2D Chess Piece Classification using CNN

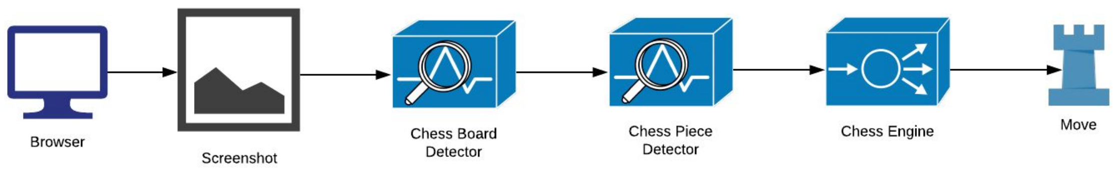
Rajtilak Indrajit
December 3, 2018

Proposal

Domain Background

Chess is a two-player strategy game based on an ancient Indian board game called Chaturanga. In the past twenty years, advancements in computing has propelled the efficacy of chess engines at game analysis.

The focus of this capstone is to take the first step to enable real-time engine analysis of online chess games through a Chess Bot. The central component involves developing a robust classifier for chess pieces. Future work will involve using this classifier to convert online chess games to the standard FEN notation, which in turn can be used for engine analysis and move suggestion. Enabling this at high level involves identifying the chess board, classifying the chess pieces, interpreting the chess position, and feeding it to an engine for analysis. This capstone will focus on chess piece classification.



There is existing work in this domain. TensorFlow Chessbot by Sameer Asari uses CNNs to perform this task on 2D chess boards and achieves fairly robust performance. Chess Piece Recognition by Anoshan Indreswaran also takes a similar approach. My goal is to build on top of these projects.

Chess is a game that I am particularly fond of and recently been following closely. The world chess championship that was held in November 2018, is still fresh in my memory. I also play Chess online for fun on lichess.org.

Problem Statement

The core problem to be solved is the classification of chess pieces dataset into the seven corresponding classes - King, Queen, Rook, Bishop, Knight, Pawn, and Blank.

Since chess pieces come in all shapes and sizes, a traditional computer vision approach to classification will fall short. Hence, the hard problem is to develop a generic and robust model that can adapt to variations in the a) shape, b) size, c) color, and d) design of the chess pieces.

The primary success metric is classification accuracy. However, it is important that the accuracy is measured on real-world datasets, using non-uniform chess pieces.

Datasets and Inputs

Data Acquisition

Internet searches revealed that there is not an existing chess image dataset that can be used for this classification task. Google Image search did not prove to be very useful either. There are some additional criteria for our dataset, we require a dataset that is non-uniform and representative of the variability of chess pieces in the real world. Hence, I decided to build a dataset of chess piece images from real-world images of chess pieces.

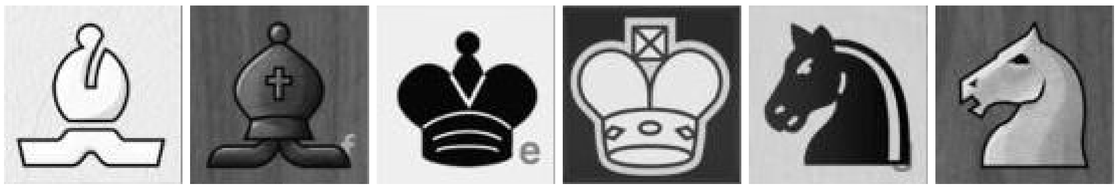
Data Characteristics

The data was created manually by screenshotting and cropping various chess piece and board designs from lichess.org, chess.com, chess24.com. The dataset consists of 1k images based of the 2D chess boards on online chess websites.

A total of 35-40 designs of chess boards was used. The dataset has images that vary in background, texture, shape, color, and design. This variability is a close approximation to online chess boards that the model is meant to cope with - online and on-screen 2D chess boards on popular chess sites.

The dataset will be used for both training, validation and testing of the model. Variability in zoom, shear, and rotation is introduced during training and testing using keras. The dataset is organized into 6 subdirectories based of the different classes of the chess pieces. K for king, Q for queen, N for knign, B for bishop, R for rook, P for pawn, and B for blank. Blank is treated as a separate class since it overall fits nicely into the multi-class classification paradigm. It is equally possible to exclude the blank class and use a rule-based system to classify it beforehand.

The dataset is well balanced, with equal number of images for most classes. However, the queen and king classes have fewer images - this is simply a function of the fact that there is only one set of king & queen per board, whereas two sets of all other pieces. The number of pawns has purposefully been limited so as to not introduce imbalance in the dataset. Here are a few of the images from the dataset.



Solution Statement

The solution uses Deep Learning and Convolutional Neural Networks to classify the chess piece images. The model trained on a dataset with ~200 images per class. The model is designed to be robust and adapt to the variability in chess board and piece design online. The primary metric is classification accuracy.

Benchmark Model

Since the solution hinges on the eigenvector approach, the most classic benchmark model would be a Support Vector Machine using an RBF kernel. The model is trained to perform multi-class classification on the image data across the seven classes. For data pre-processing and dimensionality reduction, I am following the standard image classification technique in supervised learning - Principal Component Analysis. We are likening this classification task to that of face detection. In the same vein as eigenfaces, we are generating eigen-pieces that represent the different chess pieces using Principal Component Analysis. Also, we will use GridSearchCV for tuning the hyperparameters of our Support Vector Machine. The benchmark model closely follows the example shown here.

The benchmark model using this dataset achieves an F1 score of 0.82

Evaluation Metrics

The domain of our problem requires a model that performs well on all dimensions. We need to be able to correctly classify all 64 tiles of a chess board. Getting even one of the tiles wrong would result in a wrong board position, and therefore fail to be useful. This implies that we have no room to trade off on precision and recall. Accuracy is a good enough metric to use here, since the dataset is fairly balanced overall. However, a stricter metric that would be resilient to any imbalance in dataset sizes would be the F1 score.

Hence, the metric of choice is the multi-class F1 score. For choice of averaging, I considered micro, macro, and weighted. When it comes to predicting a chess board, the model is surely more likely to see blanks and pawns, and much less likely to see the queen and the king. However, since our dataset is fairly balanced it came down to whether the model should have a bias towards any of the classes. Another factor is the problem context, a model that has a bias towards the more frequent classes is no better when it comes to predicting chess pieces, since it is an all-or-nothing result. Hence, we chose to use 'micro' averaging for our F1 score.

The metric discussed above is at a piece level. Eventually, we would like to introduce a new metric: Board Accuracy. Board accuracy would be a binary metric, true when we predict the board accurately, and False otherwise, regardless of whether we mislabelled one or ten squares.

Board Accuracy = # true board positions / total board positions

Our goal is to achieve an F1 score of 0.9 or greater, and eventually a board accuracy of 0.7+.

Project Design

Within the realm of chess piece classification, there are a number of approaches that I considered and choices to make. Here is a brief discussion of some of those choices.

Predicting Individual Pieces vs Entire Positions

The first problem to solve was how to set up the ML model. The two clear choices were to either predict at a piece-level or a board-level. A model that were to predict entire board positions seemed impossible at first glance, since the number of possible positions reaches 10^{120} as noted by the Shannon Number. Even a more conservative estimate puts this at around 10^{40}. Hence I chose not to pursue this any further.

2D vs 3D chess pieces

Another point of consideration was whether or not to support 3D chess pieces, and if yes, whether to include them as part of the same class or to separate the 2D king (2DK) from the 3D king (3DK). The challenge here turned out to be segmenting the chess board into squares such that the 3D chess pieces were contained in each square. After much trial and error, I decided to abandon 3D piece support and focus on 2D. If I were to attempt 3D again, based on the trial and error, I would separate the 3D classes from the 2D classes for some pieces (King, Queen), but perhaps keep it the same for others (Knight, Rook, Bishop).

Detecting Piece Color

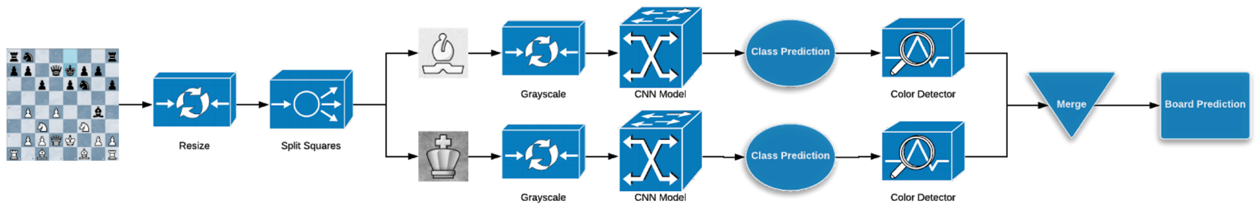
Should the model detect just the piece, or also attempt to detect the color of the piece? The general feeling here was that the color is very subjective, and it is difficult to predict in isolation. It would be a much simpler problem to predict color based on the global board level, rather than the local information at a square level. Hence, this problem was pushed further up-stream. My idea is to solve this problem using a naive pixel averaging approach to being with, and build on top over time.

Google Search vs 2D Boards

Since I had to generate the dataset myself, I had the choice of Image search or manually screenshotting and splicing images from popular online chess websites. The former approach proved to be too difficult, as there was simply too much noise in image search. Hence I decided to take the manual route, as already mentioned above in the dataset description.

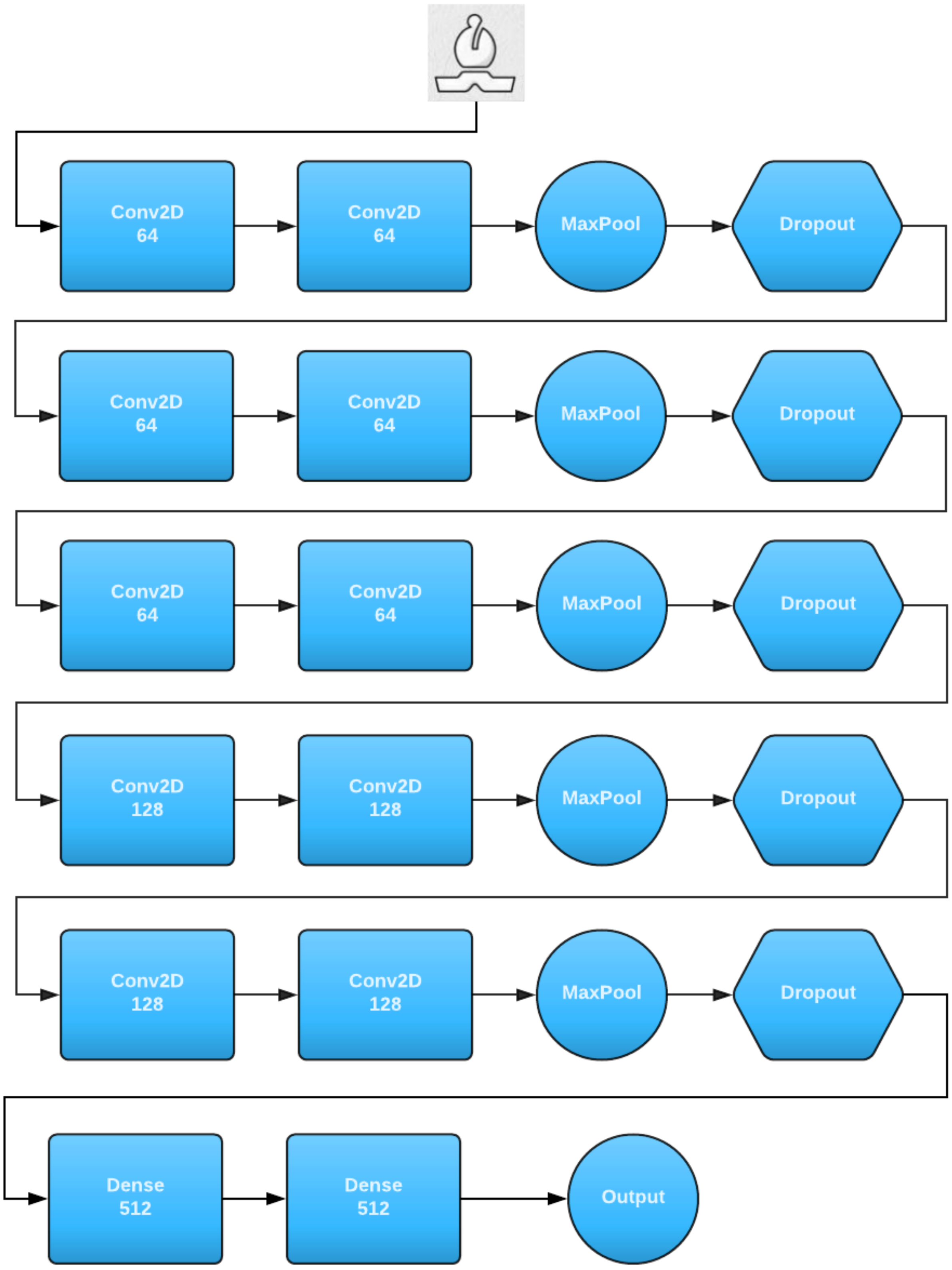
Pipeline

Ultimately, the design I landed on is the most simplistic, but also the most effective. The process steps include creating the dataset by sourcing and manually classifying the training and validation set into the corresponding image classes. The images will then be converted to grayscale and scaled to desensitize our detector to color and size variations. These images are then used to train a CNN that uses 5 convolution layers and nearly 100k trainable parameters. The resulting model will then be run on our test set to measure classification accuracy.



Model Architecture

The chosen model architecture is based closely on the VGG16 model. The model summary is as follows



Next Steps

Future work would include building other parts of the overall Chess Bot. This includes a Browser extension that supports screen capture, a chess board detector, an image splicer to convert the board to individual images, the neural network classifier (which we developed above), a piece color detector, an FEN generator that combines the detected pieces, and finally, a chess engine that will predict the next move.

Future Work

Future work can focus on a number of areas of improvement. For instance, we could start supporting 3d chess pieces instead of limiting to 2d only. Likewise, we could also support more skew and transformation of images. Similarly, the model could be extended not only to predict the piece but also the color.