IMPLEMENTACIÓN PRÁCTICA DE LIVECHESS2FEN

PRACTICAL IMPLEMENTATION OF LIVECHESS2FEN

TRABAJO FIN DE GRADO
CURSO 2023-2024

AUTORES
DAVID RODRÍGUEZ LÓPEZ
ALONSO VIVES MERINO

DIRECTORES
ALBERTO ANTONIO DEL BARRIO GARCÍA
DAVID MALLASÉN QUINTANA

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

IMPLEMENTACIÓN PRÁCTICA DE LIVECHESS2FEN

PRACTICAL IMPLEMENTATION OF LIVECHESS2FEN

Trabajo de Fin de Grado en Ingeniería Informática

Autores
DAVID RODRÍGUEZ LÓPEZ
ALONSO VIVES MERINO

DIRECTORES
ALBERTO ANTONIO DEL BARRIO GARCÍA
DAVID MALLASÉN QUINTANA

Convocatoria: Junio 2024

Doble Grado en Ingeniería Informática y Administración y Dirección de Empresas
Facultad de Informática
Universidad Complutense de Madrid

JUNIO de 2024

DEDICATORIA

David Rodríguez López

A mi familia, por su apoyo incondicional constante y a mis amigos y compañeros de clase, que han sido un gran apoyo durante estos años.

Alonso Vives Merino

A mis padres y maestros por todo el esfuerzo que han hecho para educarme, y a mis hermanos y amigos por todo lo que han influido en mi vida.

AGRADECIMIENTOS

A nuestros tutores, Alberto y David, por todo el esfuerzo que han hecho por guiarnos y apoyarnos para la realización de este proyecto a pesar de sus muchas responsabilidades, en especial por lo mucho que han agilizado su desarrollo y el interés constante que han demostrado.

RESUMEN

Implementación práctica de LiveChess2FEN

Hemos desarrollado una aplicación web cuya función principal es la retransmisión de partidas de ajedrez en formato digital, tomando fotografías del tablero físico cada vez que se realiza un movimiento. La aplicación utiliza distintos métodos de comparación de imágenes para detectar las casillas que se han modificado de una posición a otra, así como el programa LiveChess2FEN para obtener la posición del tablero mediante redes neuronales convolucionales en el caso de que no se conozca la posición anterior. El programa que detecta las casillas que se han modificado y muestra el FEN resultante, ha obtenido una precisión muy elevada bajo condiciones óptimas (de un 96%). También hemos ejecutado LiveChess2FEN en una Nvidia Jetson Nano y tomado medidas de rendimiento para comparar dicho rendimiento con el de un ordenador portátil.

Palabras clave

Ajedrez, LiveChess2FEN, FEN, Python, PHP, aplicación web, Nvidia Jetson Nano, rendimiento.

ABSTRACT

Practic LiveChess2FEN implementation

We have developed a web application whose main function is the retransmission of chess games in digital format, taking pictures of the physical board every time a move is made. The application uses different image comparison methods to detect the squares that have been modified from one position to another, as well as the LiveChess2FEN program to obtain the position of the board using convolutional neural networks in case the previous position is not known. The program, which detects the squares that have been modified and displays the resulting FEN, has obtained a very high accuracy under optimal conditions (96%). We have also run LiveChess2FEN on a Nvidia Jetson Nano and taken performance measurements to compare its performance with the performance of a laptop.

Keywords

Chess, LiveChess2FEN, FEN, Python, PHP, web application, Nvidia Jetson Nano, performance.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1 - Introducción

1.1 Motivación

En el mundo del ajedrez, la mayor parte de los torneos que se celebran no retransmiten las partidas que se juegan debido a que a día de hoy, prácticamente no hay otras opciones para retransmitir una partida que no pasen por utilizar en las partidas tableros electrónicos con sus respectivas piezas especiales para la retransmisión, los cuales tienen un precio tan elevado que la mayoría de clubes de ajedrez que realizan torneos no se pueden permitir. Un ejemplo de los precios a los que se encuentran estos artículos lo encontramos en la página web de Digital Game Technology (DGT), donde podemos encontrar un tablero de ajedrez de madera electrónico [DGTa] por 500€ y las piezas clásicas [DGTb] por 230€.

Esta falta de opciones asequibles para retransmitir partidas de ajedrez plantea una barrera para la difusión de este deporte. Sin embargo, con los grandes avances en tecnología e innovación, podemos buscar soluciones para resolver este problema con un menor coste.

En este contexto, hemos optado por desarrollar una aplicación web que sea capaz de retransmitir en vivo una partida de ajedrez utilizando una máquina que actúe como servidor y una cámara que se sitúe encima del tablero, todo esto sin la necesidad de utilizar ningún tablero de ajedrez específico. De esta manera cualquier club podría plantearse la posibilidad de diseñar una infraestructura que permita situar cámaras a cierta altura de las mesas donde se jueguen las partidas de ajedrez.

Esta aplicación de código abierto está disponible en el repositorio "Chess_Transmission_Application" en Github: https://github.com/DavidRodL/Chess_Transmission_Aplication.

1.2 Objetivos

El fin último de este proyecto es desarrollar una aplicación web simple pero funcional, mediante la cual se pueda transmitir una partida de ajedrez que se esté jugando en un tablero físico, de una manera digital.

Una vez finalizada la aplicación web, el segundo objetivo de este trabajo es el de comparar el rendimiento obtenido al ejecutar el programa LiveChess2FEN [1] [Mal+20] en un ordenador portátil, con el rendimiento que muestra este mismo programa ejecutado sobre una Nvidia Jetson Nano.

1.3 Plan de trabajo

Para lograr desarrollar con éxito una aplicación funcional de transmisión de partidas de ajedrez, hemos seguido las siguientes pasos:

- Descargar y configurar una máquina virtual con Linux (concretamente una distribución Ubuntu).
- Lograr ejecutar el programa LiveChess2FEN[1] [Mal+20] en dicha máquina.
- Desarrollar una aplicación web que obtenga imágenes de la cámara de la máquina en la que se está ejecutando y las guarde en el servidor.
- Interrelacionar la aplicación web que captura las imágenes con el programa LiveChess2FEN para que una vez obtenido el FEN de la posición, mostrarlo por la aplicación en un formato de tablero digital, utilizando la misma interfaz de tablero que usa la página web de ajedrez lichess [Lic]. Esta interfaz se llama Chessground [Chea] y es de código abierto.
- Crear un script en Python, que dadas dos carpetas con imágenes de cada casilla de un tablero en dos instantes consecutivos junto con el FEN

[1] - URL: https://github.com/davidmallasen/LiveChess2FEN

de la posición inicial, así como el turno del bando que acaba de mover, muestre tanto la jugada que se ha realizado, como el FEN de la posición final resultante. Este programa utiliza la combinación de distintos métodos de comparación de imágenes para deducir la jugada.

- Añadir la funcionalidad de este programa a la aplicación web.
- Instalar en una Jetson AGX Xavier el programa LiveChess2FEN (finalmente no nos fue posible debido a incompatibilidades con las versiones de las librerías de Python).
- Instalar en una Nvidia Jetson Nano el programa LiveChess2FEN y elegir los comandos para medir el rendimiento obtenido tanto en la Jetson Nano como en el ordenador portátil escogido para el análisis.

Capítulo 2 - Antecedentes

2.1 Notación FEN

La notación FEN (Forsyth-Edwards Notation) es un método para describir la posición de las piezas en un tablero de ajedrez. Consiste en una sola línea de texto que incluye 8 cadenas de caracteres separadas por el símbolo "/". Cada cadena representa con letras las piezas, siendo mayúsculas las blancas y minúsculas las negras, y con números las casillas vacías consecutivas; comenzando desde la octava fila hasta la primera.

Las letras pueden ser "p" para los peones, "r" para las torres, "k" para los caballos, "b" para los alfiles, "k" para el rey y "q" para la reina. Por ejemplo, al tablero de la Figura 2-1 le corresponde el FEN mostrado.

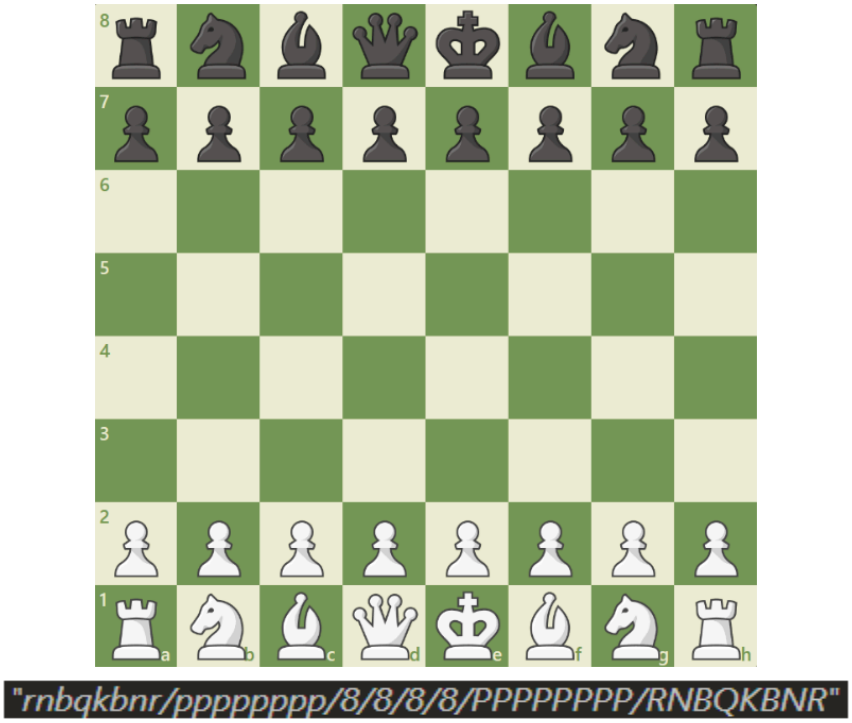


Figura 2-1. Ejemplo de un tablero de ajedrez con su FEN correspondiente. Imagen obtenida de "Chess.com" [Chec]

Esta notación es la que usaremos para representar la configuración de un tablero de manera que un computador pueda procesarla.

Sin embargo, esta notación no se interpreta de un vistazo, y para las personas que la desconozcan no es explicativa. Es por esto que existen programas que permiten mostrar un tablero de ajedrez como imagen a partir de un texto en FEN.

2.2 Interfaz Chessground

Chessground [Chea] es una interfaz de usuario de ajedrez de código abierto y libre, desarrollada para lichess.org [Lic]. Esta engloba una gran cantidad de funcionalidades, entre ellas, la de visualizar un tablero de ajedrez a partir de su notación FEN.

Para integrar esta funcionalidad en nuestra aplicación importamos la librería correspondiente y configuramos una serie de parámetros dejando todos por defecto excepto el FEN. Esta llamada la incluímos en un script dentro del archivo PHP que desarrollamos más adelante.

2.3 Métodos de comparación de imágenes

Existen muchas alternativas a la hora de comparar la similitud de dos imágenes, dependiendo del propósito para el que se requiera unas serán mejores que otras. Basándonos en diversos estudios sobre métricas utilizadas en entornos multimedia [Fer+20] y [Rod+20], para este trabajo, inicialmente se eligieron los siguientes métodos: el índice de Similitud Estructural (SSIM), la comparación de histogramas, el coeficiente de correlación de Pearson y el error cuadrático medio (MSE).

2.3.1 Índice de Similitud Estructural (SSIM)

El SSIM es una métrica utilizada para medir la similitud entre dos imágenes, basado principalmente en cómo percibimos los humanos las diferencias entre imágenes.

Según lo expuesto en el artículo "Fast structural similarity index algorithm" [Che+11], este índice se compone principalmente de tres aspectos: la luminosidad, el contraste y la estructura.

Estas variables se calculan mediante las siguientes fórmulas, siendo x e y las dos imágenes a comparar:

La luminosidad: $l(x,y) = \frac{2 \cdot \mu x \cdot \mu y + C1}{\mu x^2 + \mu y^2 + C1}$

El contraste: $c(x,y) = \frac{2 \cdot \sigma x \cdot \sigma y + C2}{\sigma x^2 + \sigma y^2 + C2}$

La estructura: $s(x,y) = \frac{\sigma xy + C3}{\sigma x \cdot \sigma y + C3}$

Donde:

μx, μy: son las medias locales de las imágenes.

σx,σy: son desviaciones medias.

σxy: es la covarianza de ambas imágenes.

C1,C2, C3: son constantes.

Por tanto el índice SSIM se calcula multiplicando los tres términos:

$$SSIM(x,y) = [l(x,y) \cdot c(x,y) \cdot s(x,y)]$$

2.3.2 Comparación de histogramas

Los histogramas son la representación gráfica de la luz en una imagen, por tanto cuando obtenemos el histograma de una fotografía, este sólo tiene en cuenta los colores de la imagen, dejando de lado atributos de forma y textura. Por tanto dos imágenes de objetos totalmente diferentes pero con el mismo color, tendrán unos histogramas similares.

El eje horizontal de un histograma se corresponde con la variedad de tonos de la imagen, mientras que el eje vertical indica la cantidad de píxeles de cada tono.

Figura 2-2. Histograma de una casilla negra con un caballo negro

Figura 2-3. Histograma de una casilla negra con un peón negro

Si comparamos los histogramas de las Figuras 2-2 y 2-3, observamos que los histogramas son similares, debido a que los tonos de ambas imágenes son muy parecidos.

2.3.3 Coeficiente de correlación de Pearson

Otra técnica para comparar dos imágenes es calcular el coeficiente de correlación de Pearson entre dichas imágenes para evaluar de esta manera su similitud. Este coeficiente toma valores en el rango de -1 a 1. Cuanto más cercanos a 1 sean los valores, existirá una mayor correlación positiva entre las imágenes y por tanto los valores de los píxeles de las dos fotografías tendrán una fuerte relación lineal.

Este coeficiente se calcula mediante la siguiente fórmula:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{(\sum(x_i - \bar{x})^2)}\sqrt{(\sum(y_i - \bar{y})^2)}}$$

Donde:

x_i, y_i: son los valores de los píxeles en las dos imágenes, respectivamente.

$\bar{x}$, $\bar{y}$: son las medias de los valores de los píxeles en las dos imágenes, respectivamente.

Este índice es sencillo de obtener, aunque tiene el principal inconveniente que presentaba el método anterior, no considera ni la estructura ni la forma de la imagen, sino que sólamente se basa en comparar todos los píxeles de forma global y por tanto no se considera su disposición espacial.

2.3.4 Error Cuadrático Medio (MSE)

Normalmente el error cuadrático medio es una métrica que sirve para evaluar la precisión de un modelo predictivo y en el contexto de las imágenes suele ser utilizado para determinar la calidad de una imagen [Leo+20], pero también lo podemos utilizar para calcular la diferencia al cuadrado promedio entre los valores de los píxeles de dos imágenes y de esta manera obtener un valor para medir la similitud entre imágenes. La fórmula para calcular el MSE es la siguiente:

$$MSE = \frac{1}{n} \sum (y_{\mathrm{i}} - x_{\mathrm{i}})^2$$

Capítulo 3 - Estado del arte

A continuación se muestran proyectos con funcionalidades similares a la aplicación que se desarrolla en este trabajo.

3.1 LiveChess2FEN

LiveChess2Fen [Mal+20] está 100% desarrollado en Python y el propio autor lo describe como "un marco de trabajo completamente funcional que digitaliza automáticamente la configuración de un tablero de ajedrez".

Para ello se parte de una foto realizada a un tablero de ajedrez a partir de la cual, el programa identifica los extremos del tablero y los de cada una de las casillas, y a continuación, se procesa cada una de las casillas tratando de identificar la pieza que hay en ellas.

Figura 3-1. Ejemplo de una foto tomada de un tablero de ajedrez y la identificación de cada vértice. Imagen obtenida del propio repositorio de LiveChess2FEN

La identificación de la pieza se hace mediante un modelo de redes neuronales entrenado para identificar 12 tipos de piezas diferentes: peón, torre, caballo, alfil, rey y dama, para cada color. De esta forma, se calcula la probabilidad de que la pieza de la casilla que se está procesando sea cada una de las posibles piezas. En el caso de la torre que vemos en la imagen superior, el modelo identifica que claramente no es una pieza blanca, y que lo más probable es que sea una torre.

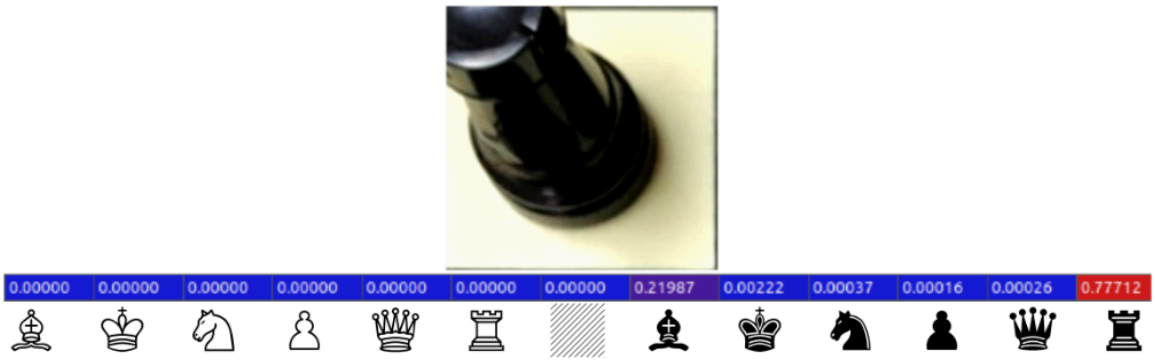


Figura 3-2. Casilla A8 del tablero de la imagen 3-1 y posterior estimación del modelo, el cual identifica que se trata de una torre negra. Esta imagen está sacada del propio repositorio de LiveChess2FEN

Una vez identificadas todas las piezas el output del programa es un FEN que permite identificar la configuración del tablero.

Figura 3-3. Resultado final del procesamiento de la imagen. Imagen obtenida del repositorio de LiveChess2FEN

En el propio repositorio del programa en Github se muestran los resultados que demuestran que el programa es claramente funcional. Sin embargo, su uso no está disponible para una persona que no tenga conocimientos informáticos avanzados, y es por esto que la consideración del desarrollo de una aplicación que haga posible su uso a través de una interfaz es interesante.

3.2 Lichess con tablero real

"Lichess with a real board" [Kar22] es un proyecto desarrollado inicialmente por el usuario de Github "Karayaman".

El programa está 100% desarrollado en Python y en palabras del autor "permite conectar un tablero de ajedrez real a *Lichess* [Lic]. *Lichess* es un servidor de ajedrez libre y gratuito de código abierto y sin publicidad mundialmente usado por millones de usuarios.

Utilizando visión artificial, el programa detecta los movimientos que se realizan en un tablero de ajedrez físico y, si es el turno del jugador en físico, envía el movimiento a los servidores de Lichess utilizando la API de tablero de Lichess. El jugador físico también debe actualizar los movimientos del contrincante en el tablero físico.

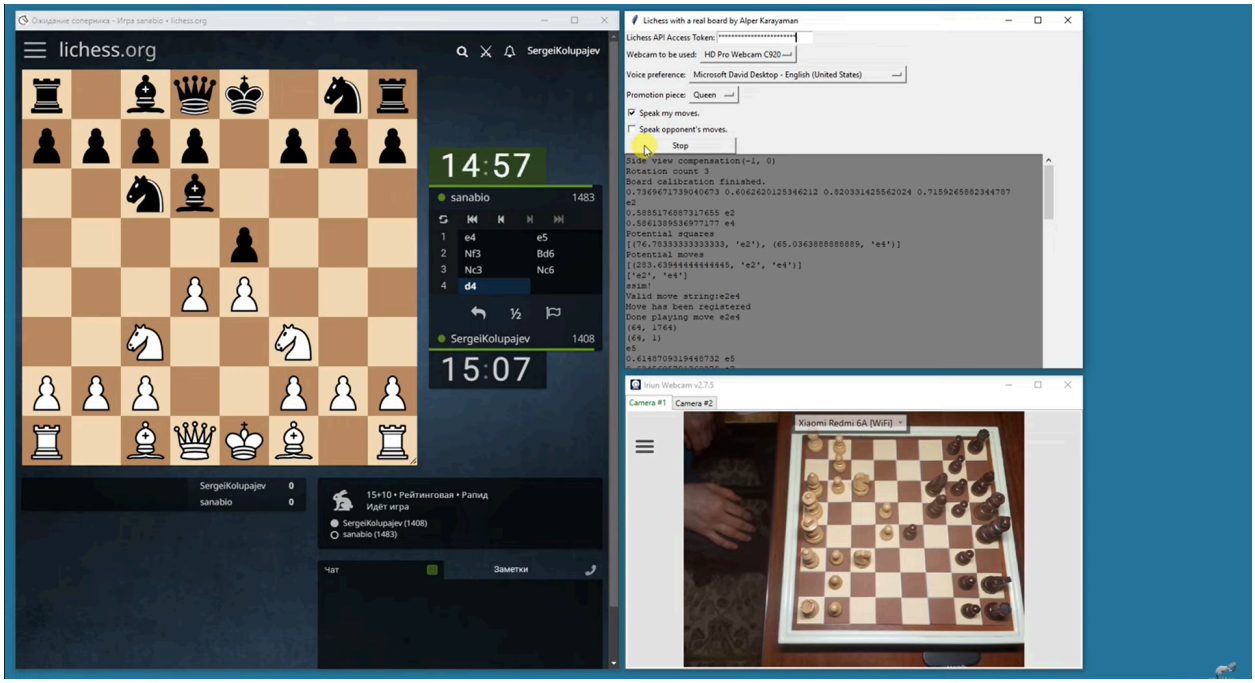


Figura 3-4. A la izquierda vemos la web de Lichess [Lic], en la esquina superior derecha vemos la API que estamos describiendo. En la esquina inferior derecha vemos la partida en físico. Imagen obtenida del repositorio de "Lichess with a real board"

De nuevo nos encontramos con una aplicación que no es fácil de usar para una persona no familiarizada con la informática. Además, cubre una necesidad ligeramente distinta que "LiveChess2FEN".

3.3 Escanner de Tableros de Ajedrez de Chessify

El escáner de Chessify [Cheb] es una herramienta avanzada de Reconocimiento Óptico de Caracteres que permite a los usuarios digitalizar posiciones de tableros de ajedrez desde diferentes plataformas, como libros, pantallas de computadora o tableros de ajedrez 3D de la vida real.

Esta aplicación funciona muy bien pero presenta la desventaja de que esta no es de código abierto, por lo que no se puede contribuir a su mejora sin realizar un acuerdo con los propietarios.

Capítulo 4 - Aplicación web

La aplicación que se ha realizado, obtiene imágenes de la cámara y las compara detectando de esta manera los cambios que se han producido de una posición de ajedrez a otra, mostrando la posición actual de la partida. Adicionalmente, se puede empezar a transmitir una partida que ya esté empezada, es decir, sin conocer la posición anterior del tablero gracias a la aplicación LiveChess2FEN [Mal+20].

4.1 Espacio de trabajo

La máquina utilizada para correr la aplicación ha sido una máquina virtual con Linux, concretamente con la versión Ubuntu 22.04.4 LTS y la versión de Python utilizada es la 3.10.12. La máquina virtual dispone de una CPU de 4 núcleos AMD Ryzen 5 4600h con radeon graphics, con una frecuencia máxima de 3,00 GHz y cuenta con una memoria RAM de 8 GB dual channel DDR4.

Para que esta máquina actúe como servidor web, hay que instalar tanto Apache como PHP. Además, es necesario que se otorguen los permisos necesarios a las carpetas donde se encuentra la aplicación, ya que cuando se accede a archivos desde PHP se realiza desde un usuario distinto al que ha iniciado la sesión.

4.2 Estructura

La aplicación consta de ficheros PHP, javascript y Python. En los siguientes apartados se detalla cómo se relacionan los distintos ficheros entre sí, diferenciando los que pertenecen a la interfaz de la aplicación (frontend), con los que tienen su lógica (backend).

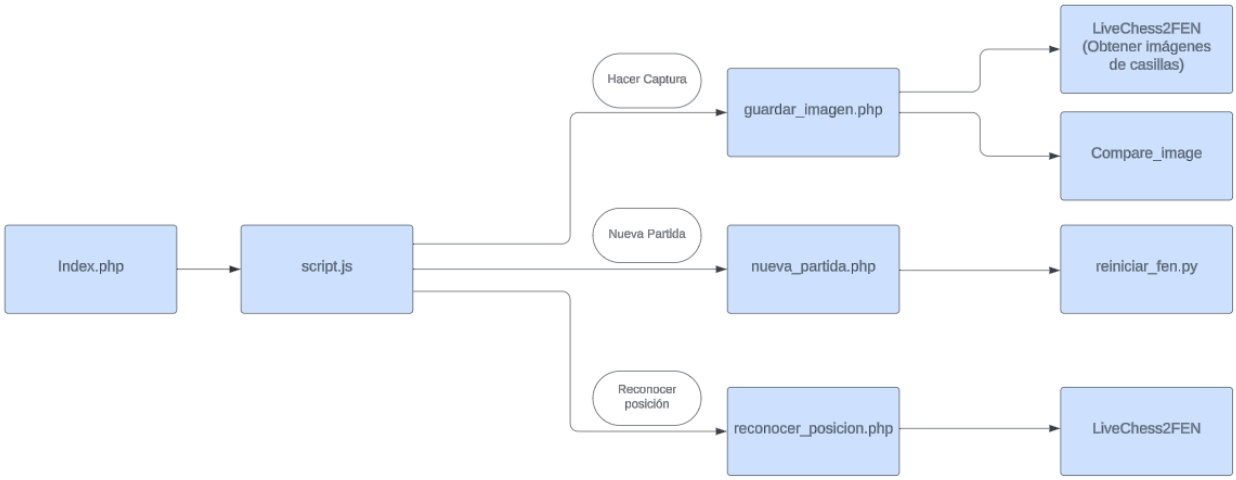


Figura 4-1. Diagrama del funcionamiento de la aplicación

4.2.1 Frontend

El frontend se compone de una serie de archivos PHP y javascript para que el usuario pueda interactuar con la aplicación. A continuación se explica la funcionalidad de dichos ficheros.

En la Figura 4-2 se muestra la interfaz final de la aplicación web.

Figura 4-2. Interfaz de la aplicación

Primeramente, la página principal de la aplicación se encuentra en index.php. En ella se muestra la cámara del usuario y más abajo un tablero de ajedrez, que se actualiza cada segundo con el FEN contenido en el archivo FEN.txt. En la parte central, se muestran tres botones cuya funcionalidad está recogida en el fichero script.js: "Nueva partida", reinicia la posición del tablero; "Hacer captura", realiza una captura al tablero enfocado por la cámara y lo compara con la imagen anterior para detectar el movimiento; y "Reconocer posición", también realiza una captura del tablero, pero

en esta ocasión se utilizan redes neuronales convolucionales para clasificar las piezas y digitalizar el tablero, por lo tanto en este caso no es necesario conocer la posición anterior.

El fichero script.js accede a la cámara y toma las capturas cuando los botones anteriormente mencionados son pulsados, excepto para el botón nueva partida, ya que cuando el usuario quiere iniciar una nueva partida la posición del tablero vuelve al estado inicial. Además de realizar las capturas y transferir las imágenes mediante $_POST al PHP correspondiente que se encargará de guardarlas en el servidor.

4.2.2 Backend

En el backend de la aplicación web encontramos los archivos PHP encargados de realizar distintas funcionalidades dependiendo del botón pulsado, estos son: nueva_partida.php, guardar_imagen.php y reconocer_posicion.php.

- En nueva_partida.php se eliminan las últimas fotos capturadas y se reinicia el FEN, mediante la ejecución de pequeños scripts en Python ejecutados con la función de PHP exec().
- Guardar_imagen.php se encarga de guardar la imagen en el servidor, eliminar la imagen de la posición de dos turnos atrás, ejecutar el programa LiveChess2fen [Mal+20] para obtener imágenes de cada casilla del tablero y el compare_image para detectar las casillas que han cambiado y de esta manera obtener el FEN de la posición y por último, cambiar el turno del jugador al que le toca mover.
- El fichero reconocer_posicion.php es similar al anterior pero toma el FEN de la salida de LiveChess2fen [Mal+20] en vez del compare_image.

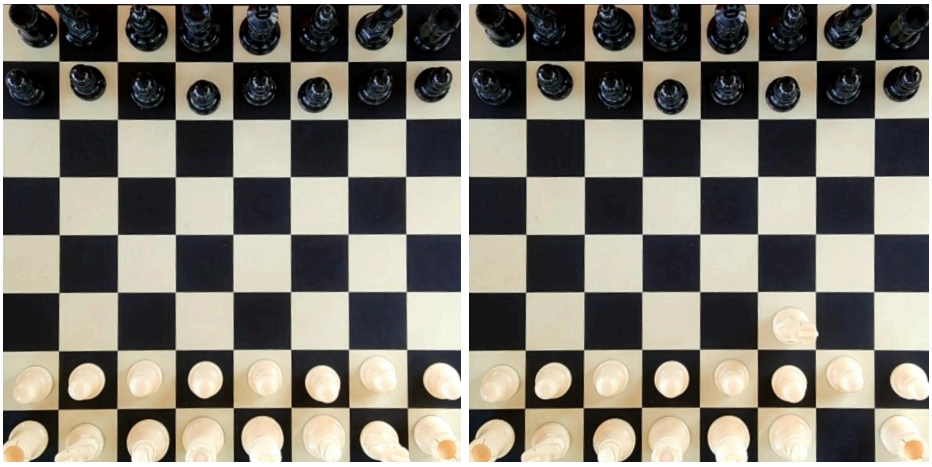
El programa de Python compare_image requiere que existan dos carpetas, las cuales han de contener 64 imágenes, una por cada casilla del tablero. De esta manera se comparan las imágenes correspondientes a las mismas casillas para

detectar las casillas que han cambiado. Al programa hay que pasarle como parámetro el FEN de la posición anterior junto con el bando al que le toca mover ('w' para las blancas y 'b' para las negras). La salida de este serán las casillas cuyos índices de similitud no son muy elevados, así como la jugada realizada y el nuevo FEN de la posición.

A continuación se explica el algoritmo de compare_image:

- Entrada: imágenes a comparar, FEN anterior y turno de juego.
- Salida: Jugada válida y nuevo FEN.

1. Verificar la existencia de las carpetas que contienen las imágenes a comparar.
2. Generar un listado con las rutas de todas las casillas agrupadas por pares (las del tablero de referencia con las de la última posición).
3. Calcular la similitud entre las imágenes mediante el índice de similitud estructural → ssim() y la correlación de histogramas → compareHist().
4. Filtrar casillas no similares identificando los índices de similitud que no superan el umbral establecido.
5. Generar jugadas válidas a partir de las casillas no similares, teniendo en cuenta la última posición conocida (FEN y turno).
6. En caso de encontrar una jugada válida, generar el nuevo FEN y devolverlo. Si no se ha encontrado ningún movimiento válido, devolver el FEN anterior.



rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w

Figura 4-3. Ejemplo de imágenes de dos posiciones, FEN inicial y turno para realizar la ejecución de compare_image

Salida del programa:

Casillas no similares:
f3: 0.83%, 0.64%
g1: 0.49%, 0.57%
Jugada: g1f3
rnbqkbnr/pppppppp/8/8/8/5N2/PPPPPPPP/RNBQKB1R

En esta ejecución se han detectado cambios en dos casillas (f3 y g1) y teniendo en cuenta el FEN y el turno que se le ha proporcionado, muestra la jugada que se ha realizado de una imagen a otra, así como el FEN de la posición resultante.

4.3 Implementación de los métodos de comparación de imágenes

4.3.1 Índice de Similitud Estructural (SSIM)

Para la aplicación desarrollada, la utilización de este índice a la hora de detectar posibles movimientos en una partida de ajedrez nos proporciona las siguientes ventajas: facilidad de implementación en Python, ya que dentro de la librería skimage se encuentra la función structural_similarity() que devuelve directamente el índice al proporcionarle las dos imágenes a comparar. Además, al

centrarse en tres aspectos, este índice puede llegar a captar mejor las diferencias entre casillas.

La principal desventaja de SSIM es que ante cambios de iluminación, puede proporcionar índices de similitud bajos a casillas que realmente no se han modificado. También hay que tener en cuenta que cualquier movimiento de la cámara o diferencia en el momento de detectar el tablero y separar las casillas, puede provocar falsos positivos, es decir, cambios en casillas que no se han visto modificadas.

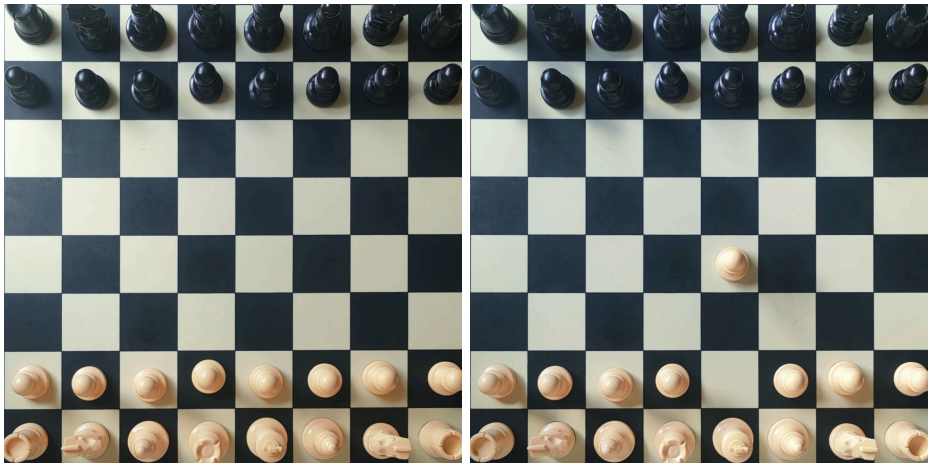


Figura 4-4. Ejemplo de dos posiciones consecutivas para calcular índices de similitud (movimiento de pieza blanca en casilla blanca, con cambio de iluminación)

0.85	0.81	0.80	0.78	0.74	0.69	0.71	0.73
0.82	0.83	0.82	0.82	0.80	0.76	0.76	0.76
0.97	0.87	0.95	0.92	0.95	0.93	0.93	0.92
0.83	0.97	0.91	0.97	0.93	0.96	0.93	0.94
0.96	0.90	0.97	0.95	**0.74**	0.95	0.96	0.92
0.92	0.96	0.95	0.98	0.97	0.97	0.96	0.97
0.83	0.89	0.93	0.69	**0.76**	0.93	0.92	0.87
0.86	0.87	0.95	0.94	0.92	0.89	0.81	0.78

Tabla 4-1. Tabla con los índices de similitud estructural por casilla correspondientes a la Figura 4-4

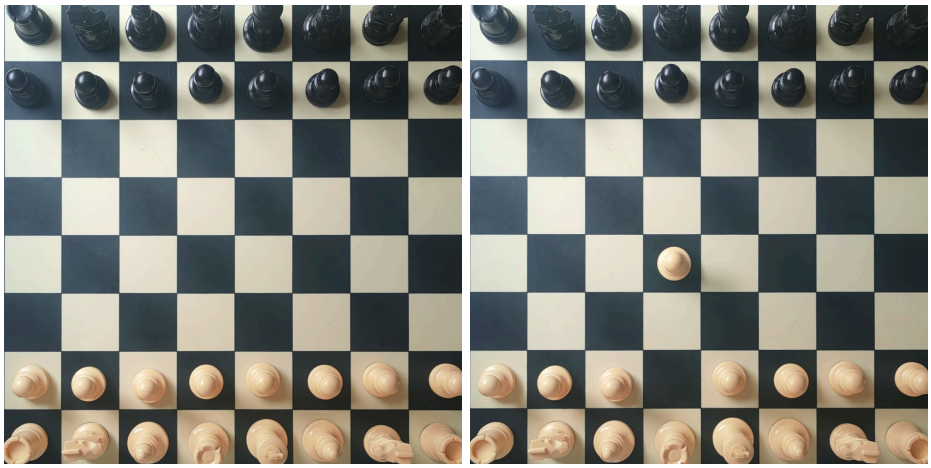


Figura 4-5. Ejemplo de dos posiciones consecutivas para calcular índices de similitud (movimiento de pieza blanca en casilla negra, sin cambio de iluminación)

0.76	0.77	0.83	0.89	0.92	0.94	0.94	0.93
0.80	0.84	0.86	0.88	0.91	0.91	0.93	0.91
0.95	0.93	0.95	0.95	0.97	0.97	0.97	0.96
0.94	0.96	0.93	0.97	0.96	0.97	0.98	0.96
0.95	0.94	0.96	**0.69**	0.94	0.97	0.96	0.96
0.95	0.96	0.95	0.96	0.96	0.96	0.96	0.95
0.80	0.81	0.80	**0.66**	0.79	0.79	0.78	0.78
0.73	0.67	0.72	0.72	0.68	0.69	0.66	0.67

Tabla 4-2. Tabla con los índices de similitud estructural por casilla correspondientes a la Figura 4-5

Como se puede apreciar en la Tabla 4-1, las casillas que se han modificado tienen unos índices de similitud de 0.74 y 0.76, esto es debido a que como las casillas modificadas son de color blanco y al igual que la pieza, el índice no es lo suficientemente bajo para asegurar que el movimiento se ha realizado en esas casillas. Además hay otras casillas que tienen unos índices de similitud ligeramente inferiores a estos valores. Esto es debido a cambios en la iluminación y diferencias a la hora de

ajustar la casilla en la detección del tablero, esto último se puede apreciar en la siguiente imagen (Figura 4-6).

Figura 4-6. Diferencia de iluminación y de encuadre de una misma casilla

Al contrario que en la Tabla 4-1, en la Tabla 4-2 si que se distinguen de forma más clara las casillas que se han visto modificadas cuyos índices SSIM son de 0.69 y 0.66. Esto es debido a que en este otro caso el movimiento se ha producido por una pieza blanca en casillas negras.

Como se ha podido comprobar, el Índice de Similitud Estructural nos puede ayudar a detectar casillas que no se han modificado, pero usándolo de manera aislada, podría dar problemas, ya que sugiere que podría haber más casillas que han cambiado debido a los problemas mencionados anteriormente.

4.3.2 Comparación de histogramas

Para nuestra aplicación, este método puede detectar fácilmente cambios en las casillas del color distinto a la pieza que se sitúa sobre ella, pero a su vez, ligeras variaciones de luminosidad pueden provocar que los histogramas de una casilla que no se ha visto modificada no sean tan similares.

A la hora de comparar dos histogramas, calculamos el coeficiente de correlación entre estos y obtenemos un número que oscila entre -1 y 1. Cuanto más alejado esté de 1, mayor diferencia existirá en los tonos de las imágenes.

Esta técnica tiene una sencilla implementación, ya que para calcular el histograma de una imagen en Python, utilizamos la función calcHist() de la librería cv2 y para comparar dos histogramas utilizamos compareHist().

0.03	0.63	0.49	0.93	0.71	0.68	0.51	0.68
0.58	0.03	0.47	0.87	0.94	0.30	0.67	0.40
-0.02	-0.06	0.10	0.78	0.25	0.56	0.01	0.19
-0.07	-0.03	-0.02	0.58	0.96	-0.03	-0.01	-0.03
-0.04	-0.05	-0.03	0.74	**0.07**	0.73	-0.01	-0.02
-0.03	-0.03	0.45	0.58	0.95	0.46	0.02	-0.02
0.26	0.23	0.57	0.78	**0.70**	0.15	0.11	0.24
0.36	0.70	0.80	0.80	0.64	0.27	0.12	0.49

Tabla 4-3. Tabla con las correlaciones de los histogramas por casilla correspondientes a la Figura 4-4

0.65	0.99	0.98	0.99	0.97	0.99	0.97	0.99
0.98	0.98	1.00	0.97	0.99	0.97	0.99	0.99
0.73	0.99	0.97	0.96	0.96	0.97	0.84	0.98
0.98	0.99	0.99	0.99	0.98	0.96	0.99	0.92
0.97	0.99	0.98	**0.51**	0.91	0.95	0.94	0.97
1.00	0.97	0.94	0.95	0.79	0.90	0.89	0.95
0.94	0.94	0.84	**0.42**	0.78	0.82	0.92	0.97
0.99	0.98	0.93	0.82	0.82	0.97	0.97	0.96

Tabla 4-4. Tabla con las correlaciones de los histogramas por casilla correspondientes a la Figura 4-5

En la Tabla 4-3, observamos que debido al cambio de iluminación, la correlación de la mayoría de las casillas es muy baja debido a que los tonos de la segunda imagen se han visto modificados con respecto a la primera.

En cambio en la Tabla 4-4, hay dos casillas que tienen unos valores de correlación mucho más bajos que el resto. Esto se debe a que en este caso no han

habido cambios de iluminación y además como el movimiento que se ha producido ha sido de un peón blanco en casillas negras, los tonos de dichas casillas se han visto fuertemente modificados.

4.3.3 Coeficiente de correlación de Pearson

Con la función corrcoef() de la librería numpy podemos calcular este coeficiente pasándole las dos imágenes como parámetros.

0.98	0.83	0.96	0.78	0.93	0.77	0.94	0.71
0.90	0.97	0.84	0.96	0.81	0.94	0.77	0.94
0.86	0.73	0.91	0.90	0.93	0.92	0.93	0.92
0.79	0.90	0.91	0.94	0.91	0.96	0.94	0.95
0.88	0.89	0.94	0.95	**0.65**	0.90	0.97	0.97
0.88	0.93	0.96	0.98	0.98	0.96	0.98	0.98
0.93	0.97	0.98	0.79	**0.77**	0.99	0.98	0.98
0.97	0.94	0.99	0.98	0.99	0.98	0.98	0.94

Tabla 4-5. Tabla con los coeficientes de correlación de Pearson por casilla correspondientes a la Figura 4-4

0.95	0.74	0.97	0.90	0.98	0.97	0.99	0.96
0.75	0.95	0.81	0.97	0.90	0.98	0.94	0.98
0.47	0.44	0.75	0.96	0.89	0.97	0.93	0.96
0.84	0.38	0.95	0.81	0.98	0.88	0.99	0.90
0.61	0.46	0.76	**0.13**	0.70	0.91	0.88	0.97
0.67	0.74	0.72	0.85	0.78	0.88	0.88	0.91
0.87	0.91	0.87	**0.21**	0.87	0.91	0.89	0.92
0.91	0.87	0.93	0.87	0.91	0.90	0.91	0.85

Tabla 4-6. Tabla con los coeficientes de correlación de Pearson por casilla correspondientes a la Figura 4-5

En este caso observamos en la Tabla 4-5 que detecta correctamente las casillas que se han visto modificadas cuyos coeficientes de correlación son de 0.65 y 0.77, a

pesar de que el movimiento se produce con una pieza blanca sobre casillas blancas y hay un cambio de iluminación entre las fotografías.

Sin embargo en la otra comparativa de imágenes, los coeficientes de correlación resultantes mostrados en la Tabla 4-6 tienen unos valores significativamente menores, a pesar de que dichas fotografías fueron tomadas bajo las mismas condiciones de iluminación.

4.3.4 Error Cuadrático Medio (MSE)

En Python se puede obtener el MSE utilizando la función mean() de numpy, pasándole como parámetro la diferencia de las dos imágenes, elevada al cuadrado.

El principal problema de utilizar el MSE en esta aplicación es que este es muy sensible a pequeñas variaciones en la imagen y por tanto cualquier pequeño cambio eleva el valor de esta métrica y podría indicar diferencias en casillas que no han sufrido ningún cambio.

112.88	102.06	88.22	38.84	65.30	53.44	81.10	50.03
117.94	104.52	92.99	47.87	27.56	88.17	49.58	80.35
145.90	115.31	120.85	44.63	77.38	36.37	114.55	58.93
109.40	106.52	109.93	30.16	12.58	101.57	79.96	140.51
93.18	122.20	118.33	27.21	**79.24**	24.04	123.54	83.81
128.36	84.05	54.34	13.02	8.89	58.38	43.08	115.18
92.61	46.87	36.72	40.48	**75.87**	82.41	104.09	97.47
59.48	71.01	33.97	62.07	63.08	115.01	87.18	105.68

Tabla 4-7. Tabla con los Errores Cuadráticos Medios por casilla correspondientes a la Figura 4-4

41.95	22.90	32.26	19.75	20.75	15.02	15.09	11.71
23.20	30.62	19.07	25.86	13.93	19.01	12.71	17.60
9.43	7.94	6.77	7.29	5.64	4.61	6.59	5.36

5.80	6.13	8.63	5.48	6.19	6.87	4.07	7.40
8.51	6.10	5.41	**55.32**	47.54	5.76	6.87	5.67
5.95	6.81	8.07	10.17	14.07	8.27	4.97	6.69
30.99	24.93	38.57	**61.42**	49.93	26.23	31.88	27.20
33.77	39.05	34.72	41.98	45.18	39.30	36.60	40.55

Tabla 4-8. Tabla con los Errores Cuadráticos Medios por casilla correspondientes a la Figura 4-5

Como se puede observar en los valores que toman los Errores Cuadráticos Medios representados en la Tabla 4-7, la mayoría de estos valores son muy elevados, ya que al haber un cambio en la iluminación entre ambas imágenes la métrica MSE aumenta considerablemente y por tanto no se pueden detectar correctamente las casillas en las que se ha producido el movimiento.

En la Tabla 4-8 las casillas con mayor MSE coinciden con el movimiento realizado, pero también hay otras casillas con un valor alto, principalmente las que se encuentran cerca de estas, debido a la sombra que produce la pieza que se ha movido en las casillas adyacentes.

4.3.5 Métodos seleccionados en la aplicación

A continuación se muestra una tabla (Tabla 4-9) con los tiempos de ejecución promedio de cada método explicado en el Apartado 2.3, estos tiempos se han obtenido ejecutando 10 veces cuatro programas distintos en python (uno por cada método) que dadas dos carpetas con imágenes de las casillas del tablero, calcula los índices correspondientes a cada método y los muestra en una tabla de 8x8. Se han añadido marcas de tiempo en el propio código Python para medir el tiempo correspondiente al cálculo de los índices en cada ejecución. La plataforma hardware en la que se han realizado estas ejecuciones ha sido la máquina virtual descrita en el Apartado 4.1.

SSIM	Histogramas	Coef. correlación	MSE
0.907	0.131	0.324	0.121

Tabla 4-9. Tabla de tiempos de ejecución promedio de 10 ejecuciones en segundos

Tras analizar los resultados obtenidos mediante cada uno de los cuatro métodos anteriores, el Índice de Similitud Estructural a pesar de ser el que más tiempo requiere es el que tiene en cuenta un mayor número de características, en cambio los otros tres métodos están mayormente influenciados por los tonos de color de la imagen y por tanto son más sensibles a sombras o cambios en la intensidad de la luz. Ninguno de los cuatro métodos por sí solos logra detectar correctamente las casillas en las que se produce el movimiento y por esto optamos por combinar los cuatro métodos de distintas maneras.

Tras varias combinaciones entre estos métodos, finalmente decidimos utilizar el Índice de Similitud Estructural con el método de la comparación de histogramas de la siguiente manera:

- Ambos índices de similitud deben ser inferiores a un valor umbral que hemos definido con un valor igual a 0.85.
- La media de ambos valores ha de ser menor a 0.75.

Con estas dos restricciones logramos disminuir considerablemente el número de casillas que la aplicación detecta como diferentes.

De esta manera, ejecutando el programa con el FEN inicial y el turno de las blancas (rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w) y con las imágenes de la Figura 4-4, obtenemos la siguiente salida:

Casillas no similares:
a8: 0.85%, 0.03%
b8: 0.81%, 0.63%
c8: 0.80%, 0.49%
e8: 0.74%, 0.71%

f8: 0.69%, 0.68%
g8: 0.71%, 0.51%
h8: 0.73%, 0.68%
a7: 0.82%, 0.58%
b7: 0.83%, 0.03%
c7: 0.82%, 0.47%
f7: 0.76%, 0.30%
g7: 0.76%, 0.67%
h7: 0.76%, 0.40%
a5: 0.83%, -0.07%
e4: 0.74%, 0.07%
a2: 0.83%, 0.26%
d2: 0.69%, 0.78%
e2: 0.76%, 0.70%
g1: 0.81%, 0.12%
h1: 0.78%, 0.49%
Jugada: e2e4
rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR

En este caso vemos que consigue detectar el movimiento que se ha producido de una imagen a otra, aunque detecta demasiadas casillas que se han modificado, esto es debido principalmente al cambio en la iluminación, ya que si volvemos a ejecutar el programa con el mismo FEN y turno pero con las imágenes de la Figura 4-5, obtenemos:

Casillas no similares:
a8: 0.76%, 0.65%
d4: 0.69%, 0.51%
d2: 0.66%, 0.42%
Jugada: d2d4
rnbqkbnr/pppppppp/8/8/3P4/8/PPP1PPPP/RNBQKBNR

En este segundo caso detecta cambios en tres casillas y esto es asumible para la funcionalidad de la aplicación, ya que mientras detecte un número muy reducido

de casillas no similares, es muy probable que consiga dar con la jugada que realmente se ha realizado.

El tiempo medio de carga de imágenes y cálculo de los dos índices del programa final de Python es de 0,952 segundos (al que habría que sumarle el tiempo que tarda en realizarse la detección del tablero), siendo este asumible si la partida que se quiere transmitir tiene un control de tiempo considerable, ya que en las partidas blitz o bullet, donde el tiempo por jugador es de pocos minutos, normalmente se realizan muchas jugadas en un corto periodo de tiempo y el programa podría no ser lo suficientemente rápido para seguir el ritmo de la partida.

4.4 Precisión de compare_image

Para medir la precisión, se ha tomado una partida de ajedrez como ejemplo (Anand, Viswanathan - Mansouri, A, Arab Asia-ch 1986, 1-0) y se han reproducido las jugadas en un tablero físico tomando fotos de la posición resultante tras cada jugada. Esto se ha realizado en dos ocasiones, la primera en el interior de una sala con luz artificial, para intentar disminuir lo máximo posible las sombras de las piezas, aunque por contrapartida, en el tablero se aprecian reflejos debido a dicha luz, por lo que las condiciones no eran las ideales. En la segunda prueba se ha colocado el tablero en una mesa junto a una ventana con mucha luminosidad natural, para conseguir unas condiciones óptimas.

En la siguiente tabla (Tabla 4-10) se muestran en las columnas 1 y 3 las jugadas que se realizaron y en las columnas 2 y 4 las jugadas que el programa detecta que se han realizado en la primera ocasión:

Jugada blancas	Jugada blancas (salida del programa)	Jugada negras	Jugada negras (salida del programa)
1. e4	e4	c5	c5
2. Cf3	Cf3	e6	e6
3. d4	d4	cxd4	cxd4
4. Cxd4	Cxd4	Cf6	Cf6
5. Cc3	c3	d6	-
6. f4	-	Cc6	Cc6
7. Ae3	Ae3	e5	Ce5
8. Cf3	Cf3	a6	-
9. Dd2	Dd2	b5	b5
10. 0-0-0	Tc1	Cg4	Cg4
11. Ag1	Ag1	exf4	-
12. Cd5	-	Ae6	Ae6
13. Ab6	Axb5		

Tabla 4-10. Resultados de las consecutivas ejecuciones de compare_image con las imágenes (en condiciones normales)disponibles en el repositorio Chess_Transmission_Aplication (Partida ejemplo1)

Como se puede comprobar en la Tabla 4-10, se han detectado correctamente 16 jugadas de las 25 totales (un 64% de las jugadas y del 67% si no se tiene en cuenta la jugada enroque). Compare_image presenta algunas limitaciones que se detallan en el siguiente apartado, entre estas limitaciones se encuentran: la no detección de enroques y la sensibilidad a la iluminación, por tanto para obtener una mayor precisión, debería de existir una iluminación más uniforme en la sala donde se

encuentra el tablero, así como una mejor colocación de la cámara, ya que esta prueba se ha realizado poniendo un tablero en el suelo y colocando la cámara con un trípode encima del tablero.

Jugada blancas	Jugada blancas (salida del programa)	Jugada negras	Jugada negras (salida del programa)
1. e4	e4	c5	c5
2. Cf3	Cf3	e6	e6
3. d4	d4	cxd4	cxd4
4. Cxd4	Cxd4	Cf6	Cf6
5. Cc3	Cc3	d6	d6
6. f4	f4	Cc6	Cc6
7. Ae3	Ae3	e5	e5
8. Cf3	Cf3	a6	a6
9. Dd2	Dd2	b5	b5
10. 0-0-0	Tc1	Cg4	Ag4
11. Ag1	Ag1	exf4	exf4
12. Cd5	Cd5	Ae6	Ae6
13. Ab6	Ab6		

Tabla 4-11. Resultados de las consecutivas ejecuciones de compare_image con las imágenes (en óptimas condiciones) disponibles en el repositorio Chess_Transmission_Aplication (Partida ejemplo2)

En la Tabla 4-11 por el contrario observamos que tan solo ha detectado erróneamente dos movimientos (uno de los cuales es el enroque, que como se menciona en el siguiente apartado, el programa no es capaz de detectarlo). Por tanto en esta ocasión se detectaron correctamente el 92% de las jugadas (96% si no se tiene en cuenta el enroque).

Por tanto, la precisión de compare_image depende en gran medida de las condiciones en las que se realicen las fotografías.

4.5 Limitaciones y posibles mejoras

La aplicación web desarrollada no siempre consigue detectar algún movimiento, lo que provoca que la transmisión deje de funcionar en cierto punto y por esto hemos añadido la opción de detectar una posición a partir de una sola imagen (utilizando el programa LiveChess2FEN [Mal+20]). A continuación se muestran las principales limitaciones de la aplicación.

4.5.1 Orientación de la cámara

Tal y como se ha diseñado la aplicación, para que esta funcione correctamente es necesario que la cámara que enfoque al tablero esté colocada en la parte superior del tablero a una cierta altura. De esta manera las imágenes que se obtendrán proporcionarán una vista aérea clara y despejada de todo el tablero.

La principal razón que está detrás de esta necesidad es que cuando se separan cada una de las casillas en distintas imágenes, las piezas que están sobre el tablero deben de aparecer tan sólo en la casilla en la que se encuentran. Por tanto, si la cámara no estuviese colocada de forma perpendicular con respecto al tablero, en la imagen que se obtendría, las piezas ocuparían parte de otras casillas y esto aumentaría el número de casillas en las que se han detectado cambios.

Este aspecto limita la portabilidad y flexibilidad de la aplicación, al tener que estar colocando de forma correcta la cámara cada vez que se quiera transmitir una partida.

Para paliar esta limitación, se podría modificar la aplicación de tal forma que indicando la posición y el ángulo de la cámara con respecto al tablero, el programa tenga en cuenta que va a detectar cambios en casillas adyacentes a las que

verdaderamente han sufrido la modificación y tratarlas como casillas similares a pesar de que los índices de similitud calculados sean bajos.

4.5.2 Movimientos especiales

En el ajedrez tanto los movimientos normales como las capturas implican movimiento de piezas en dos casillas específicas, pero hay otros movimientos en los que el movimiento se produce en más de dos casillas. El enroque se produce moviendo el rey y la torre a distintas casillas y por tanto en este caso hay cuatro casillas que se ven modificadas. Tal y como está implementada la aplicación, se detectarán las cuatro casillas modificadas y se buscará la combinación de las dos casillas en las que se produce el movimiento que en el caso del enroque corto (Figura 4-4) estas serían e1 y g1, pero el programa cuando combine las casillas que se han modificado (e1, f1, g1 y h1) probará primero con e1 y f1 y cómo ese movimiento es posible, devolverá el FEN correspondiente a la posición resultante tras la jugada Rf1 en vez de 0-0 (enroque corto).

Figura 4-7. Ejemplo de enroque corto

La captura al paso se produce cuando un peón que está en su posición inicial acaba de avanzar dos casillas y se coloca al lado de un peón enemigo, este último puede capturarlo "al paso", como si sólo hubiera avanzado una casilla en lugar de dos

(Figura 4-4). En este movimiento se modifican tres casillas (en el caso de la Figura 4-8: el peón negro de c4 pasaría a d3 y desaparecería el peón blanco de d4). El problema surge que como este movimiento sólamente es posible realizarlo en el turno siguiente al avance del peón enemigo, el FEN resultante tras este avance que permite la captura al paso tiene un parámetro extra que indica que se puede realizar la captura al paso. En la posición de la Figura 4-5 el FEN sería: r1n1k2r/ppqbppbp/8/P2pn1N1/2pP4/2P4P/1PB1QPP1/RNB2RK1 b kq d3.

Nuestro programa sólamente nos devuelve el primer parámetro del FEN y por tanto la captura al paso no se contempla como un movimiento válido.

Figura 4-8. Ejemplo de captura al paso

Por último, la promoción de un peón tampoco se podría detectar con la aplicación web, ya que está sólo detecta cambios en las casillas y cuando un peón llega a la última fila, el jugador tiene que cambiarlo por cualquier otra pieza (caballo, alfil, torre o dama).

Una forma de resolver los errores en la transmisión producidos por estos movimientos, es utilizar el botón de "Reconocer posición" para obtener el FEN de la posición resultante tras uno de estos movimientos y a partir de ahí seguir con la transmisión movimiento a movimiento con el botón "Hacer captura".

4.5.3 Sensibilidad a diferentes condiciones de iluminación

Variaciones en la luz ambiente o sombras de objetos externos al tablero, pueden causar falsos positivos o negativos en la detección de cambios en las casillas, lo que disminuye la fiabilidad de la aplicación y podría provocar que esta dejara de funcionar correctamente.

4.5.4 Limitaciones en la resolución y calidad de imágenes

La calidad de la cámara mediante la que se estén tomando las imágenes de las posiciones tiene un impacto importante a la hora de que la aplicación funcione correctamente, ya que Imágenes de baja resolución pueden dificultar la detección precisa de cambios en las casillas y también tendría un impacto negativo a la hora de utilizar LiveChess2FEN [Mal+20] para obtener la posición a partir de una imagen.

En base a las pruebas realizadas, cuando utilizamos la aplicación Droidcam para utilizar la cámara del móvil en la aplicación web, la versión gratuita proporciona una resolución de 720p, dicha resolución no es suficiente para que el programa LiveChess2FEN detecte correctamente que piezas hay en cada casilla, aunque el programa compare_image sí detecta las casillas que se han visto modificadas. Por tanto, para que la aplicación web funcione correctamente, debería de utilizarse una cámara con una mayor resolución.

Capítulo 5 - Medidas de rendimiento

Llegados a este punto, es conveniente identificar cuál de las plataformas hardware de las que disponemos es mejor para ejecutar la aplicación considerando que la opción del programa LiveChess2FEN [Mal+20] seleccionada ha sido la de ONNX runtime como motor de inferencia. El modelo elegido para realizar las comparaciones es el MobileNetV2 con $\alpha = 0,5$.

A continuación, detallamos las dos plataformas de las que disponemos, así como los diferentes comandos que hemos utilizado para obtener información de la situación de ambas máquinas durante la ejecución del programa. Los comandos mostrados a continuación se ejecutaron en el orden en el que se encuentran, por tanto la carga de trabajo en ambas plataformas es muy similar.

5.1 Plataforma hardware de ejecución

Para la ejecución de este programa, hemos contado con dos plataformas hardware diferentes.

Por un lado, disponemos de una Nvidia Jetson Nano [Nvia] cuyas especificaciones técnicas son las siguientes:

- En cuanto a su CPU, contiene el Procesador ARM Cortex-A57 MPCore de 4 núcleos, con una frecuencia máxima de 1,43 GHz.
- Tiene una GPU con arquitectura NVIDIA Maxwell con 128 núcleos NVIDIA CUDA.
- Tiene 4 GB de memoria RAM LPDDR4.

Por otro lado, contamos con una CPU utilizada a través de una máquina virtual del sistema operativo Ubuntu con la versión 22.04.4 LTS, con las siguientes especificaciones:

- Una CPU de 8 núcleos (de los cuales la máquina utiliza 4) AMD Ryzen 5 4600h con radeon graphics, con una frecuencia máxima de 3,00 GHz.
- Cuenta con una memoria RAM de 16 GB (8GB para la máquina virtual) dual channel DDR4

Ambos soportes aportan ventajas e inconvenientes a la hora de ejecutar nuestra aplicación. Es por esto que hemos tomado una serie de medidas de rendimiento para comprobar cuál de los dos facilita una mejor ejecución.

5.2 Cálculo de los tiempos de ejecución por fases

Test_lc2fen.py es una utilidad de LiveChess2FEN, cuya función es mostrar los tiempos de ejecución de las distintas funciones que realiza el programa lc2fen.py. Se ha ejecutado activando el motor de inferencia ONNX y utilizando el modelo MobileNetV2 con α = 0,5. También hemos utilizado un profiler de Python para detectar las funciones que están más tiempo en ejecución.

5.2.1 Resultados en la Jetson Nano

Test	Detección de tablero	Separación de casillas	Predicción de probabilidades	Inferencia de piezas	Conversión a notación FEN	TIEMPO TOTAL
test1	10,90014854	0,07169125	6,51337943	0,00316719	0,0001225	17,4885089
test2	3,82389838	0,0783125	1,43831333	0,00492229	0,00011818	5,34556469
test3	3,84269948	0,07802865	1,34206812	0,00353021	0,00012453	5,26645099
test4	3,38219661	0,0739751	1,62301271	0,00329505	0,00011906	5,08259854
test5	3,23611198	0,07214026	1,47813094	0,00315115	0,00011557	4,78964989

Tabla 5-1. Tiempos de ejecución en la Jetson Nano en segundos

Como se puede ver en la Tabla 5-1, en la primera ejecución el tiempo de detección del tablero es bastante mayor que en las siguientes, esto es debido a que al

ejecutarse el programa por primera vez, se tiene en cuenta también el tiempo de carga de bibliotecas y modelos necesarios para su ejecución.

Estudiando los tiempos que se muestran en la tabla, obviando la primera fila de la misma, que presenta un sobrecoste de tiempo, la detección del tablero es la fase en la que mayor tiempo de ejecución se consume, debido principalmente a que requiere de varias iteraciones que no es posible realizarlas de manera paralela para ganar tiempo. La predicción de los vectores de probabilidades de las piezas también requiere de cierto tiempo de ejecución, aunque este no es demasiado elevado debido a que el modelo de redes neuronales elegido (MobileNetV2 con α = 0,5) no es muy grande.

Al utilizar el profiler de Python, la función que más tiempo de ejecución consume es el método run de la biblioteca ONNX Runtime, utilizando aproximadamente un 34% del tiempo total. Esta función se utiliza para realizar la inferencia del modelo ONNX, la fase de detección del tablero aprovecha este modelo para acelerar esta fase gracias a la utilización de la GPU de la Jetson Nano. En segundo lugar se encuentra la carga de imágenes con cerca de un 20% del tiempo total. Un 10% del tiempo se emplea en la detección de líneas en una imágen utilizando la función HoughLinesP(). Cada una de las demás funciones llamadas en el programa no superan el 3% del tiempo total de ejecución.

5.2.2 Resultados en la CPU

Test	Detección de tablero	Separación de casillas	Predicción de probabilidades	Inferencia de piezas	Conversión a notación FEN	TIEMPO TOTAL
test1	1,98386683	0,05373065	0,43461974	0,0004855	0,00001705	2,47271976
test2	1,04227265	0,05782532	0,36525345	0,00205143	0,00002712	1,46742997
test3	1,77726297	0,05252054	0,5796461	0,00058073	0,00001973	2,41003007
test4	1,47610791	0,06404875	0,34296275	0,00066575	0,00001919	1,88380435

test5	1,33170295	0,04804374	0,50619463	0,00105779	0,00003306	1,88703217

Tabla 5-2. Tiempos de ejecución en la CPU en segundos

Al igual que ocurría con los tiempos de la Jetson Nano, las fases que más tiempo consumen en la CPU son la detección del tablero y la predicción de los vectores de probabilidades. Aunque como se puede observar, los tiempos en la CPU son significativamente menores que los de la Jetson Nano.

Observando los tiempos obtenidos con el profiler de Python, las funciones que emplean un mayor porcentaje de tiempo son las correspondientes a la detección del tablero, concretamente las que calculan la intersección entre dos líneas en un punto específico: y_intercept_x() y Compare(), tardan en torno al 22% del tiempo total. Con apenas un 5% de tiempo, encontramos la función run de la biblioteca ONNX Runtime, este porcentaje es menor que en la Jetson Nano debido a que la frecuencia de la CPU es mayor. El resto de funciones presentan unos porcentajes menores.

5.3 Ejecución del Comando Top

El comando top proporciona una vista dinámica en tiempo real del sistema, mostrando una lista con los procesos que están consumiendo más recursos (normalmente CPU y memoria) en el momento en el que este se ejecuta. Además, Top ofrece información sobre el estado general del sistema, como el porcentaje de CPU y memoria en uso, la cantidad de procesos que se encuentran en ejecución y otras estadísticas de carga del propio sistema. Esta herramienta sirve principalmente para monitorizar el rendimiento del sistema así como para identificar procesos que pueden estar consumiendo demasiados recursos.

5.3.1 Código ejecutado

El código ejecutado ha sido el siguiente:

```
top -b -d 1 > top_info.txt &
top_pid=$!
sleep 1
python3 lc2fen.py -o test1.jpg BL
sleep 1
kill $top_pid
grep "52257" top_info.txt > filtered_top_info.txt
```

Este código inicia el comando top en modo batch con actualizaciones cada segundo y redirige su salida a un archivo top_info.txt, ejecutándolo en segundo plano. Luego, captura el PID de top y lo almacena en top_pid. Después de un retraso de un segundo, se ejecuta el programa de Python lc2fen.py con la opción de ONNX activada y la imagen test1.jpg como parámetro de entrada, junto con el parámetro BL (Bottom-Left), debido a que la casilla a1 está en la parte inferior izquierda de la imagen. Posteriormente, tras otro segundo de espera, se finaliza el proceso top usando su PID. Finalmente, se utiliza grep para filtrar y guardar en filtered_top_info.txt cualquier línea del archivo top_info.txt que contenga el número "52257", que se refiere al PID específico de la aplicación lc2fen.py.

5.3.2 Resultados en la Jetson Nano

Carga del Sistema: Los valores de carga media del sistema en los últimos 1, 5 y 15 minutos son (1,50, 0,78, 0,48), indicando que el sistema no presenta una elevada carga durante la ejecución.

Uso de CPU: Pasado cierto tiempo después de comenzar a ejecutarse el programa, se observa que un 34,5% de la CPU está siendo usado por procesos de usuario y un 6,6% por procesos del sistema, mostrando un uso moderado a alto. Un

51,9% de la CPU está inactiva, lo cual indica que la CPU no parece ser un cuello de botella para este programa.

Uso de Memoria: Durante la ejecución, se observa que el porcentaje de uso de la memoria RAM por parte del proceso de Python (LiveChess2FEN) se encuentra en torno al 42% de la memoria RAM total de la Nvidia Jetson Nano, que es de 4GB.

5.3.3 Resultados en CPU

Carga del Sistema: La carga media en los últimos 1, 5 y 15 minutos es baja (1,03, 0,49, 0,39), indicando que el sistema está manejando correctamente la carga sin producirse un estrés significativo.

Uso de CPU: Después de unos segundos de ejecución, un 57,5% de la CPU es utilizado por procesos de usuario y otro 4,6% por procesos del sistema, dejando un 37,7% de la CPU inactiva.

Uso de Memoria: Al contrario que la Jetson Nano, el porcentaje de uso de la memoria RAM por parte del proceso del programa LiveChess2FEN no supera el 7% de la memoria RAM total instalada (de 8GB). Por tanto, la relación porcentaje/total es más baja, indicando una gestión más eficiente.

5.3.4 Comparación

Rendimiento del Sistema: La Jetson Nano muestra una carga del sistema y un uso de CPU similares a la CPU tradicional.

Eficiencia de la Memoria: Ambos sistemas utilizan una cantidad significativa de memoria, pero debido principalmente a la diferencia de memoria RAM máxima entre ambos sistemas, la CPU tradicional consigue manejarla mejor durante la ejecución de LiveChess2FEN.

5.4 Ejecución del Comando Free

El comando free se utiliza para mostrar la cantidad total de memoria libre, utilizada y de intercambio disponible en el sistema, así como los buffers y cachés utilizados por el kernel. Free es útil para obtener una rápida visión general del estado de la memoria del sistema, lo cual es vital para el diagnóstico de problemas de rendimiento o para asegurarse de que el sistema tiene suficiente memoria disponible para las tareas que se están ejecutando.

5.4.1 Código ejecutado

El código ejecutado ha sido el siguiente:

```
watch -n 0.5 'free -h >> memoria.txt'
```

mientras se ejecuta: `python3 lc2fen.py -o test1.jpg BL`

Usamos el comando watch para ejecutar repetidamente el comando free -h cada 0.5 segundos, que muestra el uso de memoria del sistema en formato legible, y redirige continuamente esta salida al archivo memoria.txt. Mientras este proceso de monitoreo está en marcha, se ejecuta simultáneamente el programa de Python, lc2fen.py, con los mismos parámetros que en el apartado anterior. Este conjunto de operaciones permite registrar el uso de la memoria en tiempo real mientras se ejecuta el programa en Python, útil para analizar cómo el proceso afecta el uso de recursos del sistema.

5.4.2 Resultados en la Jetson Nano

Durante la ejecución de la aplicación en la Jetson Nano, se observan cambios notables en el uso de la memoria:

Memoria total: Constante en 3,9 GB.

Memoria utilizada: Incremento de 1,7 GB hasta 3,3 GB, lo que muestra un aumento considerable en la demanda de recursos conforme se ejecuta el programa.

Memoria libre: Decremento significativo de 1,2 GB a 143 MB, reflejando una disponibilidad reducida de memoria libre a medida que avanza la ejecución.

Memoria en buffer/caché: Variación entre 996 MB y 409 MB, indicando el uso dinámico del almacenamiento en caché.

Memoria disponible: Decrecimiento de 2,4 GB a 374 MB, lo que muestra una clara disminución de la memoria a medida que se ejecuta el programa.

Swap: Utilización estable en aproximadamente 1,3 GB de 1,9 GB disponibles, indicando el uso de la memoria virtual para manejar la carga de la memoria principal.

5.4.3 Resultados en la CPU

En la CPU estándar, los cambios en el uso de memoria son los siguientes:

Memoria total: Constante en 7,7 GB.

Memoria utilizada: Ligero aumento de 1,9 GB a 2,2 GB.

Memoria libre: Reducción gradual de 496 MB a 207 MB.

Memoria en buffer/caché: Constante alrededor de 5,3 GB, lo que indica un uso efectivo y estable del almacenamiento en caché.

Memoria disponible: Disminuye ligeramente de 5,4 GB a 5,2 GB.

Swap: Muy poco uso con solo 4,0 MB utilizados de 2,0 GB disponibles, sugiriendo que la memoria principal es suficiente para la carga de trabajo actual.

5.4.4 Comparación

Capacidad de Memoria: La Jetson Nano tiene aproximadamente la mitad de la memoria total disponible en comparación con la CPU (3,9 GB vs. 7,7 GB), lo que se traduce en una capacidad mucho menor para manejar aplicaciones intensivas sin recurrir al swap.

Consumo de Memoria: El consumo de memoria en la Jetson Nano aumenta significativamente bajo carga, con una disminución notable de memoria libre y disponible. En contraste, la CPU mantiene una mayor cantidad de memoria libre y disponible, a pesar de un uso similar de memoria.

Uso del Swap: La Jetson Nano usa una porción considerable de su swap disponible, mientras que la CPU prácticamente no utiliza el swap, reflejando la mayor capacidad de memoria y manejo más eficiente de recursos.

Estabilidad y rendimiento: La máquina virtual ofrece un entorno más estable y capaz para la ejecución prolongada de aplicaciones exigentes, mientras que la Jetson Nano muestra señales de estrés bajo la misma carga, con potencial para problemas de rendimiento si la carga aumenta o se prolonga. En el caso de este programa, la fase de detección del tablero requiere de realizar un número de iteraciones de manera secuencial, haciendo que el rendimiento de la Jetson Nano disminuya en comparación con la CPU. Además, el programa requiere importar muchas librerías y la menor memoria caché disponible en la Jetson Nano, podría afectar también al rendimiento.

En conclusión, la CPU estándar maneja la carga de trabajo de manera más eficiente y efectiva, proporcionando mayor estabilidad y disponibilidad de recursos en

comparación con la Jetson Nano, que muestra limitaciones en capacidad de memoria y gestión bajo condiciones de alta demanda.

5.5 Ejecución del Comando Vmstat

El comando vmstat (virtual memory statistics) muestra diversa información acerca del sistema operativo, la memoria, los procesos, la E/S de disco, la paginación y la actividad de la CPU. Por tanto, nos proporciona una amplia visión de cómo el sistema está manejando la memoria tanto física como virtual, así como información sobre la entrada/salida y la actividad de la CPU. Es útil para medir el rendimiento del sistema e identificar posibles cuellos de botella.

5.5.1 Código ejecutado

El código ejecutado ha sido el siguiente:

```
vmstat 1 > vmstat_info.txt &
sleep 1
vmstat_pid=$!
python3 lc2fen.py -o test1.jpg BL
sleep 1
kill $vmstat_pid
```

Este código inicia vmstat con una periodicidad de 1 segundo para monitorizar el estado del sistema, y la salida se redirige a un archivo llamado vmstat_info.txt. El proceso vmstat se ejecuta en segundo plano, y el script guarda su ID de proceso en la variable vmstat_pid. Luego, se introduce una pausa de 1 segundo antes de ejecutar el programa lc2fen.py. Tras otra pausa de 1 segundo, se envía una señal para terminar el proceso vmstat usando su ID de proceso, deteniendo así el monitoreo del sistema.

5.5.2 Resultados en la Jetson Nano

En la Jetson Nano, observamos un uso intensivo tanto de CPU como de memoria, con variaciones significativas en el uso de recursos a lo largo de la ejecución:

Uso de CPU: La utilización de CPU muestra picos considerables (hasta 43% de uso de sistema y del 50% en el uso de usuario).

Memoria: La memoria libre disminuye progresivamente de 647,2 MB a valores mínimos de 63,4 MB, mostrando un alto consumo a medida que se ejecuta el programa.

Swap: El uso de swap incrementa notablemente, de 852 KB a 1,1 GB, reflejando un aumento significativo en la demanda de memoria swap.

I/O de disco: Hay un incremento notable en la actividad de disco, con picos de hasta 85,5 MB en lecturas, indicando operaciones de I/O intensivas durante el procesamiento de imágenes o datos.

5.5.3 Resultados en la CPU

En la CPU, los resultados sugieren un uso más moderado y estable de los recursos:

Uso de CPU: La CPU tiene una utilización moderada, con valores de uso de sistema que no superan el 18% y un porcentaje de uso de usuario máximo de 40%.

Memoria: La memoria libre se mantiene considerablemente estable alrededor de unos 420,7 MB, disminuyendo hasta los 148,3 MB durante la ejecución, para después volver a un valor similar al que se encontraba anteriormente.

Swap: El uso de swap es marginal (4,5 MB), indicando que la memoria física es suficiente para las demandas del programa.

I/O de disco: Hay muy poca actividad de disco, con valores insignificantes en lecturas y escrituras.

5.5.4 Comparación

Uso de CPU: El ordenador muestra una utilización de la CPU moderada, a diferencia de la Jetson Nano, que experimenta un uso por parte del sistema más intensivo, esto es así, principalmente debido a que la frecuencia máxima de la CPU es de 3 GHz, mientras que la de la Jetson Nano es de 1,43 GHz.

Gestión de Memoria: La Jetson Nano recurre intensamente al swap, sugiriendo de esta forma una insuficiencia de memoria para la carga de trabajo impuesta, mientras que el ordenador prácticamente no necesita hacer uso de la memoria swap.

Actividad de disco (I/O): La diferencia en la actividad de I/O entre las dos plataformas es significativa, siendo mucho más alta en la Jetson Nano. Esto podría deberse a una mayor necesidad de gestionar memoria a través de swap debido a la menor cantidad de memoria física disponible.

Estabilidad del sistema: La CPU ofrece un entorno más estable para la operación continua, mientras que la Jetson Nano muestra una mayor variabilidad que podría afectar el rendimiento.

En conclusión, para el programa LiveChess2FEN, la CPU estándar proporciona un rendimiento más estable y eficiente, con mejor manejo de los recursos del sistema, mientras que la Jetson Nano podría enfrentar limitaciones debido a su capacidad de procesamiento y gestión de memoria más restringida.

5.6 Ejecución del Comando Tegrastats

El comando Tegrastats [Nvib] se puede utilizar en sistemas basados en Nvidia Tegra, como es el caso de la Jetson Nano, y muestra diversas estadísticas del sistema en tiempo real. En este caso vamos a analizar la carga de la GPU mientras se ejecuta el programa lc2fen.py.

5.6.1 Código ejecutado

El código ejecutado ha sido el siguiente:

```
tegrastats --interval 5000 --logfile tegra_stats.txt
mientras se ejecuta: python3 lc2fen.py -o test1.jpg BL
```

Con esto, cada 5 segundos guardamos en un txt la información recopilada por Tegrastats mientras ejecutamos lc2fen.py.

5.6.2 Resultados en la Jetson Nano

Tras la ejecución del comando anterior, obtenemos un txt en el que se muestra una serie de líneas con información del estado del sistema en esos momentos. En este caso nos fijamos en el parámetro GR3D_FREQ que muestra el porcentaje de GPU utilizada y se observa que hasta que pasan unos cuantos segundos, la carga de la GPU es de aproximadamente el 0%, esto es debido a que se están cargando las librerías necesarias para la posterior ejecución del programa. Tras estos segundos, esta variable toma los valores: 78%, 99%, 97% y 57% para posteriormente volver al 0%, lo que indica que dicho programa utiliza toda la capacidad de procesamiento de la GPU de forma intensiva durante un breve periodo de tiempo. Este uso intensivo de la GPU es debido a que esta se utiliza para acelerar las fases de detección del tablero,

acelerando las redes neuronales convolucionales. Sin embargo la CPU se utiliza para realizar los cálculos secuenciales necesarios para detectar el tablero, ya que existen partes que no se pueden paralelizar y por tanto se ejecutan en la CPU.

Capítulo 6 - Conclusiones y trabajo futuro

6.1 Conclusiones

En este trabajo se ha presentado una aplicación web que proporciona la funcionalidad de transmitir una partida de ajedrez en tiempo real, mediante la detección de cambios en las casillas del tablero gracias a la colocación de una cámara situada encima del propio tablero. Se ha logrado integrar con éxito la funcionalidad ofrecida por LiveChess2FEN con el programa compare_image en la aplicación web final, permitiendo también la transmisión de una partida ya comenzada.

Compare_image ha demostrado tener una precisión del 64% en condiciones normales de iluminación artificial con alguna sombra. En condiciones óptimas con luz natural y sin sombras, ha tenido un 92% de precisión. Ambos resultados se han tomado en una partida donde se realizaba un enroque, un movimiento que el programa no es capaz de detectar. Despreciando este movimiento, la precisión habría sido del 67% en el primer caso y del 96% en el segundo.

Por tanto, los métodos que hemos elegido para detectar las diferencias entre las imágenes de las casillas han demostrado funcionar correctamente en condiciones óptimas y sin ejecutar los movimientos especiales: captura al paso, enroques y promociones de peón.

Respecto a los resultados obtenidos en términos de rendimiento, hemos identificado que la Jetson Nano hace un mayor uso tanto de la memoria, como de la CPU, debido a que cuenta con una memoria menor (4GB frente a los 8GB de la máquina virtual) y una frecuencia de CPU también menor (1,43GHz frente a 3GHz). Debido a la menor cantidad de memoria disponible por parte de la Jetson Nano,

hace un uso significativo del swap, mientras que la máquina virtual no la necesita utilizar.

Finalmente, es importante subrayar que la Jetson Nano también ha hecho uso de la GPU para acelerar distintas fases en la detección del tablero, y que los tiempos de ejecución han sido mayores en este caso con respecto a la máquina virtual del portátil.

También hay que tener en cuenta la diferencia de consumo entre la Jetson Nano y el ordenador portátil utilizado para el análisis, ya que la primera tiene un consumo en vatios significativamente menor que el ordenador.

6.2 Trabajo futuro

Una de las principales limitaciones de aplicación es la no detección de tres movimientos especiales en el ajedrez, es decir, la promoción, el enroque y la captura al paso. En un futuro se podría adaptar el código para garantizar así que la aplicación pueda detectar todos los movimientos de una partida en condiciones óptimas.

En cuanto a la captura al paso, sería interesante incluir dicha opción en el FEN que manejamos, tal y como se hace en la realidad, aunque habría que adaptar el programa a dicha funcionalidad. En cuanto al enroque, sería adaptar el programa para que cuando identifique que han cambiado cuatro casillas, valore la posibilidad de que se haya ejecutado un enroque. Sería interesante de nuevo ampliar el FEN para que este muestre la posibilidad de ejecutar dicho movimiento. Para la promoción, la solución más sencilla para saber en qué pieza se transforma el peón, sería volver a realizar un análisis con el LiveChess2FEN, o que el usuario directamente tenga habilitado un botón con el que pueda confirmarlo.

Por otro lado, sería interesante optimizar la aplicación para que sea capaz de aumentar el contraste de las imágenes con peor iluminación y así reduzca la brecha

en la tasa de acierto de las partidas jugadas en condiciones favorables y desfavorables, y, también, añadir la funcionalidad de detectar un nuevo movimiento de piezas sin necesidad de apretar posteriormente ningún botón, para que la aplicación se ejecute de manera completamente autónoma.

Así mismo, se podrían haber obtenido resultados interesantes ejecutando la aplicación en la Jetson AGX Xavier dado que esta es más moderna y con mejores especificaciones que la Nano.

Introduction

Motivation

In the world of chess, most of the tournaments that are held do not broadcast the games that are played because nowadays, there are practically no other options to broadcast a game that do not involve the use of electronic boards with their respective special pieces for broadcasting, which have such a high price that most chess clubs that hold tournaments can't afford. An example of the prices of these items can be found on the Digital Game Technology (DGT) website, where we can find an electronic wooden chess board [DGTa] for €500 and the classic pieces [DGTb] for €230.

This lack of affordable options to broadcast chess games presents a barrier to the diffusion of this sport. However, with the great advances in technology and innovation, we can look for solutions to solve this problem at a lower cost.

In this context, we have chosen to develop a web application that is able to broadcast live a chess game using a machine that acts as a server and a camera that is placed over the board, all this without the need to use any specific chess board. In this way any club could consider the possibility of designing an infrastructure that allows cameras to be placed at a certain height of the tables where chess games are played.

This open-source application is available in the "Chess_Transmission_Application" repository on Github: https://github.com/DavidRodL/Chess_Transmission_Aplication.

Goals

The ultimate goal of this project is to develop a simple but functional web application, through which a chess game being played on a physical chessboard can be transmitted digitally.

Once the web application is finished, the second objective of this work is to compare the performance obtained when running LiveChess2FEN [Mal+20] on a laptop computer, with the performance of the same program running on a Nvidia Jetson Nano.

Work plan

In order to successfully develop a functional chess game broadcasting application, we have followed the following steps:

- Download and configure a virtual machine with Linux (specifically an Ubuntu distribution).
- Run the LiveChess2FEN program [Mal+20] on that machine.
- Develop a web application that obtains images from the camera of the machine on which it is running and saves them on the server.
- Interconnect the web application that captures the images with the LiveChess2FEN program so that once the FEN of the position is obtained, display it by the application in a digital board format, using the same board interface used by the chess website lichess [Lic]. This interface is called Chessground [Chea] and is open source.
- Create a script in Python, that given two folders with images of each square of a board at two consecutive instants with the FEN of the initial position, as well as the turn of the side that just moved, displays both the move that has been made, as well as the FEN of the resulting final position. This program uses a combination of different image comparison methods to deduce the move.
- Add the functionality of this program to the web application.
- Install LiveChess2FEN on a Jetson AGX Xavier (finally it was not possible due to incompatibilities with the versions of the Python libraries).

- Install on a Nvidia Jetson Nano the LiveChess2FEN program and choose the commands to measure the performance obtained both on the Jetson Nano and on the laptop chosen for the analysis.

Conclusions and future work

Conclusions

In this paper we have presented a web application that provides the functionality to broadcast a chess game in real time, by detecting changes in the squares of the board thanks to the placement of a camera above the board itself. The functionality offered by LiveChess2FEN has been successfully integrated with the compare_image program in the final web application, allowing also the transmission of a game that has already started.

Compare_image has been shown to be 64% accurate under normal artificial lighting conditions with some shadowing. In optimal conditions with natural light and no shadows, it had 92% accuracy. Both results were taken in a game where a castling was performed, a move that the program is not able to detect. Disregarding this move, the accuracy would have been 67% in the first case and 96% in the second.

Therefore, the methods we have chosen to detect the differences between the images of the squares have been shown to work correctly under optimal conditions and without executing the special moves: capture on the move, castling and pawn promotions.

Regarding the results obtained in terms of performance, we have identified that the Jetson Nano makes a higher use of both memory and CPU, due to the fact that it has a smaller memory (4GB versus 8GB of the virtual machine) and a lower CPU frequency (1.43GHz versus 3GHz). Due to the smaller amount of memory available on the Jetson Nano, it makes significant use of swap, while the virtual machine doesn't need to use it.

Finally, it's important to underline that the Jetson Nano has also made use of the GPU to accelerate different phases in the detection of the board, and that the execution times have been higher in this case with respect to the virtual machine on the laptop.

Also note the difference in power consumption between the Jetson Nano and the laptop used for the analysis, as the Jetson Nano has a significantly lower watt consumption than the laptop.

Future work

One of the main limitations of the application is the non-detection of three special moves in chess, that is, promotion, castling and en passant capture. In the future, the code could be adapted to ensure that the application can detect all the moves of a game under optimal conditions.

As for the en passant capture, it would be interesting to include this option in the FEN that we handle, as it is done in reality, although the program would have to be adapted to this functionality. As for castling, it would be necessary to adapt the program so that when it identifies that four squares have changed, it evaluates the possibility that a castling has been executed. It would be interesting again to extend the FEN so that it shows the possibility of executing such a move. For promotion, the simplest solution to know what piece the pawn is transformed into, would be to redo an analysis with LiveChess2FEN, or that the user directly has a button enabled with which he can confirm it.

On the other hand, it would be interesting to optimize the application to be able to increase the contrast of images with worse lighting and therefore reduce the gap in the hit rate of games played in favorable and unfavorable conditions, and also to add the functionality to detect a new piece movement without the need to press any button afterwards, so that the application runs completely autonomously.

Also, interesting results could have been obtained running the application on the Jetson AGX Xavier since it is modern and with better specifications than the Nano.

CONTRIBUCIONES PERSONALES

David Rodríguez López

- Visualizar cómo queríamos que fuera la aplicación web final.
- Preparación del entorno de trabajo mediante la búsqueda e instalación de una máquina virtual.
- Instalación de todo lo necesario para ejecutar en la máquina el programa LiveChess2FEN.
- Preparación del entorno para utilizar la máquina como servidor PHP (instalación de Apache y PHP).
- Desarrollo junto con mi compañero de la aplicación web con sus correspondientes ficheros PHP y javascript.
- Solución de los problemas de incompatibilidades entre versiones.
- Solución del problema de los permisos de los archivos usados por la aplicación.
- Instalación de Droidcam en la máquina virtual para utilizar el móvil como cámara a la hora de utilizar la aplicación web.
- Desarrollo junto con mi compañero del programa compare_image en Python.
- Instalación junto con mi compañero de todo lo necesario para ejecutar LiveChess2FEN en la Jetson AGX Xavier (finalmente no nos fue posible debido a incompatibilidades con las versiones de las librerías de Python).
- Ejecución de los comandos para obtener las medidas de rendimiento en la Nvidia Jetson Nano.
- Realización del guión de la memoria.
- Realización del resumen y su traducción al inglés.
- Realización del apartado 1.1 donde explicamos la motivación del trabajo y su traducción al inglés.

- Realización del apartado 1.2 donde se detallan los principales objetivos del mismo y su traducción al inglés.
- Realización del apartado 1.3 correspondiente al plan de trabajo que hemos seguido y su traducción al inglés.
- Realización del apartado 2.3 donde se explican 4 métodos de comparación de imágenes:
    - 2.3.1 Explicación del Índice de Similitud Estructural.
    - 2.3.2 Explicación del método de comparación de histogramas.
    - 2.3.3 Explicación del Coeficiente de Correlación de Pearson.
    - 2.3.4 Explicación del Error Cuadrático Medio.
- Realización del capítulo 4 en el que se incluyen los siguientes apartados:
    - 4.1 Breve descripción del espacio de trabajo utilizado.
    - 4.2 Estructura de la aplicación, tanto el frontend (4.2.1) como el backend (4.2.2).
    - 4.3 Comparaciones de los 4 métodos de comparación, así como elección final de los mismos para el programa de Python.
    - 4.4 Elección de la partida a utilizar para medir la precisión de compare_image, reproducción de la partida en un tablero físico tomando fotos en dos condiciones diferentes y análisis de la precisión que obtiene compare_image.
    - 4.5 Listado de las principales limitaciones, así como posibles mejoras donde se incluyen:
        - 4.5.1 Limitaciones en función de la posición de la cámara.
        - 4.5.2 Movimientos especiales que no consigue detectar compare_image.
        - 4.5.3 Limitaciones relacionadas con la iluminación del lugar donde se encuentre el tablero físico.

        - 4.5.4 Limitaciones en función de la resolución de las imágenes que se tomen del tablero.
    - Realización de las figuras y tablas que se muestran en el capítulo 4.
- Realización del apartado 5.2 donde se muestran los tiempos de cada fase de LiveChess2FEN tanto en la Jetson Nano (5.2.1) como en la CPU (5.2.2).
- Realización de las tablas del apartado 5.2.
- Realización del apartado 5.6 correspondiente al comando tegrastat para obtener el porcentaje de uso de la GPU de la Jetson Nano durante la ejecución de LiveChess2FEN.
- Realización del manual de inicio de la aplicación.
- Revisión completa de la memoria garantizando el cumplimiento del formato.

Alonso Vives Merino

- Contacto con los tutores y desarrollo de la idea.
- Preparación del entorno de trabajo mediante la búsqueda e instalación de una máquina virtual de Ubuntu.
- Análisis y comprensión del proyecto de LiveChess2FEN.
- Instalación de todo lo necesario para ejecutar en la máquina el programa LiveChess2FEN.
- Preparación del entorno para utilizar la máquina como servidor PHP (instalación de Apache y PHP).
- Archivo constante del trabajo realizado para la posterior redacción de la memoria.
- Desarrollo junto con mi compañero de la aplicación web con sus correspondientes ficheros PHP y javascript.
- Solución de los problemas de incompatibilidades entre versiones.
- Solución del problema de los permisos de los archivos usados por la aplicación.
- Desarrollo junto con mi compañero del programa compare_image en Python.
- Ejecución de la aplicación completa y últimas correcciones.
- Instalación junto con mi compañero de todo lo necesario para ejecutar LiveChess2FEN en la Jetson AGX Xavier (finalmente no nos fue posible debido a incompatibilidades con las versiones de las librerías de Python).
- Planteamiento final del trabajo como una comparativa entre la ejecución de la aplicación en CPU y en la Jetson Nano.
- Estudio de la normativa de los TFGs de Informática en el curso 2024.
- Realización del guión de la memoria.
- Realización del apartado 2.1 y 2.2 donde explicamos la notación FEN y el módulo Chessground.

- Investigación para el estado del arte: búsqueda e identificación de los proyectos ya desarrollados similares al nuestro
- Redacción del apartado 3 en el que se desarrolla este estado del arte.
- Análisis del soporte hardware utilizado para la ejecución de la aplicación.
- Redacción del apartado 5.1 de la memoria donde se explica el soporte hardware usado.
- Redacción del apartado 5.3 en el que se desarrolla la ejecución del comando Top.
    - 5.2.1 Código ejecutado.
    - 5.2.2 Resultados en la Jetson Nano.
    - 5.2.3 Resultados en la CPU.
    - 5.2.4 Comparación de los resultados obtenidos.
- Redacción del apartado 5.4 en el que se desarrolla la ejecución del comando Free.
    - 5.4.1 Código ejecutado.
    - 5.4.2 Resultados en la Jetson Nano.
    - 5.4.3 Resultados en la CPU.
    - 5.4.4 Comparación de los resultados obtenidos.
- Redacción del apartado 5.5 en el que se desarrolla la ejecución del comando Vmstat.
    - 5.5.1 Código ejecutado.
    - 5.5.2 Resultados en la Jetson Nano.
    - 5.5.3 Resultados en la CPU.
    - 5.5.4 Comparación de los resultados obtenidos.
- Redacción del apartado en el que se desarrolla la ejecución del comando Time (eliminado).
- Revisión completa de la memoria garantizando el cumplimiento del formato.

- Redacción de conclusión y su traducción al inglés.
- Redacción del trabajo futuro y su traducción al inglés.
- Organización de la bibliografía.

BIBLIOGRAFÍA

[Mal+20] D. Mallasén Quintana, A. A. Antonio del Barrio García, M. Prieto Matías: "LiveChess2FEN: a Framework for Classifying Chess Pieces based on CNNs". arXiv:2012.06858 [cs], Dec. 2020. [Online]. Disponible: http://arxiv.org/abs/2012.06858

[Che+11] M. J. Chen y A. C. Bovik. "Fast structural similarity index algorithm". En Journal of Real-Time Image Processing Vol 6. (2011), págs. 281–287. DOI: 10.1007/s11554-010-0170-9

[DGTa] DGT. Tablero electrónico de nogal con usb. URL: https://dgtshop.com/products/chess-boards/usb-e-board-walnut-in-gift-box (visitado 05-2024)

[DGTb] DGT. Piezas electrónicas clásicas. URL: https://dgtshop.com/products/electronic-chess-pieces-2/classic (visitado 05-2024)

[Leo+20] A. León, J. Bermeo, J. Paredes y H. Torres. "Una revisión de las métricas aplicadas en el procesamiento de imágenes". En RECIMUNDO. (2020), págs. 267-273. URL: https://recimundo.com/index.php/es/article/view/874 (visitado 05-2024)

[Lic] Lichess. URL: https://lichess.org/ (visitado 06-2023)

[Chec] Chess.com. FEN. URL: https://www.chess.com/terms/fen-chess (visitado 05/2024)

[Chea] Chessground. URL: https://github.com/lichess-org/chessground (visitado 06-2023)

[Kar22] A. Karayaman. "Lichess with a real board". 2022. URL: https://github.com/karayaman/lichess-with-a-real-board/tree/main

[Cheb] Chessify. "Our Chess Scanner Feature is Now Available on the Website" URL: https://chessify.me/news/chess-scanner-on-chessify-website (visitado 05-2024)

[Nvia] Nvidia. Jetson Nano. URL: https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-nano/product-development (visitado 05-2024)

[Nvib] Nvidia. NVIDIA DRIVE OS SDK Development Guide. URL: https://docs.nvidia.com/drive/drive_os_5.1.6.1L/nvvib_docs/index.html#page/DRIVE_OS_Linux_SDK_Development_Guide/Utilities/util_tegrastats.html (visitado 05-2024).

[Fer+20] D.G. Fernández, G. Botella y A.A. Del Barrio et al. "HEVC optimization based on human perception for real-time environments". Multimed Tools Appl 79, 16001–16033 (2020). https://doi.org/10.1007/s11042-018-7033-y

[Rod+20] David Rodríguez Galiano, A.A. Del Barrio, Guillermo Botella, David Cuesta: "Efficient embedding and retrieval of information for high-resolution videos coded with HEVC". Comput. Electr. Eng. 81: 106541 (2020).

APÉNDICES

Apéndice A - Manual de inicio de la aplicación web

Para utilizar la aplicación web, es necesario realizar los siguientes pasos en una máquina con un sistema operativo Linux:

1. Primeramente hay que descargar el programa LiveChess2FEN, disponible en el siguiente enlace: https://github.com/davidmallasen/LiveChess2FEN/tree/master y realizar la instalación de los requerimientos especificados en dicho repositorio. Preferiblemente, a la hora de instalar las librerías de Python necesarias, es recomendable hacerlo en un entorno virtual.
2. Debido a que la aplicación web está desarrollada en PHP, hay que instalar Apache y PHP, para esto se puede ejecutar el siguiente comando en la terminal: sudo apt update && sudo apt install apache2 php libapache2-mod-php.
3. Tras esta instalación, reiniciamos apache con: sudo systemctl restart apache2.
4. Después debemos descargar la aplicación web y el programa de Python compare_image, disponible en el siguiente enlace: https://github.com/DavidRodL/Chess_Transmission_Aplication
5. Tras la descarga, deberemos mover la carpeta tfg a la siguiente ruta: /var/www/html y la carpeta compare_image la dejaremos en descargas, junto con la carpeta LiveChess2FEN (descargada en el paso 1).

6. A las carpetas mencionadas en el apartado anterior, hay que otorgarles todos los permisos (escritura, lectura y ejecución) para cualquier usuario.
7. Es necesario cambiar las rutas de direcciones de los archivos .php y de compara_imagenv9_final.py para que se ajusten a las de la máquina en la que se ejecute.
8. Para utilizar la aplicación, es necesario colocar una cámara encima del tablero, por lo que si no se tiene ninguna cámara, se pueden utilizar aplicaciones que conecten la cámara del teléfono con la máquina como (droidcam).
9. Tras realizar los pasos anteriores, hay que abrir un navegador y escribir: http://localhost/tfg/aplicacionv1/ y ya tendremos la aplicación web en funcionamiento.