Final Project

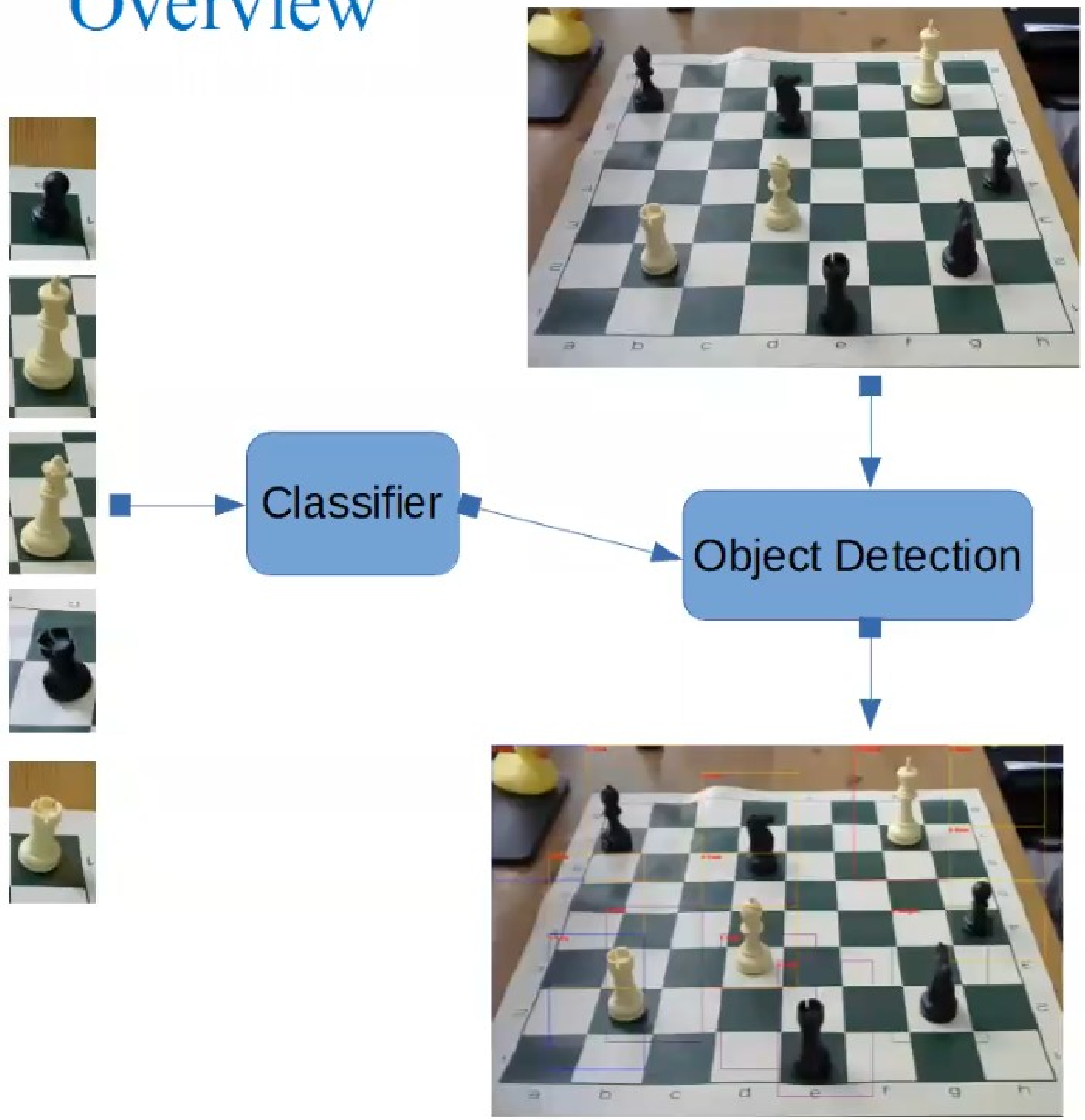
Chess Classifier

Chris Sorenson

CSCI S-89 Introduction to Deep Learning
Summer 2021
Harvard Summer School

Overview

- Goal is to detect chess pieces on a sparsely populated chess board
- A classifier was trained on images of individual chess pieces
- The classifier is then used to scan and detect pieces on an image of a chess board



Methodology

Three Steps in the Methodology:

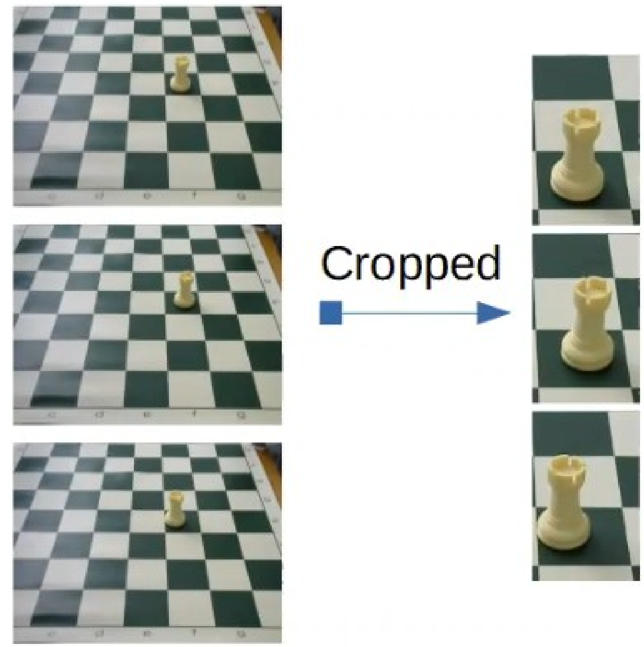
I. Generating the Data
   i. Images were taken manually
   ii. Dataset built from scratch
   iii. 13 Classes: 6 Piece Types x 2 Colors, and 1 No-Object Class

II. Building the Classifier
   i. Trained Convolutional network on the Data

III. Performing Object Detection
   i. Scan images of sparcely populated chess boards

Generating Data

- Images were taken of each piece, on a variety of squares
- Multiple images taken at square, rotating and adjusting the piece between photos
- ~2000 photos total
- Image Augmentation was used to increase the effective size of the training set
  - Shear
  - Rotation
  - Horizontal_Flip
  - Brightness



3 Pictures of White Rook at F4

Sample Results:

Lessons Learned Generating Data

- Was able to write a lot of useful scripts to vastly speed up the process of taking photos and automatically cropping to the piece, labeling, and storing
- Nevertheless much more data was needed!
  - My first 'generation' of data had ~1200 images
  - My second 'generation' of data added ~800 images
  - Classifier training went much smoother with the extra images but still hit a pretty low ceiling of effectiveness
- Identifying Absence of a piece
  - A separate "Non-Object" Category was used as a 13th category
  - Needed more training images to cover this category
    - Partially in-frame pieces
    - Sections of image that do not include chess board

Building the Classifier

Tried *many* different network architectures!

Most capped at ~60% testing accuracy

Until we had a breakthrough revelation:

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

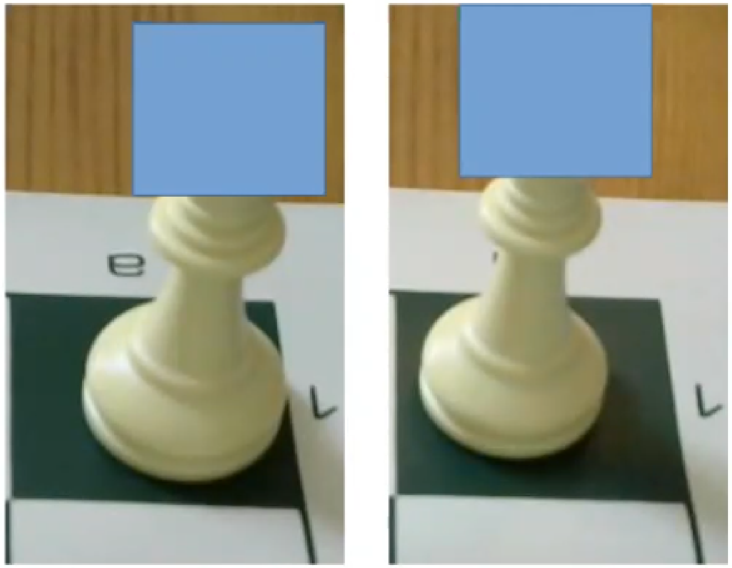
Total params: 4,458,861
Trainable params: 4,429,829
Non-trainable params: 29,032

Architecture of the Final Classifier

The Convolution Network Revelation

To the right are pictures of a White King and a White Queen with the crowns covered

Can you tell which is which?



The Convolution Network Revelation

If you guessed that the one on the left is the King, congratulations!

Though the king has a larger diameter, size is a bad predictor as it depends on distance to camera

The crown encodes the information about the piece

Therefore want to extract the complicated-but-small crown features before we lose critical information by pooling

The Convolution Network Revelation

Our final architecture uses this idea.

- 5 consecutive, small, convolutions
- Aggressively pool the image size down

The new strategy yielded immediate results – jumping testing accuracy up to ~80%.

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

Lessons Learned Generating Data

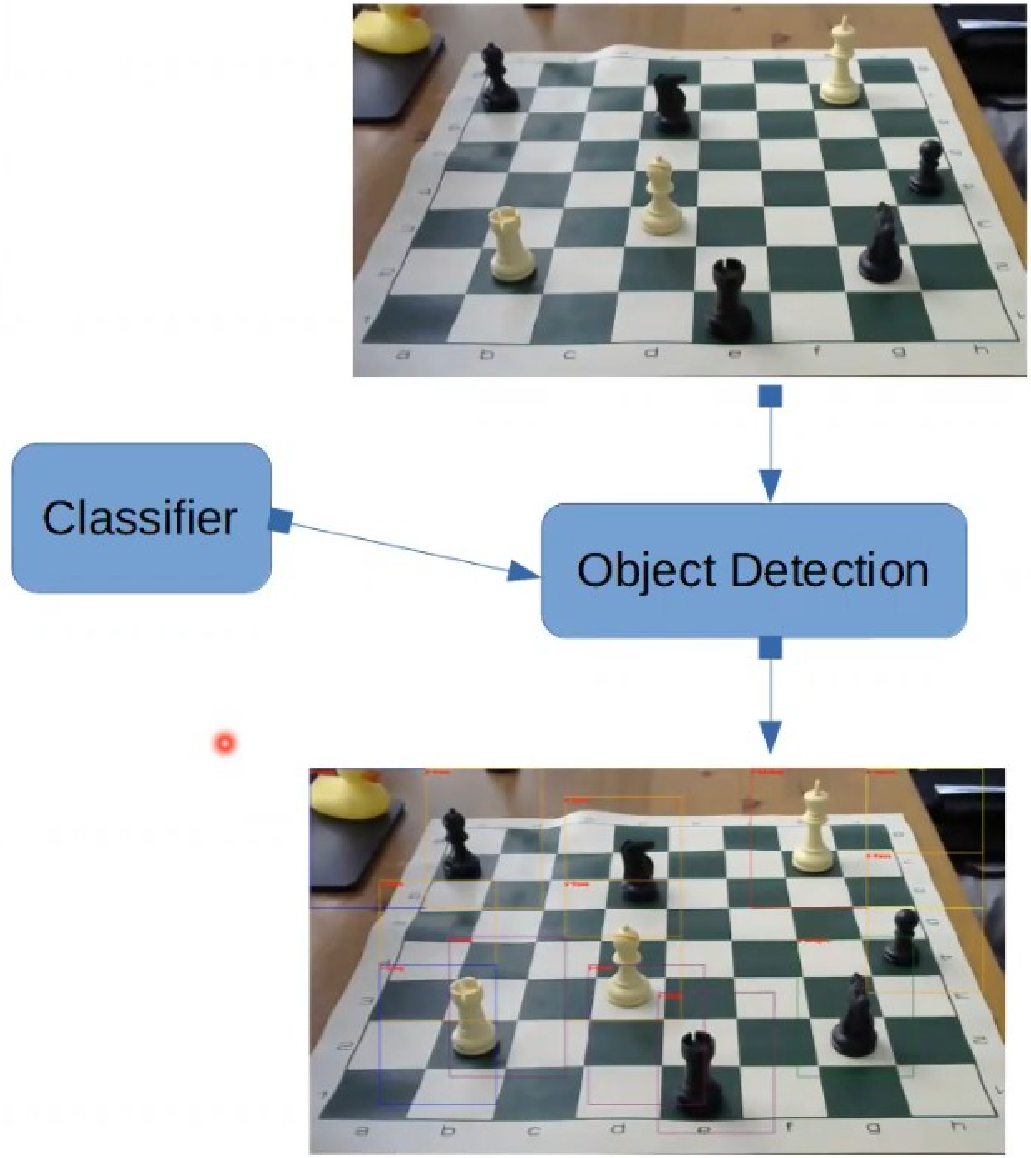
Not having enough training data remained an issue

Using the specific nature of the problem to inform the architecture had huge benefits

Object Detection

Goal: Use the classifier to scan images of chessboards and identify/locate the pieces

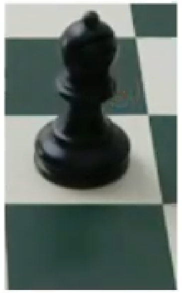
We used 2 strategies to attempt this



Object Detection: First Strategy

Method:
- Manually divide the board into 64 image-boxes, one per square
- Run the classifier on each box

Results:
- Worked so poorly I won't discuss in detail
- Bounding boxes needed to be tall enough to accommodate the tallest piece
  - As a result – shorter pieces can appear in multiple boxes

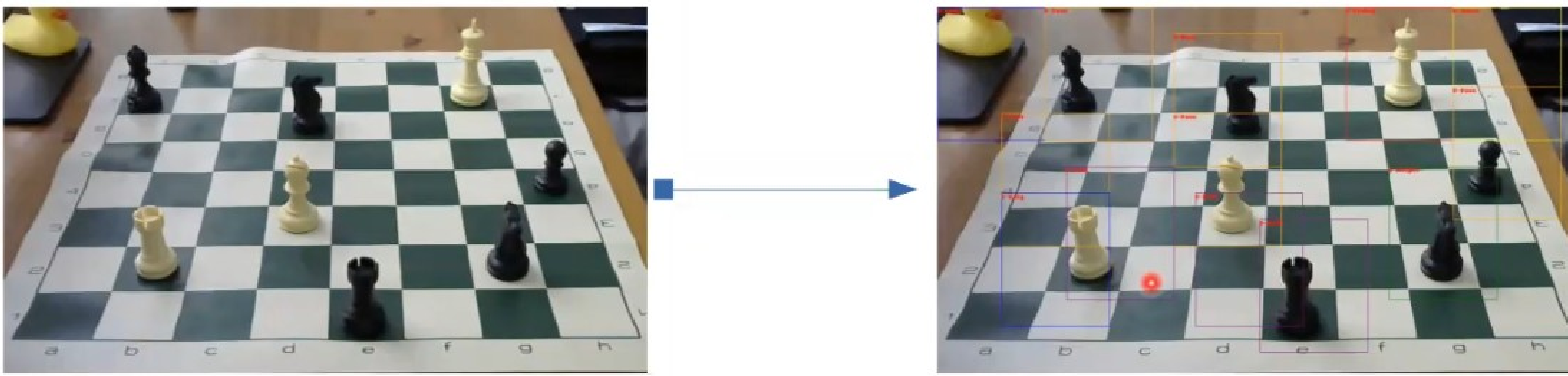
The same bishop appears in multiple boxes

Object Detection: Second Strategy

This method is adapted from “Hand-On Machine Learning ...”(Geron, page 486)

Method:
- Slide a bounding box across the board image
- Record all bounding boxes that ‘find’ pieces
- Collapse overlapping bounding boxes

Results:



Object Detection: Second Strategy

Details:

After ‘sliding’ the classifier across the image, we find many bounding boxes with relatively high (>70%) probability of containing chess pieces.

Of course the bounding boxes largely overlap – so each piece is overdetected

Original Image

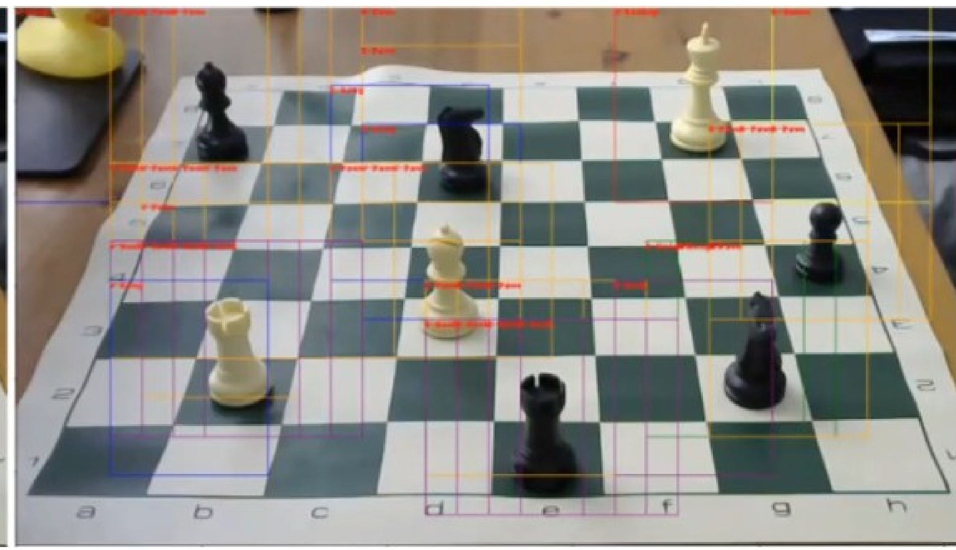


Image with all confident bounding boxes displayed

Object Detection: Second Strategy

We collapse redundant bounding boxes in the following way:

Starting with the highest-confidence bounding box:
    Remove all neighbors that largely overlap with it

Repeat until no large overlaps are left

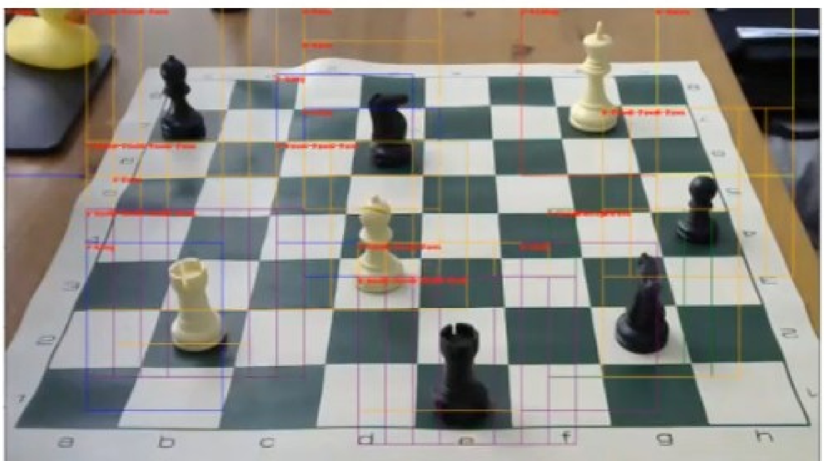


Image with all confident bounding boxes displayed

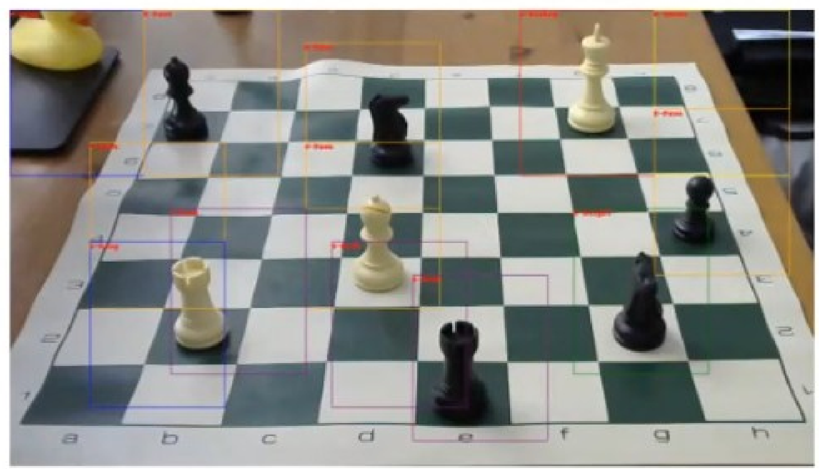


Image after removing redundant boxes

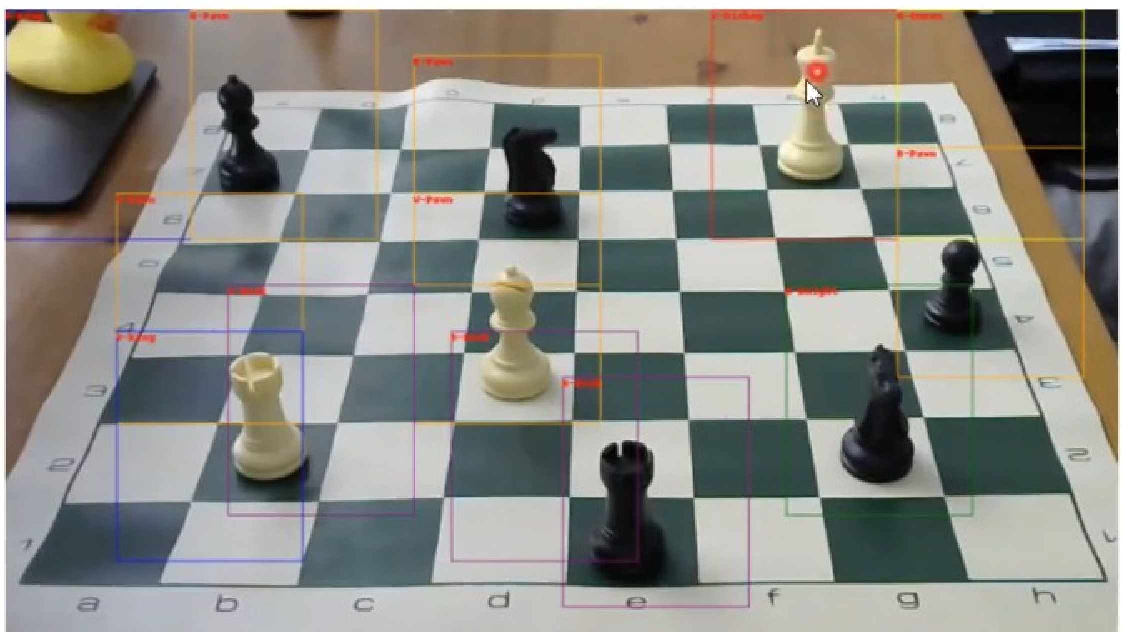
Object Detection: Lessons Learned

The problems with object detection are mostly inherited from problems with the classifier.

The classifier is bad at distinguishing certain pieces (Kings/Queens, Bishops/Pawns)

The classifier is bad at rejecting non-pieces, including partially in-frame pieces

Delightfully – it is determined to interpret the Rubber Duck as a King!



YouTube Video Presentation

YouTube video presentation:

https://www.youtube.com/watch?v=PZUjDJ6ALtw

Note: I accidentally deleted (without ever saving) these slides after recording the video. I then recreated the slides by taking screenshots from the video. That's why these slides are just big images.