JOURNAL OF
EMERGING INVESTIGATORS

Article

Augmented Reality Chess Analyzer (ARChessAnalyzer): In-Device Inference of Physical Chess Game Positions through Board Segmentation and Piece Recognition using Convolutional Neural Networks

Anav Mehta[1] **and Huzefa Mehta, PhD**
[1]Cupertino High School, Cupertino, CA, USA

SUMMARY

Chess game position analysis is important in improving one's game. One method requires entry of logged moves into a chess engine which is cumbersome and error prone. A quick and effective method for analysis is hence strongly desired. Our hypothesis was that a faster chess game entry method can be built using a combination of vision and machine learning techniques to be analyzed directly in a chess engine. To test the hypothesis, we developed the Augmented Reality Chess Analyzer (ARChessAnalyzer), a complete pipeline from a live image capture of a physical chess game, to board and piece recognition, to move analysis and finally to augmented reality overlay of the position and best move on the physical board. The chess position predictor is like a scene predictor - it is an ensemble of traditional image and vision techniques and image classifier for chess board recognition and Convolutional Neural Network (CNN) for chess piece recognition. ARChessAnalyzer was used - as the input mechanism - to compare against manual entry into StockFish which was also the chess engine used in the app. The results validate the hypothesis that both for sparse and dense chessboard populations, ARChessAnalyzer was faster than manual entry with p-value < 0.005. This app and technologies underneath will help chess learners improve their game and hopefully will be widely used in chess clubs.

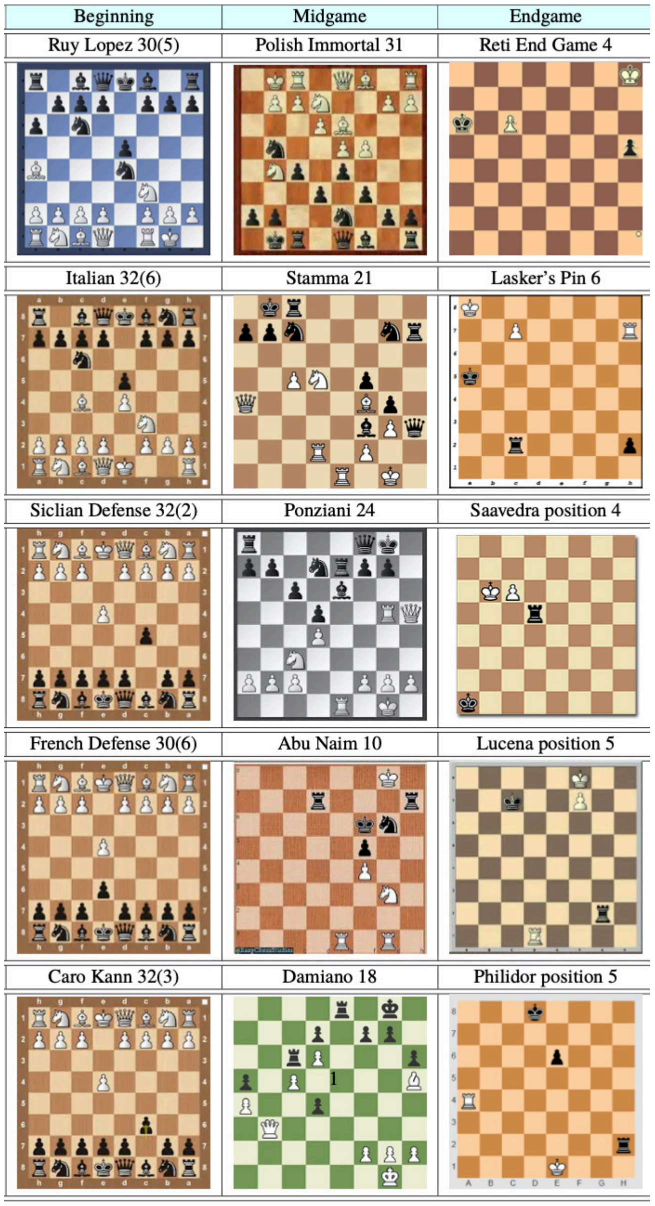
INTRODUCTION

For a player to improve one's chess game, it is important to record the moves, so that one can analyze the game later perhaps in the chess club either with a coach or using an online chess engine. A chess engine is a program where one manually enters a chess position and that analyzes chess positions, and generates the best move. Examples of chess engines are StockFish, Houdini or Komodo to name a few. The recording of moves during a chess game is a tedious manual task that is error prone and impedes the flow of the game and hinders efficient use of time. Then during post-game analysis, the handwritten moves have to be translated into the chess engine which is also error prone. Instead, imagine if one can capture a live image of the game, detect the board and all the pieces, predict the board position and analyze it to provide immediate feedback of the best move available to the player. In an age when much analysis and storage of chess games are done on computers, chess players at all levels can benefit from the ability to analyze a game immediately by taking a picture of a real-life board, as opposed to manual input. Advances in powerful algorithms and hardware computing units have allowed for deep learning algorithms to solve a wide variety of tasks which were previously deemed difficult for computers to tackle. A type of artificial neural network, called a Convolutional Neural Network (CNN), has demonstrated capabilities for highly accurate image classification after being trained on a large data set of samples (1).

Our hypothesis was as follows: a handheld app built leveraging advances in vision and machine learning will be faster than manual entry into Stockfish for a variety of opening, midgame and endgame positions from a simple live capture or a photograph. To test this hypothesis, the Augmented Reality Chess Analyzer (ARChessAnalyzer) was developed that uses a combination of techniques, using a combination of traditional vision technologies OpenCV (2), which segments the board, and a trained machine learning model AlexNet (3), which recognizes the pieces from the segmented board. The output of the piece detector is a Forsyth–Edwards Notation(FEN) (4) position string that is used by the popular chess engine StockFish (5) for analysis, and finally the engine overlays the best move along with the chess diagram on the physical board. This ensemble of algorithms was integrated into an iOS mobile app that is an augmented reality chess analysis engine. The app provides immediate feedback and helps chess players with their game.

RESULTS

To verify the hypothesis that ARChessAnalyzer was fater than manual entry, our experiments were set up as follows. An online chess analyzer (http://chess.com/analysis) which uses StockFish as its backengine was chosen for manual entry. A variety of chess diagrams which included five well known opening, midgame positions in chess literature were chosen from http://www.chess.com and their Wikipedia pages **(Figure 1).** Openings were defined as having atmost 6-7 moves or deletions from the starting board, endgames were defined as having atmost 7-8 pieces on the board and all other positions were defined as midgames. Midgames were more complicated for manual input than opening and endgame positions. Representative images of the app and manual entry of StockFish are shown in **Figure 2**. The average, standard deviation and *t*-test results were calculated and tabulated for each chess diagram **(Table 1)**. We confirmed our hypothesis that ARChessAnalyzer was faster than manual entry with statistical significance of *p*-value < 0.005.



The number in () indicates the number to be moved

Figure 1: Beginning, Midgame and Endgame Chess Diagrams (from Wikipedia) and Number of Pieces in Each Diagram

Entry Times (sec)						
Diagram	StockFish			ARChessAnalyzer		
n	Opening	Midgame	Endgame	Opening	Midgame	Endgame
1	15.21	347.92	15.90	3.20	3.22	3.40
2	18.42	222.79	12.45	2.60	3.12	3.50
3	16.80	107.91	32.67	4.50	4.25	3.30
4	17.24	100.87	15.32	3.50	2.78	2.20
5	15.56	96.24	25.67	2.90	3.23	3.10
Average(μ_0/μ_a)	16.65	175.15	20.40	3.34	3.32	3.10
Std Dev(σ_0/σ_a)	1.30	109.98	8.48	0.73	0.55	0.52
t-test results				18.23	311.82	32.99

Table 1: Chess Diagram Entry Times for StockFish and ARChessAnalyzer

DISCUSSION

The results clearly support the hypothesis that using ARChessAnalyzer reduces the analysis time across all chess diagrams. The *t*-test result for degree of freedom four for area under one tail was greater than 4.604 which corresponds to a *p*-value < 0.005. The time for manual StockFish entry was the greatest for mid-game positions **(Table 1)**. In a manual entry, one starts by modifying the board with a full board or empty board. The time for entry is related to the number of changed pieces from the starting board position of entry which can be a full or empty board. The opening chess diagram can be attained fastest with a full board and moving or removing the pieces. For end chess games the fastest entry was starting with an empty board and populating the board. For midgame diagrams one starts with a full board and pieces are moved or deleted. Hence, midgames take the most time, while openings and endgames almost have the same time for manual entry. For ARChessAnalyzer, there was no difference in entry times between opening, midgame and endgame positions. However, ARChessAnalyzer had the most time advantage (100x) compared to manual entry in midgames. For openings and endgames, the time advantage was 10 times faster.

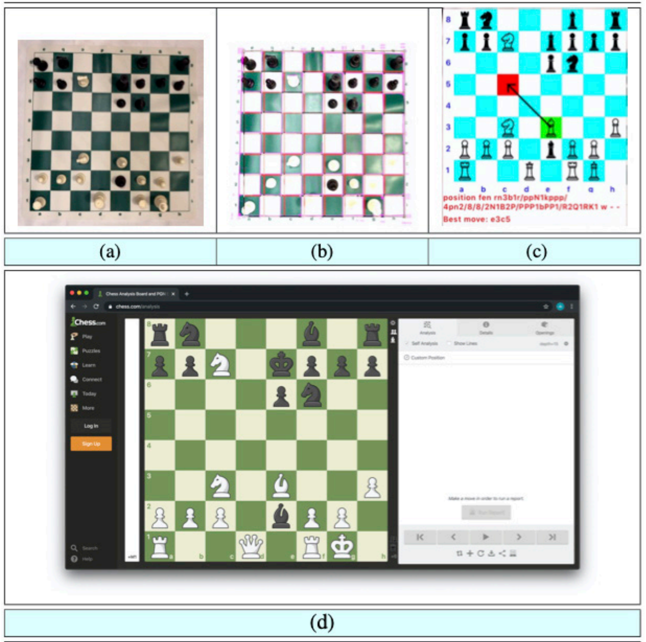

position fen rn3b1r/ppN1kppp/4pn2/8/8/2N1B2P/PPP1bPP1/R2Q1RK1 w - -
Best move: e3c5
(a)
(b)
(c)
(d)

Figure 2: Screenshots of ARChessAnalyzer App (a) Chessboard Detection (b) Canny Edge and Houghline Segmentation (c) Augmented Reality Overlay and (d) StockFish.

The following are a few things to consider going forward. First, the experience of the person in manual entry in a chess engine like StockFish is an important variable. The experiments were done by the author who is well experienced with StockFish. Further, ARChessAnalyzer beat manual entry by 10-100x. Our hypothesis would have been further strenghtened by choosing manual entry by people with all levels of chess engine experience.

Second, accuracy was a variable which was eliminated in our hypothesis. The accuracy of ARChessAnalyzer is the function of the board population. Empty squares have the highest prediction accuracy. Hence, endgames have a higher accuracy (95.24%) than midgames. which in turn have a higher accuracy (93.47%) than openings (91.02%). The overall accuracy of ARChessAnalyzer was 93.45% and it is currently being improved using better segmentation techniques and improved CNN models. It was hard to ascertain the accuracy of a human entry, since an error can be corrected on an observation before the final chess diagram comparison. An hypothesis can however be designed to prove that ARChessAnalyzer accuracy was better than human entry into StockFish. A human error can be defined to be either picking a wrong piece or placing a piece in a wrong position anytime during the entry. For accuracy, we will also need to gather larger samples and hence we would need repeated tries per diagram by multiple people with different levels of chess engine experience. We can then tabulate the errored entries from both methods and compare.

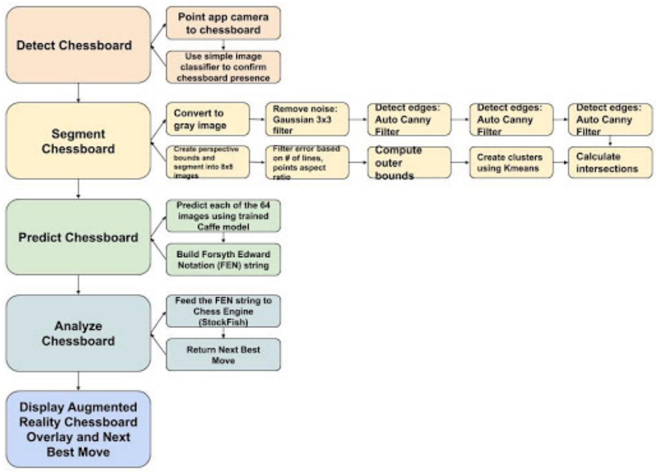

Detect Chessboard
Point app camera to chessboard
Use simple image classifier to confirm chessboard presence
Segment Chessboard
Convert to gray image
Remove noise: Gaussian 3x3 filter
Detect edges: Auto Canny Filter
Detect edges: Auto Canny Filter
Detect edges: Auto Canny Filter
Calculate intersections
Create clusters using Kmeans
Compute outer bounds
Filter error based on # of lines, points aspect ratio
Create perspective bounds and segment into 8x8 images
Predict Chessboard
Predict each of the 64 images using trained Caffe model
Build Forsyth Edward Notation (FEN) string
Analyze Chessboard
Feed the FEN string to Chess Engine (StockFish)
Return Next Best Move
Display Augmented Reality Chessboard Overlay and Next Best Move

Figure 3: ARChessAnalyzer Position Pipeline from Chessboard Detection to Augmented Reality Overlay Display.

MATERIALS AND METHODS

An ensemble of methods and tools with engineering tradeoffs was designed, while advancing the state of the art to develop the entire chess position pipeline in an iOS app **(Figure 3)**. The "Predict Chessboard" step requires generation of a trained model. **Figure 4** describes the model generation pipeline and **Figure 5** describes the preprocessing and model generation steps. This app and its technologies advance the state of the art in robust detection and segmentation of the board and piece detection using a trained AlexNet model. It is also the first of its kind hand held device app which integrates end-to-end integration of these technologies by fine-tuning accuracy and size of the model. The accuracy of the app was 93.45% and it takes 2.5-4.5s from live capture to diagram prediction.

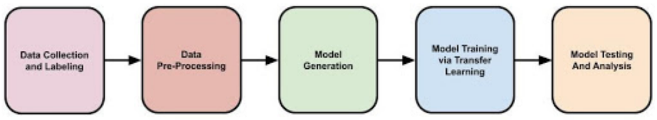

Data Collection and Labeling
Data Pre-Processing
Model Generation
Model Training via Transfer Learning
Model Testing And Analysis

Figure 4: AlexNet Model Generation Pipeline

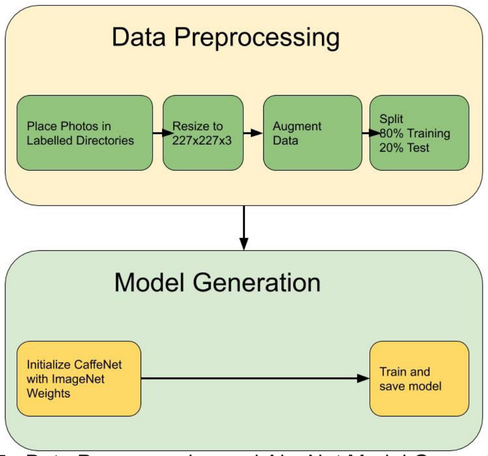

Data Preprocessing
Place Photos in Labelled Directories
Resize to 227x227x3
Augment Data
Split 80% Training 20% Test
Model Generation
Initialize CaffeNet with ImageNet Weights
Train and save model

Figure 5: Data Preprocessing and AlexNet Model Generation.

Position Detection Pipeline

The following are the steps in the position detection pipeline:

1. Detect Chessboard: The presence of a chessboard was determined using a simple binary image classifier.
2. Segment Chessboard: OpenCV image and vision techniques are then used to determine the outer bounds of the chessboard and segment the image into 64 pieces (7-10)
3. Predict Chessboard: A pretrained CNN model is used to predict the 64 image and form the position string.
4. Analyze Chessboard: The string is fed into StockFish to determine the next best move.
5. Augmented Reality Overlay: The next move and position are overlaid over the physical chessboard.

Chessboard Model Generation

The following are the steps in chessboard model generation

1. Data Collection and Labeling: A database of approximately 2,600 chess pieces was manually constructed from one tournament chess set, placed on the board and manually labelled with one of 13 classes: white and black of pawn, knight, bishop, rook, queen, king, and empty.
2. Data Preprocessing: To improve the performance of AlexNet, the data set was augmented with transformations such as cropped, flipped and blur. The data was then partitioned into training (80%) and validation (20%) sets.
3. Model Generation and Training: AlexNet (ImageNet (12) 2012 winner), employing Graphic Processing Units (GPUs) to accelerate deep learning (3) was used as the CNN in our app.

a. Transfer Learning: Weights and layers from the original AlexNet were used as a starting point and fine-tuned with pre-processed images with a batch size of 64. Transfer learning (6) leverages the previously learned low level features and requires less data to arrive at a satisfactory CNN.

b. Model Precision: The AlexNet model was fine-tuned with 32b (FP32), but it was reduced down to 16b (FP16) with CoreML tool, during model conversion, to fit in the size of the app (11).

Tools

An iOS development platform with Swift 5 and Xcode with bridges to OpenCV and StockFish was used. AlexNet was trained using Caffe with Nvidia Tesla K80 GPUs on Google Colaboratory using Python 3.7.

Experiment Setup

The author, who is experienced in StockFish, entered the diagrams **(Figure 2)** manually in StockFish. The time from start of the entry to before analysis is recorded using a stopwatch (Horo) on the Mac. If any error was made (i.e. a wrong piece was picked or a piece was placed in a wrong position), that entry was discarded. The best time of two non-errored entries was tabulated. For ARChessAnalyzer, the chess diagram is converted to a position on a physical chessboard and analyzed via the app. The time from start of the live capture of the physical to before the StockFish analysis was recorded. If the final diagram did not match the desired diagram, the entries from both methods were discarded. Only entry (and not analysis) time was considered to remove variables such as the version of StockFish and differences in underlying processors. The hypothesis is formulated with μ0 and μa corresponding to the average time taken for direct entry and ARChessAnalyzer. The null hypothesis H0 ($\mu_a \geq \mu_0$) is defined as entry by ARChessAnalyzer having the same or worse time than manual entry using StockFish and the alternate hypothesis Ha ($\mu_a < \mu_0$) is defined as entry by ARChessAnalyzer having faster time than manual entry using StockFish. The variables of the setup are the chess diagrams, time for StockFish direct entry and time for entry via ARChessAnalyzer. For each chess diagram, the time results of direct StockFish entry and ARChessAnalyzer (and respective average, standard deviation and *t*-results) were tabulated. The *t*-statistic ($t = (\mu_0 - \mu_a)/\sigma_a$) was computed, where σa was the standard deviation. Statistical significance was determined as *p*-value < 0.005 for our t-test. Since our Ha was one tailed, $t > 4.604$ was used for four degrees of freedom from the *t*-distribution table.

Received: December 8, 2019
Accepted: June 15, 2020
Published: July 17, 2020

REFERENCES

1. LeCun Y., Bengio Y. & Hinton G. (2015). ”Deep learning,” *Nature*, vol. 521, pp. 436-444.
2. Bradski G., (2000). ”The OpenCV Library,” Dr. Dobb’s *Journal: Software Tools for the Professional Programmer,* vol. 25, no. 11, pp. 120-123.
3. AlexNet– ImageNet Classification with Deep Convolutional *Neural Networks* (2018). https://neurohive.io/en/popular-networks/alexnet-imagenet-classification-with-deep-convolutional-neural-networks/
4. Edwards Notation, (2020). https://en.wikipedia.org/wiki/Forsyth%E2%80%93Edwards_ Notation
5. StockFish, (2020) https://stockfishchess.org/
6. Karpathy A., (2020) ”Transfer Learning,” *Stanford University.* http://cs231n.github.io/transfer-learning
7. Soh, L., (1997). Robust recognition of calibration charts, in: 6th International Conference on Image Processing and its Applications, *IEE.* https://doi. org/10.1049/cp:19970941, doi:10.1049/cp:19970941.
8. Zhao F., Wei C., & Tanget J., (2011) ”An automated x-corner detection algorithm (axda).” *Journal of Software.* vol. 6(5), pp. 791–797.
9. Harris, C., Stephens, M., (1988). A combined corner and edge detector., in: Alvey vision conference, *Citeseer.* pp. 10–5244.
10. Duda, R.O., Hart, P.E., (1972). Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM* 15, 11–15. http://doi.acm. org/10.1145/361237.361242, doi:10.1145/361237.361242.
11. Wang N., Choi J., Brand D., Chia-Yu Chen, Gopalakrishnan K., (2018). “Training Deep Neural Networks with 8-bit Floating Point Numbers”, *Conference on Neural Information Processing Systems (NeurIPS)*
12. Deng J., Dong W., Socher R., Li L., Kai L. and Fei-Fei L., (2009). ”ImageNet: A large-scale hierarchical image database,” *IEEE Conference on Computer Vision and Pattern Recognition.*