

**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**

[1. Introduction 3](#_Toc201189188)

[1.1. The Objective 3](#_Toc201189189)

[1.2. The Overview of Used Technologies 3](#_Toc201189190)

[2. Method Overview 4](#_Toc201189191)

[2.1. Perspective Transformation 4](#_Toc201189192)

[2.2. Edge Detection 7](#_Toc201189193)

[2.3. Chessboard Square Extraction 10](#_Toc201189194)

[2.4. Piece Detection 11](#_Toc201189195)

[3. The CNN Model 14](#_Toc201189196)

[4. Conclusions 19](#_Toc201189197)

[References 20](#_Toc201189198)

# Introduction

## The Objective

The goal of this project was to record a game of chess using a webcam. In order to achieve that, we decided to build an algorithm that would monitor each tile of the chessboard separately and use a convolutional neural network (CNN) to detect and classify chess pieces in real time.

## The 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 library – to build, train and run the convolutional neural network (CNN),
* NumPy library – for numerical operations and matrix manipulation,
* scikit-learn library – to evaluate the results of the learning phase of the CNN.

From the hardware point of view, we must have:

* a PC/laptop,
* a webcam,
* a chessboard,
* 2D images of chess pieces.



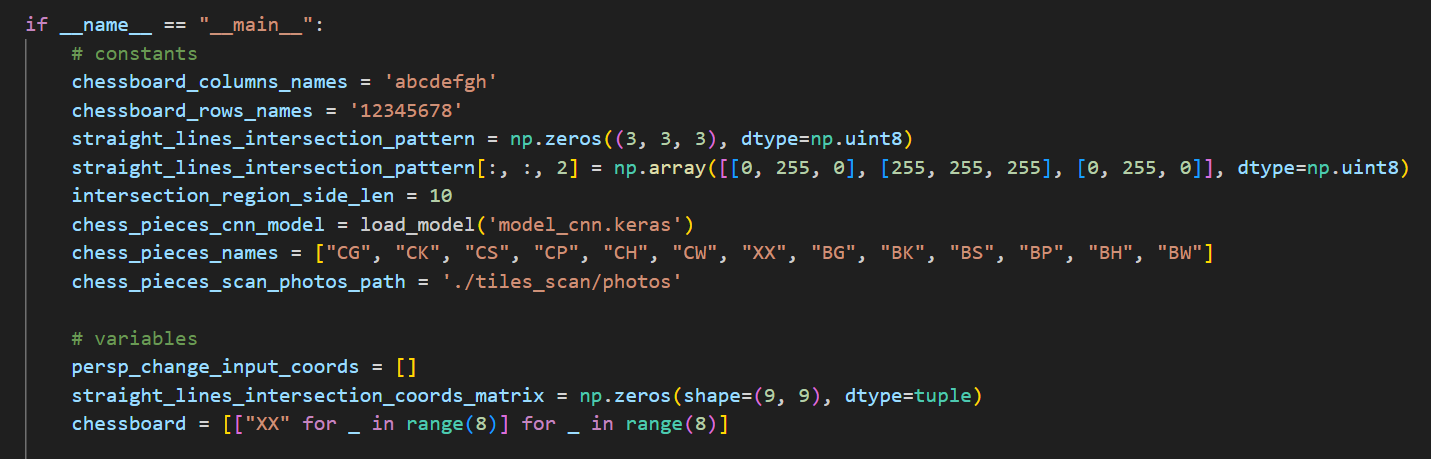
**Figure 1** – the research site

# Method Overview

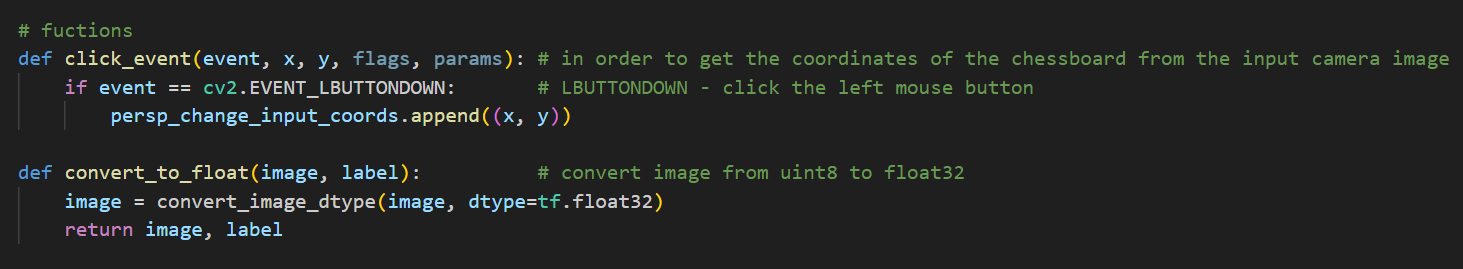
The following sections present the steps taken throughout the project. At the beginning of each subsection, a brief description of a program component is provided, followed by the corresponding code and the results obtained.

## Perspective Transformation

Let us start with presenting the defined constants, variables at the beginning of the Python file (figure 2) and the user-created functions (figure 3). They will be used in the following parts of the task.



**Figure 2** – the defined constants and variables



**Figure 3** – the user-created functions

The next goal was to set the connection with the camera. The proper definition is presented in the figure 4.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

**Figure 4** – connecting to the camera

The initial attempts to detect the chessboard structure using edge detection algorithms revealed, that the process was highly sensitive to the appearance of different objects in the camera FOV. This problem is also related to the manual tuning of the parameters used for edge detection, which was performed.

To solve this issue, a perspective transformation was applied. 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. The procedure is called the 2D homography (figures 5 and 6).

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

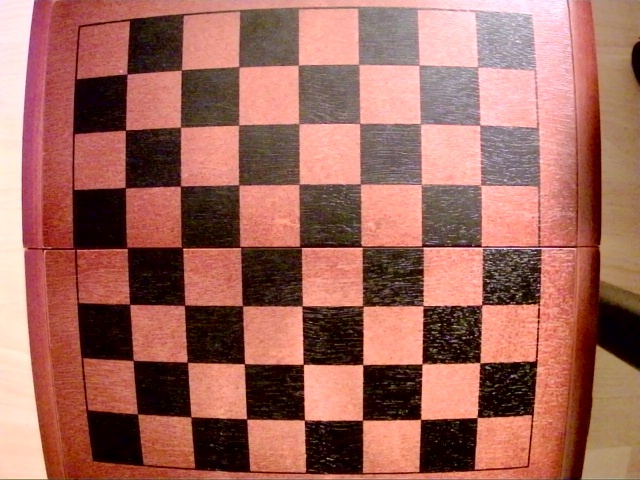
**Figure 5** – selecting corners of the chessboard

After selecting corners by the user, the perspective change matrix is calculated. Then, it is used to align chessboard in the center of the image. The input and the output for the algorithm are presented, respectfully, in the figure 7 and the figure 8.

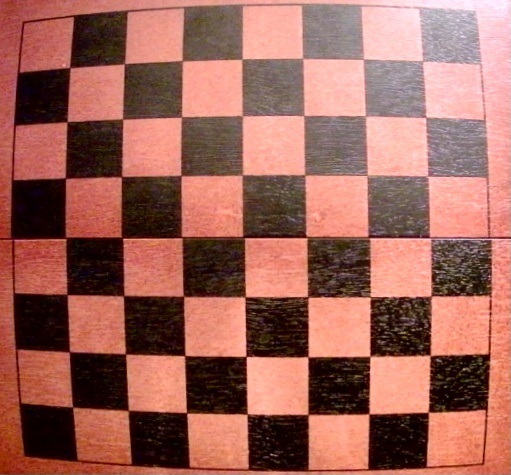
**Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.**

**Figure 6** – calculating the perspective change matrix and warping the perspective



**Figure 7** – the original image from the camera



**Figure 8** – the image after warping the 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

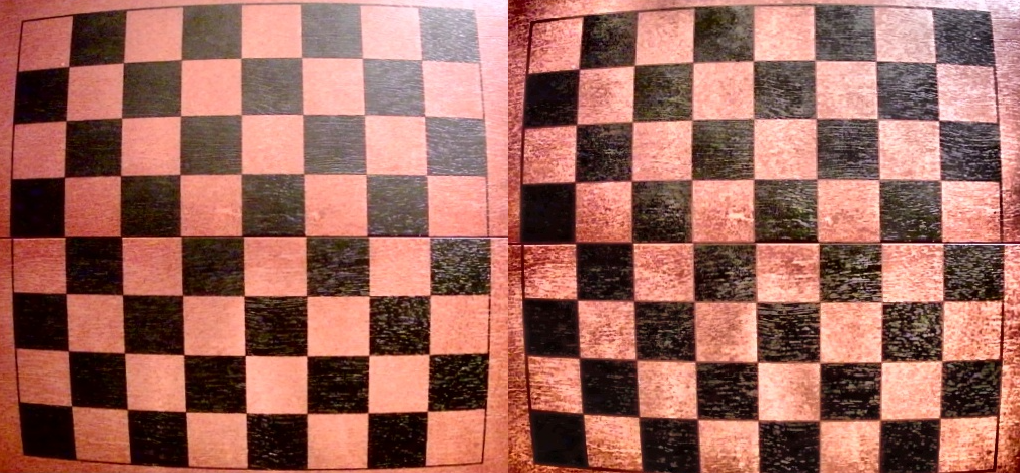
Zawartość wygenerowana przez AI może być niepoprawna.

**Figure 9** – the CLAHE algorithm

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.

The procedure is based on computing the histogram equalization on the smaller parts of the image, but the pixels that exceeds the clip limit are uniformly distributed among the rest of the bins of the histogram.

The comparison between the images before and after performing CLAHE are shown in the figure 10.



**Figure 10** – before and after the CLAHE 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.

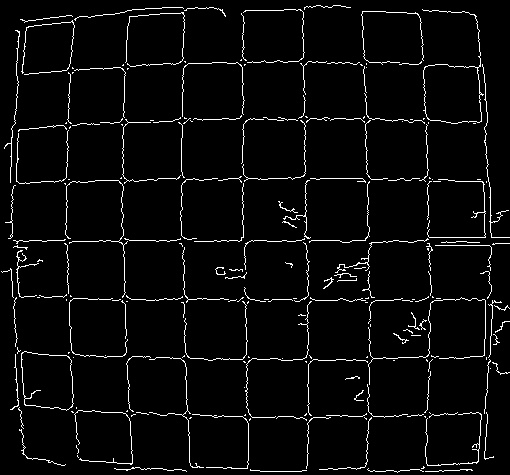
**Figure 11** – smoothing and Canny Edge Detection

Then, we performed the Canny Edge Detection method. The result is a binary image called edges, which contains only the detected edges.



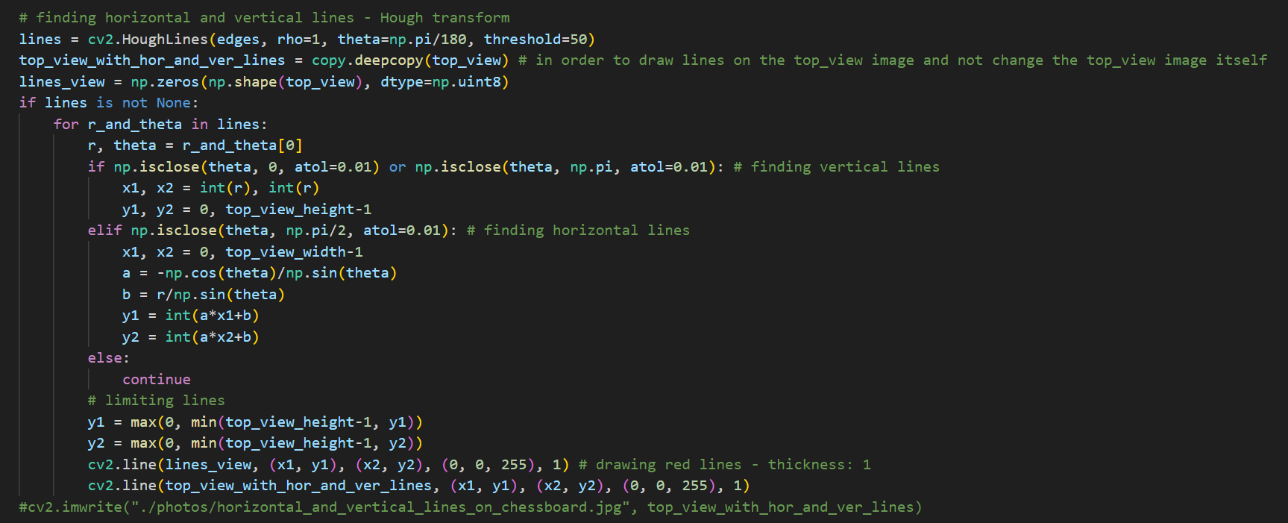
**Figure 12** – the image after smoothing

The image from the figure 12 was then used to detect edges using Canny.



**Figure 13** – the result of 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 (figure 14). The lines were drawn on the original image from the camera (figure 15).



**Figure 14** – the Hough transform

As you can see in the code. We only seek lines that are approximately horizontal or vertical.

Obraz zawierający wzór, kwadrat, Prostokąt, kafelek

Zawartość wygenerowana przez AI może być niepoprawna.

**Figure 15** – the 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 image part. 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 16** – finding intersections

This part of the code (figure 16) 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. To identify the intersection, we used the *red plus* pattern. Also, due to multiple horizontal/vertical lines being drawn close to each other, we had to set the radius for each detected intersection, so that new detection wouldn’t be found near the already marked corners.

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.**

**Figure 17** – naming the tiles

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

**Figure 18** – extracting tiles from the image

The above part of the program stores each square's name and its view in a dictionary. In the following steps, these images 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 19** – scanning each tile and preparing for the 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 20** – the example of the scanned tile

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

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

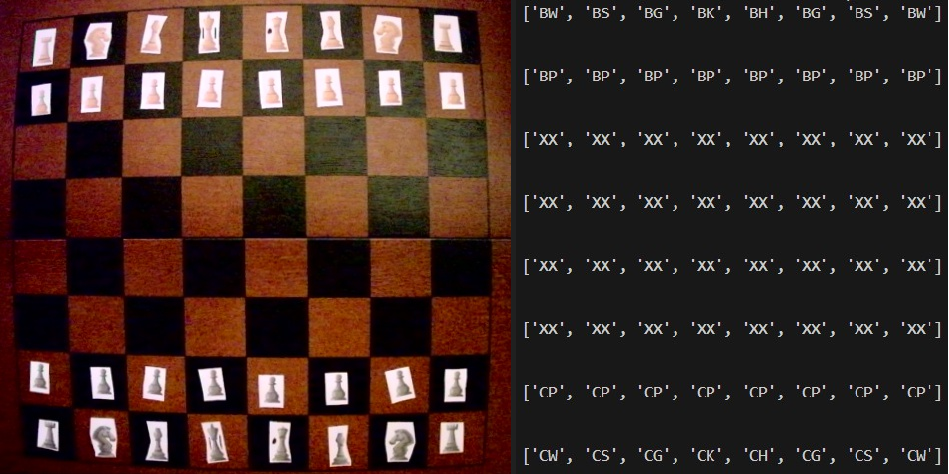
Zawartość wygenerowana przez AI może być niepoprawna.

**Figure 21** – using CNN and saving the chessboard configuration in the matrix

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 the text output in the console. The prepared results are shown in the figures 22-24.

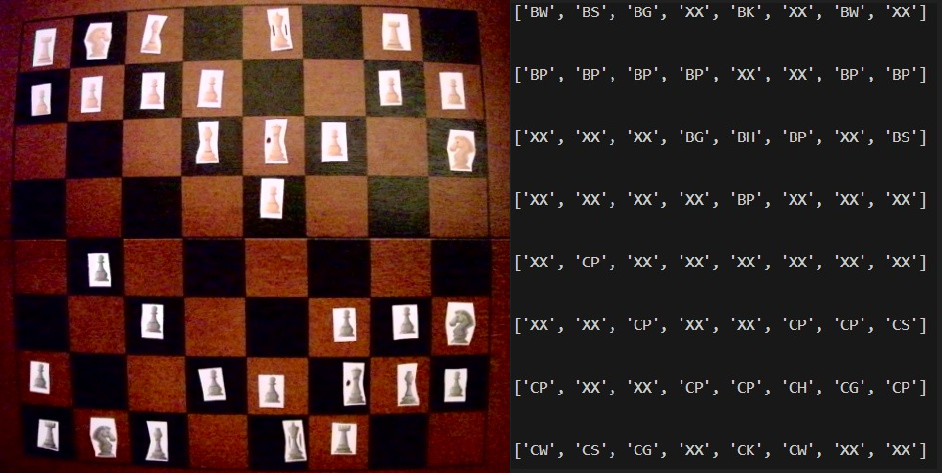


**Figure 22** – the representation of the empty board



**Figure 23** – the representation of the starting position

Each piece has its own name. (e.g. BW – *Biała Wieża,* CS – *Czarny Skoczek,* XX – empty tile).



**Figure 24** – the representation of the mid game

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

We attach the whole program to the report.

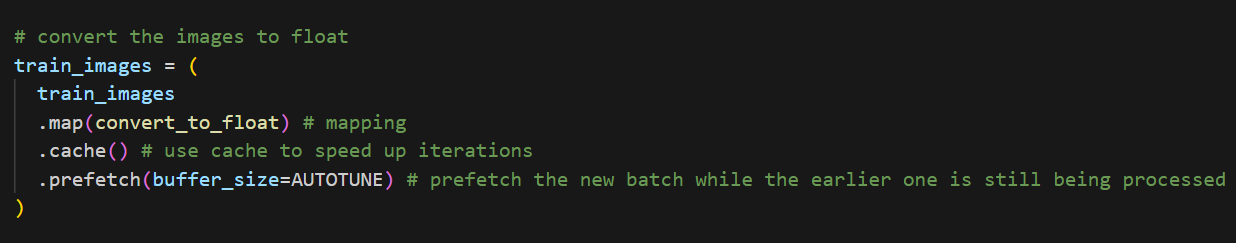
# The CNN Model

This chapter describes the stages of preparing, training, and testing the 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, that was later used in the main program.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

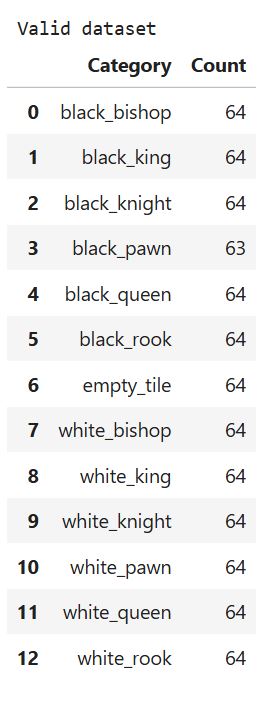
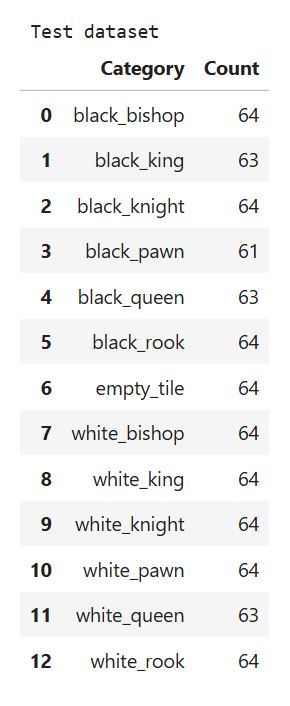
**Figure 25** – loading the data to the notebook (categorical labelling) – train data



**Figure 26** – converting the matrices to float – train data

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 (figures 25, 26). Data is split in proportion:

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



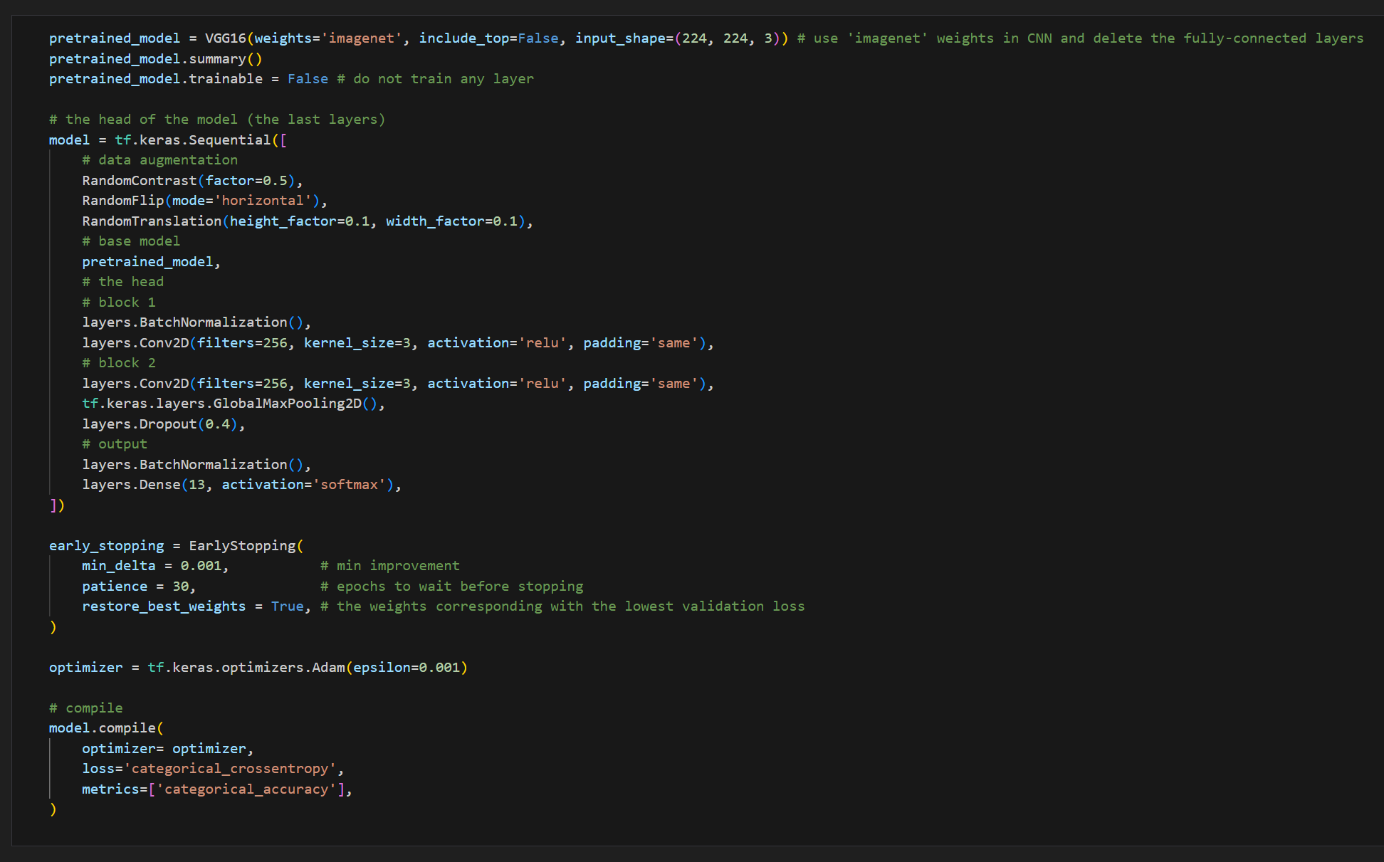
**Figure 27** – the division of the data between the sets

Additionally empty space is also classified in those sets as *empty\_tile.*

Next, the process described in the article is followed. Transfer learning (using the already trained CNN) 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 the 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 the categorical cross-entropy loss, suitable for the 13-class classification task. Early stopping is implemented to halt the training if validation loss does not improve after 30 epochs, ensuring the model does not overfit and the best weights are restored.



**Figure 28** – the CNN model

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 29** – training the CNN

Obraz zawierający tekst, zrzut ekranu, linia, diagram

Zawartość wygenerowana przez AI może być niepoprawna.

**Figure 30** – loss through epochs

Obraz zawierający tekst, zrzut ekranu, linia, diagram

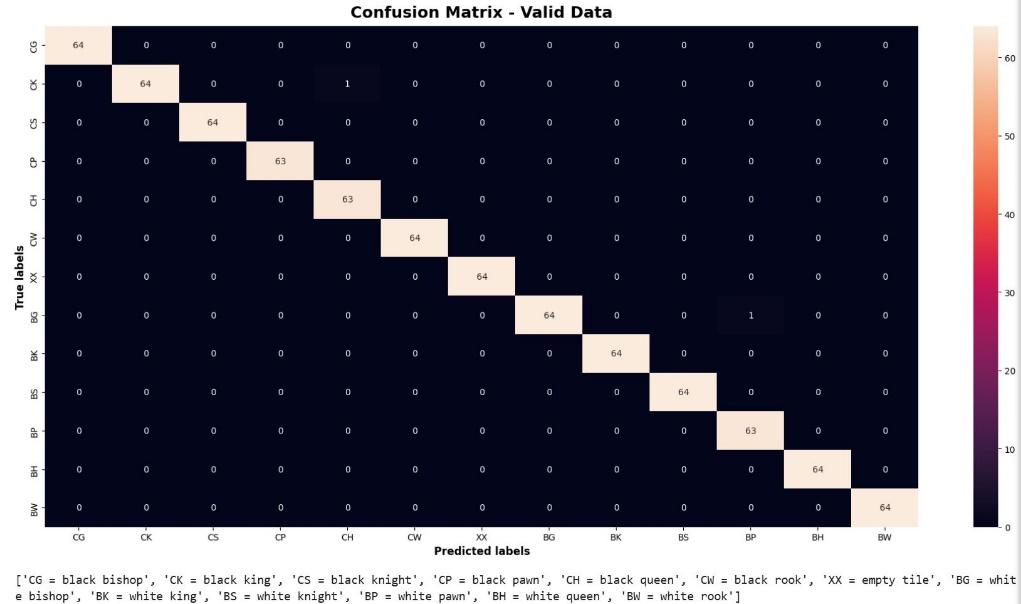
Zawartość wygenerowana przez AI może być niepoprawna.

**Figure 31** – accuracy through epochs

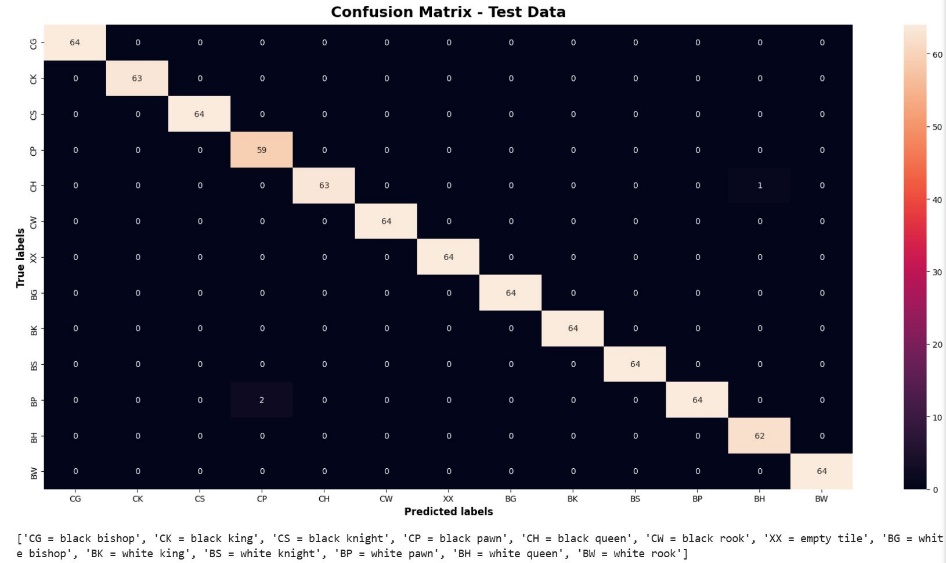
Obraz zawierający tekst, zrzut ekranu, kwadrat, Prostokąt

Zawartość wygenerowana przez AI może być niepoprawna.

**Figure 32** – the confusion matrix – train data



**Figure 33** – the confusion matrix – validate data



**Figure 34** – the confusion matrix – test data

The total accuracy of the prepared CNN 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.

We also attach the notebook that we used for the CNN training to the report.

# Conclusions

* The system successfully detects and classifies chess pieces using image processing and the 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)
* the CLAHE algorithm: (https://www.youtube.com/watch?v=tn2kmbUVK50&t=204s&ab\_channel=KevinWood%7CRobotics%26AI) – YouTube video
* 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 was used to interpret new algorithms and to help building the project.
* OpenCV and TensorFlow documentation
* the lectures