Jan Daniel Sulaiman

Chess Board image recognition

Automatically recognition chess boards images and translating them into a digital representation.

Sulaiman, Jan
5.4.2019

1. Definition

1.1. Project Overview

Computer vision techniques have advanced a lot in the previous years, the application of image recognition for various areas has increased. Also, to use it in competitive games. Chess, a game in which players constantly study to improve based on previously seen and played position, is one application in which processing of real-time image data could help to provide a new way for players to improve.

Recognition of chess piece via image and game tracking have been explored in the past, but a good portion relies on a known start position. In this project a chess position image recognition application is developed, by using a convolution neural network classifier, to output a digital representation of the input chess board. This this application a dataset of 1893 chess piece images is used. The chess board is using the Staunton Tournament Chess pieces with WE Games Tournament Roll Up Chess Board. These are the most common used sets in Chess clubs.

1.2. Problem Statement

Main objective of the project is the translation of an input image of an chess board to a digital representation. This project will take an input chess board image, provide the required preprocessing for the image, divide the image into individual squares and finally using a CNN to provide the best estimate for the piece in each given square. The output is a digital representation of the board via command line output. This is the key work to be done to use the same setup to track live games on boards and extracting key moments to fed them into the engine for analyze of the chess piece position.

1.3. Metrics

For evaluation of the models performance the following to metrics are used:

1. Log Loss = -log P(Ytrue| Ypred) = -(True log(Ypred) + (1 - Ytrue) log(1 - Ypred))
2. Accuracy = True Positive + True Negative / Totals

Log Loss also referred to as categorical cross entropy, is essential for the given problem. It is a multi-class problem and log loss defined the loss function to heavily penalize high prediction probability for wrong classed but rewards high prediction probabilities for the correct class. The assumption is that it will help the algorithm to attempt to figure out the correct piece as opposed to the most common piece in the input dataset.
Accuracy is less critical but valuable as it's a easy to understand metric which can be shown during training to track the progress of the training session.

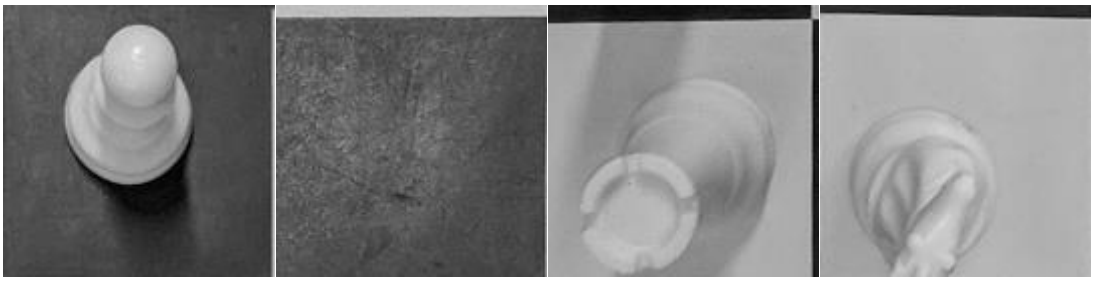
2. Definition

2.1. Data Exploration

The dataset is available on Kaggle at: https://www.kaggle.com/dcuomo56/chess-board-data

Each chess board image is converted to black and white as it is not necessary to use color for this problem. Total dataset consists of 1893 chess piece images each being 135x135 pixels. The following image is an example of an image before the split:

These next images are examples for individual piece images:



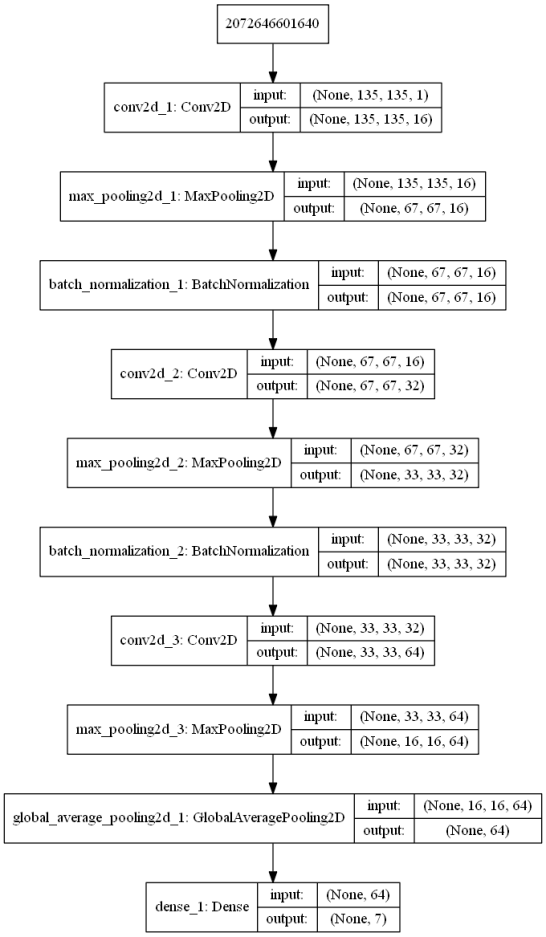
The data set is grouped in two different categories to support the neural network architecture. The groups are piece type and color of the piece. In both groups an empty one is considered square no matter which color it is. Recognizing the color of the square is not important for the problem as the color of the squares on the chessboard are fixed.
There is a high imbalance in classes as soon as it comes to empty squares due to the nature of the data collection. The used images for empty field are reduced to 300 images in total as otherwise it would outnumber the rest of the dataset. Identifying an empty square is not challenging for a neural network so the reduction of data is okay.
Data set is split into equally sized groups for training, testing and validation. After the split mislabeled data was removed from the set. There was still a minor class imbalance when comparing the king and queen to all the other piece types. Still they made about nine percent of the total data set and each other piece type about 16% of the dataset.

After further reviewing the data set two other potential risks for the training are visible. As visible in the following images:

The one problem might be that the pieces are positioned close to the edge of the square and some parts of the piece are cut off. The other problem is that neighboring pieces are on the edge of the square and therefore parts of another piece are in the image so that two pieces are shown on one. Since this should also be a general problem when recording chess games, it's a risk considered as low and rather be used to optimize the model for these cases as well.

2.2. Algorithms & Techniques

As classification algorithms two different convolutional neural networks are used. One is responsible for the identification of the piece type and the other for the color of the piece. For classification it uses the output of both models to determine the actual piece. This decision is taken since it helped to maximize the training data to identify key features. This meant one CNN is focusing on deciphering the difference between of a king and a queen regardless of color at the same time the other one is focusing on the color.
A simple CNN architecture using Keras as show in the following image was the base:



This architecture consists of three sets of Convolution, Pooling and Batch Normalization. After that s single Global Average Polling Layer is used to connect to the output layer with outputs either corresponding to seven for the pieces and three for the color CNN. During the development of the model various aspects of the neural network are tuned:

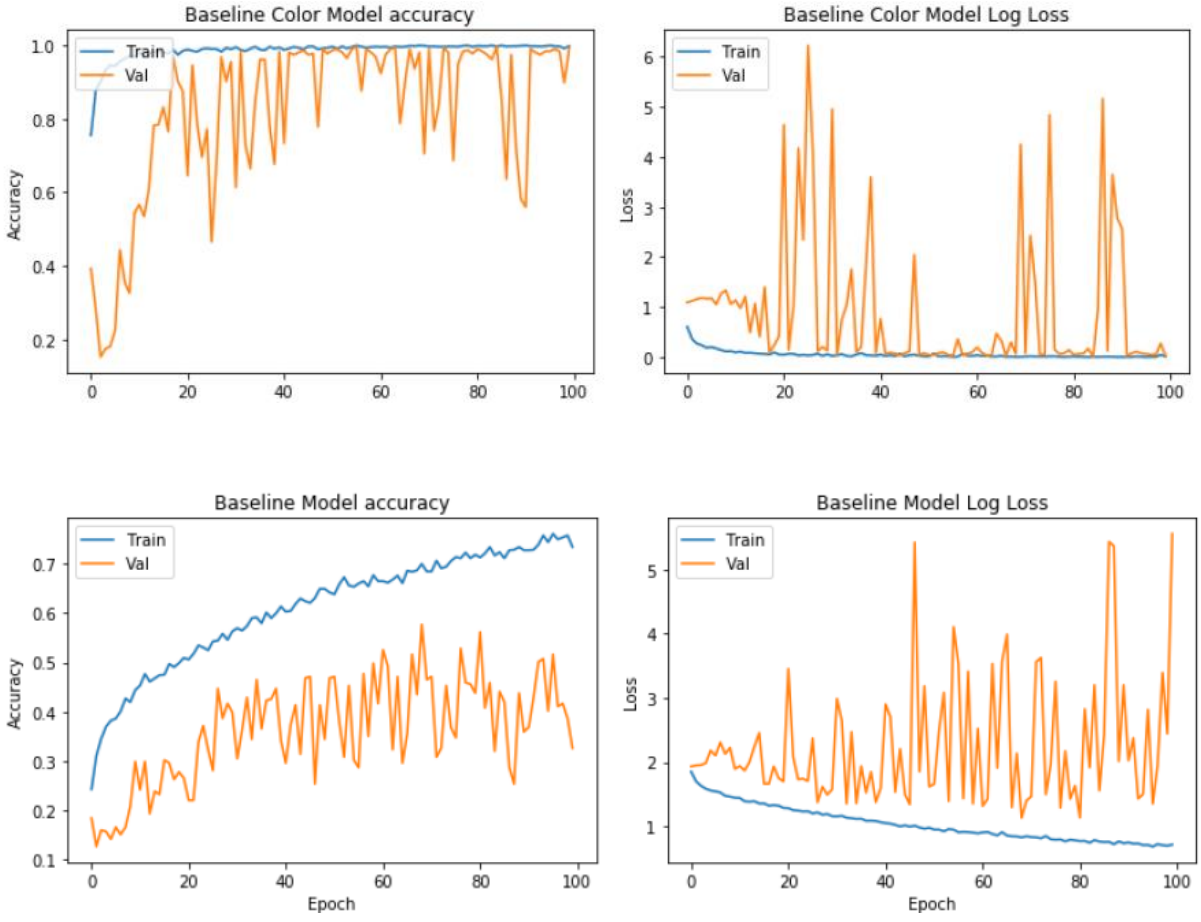
- The total Number of Layers: Increased layer adds neurons for learning
- Layer Types and Parameters used:
    - 2D Convolution Layers with specific focus on filter the number of filters and filter size to extract different key features
    - Batch Normalization for helping to increase the performance and reduce overfitting
    - Dropout Layers which minimize overfitting by focusing on a specific area of the neural network

Additionally, the following Key hyperparameters were adjusted and tuned during the training process to minimized overfitting and find an ideal solution:

- Model learning Rate
- Total Number of Epochs
- Batch Size

2.3. Benchmark Model

As benchmark model the CNN architecture previously show is used and trained against the dataset for 100 epochs, with the RMSProp optimized active. The Batch size is 32 as well as the learning rate 0,01 and no image augmentation. Here is the result for the baseline model for each color and piece classification:



	Accuracy	Log Loss
Color Model	100 %	0,005
Piece Model	51,3 %	1,23

In both cases the learning rate is too high which is an evidence of overfitting and not a stable setup for the validation set metrics. The color model is finding an optimal solution, suggesting a minor change in the learning rate would help. On the other hand, the piece model did not perform good. Both log loss and accuracy indicate a bad result. After checking the confusion matrix, the only class that had a perfect result is the empty square. Therefor all other steps taken in the further development and changes in the architecture are only made to the color model.

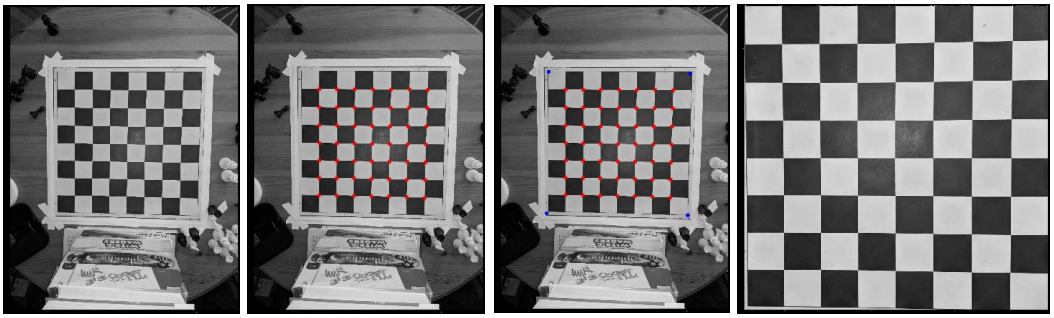
3. Methodology

3.1. Data Preprocessing

Preprocessing the image is done by the ChessBoardProcessor object. The preprocessing is done in two steps which both have similar activities. First step is using an empty chess board to find the perspective transformation for all other chess board images. Second step is to apply the perspective transformation and splice the chess board images into individual pieces.

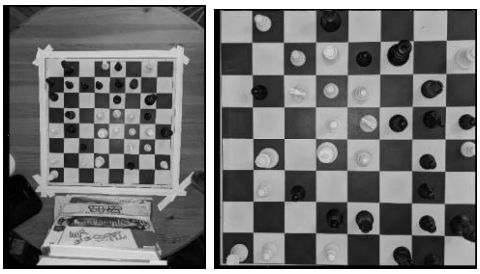
The perspective transformation is using the OpenCV library and consists of the folloowing steps:

1. Importing grayscale images of the chess board
2. Using OpenVC function to get the chess board inner corners
3. Perform a linear regression on the innter corner diagobals to fint the outer corners
4. Calculate a geometric transformation to take the outer corners and translate them to a 1080x1080 pixel image. The array for transformation is stored to a class variable M.



Once the perspective transformation is done for the baseline image, it is possible to apply it to all other images. Once transformed they are slices with the following steps:

1. Imporitng chess board image
2. Apply the perspective transformation by using cv2.warpPerspective
3. Slice the image into 135x135 squares and store the square images in a dictionary.



3.2. CNN Model Implementation

CNNModel object contains the details for the models, training, testing and prediction methods. The CNNModel is build on top of the Keras library which is leveraging TensorFlow as a backend for the deep learning algorithms. CNNModel training and testing is made up of the following steps:

1. Initiate the CNN model architecture
2. Load training, testing and validation data into Keras ImageDataGenerator, specific scaling and augmentation parameters for training data. This was used to allow augmentation to be easily added if needed and batch fitting.
3. Specify the model hyperparameters such as optimizer, learning rate, epochs, batch size
4. Specify the model loss functions and metrics (Categorical Loss and Accuracy)
5. Training the network and store the metrics to memory while saving the best weights during the process
6. After the training the results are satisfactory, the process is complete, if not training is repeated with tweaked parameters and augmentation parameters

3.3. Chess Board Predictor

The prediction for the chess board pieces is done in two different classes. The predition function is defined in the CNNModel, however the ChessBoardGenerator is executing the prediction using a passed in CNNModel object. This means that the function is wrapped in the main program and the prediction runs through these steps:

1. ChessBoardProcessor is created as object with the baseline board image path on initialization
2. Test board is loaded and split using the ChessBoard processor
3. CNNModel object is created
4. CNNModel object is compiled
5. CNNModel object selected training or pre-trianing weights depending on the program flow
6. ChessBoardGenerator objected created with a split into test board and then the CNNModel passes it on during initialization
7. ChessBoardGenerator is predicting the pieces function call, which will predict color and piece type for each square using CNNModel
8. ChessBoardGenerator output display runs to display the prediction output

3.4. Refinement

During the training there are various important aspects identified and tweaked. All of the following ones have an big impact on the models performance:

- Learning Rate: As read on various notes I researched also optimizers use the learning rate. It had an significant impact on finding a usable solution. Compared to the baseline model using the learning rate overfitting could be prevented by reducing the rate.
- Modifying and adding additional CNN Layers: Further convolutional layers are added to the architecture to pick out further relevant futures. Adjusting the filter size also helped in earlier layers. Increasing the convolutional filter size helped to identify more significant features which sort the images into the different categories.
- Mode Dense Layers: In the baseline model the output was taken from the global average pooling layer and connected it directly to the output neurons. Changing the amount of additional dense layers and size provided some increased performance due to the increased number of nodes.
- Image Augmentation: Augmentation of the training data helped the most and gave a boost to the performance of the model. With an accuracy bigger then 70% and log loss of less than one was not achieved without augmentation. It is possible that some of the complications which are in the final model could be removed after augmentation is added to it.

Augmentation is essential to help overcoming the small amount of data and consider the many orientation a piece can have in single chess board square. It is only used in the piece model.

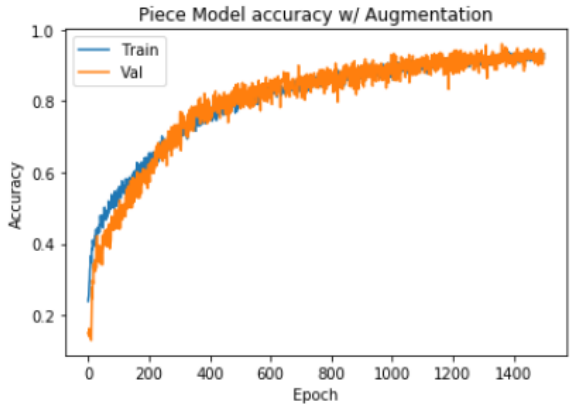
4. Model Evaluation

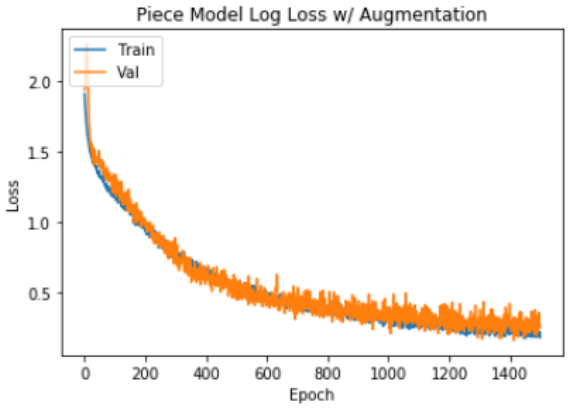
The final model for the piece type is made up of the following:

- Total of three grouping of 2D Convolution, Max Pooling 2D and Batch Normalization layers of increase filters and decreasing kernel size from five to two with relu activation function at the beginning
- Back to back 2D Convolution layers of 128 filters of of a kernel size two with relu activation layers, followed by max pooling and Batch Normalization
- Another two groupings of the same early pattern with the same number of filters and kernel size
- Global Average Pooling layer to minimize the number of parameters at the output of the convolution section of the neural network to reduce overfitting
- Two dense layers
- Dropout layer to decrease overfitting
- Final dense layer with softmax activation to give a probability an image is in a certain class

The final model weight are created in a training run with these hyperparameters:

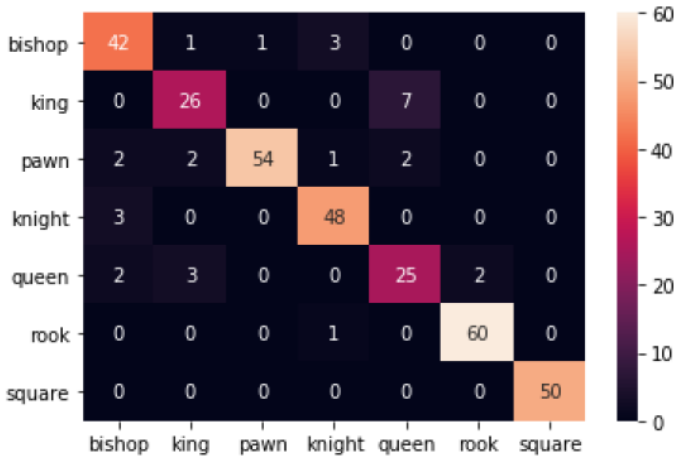
- Optimizer: RMSProp
- Learning Rate: 0,00001
- Epochs: 1500
- Batch Size: 32





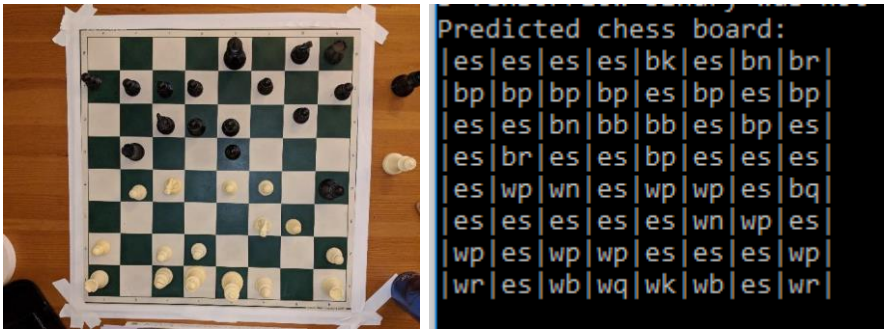
	Accuracy	Log Loss
Piece Model	91 %	0,27

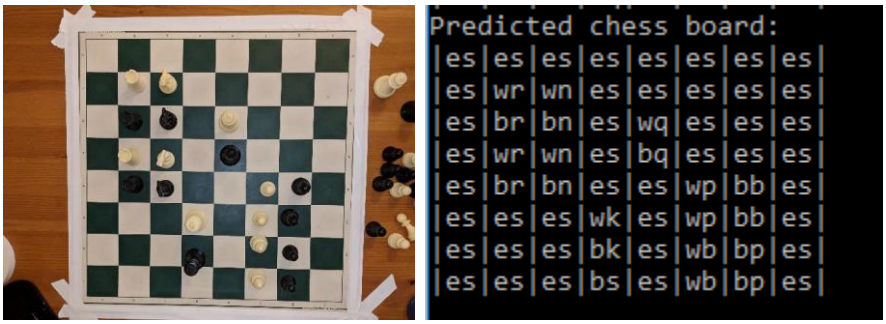
The shown results from the training are outperforming the benchmark model by a lot. The model as able to raise the piece recognition accuracy to 91% and reduce the log loss to 0,27, which is a very high increase compared to the benchmark. After taking a look at the confusion matrix, it appeared the only major issue that remained is the kind and queen identification. This is already found as potential issue in the beginning of the project. All other piece types have an accuracy over 88%.



5. Conclusion

5.1. Free Form Visualization





5.2. Reflection

The following main steps are required to solve the described problem:

1. Select the right data of chess pieces
2. Preprocessing of the images by using OpenCV
3. Developing and training a model as benchmark for the actual classifier build with Keras
4. Refining the model and training processes
5. Developing the final application which has the model included

The data gathering was not the most complicated part, as it was already available and could be reused. More challenging was to understand the basics of computer vision and start to use already existing tools like OpenCV. It took a lot of trial and error to fully understand how the corners algorithms and perspective transformation would for on the given chessboard images.

Building the actual model and defining the architecture was also a big challenge and took a lot of research to find some first common ground. This work was also the most interesting, due to the complexity but quickly seeable results for a real-world problem. Especially since this approach could also be applied to other problems then just recognizing chessboards.

5.3. Improvement

The end algorithm had an accuracy of more then 90% in digitizing the positions of each figure on the chess board. The generated output of the result is not yet usable and therefore if I would continue working on the project take a look at the following enhancements:

1. Find further images of chess boards to help gather more input data for the training phase
2. Optimize the training size to be able to handle more data, especially look into using GPU powered machines for the training to increase the calculation throughout
3. Find a better visual way for the output e.g. supporting standard formats already used by chess engines.