Chess position identification using pieces classification based on synthetic images generation and deep neural network fine-tuning

Afonso de Sá Delgado Neto
Undergraduate Student - DES - UFPE
Recife, Brazil
afonso.delgado@hotmail.com

Rafael Mendes Campello
Undergraduate Student - DES - UFPE
Recife, Brazil
rafael.campello@hotmail.com

***Abstract*—Chess pieces recognition using computer vision is a problem generally approached in various ways, with different kinds of results and complexity. Deep learning is a state of the art approach to solve problems on image recognition although facing necessity of huge data sets. This paper discusses a method to identify synthetically generated chess images on Blender using its Python API via fine-tuning of VGG16 convolutional network obtaining close to 97% accuracy on piece classification. Possible applications include automated record of real chess games and real-time play between online players using real boards.**

***Index Terms*—chess, neural networks, piece recognition, synthetic data generation.**

I. Introduction

Image classification is a task that has changed in complexity over time. In the past such tasks were heavily dependent on feature extraction and hand-design. Nowadays the state-of-the-art is deep learning due to increase in both computational power and insights about architecture of such networks, as seen in [1], [2] and [3].

There are many possible approaches to the problem. Since chess board initial position is fixed, a frequently used idea is to detect only movement of pieces with a computer vision approach and obtain full development of a game without classifying pieces.

In this paper, it is introduced a technique to recognize chess pieces depending on a segmentation of the chess board based on deep convolutional networks. The main advantage of this choice is being able to identify any chess position in any time, not only at the beginning of the game.

Differently of many related proposed work, this paper addresses an oftenly neglected approach of using a top view camera angle, which is a challenge in terms of classifying pieces due to their similarity when they are viewed from this angle. A top view camera angle provides benefits such as no occlusion problems and no need for different angles performance analysis.

It is also proposed two methods to generate the labeled data for the deep learning model, one based on real images of a chess board, with the cost of being slower and human-dependent, and other based on creating simulated images on a 3D model, which is faster but do not fully comprehend the complexity of real images.

However, the second method was used to generate the data. That means, images used to achieve final result were artificially created using 3D rendering of a chess board making possible to obtain a huge data set, which enables full capability of a deep learning approach.

All obtained results, datasets and project development files are available at `github.com/rafaelmcam/cChess`.

The remainder of this paper is structured as follows. Section II describes the works related to the main problem of this research. Section III details how data was generated from both real world chess boards and synthetically generated models. Section IV specifies the topology and training method of the employed neural network. Section V presents the obtained result. Section VI indicates applications for the presented work.

At last, Section VII provides some conclusions and condensates essential and main points regarding the work while its subsection VII-B suggests possible future work that can be done to obtain better results. .

II. Related Work

Piece movement methods such as [4] and [5] obtain above 95% accuracy without needing a training data set. These methods rely on simple algorithms such as tracking of color histogram on each square or edge detection. Although being a simple methodology, it represents great practical use.

When considering random chess positions, initial piece detection must be done. Many possibilities have been published such as [6] and [7], which obtain varying results with very different methods.

In [6] Fourier descriptors are used in segmented binary images with a small training data set of 200 images to achieve close to 100% accuracy. The main difficulty with this method is the choice of camera angle. Since the Fourier descriptor tracks the details of chess pieces shapes it has good accuracy when using camera angles less than $45°$. This choice creates two problems: it subjects pieces to occlusion and increases difficulty to identify all chess board squares. When using top view camera positions this method achieved accuracy close to 50% due to the segmented piece shapes being almost indistinguishable from each other.

In [7] SIFT and HOG feature extraction were used to obtain 60% accuracy for the SIFT detector and 80% for HOG detector. The actual performance of this classification is not well suited for practical use as visualized in the confusion matrix for the test set, as shown in Fig. 2, pieces that are not

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 02,2023 at 15:19:57 UTC from IEEE Xplore. Restrictions apply.

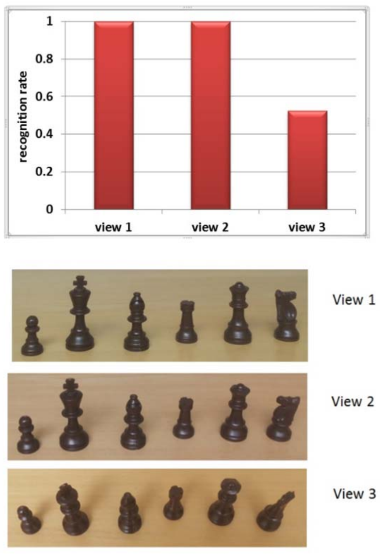
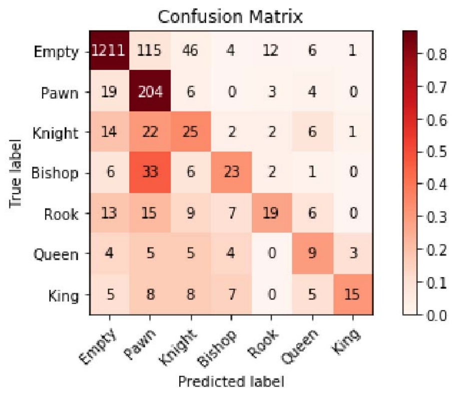


Fig. 1: Classifier not suited to top view. **Source: Cheryl Danner and Mai Kafafy** [6].

pawns or empty squares are mislabelled very frequently, which places a doubt in the meaning of the presented high accuracy.



	Empty	Pawn	Knight	Bishop	Rook	Queen	King
Empty	1211	115	46	4	12	6	1
Pawn	19	204	6	0	3	4	0
Knight	14	22	25	2	2	6	1
Bishop	6	33	6	23	2	1	0
Rook	13	15	9	7	19	6	0
Queen	4	5	5	4	0	9	3
King	5	8	8	7	0	5	15

Fig. 2: Confusion Matrix for **Source: Jialin Ding** [7].

An interesting project is done in [8], a chess playing robot is created using computer vision to detect piece movement and it is also implemented a reduced version of piece classification aiming only to detect possible promoted pawns.

Several related projects merge computer vision and robot development for real world application such as [9] and [10]. Move tracking is also discussed in [11].

Another approach is presented in [12] in a more similar manner to the one proposed in this work, which is also based on a convolutional neural network but uses different procedures for data generation, and obtained different results.

To illustrate a common problem in various proposed methods that use accuracy as only metric for detection validation, suppose a classifier able to distinguish only between empty spaces and any piece. Then, if it is assumed that all pieces are pawns, the classifier performs on a minimum of 75% accuracy if empty squares are considered valid labels and close to 50% if not.

In this work, recall and F1 score [13] will be used to avoid this bias and achieve a more meaningful result.

The work presented in [14] is more similar to this paper in the sense that it also uses neural networks and presents similar results. Main differences are the data acquisition process, real image models and camera angle used.

III. Data Generation

Deep learning heavily relies on the presence of large amounts of data. Getting good (that is, real-life, different and diverse) data is the first and main task in any project based in machine learning. For the task of chess piece recognition it is not different.

However, chess pieces data, mainly labeled chess pieces data, are difficult to find. Although there are some data sets of chess images available in a public way, they lack of diversity, labeling and, most of all, quantity. Also, since the idea for this work is to classify pieces from a top view, such kind of data is even more sparse.

To solve such problem, two ways of generating chess pieces labeled data are introduced. The first one makes use of computer vision and planned chess positions to efficiently (in human level) extract such information. The second uses simulation and rendering to automate the process on the cost of having images that do not follow a real distribution of chess pieces images.

A. Real Images Model

When dealing with real images of chess boards, the objective is to segment and label all the pieces present in the image. Of course, since factors such as camera position, color, brightness and all kinds of environmental effects interfere in the process of getting the pieces and label information, this is not an easy task. Also, it is faced a problem that is actually paradoxical to solve, since the label of the piece is needed, in order to train a model to identify the label of the piece.

Taking these limitations in consideration, it is introduced, in the next paragraphs, a computer vision pipeline to extract pieces location and detect pieces movement, without knowing the label of the piece. The labelling job must be done by a human, in a strategic way.

Let's see how this pipeline works.

1) Board Boundaries: When using a real world chess set, the first decision that must be made is the choice of camera angle and subsequent detection of the chess board boundaries itself. A top view approach was chosen since the beginning of the project. Two simple strategies were used in this paper for boundaries acquisition: Hough Line Detection [15] and user input.

In Hough Line Detection method it was necessary parameters tuning and combination of image processing algorithms

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 02,2023 at 15:19:57 UTC from IEEE Xplore. Restrictions apply.

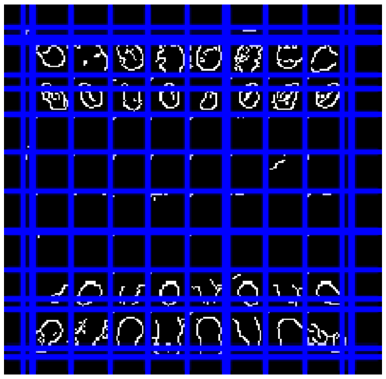


Fig. 3: Hough Lines Detection for board boundaries acquisition.

such as blurring, edge detection and Hough Lines parameters itself. Such a method will work well on a single chess board setting and will require another set of parameters tuning if a different board in a different setting is used.

After obtaining Fig. 3 a possible solution to get all square boundaries is the detection of blue lines intersections followed by applying a clustering algorithm such as K-Means, as shown in Fig. 4, which every blue dot represents a correct square boundary identification and a red dot an incorrect one.

The error at top right of Fig. 4 (in red) can be easily corrected by using the fact that square dimensions are fixed.

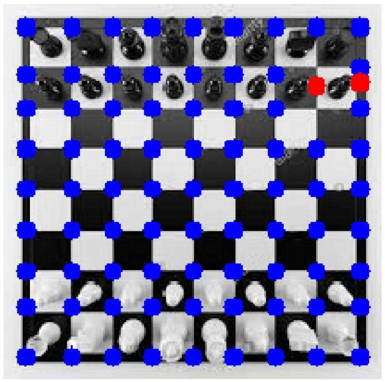
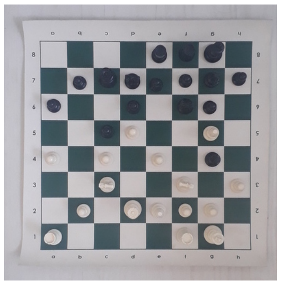


Fig. 4: Square boundaries detection after K-Means. Only two points were positioned incorrectly.

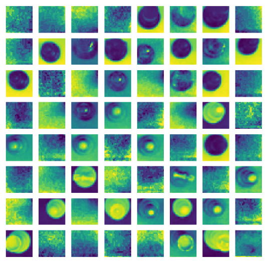
This approach needs to be readdressed every time that the board setting changes (lighting, board colors, piece style).

Additionally to Hough Lines Method, getting user input was also used during this project due to multiple settings being used.

2) Warp Perspective: After obtaining board boundaries it is a simple task to use a geometric transformation provided in OpenCV's [15] functions `getPerspectiveTransform` and `warpPerspective` to get a cropped image and access each square individually, the process visualized in Fig. 5a and Fig. 5b.



(a) Input



(b) Output.

Fig. 5: Individual Squares Access Method.

3) Piece Detection: An auxiliary function that needs to be created is a Piece Detection Function that expects a cropped square image (one square of Fig. 5b) and outputs a Boolean indicating if the square is empty or not. This task is discussed in many similar projects presented in section II and can be done in various ways.

Two approaches were developed, tested and used in this project: color histogram and edge detection.

a) Color Histogram: A possible idea is comparing a single square after a move is made on the board to determine which squares changed by detecting a meaningful change in color histogram. Fig. 6 shows that by representing in blue the grayscale histogram of the left picture and in orange the right one.

If a black piece moves from a white square to another white square this is a very easy detection for this method, as shown in Fig. 6a.

If a piece did not move, it is not expected a meaningful change in its color histogram, as shown in Fig. 6b.

If a black piece moves from a black square to another black square, the detection is more difficult but still feasible, as shown in Fig. 6c.

After obtaining the color histograms as such in Figs. 6a, 6b and 6c, it is possible to compare the similarities of both curves by calculating the cross-correlation of its normalized signals, for a null shifting, which is equivalent to the inner product of the discrete values of the histograms, seen as normalized vectors, as Eq. 1 shows.

$$\textit{Similarity} = \frac{\sum_{n=0}^{255} h_1[n]h_2[n]}{\sqrt{(\sum_{n=0}^{255} h_1[n]^2)(\sum_{n=0}^{255} h_2[n]^2)}} \tag{1}$$

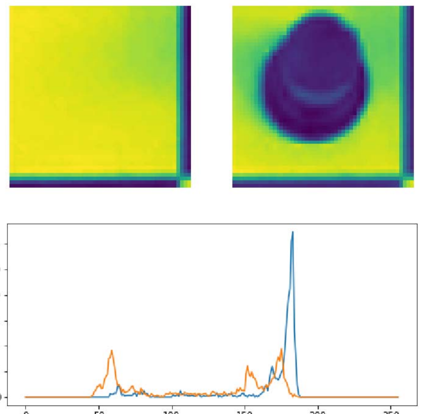
where, $h_1[n]$ and $h_2[n]$ are the histograms values for pixel intensity n.

A major issue with this method is that a big lighting change will affect directly the obtained histograms.

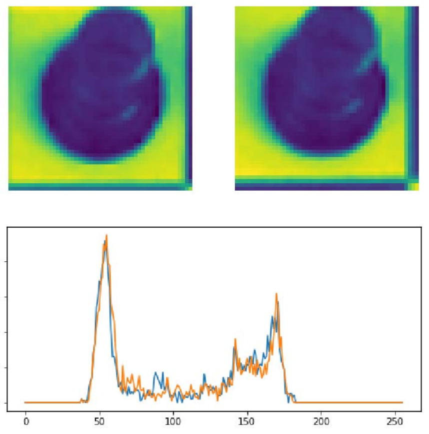
b) Edge Detection: Another approach is working with detection of edges at the center region of each square. Even with lighting variations, a correctly set Canny Edge Detector [15] will have no problem in differentiating between illumination variation and piece edges, as shown in Fig. 7.

This method also uses a single image to identify if the square is empty or not. If any significant amount of edges

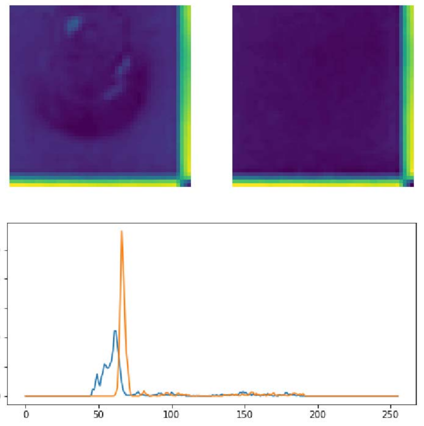
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 02,2023 at 15:19:57 UTC from IEEE Xplore. Restrictions apply.



(a) A meaningful change in color histogram is detected.



(b) A meaningful change in color histogram is not detected.



(c) A meaningful change in color histogram is detected.

Fig. 6: Color Histogram Comparison.

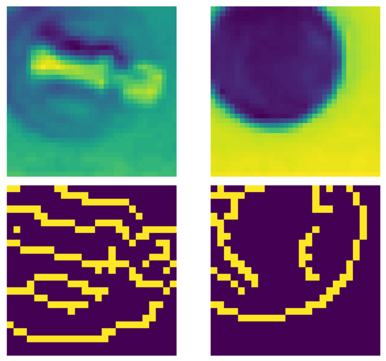
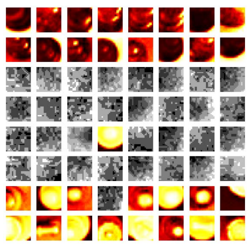
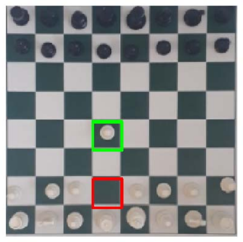


Fig. 7: OpenCV Canny Edge Detection.



(a) Piece Detection.



(b) Move detection.

Fig. 8: Movement Tracker.

are detected at the center of a square that means this square is not empty.

Using this method at the entire board we can get the result shown in Fig. 8. In Fig. 8a, red colored squares are detections of pieces, and gray colored are detections of empty space.

As shown in Fig. 8a, every square was correctly classified. In fact, for an entire game, every square in every position was correctly classified which means we can track the game from its start position in a reliable way.

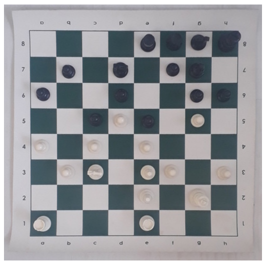
4) Game Tracker: By tracking the game from the start position and using the piece detector described in section III-A3b, a full Game Tracker is obtained. Fig. 9 shows the final output of tracking a real game of 50 movements from its beginning, no mistakes were made by the tracker, that means, in the total of 3200 (64 times 50) evaluations, no square was incorrectly evaluated.

As discussed in section II, the approach so far has great practical use, but it still does not identify which piece is present in each square, only movement starting from the beginning position.

Although, this method can be used to generate labeled data for the deep learning approach, since one can establish an initial chess position, and, by knowing which piece is moving at each time, it is possible to label the squares that changed.

Of course, this is not very practical, since, for each new labeled data of a single chess piece, it is needed an entire picture of the chess board.

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 02,2023 at 15:19:57 UTC from IEEE Xplore. Restrictions apply.


(a) Final Position.

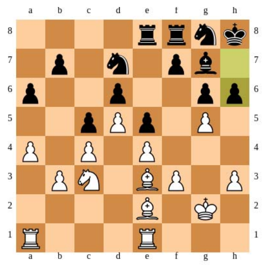

(b) Position Detection [16].

Fig. 9: Game Tracking a sample game.

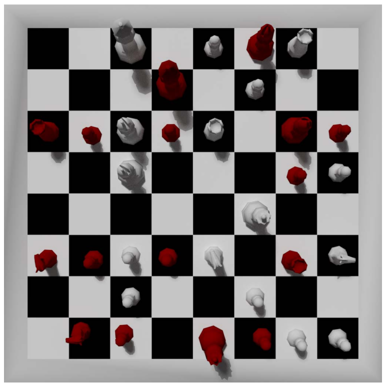


Fig. 10: 3D model design in Blender. **Source: [20]**

B. Synthetic Data Generation

So, a problem was faced: the data gathered by using the first approach was, although in good quality, in bad quantity. To proof the concept, another solution is needed to the data problem.

In such situations, it is common in the deep learning literature to generate your own data in an automated way, wherever developing real systems, as in [17], or serving as a proof of concept, as in [18].

So, for this task to be automated, Blender [19] was used. Blender is a open-source 3D modeling and rendering application that has a Python API.

A model for a chess game was created in Blender. This model, as shown in Fig. 10, can be modified in basically any way possible, given the limits of the software, using its API.

That means it is possible to generate any chess configuration, material composing the chess board and pieces, position and configuration of light, position of camera, random imperfection on pieces allocation and all other possibilities offered by Blender in an automated and parallelizable manner. All of this without losing the labeling information of data.

For this article, the configurations chosen to change were the position of pieces relative to the center of its square, that followed a gaussian in both x an y coordinates with zero mean and standard deviation of half the square side length. Also, the rotation of the pieces and the rotation of the light source followed uniform distributions from 0 to 2π.

Once stabilished all the conditions, a 3D model (Fig. 10) is created an then rendered (Fig. 11). Alongside the image, a file containing the information of the board configuration is also saved, for future segmentation of pieces.

In this work, this was done for a total of 571 chess board rendering images with 1024x1024 pixels of resolution (such as Fig. 11) taking roughly 5 minutes per image rendering, on a 8GB RAM, Intel Core i7-4500U CPU and Radeon HD 8850M GPU notebook. Something important to say is that these are the images that are going to feed, after pre-processing, the neural network. Images like Fig. 10 are just a step in the development of those final images.

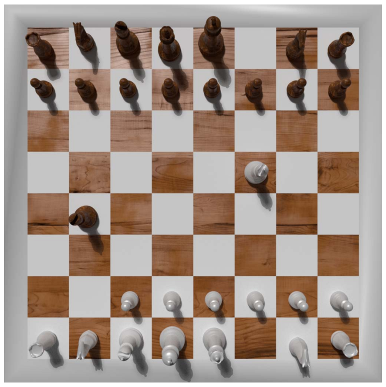


Fig. 11: Rendering of model in Blender.

Then, after rendering all images, it was possible to sectorize and label 50x50 gray scale images of pieces, shown in Fig. 12. This resolution is achieved by resizing the chess board image and it was chosen to be a fixed value, since the classification by the neural network requires a fixed size input. This set of 36544 images was the data set used in the deep learning model.

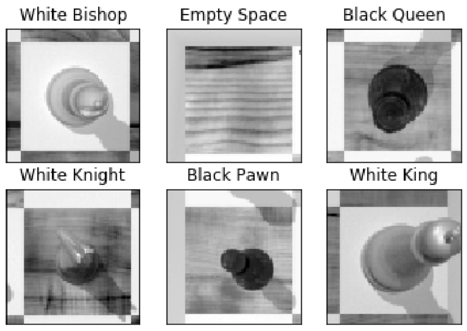


Fig. 12: Some of the sectored and labeled images.

IV. Neural Network

The chosen classification model is a neural network composed of two parts: a pre-trained VGG network [21] and a multilayer perceptron network in sequence.

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 02,2023 at 15:19:57 UTC from IEEE Xplore. Restrictions apply.

The VGG network is a deep convolutional network composed of several convolutional layers interlaced by some max-pooling layers with some fully connected layers at the end. There are actually various versions of this architecture scheme, based on the quantity of parametric layers. The one used in this work is VGG-16, that is composed of 13 convolutional layers, 2 fully connected layers and a softmax layer to output the classification of an image. Its topology is shown in Fig. 13.

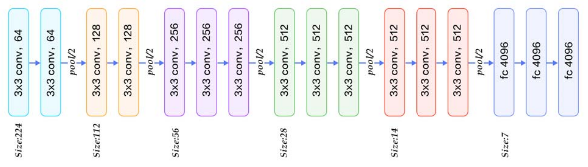


Fig. 13: Schematic of VGG-16 without its last layer.

This network was previously trained on the ImageNet [22] data set. So, the idea is to use this previous training in our favor.

Since a large convolutional network has a great capacity of discovering many details about the data, the VGG-16, in this work, is used as a feature extractor of the chess pieces images. That is, all images are pre-processed trough this network, and the final output (disregarding the last softmax layer of the network) is taken.

Then, this output, basically a lower dimensional representation of the input image, but with higher feature explicitation, is fed into a simple fully connected network, whose job is to output the piece classification.

This network is composed by 3 fully connected layers, interlaced by dropout layers with rate of 0.5. A simple schematic is shown at Fig. 14. All implementation details can be seen in the repository of this paper, available at `https://github.com/rafaelmcam/cChess`. The last layer is a softmax layer, responsible for outputting the probability of the input image being of each of the 13 possible classes.

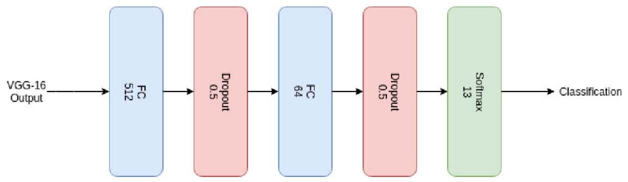
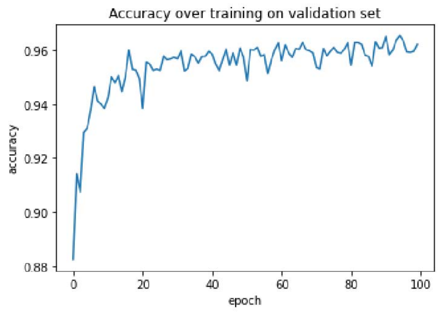


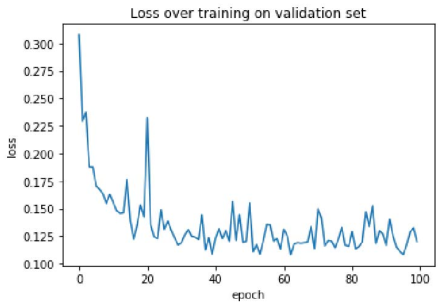
Fig. 14: Schematic of the top model.

So, actually only this last network is trained, which makes training faster and feasible on lower processing capacity computers. Training the entire network (including all VGG-16 parameters) could be computationally unfeasible to the authors.

That is the motivation towards adopting this scheme, being able to produce a high performance model by making use of all the processing used to train the VGG-16, without actually training it. Of course, this comes with a trade-off, that must be managed.



(a) Accuracy on validation.



(b) Loss on validation.

Fig. 15: Training of the network.

For training, the data set was split between a training, a validation and a test set. The division was 64%/16%/20%. Loss and accuracy on validation is shown in Fig. 15 Training was performed for 100 epochs on a batch size of 8, using Adam optimizer [23], categorical cross-entropy as loss function, implemented in Keras [24] and took approximately 21 minutes to finish on a Nvidia Tesla k80 GPU, available at *Google Colaboratory* [25].

V. Results and Comparison

In light of the difficulty regarding obtaining a big data set, the following results are consequence of the synthetic generated data acquired in Section III-B and training discussed in Section IV.

A. Confusion Matrix

First obtained result was a confusion matrix on the testing set of the Neural Network Model.

As seen in Fig. 16, the only label the model had trouble classifying was the black queen, which is usually mistaken as a black bishop. The pieces are indeed quite similar when seen from top view.

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 02,2023 at 15:19:57 UTC from IEEE Xplore. Restrictions apply.

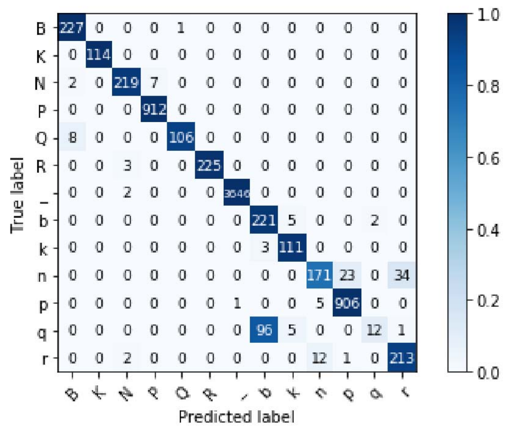


Fig. 16: Confusion matrix on test set.

B. Accuracy, Recall and F1 Score

Accuracy on this test set was 97.1%, and although this is already a higher number in comparison to many related works ([6], [7], [12]), the main advantage of the presented approach are the Recall/F1 score.

The unbalanced frequency of chess pieces, for example, pawns being roughly eight times more frequent than a king, can impact heavily on the meaning of high accuracy scores.

As already discussed in Section II, if a model that is able to identify between an empty space or an occupied space is set to evaluate every occupied space as a pawn, this classifier would already have close to 75% accuracy without been actually capable of recognizing any chess piece. That is why other scores can reflect more of the models capability of recognizing pieces.

Recall score measures the number of correct results divided by the number of results that should have been returned. Precision score is the number of correct results divided by the number of all returned results [26]. F1 score is the harmonic average between Recall score and Precision score.

Some related work only exhibit accuracy scores, that being the ratio of correctly predicted labels to the total number of labels, which means there is not enough information to calculate Recall and F1 scores, since, for that, is needed the actual prediction and real value of each sample.

Comparing to the result presented in Fig. 2 from [7], it is possible to calculate Recall and F1 Score. In this work it was achieved accuracy, Recall and F1 scores of 97.1%, 89.2% and 93.0%, while data presented in [7] achieved 78.4%, 47.0% and 58.8%, respectively.

The reason for a bigger difference between different scores in works such as [7] is the unbalanced accuracy scores obtained in different pieces classification. While achieving good accuracy on the most common labels, that being empty squares and pawns, these classifiers don't achieve good results for labeling less frequent pieces such as rooks, knights, bishops, queens and kings. The difference in Recall/F1 score when compared to accuracy score reflects this notion.

C. Synthetic Board Classification

A complete synthetic board classification in presented in Fig. 17 and exhibits 62/64 (roughly 96.87%) correct labels, corresponding to 96.8% when empty spaces are considered, and 30/32, corresponding to 93.7%, if only pieces are considered valid labels. The two misses, shown in red, are black pawns mistaken as a rook and knight.

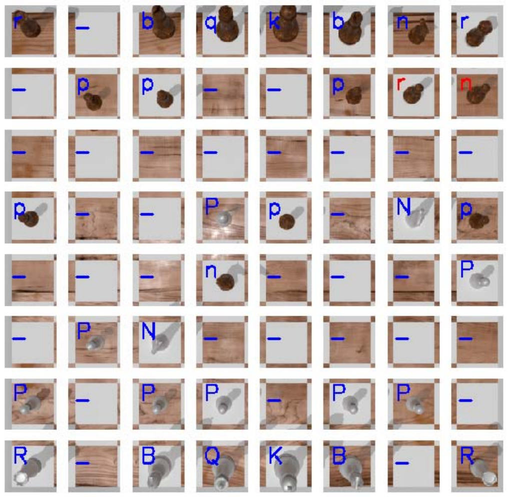


Fig. 17: Piece classification on Blender generated board.

VI. Applications

The work here done, as a chess board identifier, can provide some useful applications. In this section. are proposed two of them.

A. Automated record of real chess games

Since it is possible to get information about configuration of all pieces on a chess board, converting it in chess standard notation and files (such as `.pgn`) is easy. Therefore, it is possible to create large databases of chess games alongside their correspondent images. This application was actually implemented, tested and even used to produce figures, like Fig. 9b.

B. Real-time chess game between online players on a real chess board

If it is feasible to translate images of chess boards to chess standard files, is also possible to use such files to transmit the game information between two parts. Besides that, classification on a full board image (actually 64 classifications, one for each square, are made) takes about 0.5 seconds on the same computer configurations used in section III-B, which shows the capability of real-time playing.

That means it is possible to play chess with another person, in another place, still moving real pieces, on a real board, without having to make any modification to pieces or board except for the installation of a video camera or smartphone.

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 02,2023 at 15:19:57 UTC from IEEE Xplore. Restrictions apply.

VII. Conclusion

A. Summation

This paper exposes a method to gather data for a machine learning model from chess board images. Also, it is presented a deep learning model based on fine tuning of an already trained network, VGG-16, to achieve classification of chess pieces.

Two models to generate data were discussed, one that covers the gathering of images of real data sets, tracking the chess board and segmenting its content, and another that involves the generation of artificial data by rendering a 3D model of the chess board.

Each one of these approaches have up and downsides. The first one produces real and diverse images at the cost of being more effort and time consuming. The second one produces large amount of data at an almost inexpensive way, with the cost of producing images that do not follow reality in its entirety.

The main advantage of the presented approach of the deep learning model is the ability to attain high Recall and F1 scores even when using top view camera angle, which is a challenge, as result of images looking quite similar to each other, but offers the advantage of producing no occlusion whatsoever. Many discussed related works choose to discard right from principle top view camera angles for its difficulty for classification.

So, with this paper, it is shown that is possible to adopt this kind of approach and obtain acceptable results.

B. Future Improvements

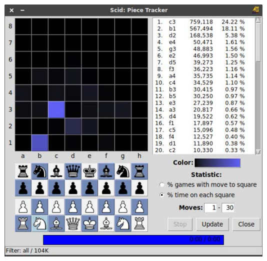
1) Piece Counter: A possible practical improvement for the classification task is the formulation of heuristic procedures taking account common chess games conditions. It is unusual in a playable game to achieve a position where there are three knights or three rooks of the same color on the board as the neural network suggests at Fig. 17, although possible.

An idea in these kind of scenarios is to assume that some knight and rook were mislabeled, in this case the probabilities calculated by the neural network should be taken into consideration by checking which knights were classified with the lowest top scores and choose as the correct label the ones with second highest scores.

2) Database Heatmaps: A promising idea is the development of a algorithm that flags already classified pieces as possible mistakes by checking if this piece is at an unusual position.

A classifier of unusual pieces positions can be done by using easily available collections of human played chess games, as the ones provided by *lichess* [27] which, as of June 2019, possesses more than 700 million games.

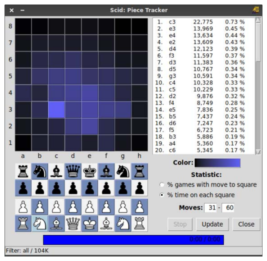
The open source chess application SCID [28] provides a *Piece Tracker* tool which generates a heatmap for each piece given a chess database. This heatmap, shown in Fig. 18, can be used to specify which classified pieces are in unnatural positions, and then perform detailed analysis, if possible with different neural network or classification method, to figure out the correct piece label.



```
 1.  c3   759,118   24.22 %
 2.  b1   567,494   18.11 %
 3.  d2   168,538    5.38 %
 4.  e4    50,471    1.61 %
 5.  g3    48,883    1.56 %
 6.  e2    46,993    1.50 %
 7.  d5    39,273    1.25 %
 8.  f3    36,223    1.16 %
 9.  a4    35,735    1.14 %
10.  c4    34,529    1.10 %
11.  b3    30,415    0.97 %
12.  b5    30,250    0.97 %
13.  e3    27,239    0.87 %
14.  a3    20,817    0.66 %
15.  d4    19,522    0.62 %
16.  f1    17,897    0.57 %
17.  c5    15,096    0.48 %
18.  f4    12,527    0.40 %
19.  d1    11,890    0.38 %
20.  c2    10,330    0.33 %
```

○ % games with move to square
● % time on each square
Moves: 1 - 30

(a) Heatmap for white queen's knight in first 30 moves.



```
 1.  c3    22,775    0.73 %
 2.  e3    13,969    0.45 %
 3.  e4    13,634    0.44 %
 4.  e2    13,609    0.43 %
 5.  d4    12,123    0.39 %
 6.  f3    11,597    0.37 %
 7.  d3    11,383    0.36 %
 8.  d5    10,767    0.34 %
 9.  g3    10,591    0.34 %
10.  c4    10,328    0.33 %
11.  c5    10,229    0.33 %
12.  d2     9,876    0.32 %
13.  f4     8,749    0.28 %
14.  e5     7,836    0.25 %
15.  b5     7,437    0.24 %
16.  d6     7,247    0.23 %
17.  f5     6,723    0.21 %
18.  b3     5,886    0.19 %
19.  a4     5,360    0.17 %
20.  c6     5,345    0.17 %
```

○ % games with move to square
● % time on each square
Moves: 31 - 60

(b) Heatmap for white queen's knight in moves 31-60.

Fig. 18: Heatmaps for white queen's knight in specific move ranges.

3) Network Design: For the development of the deep learning model, only a few basic architectures of neural networks were tried and also almost no hyper-parameter tuning was made, since the objective was only to proof the concept of the classifier.

Such ideas are, without doubt, a big improvement to be made.

4) Modifying Realism of Simulated Images with GAN's: Further improvements to the approach of synthetically generating data would be to make the images more realistic. This could be achieved by using a generative adversarial network (GAN) trained on unlabeled real chess pieces data to approximate the distribution of the synthetic generated images to the distribution of real ones without much loss of context, as it was done in [29].

References

[1] D. Cireşan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification."

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition."

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks."

[4] C. Koray and E. Sümer, "A computer vision system for chess game tracking," Başkent University.

[5] G. D. Illeperuma, "Using image processing techniques to automate chess game recording," University of Colombo.

[6] C. Danner and M. Kafafy, "Visual chess recognition," Stanford University.

[7] J. Ding, "Chessvision: Chess board and piece recognition," Stanford University.

[8] J. Meyer, "Raspberry turk - robot that can play chess." [Online]. Available: http://www.raspberryturk.com/

[9] C. Matuszek, "Gambit: An autonomous chess-playing robotic system," 2011 IEEE International Conference on Robotics and Automation.

[10] A. T.-Y. Chen and K. I.-K. Wang, "Robust computer vision chess analysis and interaction with a humanoid robot," 2016 2nd International Conference on Control, Automation and Robotics (ICCAR).

[11] V. Wang and R. Green, "Chess move tracking using overhead rgb webcam," 2013 28th International Conference on Image and Vision Computing New Zealand (IVCNZ 2013).

[12] R. Varun and S. Gupta, "Chess recognition using computer vision," The Australian National University. [Online]. Available: https://github.com/SukritGupta17/Chess-Board-Recognition

[13] "Documentation scikit-learn: sklearn.metrics.f1_score." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 02,2023 at 15:19:57 UTC from IEEE Xplore. Restrictions apply.

[14] Y. Xie, G. Tang, and W. Hoff, "Chess piece recognition using oriented chamfer matching with a comparison to cnn," in *IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018.

[15] "Documentation opencv (open source computer vision) python." [Online]. Available: https://docs.opencv.org/4.1.0/

[16] N. Fiekas, "Python chess library documentation." [Online]. Available: https://python-chess.readthedocs.io/en/latest/core.html

[17] A. V. Ankush Gupta and A. Zisserman, "Synthetic data for text localisation in natural images."

[18] S. M. Plis, D. R. Hjelm, R. Salakhutdinov, E. A. Allen, H. J. Bockholt, J. D. Long, H. J. Johnson, J. S. Paulsen, J. A. Turner, and V. D. Calhoun, "Deep learning for neuroimaging: a validation study."

[19] "Blender." [Online]. Available: www.blender.org

[20] *Komandar*, "Blender base model." [Online]. Available: https://free3d.com/pt/3d-model/chess-table-18114.html

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," University of Oxford.

[22] "Imagenet." [Online]. Available: http://www.image-net.org/

[23] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optmization."

[24] "Keras." [Online]. Available: http://keras.io

[25] "Google colaboratory - free jupyter notebook environment." [Online]. Available: https://colab.research.google.com/

[26] "Wikipedia - precision and recall." [Online]. Available: https://en.wikipedia.org/wiki/Precision_and_recall

[27] T. Duplessis, "lichess - internet chess server database." [Online]. Available: https://database.lichess.org/

[28] S. Hudson and P. Georges, "An open application to view, edit, and manage collections of chess games." [Online]. Available: http://scid.sourceforge.net/

[29] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training."

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on November 02,2023 at 15:19:57 UTC from IEEE Xplore. Restrictions apply.