PREDICTING CHESS MOVES USING IMAGES

Sakshi Jaiswal,Mihir Srivastava,GauravSingh, Arshdeep Singh
Department of Computer Science Noida Institute of Engineering andTechnology
Uttar Pradesh, India

Abstract— Chess is old, cerebral, and noble game in which two players compete against one another's mental stamina while separated by a chessboard. We use every aspect of our brains when playing chess. The number of possible movements in the chess game exponentially grows with each move from the initial 20 to the approximately 400 after the first move. After a few moves, the range of possible moves becomes so broad that neither player can imagine every scenario. Nobody imagined that in the future highspeed hardware clocks and sophisticated software would be able to tackle problems of this complexity and even outperform human intelligence.

I. INTRODUCTION

Since the 1950s, the chess game has served as a paradigm for artificial intelligence. The creation of a chess software is regarded as the first distinct challenge in the study of computer and artificial intelligence. This issue has distinct regulations and a distinct objective (capturing King). This complexity is handled by the computer as a tree issue. Each branch of the tree points to the following potentially permissible move, and the root of the tree defines the game's current state. It was believed that if we could usecomputers to solve the chess problem, we could do the same for anyother problem.

Minimax is a method for locating the optimal move (best tree). When combined with alpha beta pruning, the minimax technique [3] prevents minimax from entering branches that are unable to provide results that are superior to those of earlier branches. Even before computers existed, efforts to create chess playing machines were made. 3The first procedure for a general. Purpose computer was developed by American mathematician Claude Shannon and was based on the theoretical chess programmed algorithm.

This concept was developed into what is likely the first-ever chess software, called "TURBOCHAMP," by British mathematician and computer scientist Allan Turing in 1950. However, Turing's programmed could not be run on a computer. The first complete computer decipherable chess software was created in 1957 by a mathematician and IBM employee and ran on an IBM 704 computer. A computer chess program that could defeat world champions took around 40 years to develop. IBM developed Deep Blue, the world's greatest and brightest chess playing engine, in 1997.

The first time a chess program defeated the Chess In a six. Game encounter staged in New York, the first chess program defeated Garry Kasparov, the reigning world champion. Although the development of an autonomous chess. Laying robot would seem to have occurred in the last few years, it actually dates back to the 18thcentury.

The first so called chess playing robot, "The Turk" or "Chess Playing Automata," was introduced in 1770 by a Hungarian engineer by thename of Baron Wolfgang.

The Turk had the honor of defeating historical leaders like Benjamin Franklin and Napoleon Bonaparte.

The Turk held people's attention for 86 years till a fire at a Chinese museum in 1854 led to its destruction. Chess playing robots on an academic and industrial level has been the subject of several significant projects over the past ten years.

Some employed computerized chessboards, while others used the magnetic sensor located beneath each box to detect moves [6-7].

These methods reduce the overall project's complexity by excludingthe image processing component. The goal of the ongoing work is to create a robotic system that can play board games.eness.

A. LITERATURE REVIEW

The first research on chess perception came from the development of chess robots. It uses a camera to detect the movement of human enemies. Such robots are usually
Three classification schemes to determine each square's occupancy and (if occupied) Figure colors [2–7]. Additionally, there are several techniques for recording chess moves from video. The video uses the same strategy [8–10]. However, neither such three-way classification approach. In order to guess the current chess position, knowledge of the previous board state is required (based on). This was derived from his predictions about the occupancy and color of each square).
This information is readily available to chess robots and software that records movements, it’s not for chess recognition systems that take only a single still image as input. Aside from that, these approaches cannot recover if one move is mis predicted, Unable to identify upgrade pieces (piece upgrades occur when a pawn reachesits last rank, in this case, the player must choose whether to go Queen, Rook, Bishop or Knight..

[1] Vision a system that can only see the color of the piece cannot see what it was told to). Many techniques have been developed to address the problem of chess detection. Each piece type (Pawn, Knight, Bishop, Rook, Queen, king) and suits, mostly for the last five years.
Because chess pieces are almost indistinguishable

[2] When viewed from above, the input image is typically taken at an acute angle to the circuit board. Ding [11] relies on a scale-invariant feature transform (SIFT) and a histogram of directions, but Gradient (HOG) feature descriptors for piece classification, Danner et al. [12] also apologize. [13] Arguing that these are inappropriate due to the similarities in texture between chess and chess Instead, apply a stencil adjustment to the contour of the piece. However, Danner et al al. To make this easier, change the board colors to red and green instead of black and white problem (similar fixes have been proposed as part of other systems [3,9]).

[3] Such changes impose unreasonable restrictions on normal chess games. Several other techniques have been developed using CNNs at various stages detection

pipeline. Apologize. Compare template matching approaches and usage.

[4] Note that the CNN as part of the same work only slightly improved the accuracy (although trained with only 40 images per class). Czyzewski et al. [14] Achieve 95% accuracy Checkerboard localization from non- perpendicular camera angles by designing an iterative algorithm Generate a heatmap representing the probability that each pixel is part of the chessboard. Then use the CNN to narrow down the vertices found using the heatmap.

[5] The results of Goncalves et al. [7]. Furthermore, they Ding proposed a CNN-based piece classification algorithm for his SVM-based solution [11]. I don't see any visible improvement, but I can make improvements by thinking about it possible and legal chess positions. Recently, Mehta et al. [15] Implement extensions. We have achieved promising results in a reality app using the popular AlexNet-CNN architecture [16]. CNN has 13 classes (6 characters of each color, Use a data set of 200 samples per class. Despite using overhead from a camera perspective, we achieve an accuracy per square of 93.45% in our end-to-end pipeline (corresponding to an error rate per square of 6.55%), which is - to the best of our knowledge. It represents the current state of the art. A prerequisite for any chess detection system is to beable to recognize chess positions.

[6] Each of the chessboard and 64 squares. Once the four vertices are determined, finding the square in the bird's- eye view image is straightforward and trivial. Simple perspective transformation at other camera positions. Some of the above systems avoid this problem entirely by prompting the user to make choices interactively.

[7] The four vertices [5,7,8,12], but ideally the chess detection system should be able to parse them. Position on board without human intervention. most approachesto automatic chess Grid detection uses the Harris corner detector [3,10] or some form of line detector.

[8] Hough transforms [4,6,12,17–20] also explore other techniques such as template matching [21] and flood- filling [9]. In general, corner-based algorithms cannot do this for accurate detection of grid corners when they

[9] are obscured by parts (such as line-based detection) Algorithms seem to be the preferred solution. Such algorithms are often. The geometric properties of the chessboard allow us to compute the perspective transformation Grid lines [10,13,17] that best match the detected line. A suitable dataset for chess detection

[10] - especially at the scale needed for Deep Learning – is not currently available and is a problem many people perceive [11,14,15]. For this purpose, synthesis of training data from 3D models seems a promising method. Efficiently generate large datasets and eliminate the need for manual annotation. White. [22] Synthesizing Point Cloud Data for Volumetric CNNs Directly from 3D Chess Models Hou [23] uses renderings of 3D models as input. The approach of Weiet al. only works When the chessboard is recorded with a depth camera, Hou presents a chessboard detection system using basic neural networks, not foldable, that achieves an accuracy of only 72%.

Existing System

The first research on chess perception came from the development of chess robots. It uses a camera to detect the movement of human enemies. Such robots are usually

Three classification schemes to determine each square's occupancy and (if occupied) Figure colors [2–7]. Additionally, there are several techniques for recording chess moves from video. The video uses the same strategy [8–10]. However, neither such three-way classification approach. In order to guess the current chess position, knowledge of the previous board state is required (based on). This was derived from his predictions about the occupancy and color of each square).

This information is readily available to chess robots and software that records movements, It's not for chess recognition systems that take only a single still image as input. Aside from that, these approaches cannot recover if one move is mis predicted, Unable to identify upgrade pieces (piece upgrades occur when a pawn reaches its last rank, In this case, the player must choose whether to go Queen, Rook, Bishop or Knight.

Vision a system that can only see the color of the piece cannot see what it was told to). Many techniques have been developed to address the problem of chess detection. Each piece type (Pawn, Knight, Bishop, Rook, Queen, king) and suits, mostly for the last five years. Because chess pieces are almost indistinguishable

When viewed from above, the input image is typically taken at an acute angle to the circuit board. Ding [11] relies on a scale-invariant feature transform (SIFT) and a histogram of directions, but Gradient (HOG) feature descriptors for piece classification, Danner et al. [12] also apologize. [13] Arguing that these are inappropriate due to the similarities in texture between chess and chess Instead, apply a stencil adjustment to the contour of the piece. However, Danner et al al. To make this easier, change the board colors to red and green instead of black and white problem (similar fixes have been proposed as part of other systems [3,9]).

Such changes impose unreasonable restrictions on normal chess games. Several other techniques have been developed using CNNs at various stages detection pipeline. Apologize. Compare template matching approaches and usage.

Note that the CNN as part of the same work only slightly improved the accuracy (although trained with only 40 images per class). Czyzewski et al. [14] Achieve 95% accuracy Checkerboard localization from non-perpendicular camera angles by designing an iterative algorithm Generate a heatmap representing the probability that each pixel is part of the chessboard. Then use the CNN to narrow down the vertices found using the heatmap.

The results of Goncalves et al. [7]. Furthermore, they Ding proposed a CNN-based piece classification algorithm for his SVM-based solution [11]. I don't see any visible improvement, but I can make improvements by thinking about it possible and legal chess positions. Recently, Mehta et al. [15] Implement extensions. We have achieved

promising results in a reality app using the popular AlexNet-CNN architecture [16]. CNN has 13 classes (6 characters of each color, Use a data set of 200 samples per class. Despite using overhead from a camera perspective, we achieve an accuracy per square of 93.45% in our end-to-end pipeline (corresponding to an error rate per square of 6.55%), which is - to the best of our knowledge. It represents the current state of the art. A prerequisite for any chess detection system is to be able to recognize chess positions.

Each of the chessboard and 64 squares. Once the four vertices are determined, finding the square in the bird's-eye view image is straightforward and trivial. Simple perspective transformation at other camera positions. Some of the above systems avoid this problem entirely by prompting the user to make choices interactively.

The four vertices [5,7,8,12], but ideally the chess detection system should be able to parse them. Position on board without human intervention. most approaches to automatic chess Grid detection uses the Harris corner detector [3,10] or some form of line detector.

Hough transforms [4,6,12,17–20] also explore other techniques such as template matching [21] and flood-filling [9]. In general, corner-based algorithms cannot do this for accurate detection of grid corners when they are obscured by parts (such as line-based detection) Algorithms seem to be the preferred solution. Such algorithms are often. The geometric properties of the chessboard allow us to compute the perspective transformation Grid lines [10,13,17] that best match the detected line. A suitable dataset for chess detection - especially at the scale needed for Deep Learning – is not currently available and is a problem many people perceive [11,14,15]. For this purpose, synthesis of training data from 3D models seems a promising method. Efficiently generate large datasets and eliminate the need for manual annotation. White. [22] Synthesizing Point Cloud Data for Volumetric CNNs Directly from 3D Chess Models Hou [23] uses renderings of 3D models as input. The approach of Wei et al. only works When the chessboard is recorded with a depth camera, Hou presents a chessboard detection system using basic neural networks, not foldable, that achieves an accuracy of only 72%.

Proposed Methodology

This section details the pipeline's three main stages:

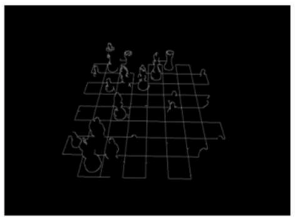
- board localization
- occupancy classification,
- piece classification

and then presents a transfer learning approach for adapting the system to unseen chess sets.

The main idea is we locate the pixel coordinates of the four chessboard corners and warp the image so that the chessboard forms a regular square grid to eliminate perspective distortion in the sizes of the chess squares before cropping them. Then, we train a binary CNN classifier to determine individual squares' occupancies and finally input the occupied squares (cropped using taller bounding boxes) to another CNN that is responsible for determining the piece types. To adapt to a previously unseen chess set, the board localisation algorithm can be reused without modification, but the CNNs must be fine-tuned on two images of the new chess set.

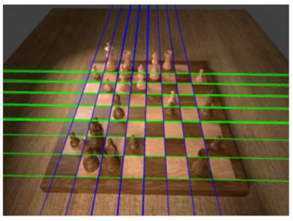
4.1. Board Localization

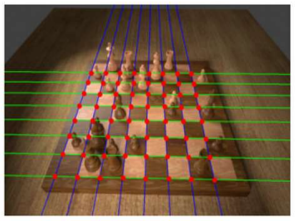
(a) Original image



(b) Detect edges

Figure 2. *Cont.*





(c) Detect lines and cluster as horizontal/vertical (d) Eliminate similar lines and compute intersection

To determine the location of the chessboard's corners, we rely on its regular geometric nature. However, all squares on the physical chessboard have the same width and height. Observed dimensions in the input image vary due to 3D perspective distortion. The chessboard consists of 64 squares arranged in an 8x8 grid, so there are 9 horizontally and 9 vertically Nine vertical lines.

- 4.1.1. Finding intersections - The first step of the algorithm detects most of the horizontal and vertical lines, Find their intersection. Convert the image to grayscale and apply Canny edge detector [25] and the result is shown in Figure 2b. Then run huff transform [26] to find lines formed by edges. There are about 200 lines, most of which are very similar. So split them into horizontal ones and remove vertical lines and remove similar ones. Experiments show this simple setup The line orientation threshold is insufficient to reliably classify them as horizontal or vertical. The camera can be tilted quite a bit, so let's make it vertical. Use aggregation instead.

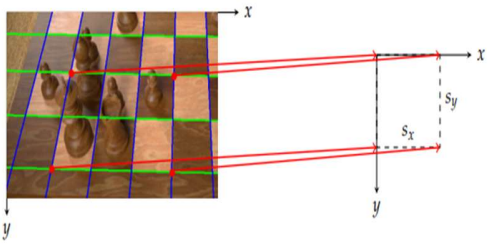
Clustering algorithm (hierarchical bottom-up algorithm

where each row starts on its own) Clusters and pairs of clusters are continuously merged in a way that minimizes dispersion within a cluster), using the smallest angle between two given lines as the distance measure. Finally, the average angle of both top clusters determines which cluster represents which. Vertical and horizontal lines (see Figure 2c). To remove similar horizontal lines, first determine the middle vertical line vertical cluster. Then find the intersection of all horizontal lines with the mean Get the vertical lines and perform DBSCAN clustering [27] to group similar lines based on them intersections, keeping only the middle horizontal lines of each group as the final set Number of horizontal lines detected. Use the same procedure in reverse for vertical lines. Calculate all intersections.

- 4.1.2. Homography calculation - Fewer than nine horizontal and vertical lines are often recognized (Figure 2d), so we need to determine if the additional lines tend to be above or above. Below a known horizontal line (and to the left or right of a known vertical line). Instead of calculating where the candidate lines are in the original image, this is simple.Warp the input image to form a regular grid with square intersections (this is I'll have to do that later to cut the samples for the occupancy and piece classifier) Think about this distorted image because the missing lines are on this grid. This projective transformation is characterized by the homography matrix H that we use. Robust RANSAC-based algorithm even with missing (or extra) rows detected from straight edges elsewhere in the image), described below.

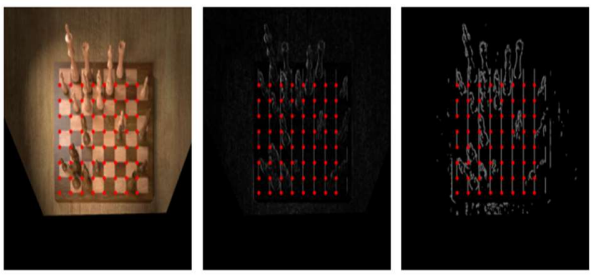
- Four random intersections in two different horizontal and vertical directions.Lines (these points represent squares on the chessboard).
- Compute the homography matrix H that maps these 4 points to a rectangle of latitude sx = 1 and height sy = 1. where sx and sy are the horizontal and vertical scale factors. Shown in Figure 3.
- Project all other intersections with H and count the number of inliers this points described by homographies to a small tolerance γ (i.e. Euclidean The distance from a given distorted point (x,y) to the point (round(x),round(y)) is less than g.
- If the size of the inlier is larger than the previous iteration, keep the inlier Instead, set the homography matrix H.
- Repeat from step 1 for sx = 2, 3, . , 8 and sy=2, 3, ..., 8 for determining the number of chess. A square that encloses the selected rectangle.
- Repeat from step 1 until at least half of the intersections are inside.
- Recompute the least-squares solution of the homography matrix H using everything Identified inliers.

Then warp the input image and the interior intersections according to the computation. Homography matrix H . You should get a result
like Figure 4a. The intersection is each is a chess square, so the x and y coordinates are quantized to be integers It has become a unit length.
Let xmin and xmax be the minimum and maximum distortion values. Similarly, ymin and ymax express the same concept in terms of the x component of the coordinate. Vertical direction.



(a) Region from the original image (b) Warped image

Figure 3. Four intersection points are projected onto the warped grid. The optimal values for the scale factors s_x and s_y are chosen based on how many other points would be explained by that choice, in order to determine the actual number of horizontal and vertical chess squares in the rectangular region from the original image. In this example, the algorithm finds $s_x = 3$ and $s_y = 2$.



(a) Intersection points from Figure 2d warped onto a regular grid (b) Horizontal gradient intensities as computed by the Sobel operator (c) Result of Canny edge detection on the horizontal gradient image

If xmax − xmin = 8, we detected all lines of the chessboard and no further processing is needed. When

Figure 4. Horizontal gradient intensities calculated on the warped image in order to detect vertical lines. The red dots overlaid on each image correspond to the intersection points found previously. Here, $x_{max} - x_{min} = 7$ because there are eight columns of points instead of nine (similarly, the topmost horizontal line will be corrected by looking at the vertical gradient intensities).

xmax − xmin < 8, as is the case in Figure 4a, we compute the horizontal gradient intensities for each pixel in the warped image in order to determine whether an additional vertical line is more likely to occur one unit to the left or one unit to the right of the currently identified grid of points. To do so, we first convolve the greyscale input image with the horizontal Sobel filter in order to obtain an approximation for the gradient intensity in the horizontal direction (Figure 4b). Then, we apply Canny edge detection in order to eliminate noise and obtain clear edges (Figure 4c). Large horizontal gradient intensities give rise to vertical lines in the warped image, so we sum the pixel intensities in Figure 4c along the vertical lines at x = xmin − 1 and x = xmax + 1 (with a small tolerance to the left and to the right). Then, if the sum of pixel intensities was greater at x = xmin − 1 than at x = xmax + 1, we update xmin ← xmin − 1, or otherwise xmax ←

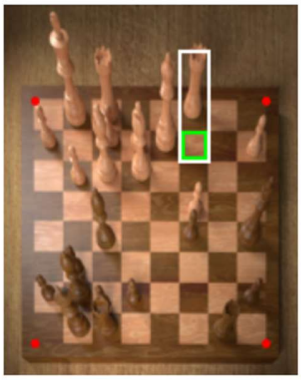
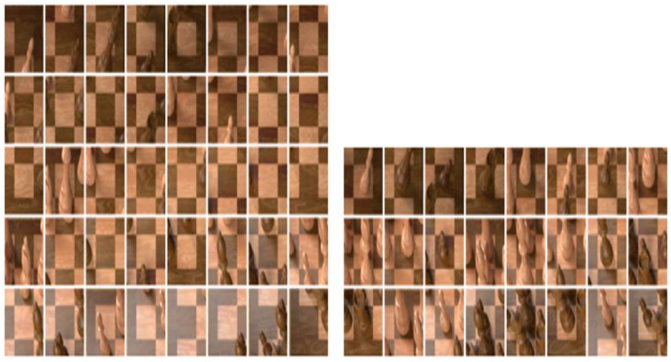


Figure 5. An example illustrating why an immediate piece classification approach is prone to reporting false positives. Consider the square marked in green. Its bounding box for piece classification (marked in white) must be quite tall to accomodate tall pieces like a queen or king (the box must be at least as tall as the queen in the adjacent square on the left). The resulting sample contains almost the entire rook of the square behind, leading to a false positive.



(a) All 40 empty samples (b) All 24 occupied samples

Figure 6. Samples for occupancy classification generated from the running example chessboard image. The squares are cropped with a 50% increase in width and height to include contextual information.

xmax + 1. We repeat this processs until xmax − xmin = 8. An analogous procedure is carried out for the horizontal lines with ymin and ymax. Finally, these four values describe the two outer horizontal and vertical lines of the chessboard in the warped image. The optimal parameters for the Hough transform and Canny edge detectors described in this section are found using a grid search over sensible parameters on a small subset of the training set.

4.2. Occupancy category

You can see that the piece classification is performed immediately after finding the four vertices. Without intermediate steps, you will get a lot of false positive results. H. is an empty square Classified as containing chess pieces (see Figure 5). To resolve this issue, first A truncated square binary classifier to determine if they are empty cut out. The squares in the distorted image are trivial because the squares are the same size (see Figure 6).

We devise six vanilla CNN architectures for the occupancy classification task, of which two accept 100 × 100 pixel input images and the remaining four require the images to be of size 50 × 50 pixels. They differ in the number of convolutional layers, pooling layers, and fully connected layers. When referring to these models, we use a 4-tuple consisting of the input side length and the three aforementioned criteria. The final fully connected layer in each model contains two output units that represent the two classes (occupied and empty). Figure 7 depicts the architecture of CNN (100, 3, 3, 3) which achieves the greatest validation accuracy of these six models. Training proceeds using the Adam optimiser [28] with a learning rate of 0.001 for three whole passes over the training set using a batch size of 128 and cross-entropy loss.

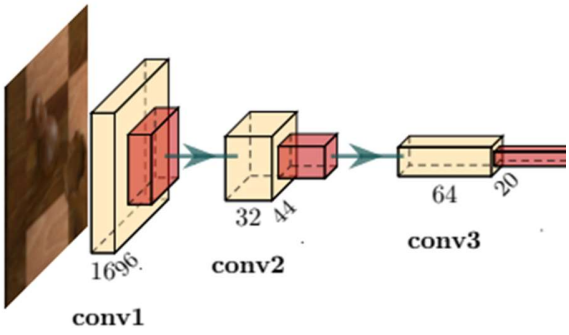


Figure 7. Architecture of the CNN (100, 3, 3, 3) netwo three-channel RGB image with 100 × 100 pixels. The kernel size of 5 × 5 and stride 1 and the final convolu with 16 filters in the first convolutional layer, the num layer. Each convolutional layer uses the ReLU activati layer with a 2 × 2 kernel and stride of 2. Finally, the a 640,000-dimensional vector that passes through two reaching the final fully connected layer with softmax ac

4.3. Piece Classification

A piece classifier takes as input a cropped image of an occupied square and outputs it. A chess piece in this square. There are 6 types of chess pieces (Pawn, Knight, Bishop, Rook, queen, king), each piece can be either white or black, so there are a dozen class. Particular care must be taken when cutting. Right after The approach described in the previous section does not provide enough information to classify the parts.

For example, consider the white king in Figure 5.

Cut only the square it is on top of His crown, which is an important feature, is not included king and queen. Instead, use a simple heuristic that expands the height of the bounds. A box of pieces further back on the board. Also the width depends on its horizontal direction position. As a further preprocessing step, we horizontally flip the bounding box of the part left side of the board to make sure the square in question is always on the bottom left of the board photo. This helps the classifier understand which parts of the sample are referenced where A large bounding box contains a contiguous portion of the image. Figure 8 shows a random one a selection of samples thus generated.

For the final shift of the piece classification task, we train a total of 6 CNNs with 12 output units. For pre-trained models, we follow the same two-step training regime, Double the number of epochs in each phase compared to the previous section. Aside from that, We are evaluating another architecture, InceptionV3 [32], which shows greater optical potential.

This much more demanding task. The remaining two models are the two most powerful models. CNN in the previous section. Select the model with the highest accuracy rating the validation set used by the chess detection pipeline.

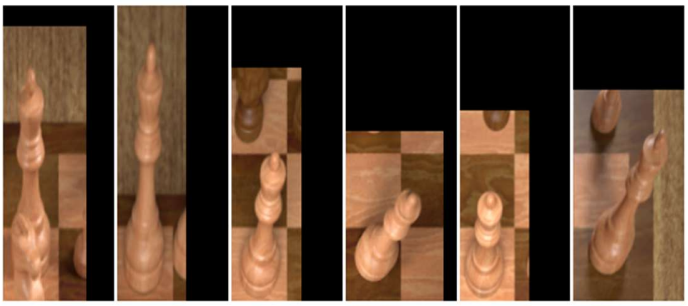


Figure 8. A random selection of six samples of white queens in the training set. Notice that the square each queen is located on is always in the bottom left of the image and of uniform dimensions across all samples.

4.4. Invisible Chess Set Tweaks

Chess sets look different. A CNN trained on a game of chess is likely to work bad for different chess set pictures because the test data is not from the same distribution. Due to the intrinsic similarity of data distributions (both If the chess set and its source and target problems are the same), you can use the form Transfer learning in a few shots. In other words, using only a small amount of data CNN is on newer distributions. The advantage of the board localization approach is that no tweaking is required. A wide variety of chess games because it uses traditional computer vision techniques such as Edge and line detection. Therefore, we can optimize the CNN without tagged records only two

An image of the starting position of the chessboard is sufficient (one from each player's point of view). Since we know the composition of the pieces (the starting position is always the same), Shown in Figure 9. You can find the board and cut out squares using the method outlined In Section 4.1, we generate training samples for fine-tuning the CNN.

Result Analysis

The piece selector's best performance was 38.30%.
The move selector produced a variety of responses. On the chess board, pieces with local movements fared noticeably better than those with global motions. The

The accuracy of the pawn, knight, and king calculations were 52.20%, 56.15%, and 47.29%, respectively. The queen, rook, and bishop, which have global moves, were predicted with 5 considerably less accuracy (26.53%, 26.25%, and 40.54%, respectively).

Table 1. Piece Selection Accuracy

Metric	Accuracy
Piece Selection	38.30%

Table 2. Move Prediction Accuracy

Move Selector	Accuracy
Pawn	52.20%
Rook	29.25%
Knight	56.15%
Bishop	40.54%
Queen	26.53%
King	47.29%

This can be ascribed to the fact that local pieces have fewer spots they can go to as well as the convolution layers' ability in providing relevant characteristics that allow the algorithm to assess the local circumstances around the piece. The first claim is probably accurate. The accuracy of the local pieces is decreased by between 20% and 34% when the second convolution layer is removed, but the accuracy of the global pieces is unaffected.
In contrast, deleting the second affine layer has a similar effect by lowering the accuracy of the local pieces to between 32% and 38% while increasing the accuracy of the global pieces to between 15% and 21%.

Clipping

Because they do not take into account selecting a valid piece or making a lawful move, the accuracy ratings listed above do not accurately reflect the validation accuracy ratings of the model. This is because we intended to evaluate our network solely as a classification task, fully off-model, and without any outside influence whatsoever on the game's characteristics or regulations.
Clipping can only improve validation accuracy by forcing the algorithm to select a right piece or lawful move. Naturally, we saw that the algorithm correctly predicted three times as much when we cut the pieceselector network in the first few epochs, with the effects levelling off by the third epoch and completing convergence by the fourth or fifth.

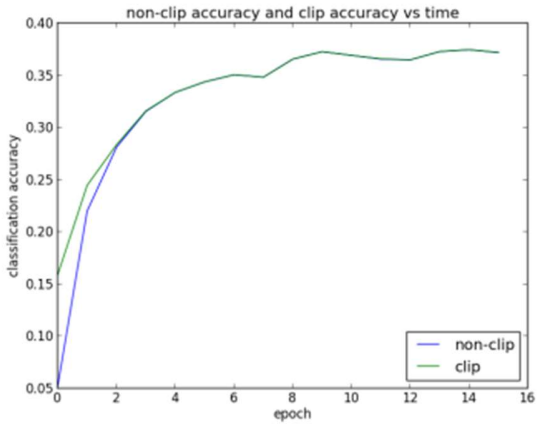


Figure 3. Clipping Vs. Non-Clipping of Illegal Moves

Trade-off between Move Legality and Optimality:

The effectiveness of the network's ability to categorise chess moves as legal raises an intriguing question: How can the network categorise plays as both legal and optimal? Given that it always takes legal action,

Exists a trade-off between a prediction that is favourable and one that is legal? As a result, the moveselector network must also have features and architectures intended to ensure that a legal move is selected, the activations of which may "counteract" those intended to compute the optimal move since the moveselector network lacks context regarding which piece has already been "picked" from the piece-selector. If this were to be further researched, methods for performing the categorization task with an awareness of the game's rules .

Performance, Saturation, and Limitations

As expected, dropping out had no beneficial demonstrable effects on performance. All of the data on the board are linearly important in contributing, as seen by its linear reduction to the accuracy of the whole.

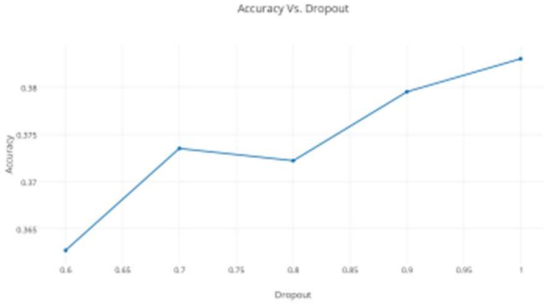


Figure 4. Accuracy Vs. Dropout Rate

One of our study's most unexpected findings is that

Table 3. Dropout Rate Vs. Accuracy

Dropout Rate	Accuracy
0.6	0.362700
0.7	0.373500
0.8	0.372200
0.9	0.379500
1.0	0.383000

Piece dependencies make sure that for better predictions, as much knowledge about the activations and the initial input is required.
When the resulting net was trainable, increasing the size of the network with both affine layers and convolution layers did not improve performance. Pooling did not have the same impact on performance as we had anticipated, suggesting that only a small subset of the attributes may have the most impact. The fact that the filters were doubled served as confirmation.

Figure 5. Accuracy Vs. Regularization

Against The Computer

100 games were played between the Sunfish chess engine and the AI.

games. The other games were lost to the engine, and 26 games ended in draws.

When capturing pieces, the AI makes wise decisions. This is probably due to the fact that pawn defences used in piece captures get quite active when a capture is made with them. There is evidence that the AI learned the fundamentals of attacking. For instance, the AI was observed regularly attempting to "fork" the rook and the queen in games played against the researchers, yet it was always blind to the fact that a king, knight, or bishop was protecting that sector. These "single piece disasters" happen regularly, when a 1-piece variation from prior examples makes a significant impact.

Reference

1. Wölflein, G.; Arandjelovi´c, O. Dataset of Rendered Chess Game State Images; OSF, 2021; [CrossRef] 2. Urting, D.; Berbers, Y. MarineBlue: A Low-Cost Chess Robot. In International Conference Robotics and Applications; IASTED/ACTA Press: Salzburg, Austria, 2003. 3. Banerjee, N.; Saha, D.; Singh, A.; Sanyal, G. A Simple Autonomous Chess Playing Robot for Playing Chess against Any Opponent in Real Time. In International Conference on Computational Vision and Robotics; Institute for Project Management: Bhubaneshwar, India, 2012. 4. Chen, A.T.Y.; Wang, K.I.K. Computer Vision Based Chess Playing Capabilities for the Baxter Humanoid Robot. In Proceedings of the International Conference on Control, Automation and Robotics, Hong Kong, China, 28–30 April 2016. 5. Khan, R.A.M.; Kesavan, R. Design and Development of Autonomous Chess Playing Robot. Int. J. Innov. Sci. Eng. Technol. 2014, 1, 1–4. 6. Chen, A.T.Y.; Wang, K.I.K. Robust Computer Vision Chess Analysis and Interaction with a Humanoid Robot. Computers 2019, 8, 14. [CrossRef] 7. Gonçalves, J.; Lima, J.; Leitão, P. Chess Robot System : A Multi-Disciplinary Experience in Automation. In Spanish Portuguese Congress on Electrical Engineering; AEDIE: Marbella, Spain, 2005. 8. Sokic, E.; Ahic-Djokic, M. Simple Computer Vision System for Chess Playing Robot Manipulator as a Project-Based Learning Example. In Proceedings of the IEEE International Symposium on Signal Processing and Information Technology, Sarajevo, Bosnia and Herzegovina, 16–19 December 2008. 9. Wang, V.; Green, R. Chess Move Tracking Using Overhead RGB Webcam. In Proceedings of the International Conference on Image and Vision Computing New Zealand, Wellington, New Zealand, 27–29 November 2013. 10. Hack, J.; Ramakrishnan, P. CVChess: Computer Vision Chess Analytics. 2014. Available online: https://cvgl.stanford.edu/teaching/cs231a_winter1415/prev/projects/chess.pdf (accessed on 30 May 2021). 11. Ding, J. ChessVision: Chess Board and Piece Recognition. 2016. Available online: https://web.stanford.edu/class/cs231a/prev_projects_2016/CS_231A_Final_Report.pdf (accessed on 30 May 2021). 12. Danner, C.; Kafafy, M. Visual Chess Recognition; 2015. Available online: https://web.stanford.edu/class/ee368/Project_Spring_1415/Reports/Danner_Kafafy.pdf (accessed on 30 May 2021). 13. Xie, Y.; Tang, G.; Hoff, W. Chess Piece Recognition Using Oriented Chamfer Matching with a Comparison to CNN. In Proceedings of the IEEE Winter Conference on Applications of Computer Vision, Lake Tahoe, NV, USA, 12–15 March 2018. J. Imaging 2021, 7, 94 18 of 18 14. Czyzewski, M.A.; Laskowski, A.; Wasik, S. Chessboard and Chess Piece Recognition with the Support of Neural Networks. Found. Comput. Decis. Sci.. 2020, 45, 257–280. [CrossRef] 15. Mehta, A.; Mehta, H. Augmented Reality Chess Analyzer (ARChessAnalyzer). J. Emerg. Investig. 2020, 2. 16. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. Commun. ACM 2017, 60, 84–90. [CrossRef] 17. Tam, K.; Lay, J.; Levy, D. Automatic Grid Segmentation of Populated Chessboard Taken at a Lower Angle View. In Proceedings of the Digital Image Computing: Techniques and Applications, Canberra, Australia, 1–3 December 2008.

18. Neufeld, J.E.; Hall, T.S. Probabilistic Location of a Populated Chessboard Using Computer Vision. In Proceedings of the IEEE International Midwest Symposium on Circuits and Systems, Seattle, WA, USA, 1–4 August 2010. 19. Kanchibail, R.; Suryaprakash, S.; Jagadish, S. Chess Board Recognition, 2016. Available online: http://vision.soic.indiana.edu/b657 /sp2016/projects/rkanchib/paper.pdf (accessed on 30 May 2021). 20. Xie, Y.; Tang, G.; Hoff, W. Geometry-Based Populated Chessboard Recognition. In International Conference on Machine Vision; SPIE: Munich, Germany, 2018. 21. Matuszek, C.; Mayton, B.; Aimi, R.; Deisenroth, M.P.; Bo, L.; Chu, R.; Kung, M.; LeGrand, L.; Smith, J.R.; Fox, D. Gambit: An Autonomous Chess-Playing Robotic System. In Proceedings of the IEEE International Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011. 22. Wei, Y.A.; Huang, T.W.; Chen, H.T.; Liu, J. Chess Recognition from a Single Depth Image. In Proceedings of the IEEE International Conference on Multimedia and Expo, Hong Kong, China, 10–14 July 2017. 23. Hou, J. Chessman Position Recognition Using Artificial Neural Networks. Available online: http://citeseerx.ist.psu.edu/viewdoc/ download?doi=10.1.1.89.4390&rep=rep1&type=pdf (accessed on 30 May 2021). 24. Bilali´c, M.; Langner, R.; Erb, M.; Grodd, W. Mechanisms and Neural Basis of Object and Pattern Recognition. J. Exp. Psychol. 2010, 139, 728. [CrossRef] [PubMed] 25. Canny, J. A Computational Approach to Edge Detection. IEEE Trans. Pattern Anal. Mach. Intell. 1986, 679–698. [CrossRef] 26. Duda, R.O.; Hart, P.E. Use of the Hough Transformation to Detect Lines and Curves in Pictures. Commun. ACM 1972, 15, 11–15. [CrossRef] 27. Ester, M.; Kriegel, H.P.; Sander, J.; Xu, X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In International Conference on Knowledge Discovery and Data Mining; AAAI Press: Portland, OR, USA, 1996. 28. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. In Proceedings of the International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015. 29. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015. 30. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, LasVegas, NV, USA, 27–30 June 2016. 31. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 20–25 June 2009. 32. Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; Wojna, Z. Rethinking the Inception Architecture for Computer Vision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 J