

Detección y Clasificación Automática de Tableros de Ajedrez para la Generación de Notación FEN mediante Redes Neuronales Convolucionales

Pablo Gálvez Ortigosa
Eduardo Rodríguez Cao
José Luis Mera Cardoso
Mario Megías Mateo

January 13, 2025

Índice

1 Introducción

2 Estado del arte

3 Dataset

4 Métodos

Detección de tablero

Clasificación por ocupación

Clasificación por piezas

Detalles técnicos e implementación

5 Experimentos

6 Resultados

7 Conclusiones

1 Introducción

2 Estado del arte

③ Dataset

4 Métodos

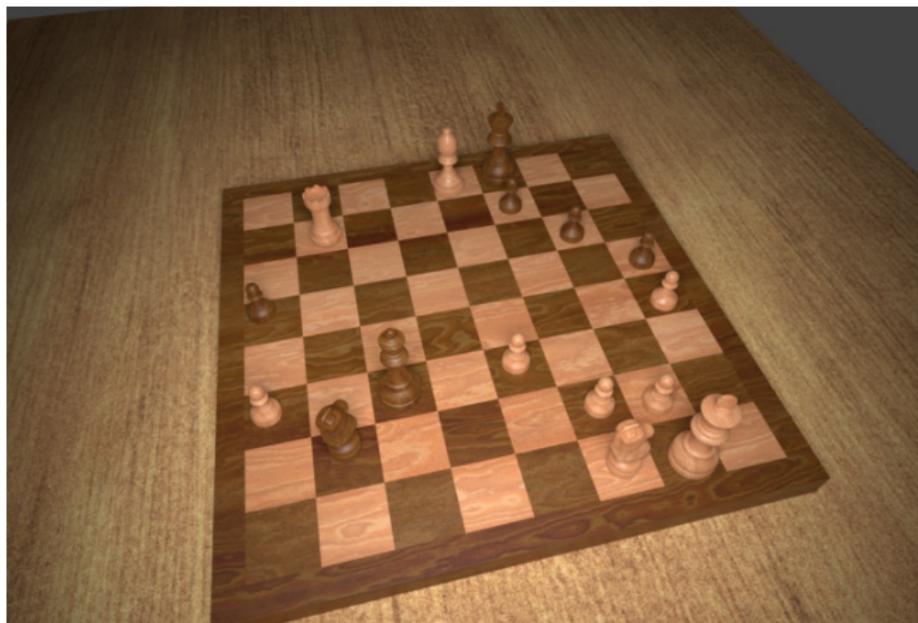
5 Experimentos

6 Resultados

7 Conclusiones

Introducción

Objetivo: identificar la configuración que siguen las piezas de ajedrez de un tablero (en notación FEN) a partir de una imagen.



Introducción



Figure 1: Ejemplo de FEN tomado de chess.com

Introducción

- Modelo con tres etapas: detector de tablero, clasificador de ocupación de casilla y clasificador de pieza en la casilla.

Introducción

- Modelo con tres etapas: detector de tablero, clasificador de ocupación de casilla y clasificador de pieza en la casilla.
- Combinamos métodos tradicionales con aprendizaje profundo (CNNs).

Introducción

- Modelo con tres etapas: detector de tablero, clasificador de ocupación de casilla y clasificador de pieza en la casilla.
- Combinamos métodos tradicionales con aprendizaje profundo (CNNs).
- Dataset 3D generado con Blender.

Estado del arte

- Modelos end-to-end con CNNs [3, 7].

Estado del arte

- Modelos end-to-end con CNNs [3, 7].
 - Modelos separados en etapas:

Estado del arte

- Modelos end-to-end con CNNs [3, 7].
 - Modelos separados en etapas:
 - Detección de casillas con detectores de líneas rectas [3].

Estado del arte

- Modelos end-to-end con CNNs [3, 7].
 - Modelos separados en etapas:
 - Detección de casillas con detectores de líneas rectas [3].
 - Detector de esquinas Harris [1].

Estado del arte

- Modelos end-to-end con CNNs [3, 7].
 - Modelos separados en etapas:
 - Detección de casillas con detectores de líneas rectas [3].
 - Detector de esquinas Harris [1].
 - SIFT o HoG [4].

Estado del arte

- Modelos end-to-end con CNNs [3, 7].
 - Modelos separados en etapas:
 - Detección de casillas con detectores de líneas rectas [3].
 - Detector de esquinas Harris [1].
 - SIFT o HoG [4].
 - Otros modelos de dos-tres etapas que han tenido éxito [9].

Estado del arte

- Modelos end-to-end con CNNs [3, 7].
- Modelos separados en etapas:
 - Detección de casillas con detectores de líneas rectas [3].
 - Detector de esquinas Harris [1].
 - SIFT o HoG [4].
 - Otros modelos de dos-tres etapas que han tenido éxito [9].
- Modelo que usamos nosotros propuesto por Wölflein [11]:

Estado del arte

- Modelos end-to-end con CNNs [3, 7].
 - Modelos separados en etapas:
 - Detección de casillas con detectores de líneas rectas [3].
 - Detector de esquinas Harris [1].
 - SIFT o HoG [4].
 - Otros modelos de dos-tres etapas que han tenido éxito [9].
 - Modelo que usamos nosotros propuesto por Wölflein [11]:
 - Detector de tablero.

Estado del arte

- Modelos end-to-end con CNNs [3, 7].
- Modelos separados en etapas:
 - Detección de casillas con detectores de líneas rectas [3].
 - Detector de esquinas Harris [1].
 - SIFT o HoG [4].
 - Otros modelos de dos-tres etapas que han tenido éxito [9].
- Modelo que usamos nosotros propuesto por Wölflein [11]:
 - Detector de tablero.
 - Clasificador por ocupación.

Estado del arte

- Modelos end-to-end con CNNs [3, 7].
- Modelos separados en etapas:
 - Detección de casillas con detectores de líneas rectas [3].
 - Detector de esquinas Harris [1].
 - SIFT o HoG [4].
 - Otros modelos de dos-tres etapas que han tenido éxito [9].
- Modelo que usamos nosotros propuesto por Wölflein [11]:
 - Detector de tablero.
 - Clasificador por ocupación.
 - Clasificador por piezas.

1 Introducción

2 Estado del arte

3 Dataset

4 Métodos

5 Experimentos

6 Resultados

7 Conclusiones

Dataset

“Rendered Chess Game State Images” de Open Science Framework [10]. 4888 imágenes 1200 x 800 (90% de entrenamiento, 3% de validación y 7% de test).



Figure 2: Tablero con visualización de los datos de su JSON

Dataset preprocesado para la clasificación por ocupación

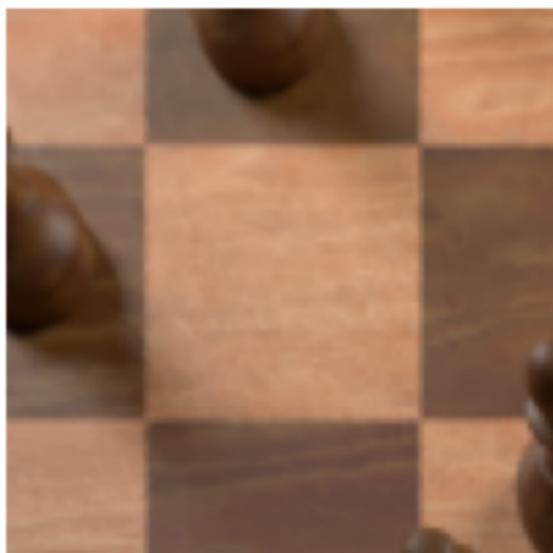


Figure 3: Casilla vacía

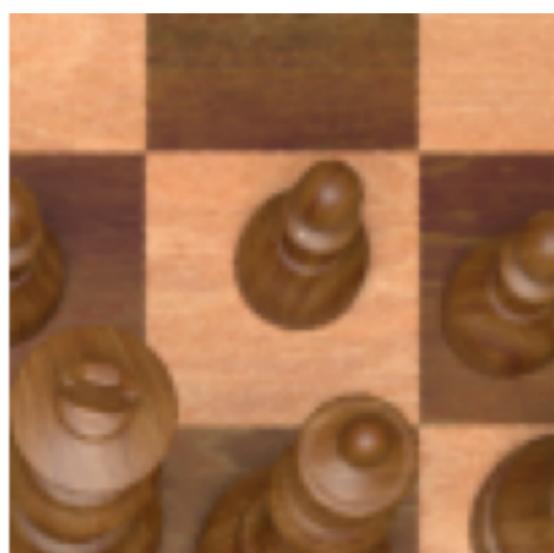


Figure 4: Casilla ocupada

Dataset preprocesado para la clasificación de piezas



Figure 5: Alfil negro



Figure 6: Peón blanco

1 Introducción

2 Estado del arte

3 Dataset

4 Métodos

Detección de tablero

Clasificación por ocupación

Clasificación por piezas

Detalles técnicos e implementación

5 Experimentos

6 Resultados

1 Introducción

2 Estado del arte

3 Dataset

4 Métodos

Detección de tablero

Clasificación por ocupación

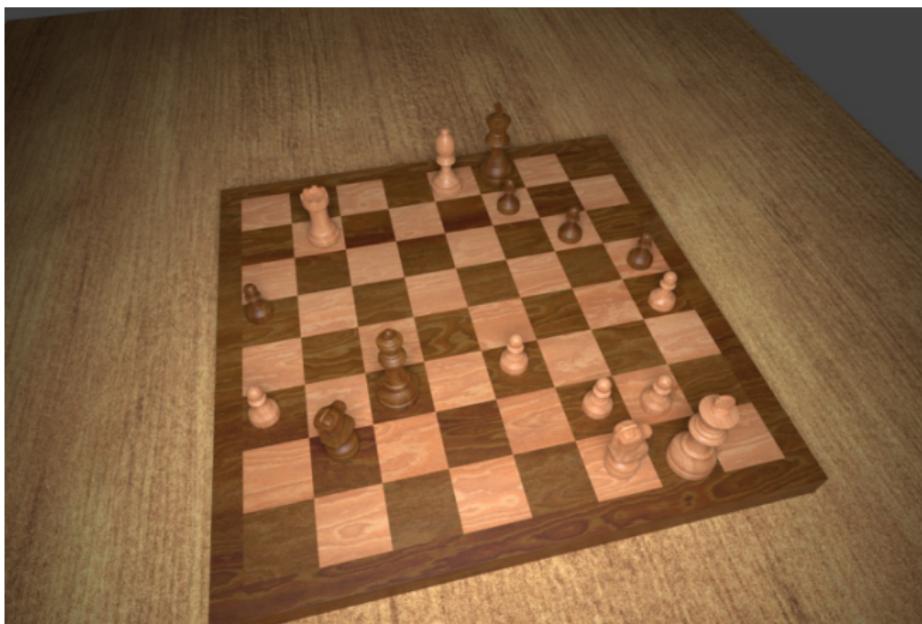
Clasificación por piezas

Detalles técnicos e implementación

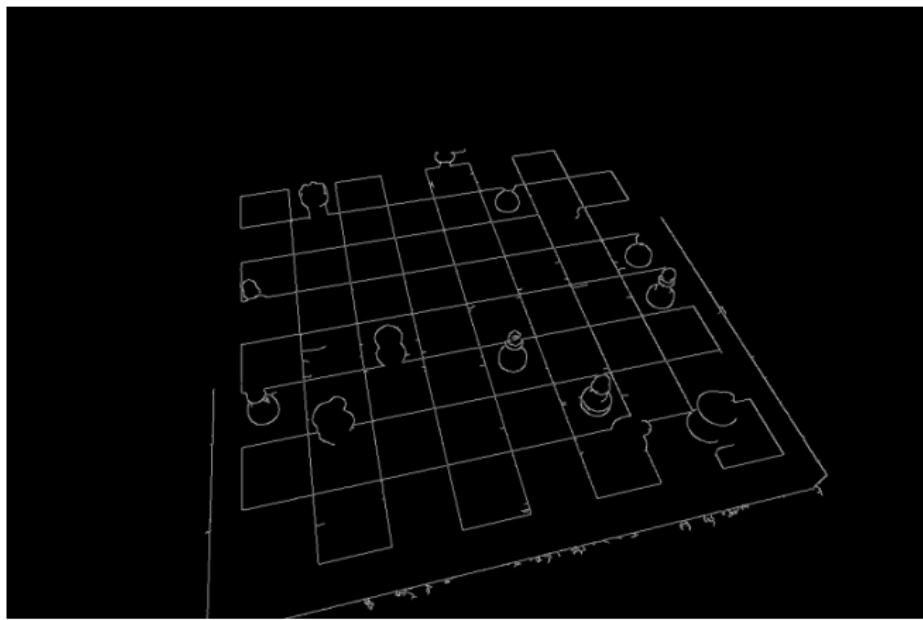
5 Experimentos

6 Resultados

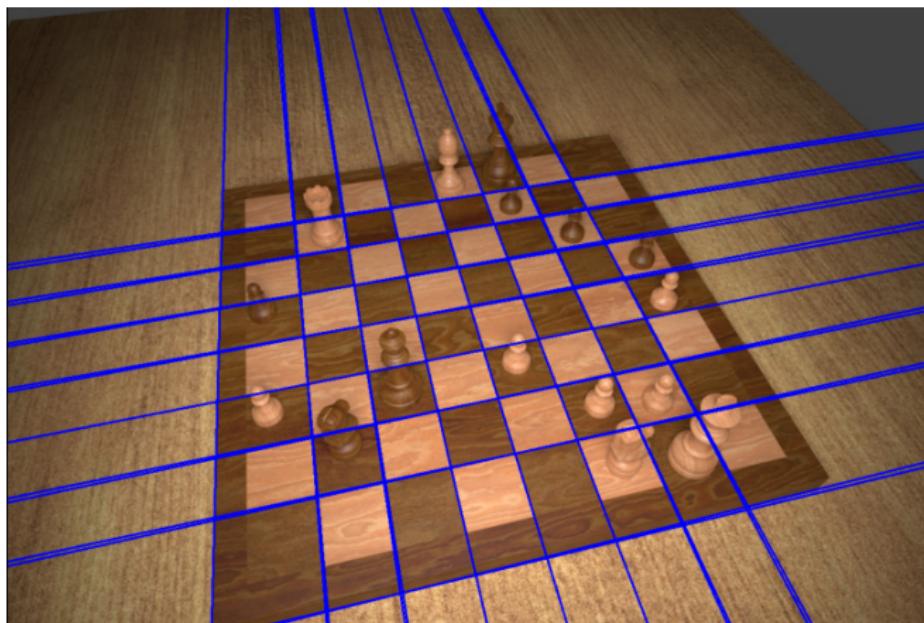
Ejemplo de tablero que usamos



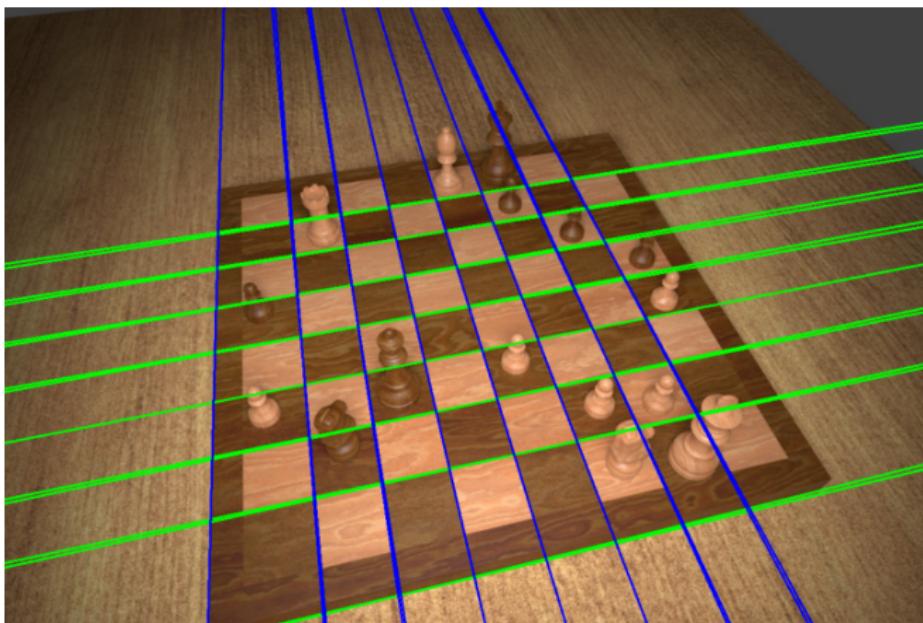
Detector de bordes Canny



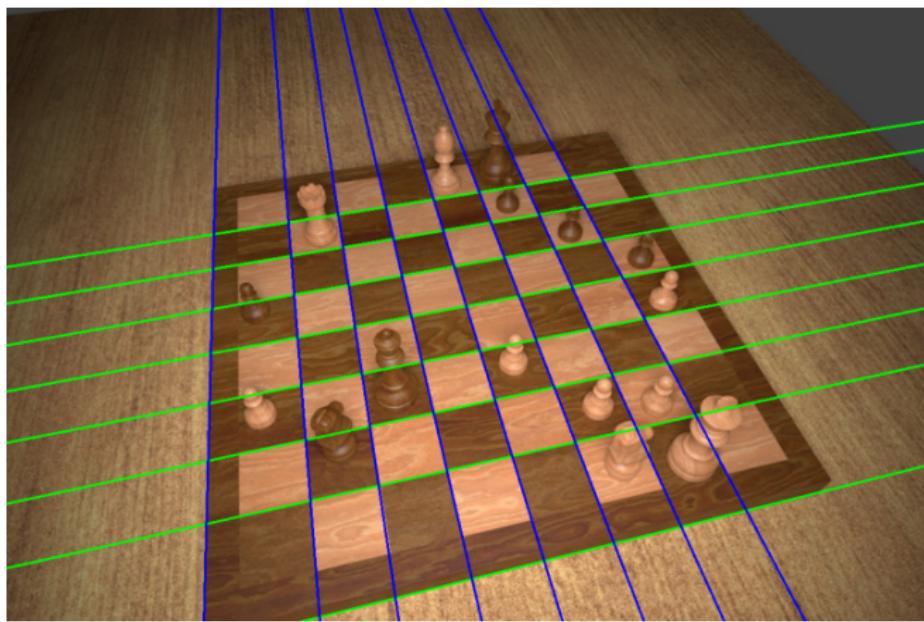
Detección de rectas con transformada de Hough



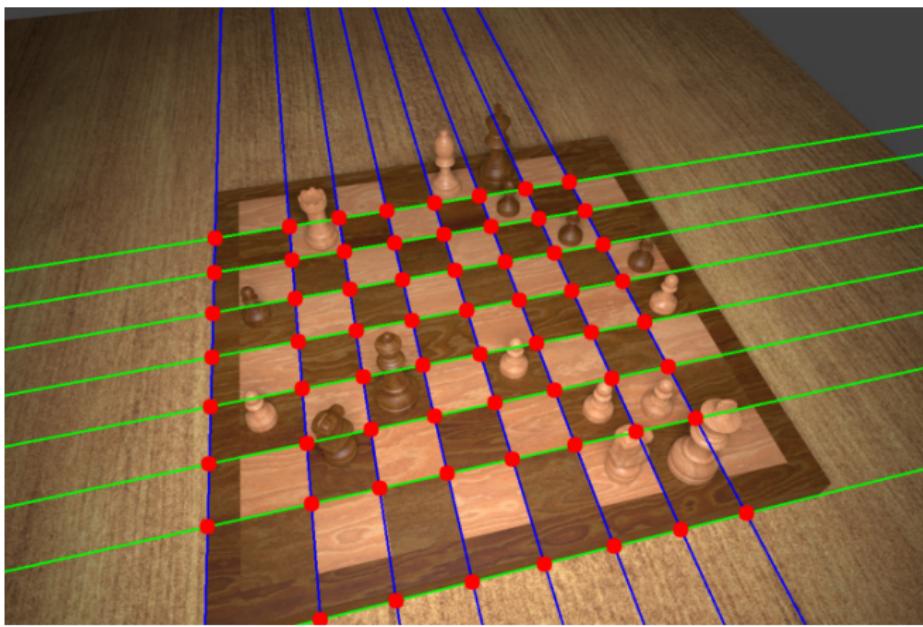
Clustering aglomerativo para distinguir rectas verticales y horizontales



Eliminación de líneas similares con DBSCAN



Puntos de intersección



Cálculo de homografía con RANSAC

- **Problema 1:** Rectas faltantes.



Figure 7: Tablero original

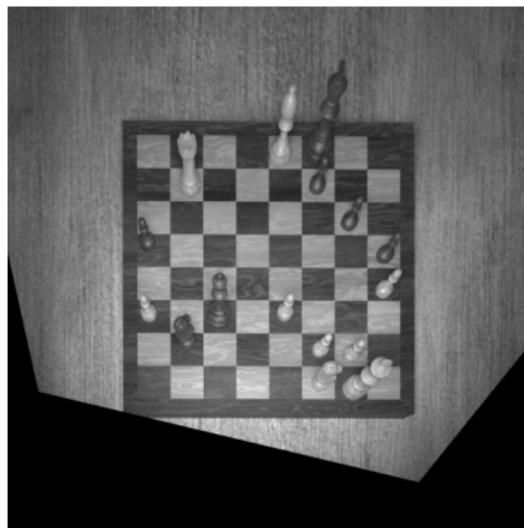


Figure 8: Tablero distorsionado

Cálculo de homografía con RANSAC

- **Problema 1:** Rectas faltantes.
- **Problema 2:** Rectas incorrectamente detectadas.



Figure 7: Tablero original

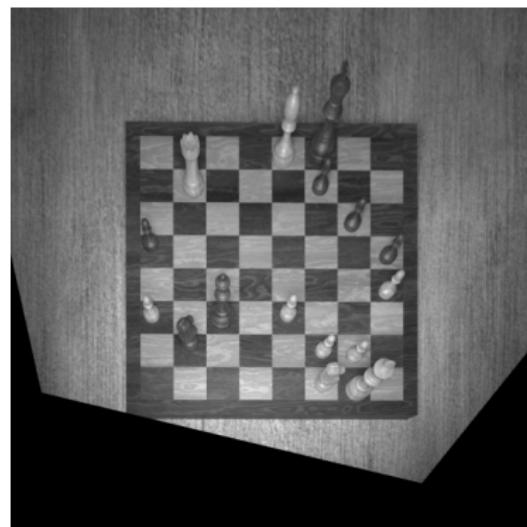
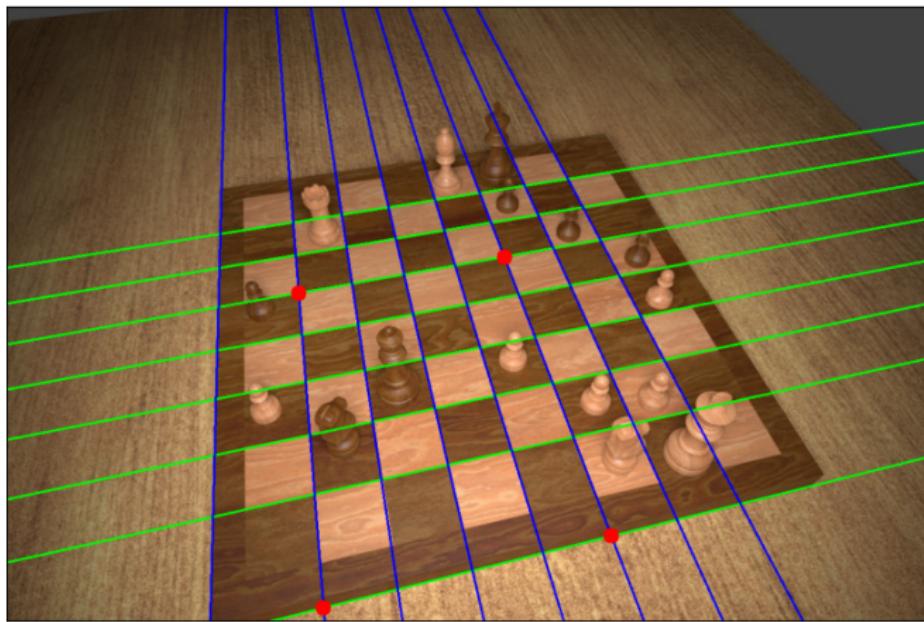
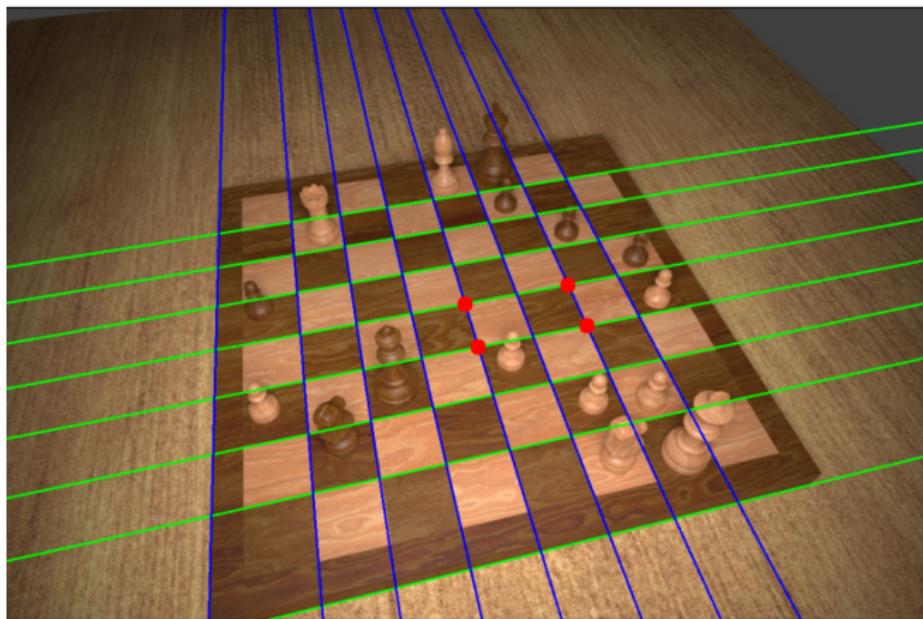


Figure 8: Tablero distorsionado

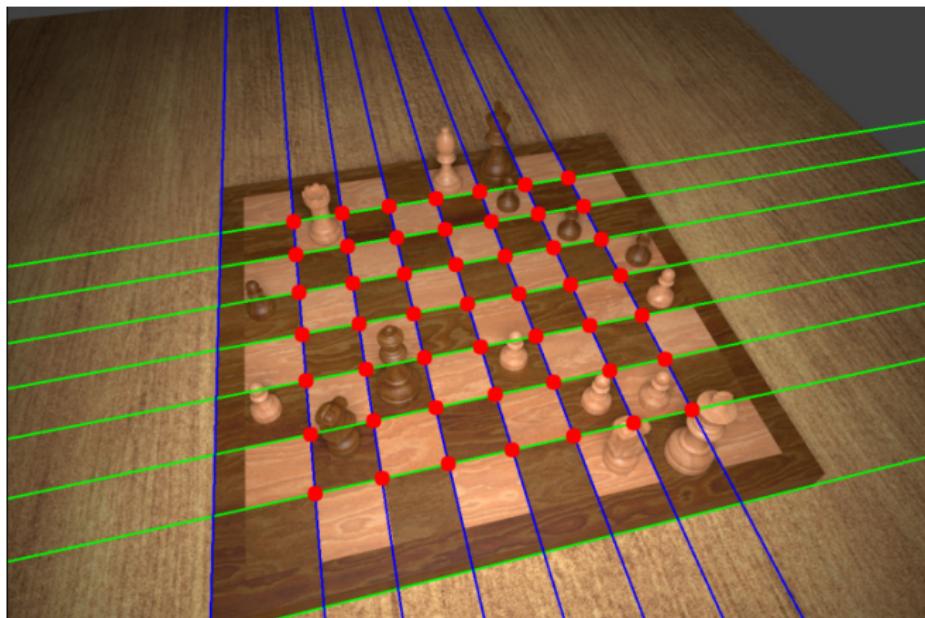
RANSAC: muestreo malo con 0 inliers



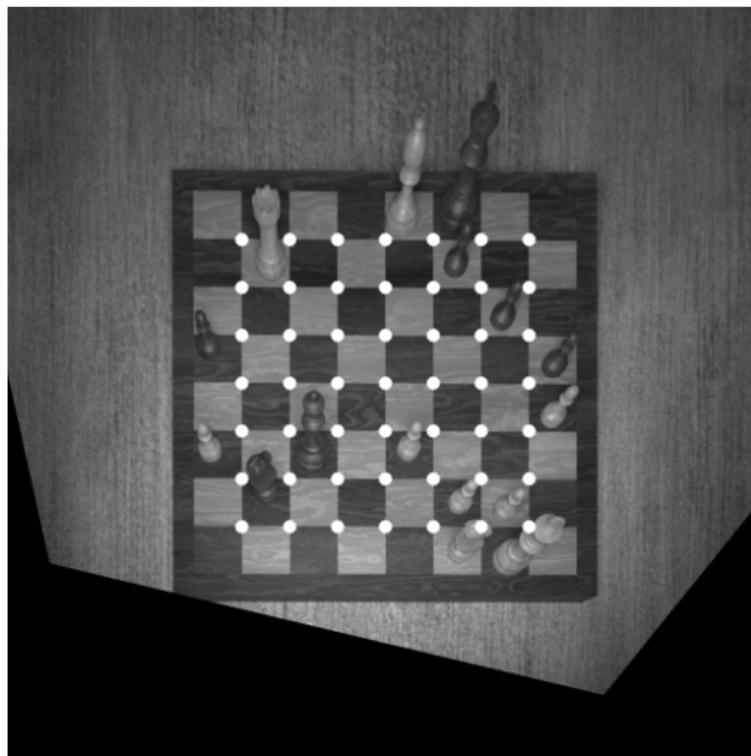
RANSAC: muestreo bueno con 49 inliers



RANSAC: mejor resultado obtenido



Vista desde arriba tras homografía (con puntos inliers)



Refinamiento de bordes verticales con Sobel y Canny

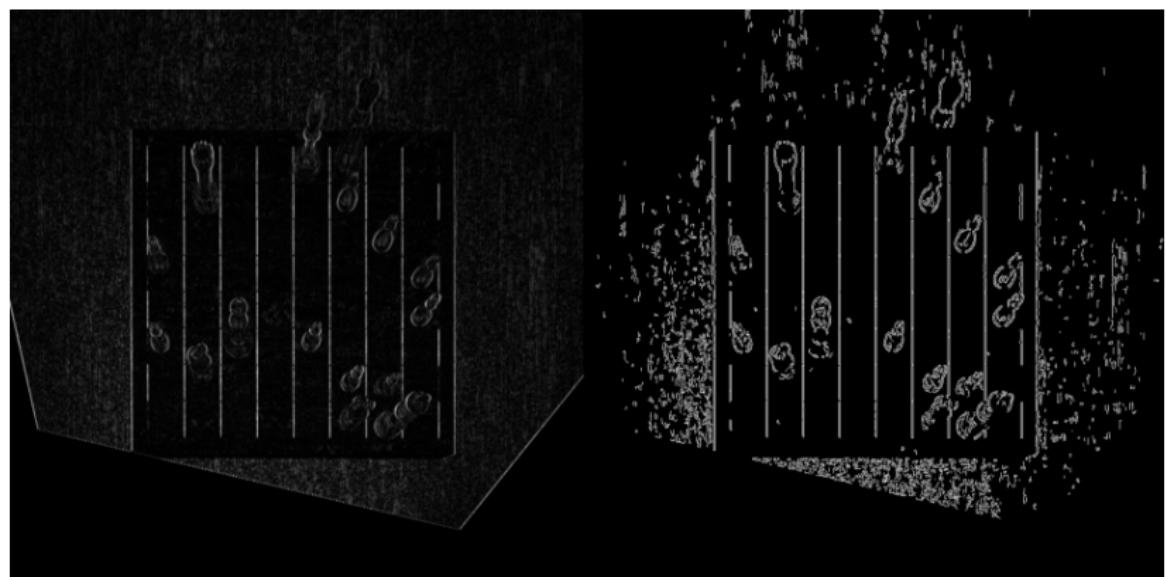


Figure 9: Sobel horizontal (izquierda) y Canny (derecha)

Refinamiento de bordes horizontales con Sobel y Canny

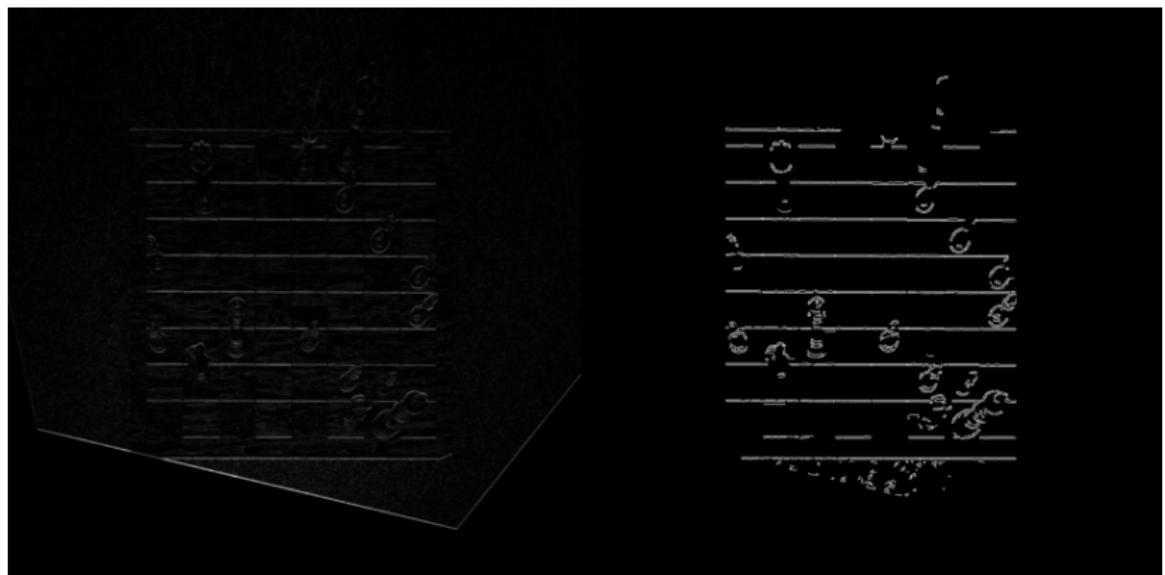


Figure 10: Sobel vertical (izquierda) y Canny (derecha)

Esquinas encontradas



1 Introducción

② Estado del arte

③ Dataset

4 Métodos

Detección de tablero

Clasificación por ocupación

Clasificación por piezas

Detalles técnicos e implementación

5 Experimentos

6 Resultados

Clasificación por ocupación

- CNN con 3 capas de convolución, 3 de pooling y 3 densas
- ResNet18 preentrenada en ImageNet



Clasificación por ocupación

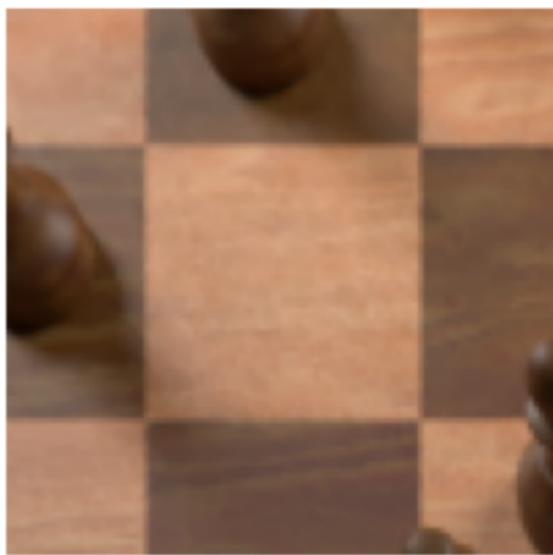


Figure 11: Casilla vacía

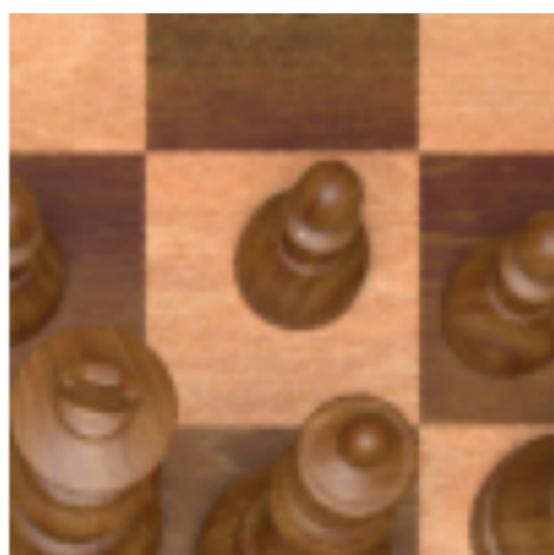


Figure 12: Casilla ocupada

1 Introducción

2 Estado del arte

3 Dataset

4 Métodos

Detección de tablero

Clasificación por ocupación

Clasificación por piezas

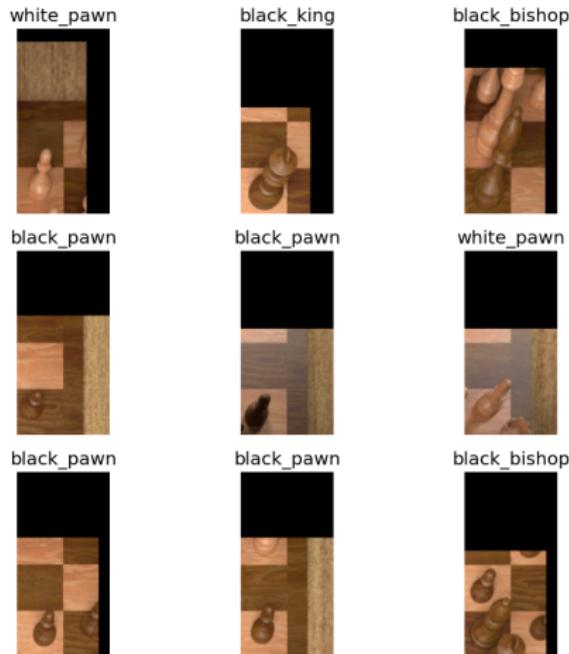
Detalles técnicos e implementación

5 Experimentos

6 Resultados

Clasificación por piezas

- CNN con 3 capas de convolución, 3 de pooling y 3 densas
- ResNet18 preentrenada en ImageNet



Clasificación por piezas



Figure 13: Alfil negro

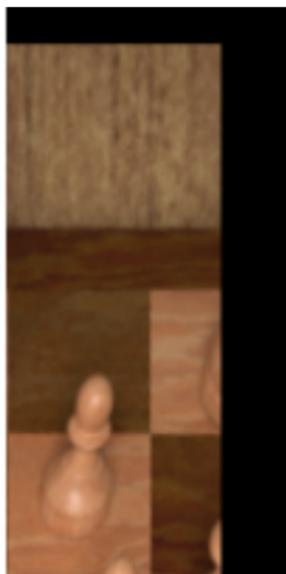


Figure 14: Peón blanco

1 Introducción

2 Estado del arte

3 Dataset

4 Métodos

Detección de tablero

Clasificación por ocupación

Clasificación por piezas

Detalles técnicos e implementación

5 Experimentos

6 Resultados

Detalles técnicos e implementación

- Librerías usadas: Scikit-Learn [8], OpenCV [2], FastAI[6], chess[5].
- Los parámetros de los algoritmos tales como Canny y Hough se han encontrado haciendo grid search sobre un conjunto de parámetros razonable en un subconjunto pequeño del dataset.

1 Introducción

2 Estado del arte

3 Dataset

4 Métodos

5 Experimentos

6 Resultados

7 Conclusiones

Experimentos

- Entrenamos en el dataset preprocesado durante 5 épocas con batchsize 32.

Experimentos

- Entrenamos en el dataset preprocesado durante 5 épocas con batchsize 32.
 - Función de pérdida Cross Entropy Loss y optimizador Adam.

Experimentos

- Entrenamos en el dataset preprocesado durante 5 épocas con batchsize 32.
 - Función de pérdida Cross Entropy Loss y optimizador Adam.
 - Dos modelos: CNN con learning rate del finder de fastai y 0.01 para la ResNet18.

Experimentos

- Entrenamos en el dataset preprocesado durante 5 épocas con batchsize 32.
- Función de pérdida Cross Entropy Loss y optimizador Adam.
- Dos modelos: CNN con learning rate del finder de fastai y 0.01 para la ResNet18.
- Tiempo de entrenamiento: 60 minutos y 1.3 horas respectivamente en ocupación, 22 minutos y 1.13 horas en piezas.

Experimentos

- Entrenamos en el dataset preprocesado durante 5 épocas con batchsize 32.
- Función de pérdida Cross Entropy Loss y optimizador Adam.
- Dos modelos: CNN con learning rate del finder de fastai y 0.01 para la ResNet18.
- Tiempo de entrenamiento: 60 minutos y 1.3 horas respectivamente en ocupación, 22 minutos y 1.13 horas en piezas.
- Se usó una GPU NVIDIA T4.

1 Introducción

2 Estado del arte

3 Dataset

4 Métodos

5 Experimentos

6 Resultados

7 Conclusiones

Detector de tablero

- Distancia en píxeles entre esquinas predichas y verdaderas.

Detector de tablero

- Distancia en píxeles entre esquinas predichas y verdaderas.
 - Accuracy, considerando como error cuando la diferencia entre la esquina calculada con la real es mayor de 1% de la anchura de la imagen.

Detector de tablero

- Distancia en píxeles entre esquinas predichas y verdaderas.
 - Accuracy, considerando como error cuando la diferencia entre la esquina calculada con la real es mayor de 1% de la anchura de la imagen.
 - **Resultado:** accuracy de 100% y un error medio de 1.36 píxeles.

Clasificador por ocupación

Modelo	Accuracy	F1-weighted
CNN	99.75%	99.75%
Resnet18	100%	100%

Table 1: Evaluación de distintos modelos del clasificador de ocupación en el conjunto de validación

Evaluada en el conjunto de test, la Resnet18 tiene *accuracy* 99.97% y *F1-weighted* 99.97%.

Clasificador por piezas

Modelo	Accuracy	F1-weighted
CNN	95.7%	95.7%
Resnet18	99.94%	99.94%

Table 2: Evaluación de distintos modelos del clasificador de piezas en el conjunto de validación

Evaluada en el conjunto de test, la Resnet18 tiene *accuracy* 99.97% y *F1-weighted* 99.97%.

Pipeline completo

Métrica	Val	Test
Número medio de casillas incorrectas por tablero	0.01	0.02
Tableros sin errores (%)	99.32	97.66
Tableros con ≤ 1 errores (%)	100	100
Tasa de error por casilla (%)	0.01	0.04
Accuracy del detector de esquinas (%)	100	100
Accuracy del clasificador de ocupación por casilla (%)	100	99.97
Accuracy del clasificador de piezas por casilla (%)	99.99	100

Table 3: Evaluación del pipeline completo en el conjunto de validación y test

1 Introducción

2 Estado del arte

3 Dataset

4 Métodos

5 Experimentos

6 Resultados

7 Conclusiones

Conclusiones

- **Objetivo alcanzado:** automatizar la detección del estado de juego en un tablero de ajedrez a partir de una imagen.

Conclusiones

- **Objetivo alcanzado:** automatizar la detección del estado de juego en un tablero de ajedrez a partir de una imagen.
 - **Modelo final:** detector de tablero con métodos clásicos y dos ResNet18 finetuneadas.

Conclusiones

- **Objetivo alcanzado:** automatizar la detección del estado de juego en un tablero de ajedrez a partir de una imagen.
- **Modelo final:** detector de tablero con métodos clásicos y dos ResNet18 finetuneadas.
- **Resultado:** excelente desempeño en el conjunto de prueba, clasificando todos los tableros con un máximo de un error y logrando un promedio de tan solo 0.02 casillas incorrectas por tablero.

Conclusiones

- **Objetivo alcanzado:** automatizar la detección del estado de juego en un tablero de ajedrez a partir de una imagen.
- **Modelo final:** detector de tablero con métodos clásicos y dos ResNet18 finetuneadas.
- **Resultado:** excelente desempeño en el conjunto de prueba, clasificando todos los tableros con un máximo de un error y logrando un promedio de tan solo 0.02 casillas incorrectas por tablero.
- **Trabajo futuro:**

Conclusiones

- **Objetivo alcanzado:** automatizar la detección del estado de juego en un tablero de ajedrez a partir de una imagen.
- **Modelo final:** detector de tablero con métodos clásicos y dos ResNet18 finetuneadas.
- **Resultado:** excelente desempeño en el conjunto de prueba, clasificando todos los tableros con un máximo de un error y logrando un promedio de tan solo 0.02 casillas incorrectas por tablero.
- **Trabajo futuro:**
 - Transfer learning a datasets nunca vistos.

Conclusiones

- **Objetivo alcanzado:** automatizar la detección del estado de juego en un tablero de ajedrez a partir de una imagen.
- **Modelo final:** detector de tablero con métodos clásicos y dos ResNet18 finetuneadas.
- **Resultado:** excelente desempeño en el conjunto de prueba, clasificando todos los tableros con un máximo de un error y logrando un promedio de tan solo 0.02 casillas incorrectas por tablero.
- **Trabajo futuro:**
 - Transfer learning a datasets nunca vistos.
 - Experimentar con transformers como CLIP y ViT y compararlos con nuestras CNNs.

Bibliografía |

- [1] N Banerjee, D Saha, A Singh, and G Sanyal. A simple autonomous chess playing robot for playing chess against any opponent in real time. In *International Conference on Computational Vision and Robotics; Institute for Project Management: Bhubaneshwar, India*, 2012.
- [2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
- [3] Maciej A. Czyzewski, Artur Laskowski, and Szymon Wasik. Chessboard and chess piece recognition with the support of neural networks. 2020.
- [4] Jialin Ding. Chessvision : Chess board and piece recognition. 2016.

Bibliografía II

- [5] Niklas Fiekas. python-chess: A chess library for python, 2017. Accessed on December 25, 2024.
- [6] Jeremy Howard and Sylvain Gugger. Fastai: A layered api for deep learning. *Information*, 11(2):108, 2020.
- [7] Athanasios Masouris and Jan van Gemert. End-to-end chess recognition. *ArXiv*, abs/2310.04086, 2023.
- [8] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Bibliografía III

- [9] David Mallasén Quintana, Alberto Antonio del Barrio García, and Manuel Prieto Matías. Livechess2fen: a framework for classifying chess pieces based on cnns. 2020.
- [10] Georg Wölflein and Ognjen Arandjelović. Dataset of rendered chess game state images. 2021.
- [11] Georg Wölflein and Ognjen Arandjelović. Determining chess game state from an image. *Journal of Imaging*, 7(6), 2021.