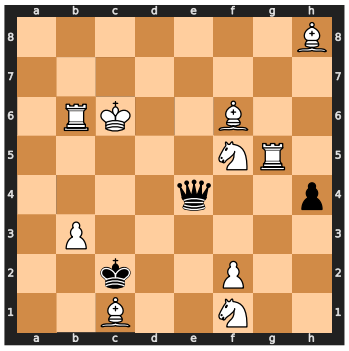
AI Chess Master Final Project

Technical Notes

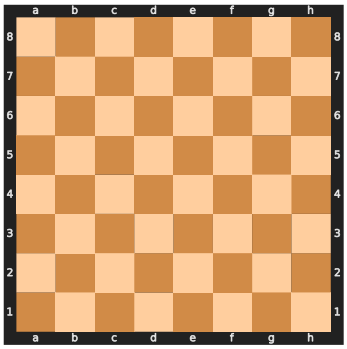
Author: Ibrahim Sobh
Group: AIS 2021

1. Introduction:

This project tackles the problem of detecting chessboards positions captured in 2D Chess Boards images. A chessboard board consists of 64 squares (eight rows and eight columns) and the chess pieces that are placed on a board.



7B-8-1RK2B2-5NR1-4q2p-1P6-2k2P2-2B2N2



8-8-8-8-8-8-8-8

The problem of recognizing a chessboard from an image can be difficult, with a major issue mostly being the quality of images (lighting issues and low resolutions issues).

For this reason, most current methods for chessboard recognition typically perform certain simplifications. This includes utilizing only a single chessboard style during experiments, capturing images with adirect overhead view of the chessboard (in Case of 2D Chess Board Recognition) or at a convenient angle in advanced cases (in Case of 3D Chess Board Recognition).

The state of the art to solve this problem is to use a CNN's that feeds on the chess Board blocks (for each image we chop it into 64 blocks representing 64 position/ possible piece). Afterwards we encode the labels into classes (multiclass classification) to get the result and then we decode the results to reproduce the images of predicted FEN's.

Chessboards Forsyth–Edwards Notation (FEN for short) is a standard notation for describing a particular board position of a chess game. The purpose of FEN is to provide all the necessary information to restart a game from a particular position, FEN is used to define initial positions oth. Usually (Capital Letters for **White**) (small Letters for **Black**)

Example in the picture above: 7B-8-1RK2B2-5NR1-4q2p-1P6-2k2P2-2B2N27

7B -> means & empty spaces then Bishop (**White**)
8 -> means empty row
1RK2B2 -> means one empty spaces , a Rook(**White**), a King **White**),2 spaces … etc.

2. Methodology:

Dataset From Kaggle: https://www.kaggle.com/datasets/koryakinp/chess-positions?datasetId=115231

Dataset Content

- 100000 images of a randomly generated chess positions of 5-15 pieces (2 kings and 3-13 pawns/pieces)
- Images were generated using 28 styles of chess boards and 32 styles of chess pieces totaling 896 board/piece style combinations (Images were generated using this custom-build tool)
- All images are [400 by 400 pixels]. Training set: 80000 images // Test set: 20000 images
  Pieces were generated with the following probability distribution:
  30% for Pawn, 20% for Bishop, 20% for Knight, 20% for Rook,10% for Queen

Approach:

After performing a full on EDA I have concluded that the data is pretty much clean and have the same Ratios, Weights and Sizes and that the thing that I need to focus on is the image resolution, number of pixels per inch and the way to divide the blocks evenly into 64 pieces while preserving the shape and the smaller details of the pieces themselves
I tried a PCA but the results were not encouraging as only about 5-10% of features loss were enough to make the pieces unrecognizable check notebook for examples

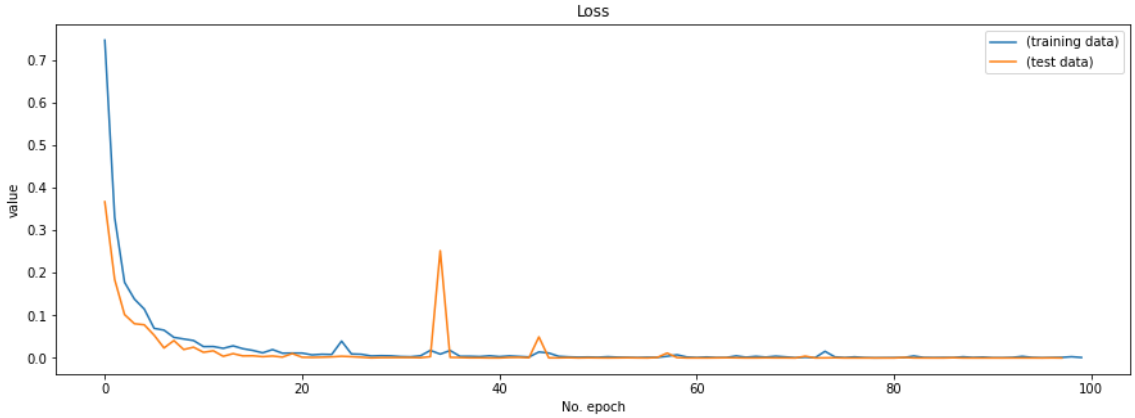
Afterwards, the decision I had to think about is how to decrypt the label because the labels are a compound of several pieces' positions at the end, I went to Kaggle and I had taken a function that do the Encoding/Decoding of the labels because my way of one hot encoding was not efficient.
My Choice of a CNN was first because I am dealing with a pretty big Dataset and pretty diverse classes and that simple models were not going to perform as good as a CNN in tackling such a problem as they will need a lot of fine tuning and they won't be reliable in all cases.

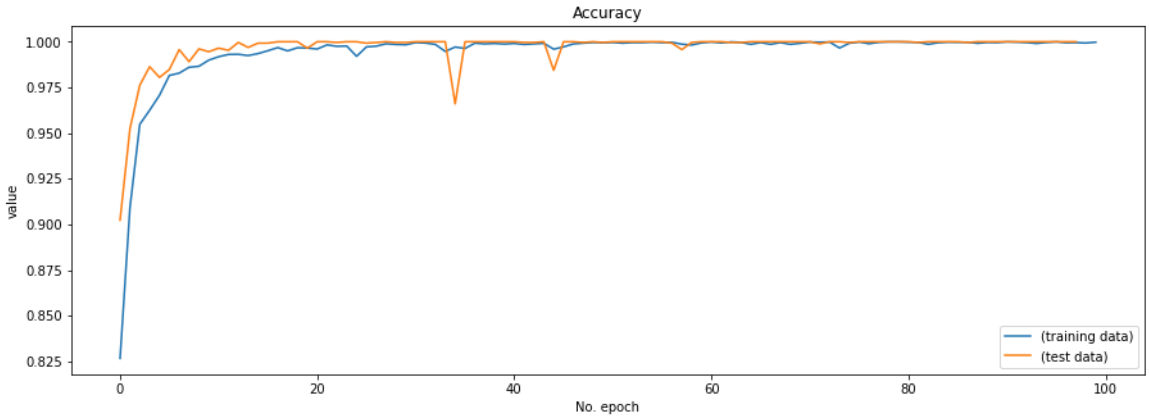
I have used 2 convolution layer with 2 different kernel a (3,3) one to focus on the pieces and then a (5,5) in order to look deeper into the details between these 2 I used a Maxpooling and I ended with a flattening layer and a SoftMax dense layer.
Regularization was needed while training so I used also a dropout of (0.35)
I compiled the model and trained the model and then saved the history as well as model weights in order to use later.

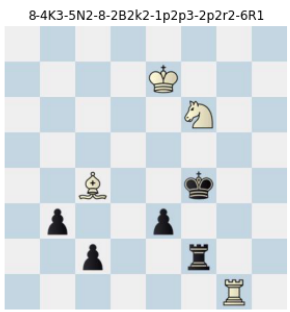
I trained with 20 K train, and 4 K test images which was more than enough to get ~99% accuracy, I shuffled those data to have the best possible distribution.

I experimented with the changing the resolution, size of the images (the smaller the picture the less details I have and the more accuracy I lost) and increasing the complexity of the model made the model overfit quickly, so I decided to leave the CNN structure simple and play with the Kernels only.

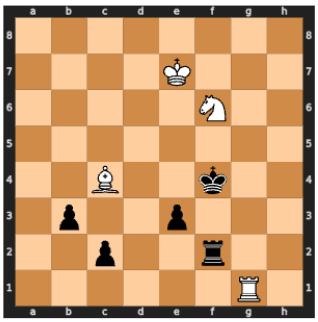
3.Results:



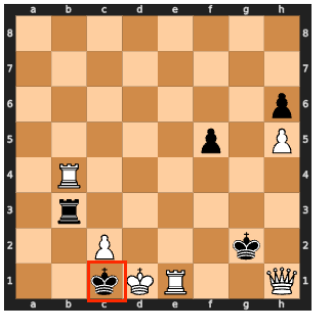




predicted FEN : 8-4K3-5N2-8-2B2k2-1p2p3-2p2r2-6R1



Actual FEN: 8-8-7p-5p1P-1R6-1r6-2P3k1-2qKR2Q
predicted FEN : 8-8-7p-5p1P-1R6-1r6-2P3k1-2kKR2Q



Precison: 0.999
Recall: 0.999
F1 Score: 0.999
Accuracy: 0.999

In general, the results look really good in the following slide I'll explain more about it

The model overfitting was avoided using dropouts (which explains why the accuracy on validation seems to outperform training sometimes in the epochs).
After the 50th epoch we can see that the model loss seems to stabilize, and the model performance is amazingly good. (Same thing for accuracy.)
Looking at the scores F1, Accuracy, precision and Recall I was a little confused until I toke a look at the confusion matrix and number of misclassified (between 5-20 out of 4000 depending on the randomness of the shuffle of the train) which explains a lot why they are equal. To improve the randomness of outliers I should have used K-Fold Validation instead of normal Validation to have a better look at the outliers even if their count is small.

4. Conclusion and perspectives

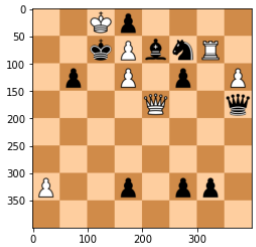
What is even more interesting plotting those outliers made me realize that the model sometimes gets confused because of the shades on one of the pieces inside a block , for example it doesn't consider it as a piece if the block is dark and the piece has the same shade of dark (I had a lot of outliers where the mistake was that the model didn't even recognize the piece in place so it is an empty block while it should be filled)
And some other time it gets confused because the details of 2 pieces are too similar like or too distinct (in the train we have chessboard with different themes for the same FEN each time the same pieces have slightly different shapes but sometimes very different) which explains the confusion the model Had.

I also created my own image generator using Cairo SVG and Chess SVG to test my own random samples!! ☺

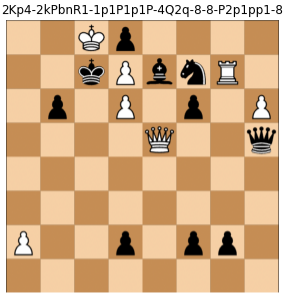
```
from cairosvg import svg2png

def generate_random_image_from_FEN(FEN,optional_path='../random/'):
    board = chess.Board(FEN)
    boardsvg = chess.svg.board(board, size=400,coordinates=False)
    FEN = FEN.replace('/', '-')
    return svg2png(bytestring=boardsvg,write_to=optional_path+FEN+'.jpeg')

generate_random_image_from_FEN('2Kp4/2kPbnR1/1p1P1p1P/4Q2q/8/8/P2p1pp1/8')
generate_random_image_from_FEN('8/8/8/8/8/8/8/8')
```

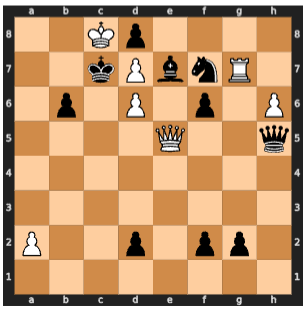
```
img  = mpimg.imread('../random/2Kp4-2kPbnR1-1p1P1p1P-4Q2q-8-8-P2p1pp1-8.jpeg')
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x2b7ac0130>
```



2Kp4-2kPbnR1-1p1P1p1P-4Q2q-8-8-P2p1pp1-8





Lastly to improve the model I think it is best if I train the model on learning **deeper** details of the pieces in such way it can also learn the smallest details of those pieces so it doesn't confused between them, and the second thing I propose is to try to maybe filter the picture or apply some kind of transformation to make the confusing shades look more distinct (playing with contrast maybe).