Chris Sorenson
CSCI-89, Intro Deep Learning
08/06/2021

Chess Piece Classifier and Object Detection

Abstract:

This paper outlines a three part process. The first part was the creation of ~2000 images of chess pieces. The second part was using these images to train a convolutional neural network that could classify the images as any of the twelve standard chess pieces or as a non-piece. The third and final part was creating a rudimentary image detection program using the network to locate and identify chess pieces on a sparsely populated board.

The project was moderately successful. The classifier achieved ~82% accuracy on test data. The object detection was limited by the effectiveness of the classifier – and tends to overpredict existence of pieces and miscategorize other pieces.

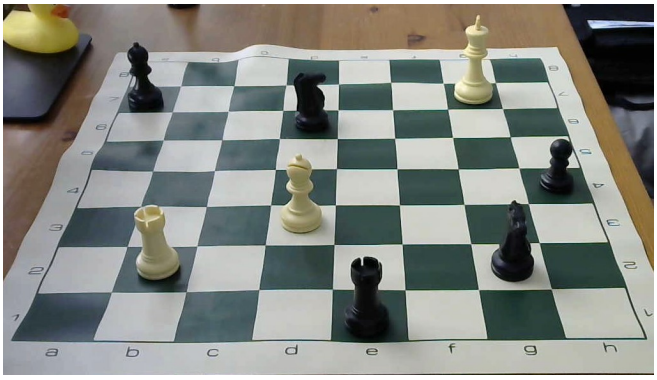


Figure 2: Input Image: Sparse Board

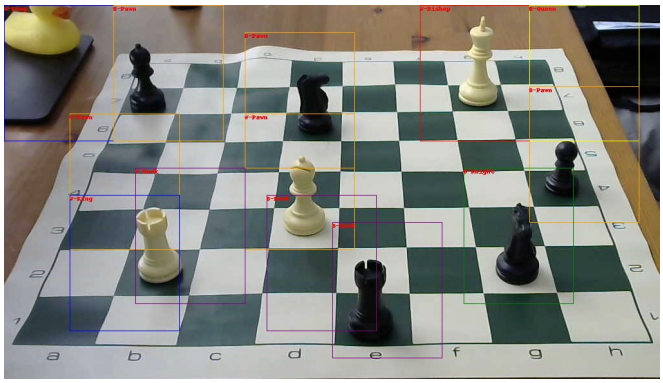


Figure 1: Output Image with Bounding Boxes Identifying Pieces

Configuration and Setup Details:

All code necessary to recreate the project is laid out in the Jupyter Notebooks for the Classifier and Object Detection.

No unusual packages are required. No known version requirements exist. External libraries include:

- tensorflow
- keras
- matplotlib (non-necessary – just for viewing data)
- sklearn
- pandas

- numpy
- PIL

The input data (images of chess pieces and boards) used for this project were all custom created and are not publicly available. In the truly unthinkable circumstance that somebody wants the data – drop me an email. It's <10MB worth of images.

The Data:

The data was custom created for this project – as no satisfyingly labeled and sufficiently large datasets were found.

Pictures of the board and pieces were taken from a variety of angles meant to imitate the approximate eye position of a chess player.

For expediency in generating the data – several pictures were taken with a piece on a given square. The piece was adjusted (rotated and shifted) between photos to increase the diversity of the dataset. The images were programatically cropped and labelled

About 2000 input images were created – evenly spread across the thirteen classes.

Data augmentation was used to artificially increase the scope of the data set.

The following augmentations were randomly performed using Keras.ImageDataGenerator:

- rotation (range 20)
- shear (range of .2)
- horizontal flip

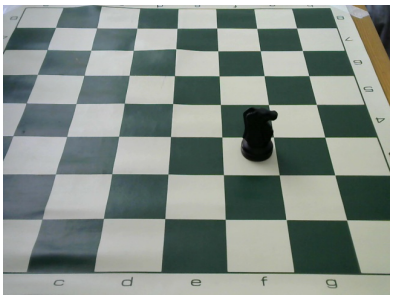


Figure 3: Original Image

Other types of augmentation, such as vertical flip, were deemed not helpful.

Figure 4: Cropped Image

The Data Mistakes:

Insufficient Data was collected.

An original batch of ~1200 images resulted in difficulties training an effective classifier. Adding ~800 to the dataset greatly improved the classifiers – but 2000 images is still a very small dataset and the classifier had a lot of room to improve further.

Insufficient training on 'No-Object' Images. There needed to be more images of partially out-of-frame pieces, as well of images not of the chess board to improve the classifiers ability to detect non pieces.

The Classifier Architecture:

The architecture of the classifier is summarized in Fig5.

The input is a 100x90 image.

The general structure is not particularly unusual. A series of convolutions and pooling layers are followed by a flattening layer, and several Dense layers funnel to a 13-dimensional output.

ReLu is used as the activation at all layers – except the output layer where Softmax is used

All convolutions have a filter size of 3x3, except the first which uses a 5x5 filter.

The Dense layers are aggressively regularized. Each is followed by a .4 dropout layer, and each also has l2 regularization with a factor or 10^-5.

All other features are either inferrable from Figure 5, and keras defaults are used everywhere else.

The classifier is compiled using Categorical CrossEntropy for loss and adam as the optimizer.

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 96, 86, 128)	9728
conv2d_8 (Conv2D)	(None, 94, 84, 140)	161420
conv2d_9 (Conv2D)	(None, 92, 82, 150)	189150
conv2d_10 (Conv2D)	(None, 90, 80, 160)	216160
conv2d_11 (Conv2D)	(None, 88, 78, 170)	244970
conv2d_12 (Conv2D)	(None, 86, 76, 180)	275580
batch_normalization_2 (Batch	(None, 86, 76, 180)	720
max_pooling2d_2 (MaxPooling2	(None, 28, 25, 180)	0
conv2d_13 (Conv2D)	(None, 26, 23, 256)	414976
max_pooling2d_3 (MaxPooling2	(None, 8, 7, 256)	0
flatten_1 (Flatten)	(None, 14336)	0
batch_normalization_3 (Batch	(None, 14336)	57344
dense_3 (Dense)	(None, 200)	2867400
dropout_2 (Dropout)	(None, 200)	0
dense_4 (Dense)	(None, 100)	20100
dropout_3 (Dropout)	(None, 100)	0
dense_5 (Dense)	(None, 13)	1313

Total params: 4,458,861
Trainable params: 4,429,829
Non-trainable params: 29,032

Figure 5: Classifier Architecture

Justifying the Classifier Architecture:

The current architecture was the product of extensive experimentation and achieved ~82% test accuracy.

Prior versions of the model peaked at ~60%.

Five convolutional layers are believed to be the key advantage of the current architecture. For identifying chess pieces, the ‘crown’ of the piece is a disproportionately detailed and critical section of the photo. As such it is necessary to extract all possible detail from this small-but-vital region of the photo before pooling the image. As a proof of this concept – consider the images below. It is nearly

impossible to determine which is the King and which is the Queen without access to the crown. For the curious – the King is on the left and the Queen on the right

Efficacy of the Classifier:

Overall the classifier has an 82% accuracy on test data – but it's worth examining where it went wrong.

The most illustrative summary of the classifier is the confusion matrix of its results on testing data(Fig6).

A few optimistic observations:

- The classifier never miscategorizes one color piece as the other color
- The classifier does very well on the None-Object Data

A few pessimistic observations:

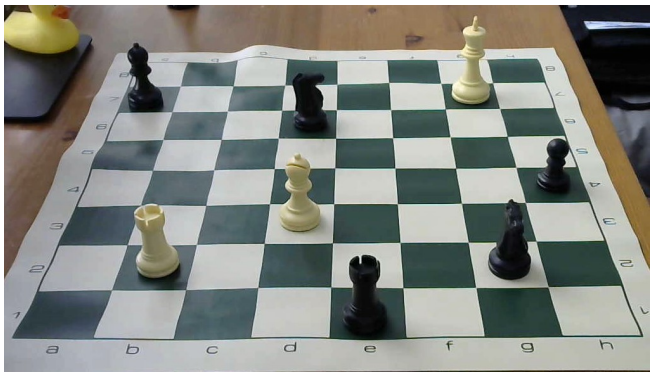
- Kings and Queens are still often miscategorized as eachother
- Pawns and Bishops are often miscategorized as eachother

Though far from perfect, this classifier can still be used in object detection – as seen in the next section.

	B-Bishop	B-King	B-Knight	B-Pawn	B-Queen	B-Rook	None	W-Bishop	W-King	W-Knight	W-Pawn	W-Queen	W-Rook
B-Bishop	36	0	0	4	1	3	1	0	0	0	0	0	0
B-King	2	30	0	0	12	0	1	0	0	0	0	0	0
B-Knight	0	0	37	1	0	7	0	0	0	0	0	0	0
B-Pawn	2	0	4	35	0	1	3	0	0	0	0	0	0
B-Queen	11	3	0	0	27	4	0	0	0	0	0	0	0
B-Rook	0	0	0	0	0	45	0	0	0	0	0	0	0
None	0	1	1	0	0	0	43	0	0	0	0	0	0
W-Bishop	0	0	0	0	0	0	1	39	0	0	1	3	1
W-King	0	0	0	0	0	0	0	1	22	0	0	21	1
W-Knight	0	0	0	0	0	0	0	14	0	25	2	0	4
W-Pawn	0	0	0	0	0	0	3	0	0	0	42	0	0
W-Queen	0	0	0	0	0	0	0	3	1	0	0	41	0
W-Rook	0	0	0	0	0	0	0	0	0	0	0	0	45

Figure 6: Classifier Results on Testing Data

The Object Detection:



The goal was to identify and locate pieces on a board like the one here. Note that the board is sparsely populated – so as to make my life a little easier.

Two variations of object detection were attempted. The first failed – and is only discussed briefly here.

The first method of object detection was to manually predefine the bounding boxes for each of the 64 squares of the chess board. Though labor-intensive – this method could be useful in contexts where the square positions are fixed, for example at a filmed chess tournament where a static board is filmed by a static camera.

However this method worked very poorly – exaggerating the limitations of the classifier. In particular – it grossly over predicted the number of pieces on the board, largely for the following reason.

The bounding boxes needed to be tall enough to accommodate the tallest piece – which by necessity of the camera angle, meant the bounding boxes included multiple squares. A short piece, particularly a pawn, could therefore be included in multiple bounding boxes

The second technique at object detection was much more successful.

Adapting the object detection strategy laid out in the Geron textbook (pp 486) – the classifier was ‘slid’ across the image at fixed intervals, left to right and top to bottom typewriter style.

The result was a set of predictions for each location – which was then culled to only include predictions with a 75% confidence or greater. The result is shown below.

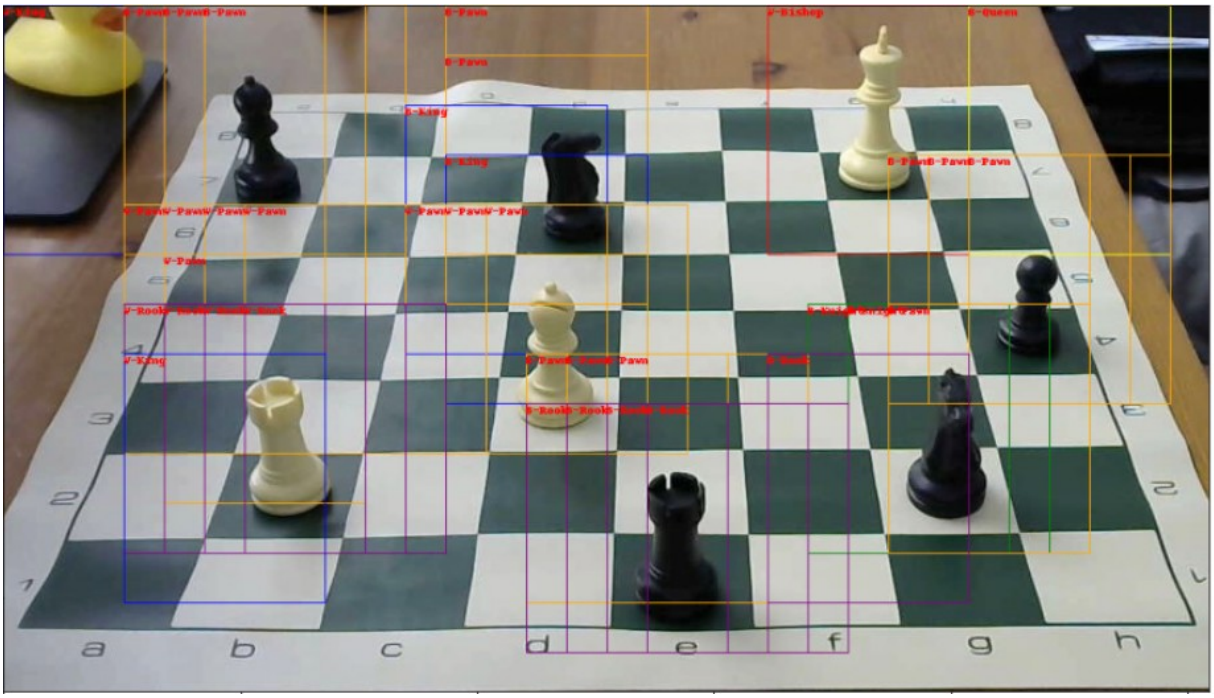
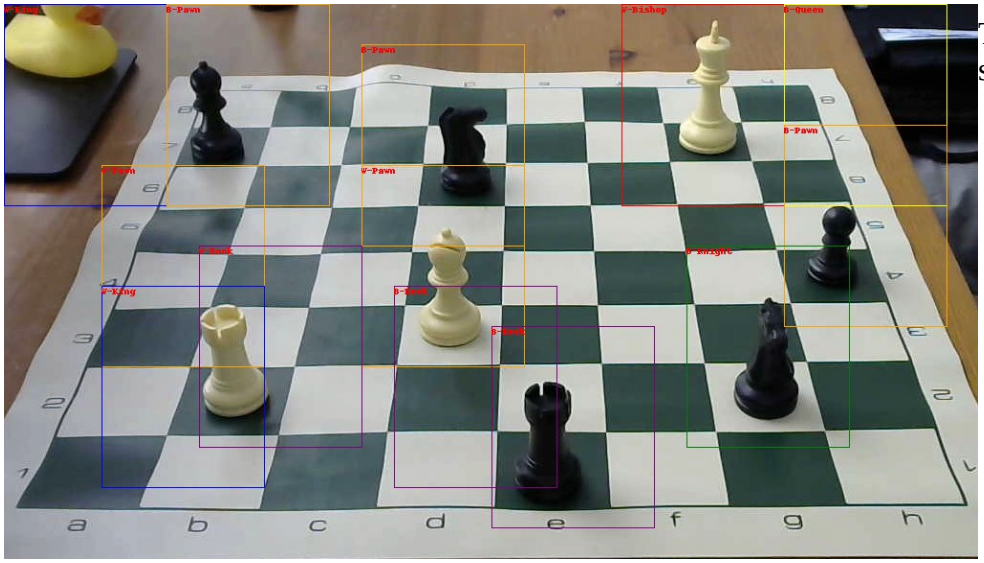


Figure 7: All Bounding Boxes with >75% confidence

To solve the issue of overlapping bounding boxes – we used the following algorithm.

Starting with the bounding box with the highest confidence – all neighboring bounding boxes are deleted. Continue with the next-highest confidence bounding box until no overlaps remain.



The result is shown here:

Figure 8: Final Bounding Box Predictions

This object detection scheme was moderately successful. Pieces are almost always identified as pieces – though are often miscategorized as the incorrect piece. Non-objects, such as the rubber duck in the upper left of Fig 7 are also regularly categorized as pieces.

Final Analysis:

The flaws in data were magnified in the classifier, and magnified further in the object detection scheme.

Increasing the size and variety of the dataset – particularly for the None-Object category – would likely greatly improve the classifier, in turn improving the object detection scheme.