IKT213 - Machine Vision

ACRONYM - Final Release

Advanced **C**hessboard **R**ecognition and **O**ptical **N**otation **Y**ielding **M**echanism

---

Autumn 2023

Robin Meier, Robin Schwarz, Florian Walter

November 20, 2023

Mandatory Group Declaration

Each student is solely responsible for familiarizing themselves with the legal aids, guidelines for their use, and rules regarding source usage. The declaration aims to raise awareness among students of their responsibilities and the consequences of cheating. Lack of declaration does not exempt students from their responsibilities.

1.	We hereby declare that our submission is our own work and that we have not used other sources or received any help other than what is mentioned in the submission.	Yes
2.	**We further declare that this submission:** • Has not been used for any other examination at another department/university/-college domestically or abroad. • Does not reference others' work without it being indicated. • Does not reference our own previous work without it being indicated. • Has all references included in the bibliography. • Is not a copy, duplicate, or transcription of others' work or submission.	Yes
3.	We are aware that violations of the above are considered to be cheating and can result in cancellation of the examination and exclusion from universities and colleges in Norway, according to the Universities and Colleges Act, sections 4-7 and 4-8 and the Examination Regulation, sections 31.	Yes
4.	We are aware that all submitted assignments may be subjected to plagiarism checks.	Yes
5.	We are aware that the University of Agder will handle all cases where there is suspicion of cheating according to the university's guidelines for handling cheating cases.	Yes
6.	We have familiarized ourselves with the rules and guidelines for using sources and references on the library's website.	Yes
7.	We have in the majority agreed that the effort within the group is notably different and therefore wish to be evaluated individually. Ordinarily, all participants in the project are evaluated collectively.	No

Publishing Agreement

Authorization for Electronic Publication of Work The author(s) hold the copyright to the work. This means, among other things, the exclusive right to make the work available to the public (Copyright Act. §2).

Theses that are exempt from public access or confidential will not be published.

We hereby grant the University of Agder a royalty-free right to make the work available for electronic publication:	Yes
Is the work confidential?	No
Is the work exempt from public access?	No

Contents

1 Project Idea

The idea behind the Advanced Chessboard Recognition and Optical Notation Yielding Mechanism (or short ACRONYM) is to convert an image of an over-the-board chess game into a digital representation. Users should be able to take a picture on their phone which then gets evaluated and converted into then Forsyth–Edwards chess notation system (FEN).

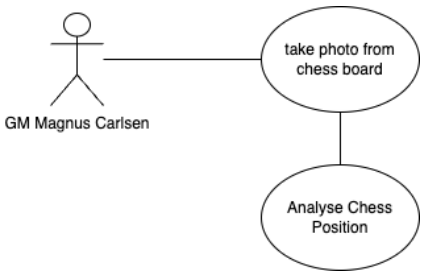


Figure 1: Use Case Diagram

1.1 Project Goal

For this project multiple goals were defined during the planning phase. The system does not have to be near-real-time, but a maximum inference time of ten seconds is aimed at. Regarding accuracy, always correct results are sadly not realistic, so the goal of 98.4% is set, which corresponds to one mistake every 64 identified squares. Another, supplemental goal, is to implement the image processing pipeline if possible without any machine learning algorithms to identify the pieces.

2 Project Background

The task of chessboard piece recognition is a far spread challenge and has been attempted by numerous people. The ways, in which the recognition is made varies, but similar techniques were observed. Multiple approaches first detect the chessboard and cut it out for further analysis [1], [2], [3], while Czyzewski et al. [2] and Saurab [3] use a heatmap based approach, Nisan [1] uses a YOLOv8 model to detect chessboard corners. Nelson [4] skips detecting the board completely and directly trains a YOLOv3 model to identify the pieces in the image. Another similarity in previous work is the use of the Hough Line Transform to detect the chessboard and locate separate squares on the board [5] and merging similar lines into one [2]. Most approaches use machine learning models to identify pieces, with Nelson [4] and Nisan [1] using YOLO over the full image and Ding [6] and Czyzewski et al. [2] using SVMs on cutouts of the pieces. Czyzewski et al. additionally use the Stockfish engine to evaluate the probability of the predicted position to actually exist and take the physical properties of the pieces into account. Orémuš [5] evaluates two variants for recognizing pieces. Variant A relying on template matching the contour of the pieces and variant B using the grayscale intensity distribution of the piece to differentiate them. Our implementation uses this previous work as inspiration in different parts of the pipeline. The Houg Line Transform is used to identify different squares and the grayscale intensity distribution in the piece recognition step. Also recognition is done on cutouts of pieces instead of the full board. Different from previous similar work however, we do not use SVM but a deep convolutional network as described by Simonyan and Zisserman [7].

3 Proposed Solution

During the 2023 fall semester course "IKT213 - Machine Vision" at the University of Agder in Grimstad, a working prototype should be designed and implemented which follows the general idea described above.

3.1 Algorithm

To achieve the previously introduced project idea, an image processing pipeline will be implemented in Python using OpenCV. The pipeline consists of the following steps:

1. Detect the boundaries of the board
2. Transform the perspective to isolate the board itself
3. Detect the squares on the board
4. Identify pieces on the squares
5. Convert the internal data structure to FEN notation

For the course, the application should only be able to identify pieces on a specific chessboard and figure set. More boards may be introduced later, but are not part of the release. Additionally, to reduce the workload required for board rotation and piece identification, from a specific angle the image has to be taken from is enforced.

3.2 Architecture

As already mentioned in section 1, the user shall be able to analyse an image by taking a picture of the board with their phone. For this purpose, an app will be created, using the cross-platform framework Flutter [8]. When an image is taken, it will then be sent to a Python Flask server, where it will be analysed using OpenCV and the notation will be sent back to the app, where it is displayed to the user.

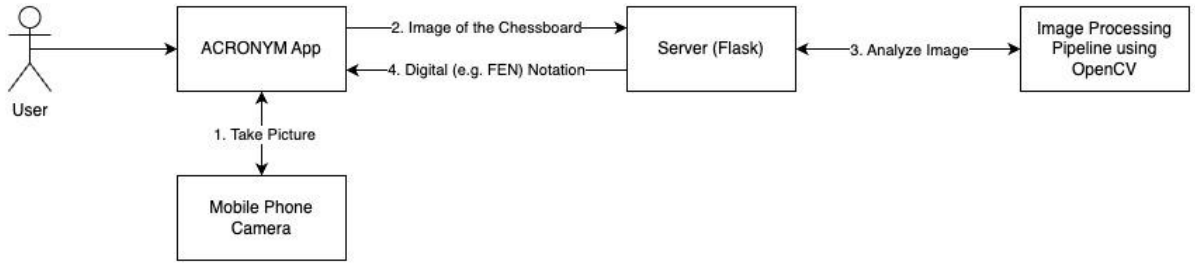


Figure 2: System Architecture and Interaction Diagram

4 Data collection

4.1 Dataset game_1

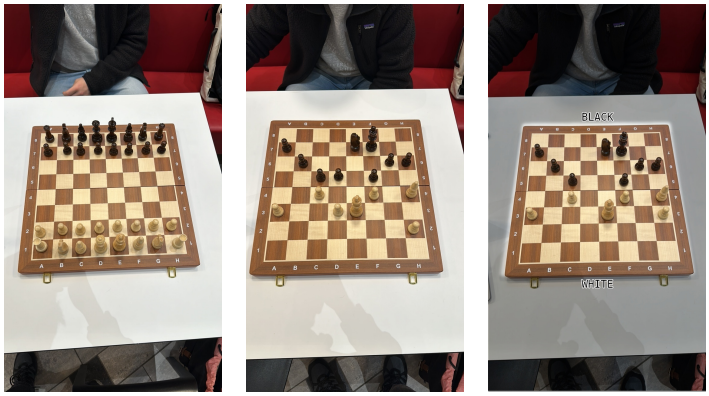
To test accuracy of the recognition mechanism and to tune hyperparameters, a dataset of different chess positions was created. The dataset was made using a beta version of the mobile application, with fixed camera resolution to 720p. A first collection of 90 images was created during a game of chess, with focus on using different camera angles for each image.

(a) Starting Position (b) Move 20 (c) Move 72

Figure 3: Example images from game_1 dataset

4.2 Dataset game_2

Due to difficulties recognising chess board corners on images with varying perspectives, a fixed perspective for image capturing was defined. To achieve a similar perspective for every image, an overlay was implemented in the mobile application and a second dataset of 99 images with corresponding FEN notation and fixed perspective was created during a second game of chess.



(a) Starting Position (b) Move 65 (c) E.g. App Overlay

Figure 4: Example images from game_2 dataset

4.3 Dataset random

Dataset game_2 had some shortcomings in terms of piece position variety, like rooks and queens rarely leaving their starting squares. For the CNN to learn more varied positions of pieces, a third dataset was created. This dataset still uses the fixed angle of game_2, but doesn't cover a game anymore. Instead 20 random positions using all available pieces were generated using an online tool [9], which returned random positions with corresponding FEN notations, which then were setup on the board.

4.4 Data Preprocessing for CNN

From game_2 dataset a collection of 6336 images and from the random dataset 1280 images of individual pieces and empty squares were generated. The pictures were generated using the square corners found in the image (Section 5.2) from which the images for every square on the board was cropped to a specific width and height. The width and height are dependent on the center point of the square and the rank on which the piece is located. After cropping the individual squares, the images were sorted by type and colour and stored in a database for training the CNN model.

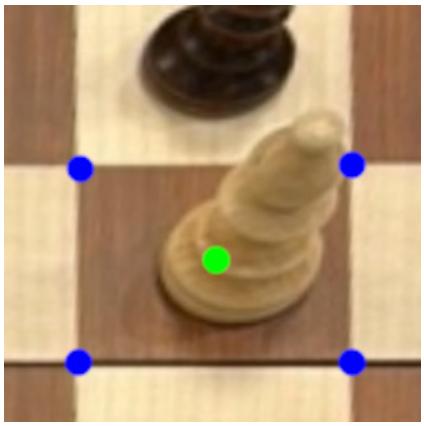


Figure 5: Cutout for CNN

In figure 5 the blue points represent the corners of the square. The following calculations are based on the coordinates of the top left point (x, y) and the width and height of the square (w, h).

Calculating the midpoint (figure 5, green point):

$$m = \left(x + \frac{w}{2}, y + \frac{h}{2}\right) \tag{1}$$

Based on the midpoint, the cutout rectangle's points are calculated as followed:

$$P_{top\_left\_x} = m_x - s_w * \frac{w}{2} \tag{2}$$

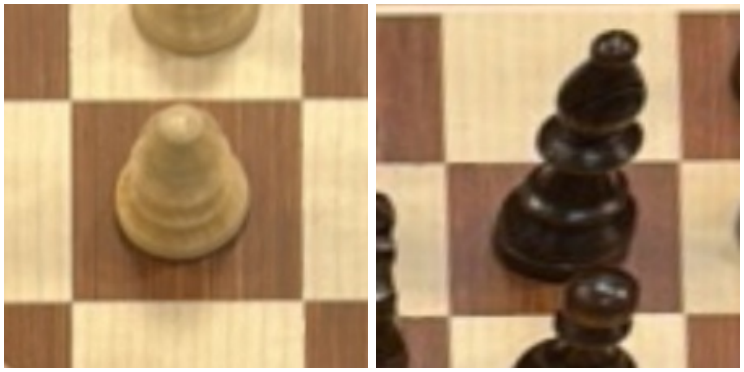
$$P_{top\_left\_y} = m_y - \left(1 + \frac{r-1}{7}\right) * s_h * \frac{h}{2} \tag{3}$$

$$P_{bottom\_right\_x} = m_x + s_w * \frac{w}{2} \tag{4}$$

$$P_{bottom\_right\_y} = m_y + s_h * \frac{h}{2} \tag{5}$$

with the following parameter: height scale factor $s_h = 5/3$, width scale factor $s_w = 3/2$ and chess board rank r.

Depending on the rank the piece is standing on, the cutout of the piece must be a different height (see figure 6). This is due to the perspective transform of the board. Pieces closer to the camera appear more top down after the transformation and therefore need less area in the cropped image than pieces further away from the camera. If the height of the image would not be scaled based on the rank, the crop would either cut of the top of pieces further away or potentially include more than one full piece on crops of pieces closer to the camera. As seen in equation (3), the rank of the piece is taken into account and according to r the distance from the center point to the top left corner is scales by a factor between 1 and 2.



(a) Bishop on 2nd rank (b) Bishop on 7th rank

Figure 6: Height difference of pieces on different ranks

As a final step, because the crops of the pieces are varying in size, the images are resized to 224x244 pixels to then be processed and identified by the CNN.

5 Image Processing Pipeline

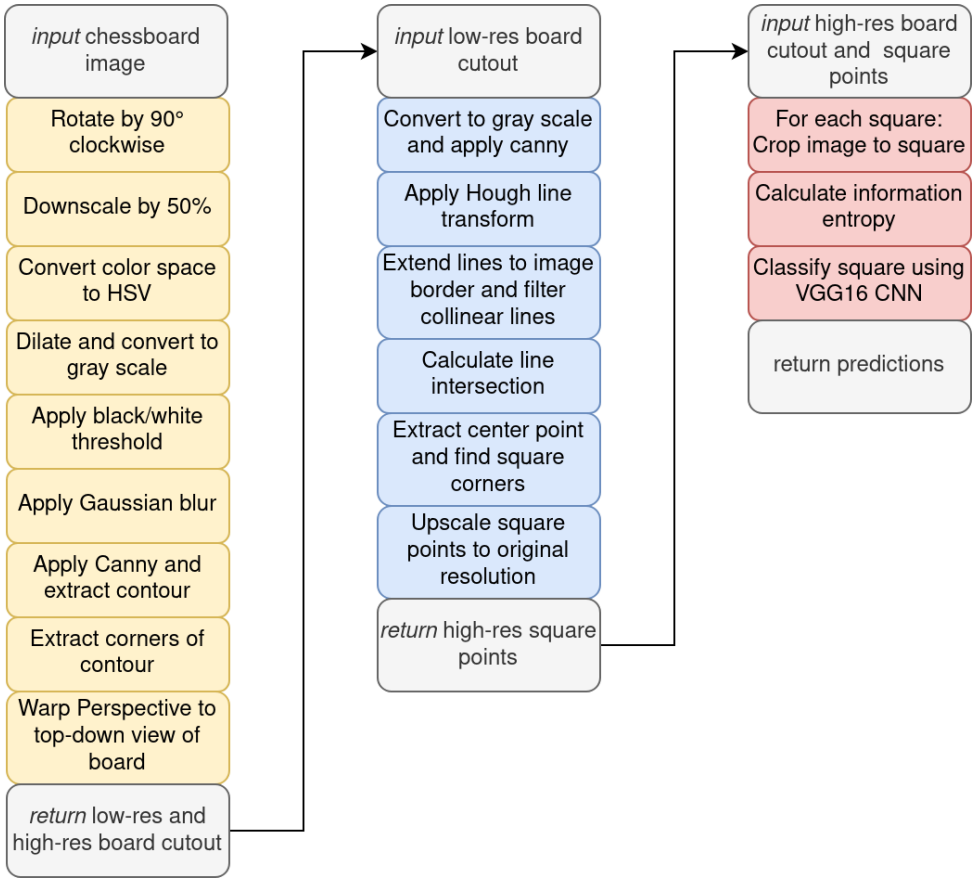
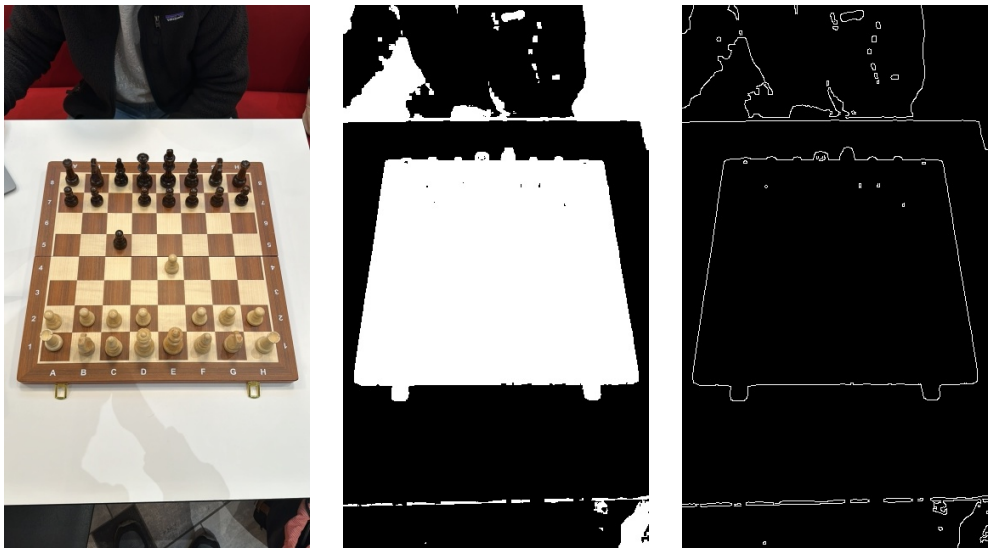


Figure 7: Overview of image processing pipeline

Figure 7 gives an overview of the whole image processing pipeline. The pipeline can be split up into three parts: Pre-processing the image and cutting out the board (yellow), extracting the corner information of the individual squares (blue) and classifying the pieces (red). The following sections describe the sub-pipelines in more detail.

5.1 Board Cutout

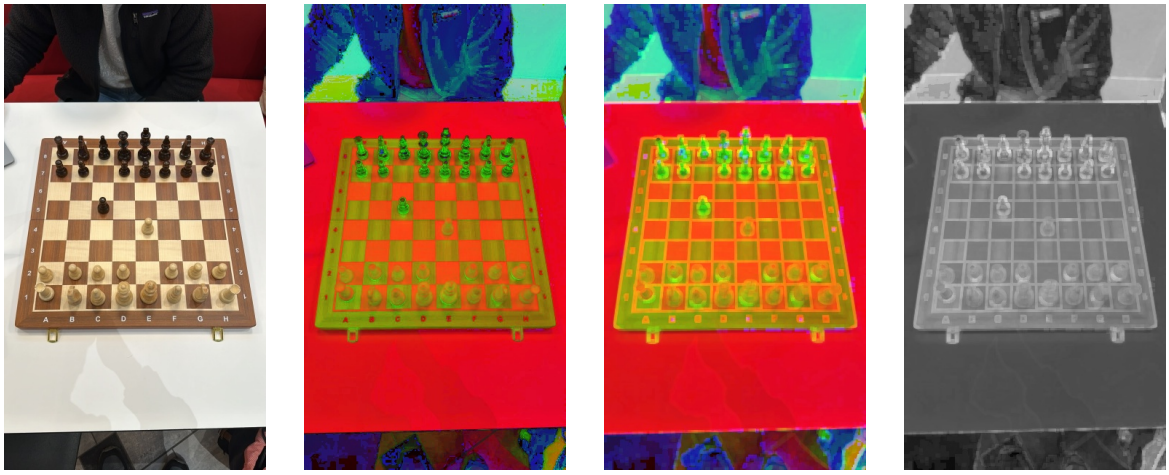
The first step to detecting a chessboard position is to know where the chess board is located in the image. To achieve this, the image is first scaled down to half it's original size (as the image size is fixed through the app, there is no need to dynamically downscale the image) and then grey scaled and blurred. To isolate the board in the image, a threshold is applied to the greyscale image. From there, the edges can easily be detected using a canny edge detector.



(a) input image (b) thresholding (c) edge detection

Figure 8: Steps to detect edges of the chessboard

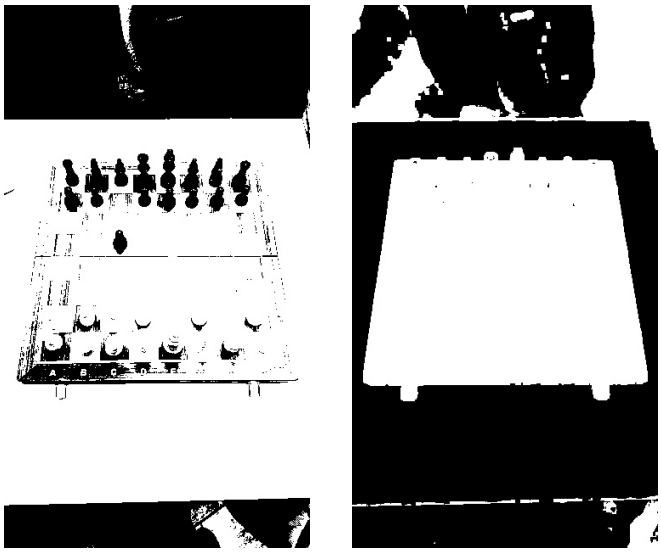
On most images, the background of the table was not very distinguishable from the white squares, which resulted in the threshold selecting the entire table instead of just the chessboard. To prevent this, an additional step has been inserted in the pipeline. Instead of converting the image directly into greyscale, it is first converted into HSV colour space to retain the colour information instead of just the intensity. Afterwards, the image is dilated so that the black squares "bleed" into the white squares to make them darker and more distinguishable from the table (see figure 9).



(a) input image (b) HSV color space (c) dilation (d) grey scaled

Figure 9: HSV dilation process

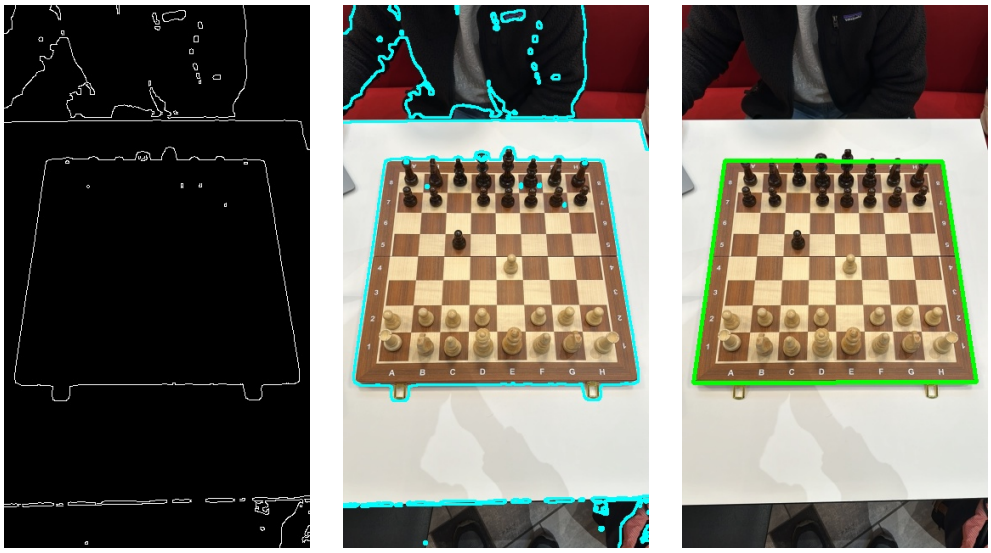
After doing this, the image can then be grey scaled and a threshold applied to extract a mask for the chessboard. The difference the dilation process causes can be seen in figure 10.



(a) without HSV dilation (b) with HSV dilation

Figure 10: Comparison of the impact of HSV dilation on the thresholding step

Once the edges have been identified, the `cv2.findContours` function is used to extract the continuous contours of all the edges. As the chessboard is expected to be the main thing on the image, the contours are then sorted based on the area they take up in the image and the largest one of these is selected to be the chessboard contour. Finding the corners from here can be achieved using the Ramer-Douglas-Peucker algorithm (e.g. using `cv2.approxPolyDP`), which combines points within a certain threshold into a single line and in the end is left with only the most significant corner points.

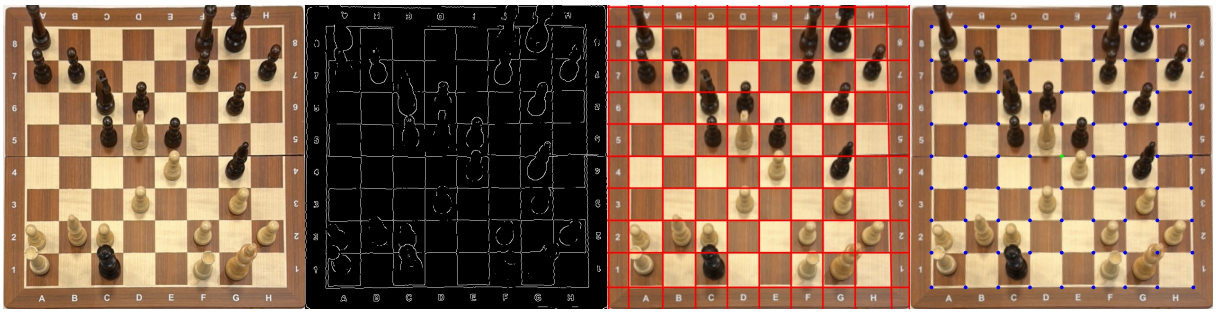


(a) edges (b) contours (c) approximated polygon

Figure 11: Steps from edge detection to board corners

This way of finding the chessboard has a drawback though, as the chessboard has to be entirely within the picture. When a corner is slightly cut off, the contour might not be continuous any more and would fail in the next step.

5.2 Square Detection



(a) warped perspective (b) canny (c) detected lines (d) detected squares

Figure 12: Steps to detect squares from board image

Find Lines of Squares

After detecting the corners of the chessboard, the OpenCV-function `cv2.getPerspectiveTransform()` returns the transformation matrix with which the function `cv2.warpPerspective()` warps the 4 corners of the board to a square image with resolution of 640 × 640 pixel (from figure 11c to figure 12a) [10]. A Canny-Edge-Detection algorithm (`cv2.canny()`) is then used to find the lines of the board's squares (see figure 12b) and with the function `cv2.HoughLinesP()` lines, which are longer than a certain threshold (in this case 50 pixel) are returned as coordinates of start and endpoint of each line.

Extend Lines

With the following algorithm, the lines found in the image are extended to the edge of the image. This is important for calculating the intersections later. In a first step the algorithm calculates a polynomial 1st order's factors a_0 and a_1 from the start and endpoint of a line:

$$a_1 = \frac{y_1 - y_0}{x_1 - x_0}, \quad a_0 = y_1 - a_1 * x_1 \tag{6}$$

Based on these factors the start and endpoint of the extended lines are determined:

$$\text{Linepoints} = \begin{cases} P_{start} = (x_0, 0), P_{end} = (x_0, 640) & \text{, if } a_1 = \pm\infty \\ P_{start} = (0, a_0), P_{end} = (640, a_1 * 640 + a_0) & \text{, if } 0 \leq a_0 \leq 640 \\ P_{start} = \left(\frac{-a_0}{a_1}, 0\right), P_{end} = \left(\frac{640 - a_0}{a_1}, 640\right) & \text{, otherwise} \end{cases} \tag{7}$$

To avoid the chance of having multiple lines for the same square line, collinear lines are removed using a threshold (in this case within 35 pixel) that removes all but one line within this boundaries (see figure 12c).

Find Intersections

With the calculated lines it is possible to calculate the intersections and therefore the square corners of the chess board. For calculating the intersections, the following factors have to be calculated in advance:

$$h_1 = \frac{H_{end\,y} - H_{start\,y}}{H_{end\,x} - H_{start\,x}}, \quad h_0 = H_{end\,y} - h_1 * H_{end\,x} \tag{8}$$

$$v_1 = \frac{V_{end\,y} - V_{start\,y}}{V_{end\,x} - V_{start\,x}}, \quad v_0 = V_{end\,y} - v_1 * V_{end\,x} \tag{9}$$

With the factors calculated, the intersections can be found:

$$\text{Intersection} = \begin{cases} P = (V_{start\,x}, h_1 * V_{start\,x} + h_0) & \text{, if } v_1 = \pm\infty \\ P = (\frac{h_0 - v_0}{v_1 - h_1}, h_1 * \frac{h_0 - v_0}{v_1 - h_1} + h_0) & \text{, otherwise} \end{cases} \tag{10}$$

If one of the points is on the image's boarder, it is automatically discarded from, because it is not further needed.

Find Center-point

To identify the important points for the squares, the center-point of the chess board has to be determined. Due to the images square size, the midpoint is the point closest to the image's center. Therefore the center-point can be determined as followed (see green point in figure 26a):

$$M = \min(\|(320, 320) - P_i\|), P_i \in \text{Intersections} \tag{11}$$

Find Square Corner Points

After finding the center point, the 8 points on the center's horizontal line that are closest to the center point are identified using the distance between each point and the center-point (see figure 26a). For each of the 9 points found on the center horizontal line, the 8 points closest to the point and its vertical line are calculated using again the distance between each point and its center-line-point (see figure 26b). These 81 points are then stored in 2D-Array sorted from top left (figure 12d: square A8) to bottom right (figure 12d: square H1).

5.3 Piece Classification

Architecture

To classify the pieces, a convolutional neural network (CNN) using the VGG16 architecture is used. For the convolutional layers, the pretrained weights based on the ImageNet dataset are utilized. The output of the convolutional layers is flattened and fed into dense layers of size 4096 and 1072. Then a dropout layer is applied to reduce overfitting. The last layer contains 13 outputs, one for each piece type and one for an empty square.

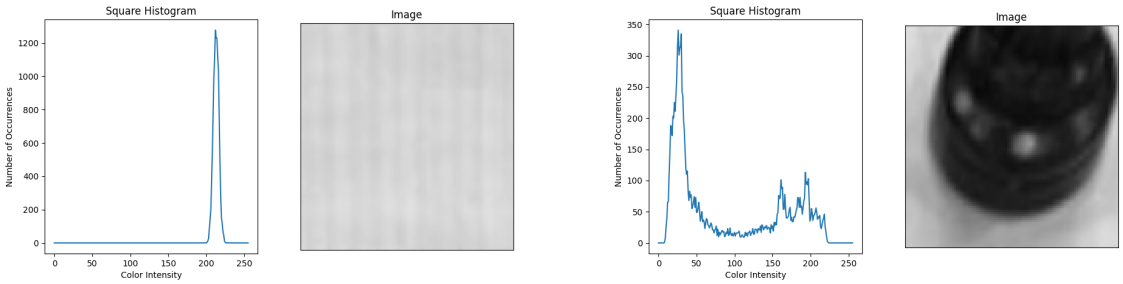
Training

During training the weights in the convolutional layers were locked and only the dense layers were updated. The data used, was split into training, validation and testing sub-datasets with the ratio 70%:15%:15%. To reduce the risk of overfitting, early stoppage was implemented on the validation loss, with a patience of 10 epochs.

Pipeline Operations

After extracting the square location data (5.2), each square is cut out individually (4.4), to be used for inference. The inference time varies between 200-600ms, which would lead to an overall run time of about 25 seconds to predict every square. To reduce the number of inference operations, the Shannon entropy, i.e. the average level of information for each squares grayscale intensity distribution calculated using function 12, with χ being the probability of each value. Figure 13 shows two examples of grayscale intensity distribution histograms and their calculated entropy. If the entropy is < 1.5, the square is marked as empty and not fed into the CNN (13a). If the entropy is $>= 1.5$ a prediction of the square is made (13b). After every square is processed, the predictions are returned.

$$H(X) := -\sum_{x \in \chi} p(x)\, log\, p(x), \quad p : \chi \to [0, 1] \tag{12}$$



(a) $H(X) = 0.907$ (b) $H(X) = 5.844$

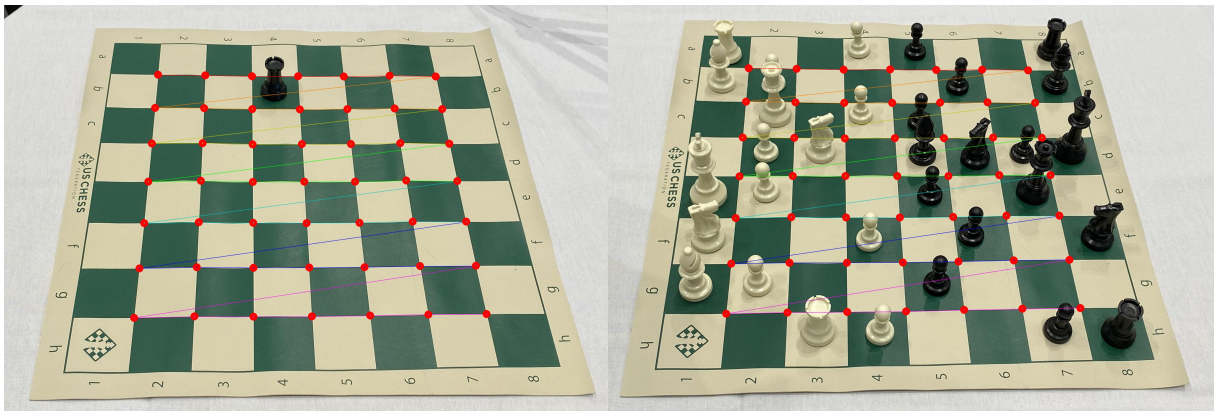
Figure 13: Histograms of grayscale squares

6 Results

During the course of the project, multiple different methods were experimented with. Following are the results of the different experiments and their success rates.

6.1 Chessboard detection using findChessboardCornersSB

An initial attempt was made, to detect the chessboard in an image using the built in function `cv2.findChessboardCornersSB()`. Per documentation, "The function attempts to determine whether the input image is a view of the chessboard pattern and locate the internal chessboard corners." [11]. This approach worked well on the Roboflow chess piece dataset [12], but whenever own images were used, no board was detected. Because the physical chessboard was already acquired, this approach was scrapped in favour of a custom detection pipeline.



(a) Detection on empty chessboard (b) Detection on full chessboard

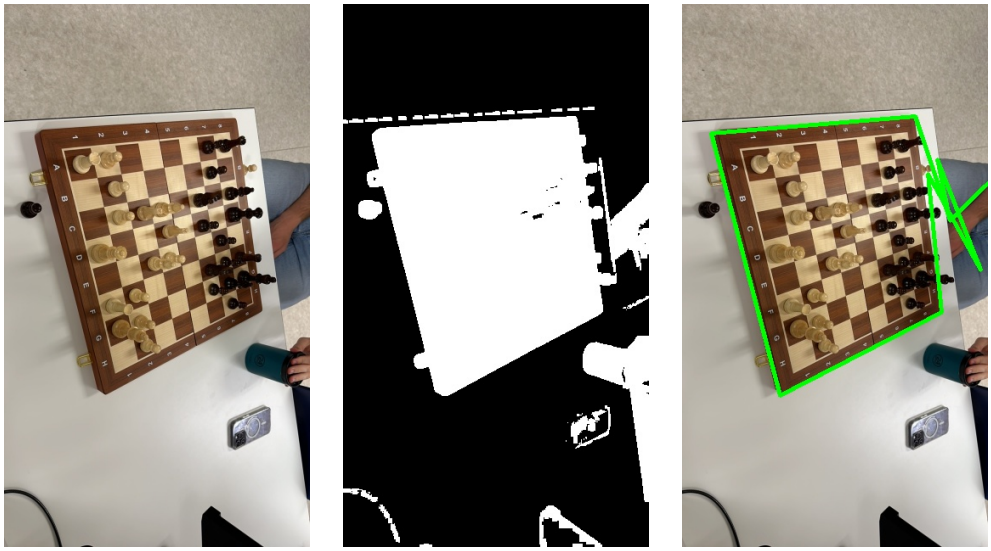
Figure 14: Results of findChessboardCornersSB on Roboflow dataset [12]

6.2 Chessboard detection using custom implementation

While the custom implementation only achieves to detect the chessboard in 18% of the images from the first dataset, the second dataset can achieve an accuracy of 91%.

(a) input image (b) threshold (c) approximated polygon

Figure 15: Failure to detect chessboard in image of first dataset due to captured figures standing too close to the edge of the board



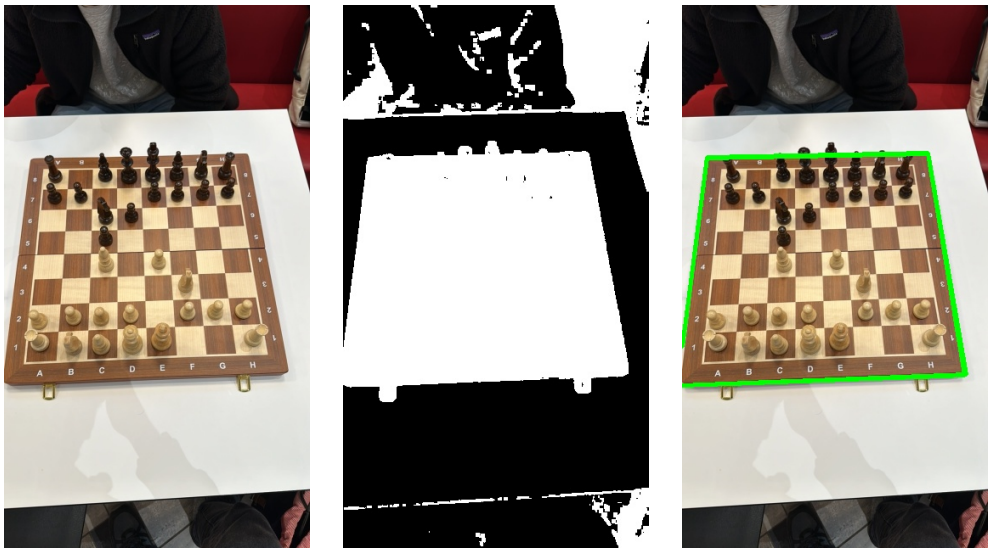
(a) input image (b) threshold (c) approximated polygon

Figure 16: Failure to detect chessboard in image of first dataset due to table corner too close to the chessboard

Most of the failures in the first dataset can be attributed to the captured white pieces standing too close to the edge of the board, which results in the threshold including them as a part of the board and therefore the approximated polygon not being a square in the end (as shown in figure 15). Other common failures originate from one of the corners not being in the image itself, or perspectives where the board was too close to the edge of the table and different parts of the image got included in the contour (such as figure 16).

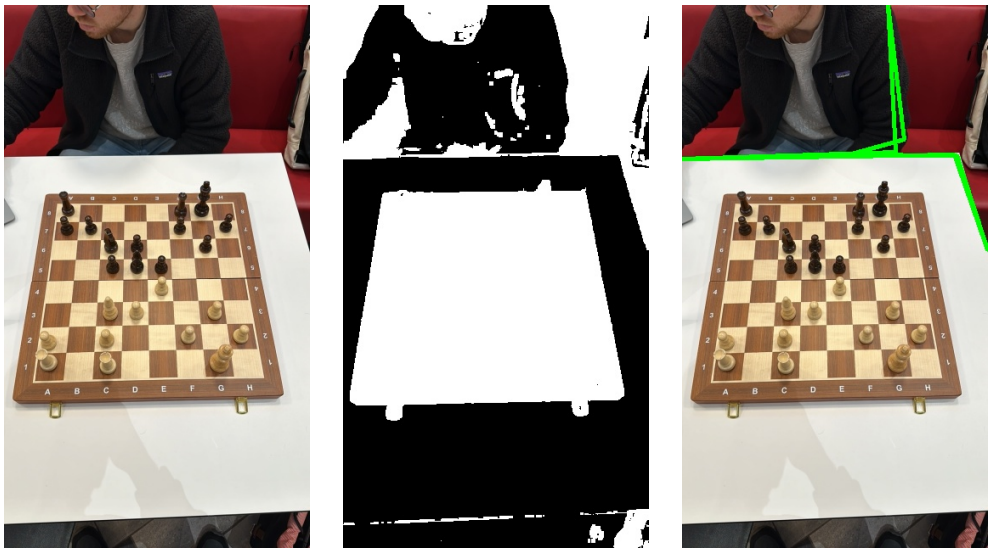
In the second dataset, the primary point of failure was the picture not containing one of the corners of the board (such as figure 17), which contributed to 7 out of the 10 failures in total. Without these human errors, the board detection would achieve an accuracy of

97% for this dataset. The other three failures consist of the image threshold containing some other larger contour in the image, as shown in figure 18.



(a) input image (b) threshold (c) approximated polygon

Figure 17: Failure to detect chessboard in image of second dataset due to cut off left bottom corner in input image



(a) input image (b) threshold (c) approximated polygon

Figure 18: Failure to detect chessboard in image of second dataset due to chessboard not being the largest contour in the image

6.3 Piece color identification using entropy and thresholds

While experimenting with the Shannon entropy, an attempt was made to use it for differentiating black and white pieces, instead of only using it to detect empty squares. Empty squares would be determined using the entropy and then using the average color values in the center of the square, the color would be determined. Using this approach no accuracy higher than 80% was achieved, as seen in figure 19, so this idea was dropped.

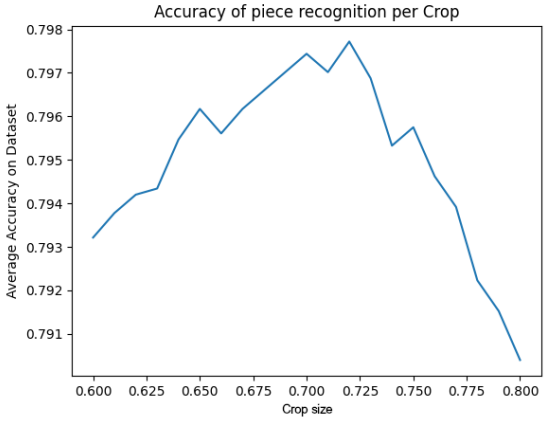


Figure 19: Accuracy of different amounts of cropping before calculating color average

6.4 Piece identification using SIFT

Initially, the idea has been to use scale-invariant feature transform (SIFT) to detect the pieces and a first prototype had been implemented. However, it became clear very quickly, that this approach would need a lot of tuning in order to be viable. While the detected key points (as shown in figure 20) seem quite reasonable, the found matches were just completely out of hand and did not have anything to do with the structure of the figure any more. Instead, the matches would just connect to some key points on the grain of the wooden chessboard, instead of actually matching the points on the piece, which can be seen in figure 21. Before spending a lot of time on trying to get SIFT to produce accurate results, it has been decided to check out neural networks first, which turned out to be much easier.

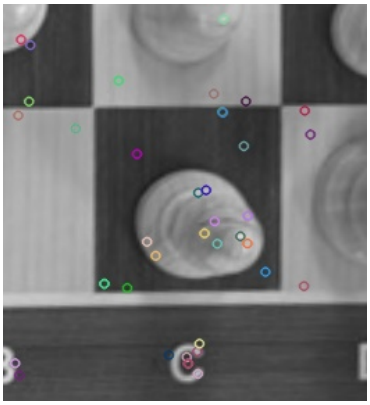
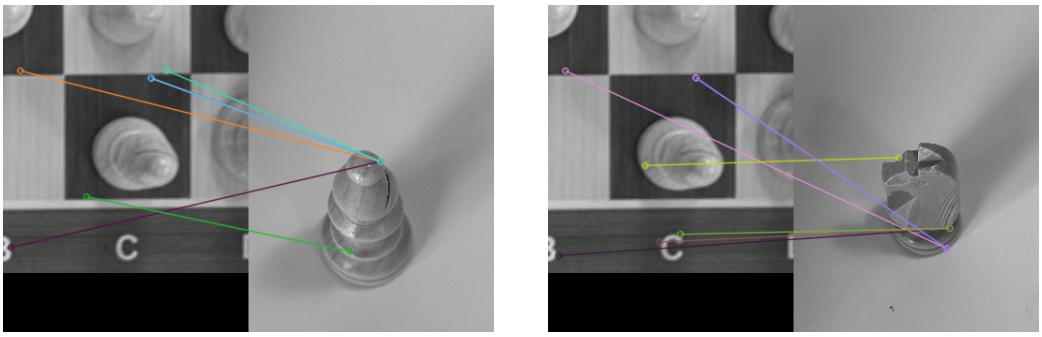


Figure 20: Identified key points using SIFT



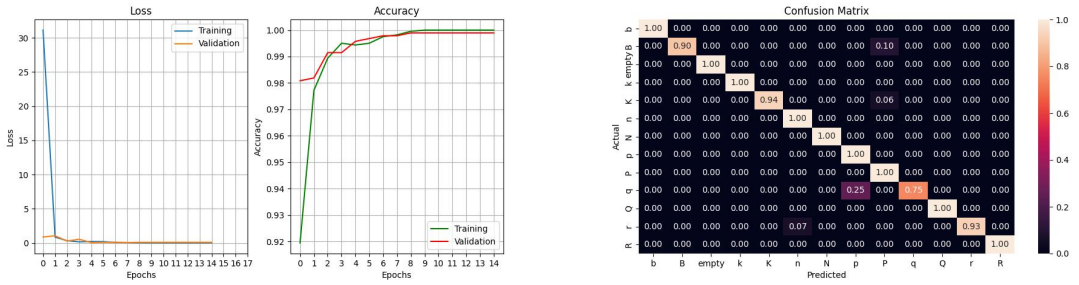
(a) best matches against bishop (b) best matches against knight

Figure 21: Best matches found using the key points from figure 20 using the SIFT prototype

6.5 Piece identification using CNN

Training using game_2

In a first cycle of training using the preprocessed dataset generated from game_02 (4.2), an accuracy of nearly 100% was reached after only about 10 epochs on the training and validation datasets (22a). Testing showed also very good scores with an average accuracy of 99%, only mislabeling white bishops 10% of the time, white king 6% of the time, black queens 25% of the time and black rooks 7% of the time (22b). Testing on random positions resulted in an accuracy of only 78.4%. This could be due to misrepresentations in the dataset. During the game portrayed in dataset 2, blacks rooks and queen rarely left the back rank. This led to there always being the bottom board edge in the image, which may have trained the model on the edge instead of the actual figures. Because of this the model after training cycle one, was very bad at predicting the black queen and rooks, if they where anywhere, but the last rank. Additionally the testing with real images showed, that kings and queens get often confused. This may be led back to their relative rareness in contrast to empty squares or pawns. (See Appendix A.1 for results)

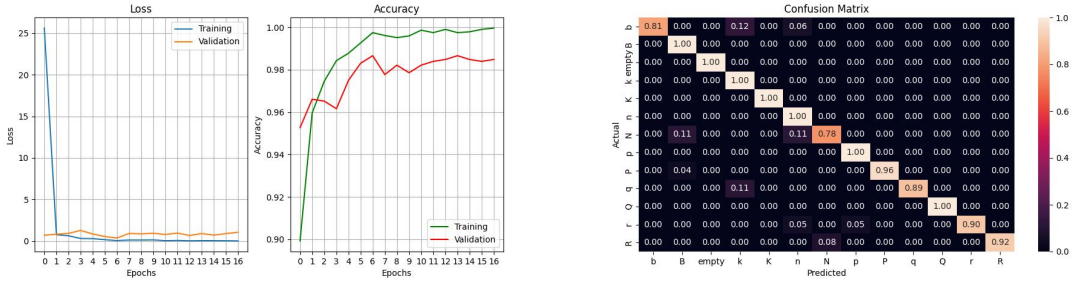


(a) Accuracy and training loss (b) Confusing Matrix

Figure 22: Results of training CNN classifier

Training with addition of random positions

To try and alleviate the above mentioned issues, the dataset described in section 4.3 was appended to the game_2 dataset and training was run again. This led to worse performance on the validation set and testing accuracy stayed the same at 99%. Testing the model on new random positions led to worse accuracy than the weights, only trained on game _2, possibly due to overfitting. (See Appendix A.1 for results) Out of this reason, this experiment was scrapped in favour of the game_2 weights.



(a) Accuracy and training loss

(b) Confusing Matrix

Figure 23: Results of training CNN classifier with additional random positions

7 Conclusion & Challenges

During implementation, one big challenge quickly emerged: Consistency. How can the corners of the board and positions consistently be detected and figures correctly be identified, when images may be taken from different angles and image quality and aspect ratio could vary? To reduce variables, the decision was made to fix camera resolution in the app to 720p, only support one specific type of chessboard and fix the angle the image has to be taken from using an overlay. This reduced complexity tremendously and made the identification of the board much more reliable. In general, the board cutout pipeline took a long time to get working consistently and turned out rather convoluted.

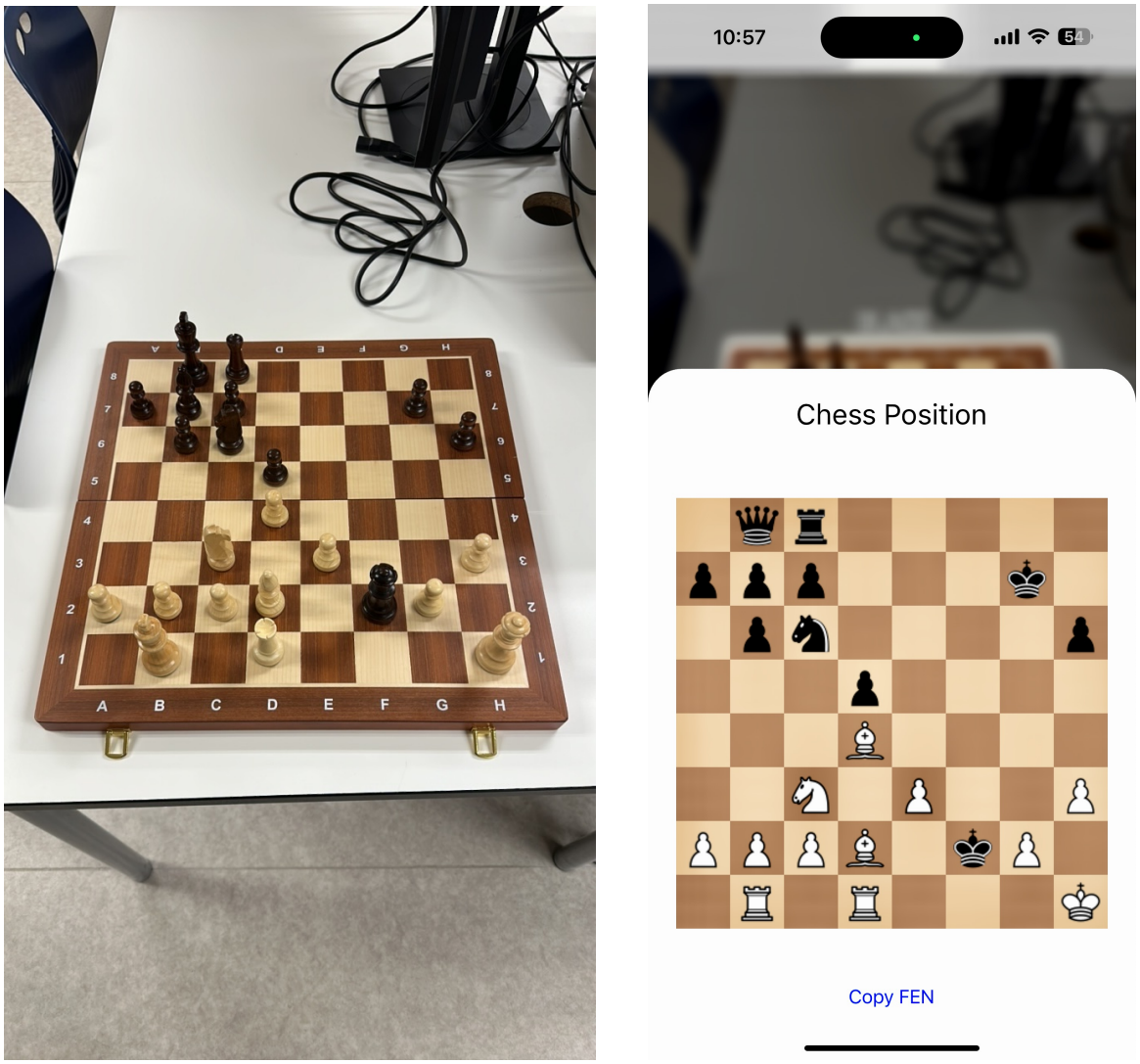
Previous work by Orémuš shows that implementing a working solution without using any machine learning is possible [5]. This, however, was not achievable for us in the given time frame, as fine-tuning SIFT and generating high value features, would have taken considerably more time. The switch to using the VGG16 convolutional network with the ImageNet weights, provided a fast and relatively reliable solution. The goal of reaching 98.4% accuracy was not reached, instead it was only possible to reach about 78% on completely random chess positions and about 90% on actually possible game states. Improvements could be made by tuning the model training process or implementing additional validation mechanisms like the piece position probability described in Czyzewski et al. [2]. Due to the usage of the large VGG16 model, the inference time also tremendously increased and on the server requires about 14 seconds to be evaluated. Locally, however, on a MacBook Pro with M1 Max chip, the total pipeline takes about 6 seconds.

Overall it can be said that the base implementation is solid, but on every step there are some improvements that can be made to further improve the accuracy and speed.

A Appendix

A.1 CNN Results

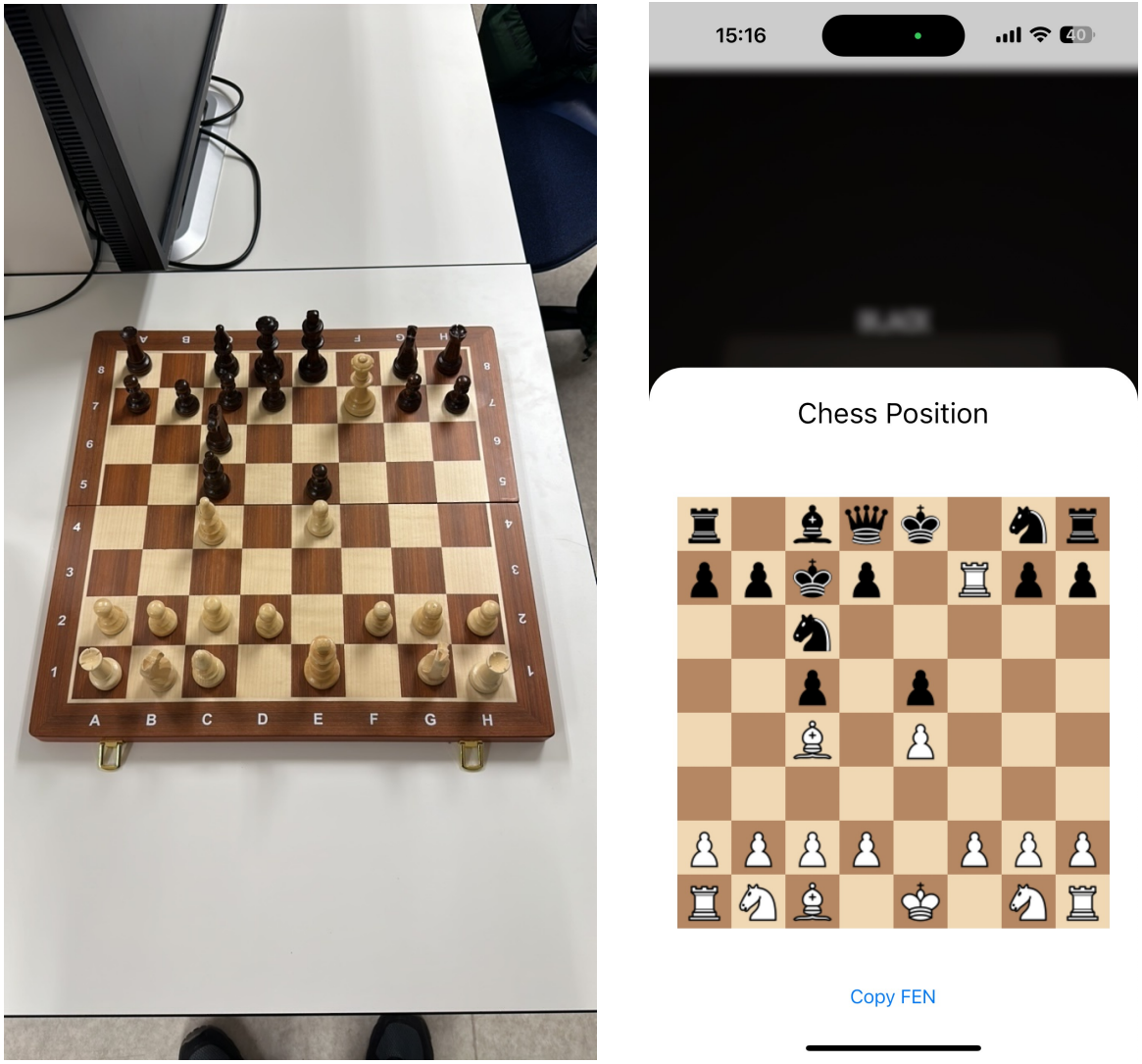
Training Cycle One



(a) Original board (b) Result

Figure 24: Results of analysing an image not in dataset

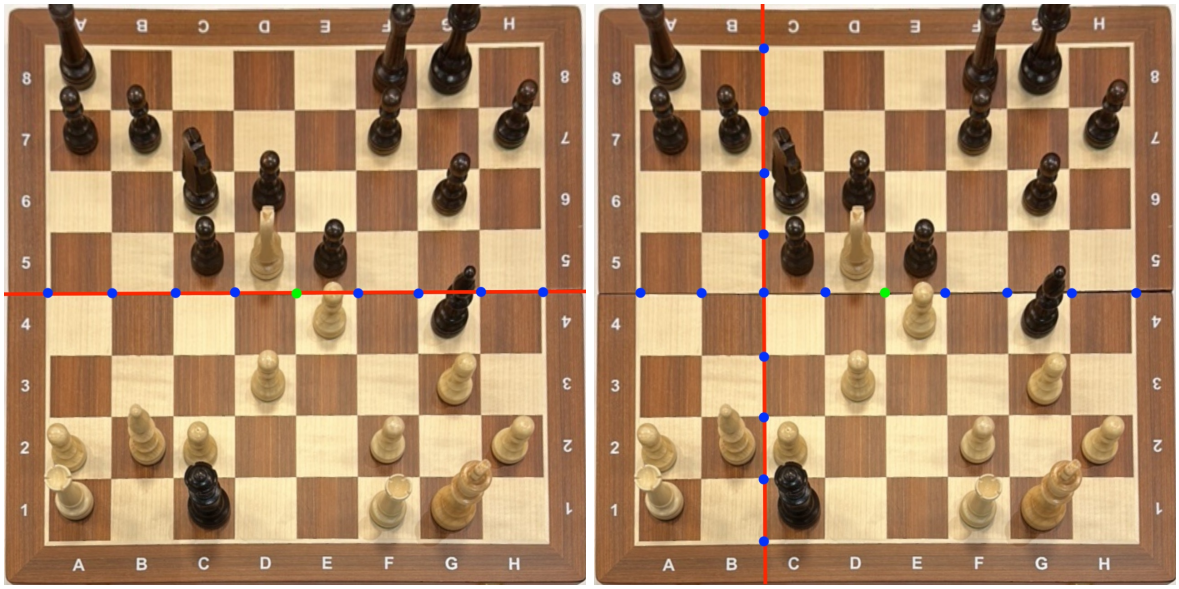
Training Cycle Two



(a) Original board (b) Result

Figure 25: Results of analysing an image not in dataset

A.2 Identify Square Corners from Intersections



(a) Points on center-horizontal Line (b) Vertical Points of Squares

Figure 26: Identify Square Corners from Intersections

References

[1] S. Nisan. "Represent chess boards digitally with computer vision," Roboflow Blog. (Mar. 10, 2023), [Online]. Available: https://blog.roboflow.com/chess-boards/ (visited on 11/13/2023).

[2] M. A. Czyzewski, A. Laskowski, and S. Wasik, *Chessboard and chess piece recognition with the support of neural networks*, Jun. 23, 2020. arXiv: 1708.03898[cs]. [Online]. Available: http://arxiv.org/abs/1708.03898 (visited on 11/13/2023).

[3] B. Saurabh. "Convert a physical chessboard into a digital one," Bakken & Baeck Tech. (Nov. 13, 2019), [Online]. Available: https://tech.bakkenbaeck.com/post/chessvision (visited on 11/13/2023).

[4] J. Nelson. "Training a YOLOv3 object detection model with custom dataset," Roboflow Blog. (Jan. 9, 2020), [Online]. Available: https://blog.roboflow.com/training-a-yolov3-object-detection-model-with-a-custom-dataset/ (visited on 11/13/2023).

[5] Z. Orémuš, "Chess position recognition from a photo," Ph.D. dissertation, Masarykova Univerzita, Fakulta Informatiky, Brno, 2018. [Online]. Available: https://is.muni.cz/th/meean/Master_Thesis.pdf (visited on 11/13/2023).

[6] J. Ding, "ChessVision: Chess board and piece recognition," Ph.D. dissertation, Stanford University, 2016. [Online]. Available: https://web.stanford.edu/class/cs231a/prev_projects_2016/CS_231A_Final_Report.pdf (visited on 11/13/2023).

[7] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, Apr. 10, 2015. DOI: 10.48550/arXiv.1409.1556. arXiv: 1409.1556[cs]. [Online]. Available: http://arxiv.org/abs/1409.1556 (visited on 11/06/2023).

[8] "Flutter - build apps for any screen." (), [Online]. Available: https://flutter.dev/ (visited on 10/13/2023).

[9] "Bernd's random-FEN-generator." (2014), [Online]. Available: http://bernd.bplaced.net/fengenerator/fengenerator.html (visited on 11/13/2023).

[10] "Geometric image transformations," Geometric Image Transformations. (Nov. 7, 2023), [Online]. Available: https://docs.opencv.org/4.x/da/d54/group__imgproc__transform.html#gaf73673a7e8e18ec6963e3774e6a94b87 (visited on 11/08/2023).

[11] "OpenCV: Camera calibration and 3d reconstruction." (Nov. 13, 2023), [Online]. Available: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga93efa9b0aa890de240ca32b11253dd (visited on 11/14/2023).

[12] "Chess pieces object detection dataset," Roboflow. (Feb. 2021), [Online]. Available: https://public.roboflow.com/object-detection/chess-full (visited on 11/14/2023).