Vision-based Robotic Chess Player

Mingyu Wang

A thesis

submitted in partial fulfillment of the

requirements for the degree of

Master of Science in Mechanical Engineering

University of Washington

2021

Committee:

Xu Chen, Chair

Santosh Devasia

Sawyer Fuller

Program Authorized to Offer Degree:

Mechanical Engineering

University of Washington

Abstract

Vision-Based Robotic Chess Player

Mingyu Wang

Chair of the Supervisory Committee:
Xu Chen
Mechanical Engineering

A collaborative robot is a type of robot built to interact with humans in a shared workspace physically. This thesis will introduce a system that manipulates a collaborative robot to play a chess game with a human. This system heavily relies on computer vision to plan the robot's action. The thesis will present a workflow of pose estimation about a chessboard's position to the robot base and the process of using transfer learning to develop a pre-trained neural network to detect the chess pieces. The detection of the chess piece is based on images collected in real-time, and one major factor that influences the picture is the lighting condition. Some suggest that data augmentation can improve the neural network's ability against light variation. This thesis compares the performance of neural networks trained with the computer augmented dataset and real collected dataset, and studies the data augmentation effects about the neural network. Results will suggest that the real collected dataset is still better than the computer augmented dataset, and data augmentation can improve the neural network's performance but has limitations.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Number Page

Acknowledgments

I would like to express my tremendous appreciation to Professor Xu Chen for giving me this opportunity to join his lab and gain the knowledge and experience from this thesis project. I also would like to thank my mentor, Hui Xiao, for his guidance and the immense support of this research. His advice and help have made this work progress steadily. Additionally, I would like to thank Professor Santosh Devasia and Professor Sawyer Fuller for becoming my committee members. Lastly, I would like to thank my family and friends, who have always been a constant source of support and encouragement.

Chapter 1: Introduction

1.1 Thesis Introduction

The world is marching into the autonomous era as humans continuously improve production efficiency and reduce costs. A consensus is that cooperation between humans and robots can work much better than just human or just robot teams alone. One problem that restraints the robot's implementation is that it cannot precept and react to its surroundings. Hence, computer vision is introduced to the robot system to solve this obstacle. With the visual information, the robot can have the ability to precept the surrounding environment and make necessary adjustments and decisions.

The thesis carried this idea and aimed to build a system that uses the visual information detected by computer vision and drives a collaborative robot to interact with humans and to finish a task together. The chess game was selected as the task. This game is highly interactive. The robot will react after a human makes a move. More than that, the system needs to detect the location of the chessboard and each chess piece on the fly. Therefore, this task involves problems in detection, decision, and action.

Wining the chess game was the first objective of this thesis. Therefore, searching for a powerful chess engine is necessary. Secondly, developing a program that can detect the chessboard's location was important. The thesis utilizes the functions provided by OpenCV to create a workflow of series image processing and traditional detection methods to solve this problem. The third task was detecting and locating the chess piece on the chessboard. Transfer learning was considered to solve this challenge. We first needed to select an appropriate neural network, training method, and fully connected layers. The next was collecting the dataset.

In industry, applying a collaborative robot in the production line will also face this thesis's problems. Hence this thesis can give insight into the application of robots.

1.2 Related Project

There is a paper called “Gambit: An Autonomous Chess-Playing Robotic System” [1]. This paper introduces a robotic arm system integrated with computer vision to play a chess game with a human player. The robot in the article had two visual sensors, one depth camera, and one small USB camera. The main goal was to develop a system that could successfully pick up the chess piece no matter where the piece was placed in the square and move to the correct square without colliding with other pieces. The overlaps between the paper and our thesis are locating the chessboard and detecting the pieces on it.

The prior system first detects the corner points on the chessboard's 2D (RGB) image to locate the board and the grid of squares. Then it incorporates the depth information of corner points and uses the RANSAC algorithm to find the plane where most points lied. It chooses the best match of all 3D points on the board plane with a template of the eight-by-eight contiguous chessboard cells to localize the board. Next, the system uses the point cloud consisting of all points above the board plane to detect the chessboard occupancy. It projects these points on the chessboard plane and examines which square has points in it.

The paper proposes a detection and recognition system that consists of four hierarchical classifiers [1]: “a square detector, a binary piece/background detector, a binary piece color (white/black recognizer, and two piece-type recognizers, each with the six class labels {B, N, K, P, Q, R}”. The square detector is used to find the board squares. The author extracted histograms of oriented gradients feature of 20 square templates, and the system uses those features to detect.

The piece/background detector uses a trained SVM to reconfirm a square is empty or non-empty. Finally, the features of the chess piece classifier are concatenated SIFT and kernel descriptors.

The most significant difference between the paper's system and ours is that ours only relies on one 2D RGB camera. We explored and stretched the capability of a 2D RGB camera. Our system uses that camera to capture the image for locating the chessboard and detecting the chess pieces. Besides, we hardcoded the chess pieces' size into the program, so the robot actions had already been planned not to hit other pieces during the execution.

1.3 Locating Object from The Image

1.3.1 Perspective Transformation

The image formation of a Pin-hole camera with a lens is the perspective transformation. A convention for describing this idea is using the central perspective imaging model. The model is shown below.

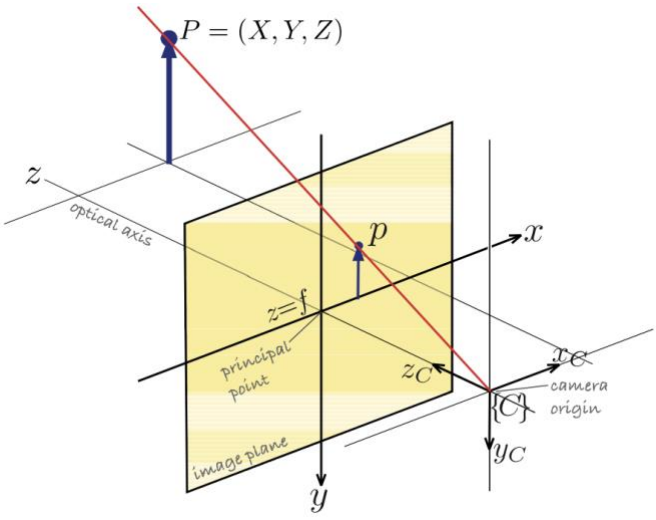

Figure 1.1 The central perspective imaging model [2]

The first important part is the camera frame, which is the $\{C\}$ in the image. It is the frame used to describe the location of points to the camera. The frame's Z-axis, Z_C, is collinear with the principal axis of the camera's lens and pointing out from the camera. An image plane is a plane for

forming a non-inverted image. The center of this plan is on the z-axis, at $z = f$, where f is the focal length. Let a point $P = (X, Y, Z)$ be reprojected on the image as p. The point p is calculated using similar triangles because light transmits straight and converges on the origin because of image formation. The function of the x and y value of point p is shown below:

$$x = f * \frac{X}{Z},\ y = f * \frac{Y}{Z} \tag{1.1}$$

The camera uses a plane of image sensors to capture light and forms an image, so the previously calculated image point is caught inside an image sensor in pixels. They are commonly square, and their edge length is ρ. The coordinate vector inside the image plane includes width index u and height index v. Each index starts from zero, and the index of the center pixel, where the camera frame's Z-axis passes through, is (u_0, v_0). The pixel index is calculated by:

$$u = \frac{x}{\rho} + u_0,\ v = \frac{y}{\rho} + v_0 \tag{1.2}$$

The process of the coordinate transformation can be written in matrix form, with the coordinate written in the homogenous form:

$$\begin{pmatrix} u' \\ v' \\ w \end{pmatrix} = \begin{pmatrix} 1/\rho & 0 & u_0 \\ 0 & 1/\rho & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{1.3}$$

Then, u and v can be obtained by dividing u' and v' by w.

1.3.2 Pose Estimation

Pose estimation estimates an object's location and orientation to the camera. It is developed from perspective transformation.

In the previous introduction, the coordinate point *P* is in the camera frame, but now the coordinate is in the object's frame for pose estimation. Thus, there is a transformation matrix from

the object's frame to the camera frame, and that transformation is the pose of the object to the camera. The transformation matrix's structure follows:

$$\begin{pmatrix} R & T \\ 0 & 1 \end{pmatrix} \tag{1.4}$$

where R is a three-by-three rotation matrix. The T is a three by one translation vector. The zero is a one-by-three zero vector, and 1 is the scaler. The previous matrix form of the coordinate transformation is then modified to become:

$$\begin{pmatrix} u' \\ v' \\ w \end{pmatrix} = \begin{pmatrix} 1/\rho & 0 & u_0 \\ 0 & 1/\rho & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{1.5}$$

The vector, (u', v', w), is set to $(u, v, 1)$ for further processing. This is based on the property of the homogenous form. The goal for pose estimation is solving the transformation matrix and the twelve unknowns. Hence, four points' pixel information and coordinates are needed.

The thesis uses the OpenCV's built-in function, solvePNP, to solve this problem. This built-in function can take more than four points' information. It is an iterative method based on a Levenberg-Marquardt optimization. First, the function randomly assigns the parameters to the transformation matrix and finds the coordinates' reprojected pixel accordingly. The function then calculates reprojection error, the sum of squared distances between the feature extracted points and their projections. Finally, the function minimizes this error by the optimization algorithm until the error value is under the threshold.

1.4 Transfer Learning

1.4.1 Transfer Learning Introduction

Transfer Learning is one popular method in the application of neural networks. The main idea of transfer learning is to use a neural network trained on a similar task as a start point to develop a new neural network to solve a given task.

Pan and Yang gave the formal definition of Transfer Learning in reference [3]. They first defined the concept of domain and task. The domain D consists of a feature space U and a marginal probability distribution $P(X)$ over the feature space, where $X = u_1, \dots, u_n \in U$, and D is expressed as $D = \{U, P(X)\}$. The task T consists of a label space Y and a conditional probability distribution $P(Y|X)$, which is learned from the training data consisting of pairs $u_i \in U$ and $y_n \in Y$. The formal definition is shown below [3]:

> "(Transfer Learning) Given a source domain D_S and learning task T_S, a target domain D_T and learning task T_T, transfer learning aims to help improve the learning of the target predictive function $f_T(\cdot)$ in D_T using the knowledge in D_S and T_S, where $D_S \neq D_T$, or $T_S \neq T_T$."

The motivation behind transfer learning is solving the lack of data and the vast training time of building a raw network. The size of a dataset for training a raw network to have promising performance in the general condition is enormous. Collecting and labeling vast amounts of data for supervised models can be complex, relative to the effort it needs to mark data points. Even if there is enough data, the time spent for a neural network to learn from that dataset is a lot because of the size of data and the randomly assigned initial parameters. Besides, most deep learning models have a spectacular performance on specific tasks but have a significant performance loss even when applied to a new similar task. The desire to leverage the knowledge from pre-trained models so that the neural network application can go beyond specific tasks and domains is also a motivation for transfer learning.

1.4.2 Transfer Learning Application

There are three major scenarios for applying transfer learning on convolution network (ConvNet) [4]: ConvNet as fixed feature extractor, fine-tuning the ConvNet, and Pretrained models.

ConvNet as a fixed feature extractor is taking a ConvNet pretrained on other datasets, replacing the last-fully connected layer, and fixing the parameter of the remaining structure as a fixed feature extractor. Therefore, when training this setup, the transfer learning only updates the last layer's parameter.

Fine-tuning the ConvNet also replaces the last-fully connected layer first. However, it does not treat the rest of the parameters as fixed values, which means that the parameters of the entire neural network get updated during the training process. The difference from training the raw neural network is that the transfer learning starts with the parameter not being entirely random. That is an advantage compared with training from the raw neural network.

Pretrained models usually use other released final ConvNet checkpoints for fine-tuning. Whether choosing fine-tuning or a fixed feature depends on the situation. There are four major scenarios where fine-tuning will be a good choice [4]:

> "1. *New dataset is small and similar to the original dataset*. Since the data is small, it is not good to fine-tune the ConvNet due to overfitting concerns. Since the data is similar to the original data, we expect higher-level features in the ConvNet to be relevant to this dataset as well. Hence, the best idea might be to train a linear classifier on the CNN codes.
> *2. New dataset is large and similar to the original dataset*. Since we have more data, we can have more confidence that we won't overfit if we were to try to fine-tune through the full network.
> 3. *New dataset is small but very different from the original dataset*. Since the data is small, it is likely best to only train a linear classifier. Since the dataset is very different, it might not be best to train the classifier form the top of the network, which contains more dataset-specific features. Instead, it might work better to train the SVM classifier from activations somewhere earlier in the network.
> 4. *New dataset is large and very different from the original dataset*. Since the dataset is very large, we may expect to afford to train a ConvNet from scratch. However, in practice it is very often still beneficial to initialize with weights from a pretrained model. In this case, we would have enough data and confidence to fine-tune through the entire network."

Chapter 2: The Robot System

2.1 Hardware and Software

Three hardware pieces are used to build this system: a UR5e robotic arm, a gripper, and a camera. This camera is a fixed focal length camera and mounts on the top of the gripper.

The software of this system relies on the ROS platform to connect each part. In addition, OpenCV is used for visual detection, and PyTorch is adopted to form a neural network for image classification. Fig. 2.1 shows the program structure.

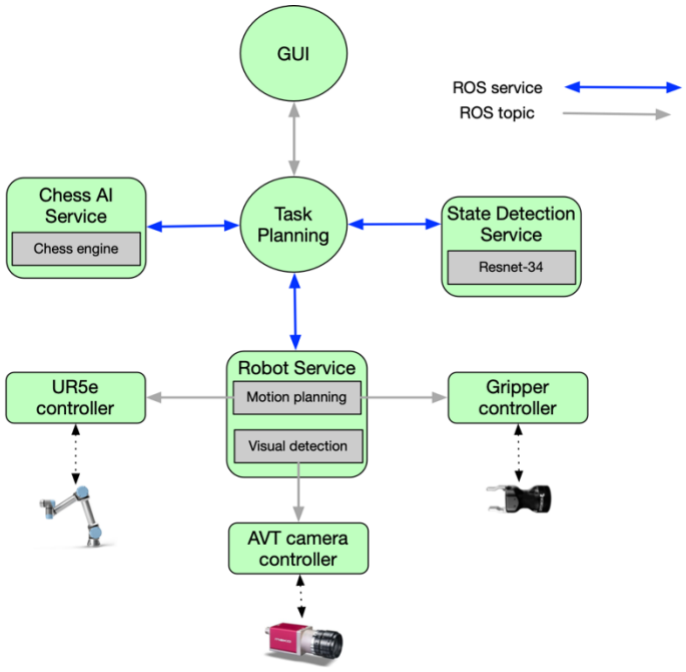

Figure 2.1 System's software structure map

The user interacts with the system through the GUI node. The GUI node connects to the task planning. The task planning subscribes the message from the GUI node and executes the corresponding function. Three ROS services link to the task planning: Chess AI Service, Chessboard State detection service, and robot service. The chess AI service generates a chess move. The state detection service loads the neural network to detect the chessboard state. The robot service has two parts: motion planning and visual detection. The optical detection here estimates

the chessboard pose relative to the robot base. The motion planning calculates the waypoints for the robot arm to reach and sends the message to open or close the gripper.

2.2System Workflow

The robot first moves to the standby position and takes an image of an empty chessboard.

Figure 2.2 Robot at the standby position

The system detects the chessboard and estimates its pose to the robot base. Then, with the pose information, the system calculates the inspection position at which the camera can have a closer view, and the chessboard can fit into the picture. The first step of this calculation process is to define four pixels that the chessboard's corners need to align with. Secondly, the system sets the chessboard's center as the origin and recalculates the chessboard corner's coordinate value. The idea of calculating the camera's height away from the chessboard is like equation 1.1. The point coordinate and corresponding pixel value are given, so the height could be obtained using the similar triangular theory. Finally, the system has the desired camera's position, multiplies with the transformation matrix obtained by the pose estimation, and gets the inspection position to the robot base. Then the robot moves to that position, and the system does the estimation again, and the robot moves back to the standby position.

Figure 2.3 Robot at the inspection position

The system also detects each square's corner information for the upcoming chessboard state detection. Finally, a chess piece can be placed on the chessboard after preparations, and the human player can make the first move.

After the human player moves, the robot will move back to the top of the chessboard and detect the chessboard state. The system prints the detection result on the GUI window. A human performs the final verification to ensure the detection is a hundred percent correct and then sends it to the chess engine to search for the best move. According to the command from the chess engine, the system plans the action that the robot needs to do and sends that series of steps to the robot motion driver to execute.

Figure 2.4 Robot doing an action

2.3 Chess AI Service

An external chess engine searches for a move. We use a python package called python-chess to load the engine. This chess engine is called stockfish 11 [5], configured to the most challenging model. The default output of the engine only contains four characters, for example, e2e4, in which e2 is the start square, and e4 is the end square of the move. However, this message is oversimplified. The move could be just moving a piece or capturing a piece or other actions. These different moves led to further action planning. Therefore, to output a more informative message, we use functions in the python-chess library to check what type of action this move is. The system then arranges the message and sends it to the action section for further processing. One example message is "e2e4,no,no": the first "no" means this move is not capturing a piece, and the second means this move is not castling.

2.4 Robot Action Planning and Execution

The proposed action section has the planning part and execution part. The action section first decodes the message. Then, it checks which action is this move and then calculates the start

square and end square positions from the information provided by pose estimation. Next, the system checks which piece needs to be picked up via the chessboard state then calculates the pickup position. Since some chess piece moves in a particular pattern, during the planning process, the system needs to know how much this piece needs to be raised to not hit other pieces during the action. Next, the planning program calculates key trajectory waypoints and constructs a list of activities that include waypoints and gripper action. Then, it sends to the robot execution part a package for robot motion planning and controlling. My mentor Hui Xiao developed this package. It moves the robot from the current waypoint to the goal waypoint in a straight path.

2.5 Pose Estimation Workflow and Limitation

The pose estimation of the chessboard is a crucial step for this system. The first step is placing the chessboard on the workspace and inside the image frame and then taking a picture of the empty chessboard at the standby position. No chess piece is placed on the chessboard at this point. The system uses the OpenCV's built-in function, goodFeaturesToTrack, to extract the possible points in the image (Fig. 2.5).

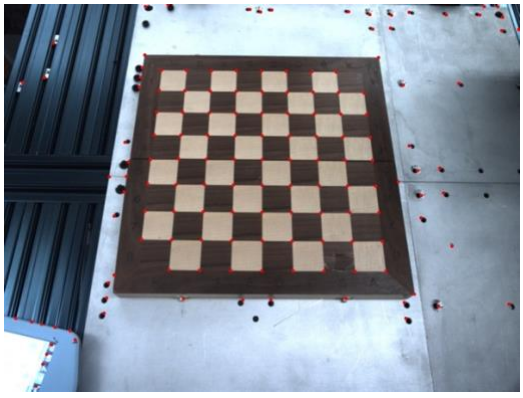

Figure 2.5 Extracted points in the image

Since the points on the chessboard have a unique pattern, the system takes each point as the center to crop a small square and performs the K-means clustering. The total number of clusters is set to two (Fig. 2.6), and after this process, this small image's color becomes only black and white.

Figure 2.6 K-mean clustering of a point's image

The system then checks a small segment of pixel value at each corner in this small image. If the corner has only one color and its diagonal counterpart also has the same color, the system identifies it as a point on the chessboard. Although this method works well to find points inside the chessboard most of the time, it does fail very few times when some points coincidently satisfy the criteria. The black box on the upper left of Fig. 2.7 shows an example of this noise. The solution is just moving that noise object out from the image frame physically. The system uses the points inside the chessboard to perform the pose estimation. However, lighting and overexposure would affect the result of extracting points from the image. Inside Fig. 2.7, The sizeable black box circles the area where extracted points are missing or not precisely alienate actual chessboard points.

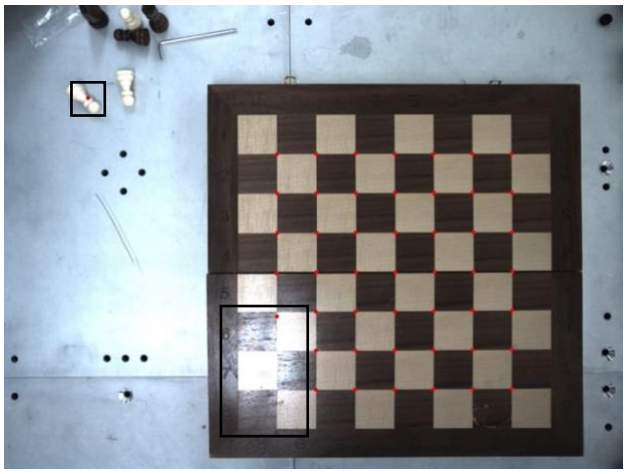

Figure 2.7 Unwanted point and missing points due to noise

Based on the observation, we found that the lighting noise only affects a particular area, and in that area, the points that are supposed to be detected are always missing. Hence, the system needs to select a set of feasible points. The system first finds a pair of points with the longest distance between them. This pair of points is the chessboard's vertices, and the points are always alien with actual chessboard points because of no lighting noise. With the vertices information, the system chooses the one with a lower histogram value than others. The histogram value indicates the image's brightness, where lower brightness is affected less by the overexposure noise. Then the system collects nine points of that corner, lists them in order, and sends them to OpenCV's built-in pose estimation function to solve the chessboard pose to the camera.

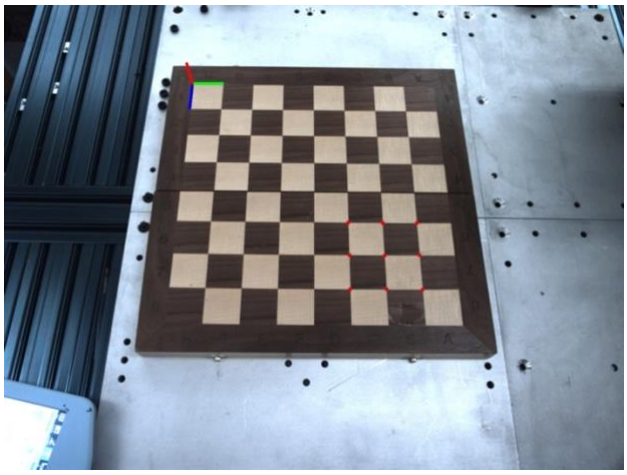

Figure 2.8 Pose estimation result

The advantage of this pose estimation feature is that the robot can adjust its motion without precisely placing the chessboard at pre-defined fixed locations. The disadvantage is that the pose estimation accuracy is not entirely satisfying using a 1.92-megapixel camera. For example, given a point's coordinate in the chessboard base, the final motion error of the robot could have 1 to 5 millimeters of error. Usually, the error in the z-direction is more significant. Also, sometimes the orientation of the coordinate system is not parallel to the chessboard surface. The first factor that can cause this inaccuracy is that the feature points extracted from the image are not perfectly

aligned. The second factor is that when the camera is far from the target, a pixel corresponds to a relatively large area in the real world, so even a tiny error in points extraction can cause non-negligible effects on the pose estimation. The current solution is modifying the z direction's translation entry in the pose estimation matrix to a measured value and rotating the orientation of the matrix so that the x-y surface is parallel to the horizontal surface.

2.6 Detect the Chessboard State

The deep learning method is adopted as a tool to detecting the chessboard state. The overall detection system needs to detect the pieces on the chessboard and locate each piece on the board.

2.6.1 Image Detection: YOLO Neural Network

This problem was first treated as image detection. The reason is that the image detection detects the object in the image and draws a bounding box around the object, satisfying the detecting the locating requirements. Then, we initially tried the Yolo neural network [6]. The highlight feature of this neural network is fast detection with relatively high accuracy. The neural network was trained with an online dataset provided by Roboflow. The dataset is called “Chess Piece Dataset” [7]. It is an augmented dataset. The image type of the data is JPG. The size of this augmented dataset is 693 images. The size of the original dataset that the augmented dataset originated from is 289 images. The augmented dataset splits the data into three parts: train, valid, and test. There are 606 images in the train, 58 images in the valid, and 29 images in the test. A detection result is presented below. From the image, it is obvious that the neural network still requires further adjustment to adapt to our system, but the outcome shows that it can achieve the task.

Figure 2.9 Yolo detection result

However, this method was no longer used in our system. The first reason was that, in this task, the priority was not fast detection but accuracy. Second, it was not straightforward to know which square had a piece by using the bounding box. Besides, it would be even more complicated when occlusion happened because the bounding box could be cover by another piece. The third was that the process of collecting the data is time-consuming. Therefore, the thesis modified the goal from image detection to image classification.

2.6.2 Image Classification: Resnet

There are many neural network models for image classification provided by PyTorch. Comparing each model's Top1 error, inference time on GPU, and model size (Fig. 2.10), Resnet-50, 34, and 18 were chosen as candidates for our task.

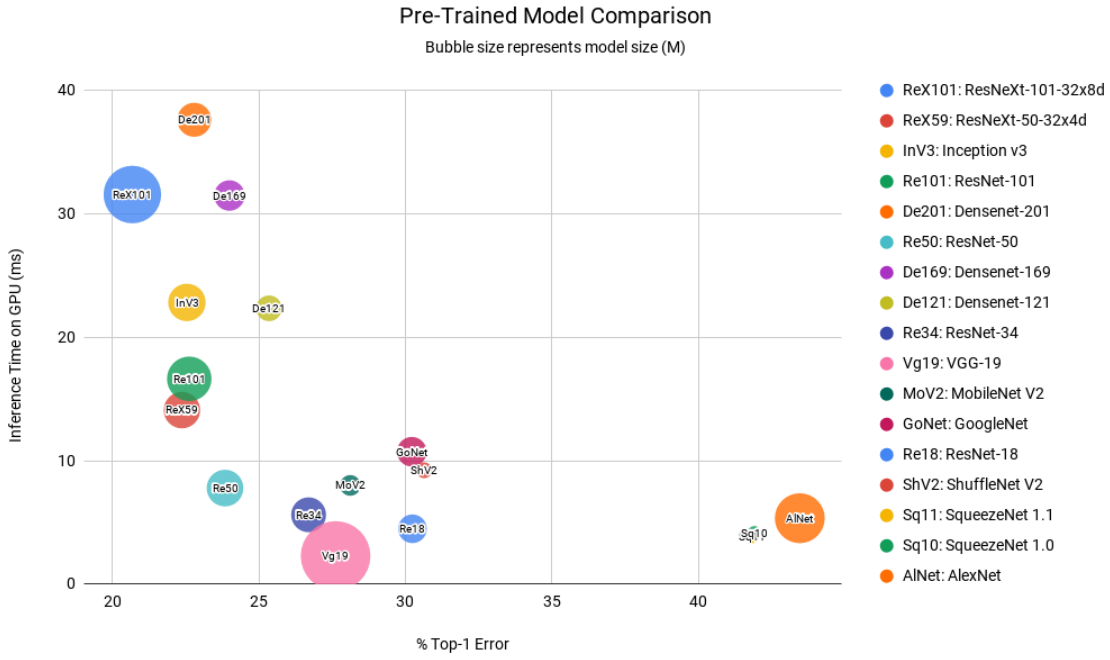

Figure 2.10 Plot of the pre-trained models [8]

For transfer learning, there are three scenarios: fix feature extractor, fine-tuning, and pre-trained model. The thesis chose to fine-tune the neural network since the system augmented the collected dataset to a relatively large size. Therefore, the dataset is somewhat similar to the dataset used for pre-trained data.

We augmented the collected 832 chessboard images to 3328 data frames and used the Adam optimizer to learn the parameter to prevent overfitting. The mechanism of this optimizer has the same effect as L2 regularization [9]. We then collected a test dataset under the same lighting condition as when collecting the training dataset.

We replaced the last fully connected layer with a linear layer and trained all three neural networks for 25 epochs. The performance is shown below in Table 2.1.

Dataset	Resnet-18	Resnet-34	Resnet-50
Validation	96%	93%	89%
Test	96%	90%	75%

Table 2.1 Performance of Resnet-20, Resnet-34, and Resnet-50 with a linear fully connected layer

The Resnet-18 had the best performance and followed by Resnet-34, and Resnet-50 was the worst. The explanation is that all these neural networks are trained with the ImageNet dataset with 200 classes. The difference of Top1 error between them is relatively not that large, so they all have sufficient ability to fit a 13-class dataset. On the other hand, Resnet-50 had a bad outcome because the time required for training was relatively more extended than the other, and since the training epoch was fixed, the training time for Resnet-50 was not enough.

For the second attempt, the system added a softmax layer after the linear layer. The softmax function is for multi-class classification. It assigns each class with a decimal probability, and all probability values add up to one. This feature helps the training to converge faster. The new layout was still trained and tested on the same training and testing dataset. The performance is shown below.

Dataset	Resnet-18	Resnet-34
Validation	96%	93%
Test	97%	88%

Table 2.2 Performance of Resnet-20 and Resnet-34 with a softmax layer

Adding the softmax did improve 1 percent for Resnet-18 during testing. However, the Resnet-34's test performance decreases by 5%. Overfitting could have caused this, so we added a drop-off layer to the current Resnet-34 layout and trained again. The performance is shown in Table 2.3.

Dataset	Resnet-34(drop off)
Validation	96%
Test	95%

Table 2.3 Performance of Resnet-34 with a drop-off layer and a softmax layer

The data shows that Resnet-18 with a softmax layer and Resent-34 with a drop-off softmax layer had an equivalent promising performance under the same lighting condition of the training dataset. A sample detection output is shown below.

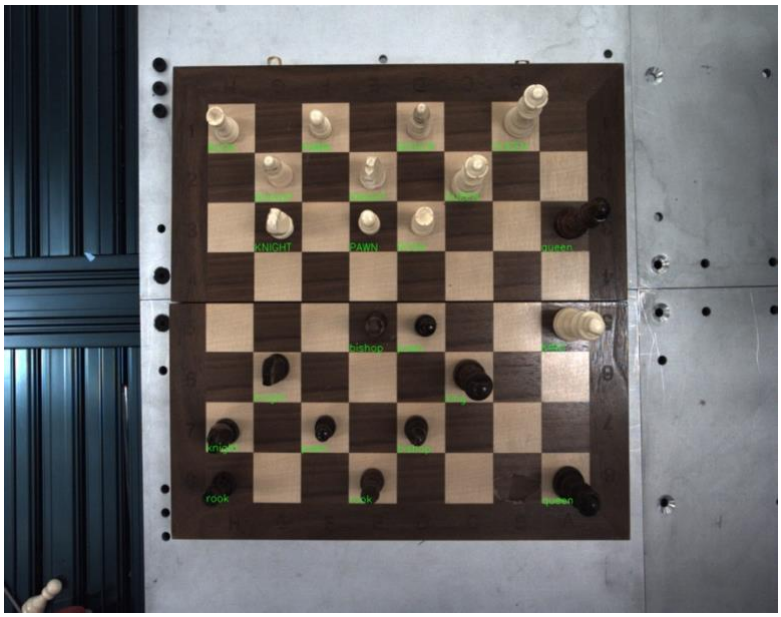

Figure 2.11 Resnet-18 detection result

Chapter 3: Lighting Variant Experiment

3.1Define the Goal of the Experiment

The previous training shows that Resnet-18 with a softmax layer and Resent-34 with a drop-off softmax layer had an equivalent promising performance under a lightning condition, which is the same as when collecting the training data. This thesis explores further the performance of these two models when lighting changes. There are three goals for this study. The first is to know how severe the lighting change can affect a trained neural network. The second is to identify

whether brightness variation during the data augmentation process could help the neural network's performance. The third goal compares the performance between the same neural network trained on data augmentation and trained on real collected light varying training dataset.

3.2 Experiment and Result

The first preparation was to collect the light varying test dataset. The test dataset had twenty images divided into four categories: insufficient light, light coming from the left side, light coming from the right side, and light coming from a random side. The images below show a chessboard state under these four different conditions.

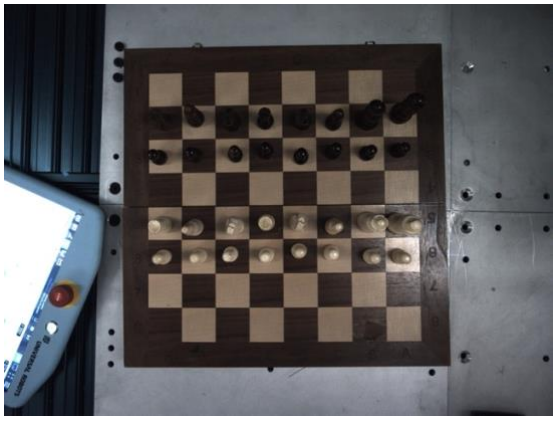

Figure 3.1 Insufficient light

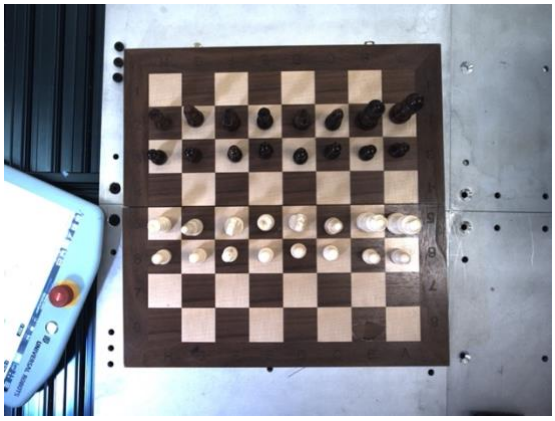

Figure 3.2 light coming from the right side

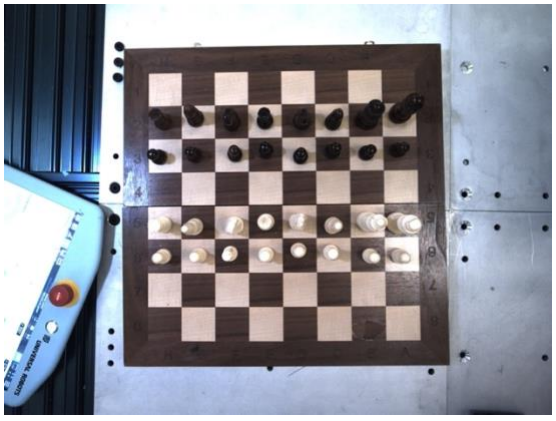

Figure 3.3 light coming from the left side

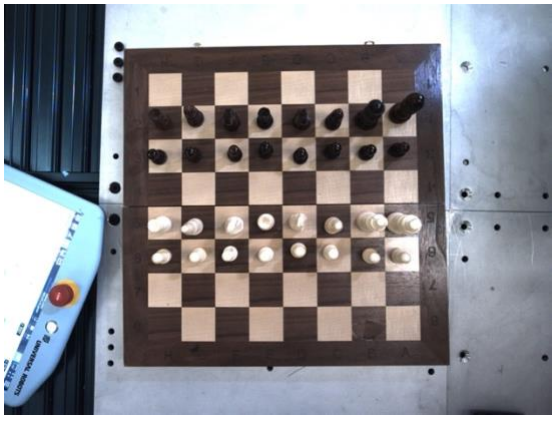

Figure 3.4 light coming from random side

This test dataset was named the "varied" dataset, and the previous test dataset with regular light conditions was called the "regular" dataset.

The previously trained Resnet-18 and Resnet-34 were tested on the light varied test data. The result is shown below.

Test data	Resnet-18	Resnet-34(drop off)
Varied light	68%	65%

Table 3.1 The performance of the trained Resnet-18 and Resnet-34 (drop off) on the varied dataset

The accuracy of the two neural networks had a dramatic decrease. Compared with testing on the "regular" dataset, the Resnet-18 decreased 29%, and Resnet-34 decreased 30%. Therefore, it was evident that the light condition of the environment has a significant effect on neural network detection.

The thesis explored how much data augmentation can help to improve performance. We used the Albumentation, a recommended data augmentation package, to augment the dataset, including the brightness variation. The total size of the new augmented dataset was still 3328 images. The brightness variation was achieved by applying the provided function, RandomBrightness. The possibility that an image would have a brightness variation was set to be 0.33. The performance of the neural network trained by this augmentation dataset is shown below.

Test data	Resnet-18	Resnet-34(drop off)
Regular	97%	94%
Varied	72%	62%

Table 3.2 The performance of the Resnet-18 and Resnet-34 (drop off) trained with an augmented dataset with brightness variation

The performance of Resnet-18 tested on the "regular" dataset remained the same and had a 5% increase on the "varied" data set. However, the Resnet-34 with drop-off layer performance was even worse. The reason could be the neural network could not fit the new augmented data.

Therefore, the drop-off layer was removed from the Resnet-34 and trained again. The performance is shown below.

Test data	Resnet-34
Regular light	97%
Varied light	73%

Table 3.3 The performance of the Resnet-34 trained with an augmented dataset with brightness variation

The accuracy of Resnet-34 tested on the "regular" dataset was bouncing back to 97%, and the performance on varied light reached 73%. Although the neural network's performance increased, the degree of improvement was limited.

Due to the limited performance of the neural network trained with the computer augmented dataset, collecting a new dataset with the same size as the augmented dataset was necessary. The data were collected under natural lighting variation. We collected additional 2496 images and combined them with previously collected 832 images. The total number of this new dataset was still 3328. In this dataset, a quarter of the dataset was contained under low-light condition. Another quarter was collected under more light applying to normal lighting condition, and the last quarter was collected under random lighting condition. Comparing the performance of the same neural network trained under different datasets can suggest whether the data augmentation does not positively affect performance or the lighting variation in the test dataset is too large to detect. The performance of the algorithm trained with this new dataset is shown below.

Test data	Resnet-18	Resnet-34
Regular	98%	98%
Varied	87%	92%

Table 3.4 The performance of the neural networks trained real collected dataset

The performance had improved. Both neural networks were enhanced by 1% for the "regular" dataset. The performance of Resnet-18 tested under varied light improved by 15% from the network trained under the computer augmented dataset. The performance of Resnet-34 improved even more, by 19%. The data clearly shows that the neural networks trained with a real collected dataset outperform those trained with a computer augmented dataset.

3.3 Discussion

Comparing the difference between data in the two-dataset led to an explanation for this outcome. For example, here are three different images of the same square.

Figure 3.5 Regular light

Figure 3.6 Augmented

Figure 3.7 Real lighting variation

The image obtained by brightness augmentation turns whiter than the regular lighting condition image. However, the image with real light intensity change had more effective variation. The color in the actual light variant picture is different from the regular shot, some areas of the piece are brighter, and some areas are dimmer.

This experiment suggests that the augmentation model of changing the image brightness cannot vividly simulate the natural light variation. Therefore, it cannot help the neural network to identify the lighting change. Data augmentation can improve the performance when the neural network only works for unchanged lighting conditions. It is recommended to control the lighting condition to cover any physical variations.

Chapter 4: Future Work

4.1 Detecting Chess Piece inside the Square

Currently, the system does not locate the exact position of the chess piece inside the square. This design limitation causes trouble when the chess piece is not at the center of the square. As mentioned in section 2.5, the pose estimation has certain errors, so the chess piece would sometimes not be placed at the center by the robotic arm. This error could accumulate after several times of pick and place, and finally, the gripper would hit the chess piece and fail to pick it up. If the system can detect the piece's location, it can adjust the error.

To accomplish this task, the system will need to detect the piece’s contour and pose. The system will need to construct the computer model of each chess piece and train the six dimensions pose estimation algorithm to achieve this task.

4.2 Light Variant Simulator for Data Augmentation

This thesis has discussed the shortage of currently adopted data augmentation in generating the image data for neural networks to learn the light variations. Suppose a simulator can compellingly create the effect of the light coming from a different direction. In that case, the neural network's performance trained with the computer augmented dataset may improve. Thus, searching or developing a data augmentation could be a future work to do.

BIBLIOGRAPHY

[1] Matuszek, Cynthia, et al. "Gambit: An autonomous chess-playing robotic system." 2011 IEEE International Conference on Robotics and Automation. IEEE, 2011.

[2] Robotics, Vision, and Control: Fundamental Algorithms in MATLAB®, by Peter I. Corke, Springer, 2017.

[3] Pan, Sinno Jialin, and Qiang Yang. "A survey on transfer learning." IEEE Transactions on knowledge and data engineering 22.10 (2009): 1345-1359.

[4] CS231n Convolutional Neural Networks for Visual Recognition, cs231n.github.io/transfer learning/.

[5] Official-Stockfish. “Official-Stockfish/Stockfish.” GitHub, 1 Sept. 2008, github.com/official stockfish/Stockfish/blob/master/Copying.txt.

[6] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[7] Roboflow. “Chess Pieces Object Detection Dataset.” Roboflow, 1 Apr. 2021, public.roboflow.com/object-detection/chess-full.

[8] Shrimali, Vishwesh. “Image Classification Using Pre-Trained Models in PyTorch: Learn OpenCV.” Learn OpenCV | OpenCV, PyTorch, Keras, Tensorflow Examples and Tutorials, 4 May 2021, learnopencv.com/pytorch-for-beginners-image-classification-using-pre-trained-models/.

[9] Loshchilov, Ilya, and Frank Hutter. "Decoupled weight decay regularization." arXiv preprint arXiv:1711.05101 (2017).