School of Computing
FACULTY OF ENGINEERING AND
PHYSICAL SCIENCES

Final Report

Chessboard State Prediction with Multitask Learning

Calum Wallbridge

Submitted in accordance with the requirements for the degree of Computer Science

2021/22

<Module code and name>

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Final Report	PDF file	Uploaded to Minerva (DD/MM/YY)
<Example> Scanned participant consent forms	PDF file / file archive	Uploaded to Minerva (DD/MM/YY)
<Example> Link to online code repository	URL	Sent to supervisor and assessor (DD/MM/YY)
<Example> User manuals	PDF file	Sent to client and supervisor (DD/MM/YY)

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Student) _____

© 2021/22 The University of Leeds and Calum Wallbridge

Summary

<Concise statement of the problem you intended to solve and main achievements (no more than one A4 page)>

Immediatley explain what you've added compared to other implementations.

Acknowledgements

<The page should contain any acknowledgements to those who have assisted with your work. Where you have worked as part of a team, you should, where appropriate, reference to any contribution made by other to the project.>

Note that it is not acceptable to solicit assistance on 'proof reading' which is defined as the "the systematic checking and identification of errors in spelling, punctuation, grammar and sentence construction, formatting and layout in the test"; see
`https://www.leeds.ac.uk/secretariat/documents/proof_reading_policy.pdf`

Contents

Chapter 1

Introduction and Background Research

1.1 Introduction

Algorithms such as Deep Blue [1], AlphaZero [] and more recently Player of Games[] have enabled computers to out smart the smartest humans at the game of Chess. Unfortunately all these algorithms are bound to the digital world, rendered useless when competing against humans on a real board. This project aims to explore a major component of this: vision.

Consider the vision problem for chess to be two-fold: what is the current board state and where are all of the pieces? With this information, in combination with the previous algorithms and a robot arm, the computer is no longer bound to the digital world. In particular this project will focus on the former, that is, to produce and present a solution for determining the state of a chess board from a video stream. A solution reliable enough to live up to the likes of AlphaZero in a robotic system. Such a solution could be immediately useful for problems such as chess analysis from a real board.

There will be a focus on deep learning techniques, with consideration for best practice in operations. There is also the aim to share the tools to more easily manage and create new datasets in this area. Something called for by [] as a serious challenge and priority for future research.

1.2 Literature Review

For humans the hard part of chess is planning, this is not the case for computers, instead recognition and localisation of objects in 3D space, along with manipulation, present much greater challenges. []

Make reference to humans huge allocation of resources to vision. [] Why is it so hard for computers then? It's an inverse problem. Compare to solving the decision problem (minimax). The statistal calculation of whether to trade Queen's or block with a pawn has now become trival.

1.2.1 A Short History of Computer Vision

Classical Techniques

How we can use classical techniques to understand images. Neural networks have taken over almost all of the heavy lifting for high level inference. Give an indication of time scale here.

Image Recognition

the way back in 1980 [] convolutions showed promise in simple computer vision tasks, convolutions since have showed extreme promise [] Le Ye Cunn's work with convolutions [] and MNIST [] and more recently ImageNet [] having become incredibly well known. New methods such as Transformers from the world have NLP have generalised the convolution operation have proved very successful and lots of work here has been applied to vision. [] [] [] (Attention is all you need, ViT, generic model from deepmind)

Object Detection

But in most applications there will be many things we want to recognise in an image. The RCNN [] enters. How Faster-RCNN improves on this []. Why YOLO [] has been so successful (realtime). Transformers are applicable here too.

Instance Segmentation

Why bounding boxes are not enough. What is segmentation? Why instance segmentation is what we actually want. []

Adding More Dimensions

The real world is not percepived in static 2d images. How do we add an understanding of 3 dimensions and time in our computer vision models? [] Important for localisation in the real world. Important for understanding things like object permenance.

1.2.2 Tools

Dataset Collection??

Chess Engine??

Deep Learning Library??

1.2.3 Computer Vision for Chess

Despite chess being a very narrow application of computer vision, the amount of research effort gone into the problem of determining board state is not insignificant. A variety of approaches have been tried and tested for which the following section will attempt to fairly summarise.

As in [] we will futher split the vision problem into two further problems for analysis: board detection and piece recognition.

Board Detection

The problem of board detection is not specific to chess but also receives heavy research from other applications such as camera calibration []. The built in camera calibration functions in

opencv [] and matlab [] are used in many previous works [] which provide a quick and precise solution for board detection, but becomes unusable when any pieces are present on the board. This forced those authors to take the approach of an initial setup stage at inference, making the solution unfit to changes in board position during inference.

Due to a chessboard's simple features many early works of line and corner point detection can be applied. For example Hough transforms [] are used to detect the lines of a chessboard []. Corner point detection methods such as the Harris and Stephens's [] were also common among solutions [], with some authors combining approaches with further processing such as canny edge detection [] to yield more reliable results.

ChESS was another corner detection algorithm that out performed the Harris and Stephen's algorithm []. This was, perhaps interestingly, created for real-time measurement of lung function in humans, further demonstrating the attention chess board detection has received due to its general applicability.

There are many other algorithms that require simplifications such as green-red chessboards [], multiple camera angles [], or even user input for entering the corners of the chessboard [].

The most impressive work came out of Poznan University of Technology which proposes many interesting ideas that perform more reliably in a wider range of difficult situation such as pieces being present on the board []. They employ an iterative approach with each iteration containing 3 sub-tasks: line segment detection, lattice point search and chessboard position search. In each iteration of line detection a canny lines detector [] is used on many preprocessed variations of the input image to maximize line segment detections which are then merged using a linking function. The lattice point search starts with the intersection of all merged lines as input, converting these intersection points to a 21x21 pixel binary image of the surrounding area and runs them through a classifier to remove outliers. The addition of a neural network as a classifier greatly improves the generality of the proposed solution as it can be resistant to lattice points that are partially covered by a chess piece. The final sub-task then creates a heatmap over the original image that represents the likelihood of a chessboard being present. Under the hood this is done by calculating a polyscore for the set of most likely quadrilaterals formed by the lines of the first stage. The polyscore is a function of the area of the quadrilateral and the lattice points contained within it. It is the quadrilateral that produces the highest polyscore that is used to crop the image for input to the next iteration until the quadrilateral points converge.

Piece Recognition

Piece recognition has proved more difficult []. Most chess vision systems avoid classifying pieces by type all together []. These approaches typically get around this by requiring the board to start in a known position. From this known state the normal rules of chess can be used to infer what pieces are where after each move. Simpler methods require human input to prompt when a move has been taken [], more sophisticated attempt to do this move detection automatically.

These automatic move detection methods tend to all follow the same overarching processes of thresholding, whether on color [] or even the edges detected with each square []. Most authors recognise the dependance this approach has lighting variations, with a few deciding to use Otsu

thresholding [] to minimise the negative impact when lighting changed. While this improved results for what may be considered normal lighting condition they still suffered.

the reference color of all 4 variations (white square, black square, white piece, black piece). All of these then only work in situation where a series of moves are to be recorded, not the chessboard state at any given moment. In other words, given the standard chess starting FEN and a piece with moves being detected instead using normal chess rules.

There were a couple of methods that stood out from the rest in different ways. One used fourier descriptors to model the pieces and the other modelled the pieces in a 3d modelling software and used template matching to determine piece type. Fourier method was very sensitive to change in angles, prefferring a side view angle that unfortunately cause too many occlusions to be practical. The template matching approach took over 150 seconds on average to predict board state from one image which does not lend itself to interactive play.

Go to standford dude and the heatmap guys as the best approach out there. They use SIFT and hard coded color alogrithms. heatmap guys improved on this only by adding more restrictions by assuming the board much be valid and making statistical assumptions on what state is most likely. Oh and a HOG method. The one that said SIFT didn't work well because the lack of texture.

More recently another group of methods have surfaced using neural networks, specifically convolutional neural netowrks (CNNs) []. One of these used a pretrained Inception-ResNet-v2 model [] and only had 6 classes, resorting to the more tradition approaches for color detection, in particular binary thresholding with added morphological transformations to reduce noise as seen in previous works []. Interestingly the six chosen classes were 'empty', 'pawn', 'knight', 'bishop', 'rook' and 'king_or_queen' as they claim kings and queens can be difficult even for human eyes to distinguish. Because of the choice of classes this method falls back to relying on a chess engine to determine piece type, which while usually correct for normal games of play makes the method unusable for games played with a variation on the normal rules of play. The other two methods used a simpler CNN structure similar to that of VGG [] with 13 classes, one for every piece and the empty square.

1.2.4 Prior Work From the Author

Mention robotic arm for two counter board games and automatic differentiation library.

Chapter 2

Methods

<Everything that comes under the 'Methods' criterion in the mark scheme should be described in one, or possibly more than one, chapter(s).>

2.1 Data Collection

At the heart of any machine learning project is the data. It is as important, often more important, than the code and presents many interesting challenges. **Why is this?** Discussed in the following sections are some of the challenges and decisions that were considered.

2.1.1 Sensors

The eminent challenge is acquiring data in the first place. This is highly context dependant, but as vision is primarily focused on spatial awareness the discussion will be limited to the sensors that can measure it.

Sensor choice is an important choice for any robotics application as there are important tradeoffs, as with any engineering challenge, which must be considered.

Outline some of the tradeoffs between spatial sensors

One important distinction to make is the difference between training and inference. Requirements at the time of training my differ significantly to the requirements at inference. Processing power, energy supply and realtime operation are some of the constraints that will have to be met when considering different sensors.

Talk about single camera.

The sensor used throughout this project is the RealSense SR305 which is a RGB-D camera using structured and coded light to determine depth, it functions best indoors or in a controlled lighting situation. For the reasons outlined above the RGB camera stream is mainly relied upon but there will be some discussion and comparision of piece detection with the depth sensor.

Talk about the generic camera

2.1.2 Auto-Labelling

Talk about using a simulator.

A closely related challenge of acquiring the data is that of labelling it too. During Literature Review multiple past authors have stated the availability of datasets for chess piece recognition is sparse [] with some emphasizing dataset collection took the large majority of their time []. It is also widely known that neural networks scale with the number of examples [], which will be

explicitly explored for Chess Vision in Results. This however poses the question: how do we get access to a lot of labelled data for chess?

Unlike techniques in [] **Some examples of other auto labelling techniques** the approach taken here was to maximize speed of collection and flexibility.

Portable Game Notation (PGN) is a common format for recording chess games as a series of moves and are widely available online. All the PGN files used in the project were used from [] which has over one million games. Utilising this data not only has the benefit of an abundance of chess games but also that the recorded games are real and contain positions more likely to appear in game play.

A program is developed for recording these games with the generic camera interface as previously described. The program takes screenshots upon user input (with the [Enter] shortcut) displaying the move number and image as a result for visual feedback before saving to disk. These games can then be automatically labelled using the matching PGN file.

After the development of this pipeline it was possible to collect over 2,500 *unique* labelled images in under two hours.

2.1.3 Dataset Versioning

With all this data the next challenge becomes self evident. It is concerned with the question: How do we manage all of this? Some of the problems... and why you need versioning... Transitioning from git lfs to aws s3. Perhaps a quick mention of other solutions. How the Game and Labeller classes solves some of these challenges for us.

2.2 Model Architecture

Given this data what is are model meant to do.

2.2.1 Experiment Tracking

Express importance of experiment tracking. Some of the solutions [] and their tradeoffs. Why guild was chosen and how it was used.

2.2.2 Board Segmentation

Aruco markers are chosen for board corner point detection as a very simple method. With the corner points of the board the perspective transformation is calculated using Gaussian elimination [**?**], as demonstrated in **??**. Although Aruco markers require customizing the environment they are very fast at inference and allow the focus to remain on piece recognition, from which Literature Review showed is less studied and reliable. In a more holistic solution the iterative heatmap approach proposed in [**?**] would be recommended.

2.2.3 Piece Recognition

Now that the board has been segmented including all of it's squares, it is time to layout the approach for how to determine what piece, if any, occupy each square. To start, a good baseline is found. A good baseline is a simple model to understand and easy to get decent results with. For classification, the pathological baseline could be a random model, which could easily be extended to be weighted by count of examples for each class in the training set. [Add mathematical formulation for demonstration (categorical / multinomial distribution)]

This is common practice in exploratory machine learning [] so that the transitions made are always from a known and working state. It becomes very easy to see if an experiment is not working and easy to go back.

Keeping to this strategy, the multilayer perception (MLP) or fully connected network [] was the first neural network to be explored. By starting with the MLP, all complexity from the network is stripped away so the more extraneous elements such as the training loop and evaluation metrics can be built and tested. [perhaps mention purposeley overfitting] Once the full training and evaluation structure is functional, new features can be incrementally added and architectures explored.

As mentioned in Literature Review convolution operations, and in particular differentiable convolutional operations have had monumental impact on the field of computer vision and so this was the next experiment. Different architectures such as ResNets [], ConvNext [] and ViT were explored as well.

Quite quickly, especially with a limited dataset, overfitting becomes a major problem to overcome and so many experiments were positioned To solve this problem. Pooling, dropout layers and skip connections are amoung some of these.

ConvNext https://arxiv.org/pdf/2201.03545.pdf

Optimizer

An important part of any neural network training loop is the optimiser and so we will explore the effect of swapping these out.

[put into results] AdamW was found produce more attractive results from comparing SGD with/without momentum and Adam and AdamW [] Learning Rate, Weight Regularization.

Batch Normalization

https://arxiv.org/abs/1502.03167 and renorm for small batch sizes
https://arxiv.org/abs/1702.03275

Dropouts

https://arxiv.org/abs/1207.0580 shouldn't be used after convolution layers [] there has been some work https://arxiv.org/abs/1904.03392

Pooling and DropPaths

Transfer Learning

There appears to be a trend occuring in the deep learning space. Some organisation spends millions training an impossibly large nerual network and others more and more are using these models, often fine-tuning for their own use cases. [] uses these large models as fixed feature extractors.

This appraoch makes sense as it is inpractical to retrain huge nerual networks that take weeks, millions of dollars and wasteful amounts of energy to train [].

In the case of CNNs we can see the features that kernals in the early layers learn [] are often very simple shapes and will be common for all computer vision tasks. This will explored this further in the results section as we visualise kernals from both random initalised models and pretrained models.

Multitask Learning

2.2.4 Augmentation

Data augmentation is strategy every machine learning practioner wants to have in their aresonal. It addresses the data problem, allows us to truly leverage the data we have and generalise our models further. The MINST [] dataset itself was created using data augmentation. In the case of chess piece classification, the correct augmentations or transformations should be chosen. Go onto list the transformations used and why.

2.3 Recording a Chess Game to PGN

The goal of this section is to discuss methods for using the proposed piece classifier to record an entire game of chess, played on a real board, to a PGN formatted file.

The general approach is to generate a board state at each fetched frame. That is to segment the board and each of it's squares, send each square through a forward pass of the piece classifier described above and finally collate the predictions together into a board state. A board state is a generic term that in actuality could be many things, but in this case it is sufficient enough to think of it as Forsyth-Edwards Notation (FEN). This board state can then be compared with the previous board state and if any difference is detected then that move is added as a child node. This tree structure gives flexibility for many variations of the same game to be recorded to later be parsed and encoded to PGN.

2.3.1 User Input

2.3.2 Chess Engines

As leveraged by others before [] a chess engine and other statistical methods [] can be used to increase the reliability of board state prediction when the normal rules of chess can be assumed to be abided by. The piece classifier described above makes no such assumptions, but as the output of this module is a PGN file (which in itself assumes the normal rules of chess??) it is

safe to assume them here without loss of generality. In this case leveraging the rules of chess means that we assume the starting state of the board and only allow legal moved to be recorded.

To do this, two concepts are introduced: *VisionState* and *BoardState*. With some simplification, VisionState is a lightweight representation of board as the model sees it in the last fetched frame. The BoardState starts at a known state and is then only updated if the VisionState at any time step represents a legal move from the BoardState. When the BoardState reaches a terminal state or otherwise receives a cancel signal in cases of a draw or resignation the game tree is parsed and the PGN saved to disk.

TALK about why

To add even more redundancy the VisionState has a memory of length N, where memory is in an average of states over the last N frames.

2.3.3 Motion

One factor that was found to still sometimes break this system was motion. Moving pieces across squares and hands flailing over the board confused the model. Even with the separation of the BoardState and memory to remove anomalous predictions - especially when motion persisted for longer periods.

In a lot of these situations the actual board state in undefined. That is because a piece has been lifted and so most now be moved but not yet let go and so it's definition may be undetermined. Because of this is it not unreasonable to halt inference all together. User input to indicate when a move has been completed is a common strategy [], but goes against the purpose of building such a system all together - autonomy. Instead a motion detector is employed. Even the naive motion detector of using a threshold over the absolute difference between each consecutive frame was found to be sufficient for removing these disturbances. Some other methods such as SIFT and SURF were also explored but found to be more computationally expensive that necessary.

Chapter 3

Results

3.1 Model Evaluation

Throughout development, every change was monitored and evaluated. As expressed in Methods some particular design decisions and tools greatly helped at making that possible.

3.1.1 Board Segmentation

The decision to use aruco markers, while impractical for release, aided development in a couple of ways. Firstly, they're reliable []. To correctly classify pieces, the segmentation of board squares was critical as a small error of even 10mm could completely throw the model off as due to added perspective shift a square could now contain two pieces. Compared to the iterative heatmap method [] aruco marker are not the best, but in a controlled environment this did not become a problem and more importantly they worked even with pieces on the board which enabled recalibration during inference and while collecting data. This is unlike a lot of other proposed solutions []. Secondly they're fast []. The iterative heatmap method can take around 5 seconds to segment the board which while not terrible can add up when relabelling lots of data for many model experiments.

As this project focused on piece recognition and the development of an inference system, aruco markers proved to be a sensible choice. However, they come with deal breaking consequences if this method is to be used in production with many boards. It's just too impractical to print and fix markers every time a new board is to be used with the system and again goes directly against the main purpose of this project: autonomy.

Time comparison for the 3. And compare accuracy of heatmap against aruco.

3.1.2 Multitask Learning

How does it compare?

3.1.3 Piece Recognition

Use top-1 and top-5 to measure performance of different models. Will include basic evaluation. What happens what you increase layers, use more data, data augmentation. Include Recall / Specificity / Sensitivity

A benefit of having the depth sensor is an easier way to detech piece presence. Fixed threshold vs clustering. Adding a margin. How we actually did it. Using paried T-test to evaulate.

Using a neural network instead as an additional class with our piece recognition network. Compare that two having a two stage network, the first for piece detection and the second for recognition.

3.1.4 Hyperparameter Tuning

Augmentation as well? Talk about grayscale (a lot of previous works used gray scale do performant CNNs aid from gray or hinder?)

3.1.5 Deep Dive into CNNs

Visualising convolutional layers to analyse effectiveness.

3.2 Realtime Analysis

Frames per second. Problems.

3.2.1 Trials

table of end-to-end experiments

3.3 Comparison to Existing Solutions

Chessboard 1

Method	Accuracy
proposed	94%
[]	78%
[]	65%

Talk about speed of inference and any other limitations of both presented and other work.

Chapter 4

Discussion

<Everything that comes under the 'Results and Discussion' criterion in the mark scheme that has not been addressed in an earlier chapter should be included in this final chapter. The following section headings are suggestions only.>

4.1 Conclusions

4.1.1 Piece Recognition

4.1.2 Dataset Management

4.2 Ideas for future work

Firstly it would be nice to explore these methods with more extreme camera angles. This would probably include extending the labeller too add more margin in one direction to account for the perspective. If assumptions about the environment are to be kept minimal then localising pieces in 3 dimensional space should be a requirement for robotic manipulation to be possible.

It is the author's belief that "3d reconstruction" would kill two birds with one stone and the direction that future research should focus.

Explain. Similar work in other areas.

References

[1] D. Parikh, N. Ahmed, and S. Stearns. An adaptive lattice algorithm for recursive filters. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 28(1):110–111, 1980.

Appendix A

Self-appraisal

<This appendix should contain everything covered by the 'self-appraisal' criterion in the mark scheme. Although there is no length limit for this section, 2—4 pages will normally be sufficient. The format of this section is not prescribed, but you may like to organise your discussion into the following sections and subsections.>

A.1 Critical self-evaluation

A.2 Personal reflection and lessons learned

Surpised at how effective transfer learning is and the significance of it's place in the future. The importance of experiment tracking for research.

A.3 Legal, social, ethical and professional issues

<Refer to each of these issues in turn. If one or more is not relevant to your project, you should still explain *why* you think it was not relevant.>

A.3.1 Legal issues

A.3.2 Social issues

A.3.3 Ethical issues

A.3.4 Professional issues

Appendix B

External Material

<This appendix should provide a brief record of materials used in the solution that are not the student's own work. Such materials might be pieces of codes made available from a research group/company or from the internet, datasets prepared by external users or any preliminary materials/drafts/notes provided by a supervisor. It should be clear what was used as ready-made components and what was developed as part of the project. This appendix should be included even if no external materials were used, in which case a statement to that effect is all that is required.>