Get Chess Board

Chess Screenshot → Analysis Board

r1b1k2r/pp2ppbp/2n3p1/q7/2BPP3/4B3/P3NPPP/R2QK2R

Motivation & Goals

Motivation

Watching chess on YouTube:
-But why not <random_chess_move>??
The video moves on without hearing you…

In a parallel universe:
The video still moves on without hearing you… but you open that position on a chess engine.

Goals

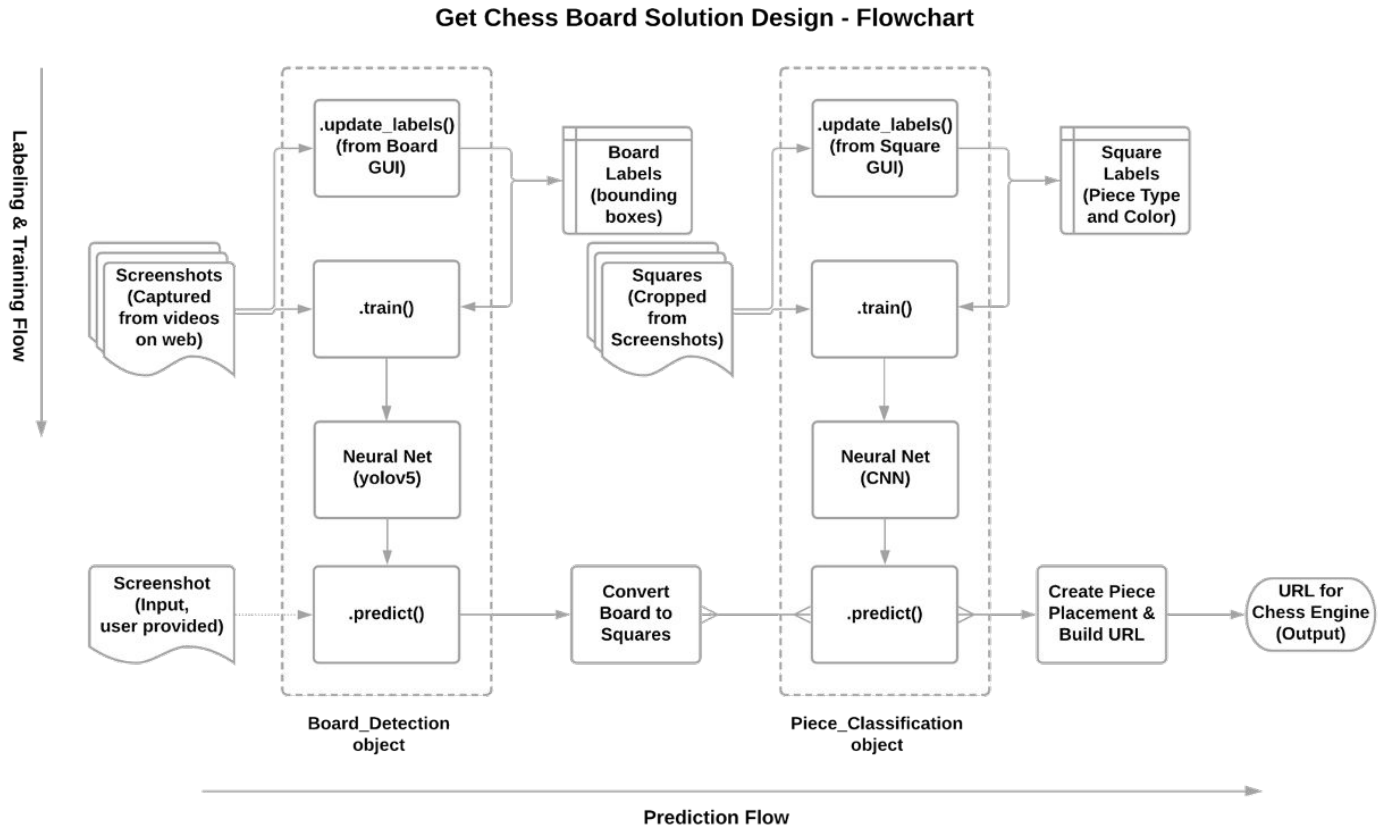
Decrease the friction to study a chess game played on a video platform. Some benefits:

1. **Shorten the feedback-loop on learning by making the analysis board available**
2. **Introduce a programmatic tool for capturing chess games from videos**
3. **Make watching chess more fun!**

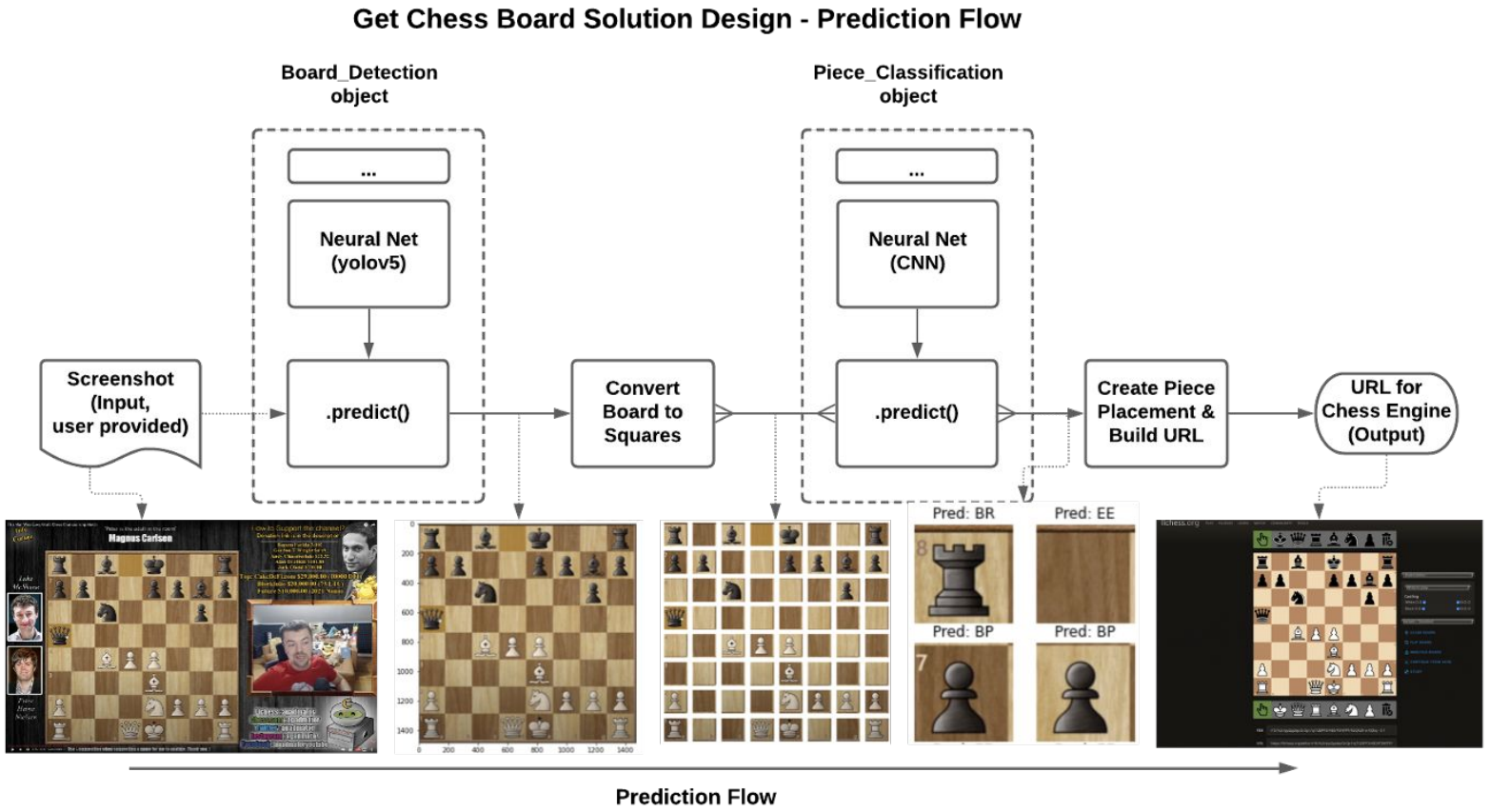
Project Objective: Take in a screenshot of a chess game on a video platform and return a string the represents the piece placement.

Methodology Design

-Break the problem down into manageable and relatively independent subproblems.



Focusing On Prediction Flow



Methodology Steps

0. **Data Acquisition & Labeling GUI:**
    a. **for Boards**
    b. **for Squares**
1. **Board Detection**
2. **Convert Board to Squares**
3. **Piece Classification**
4. **Creating Piece Placement & Building URL**

Steps: Data Acquisition & Labeling GUI - Boards

Design Goal: Lessen the cognitive load of the labeler. Abstract away the data handling.

To this end, the user should be able to:

- **-Label a chessboard object (set a bounding box) within the GUI with a mouse drag**
- **-Label multiple objects**
- **-Delete any labels that are not desired**
- **-Label screenshots in succession**
- **-Move to the next image (or exit) without worrying about the saving status**

Steps: Data Acquisition & Labeling GUI - Boards

- **Labeled around 80 screenshots in 2-3 hours**
- **Label data held in a csv - two rows can be seen in the image**
- **Has functions to handle randomization into training, validation and test**
- **Handles any conversions and requirements for yolov5.**

Steps: Data Acquisition & Labeling GUI - Squares

Design Goal: Lessen the cognitive load of the labeler. Abstract away the data handling.

To this end, the user should be able to:

- **-Label a square with mouse click - use radio buttons for their toggle property**
- **-Use minimum clicks:**
  - **-Default state of any square is set to E(mpty) - more than half of squares are empty**
  - **-Button added to confirm “Human Check” wholesale for the screen**
- **-Label sets of squares in succession**
- **-Move to the next image (or exit) without worrying about the saving status**

Steps: Data Acquisition & Labeling GUI - Squares

- **Labeled around 5600 squares in 5-6hours**
- **Label data held in a csv - sixteen rows can be seen in the image**
- **Has functions to handle randomization into training, validation and test**
- **Handles any conversions and requirements for yolov5.**

Steps: Data Acquisition & Labeling GUI - Square Counts

Count of Pieces by Color and Type – includes E(mpty) squares

	fname							
PcType-PRNBQKE	B	E	K	N	P	Q	R	All
PcColor-BWE								
B	115.0	-	87.0	110.0	524.0	64.0	143.0	1043
E	-	3540.0	-	-	-	-	-	3540
W	118.0	-	87.0	114.0	531.0	66.0	143.0	1059
All	233	3540	174	224	1055	130	286	5642

- **Empty squares more than half (62.7%), Pawns (18.7%), Rooks (5.1%), Bishops (4.1%), kNights (4%), Kings (3.1%), Queens (2.3%)**
- **Labels indicate unbalanced classes for classification**
- **With color split - the unbalance becomes more pronounced.**
- **Silver lining is black and white pieces seem balanced.**

Steps: Board Detection - Training

-Uses a relatively lightweight detection algorithm: yolov5 implemented on PyTorch, see yolo for the originating paper.**

-Goal: Detect the chessboard: Algorithm producing bounding boxes and chessboards being a box helps.

-Training:

- **-used 64 screenshots for training and 18 for validation**
- **-Initial weights are those of the small pretrained model provided by yolov5**

-Results:

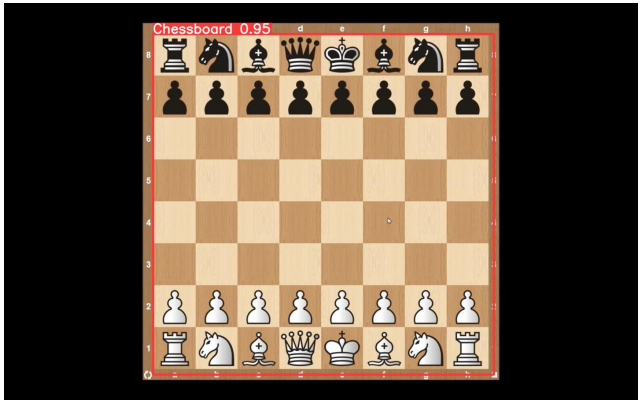
- **-Each epoch took ~4 minutes for a total of 599 (best @ 499) epochs at 39 hours.**
- **-Over validation set mAP@.5=0.96 and mAP@.5:.95=95.3**

```
Results of the experiment (actual best set, took over 39 hours, best epoch 498, end at 599.)
     Epoch   gpu_mem       box       obj       cls    labels  img_size
  498/1499        0G    0.0154  0.006404         0        49      1440: 100%|██████████| 4/4 [04:00<00:00, 60.11s/it]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|██████████| 1/1 [00:05<00:00,  5.33s/it]
                 all         18         24       0.96          1      0.987      0.953
     Epoch   gpu_mem       box       obj       cls    labels  img_size
  598/1499        0G   0.01687  0.004755         0        52      1440: 100%|██████████| 4/4 [03:32<00:00, 53.17s/it]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|██████████| 1/1 [00:05<00:00,  5.32s/it]
                 all         18         24       0.96          1       0.96      0.911
Stopping training early as no improvement observed in last 100 epochs. Best results observed at epoch 498, best model saved as best.pt.
To update EarlyStopping(patience=100) pass a new patience value, i.e. `python train.py --patience 300` or use `--patience 0` to disable EarlyStopping.
599 epochs completed in 39.422 hours.
```

*https://github.com/ultralytics/yolov5
**https://pjreddie.com/media/files/papers/yolo.pdf

Steps: Board Detection - Prediction on Test Set

Correct Classification



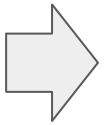
Correct Classification

False Positive

False Negative

Steps: Convert Board to Squares

Split the output from Board Prediction step into 64 squares



Steps: Piece Identification - Goal and Problem Notes

-Uses a CNN in Tensorflow with two Convolution&MaxPooling layers followed by two dense layers (one SoftMax). Similar to MNIST.

-Goal: Classify a square as empty (EE) and if not state the piece.

13 Classes: EE, WP, WR, WN, WB, WQ, WK, BP, BR, BN, BB, BQ, BK.

-Notes:

Solution has to generalize well across several dimensions:

- **Difference in shapes across sets**
- **Variation in chess square colors**
- **Pieces looking alike (e.g. kings and queens)**
- **Imbalance in multiclass data in favor of empty**
- **Improperly captured squares**

A naive calculation for 3% board error rate: (0.97)^(1/64) = 0.995

Steps: Piece Identification - Training

-Training

- 4231 squares for training, 1128 for validation at 80x80 during model input

- No pretrained weights (training from scratch)

- Callback function to save “the best” model: validation metric used if validation exists

-Final metric used: Cohen’s Kappa which has some adjustment for class imbalance. The other was ‘loss’.

Results:

-Best epoch as follows:

```
Epoch 00031: val_cohen_kappa improved from 0.97573 to 0.97573, saving model to
data/model/piece-train/exp/cnn_pieces.h5
133/133 - 5s - loss: 3.3121e-06 - accuracy: 1.0000 - cohen_kappa: 1.0000 - val_loss: 0.2934 -
val_accuracy: 0.9858 - val_cohen_kappa: 0.9757- 5s/epoch - 40ms/step
```

-Model ran for 40 epocs - ~5 seconds each.

Steps: Piece Identification - Training Analysis

Confusion Matrix for the Training Set – _pred stands for model prediction

	EE	WP	WR	WN	WB	WQ	WK	BP	BR	BN	BB	BQ	BK
EE_pred	2658	0	0	0	0	0	0	0	0	0	0	0	0
WP_pred	0	389	0	0	0	0	0	0	0	0	0	0	0
WR_pred	0	0	108	0	0	0	0	0	0	0	0	0	0
WN_pred	0	0	0	83	0	0	0	0	0	0	0	0	0
WB_pred	0	0	0	0	91	0	0	0	0	0	0	0	0
WQ_pred	0	0	0	0	0	50	0	0	0	0	0	0	0
WK_pred	0	0	0	0	0	0	70	0	0	0	0	0	0
BP_pred	0	0	0	0	0	0	0	392	0	0	0	0	0
BR_pred	0	0	0	0	0	0	0	0	106	0	0	0	0
BN_pred	0	0	0	0	0	0	0	0	0	80	0	0	0
BB_pred	0	0	0	0	0	0	0	0	0	0	91	0	0
BQ_pred	0	0	0	0	0	0	0	0	0	0	0	50	0
BK_pred	0	0	0	0	0	0	0	0	0	0	0	0	63

400 Randomly Chosen Correct Classifications [Training Set - 4231 Images]

-For correct classifications, most of the data seems to come from ‘clean cut’ squares.

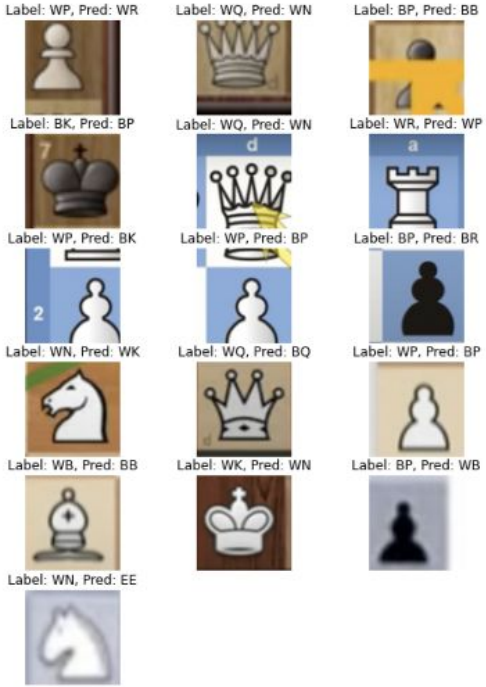
-Model seems robust to the color of chessboard square.

Steps: Piece Identification - Prediction on Validation Set

Confusion Matrix for the Validation Set – _pred stands for model prediction

	EE	WP	WR	WN	WB	WQ	WK	BP	BR	BN	BB	BQ	BK
EE_pred	707	0	0	1	0	0	0	0	0	0	0	0	0
WP_pred	0	109	1	0	0	0	0	0	0	0	0	0	0
WR_pred	0	1	28	0	0	0	0	0	0	0	0	0	0
WN_pred	0	0	0	24	0	2	1	0	0	0	0	0	0
WB_pred	0	0	0	0	19	0	0	1	0	0	0	0	0
WQ_pred	0	0	0	0	0	10	0	0	0	0	0	0	0
WK_pred	0	0	0	1	0	0	13	0	0	0	0	0	0
BP_pred	0	2	0	0	0	0	0	103	0	0	0	0	1
BR_pred	0	0	0	0	0	0	0	1	27	0	0	0	0
BN_pred	0	0	0	0	0	0	0	0	0	22	0	0	0
BB_pred	0	0	0	0	1	0	0	1	0	0	20	0	0
BQ_pred	0	0	0	0	0	1	0	0	0	0	0	11	0
BK_pred	0	1	0	0	0	0	0	0	0	0	0	0	19

All Incorrect Classifications [Validation Set - 1128 Images]



-For correct classifications, most of the data seems to come from ‘clean cut’ squares.

-Model seems robust to the color of chessboard square.

Steps: Create Piece Placement & Build URL - Placement

Use FEN Piece Placement String notation and insert the remaining 5 strings (Active Player, Castling Availability, En Passant Target, Half Move, Full Move) so that any mistakes can be fixed quickly in the engine.

```
r.b.k..r
pp..ppbp
..n...p.
q.......
..BPP...
....B...
P...NPPP
R..QK..R
```

FEN Piece Placement String: r1b1k2r/pp2ppbp/2n3p1/q7/2BPP3/4B3/P3NPPP/R2QK2R

Full FEN String: r1b1k2r/pp2ppbp/2n3p1/q7/2BPP3/4B3/P3NPPP/R2QK2R w KQkq - 0 1

Steps: Create Piece Placement & Build URL - Build URL

Final step for this project - observe Lichess API and create a URL. Could be extended to other sites/engines.

https://lichess.org/editor?fen=r1b1k2r%2Fpp2ppbp%2F2n3p1%2Fq7%2F2BPP3%2F4B3%2FP3NPPP%2FR2QK2R+w+KQkq+-+0+1

Conclusion + Next Steps

Project has achieved what it has set out to do:

Take in a screenshot of a chess game on a video platform and return a string the represents the piece placement.

Some next steps could be:

- **Add-ons:**
  - **Write a new class wraps the end-to-end process**
  - **Make model available in Docker for widespread use**
  - **Deploy on the cloud for even wider adoption**
- **Model Improvements:**
  - **Experiments to judge relative impact of board detection vs piece classification for appropriate prioritization**
  - **Test to see if classification would improve by a hierarchical classification**
  - **Generate more training and validation data**
  - **Test to see if a model could be estimated the direction of play**

Thank you + Open Floor for Discussion

Goal: Decrease the friction to study a chess game played on a video platform

Project Objective: Take in a screenshot of a chess game on a video platform and return a string the represents the piece placement.

Contributions:

-End-to-end neural network based piece placement generation from a screenshot.

-Transfer learning from an object detection model (via PyTorch)

-Bottom up training of a fully connected classifier (Tensorflow)

-Creation of chess specific (& easily extendable) platform independent GUI.