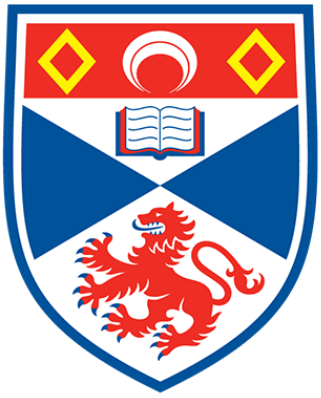
MSci Dissertation



University of
St Andrews

Chess recognition using machine learning

Author:
Georg Wölflein

Supervisor:
Dr. Ognjen Arandjelović

8th December 2020

Abstract

'Tis very abstract indeed.

Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project report is XX,XXX words long.

In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work.

Georg Wölflein

Contents

List of Figures

List of Tables

Notation

This report will follow notation conventions established in the deep learning community, in particular those described in Goodfellow *et al.* [1].

...

list symbols, etc.

Improving your weaknesses has the potential for the greatest gains.

Garry Kasparov

1 Introduction

Former World Chess Champion Garry Kasparov remarks the epigraph above in his most recent book [2] – a profound observation that applies even outside of chess. With regard to chess in particular, Kasparov implies that you must identify your mistakes and weaknesses in order to improve as a player, and to do so, you must analyse your own games.

Amateur chess players can analyse games they played online without much effort because the moves are recorded automatically. However, to analyse over-the-board games[1], players must tediously enter the position in the computer piece by piece. A casual over-the-board game between two friends will often reach an interesting position[2]. After the game, the players will want to analyse that position on a computer, so they take a photo of the position. On the computer, they need to drag and drop pieces onto a virtual chessboard until the position matches the one they had on the photograph, and then they must double-check that they did not miss any pieces.

The goal of this project is to develop a system that is able to map a photo of a chess position to a structured format that can be understood by chess engines, such as the widely-used Forsyth–Edwards Notation (FEN) [3], in order to automate this laborious task.

[1] Usually, players will invest more effort in over-the-board games, both in terms of time and deep thinking. These games will also involve a greater psychological aspect as a result of being able to observe the opponent's expressions. As such, analysing these games should be even more interesting and fruitful.

[2] For example, one of the players might have a few moves that look promising, but is also considering a line with a piece sacrifice. If he decides to play it safe, he will likely want to analyse the piece sacrifice on the computer after the game.

1.1 Context survey

Determining the game state of a chess board, also known as *chess recognition*, is a problem in computer vision whereby an algorithm is tasked with recovering the configuration of pieces from an image of a chessboard. Early work on chess recognition in the 1990s focused on extracting typeset games from printed material [4]. In recent years, the problem of parsing two-dimensional chess images has effectively been solved using conventional machine learning techniques [5] and deep learning [6], [7]. However, recognising chess positions from physical chessboards as opposed to artificial two-dimensional images poses a much more interesting and challenging problem that finds practial application in chess-playing robots, augmented reality, and aiding amateur chess players[3].

Chess robots Initial research into chess recognition emerged from the development of chess robots that included a camera to detect the human opponent's moves from a top-down overhead perspective. The difficulty of distinguishing between chess pieces from a bird's-eye-view due to their similarity is noted in many papers; as a result, chess robots typically implement a three-way classification system that for every square attempts to determine whether it contains a piece, and if so, the piece's colour. Various approaches have been explored including employing manual thresholding [9]–[12] and clustering [13] in different colour spaces, as well as differential imaging (classifying based on the per-pixel difference between two images) [14], [15]. Although the *Gambit* robot proposed by Matuszek *et al.* [16] does not require a bird's-eye view over the chessboard and uses a depth camera to more reliably detect the occupancy of each square, it employs the three-way classification strategy using a linear support vector machine (SVM) to determine the piece colour.

Chess move recording Several techniques for recording chess moves from video footage have been proposed that follow a similar three-way occupancy and colour classification scheme, both from a top-down perspective [8], [17] as well as from a camera positioned at an acute angle to the board [18]. However, in any three-way classification approach, the robot or move recorder requires knowledge of the previous board state in addition to its predictions for each square's occupancy and piece colour to deduce the last move. While this information is readily available to a chess robot or move recording software, this is not the case for a chess recognition system that should deduce the position from a single still image. Furthermore, these approaches experience severe shortcomings in terms of their inability to recover once a single move was predicted incorrectly and fail to identify promoted pieces[4] [9].

Single-image chess recognition A number of techniques have been developed to address the issue of chess recognition from a single image. Unlike move recording software or chess robots, it does not suffice to only determine the occupancy and colour of each square, but each piece must be identified.

[3]Electronic chess sets are impractical and very costly [8], thus solutions for chess recognition using just a photo of an unmodified chess board are more compelling for amateur chess players.

[4]Piece promotion occurs when a pawn reaches the last rank, in which case the player must choose to promote to a queen, rook, bishop or knight. Evidently, a vision system that can only detect the piece's colour is unable to detect what it was promoted to.

These techniques must implement a classification algorithm for each piece type (pawn, knight, bishop, queen, and king) of each colour which poses a significantly more difficult problem, attracting research mainly in the last five years. From a bird's-eye view, the pieces are nearly indestinguishable, so the photo is usually taken at an angle to the board. Ding [19] proposes a piece classifier that uses one-versus-rest SVMs trained on scale-invariant feature transform (SIFT) and histogram of oriented gradients (HOG) feature descriptors, achieving an accuracy of 85%. Danner and Kafafy [20] as well as Xie *et al.* [21] claim that SIFT and HOG provide inadequate features for the problem of piece classification due to the similarity in texture between chess pieces, and instead focus on the pieces' outlines. As such, Danner and Kafafy [20] use Fourier descriptors calculated for the pieces' contours, but this requires a manually-created database of piece silhouettes. Furthermore, they modify the board colours to red and green instead of black and white, in order distinguish the pieces from the board more easily[5]. On the other hand, Xie *et al.* [21] perform contour-based template matching with an interesting caveat: the camera angle is calculated based on the perspective transformation of the chessboard, and then depending on the angle, different templates are utilised for matching the chess pieces. As part of the same work, Xie *et al.* developed another approach that instead utilised convolutional neural networks (CNNs), but found that their original template-matching technique achieved superior results in terms of speed and accuracy in low-resolution images. However, it is important to note that their CNNs were trained on only 40 images per class and deep learning methods tend to excel when trained on larger datasets.

Chessboard detection A prerequisite to any chess recognition system is the ability to detect the location of the chessboard and each of the 64 squares. Once the four corner points have been established, finding the squares is trivial for pictures captured in bird's-eye view, and only a matter of a simple perspective transformation in the case of other camera positions. While finding the corner points of a chessboard is frequently used for automatic camera calibration due to the regular nature of the chessboard pattern [22], [23], techniques designed for this purpose tend to perform poorly when there are pieces on the chessboard that occlude lines or corners. Some of the aforementioned chess robots [13], [14], [17] as well as the single-image recognition system proposed by Danner and Kafafy [20] circumvent this problem entirely by prompting the user to interactively select the four corner points, but ideally a chess recognition system should be able to parse the position on the board without human intervention. Most approaches for automatic chess grid detection utilise either the Harris corner detector [11], [18] or a form of line detector based on the Hough transform [12], [15], [20], [24]–[27], although other techniques such as template matching [16] and flood fill [8] have been explored. In general, corner-based algorithms are unable to accurately detect grid corners when they are occluded by pieces, thus line-based detection algorithms appear to be the favoured solution. Such algorithms often take advantage of the geometric nature of the chessboard which allows to compute a perspective transformation of the grid lines that

[5]Similar board modifications have also been proposed as part of chess robots [11] and chess move trackers [8], but any such modification imposes an unreasonable constraint on normal chess games.

best matches the detected lines [18], [21], [24]. However, lines found in the background of the photo can often cause failure modes. A recent chess grid detection algorithm that is highly successful even on populated boards is descibed by Xie *et al.* in [27]. They apply several clustering algorithms on the lines detected via a Hough transform in order to find the horizontal and vertical grid lines belonging to the chessboard, and use this algorithm as a preprocessing step in their template-matching piece classification technique [21] described above.

Chess recognition using CNNs Since Xie *et al.* pioneered the use of CNNs in the domain of chess recognition from monocular images in 2018[6], a few more techniques have been developed that employ CNNs at various stages in the recognition pipeline. Czyzewski *et al.* [29] achieve an accuracy of 95% on chessboard detection from non-vertical camera angles by designing an iterative algorithm that gereates heatmaps over the input image representing the likelihood of each pixel being part of the chessboard. They then employ a CNN to refine the corner points that were found using the heatmap, outperforming the results obtained by Gonçalves *et al.* [13]. Furthermore, they compare a CNN-based piece classification algorithm to the SVM-based solution proposed by Ding [19] and find no notable improvement, but manage to obtain major improvements by implementing a probabilistic reasoning system that uses the open source Stockfish chess engine [30] as well as chess statistics [31]. Although reasoning techniques were already employed for refining the predictions of chess recognition systems before [20], [25], Czyzewski *et al.* demonstrate the potential of combining information obtained from a chess engine with large-scale chess statistics. Very recently, Mehta and Mehta [32] implemented an augmented reality app using the popular *AlexNet* CNN architecture introduced by Krizhevsky *et al.* [33], achieving promising results. Despite using an overhead camera perspective and not performing any techniques to ensure probable and legal chess positions, Mehta and Mehta achieve an end-to-end accuracy of 93% for the entire chessboard detection and piece classification pipeline.

Datasets The lack of adequate datasets for chess recognition has been recognised by many [19], [29], [32]. Although Czyzewski *et al.* [29] published a dataset of chessboard lattice points that are difficult to predict [34], large datasets – especially at the scale required for deep learning – are not available as of now. Using synthesised data in the training set is an efficient means of creating sizable datasets while minimising the manual annotation efforts [28], [29], [35]. Czyzewski *et al.* distort some input images in order to simulate different camera perspectives on the chessboard corners. However, a more promising method seems to be the use of three-dimensional models. Wei *et al.* [28] synthesise point cloud data for their volumetric CNN directly from three-dimensional chess models and Hou [35] use renderings of three-dimensional models as input. Yet Wei *et al.* [28]'s approach works only if the chessboard was captured with a depth camera and Hou [35] presents a chessboard recognition system using a simple

[6]Wei *et al.* [28] developed a chess recognition system using a volumetric CNN one year previously, but this approach requires three-dimensional chessboard data obtained from a depth camera. Their approach achieved a per-class accuracy over 90% except for the "king" class, was trained on computer-aided design (CAD) models, and evaluated on real three-dimensional images (point clouds) of a chessboard.

artificial neural network (ANN) that is not convolutional and hence achieves an accuracy of only 72%.

1.2 Objectives

1.2.1 Primary

1. Perform a literature review of available methods for parsing chess positions from photos.
2. Develop an algorithm for detecting the corners of the chessboard as well as the squares.
3. Develop an algorithm for recognising the chess pieces.
4. Develop an algorithm that uses the outputs from (2) and (3) in order to compute a probability distribution over each piece in each square.
5. Evaluate the performance of the developed algorithms.

1.2.2 Secondary

1. Create a large labelled dataset of synthesised chessboard images using 3D models.
2. Implement an algorithm that takes as input the raw probability distribution of each piece in each square and outputs a likely Forsyth–Edwards Notation [3] (FEN) description.
3. Implement a simple web API that performs the inference pipeline for an input image, returning the FEN description.

1.2.3 Tertiary

1. Develop a web app that allows the user to upload an image of the chess board to obtain the FEN description.

1.3 Ethics

There are no ethical issues raised by this project, as indicated in the signed ethics form in appendix E.

To become good at anything you have to know how to apply basic principles. To become great at it, you have to know when to violate those principles.

Garry Kasparov

2 Background

This chapter will introduce some of the basic concepts involved in ANNs, CNNs, and transfer learning required to understand this report. Readers familiar with this subject matter should feel free to skip this chapter and resume at chapter 3.

2.1 Supervised learning

At its core, the purpose of an ANN is to infer a function that maps some input to some output, based on sample input-output pairs. In machine learning, we call this a *supervised learning* task, and there are a great number of machine learning models (not only ANNs) that have been developed for this task. We shall briefly examine the two main disciplines within supervised learning: *regression* and *classification*.

2.1.1 Regression

A regression model captures the relationship between multiple input variables and one output variable. As such, it can be defined a mathematical function of the form $f : \mathbb{R}^n \to \mathbb{R}$ given by

$$f(\boldsymbol{x}) = \hat{y} = y + \epsilon \tag{2.1}$$

that models the relationship between a n-dimensional feature vector $\boldsymbol{x} \in \mathbb{R}^n$ of independent (*input*) variables and the dependent (*output*) variable $y \in \mathbb{R}$. Given a particular $\boldsymbol{x}$, the model will produce a *prediction* for y which we shall denote $\hat{y}$. Here, the additive error term ϵ represents the discrepancy between y and $\hat{y}$, i.e. the difference between the predicted and observed output.

A labelled dataset for a regression task consists of m tuples of the form $\langle \boldsymbol{x}_i, y_i \rangle$ for $i = 1, \ldots, m$. For each feature vector $\boldsymbol{x}_i$ (a row vector), the corres-

ponding y_i represents the observed output, or *label* [36]. We use the vector

$$\boldsymbol{y} = \begin{bmatrix} y_1 & y_2 & \cdots & y_m \end{bmatrix}^\top \tag{2.2}$$

to denote all the labelled outputs in the dataset, and the $m \times n$ matrix

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_m \end{bmatrix}^\top \tag{2.3}$$

for representing the corresponding feature vectors.

2.1.2 Classification

Classification is a task that finds greater applicability within this project. As the name implies, a classification model tries to determine which category each input sample belongs to from a predefined set of classes $\mathcal{C}$. To ease notation, we shall let $\mathcal{C} = \{1, 2, \ldots, C\}$ where C is the total number of classes. In practical terms, the elements in $\mathcal{C}$ could represent any type of mathematical or non-mathematical object, and in that case we only require a one-to-one mapping from those objects to $\mathcal{C}$. Furthermore, in the context of this report, each input sample can only belong to one class, and we will interpret the output of the model to represent the probabilities of the input sample belonging to each of the classes $\mathcal{C}$. Therefore, a classification model is represented by a vector-valued function $\boldsymbol{f} : \mathbb{R}^n \to [0, 1]^{|\mathcal{C}|}$ instead of a scalar function as in eq. (2.1), namely

$$\boldsymbol{f}(\boldsymbol{x}) = \hat{\boldsymbol{y}} = \boldsymbol{y} + \boldsymbol{\epsilon}. \tag{2.4}$$

Here, $\boldsymbol{f}(\boldsymbol{x})$ actually represents a probability mass function (PMF) where the i^{th} component $\hat{y}_i$ of the output vector $\hat{\boldsymbol{y}}$ represents the probability that the input sample $\boldsymbol{x}$ belongs to the class $i \in \mathcal{C}$. It follows that

$$\sum_{i=1}^{|\mathcal{C}|} \hat{y}_i = 1. \tag{2.5}$$

The observed output $\boldsymbol{y}$ is a row vector that uses a one-hot encoding (OHE) to represent the sample's class. This means that for a given class $c \in \mathcal{C}$, the components of $\boldsymbol{y}$ are given by

$$y_i = \begin{cases} 1 & i = c \\ 0 & \text{otherwise} \end{cases},$$

which retains the property described in eq. (2.5). Since $\boldsymbol{f}$ represents a PMF, the one-hot encoded vector essentially states that the probability of class c is 100% and all other classes have a probability of 0%. Notice that our outputs are now the vectors $\boldsymbol{y}_i$ instead of the scalars y_i in the regression task. Hence a labelled classification dataset will consist of m tuples of the form $\langle \boldsymbol{x}_i, \boldsymbol{y}_i \rangle$, meaning that instead of the vector $\boldsymbol{y}$ from section 2.1.2, we must use a matrix

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{y}_1 & \boldsymbol{y}_2 & \cdots & \boldsymbol{y}_m \end{bmatrix}^\top \tag{2.6}$$

to denote the targets. Each row in $\boldsymbol{Y}$ represents the one-hot encoded class label for that sample. The input matrix $\boldsymbol{X}$ remains as defined in eq. (2.3).

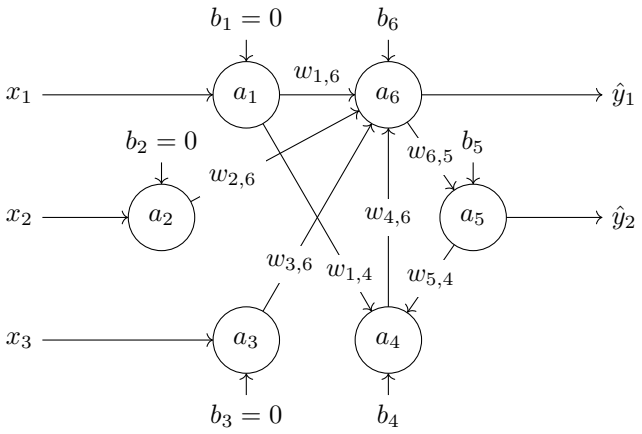


Figure 2.1: An artificial neural network with six neurons that could be used to decide a classification problem. The inputs x_1, x_2, x_3 are set as the activations of the first three units (thus b_1, b_2, b_3 have no effect) and the outputs $\hat{y}_1, \hat{y}_2$ are the activations received from the final two units. Only some weights are shown in the diagram.

2.2 Artificial neural networks

ANNs take inspiration from the human brain and can be regarded as a set of interconnected neurons. More formally, an ANN is a directed graph of n neurons (referred to as *nodes* or *units*) with weighted edges (*links*). Each link connecting two units i and j is directed and associated with a real-valued weight $w_{i,j}$.

A particular unit i's *excitation*, denoted z_i, is calculated as the weighted sum

$$z_i = \sum_{j=1}^{n} w_{j,i} a_j + b_i \tag{2.7}$$

where $a_j \in \mathbb{R}$ is another unit j's *activation* and $b_i \in \mathbb{R}$ is the i^{th} unit's *bias*. In this model, if there exists no link between unit i and a particular j then simply $w_{i,j} = 0$ and therefore j will not contribute to i's excitation. Figure 2.1 gives an example how such a network could look like.

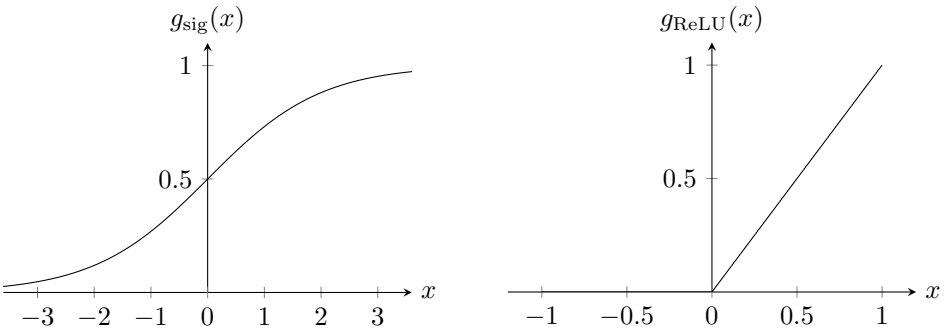
The unit i's activation is its excitation applied to a non-linear *activation function*, $g : \mathbb{R} \to \mathbb{R}$. We have

$$a_i = g(z_i) = g\left(\sum_{j=1}^{n} w_{j,i} a_j + b_i\right). \tag{2.8}$$

Activation functions In its original form in 1943, McCulloch and Pitts defined the neuron as having only binary activation [37]. This means that in our model from eq. (2.8), we would require $a_i \in \{0, 1\}$ and hence an activation function of the form $g_{\text{step}} : \mathbb{R} \to \{0, 1\}$ like the Heaviside step function[1]

$$g_{\text{step}}(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}.$$

[1] In fact, McCulloch and Pitts defined the activation to be zero when $x < \theta$ for a threshold parameter $\theta \in \mathbb{R}$ and one otherwise, but in our model the bias term b_i acts as the threshold.



(a) Sigmoid

(b) Rectified linear unit

Figure 2.2: Plots of the the two most common activation functions.

Commonly used activation functions in modern neural networks include the sigmoid

$$g_{\text{sig}}(x) = \frac{1}{1 + e^{-x}}$$

and the rectified linear unit (ReLU) [38]

$$g_{\text{ReLU}} = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \tag{2.9}$$

which are depicted in fig. 2.2. Unlike g_{step}, the range of these these activation functions is the real numbers, and the functions themselves are differentiable which is an advantage for being able to use gradient descent to optimise the weights [39, p. 729].

Rectified units do not suffer from the so-called *vanishing gradient effect* [38]. This phenomenon occurs with sigmoid activation functions when they reach high saturation, i.e. when the input is significantly far from zero such that the gradient is almost horizontal (see fig. 2.2(a)), and is especially prevelant in deep neural networks[2]. As a result, the ReLU activation function (or variants thereof) are the most popular choice nowadays. Furthermore, the computational cost of the function itself as well as its gradient is cheap.

2.2.1 Feedforward neural networks

Our definition of ANNs is so far still very general and makes virtually no restrictions on the graph of the network. It turns out that it is difficult to propagate activations between neurons when they exhibit cycles, as is the case with the last three units in fig. 2.1. Thus we impose a constraint that the nodes of the network are not allowed to form cycles, and as a result the network becomes a directed acyclic graph (DAG). This class of ANNs are referred to as *feedforward neural networks*.

[2]Deep neural networks are ANNs with many layers.

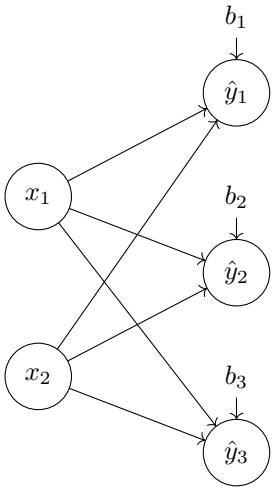


Figure 2.3: A SLP with two inputs and three outputs.

2.2.1.1 Single-layer perceptron

The most basic type of feedforward neural network is the single-layer perceptron (SLP). It consists of n_0 input units that are directly connected to n_1 output units, as illustrated in fig. 2.3. There are no connections between input units, and no connections between output units. Likewise, there are no connections from output units to input units. The only connections in the network originate from input units and feed to output units. In fact, that is where the term *feedforward* arises: the network exhibits neither backwards nor intra-layer connections. The connections themselves are of course weighted as in any ANN. Due to the unidirectional nature of the links, we can continue to use the notation $w_{i,j}$ to denote weights, but now i refers to the input unit and j refers to the output unit.

From eq. (2.8), we can compute the values of the three output units in fig. 2.3 as

$$\hat{y}_j = g\left(\sum_{i=1}^{n} w_{i,j}x_i + b_j\right) \tag{2.10}$$

for $j = 1, 2, 3$. Mathematically, a SLP is represented by a function $\boldsymbol{f} : \mathbb{R}^{n_0} \to \mathbb{R}^{n_1}$ that maps an input vector $\boldsymbol{x} \in \mathbb{R}^{n_0}$ to an output vector $\hat{\boldsymbol{y}} \in \mathbb{R}^{n_1}$. Let us use the $n_0 \times n_1$ matrix $\boldsymbol{W}$ to contain all weights such that

$$\boldsymbol{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n_1} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n_1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_0,1} & w_{n_0,2} & \cdots & w_{n_0,n_1} \end{bmatrix}.$$

and the vector

$$\boldsymbol{b} = \begin{bmatrix} b_1, b_2, \ldots, b_{n_1} \end{bmatrix}$$

to represent the biases. Since the output vector $\hat{\boldsymbol{y}}$ is simply the concatenation of the output units, the summation in eq. (2.10) we finally define the function

of a SLP as

$$\boldsymbol{f}_{\boldsymbol{W},\boldsymbol{b}}(\boldsymbol{x}) = \hat{\boldsymbol{y}} = \boldsymbol{g}\left(\boldsymbol{w}^\top \boldsymbol{x} + \boldsymbol{b}\right) \tag{2.11}$$

where $\boldsymbol{g}(\cdot)$ applies the activation function g pointwise. The process of computing the activations of the output units given the input units is often called *forward propagation* or *forward pass* [36].

2.2.1.2 Multi-layer perceptron

As the name implies, a multi-layer perceptron (MLP) simply stacks multiple SLPs on top of each other, such that each layer's outputs are connected to the next layer's inputs (except for the final layer, where the outputs represent $\hat{\boldsymbol{y}}$). A MLP with L layers can be expressed mathematically by composing L SLPs $\boldsymbol{f}_1, \boldsymbol{f}_2, \ldots, \boldsymbol{f}_L$:

$$\boldsymbol{f}(\boldsymbol{x}) = (\boldsymbol{f}_1 \circ \boldsymbol{f}_2 \circ \cdots \circ \boldsymbol{f}_L)(\boldsymbol{x}). \tag{2.12}$$

Each SLP inside the MLP is constitutes a logical module of the MLP which we call a *layer*. Due to the fact that each layer has its own weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$, we will use the parenthesised superscript notation $\boldsymbol{W}^{(l)}$ and $\boldsymbol{b}^{(l)}$ to denote the weights and biases in the l^{th} layer. Further, the notation $b_j^{(l)}$ refers to the j^{th} unit's bias in layer l, and $w_{i,j}^{(l)}$ is the weight of the connection from unit i in layer $l-1$ to unit j in layer l. This notation follows Goodfellow *et al.* [1] and is customary in describing neural networks We shall use $n_0, n_1, \ldots, n_L$ to describe the number of units in each layer. Layer zero is considered the *input layer*, meaning that n_0 must be equal to the dimensionality of $\boldsymbol{x}$. The layers $1, 2, \ldots, L-1$ are the *hidden layers* because they are not directly connected to the input or output units. Finally, we refer to layer L as the *output layer*, since its units represent the prediction $\hat{\boldsymbol{y}}$, and so n_L must be equal to the dimensionality of $\hat{\boldsymbol{y}}$. Figure 2.4 gives an example of a MLP architecture. It can clearly be seen that since MLPs are simply nested SLPs, they still form DAGs and thus are feedforward networks. In fact, such networks are usually *fully-connected feedforward networks* because all units from a particular layer connect to all units from the previous layer.

2.2.2 Backpropagation

Training with regard to neural networks refers to the process of altering a network's weights and biases with the goal of achieving an optimal configuration that reduces the error of the predictions, i.e. how far they are 'off'. This is how the network facilitates *learning* the input-output function from eq. (2.1) that we introduced at the very beginning of this chapter in the context of supervised learning.

Backpropagation with gradient descent is an iterative algorithm for training neural networks that, provided a suitable learning rate α, is guaranteed to converge to a *local minimum*. The main idea is as follows:

1. Calculate the derivative of the loss function with respect to the current trainable parameters $\boldsymbol{p}$ (weights and biases) as $\boldsymbol{\Delta p} = \frac{\delta L}{\delta \boldsymbol{p}}(\boldsymbol{p})$.

2. Take a step in the negative direction of this gradient, i.e. update the trainable parameters $\boldsymbol{p} \leftarrow \boldsymbol{p} - \alpha \boldsymbol{\Delta p}$ where $\alpha \in \mathbb{R}$ is the learning rate.

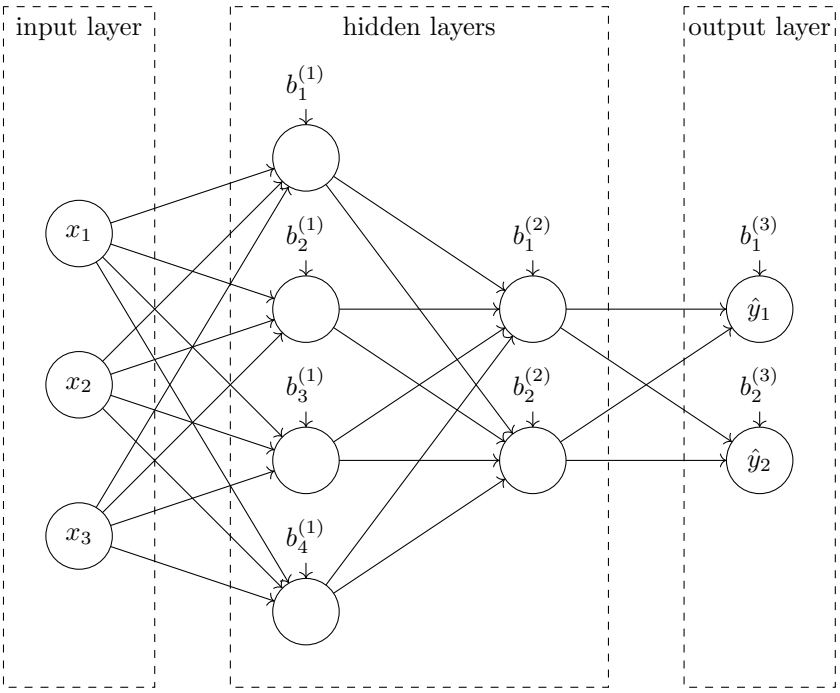


Figure 2.4: A MLP with three inputs, two hidden layers, and two outputs.

3. Repeat steps 1 and 2 until a predefined convergence criterion is met.

Figure 2.5 shows the steps that this algorithm would make on a simple error-weight surface with only one parameter.

To explain the backpropagation algorithm, we shall examine the case of a regression task, meaning that the network has only one output unit. The algorithm can, of course, be generalised to classification problems with multiple output units. Let $\{\langle \boldsymbol{x}_i, y_i \rangle\}_{i=1}^m$ be a labelled regression dataset as defined in section 2.1.1.

We must first introduce a measure indicating how good our predictions are. For simplicity, we will use the mean squared error (MSE), although note that for classification problems, the cross-entropy loss is preferred. The MSE of a set of predictions $\hat{\boldsymbol{y}}$ is the average squared difference between the predictions and targets,

$$E\left(\hat{\boldsymbol{y}}, \boldsymbol{y}\right) = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}_i - y_i\right)^2 = \left\|\hat{\boldsymbol{y}} - \boldsymbol{y}\right\|_2^2 . \tag{2.13}$$

The reader should not confuse this notation with that of a classification problem; here, we use $\boldsymbol{y}$ to denote the targets for each sample in the dataset and $\hat{\boldsymbol{y}}$ to denote the corresponding predictions.

We can formulate a loss function

$$L\left(\hat{\boldsymbol{y}}, \boldsymbol{y}\right) = \frac{1}{2} \sum_{i=1}^{m} \left(\hat{y}_i - y_i\right)^2 \tag{2.14}$$

that behaves similarly to eq. (2.13) in that reducing the loss function will also reduce the MSE, but it will be easier to work with.

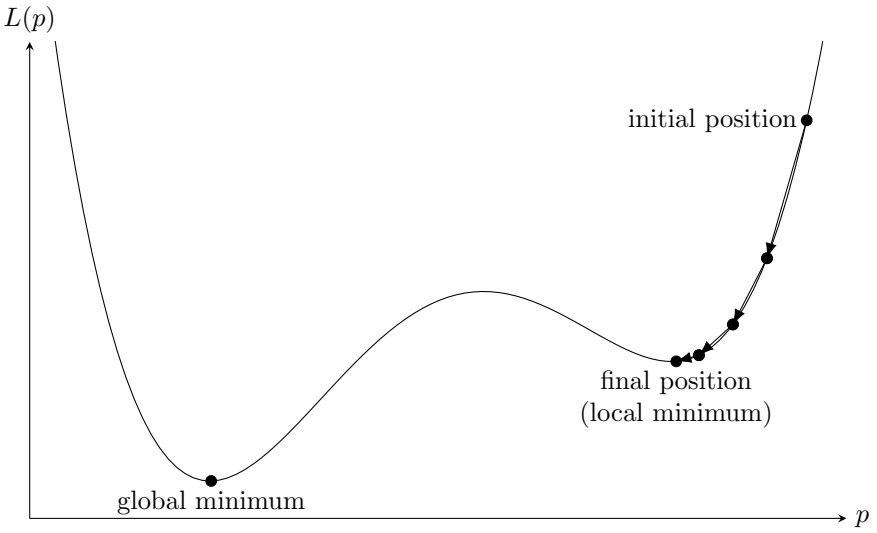


Figure 2.5: An illustration gradient descent training on an error-weight surface with only one parameter (not drawn to scale). Gradient descent minimises the objective loss function and converges to a local minimum. In practice, deep neural networks have parameters in the order of one million, resulting in a high-dimensional error surface, and local minima often turn out to be 'good enough' in that their error is only slightly higher than the global minimum [40].

2.2.2.1 Single-layer perceptron

In a SLP, we can easily express the loss in terms of the weights and biases by combining eqs. (2.11) and (2.14) as

$$L = \frac{1}{2} \sum_{i=1}^{m} \left(g\left(\boldsymbol{w} \boldsymbol{x}_i + b \right) - y_i \right)^2. \tag{2.15}$$

Calculating the derivative of the loss function with respect to each of the trainable parameters is a core part of the gradient descent algorithm. The trainable parameters in our network are $\boldsymbol{w}$ and b, so we will differentiate L with respect to each of these. We obtain the partial derivative of the loss with respect to the bias as

$$\begin{aligned} \frac{\delta L}{\delta b} &= \sum_{i=1}^{m} \left(g\left(\boldsymbol{w}^\top \boldsymbol{x}_i + b \right) - y_i \right) \frac{\delta}{\delta b} \left(g\left(\boldsymbol{w}^\top \boldsymbol{x}_i + b \right) - y_i \right) \\ &= \sum_{i=1}^{m} \left(g\left(\boldsymbol{w}^\top \boldsymbol{x}_i + b \right) - y_i \right) g'\left(\boldsymbol{w}^\top \boldsymbol{x}_i + b \right), \end{aligned} \tag{2.16}$$

and similarly we can differentiate with respect to the weights

$$\begin{aligned} \frac{\delta L}{\delta \boldsymbol{w}} &= \sum_{i=1}^{m} \left(g\left(\boldsymbol{w}^\top \boldsymbol{x}_i + b \right) - y_i \right) \frac{\delta}{\delta \boldsymbol{w}} \left(g\left(\boldsymbol{w}^\top \boldsymbol{x}_i + b \right) - y_i \right) \\ &= \sum_{i=1}^{m} \left(g\left(\boldsymbol{w}^\top \boldsymbol{x}_i + b \right) - y_i \right) g'\left(\boldsymbol{w}^\top \boldsymbol{x}_i + b \right) \boldsymbol{x}_i. \end{aligned} \tag{2.17}$$

Here, the derivative $g'(\cdot)$ depends on the choice of activation function $g(\cdot)$.

At each iteration, the gradient descent algorithm updates the trainable parameters by taking a step in the direction opposite to the gradient. The size of this step is governed by the learning rate $\alpha \in \mathbb{R}$. Mathematically, this is expressed as

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \frac{\delta L}{\delta \boldsymbol{w}}(\boldsymbol{w}), \tag{2.18}$$

$$b \leftarrow b - \alpha \frac{\delta L}{\delta b}(b). \tag{2.19}$$

2.2.2.2 Multi-layer perceptron

To apply backpropagation in MLPs, we must first compute the gradient of the loss with resepect to each trainable parameter (i.e. each layer's weights and biases). We must compute $\frac{\delta L}{\delta \boldsymbol{W}_l}$ and $\frac{\delta L}{\delta \boldsymbol{b}_l}$ for each layer $l = 1, 2, \ldots, L$; the exact derivation is left as an exercise to the reader due to the scope of this report. The main idea is realising that because MLPs are simply nested SLPs, we can apply the chain rule when calculating the derivative of eq. (2.12). Similar to eqs. (2.18) and (2.19), we update the weights and biases by applying the rule

$$\boldsymbol{W}_l \leftarrow \boldsymbol{W}_l - \alpha \frac{\delta L}{\delta \boldsymbol{W}_l}(\boldsymbol{W}_l), \tag{2.20}$$

$$\boldsymbol{b}_l \leftarrow \boldsymbol{b}_l - \alpha \frac{\delta L}{\delta \boldsymbol{b}_l}(\boldsymbol{b}_l). \tag{2.21}$$

A major limitation of gradient descent as we described it here is the summation in eqs. (2.16) and (2.17) that is applied over the whole dataset of m samples. This becomes impractical for large datasets due to memory consumption and computational complexity, so instead we commonly split the dataset into batches and estimate the gradients for each batch in a process known as *stochastic gradient descent*. Furthermore, we typically apply more sophisticated optimisation algorithms than the update rules outlined in eqs. (2.20) and (2.21) to improve the speed and quality of convergence. A very popular choice that demonstrates empirically demonstrates good results in many settings is the Adam optimiser [41]. However, such details are unfortunately outwith the scope of this report.

2.3 Convolutional neural networks

2.4 Transfer learning

I don't look at computers as opponents. For me it is much more interesting to beat humans.

Magnus Carlsen

3 Data synthesis

Studies in human cognition by Bilalić *et al.* [42] and Zhou [43] compared skilled chess players with novices in terms of their ability to remember a chess position for a short amount time confirmed that highly skilled players outperform novices at this task. Perhaps more interestingly, both studies found that skilled players remembered *random* positions (where pieces are positioned on random squares, not necessarily obeying the rules of chess) significantly less accurately than they did positions from actual chess games. Thus it stands to reason that in general, highly skilled chess players exhibit a more developed pattern recognition ability for chess positions than novices, but this ability is specific to positions that conform to the rules of chess and are likely to occur in actual games.

This project aims to develop a similar pattern recognition ability using machine learning, and therefore our dataset will consist of positions from real chess games. In doing so, we automatically ensure that the chess positions are legal according to chess rules such that a probabilistic reasoning system could later yield sensible results in a post-processing step (see Objective 2.2).

3.1 Chess positions

The positions are generated from a publicly available dataset of 2,851 games played by current World Chess Champion Magnus Carlsen [44]. Each move in each game is included in our dataset with a probability of 2%, and duplicate positions are discarded. A total of 4,888 chess positions are obtained in this manner and saved in FEN format.

3.2 Three-dimensional renders

In order to obtain realistic images of these chess positions, we employ a three-dimensional model of a chess set on a wooden table. Chess pieces are placed

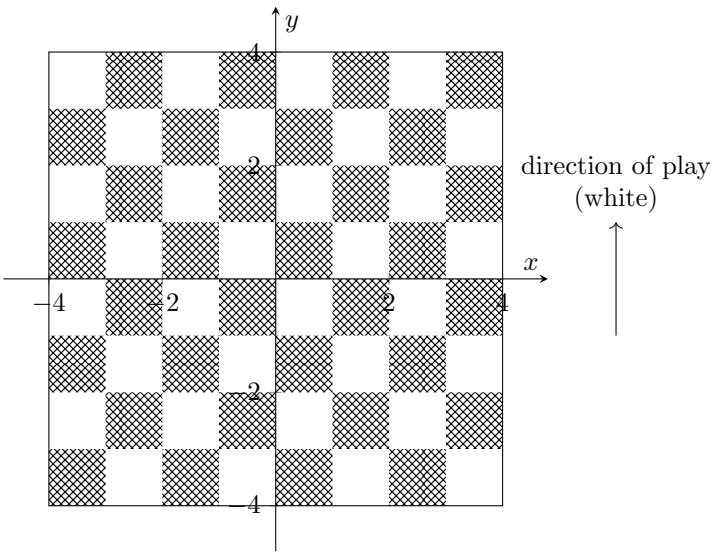


Figure 3.1: Overhead view of the coordinate system on the chessboard. The z-axis (not shown) points upward, normal to the chessboard surface, and the board is oriented like a chess game would be set up, i.e. the bottom right square is white. White's direction of play (the direction in which pawns are advanced) coincides with the y-axis.

on the board squares according the given FEN description. Different camera angles and lighting setups are chosen in a random process in order to maximise diversity in the dataset.

Let us consider a three-dimensional Cartesian coordinate system whose origin lies at the centre point of the chessboard's surface, as depicted in fig. 3.1. The chessboard lies on the plane formed by the x and y axes, and the chess squares are of unit length.

Pieces The pieces are positioned on the squares as dictated by the particular FEN description. However, instead of positioning them at the centre in their respective squares, they are randomly rotated and positioned with a random offset to emulate the conditions in real chess games. More specifically, the x and y position of a piece in file i and rank j is sampled from a bivariate normal distribution given by

$$\mathbf{p}_{i,j} \sim \mathcal{N}\left(\begin{bmatrix} i \\ j \end{bmatrix} - \frac{7}{2}, \frac{\boldsymbol{I}_2}{10}\right).$$

Here, we assume that i and j are zero-indexed, i.e. the square *a1* corresponds to $i = j = 0$. The reason for shifting the mean by $\frac{7}{2}$ above is that since the origin lies at the midpoint of the board, the mean must be shifted four units to the left (or downwards for the y-axis), but since the normal distribution should be centred on the midpoint of the square, we must add one half. Due to the fact that the x and y axes are perpendicular, the two components of $\mathbf{p}$ will be independent and thus can be modelled with a covariance matrix that is a multiple of the identity matrix $\boldsymbol{I}_2$. Experiments showed that a variance of $\frac{1}{10}$ achieved realistic results. Finally, the piece's rotation about its z-axis is sampled from a uniform distribution over the half-open interval $[0, 2\pi)$.

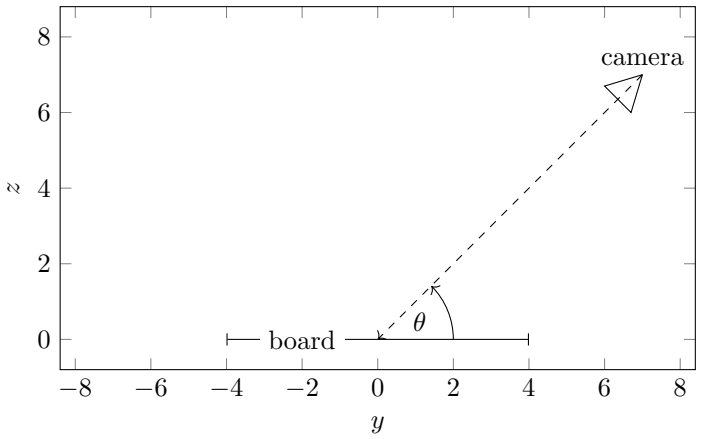


Figure 3.2: Side view of the camera setup for the scenario where it is white to move.

Camera The camera is aligned such that it points directly at the origin (i.e. the centre of the board). It is positioned with only a small offset from the yz-plane to ensure that the view over the chessboard is similar to the current player's perspective. A slight perturbation to the x-component of the camera position is introduced according to a normal distribution with $\mu = 0$ and $\sigma = 0.8$ since the player will not usually be positioned exactly in the middle in front of the board. An angle θ is chosen uniformly in the range $\left[\frac{\pi}{4}, \frac{\pi}{3}\right]$ to represent the angle that the camera makes with the board's surface (see fig. 3.2) if white is to play[1]. This range is chosen because human players would typically choose a camera angle between 45 and 60 degrees to ensure maximum visibility of the pieces. The two remaining components (y and z) of the camera's location is then obtained using a simple trigonometric calculation such that the distance from the camera to the origin is 11 units, a length that allows the camera to capture the entire board.

Lighting For each chess position, a random choice is made between two different lighting scenarios, each having equal probability of being employed.

1. The first lighting mode tries to emulate a *camera flash*. To do so, a spotlight is set up with the same location and orientation as the camera. As a result, the scene is light up quite well with no large shadows, as it can be seen in fig. 3.3(a).

2. In the other lighting mode, two spotlights are set up in the scene. Their x and y coordinates are constrained such that they lie on a circle centred at the origin of the coordinate system with radius 10 on the xy-plane, as depicted in fig. 3.4, but each spotlight's location along the circumference is sampled uniformly. Furthermore, each spotlight's z-component is sampled uniformly in the range $[5, 10)$. Finally, the for each spotlight, a focus point on the chessboard surface (i.e. the xy plane) is sampled

[1]On the other hand, if it is black to play, the perspective must be from the other side of the board, so θ is chosen in the range $\left[\frac{2\pi}{3}, \frac{3\pi}{4}\right]$ which is equivalent to reflecting the camera position about the xz-plane.

(a) camera flash

(b) spotlights

Figure 3.3: Two samples from the synthesised dataset showing both types of lighting.

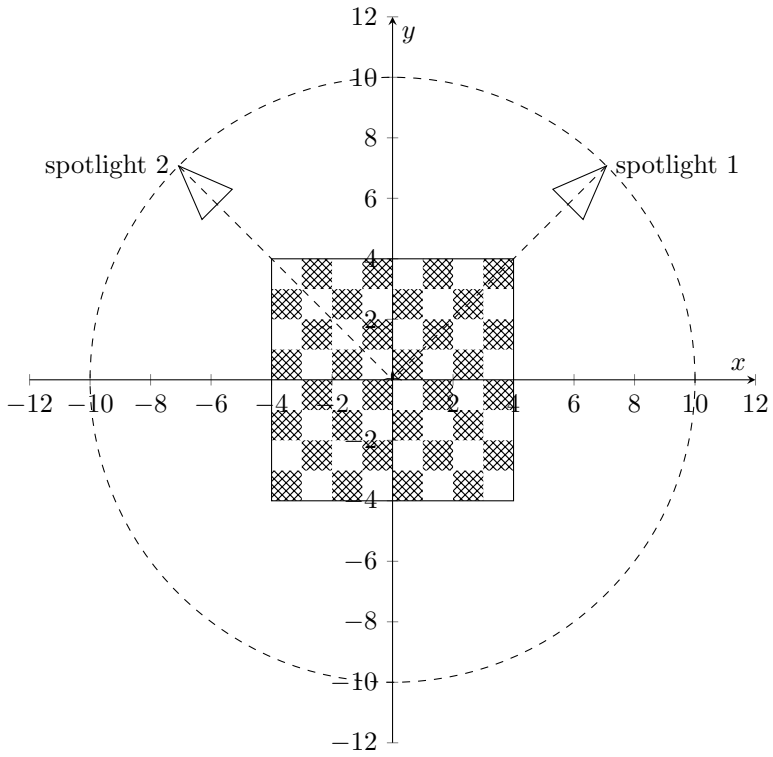


Figure 3.4: Overhead view of the chessboard with two spotlights. The spotlights are constrained to the dashed circle such that their distance to the origin amounts to 10 units when disregarding the z-component.

from $\mathcal{N}\left(\begin{bmatrix}0\\0\end{bmatrix}, \frac{5}{2}\boldsymbol{I}_2\right)$ and the corresponding spotlight is rotated such that it points in that direction. Consequently, there is greater variabililty in the lighting because the spotlights could be pointing at different areas of the board, thus producing different types of shadows. Figure 3.3(b) shows an example rendering where the lighting produced by the spotlights is poorer than the camera flash mode.

3.3 Automated labelling

use the chesscog.data_synthesis.vis script

todo: explain how dataset was split into train/val/test

Of chess, it has been said that life is not long enough for it, but that is the fault of life, not chess.

William Napier

4

Recognising chess positions

The chess recognition system is responsible for taking an RGB image of a chessboard with pieces on it and ultimately produce a FEN string representing the predicted chess position. The pipeline, depicted in fig. 4.1, consists of four main stages which shall be described in detail in the subsequent sections.

The first step is locating the chessboard in the image. More specifically, we are interested in finding the pixel coordinates of the four chessboard corners. Using these corner points, we will warp the image such that the chessboard will form a regular square grid, to eliminate perspective distortion in the sizes of the chess squares. Next, we train a binary CNN classifier to determine the occupancy of each square. Each of the occupied squares will then be input to another CNN that is responsible for determining the type of piece. Finally, we analyse the output probabilities of the piece classifier to produce a FEN string representing the position.

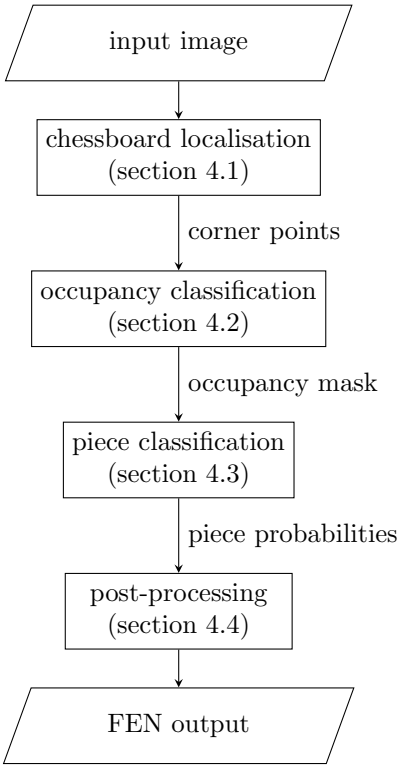


Figure 4.1: Overview of the chess recognition pipeline.

4.1 Board localisation

To determine the location of the chessboard's corners, we will rely on its regular geometric nature. Each square on the physical chessboard has the same width and height, even though their observed dimensions in the input image will vary due to 3D perspective distortion.

4.1.1 Finding intersection points

A chessboard consists of 64 squares that are arranged in an 8×8 grid, thus there are nine horizontal and nine vertical lines. In the first step of our algorithm, we would like to detect the majority of these lines and find their intersection points. Before conducting any futher computation, we resize the image to a width of 1,200 pixels, keeping the aspect ratio.

4.1.1.1 Edge detection

To detect edges in the input image, we convert the RGB image to grayscale and apply the Canny edge detector [45], the result of which is shown in fig. 4.2(c). Canny edge detection is a multi-stage algorithm that first reduces noise in the image by applying a Gaussian blur, then calculates pixel intensity gradients, performs non-maximum supression, and finally refines the results using hysteresis thresholding [45], although the precise details of this algorithm go beyond the scope of this report.

4.1.1.2 Line detection

Next, we perform the so-called Hough transform [46], [47], in order to detect lines that are formed by the edges. To this end, we will represent lines in Hesse normal form, meaning that that they are parameterised by the angle θ that the normal vector makes to the x-axis, and the distance ρ to the origin. Figure 4.3 explains this geometrically and gives rise to the equation of a line in Hesse normal form,

$$\rho = x \cos\theta + y \sin\theta. \tag{4.1}$$

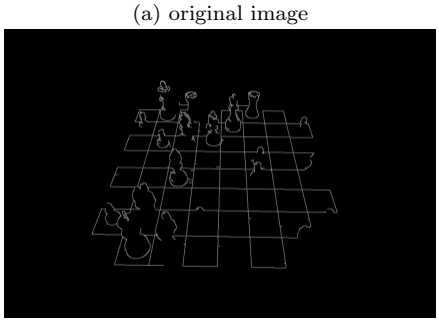
In fact, for a particular point (x, y), eq. (4.1) represents the family of lines passing through it. Here, each pair of (ρ, θ) values represents one particular line, and we only need to consider pairs where $\rho \geq 0$ and $0 \leq \theta < 2\pi$ to parameterise all lines.

Plotting the pairs of (ρ, θ) values for a particular point in image space will give a sinusoidal curve that represents all lines passing through that point. We can take multiple points in image space and plot their sinusoids. Then, each intersection in (ρ, θ) space will represent a line passing through the corresponding points, as illustrated in fig. 4.4.

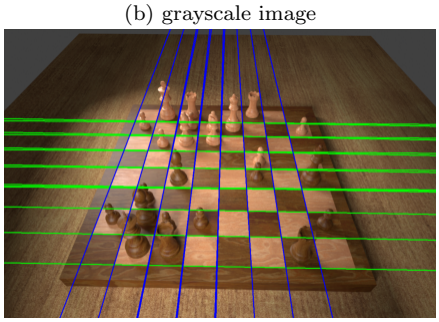
For a given threshold t, the Hough transform essentially plots the sinusoids for all edge points in the input image (which we obtained using the Canny edge detection algorithm) and outputs intersection points in (ρ, θ) space that are on at least t different sinusoids. Consequently, the lines identified by the Hough transform are corroborated by at least t edge points in the image.

(a) original image

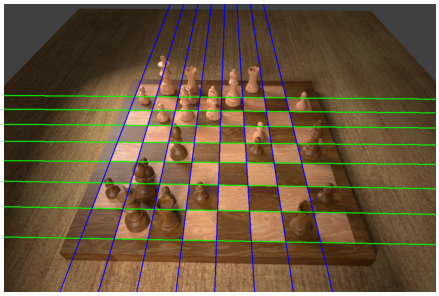
(b) grayscale image



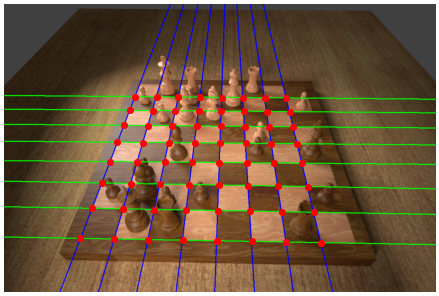
(c) detected edges



(d) detected lines, clustered into horizontal (green) and vertical (blue) lines



(e) elimination of similar lines



(f) intersection points (red)

Figure 4.2: The process of determining the intersection points on the chessboard.

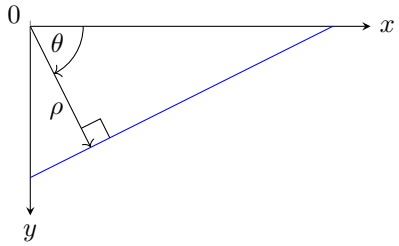
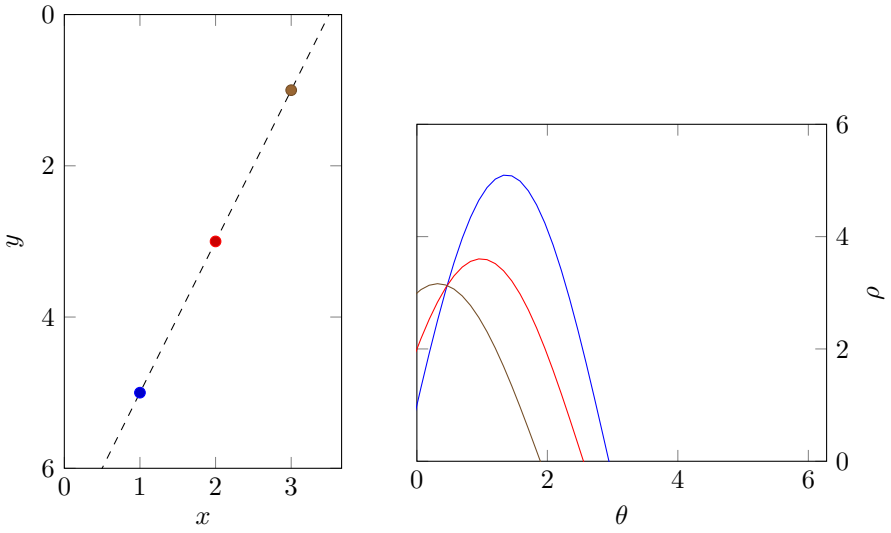


Figure 4.3: A line in Hesse normal form is parameterised by the angle θ to the x-axis and distance ρ from the origin. The line is defined by the equation $\rho = x\cos\theta + y\sin\theta$. Equivalently, in slope-intercept form, we have $y = -x\cot\theta + \frac{\rho}{\sin\theta}$ provided that $\cos\theta \neq 0$, i.e. θ is not a multiple of π. Notice that the y-axis is pointing down because the origin of the coordinate system is the top left of the image.



(a) three points in image space

(b) lines passing through these three points in (ρ, θ) space

Figure 4.4: An example of how lines in image space are represented in (ρ, θ) space. The family of lines passing through a particular point in image space (a) is represented by a sinusoid in (ρ, θ) space (b). The intersection point of the three curves represents the line passing through all three points.

4.1.1.3 Clustering

On the chessboard images, the Hough transform typically yields around 200 lines, most of which are very similar. We will first split them into horizontal and vertical lines and then eliminate similar lines. Experiments showed that simply setting thresholds for the angle θ is insufficient for robustly classifying lines as horizontal or vertical. This is because the camera is often tilted quite severely in the dataset (and might be the case in real chessboard photos taken by humans, too). Instead, we employ agglomerative clustering which is a bottom-up hierarchical clustering algorithm. In this algorithm, each line starts off in its own cluster, and as the algorithm progresses, pairs of clusters are merged in a manner that aims to minimise the variance within the clusters. We cluster the lines based only on their direction, i.e. their angle θ, and use the smallest angle between two given lines as the distance metric. Finally, we use the mean angle of both top-level clusters to determine which cluster represents the vertical lines and which the horizontal lines. The clustered lines are depicted in fig. 4.2(d).

Next, we must eliminate similar lines by finding clusters of similar lines. To eliminate similar horizontal lines, we first find the mean vertical line by considering the ρ and θ values associated to the lines in the vertical cluster. Then, we find the intersection points of all the horizontal lines with the mean vertical line and perform a DBSCAN clustering [48] to group similar lines based on these intersection points. We use the mean ρ and θ values of the lines in each group to represent the final set of discovered horizontal lines. The same procedure is applied vice-versa for the vertical lines, the result of which is shown in fig. 4.2(e).

4.1.1.4 Intersection points

It remains to find the intersection points of the horizontal and vertical lines. Given two lines parameterised by (ρ_1, θ_1) and (ρ_2, θ_2) respectively, we can find their point of intersection by observing that their equations in Hesse normal form (eq. (4.1)) forms a system of two linear equations that can be solved for x and y. We have

$$\begin{aligned}\rho_1 &= x\cos\theta_1 + y\sin\theta_1\\ \rho_2 &= x\cos\theta_2 + y\sin\theta_2.\end{aligned}$$

Rearranging for x and y, we obtain after some algebraic manipulation

$$x = \frac{\rho_2\sin\theta_1 - \rho_1\sin\theta_2}{\cos\theta_2\sin\theta_1 - \cos\theta_1\sin\theta_2}, \tag{4.2}$$

$$y = \frac{\rho_2\cos\theta_1 - \rho_1\cos\theta_2}{\sin\theta_2\cos\theta_1 - \sin\theta_1\cos\theta_2}. \tag{4.3}$$

Finally, using eqs. (4.2) and (4.3) on each pair of horizontal and vertical lines, we can find the intersection points as depicted in fig. 4.2(f).

4.1.2 Finding the homography

Once we have obtained the intersection points, the next step is to project them on to a regular grid. Using this projection, known as a *homography*, we can warp the original image to eliminate the perspective distortion.

4.1.2.1 Computing the homography matrix

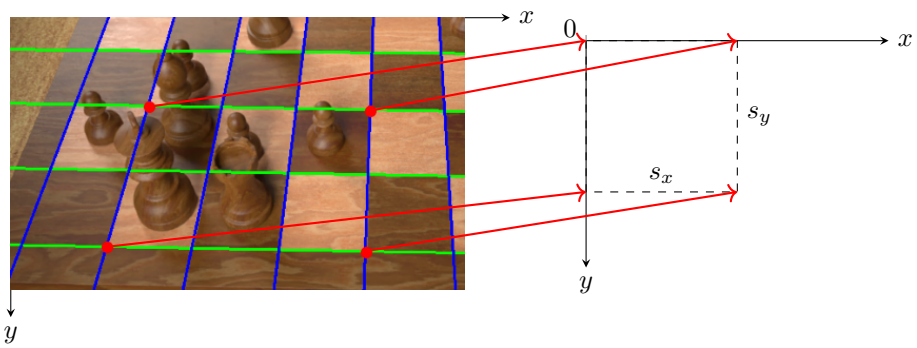
The projection is described by a *homography matrix* $\boldsymbol{H} \in \mathbb{R}^{3\times3}$ [49] mapping any point $\boldsymbol{p}$ from the original image to the corresponding point $\boldsymbol{p'}$ in the warped image using the relation

$$\boldsymbol{H}\boldsymbol{p} = q\boldsymbol{p'} \tag{4.4}$$

up to a scalar factor $q \in \mathbb{R}$. Here, we consider $\boldsymbol{p}$ and $\boldsymbol{p'}$ as 2D *homogenous coordinate vectors*, which are three-component vectors that extend the Euclidean plane with points at infinity. More practically, a point (x, y) in Euclidean space corresponds to the set of homogenous coordinates $\begin{bmatrix}qx & qy & q\end{bmatrix}^\top$ for any $q \in \mathbb{R}, q \neq 0$. In practical terms, we can thus convert a homogenous coordinate to the corresponding Euclidean coordinate by taking its first two components and dividing by the third.

To compute a homography, we require four source points on the original image and four corresponding destination points in the warped image. We can choose four instersection points that lie on two distinct horizontal and two distinct vertical lines (according to the lines found in fig. 4.2(e)) and therefore represent a rectangle on the chessboard. Let the 3×4 matrix $\boldsymbol{P}$ contain these points' homogenous coordinates as column vectors in clockwise order.

Let $\boldsymbol{P'}$ be a matrix of the same size as $\boldsymbol{P}$ containing this rectangle's coordinates in clockwise order. In the warped space, the squares of the chessboard will be of unit length. Therefore, we can map the four points from the original image to a rectangle whose top left vertex coincides with the origin and whose side lengths are s_x and s_y (see fig. 4.5). We have



(a) region of the original image (b) warped image

Figure 4.5: Projection of four intersection points from the original to the warped image.

$$\boldsymbol{P'} = \begin{bmatrix} 0 & 0 & 1 \\ s_x & 0 & 1 \\ s_x & s_y & 1 \\ 0 & s_y & 1 \end{bmatrix}^\top .$$

Since the homography matrix maps points from the original image to the output image, we obtain from eq. (4.4) the relation

$$\boldsymbol{HP} = \boldsymbol{P'}(\boldsymbol{I}_4\boldsymbol{q}) \tag{4.5}$$

where the vector $\boldsymbol{q} \in \mathbb{R}^4$ represents the scale factor for each point. Equation (4.5) describes a system of linear equations that we can solve computationally for $\boldsymbol{H}$. Then, we can use eq. (4.4) to project all other intersection points to the warped image.

4.1.2.2 Finding the optimal homography

Although fig. 4.2(f) has no intersection points that are outliers (because all of the identified lines do, in fact, correspond to squares on the chessboard), our algorithm must be robust in the face of lines identified in the image that do not correspond to squares on the chessboard. Figure 4.6 depicts one such example; however, in non-synthetic chessboard images, especially with other objects in the image that exhibit straight lines, there is often an even greater number of outliers.

Random sample consensus (RANSAC) [50] is a non-deterministic iterative model fitting algorithm that can robustly estimate a model's parameters in such a manner that is not significantly influenced by outliers. It can be applied to a variety of situations where a model must be fit to a dataset containing outliers. In our case, the dataset consists of the intersection points (some of which are outliers due to detecting irrelevant lines) and the model is a homography that corresponds to the inlier intersection points. The RANSAC algorithm consists of two main steps:

1. Randomly sample a set of observations from the dataset that contains the minimum number of samples sufficient to fit the model. We require four

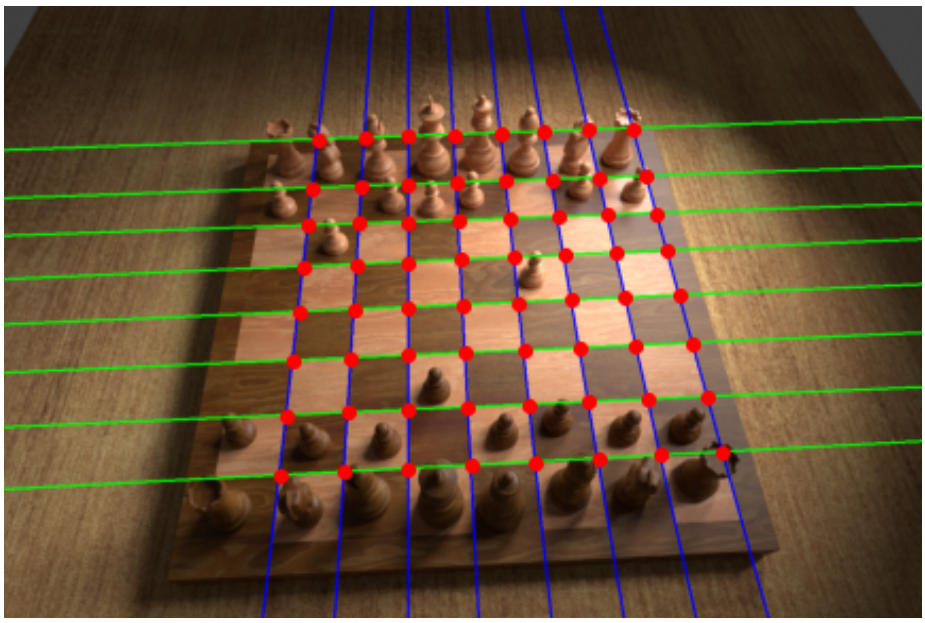


Figure 4.6: A sample from the training set with an incorrectly identified line. The top horizontal line is at the border of the chessboard instead of the top chessboard squares.

points to compute a homography, so this is the number of intersection points we must sample in this step.

2. Fit the model to these samples, and then determine which observations are explained by the model within some threshold (the threshold should be just large enough to account for the expected noise in the dataset). This is the set of inliers, and if the size of this set is greater than the inliers discovered in the previous iteration, we will retain this model instead of the previous. Repeat from step 1 until reaching a specified number of iterations.

Applied to our problem, choosing any four intersection points in step 1 would be quite unwise because we would not know which points they represent on the chessboard, making it impossible to map them to the warped image such that the chess squares are snapped to the grid of whole numbers. Instead, we will randomly choose four points that we believe to form a rectangle on the chessboard. This means that we will select two horizontal and two vertical lines at random, and choose the four intersection points of these lines as the random sample. We know that these four points will form a rectangle in the warped image, so we can map to a rectangle as explained in section 4.1.2.1 and illustrated in fig. 4.5.

The RANSAC-based method of finding the inlier intersection points is described in Algorithm 1. To simplify notation in the algorithm, we use the function $h : \mathbb{R}^2 \to \mathbb{R}^3$ to convert Euclidean coordinates to homogenous coordinates and $h^{-1} : \mathbb{R}^3 \to \mathbb{R}^2$ to denote the reverse operation. These functions can

Algorithm 1 RANSAC-based algorithm for finding the optimal homography.

Input: the set of intersection points $\mathcal{P} \subset \mathbb{R}^2$

Parameters: minimum number of iterations $i_{\min}$, maximum number of iterations $i_{\max}$, minimum number of inliers c, inlier threshold $\epsilon_{\max}$

Output: a bijective mapping $m : \mathcal{I} \to \mathcal{W}$ from a set of inlier points $\mathcal{I} \subseteq \mathcal{P}$ to a set of warped points $\mathcal{W} \subset \mathbb{Z}^2$ that is given by its graph $\mathcal{M} \subset \mathcal{I} \times \mathcal{W}$

$\mathcal{P} \leftarrow \{h(\boldsymbol{p}) : \boldsymbol{p} \in \mathcal{P}\}$ ▷ convert to homogenous coordinates
$\mathcal{I}, \mathcal{W}, \mathcal{M} \leftarrow \emptyset, \emptyset, \emptyset$ ▷ initialise outputs
$i \leftarrow 0$ ▷ number of iterations
while $(i < i_{\min}) \vee (|\mathcal{I}| < c)$ **do**
  $x_1, x_2 \leftarrow$ choose$(\{1, 2, \ldots, x_{\max}\})$ ▷ randomly choose such that $x_1 < x_2$
  $y_1, y_2 \leftarrow$ choose$(\{1, 2, \ldots, y_{\max}\})$ ▷ randomly choose such that $y_1 < y_2$
  $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4 \leftarrow$ intersection points in $\mathcal{P}$ corresponding to the four points described by vertical lines x_1, x_2 and horizontal lines y_1, y_2 (in clockwise order)
  $\boldsymbol{P} \leftarrow \begin{bmatrix} \boldsymbol{p}_1 & \boldsymbol{p}_2 & \boldsymbol{p}_3 & \boldsymbol{p}_4 \end{bmatrix}^\top$ ▷ matrix of original points
  for all $s_x \in \{1, 2, \ldots, 8\}$ **do**
    for all $s_y \in \{1, 2, \ldots, 8\}$ **do**
      $\boldsymbol{P'} \leftarrow \begin{bmatrix} 0 & 0 & 1 \\ s_x & 0 & 1 \\ s_x & s_y & 1 \\ 0 & s_y & 1 \end{bmatrix}^\top$ ▷ matrix of warped points
      $\boldsymbol{H} \leftarrow$ homography matrix calculated for $\boldsymbol{P}$ and $\boldsymbol{P'}$ using eq. (4.5)
      $\mathcal{I}_C \leftarrow \emptyset$ ▷ candidate set of inliers for this iteration
      $\mathcal{W}_C \leftarrow \emptyset$ ▷ candidate set of warped points
      $\mathcal{M}_C \leftarrow \emptyset$ ▷ candidate mapping
      for all $\boldsymbol{p} \in \mathcal{P}$ **do** ▷ iterate over all intersection points
        $\boldsymbol{p'} \leftarrow h^{-1}(\boldsymbol{H}\boldsymbol{p})$ ▷ warp the point according to our model
        $\boldsymbol{q} \leftarrow \text{round}(\boldsymbol{p'})$ ▷ snap to grid (round element-wise to nearest whole number)
        if $\boldsymbol{q} \in \mathcal{W}_C$ **then** ▷ ignore this point if we already mapped it
          continue
        end if
        $\epsilon \leftarrow \|\boldsymbol{p'} - \boldsymbol{q}\|_1$ ▷ calculate the error
        if $\epsilon \leq \epsilon_{\max}$ **then** ▷ determine whether we have an inlier
          $\boldsymbol{p} \leftarrow h^{-1}(\boldsymbol{p})$
          $\mathcal{I}_C \leftarrow \mathcal{I}_C \cup \{\boldsymbol{p}\}$ ▷ add the inlier to our candidate mapping
          $\mathcal{W}_C \leftarrow \mathcal{W}_C \cup \{\boldsymbol{q}\}$
          $\mathcal{M}_C \leftarrow \mathcal{M}_C \cup \{(\boldsymbol{p}, \boldsymbol{q})\}$
        end if
      end for
      if $|\mathcal{I}_C| > |\mathcal{I}|$ **then** ▷ update the set of inliers if we found more
        $\mathcal{I}, \mathcal{W}, \mathcal{M} \leftarrow \mathcal{I}_C, \mathcal{W}_C, \mathcal{M}_C$
      end if
    end for
  end for
  if $i \geq i_{\max}$ **then break end if** ▷ guard against iterating indefinitely
  $i \leftarrow i + 1$
end while

be defined as

$$h(x, y) = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

and

$$h^{-1}(x, y, z) = \begin{bmatrix} x/z \\ y/z \end{bmatrix}$$

provided that $z \neq 0$.

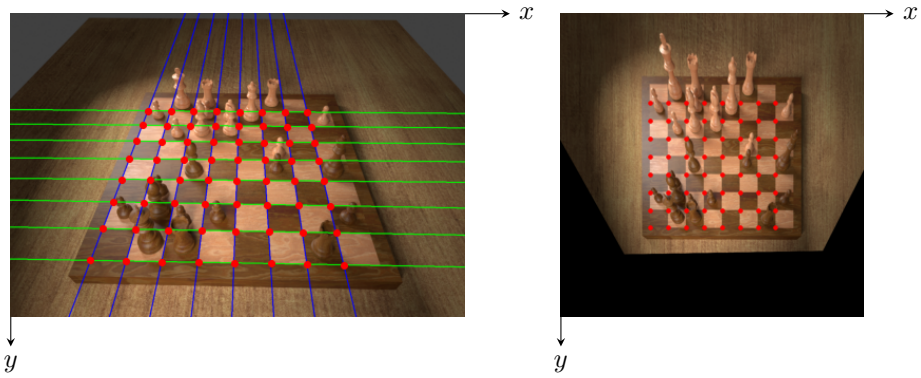
The input to algorithm 1 is the set of intersection points $\mathcal{P} \subset \mathbb{R}^2$ in the original image. Its output is a bijection $m : \mathcal{I} \to \mathcal{W}$ from a set of inlier points $\mathcal{I} \subseteq \mathcal{P}$ to a set of warped points $\mathcal{W} \subset \mathbb{Z}^2$. Notice that the warped points are represented by two-dimensional vectors whose components are whole numbers, meaning that they lie on a grid of whole numbers. Each square on the chessboard is represented by a square of unit length on the grid. The chessboard itself is represented by a 8×8 square on that grid. However, since the set of inliers will likely not contain all vertices of the chessboard squares (for example because some lines were not recognised as seen in figs. 4.2(e) and 4.6) we do not yet know the origin of that 8×8.

Once we have computed the output of algorithm 1 and obtained the mapping m of points from the original image to points on the grid of the warped image, we can recompute the homography matrix $\boldsymbol{H}$ using eq. (4.5) using *all* inliers to obtain a more accurate homography that is based on more than just four points. To do so, we take the matrix of original points $\boldsymbol{P}$ to be a $3 \times |\mathcal{I}|$ matrix whose columns are populated by all the intersection points $\mathcal{I}$ as homogenous coordinate vectors. The matrix of warped points $\boldsymbol{P'}$ of the same size will be populated by the corresponding points in warped space $\mathcal{W}$ according to the bijection m. Using the new $\boldsymbol{P}$ and $\boldsymbol{P'}$ matrices, eq. (4.5) becomes an overdetermined linear system, so we will compute $\boldsymbol{H}$ as the least squared error solution instead of the exact solution. Finally, we can use the homography matrix to project the pixels of the original image to obtain a warped image as in fig. 4.7.

4.1.2.3 Optimisations

Algorithm 1 aims to outline the method of determining the inlier points in a manner that is easy to follow. In practise, the employed algorithm will be different in order to achieve improved performance. Implementing these differences in algorithm 1 would have made it too convoluted to follow, so they shall instead briefly be summarised below:

- The for loop on line 16 can be vectorised and replaced by matrix operations.
- The computation of the homography matrix on line 12 occurs 64 times per iteration of the outer loop, due to the two for loops on lines 9 and 10. Instead, it should be moved before the two aforementioned for loops and computed for the scale $s_x = s_y = 1$. To ensure the points are mapped correctly, we must add the instruction $\boldsymbol{p'} \leftarrow \boldsymbol{p'} \odot \begin{bmatrix} s_x & s_y \end{bmatrix}^\top$ immediately after line 17 in order to scale the warped points.
- The for loops on lines 9 and 10 do not need to be nested; instead, one could first consider only the x-component of the vectors when calculating



(a) original image with intersection points

(b) warped image

Figure 4.7: The original image is warped using the computed homography matrix $\boldsymbol{H}$. Detected lines and intersection points are overlaid in the original image as per fig. 4.2(e). The corresponding intersection points are marked in red in the warped image and lie on a regular unit-length grid.

the errors for each scale s_x, and then do the same for the y-components and s_y. Finally, we would choose the best s_x and s_y (with the least accumulated error), compute the joint error, and use the information to update the mappings if necessary. This is the reason for using the ℓ_1 norm for computing the error on line 22, since it can be easily decomposed into the two directional components.

4.1.3 Inferring missing lines

It was noted at the end of section 4.1.2.2 that while the result of algorithm 1 projects the inlier intersection points onto a regular grid of whole numbers, that result does not explicitly indicate the location of the 8×8 sub-grid that represents the chessboard. To understand the set of inliers, we shall first compute the minimum and maximum x and y values of the warped points. We shall use the notation

$$x_{\min} = \min \mathcal{W}_x, \qquad x_{\max} = \max \mathcal{W}_x, \tag{4.6}$$

$$y_{\min} = \min \mathcal{W}_y, \qquad y_{\max} = \max \mathcal{W}_y, \tag{4.7}$$

where

$$\mathcal{W}_x = \left\{ x : \begin{bmatrix} x & y \end{bmatrix}^\top \in \mathcal{W} \right\} \qquad \text{and} \qquad \mathcal{W}_y = \left\{ y : \begin{bmatrix} x & y \end{bmatrix}^\top \in \mathcal{W} \right\}. \tag{4.8}$$

The $x_{\min}$ value describes the leftmost vertical line in the warped image given by the equation $x = x_{\min}$, and the value of $x_{\max}$ describes the rightmost vertical line. Similarly, the values of $y_{\min}$ and $y_{\max}$ represent the top and bottom lines. Examining the range of x-values ($x_{\max} - x_{\min}$) gives an indication of whether the vertical lines we identified represent the whole chessboard, part of it, or even a range that is wider than the chessboard itself, horizontally. We thus distinguish between three cases based on the x-values which we shall outline below.

Case 1. $x_{\max} - x_{\min} = 8$.

In this case, no action is required because we detected the two outer vertical lines. The leftmost vertical line of the chessboard is located at $x = x_{\min}$ and the rightmost line is at $x = x_{\max}$ on the warped image.

Case 2. $x_{\max} - x_{\min} > 8$.

This case arises when the vertical line $x = x_{\min}$ is left of the left edge of the chessboard and/or the vertical line $x = x_{\max}$ is right of the right edge of the chessboard. In other words, the range of x-values covers more than just the chessboard. To remedy this, we will iteratively update $\mathcal{W}_x$ by applying

$$\mathcal{W}_x \leftarrow \mathcal{W}_x \setminus \{x_{\min}, x_{\max}\}$$

and recomputing $x_{\min}$ and $x_{\max}$ by eq. (4.6) until $x_{\max} - x_{\min} \leq 8$. Then, we update the set of warped points $\mathcal{W}$ as

$$\mathcal{W} = \left\{ \begin{bmatrix} x & y \end{bmatrix}^\top : \begin{bmatrix} x & y \end{bmatrix}^\top \in \mathcal{W},\ x \in \mathcal{W}_x \right\},$$

retaining only the warped points whose x-component is in the interval $[x_{\min}, x_{\max}]$. We also remove the corresponding points from $\mathcal{I}$ and the mapping m's graph $\mathcal{M}$. Then, we fall either in case 1 or 3, depending on the values of $x_{\min}$ and $x_{\max}$.

Case 3. $x_{\max} - x_{\min} < 8$.

This most common scenario arises when the grid that should represent the chessboard in the warped image is smaller than eight units in width. Figure 4.7 illustrates this case makes it clear that it arises because either one or both of the outermost vertical lines were not detected on the chessboard. Determining whether the grid should be expanded towards the left or the right involves futher processing of the image which will be outlined below.

We will calculate the horizontal intensity gradients in the warped image for each pixel in order to determine whether an additional vertical line is more likely to occur one unit to the left or one unit to the right of the currently identified grid. To do so, we first convert the RGB image to grayscale and then apply the horizontal Sobel filter which is a discrete differentiation operator that provides an approximation for the gradient intensity in the horizontal direction. Large horizontal gradient intensities will give rise to vertical lines in the warped image which can be used in order to determine whether the chessboard should be expanded to the left or right.

The Sobel filter is widely used in image processing and consists of two kernels that are convolved over the input image, a horizontal and a vertical one. Typically, the results of both convolution operations is calculated, and then some simple trigonometry can be applied in order to determine the gradient directions for each pixel. However, since we are only interested in determining the gradients in the horizontal direction, it suffices to use the horizontal Sobel kernel

$$\boldsymbol{S}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}. \tag{4.9}$$

Given an intensity image (i.e. a grayscale image) of dimensions $h \times w$ that is represented by the matrix $\boldsymbol{A} \in \mathbb{R}^{h \times w}$, we compute the horizontal

gradient intensity at each pixel by

$$\boldsymbol{G}_x = \boldsymbol{S}_x * \boldsymbol{A} \tag{4.10}$$

where $\boldsymbol{G}_x$ is of the same dimensionality as $\boldsymbol{A}$ and contains the gradient intensities. Then, we can apply the Canny edge detector (as we did in section 4.1.1) to image represented by $\boldsymbol{G}_x$ in order to eliminate noise and obtain clear edges. Figure 4.8 shows the results of these operations on the running example.

On that image we shall sum the pixel internsities on the vertical lines at $x = x_{\min} - 1$ and $x = x_{\max} + 1$ (with a small tolerance to the left and to the right). Then, if the sum of pixel intensities was greater at $x = x_{\min} - 1$ than $x = x_{\max} + 1$, we will update $x_{\min} \leftarrow x_{\min} - 1$, or otherwise $x_{\max} \leftarrow x_{\max} + 1$. We repeat this process until $x_{\max} - x_{\min} = 8$.

Although the three cases above examine only the x-values to locate the horizontal position of the chessboard, it is trivial to see how this method applies to determine the vertical position of the chessboard using the y-values, too. However, the main caveat the reader should acknowledge is that we must use the vertical sobel kernel

$$\boldsymbol{S}_x = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}. \tag{4.11}$$

instead of the horizontal one given in eq. (4.9), and as a consequence eq. (4.10) becomes

$$\boldsymbol{G}_y = \boldsymbol{S}_y * \boldsymbol{A}. \tag{4.12}$$

After following the instructions according to the three cases above for both the horizontal and vertical directions, we will have obtained the $x_{\min}, x_{\max}, y_{\min}, y_{\max}$ values that describe the location of the chessboard in the warped image. As such, the coordinates of the four corners of the chessboard on the warped image can be expressed as

$$\begin{aligned} \boldsymbol{p}'_1 &= \begin{bmatrix} x_{\min} & y_{\min} \end{bmatrix}^\top & \boldsymbol{p}'_2 &= \begin{bmatrix} x_{\max} & y_{\min} \end{bmatrix}^\top \\ \boldsymbol{p}'_3 &= \begin{bmatrix} x_{\max} & y_{\max} \end{bmatrix}^\top & \boldsymbol{p}'_4 &= \begin{bmatrix} x_{\min} & y_{\max} \end{bmatrix}^\top. \end{aligned}$$

The points are supplied in clockwise order, starting from the top left.

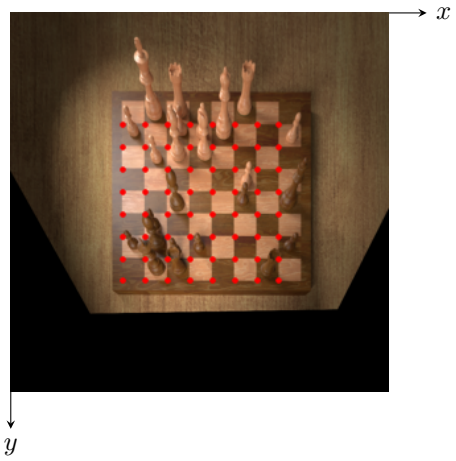
Finally, we can use the inverse of the homography matrix in order to project these points back onto the original image by inverting eq. (4.4). Notice that the points must be converted to homogenous coordinates and back again, so the equation becomes

$$\boldsymbol{p}_i = h^{-1}\left(\boldsymbol{H}^{-1} h(\boldsymbol{p}'_i)\right) \tag{4.13}$$

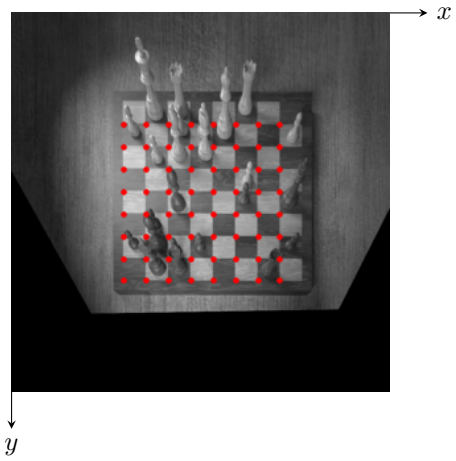
for $i = 1, 2, 3, 4$. The result is depicted in fig. 4.9.

4.1.4 Determining optimal parameters

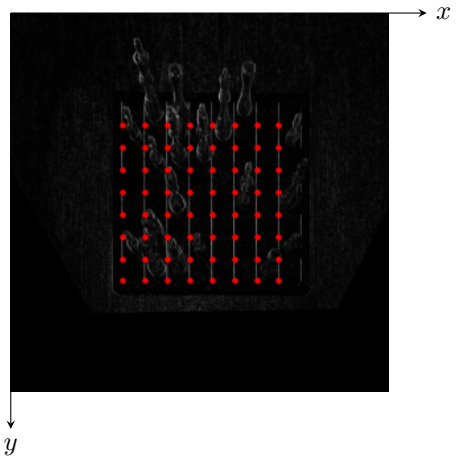
The method of finding the four corner points of the chessboard described in the previous sections relies on numerous parameters whose optimal values are yet to be determined. The most important parameters are summarised below:



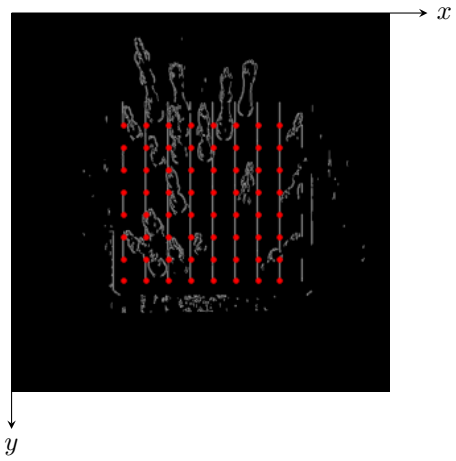
(a) warped image from fig. 4.7(b)



(b) conversion to grayscale

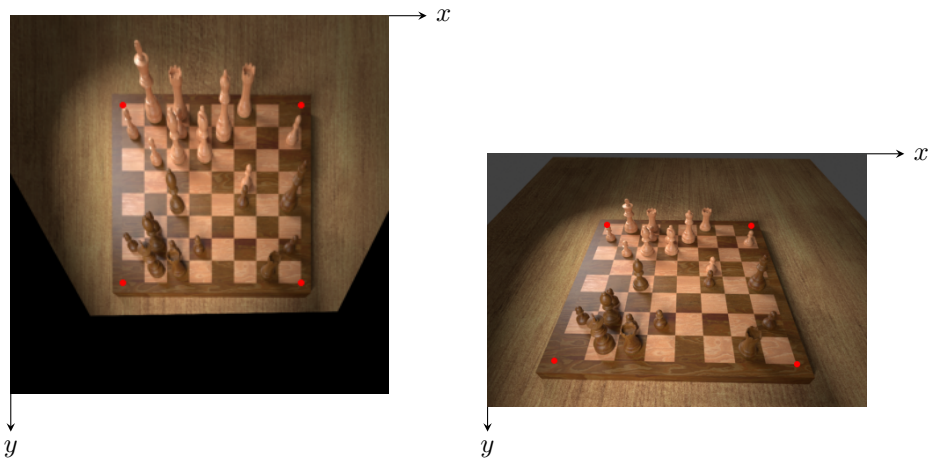


(c) horizontal gradient intensities as computed by the Sobel operator (the colour scale shows low values in black and high values in white)



(d) result of Canny edge detection on the horizontal gradient image

Figure 4.8: Horizontal gradient intensities calculated on the warped image in order to detect vertical lines. The red dots correspond to the intersection points in $\mathcal{I}$. Here, $x_{\max} - x_{\min} = 7$ because there are eight columns of points instead of nine. This is because the rightmost vertical line was not detected.



(a) warped image with detected corner points $\boldsymbol{p}'_i$

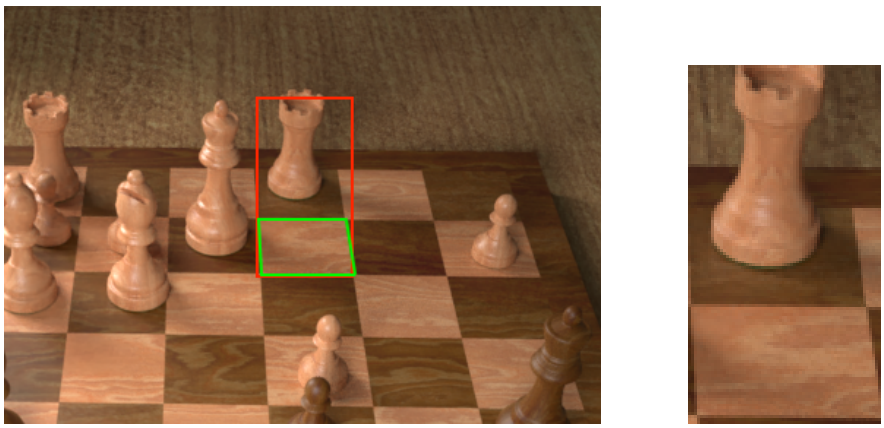
(b) original image with reprojected corner points $\boldsymbol{p}_i$

Figure 4.9: The identified corner points are projected back from the warped image onto the original image using the inverse homography matrix.

- low and high thresholds of the final stage of the Canny edge detector (hysteresis);
- threshold t of the Hough transform as explained in section 4.1.1;
- minimum number of iterations $i_{\min}$, maximum number of iterations $i_{\max}$, minimum number of inliers c, and the inlier threshold $\epsilon_{\max}$ from algorithm 1;
- low and high Canny edge detection thresholds for the horizontal and vertical gradient images, as well as the relative width of the horizontal and vertical lines from section 4.1.3.

The optimal parametervalues are decided by performing a grid search using sensible presets. A corner is classified as being detected accurately if the prediction is within ten pixels of the groundtruth label, which given the fact that the width of the image is 1,200 pixels ensures that the predictions must be very close to the groundtruth. If at least one corner is detected inaccurately, that particular sample is determined to be inaccurate. We perform the grid search only on a small subset of the training set because the number of different parameter sets to be tested is in the order of a thousand. The set of parameters achieving the best performance on this subset of the training set is retained as the final configuration.

On the entire training set of 4,400 samples, the corner detection algorithm yielded inaccurate predictions in 13 cases, so its accuracy is 99.71%. The validation set of size 146 sees no mistakes and thus achieves an accuracy of 100.00%. As such, we conclude that the corner detection algorithm is very robust to unseen samples from the dataset and did not overfit as a result of the grid search.



(a) region from the original image with marked square

(b) cropped sample

Figure 4.10: An example illustraing why an immediate piece classification approach is inclined to reporting false positives. Consider the square marked in green in the original image (a). The bounding box for piece classification (marked in red) must be quite tall because the square might contain a tall piece such as a queen or king (the box must be at least as tall as the the queen in the adjacent square on the left). The resulting sample, depicted in (b), contains almost the entire rook of the square behind. Thus, a piece classifier might classify this square as containing a rook instead of being empty.

4.2 Occupancy classification

Empirical experiments showed that performing piece classification directly after detecting the four corner points with no intermediate step yields a large number of false positives, i.e. empty squares being classified as containing a chess piece. One common scenario where the trained classifier failed is illustrated in fig. 4.10. Notice that squares further away from the camera must be cropped with increasingly taller bounding boxes. If a particular square is empty but its bounding box includes the piece from the adjacent square as in fig. 4.10(b), the trained classifier was inclined to report a false positive.

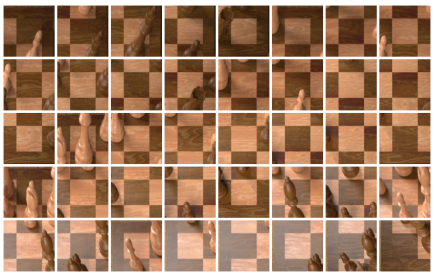
To solve this problem, a binary classifier is trained on cropped squares to decide whether they are empty or not. Before cutting out the squares from the original image, the input image is warped to a two-dimensional overhead view by means of a projective transformation. This ensures that all squares are of equal size and that the corners form right angles (which is not the case in the original image due to perspective distortion).

Finally, we can use $\boldsymbol{H}$ to apply the projective transformation to the original image such as depicted in fig. 4.11(a), obtaining the warped image in fig. 4.11(b).

Cropping the squares from the warped image is trivial because the squares are of equal size. In training the occupancy classifiers, it is conjectured that it would be useful to include contextual information with each square; therefore, the squares are not cropped tightly around their boundaries but instead with a 50% increase in length on all four sides, as shown in figs. 4.11(c) and 4.11(d). This might aid the classifier's decision in difficult situations where a chess piece from another square reaches into the cropped one due to the camera perspective.

(a) original

(b) warped



(c) all 32 empty samples

(d) all 24 occupied samples

Figure 4.11: The process of obtaining samples for occupancy classification from a chessboard image. First, the original image (a) is warped to a two-dimensional overhead view, (b). Then, all squares are cropped (with a 50% increase in width and height to include contextual information). Finally, the cropped squares are annoted using the FEN groundtruth as either empty (c) or occupied (d).

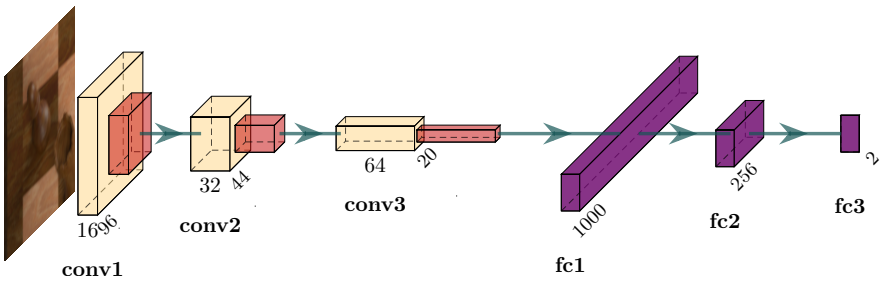


Figure 4.12: Architecture of the CNN $(100, 3, 3, 3)$ network for occupancy classification. The input is a three-channel RGB image with 100×100 pixels. There are two convolutional layers (yellow) with a kernal size of 5×5 and stride 1, meaning that each convolutional layer reduces the width and height by 4. The final convolutional layer has a kernel size of 3×3, thus only reducing the input size by two. Starting with 16 filters in the first convolutional layer, the number of channels is doubled in each subsequent layer, as is common practice in CNNs [51]. Each convolutional layer uses the ReLU activation function and is followed by a max pooling layer with a 2×2 kernel that is moved with a stride of 2 such that the width and height are halved. Finally, the output of the last pooling layer is reshaped to a 640,000-dimensional vector that passes through two fully connected ReLU-activated layers before reaching the final fully connected layer with softmax activation.

4.2.1 Training CNNs

Six CNN architectures are devised for the occupancy classification task, of which two accept 100×100 pixel input images and the remaining four require the images to be of size 50×50 pixels. They differ in the number of convolution layers, pooling layers, and fully connected layers. When referring to these models, we shall use a 4-tuple consisting of the input side length and the three aforementioned criteria. Figure 4.12 depicts the architecture of the CNN $(100, 3, 3, 3)$ model which achieves the greatest validation accuracy of these six models. The final fully connected layer in each model contains two output units that represent the two classes (occupied and empty). The models are trained using the cross-entropy loss function on the outputs. Training proceeds using the popular *Adam* optimizer [41] with a learning rate of 0.001 for three whole passes over the training set using a batch size of 128. After every 100 steps, the model's loss and accuracy is computed over the entire validation set, the results of which are reported in fig. 4.13. The model converges smoothly to a very low loss value, achieving a training accuracy of 99.70% and validation accuracy of 99.71%. Due to the fact that the difference between training and validation accuracy is very small (in fact, the validation accuracy even happens to be slightly above the training accuracy), we conclude that the model does not overfit the training set.

4.2.2 Transfer learning on deeper models

VGG [51] ResNet [52] AlexNet [33] ImageNet [53]

todo: explain how this was done and explain tradeoff (significantly more parameters in model)

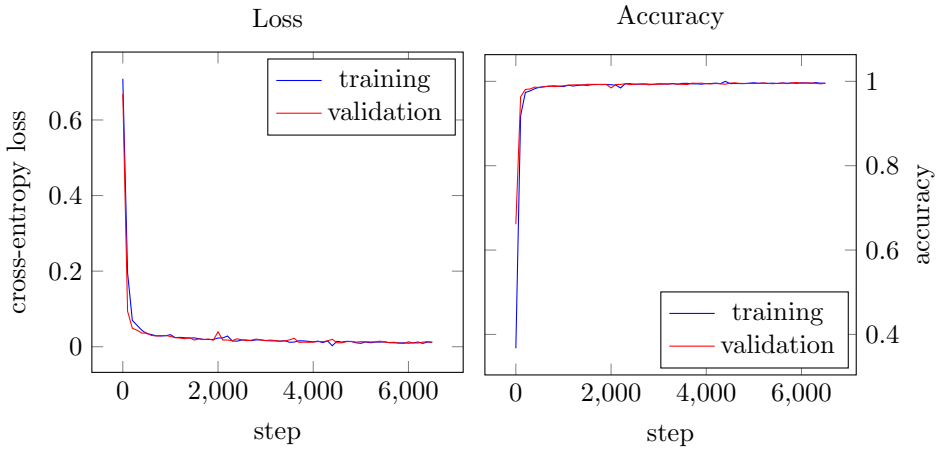


Figure 4.13: Loss and accuracy during training on both the training and validation sets for the CNN $(100, 3, 3, 3)$ model. The best validation accuracy is 99.71%.

4.2.3 Analysis

Each model is trained separately on the dataset of squares that are cropped to include contextual information (by increasing the bounding box by 50% in each direction), and the same samples except that the squares are cropped tightly. In each case, the model trained on the samples that contained contexual information outperfomed its counterpart trained on tightly cropped samples, confirming the hypothesis that the information around the square itself is useful.

4.3 Piece classification

Now that the occupancy of each square on the chessboard can be detected to a high degree of accuracy, the next step is to classify the piece in each of the occupied squares. We require a 12-way classifier that takes as input a cropped image of an occupied square and will output the chess piece on that square. There are six types of chess pieces (pawn, knight, bishop, rook, queen, and king), and each piece can either be white or black in colour, thus there are 12 classes of pieces.

We must pay some special attention to how the pieces are cropped. Simply following the approach described in section 4.2 for cropping squares does not give enough information to classify pieces. Especially tall pieces at the back of the board would be cropped in a manner such that only the bottom part of the piece remains in the region of interest (ROI). Consider for example the white king in fig. 4.11. Cropping only the square it is located in would not include its crown which is an important feature needed to distinguish between kings and queens. Instead, we must choose a rectangular bounding box that is tall enough to account for the perspective distortion.

The camera perspective causes another phenomenon that we must account for: pieces on the left of the board tend to 'slant' to the left, and vice-versa on the right. This is due to the vanishing point of the normal vectors on the chessboard surface being roughly centred horizontally with regards to the location of the

	model	parameters	accuracy	precision	recall	errors	training accuracy
✓	ResNet [52]	$1.12 \cdot 10^7$	99.96%	1.000	0.999	4	99.93%
✓	VGG [51]	$1.29 \cdot 10^8$	99.95%	0.999	0.999	5	99.96%
✗	VGG [51]	$1.29 \cdot 10^8$	99.94%	0.999	0.999	6	99.93%
✗	ResNet [52]	$1.12 \cdot 10^7$	99.90%	0.999	0.998	9	99.94%
✓	AlexNet [33]	$5.7 \cdot 10^7$	99.80%	0.998	0.996	19	99.74%
✗	AlexNet [33]	$5.7 \cdot 10^7$	99.76%	0.998	0.995	22	99.76%
✓	CNN $(100, 3, 3, 3)$	$6.69 \cdot 10^6$	99.71%	0.997	0.995	27	99.70%
✓	CNN $(100, 3, 3, 2)$	$6.44 \cdot 10^6$	99.70%	0.996	0.995	28	99.70%
✗	CNN $(100, 3, 3, 2)$	$6.44 \cdot 10^6$	99.64%	0.996	0.993	34	99.61%
✓	CNN $(50, 2, 2, 3)$	$4.13 \cdot 10^6$	99.59%	0.993	0.995	38	99.62%
✓	CNN $(50, 3, 1, 2)$	$1.86 \cdot 10^7$	99.56%	0.997	0.991	41	99.67%
✓	CNN $(50, 3, 1, 3)$	$1.88 \cdot 10^7$	99.56%	0.993	0.994	41	99.66%
✓	CNN $(50, 2, 2, 2)$	$3.88 \cdot 10^6$	99.54%	0.993	0.994	43	99.64%
✗	CNN $(50, 2, 2, 3)$	$4.13 \cdot 10^6$	99.52%	0.993	0.993	45	99.57%
✗	CNN $(100, 3, 3, 3)$	$6.69 \cdot 10^6$	99.50%	0.988	0.997	47	99.55%
✗	CNN $(50, 3, 1, 2)$	$1.86 \cdot 10^7$	99.50%	0.993	0.992	47	99.44%
✗	CNN $(50, 2, 2, 2)$	$3.88 \cdot 10^6$	99.44%	0.995	0.989	52	99.54%
✗	CNN $(50, 3, 1, 3)$	$1.88 \cdot 10^7$	99.39%	0.993	0.989	57	99.41%

Table 4.1: Performance of all occupancy classification models on the validation set. For the CNN models, the 4-tuple denotes the length of the square input size in pixels, the number of convolution layers, the number of pooling layers, and the number of fully connected layers. The check mark in the left column indicates whether the input samples contained contextual information (cropped to include part of the adjacent squares). In the penultimate column, the total number of misclassifications in the validation set are reported (the validation set consists of 9,346 samples). The training accuracy is given in the rightmost column for comparison. Notice that there is no significant difference between the validation and training accuracies, indicating that none of the models suffer from overfitting.

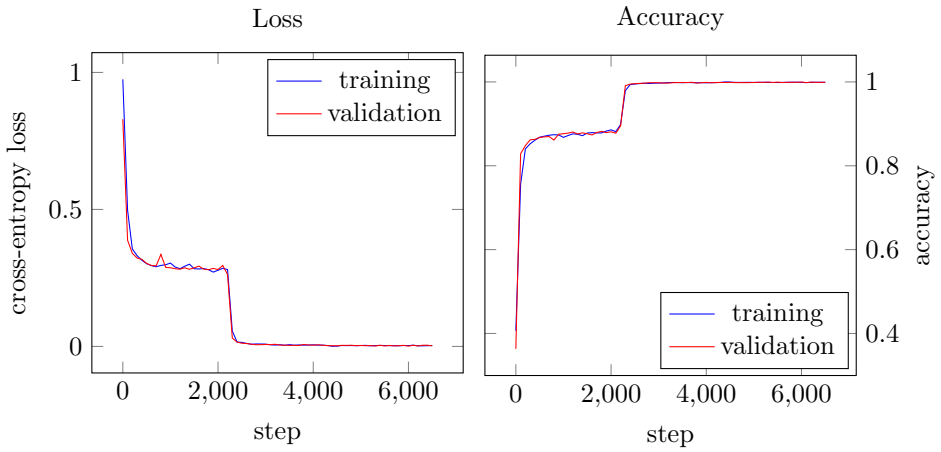


Figure 4.14: Loss and accuracy during training on both the training and validation sets for the ResNet model. The best validation accuracy is 99.96%.

Figure 4.15: The four samples that the ResNet model misclassified in the validation set. The left sample depicts an empty square, and the remaining three are of occupied squares.

chessboard itself[1], as illustrated in fig. 4.16. Hence, we must extend the ROI vertically in the appropriate direction.

To obtain the ROIs, we first warp the input image of the chessboard as described in section 4.2 and exemplified in fig. 4.11(b). At first, each pieces bounding box will correspond to the square it is located on, i.e. its width and height will be that of the square[2]. Depending on the rank r, the height is increased by

$$h_{\text{inc}}(r) = \frac{2r+5}{7}$$

where the unit of measurement is the height of the chessboard squares. This represents an arithmetic progression such that the bounding box for pieces in the first rank (bottom row of the chessboard, i.e. $r = 1$) will be increased by one square, and pieces in the top row ($r = 8$) will have their height increased by three units.

The increase in width is dependent on the file f and given by the piecewise-defined arithmetic progression

$$w_{\text{inc}}(f) = \begin{cases} -\frac{f}{4} & f \leq 4 \\ \frac{f-4}{4} & f > 4. \end{cases}$$

A negative increase in width means that the bounding box is extended to the left, whereas a positive increase means it is widened to the right. Thus, pieces on the left half of the board ($1 \leq f \leq 4$) will have their bounding boxes extended to the left, and vice-versa on the right (for $4 < f \leq 8$).

Experiments demonstrate that this heuristic generates bounding boxes that are large enough to contain even the tallest pieces in most cases, whilst not being needlessly large. As a further preprocessing step, the cropped ROIs for all pieces on the left side of the board ($1 \leq f \leq 4$) are flipped vertically, such that the square that the piece stands on will always be in the bottom left of the image. This may help the classifier understand what piece is being referred to in samples where the larger bounding box includes adjacent pieces in the image.

[1]Unfortunately, it is not possible to compute the vanishing point of the normals based solely on the information available in the input images as this would require knowledge of the intrinsic and extrinsic camera parameters [54]. The aim of this work is to recognise chess positions from just the input image without any further information; therefore, we devise a heuristic based on the observation that the vanishing point tends to be vertically below the chessboard and roughly horizontally centred.

[2]Note that due to the warped perspective, all squares have equal width and height.

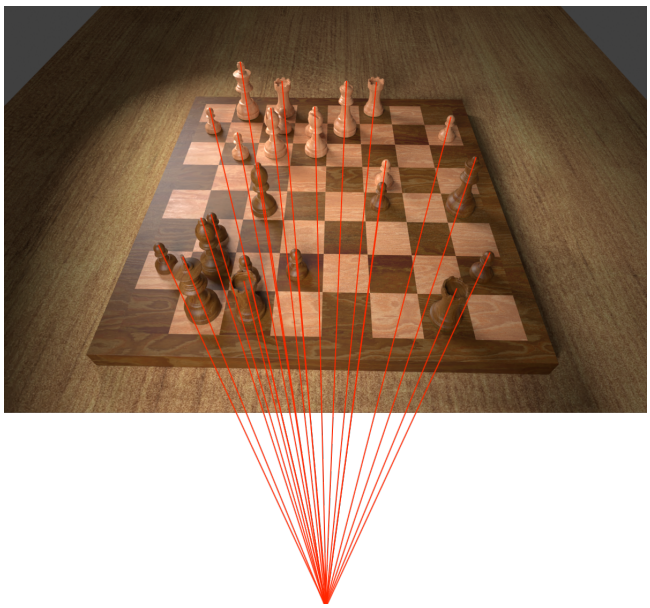


Figure 4.16: The normals of the chessboard surface (corresponding to the direction the pieces are pointing, marked in red) converge to a single vanishing point which is below the image. We will assume that the vanishing point is roughly horizontally centred with the chessboard, as this corresponds to how chessboards are usually photographed. As a result, pieces on left 'lean' left, and vice-versa on the right.

Figure 4.17 shows a random selection of the ROIs corresponding to white queens in order to demonstrate that the bounding boxes are large enough to contain even tall pieces.

4.3.1 Training CNNs

Similar to section 4.2, we train several models on

talk about change in number of epochs etc

4.4 Preparing the results

model	parameters	accuracy	errors	training accuracy
InceptionV3 [55]	$2.44 \cdot 10^7$	100.00%	0	99.98%
VGG [51]	$1.29 \cdot 10^8$	99.94%	2	99.84%
ResNet [52]	$1.12 \cdot 10^7$	99.91%	3	99.93%
AlexNet [33]	$5.71 \cdot 10^7$	99.02%	31	99.51%
CNN $(100, 3, 3, 2)$	$1.41 \cdot 10^7$	96.94%	97	99.62%
CNN $(100, 3, 3, 3)$	$1.44 \cdot 10^7$	96.90%	98	99.49%

Table 4.2: Performance of all piece classifiers on the validation set.

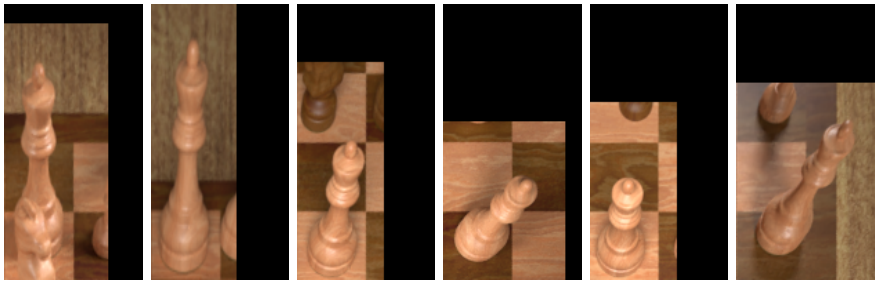


Figure 4.17: A random selection of six samples of white queens in the training set. Notice that the square each queen is located on is always in the bottom left of the image and of uniform dimensions across all samples.

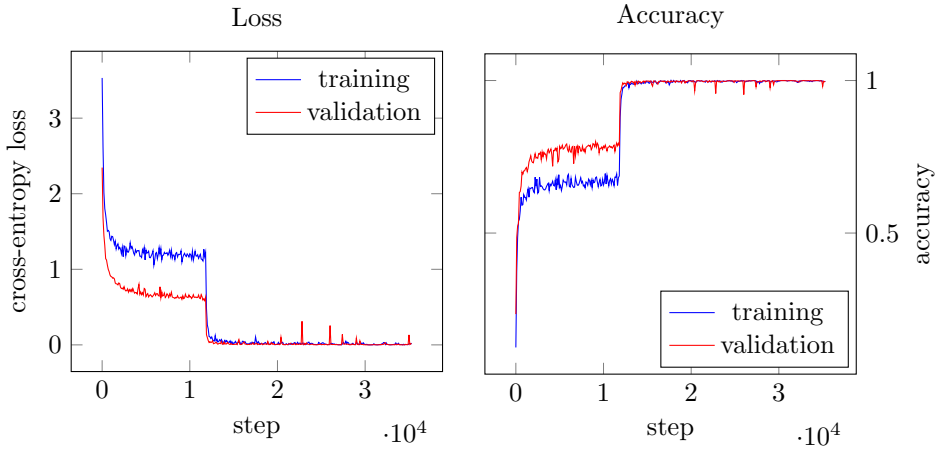


Figure 4.18: Loss and accuracy during training on both the training and validation sets for the InceptionV3 model. The best validation accuracy is 100.00%.

Life is a kind of chess, in which we have often points to gain, and competitors or adversaries to contend with.

Benjamin Franklin

5 Implementation

5.1 Overview

Explain the use of:

- Repositories
- CI/CD pipelines
- Testing

5.2 Data synthesis

5.3 Configuration management

the Python Package Index (PyPI)

5.4 Board localisation

5.5 Training the classifiers

5.6 Transfer learning

5.7 Web app

Chess is the struggle against the error.

Johannes Zukertort

6 Evaluation

6.1 Dataset

6.2 Critical appraisal

You may learn much more from a game you lose than from a game you win.

José Capablanca

7 Conclusion

7.1 Future work

Acronyms

ANN artificial neural network. 5, 6, 8–10

CAD computer-aided design. 4

CNN convolutional neural network. 3, 4, 6, 20, 36–38, 40

DAG directed acyclic graph. 9, 11

FEN Forsyth–Edwards Notation [3]. 5, 15, 16, 20, 35

HOG histogram of oriented gradients. 3

MLP multi-layer perceptron. v, 11, 12, 14

MSE mean squared error. 12

OHE one-hot encoding. 7

PMF probability mass function. 7

PyPI the Python Package Index. 42

RANSAC random sample consensus. 25–27

ReLU rectified linear unit. 9, 36

RGB red, green, blue. 20, 21, 30, 36

ROI region of interest. 37, 39, 40

SIFT scale-invariant feature transform. 3

SLP single-layer perceptron. v, 10, 11, 13, 14

SVM support vector machine. 2, 3

Bibliography

[1] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016.

[2] G. Kasparov and M. Greengard, *Deep Thinking: Where Machine Intelligence Ends and Human Creativity Begins*. 2018, ISBN: 978-1-4736-5351-1.

[3] S. J. Edwards, *PGN Standard*. Mar. 1994. [Online]. Available: `http://archive.org/details/pgn-standard-1994-03-12`.

[4] H. Baird and K. Thompson, 'Reading chess,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 6, pp. 552–559, Jun. 1990.

[5] I. M. Khater, A. S. Ghorab and I. A. Aljarrah, 'Chessboard recognition system using signature, principal component analysis and color information,' in *International Conference on Digital Information Processing and Communications*, Jul. 2012, pp. 141–145.

[6] A. Sameer, *Tensorflow_chessbot*, Sep. 2020. [Online]. Available: `https://github.com/Elucidation/tensorflow_chessbot`.

[7] A. Roy, *Chessputzer*, Jul. 2020. [Online]. Available: `https://github.com/metterklume/chessputzer`.

[8] V. Wang and R. Green, 'Chess move tracking using overhead RGB webcam,' in *International Conference on Image and Vision Computing New Zealand*, Nov. 2013, pp. 299–304.

[9] T. Cour, R. Lauranson and M. Vachette, 'Autonomous Chess-playing Robot,' École Polytechnique, Palaiseau, France, Jul. 2002. [Online]. Available: `http://www.timotheecour.com/papers/ChessAutonomousRobot.pdf`.

[10] D. Urting and Y. Berbers, 'MarineBlue: A low-cost chess robot,' in *International Conference Robotics and Applications*, Salzburg, Austria, Jun. 2003, pp. 76–81.

[11] 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*, vol. 58, Bhubaneshwar, India: Interscience Research Network, Aug. 2012, pp. 17–22.

[12] A. T.-Y. Chen and K. I.-K. Wang, 'Computer vision based chess playing capabilities for the Baxter humanoid robot,' in *International Conference on Control, Automation and Robotics*, Apr. 2016, pp. 11–14.

[13] J. Gonçalves, J. Lima and P. Leitão, 'Chess robot system : A multi-disciplinary experience in automation,' in *Spanish Portuguese Congress on Electrical Engineering*, 2005.

[14] R. A. M. Khan and R. Kesavan, 'Design and development of autonomous chess playing robot,' *International Journal of Innovative Science, Engineering & Technology*, vol. 1, no. 1, 2014.

[15] A. T.-Y. Chen and K. I.-K. Wang, 'Robust Computer Vision Chess Analysis and Interaction with a Humanoid Robot,' *Computers*, vol. 8, no. 1, p. 14, 1 Mar. 2019.

[16] C. Matuszek, B. Mayton, R. Aimi, M. P. Deisenroth, L. Bo, R. Chu, M. Kung, L. LeGrand, J. R. Smith and D. Fox, 'Gambit: An autonomous chess-playing robotic system,' in *IEEE International Conference on Robotics and Automation*, May 2011, pp. 4291–4297.

[17] E. Sokic and M. Ahic-Djokic, 'Simple Computer Vision System for Chess Playing Robot Manipulator as a Project-based Learning Example,' in *IEEE International Symposium on Signal Processing and Information Technology*, Dec. 2008, pp. 75–79.

[18] J. Hack and P. Ramakrishnan, 'CVChess: Computer Vision Chess Analytics,' Stanford University, 2014. [Online]. Available: `https://web.stanford.edu/class/cs231a/prev_projects_2015/chess.pdf`.

[19] J. Ding. (2016). 'ChessVision : Chess Board and Piece Recognition,' [Online]. Available: `https://web.stanford.edu/class/cs231a/prev_projects_2016/CS_231A_Final_Report.pdf`.

[20] C. Danner and M. Kafafy. (2015). 'Visual Chess Recognition,' [Online]. Available: `https://web.stanford.edu/class/ee368/Project_Spring_1415/Reports/Danner_Kafafy.pdf`.

[21] Y. Xie, G. Tang and W. Hoff, 'Chess Piece Recognition Using Oriented Chamfer Matching with a Comparison to CNN,' in *IEEE Winter Conference on Applications of Computer Vision*, Mar. 2018, pp. 2001–2009.

[22] A. De la Escalera and J. M. Armingol, 'Automatic Chessboard Detection for Intrinsic and Extrinsic Camera Parameter Calibration,' *Sensors*, vol. 10, no. 3, pp. 2027–2044, 3 Mar. 2010.

[23] S. Bennett and J. Lasenby, 'ChESS – Quick and robust detection of chessboard features,' *Computer Vision and Image Understanding*, vol. 118, pp. 197–210, Jan. 2014.

[24] K. Tam, J. Lay and D. Levy, 'Automatic Grid Segmentation of Populated Chessboard Taken at a Lower Angle View,' in *Digital Image Computing: Techniques and Applications*, Dec. 2008, pp. 294–299.

[25] J. E. Neufeld and T. S. Hall, 'Probabilistic location of a populated chessboard using computer vision,' in *IEEE International Midwest Symposium on Circuits and Systems*, Aug. 2010, pp. 616–619.

[26] R. Kanchibail, S. Suryaprakash and S. Jagadish, 'Chess Board Recognition,' 2016. [Online]. Available: `http://vision.soic.indiana.edu/b657/sp2016/projects/rkanchib/paper.pdf`.

[27] Y. Xie, G. Tang and W. Hoff, 'Geometry-based populated chessboard recognition,' in *International Conference on Machine Vision*, vol. 10696, International Society for Optics and Photonics, Apr. 2018, p. 1 069 603.

[28] Y.-A. Wei, T.-W. Huang, H.-T. Chen and J. Liu, 'Chess recognition from a single depth image,' in *IEEE International Conference on Multimedia and Expo*, Jul. 2017, pp. 931–936.

[29] M. A. Czyzewski, A. Laskowski and S. Wasik. (Jun. 2020). 'Chessboard and chess piece recognition with the support of neural networks.' arXiv: 1708.03898.

[30] T. Romstad, M. Costalba and J. Kiiski, *Stockfish*, Sep. 2020. [Online]. Available: https://github.com/official-stockfish/Stockfish.

[31] M. Acher and F. Esnault. (Apr. 2016). 'Large-scale Analysis of Chess Games with Chess Engines: A Preliminary Report.' arXiv: 1607.04186.

[32] A. Mehta and H. Mehta, 'Augmented Reality Chess Analyzer (ARChess-Analyzer): In-Device Inference of Physical Chess Game Positions through Board Segmentation and Piece Recognition using Convolutional Neural Networks,' *Journal of Emerging Investigators*, Jul. 2020.

[33] A. Krizhevsky, I. Sutskever and G. E. Hinton, 'ImageNet classification with deep convolutional neural networks,' *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[34] M. A. Czyzewski, A. Laskowski and S. Wasik, 'LATCHESS21: Dataset of damaged chessboard lattice points (chessboard features) used to train LAPS detector (grayscale/21x21px),' RepOD, 2018.

[35] J. Hou, 'Chessman Position Recognition Using Artificial Neural Networks.'

[36] A. Burkov, *The Hundred-Page Machine Learning Book*. Quebec, Canada: Andriy Burkov, 2019.

[37] W. S. McCulloch and W. Pitts, 'A logical calculus of the ideas immanent in nervous activity,' *Bulletin of Mathematical Biophysics*, Dec. 1943.

[38] X. Glorot, A. Bordes and Y. Bengio, 'Deep Sparse Rectifier Neural Networks,' in *International Conference on Artificial Intelligence and Statistics*, JMLR Workshop and Conference Proceedings, Jun. 2011.

[39] S. J. Russell, P. Norvig and E. Davis, *Artificial Intelligence: A Modern Approach*, 3rd. Upper Saddle River: Prentice Hall, 2010.

[40] Y. LeCun, Y. Bengio and G. Hinton, 'Deep learning,' *Nature*, vol. 521, no. 7553, pp. 436–444, 7553 May 2015.

[41] D. P. Kingma and J. Ba. (Jan. 2017). 'Adam: A Method for Stochastic Optimization.' arXiv: 1412.6980.

[42] M. Bilalić, R. Langner, M. Erb and W. Grodd, 'Mechanisms and neural basis of object and pattern recognition: A study with chess experts,' *Journal of Experimental Psychology*, vol. 139, no. 4, pp. 728–742, 2010.

[43] Q. Zhou. (May 2018). 'Pattern recognition in chess,' ChessBase, [Online]. Available: https://en.chessbase.com/post/pattern-recognition-in-chess.

[44] 64 Squares. (Feb. 2020). 'Magnus Carlsen Chess Games,' PGN Mentor, [Online]. Available: https://www.pgnmentor.com/players/Carlsen/.

[45] J. Canny, 'A Computational Approach to Edge Detection,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986.

[46] P. V. C. Hough, 'Method and means for recognizing complex patterns,' U.S. Patent 3069654A, Dec. 1962.

[47] R. O. Duda and P. E. Hart, 'Use of the Hough transformation to detect lines and curves in pictures,' *Communications of the ACM*, vol. 15, no. 1, pp. 11–15, Jan. 1972.

[48] M. Ester, H.-P. Kriegel, J. Sander and X. Xu, 'A density-based algorithm for discovering clusters in large spatial databases with noise,' in *International Conference on Knowledge Discovery and Data Mining*, Portland, Oregon, Aug. 1996.

[49] R. Szeliski, 'Image formation,' in *Computer Vision: Algorithms and Applications*, London: Springer, 2011, pp. 27–86, ISBN: 978-1-84882-935-0.

[50] M. A. Fischler and R. C. Bolles, 'Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,' *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981.

[51] K. Simonyan and A. Zisserman, 'Very Deep Convolutional Networks for Large-Scale Image Recognition,' in *International Conference on Learning Representations*, San Diego, USA, May 2015.

[52] K. He, X. Zhang, S. Ren and J. Sun, 'Deep Residual Learning for Image Recognition,' in *IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2016, pp. 770–778.

[53] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, 'ImageNet: A large-scale hierarchical image database,' in *IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 248–255.

[54] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge: Cambridge University Press, 2004.

[55] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, 'Rethinking the Inception Architecture for Computer Vision,' in *IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2016, pp. 2818–2826.

A

Testing summary

B

User manual: chesscog

B.1 Data synthesis

```
cd /Applications/Blender.app/Contents/Resources/2.90/python/bin
./python3.7m -m ensurepip
./python3.7m -m pip install --upgrade pip
./python3.7m -m pip install python-chess
```

C

User manual: web app

D
User manual: recap package

E

Ethics self-assessment form

There are no ethical issues raised by this project. The self-assessment form is attached on the next page.

UNIVERSITY OF ST ANDREWS
TEACHING AND RESEARCH ETHICS COMMITTEE (UTREC)
SCHOOL OF COMPUTER SCIENCE
PRELIMINARY ETHICS SELF-ASSESSMENT FORM

This Preliminary Ethics Self-Assessment Form is to be conducted by the researcher, and completed in conjunction with the Guidelines for Ethical Research Practice. All staff and students of the School of Computer Science must complete it prior to commencing research.

This Form will act as a formal record of your ethical considerations.
Tick one box

☐ **Staff Project**
☐ **Postgraduate Project**
☒ **Undergraduate Project**

Title of project

Identifying chess positions using machine learning

Name of researcher(s)

Georg Wölflein

Name of supervisor (for student research)

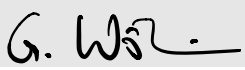
Dr Oggie Arandjelović

OVERALL ASSESSMENT (to be signed after questions, overleaf, have been completed)

Self audit has been conducted **YES** ☒ **NO** ☐

There are no ethical issues raised by this project

Signature Student or Researcher



Print Name

Georg Wölflein

Date

11.09.2020

Signature Lead Researcher or Supervisor

Print Name

Ognjen Arandjelovic

Date

15/09/2020

This form must be date stamped and held in the files of the Lead Researcher or Supervisor. If fieldwork is required, a copy must also be lodged with appropriate Risk Assessment forms. The School Ethics Committee will be responsible for monitoring assessments.

Computer Science Preliminary Ethics Self-Assessment Form

Research with human subjects

Does your research involve human subjects or have potential adverse consequences for human welfare and wellbeing?

YES ☐ **NO** ☒

If YES, full ethics review required
For example:
Will you be surveying, observing or interviewing human subjects?
Will you be analysing secondary data that could significantly affect human subjects?
Does your research have the potential to have a significant negative effect on people in the study area?

Potential physical or psychological harm, discomfort or stress

Are there any foreseeable risks to the researcher, or to any participants in this research?

YES ☐ **NO** ☒

If YES, full ethics review required
For example:
Is there any potential that there could be physical harm for anyone involved in the research?
Is there any potential for psychological harm, discomfort or stress for anyone involved in the research?

Conflicts of interest

Do any conflicts of interest arise?

YES ☐ **NO** ☒

If YES, full ethics review required
For example:
Might research objectivity be compromised by sponsorship?
Might any issues of intellectual property or roles in research be raised?

Funding

Is your research funded externally?

YES ☐ **NO** ☒

If YES, does the funder appear on the 'currently automatically approved' list on the UTREC website?

YES ☐ **NO** ☐

If NO, you will need to submit a Funding Approval Application as per instructions on the UTREC website.

Research with animals

Does your research involve the use of living animals?

YES ☐ **NO** ☒

If YES, your proposal must be referred to the University's Animal Welfare and Ethics Committee (AWEC)

University Teaching and Research Ethics Committee (UTREC) pages
http://www.st-andrews.ac.uk/utrec/