COMPUTER VISION
Final Project: AI Chess Master

1. Introduction

This article will discuss the problem of identifying pieces and their positions in chessboard pictures. Object recognition algorithms generally detect objects from two dimensions: classification and localization. The first pertains to recognizing the object's characteristics, while the second refers to the positioning object's location. Traditional detection techniques often share the region box (Proposal) and feature extractor (Sibling head) of the probable existence of objects, learning classification and regression simultaneously. This detection method's weakness is that the recognition accuracy is poor and the classification confidence of the final output image frame differs from the accuracy of the detection frame.

This was shown to be due to the difference between classification tasks and regression tasks, wherein classification tasks place a stronger priority on areas with rich semantic information and regression tasks place a greater emphasis on object boundaries.
As a result, the detection outcomes will be impacted by sharing the region box (Proposal) and feature extractor (Sibling head) of the object's probable existence.

A similar effort employs computer vision methods and convolutional neural networks (CNN) to develop a classification algorithm to ascertain the location of the pieces on the chessboard for the chessboard detection challenge. In order to see the results, the end program outputs a 2D image of the chessboard while saving the complete image and visualizing it.

The project uses computer vision techniques to find the features of the chessboard, and then converts these features to the coordinates of the outer boundary and 64 separate squares. The process is centered on the intersection of intersecting horizontal and vertical lines generated by Canny edge detection and Hough transform. And use hierarchical clustering to group intersections by distance and average the groups to create the final coordinates (the process is shown in Figure 1.).

Finally, by using transfer learning, the chess piece features are learned and classified using VGG16. The evaluation results of the project are:

- Pros: Null - 99% accuracy, 100% recall; White and Black (WP and BP) - F1 score around 95%.
- Cons: White Knight (WN) - High recall (98%) but low accuracy (65%); White Bishop (WB) - lowest recall at 74%.

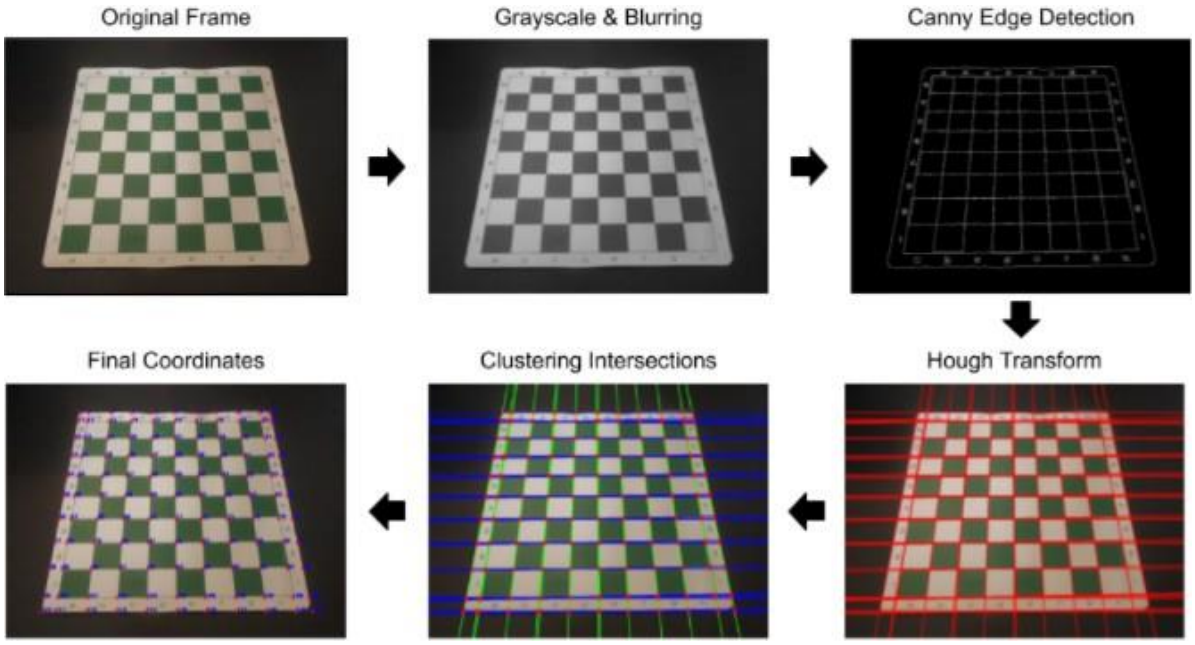

Figure 1. Full Chessboard Detection Process [2]

2. Methodology

1. Dataset

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
2 Kings are guaranteed to be on the board.

Labels are in a filename in Forsyth–Edwards Notation format, but with dashes instead of slashes.

2. Model Explanation

I've selected a Kaggle [4] solution for this. Given that there are only the two primary pawn classifications in this puzzle—black and white— Additionally, the chessboard's color is irrelevant and can be regarded as empty.

To decrease the dimension, our approach first just scans the image in grayscale. The author then separates the chessboard into 64 sub-pictures in accordance with the grid and organizes them in an array as the input of the model due to the uniform size and format of all the images. Extract the label for the image from the file title at the same time and use one hot encoding to convert it from a string to a numeric value. It then classifies sub-images using a CNN. White (B, K, Q, R, P, N), black (B, K, Q, R, P, N), and empty make up the total of 13 classes.

For the CNN model, the author used the same architecture as in the Keras official documentation example. The original model is designed for the classification of the CIFAR10 dataset [5], which consists of 60,000 32*32 color images in 10 classes. Since the two cases are highly similar, both are to classify small-size images, so the author chooses to transplant the original model and only make appropriate adjustments for the input and output.
The CNN architecture summary can be seen in Figure 2. During training, this CNN uses "rmsprop" as the optimizer, "categorical_crossentropy" as the loss function, and "accuracy" as the metrics.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 50, 50, 32)        320
_________________________________________________________________
activation_1 (Activation)    (None, 50, 50, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 48, 48, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 48, 48, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 24, 24, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 24, 24, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 22, 22, 64)        36928
_________________________________________________________________
activation_4 (Activation)    (None, 22, 22, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 11, 11, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 11, 11, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 7744)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3965440
_________________________________________________________________
activation_5 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 13)                6669
_________________________________________________________________
activation_6 (Activation)    (None, 13)                0
=================================================================
Total params: 4,037,101
Trainable params: 4,037,101
Non-trainable params: 0
_________________________________________________________________
```

Figure 2. The summary of the CNN model.

3. Main library

- OpenCV: We use the OpenCV library to read the image file. Since chess only has white/black pieces, and we do not care about the chessboard (we define background as empty), we use cv2.IMREAD_GRAYSCALE to read the image in gray mode to reduce the dimension of images.
- Keras: This library is imported to build the CNN model.
- Sklearn: The method of classification_report from Sklearn is used to generate the text report for classification.

4. Data processing

There is no need to modify the photographs because every picture in this collection has the same standard size and format. Additionally, since this dataset contains 100,000 photos and the final model performs excellently, no augmentation is necessary.

In addition, according to the settings of this dataset, most areas of the chessboard will be empty, so the sub-images with empty labels will occur most often. Therefore, in the model training process, to balance the data distribution, this project trains the empty and non-empty images in a ratio of 1:2 to avoid the negative impact of data imbalance on the training results.

5. Feature engineering

Since each chessboard is 8*8 in size, we can subdivide each chessboard image into 64 sub-images in order for feature extraction and classification. In this way, the whole problem is changed from the feature classification and localization of the chessboard image to a simple classification problem of 64 sequentially arranged sub-images. Figure 3. Shows the subdivision example of the image in Figure 4.

Figure 3. 64 sub-images divided from the original image.

6. Dimensionality reduction

This model uses cv2.IMREAD_GRAYSCALE method to read the images in gray mode to reduce the dimension of each image from (400,400,3) to (400,400,1). An example can be seen in Figure 4.

Figure 4. The raw image and the gray mode of it after dimensionality reduction.

3. Results

1. Performance

The testing process has an accuracy of 99.78% However, after comparing the solutions of other people on Kaggle, the biggest advantage of this model is that its overall time complexity is very low.

2. Metrics

For the training process, the model uses “accuracy ” as the evaluation metric. For the testing process, by using the classification_report method from Sklearn, it generates a text summary of the precision, recall, and F1 score for each class. As shown in Figure 5.

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	17700
1.0	1.00	1.00	1.00	17695
2.0	1.00	1.00	1.00	20000
3.0	1.00	1.00	1.00	20000
4.0	1.00	1.00	1.00	8785
5.0	1.00	1.00	1.00	8768
6.0	1.00	1.00	1.00	17536
7.0	1.00	1.00	1.00	17889
8.0	1.00	1.00	1.00	18119
9.0	1.00	1.00	1.00	17930
10.0	1.00	1.00	1.00	17606
11.0	1.00	1.00	1.00	17628
12.0	1.00	1.00	1.00	1080344
accuracy			1.00	1280000
macro avg	1.00	1.00	1.00	1280000
weighted avg	1.00	1.00	1.00	1280000

Figure 5. The classification report of the testing.

4. Conclusion

This project shows how to categorize chessboard photos and generate the FEN names that go with them. The strategy separates the board into 64 smaller images using the divide and conquer principle, and then employs CNN to train all of the smaller images in the training set for feature classification. Finally, it generates FEN names and classifies test photos using the trained model.

5. Reference

1. Underwood, A. (2021, December 16). *Board Game Image Recognition using Neural Networks - Towards Data Science*. Medium. https://towardsdatascience.com/board-game-image-recognition-using-neural-networks-116fc876dafa
2. Koryakin, P. (2019). *Chess Positions*. Kaggle. https://www.kaggle.com/koryakinp/chess-positions
3. Y. (2019, November 10). *Chess Positions FEN generator*. Kaggle. https://www.kaggle.com/yeahlan/chess-positions-fen-generator