Chess Game Tracking Using Computer Vision and Deep Learning

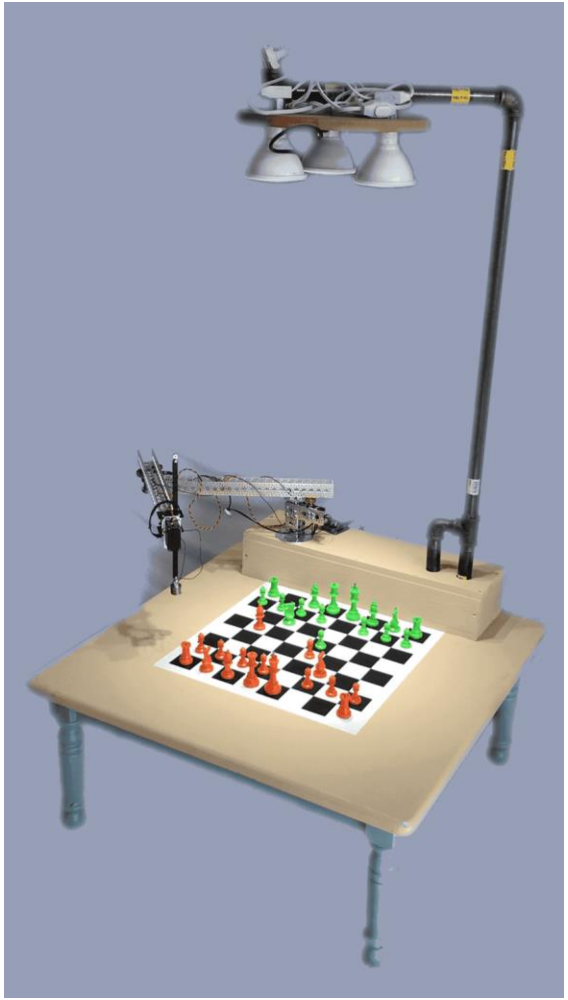
Jerry Liu

Motivation

- In tournaments, players must manually record every move played
- There are specialized chess sets that automatically record every move played, but they are expensive
- Tracking each game with computer vision is an inexpensive solution

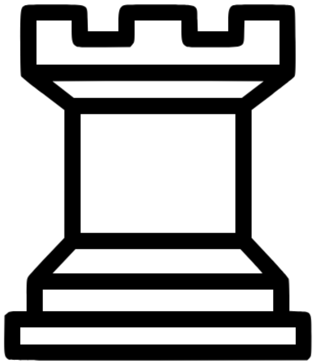
Previous Work

- Previous work has used an overhead view of the chessboard
    - Difficult to distinguish between certain pieces (e.g. pawns and bishops)
- Specialized colored chess sets (e.g. red and green pieces)



My Project

- Track a chess game using computer vision
  - Camera mounted over chessboard
    - 45 degree angle
  - Piece Detection
    - Pickle Module and Keras Library
  - Board Detection
    - OpenCV Library



Piece Detection

- Convolutional Neural Network
  - Training with database of chess pieces
    - Google Images
    - Manual Collection
  - Keras

Database of Pieces

- 7 folders of images corresponding to each piece
  - Empty square, pawn, knight, bishop, rook, queen , king
- Started off using only sample images scraped from Google Images
- Later, added pictures manually taken at angles

Google Images Pictures

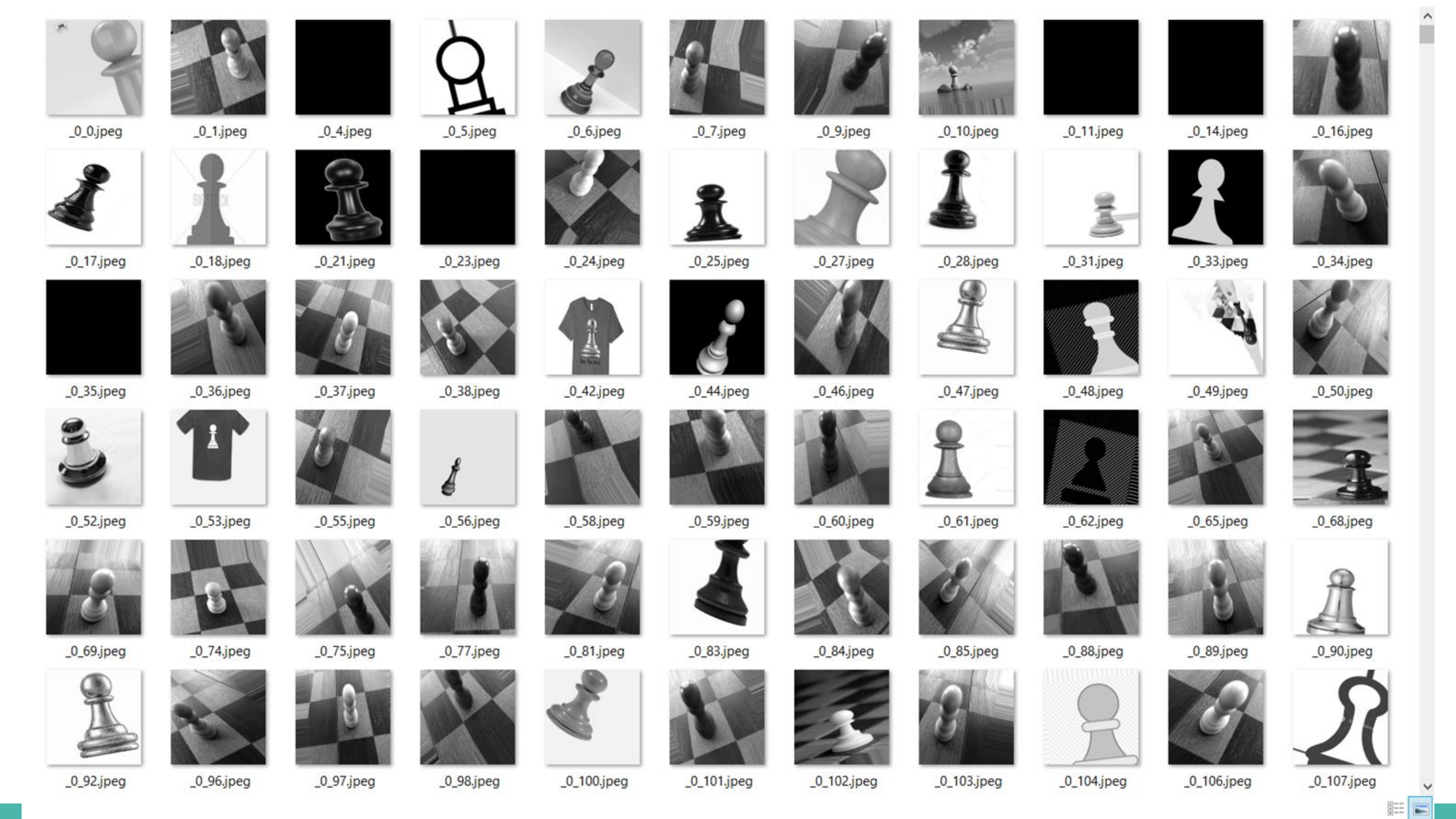
- Google Images as source
  - Used Javascript to create a txt file of all the urls of the images for each piece
  - Used Python to download the images into folders of respective pieces
- Folders containing images of each chess piece
- Filtered bad images to approx. 200-300 base examples for each folder

Manual Collection

- Pieces from Google Images were not representative of the pieces I wanted to test
- Manually collected data at 30 - 60 degree angles
  - 200 samples per piece

Data Standardization and Magnification

- Using Keras to standardize and augment training examples
  - Standardized to 100 x 100 grayscale .jpg files
  - Reflections and Rotations
  - Augmented to approx. 4500-6000

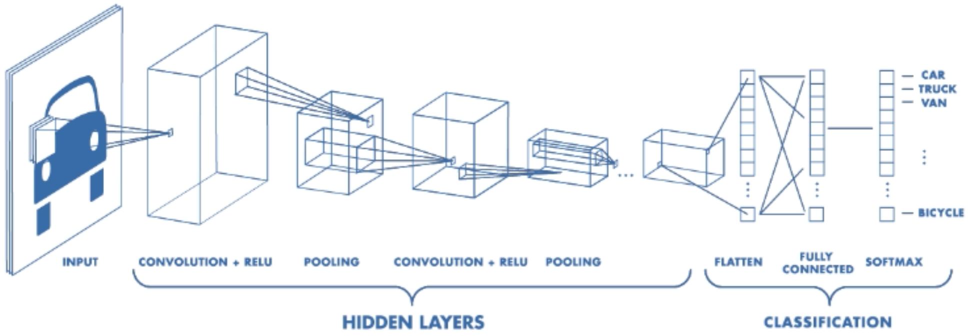

_0_0.jpeg
_0_1.jpeg
_0_4.jpeg
_0_5.jpeg
_0_6.jpeg
_0_7.jpeg
_0_9.jpeg
_0_10.jpeg
_0_11.jpeg
_0_14.jpeg
_0_16.jpeg
_0_17.jpeg
_0_18.jpeg
_0_21.jpeg
_0_23.jpeg
_0_24.jpeg
_0_25.jpeg
_0_27.jpeg
_0_28.jpeg
_0_31.jpeg
_0_33.jpeg
_0_34.jpeg
_0_35.jpeg
_0_36.jpeg
_0_37.jpeg
_0_38.jpeg
_0_42.jpeg
_0_44.jpeg
_0_46.jpeg
_0_47.jpeg
_0_48.jpeg
_0_49.jpeg
_0_50.jpeg
_0_52.jpeg
_0_53.jpeg
_0_55.jpeg
_0_56.jpeg
_0_58.jpeg
_0_59.jpeg
_0_60.jpeg
_0_61.jpeg
_0_62.jpeg
_0_65.jpeg
_0_68.jpeg
_0_69.jpeg
_0_74.jpeg
_0_75.jpeg
_0_77.jpeg
_0_81.jpeg
_0_83.jpeg
_0_84.jpeg
_0_85.jpeg
_0_88.jpeg
_0_89.jpeg
_0_90.jpeg
_0_92.jpeg
_0_96.jpeg
_0_97.jpeg
_0_98.jpeg
_0_100.jpeg
_0_101.jpeg
_0_102.jpeg
_0_103.jpeg
_0_104.jpeg
_0_106.jpeg
_0_107.jpeg

Saving my Data

- I used the Python pickle module to save the files of my training folders into individual files
- Each category of piece corresponded to a label
  - e.g. empty = 0, pawn = 1, knight = 2 etc.
- x.pickle contained each image and y.pickle contains the corresponding label

Convolutional Neural Network (CNN)

- A convolutional neural network is a type of deep neural network, most commonly applied to analyzing pictures and visual data
- Uses hidden [illegible]es from an image



Training

- Training remotely to the Duke Computer in Rm 202
  - Strong GPU - shorter training time
- CNN structure:
  - 6 convolutional layers - RELU activation function
  - 5 dropout layers
  - Fully-connected layer with 1000 neurons - RELU
  - Fully-connected layer with 7 neurons (corresponds to each piece) - Softmax
- To about 92% accuracy

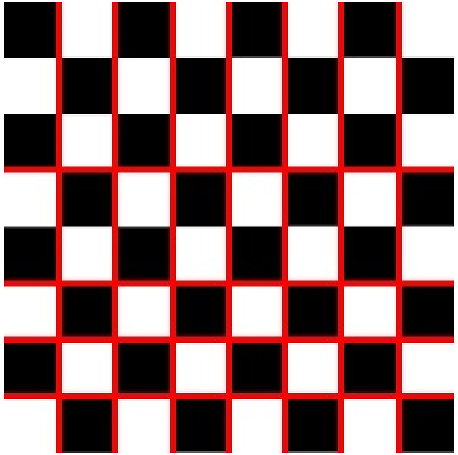
```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 100, 100, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 50, 50, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 50, 50, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 50, 50, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 25, 25, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 25, 25, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 25, 25, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 128)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 12, 12, 256)       295168
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 6, 6, 256)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 6, 6, 512)         1180160
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 3, 3, 512)         0
_________________________________________________________________
dropout_4 (Dropout)          (None, 3, 3, 512)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 3, 3, 4024)        18546616
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 1, 1, 4024)        0
_________________________________________________________________
dropout_5 (Dropout)          (None, 1, 1, 4024)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1000)              4025000
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 7007
=================================================================
Total params: 24,147,199
Trainable params: 24,147,199
Non-trainable params: 0
```

Saving the Model

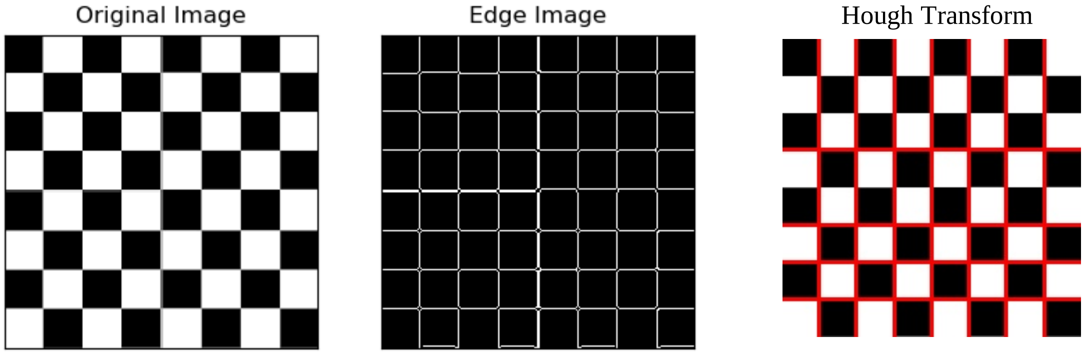
- I saved my neural network in a file called chess.h5
- Using this file, I created a python classifier program that when given an image, classifies it as one of 7 categories (i.e. empty, pawn, knight...)

Board Detection

- Line Detection
  - Canny Edge Detector
  - Hough Transform
- Corner Detection
  - Intersections between perpendicular lines
  - Non-maximum suppression



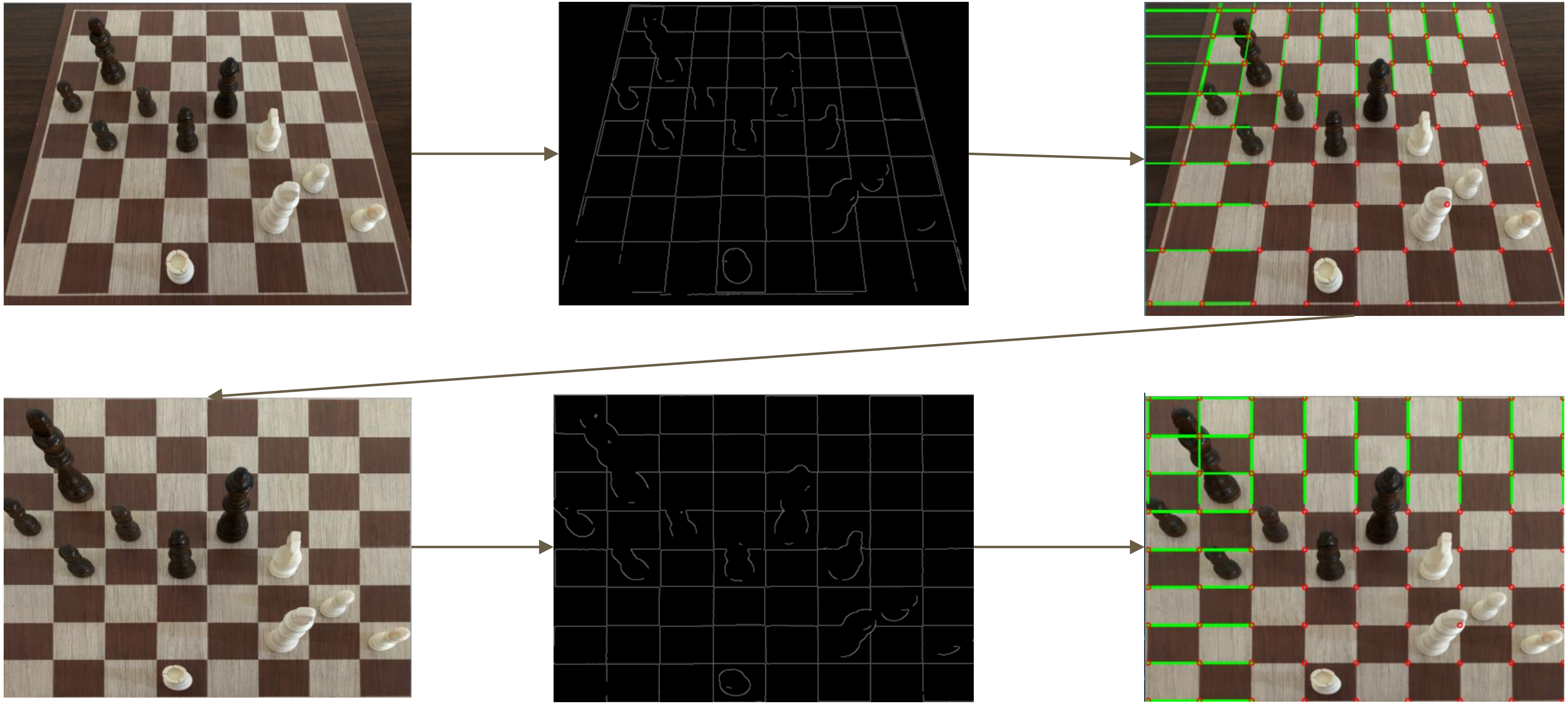
Canny Edge Detector and Hough Transform



Perspective Transform

- I used the far corners of the board to create a perspective transform of the image
- Then, I found the lines and corners again in the new image

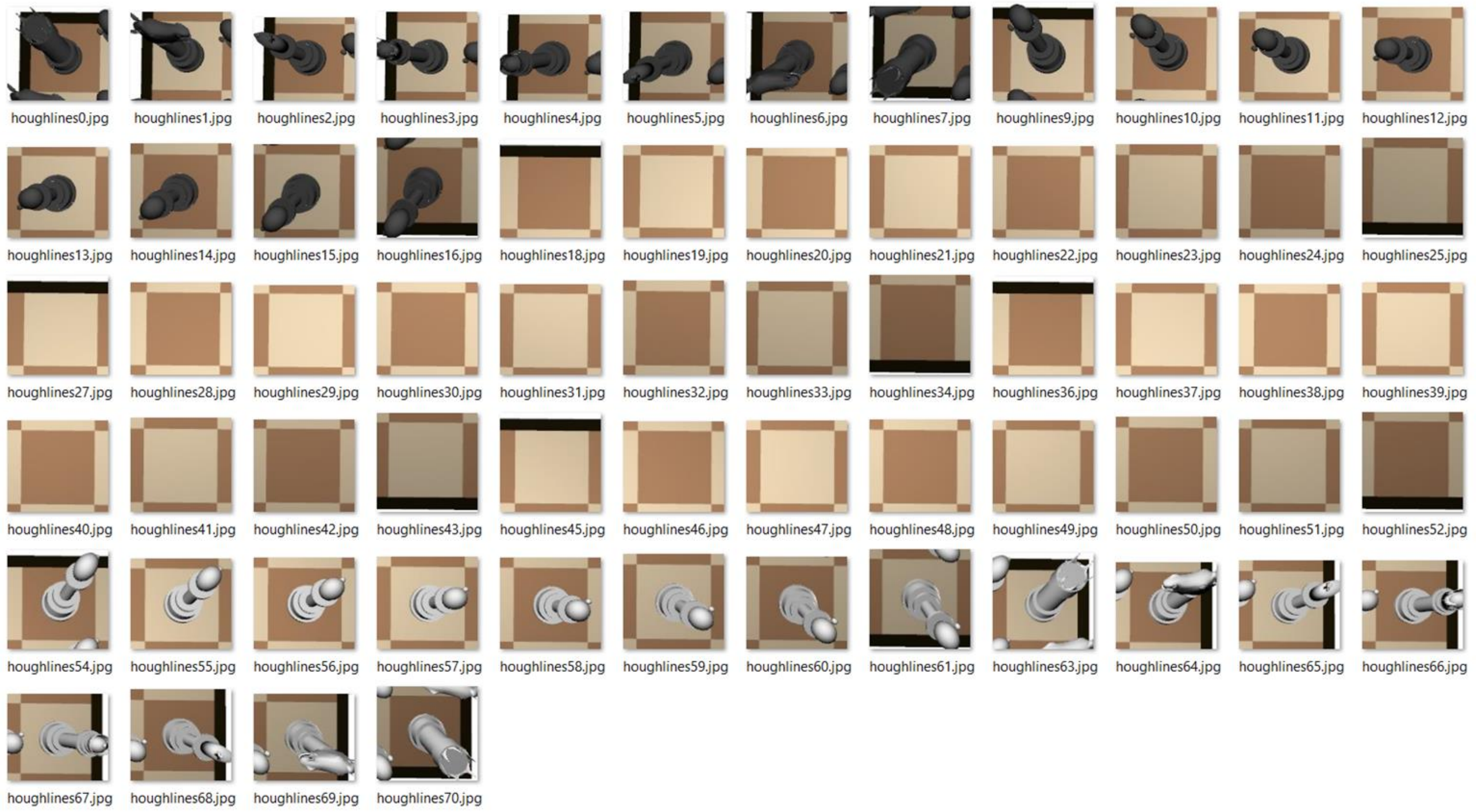
Perspective Transform

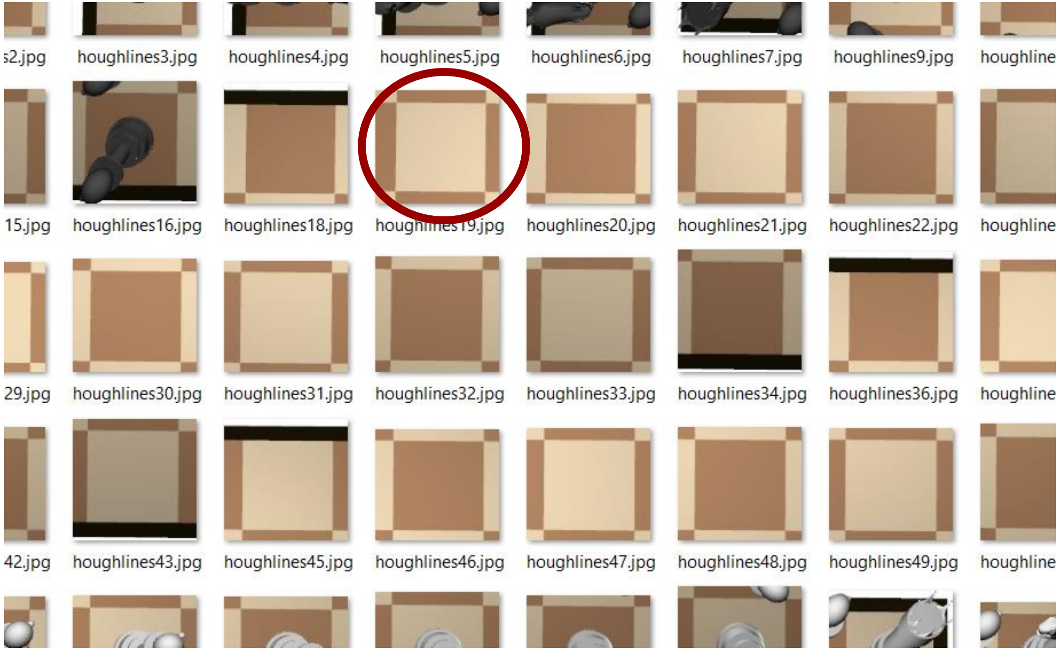


Process

- 64 images are cut out and processed - uses corners
- Each picture is run through the classifier to predict the piece (or if it's an empty square)
- The predicted board is then constructed using this information

Demonstration


houghlines0.jpg
houghlines1.jpg
houghlines2.jpg
houghlines3.jpg
houghlines4.jpg
houghlines5.jpg
houghlines6.jpg
houghlines7.jpg
houghlines9.jpg
houghlines10.jpg
houghlines11.jpg
houghlines12.jpg
houghlines13.jpg
houghlines14.jpg
houghlines15.jpg
houghlines16.jpg
houghlines18.jpg
houghlines19.jpg
houghlines20.jpg
houghlines21.jpg
houghlines22.jpg
houghlines23.jpg
houghlines24.jpg
houghlines25.jpg
houghlines27.jpg
houghlines28.jpg
houghlines29.jpg
houghlines30.jpg
houghlines31.jpg
houghlines32.jpg
houghlines33.jpg
houghlines34.jpg
houghlines36.jpg
houghlines37.jpg
houghlines38.jpg
houghlines39.jpg
houghlines40.jpg
houghlines41.jpg
houghlines42.jpg
houghlines43.jpg
houghlines45.jpg
houghlines46.jpg
houghlines47.jpg
houghlines48.jpg
houghlines49.jpg
houghlines50.jpg
houghlines51.jpg
houghlines52.jpg
houghlines54.jpg
houghlines55.jpg
houghlines56.jpg
houghlines57.jpg
houghlines58.jpg
houghlines59.jpg
houghlines60.jpg
houghlines61.jpg
houghlines63.jpg
houghlines64.jpg
houghlines65.jpg
houghlines66.jpg
houghlines67.jpg
houghlines68.jpg
houghlines69.jpg
houghlines70.jpg



s2.jpg
houghlines3.jpg
houghlines4.jpg
houghlines5.jpg
houghlines6.jpg
houghlines7.jpg
houghlines9.jpg
houghline
15.jpg
houghlines16.jpg
houghlines18.jpg
houghlines19.jpg
houghlines20.jpg
houghlines21.jpg
houghlines22.jpg
houghline
29.jpg
houghlines30.jpg
houghlines31.jpg
houghlines32.jpg
houghlines33.jpg
houghlines34.jpg
houghlines36.jpg
houghline
42.jpg
houghlines43.jpg
houghlines45.jpg
houghlines46.jpg
houghlines47.jpg
houghlines48.jpg
houghlines49.jpg
houghline

empty: 100.00%
pawn: 0.00%
rook: 0.00%
bishop: 0.00%
queen: 0.00%
knight: 0.00%
king: 0.00%

```
('A1', 'pawn')
('B1', 'rook')
('C1', 'empty')
('D1', 'rook')
('E1', 'rook')
('F1', 'rook')
('G1', 'rook')
('H1', 'pawn')

('A2', 'knight')
('B2', 'bishop')
('C2', 'empty')
('D2', 'pawn')
('E2', 'empty')
('F2', 'pawn')
('G2', 'knight')
('H2', 'pawn')

('A3', 'knight')
('B3', 'bishop')
('C3', 'pawn')
('D3', 'empty')
('E3', 'empty')
('F3', 'empty')
('G3', 'empty')
('H3', 'empty')

('A4', 'empty')
('B4', 'empty')
('C4', 'empty')
('D4', 'empty')
('E4', 'empty')
('F4', 'empty')
('G4', 'empty')
('H4', 'empty')

('A5', 'empty')
('B5', 'empty')
('C5', 'empty')
('D5', 'empty')
('E5', 'empty')
('F5', 'empty')
('G5', 'empty')
('H5', 'empty')
```

Problems

- Inaccurate Classification
- Piece obstruction
- Difficulty in piece capture for each square
  - Multiple pieces in one picture

Unfinished Work

- Neural network needs fine tuning
- Needs ability to distinguish pieces by color

Acknowledgements

- I would like to acknowledge my Computer Systems Research Lab director, Dr. Zacharias

Questions?