TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN
HỆ THỐNG THÔNG MINH

HỆ THỐNG A.I.
HỖ TRỢ ĐÁNH CỜ TƯỚNG

CÁN BỘ DOANH NGHIỆP HƯỚNG DẪN: Trần Anh Tú
Trần Đình Quý
Đặng Thanh Hào

GIẢNG VIÊN ĐỒNG HƯỚNG DẪN: TS. Huỳnh Hữu Hưng

STT NHÓM: **2** HỌ VÀ TÊN SINH VIÊN	LỚP HỌC PHẦN ĐỒ ÁN	ĐIỂM QUÁ TRÌNH (do GVHD & CBDN ghi)	ĐIỂM BẢO VỆ (do Hội đồng chấm ghi)
Ngô Văn Anh Tuấn	18N16B		
Phan Văn Ngọc Hiếu	18N16B		

ĐÀ NẴNG, 12/2020

TÓM TẮT ĐỒ ÁN

Đồ án này của chúng mình sẽ áp dụng Computer Vision và A.I. để hỗ trợ người dùng học hỏi, có tầm nhìn sâu hơn về một bộ môn nào đó, cụ thể, nhóm chúng mình đã chọn bộ môn Cờ Tướng (Chinese Chess), đặt ra vấn đề chính cần phải giải quyết cho đồ án là: Cung cấp một bức ảnh có chứa một bàn cờ tướng, hãy đề xuất những nước đi được cho là nước đi tốt. Từ vấn đề, chúng mình đề xuất một giải pháp để từ hình bàn cờ lấy ra được trạng thái bàn cờ đó và lưu trữ vào máy tính. Sau đó, từ trạng thái bàn cờ mà đưa ra những nước đi sẽ giúp người chơi có lợi thế. Như vậy, có thể tóm tắt các bước giải quyết của chúng ta thành hai mô hình: hệ thống phân lớp (classifier system) dùng để "hiểu" bàn cờ, và hệ thống đề xuất (recommender system). Phần cuối cùng, chúng mình sẽ trình bày về những kết quả và số liệu, đại khái là hệ thống đã có thể "hiểu" hình của một bàn cờ có độ chính xác cao, tuy nhiên chỉ trong điều kiện ánh sáng, quân cờ, góc chụp tiêu chuẩn. Hệ thống cũng có thể đề xuất khá tốt. Và cả hai yêu cầu được hệ thống giải quyết với thời gian phản hồi chấp nhận được.

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

CHƯƠNG I. GIỚI THIỆU

1. Nguồn cảm hứng

Cờ Tướng, cờ Vua, cờ Vây,… với bất kỳ loại cờ nào, chủ đề *chơi cờ tự động (automated chess)*, *robot đánh cờ*,… là một chủ đề tạo được sự thú vị cho rất nhiều người hay nhiều nhóm đam mê thực hiện với mục đích học tập, nghiên cứu. Rất nhiều các project cùng chủ đề này rất phong phú về cách tiếp cận, ví dụ như:

- Sử dụng mặt bàn và đế quân cờ là nam châm, di chuyển quân cờ bằng từ tính.
- Sử dụng một cánh tay robot (SCARA, Cartesian coordinate robot,…) để di chuyển quân cờ.
- Cũng sử dụng từ tính, theo dõi trạng thái quân cờ. Tuy nhiên cách này chỉ biết có tồn tại quân cờ tại vị trí hay không, chứ không biết là quân gì.
- Sử dụng một camera phía trên theo dõi bàn cờ. Lúc này, có thể lựa chọn theo dõi có hay không, hoặc nhận dạng quân cờ đó là quân cờ gì.

Trong tất cả các project mà nhóm đã tham khảo, có lẽ truyền cảm hứng nhiều nhất chính là project Raspberry Turk – một cánh tay robot đánh cờ Vua, sử dụng xử lý đồ họa và A.I. để đọc bàn cờ và một Engine A.I. cờ Vua làm backend.

2. Yêu cầu của đồ án

Vấn đề cốt lõi của đồ án là: "Giả thuyết cho một bức hình có chứa bàn cờ Tướng (chinese chess), hãy đề xuất các nước đi mà sẽ tối đa hóa xác suất chiến thắng". Về mặt cơ bản, đây cũng chính là vấn đề cốt lõi của tất cả đồ án khác mà liên quan tới automated chess đều phải giải quyết. Cụ thể hơn, có hai điều mà hệ thống cần thực hiện tốt:

- Nhận diện chính xác trạng thái bàn cờ có trong khung hình.
- Đề xuất các nước đi hợp lệ, hợp lý. Tối thiểu phải đi được các nước mà hiển nhiên sẽ có lợi, ví dụ như nước chiếu hết.

Ngoài ra, để thêm tính độc đáo, nhóm đã quyết định thêm thành phần Web Server cho đồ án. Là một Web server thì phải có thể phục vụ được cho nhiều người, và stream hình ảnh thông tin về ván cờ hiện tại. "Thông tin" này bao gồm:

- Khung hình trực tiếp từ camera

- Hình ảnh tiền xử lý của khung hình bàn cờ
- Hình vẽ đồ họa bàn cờ ảo – tượng trưng cho bàn cờ máy tính đang thấy.
- Các nước đi được đề xuất.

3. Giải pháp tổng quan

Về phần cứng, nhóm lựa chọn nền tảng để triển khai đồ án là máy tính nhúng Raspberry Pi, tiện thể sử dụng module Pi Camera như là camera chuyên dụng để quay hình bàn cờ. Chân đứng set-up cho đồ án có thể tùy cơ ứng biến, nhưng phải đảm bảo ánh sáng cho bàn cờ.

Về nhận diện bàn cờ, sau khi có được khung hình, ta sẽ xử lý đồ họa, cô lập được khu vực mà là bàn cờ trong khung hình đó. Sau đó, căng khu vực ra thành một hình chữ nhật đứng, ta có thể lấy được hình tại những vị trí của bàn cờ bằng cách cắt hình chữ nhật đứng đó thành nhiều ô li, theo một tỉ lệ định sẵn. Với mỗi ô li, nhóm sử dụng một model Tensorflow do nhóm tự tạo, tự train để phân lớp xem ô li đó là gì (không có gì, là quân tốt đen, là quân xe đỏ,…).

Về đề xuất nước đi, nhóm sử dụng một Engine AI cho cờ Tướng có sẵn có kích thước nhỏ gọn, nhẹ, có thời gian phản hồi nhanh.

Về web server, nhóm sử dụng module http.server có sẵn kèm theo khi cài ngôn ngữ lập trình Python 3.

4. Phân công công việc

Bảng 1. Bảng phân công công việc các thành viên

Công việc	**Người đảm nhiệm**
Xây dựng chân đứng cho project	Hiếu
Code tiền xử lý khung hình bàn cờ	Tuấn
Tạo và train model Tensorflow nhận diện quân cờ	Tuấn
Sinh và phân loại thủ công dataset để train model phân lớp quân cờ	Hiếu
Code vẽ đồ họa bàn cờ ảo	Hiếu
Code front-end cho web server	Hiếu
Code back-end cho web server	Tuấn
Trích dẫn và code hàm gọi engine đề xuất nước cờ	Tuấn

CHƯƠNG II. GIẢI PHÁP

1. LINH KIỆN CHÍNH CỦA HỆ THỐNG

Dưới đây là danh sách các linh kiện được sử dụng:

- **Máy tính nhúng Raspberry Pi 3 Model B**

  Nhóm quyết định lựa chọn này là vì tính nhỏ gọn nhưng đa năng của Raspberry Pi. Pi của nhóm được gắn 1 heatsink lên CPU với 1 quạt tản nhiệt 5V. Raspberry của chúng ta sẽ thực hiện nhiều tác vụ nặng (xử lý hình ảnh, server,…) nên việc lắp hệ thống tản nhiệt là cần thiết.

Hình 1. Hệ thống tản nhiệt cho Raspberry Pi 3B

- **Module camera cho Raspberry Pi 3B**

  Cụ thể, nhóm sử dụng **Camera Module v1** cùng với dây cáp dài. Đây là camera 5 Megapixels, có độ phân giải lên đến **2592 × 1944**, sử dụng cổng CSI 15-pins có trên Raspberry Pi 3.

- **LED thanh làm nguồn sáng cho bàn cờ**

  Nhóm sử dụng ba đoạn LED thanh, nguồn 9V, dài khoảng 10cm, mắc nối tiếp với nhau để cho khi gắm 1 đầu thì cả 3 thanh đều sáng.

- **Một bộ bàn cờ tướng nhựa**

  Bàn cờ tướng này có màu vàng, kích cỡ ~**35cm**×~**30cm**. Được đặt trên một tấm bạt, bảng hoặc giấy, có nền trắng.

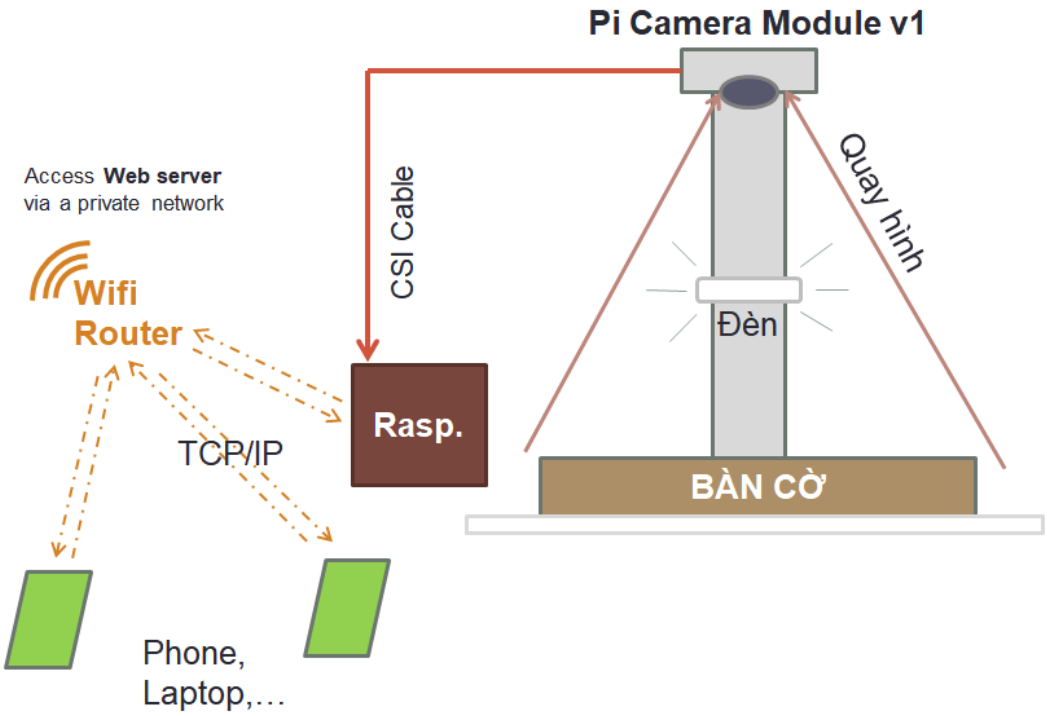
- **Một wifi router**

  Wifi Router được sử dụng để phát ra một mạng cục bộ để Raspberry có thể làm Web Server cho mạng đó. Ngoài ra, Raspberry Pi có thể được cấu hình để trở thành một Access Point, cho nên Router không thực sự cần thiết lắm.

Bảng 2. Chi phí tổng cộng các thành phần chính của đồ án

Thành phần	Chi phí
Raspberry Pi 3 Model B + Bộ nguồn	1.250.000 VNĐ
Heatsink + Quạt tản nhiệt cho CPU	21.000 VNĐ
Pi Camera Module v1	265.000 VNĐ
Pi Camera Extended CSI Cable	90.000 VNĐ
Bộ bàn cờ tướng nhỏ	40.000 VNĐ
LED thanh + Bộ nguồn	50.000 VNĐ
Tổng cộng	**1.716.000 VNĐ**

2. SƠ ĐỒ CỦA HỆ THỐNG



Hình 2. Sơ đồ các thành phần chính của đồ án.

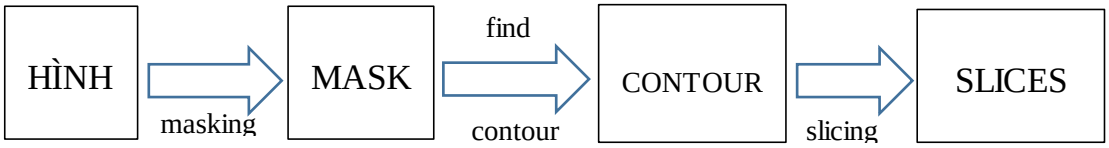
Hình ảnh của bàn cờ sẽ được thu hình thông qua Pi Camera và truyền đến Raspberry thông qua dây chuyên dụng. Mọi xử lý sẽ diễn ra tại Raspberry. Sau đó, với chức năng là một Web Server, stream các kết quả của xử lý đó lên web với giao thức HTTP, thông qua một mạng cục bộ riêng.

3. CÁC CÔNG ĐOẠN XỬ LÝ

1. Tiền xử lý - Trích các vị trí bàn cờ từ khung hình:

Để thực hiện công đoạn này, nhóm sử dụng thư viện OpenCV để giải quyết các xử lý đồ họa cần thiết. Một hàm thuộc thư viện này sẽ có dạng `cv.function()`.

Một khung hình trực tiếp từ camera sẽ có chứa bàn cờ, tuy nhiên để từ đó mà cô lập được khu vực chỉ chứa bàn cờ, và rồi chia nhỏ khu vực đó ra thành nhiều ô li mà mỗi ô li là một vị trí của bàn cờ là một quá trình không dễ.



Hình 3.Ba bước của công đoạn Tiền xử lý (Masking – Find contour – Slicing)

Công đoạn xử lý hình ảnh này gồm nhiều bước nhỏ:

a. Masking - Xác định khu vực trong bức hình là khu vực bàn cờ

Bởi vì màu bàn cờ khác với màu nền, nhóm đã tận dụng điều này để xác định khu vực của bàn cờ sử dụng phương pháp đặt ngưỡng màu. Trước khi bắt đầu, khung hình sẽ được chuyển sang không gian màu HSV. Bởi vì ở không gian màu RGB, một màu được xác định bởi ba kênh, điều này gây khó khăn trong việc tách một khu vực màu ra rất nhiều. Với không gian màu HSV, việc tách biệt một mảng màu dễ hơn rất nhiều.

Các bước thực hiện:

- Đầu tiên, làm mờ hình ảnh. Điều này giúp giảm thiểu các điểm ảnh bị nhiễu.

```
blurred = cv2.blur(image, (5, 5))
# làm mờ hình ảnh với kernel 5x5
```

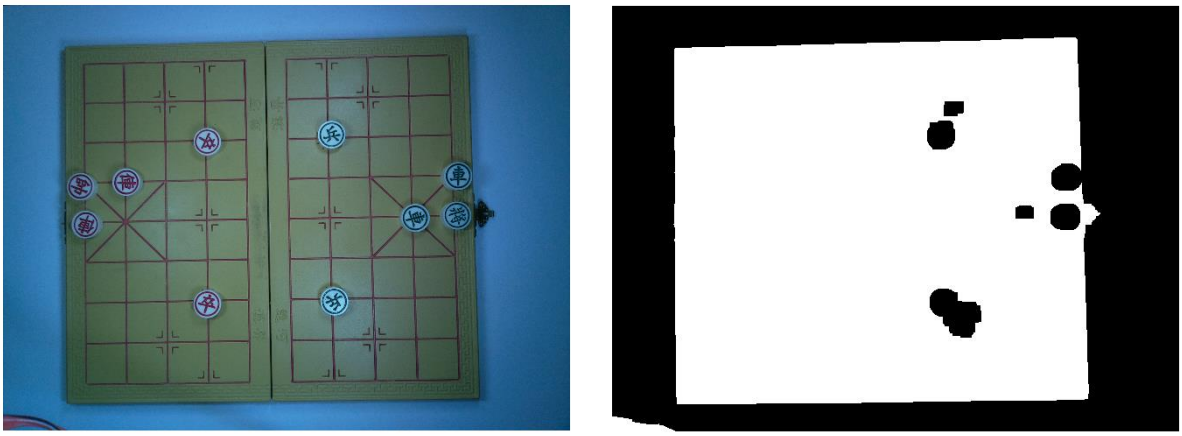
- Chuyển hình ảnh về không gian màu HSV

```
hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
```

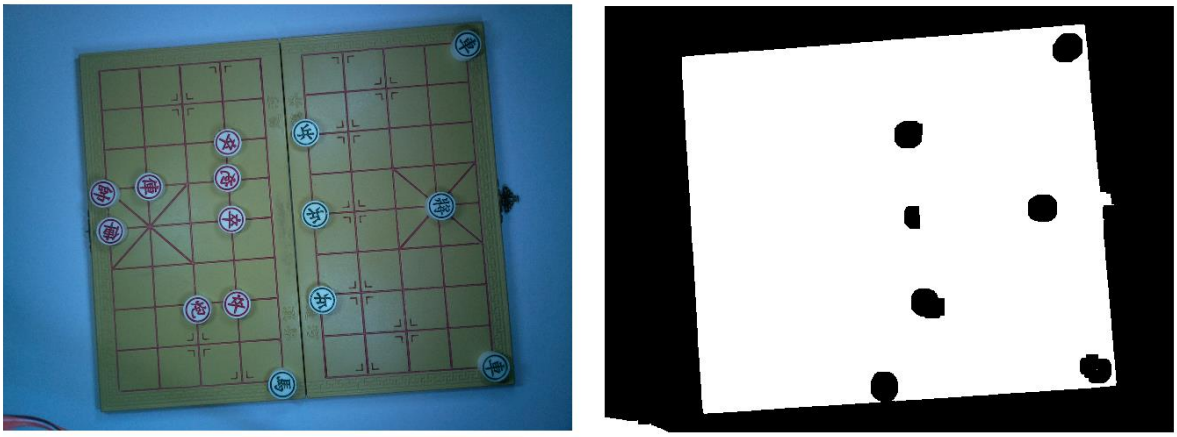
- Sau đó, sử dụng ngưỡng thấp, ngưỡng cao và hàm `cv.inRange()` để tách màu:

```
LOW_HUE, LOW_SAT, LOW_VAL = 91, 0, 0
UPPER_HUE, UPPER_SAT, UPPER_VAL = 180, 255, 255
LOW_HSV = [LOW_HUE, LOW_SAT, LOW_VAL]
UPPER_HSV = [UPPER_HUE, UPPER_SAT, UPPER_VAL]
mask = cv2.inRange(hsv, lower_yellow, upper_yellow)
```

Các giá trị hằng số được dùng để làm ngưỡng phải phụ thuộc vào điều kiện ánh sáng và màu sắc của vật thể và nền, cho nên có thể các thông số này sẽ cho kết quả sai đối với môi trường khác. Ứng với mỗi ngữ cảnh, môi trường, nên thử nghiệm các hằng số khác để có kết quả tốt.



Hình 4. Ví dụ 1 cho bước Masking



Hình 5. Ví dụ 2 cho bước Masking

b. Find Contour - Xác định tọa độ các góc của bàn cờ

Với Mask chính là khu vực có chứa bàn cờ, bước tiếp theo của nhóm là sẽ tìm ra tọa độ các góc của bàn cờ. Điều này có thể thực hiện được bằng cách xác định đường bao (contour) của mask, sau đó xấp xỉ (approximate) đường bao đó thành một tứ giác 4 cạnh.

Các bước thực hiện:

- Tìm các đường bao có trong mask:

Hàm `cv.findContours()` có chức năng tìm tất cả đường bao có trong một bức hình. Một bức hình có thể chứa nhiều mảng màu khác nhau, vì vậy cũng có thể chứa nhiều đường bao khác nhau.

```
contours, hierarchy = cv.findCountours(mask)
```

Ngoài đường bao, hàm còn trả về `hierarchy`, mô tả mối quan hệ của các đường bao, ví dụ như quan hệ lồng nhau,… Chúng ta có thể bỏ qua kết quả trả về `hierarchy` vì nó không thực sự giúp ích cho bài toán.

- Tìm đường bao thật sự của bàn cờ:

Vì gồm nhiều đường bao, nên chúng ta phải thực hiện một lựa chọn đường bao mà sẽ là đường bao của bàn cờ. Nhóm sử dụng tiêu chí lựa chọn tham lam là chọn đường bao có chu vi lớn nhất, tuy nhiên nhóm cũng cho rằng tiêu chí diện tích lớn nhất cũng sẽ cho kết quả như mong muốn.

Chu vi của một đường bao contour có thể được tính bằng:

```
cv.arcLength(contour)
```

Diện tích của một đường bao contour có thể được tính bằng:

```
cv.contourArea(contour)
```

- Approximate đường bao thành một tứ giác:

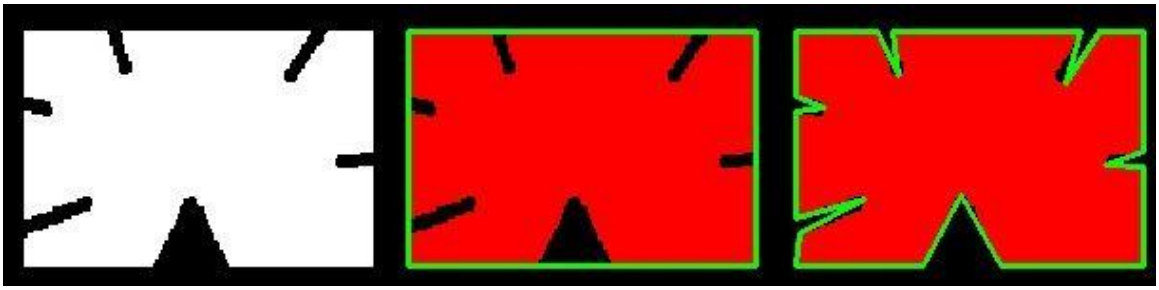
Đường bao cũng là một đa giác nhiều đỉnh. Bước này sẽ tối giản đa giác này về ít đỉnh hơn, mà cụ thể là 4 đỉnh. Để thực hiện, nhóm khai báo một tham số `epsilon` và thực hiện tối giản sử dụng `cv.approxPolyDP()`:

```
epsilon = 0.1 * cv.arcLength(contour, True)
```

```
approx = cv.approxPolyDP(contour, epsilon, True)
```

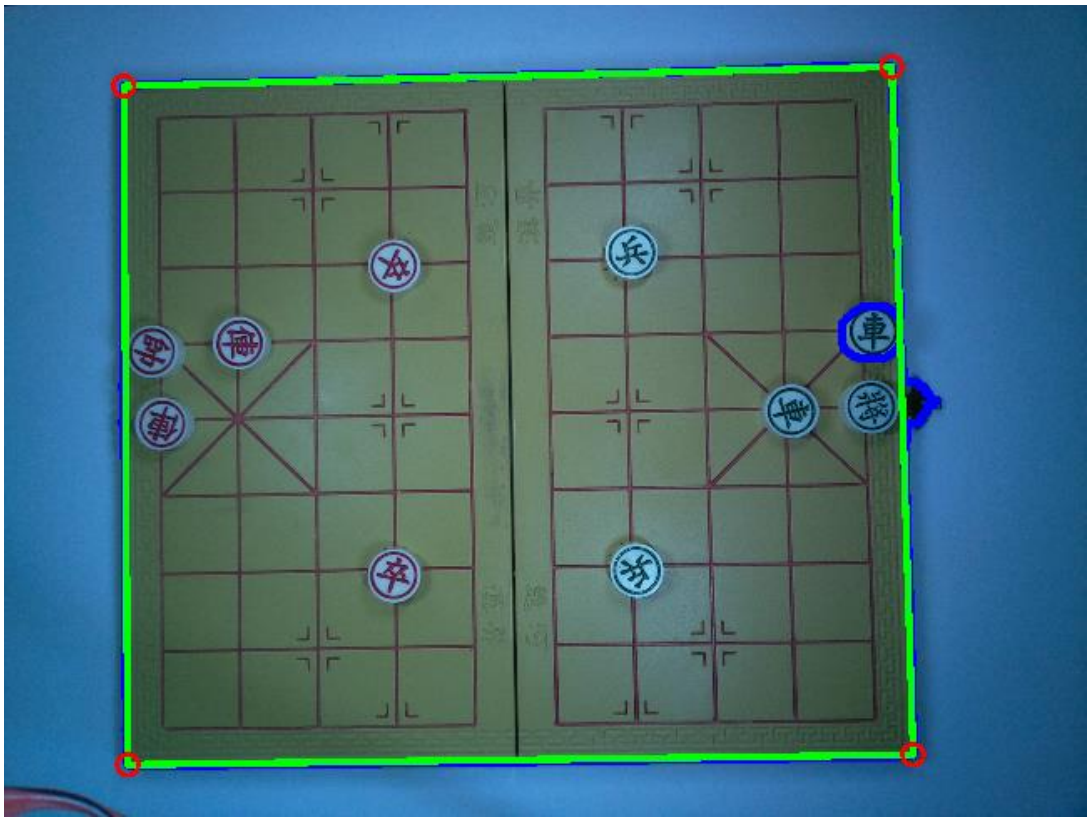
Ở đây, tham số `epsilon` được định nghĩa là khoảng cách lớn nhất từ `contour` đến `approx`. Có thể được hiểu như là một tham số chính xác. Tham số `True` ở trên hai hàm là tham số `Boolean` tượng trưng cho đường bao này có khép kín hay không.

Phía dưới mô tả kết quả của hàm `approxPolyDP()` với các `epsilon` khác nhau. Đường viền màu xanh chính là approx. Với hình ở giữa, `epsilon` là 10% độ dài đường bao. Với hình ở bên phải, `epsilon` là 1% độ dài đường bao .



Hình 6. Ảnh hưởng của tham số Epsilon trong hàm cv.approxPolyDP()

Đường bao trong thư viện được định nghĩa bởi tọa độ của các đỉnh của đa giác đó. Nên từ kết quả approx ta có thể tìm được tọa độ các góc của bàn cờ.



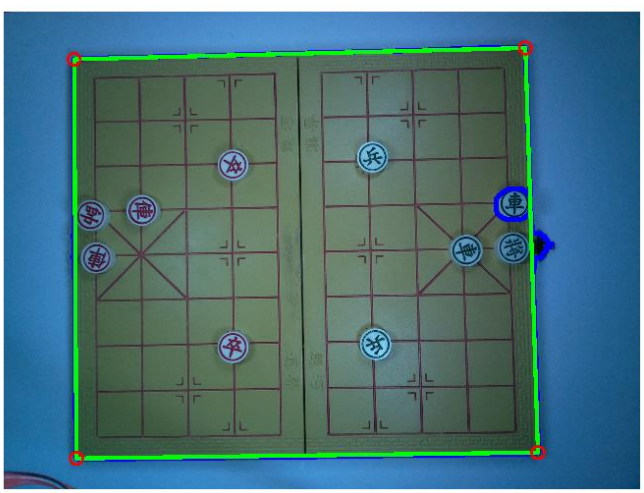
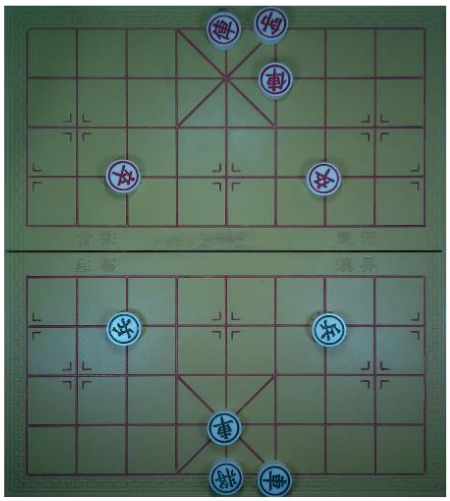
Hình 7. Ví dụ cho bước Find contour - các vòng tròn đỏ là các góc của bàn cờ được tìm thấy bởi chương trình.

c. Slicing - Trích các vị trí bàn cờ ra

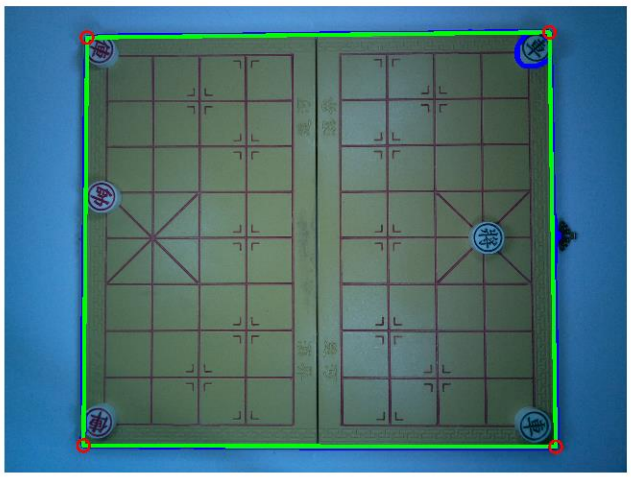
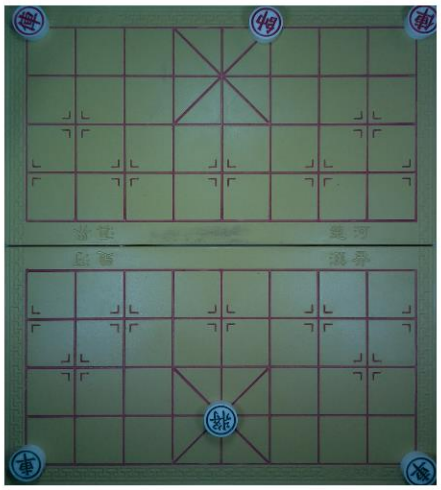
Sau khi xác định được bốn góc của bàn cờ, chúng ta sẽ sử dụng `cv.warpPerspective()` để căng bàn cờ về dạng một hình chữ nhật đứng. Để rồi từ đó, ta cắt bàn cờ thành 10 đường ngang và 9 đường dọc cách đều nhau, tạo nên 90 ô li mà mỗi ô chính là một vị trí của bàn cờ.

Công đoạn này gồm hai bước:

- Từ 4 góc bàn cờ, căng bàn cờ về dạng hình chữ nhật đứng:

Hình 8. Ví dụ 1 cho warping - bước Slicing

Hình 9. Ví dụ 2 cho warping - bước Slicing

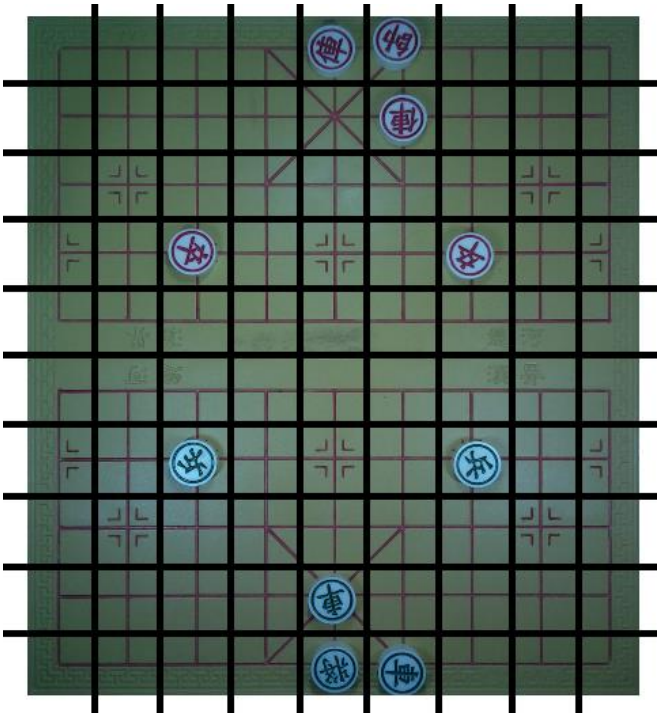
Bản cờ đích luôn có kích cỡ theo tỉ lệ 10:9.

```
topdown = [[0,0],  [0, 500], [450, 500], [450, 0]]
# với 450, 500 là kích cỡ của bàn cờ, tỉ lệ 9:10
```

Định nghĩa topdown là 4 góc của bức hình mới mà chúng ta muốn warp thành. Chú ý rằng thứ tự các đỉnh được mô tả trong Contour và trong Topdown phải tương ứng nhau, nghĩa là góc trái trên ứng với góc trái trên, và các điểm liệt kê theo một chiều nhất định. Vì vậy, nhóm đã sort lại các điểm của Contour theo tiêu chí: các điểm được liệt kê theo ngược chiều đồng hồ, và điểm đầu tiên với điểm thứ hai tạo thành cạnh ngắn nhất.

```
M = cv.getPerspectiveTransform(contour, topdown)
warped = cv.warpPerspective(image, M, (450, 500))
```

`warped` bây giờ chính là hình ảnh bàn cờ đã được căng chỉnh. Ta sẽ cắt nó dọc và ngang theo tỉ lệ, để trích ra được 90 vị trí của bàn cờ.



Hình 10. Mô tả vị trí đường cắt - bước Slicing

2. Model nhận diên quân cờ - phân lớp từng ô li của bàn cờ:

Yêu cầu của công đoạn này là phải cho biết được hình ảnh ô vị trí của bàn cờ là hình gì: Rỗng, Quân Tốt Đỏ, Quân Xe Đen… Mỗi quân cờ đều có một ký tự trên đó, và bộ cờ mà nhóm sở hữu có một đặc điểm đó là cùng một loại cờ, nếu khác màu sẽ khác ký tự:

Hình 11. Ví dụ cho bộ quân cờ tướng và ký tự của chúng

Dựa vào điều này, nhóm có thể chuyển bài toán nhận diện quân cờ thành bài toán nhận diện ký tự chữ, có thể được giải quyết bằng các mô hình Mạng thần kinh tích chập (Convoluted Neural Network). Cụ thể, cấu trúc model của nhóm được xây dựng bởi các lớp sau:

- Lớp tích chập (convolutional layer)

  Một lớp tích chập bao gồm ma trận lọc (filter) và ma trận đặc trưng (feature map). Filter chính là những "neuron" của lớp, có giá trị mỗi ô trong ma trận là trọng số cần được huấn luyện. Ma trận đặc trưng chính là output của một filter được áp dụng trên lớp ngay trước lớp này. Để thu được ma trận đặc trưng, filter sẽ được di chuyển từng pixel một trên lớp phía trước, tại mỗi vị trí sẽ thực hiện một phép toán tích chập để tính giá trị kích hoạt (activation) của ô tương ứng trong ma trận đặc trưng.

  Lớp tích chập có mục đích là trích ra được những đặc trưng của lớp trước. Lớp trước này có thể là hình ảnh gốc, hoặc là một lớp đặc trưng nữa.

- Pooling layer

  Lớp Pooling có tác dụng down-sample lớp phía trước nó. Nói nôm na là lớp này sẽ giảm thiểu sự chi tiết của lớp phía trước, mục đích làm chung chung dữ liệu vào, tránh hiện tượng model học thuộc dữ liệu khiến độ chính xác dự đoán thực tế bị giảm đi.

  Lớp cũng có một filter, nhưng thường kích cỡ nhỏ hơn nhiều và di chuyển trên lớp input có khoảng cách bằng với kích cỡ filter, để tránh hiện tượng overlap. Lớp này thông thường rất đơn giản, ví dụ như chỉ lấy trung bình hoặc là lấy giá trị tối đa của lớp input để tạo ma trận đặc trưng.

- Lớp kết nối dày đặc (fully-connected layer)

Lớp này là một lớp phẳng bình thường, có kết nối dày đặc. Thông thường chúng sẽ có một hàm kích hoạt không tuyến tính (non-linear) hoặc kích hoạt softmax nếu yêu cầu đầu ra là một xác suất.

Lớp này được sử dụng ở cuối mạng để hợp nhất các đặc trưng lại và đưa ra dự đoán.

Công đoạn tạo ra model:

a. **Thu thập dữ liệu để huấn luyện**

Bằng cách thực hiện bước **Tiền xử lý** nhiều lần với nhiều bàn cờ khác nhau, nhóm có được với mỗi cấu hình bàn cờ là 90 ô li. Sau đó, nhóm đã phân loại thủ công tập dữ liệu này thành 15 lớp: 7 lớp quân đỏ, 7 lớp quân đen và 1 lớp là rỗng, tổ chức theo từng thư mục mỗi lớp.

b. **Xây dựng và huấn luyện model**

Nhóm sử dụng thư viện Tensorflow (version 2.2.0 + GPU) kết hợp ngôn ngữ Python để xây dựng model. Model sử dụng liên tiếp các lớp tích chập và lớp pooling chồng lên nhau, với filter của lớp tích chập ở càng sâu thì càng to.

Đây là tổng quan về model của nhóm:

Bảng 3. Summary các lớp của model phân lớp quân cờ

STT	Loại layer	Filter size	Output shape	Param #
1	Input – (Rescaling)	No	(50, 50, 1)	0
2	Conv2D_1	32×32	(50, 50, 32)	320
3	MaxPooling2D_1	2×2	(25, 25, 32)	0
4	Conv2D_2	64×64	(25, 25, 64)	18496
5	MaxPooling2D_2	2×2	(12, 12, 64)	0
6	Conv2D_3	128×128	(12, 12, 128)	73856
7	MaxPooling2D_3	2×2	(6, 6, 128)	0
8	Conv2D_4	256×256	(6, 6, 256)	295168
9	MaxPooling2D_4	2×2	(3, 3, 256)	0
10	Dense	No	(15)	7695

Total params: 1,575,695

Model trên cho kết quả chính xác là rất cao chỉ trong 50 bước train từ lúc đầu. Với tập dữ liệu đã nói đến, model cuối cùng nhóm sử dụng được train

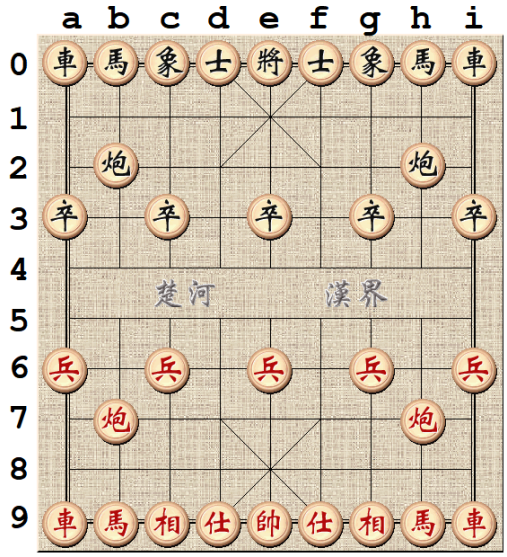
1000 bước, thời gian dành ra để train là xấp xỉ 12 tiếng, trên GPU Nvidia GTX 960M.

- Chuyển model từ Tensorflow thành Tensorflow-Lite:

  Model Tensorflow của chúng mình đã được thử chạy trên Raspberry cho kết quả như mong muốn, tuy nhiên thời gian phản hồi lâu. May thay, tồn tại một lựa chọn là chuyển đổi sang model Tensorflow-Lite (TF-Lite) mà không làm ảnh hưởng đến độ chính xác của model. TF-Lite là một phiên bản gọn nhẹ của thư viện Tensorflow được thiết kế để chạy trên các hệ thống nhúng. Việc chuyển đổi này đã giúp giảm thời gian phản hồi đi đáng kể, cho phép hoạt động trên thời gian thực.

3. Vẽ đồ họa bàn cờ tướng ảo

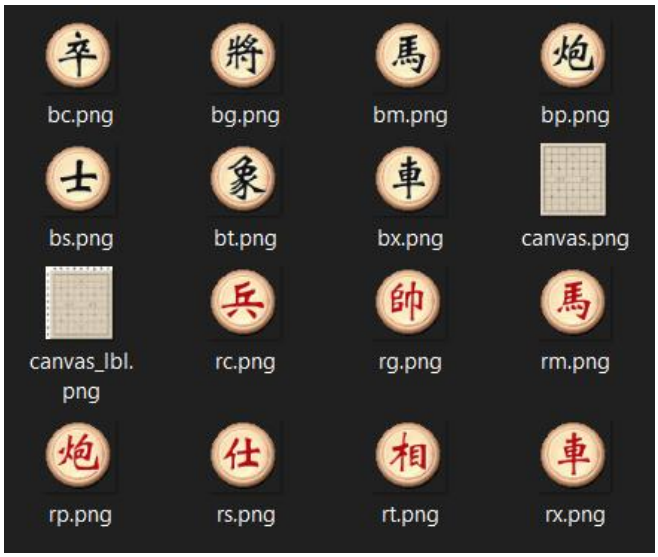
Nhiệm vụ của bước này là phải đọc vào một chuỗi ký tự với độ dài là 90, mỗi ký tự sẽ tượng trưng cho một vị trí của bàn cờ tướng, sau đó, trả về một bức ảnh là đồ họa của bàn cờ tướng được mô tả bởi chuỗi ký tự được cho.




```
xmtsgstmx
---------
-p-----p-
c-c-c-c-c
---------
---------
C-C-C-C-C
-P-----P-
---------
XMTSGSTMX
```

Hình 12. Ví dụ 1 cho bước vẽ bàn cờ ảo - Bên trái là bàn cờ đã vẽ, bên phải là chuỗi ký tự mô tả

Nhóm tìm và sử dụng một folder hình ảnh bao gồm: hình của bàn cờ, hình các quân cờ,… để làm texture cho đồ họa.



Hình 13. Texture sử dụng cho bước vẽ bàn cờ ảo

Ta có thể vẽ một hình A lên hình B bằng cách thay đổi các giá trị ở trong ma trận B thành ma trận A. Đoạn code sau đây được dùng để vẽ một quân cờ (piece) lên bàn cờ (canvas):

```
def placePiece(px, py, piece, canvas, mx=2, my=3, UNIT_SIZE=57):
    w, h, _ = piece.shape
    ox, oy = px * UNIT_SIZE, py * UNIT_SIZE
    for x in range(w):
        for y in range(h):
            if piece[x][y][3] != 0: # pixel is not transparent
                canvas[ox+x+mx][oy+y+my] = piece[x][y]
```

4. Đề xuất nước đi – Thuật toán tìm kiếm MTD(*f*)

Hệ thống sử dụng một Engine cờ Tướng làm Back-end. Engine này có tên là elephantfish, sử dụng thuật toán tìm kiếm MTD-f làm cốt lõi.

MTD (Memory-enhanced Test Driver) là một thuật toán tìm kiếm minimax, đơn giản và hiệu quả hơn nhiều thuật toán khác như NegaScout/PVS. Mã giả của MTD-f như sau:

```
int MTDF(node_type root, int f, int d) {
 int g = f;
 int upperbound := +INFINITY;
 int lowerbound := -INFINITY;
```

```
do {
      int beta;
      if (g == lowerbound) beta = g + 1;
      else beta = g;
      g = AlphaBetaWithMemory(root, beta - 1, beta, d);
      if (g < beta) upperbound = g;
      else lowerbound = g;
} while (lowerbound >= upperbound);
return g;
}
```

Thuật toán sẽ gọi `AlphaBetaWithMemory` một vài lần với khung tìm kiếm là `[beta-1, beta]` có kích cỡ bằng 0 (zero-size). Vì `AlphaBetaWithMemory` sẽ trả về một giá trị là biên của giá trị minimax thật sự, ban đầu biên của minimax sẽ là `lowerbound` và `upperbound`, mỗi vòng lặp cập nhật chặn dưới hoặc chặn trên, dần thu hẹp vùng giá trị của minimax lại, hội tụ đến giá trị minimax thực sự.

Hàm `AlphaBetaWithMemory` thực chất là thuật toán `AlphaBeta` thông thường, nhưng có sự dụng một bảng ghi nhớ (transposition table) để lưu lại những nhánh đã tìm kiếm. Hơn nữa, lời gọi `AlphaBeta` với khung tìm kiếm là `[beta, beta+1]` không đúng với mục đích ban đầu của `AlphaBeta` là khung càng rộng càng tốt (ví dụ như $[-\infty, +\infty]$),, đã làm cho thuật toán đa phần sẽ fail-low (giá trị trả về dưới beta) hoặc fail-high (giá trị trả về trên beta). Điều này cho phép ta có thể cập nhật các chặn của giá trị minimax như đã nói.

Ý tưởng của thuật toán có phần nào tương tự như thuật toán tìm kiếm nhị phân (binary search) khi cũng cập nhật chặn trên và chặn dưới để hội tụ đến một giá trị thật sự. Ưu điểm của sử dụng triển khai binary search là vì nó cho độ phức tạp thời gian chạy là:

$$\boldsymbol{O}(\log_2 |upperbound - lowerbound + 1| * C),$$

với C là chi phí lời gọi hàm AlphaBeta

Engine mà nhóm tham khảo đã triển khai MTD-f theo hướng này.

5. Web Server – HTTP.Server của Python

Nhóm sử dụng module HTTP Server có sẵn trong thư viện của Python để xây dựng nên Web Server. Trang web của nhóm sẽ bao gồm 2 trang lớn:

- Trang Home: cho biết thông tin của nhóm
- Trang Dashboard: tại đây, người dùng có thể chọn giữa các Tab, với mỗi Tab là một stream video. Có tổng cộng 4 tab: Camera, Masking, Contour và AI. Ba tab đầu sẽ stream nội dung như tên của chúng, còn tab AI sẽ gồm hai phần:
    - Phần Canvas: hiển thị đồ họa bàn cờ ảo. Bàn cờ ảo này sẽ tương ứng với những gì mà Raspberry đang thấy ở bàn cờ tướng thật.
    - Phần Suggestion: hiển thị các nước đi được đề xuất cho bàn cờ ở canvas.

Để triển khai Web Server, nhóm đã đặt ra hai điều cần chú ý sau:

c. **Chỉ nên có một thực thể video stream tồn tại:**

Một web server có thể một lúc phục vụ nhiều client, cùng với việc module HTTP Server của Python có hỗ trợ phân luồng để xử lý một request, cho nên có thể vô tình mà triển khai của nhóm tạo ra nhiều videos stream tuy nhiên lại stream một khung hình y đúc nhau. Điều này làm Raspberry chạy chậm lại và rất lãng phí.

Để giải quyết điều này, nhóm đã tạo một thực thể VideoProcessor toàn cục, và cho thực thể này chạy trên một luồng riêng biệt, liên tục ghi hình và xử lý chúng vào các thực thể stream public. Ngoài ra, nhóm còn tạo các phương thức lấy dữ liệu một cách an toàn với luồng (thread-safe), cho phép lấy khung hình hiện tại của một stream bất kỳ (Camera, Masking, Contour, AI)

d. **Nghẽn cổ chai bởi các công đoạn yêu cầu xử lý nặng:**

Trong tất cả các công đoạn, công đoạn AI là tiêu tốn nhiều tài nguyên nhất, khi phải đồng thời sử dụng model A.I. để hiểu bàn cờ, vẽ lại bàn cờ ảo, và chạy Engine đề xuất các nước đi tốt. Nếu thực thể VideoProcessor của nhóm chỉ chạy đơn luồng thì sẽ khiến các công đoạn nhẹ hơn phải chờ công đoạn này, cho nên nhóm đã tách làm 2 luồng xử lý: luồng chính sẽ làm việc nhẹ và tạo ra luồng phụ nếu chưa có, luồng phụ sẽ xử lý công đoạn A.I. và tự xóa bản thân nếu đã xong việc.

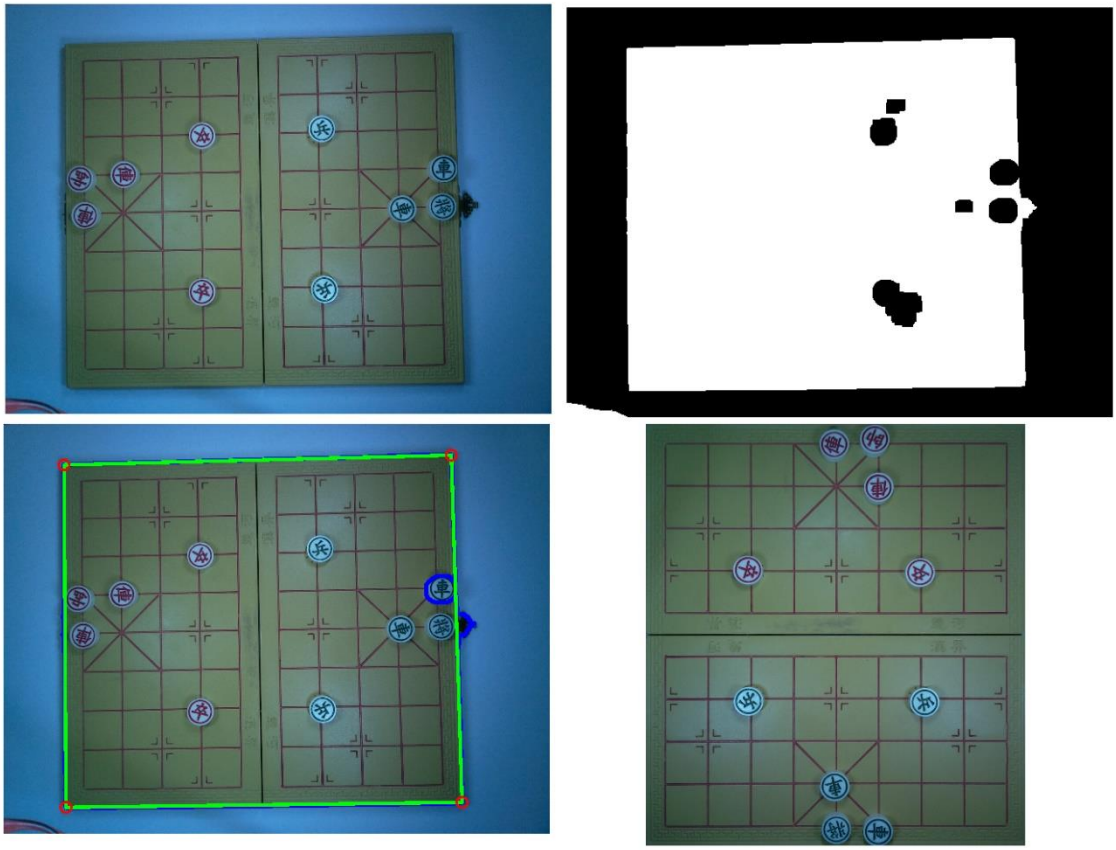
CHƯƠNG III. KẾT QUẢ

1. Tiền xử lý

Với các điều kiện như sau:

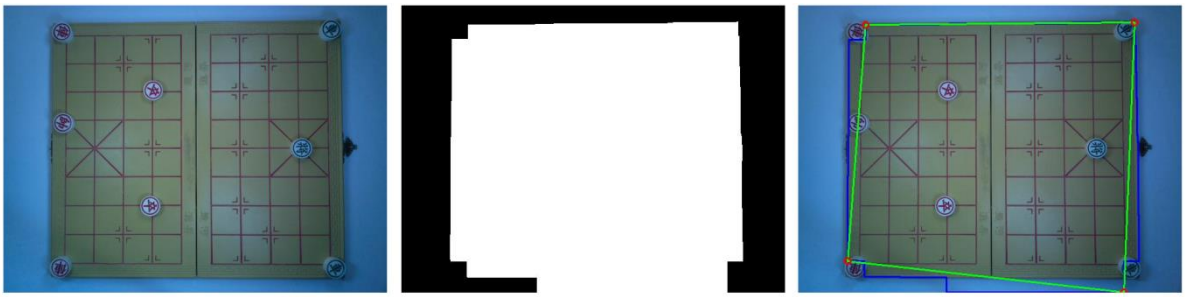
- Trong khung hình, khoảng cách từ biên bàn cờ đến biên khung hình phải lớn hơn xấp xỉ 2cm.
- Điều kiện ánh sáng đều trên toàn bàn cờ, không có chỗ tối chỗ sáng.
- Các góc bàn cờ phải luôn hiện diện.

Chương trình luôn xác định đúng các góc bàn cờ, khiến việc trích các ô li của bàn cờ cũng đúng như mong muốn.



Hình 14. Ví dụ hoàn chỉnh của công đoạn tiền xử lý

Trong một vài trường hợp, ví dụ như khi có một quân cờ ở sát một góc của bàn cờ sẽ khiến góc bàn cờ đó trở nên không nhận dạng được, khiến các bước tiếp theo cũng sai.



Hình 15.Một trường hợp mà bước tiền xử lý thất bại - khi một góc bàn cờ bị che khuất hoặc biên bàn cờ quá gần với biên của bức ảnh.

Ngoài các trường hợp biên như trên, nhóm chưa phát hiện được thêm những trường hợp khác mà quá trình tiền xử lý không tìm được 4 góc của bàn cờ.

2. Model nhận diên quân cờ - phân lớp ô li của bàn cờ:

a. Thu thập dữ liệu để huấn luyện

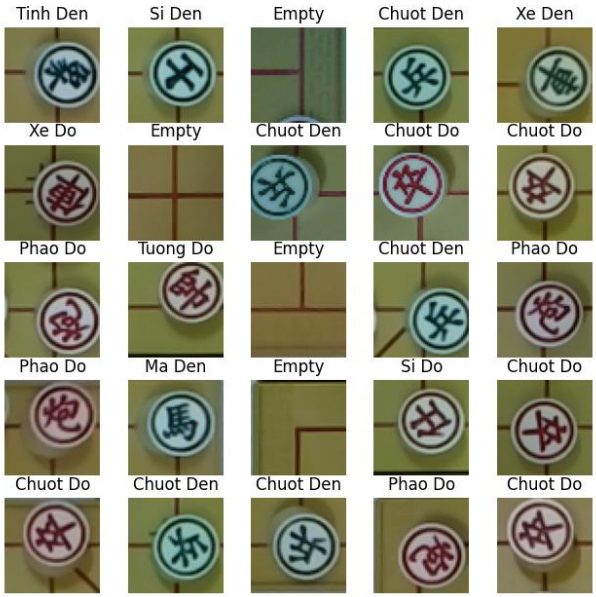
Tính đến thời điểm của bài báo cáo này, nhóm đã chụp và phân loại thủ công xấp xỉ 90 bàn cờ khác nhau. Sau khi trích một lượng nhỏ ra để làm tập đánh giá cuối cùng, tập dataset còn lại (train + validate) của nhóm gồm có:

Bảng 4. Số lượng mỗi class trong dataset

Class	Số lượng		
Tốt đen	424	Tốt đỏ	419
Pháo đen	170	Pháo đỏ	168
Xe đen	172	Xe đỏ	168
Mã đen	170	Mã đỏ	170
Tịnh đen	168	Tịnh đỏ	167
Sĩ đen	169	Sĩ đỏ	169
Tướng đen	86	Tướng đỏ	84
		Ô rỗng	542

Trong đó, có 3000+ ô là ô rỗng không được sử dụng. Ngoài ra, nhóm còn sinh thêm dữ liệu bằng hai phép biến đổi đơn giản: xoay mỗi sample đi 90 độ, 180 độ, 270 độ; thêm độ sáng ngẫu nhiên vào mỗi sample.

Tổng cộng, nhóm thu thập được **3246 sample**, với mỗi sample nhóm sinh được 40 sample nữa. Vậy kích cỡ tập dataset cuối cùng là **129840 sample**.



Hình 16. Ví dụ một vài mẫu trong tập dataset

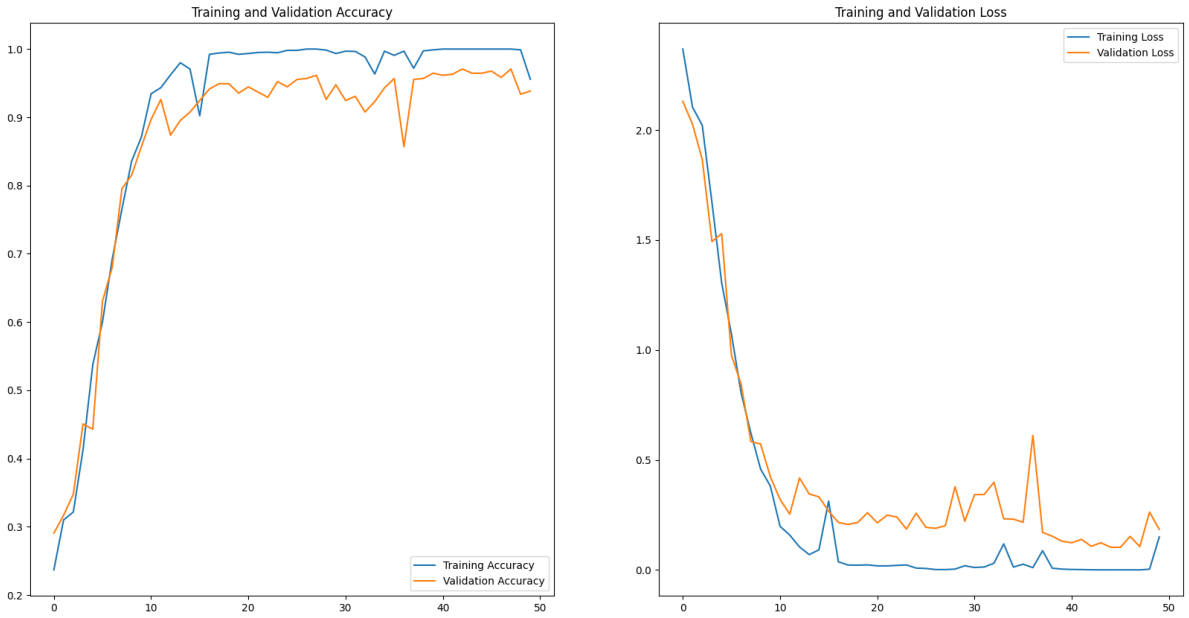
b. Xây dựng và huấn luyện model

Trên thực tế đây không phải là model đầu tiên của nhóm. Nhóm đã tạo ra 3 model trước model này và tất cả đều thất bại khi chỉ đạt được độ chính xác là ~60%. Model này là model thành công nhất của nhóm khi đạt được đến 99.99% tập dữ liệu validate và 100% tập đánh giá cuối cùng.

Nhóm giả thiết điều kiện để đảm bảo model luôn phân lớp đúng là:

- Quân cờ có ký tự không bị mất nét do nằm ở ngoài ô li, hay do có vật cản.
- Điều kiện ánh sáng không quá tối, không quá sáng.

Dưới đây là một vài hình ảnh về tính hiệu quả của model:



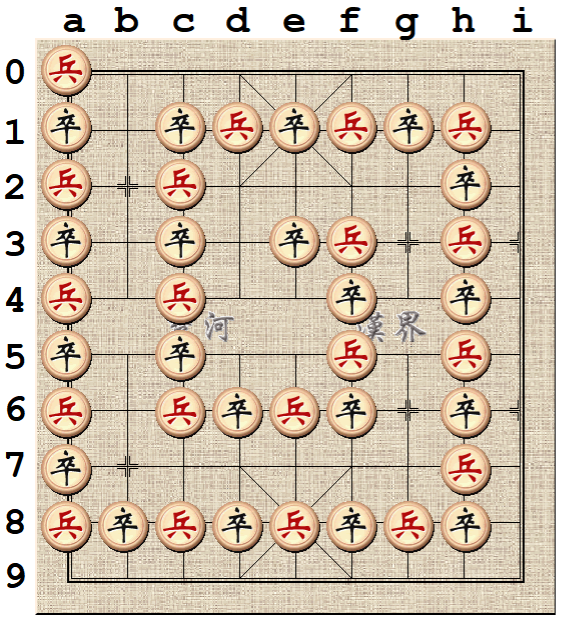
Hình 17. Plot độ chính xác (bên trái) và mất mát (bên phải) của model trong 50 bước huấn luyện đầu - Accuracy có xu hướng tăng và Loss có xu hướng giảm.

Hình 18. Dự đoán của model trên một phần của tập dataset đánh giá cuối cùng

3. Vẽ đồ họa bàn cờ tướng ảo

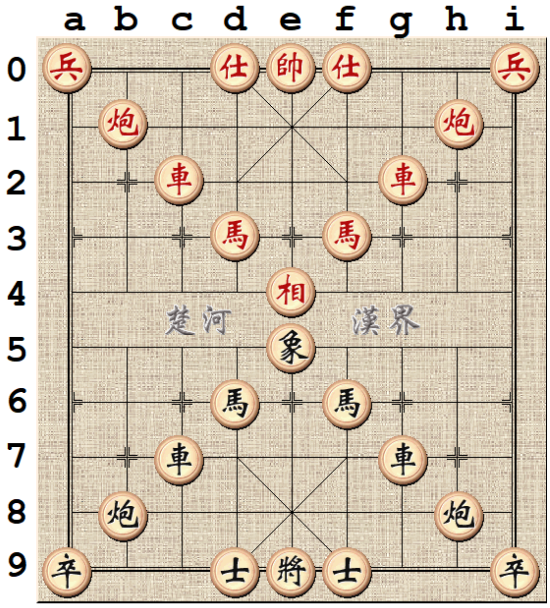
Đây là một công đoạn có thời gian xử lý khá lâu, khi ứng với mỗi quân cờ, nhóm gán từng pixel một. Nhóm đã có một phép tối ưu là lưu trữ những bàn cờ đã vẽ, tìm bàn cờ gần giống nhất rồi vẽ lên đó, giảm được thời gian chạy trung bình xuống.

Một vài hình ảnh ví dụ:




```
C--------
c-cCcCcC-
C-C----c-
c-c-cC-C-
C-C--c-c-
c-c--C-C-
C-CcCc-c-
c------C-
CcCcCcCc-
---------
```

Hình 19. Ví dụ 1 cho bước Vẽ bàn cờ ảo



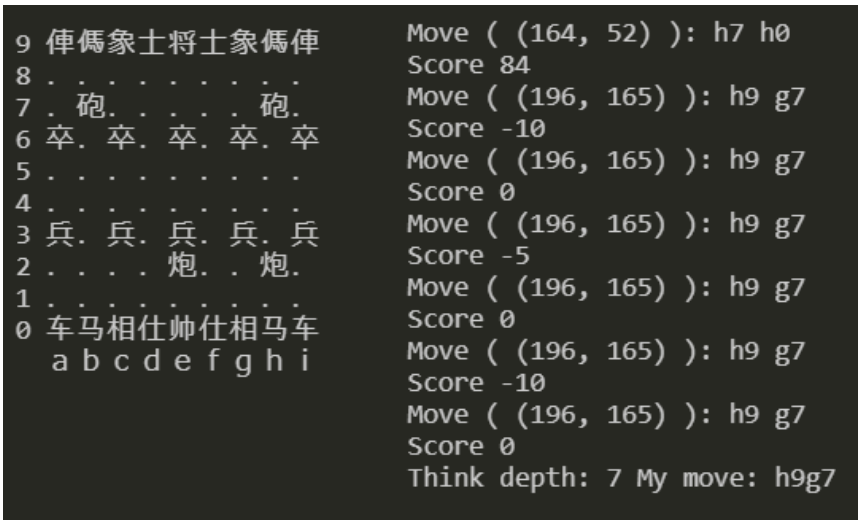

```
C--SGS--C
-P-----P-
--X---X--
---M-M---
----T----
----t----
---m-m---
--x---x--
-p-----p-
c--sgs--c
```

Hình 20. Ví dụ 2 cho bước Vẽ bàn cờ ảo

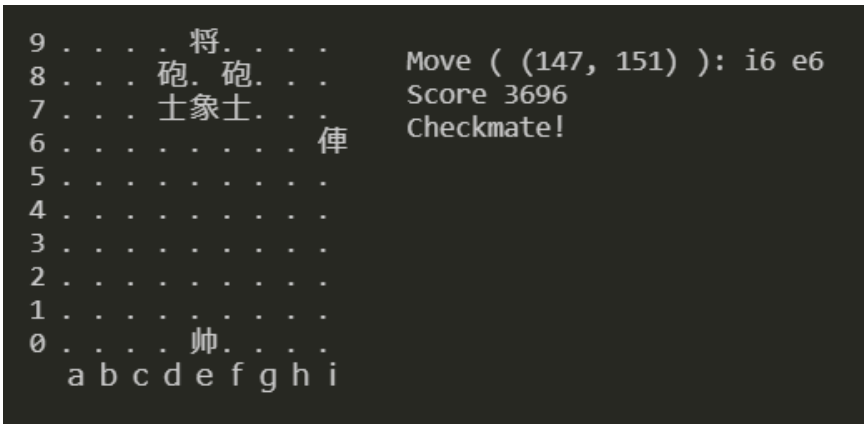
4. Đề xuất nước đi

Engine có thời gian phản hồi tốt, có lựa chọn tăng giảm thời gian suy nghĩ giúp engine tìm thấy những nước đi hay hơn. Chúng mình không có nhiều kinh nghiệm về thế trận cờ tướng cho nên không thể nhận xét chi tiết về các nước đi được đề xuất. Tuy nhiên, các nước đi hiển nhiên có lợi như: Chiếu hết, nước forking, nước đổi quân có giá trị hơn,… thì engine có khả năng tìm được. Trên lý thuyết, cho thời gian suy nghĩ càng nhiều thì engine sẽ càng có khả năng tìm được các nước tốt nhất.

Một vài hình ảnh hoạt động của Engine:



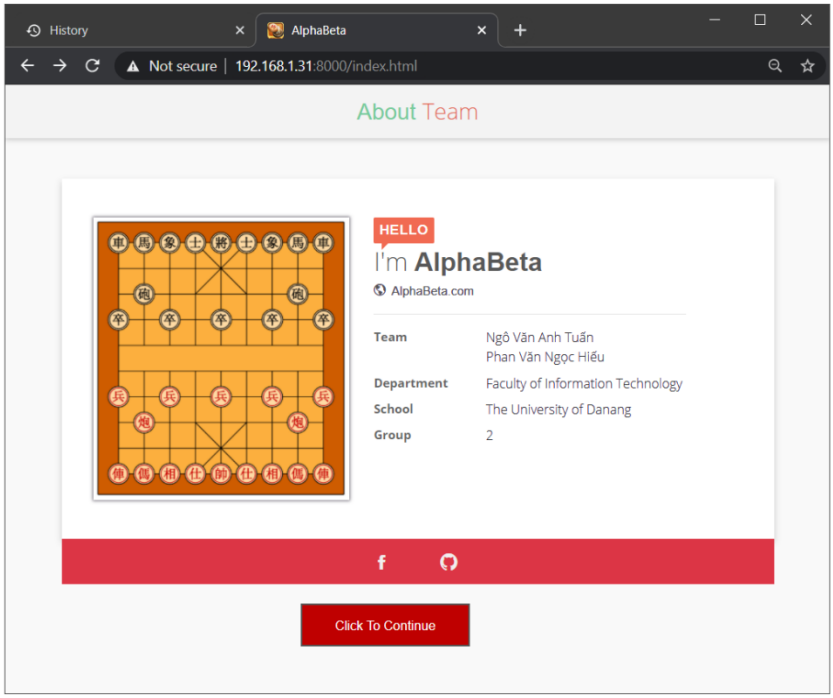
Hình 21. Ví dụ 1 về Đề xuất nước đi



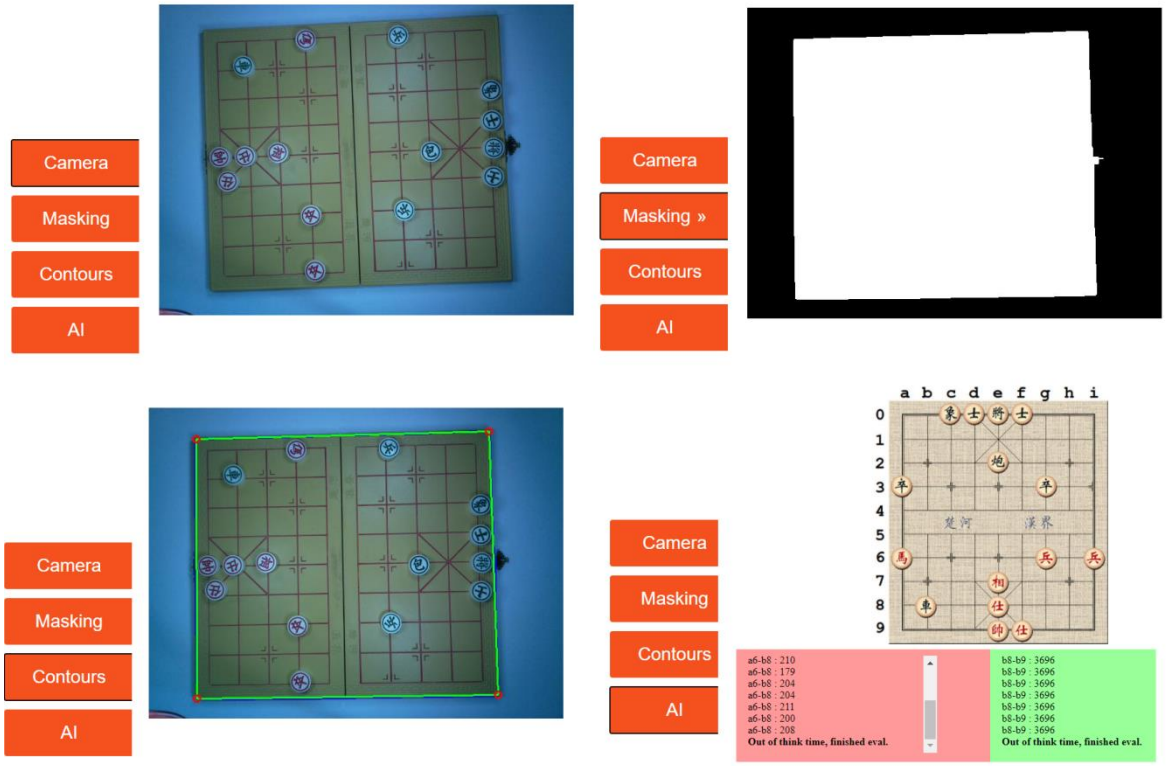
Hình 22. Ví dụ 2 về Đề xuất nước đi

5. Web Server

Một vài hình ảnh trang web của nhóm:



Hình 23. Hình ảnh trang Index



Hình 24. Lần lượt hình ảnh của các Tab trong trang Dashboard

Web Server có khả năng phục vụ nhiều client mà không làm tốc độ xử lý bị ảnh hưởng quá nhiều. Nhóm đã benchmark thời gian phản hồi của các công đoạn và thu thập số liệu lại như sau:

- Tab Camera, Masking, Contour

Bảng 5. Thời gian phản hồi các bước trong Tiền xử lý (đơn vị giây)

Lần	Camera	Masking	Contour
1	0.0002	0.1872	0.0278
2	0.0004	0.1633	0.0358
3	0.0010	0.1697	0.0322
4	0.0004	0.1524	0.0372
5	0.0007	0.1613	0.0311
6	0.0005	0.1650	0.0294
7	0.0004	0.1855	0.0329
8	0.0006	0.1643	0.0268
9	0.0006	0.1471	0.0261
10	0.0007	0.1455	0.0374
Trung bình	**0.00055**	**0.16413**	**0.03167**

- Nhận diện và vẽ (canvas) của tab AI

Bảng 6. Thời gian phản hồi của tab AI (đơn vị giây)

Lần	1	2	3	4	5	6
Canvas	4.902	4.305	7.835	9.908	9.908	7.06
7	8	9	10	11	12	**TB**
7.723	8.006	12.124	7.723	7.016	8.932	**7.9535**

Thời gian trên được thu thập sau mỗi lần bàn cờ bị thay đổi nhiều vị trí cùng một lúc. Trên thực tế, một lúc bàn cờ chỉ thay đổi 2 vị trí, nên thời gian chạy thực tế sẽ nhỏ hơn nhờ vào một vài phép tối ưu.

Riêng thời gian chạy của **Đề xuất nước đi** được giới hạn bởi thời gian suy nghĩ (Think time) được định nghĩa trong Engine, nên nhóm không cần quan tâm đến công đoạn này.

CHƯƠNG IV. KẾT LUẬN

Sản phẩm hoạt động chính xác và ổn định trong môi trường và điều kiện phổ thông đã được đặt ra tại phần Giải pháp. Sản phẩm còn có tốc độ phản hồi nhanh, tuy vẫn là chậm nếu so với tiêu chuẩn của thời gian thực, nhưng thời gian này nhóm cho rằng có thể chấp nhận được. Sản phẩm đã đạt được đúng mong muốn của các thành viên trong nhóm.

Nếu có thêm thời gian và kinh phí để tiếp tục phát triển project, nhóm sẽ tập trung vào một số tính năng sau:

- Web Server: Tối ưu thêm back-end. Làm cho công đoạn AI có tốc độ ngang bằng với các công đoạn khác. Làm cho front-end web đẹp hơn.
- Web Server: Có khả năng lưu game đang chơi và mở ra lại hay quay lui.
- Engine cờ: Thành một dịch vụ riêng chạy song song với Web Server, không còn bắt đầu chạy mỗi khi nó được gọi nữa.
- Tự động chơi cờ: Thêm một cánh tay robot hay một hình thức để máy tính có thể tương tác với bàn cờ vật lý.

CHƯƠNG V. TÀI LIỆU THAM KHẢO

- Raspberry Turk
- Raspberry Pi Camera pinout
- Setting up a Raspberry Pi 3 as an Access point
- OpenCV documentation: Thresholding Operations using inRange
- OpenCV documentation: Contour Features
- OpenCV documentation: Geometric Image Transformations
- OpenCV Shape detection
- Stackoverflow: OpenCV shape detection
- Detect shape using Hough Transform
- Overview Of Convolutional Neural Network In Image Classification
- A Gentle Introduction to Pooling Layers for Convolutional Neural Networks
- How Do Convolutional Layers Work in Deep Learning Neural Networks?
- Tensorflow documentation: TensorFlow Lite converter
- Github: Chinese Chess engine (Elephantfish)
- Github: Chess engine (Sunfish)
- Chess programming: Piece-Square Tables
- MTD(f) - A minimax algorithm faster than NegaScout
- Wikipedia: MTD-f
- http.server - Python Documentation 3.9.1
- Gist Github: simple_mjpeg_streamer_http_server