Đại Học Thăng Long
Khoa Công Nghệ Thông Tin

Tự Động Chấm Điểm Trắc Nghiệm Trung Học Phổ Thông Quốc Gia Bằng Thị Giác Máy Tính

Sinh viên:
A43968-Nguyễn Huy Mạnh

Giảng viên Hướng dẫn:
GV.Dương Văn Lạc

Hà Nội, Tháng 9 Năm 2024

Mục lục

Danh sách hình vẽ

Danh sách bảng

Chương 1

Tổng quan bài toán

1.1 Mô tả khái quát mục tiêu bài toán

Bài thi Trung học phổ thông (THPT) Quốc Gia là một sự kiện vô cùng quan trọng trong hệ thống giáo dục của Việt Nam. Đây là kỳ thi có quy mô lớn nhất trong cả nước, không chỉ để đánh giá trình độ học sinh sau 12 năm học tập mà còn là cơ sở quan trọng để xét tuyển vào các trường đại học và cao đẳng. Hàng năm, hàng triệu học sinh từ khắp các tỉnh thành tham gia vào kỳ thi này. Để đảm bảo tính khách quan, chính xác và công bằng trong quá trình chấm điểm, nhu cầu tự động hóa đang trở thành một xu hướng cấp bách, đặc biệt khi bài thi chủ yếu được thực hiện trên giấy.

Hiện nay, quá trình chấm thi thường được thực hiện thủ công bởi giáo viên và cán bộ chấm thi. Tuy nhiên, điều này không chỉ tốn kém thời gian mà còn dễ xảy ra các sai sót không mong muốn, như lỗi do con người hoặc sự chênh lệch trong quan điểm chấm điểm giữa các giám khảo. Điều này đặt ra yêu cầu phát triển các hệ thống chấm thi tự động, nhằm tối ưu hóa thời gian, nâng cao tính chính xác và công bằng, cũng như giảm bớt gánh nặng công việc cho đội ngũ giáo viên.

Trong bối cảnh sự phát triển mạnh mẽ của trí tuệ nhân tạo (AI) và thị giác máy tính (Computer Vision), việc áp dụng các công nghệ này vào chấm điểm bài thi trở nên khả thi và hiệu quả hơn bao giờ hết. Một hệ thống chấm điểm tự động có thể phân tích và đánh giá các bài thi viết hoặc các bài thi có cấu trúc tự luận, trắc nghiệm thông qua hình ảnh quét. Công nghệ thị giác máy tính sẽ giúp hệ thống nhận diện chữ viết, so sánh kết quả với các đáp án chuẩn, và đưa ra điểm số dựa trên các tiêu chí định sẵn.

Mục tiêu chính của đề tài này là phát triển một hệ thống chấm điểm tự động dành riêng cho bài thi THPT Quốc Gia dựa trên hình ảnh của các bài thi được chụp lại hoặc quét. Hệ thống sẽ sử dụng các thuật toán nhận diện và phân tích hình ảnh tiên tiến, có khả năng nhận biết và phân biệt giữa các loại bài thi như trắc nghiệm và tự luận. Đối với bài thi trắc nghiệm, hệ thống có thể dễ dàng nhận diện các câu trả lời thông qua việc phát hiện các lựa chọn được tô đen trên phiếu trả lời. Còn với bài thi tự luận, hệ thống sẽ ứng dụng công nghệ nhận diện chữ viết tay (OCR - Optical Character Recognition) để phân tích nội dung, từ đó đưa ra các đánh giá dựa trên các tiêu chí về tính đúng đắn, logic, và cấu trúc của câu trả lời.

1.2 Một số nghiên cứu liên quan

Viola, P., Jones, M. (2001). Rapid Object Detection Using a Boosted Cascade of Simple Features. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). Nghiên cứu này đã giới thiệu một phương pháp mới cho việc phát hiện đối tượng trong hình ảnh một cách nhanh chóng và hiệu quả, thông qua việc sử dụng một chuỗi đặc trưng đơn giản được tăng cường bằng kỹ thuật cascade. Đây là nền tảng quan trọng trong các ứng dụng liên quan đến thị giác máy tính, bao gồm cả nhận diện chữ viết và các đối tượng trong hình ảnh. Phương pháp Viola-Jones được áp dụng rộng rãi trong việc phát hiện khuôn mặt, và có thể được mở rộng cho các đối tượng khác trong bối cảnh chấm điểm tự động dựa trên hình ảnh của bài thi.

Krizhevsky, A., Sutskever, I., Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems (NIPS). Công trình này đã đánh dấu bước tiến quan trọng trong lĩnh vực học sâu (Deep Learning) và thị giác máy tính. Nhóm nghiên cứu đã sử dụng mạng nơ-ron tích chập sâu (CNN) để huấn luyện trên tập dữ liệu lớn ImageNet, đạt được kết quả ấn tượng trong việc phân loại hình ảnh. Mạng CNN mà nghiên cứu này phát triển có thể được áp dụng hiệu quả cho các bài toán nhận diện và phân tích hình ảnh trong hệ thống chấm điểm tự động, đặc biệt là đối với các bài thi viết tay hay hình ảnh có độ phức tạp cao.

OpenCV – Thư viện mã nguồn mở dành cho xử lý hình ảnh và thị giác máy tính. OpenCV là một trong những công cụ mạnh mẽ và phổ biến nhất trong lĩnh vực xử lý ảnh và thị giác máy tính. Nó cung cấp nhiều hàm và phương pháp giúp phát triển các ứng dụng liên quan đến xử lý hình ảnh, nhận diện đối tượng, phân đoạn ảnh, và xử lý video. Trong bối cảnh chấm thi tự động, OpenCV có thể được sử dụng để xử lý và phân tích hình ảnh của các bài thi, chẳng hạn như việc nhận diện ô tròn được tô trong bài thi trắc nghiệm, hoặc nhận diện chữ viết tay trong bài thi tự luận thông qua việc kết hợp với các mô hình nhận diện ký tự quang học (OCR).

Chương 2

Cơ sở lý thuyết

2.1 Các khái niệm cơ bản

Trong lĩnh vực thị giác máy tính và xử lý ảnh, có nhiều khái niệm cơ bản cần nắm vững để hiểu rõ cách xử lý và phân tích hình ảnh. Dưới đây là một số khái niệm quan trọng:

2.1.1 Ảnh

Ảnh kỹ thuật số là dạng biểu diễn số của hình ảnh, được cấu thành bởi các điểm ảnh (pixels). Mỗi điểm ảnh chứa thông tin về màu sắc hoặc độ sáng của hình ảnh tại một vị trí cụ thể. Có hai loại ảnh chính:

Ảnh màu (Color Image)

Ảnh màu chứa thông tin về màu sắc của từng điểm ảnh. Một ảnh màu thường được biểu diễn bằng ba kênh màu chính là *Red*, *Green*, và *Blue* (RGB). Mỗi kênh màu lưu trữ thông tin về cường độ của màu đó tại mỗi điểm ảnh, với giá trị thường nằm trong khoảng từ 0 đến 255. Kết hợp ba giá trị này sẽ tạo ra một màu cụ thể. Ví dụ, màu trắng được biểu diễn khi cả ba kênh đều có giá trị 255, trong khi màu đen được biểu diễn khi cả ba kênh đều bằng 0.

- Kênh đỏ (Red): Giá trị cường độ màu đỏ.
- Kênh xanh lá (Green): Giá trị cường độ màu xanh lá.
- Kênh xanh dương (Blue): Giá trị cường độ màu xanh dương.

Ảnh xám (Grayscale Image)

Ảnh xám chỉ chứa thông tin về độ sáng (mức xám) của các điểm ảnh mà không chứa thông tin về màu sắc. Mỗi điểm ảnh trong ảnh xám thường có giá trị từ 0 (đen) đến 255 (trắng). Ảnh xám được sử dụng phổ biến trong các bài toán nhận diện hình dạng, phát hiện cạnh, và các bài toán thị giác máy tính khác, vì giảm số lượng kênh màu giúp giảm độ phức tạp tính toán.

2.1.2 Chuyển đổi ảnh Grayscale

Việc chuyển đổi từ ảnh màu sang ảnh xám là một bước quan trọng để giảm số lượng thông tin cần xử lý trong nhiều ứng dụng xử lý ảnh. Quá trình này thường được thực hiện bằng cách tính toán giá trị trung bình có trọng số của các kênh màu (đỏ, xanh lá, xanh dương) tại mỗi điểm ảnh để tạo ra giá trị mức xám tương ứng. Công thức chuyển đổi phổ biến là:

$$GrayValue = 0.299 \times R + 0.587 \times G + 0.114 \times B \tag{2.1}$$

Trong đó R, G, B lần lượt là giá trị cường độ của các kênh màu đỏ, xanh lá, và xanh dương tại mỗi điểm ảnh.

2.1.3 Làm mịn hình ảnh (Image Smoothing)

Làm mịn hình ảnh là kỹ thuật dùng để giảm nhiễu hoặc làm mờ hình ảnh bằng cách áp dụng các bộ lọc hạ bậc thấp (*low-pass filter*). Quá trình này giúp loại bỏ các thông tin tần số cao như nhiễu hoặc cạnh sắc trong hình ảnh, làm cho các đối tượng trở nên trơn tru hơn. Phương pháp phổ biến để làm mịn là sử dụng bộ lọc Gaussian hoặc bộ lọc trung bình, với bộ lọc kernel là một ma trận nhỏ được áp dụng cho từng vùng nhỏ của ảnh.

Công thức của bộ lọc trung bình có thể biểu diễn như sau:

$$I'(x,y) = \frac{1}{K^2} \sum i = -\frac{K}{2}^{\frac{K}{2}} \sum j = -\frac{K}{2}^{\frac{K}{2}} I(x+i, y+j) \tag{2.2}$$

Trong đó $I'(x,y)$ là giá trị điểm ảnh sau khi làm mịn, $I(x+i, y+j)$ là giá trị điểm ảnh gốc trong vùng lân cận, và K là kích thước của kernel.

2.1.4 Xóa viền hình ảnh (Image Border Removal)

Xóa viền hình ảnh là quá trình loại bỏ các phần viền không mong muốn của hình ảnh. Các phần viền này có thể là nhiễu, các thông tin không liên quan, hoặc các vùng hình ảnh nằm ngoài phạm vi quan tâm. Quá trình này giúp làm sạch và tập trung vào phần chính của hình ảnh, tăng tính hiệu quả khi phân tích và xử lý.

2.1.5 Cắt ảnh (Image Cropping)

Cắt ảnh là kỹ thuật dùng để loại bỏ các phần không cần thiết của hình ảnh và chỉ giữ lại phần hình ảnh cần phân tích. Quá trình này được thực hiện bằng cách xác định một vùng hình chữ nhật trong ảnh gốc và trích xuất phần hình ảnh nằm trong vùng đó. Cắt ảnh thường được sử dụng để tập trung vào các đối tượng cụ thể trong hình ảnh hoặc để giảm kích thước ảnh, phục vụ cho các bài toán nhận diện, phân loại, và phát hiện đối tượng.

Ví dụ, nếu ta có một ảnh với kích thước $M \times N$, việc cắt ảnh từ vị trí $(x1, y1)$ đến $(x2, y2)$ sẽ tạo ra một ảnh mới với kích thước $(x2 - x1) \times (y2 - y1)$.

2.2 Các thư viện được sử dụng

Trong quá trình phát triển hệ thống chấm điểm tự động bằng thị giác máy tính, nhiều thư viện hỗ trợ xử lý ảnh và xây dựng giao diện đã được sử dụng. Dưới đây là danh sách các thư viện chính cùng với mô tả chi tiết:

2.2.1 Numpy

Numpy là một trong những thư viện cơ bản cho tính toán khoa học và số học trong Python. Nó cung cấp các công cụ mạnh mẽ để thao tác với ma trận, mảng đa chiều, và các phép tính toán học phức tạp. **NumPy** cho phép thực hiện các phép toán trên mảng với tốc độ cao nhờ tối ưu hóa việc xử lý theo vector hoá, so với các phương pháp thông thường trong Python.

Một số tính năng nổi bật của **NumPy** bao gồm:

- Tạo và thao tác các mảng đa chiều (multidimensional arrays).
- Hỗ trợ các phép toán ma trận, đại số tuyến tính, thống kê.
- Khả năng tích hợp với các thư viện khác như **SciPy**, **Pandas**.
- Tính toán nhanh nhờ khả năng xử lý hiệu quả với dữ liệu lớn.

Mảng trong NumPy có thể được tạo như sau:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
```

2.2.2 OpenCV

OpenCV (Open Source Computer Vision Library) là một thư viện mã nguồn mở nổi tiếng dành cho thị giác máy tính và xử lý ảnh. **OpenCV** cung cấp hàng loạt các công cụ và thuật toán mạnh mẽ để thực hiện các tác vụ như xử lý hình ảnh, phát hiện đối tượng, theo dõi đối tượng, và nhiều ứng dụng khác liên quan đến thị giác máy tính.

OpenCV hỗ trợ rất nhiều ngôn ngữ lập trình như C++, Python, Java,... nhưng phiên bản Python là phổ biến nhất trong các dự án liên quan đến học máy và trí tuệ nhân tạo. Một số tính năng của **OpenCV** bao gồm:

- Phát hiện đối tượng và nhận diện khuôn mặt.
- Xử lý video và phát hiện chuyển động.
- Thao tác với hình ảnh như đọc, ghi và chuyển đổi giữa các định dạng ảnh khác nhau.
- Phân tích hình ảnh như phát hiện cạnh, lọc ảnh, biến đổi hình học.

Ví dụ, để đọc và hiển thị một ảnh, có thể sử dụng OpenCV như sau:

```
import cv2
image = cv2.imread('image.jpg')
cv2.imshow('Image', image)
cv2.waitKey(0)
```

2.2.3 Imutils

Imutils là một thư viện Python nhằm đơn giản hóa và cải thiện quá trình xử lý và hiển thị hình ảnh khi sử dụng OpenCV. Nó không thay thế OpenCV mà bổ sung và cung cấp các tiện ích hữu ích giúp làm việc với hình ảnh dễ dàng hơn.

Các tính năng quan trọng của **Imutils** bao gồm:

- Xoay ảnh (image rotation) dễ dàng mà không cần viết các hàm thủ công.
- Thay đổi kích thước ảnh (image resizing) với các hàm đơn giản.
- Xử lý việc cắt ảnh (image cropping) và các thao tác hình ảnh khác như lật, làm mờ.

Ví dụ, để xoay một hình ảnh với **Imutils**, ta có thể làm như sau:

```
import imutils
rotated = imutils.rotate(image, 90)
```

2.2.4 Tkinter

Tkinter là thư viện chuẩn của Python để xây dựng giao diện người dùng đồ họa (GUI). Đây là thư viện rất hữu ích khi cần phát triển các ứng dụng giao diện trực quan, cho phép người dùng tương tác trực tiếp với hệ thống chấm điểm tự động qua các cửa sổ đồ họa.

Vì **Tkinter** là một phần của thư viện chuẩn Python, không cần cài đặt thêm bất kỳ phần mềm nào khác. Nó cung cấp một tập hợp các widget phong phú, bao gồm các nút (buttons), nhãn (labels), hộp văn bản (text boxes), và cửa sổ (windows).

Ví dụ, để tạo một cửa sổ đơn giản bằng **Tkinter**, ta có thể viết:

```
import tkinter as tk
window = tk.Tk()
window.title('Simple GUI')
window.mainloop()
```

2.2.5 PIL (Python Imaging Library)

PIL là một thư viện mạnh mẽ và linh hoạt cho xử lý hình ảnh trong Python. Thư viện này cho phép tạo, chỉnh sửa và xử lý hình ảnh với nhiều định dạng khác nhau. **PIL** đã được thay thế bởi **Pillow**, một nhánh phát triển của PIL, nhưng vẫn rất phổ biến trong các dự án xử lý hình ảnh.

Một số tính năng của **PIL** bao gồm:

- Đọc và ghi hình ảnh với nhiều định dạng như JPEG, PNG, BMP, GIF.
- Thay đổi kích thước, cắt, và xoay hình ảnh.
- Áp dụng các bộ lọc như làm mờ, tăng cường độ sáng, và phát hiện cạnh.

Ví dụ, để mở và hiển thị một ảnh bằng **PIL**, có thể sử dụng mã sau:

```
from PIL import Image
img = Image.open('image.jpg')
img.show()
```

Chương 3

Mô tả dữ liệu

3.1 Dữ liệu ban đầu (Ảnh)

Dữ liệu đầu vào của hệ thống chấm điểm tự động là hình ảnh các bài thi được quét từ các phiếu trả lời của học sinh tham gia kỳ thi THPT Quốc Gia. Mỗi ảnh đại diện cho bài thi của một học sinh cụ thể, bao gồm các câu trả lời được đánh dấu trên phiếu thi. Dữ liệu này thường được chụp hoặc quét ở độ phân giải cao nhằm đảm bảo chất lượng hình ảnh tốt nhất để xử lý và phân tích.

Các bài thi trong kỳ thi THPT Quốc Gia bao gồm nhiều môn khác nhau, được chia thành các khối thi cụ thể như sau:

3.1.1 Môn Toán

Bài thi Toán gồm 50 câu hỏi trắc nghiệm. Mỗi câu hỏi có các lựa chọn A, B, C, D, và thí sinh sẽ chọn đáp án mà họ cho là đúng bằng cách tô đậm vào ô tương ứng trên phiếu trả lời. Các câu trả lời này cần được hệ thống nhận diện chính xác để đánh giá kết quả của thí sinh.

3.1.2 Môn Tiếng Anh

Bài thi Tiếng Anh có tổng cộng 40 câu hỏi trắc nghiệm. Tương tự như môn Toán, thí sinh sẽ tô các ô đáp án A, B, C, D. Dữ liệu ảnh của bài thi này phải đảm bảo hệ thống có thể phân biệt rõ các vùng chứa câu hỏi, đáp án và nhận diện được lựa chọn của thí sinh.

3.1.3 Các môn tổ hợp (Khoa học tự nhiên và Khoa học xã hội)

Các môn tổ hợp bao gồm 120 câu hỏi, với mỗi môn tổ hợp chia thành ba bài thi nhỏ. Mỗi bài thi sẽ gồm các môn như: Vật Lý, Hóa Học, Sinh Học (đối với tổ hợp Khoa học tự nhiên) hoặc Lịch Sử, Địa Lý, Giáo Dục Công Dân (đối với tổ hợp Khoa học xã hội). Tổng số câu hỏi của các môn tổ hợp là 120 câu, và yêu cầu hệ thống phải phân biệt được từng môn để chấm điểm chính xác.

3.1.4 Yêu cầu về chất lượng ảnh

Để đảm bảo quá trình nhận diện và chấm điểm diễn ra chính xác, các ảnh bài thi đầu vào cần thỏa mãn các điều kiện sau:

- Ảnh phải được quét hoặc chụp với độ phân giải cao, đủ để nhận diện rõ ràng các chi tiết nhỏ như ô trắc nghiệm và các vùng chứa câu hỏi.

- Ảnh phải được căn chỉnh đúng, không bị nghiêng hay lệch để tránh sai sót trong việc nhận diện các đáp án được tô.
- Ảnh phải sạch, không có các dấu mờ hoặc nhiễu ảnh gây khó khăn cho quá trình xử lý và nhận diện.

Dữ liệu này đóng vai trò nền tảng để hệ thống thị giác máy tính có thể áp dụng các thuật toán nhận diện, phân tích và chấm điểm một cách chính xác và hiệu quả. Mỗi môn thi yêu cầu một cách xử lý riêng biệt do đặc thù của số lượng câu hỏi và cách tổ chức bố cục phiếu trả lời, đòi hỏi hệ thống phải linh hoạt và chính xác trong việc nhận diện hình ảnh.

3.2 Xử lý ảnh

3.2.1 Lọc các đối tượng

Hàm `getContours` được sử dụng để xác định và lọc các đối tượng trong một hình ảnh dựa trên diện tích và số lượng cạnh của chúng. Hàm này rất hữu ích trong việc xử lý và phân tích hình ảnh, giúp hệ thống tập trung vào các đối tượng quan trọng. Dưới đây là mô tả chi tiết cách hoạt động của hàm.

Cấu trúc của hàm

```
def getContours(img, cThread=[100,100], minArea=1000, filter=4):
```

- `img`: Hình ảnh đầu vào cần xử lý.
- `cThread=[100,100]`: Danh sách chứa ngưỡng dưới và ngưỡng trên để phát hiện cạnh Canny.
- `minArea=1000`: Diện tích tối thiểu của một contour để được coi là hợp lệ.
- `filter=4`: Số cạnh yêu cầu của các đối tượng được lọc. Ví dụ, `filter=4` lọc những đối tượng có 4 cạnh (hình chữ nhật hoặc hình vuông).

Các bước thực hiện

1. **Chuyển đổi ảnh sang ảnh xám**:

   ```
   imgGray = cv2.cvtColor(img, cv2.COLOR BGR2GRAY)
   ```

   Hình ảnh màu (BGR) được chuyển đổi sang ảnh xám (grayscale) để dễ dàng xử lý hơn, giúp giảm độ phức tạp khi không cần phân tích màu sắc.

2. **Làm mờ ảnh bằng Gaussian Blur**:

   ```
   imgBlur = cv2.GaussianBlur(imgGray, (5, 5), 1)
   ```

   Hàm `cv2.GaussianBlur` được sử dụng để làm mịn hình ảnh, giảm nhiễu và các chi tiết không cần thiết.

3. **Phát hiện cạnh bằng Canny**:

```
imgCanny = cv2.Canny(imgBlur, cThread[0], cThread[1])
```

Phép phát hiện cạnh Canny được áp dụng để tìm biên của các đối tượng trong ảnh.

4. **Làm dày các cạnh bằng phép Dilation**:

```
kernel = np.ones((5,5))
imgDilation = cv2.dilate(imgCanny, kernel, iterations=3)
```

Hàm `cv2.dilate` sử dụng kernel để làm dày các cạnh phát hiện trong ảnh, giúp làm rõ các đường biên của đối tượng.

5. **Làm mỏng các cạnh bằng phép Erosion**:

```
imgThre = cv2.erode(imgDilation, kernel, iterations=2)
```

Phép erode giúp làm mỏng lại các đường biên sau khi đã làm dày để loại bỏ các chi tiết nhiễu nhỏ.

6. **Tìm các contours trong ảnh**:

```
contours,   = cv2.findContours(imgThre,
cv2.RETR EXTERNAL, cv2.CHAIN APPROX SIMPLE)
```

Hàm `cv2.findContours` được sử dụng để tìm tất cả các contours trong ảnh sau khi đã qua các bước xử lý.

7. **Lọc các contours hợp lệ**:

```
final countours = []
for contour in contours:
    area = cv2.contourArea(contour)
    if area > minArea:
        peri = cv2.arcLength(contour, True)
        approx = cv2.approxPolyDP(contour, 0.02*peri, True)
        bbox = cv2.boundingRect(approx)
        if filter > 0:
            if len(approx) == filter:
                final countours.append([len(approx), area, approx,
        else:
            final countours.append([len(approx), area, approx, bbox
```

- Vòng lặp này duyệt qua tất cả các contours tìm được. - Các contours có diện tích lớn hơn `minArea` sẽ được tiếp tục xử lý, bao gồm việc xấp xỉ thành đa giác (sử dụng hàm `cv2.approxPolyDP`) và tính bounding box. - Nếu giá trị `filter` lớn hơn 0, hàm sẽ chỉ giữ lại các contours có số cạnh khớp với giá trị `filter`.

8. **Sắp xếp các contours theo diện tích**:

```
final countours = sorted(final countours, key=lambda x: x[1], rever
```

Các contours hợp lệ được sắp xếp theo diện tích từ lớn đến nhỏ.

9. **Trả về kết quả**:

```
return img, final countours
```

Hàm trả về hình ảnh gốc và danh sách các contours hợp lệ.

```python
def getContours(img,cThread=[100,100],minArea=1000, filter=4):
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    imgBlur = cv2.GaussianBlur(imgGray, (5, 5), 1)
    imgCanny = cv2.Canny(imgBlur,cThread[0],cThread[1])
    kernel = np.ones((5,5))
    imgDilation = cv2.dilate(imgCanny, kernel, iterations = 3)
    imgThre = cv2.erode(imgDilation, kernel, iterations = 2)

    contours, _ = cv2.findContours(imgThre, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    final_countours = []
    for contour in contours:
        area = cv2.contourArea(contour)
        if area > minArea:
            peri = cv2.arcLength(contour,True)
            approx = cv2.approxPolyDP(contour,0.02*peri,True)
            bbox = cv2.boundingRect(approx)
            if filter > 0:
                if len(approx) == filter:
                    final_countours.append([len(approx),area,approx,bbox,contour])
            else:
                final_countours.append([len(approx),area,approx,bbox,contour])
    final_countours = sorted(final_countours, key = lambda x:x[1], reverse=True)

    return img, final_countours
```

Hình 3.1: Đoạn mã lọc các đối tượng

3.2.2 Cắt hình ảnh

Để thực hiện phép biến đổi hình ảnh và cắt chỉ giữ lại phần quan trọng, chúng tôi đã xây dựng hàm `wrapImage` với mục đích áp dụng các phép biến đổi hình học (warp) dựa trên các điểm xác định (contours) trong hình ảnh đầu vào. Hàm này rất hữu ích trong các ứng dụng như quét tài liệu, nhận dạng văn bản từ ảnh hoặc cắt các phần cụ thể từ hình ảnh. Dưới đây là mô tả chi tiết về cách hoạt động của hàm này.

Mô tả hàm `wrapImage`

```
def wrapImage(img, points, widthImg, heightImg, pad=0):
```

- `img`: Đây là hình ảnh đầu vào cần biến đổi. Hình ảnh này thường là một bức ảnh chứa các vùng hoặc đối tượng cần được cắt ra.
- `points`: Danh sách các điểm (contours) được sử dụng để xác định phép biến đổi. Mỗi điểm trong danh sách này đại diện cho một góc của vùng cần quan tâm. Danh sách này phải chứa chính xác 4 điểm.
- `widthImg`, `heightImg`: Đây là kích thước mục tiêu của hình ảnh sau khi biến đổi. Hình ảnh đầu ra sẽ có kích thước tương ứng với `widthImg` và `heightImg`.
- `pad`: Tham số này dùng để loại bỏ lề (padding) từ hình ảnh đã được biến đổi. Giá trị mặc định là 0, tức là không có lề bị loại bỏ.

3.2.3 Các bước hoạt động của hàm

1. **Tìm và sắp xếp các điểm contour hợp lệ**:

```
points = get 4 contour(points)
```

   Hàm `get 4 contour` được gọi để đảm bảo rằng danh sách `points` chứa chính xác 4 điểm, đại diện cho các góc của vùng cần biến đổi. Các điểm này có thể không được cung cấp theo thứ tự, do đó hàm này sẽ xác định và sắp xếp các điểm theo thứ tự: góc trên bên trái, góc trên bên phải, góc dưới bên trái và góc dưới bên phải.

2. **Chuẩn bị các điểm để biến đổi**:

```
pts1 = np.float32(points)
pts2 = np.float32([[0, 0], [widthImg, 0],
                   [0, heightImg], [widthImg, heightImg]])
```

   Tạo hai mảng `pts1` và `pts2`:

   - `pts1`: chứa tọa độ của 4 điểm được xác định từ danh sách `points`. Đây là các điểm gốc cần biến đổi.
   - `pts2`: chứa tọa độ của 4 điểm tương ứng trong hình ảnh đích, với kích thước `widthImg` và `heightImg`. Các điểm này được định nghĩa là 4 góc của hình chữ nhật đích.

3. **Tính toán ma trận biến đổi**:

```
matrix = cv2.getPerspectiveTransform(pts1, pts2)
```

   Hàm `cv2.getPerspectiveTransform` tính toán ma trận biến đổi dựa trên hai bộ tọa độ điểm `pts1` và `pts2`. Ma trận này sẽ được sử dụng để thực hiện phép biến đổi phối cảnh (perspective transformation).

4. **Áp dụng phép biến đổi hình ảnh**:

```
wrap = cv2.warpPerspective(img, matrix,
                           (widthImg, heightImg))
```

Hàm `cv2.warpPerspective` thực hiện phép biến đổi phối cảnh lên hình ảnh đầu vào `img` bằng cách sử dụng ma trận biến đổi `matrix`. Kết quả trả về là một hình ảnh đã được biến đổi và cắt theo vùng chỉ định.

5. **Loại bỏ lề nếu cần thiết**:

```
wrap = wrap[pad:wrap.shape[0]-pad, pad:wrap.shape[1]-pad]
```

Nếu tham số `pad` lớn hơn 0, hàm sẽ loại bỏ lề khỏi hình ảnh đích bằng cách cắt bỏ phần biên dựa trên giá trị `pad`. Phần lề này có thể là các vùng không mong muốn xung quanh hình ảnh.

Hàm `get 4 contour`

Hàm `get 4 contour` nhận vào một danh sách các điểm (contours) và trả về một mảng chứa chính xác 4 điểm đại diện cho các góc của vùng cần biến đổi. Việc sắp xếp lại các điểm này rất quan trọng để đảm bảo rằng phép biến đổi sẽ diễn ra đúng cách. Dưới đây là các bước hoạt động của hàm:

1. **Tính trung bình tọa độ của các điểm**:

```
center = np.mean(points, axis=0).astype(int)
```

Hàm tính toán trung bình tọa độ của tất cả các điểm trong `points`, kết quả là điểm trung tâm `center`. Điểm này được sử dụng để phân loại các điểm thành hai nhóm: nhóm điểm nằm phía trên và nhóm điểm nằm phía dưới điểm trung tâm.

2. **Phân chia các điểm thành hai nhóm**:

```
points above center = np.array([point.squeeze()
                for point in points if point.squeeze()[1] < center[
points below center = np.array([point.squeeze()
                for point in points if point.squeeze()[1] >= center
```

Các điểm trong danh sách `points` được phân chia thành hai nhóm dựa trên vị trí của chúng so với điểm trung tâm:

- `points above center`: chứa các điểm có tọa độ y nhỏ hơn điểm trung tâm.
- `points below center`: chứa các điểm có tọa độ y lớn hơn hoặc bằng điểm trung tâm.

3. **Xác định các điểm góc**:

```
top left = points above center[np.argmin(points above center[:, 0])
top right = points above center[np.argmax(points above center[:, 0]
botton left = points below center[np.argmin(points below center[:,
botton right = points below center[np.argmax(points below center[:,
```

Các điểm góc được xác định dựa trên tọa độ x và y của chúng:

```python
def get_4_contour(points):
    center = np.mean(points, axis=0).astype(int)

    points_above_center = np.array([point.squeeze() for point in points if point.squeeze()[1] < center[0][1]])
    points_below_center = np.array([point.squeeze() for point in points if point.squeeze()[1] >= center[0][1]])

    top_left = points_above_center[np.argmin(points_above_center[:, 0])]
    top_right = points_above_center[np.argmax(points_above_center[:, 0])]
    botton_left = points_below_center[np.argmin(points_below_center[:, 0])]
    botton_right = points_below_center[np.argmax(points_below_center[:, 0])]
    return np.array([[top_left], [top_right], [botton_left],[botton_right]])
```

Hình 3.2: Đoạn mã hàm get contour

```python
def wrapImage(img, points, widthImg, heightImg, pad = 0):
    points= get_4_contour(points)
    pts1 = np.float32(points)
    pts2 = np.float32([[0, 0],[widthImg, 0], [0, heightImg],[widthImg, heightImg]])

    matrix = cv2.getPerspectiveTransform(pts1, pts2)
    wrap = cv2.warpPerspective(img, matrix, (widthImg, heightImg))
    wrap = wrap[pad:wrap.shape[0]-pad,pad:wrap.shape[1]-pad]
    return wrap
```

Hình 3.3: Đoạn mã hàm Wrap Image

- `top left`: điểm có tọa độ x nhỏ nhất trong `points above center`.
- `top right`: điểm có tọa độ x lớn nhất trong `points above center`.
- `botton left`: điểm có tọa độ x nhỏ nhất trong `points below center`.
- `botton right`: điểm có tọa độ x lớn nhất trong `points below center`.

4. **Trả về các điểm góc**:

```
return np.array([[top left], [top right],
                 [botton left], [botton right]])
```

Hàm trả về một mảng chứa 4 điểm đại diện cho các góc của hình ảnh hoặc vùng quan tâm.

Chương 4

Xây dựng mô hình bài toán

4.1 Mô tả bài toán

Nhận diện số báo danh và mã đề thi: Các phần thông tin khác trên phiếu thi như số báo danh và mã đề thi cũng cần được nhận diện và trích xuất từ hình ảnh. Điều này có thể thực hiện qua các kỹ thuật nhận diện ký tự quang học (OCR) hoặc các phương pháp thị giác máy tính khác.

So sánh với đáp án chính thức: Sau khi trích xuất các đáp án từ bài thi, hệ thống sẽ so sánh chúng với đáp án chính thức đã được nhập trước đó. Phương pháp so sánh có thể bao gồm việc đối chiếu trực tiếp hoặc sử dụng các thuật toán tính điểm tự động.

Tính toán điểm số và kết quả: Dựa trên sự so sánh giữa đáp án thí sinh và đáp án chính thức, hệ thống sẽ tính toán điểm số cho từng phần của bài thi và tổng hợp điểm số cuối cùng. Kết quả làm bài, số báo danh, và mã đề thi của thí sinh sẽ được trả về dưới dạng thông báo kết quả.

4.2 Xây dựng mô hình

4.2.1 Cắt vùng phiếu

Để xử lý và trích xuất các phần khác nhau của phiếu trả lời từ một hình ảnh đầu vào, chúng tôi đã phát triển một số hàm xử lý ảnh. Dưới đây là mô tả chi tiết về từng hàm, giải thích cách chúng hoạt động và ứng dụng của chúng trong việc xử lý ảnh phiếu trả lời.

Hàm findFullAnswerSheet

Hàm `findFullAnswerSheet` nhận một đường dẫn đến hình ảnh bài thi (`pathImage`) và cắt cũng như biến đổi hình ảnh ban đầu để tạo ra một ảnh vùng phiếu trả lời đầy đủ với kích thước mục tiêu.

```
def findFullAnswerSheet(pathImage, width=1830, height=2560): img = cv2.imread(pathImage
```

- **Đầu vào**: Đường dẫn đến hình ảnh bài thi.
- **Xử lý**:
  - Đọc hình ảnh từ đường dẫn.
  - Sử dụng hàm `getContours` để phát hiện các đối tượng trong hình ảnh.
  - Xác định và sắp xếp các điểm góc của vùng phiếu trả lời bằng hàm `get 4 contour`.

– Cắt và biến đổi hình ảnh thành kích thước mục tiêu bằng hàm `wrapImage`.

- **Đầu ra**: Ảnh vùng phiếu trả lời đầy đủ với kích thước mục tiêu.

Hàm gettestcodeimage

Hàm `get test code image` nhận một ảnh vùng phiếu trả lời và trả về một ảnh con chứa phần mã đề thi.

```
def get test code image(image): height, width, channels = image.shape per width = [0.87
        (per height[1]*height), int(per width[0]*width)
        (per width[1]*width)]
```

- **Đầu vào**: Ảnh vùng phiếu trả lời.
- **Xử lý**:
  – Xác định vùng chứa mã đề thi dựa trên tỷ lệ phần trăm chiều rộng và chiều cao của ảnh.
- **Đầu ra**: Ảnh con chứa phần mã đề thi.

Hàm get student code image

Hàm `get student code image` tương tự như hàm `get test code image`, nhưng trả về ảnh con chứa phần số báo danh.

```
def get student code image(image): height, width, channels = image.shape per width = (0
        (per height[1]*height), int(per width[0]*width)
        (per width[1]*width)]
```

- **Đầu vào**: Ảnh vùng phiếu trả lời.
- **Xử lý**:
  – Xác định vùng chứa số báo danh dựa trên tỷ lệ phần trăm chiều rộng và chiều cao của ảnh.
- **Đầu ra**: Ảnh con chứa phần số báo danh.

Hàm get sheet ans image

Hàm `get sheet ans image` trích xuất và trả về phần vùng chứa các câu trả lời trong ảnh vùng phiếu trả lời.

```
def get sheet ans image(image): height, width, channels = image.shape per height = (0.3
        (per height[1]*height), int(per width[0]*width)
        (per width[1]*width)] return full sheet
```

- **Đầu vào**: Ảnh vùng phiếu trả lời.
- **Xử lý**:

- Xác định vùng chứa các câu trả lời dựa trên tỷ lệ phần trăm chiều rộng và chiều cao của ảnh.

- **Đầu ra**: Ảnh con chứa phần vùng câu trả lời.

Hàm get part sheet ans image

Hàm `get part sheet ans image` phân chia hình ảnh vùng câu trả lời thành 4 phần tương ứng với các góc của vùng câu trả lời.

```
def get part sheet ans image(image, dis=10): height, width, channels = image.shape dis
        (round((width + dis) / 2 - dis, 0))], image[:, int(round((width + dis) / 2, 0)):
        (round((A.shape[1] + dis) / 2 - dis, 0))], A[:, int(round((A.shape[1] + dis) / 2
        (round((B.shape[1] + dis) / 2 - dis, 0))], B[:, int(round((B.shape[1] + dis) / 2
```

- **Đầu vào**: Ảnh vùng câu trả lời và tham số khoảng cách (`dis`).

- **Xử lý**:

  - Chia hình ảnh thành 4 phần tương ứng với các góc của vùng câu trả lời dựa trên khoảng cách được xác định.

- **Đầu ra**: Danh sách chứa 4 phần của ảnh tương ứng với các góc của vùng câu trả lời.

```python
def get_test_code_image(image):
    height, width, channels = image.shape
    per_width = [0.876, 0.94]
    per_height = (0.0975, 0.298)
    return image[int(per_height[0]*height):int(per_height[1]*height),\
                 int(per_width[0]*width):int(per_width[1]*width)]

def get_student_code_image(image):
    height, width, channels = image.shape
    per_width = (0.727, 0.845)
    per_height = (0.0975, 0.298)
    return image[int(per_height[0]*height):int(per_height[1]*height),\
                 int(per_width[0]*width):int(per_width[1]*width)]

def get_sheet_ans_image(image):
    height, width, channels = image.shape
    per_height = (0.326, 0.945)
    per_width = (0.055, 0.925)
    full_sheet = image[int(per_height[0]*height):int(per_height[1]*height),\
                 int(per_width[0]*width):int(per_width[1]*width)]
    return full_sheet

def get_part_sheet_ans_image(image,dis=10):
    height, width, channels = image.shape
    dis = 0.03562 * width
    A, B = image[:,:int(round((width+dis)/2 - dis,0))], image[:,int(round((width+dis)/2,0)):]
    A1,A2 = A[:,:int(round((A.shape[1]+dis)/2 - dis,0))], A[:,int(round((A.shape[1]+dis)/2,0)):]
    B1,B2 = B[:,:int(round((B.shape[1]+dis)/2 - dis,0))], B[:,int(round((B.shape[1]+dis)/2,0)):]
    return [A1,A2,B1,B2]
```

(variable) height: Any

Hình 4.1: Đoạn mã các hàm cắt vùng chiếu

```python
def get_student_code(img_sbd, thresh_value = 150):
    # Xử lý ảnh
    gray = cv2.cvtColor(img_sbd, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (5, 5), 0)
    # Xoá viền
    kernel = np.ones((5,5))
    imgDilation = cv2.dilate(blur, kernel, iterations = 2)
    # Phóng to phần tô màu
    erosion = cv2.erode(imgDilation, kernel, iterations = 2)
    # Chuyển thành giá trị nhị phân, giá trị nào dưới 100 về 0, ngược lại 255 (đen và trắng)
    _, thresh = cv2.threshold(erosion, thresh_value, 255, cv2.THRESH_BINARY)

    # cv2_imshow(thresh)
    column_width = img_sbd.shape[1] // 6
    bubble_positions = [int(i*(img_sbd.shape[0]//10)) for i in range(11)]

    student_ID = ""
    for i in range(6): # Có 6 cột số báo danh
        column = thresh[:,i*column_width:(i+1)*column_width]
        selected_number = None
        min_mean = float('inf')

        for pos in range (10): # Có 10 dòng từ 0-9
            start, end = bubble_positions[pos], bubble_positions[pos+1]
            mean_value = np.mean(column[start:end, :])
            # Tìm giá trị trung bình nhỏ nhất
            # Nhỏ nhất là chứa nhiều phần tử màu đen nhất, tức chọn ô này
            if mean_value < min_mean:
                min_mean = mean_value
                selected_number = pos
        student_ID += str(selected_number)
    return student_ID
```

Hình 4.2: Đoạn mã lấy số báo danh

4.2.2 Lấy số báo danh

Hàm get student code(img sbd, thresh value = 150): img sbd: Đây là hình ảnh chứa số báo danh của học sinh.11 thresh value: Ngưỡng (threshold) để chuyển hình ảnh thành ảnh nhị phân. Giá trị mặc định là 150. Hàm này có các bước chính: + Chuyển hình ảnh thành hình ảnh xám. + Áp dụng Gaussian Blur để làm mịn hình ảnh. + Xoá viền của hình ảnh bằng cách sử dụng phép dilation và erosion. + Chuyển hình ảnh thành ảnh nhị phân, trong đó giá trị dưới ngưỡng thresh value được đặt thành 0 (đen), và ngược lại được đặt thành 255 (trắng). + Sau đó, hình ảnh được chia thành 6 cột bằng cách lấy tổng số cột và chia cho 6. Các vị trí của ô tròn trong cột được xác định bằng cách lấy tổng số dòng và chia cho 10, tạo ra một danh sách bubble positions chứa vị trí dòng của các ô tròn. Hàm sau đó duyệt qua từng cột của hình ảnh, và trong mỗi cột, tìm dòng (số từ 0 đến 9) mà có giá trị trung bình (mean) của ô tròn nhỏ nhất (tức là ô tròn đó chứa nhiều màu đen nhất). Giá trị này được thêm vào chuỗi student ID làm số báo danh. Sau khi duyệt qua tất cả các cột, hàm trả về số báo danh

4.2.3 Lấy mã đề thi

Hàm này hoàn toàn tương tự với get student code, nhưng được sử dụng để nhận dạng và trích xuất mã đề thi từ hình ảnh. Và nó được chia thành 3 cột thay vì 6. Chức năng của cả hai

```python
def get_test_code(img, thresh_value = 150):
    # Xử lý ảnh
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (5, 5), 0)
    # Xoá viền
    kernel = np.ones((5,5))
    imgDilation = cv2.dilate(blur, kernel, iterations = 2)
    # Phóng to phần tô màu
    erosion = cv2.erode(imgDilation, kernel, iterations = 2)
    # Chuyển thành giá trị nhị phân, giá trị nào dưới 100 về 0, ngược lại 255 (đen và trắng)
    _, thresh = cv2.threshold(erosion, thresh_value, 255, cv2.THRESH_BINARY)
    # cv2_imshow(thresh)
    column_width = img.shape[1] // 3
    bubble_positions = [int(i*(img.shape[0]//10)) for i in range(11)]

    test_ID = ""
    for i in range(3): # Có 6 cột số báo danh
        column = thresh[:,i*column_width:(i+1)*column_width]
        selected_number = None
        min_mean = float('inf')

        for pos in range (10): # Có 10 dòng từ 0-9
            start, end = bubble_positions[pos], bubble_positions[pos+1]
            mean_value = np.mean(column[start:end, :])
            # Tìm giá trị trung bình nhỏ nhất
            # Nhỏ nhất là chứa nhiều phần tử màu đen nhất, tức chọn ô này
            if mean_value < min_mean:
                min_mean = mean_value
                selected_number = pos
        test_ID += str(selected_number)
    return test_ID
```

Hình 4.3: Đoạn mã lấy số đề thi

hàm là trích xuất thông tin số học sinh và mã đề thi từ hình ảnh dựa trên sự tương quan giữa các ô tròn và nền của hình ảnh. Dưới đây là hình ảnh hàm lấy mã đề thi của thí sinh

4.2.4 Lấy đáp án

Hàm `get my ans` được sử dụng để xác định các đáp án của thí sinh từ một hình ảnh chứa các câu trả lời trên phiếu trả lời. Hàm thực hiện các bước xử lý ảnh và phân tích để xác định các câu trả lời được chọn. Dưới đây là mô tả chi tiết về cách hoạt động của hàm:

- **Đầu vào**:
  - `image ans`: Ảnh chứa các câu trả lời của thí sinh.
  - `ANSWER KEY`: Danh sách các đáp án đúng cho các câu hỏi.
  - `thresh value`, `limit value`, `start` : Các tham số để điều chỉnh ngưỡng phân loại và khu vực cắt của ảnh.
- **Xử lý**:
  - Chuyển ảnh sang màu xám và áp dụng làm mờ Gaussian.
  - Sử dụng phép dãn và co lại để cải thiện chất lượng ảnh.

- Chuyển ảnh thành nhị phân với ngưỡng xác định.
- Cắt vùng câu trả lời và phân tích để xác định đáp án bằng cách tính giá trị trung bình trong các vùng.
- So sánh đáp án của thí sinh với đáp án đúng và đánh dấu trên ảnh kết quả.

- **Đầu ra**: Danh sách các đáp án của thí sinh.

Hàm `get my ans` thực hiện việc phân tích và so sánh các đáp án của thí sinh với đáp án đúng, đồng thời đánh dấu các câu trả lời đúng và sai trên hình ảnh kết quả.

```
        def get_my_ans(image_ans, ANSWER_KEY, thresh_value = 150,
            limit_value = 240, start_ = 55):
            translate = {"A": 0, "B": 1, "C": 2, "D": 3}
            revert_translate = {0: "A", 1: "B", 2: "C", 3: "D", -1: "N
                "}
            img = image_ans[:,start_:]

            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            blur = cv2.GaussianBlur(gray, (5, 5), 0)
            kernel = np.ones((5,5))
            imgDilation = cv2.dilate(blur, kernel, iterations = 2)
            erosion = cv2.erode(imgDilation, kernel, iterations = 2)
            _, thresh = cv2.threshold(erosion, thresh_value, 255, cv2.
                THRESH_BINARY)

            ans_char = ['A', 'B', 'C', 'D']
            height_sub = img.shape[0] // 6
            n_part, n_questions = 6, 5
            pad = 5
            my_answers = []

            for i_part in range(n_part):
                sub_img = thresh[height_sub*i_part+pad*3:height_sub*(
                    i_part+1)-pad*2, pad:-pad]
                bubble_positions = [int(idx*(sub_img.shape[0]//
                    n_questions)) for idx in range(n_questions+1)]

                for j in range(n_questions):
                    start, end = bubble_positions[j], bubble_positions
                        [j+1]
                    row_1_question = sub_img[start:end, :]
                    min_mean = float('inf')
                    selected_ans = None
                    width_sub = row_1_question.shape[1] // 4

                    for i_ans in range(4):
                        mean_value = np.mean(row_1_question[:, i_ans*
                            width_sub:(i_ans+1)*width_sub])
                        if mean_value < limit_value and mean_value <
                            min_mean:
                            min_mean = mean_value
                            selected_ans = ans_char[i_ans]

```

```
                    my_answers.append(selected_ans if selected_ans is
                        not None else "-")

            r = 15
            for part in range(n_part):
                start_h, start_w = height_sub * part + 45, 35 + start_

                try:
                    for quest in range(n_questions):
                        idx = part * n_questions + quest
                        s_w = start_w + 74 * translate[ANSWER_KEY[idx
                            ]]
                        if my_answers[idx] == ANSWER_KEY[idx]:
                            cv2.circle(image_ans, (s_w, start_h), r,
                                (0, 255, 0), 3)
                        else:
                            w_wrong_ans = start_w + 74 * translate[
                                my_answers[idx]]
                            cv2.circle(image_ans, (w_wrong_ans,
                                start_h), r, (0, 0, 255), 3)
                            cv2.circle(image_ans, (s_w, start_h), r,
                                (255, 0, 0), 3)
                        start_h += 45
                except:
                    break

            return my_answers
```

Listing 4.1: Đoạn mã lấy đáp án

Chương 5

Giao diện người dùng

Chương trình chấm điểm tự động THPTQG được phát triển nhằm mục đích hỗ trợ quá trình chấm điểm bài thi trắc nghiệm của học sinh trong kỳ thi THPT Quốc gia. Giao diện thân thiện, dễ sử dụng và có thể tải ảnh bài thi, phân tích, và trả kết quả ngay lập tức.

5.1 Các bước sử dụng

Bước 1: Tải ảnh bài thi lên

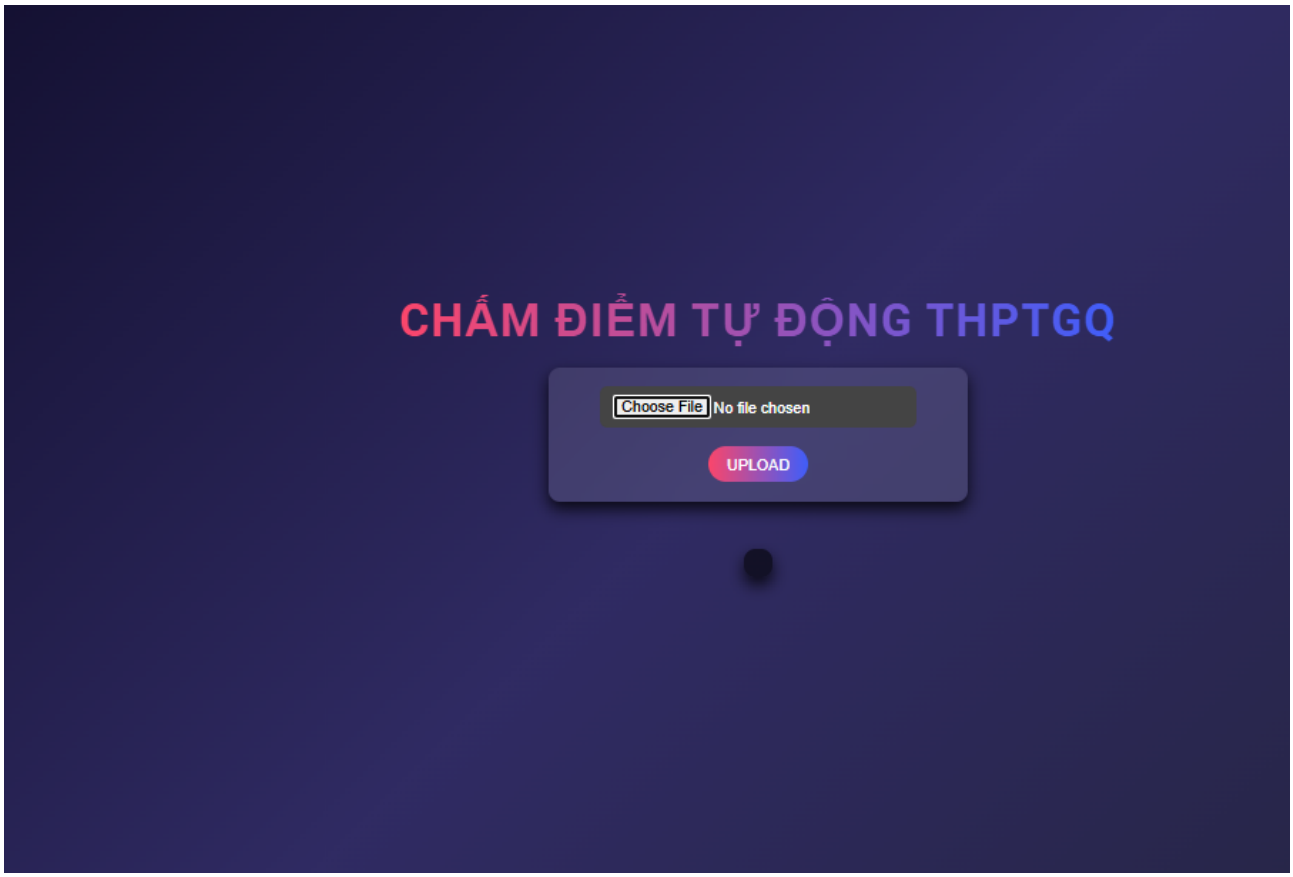
- Trên giao diện chính, bấm nút *Choose File* để chọn ảnh bài thi trắc nghiệm mà bạn muốn chấm điểm.
- Sau khi chọn ảnh, bấm nút *Upload* để tải ảnh lên hệ thống.

Bước 2: Chờ quá trình xử lý ảnh

- Sau khi ảnh được tải lên, hệ thống sẽ tự động phân tích và chấm điểm. Trong thời gian này, bạn có thể thấy ảnh được tải lên và đang được xử lý.

Bước 3: Hiển thị kết quả

- Sau khi quá trình chấm điểm hoàn tất, hệ thống sẽ hiển thị kết quả bao gồm: mã sinh viên, mã đề thi, và điểm số.
- Đồng thời, ba hình ảnh sẽ được hiển thị:
    1. Ảnh bài thi gốc được tải lên.
    2. Ảnh đã qua xử lý.
    3. Ảnh chấm điểm, trong đó các câu trả lời đúng và sai sẽ được đánh dấu.



Hình 5.1: Giao diện bắt đầu

Hình 5.2: Giao diện kết quả

Chương 6

Kết Luận ,Đánh giá và Phát Triển

Sau khi hoàn thành dự án hiện tại, chúng ta có thể mở rộng ứng dụng để hỗ trợ việc chấm điểm cho nhiều loại bài thi khác nhau, chẳng hạn như bài thi TOEIC hoặc bất kỳ bài trắc nghiệm nào khác. Điều này yêu cầu một số thay đổi và cải tiến để ứng dụng có thể đáp ứng nhu cầu đa dạng của người dùng. Dưới đây là các tính năng và cải tiến chi tiết có thể được thêm vào:

Cấu Hình Đáp Án

- **Tùy Chọn Đáp Án**: Cho phép người dùng cấu hình bộ đáp án cho từng loại bài thi cụ thể, bao gồm chọn loại đáp án như A, B, C, D, hoặc các lựa chọn khác (E, F, G, v.v.).
- **Số Lượng Câu Hỏi**: Người dùng có thể chỉ định số lượng câu hỏi trong bài thi.
- **Định Dạng Đáp Án**: Cung cấp các tùy chọn định dạng đáp án khác nhau, bao gồm các ô trắc nghiệm dạng hình tròn, vuông, hoặc các kiểu khác tùy thuộc vào thiết kế bài thi.

Giao Diện Người Dùng Tùy Chỉnh

- **Tải Lên Hình Ảnh**: Cải thiện giao diện người dùng để người dùng dễ dàng tải lên hình ảnh bài thi từ máy tính hoặc thiết bị di động.
- **Cấu Hình Cài Đặt**: Cung cấp các tùy chọn cấu hình rõ ràng và dễ sử dụng để người dùng có thể thiết lập cài đặt cho từng bài thi, như vị trí đáp án, kích thước ô đáp án, v.v.

Hỗ Trợ Nhiều Định Dạng Hình Ảnh

- **Định Dạng Hình Ảnh**: Mở rộng ứng dụng để hỗ trợ nhiều định dạng hình ảnh đầu vào, chẳng hạn như JPG, JPEG, PNG, và PDF. Điều này yêu cầu triển khai các thư viện xử lý ảnh và PDF để trích xuất hình ảnh từ các tài liệu đa dạng.

Xác Định Vị Trí Đáp Án

- **Tự Động Xác Định**: Phát triển các thuật toán để tự động xác định vị trí của các ô đáp án trên hình ảnh bài thi dựa trên các thông số cấu hình của người dùng.
- **Điều Chỉnh Vị Trí**: Cung cấp công cụ cho phép người dùng điều chỉnh và xác nhận vị trí của các ô đáp án trước khi bắt đầu quá trình chấm điểm.

Giao Diện Trực Quan

- **Giao Diện Tương Tác**: Cung cấp một giao diện người dùng trực quan để người dùng có thể dễ dàng xác nhận và điều chỉnh vị trí của các ô đáp án.
- **Xem Trước Kết Quả**: Hiển thị trước kết quả chấm điểm để người dùng có thể xem lại và chỉnh sửa trước khi lưu trữ kết quả chính thức.

Thống Kê và Lưu Trữ Kết Quả

- **Hiển Thị Kết Quả**: Hiển thị kết quả chấm điểm chi tiết cho người dùng, bao gồm điểm số và các đáp án đúng/sai.
- **Lưu Trữ Lịch Sử**: Lưu trữ lịch sử kết quả của các bài thi trước đó để người dùng có thể theo dõi và xem lại các kết quả cũ.

Bảo Mật và Quản Lý Người Dùng

- **Bảo Mật Dữ Liệu**: Đảm bảo tính riêng tư và bảo mật thông tin bài thi của người dùng bằng cách mã hóa dữ liệu và triển khai các biện pháp bảo mật.
- **Quản Lý Tài Khoản**: Cung cấp khả năng quản lý tài khoản người dùng, bao gồm đăng ký, đăng nhập, và quản lý quyền truy cập.

Hỗ Trợ Nhiều Loại Đáp Án

- **Các Loại Đáp Án**: Mở rộng khả năng hỗ trợ cho nhiều loại đáp án, từ các đáp án cơ bản (A, B, C, D) đến các kiểu đáp án phức tạp hơn như E, F, G, tùy theo yêu cầu của từng bài thi.

Chia Sẻ Kết Quả

- **Chia Sẻ Qua Email**: Cho phép người dùng gửi kết quả chấm điểm qua email trực tiếp từ ứng dụng.
- **Chia Sẻ Qua Mạng Xã Hội**: Tích hợp tính năng chia sẻ kết quả qua các mạng xã hội phổ biến.

Tích Hợp AI và Máy Học

- **Thuật Toán AI**: Cải tiến ứng dụng bằng cách sử dụng các thuật toán AI và máy học để nâng cao độ chính xác và hiệu quả của quá trình chấm điểm. Điều này có thể bao gồm việc sử dụng mô hình học sâu để nhận diện và phân tích các ô đáp án.