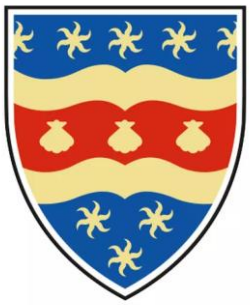
**Chess Piece Recognition: A Computer Vision and Machine Learning Approach**

by

**Rachel Ireland-Jones**

A report submitted to the

**University of Plymouth**



in partial fulfilment for the degree of BEng (Hons) Robotic

August 2023 12032 Words

**Abstract**

This report describes the development of a machine learning algorithm for chess detection and analysis. The algorithm detects the chess pieces and their locations on a board, displays the game status on a graphical user interface (GUI) and generates a game code that can be used to resume a physical game online. The report reviews the relevant machine learning methods for object detection, computer vision techniques for chessboard detection and training procedures for new models. The report evaluates and compares fifteen models created with Roboflow, Ultralytics Hub, YOLOv5 and Python using five datasets on three platforms. The evaluation metrics include mean average precision (mAP), recall, precision and object and class loss graphs. The report finds that Roboflow is the most integrated solution, but local training provides more flexibility and customization. The report identifies the main limitations and future improvements of the algorithm: real time detection and dataset diversity. Real time detection is feasible with this system but slow and resource-demanding, so the final design does not run in real time. Dataset diversity could be increased by using more images with different chess sets, scenarios, and cameras with better resolution.

**Acknowledgements**

Firstly, I would like to thank my project supervisor Ian Howard for the support and guidance provided in this project as well as teaching the machine learning module that inspired this topic choice.

Secondly, thanks should also go to the whole of the staff of Electronics and Robotics. Most importantly, those who are rarely thanked but always appreciated, to all the technicians, your willing assistance has impacted so many and is undeniably one of the best parts of this degree.

A large thank you must go to the team at Roboflow for providing extra credits and resources so that I could train my dataset multiple times, this meant I could experiment far further with my dataset and training evaluation.

Although far too many people have come and gone in my five years in Plymouth, my experience here has been centred around the Ultimate Frisbee team. The whole team, current students and alumni alike has shaped me into the person I am today. Special thanks to Zoe for dragging me out to swim and helping me find a healthier split between work and fun.

To my supporters elsewhere, my parents, my siblings, and my friends, your belief in me has kept me motivated and helped me continue even through the tougher moments.

My brother Edward was kind enough to spend time proof reading, which was quite the task, especially finding all my misplaced commas, thank you so much.

Finally, to my partner Ben, I am grateful for the understanding and the patience you have given me this year, especially in the stressful moments. Thank you for the laughter, food, and hugs.

## Glossary

| | |
|---|---|
| AI | Artificial Intelligence |
| CNN | Convolutional Neural Network |
| CSP | Cross-Stage Partial |
| CV | Computer Vision |
| DNN | Deep Neural Network |
| FIDE | Fédération Internationale des Échecs (International Federation of Chess) |
| MAE | Mean Absolute Error |
| mAP | Mean Average Precision |
| ML | Machine Learning |
| NN | Neural Network |
| R-CNN | Region-based Convolutional Neural Networks |
| SAN | Standard Algebraic Notation |
| SPPF | Spatial Pyramid Pooling with Factorization |
| SSD | Single Shot MultiBox Detector |
| YOLO | You Only Look Once |

# Contents

# 1. Introduction

Chess is more than just a game. It is a complex challenge for human intelligence, creativity, and strategy, as well as a rich source of problems for computer science to try and solve, especially in the fields of computer vision and machine learning. This dissertation will focus on two of these problems, chessboard, and chess piece recognition.

Chess piece recognition is the task of automatically identifying the positions and types of chess pieces on a chessboard, given an image or a video of the board. This task has multiple potential benefits such as allowing the players to play against competitive AI bots using physical boards, enabling online analysis and commentary of live games, and facilitating the development of chess education and training tools.

However, chess piece recognition is also a challenging problem due to factors that affect the quality and accuracy of the recognition. One difficulty is the diversity of chess pieces, which vary in size, shape, colour, and material. For example, even among the approved Staunton pieces, there are subtle differences in design and proportion that may confuse the recognition algorithm. Another problem can be the dynamic nature of the game, which requires the algorithm to manage changes in the board state, such as moves, captures, promotions, castling, en passant and checkmate.

The second problem discussed in this dissertation is chessboard detection, which is the task of locating and segmenting the chessboard from the background. This is a challenging problem due to issues arising from lighting conditions and the angle of the images and video which may cause shadows, reflections, or occlusions. As well as this, the chess pieces often cause additional occlusion and shadows which enhance the difficulty of this task.

The motivation for this dissertation is not only academic but also personal. Chess is a popular game that has been played for centuries. The history of modern chess dates back to

the 18th Century, and automated chess bots have been created since, the first attempt at an automated chess bot was the Mechanical Turk (Connaughton, 2008) in 1770, although it was later discovered to be a hoax. Two decades later, the Deep Thought computer was created by Feng-Hsiung Hsu at Carnegie Mellon University. It was the first chess computer capable of beating a chess grandmaster in a regular tournament game. (Hsu, et al., 1995)

As the field of computer vision and machine learning has advanced, chess computers have improved extensively. However, I have always been fascinated by the idea of playing against a competitive AI bot by using a physical board instead of a screen. I believe that this would enhance the experience and enjoyment of playing chess for players like me as well as allow for further analysis of the chessboard due to the physical aspect.

Moreover, chess is not only a game for experts or enthusiasts. It is also a game for everyone. It is estimated that up to eight hundred million people know how to play a basic game of chess. (The Chess Journal, 2021). The interest in chess has increased significantly in recent years, initially this rise was due to the COVID-19 pandemic as many people were forced to stay at home and look for new ways to entertain themselves and connect with others. Chess was a perfect choice, as it is a game that can be played online or offline, solo or with friends, and requires only a board and pieces.

The Netflix show The Queen's Gambit, which was released in October 2020, and the emergence of chess streaming, also boosted the interest in chess among the general public. For example, the Pogchamps tournament, which started in June 2020, featured popular streamers competing against each other in chess. Many streamers also collaborated with chess masters, such as Hikaru Nakamura and Eric Rosen, who have their own channels and offer coaching, commentary, and analysis.

However, the rise in chess didn't pause when the world opened up, in fact, according to Chess.com, a popular online chess server, there were more daily users in January 2023

than ever before, with 300,000 new members joining in one day and with an extraordinary 31 million games played on January 20 2023 alone (Chess.com, 2023). These factors show that chess is not only an academic or technical problem but that it is also a cultural and social phenomenon that deserves attention, and exploration.

This dissertation presents a machine learning-based approach to detecting chess pieces and their locations on a chessboard. The approach uses state-of-the-art machine learning techniques to accurately identify the positions of chess pieces in images of chessboards. A graphical user interface (GUI), which displays the current game status, has also been developed, allowing users to easily visualise the state of the game, and have included the research taken and beginning development of a chessboard detection algorithm.

The remainder of this dissertation is organised as follows. Chapter 2 reviews the relevant literature on detecting chessboards and chess pieces in images using machine learning techniques. Chapter 3 describes the methodology for developing the machine learning algorithm and implementing the GUI. Chapter 4, presents the results of the experiments and evaluates the performance of the approach. Finally, Chapters 5 and 6, discusses the implications of the findings and suggest directions for future research.

## 2. Research

The game of chess is one of the oldest and most popular games in the world. It is also a rich and complex domain for artificial intelligence and machine learning research. Chess poses various challenges and opportunities for developing and testing intelligent systems that can learn from data, experience, and feedback, as well as exhibit human-like abilities in decision making, problem solving and strategic thinking. This literature review surveys and analyses the existing research and literature on applying machine learning techniques to various aspects of chess. It also identifies the main research questions, objectives and methods used in this field. The main research fields that will guide the literature review are:

- Machine learning and image recognition: This section will cover the different types of machine learning methods and applications that have been used to detect objects of interest in images or videos, such as cars, people, animals, etc. It will also explore how machine learning can be used to extract and interpret visual information from chessboards using computer vision techniques, such as image processing, image recognition, object detection, and object tracking.
- Chessboard detection using computer vision techniques: This section will cover the different methods and practices of chessboard detection that can be utilised and compared in order to create the best method. It will also explain how this task is related to the research question or objective.

**Machine learning and Image Recognition.**

Machine learning and object detection is a field that covers the diverse types of machine learning methods and applications that have been used to detect objects of interest in images or videos. Object detection involves finding the location and the category of each object in an image or a video. It is a crucial task in computer vision that has many practical

and scientific implications, such as security, surveillance, autonomous driving, robotics, and more. As Zhao, et al. (2019) defines, “the problem definition of object detection is to determine where objects are located in a given image (object localization) and which category each object belongs to (object classification)”.

Machine learning is a branch of artificial intelligence that enables computers to learn from data and perform tasks that are difficult or impossible to program explicitly. Machine learning methods can be classified into four main types: supervised, unsupervised, semi-supervised, and reinforcement learning. Supervised learning uses labelled data to train a model that can predict the output for new input data. Unsupervised learning uses unlabelled data to discover hidden patterns or structures in the data. Reinforcement learning uses feedback from the environment to learn an optimal policy for an agent to achieve a goal.

This field will explore the details and differences of these methods, as well as their advantages and limitations for object detection tasks. It will also review some of the benchmark datasets and evaluation metrics used in object detection research. Furthermore, it will discuss some of the current challenges and future directions for machine learning and object detection. Object detection is important for chess research because it can help to create intelligent systems that can play, analyse, recognise, and generate chess positions and moves.

**Approaches to Image Recognition**

Image recognition is the process of identifying and detecting objects or features in digital images. This technology has a wide range of applications, including security, surveillance, healthcare, and entertainment. There are two main approaches to image recognition: classical and modern.

Classical approaches to image recognition involve the use of well-established computer vision techniques such as feature descriptors for object detection (O'Mahony, et al.,

2019). Before the emergence of deep learning, a step called feature extraction was carried out for tasks such as image classification. Features are interesting, descriptive, or informative patches in images. Two common methods classifying the overall image are maximum likelihood and minimum distance.

Maximum likelihood classification is a way of sorting things into groups based on how likely they are to belong to each group. It looks at the data and picks the most probable group for each thing. This method works better than others when the data is normal and similar within each group.

Minimum distance classification is another way of sorting things into groups based on how close they are to the average of each group. It measures the distance between each thing and the group's average and picks the closest one. This method is used to sort new data that we do not know much about. The distance shows how similar or different the data and the group are. The smaller the distance, the more similar they are.

Common classical machine learning techniques such as Viola-Jones Object Detection and Scale Invariant Feature Transforms were also used for object detection traditionally. These would detect a number of common features across an image and classify them using logistic regression and histograms. (Chopra, 2019)

Novel approaches to image recognition use deep learning techniques with neural networks architecture such as RetinaNet, SSD (Single Shot MultiBox Detector), and YOLO (You Only Look Once). Deep learning is a subset of machine learning that uses neural networks with multiple layers to learn and make predictions or decisions. It is made up of many interconnected processing nodes, called neurons, which learn from complex patterns in data to solve problems. Neural networks are used in many different fields, including image recognition, where they can be trained to identify and classify objects within images. In the context of image recognition, deep learning models are trained on large datasets of images to

learn how to identify and classify objects within images. One of the most prominent approaches in deep learning for image recognition is the use of convolutional neural networks (CNNs).

A CNN is a type of artificial neural network commonly used in image recognition and processing tasks. CNNs are specifically designed to process pixel data and are used in a variety of applications, including image and video recognition, image classification, and natural language processing. CNNs use a mathematical operation called convolution in place of general matrix multiplication in at least one of their layers. Convolution is a process that involves sliding a small filter or kernel over the input data and computing the dot product between the filter and the input at each location. This allows the network to learn local patterns or features in the input data. A typical CNN architecture consists of an input layer, multiple hidden layers, and an output layer. The hidden layers often include convolutional layers, which perform the convolution operation, as well as pooling layers, which reduce the spatial dimensions of the data, and fully connected layers, which perform classification.

Both classical and modern approaches to image recognition have their strengths and weaknesses. Classical approaches are well-established and have been used successfully for many years, while modern approaches such as deep learning techniques have shown great promise in achieving high accuracy and speed.

Under the image recognition umbrella there are multiple common methods. Image recognition encompasses several methods, including image classification, object recognition, and object detection. Image classification assigns an image to a category based on its content. Object recognition identifies objects within an image without providing their precise locations. Object detection identifies and locates objects using bounding boxes.

*Table 1 - Differences between Image Recognition methods*

| TASK | OBJECT RECOGNITION | OBJECT DETECTION | IMAGE CLASSIFICATION |
|---|---|---|---|
| **IDENTIFIES OBJECTS WITHIN AN IMAGE** | Yes | Yes | No |
| **PROVIDES LOCATIONS OF OBJECTS USING BOUNDING BOXES** | No | Yes | No |
| **ASSIGNS AN IMAGE TO ONE OF SEVERAL PREDEFINED CATEGORIES** | No | No | Yes |

Ultimately, the choice of which method to use depends on the specific requirement of the task. In this project, the aim is to find the location and type of each chess piece in an image, and as such, the focus will be on object detection.

**Object Detection Models**

There are two types of object detection models: two-stage detectors, and single-stage detectors.

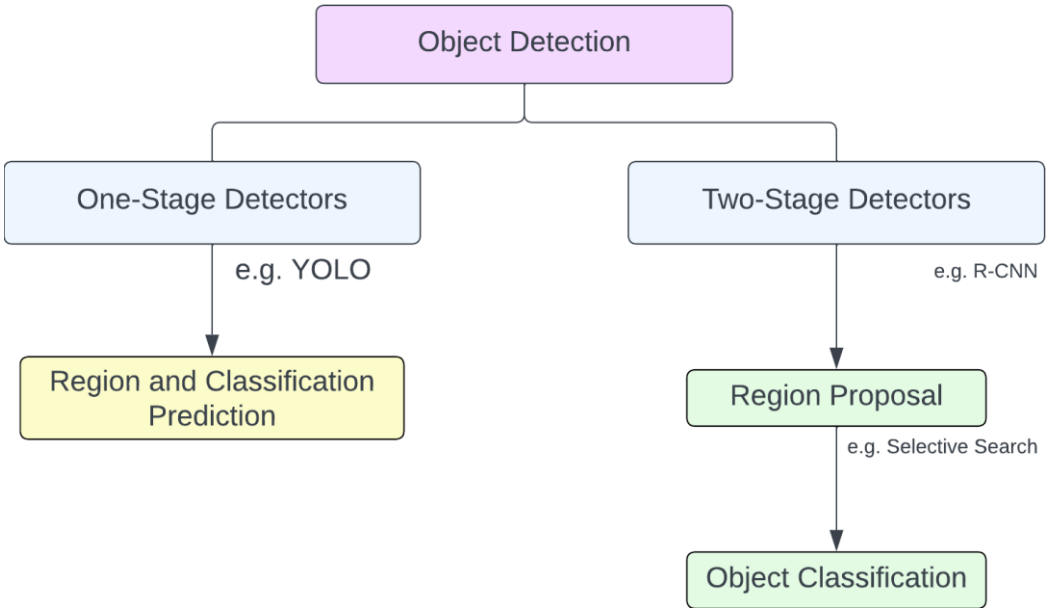



*Figure 1 - One vs Two Stage Detectors*

In single-stage detectors, YOLO for example, object classification and bounding-box regression is done directly whilst two-stage detectors, such as R-CNN (Region-based Convolutional Neural Networks), "consist of two key stages: region proposal and object classification" (Shree, 2023). In general, this difference means that one stage detectors are typically faster and simpler to implement than two stage detectors, however they often lack accuracy and provide less robust solutions. A full table of advantages and disadvantages is below in Table 2.

In order to decide which type of detector to use, it is important to consider the use case and the priorities for the project. In this project, a key element is the speed of detection and therefore a one-stage detector is preferable. This will enable faster prototyping and the possibility of testing real-time detection.

*Table 2 - Advantages and Disadvantages of one and two stage detectors*

| | Advantages | Disadvantages |
|---|---|---|
| One Stage Detector | Faster<br>Simpler<br>Robust to scale changes | Lower Accuracy<br>Less robust to occlusions. |
| Two Stage Detectors | Accuracy<br>Better at localisation<br>Robust to noise | Slower<br>More complex. |

Some common examples of one-stage object detectors include YOLO, SSD and DetectNet. The model chosen for this project is from the YOLO family. The YOLO family consists of eight models at this time, the first of these models (YOLOv1) was first introduced in 2016 with YOLOv2 and YOLOv3 being released soon after by the same author, Joseph Redmon. Since then, the YOLO family has evolved with new versions and new authors. A summary of these changes, taken from the Ultralytics website can be found in section Chapter -758646944(A2 - A History of YOLO). I chose to use YOLOv5 for this project as it is well established with good documentation to follow and refer to.

**YOLOv5 Architecture**

Like all CNNs, the YOLOv5 network consists of an input layer, hidden layers, and an output layer. In total YOLOv5 is reported to have 191 layers (Ultralytics, 2023) and like other single stage detectors, the architecture consists of three components, the backbone, the neck, and the head. The backbone is the main body of the network and is a pre-trained convolutional neural network that aggregates and forms image features at different granularities (Horvat, et al., 2022). In YOLOv5, the backbone is designed using a new CSP-Darknet53 structure which consists of fifty-three layers, the main purpose of these layers is to extract features from the input images. These layers follow on from the design of YOLOv3,

"CSP-Darknet53 is the convolutional network Darknet53 used as the backbone for YOLOv3, to which the authors [of YOLOv5] applied the CSP network strategy" (Imane, n.d.). Cross-Stage Partial (CSP) layers are used to reduce the computational cost and memory used by CNNs, typically by splitting and concatenating the feature maps in different stages.

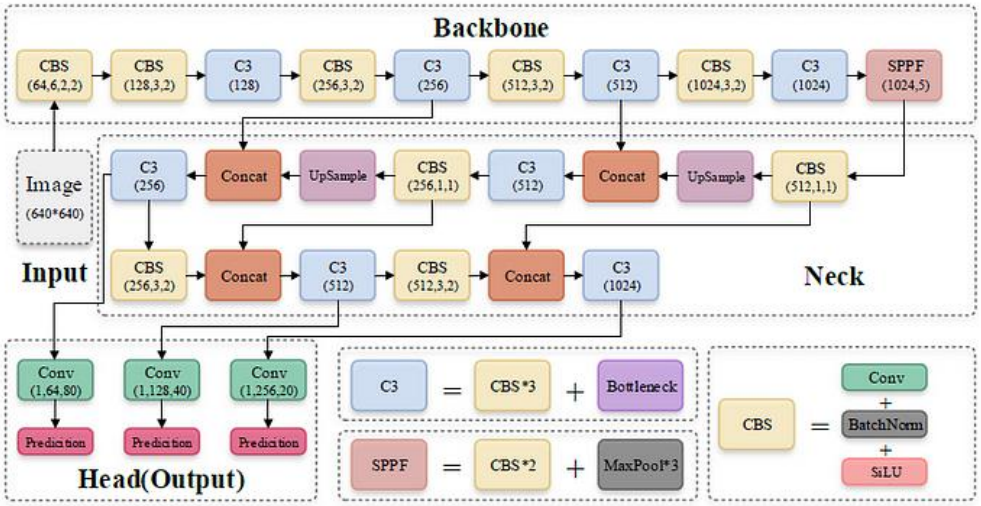



*Figure 2 – Default Network Structure of YOLOv5 (Liu, et al., 2022)*

The neck of YOLOv5 contains a Spatial Pyramid Pooling with Factorization (SPPF). This is an updated version of Spatial Pyramid Pooling (SPP) which takes around half the time to process. This layer is essential for capturing context information at multiple scales by pooling features with different window sizes. It also contains a PANet layer, which is used to combine features from various levels together. (Ultralytics, 2023)

The head of the model generates the final output, and in this case, it has not been altered from the YOLOv3 model. YOLOv3's head structure is composed of fifty-three layers, which are used for the bounding box regression and output generation.

**YOLOv5 Size**

Ultralytics provide five pre-trained models of various sizes which can be used for various tasks such as object detection or custom training. The smallest of these models is YOLOv5n, at just 4MB in size (Ultralytics, 2023), is also the fastest, achieving 37.2mAP

with an inference time on a V100 GPU of 6.3ms when deployed on the COCO dataset. The largest, YOLOv5x, is 166MB and the most accurate, achieving 50.7 mAP however there is a large speed trade off as the inference time has almost doubled. For this project, the size of the model chosen was YOLOv5m. This model is only 41MB and does not focus on either accuracy or speed, instead having a good combination of the two.

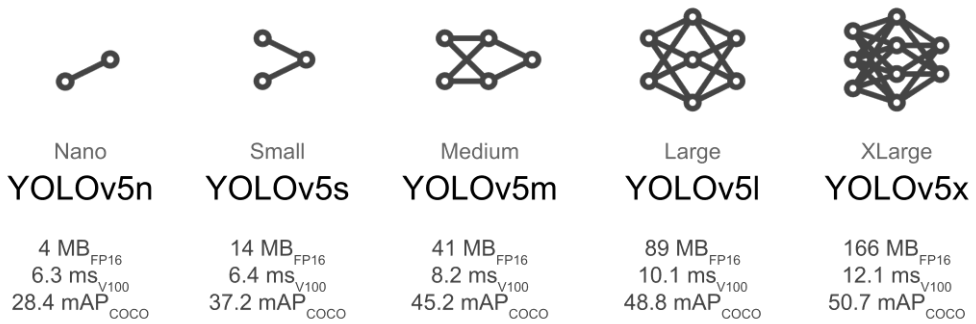



*Figure 3 - Comparing YOLov5 model statistics (Ultralytics, 2023)*

Transfer learning is a common machine learning task that involves taking a model pre-trained on a large dataset and adapting it to a new task with a smaller dataset. There are multiple methods available to perform custom training with transfer learning on YOLOv5. One of these is the Ultralytics implementation which can be performed locally on any computer with a GPU. The implementation allows customisation with optional layer freezing, model configuration and weight specification. During the transfer learning, during each epoch forward propagation and backward propagation are performed. Forward propagation is used to compute the output using the current weights and activation functions and then at the end of the epoch, the weights of the neural network are updated with backward propagation. This is where the error is calculated using the loss function and the learning rate (a fraction of the gradient of the error) is then used to adjust the weights. The combination of these two processes is called backpropagation and is an efficient way to perform gradient descent.

## Chess

### Staunton Pattern Design

Chess pieces come in many shapes and sizes however, the most widely used design is the Staunton pattern. The design of the Staunton pieces is credited to Nathaniel Cook; however, they are named after the top English chess-master at the time, Howard Staunton. This style was first available in 1849, quickly became popular and is now the standard style in international chess. The Staunton pattern is also recommended for use in competition by FIDE, the international chess governing body.

*Figure 4 - Original Staunton Chess Pieces (Wikipedia, 2018)*

The design is widely considered to have been based upon Victorian society and whilst "some variation is tolerated, there are several key distinguishing characteristics that define a set as a Staunton" (Stamp, 2013). For example, the king and queen have crowns, and are the tallest pieces on the board. The knight is a horse's head, and the bishop is designed after a bishop's hat (called a Mitre). More recently, newer variations of the Staunton chess set have developed and in 2013, the organisation World Chess asked Daniel Weil to redesign the pattern (Montgomery, 2013) in an attempt to re-invigorate interest in chess. Although the differences are subtle, the modern design is more simplified, and whilst the pieces still include these key features, they lack the original detail which kept them distinctly shaped.

*Figure 5 – Common Modern Staunton Chess Piece Design (Fruugo)*

There is no strict formula when creating chess pieces, even for the most popular design, and so due to the variations over time, although simple enough to distinguish with human eyes, computer vision techniques can struggle to identify modern pieces on a board. In order to overcome these challenges, deep learning models such as CNNs are commonly used to learn the features of the pieces from datasets of images. (Czyzewski, et al., 2020).

**Notation**

There are various methods of chess notation, however the official system of the FIDE is Standard Algebraic Notation (SAN) (Chess Programming Wiki, 2018). SAN is a system of symbols and letters that is used to record the moves and positions of pieces in a chess game. The notation consists of two parts: the name of the piece and the square where it is located. The name of the piece is denoted by a capital letter, such as K for king, Q for queen, R for rook, B for bishop, N for knight, and P for pawn. The square where the piece is located is denoted by a combination of a letter and a number, such as a1, b2, c3, etc. The letter indicates the column or file of the square, from a to h, while the number indicates the row or rank of the square, from 1 to 8. For example, Qd4 means that the queen is on the square d4.

Whilst SAN is used to notate moves in chronological order, Forsyth-Edwards Notation can be used to record the current state of chess games in plain text with one single line of ASCII. (ChessProgramming Wiki, 2018) One FEN string consists of six fields

separated by spaces, however for this project only the first field is of interest: piece placement.

Piece placement is determined by rank, starting at rank eight and continuing down to rank one. Each rank is then scanned from file ‘a’ to ‘h’. Using the same notation as SAN, where uppercase letters denote white pieces and lowercase letters denote black pieces, the pieces in each rank are noted. Squares without pieces are notated as number 1, and if the adjacent square is also empty, the number will increase in turn. For example, in Figure 6, the seventh rank contains four black pawns and a white knight. This would be notated as “`p2p1pNp`” with the number 2 denoting the gaps in b7 and c7, and the number 1 denoting the gap in e7. This system is used for each rank and then they are concatenated together using ‘/’ to return the final fen string. For example, in the game in Figure 6, the final FEN notation would be: “`r1b1k1nr/p2p1pNp/n2B4/1p1NP2P/6P1/3P1Q2/P1P1K3/q5b1`”. This notation can be used to denote the game at a moment in time however, it tells us nothing about the moves that lead up to this point.


*Figure 6 – Chessboard layout example (Chess.com, 2020)*

**Computer Vision**

"Computer Vision is a field of AI that enables computers to derive meaningful information from digital images, videos and other visual inputs." (IBM). It can be used to automate tasks and detect features that computers and systems otherwise struggle with. Within the context of detecting chessboards and chess pieces, there are some common techniques that have been developed.

Although not always used in the context of a chess game, detecting the features of a chessboard is a task that is frequently used for camera calibration and feature extraction. Camera calibration is a process of estimating information such as focal length, lens distortion and orientation from images. In order to do this, chessboard detection techniques have been developed. These are methods of finding the inner corners of a chessboard pattern in an image to use as calibration markers. Some examples of these methods include the Harris Corner Detector, the SUSAN Corner Detector and the EDLines Algorithm. OpenCV also provides a function findChessboardCorners that uses an alternative method, focusing on finding rhombus shapes and combining them to define a grid. However, these methods are not tailored for the specific task of detecting chessboards within a chess game scenario, which requires the outer corners of a grid to also be detected. Other factors such as lighting, perspective, occlusion, and chess piece movement can also affect the accuracy and robustness of detection. Therefore, throughout this project, an alternative method that combines elements from these existing techniques was developed.

Part of this new method included HoughLines detection. This is a method to detect straight lines in an image using the Hough Transform. This is extremely useful when detecting grid patterns as they are made up of straight lines. The Hough Transform technique works by utilising a voting procedure, each potential feature is transformed into parameter space and the points that receive the most votes correspond to the most likely lines in the

image. Hough Transform generates the best results after an edge detection process has been applied, reducing the complexity of the image. (Bhatt, 2022)

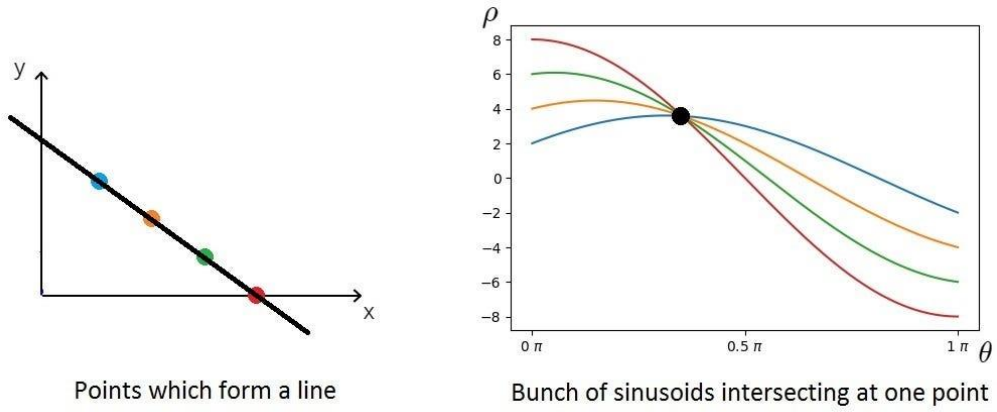



*Figure 7 - Image space vs Parameter space*

As Figure 7 demonstrates, one point in parameter space can be used to identify all points on a line in image space. Whilst image space uses x and y, parameter space uses theta and rho to describe the position of the line.

**Dataset Annotation**

Data annotation is the process of adding labels and tags to raw data such as images and videos in order to make them suitable for machine learning models. Data annotation can be performed with text editors, however the rise in popularity and efficiency in machine learning has also inspired many new data annotation tools to be created. These tools are typically software solutions that can help by creating guidelines, managing workflows, and validating results. Object detection tools are now commonly used within machine learning and help streamline the process of creating bounding boxes and labels on the training and validation datasets.

Data annotation tools can vary in their capabilities as well as the costs, availability, and capacity. To decide which tool to use in this project, the table below summarises the primary features for four popular tools.

*Table 3 - Data Annotation Tool Comparison*

| Tool | Key Features | Cost | YOLOv5 Compatible | Chosen! |
|---|---|---|---|---|
| V7 | - customisable<br>- model training<br>- assisted labelling | - £99 p/m | - Yes | |
| Roboflow | - AI enabled labelling<br>- Model Training<br>- Dataset Health Check<br>- deployment via Roboflow API with python | - free, three training credits<br>- supply students with extra credits<br>- enterprise plan available | Yes! Partnered with YOLOv5 | Yes! |
| LabelBox | - AI enabled labelling tools<br>- data management<br>- label performance analytics | - free up to five thousand images<br>- enterprise plan available | - Yes | |
| SuperAnnotate | - end to end<br>- video annotation<br>- AI-assisted labelling | - 14-day free trial, price not specified | - Yes | |
| Dataloop | - annotation<br>- model evaluation<br>- model improvement<br>- Automation & production pipeline using Python SDK | - free trial, price not specified | -Yes | |

# 3. Methodology

## Chessboard Recognition

The aim of this objective was to develop a system for recognising the chessboard and its squares from an image of the board. To achieve this, two different techniques were combined to extract the relevant features of the board: `findContours,` and `HoughLines.` These techniques are widely used in computer vision and image processing and are implemented in the OpenCV library (OpenCV). The following paragraphs provide a detailed description of each technique and how it was applied.

`findChessboardCorners` is a technique that is specifically designed for finding the internal corners of a chessboard pattern. It takes a grayscale image as input and returns an array of corner points, each represented by two coordinates. The function also performs some subpixel refinement to increase the accuracy of the corner detection. The function assumes that the chessboard pattern is regular and has a known number of rows and columns, which are specified as parameters.

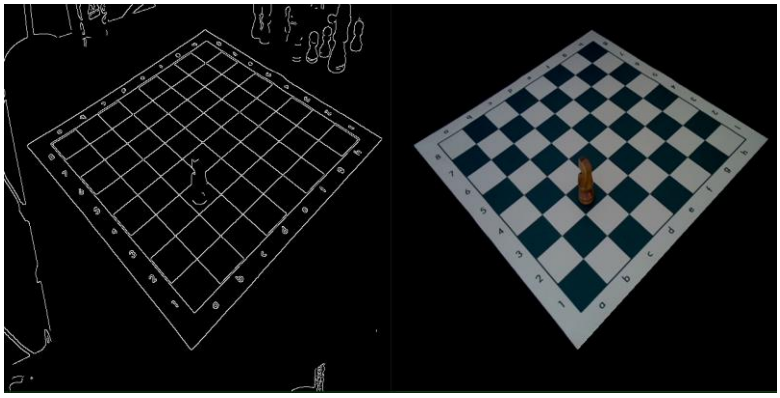


*Figure 8 - Using findContours to find the whole chessboard area and then mask it so the background is removed.*

`findContours` is a technique that is generalised for finding contours or boundaries of objects in an image. It takes a binary image as input and returns a hierarchy of contours, each

represented by a list of points. The function also applies some simplification and approximation algorithms to reduce the number of points in each contour. The function can also classify contours into external or internal ones, depending on whether they are outermost or nested within another contour. In this method, the findContours function was used to detect the contours of the edge of the chessboard, after applying some preprocessing steps to enhance its visibility and then the required region was masked to remove other elements of the image. Hough Lines was then performed upon the masked chessboard area.

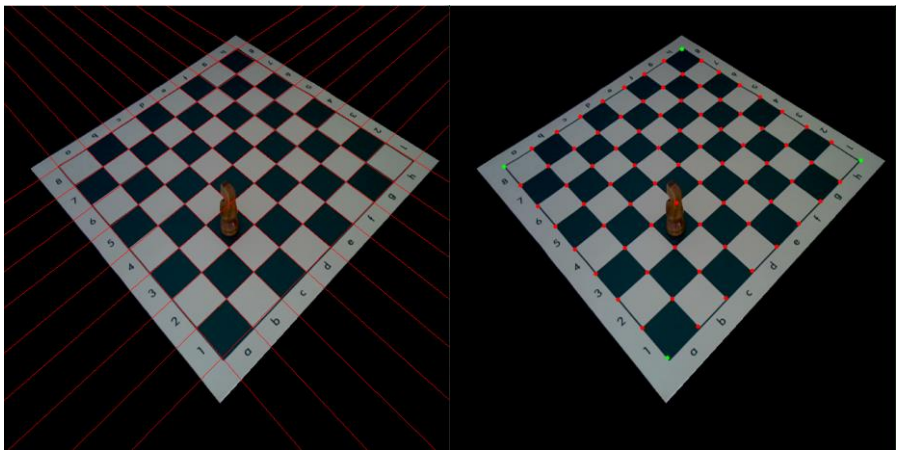


*Figure 9 - Using HoughLines to detect the squares (left) and then drawing the corners by finding the intersections of these lines (right).*

`HoughLines` is a technique based on the Hough transform, which is a mathematical method for finding straight lines in an image. The Hough transform converts each point in the image to a sinusoidal curve in a parameter space, where the intersections of the curves correspond to potential lines. The HoughLines function in OpenCV takes an edge-detected image as input and returns an array of lines, each represented by two parameters: the distance from the origin and the angle of the normal vector. The function also allows specifying a threshold for the minimum number of votes or intersections required for a line to be detected, as well as the resolution of the parameter space. In this case, the HoughLines function was used to detect the horizontal and vertical lines of the chessboard, which define its boundaries

and its squares. The edge-detected image was obtained by applying a Canny edge detector to the original image, with appropriate parameters for noise reduction and edge strength. The output of the HoughLines function was then filtered and sorted to obtain only the most relevant lines for the chessboard recognition and remove the outside lines which represented the chessboard's border. Finally, the intersections of the remaining lines were calculated to find the four corners of each chessboard square.

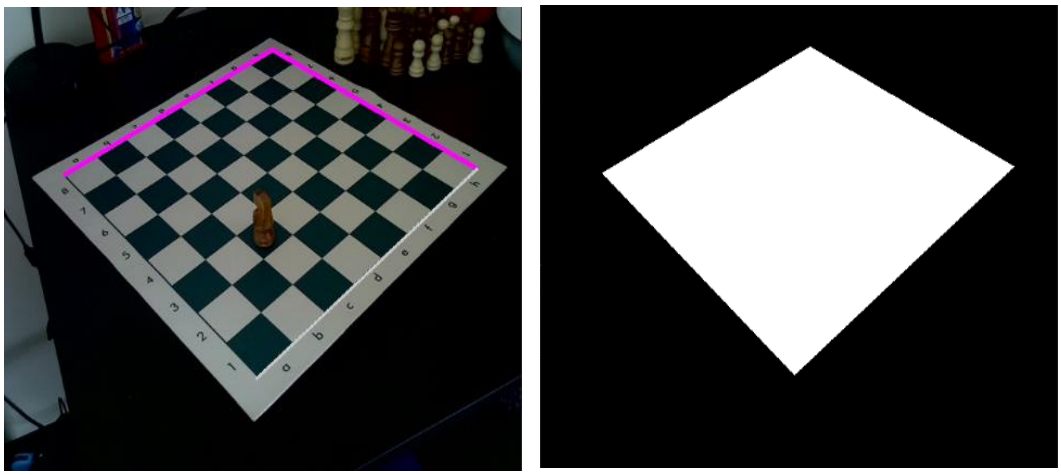


*Figure 10 – Left, user selection of the three corners of the chessboard, and right, the mask created from this action.*

In some cases, the techniques described above may fail to detect the chessboard or its squares correctly, due to factors such as poor lighting, occlusion, distortion, or noise. To handle these situations, a back-up method was implemented, which allowed the user to manually specify the corner locations of the chessboard using mouse clicks on the image. The user was instructed to click on the four outer corners of the chessboard, starting from the top-left corner and proceeding clockwise. The coordinates of the clicked points were then stored in an array. This array is then used to calculate a transformation matrix using the `getPerspectiveTransform` function. The inverse matrix is also calculated. This initial matrix is then used to warp the original image to obtain a rectified view of the chessboard, where each square has equal size and shape. A list of square corners is then generated using

`np.linspace` and the shape of the rectified view. The inverse matrix is applied to this list to determine the equivalent location of these corners on the original image.

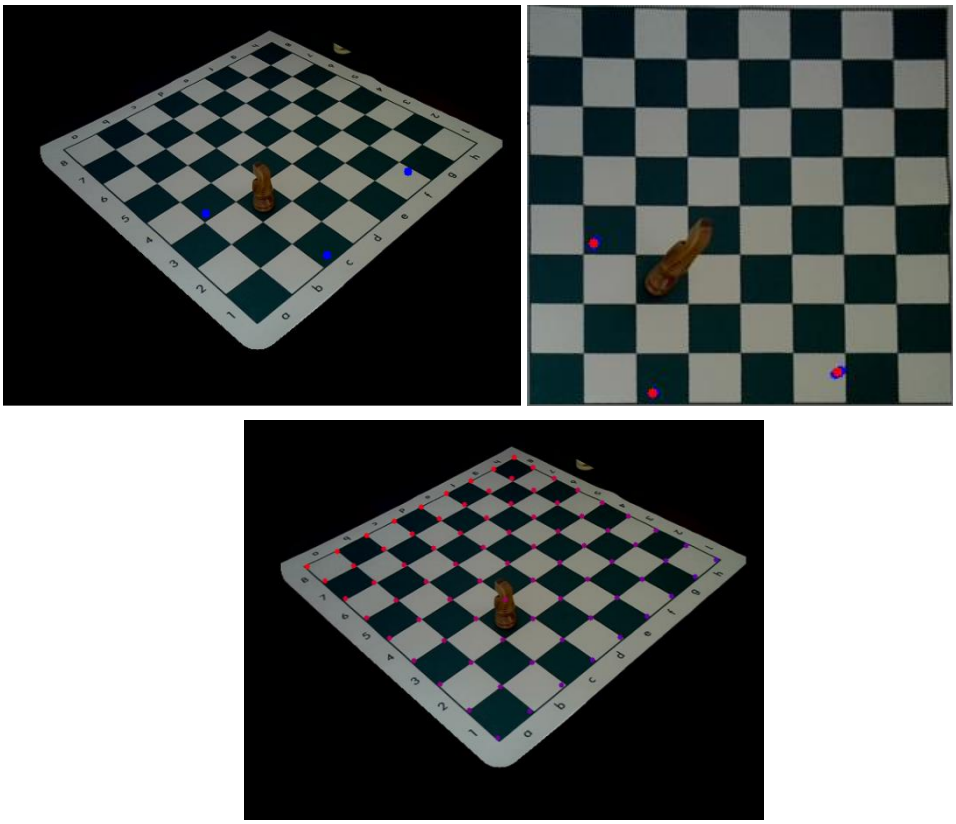


*Figure 11 – Top Left, choosing three points to test the accuracy of transform matrices.*

*Top Right, redrawing these points using on rectified image*

*Bottom, drawing all corners of each square on original image*

**Chess Piece Recognition**

In this section, the methods and tools described have been used to create a dataset for machine learning and train a computer vision model to recognise and represent the state of a chess game from a webcam or image input. The main steps of the methodology are:

- Data collection: Creating a collection of images of a chessboard and pieces using a webcam. The main aim was to have a diverse and representative sample of images with different angles, lighting conditions, and board colours.
- Data annotation: Following the research laid out in the research section of this paper, software by Roboflow was used to annotate the images. Roboflow (Dwyer & Nelson, 2022) is an online platform for computer vision datasets that allows users to label images with bounding boxes and export the dataset into various formats which are compatible with different machine learning frameworks such as COCO JSON and YOLO TXT.

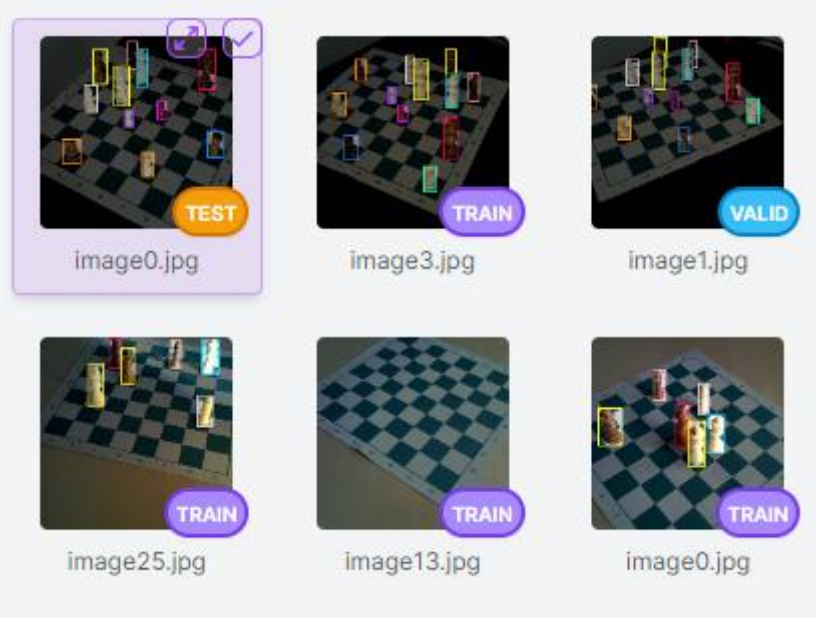



*Figure 12 - Examples of annotated images in Roboflow, including one "null" image with zero annotations. The images are split into training, testing and validation datasets.*

- Data pre-processing: Roboflow was also used to apply various preprocessing techniques to the images in the dataset. These help to improve the diversity of the dataset further and reduce unconscious bias. Roboflow also provides the ability to split the dataset into training, validation and testing sets with user-determined ratios.
- Model training: Multiple methods of model training were utilised to customise the models however all of the methods used were based on YOLOv5. YOLOv5 is fast, accurate and scalable which makes it suitable for use. The three methods are described below:
    - **Roboflow Train**: This method involved using Roboflow's AutoML product Roboflow Train, a service to automatically train a model on Roboflow's cloud infrastructure. This system uses a COCO checkpoint for object detection projects and automatically chooses the metrics and parameters that it thinks will provide optimal results and performs the training for the user with just one click of a mouse. Models trained using this method require the Roboflow hosted API to access predictions. This makes it quite simple, however computationally slow.
    - **Ultralytics HUB and Google Notebooks**: This method involved using the Ultralytics Hub to fine-tune a pre-trained YOLOv5 model on Google Colab. Google Colab (Google, 2022) is a cloud-based service that allows users to run Python code in notebooks using Google's free GPU resources. Ultralytics Hub (Ultralytics, 2023) is a web-based platform that provides access to pre-trained YOLOv5 models that can be fine-tuned on custom datasets using Google Colab. It also provides assorted options for customising the model hyperparameters and data pipeline as well as visualising the results. The YOLOv5s model from Ultralytics Hub was used and fine-tuned on the dataset

using Google Colab. The models using this method were trained for one hundred epochs using a batch size of sixteen and a learning rate of 0.01. Models trained with this method use the Ultralytics HUB hosted API to run deployment and predict images.

- **YOLOv5 Local Training:** This method involved installing and configuring the YOLOv5 software on a local machine. The software provided various options for customising the model architecture, hyperparameters and data pipeline. The Comet ML library was also used to track and visualise the training metrics such as loss, accuracy, precision, recall and mAP. Comet ML is a cloud-based platform that allows users to log and compare experiments across different models, datasets, and frameworks. The model was trained on a local GPU which was an NVIDIA GeForce GTX 1060 SUPER with 12 GB of memory and each model was trained for fifty epochs with a batch size of sixteen. Models trained using this system can be run locally which is a large benefit, however the speed of this deployment will depend on the hardware limitations of the deployment device.

- Model evaluation: By comparing various metrics such as precision, recall and mean average precision (mAP) from each trained model, it is possible to evaluate the performance of the models before use on a testing set of images. When a satisfactory model had been trained, this model was deployed onto new images using the Roboflow hosted API and the results were compared to ground truth data.

**Chess Piece Location and Notation**

After obtaining the representation of the chessboard and its squares, the next step was to identify the location of each piece on the board and display it using chess notation. To find the location of each piece on the board, a technique was employed that involved two main steps: detecting the bounding boxes of the pieces and comparing them to the squares on the chessboard.

The first step was to detect the bounding boxes of the pieces the machine models trained previously. Each type of model required slightly different technique and syntax to work in python however the output of each one was the same, an array of bounding boxes and labels for each piece on the board.

The second step was to compare each bounding box to each square on the chessboard and find the best match. To do this, a simple algorithm was devised, based on the assumption that the lower third of each bounding box will occupy almost entirely the square the piece is on. The algorithm involved taking the lower third of each bounding box, which corresponds to the base of each piece, and computing its intersection area with each square on the chessboard. The intersection area was calculated as the product of the width and height of the overlapping region between two rectangles. The algorithm then selected the square with the largest intersection area as the best match for each bounding box.

The last step was to convert each prediction, made up of a bounding box and a square, into chess notation, using their labels and coordinates. The label of each bounding box provided the name of the piece, while the coordinates of each square provided its location on the board. For example, if a bounding box with label Q had a matching square with coordinates (3, 4), this would translate into Qd4 in chess notation. The output of this step was an array of chess notation symbols for each piece on the board. By applying this technique, it

was possible to display the current FEN notation of the game state and use this notation to create a 2D representation of the game.

In order to display the 2D image of the chess game on screen, a python module `fentoboardimage` (Krawiec, 2023) was imported, this module has a function, `fenToImage`, that returns a PIL image of a chessboard when it is given a FEN code in string format. Once the image was created, a border that contained the rank-and-file notation was added to make it simpler to read the image.

**Graphical User Interface**

To create a simple graphical user interface (GUI) for this project Python was utilised alongside the `pygame, pygame_gui` modules. The first step was to design the layout of the GUI, which involved deciding on the placement and arrangement of various GUI elements, such as buttons, input fields, and display areas. A simple design, which showed the webcam image, the detected state of play and the FEN code of the board layout was chosen. Figure 13 shows the initial sketch of this design which highlights that this GUI is to make a simple way to confirm the success of the project, not create a fancy system.

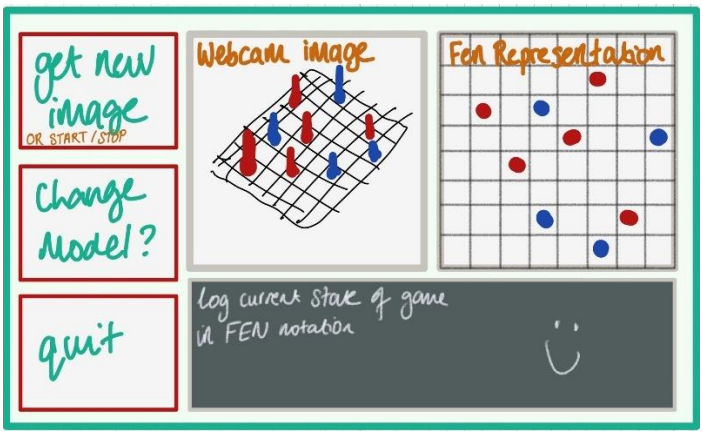



*Figure 13 - Initial GUI sketch*

Once the design was finalised, it was implemented using `pygame` and its `pygame_gui` module. This involved creating instances of various GUI elements, such as buttons and text

boxes, and adding them to the `pygame` application. Functionality was then added to the GUI that allowed the user to get a new frame from the webcam by adding a button.

After obtaining the new frame, the chess piece detection algorithm runs and the results of processing the frame are then displayed on the GUI. This involved drawing bounding boxes around the detected chess pieces and displaying their positions. The final step in the methodology was to test and refine the GUI and its underlying algorithms. This involved manually interacting with the GUI to ensure that it behaved as expected.

# 4. Results

## Chessboard Recognition

The results of the chessboard detection methodology showed that the combination of the `findContours` and `HoughLines` functions in OpenCV was effective in detecting the boundaries and lines of the chessboard in most cases. The automatic detection method was able to accurately identify the corners of the chessboard and its squares in situations where the board was clearly visible but struggled with occlusion and when there was a lot of complexity in the background of the image.

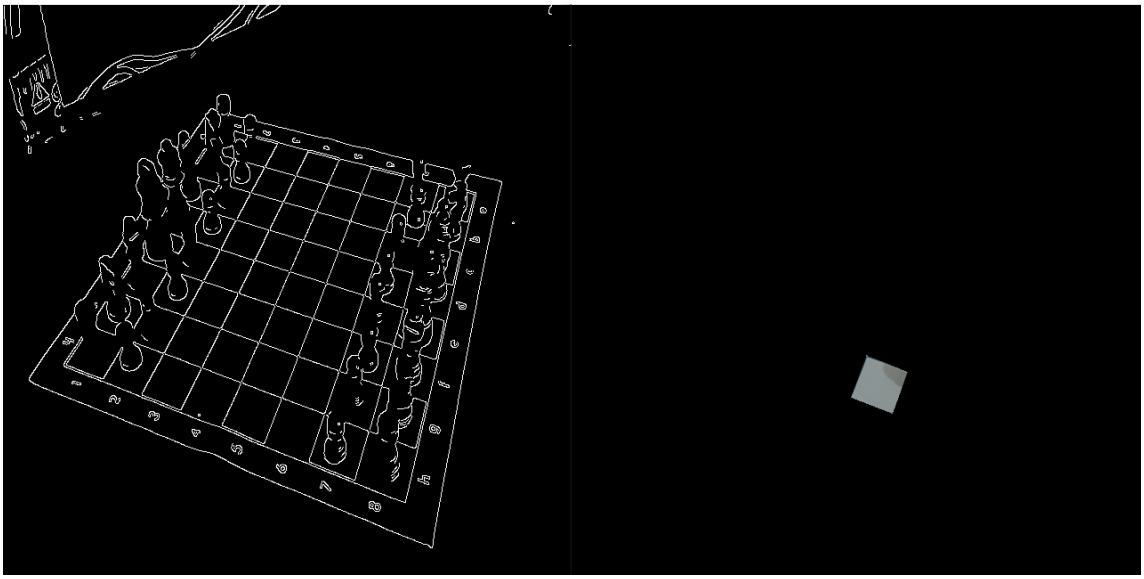


*Figure 14 - Occlusion by the chess pieces causes the findContours stage to fail.*

In cases where the automatic detection method failed due to factors such as poor lighting, occlusion, distortion, or noise, the backup method that allowed the user to manually specify the corner locations of the chessboard using mouse clicks proved to be a reliable alternative. The user was able to accurately identify the corners of the chessboard, and the calculated transformation matrix produced a rectified view of the chessboard with equal size and shape for each square which could be utilised to automatically create corners.

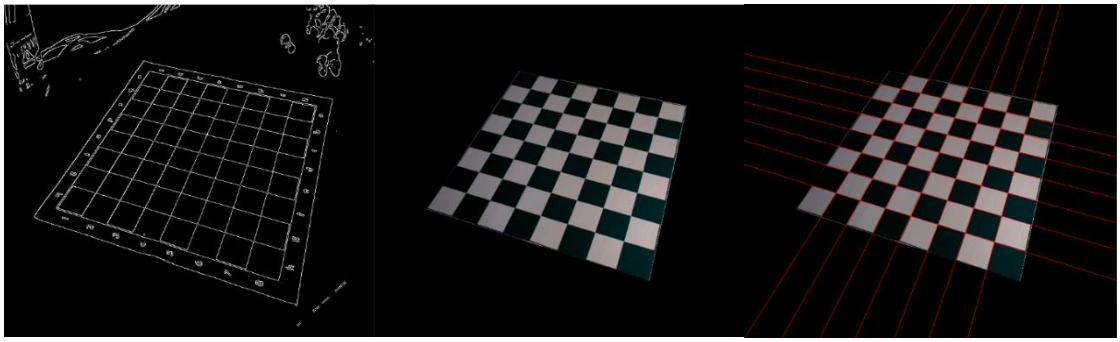


*Figure 15 – Faulty chessboard detection due to lighting*

In the scenario in Figure 15, a change in the lighting conditions changed the results of the findContours stage of the algorithm and caused the border of the chessboard to be removed while the mask step occurred. This means that the only lines detected are the internal lines of the squares, and not the major border around the edge. This means that although the detection has worked well, a seven-by-seven chessboard is returned instead of an eight-by-eight chessboard.

In Figure 16, the chess pieces on the board cause occlusion and although the mask is successful, the HoughLines function struggles to find all the lines. As the corners cannot all be detected, this attempt will also return an error and prompt manual detection.

*Figure 16 – Faulty chessboard detection due to occlusion.*

Overall, the results demonstrate that the methodology used for chessboard detection was effective in the right conditions, accurately identifying the chessboard and its squares in

an image. The combination of automatic and manual detection methods provided a robust solution for handling a variety of challenging situations.

**Chess Piece Detection**

For this objective, I aimed to detect and classify images of chessboards using YOLOv5, a state-of-the-art object detection model. As described in the methodology section, I used Roboflow, a platform for creating and managing computer vision datasets, to annotate and preprocess the images. I also used three different training methods to compare their efficiency and accuracy: local training on a GPU with YOLOv5 software and Comet ML for tracking, Roboflow Train, and Ultralytics Hub on Google Colab. I experimented with five versions of the dataset, each with different characteristics and challenges. The following table, Table 4, summarises the key features and results of each dataset and model:

*Table 4 - Summary of datasets and trained models.*

| Dataset | Images | Classes | Preprocessing | Augmentations | Local Training | AutoML | Ultralytics Hub |
|---|---|---|---|---|---|---|---|
| V1 | 32 | 8 | Resize orient | Crop brightness flip | - poor<br>- mAP 40.3%<br>- 50 epochs (30 min) | - poor<br>- mAP 98.9%<br>- 299 epochs (10 m) | - poor<br>- mAP 78.1%<br>- 100 epochs (30m) |
| V3 | 424 | 6 | Resize orient | Rotation, brightness, grayscale, exposure, saturation | - good<br>- mAP 97.9%<br>- 50 epochs (10m) | - good<br>- mAP 98.9%<br>- 299 epochs (25m) | - good<br>- mAP 92.9%<br>- 100 epochs (5h 10m) |
| V8 | 446 | 12 | Resize orient rename | Rotation, brightness, grayscale, exposure, saturation | - medium<br>- mAP 90.0%<br>- 50 epochs (11.5h) | - medium<br>- mAP 98.7%<br>- 97 epochs (45m) | - medium<br>- mAP 79.4%<br>- 100 epochs (11.5h) |
| V12 | 505 | 12 | Resize crop rename | Brightness, contrast, saturation, rotation, flip, exposure | - good<br>- mAP 98.4%<br>- 50 epochs (8h) | - good<br>- mAP 99.0%<br>- 122 epochs (35m) | - good<br>- mAP 98.7%<br>- 100 epochs (8h) |
| V15 | 505 | 6 | Resize crop rename | Brightness, contrast, saturation, rotation, flip, exposure | - good<br>- mAP 98.7%<br>- 50 epochs (13h) | - good<br>- mAP 99.4%<br>- 232 epochs (60m) | - good<br>- mAP 99.2%<br>- 100 epochs (13h) |

**Model Training**

The first version of the dataset, V1, had only 32 images which was too small to train a robust model. In this dataset the black and white chess pieces were treated as the same class and two additional classes were annotated: “chessboard” and “hand.” These classes were removed from later datasets as they were not relevant to the task. The preprocessing was minimal, only resizing and orienting the images. The augmentations were also simple, only cropping, adjusting brightness, and flipping bounding boxes. Three augmented outputs were created per training image. This dataset performed poorly in all three training methods as expected due to the lack of data. It took only 30 minutes to train for 50 epochs on a local GPU and 20 minutes on Google Colab.

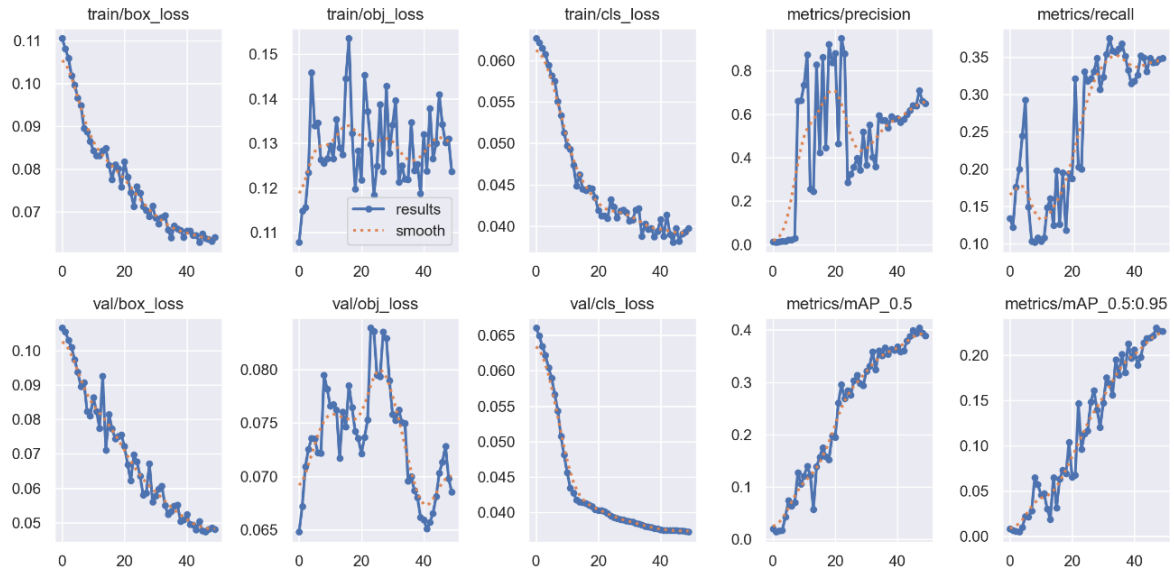



*Figure 17 – V1 local GPU training results*

The second version of the dataset, V3, had more images than V1 (424), and the number of classes was reduced from eight to six by removing the “chessboard” and “hand” classes and keeping the black and white pieces as one class. The preprocessing was similar to V1, only performing resizing and re-orienting upon the images. The augmentations were more diverse than V1 including random rotation, brightness, grayscale, exposure, and

saturation adjustments. Three augmented outputs were created per training image. This dataset performed well in all three training methods despite having less data than ideal. It achieved a max mAP of 0.973 on the test set with a confidence of 0.5. It took five hours and ten minutes to train for 50 epochs on a local GPU and 1 hour and 5 minutes on Google Colab.

The graphs in Figure 17 and Figure 18 show how much difference these changes, particularly the increase in photos made to the training. For example, it is clear that the object loss, which in V1 was erratic and showed little correlation, followed a far more logarithmic nature which suggests better learning.

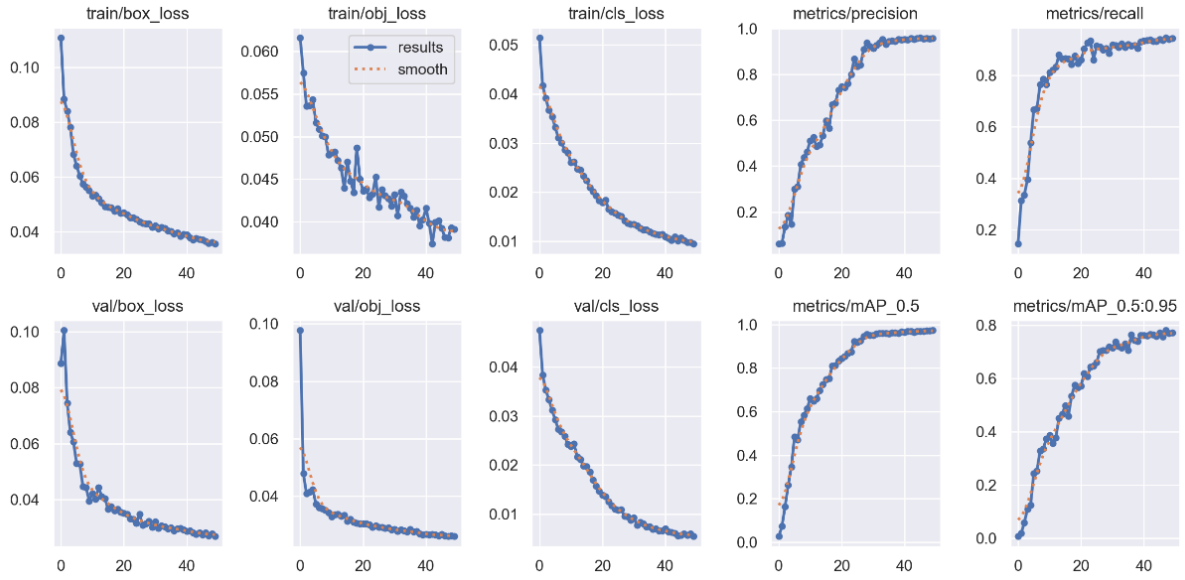



*Figure 18 – V3 local GPU training results*

The third version of the dataset, V8, had more images and classes than the previous ones as I decided to separate the black and white pieces into different classes. However, this increased the complexity of the task and required more data to train the model. The preprocessing was similar to the previous versions only resizing, orienting, and renaming the classes. The augmentations were also similar to V3, but I increased the number of outputs per image from three to six. However, these augmentations did not account for the variations in lighting angle perspective or background that could affect the detection of chess pieces.

Therefore, this dataset performed poorly in all three training methods. Moreover, it took a long time to train on a local GPU and actually scored a lower mAP than the V3 model.

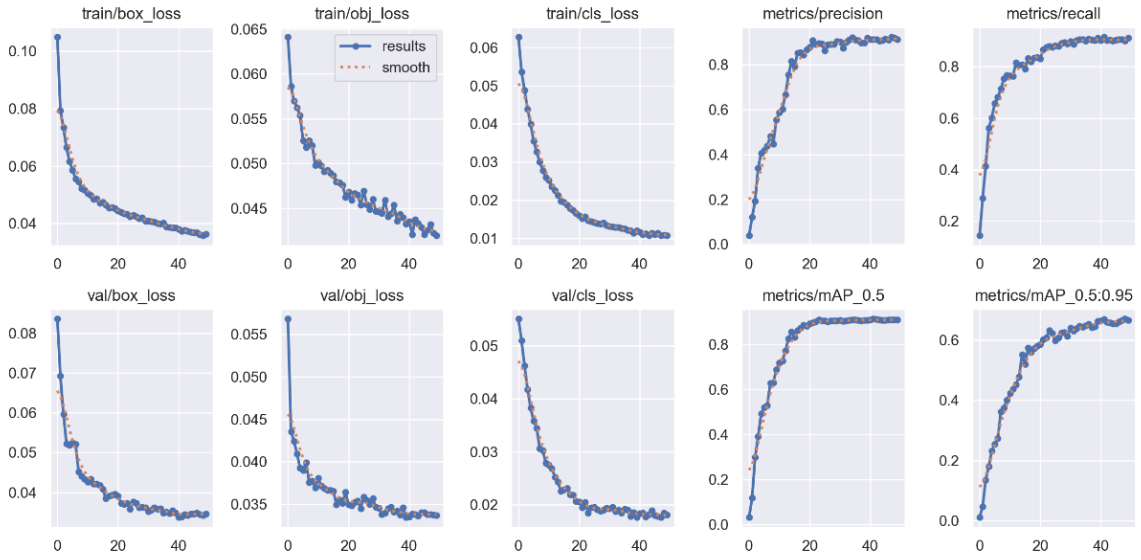



*Figure 19 - V8 local GPU training results*

The fourth version of the dataset, V12, had more images and diversity than the previous ones as I focused on improving the lighting and quality of the images. The number of classes remained the same as V8. The preprocessing included resizing, cropping, and renaming the classes. The augmentations included more adjustments for brightness, contrast, and saturation as well as rotations, flips and exposure changes. I also quadrupled the number of images and instances per class by applying augmentations to each image. This resulted in a total of 1400 training images. Whilst training with Roboflow Train, I used a checkpoint from V3 to speed up the process. This dataset performed well in all three training methods. It also took less time to train on a local GPU, 8 hours for fifty epochs, than V8. The results shown in Figure 20 indicate that the majority of the learning is completed in the first 20-25 epochs, with learning slowing after this turning point.

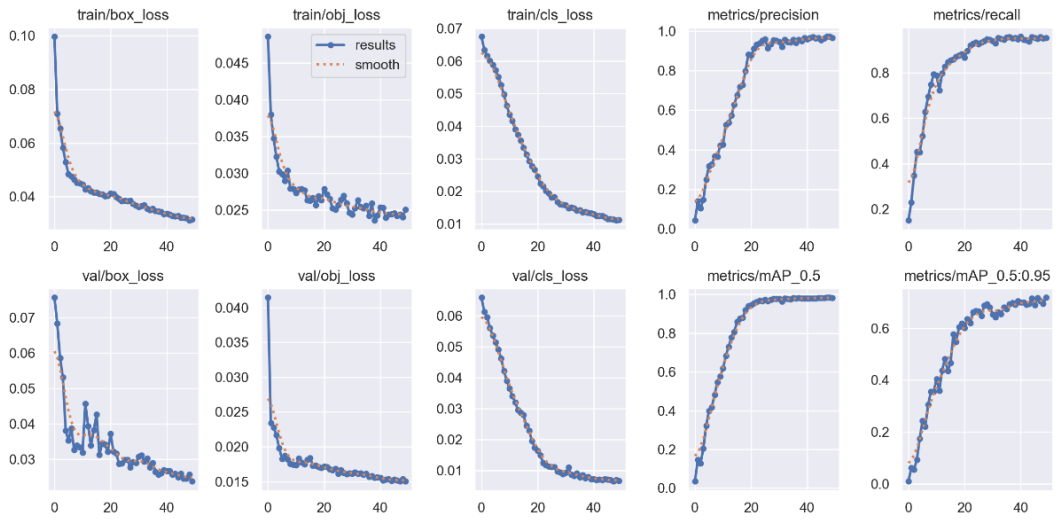



*Figure 20 - V12 local GPU training results*

The final version of the dataset, V15, was almost identical to V12, however this time in the preprocessing step the black and white pieces were combined into one. This meant there were six overall classes one for each piece type. Preprocessing and augmentation followed the same steps as in V12 and the total number of training images was multiplied by six for a total of 2100 training images. Within the Roboflow training method the model trained from the previous V12 checkpoint. This model took 12 hours and 50 minutes to train for fifty epochs on a local GPU and 120 minutes on Google Colab.

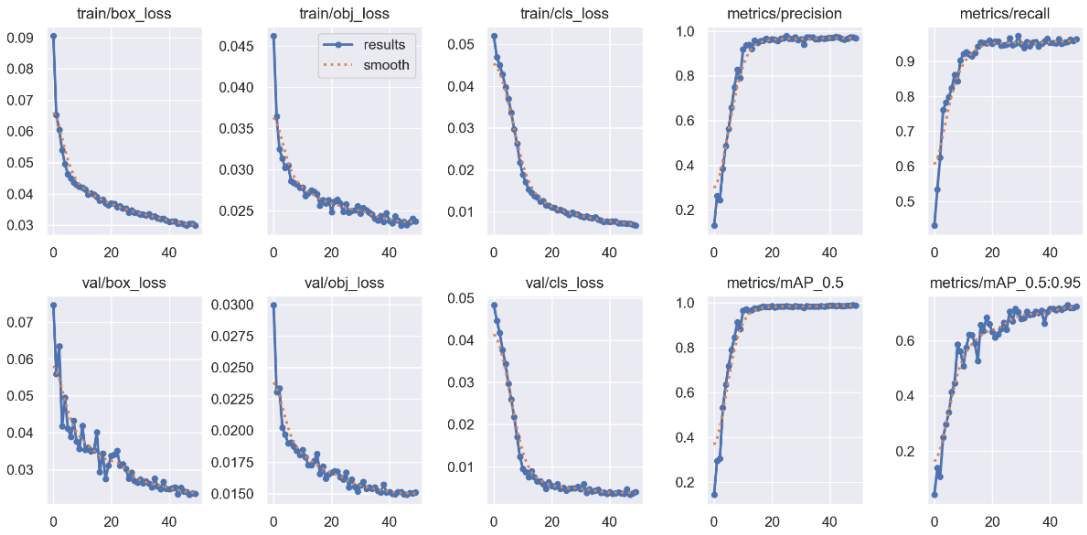



*Figure 21 - V15 local GPU training results*

**Model Evaluation**

The main metrics used to evaluate the effectiveness of these models were object and class loss, precision, recall and mean average precision (mAP). Object loss measures how successfully a model predicts the location and size of objects in an image and Class loss is a metric that measures the accuracy of class predictions for each object in an image. In this project, the final object loss and class loss percentages when the model has finished training, can give an indication into the likelihood that the predicted bounding boxes and class predictions will be accurate during deployment.

Precision measures how many of the predicted objects are correct, calculating the ratio of true positives to the sum of true and false positives. Recall measures how many of the correct objects are predicted, calculating the ratio of true positives to the sum of true positives and false negatives. High precision indicates that there are few false predictions whilst high recall indicates that there are few misses. By calculating the area under a precision-recall graph, the average precision can be calculated. This then can be used to calculate the mAP. A high mAP means the model has high precision and recall for all classes at various thresholds.

If a model has a high mAP, it does not necessarily lead to a successful deployment of a model. Issues can be caused by overfitting, lack of training data, lack of representation and diversity in the training data or generalisation of new data.

The graphs in Figure 17 to Figure 21, can be used to compare each of these metrics and evaluate the likely success of deployment. From these graphs, the following table (was created, which allows simple comparison between each of these metrics. Whilst this is possible using the graphs, their axes are all adjusted to create the clearest image which can mislead easily. Using the table, it is clear that V15 scored the best across all metrics and V12 was a close second on all but one, class loss.

*Table 5 - Training metric comparison across all models (local training)*

| Model | V1 | V3 | V8 | V12 | V15 |
|---|---|---|---|---|---|
| **Object Loss** | 0.12 | 0.039 | 0.042 | 0.025 | 0.024 |
| **Class Loss** | 0.039 | 0.0095 | 0.011 | 0.011 | 0.0067 |
| **Precision** | 0.649 | 0.958 | 0.912 | 0.966 | 0.967 |
| **Recall** | 0.349 | 0.944 | 0.9 | 0.954 | 0.963 |
| **mAP** | 0.388 | 0.973 | 0.91 | 0.982 | 0.988 |

One interesting thing about the results is that V8 is worse than V3 in all categories. This seems unlikely as more variety of images were added; however, this was the point at which the number of classes annotated in each image changed from six to twelve and the models started to distinguish between black and white chess pieces which could be part of this reduction in accuracy.

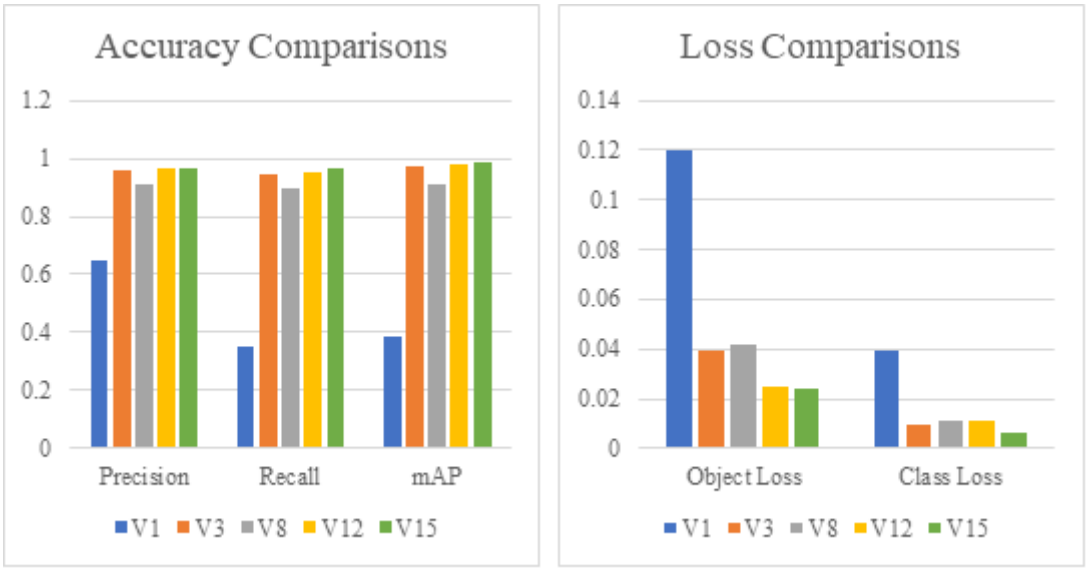



*Figure 22 - Graph showing results from Table 5.*

**Method Evaluation**

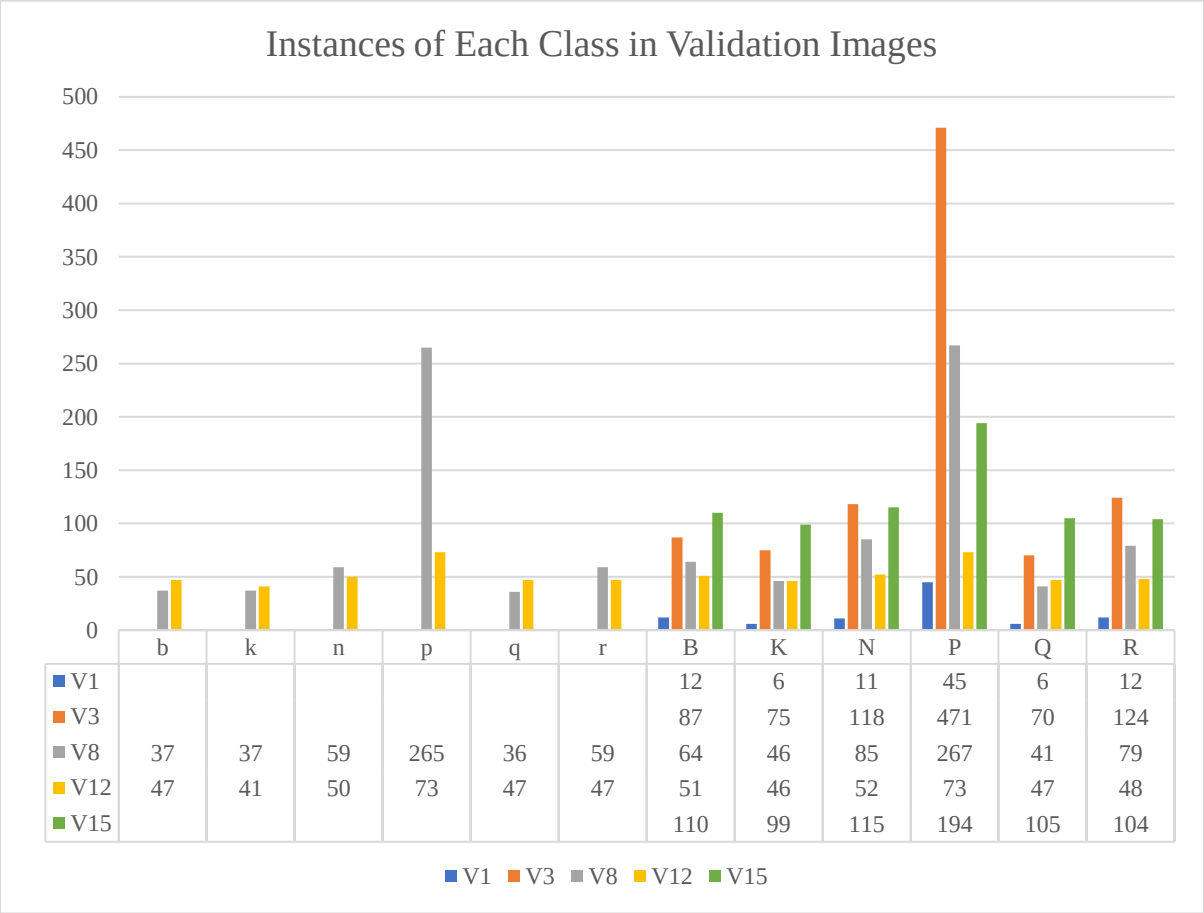
The three training methods have different advantages and disadvantages. The local training with a GPU and YOLOv5 software as well as using Comet ML for tracking was the most flexible and customizable option, with simple ways to change the batch size, number of epochs and many more parameters. However, it was also the most time-consuming and resource-intensive option as it required installing and configuring various software packages and libraries as well as managing the GPU memory and storage. It took so much longer than the other methods that these additional options did not seem to really justify the extra time and computational power required for each training session.

Roboflow's Train with AutoML cloud service was the easiest and fastest option as it automated the entire training process and also provided a user-friendly interface for monitoring and evaluating the models. However, it offered fewer metrics and evaluation methods than the other options which made it harder to compare results. It also did not allow customisation of the model and parameters, instead automatically choosing what it considered the best configuration for the user. The most beneficial detail about Roboflow Train is that it trains until it no longer detects improvement. This means that whilst using the other methods the models only trained for 50 or 100 epochs, this method trained for up to three hundred epochs, even though it took less time to do so.

The Ultralytics Hub on Google Colab was a middle ground between the local training and Roboflow Train as it provided a pre-trained YOLOv5 model that could be fine-tuned on a dataset using Google's free, cloud-based GPU resources. However, it also had some limitations such as requiring a stable internet connection and having a time limit for each session. It also does not automatically save the output data from the training session which means metrics such as duration and metrics about individual epochs cannot be accessed after it is finished.

**Comparing the datasets and models**

As well as the type of training and the dataset upon which the model is trained, there are some other key factors that can determine the success of a machine learning model. These include the number and quality of the annotations, and the number and clarity of the classes.




| | b | k | n | p | q | r | B | K | N | P | Q | R |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| V1 | | | | | | | 12 | 6 | 11 | 45 | 6 | 12 |
| V3 | | | | | | | 87 | 75 | 118 | 471 | 70 | 124 |
| V8 | 37 | 37 | 59 | 265 | 36 | 59 | 64 | 46 | 85 | 267 | 41 | 79 |
| V12 | 47 | 41 | 50 | 73 | 47 | 47 | 51 | 46 | 52 | 73 | 47 | 48 |
| V15 | | | | | | | 110 | 99 | 115 | 194 | 105 | 104 |

*Figure 23 - Number of Instances of each Class*

In general, having more training data can improve the performance of a machine learning model, as it allows the model to learn more about the relationships between the features and the target variable. However, if the images that a model is trained on has been badly labelled, more training data might not help and might actually hinder the training of the model. After training V8 and finding that the previous version was actually more accurate than V8, the quality of the labelling was a possible reason. More images had been added between the two datasets and the images had also been re-labelled to contain twelve instead

of six classes. The YOLOv5 documentation says “Labels must closely enclose each object. No space should exist between an object and its [sic] bounding box. No objects should be missing a label.” (Ultralytics, 2020) . This meant that before training V12, all the images were re-examined, checked, and relabelled to improve the training chances.

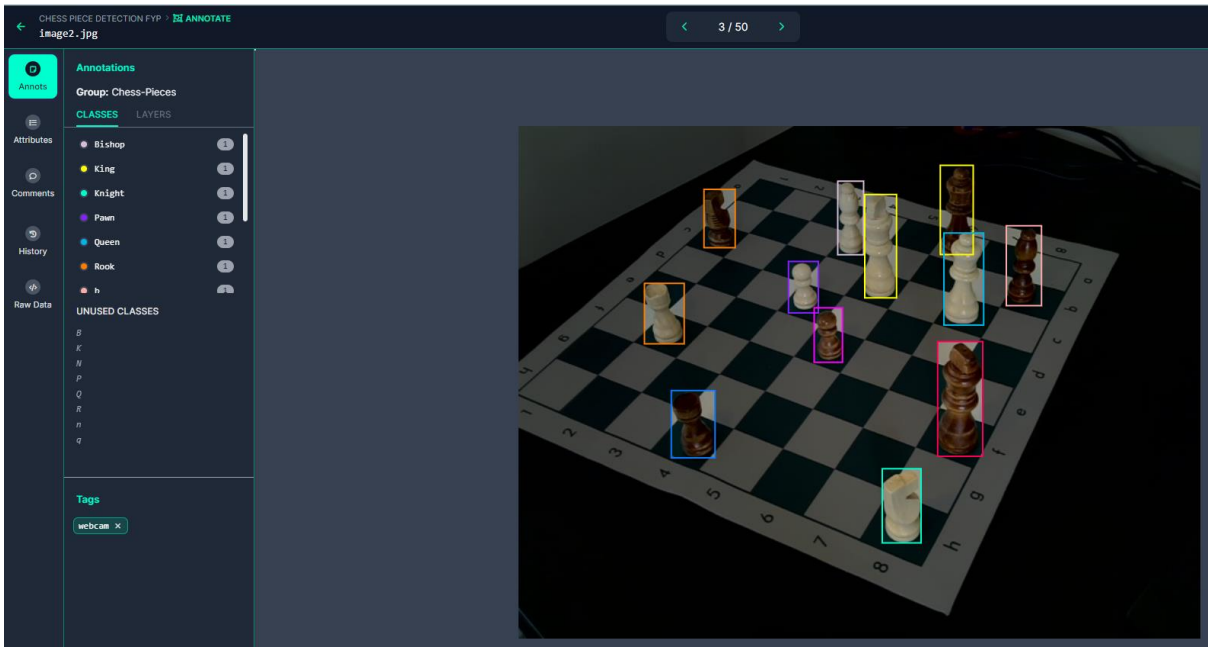



*Figure 24 - An example of dataset annotation in Roboflow*

Another key factor than the number of training instances is having a balanced distribution of annotations for each class. When the classes in the training data are imbalanced, the machine learning model may have difficulty learning to accurately classify the minority classes. This can result in a biased model that performs well on the majority class but poorly on the minority class. As Figure 23 shows, in the V1, V3 and V8, the Pawn classes (‘p’ and ‘P’) had far more instances than the others as there are sixteen pawns on each chessboard.

To address this issue, over-sampling was applied to the rest of the classes and under-sampling applied to the pawn class. This means that some images with too many pawns were

removed from the training dataset and other classes had more images added. Although it was not completely equal, by the V15 dataset, the class balance was far more equal.

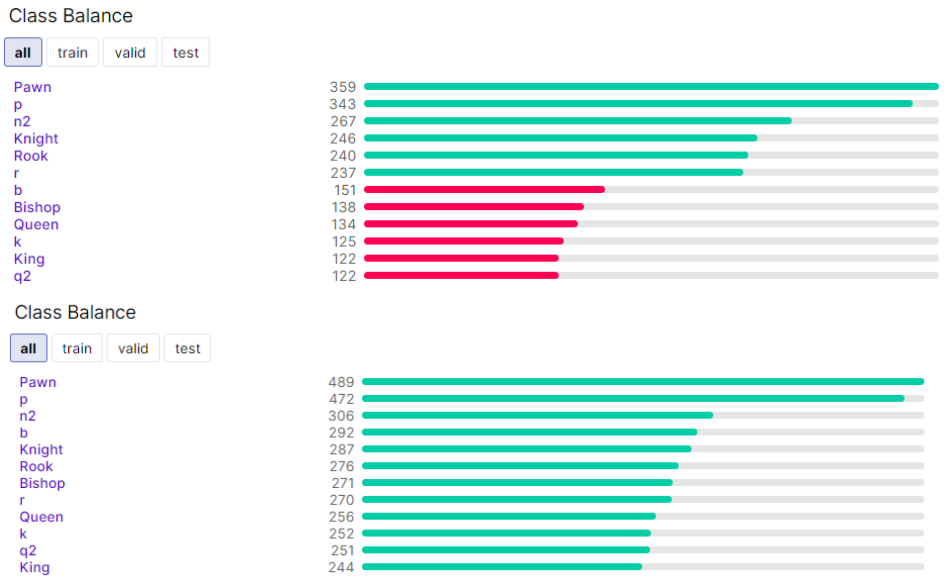



*Figure 25 - Roboflow class balance graphs (V3 above, V15 below)*

In general, as the number of classes increases, the complexity of the classification problem increases, and more training data may be required to achieve good performance.

The five versions of the dataset that I experimented with showed how factors such as the number of images classes pre-processing, and augmentations affected the performance of the models. The first version (V1) demonstrated that having too few images and classes resulted in poor performance regardless of the training method. It also proved that even if the mAP score is high, this does not automatically mean the model works well when deployed to real scenarios. The second version (V3) showed that increasing the number of images and applying diverse augmentations improved the performance significantly. Version 8 revealed that increasing the number of classes without increasing the number of images or the diversity of the data actually reduces the performance despite increasing the training time. The fourth version (V12) proved that improving the quality and diversity of the images as well as applying more augmentations increased the performance. It also showed that ensuring

that the number of instances per class were balanced helped the model train each class equally. The final version (V15) confirmed that reducing the number of classes while keeping the same number of images and augmentations improved the performance further.

**Chess Piece Location and Notation**

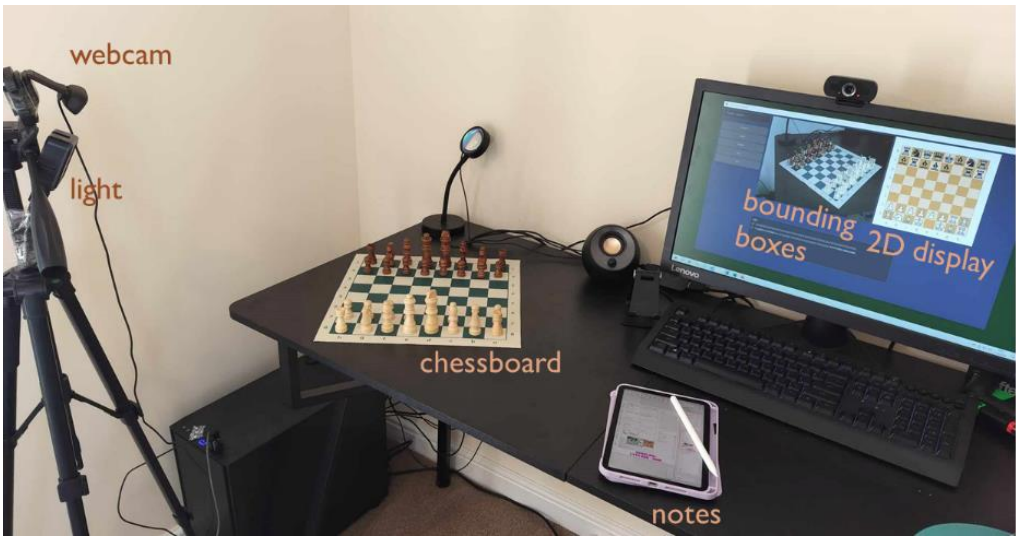



*Figure 26 - Testing setup.*

Having compared each model's metrics and validation results, it was also essential to test the models in use. To do this a test was conducted to evaluate the accuracy of the notation aspect of the chess detection system. In this test, the first thirty moves of a real chess game were recreated using a webcam. The goal of this test was to compare the output notation results generated by the system with the actual moves that were played in the game.

By comparing the output notation results with the real moves, it was possible to assess the accuracy of the system in detecting and recording the movements of the chess pieces. This allowed for simple identification of discrepancies or errors in the system's performance.

To automate the test, using the Ultralytics models, the algorithm iterated through each image and compared the predictions from the machine learning models to the known game

state, printing the number of errors and their position as it went. The first test, performed using model V12, had a total of 48 errors across the thirty moves.

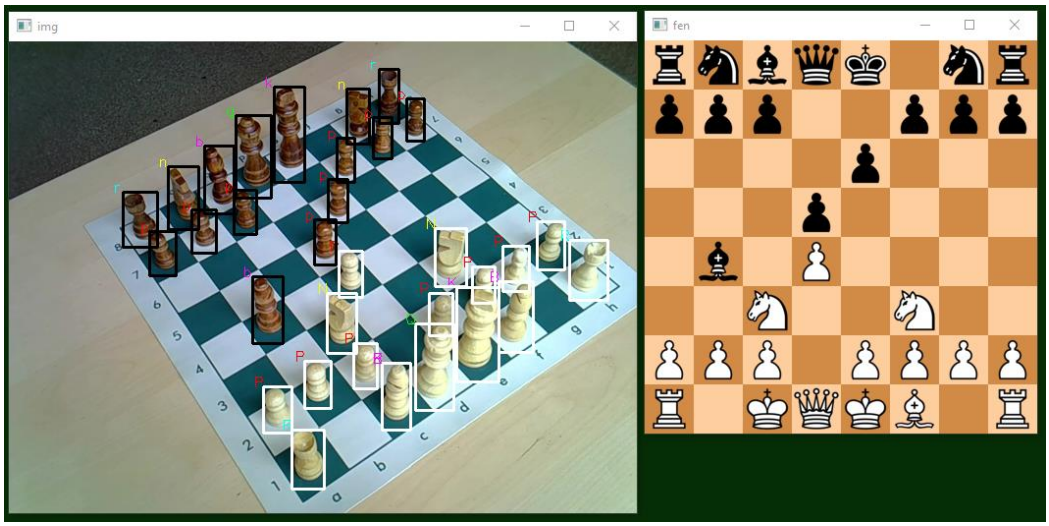



*Figure 27 - Testing the accuracy of the notation and the real game state (V12)*

Upon further investigation, it was determined that although there were positional errors and prediction errors, the majority of errors were caused by incorrect object detection predictions. In order to replicate the test, this chess game was replicated again, with two new sets of images taken in a new location with drastically different lighting scenarios and backgrounds to see if this changed the results, and the test was repeated using all five models trained using Ultralytics hub.

```
./MachineLearning/tests/2/27.jpg V15 Testing
a7: is   should be P
c6: is R should be P
FENCODE: R1BB1RK1/1P111PP1/1BR1P111/111P1N1P/1P1P11NP/P1NBPNP1/11P11P11/R11QK11R

./MachineLearning/tests/2/28.jpg V15 Testing
a7: is R should be P
c2: is B should be P
FENCODE: R1BR11K1/RP111PP1/1BP1P111/111P1N1P/1P1P11NP/P1NBPNP1/11B11P11/R11QK11R

./MachineLearning/tests/2/29.jpg V15 Testing
a7: is R should be P
c2: is Q should be P
f2: is R should be P
FENCODE: R1BR11K1/RP111PP1/1BP1P111/111P1N1P/1P1P11NP/P1NBPNP1/11Q11R11/R11Q1RK1

Test 2 finished: 69 errors.
Model V15 testing complete.
```

*Figure 28 - Terminal output from notation tests*

Each test checked the contents of a total 1920 squares across the thirty images, allowing for an extensive test of each model. A table of the results is below, with the best performing model highlighted in green and the worst in red. As half of the squares on the board are empty at any given time, a model that did not detect a single piece in any square would return a score of 960 errors and be considered 50% accurate. However, 50% accuracy could also indicate a balance of false positives and false negatives.

*Table 6 – Chess Piece Location Accuracy Test*

| | V1 | | V3 | | V8 | | V12 | | V15 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Errors | % Accuracy | Errors | % Accuracy | Errors | % Accuracy | Errors | % Accuracy | Errors | % Accuracy |
| **Test 0** | 930 | 51.56% | 14 | 99.27% | 928 | 51.67% | 24 | 98.75% | 1 | 99.95% |
| **Test 1** | 949 | 50.57% | 256 | 86.67% | 927 | 51.72% | 65 | 96.61% | 24 | 98.75% |
| **Test 2** | 1059 | 44.84% | 402 | 79.06% | 921 | 52.03% | 150 | 92.19% | 69 | 96.41% |
| Total | 2938 | 48.99% | 672 | 88.33% | 2776 | 51.81% | 239 | 95.85% | 94 | 98.37% |

As is indicated in Table 6, the results in general mirror the results from each of the training metrics, with V3, V12 and V15 vastly outperforming V1 and V8.

*Figure 29 - Images from each test*

It is easy to see that V15 outperformed the rest of the tests by quite a margin, with V12 and V3 close in second, although it is important to keep in mind that V1, V3 and V15 only detect six classes so errors that stem from detecting the colour incorrectly are not counted here. One key takeaway is that every model performed worse in test one, which had darker lighting.

It is also important to note that three models do not detect for black and white pieces separately, to identify if the difference in results between V12 and V15 could be attributed to this, the full output from the tests were analysed which revealed that over 50% of the errors in V12 were colour errors. This analysis also showed that Pawns caused the largest number of errors by a large margin, and these errors were typically that an alternative piece has been predicted where a pawn is positioned. This is likely due in part because pawns have less distinct features than the other pieces.

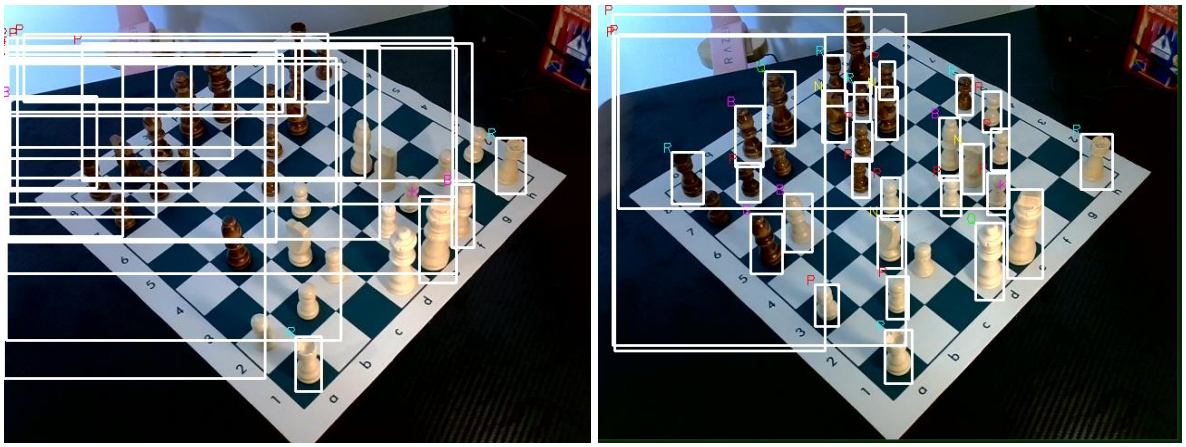


*Figure 30 - V1 and V3 testing.*

This testing system also made it clear that the later models were better at predicting the bounding boxes for the detected pieces, Figure 30 shows how in both V1 and V3 testing, the errors are caused by huge bounding boxes which cause an incorrect analysis of the location on the board. Although the algorithm only considers the bottom third of the bounding box for positional analysis, if a bounding box is predicted wrongly, this localisation method will not work. These events were reduced by model V12, likely due to the annotation adjustment discussed before.

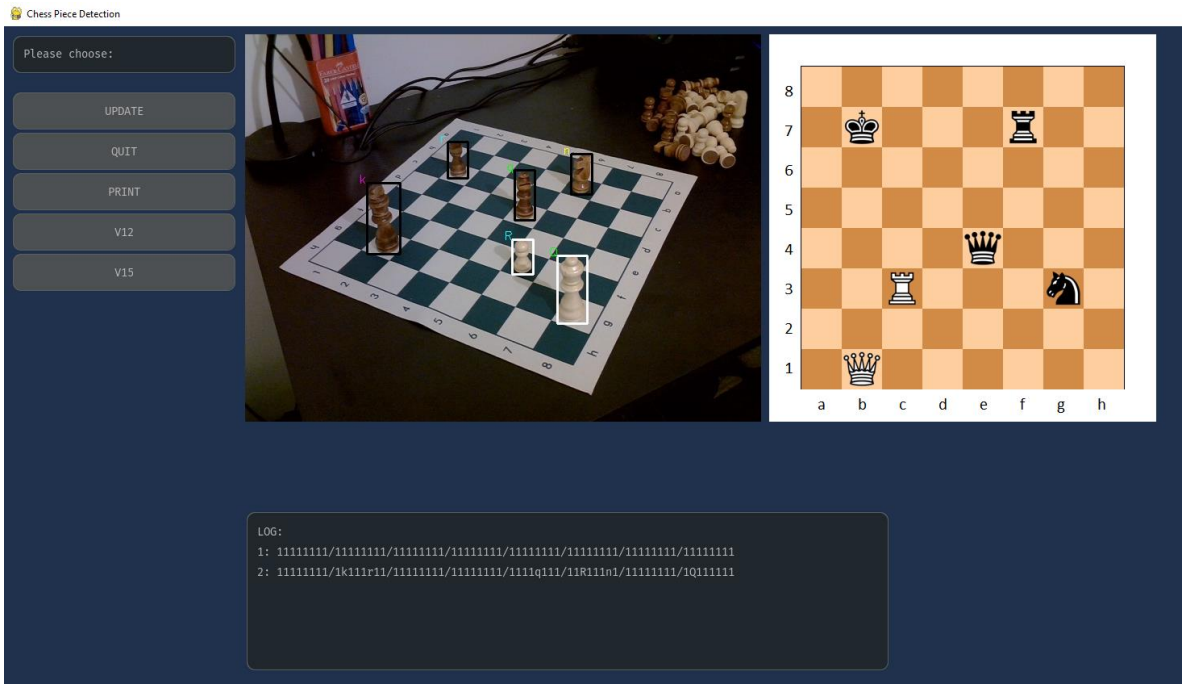



*Figure 31 - The final displayed output.*

Once the testing was complete, the model for the final demonstration was chosen to be V12 run on the Roboflow hosted API. This model was chosen as it produced accurate results under the right conditions and also was one of the models to have twelve classes. This meant that the predictions would be given with black and white pieces separately identifiable, and the FEN image would be able to display the whole game scenario. Figure 31 displays the final product of this project, with a live display of both the predicted bounding boxes and a 2D visualisation of the state of play.

## 5. Future Work

Although all of the initial objectives laid out in this project have been achieved, there are still a few areas that could be improved with some future work. A few suggestions for continuing are detailed below.

Firstly, although the V15 model had a high accuracy rate in testing, the machine learning model could be improved further. One way to do this would be to add more images to the dataset, YOLOv5 documentation recommends “1500 images per class” which would mean increasing the number of images by a factor of five. Although this is not always necessary, as my results show, it would be expected that this would improve the robustness of the model’s performance. As well as increasing the number of images, it is important to note that focusing on improving the diversity in lighting and camera, including images taken from different angles, or with a new chessboard and pieces would also be recommended.

Secondly, developing the software solution to allow for real-time detection would be a beneficial change. This would involve changing from using the hosted Roboflow API, and instead using a locally trained model or an alternative system which runs locally. By increasing the efficiency of the wrapper script, adapting the code to be a multi-threaded solution, or changing the GUI to a faster alternative than `pygame`, the speed could be further improved and allow for real-time detection to be implemented.

Finally, in order to improve the displayed results, simple logical arguments could be implemented to compare the prediction results from the model to the number of pieces that are known to be possible on a chessboard. For example, in this solution, pawns were often mistaken for rooks, however if there can only be a maximum of four rooks on the board at one time, then perhaps only the four rooks with the highest confidence levels should be displayed and the prediction can be run again on the remaining pieces.

## 6. Conclusion

In this project, I created a machine learning model for identifying chess pieces on a chessboard and tracking the state of a chess game. This included producing and annotating a dataset, training a YOLOv5 based machine learning model and deploying it upon live images from a webcam. By using Hough transforms and contour detection, I also successfully designed a chessboard detection algorithm that enabled the conversion of the detected tiles into chess grid references. The graphical user interface I developed uses these tools to translate the state of a real-world chess game into FEN notation, which could then be used to continue a game online.

The model I trained with Roboflow achieved a mean average precision of 98.37% which outperformed the other models trained throughout the project. By combining the CNN model and computer vision techniques for the detection of the chessboard, this solution was able to track the state of a game with an accuracy of 95.6% under good conditions.

However, this implementation of the algorithm also had some limitations. Training on Roboflow restricted the customisation and fine-tuning of the model parameters, while the local GPU trained models were restricted by inefficient hardware. Using the Roboflow platform for the deployed model also had issues as the data transfer to the host servers caused some latency and meant that real-time detection was not possible. To improve the presented solution, a few areas could be improved, these are discussed in more detail in the previous section.

This project shows how machine learning and computer vision can be used to create a robust and efficient solution for chessboard analysis. By combining different techniques and overcoming various challenges, this project demonstrates the potential of applying these technologies to other domains that require similar capabilities.

## 7. References

Bhatt, D., 2022. *A Complete Guide to Hough Transform.* [Online]
Available at: https://www.analyticsvidhya.com/blog/2022/06/a-complete-guide-on-hough-transform/

Bochkovskiy, A., 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection. *ArXiv,* Issue 2004.10934.

Chess Programming Wiki, 2018. *Algebraic Chess Notation.* [Online]
Available at: https://www.chessprogramming.org/Algebraic_Chess_Notation#SAN

Chess.com, 2020. *Fen Chess.* [Online]
Available at: https://www.chess.com/terms/fen-chess

Chess.com, 2023. *Chess Is Booming! And Our Servers Are Struggling..* [Online]
Available at: https://www.chess.com/blog/CHESScom/chess-is-booming-and-our-servers-are-struggling

ChessProgramming Wiki, 2018. *Forsyth-Edward Notation.* [Online]
Available at: https://www.chessprogramming.org/Forsyth-Edwards_Notation

Chopra, E., 2019. *Various Object Detection Techniques.* [Online]
Available at: https://iq.opengenus.org/object-detection-techniques/

Connaughton, K., 2008. *ChessOrb.* [Online]
Available at: http://www.chessorb.com/history-of-computer-chess.html

Czyzewski, M. A., Laskowski, A. & Wasik, S., 2020. Chessboard and Chess Piece Recognition with the Support of Neural Networks. *Foundations of Computing and Decision Sciences,* 45(4), pp. 257-280.

Divvala, S., Redmon, J., Farhadi, A. & Girshick, R., 2016. *You only look once: Unified, real-time object detection..* s.l., IEEE conference on computer vision and pattern recognition..

Dwyer, B. & Nelson, J., 2022. *Roboflow (V1.0) [Software].* [Online]
Available at: https://roboflow.com

Fruugo, n.d. *Staunton Chess Set.* [Online]
Available at: https://img.fruugo.com/product/0/85/948122850_max.jpg

Google, 2022. *Google Colab.* [Online]
Available at: https://colab.google/

Horvat, M., Jelecevic, L. & Gledec, G., 2022. *A Comparative Study of YOLOv5 models performance for image localization and classification.* Dubrovnik, CECIIS Intelligent Systems.

Hsu, F.-h., Campbell, M. S. & Hoane, A. J., 1995. *Deep Blue System Overview.* [Online]
Available at: https://dl.acm.org/doi/pdf/10.1145/224538.224567

IBM, n.d. *What is Computer Vision?.* [Online]
Available at: https://www.ibm.com/topics/computer-vision

Imane, C., n.d. *YOLOv5 Model Architecture.* [Online]
Available at: https://iq.opengenus.org/yolov5/

Jocher, G., 2020. *YOLOv5.* [Online]
Available at: https://github.com/ultralytics/yolov5

Jocher, G., Chaurasia, A. & Qiu, J., 2023. *Ulralytics by Ultralytics.* [Online]
Available at: https://github.com/ultralytics/ultralytics
[Accessed 10 07 2023].

Krawiec, R., 2023. *FenToBoardImage.* [Online]
Available at: https://pypi.org/project/fentoboardimage/

Liu, H., Sun, F., Gu, J. & Deng, L., 2022. SF-YOLOv5: A Lightweight Small Object Detection Algorithm Based on Improved Feature Fusion Mode. *Sensors,* 22(15), p. 5817.

Meituan, 2022. YOLOv6: A single-stage object detection framework for industrial applications. *ArXiv,* Issue abs/2209.02976.

Montgomery, A., 2013. *Daniel Weil redesigns the chess set.* [Online] Available at: https://www.designweek.co.uk/issues/march-2013/daniel-weil-redesigns-the-chess-set/

O'Mahony, N. et al., 2019. Deep learning vs traditional computer vision. *Advances in Computer Vision: Proceedings of the 2019 Computer Vision Conference,* 1(1).

OpenCV, n.d. *OpenCV Tutorials.* [Online] Available at: https://docs.opencv.org/4.x/d6/d10/tutorial_py_houghlines.html

Redmon, J. & Farhadi, A., 2017. s.l., IEEE conference on computer vision and pattern recognition.

Redmon, J. & Farhadi, A., 2018. Yolov3: An incremental improvement.. *ArXiv,* Volume abs/1804.02767.

Shree, L., 2023. *One-Stage vs. Two-Stage Detectors in Object Detection.* [Online] Available at: https://lakshmishreea1.hashnode.dev/one-stage-vs-two-stage-detectors-in-object-detection-unveiling-the-performance-trade-offs

Stamp, J., 2013. *How the Chess Set Got Its Look and Feel.* [Online] Available at: https://www.smithsonianmag.com/arts-culture/how-the-chess-set-got-its-look-and-feel-14299092/

The Chess Journal, 2021. *How Many Chess Players Are There in The World? (Finally Solved).* [Online] Available at: https://www.chessjournal.com/how-many-chess-players-are-there/

Ultralytics, 2020. *Ultralytics YOLO Docs.* [Online] Available at: https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results/

Ultralytics, 2023. *Ultralytics Hub.* [Online]

Available at: https://ultralytics.com/hub

Ultralytics, 2023. *YOLOv5 Docs.* [Online]

Available at: https://docs.ultralytics.com/yolov5

Wang, C., Bochkovskiy, A. & Liao, H. Y. M., 2023. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors.* s.l., Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.

Wikipedia, 2018. *Staunton Chess Set.* [Online]

Available at: https://en.wikipedia.org/wiki/Staunton_chess_set

Zhao, Z.-Q., Zheng, P., Xu, S.-t. & Wu, X., 2019. Object Detection with Deep Learning: A Review.. *IEEE Transactions on Neural Networks and Learning Systems,* 30(11), pp. 3212-3232.

## 8. List of Tables and Figures

# 9. Appendices

## A1 - Project Management

The objective of this project is to design and implement a software system that can identify and represent the state of a chess game from a webcam or image input. The project consists of four main tasks that are essential to achieve the desired outcome: (1) detection of the chessboard as the region of interest; (2) detection of the chess pieces on the board; (3) estimation of the positions and types of these pieces; and (4) visualisation of these components in a 2D format. By combining these aspects, this project will provide a valuable tool for chess enthusiasts and learners to examine and improve their skills by accessing any chess game using a camera device.

Chess Piece Recognition: A Computer Vision and Machine Learning Approach

Rachel Ireland-Jones

| | |
|---|---|
| Project Start Date: | 27 April 2023 |
| Project End Date: | 14 August 2023 |
| Today's Date: | 19 June 2023 |
| Time Left: | 1 months, 13 days |

Key: Finished; Ongoing; Meeting – 28 March (10am)

| Semester 1 | % Done | Proposed Start | Proposed End | Actual Start | Actual End |
|---|---|---|---|---|---|
| **Project Planning** | 100 | **27-Mar** | **10-Apr** | **27-Mar** | **17-Apr** |
| 1 Supervisor Decision | 100 | 27-Mar | 27-Mar | 27-Mar | 27-Mar |
| 2 Project Title | 100 | 27-Mar | 10-Apr | 01-Apr | 07-Apr |
| 3 Aims and Objectives | 100 | 27-Mar | 10-Apr | 01-Apr | 16-Apr |
| 4 Project Proposal | 100 | 27-Mar | 10-Apr | 01-Apr | 17-Apr |
| **Research** | 90 | **10-Apr** | **15-May** | **15-Apr** | **09-Jun** |
| 5 Chessboard Detection | 60 | 10-Apr | 01-May | 15-Apr | 15-May |
| 6 Chess Piece Detection | 100 | 10-Apr | 01-May | 17-Apr | 15-May |
| 7 Dataset Labelling | 100 | 17-Apr | 10-May | 01-May | 09-Jun |
| 8 Machine Learning | 100 | 20-Apr | 15-May | 01-May | 09-Jun |
| **Design** | 68 | **01-May** | **15-Jul** | **05-May** | **05-Jun** |
| 9 Chess Piece Dataset | 80 | 01-May | 05-Jun | 10-May | |
| 10 Chess Piece Detection Software | 80 | 01-May | 05-Jun | 20-May | |
| 11 Chess Board Detection | 40 | 01-May | 10-Jun | 05-May | 05-Jun |
| 12 2D Display | 70 | 05-Jun | 15-Jul | 12-Jun | |
| 13 GUI | 70 | 05-Jun | 15-Jul | 01-Jul | |
| **Documentation** | 35 | **27-Apr** | **13-Aug** | **03-Apr** | **09-Aug** |
| 14 Lab Book | 80 | 27-Apr | 10-Aug | 03-Apr | |
| 15 Video | 10 | 25-Jul | 08-Aug | 01-Aug | |
| 16 Viva Presentation | 10 | 20-Jul | 08-Aug | 31-Jul | |
| 17 Final Report | 40 | 01-Jul | 13-Aug | 20-Jul | 09-Aug |

APRIL 27 03 10 17

*Figure 32- A screenshot of the project Gannt chart from 19 June 2023.*

This project was conducted in approximately four months, from April 2023 until August 2023, and although the objectives and methods developed throughout the process, a project management plan was developed and followed throughout this time. Large parts of this management were completed in digital notebooks focusing on the initial plan and then

weekly summaries and notes. However, other tools such as a Gantt chart and kanban tables were also used. To manage my time efficiently, I used a Gantt chart throughout this project. This is a technique to visualise and monitor the activities and progress of the project over time. I divided the activities into four categories: project planning, research, design, and documentation. Each category consisted of several subtasks that were also tracked.

In addition, I used the Gantt chart to identify and resolve dependencies between tasks. For example, I realised that as I was struggling with the chessboard detection element of the project, I had not been able to start the chess piece location phase as this was a dependency. To continue making progress, I decided to separate these issues and work on them independently, creating an alternative chessboard location solution that did not rely on computer vision. This way, I could advance in each area individually and combine the two methods together when they were complete.

One of the major challenges I faced in this project was creating accurate and reliable algorithms for chess piece and chessboard detection. These elements both took longer than expected to complete due to software and hardware limitations. This resulted in multiple ‘false-starts’ where I had to completely restart the attempt with a different software or method. Although this was frustrating and affected my project plan, it also helped me understand the task better and enhance my Python programming and experience with multiple software options.

**A2 - A History of YOLO**

(Jocher, et al., 2023)

| | |
|---|---|
| YOLO | A popular object detection and image segmentation model, V1 was developed by Joseph Redmon and Ali Farhadi at the University of Washington. Launched in 2015, YOLO quickly gained popularity for its high speed and accuracy. |
| YOLOv2 | Released in 2016, improved the original model by incorporating batch normalisation, anchor boxes, and dimension clusters. |
| YOLOv3 | Launched in 2018, further enhanced the model's performance using a more efficient backbone network, multiple anchors, and spatial pyramid pooling. |
| YOLOv4 | Released in 2020, introducing innovations like Mosaic data augmentation, a new anchor-free detection head, and a new loss function. |
| YOLOv5 | Further improved the model's performance and added new features such as hyperparameter optimization, integrated experiment tracking and automatic export to popular export formats. |
| YOLOv6 | Open sourced by Meituan in 2022 and is in use in many of the company's autonomous delivery robots. |
| YOLOv7 | Added additional tasks such as pose estimation on the COCO key points dataset. |
| YOLOv8 | The latest version of YOLO by Ultralytics. As a cutting-edge, state-of-the-art (SOTA) model, YOLOv8 builds on the success of previous versions, introducing new features and improvements for enhanced performance, flexibility, and efficiency. YOLOv8 supports a full range of vision AI tasks, including detection, segmentation, pose estimation, tracking, and classification. This versatility allows users to leverage YOLOv8's capabilities across diverse applications and domains |

| MODEL | RELEASE DATE | AUTHOR(S) | PAPER/SOURCE |
|---|---|---|---|
| YOLOV1 | 2016 | Joseph Redmon, et al. | You only look once: Unified, real-time object detection. (Divvala, et al., 2016) |
| **YOLOV2** | 2017 | Joseph Redmon, Ali Farhadi | YOLO9000: Better, Faster, Stronger (Redmon & Farhadi, 2017) |
| **YOLOV3** | 2018 | Joseph Redmon, Ali Farhadi | YOLOv3: An Incremental Improvement (Redmon & Farhadi, 2018) |
| **YOLOV4** | 2020 | Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao | YOLOv4: Optimal Speed and Accuracy of Object Detection (Bochkovskiy, 2020) |
| **YOLOV5** | 2020 | Glenn Jocher and his team at Ultralytics | YOLOv5 (Jocher, 2020) |
| **YOLOV6** | 2022 | Meituan Inc. (Chuyi Li et al.) | YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications (Meituan, 2022) |
| **YOLOV7** | 2022 | WongKinYiu and Alexey Bochkovskiy (AlexeyAB) | YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors (Wang, et al., 2023) |
| **YOLOV8** | 2023 | Glenn Jocher at Ultralytics | YOLOv8 (Jocher, et al., 2023) |

## A3 - System Requirements

This appendix provides the details of the hardware and software that were used to run the python software project developed in this dissertation.

### *Hardware Requirements*

- PC with a quad-core Intel Core i5 or similar.
- GPU (GeForce NVIDIA 1060 SUPER)
- A high-speed network connection

### *Software Requirements*

- Windows OS
- Python 3.1
- Pip 22.3.1

The project also depends on the following packages, which can be installed using pip:

- pygame and pygame_gui
- cv2
- numpy
- math
- IPython
- fentoboardimage
- roboflow
- comet_ml
- yaml
- tensorboard

And the following modules, which can be installed from GitHub.

- ultralytics
- YOLOv5