

**Chess game registration**

Vision Systems in Robotics

Authors: Jan Gallina  
 Jakub Górski

Field of study: Automatic Control and Robotics

Specialization: Komputerowe Systemy Sterowania

Semester: I

Cracow, 2025 r.

**Table of contents**

Table of contents

[1. Introduction 3](#_Toc201090446)

[1.1. Objective 3](#_Toc201090447)

[1.2. Overview of used technologies 3](#_Toc201090448)

[2. Method Overview 3](#_Toc201090449)

[2.1. Perspective Transformation 3](#_Toc201090450)

[2.2. Edge detection 5](#_Toc201090451)

[2.3. Chessboard Square Extraction 8](#_Toc201090452)

[2.4. Piece Detection 9](#_Toc201090453)

[3. CNN Model 11](#_Toc201090454)

[4. Conclusion 15](#_Toc201090455)

[References 16](#_Toc201090456)

# Introduction

## Objective

The goal of this project was to record a game of chess using a webcam. We decided to use a convolutional neural network (CNN) to detect and classify chess pieces on a real chessboard in real time.

## Overview of used technologies

During this project, we made use of various technologies from different areas. To accomplish the project goals, the following tools and libraries were used:

* Python language – main programming language used for implementation
* OpenCV – library for image processing and computer vision tasks
* TensorFlow – to build, train and run the convolutional neural network (CNN)
* NumPy – for numerical operations and matrix manipulation

# Method Overview

The following sections present the steps taken throughout the project. At the beginning of each subsection, a brief description of a program component is provided, followed by the corresponding code and the results obtained.

## Perspective Transformation

Initial attempts to detect the chessboard structure using edge detection revealed that the process was highly sensitive to even slight changes in the camera angle or board position. Since the parameters used for edge detection were manually tuned, any small shift in perspective caused the algorithm to fail in detecting the correct lines and intersections.

To solve this issue, a perspective transformation was applied before performing edge detection. At the start of the program, the user is asked to manually select the four corners of the chessboard using the mouse, based on the live webcam feed. These points are then used to calculate a transformation matrix, which adjusts the image so that the chessboard is properly aligned and centered. This step significantly improves the stability and accuracy of edge detection in the later stages of the pipeline.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

Figure 1 - selecting corners of the chessboard

After selecting corners by the user, perspective change matrix is calculated. Then it is used to align chessboard in the center of the image.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

Figure 2 - calculating perspective change matrix and warping perspective

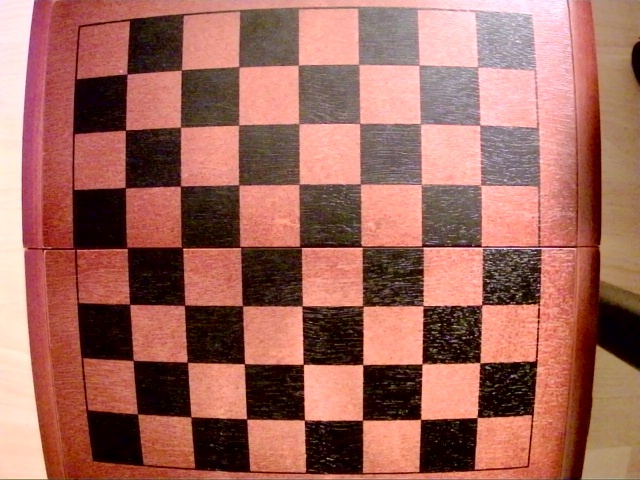


Figure 3 - original image from the camera

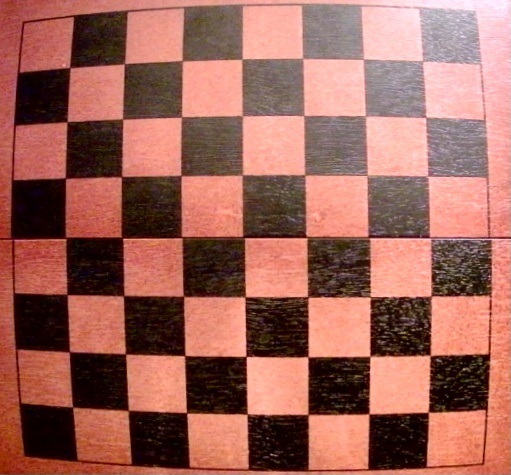


Figure 4 - image after warping perspective

After these operations, the image contains only the chessboard, with all other elements removed. It is centered and perfectly aligned with the camera view, which makes the edge detection algorithm more reliable.

## Edge detection

After applying the perspective transformation, the next step was to detect the edges of the chessboard using the Canny Edge Detection algorithm, introduced during lectures. Based on the transformed image of the empty chessboard (without pieces), the algorithm identifies the edges. Then, using the Hough Transform, the detected edges are drawn as straight lines on the image. This approach makes it possible to extract all individual squares of the chessboard. However, without the earlier perspective correction, this method was too sensitive to variations in the camera view.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

Figure 5 - Clache algoritm

The CLAHE (Contrast Limited Adaptive Histogram Equalization) algorithm was used to enhance the visibility of details in both overexposed and underexposed areas of the image captured by the webcam. This step helped to improve the overall contrast of the chessboard image.

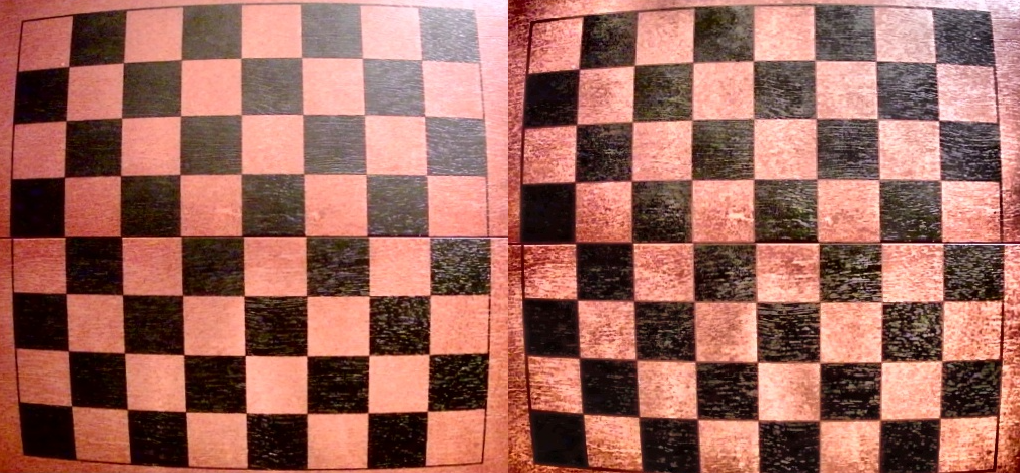


Figure 6 - before and after Clache algorithm

The image on the right is more consistent in lighting and it doesn’t have overexposed and underexposed areas.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

Figure 7 - Canny edge detection

To perform Canny edge detection effectively, it is essential to apply a Gaussian filter to smooth the image. The result is a binary image called edges, which contains only the detected edges.



Figure 8 - image after smoothing

This image was then used to detect edges using canny.

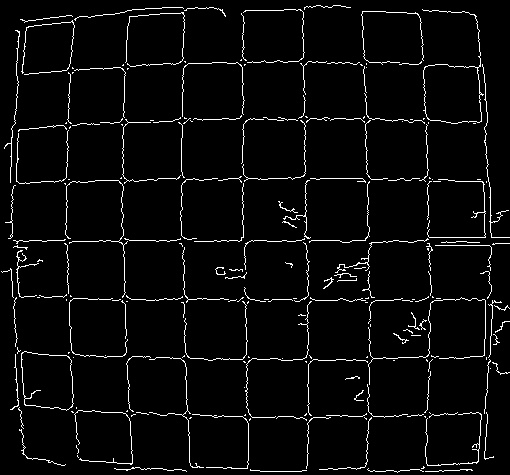


Figure 9 - Canny edge detection

Next, the Hough Transform was used to detect vertical and horizontal lines, which are necessary for the process of extracting individual chessboard squares. The lines were drawn on the original image from the camera.

Obraz zawierający wzór, kwadrat, Prostokąt, kafelek

Zawartość wygenerowana przez AI może być niepoprawna.

Figure 10 - Image from camera with vertical and horizontal lines

## Chessboard Square Extraction

Using the lines detected with the Hough Transform, the image was divided into 64 individual squares. The first step in extracting the squares was to find the intersection points of the detected lines. Then, a dictionary was created that mapped each square’s name (e.g., C1, B8) to its coordinates in the transformed image. These square regions were later used by the neural network to detect chess pieces.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

Figure 11 - finding intersections

This part of the code identifies the intersection points of the detected vertical and horizontal lines, which correspond to the corners of the chessboard squares. These points are stored in a matrix and later used to extract individual tiles from the board.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

Figure 12 - extracting tiles from image

The above part of the program stores each square's name and its coordinates from the camera image in a dictionary. In the following steps, these coordinates are used to identify which chess piece, if any, is present on each square.

## Piece Detection

The final step of the program is detecting whether a chess piece is present on a given square. Iterating through all the squares identified in the previous step, the convolutional neural network predicts the probability of each square containing a specific piece. Based on this, the program creates an interpretation of the board state, which is printed to the console. After each move, the user must press the “a” key to trigger a new scan of the board. The current board layout is then displayed in the console.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

Figure 13 - scanning each tile and preparing for CNN

Each tile is scanned and saved to a directory. The image is then resized and converted to float format to be processed by the CNN.

Obraz zawierający tekst, sztuka

Zawartość wygenerowana przez AI może być niepoprawna.

Figure 14 - scanned tile

This is what a scanned tile looks like. Based on this image, the CNN will predict which piece is present on the tile.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Zawartość wygenerowana przez AI może być niepoprawna.

Figure 15 - using CNN and saving chessboard configuration to string representation

After each move, the user can press the "a" key on the keyboard, and the program will detect the current arrangement of pieces on the chessboard and display its interpretation as text output in the console.



Figure 16 - representation of empty board

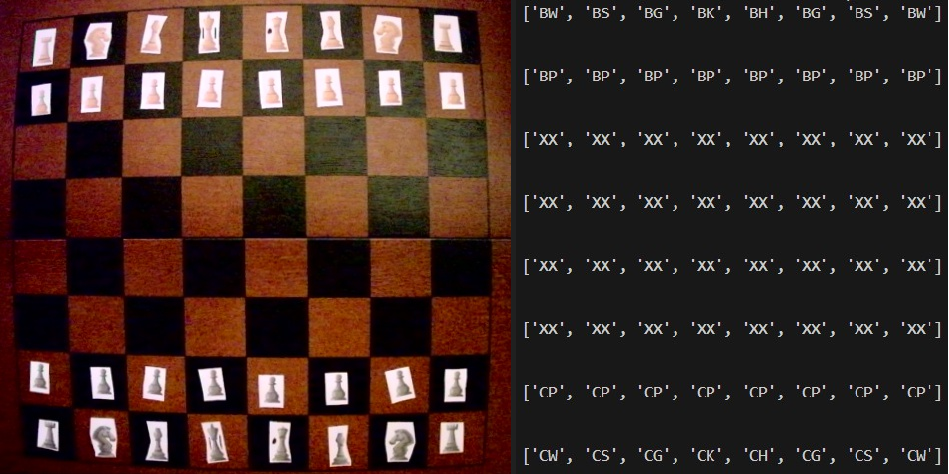


Figure 17 - representation of starting position

Each piece has his own name. (e.g. BW – *Biała Wieża,* CS – *Czarny Skoczek).*

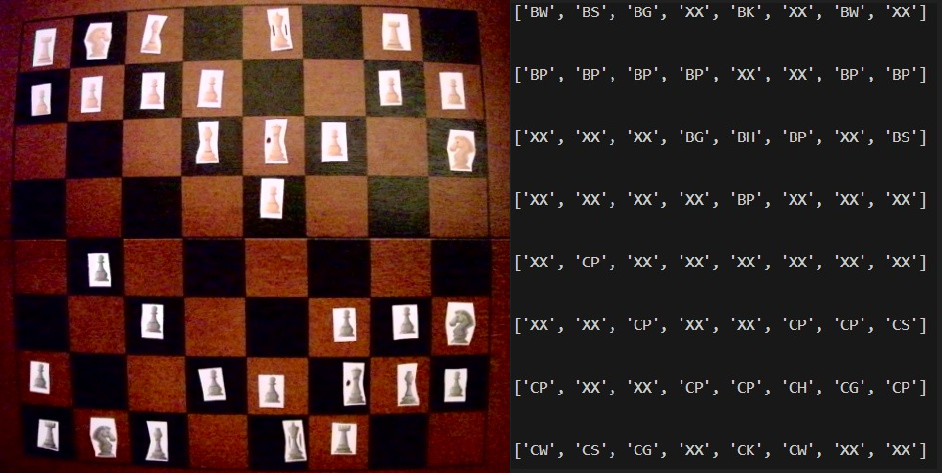


Figure 18 - representation mid game

In each scenario, the neural network correctly interpreted the image from the camera of the arranged figures.

# CNN Model

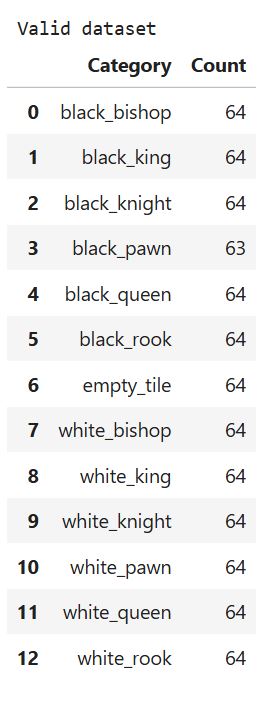
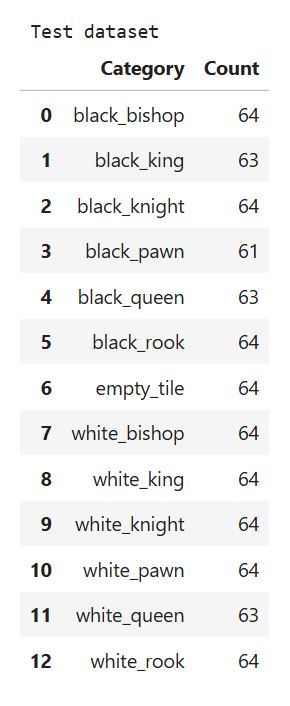
This chapter describes the stages of preparing, training, and testing a neural network for recognizing chess pieces on a chessboard. This solution is largely based on the article *"Detecting Chess Pieces with a CNN"* by Thomas S. Visser. The network was created in a separate program and saved to a .keras file, then later used in the main program.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

The first step after importing the libraries is to load the data into the test, validation, and training sets using a function from the TensorFlow library. Data is split in proportion:

* 60% training set
* 20% validate set
* 20% test set



Pieces are correctly identified by the CNN. Additionally empty space is also classified in this sets as *empty\_tile.*

Next, the process described in the article is followed. Transfer learning is applied by using a pre-trained VGG16 model trained on the large ImageNet dataset. The early convolutional layers are frozen to preserve their learned low-level features, while only the later layers and a new "head" are trained on the smaller chess piece dataset.

The head consists of two hidden layers with 256 nodes each, batch normalization layers to normalize inputs and speed up training, and a dropout rate of 0.4 to prevent overfitting. Data augmentation techniques like random contrast, horizontal flips, and slight translations are used to increase data diversity.

The model is compiled with the Adam optimizer and categorical cross-entropy loss, suitable for the 13-class classification task. Early stopping is implemented to halt training if validation loss does not improve after 30 epochs, ensuring the model does not overfit and the best weights are restored.

Next, the network is trained, and the results are displayed on various plots where the improvement of the network’s performance can be visually observed from epoch to epoch.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

Figure 19 - training the CNN

Obraz zawierający tekst, zrzut ekranu, linia, diagram

Zawartość wygenerowana przez AI może być niepoprawna.

Figure 20 - Loss through epochs

Obraz zawierający tekst, zrzut ekranu, linia, diagram

Zawartość wygenerowana przez AI może być niepoprawna.

Figure 21 - Accuracy through epochs

Obraz zawierający tekst, zrzut ekranu, kwadrat, Prostokąt

Zawartość wygenerowana przez AI może być niepoprawna.

Figure 22 - Confusion Matrix - Train Data

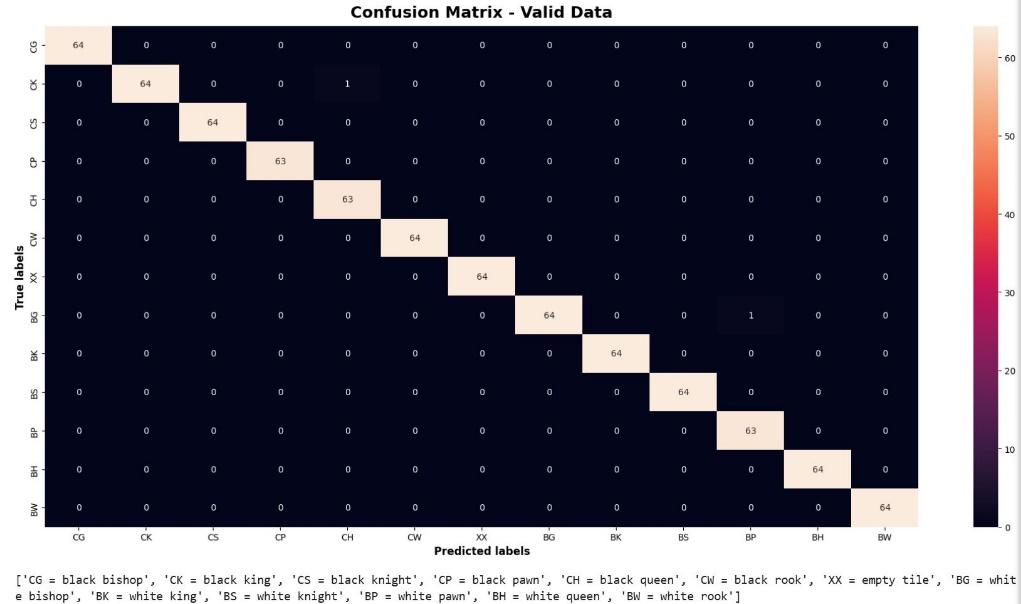


Figure 23 - Confusion Matrix - validate Data

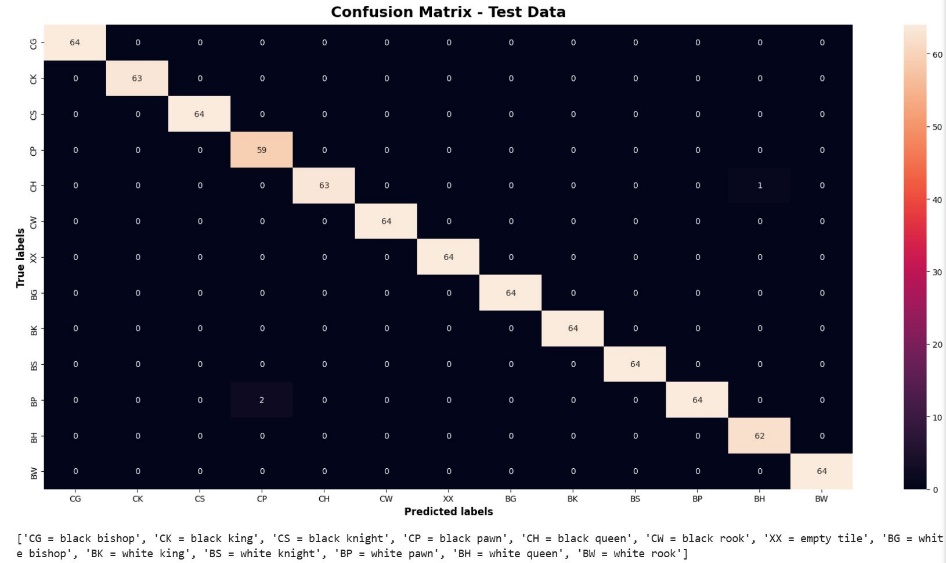


Figure 24 - Confusion Matrix - Test Data

The total accuracy across all sets was above 0.966. The lowest result was obtained on the test set. These results clearly show that the neural network performs very well in recognizing chess pieces and can be reliably used in the project.

# Conclusion

* The system successfully detects and classifies chess pieces using image processing and a trained CNN. It works in real-time and provides an accurate board representation.
* During the board recognition stage, all pieces must be removed from the chessboard. This is necessary for the Canny edge detection algorithm to correctly identify the grid lines of the board.
* At first, the dataset for training the neural network was prepared using photos of chess pieces printed in 3D. However, this approach encountered significant issues. Pieces such as bishops and pawns were difficult to distinguish in top-down images, and when the camera was positioned beside the board, taller pieces could obscure shorter ones. As a result, some pieces became invisible and could not be correctly classified by the network. For this reason, the decision was made to use 2D-printed chess pieces instead.
* The network was trained using a GPU, which significantly reduced computation time.
* Lighting conditions turned out to be a critical factor affecting recognition performance. For the network to function properly, the lighting had to remain consistent throughout the game. This requirement posed a major challenge when designing the testing setup.

# References

- TheAILerner (https://theailearner.com/tag/cv2-getperspectivetransform/, https://theailearner.com/tag/cv2-warpperspective/) - perspective change - 2D homography (method and functions)

- CLAHE: <https://www.youtube.com/watch?v=tn2kmbUVK50&t=204s&ab_channel=KevinWood%7CRobotics%26AI>

- Kaggle - "Detecting Chess Pieces with a CNN" - https://www.kaggle.com/code/thomassvisser/detecting-chess-pieces-with-a-cnn (Apache 2.0 open source license)

- chess pieces ("Designed by brgfx / Freepik")

- ChatGPT

- OpenCV and Tensorflow documentation

- lectures