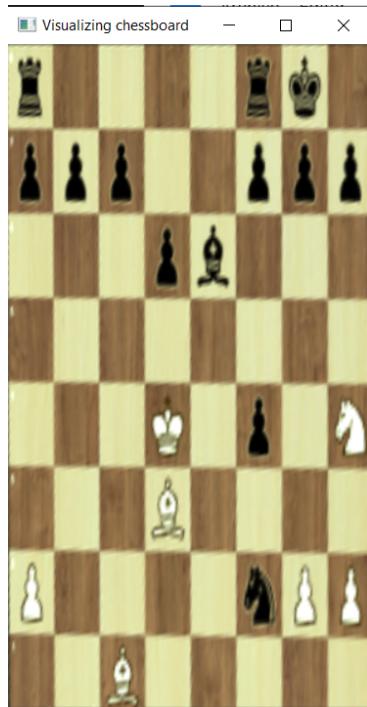


Figura 24: Tablero seccionado

Esta representaría la iteración definitiva del proyecto, que más adelante se complementaría con la funcionalidad de redimensionar las imágenes de entrada a dimensiones adecuadas tanto en anchura como en altura para lograr una adaptación óptima al tamaño de la consola.

Por otro lado, se incorporó una mejora al programa para afrontar situaciones en las que la detección semiautomática no cumpliera con las expectativas. En tales casos, se brindaría la opción al usuario de seleccionar el tablero directamente, con el propósito de agilizar la obtención del resultado deseado y proporcionar una experiencia más fluida y satisfactoria.

5.3. Desarrollo del algoritmo de seccionamiento

Este desarrollo se inició con la creación de un algoritmo que, una vez que el tablero era detectado, se enfocaba en el proceso de recortar dicho tablero y transformarlo en imágenes individuales, las cuales eran almacenadas en una ubicación específica. Para llevar a cabo este proceso, el algoritmo se basó en las dimensiones de ancho y alto de la imagen original. Dado que un tablero de ajedrez presenta casillas de tamaño uniforme, se realizó una división en 8 secciones, representando las casillas, mediante la cual se recorría la imagen de izquierda a derecha. Esta secuencia continuaba hasta completar una fila antes de avanzar a la siguiente.

Sin embargo, tras la implementación inicial y su uso conjuntamente con la inteligencia artificial, se observó un incremento en el tiempo de ejecución. Para abordar este problema y optimizar el proceso, se propuso un enfoque alternativo. La modificación consistió en la incorporación de una interacción constante entre el algoritmo y el modelo clasificador. A medida que las casillas se recortaban, el modelo clasificador generaba resultados específicos para cada una de ellas. Estos resultados se almacenaban en una matriz, la cual se adaptaba posteriormente para asemejarse a una notación FEN[1].

Esta mejora no solo disminuyó significativamente el tiempo de ejecución, sino que también optimizó la precisión del proceso de recorte y clasificación, aportando resultados más coherentes y cohesivos. En resumen, esta evolución del algoritmo reforzó su capacidad para abordar de manera eficiente y precisa la identificación y segmentación de las casillas del tablero de ajedrez.

5.4. Desarrollo del clasificador

En el marco de este trabajo de investigación, narramos la evolución del proceso de desarrollo y mejora de nuestro modelo clasificador. En la fase inicial, se tomó la decisión de construir el modelo desde sus bases, aprovechando la plataforma TensorFlow[13] como herramienta fundamental. Durante este proceso, se generaron tres variantes de modelos, cada una de ellas representando una iteración en el proceso de optimización.

Es importante comprender algunos términos técnicos que sustentan este proceso del desarrollo del clasificador:

- Tamaño de Lote (Batch Size): El tamaño de lote se refiere a la cantidad de muestras de datos que se utilizan en cada iteración de entrenamiento para actualizar los parámetros del modelo. En lugar de ajustar el modelo con una sola muestra a la vez, el tamaño de lote permite procesar varias muestras juntas, lo que puede mejorar la eficiencia del entrenamiento.
- Maxpooling: El maxpooling es una operación en la que se reduce el tamaño de una matriz o imagen dividiéndola en regiones y seleccionando el valor máximo de cada región. Esta operación ayuda a reducir la cantidad de información al resaltar las características más prominentes y descartar detalles menos importantes.
- Flatten: La operación de aplanar transforma una estructura de datos bidimensional, como una matriz, en una estructura unidimensional, como un vector. En el contexto de las redes neuronales, es común usar flatten después de capas convolucionales o de maxpooling, para transformar la información de la imagen en una forma que pueda ser procesada por capas densas.
- Dense: Una capa densa (también llamada capa completamente conectada) es una capa en una red neuronal en la que cada neurona o nodo está conectado con cada neurona de la capa anterior. Cada conexión tiene un peso asociado, que se ajusta durante el entrenamiento. Esto permite que la capa capture relaciones complejas entre las características de entrada y las utilice para tomar decisiones.
- Dropout: El dropout es una técnica de regularización en la que, durante el entrenamiento, se apagan (se "dejan caer") aleatoriamente un porcentaje de neuronas en una capa.

Esto previene que la red se vuelva demasiado dependiente de ciertas neuronas y evita el sobreajuste. El dropout simula la presencia de diferentes subconjuntos de neuronas en cada iteración de entrenamiento, lo que hace que la red sea más robusta y generalice mejor a nuevos datos.

Una vez se comprenden estos conceptos se comienza a diseñar la arquitectura de los modelos:

El primer modelo fue diseñado con un enfoque introductorio. Se incorporaron tres capas convolucionales, cada una constando de 32 neuronas, seguidas por capas de maxpooling correspondientes. La estructura culminó en una capa “flatten” y una capa de “dropout”. La capa final, de tipo “dense”, realizaba la clasificación empleando la función softmax. Sin embargo, debido a que el modelo era básico, no logró realizar clasificaciones precisas entre ciertas categorías, concretamente alfiles con peones y reinas con torres, además de no discernir los colores de manera adecuada.

El rendimiento nefasto del primer modelo condujo a la reformulación en un segundo intento. Este segundo modelo presentó seis capas convolucionales, cada una con 128 neuronas. Además, se implementaron varias capas de maxpooling y dropout para prevenir el sobreajuste.

A pesar de su mayor complejidad, el segundo modelo no logró superar los desafíos en la clasificación de ciertas piezas y en la identificación precisa de colores. Tras un análisis exhaustivo, se exploró una alternativa que implicó el aprovechamiento del preentrenamiento y la transferencia de aprendizaje.

La transferencia de aprendizaje[22] consiste en aplicar el conocimiento de un modelo previamente entrenado en una tarea similar. En esta instancia, optamos por el modelo VGG16, que previamente demostró habilidad en la detección de objetos en imágenes. Ajustamos la capa final del VGG16 para adaptarlo a la tarea de clasificación de piezas de ajedrez.

Este enfoque de transferencia de aprendizaje permitió utilizar el conocimiento ya adquirido y aplicarlo en específico a la clasificación de piezas de ajedrez. Para esto se necesitaba adaptar las imágenes a un tamaño de (224 x 224) y realizar un pequeño ajuste en la última capa reduciendo el resultado a nuestras 13 categorías. Aunque los resultados fueron bastante buenos, consiguiendo discernir todas las categorías de piezas ocurría que presentaba errores al concretar el color de la pieza.

Observando estos resultados se decidió explorar otros recursos para encontrar una solu-

ción a este problema. Se buscó otro tipo de modelo, en este caso se optó por MobileNetV2 debido a su eficiencia de recursos y una mayor velocidad de inferencia aunque obtendríamos una precisión moderada debido a su eficiencia.

Tras indagar en busca de información para la aplicación de este modelo con la finalidad de aplicar la técnica de “transfer learning”[22], se encontró una serie de tutoriales que utilizan este modelo en la plataforma Pytorch[23]. Por lo que observando que era más intuitiva y con una amplia variedad de recursos en línea, se decidió seguir el desarrollo en esta plataforma.

Aunque para implementar transfer learning con el modelo MobileNetV2 en Pytorch, hay que definir una serie de funciones, que no eran en Tensorflow:

- Cargador de modelo preentrenado: En este apartado se carga el modelo preentrenado para posteriormente quitarle la última capa y sustituirla por una capa que incluya el número de clases que se desean. Este es común a Tensorflow.
- Cargar y procesar los datos. En este apartado se utiliza los dataloaders y transformaciones. Los DataLoaders y las transformaciones son componentes esenciales para cargar y preparar tus datos antes de alimentarlos al modelo durante el entrenamiento o la evaluación. También te separan el conjunto de datos en tres subconjuntos: entrenamiento, validación y test.
- Funciones de pérdida y optimizador: La función de pérdida es una medida de qué tan diferente son las predicciones del modelo en comparación con las etiquetas verdaderas de los datos de entrenamiento. Mientras que el optimizador es responsable de ajustar los pesos del modelo según la función de pérdida calculada durante el entrenamiento.

Vistas estas funciones se procedió primero a modificar la estructura de MobileNetV2, de la forma anteriormente mencionada. Se obtuvo la siguiente arquitectura:

Layer (type)	Output Shape	Param #
Conv2d-1	[-, 32, 112, 112]	864
BatchNorm2d-2	[-, 32, 112, 112]	64
ReLU6-3	[-, 32, 112, 112]	0
Conv2d-4	[-, 32, 112, 112]	288
BatchNorm2d-5	[-, 32, 112, 112]	64
ReLU6-6	[-, 32, 112, 112]	0
Conv2d-7	[-, 32, 112, 112]	512
BatchNorm2d-8	[-, 32, 112, 112]	64
InvertedResidual-9	[-, 16, 112, 112]	32
Conv2d-10	[-, 16, 112, 112]	0
BatchNorm2d-11	[-, 16, 112, 112]	1,536
ReLU6-12	[-, 16, 112, 112]	192
Conv2d-13	[-, 16, 56, 56]	864
BatchNorm2d-14	[-, 16, 56, 56]	192
ReLU6-15	[-, 16, 56, 56]	0
Conv2d-16	[-, 16, 56, 56]	2,364
BatchNorm2d-17	[-, 16, 56, 56]	48
InvertedResidual-18	[-, 16, 56, 56]	0
Conv2d-19	[-, 16, 56, 56]	3,456
BatchNorm2d-20	[-, 16, 56, 56]	288
ReLU6-21	[-, 16, 56, 56]	0
Conv2d-22	[-, 16, 56, 56]	1,296
BatchNorm2d-23	[-, 16, 56, 56]	288
ReLU6-24	[-, 16, 56, 56]	0
Conv2d-25	[-, 16, 56, 56]	3,456
BatchNorm2d-26	[-, 16, 56, 56]	48
InvertedResidual-27	[-, 16, 56, 56]	0
Conv2d-28	[-, 16, 56, 56]	3,456
BatchNorm2d-29	[-, 16, 56, 56]	288
ReLU6-30	[-, 16, 56, 56]	0
Conv2d-31	[-, 16, 28, 28]	1,296
BatchNorm2d-32	[-, 16, 28, 28]	288
ReLU6-33	[-, 16, 28, 28]	0
Conv2d-34	[-, 16, 28, 28]	4,668
BatchNorm2d-35	[-, 16, 28, 28]	64

(a)

(b)

Figura 25: Capas de Arquitectura

Layer (type)	Output Shape	Param #
InvertedResidual-72	[-, 64, 14, 14]	0
Conv2d-73	[-, 384, 14, 14]	24,576
BatchNorm2d-74	[-, 384, 14, 14]	768
ReLU6-75	[-, 384, 14, 14]	0
Conv2d-76	[-, 384, 14, 14]	3,456
BatchNorm2d-77	[-, 384, 14, 14]	768
ReLU6-78	[-, 384, 14, 14]	0
Conv2d-79	[-, 64, 14, 14]	24,576
BatchNorm2d-80	[-, 64, 14, 14]	128
InvertedResidual-81	[-, 64, 14, 14]	0
Conv2d-82	[-, 384, 14, 14]	24,576
BatchNorm2d-83	[-, 384, 14, 14]	768
ReLU6-84	[-, 384, 14, 14]	0
Conv2d-85	[-, 384, 14, 14]	3,456
BatchNorm2d-86	[-, 384, 14, 14]	768
ReLU6-87	[-, 384, 14, 14]	0
Conv2d-88	[-, 64, 14, 14]	24,576
BatchNorm2d-89	[-, 64, 14, 14]	128
InvertedResidual-90	[-, 64, 14, 14]	0
Conv2d-91	[-, 384, 14, 14]	24,576
BatchNorm2d-92	[-, 384, 14, 14]	768
ReLU6-93	[-, 384, 14, 14]	0
Conv2d-95	[-, 384, 14, 14]	3,456
BatchNorm2d-96	[-, 384, 14, 14]	768
ReLU6-97	[-, 384, 14, 14]	0
Conv2d-98	[-, 96, 14, 14]	36,864
InvertedResidual-99	[-, 96, 14, 14]	192
Conv2d-100	[-, 96, 14, 14]	0
BatchNorm2d-101	[-, 96, 14, 14]	55,296
ReLU6-102	[-, 96, 14, 14]	1,152
Conv2d-103	[-, 96, 14, 14]	5,184
BatchNorm2d-104	[-, 96, 14, 14]	1,152
ReLU6-105	[-, 96, 14, 14]	0
Conv2d-106	[-, 96, 14, 14]	55,296
BatchNorm2d-107	[-, 96, 14, 14]	192
InvertedResidual-108	[-, 96, 14, 14]	0
Conv2d-109	[-, 96, 14, 14]	55,296
BatchNorm2d-110	[-, 96, 14, 14]	1,152
ReLU6-111	[-, 96, 14, 14]	0
Conv2d-112	[-, 96, 14, 14]	5,184
InvertedResidual-117	[-, 96, 14, 14]	0
Conv2d-118	[-, 576, 14, 14]	55,296
BatchNorm2d-119	[-, 576, 14, 14]	1,152
ReLU6-120	[-, 576, 14, 14]	0
Conv2d-121	[-, 576, 7, 7]	5,184
BatchNorm2d-122	[-, 576, 7, 7]	1,152
ReLU6-123	[-, 576, 7, 7]	0
Conv2d-124	[-, 160, 7, 7]	92,160
BatchNorm2d-125	[-, 160, 7, 7]	320
InvertedResidual-126	[-, 160, 7, 7]	0
Conv2d-127	[-, 960, 7, 7]	153,600
BatchNorm2d-128	[-, 960, 7, 7]	1,920
ReLU6-129	[-, 960, 7, 7]	0
Conv2d-130	[-, 960, 7, 7]	8,640
BatchNorm2d-131	[-, 960, 7, 7]	1,920
ReLU6-132	[-, 960, 7, 7]	0
Conv2d-133	[-, 960, 7, 7]	153,600
BatchNorm2d-134	[-, 960, 7, 7]	320
InvertedResidual-135	[-, 960, 7, 7]	0
Conv2d-136	[-, 960, 7, 7]	153,600
BatchNorm2d-137	[-, 960, 7, 7]	1,920
ReLU6-138	[-, 960, 7, 7]	0
Conv2d-139	[-, 960, 7, 7]	8,640
BatchNorm2d-140	[-, 960, 7, 7]	1,920
ReLU6-141	[-, 960, 7, 7]	0
Conv2d-142	[-, 160, 7, 7]	153,600
BatchNorm2d-143	[-, 160, 7, 7]	320
InvertedResidual-144	[-, 160, 7, 7]	0
Conv2d-145	[-, 960, 7, 7]	153,600
BatchNorm2d-146	[-, 960, 7, 7]	1,920
ReLU6-147	[-, 960, 7, 7]	0
Conv2d-148	[-, 960, 7, 7]	8,640
BatchNorm2d-149	[-, 960, 7, 7]	1,920
ReLU6-150	[-, 960, 7, 7]	0
Conv2d-151	[-, 320, 7, 7]	307,200
BatchNorm2d-152	[-, 320, 7, 7]	640
InvertedResidual-153	[-, 320, 7, 7]	0
Conv2d-154	[-, 1,280, 7, 7]	409,600
BatchNorm2d-155	[-, 1,280, 7, 7]	2,560
ReLU6-156	[-, 1,280, 7, 7]	0
Dropout-157	[-, 1,280]	0

(a)

(b)

Figura 26: Capas de arquitectura

Layer (type)	Output Shape	Param #
InvertedResidual-135	[-, 160, 7, 7]	0
Conv2d-136	[-, 960, 7, 7]	153,600
BatchNorm2d-137	[-, 960, 7, 7]	1,920
ReLU6-138	[-, 960, 7, 7]	0
Conv2d-139	[-, 960, 7, 7]	8,640
BatchNorm2d-140	[-, 960, 7, 7]	1,920
ReLU6-141	[-, 960, 7, 7]	0
Conv2d-142	[-, 160, 7, 7]	153,600
BatchNorm2d-143	[-, 160, 7, 7]	320
InvertedResidual-144	[-, 160, 7, 7]	0
Conv2d-145	[-, 960, 7, 7]	153,600
BatchNorm2d-146	[-, 960, 7, 7]	1,920
ReLU6-147	[-, 960, 7, 7]	0
Conv2d-148	[-, 960, 7, 7]	8,640
BatchNorm2d-149	[-, 960, 7, 7]	1,920
ReLU6-150	[-, 960, 7, 7]	0
Conv2d-151	[-, 320, 7, 7]	307,200
BatchNorm2d-152	[-, 320, 7, 7]	640
InvertedResidual-153	[-, 320, 7, 7]	0
Conv2d-154	[-, 1,280, 7, 7]	409,600
BatchNorm2d-155	[-, 1,280, 7, 7]	2,560
ReLU6-156	[-, 1,280, 7, 7]	0
Dropout-157	[-, 1,280]	0
Linear-158	[-, 1, 13]	16,653

Figura 27: Capas de arquitectura

Partiendo de esta arquitectura, se llevaron a cabo cuatro variantes distintas de modelos clasificadores, cada uno entrenado con conjuntos de datos específicos. Esta diversificación fue el resultado de un enfoque progresivo en el proyecto, con el objetivo de garantizar el rendimiento del clasificador frente a una serie de desafíos visuales.

A continuación veremos la progresión en la creación de modelos, con sus respectivos resultados y los nuevos desafíos que se tendrían que implementar una vez probado con las diferentes imágenes del dataset. Por otro lado se mostrarán las gráficas producidas durante el entrenamiento y pruebas.

El primer modelo se entrenó con imágenes de alta calidad y posicionamiento preciso dentro de un encuadre central a la casilla correspondiente. Los resultados obtenidos sobre estas imágenes fueron bastante buenos como podemos ver en las siguientes gráficas:

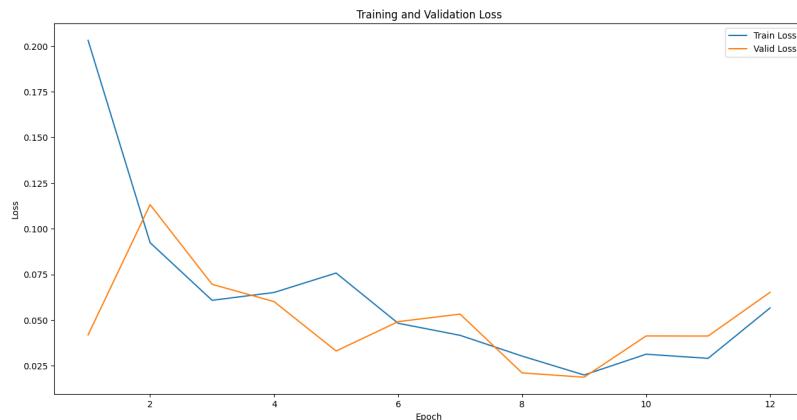


Figura 28: Pérdida en Entrenamiento y Validación del primer modelo



Figura 29: Precisión en Entrenamiento y Validación del primer modelo

En estas gráficas podemos ver la pérdida y la precisión que se producen dentro de las épocas de entrenamiento y la validación de este modelo. Se observa que a medida que se mejora la precisión del clasificador disminuye la pérdida en cada época, menos las últimas épocas. Estos resultados también lo podemos observar en la matriz de confusión en la que vemos que obtenemos pocos falsos positivos y negativos mientras que la tasas de verdaderos positivos es muy alta.

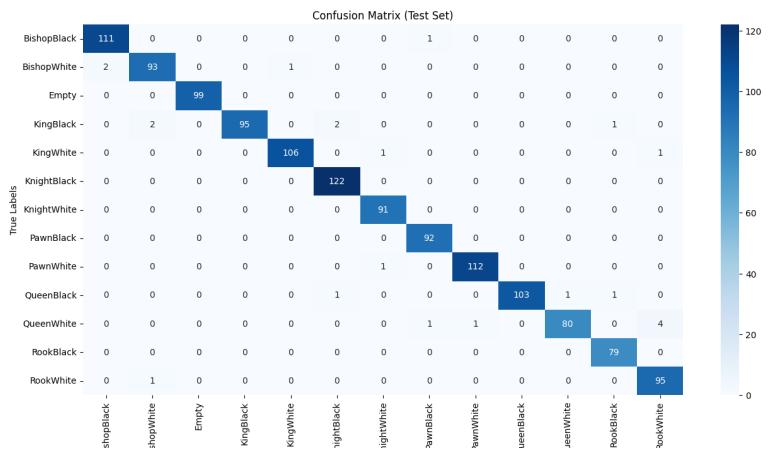


Figura 30: Matriz de Confusión del primer modelo

Sin embargo cuando el clasificador se llevó a la práctica, se enfrentó a imágenes con ciertos

grados de distorsión, donde las predicciones mostraron fallos significativos, lo que puso de manifiesto la necesidad de abordar estas variaciones.

Para abordar esta limitación, se desarrolló el segundo modelo, el cual incorporó variaciones de iluminación y ruido gaussiano en las imágenes de entrenamiento, además de las imágenes nítidas. Los resultados fueron los siguientes:

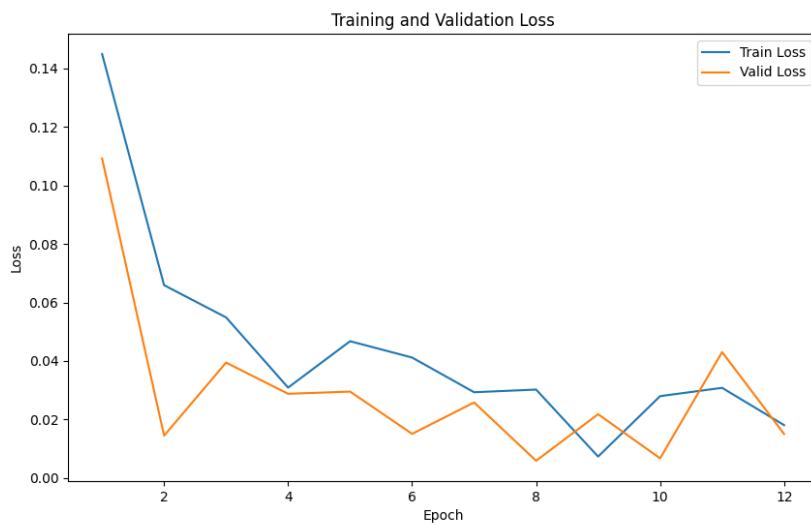


Figura 31: Pérdida en Entrenamiento y Validación del segundo modelo



Figura 32: Precisión en Entrenamiento y Validación del segundo modelo

En estas gráficas obtenemos mejor desempeño por épocas tanto en precisión como en pér-

dida, aunque empezamos a observar la necesidad de acortar el número de etapas en busca de evitar fallos en la precisión como podemos observar en la undécima etapa. Además, observando la matriz de confusión obtenemos que los falsos positivos y negativos son cada vez menos frecuentes. Obteniendo así mejor tasa de verdaderos positivos que el anterior modelo.

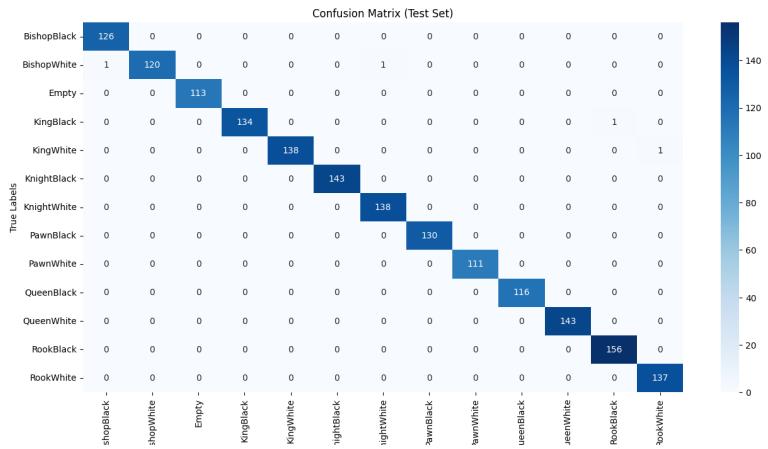


Figura 33: Matriz de Confusión del segundo modelo

Aunque este enfoque fortaleció la capacidad del modelo, se observó que enfrentar un posicionamiento deficiente de las casillas resultaba en predicciones confusas, particularmente en tableros donde las piezas compartían formas similares. Por otro lado, el exceso en el número de épocas provoca disminución en la precisión.

Con el propósito de abordar estas cuestiones, se realizó el tercer modelo. En esta iteración, se amplió el conjunto de datos del segundo modelo con imágenes que presentaban posicionamiento inadecuado, lo que llevaba a secciones fragmentadas. Además se disminuyó el número de épocas. Los resultados fueron los siguientes:

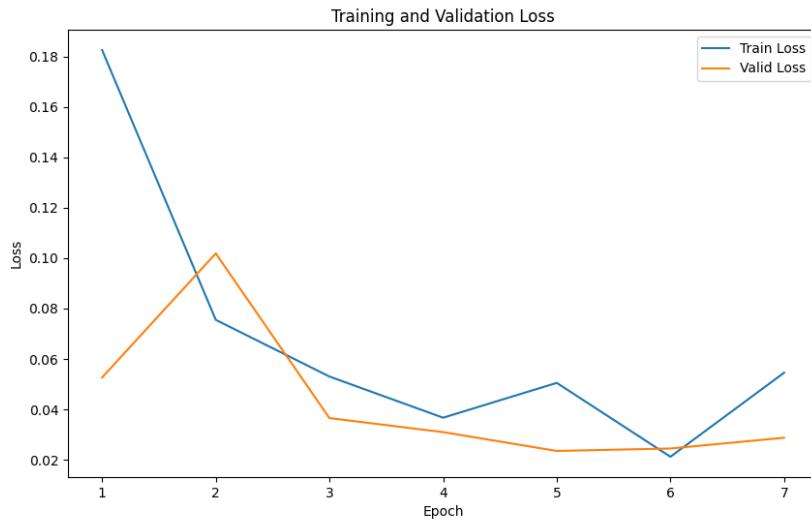


Figura 34: Pérdida en Entrenamiento y Validación del tercer modelo

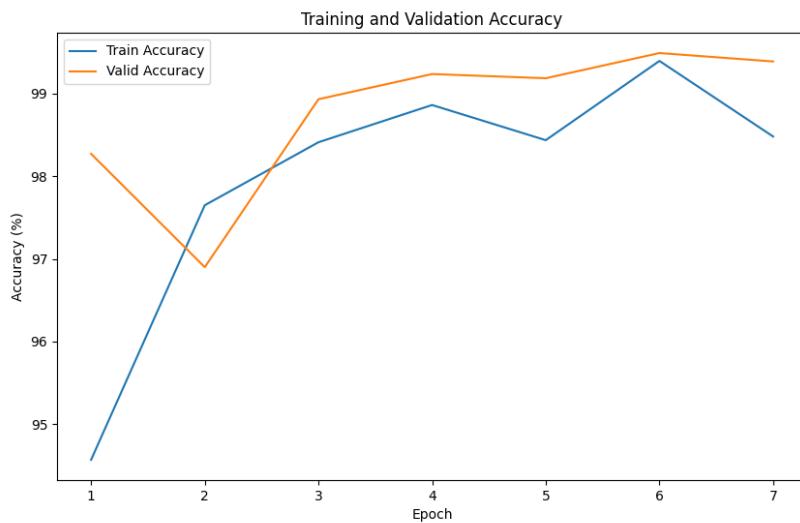


Figura 35: Precisión en Entrenamiento y Validación del tercer modelo

En estas gráficas podemos observar como la precisión disminuye tanto al inicio, que se considera normal, como al final, que se considera anormal. Esto se debe a la reducción significativa de las épocas de entrenamiento y validación acompañado por un aumento en el grado de dificultad de la imagen. Por otro lado, al observar la matriz de confusión en busca de estos errores producidos, obtenemos que la precisión en la clasificación ha sido incluso mejor que el segundo modelo.

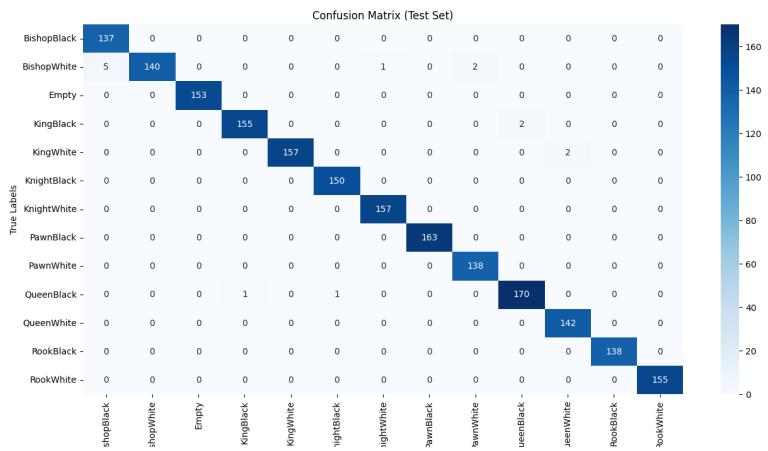


Figura 36: Matriz de Confusión del tercer modelo

Si bien este enfoque mejoró los resultados, surgió un nuevo desafío: al detectar tableros de ajedrez y pasar por cada casilla al clasificador, se detectó una pérdida de resolución y una pixe-lación en las imágenes. Esto generó confusión en la clasificación de formas, como la reina y la torre. Otro aspecto a detallar es que al incrementar el nivel de dificultad al detectar las imáge-nes se incrementaría el número de épocas para evitar el suceso producido por el tercer modelo.

En última instancia, el cuarto modelo incorporó imágenes fuertemente pixeladas, logradas mediante el ajuste del tamaño de las imágenes mediante aumentos y reducciones, además se aumentó el número de épocas. Se obtuvieron los siguientes resultados:

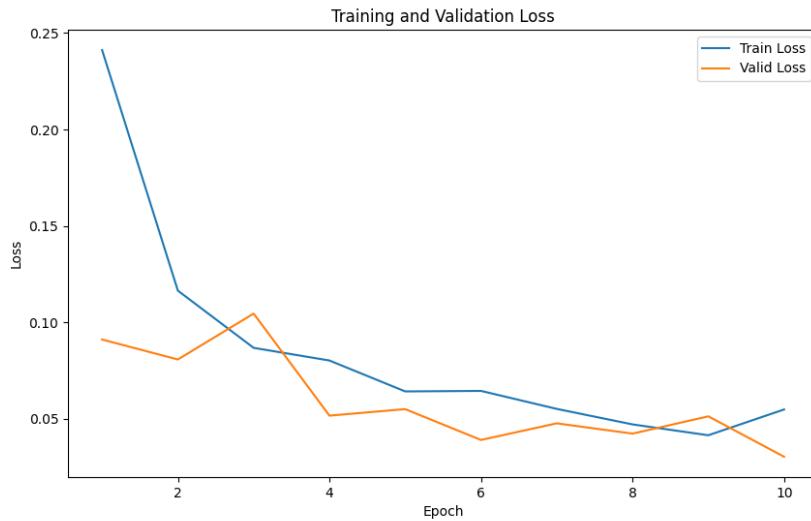


Figura 37: Pérdida en Entrenamiento y Validación del cuarto modelo

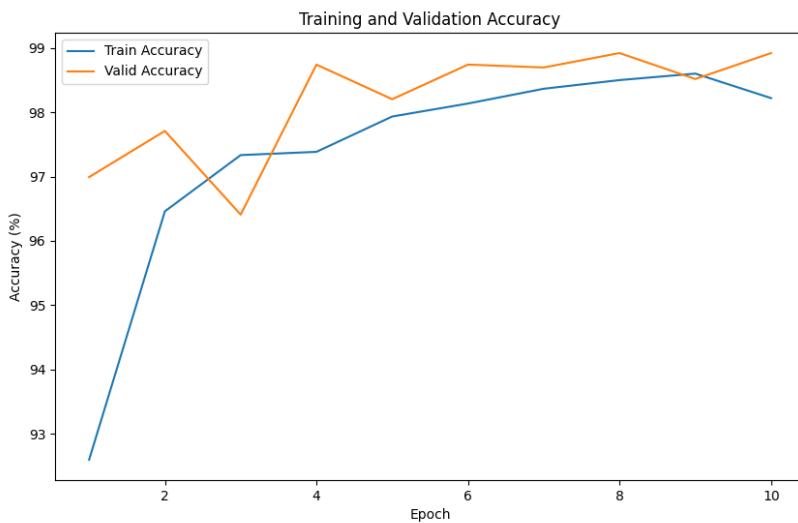


Figura 38: Precisión en Entrenamiento y Validación del cuarto modelo

Como podemos observar en las gráficas, la precisión del modelo aumentó significativamente a la vez que se disminuyó la pérdida, sin producirse valores anormales como el tercer modelo. Por otro lado, la matriz de confusión, aunque presenta más falsos negativos y positivos que los anteriores modelos, sigue presentando una tasa de verdaderos positivos bastante alta.

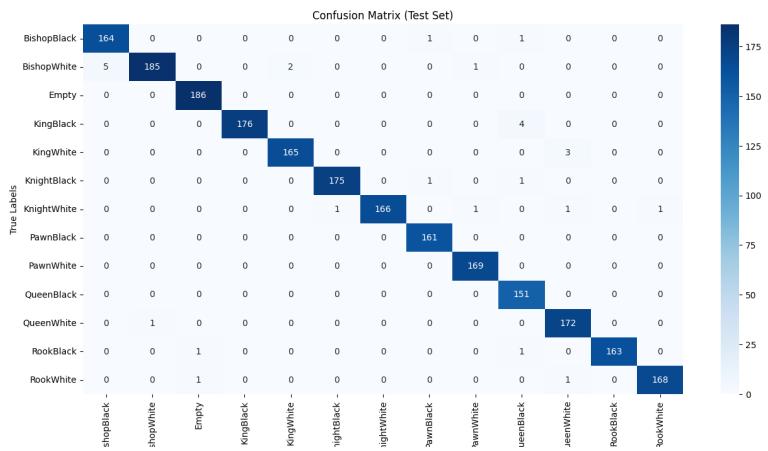


Figura 39: Matriz de Confusión del cuarto modelo

Este enfoque reforzó la robustez del modelo, ya que se demostró que era capaz de lidiar de manera efectiva con las variaciones y procesamientos previos a los que las imágenes habían sido sometidas.

Este proceso de desarrollo iterativo permitió abordar gradualmente las limitaciones identificadas y mejorar la capacidad del clasificador para reconocer y categorizar con precisión las piezas de ajedrez en una amplia gama de condiciones visuales, siendo así más robusto ante estas condiciones.

5.5. Desarrollo del algoritmo tipicador de notación FEN

Durante la fase final de este proyecto, se tomó la decisión de integrar el algoritmo de seccionamiento, junto la función encargada de cargar el modelo clasificador y la función que utiliza este modelo para predecir las piezas alojadas en las casillas.

Este enfoque permitió crear una estructura que contendría todas las piezas del tablero en un orden secuencial, lo cual facilitaría su posterior reorganización mediante una serie de funciones para generar la notación FEN[1].

Esta notación cumple los siguientes requisitos:

- La anotación se hace por filas, empezando siempre por la octava y terminando por la primera: fila 8/fila 7/fila 6/ fila 5/ fila 4/fila 3/ fila 2/ fila 1. En ocasiones se pone cada fila en una línea de texto.
- Las piezas blancas se anotan en mayúsculas: R: rey, D: dama, T: torre, A: alfil, C: caballo, P: Peón. Lo mismo para las negras, pero con letras minúsculas.
- En cada fila iremos anotando cada pieza por orden comenzando siempre por la izquierda. Si nos encontramos casillas vacías pondremos el número de ellas que haya seguidas.
- Una fila que está completamente vacía se anota simplemente con el número 8.
- Se realiza el mismo proceso con cada una de las filas

Una vez entendido las reglas básicas de esta notación se diseñaron las siguientes funciones específicas para normalizar la cadena de caracteres que proporcionaría el modelo clasificador:

- División en filas: Esta función se encargaría de introducir un marcador / cada 8 caracteres, correspondiendo al tamaño de una fila de ajedrez.
- Eliminación de repeticiones numéricas: Esta función tendría la tarea de reducir las repeticiones consecutivas de "1", que representan casillas vacías, sustituyéndolas por la suma total. Cabe mencionar que esta función sería implementada en el proceso posterior y no se requeriría en este contexto, se mantuvo como referencia.

- Conversión a notación FEN: Mediante la combinación de la función de división en filas y la implementación de la función de eliminación de repeticiones numéricas, se obtendría el código resultante que representa la notación FEN, que es esencial en la categorización y representación del estado de un tablero de ajedrez.

6 Fase de Pruebas

6.1. Resultados del desarrollo

Con objetivo de medir la precisión de los aspectos más importantes de la fase de desarrollo como lo son la detección del tablero y la clasificación de piezas, aunque posteriormente se sustituiría por la implementación de una prueba general de la aplicación debido a que abarca el rendimiento de ambas partes en conjunto. Con estos objetivos en mente, se han llevado a cabo las siguientes pruebas:

6.1.1. Pruebas sobre algoritmo de detección

Con el objetivo de comprobar el rendimiento y la eficiencia del algoritmo de detección del tablero de ajedrez, se realizaron un serie de pruebas mediante el uso del conjunto de pruebas previamente registrado.

Para ello se obtendría la tasa de acierto correspondiente entre las coordenadas detectadas y las coordenadas exactas del tablero en la misma imagen. Esto se realiza mediante la métrica “Intersection over Union”. Esta métrica se utiliza para evaluar qué tan bien se superponen o se ajustan las regiones detectadas por un algoritmo en comparación con las regiones de referencia, dando como resultado un porcentaje de precisión que se comprende entre 0 y 1.

La métricas usadas para saber la precisión del tablero son la media y la desviación típica, debido a que necesitamos un valor medio representativo de todos los casos realizados en cada plataforma y como de dispersos están los datos.

Antes de presentar los datos, es importante resaltar que estas pruebas se llevaron a cabo en la sección semi-automatizada. Esto se debe a que, en casos donde el tablero no sea detectado, se espera que la selección sea realizada manualmente por parte del usuario.

En la imagen siguiente, se presenta una visualización del rendimiento en cada plataforma, tanto en entornos reales como en entornos sintéticos. Además, se proporciona un análisis se-

parado del rendimiento en los entornos reales y en los entornos sintéticos:

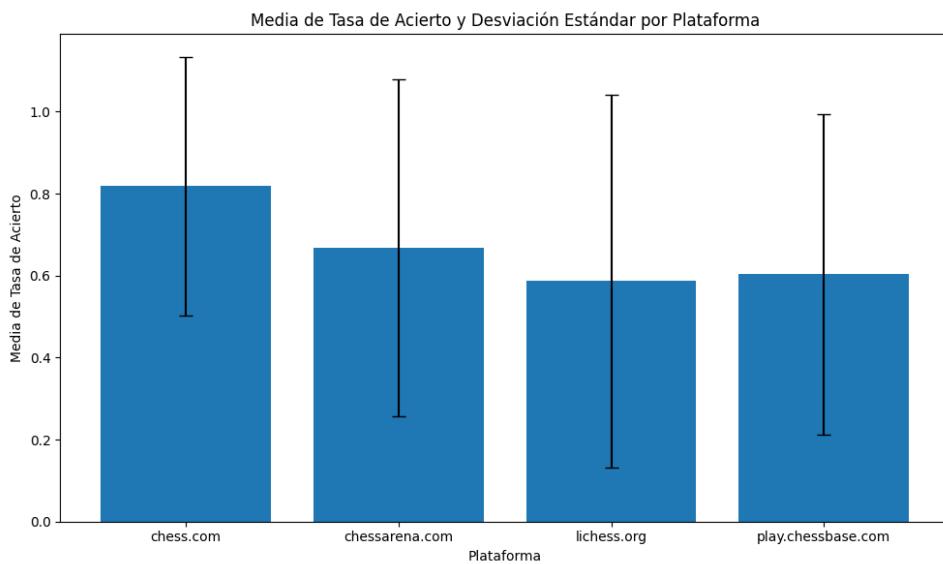


Figura 40: Tasas de acierto por plataforma

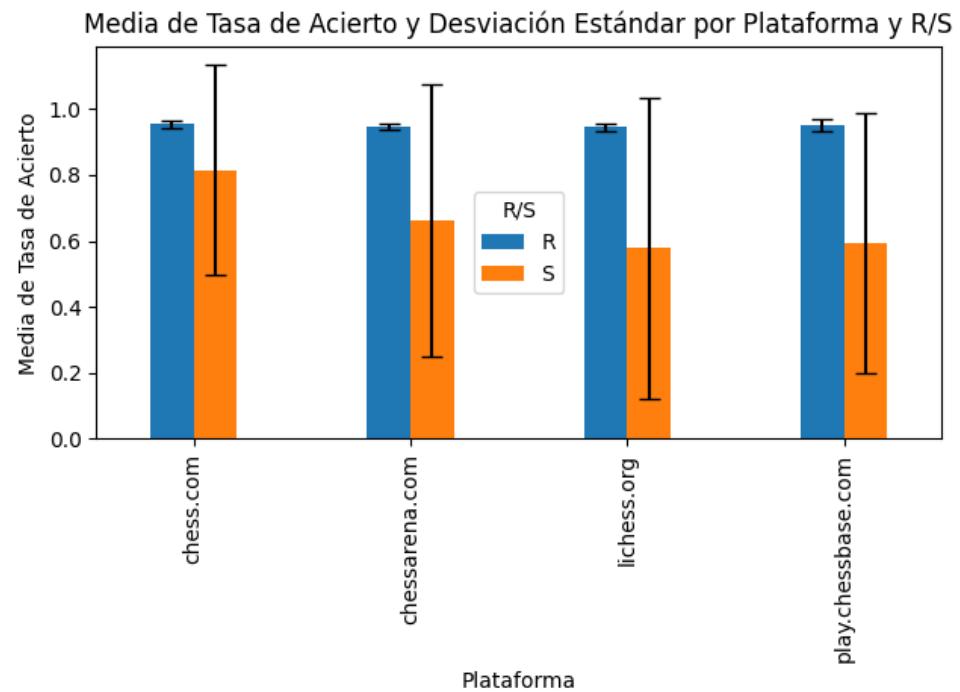


Figura 41: Tasas de acierto en plataformas reales y su versión sintética

A la vista de los resultados, se observa que hay distintos parámetros que están involucrados produciendo que la detección del tablero no sea tan exacta. Por lo que procedemos a observar como afecta cada parámetro a la tasa de acierto, para comprender la naturaleza de los datos obtenidos.

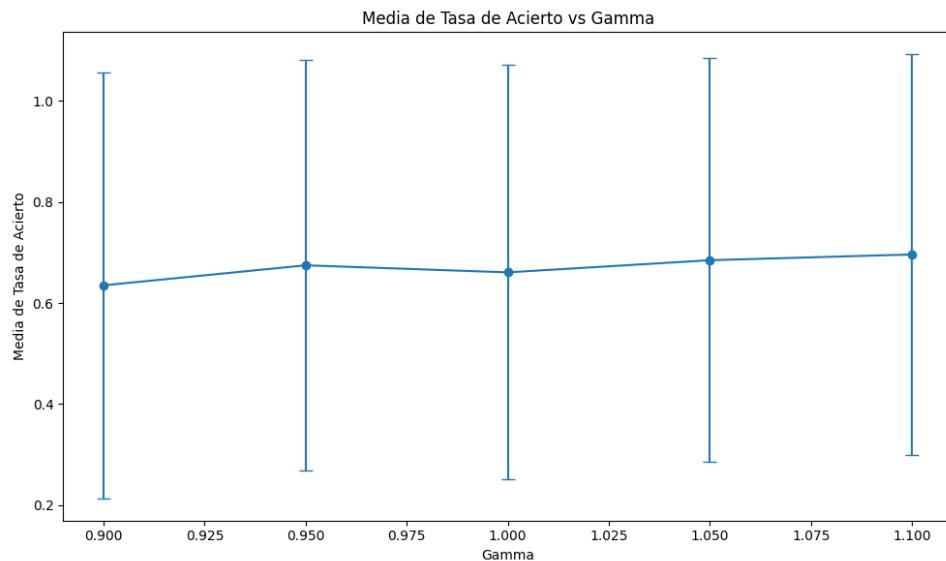


Figura 42: Influencia del parámetro gamma

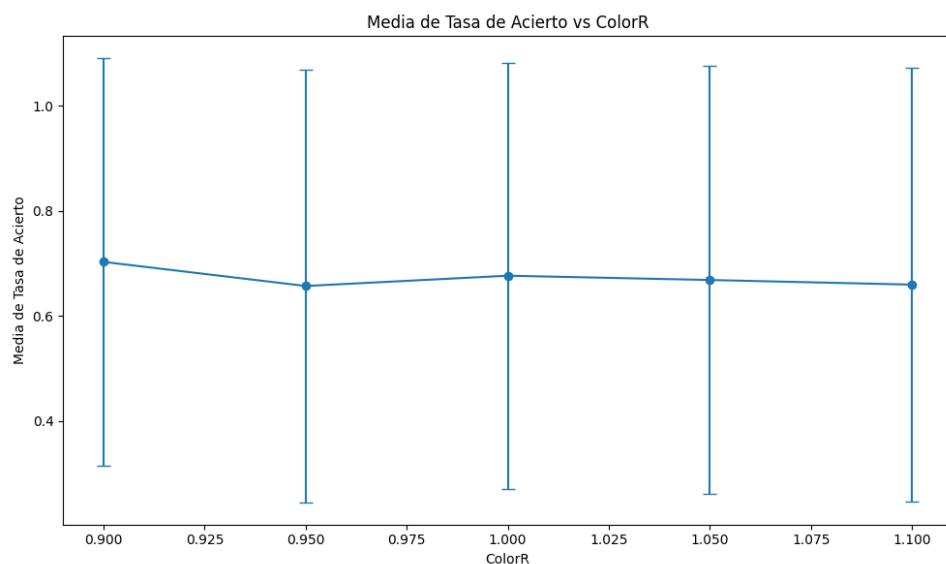


Figura 43: Influencia del parámetro color rojo

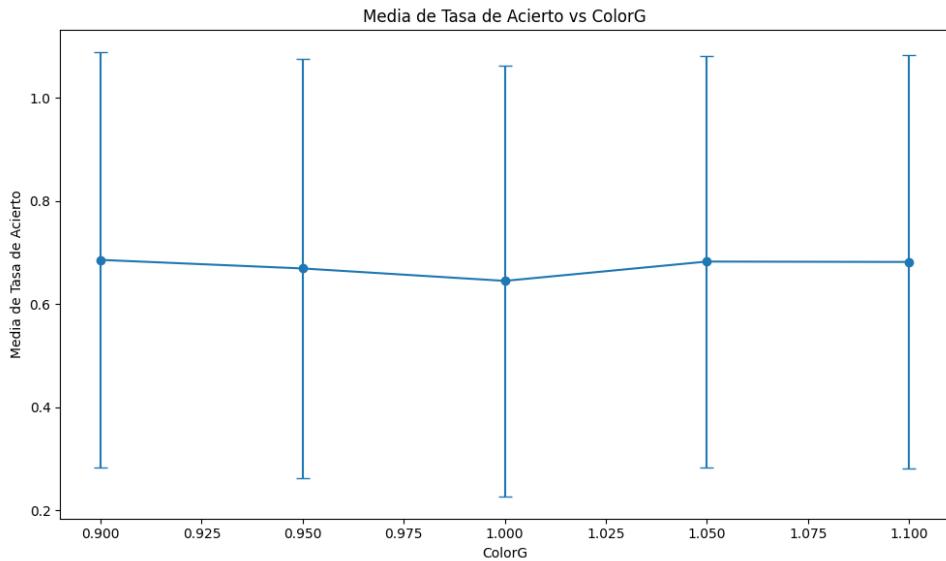


Figura 44: Influencia del parámetro color verde

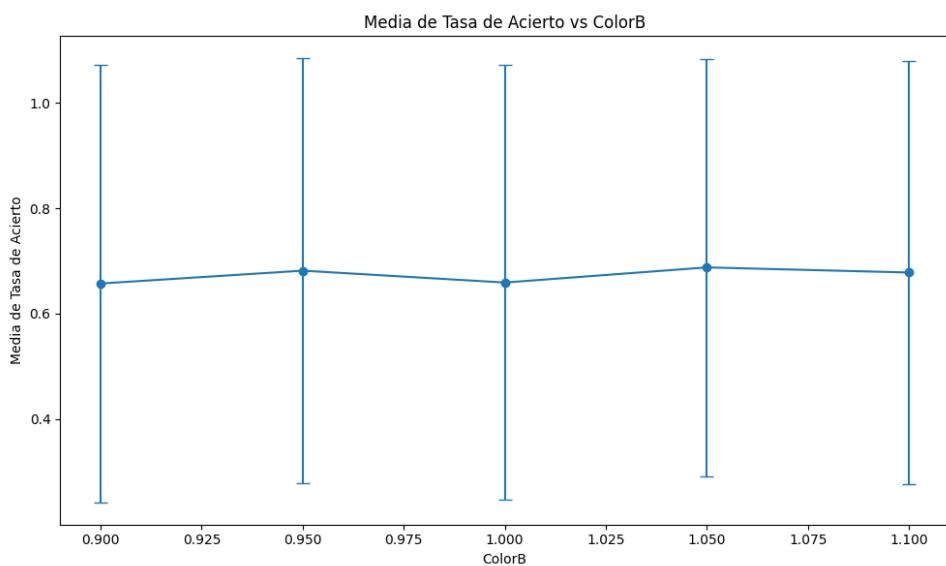


Figura 45: Influencia del parámetro color azul

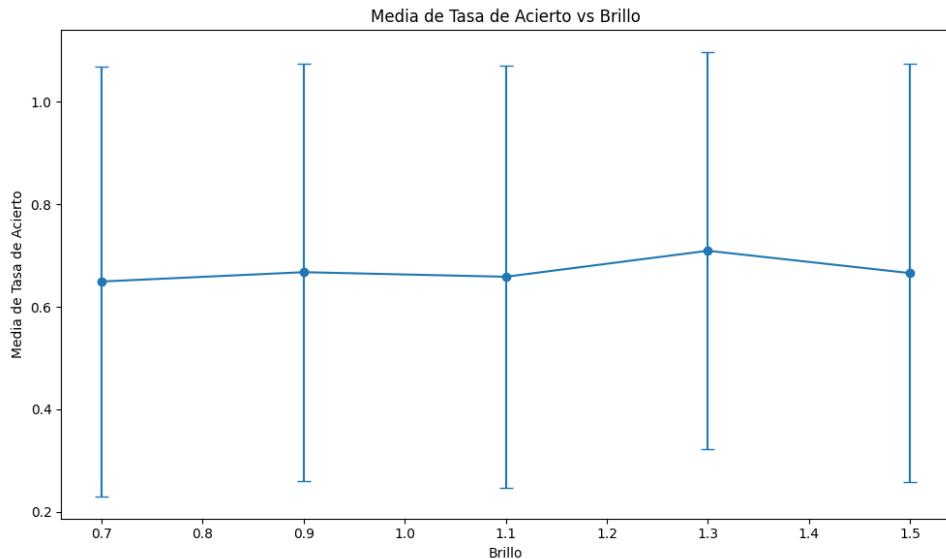


Figura 46: Influencia del parámetro brillo

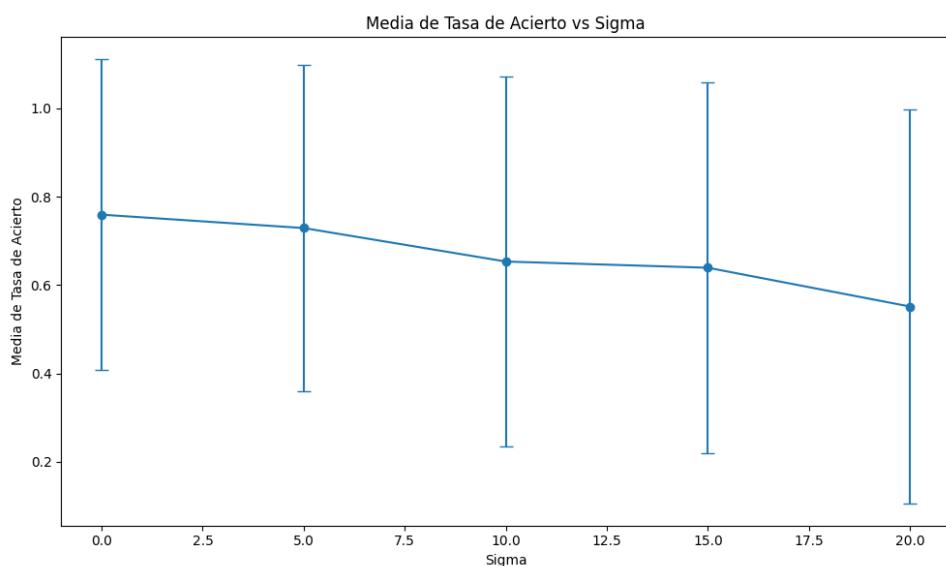


Figura 47: Influencia del parámetro sigma

Tal como se aprecia en las representaciones gráficas de los parámetros, se entiende que los factores que más inciden en la complicación del proceso de detección del tablero son el brillo y el valor sigma. Respecto al brillo, se puede constatar que tanto valores demasiado bajos como excesivamente altos conllevan a una reducida probabilidad de acierto en la detección del tablero. En cuanto al valor sigma, que corresponde al nivel de ruido gaussiano, se observa que un aumento en su magnitud resulta en resultados igualmente desfavorables. Tal como se apre-

cia en las representaciones gráficas de los parámetros, se desprende que los factores que más inciden en la complicación del proceso de detección del tablero son el brillo y el valor sigma. Respecto al brillo, se puede constatar que tanto valores demasiado bajos como excesivamente altos conllevan a una reducida probabilidad de acierto en la detección del tablero. En cuanto al valor sigma, que corresponde al nivel de ruido gaussiano, se observa que un aumento en su magnitud resulta en resultados igualmente desfavorables.

6.1.2. Resultados del programa completo

Tras la finalización del desarrollo del programa, se procedió a llevar a cabo pruebas utilizando el conjunto de datos empleado en la creación y en la realización de pruebas del algoritmo de detección del tablero. El objetivo primordial de estas pruebas radicaba en evaluar el desempeño de la aplicación en la detección y clasificación del tablero. Los resultados obtenidos se detallan a continuación:

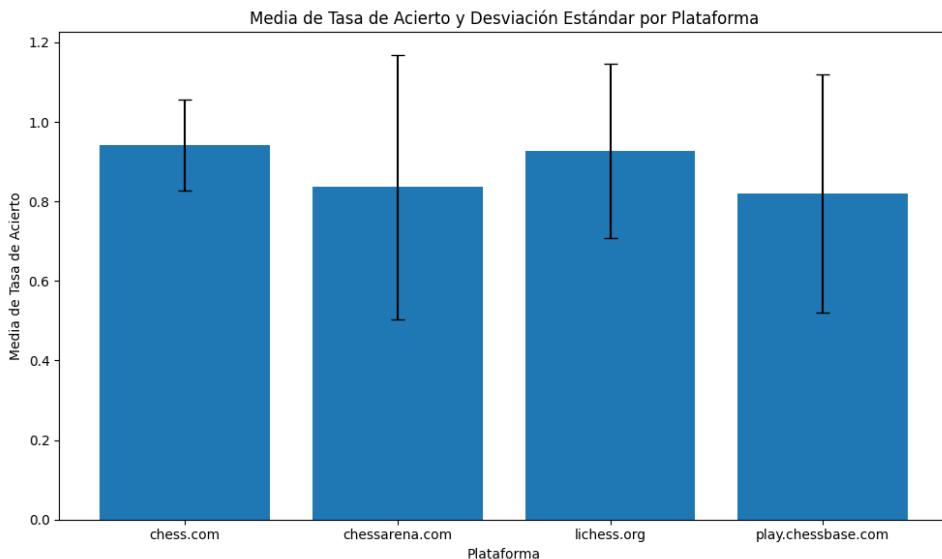


Figura 48: Resultados por plataforma

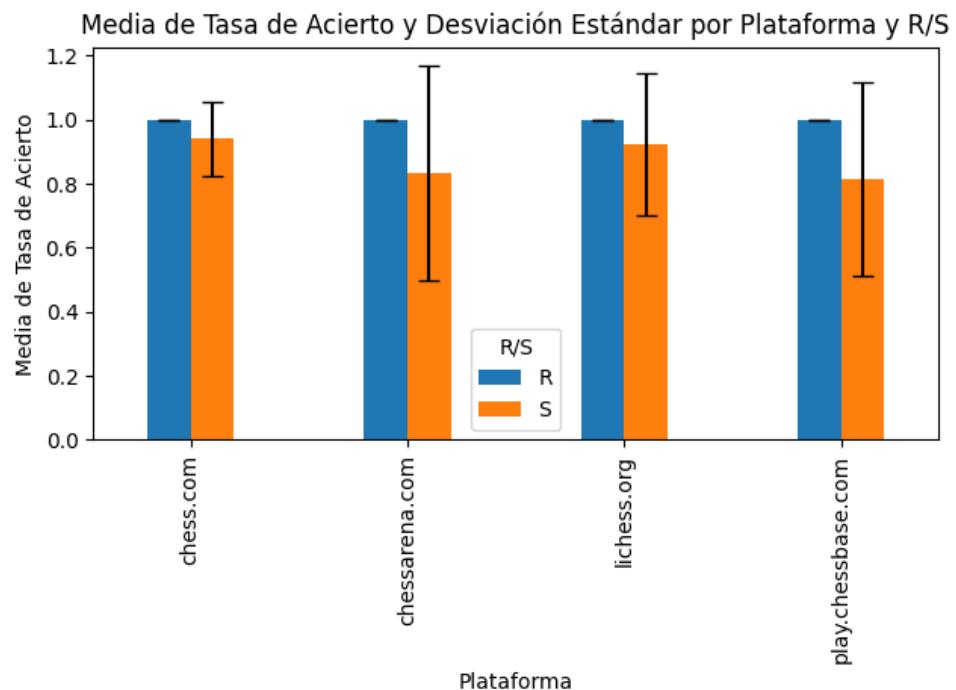


Figura 49: Resultados por plataforma sobre imágenes reales y sintéticas

A la vista de los resultados, se observa que hay variaciones en el resultado final del programa. Por lo que procedemos a observar como afecta cada parámetro a la tasa de acierto, bajo el mismo método practicado al algoritmo de detección:

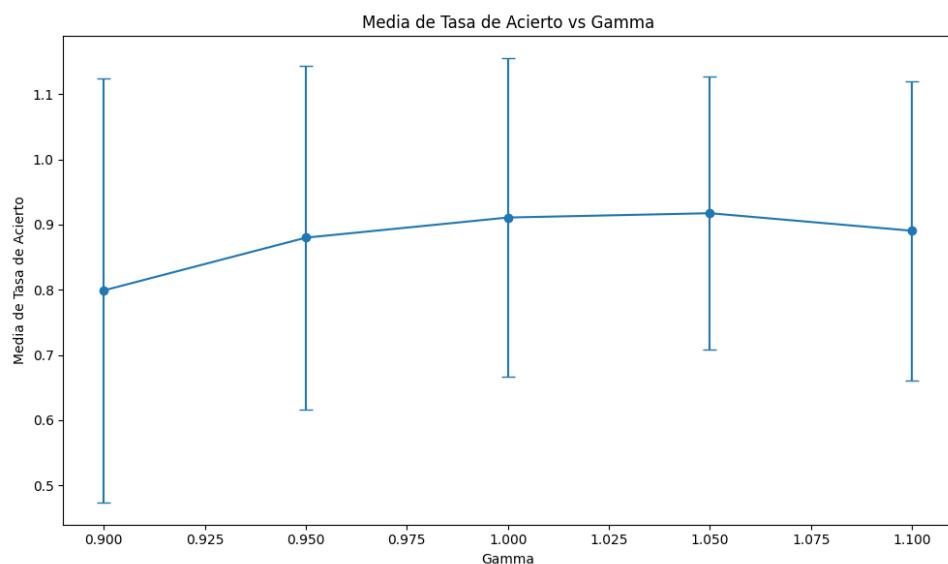


Figura 50: Influencia el parámetro gamma

Cómo vemos en la gráfica, cuánto menor es el parámetro gamma, menor contraste, más oscuridad presenta la imagen. Este hecho produce que el tablero junto con las piezas sea más difícil de detectar, debido a que no se puede diferenciar correctamente las líneas y las formas de las piezas.

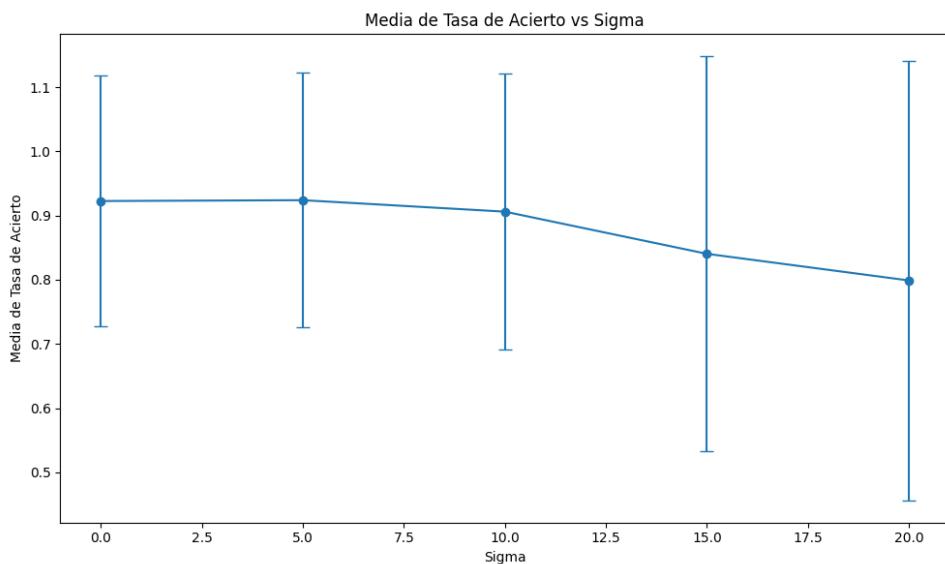


Figura 51: Influencia del parámetro sigma

En esta gráfica podemos observar como el parámetro sigma encargado de la producción de ruido gaussiano, solo afecta cuánto mayor sea el parámetro debido a que la producción de puntos en la imagen dificulta la tarea de detección de líneas y la clasificación de piezas, debido a que por una parte se crean demasiadas líneas que son imposibles de procesar y por otro lado, no se puede diferenciar las formas piezas correctamente.

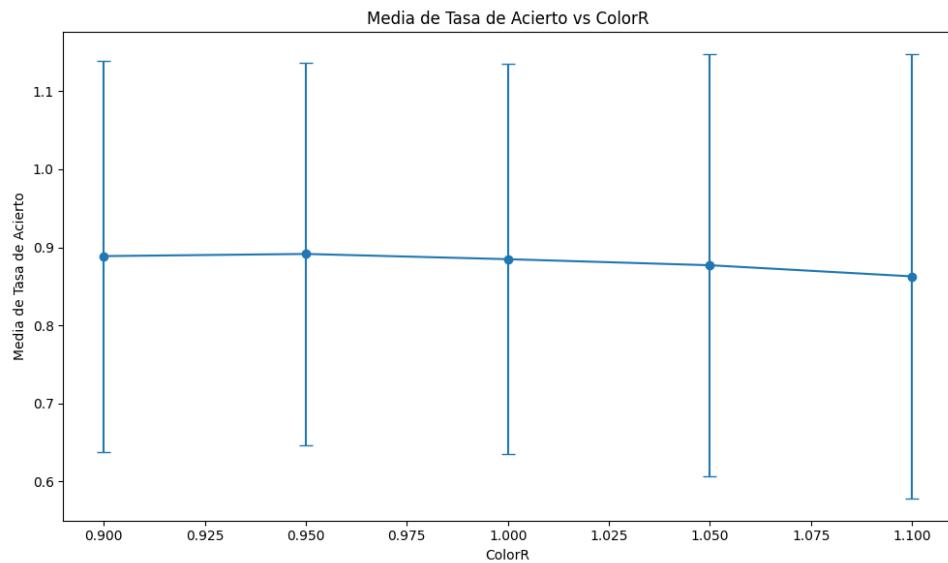


Figura 52: Influencia del parámetro color rojo

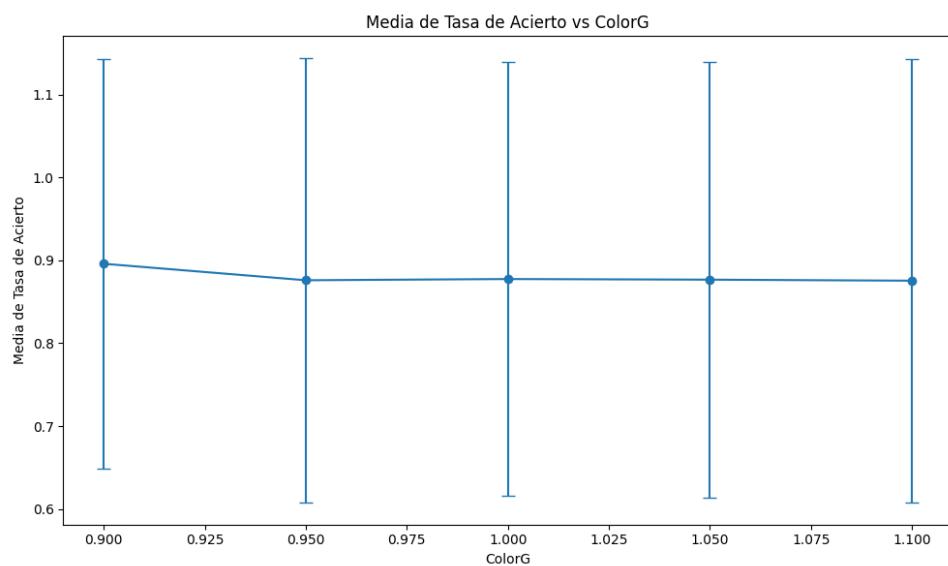


Figura 53: Influencia del parámetro color verde

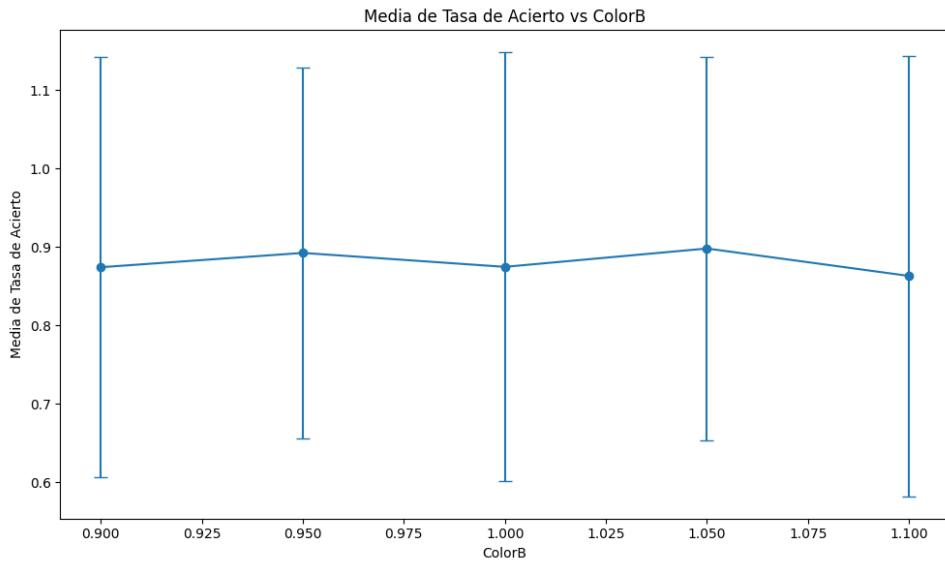


Figura 54: Influencia del parámetro color azul

En las gráficas relacionadas con el color, podemos observar que el color determinante es el rojo, debido a que presenta una relación de menor tasa de acierto con respecto al aumento del mismo, mientras que en los parámetros verde y azul se mantienen en la mayoría de las ocasiones.

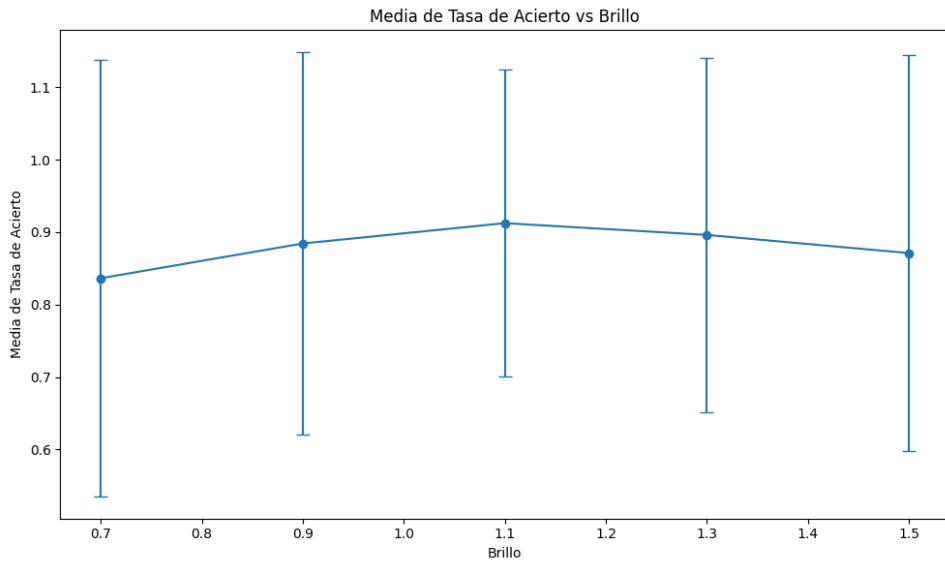


Figura 55: Influencia del parámetro brillo

La influencia del brillo en la detección y la clasificación es evidente, en ambas partes podemos observar como un brillo bajo o intenso puede producir un impacto negativo en ambos

procesos.

Los análisis completos del programa indican la presencia de cuatro categorías de parámetros que ejercen un impacto directo en el rendimiento del mismo. Estos parámetros son los siguientes: el exceso o la deficiencia en la luminosidad, el aumento del valor del parámetro sigma, la saturación excesiva en el canal de color rojo y la reducción del parámetro gamma. La existencia de cualquiera de estos parámetros en la imagen provoca una disminución en la precisión del programa, una situación que no se verificaría bajo circunstancias normales.

7 Conclusiones y Líneas futuras

7.1. Conclusiones

La memoria abarca en detalle el proceso realizado para desarrollar una aplicación de escritorio con la capacidad de detectar tableros de manera semiautomática y generar notaciones que permitan reconstruirlos con precisión a partir de las capturas originales. El objetivo principal ha sido afrontar este desafío mediante la combinación de diversos enfoques y técnicas provenientes de la visión por computador e inteligencia artificial.

En pos de lograr una detección precisa de los tableros, se han explorado diferentes métodos. Destaca entre ellos el empleo de la transformada de Hough, tanto en su variante tradicional como probabilística. Además, se han desarrollado algoritmos específicos basados en estos métodos de detección de líneas. Un enfoque particular ha consistido en aplicar máscaras para identificar figuras geométricas relacionadas con los tableros, lo que ha mejorado la precisión en algunas detecciones. También se ha implementado la delimitación de zonas en las proximidades de los puntos de interés para identificar líneas perpendiculares y definir con precisión los puntos de intersección, aunque este método puede ser sensible al tamaño de la zona.

En el proceso de mejora de la detección, se han experimentado con diversas técnicas de preprocesamiento de imágenes. Esto ha incluido la utilización de filtros gaussianos para reducir el ruido, el empleo del operador Sobel para resaltar bordes y líneas, la aplicación del operador Laplaciano para identificar características, así como la conversión de imágenes a escala de grises. También se han realizado ajustes en el brillo y contraste, y se ha aplicado el algoritmo de detección de bordes de Canny. Esto ha resultado en la creación de un sistema adaptable de detección de bordes que combina un filtro de mediana, un filtro gaussiano y el algoritmo de Canny.

En cuanto al desarrollo del clasificador, se ha trabajado inicialmente con la plataforma Tensorflow. Sin embargo, considerando la complejidad y recursos requeridos para crear un modelo clasificador desde cero, se ha optado por aprovechar modelos preentrenados de redes convo-

lucionales. Esta elección ha acelerado el proceso de obtener un clasificador personalizado para los propósitos específicos del proyecto. Se ha contemplado la opción de utilizar PyTorch, que brinda mayor flexibilidad para cargar, modificar y adaptar modelos existentes a las necesidades del proyecto. Finalmente, la obtención del clasificador se ha llevado a cabo utilizando la plataforma PyTorch.

En conjunto, esta labor de investigación y desarrollo ha proporcionado un entendimiento profundo de las técnicas de visión por computador e inteligencia artificial necesarias para abordar el desafío planteado. La convergencia de enfoques, desde la detección de tableros a través de métodos como la transformada de Hough y el uso de máscaras, hasta la mejora de la precisión mediante técnicas de preprocesamiento y la adaptación de modelos de redes convolucionales preentrenados, ha sentado las bases para una aplicación de escritorio capaz de detectar tableros y generar notaciones. Cabe mencionar que, aunque la solución es sólida en general, presenta cierta vulnerabilidad ante variaciones en el tamaño, cambios de contraste y brillo o presencia de ruido exacerbados que influyen directamente a la detección y clasificación.

7.2. Líneas futuras

Dentro de las posibilidades que se entrelazan con este trabajo, es posible explorar diversas orientaciones que servirían para ampliar y enriquecer aún más la funcionalidad y la utilidad inherentes a la aplicación desarrollada. Algunas de estas oportunidades son las siguientes:

- Refinamiento del algoritmo de detección: El enfoque de esta dirección sería alcanzar una mejora sustancial en la obtención del tablero a través de un proceso completamente automático. De esta manera, se otorgaría al usuario el rol exclusivo de proporcionar la imagen al programa, delegando al algoritmo la tarea de realizar una detección precisa. Este procesamiento automático podría requerir el uso de inteligencia artificial para la detección del tablero y posteriormente aplicar los tratamientos necesarios.
- Implementación de una interfaz intuitiva: Otra posibilidad sería diseñar una interfaz de usuario que comprenda y organice las opciones disponibles para el usuario. Esta interfaz no solo facilitaría el proceso de interacción, sino que también contribuiría a agilizar el aprendizaje y el uso de la aplicación en general.

- Integración en plataformas de ajedrez en línea: Una vía de desarrollo valiosa sería permitir que los usuarios carguen imágenes de sus partidas directamente desde plataformas de ajedrez en línea hacia la aplicación. Esto proporcionaría una detección y notación expedita y precisa de los tableros, lo que podría tener un impacto significativo en la experiencia de juego.
- Generación de análisis y estadísticas: Una dirección interesante sería desarrollar un módulo de análisis que examine las partidas notadas y proporcione estadísticas sobre el desempeño del jugador. Esto podría incluir información sobre las jugadas más comunes, las aperturas preferidas y las áreas en las que el jugador podría mejorar. Además, podría ofrecer sugerencias basadas en el análisis de movimientos y patrones previos.

En resumen, estas oportunidades encajan de manera coherente con el proyecto y su potencial evolutivo, permitiendo que la aplicación crezca en funcionalidades y se ajuste a las necesidades cambiantes de los usuarios.

Apéndice A

Apéndice: Manual de Instalación

Este manual tiene la funcionalidad de realizar la instalación del trabajo. Este trabajo se puede instalar de dos formas, manual y automática:

Forma manual:

Instala Python, la versión utilizada en el desarrollo es la 3.11.4, aunque cualquier versión por encima de la 3.9 es compatible con este proyecto. Se puede encontrar en este enlace: <https://www.python.org/downloads/>

Descarga el proyecto proporcionado, o bien dirígete al siguiente enlace donde también se almacena el proyecto: <https://github.com/sergio4/ChessProject>

Puedes crear un entorno virtual para el despliegue y utilización del proyecto(opcional):

```
>python -m venv venv  
>venv\Scripts\activate.bat
```

Posterior a esto, debes instalar las dependencias y librerías asociadas al proyecto(si has hecho el entorno virtual debes de hacerlo sobre ese entorno):

```
>python -m pip install -r requirements.txt
```

Introduce la foto en el dataset del proyecto que se encuentra dentro de la siguiente ruta:

```
\transform_images\dataset
```

Por último, ejecutas en la linea de comandos del sistema lo siguiente:

```
>C:\Users\...\ProyectoChess>python "ruta_al_archivo\image2Fen.py  
" "nombre_de_tu_imagen"
```

Con esto ya empezaría a funcionar el programa.

Forma automática:

Cabe resaltar que la forma automática no crea un entorno virtual, debido a que es un paso opcional del usuario final, pero puede ser ejecutado dentro del entorno virtual y funcionar de la misma manera:

Instala Python, la versión utilizada en el desarrollo es la 3.11.4, aunque cualquier versión por encima de la 3.9 es compatible con este proyecto. Se puede encontrar en este enlace: <https://www.python.org/downloads/>

Descarga el proyecto proporcionado, o bien dirígete al siguiente enlace donde también se almacena el proyecto y descargue el archivo setup.py: <https://github.com/serg6io4/ChessProject>

Puedes crear un entorno virtual para el despliegue y utilización del proyecto(opcional):

```
>python -m venv venv  
>venv\Scripts\activate.bat
```

Ejecute el archivo setup.py, si tiene un entorno virtual ejecute el archivo en el entorno virtual:

```
>python "ruta_hasta_archivo\setup.py" "  
ruta_para_guardar_directorio\nombre_directorio"
```

Con este comando se ejecutará setup.py, que se encargará de descargar todo el sistema de archivos y dependencias que necesite el mismo proyecto.

Introduce la foto en el dataset del proyecto que se encuentra dentro de la siguiente ruta:

```
\transform_images\dataset
```

Por último, ejecutas en la linea de comandos del sistema lo siguiente:

```
>C:\Users\...\ProyectoChess>python "ruta_al_archivo\image2Fen.py"  
" "nombre_de_tu_imagen"
```

Con esto ya empezaría a funcionar el programa.

Apéndice B

Apéndice: Manual de Usuario

Dentro de este manual se especifica los pasos a realizar para un correcta ejecución de la aplicación:

Primero debes meter tu imagen en el dataset correspondiente, para ello nos vamos a la ruta específica, que será en la carpeta dataset1, que está en transformimages, como vemos en la siguiente imagen:

Nombre	Fecha de modificación	Tipo	Tamaño
chess24-0010-1690456332487.png	27/07/2023 13:12	Archivo PNG	188 KB
chess24-0010-1690456332487.txt	27/07/2023 13:12	Documento de tex...	1 KB
chess24-0010-1690456332487-color.png	27/07/2023 13:12	Archivo PNG	167 KB
chess24-0010-1690456332487-colorgaus...	27/07/2023 13:12	Archivo PNG	321 KB
chess24-0010-prediction.txt	26/07/2023 11:45	Documento de tex...	1 KB
ejemplo.png	13/08/2023 14:03	Archivo PNG	300 KB
fideonlinearena-0001.png		Tipo de elemento: Archivo PNG	191 KB
fideonlinearena-0001.txt		Dimensiones: 1920x1980.06	Documento de tex...
fideonlinearena-0001-1690456374091.png	27/07/2023 13:12	Tamaño: 299 KB	Archivo PNG

Figura 56: Insertar imagen en el dataset

Una vez tenemos la imagen dentro del dataset, vamos a nuestro símbolo de sistema:



Figura 57: Accediendo a la terminal

Nos situamos en la carpeta principal del proyecto, que debería ser ProyectoChess. En este punto ejecutamos el comando para usar el programa que en este ejemplo, se observa de la siguiente manera:

```
C:\Users\sergi\Desktop\ProyectoChess>python C:\Users\sergi\Desktop\ProyectoChess\Code\image2fen.py chess-0002.png
```

Figura 58: Ejecutar el comando

Al ejecutar el comando, se abrirá una ventana con la imagen y tendrá que seleccionar los puntos donde se observa el tablero:

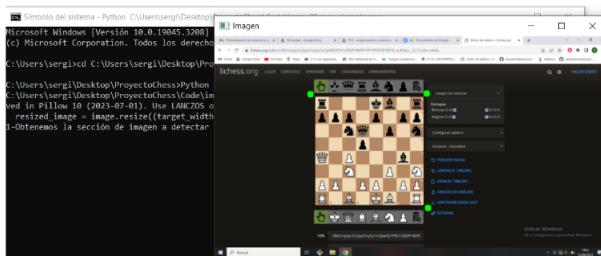


Figura 59: Seleccionando puntos

Posterior a esto, le irá mostrando pantallas en las que se detectan los bordes de la imagen detectados, los puntos de intersección correspondiente a las líneas detectadas y la imagen final que se le pasará al clasificador:

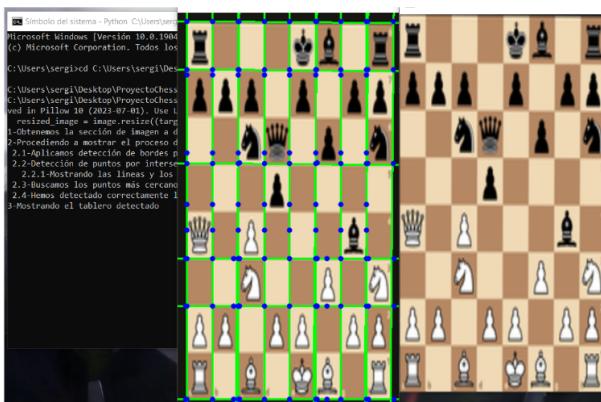


Figura 60: Mostrando Intersecciones y Líneas

Por último el programa detectará las piezas de cada una de las casillas dando como resultado una notación FEN correspondiente al tablero:

```

1-Obtenemos la sección de imagen a detectar
2-Procediendo a mostrar el proceso de detección
2.1-Aplicamos detección de bordes por algoritmo Canny
2.2-Detección de puntos por intersecciones de líneas detectadas por Hough
2.2.1-Mostrando las líneas y los puntos encontrados en la imagen
2.3-Buscamos los puntos más cercanos a la esquina del tablero
2.4-Hemos detectado correctamente los puntos
3-Mostrando el tablero detectado
4-Redimensionando el tablero a cuadrado perfecto
5-Cargando el modelo para predecir piezas
6-Realizando tipificación del código al código FEN
FEN NOTATION:
r3kb1r/ppp1p1pp/2nq1p1n/3p4/Q1P3b1/2N2P1N/PP1PP1PP/R1B1KB1R
  
```

Figura 61: Presentación de pasos y obtención de la notación FEN

En caso de que no detecte el tablero, tendrá que seleccionar las esquinas del tablero de forma precisa, como se ve a continuación:

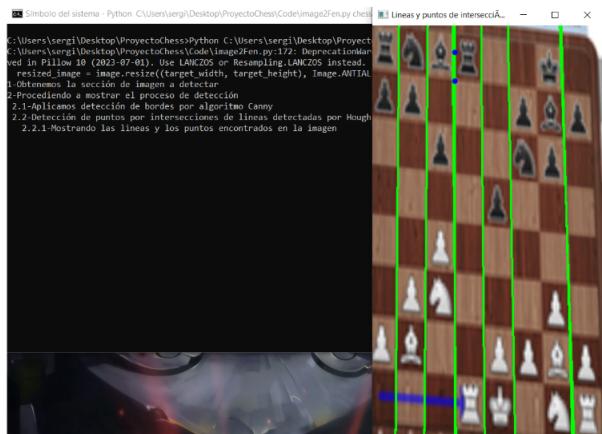


Figura 62: No se detecta el tablero

En este caso no se detecta las intersecciones correspondientes por lo que procedemos a realizar una selección precisa de las esquinas:

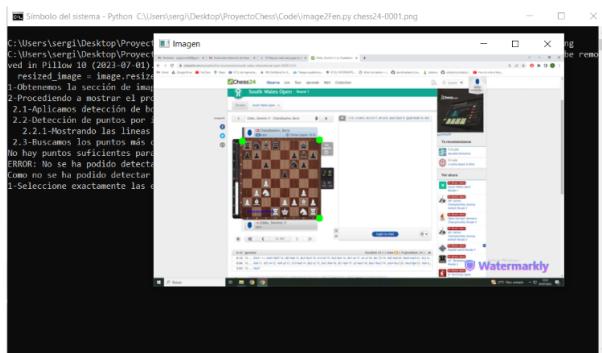


Figura 63: Selección precisa de puntos

Después de esto, el programa cortará la imagen:

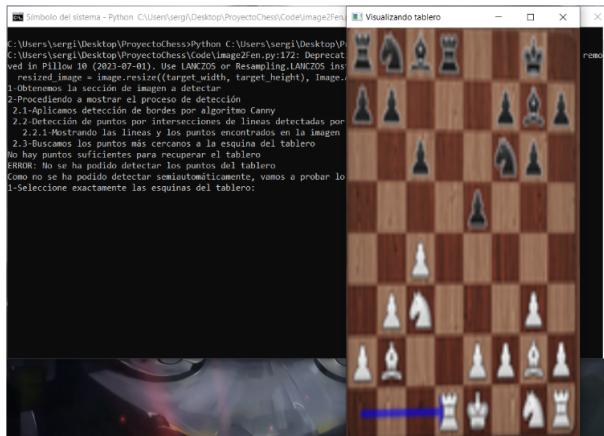


Figura 64: Imagen recortada

Y se la pasará al clasificador que procederá a redactar la notación FEN:

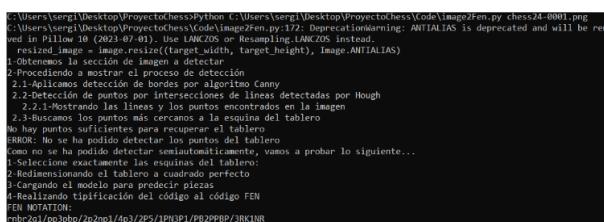


Figura 65: Obtención Notación FEN

Una vez obtenida la notación, y mediante el uso de una plataforma de ajedrez online puedes recrear la partida desde el punto capturado. Una de las plataformas que te permite realizar esto es el editor de tablero de lichess.org: <https://lichess.org/editor>

Apéndice C

Apéndice: Glosario

Canny: Un algoritmo de detección de bordes en imágenes que utiliza una serie de operaciones, como el filtrado Gaussiano y el cálculo del gradiente, para identificar los bordes en una imagen.

Big Data: Refiere al manejo y análisis de grandes volúmenes de datos que exceden la capacidad de las herramientas de software tradicionales para gestionarlos y procesarlos.

Filtro Gaussiano: Un tipo de filtro utilizado para suavizar una imagen eliminando el ruido y detalles pequeños, aplicando una función Gaussiana ponderada a los píxeles.

Filtro Mediana: Un filtro utilizado para suavizar una imagen mediante la sustitución de cada píxel por el valor mediano de los píxeles en su vecindario, lo que ayuda a eliminar el ruido impulsivo.

Kernel: Una matriz utilizada en procesamiento de imágenes para aplicar operaciones como convoluciones y filtros a una imagen. El tamaño y los valores del kernel determinan el efecto que tendrá en la imagen.

Machine Learning: Aprendizaje automático, un campo de la inteligencia artificial que se basa en la capacidad de las máquinas para aprender a partir de datos y mejorar su rendimiento en tareas específicas sin ser programadas explícitamente.

Matriz Homográfica: Una matriz utilizada en la geometría proyectiva para describir las transformaciones proyectivas entre dos sistemas de coordenadas en una imagen.

MobileNetV2: Una arquitectura de red neuronal convolucional (CNN) diseñada para tareas de visión por computadora, que se enfoca en la eficiencia computacional y el rendimiento en dispositivos móviles.

Notación FEN: Notación de Álgebra de Notación Descriptiva (Forsyth-Edwards Notation), utilizada para describir la disposición de las piezas en un tablero de ajedrez.

Operador Laplaciano: Un operador utilizado en el procesamiento de imágenes para resaltar regiones de cambios bruscos en la intensidad de los píxeles, lo que ayuda a detectar bordes y características.

Operador Sobel: Un operador utilizado para detectar bordes en imágenes calculando las derivadas en las direcciones horizontal y vertical, lo que resalta los cambios de intensidad en

esas direcciones.

Funciones de Activación: Las funciones de activación son componentes clave en las redes neuronales artificiales y se utilizan para introducir no linealidad en el modelo. Estas funciones determinan la salida de una neurona o nodo en función de su entrada. Aquí hay algunas funciones de activación comunes:

ReLU (Rectified Linear Activation): Una función de activación que retorna cero si el valor es negativo y el valor mismo si es positivo.

Sigmoide: Una función de activación que transforma valores en el rango $(-\infty, \infty)$ a un rango entre 0 y 1, adecuada para problemas de clasificación binaria.

Softmax: Una función de activación utilizada en la capa de salida de una red neuronal para convertir valores en una distribución de probabilidades.

VGG16: Una arquitectura de red neuronal convolucional (CNN) que consta de 16 capas convolucionales y totalmente conectadas, ampliamente utilizada para tareas de clasificación de imágenes.

Bibliografia

- [1] Nosde21. *La notación FEN en ajedrez, ¿la conoces?* 2020. URL: <https://www.chess.com/es/blog/NosdeChess/la-notacion-fen-en-ajedrez-la-conoces> (visitado 18-07-2023).
- [2] H. Ñaupas Paitán et al. *Metodología de la Investigación Cuantitativa-Cualitativa y Redacción de la Tesis*. Ediciones de la U, 2019.
- [3] C. Rodríguez y R. Dorado Vicente. “¿Por qué implementar Scrum?” En: *Revista ONTARE* 3.1 (2015), págs. 125-144. ISSN: 2745-2220.
- [4] R. González Duque. *Python para todos*. 2011.
- [5] Python Software Foundation. *Python*. 2023. URL: <https://www.python.org/> (visitado 02-04-2023).
- [6] Visual Studio Code. *Página principal*. 2023. URL: <https://code.visualstudio.com> (visitado 02-04-2023).
- [7] Microsoft. *Getting Started with Visual Studio IDE*. 2023. URL: <https://learn.microsoft.com/es-es/visualstudio/get-started/visual-studio-ide?view=vs-2022> (visitado 04-04-2023).
- [8] OpenCV. *OpenCV Documentation*. 2023. URL: <https://docs.opencv.org/> (visitado 04-04-2023).
- [9] NumPy. *Página oficial de NumPy*. 2023. URL: <https://numpy.org/> (visitado 04-04-2023).
- [10] Torchvision. *Página oficial de torchvision*. 2017. URL: <https://pytorch.org/vision/stable/index.html> (visitado 10-05-2023).
- [11] Oracle. *What Is PyTorch?* 2022. URL: <https://developer.oracle.com/es/learn/technical-articles/what-is-pytorch> (visitado 07-07-2023).
- [12] Puentes Digitales. *Todo lo que necesitas saber sobre TensorFlow: La plataforma para inteligencia artificial de Google*. 2018. URL: <https://puentesdigitales.com/2018/02/14/todo-lo-que-necesitas-saber-sobre-tensorflow-la-plataforma-para-inteligencia-artificial-de-google/#:~:text=TensorFlow%20es%20el%20sistema%20de,el%20mundo%20del%20Deep%20Learning.> (visitado 12-05-2023).

- [13] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, 2019.
- [14] E. Stevens y L. Antiga. *Deep Learning with PyTorch: Build, Train, and Tune Neural Networks Using Python Tools*. 2020.
- [15] IBM. *Computer Vision*. 2019. URL: <https://www.ibm.com/es-es/topics/computer-vision> (visitado 04-05-2023).
- [16] A. Dertat. *Applied Deep Learning - Part 4: Convolutional Neural Networks*. Nov. de 2017. URL: <https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2> (visitado 17-06-2023).
- [17] G. Bradski y A. Kaehler. *Learning OpenCV*. O'Reilly Media, 2008.
- [18] S. Russell y P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 2020.
- [19] S. Raschka y V. Mirjalili. *Python Machine Learning*. Marcombo, 2019.
- [20] Ringa Tech. *Funciones de activación a detalle (Redes neuronales)* [Video]. 2022. URL: https://www.youtube.com/watch?v=_0wdproto34&list=PLf12PqY9KBmoiQOMpOHL-67JMSLr1wD8a&index=11&ab_channel=RingaTech.
- [21] M. V. Tercero. *Aprendiendo el patrón módulo*. 2018. URL: <https://medium.com/@mvtercero85/aprendiendo-el-patr%C3%B3n-m%C3%B3dulo-fc68489301bc> (visitado 17-05-2023).
- [22] PyTorch. *Transfer Learning Tutorial*. 2023. URL: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html (visitado 31-07-2023).
- [23] PyTorch. *CIFAR-10 Tutorial*. 2023. URL: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html (visitado 31-07-2023).



UNIVERSIDAD
DE MÁLAGA | **uma.es**

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga