ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN - ĐIỆN TỬ
BỘ MÔN VIỄN THÔNG

———— * ————

BÁO CÁO THỰC TẬP TỐT NGHIỆP

VIẾT CHƯƠNG TRÌNH XOAY ẢNH VÀ DETECT FACE VỚI KHOẢNG CÁCH MẮT CỐ ĐỊNH

HỘI ĐỒNG: KHOA ĐIỆN - ĐIỆN TỬ
GVHD: Th.S ĐẶNG NGUYÊN CHÂU
—o0o—
SVTH: LÊ NGỌC LỢI (1511879)

TP. HỒ CHÍ MINH, 09/2020

Lời cảm ơn / Lời ngỏ

Để hoàn thành đề tài thực tập này, em tỏ lòng biết ơn sâu sắc đến thầy Đặng Nguyên Châu đã hướng dẫn tận tình trong suốt quá trình thực hiện.

Em chân thành cám ơn quý thầy, cô trong khoa Điện - Điện tử, trường đại học Bách Khoa thành phố Hồ Chí Minh đã tận tình truyền đạt kiến thức trong những năm em học tập ở trường. Với vốn kiến thức tích lũy được trong suốt quá trình học tập không chỉ là nền tảng cho quá trình thực hiện đề tài mà còn là hành trang để bước vào đời một cách tự tin.

Cuối cùng, em xin chúc quý thầy, cô dồi dào sức khỏe và thành công trong sự nghiệp cao quý.

Mục lục

Danh sách hình vẽ

Chương 1

Giới thiệu

1.1 Giới thiệu đề tài

Viết một chương trình xoay ảnh sao cho đường thẳng nối tâm hai mắt nằm ngang song song với trục Ox. Khoảng cách hai tâm mắt cố định ở kích thước 80 pixels, và kích thước ảnh 160x160 pixels.

1.2 Mục tiêu của đề tài

Mục tiêu của đề tài là nghiên cứu, hiểu và thực hiện một số phương pháp để phát hiện mắt, trích tọa độ tâm, xoay mắt, thay đổi tỷ lệ ảnh, phát hiện gương mặt và cắt ảnh.

Một số vấn đề đặt ra:

- Làm thế nào để giải quyết bài toán trên?
- Cách tiếp cận như thế nào?
- Những đã đã và hiện đang được sử dụng?
- Hướng cải tiến?

Như vậy để thực hiện theo đúng mục tiêu của đề tài cần xác định một số công việc phải giải quyết như sau:

- Trước tiên phải tìm hiểu những thuật toán phát hiện mắt đã và đang được sử dụng hiện nay xem thuật toán nào có thể phát hiện chính xác hơn, phát hiện được nhiều góc ảnh hơn để từ đó áp dụng vào bài toán của mình.
- Tìm hiểu các hàm tìm tâm ảnh, xoay ảnh, thay đổi tỷ lệ ảnh với thư viện OpenCv và PIL của python.
- Lựa chọn thuật toán phù hợp với bộ dữ liệu
- Lên kế hoạch hiện thực, phát triển chương trình và kiểm thử.

Chương 2

Kiến thức nền tảng

2.1 Tổng quan về google colab

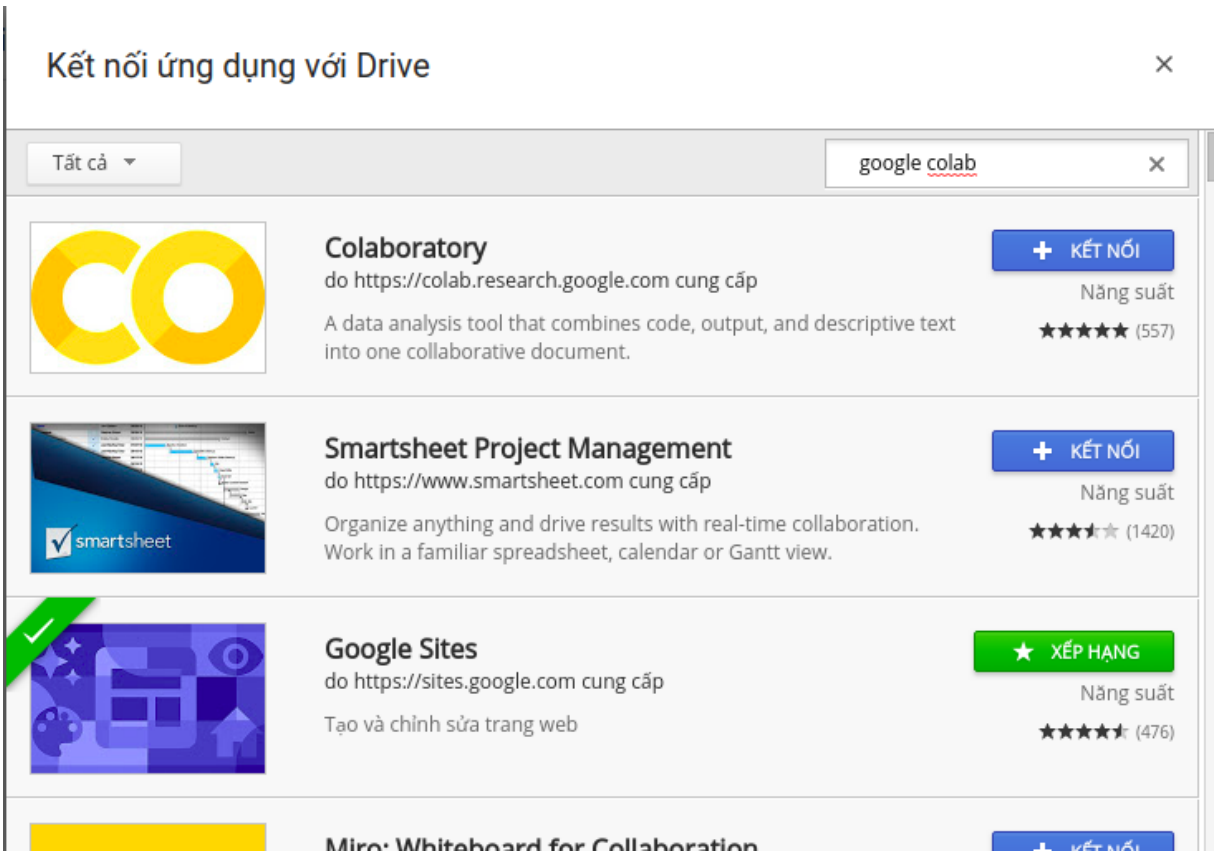
2.1.1 Giới thiệu về google colab

Machine Learning/Deep Learning đang phát triển với tốc độ rất nhanh. Để viết một chương trình sử dụng framework về Deep Learning như TensorFlow, Keras hay Pytorch, em có thể sử dụng bất kì Python IDE nào như PyCharm, Jupyter Notebook hay Gedit. Tuy nhiên, do những thuật toán Machine Learning/Deep Learning yêu cầu hệ thống phải có tốc độ và khả năng xử lý cao (thông thường dựa trên GPU), mà máy tính của em thì không được trang bị GPU. Rất nhiều người học và nhà nghiên cứu chọn giải pháp là thuê những dịch vụ tính toán trên AWS. Từ đó, Google cho ra đời một dịch vụ hoàn toàn miễn phí dành cho cộng đồng nghiên cứu AI, phát triển các ứng dụng Deep Learning bằng việc cung cấp GPU và TPU miễn phí - Đó là Google Colab. Google colab là một dich vụ đám mây miễn phí, hiện nay có hỗ trợ GPU (Tesla K80) và TPU (TPUv2). Do được phát triển dựa trên Jupiter Notebook nên việc sử dụng Google Colab cũng tương tự như việc sử dụng Jupyter Notebook. Google Colab là một công cụ lý tưởng để rèn luyện kĩ năng lập trình với ngôn ngữ Python thông qua các thư viện của deep learning. Google Colab cài đặt sẵn những thư viện rất phổ biến trong nghiên cứu Deep Learning như PyTorch, TensorFlow, Keras và OpenCV.

2.1.2 Thiết lập trên Colab

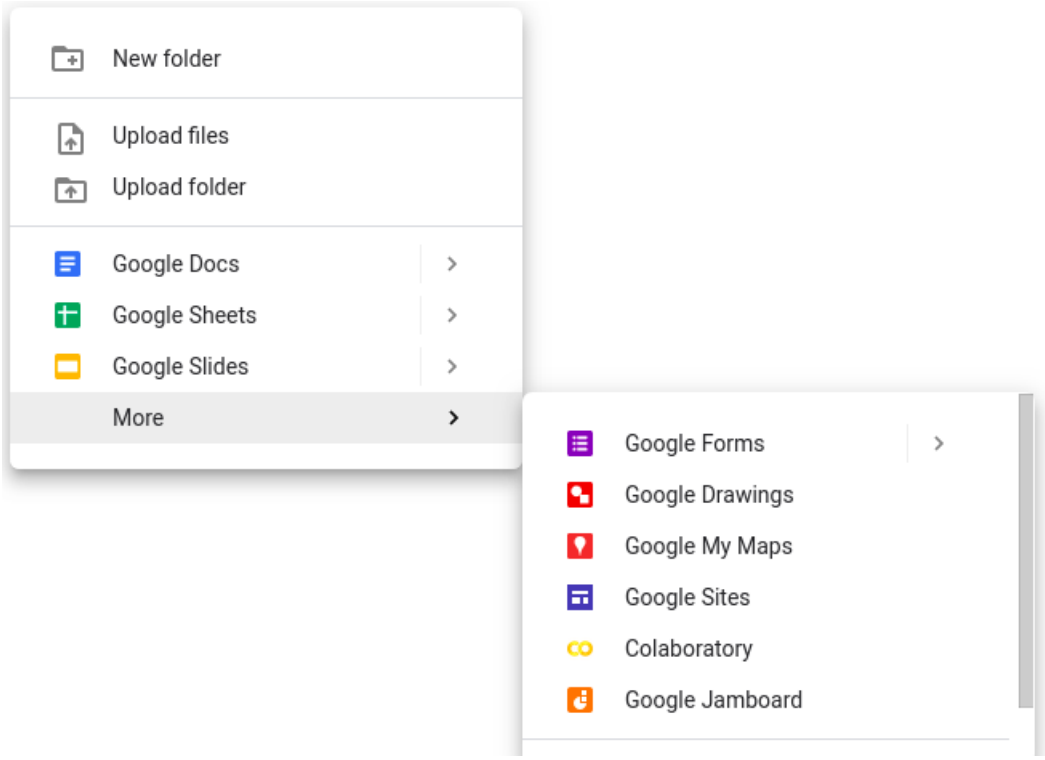
Tạo notebook

- Vào Drive, tạo một thư mục bất kì. Ví dụ như AICOLAB
- Click chuột phải, rồi chọn More –> Connect more apps. Sau đó search Google Colab và chọn Connect.



Hình 2.1: Cài Colab trong Store

- Tiếp theo, chúng ta chọn biểu tượng của Colaboratory



Hình 2.2: Kết nối Colab với Drive

Để sử dụng GPU nhấp vào “Runtime” trên thanh công cụ, chọn “Change runtime type” và chọn GPU trong “Hardware Accelerator”. Tùy theo từng thời điểm google sẽ cung cấp dung lượng cho thiết bị GPU. Để kiểm tra thông tinh dung lượng được google cung dùng câu lệnh:

```
[ ]  gpu_info = !nvidia-smi
     gpu_info = '\n'.join(gpu_info)
     if gpu_info.find('failed') >= 0:
       print('Select the Runtime > "Change runtime type" menu to enable a GPU accelerator, ')
       print('and then re-execute this cell.')
     else:
       print(gpu_info)
```

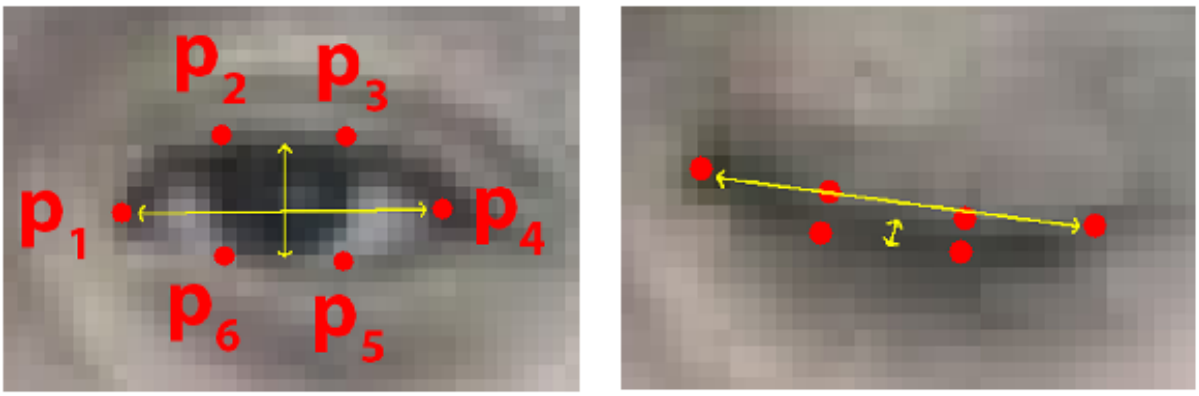
```
Tue Jul 14 01:06:57 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.51.05    Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   73C    P8    35W / 149W |      0MiB / 11441MiB |      0%      Default |
|                               |                      |                 ERR! |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

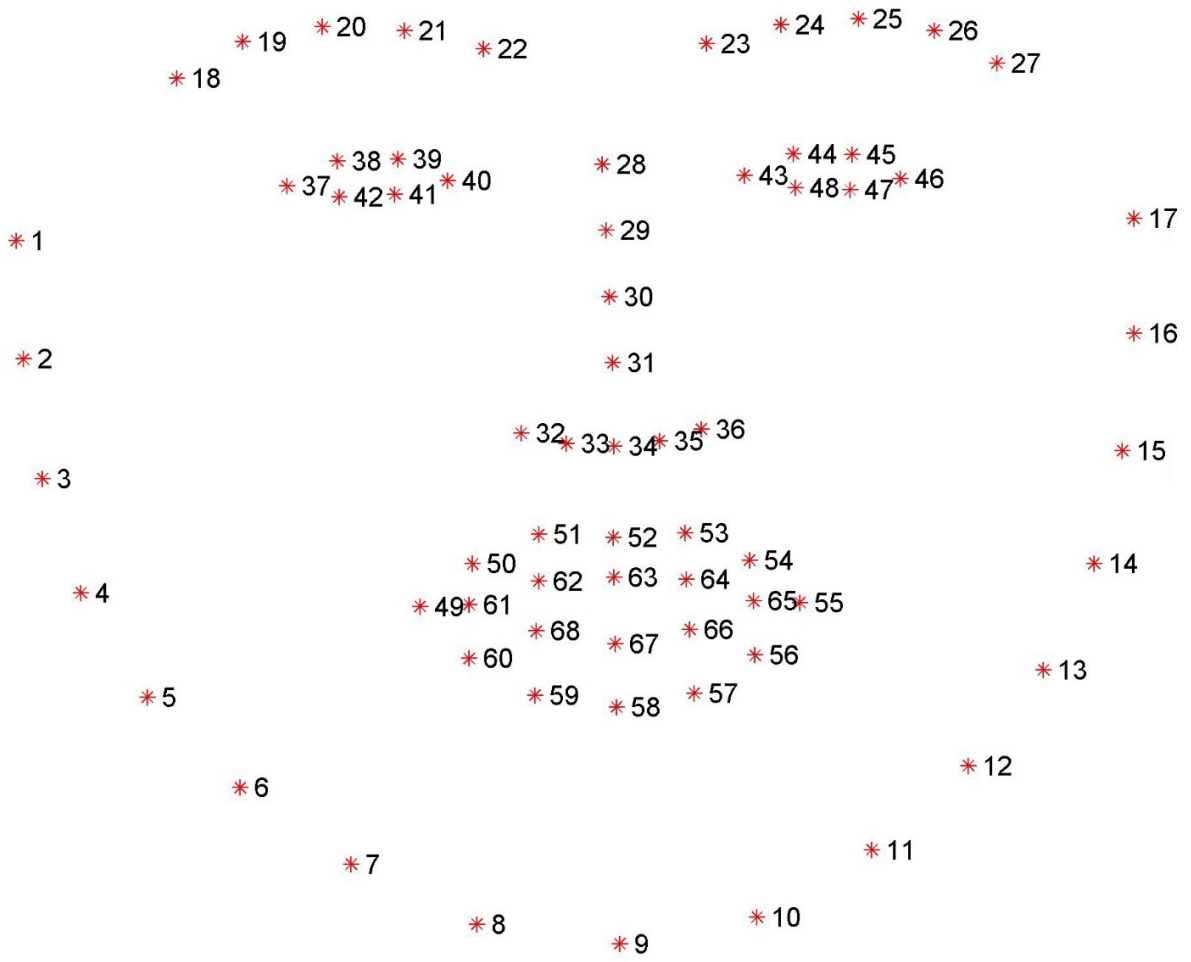
Hình 2.3: Xem GPU được hỗ trợ trong Colab

2.2 Thuật Toán Phát Hiện Mắt

Ban đầu em dung Haar Cascade để phát hiện gương mặt và mắt, mặc dù thuật toán này chạy tốt đối với những ảnh tải từ trên mạng xuống như lại không làm việc tốt trên tập dữ liệu của thầy nên em đã chuyển sang thuật toán khác là facial landmark detector bên trong thư viện dlib. Thuật toán này được sử dụng để ước lượng vị trí 68 điểm tọa độ (x,y) ánh xạ cấu trúc của gương mặt.



Hình 2.4: Minh họa thuật toán có thể phát hiện cả mắt nhắm lẫn mở

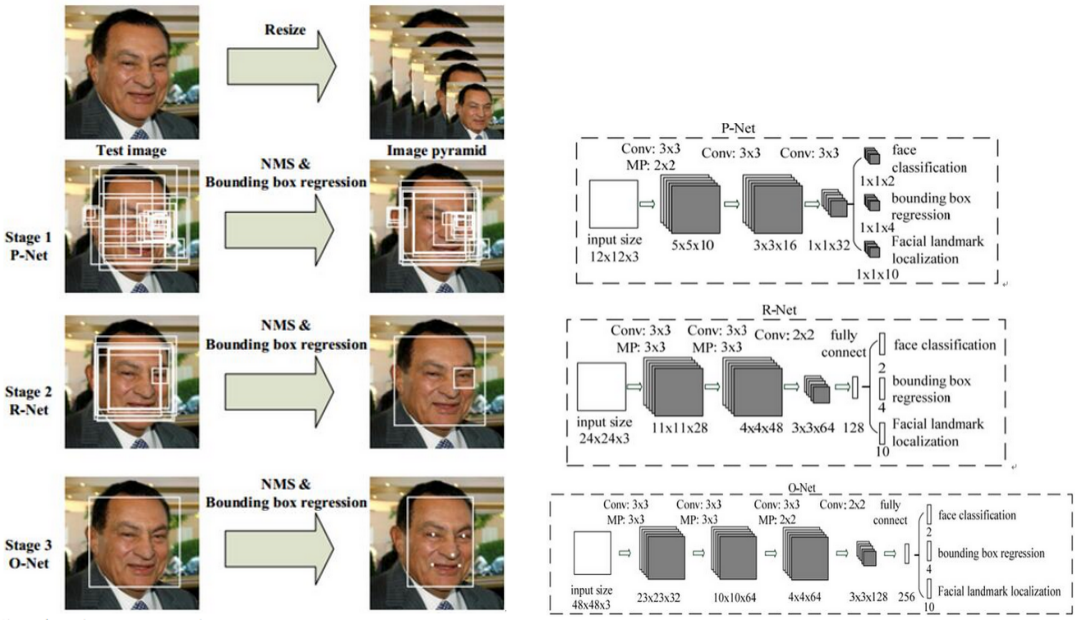


Hình 2.5: Minh họa 68 điểm tọa độ trên gương mặt từ bộ dữ liệu iBUB 300-W mà dlib đã được huấn luyện

2.3 Thuật toán phát hiện gương mặt

2.3.1 MTCNN

Em sử dụng thuật toán MTCNN (Multi-Task Convolutional Neural Network) cho việc phát hiện gương mặt, tức là tìm và trích xuất gương mặt từ ảnh. Đây là một mô hình học sâu hiện đại trong việc phát hiện gương mặt được công bố vào năm 2016.



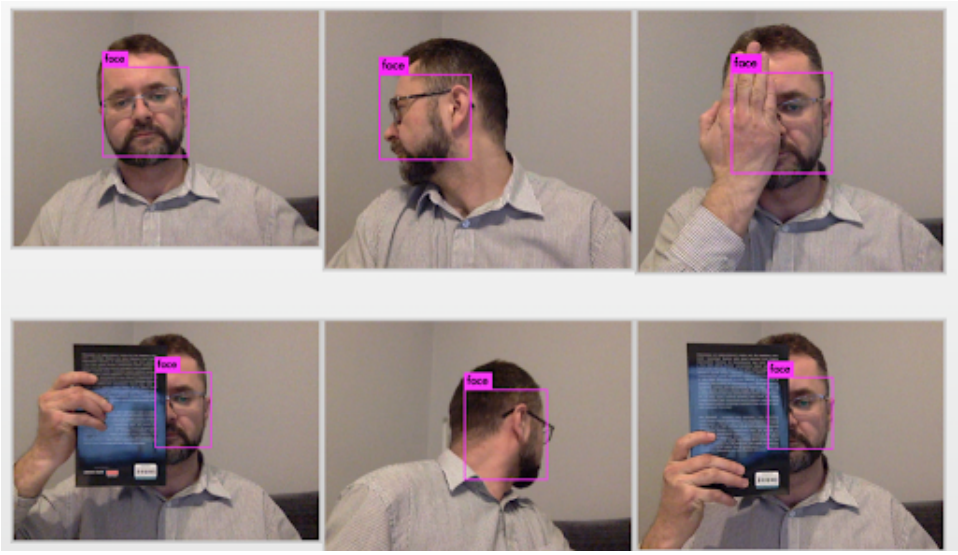
(a) Ảnh thật (b) Cấu trúc

Hình 2.6: Pipline của cascaded framework bao gồm 3 giai đoạn multi-task deep convolutional network

2.3.2 Yolo-Face

Sau khi dùng MTCNN để detect gương mặt thì còn một số ảnh thuật toán không detect được, vì thế em đã tìm hiểu và áp dụng thuật toán Yolo-Face, thuật toán này mặc dù không detect được tất cả ảnh lỗi, nhưng cũng đã tối thiểu hóa đi số lượng ảnh lỗi, từ đó nâng cao độ chính xác hơn.

Thuật toán này áp dụng một mạng nơ-ron duy nhất cho toàn bộ hình ảnh. Mạng này chia hình ảnh thành các vùng và dự đoán các bounding box và xác suất cho từng vùng. Các bounding box này được tính theo xác suất dự đoán.



Hình 2.7: Minh họa thuật toán Yolo-Face

Chương 3

Tiến hành thực hiện

3.1 Hướng tiếp cận

Với bộ dữ liệu đầu vào là 65.000 ảnh được chia làm nhiều thư mục, việc đầu tiên là em đưa ảnh lên google drive với 4 thư mục con như sau:

- Original: thư mục dữ liệu gốc
- Result: thư mục kết quả sau khi chuyển đổi
- err-rects: thư mục lỗi do không nhận diện được mắt
- err-face: thư mục lỗi do không nhận diện được gương mặt

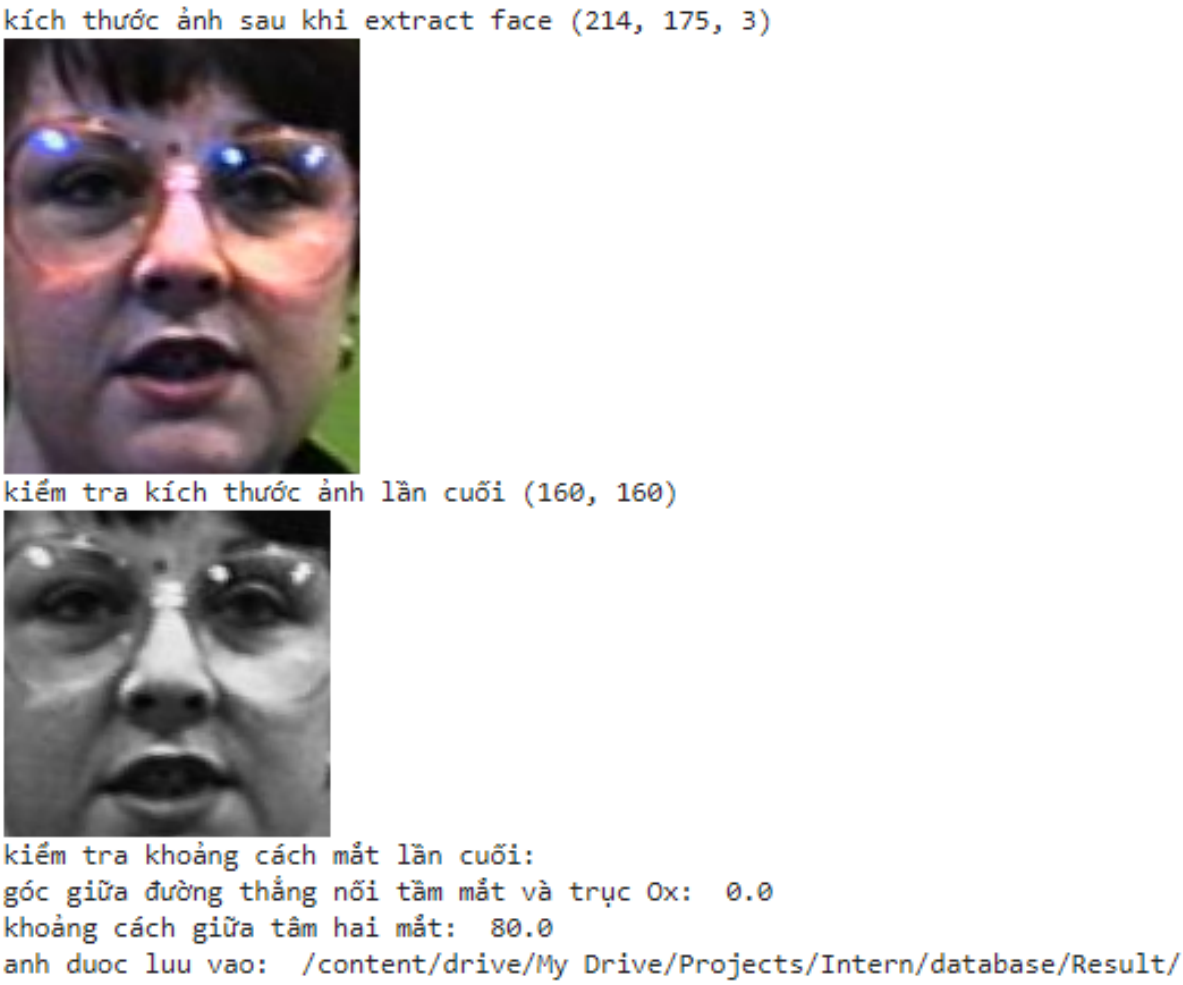
Đầu tiên em sẽ nhận dạng mắt –> trích tọa độ tâm của mắt –> tính góc tạo bởi đường thẳng nối hai tâm của mắt và trục Ox –> tính khoảng cách giữa hai tâm mắt.

Hình 3.1: Thông số của ảnh gốc trước khi xử lý

Tiếp theo em sẽ xoay ảnh với góc angle đã tính ở trên, thay đổi tỷ lệ ảnh bằng cách rescale ảnh để đưa về khoảng cách 80 pixels.

Hình 3.2: Sau khi dùng dlib xử lý một phần

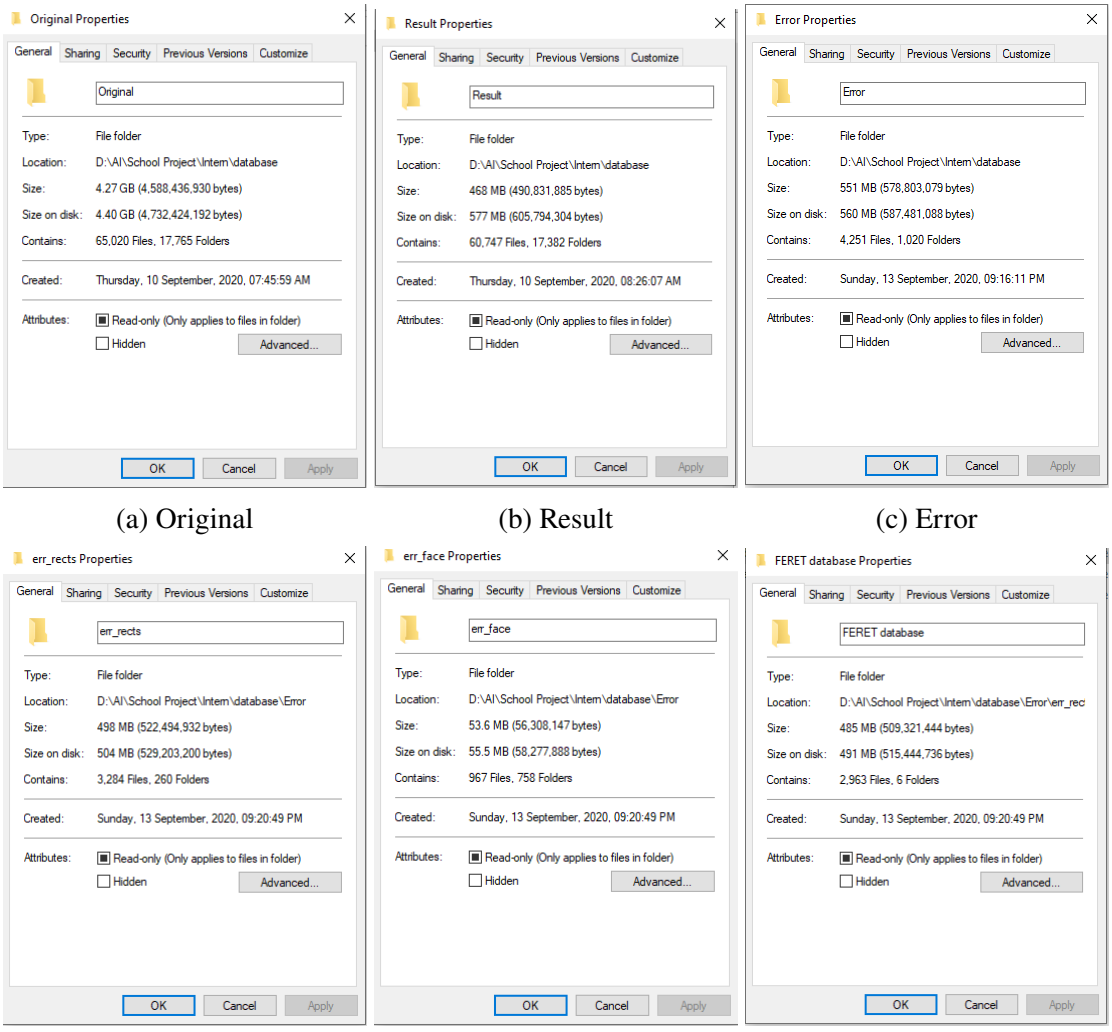
Tiếp theo em sẽ dùng MTCNN để phát hiện, trích tọa độ của gương mặt, và cắt face ra với width và height đã trích ra ở trên. Sau đó em crop ảnh với kích thước 160x160. Cuối cùng em gọi hàm tính khoảng cách hai mắt, góc tạo bởi 2 mắt và Ox, kích thước ảnh lần cuối. Và kết quả cho ra hoàn toàn như mong muốn.



Hình 3.3: Kết quả cuối cùng sau khi xử lý xong

3.2 Kết quả đạt được

Với bộ dữ liệu gốc khoảng 65.000 ảnh, kết quả cho thấy khá tốt khi có tới hơn 60.800 ảnh cho kết quả chính xác và khoảng 4200 ảnh lỗi với tỷ lệ chính xác khoảng 93,5%. Trong số các ảnh lỗi thì ảnh lỗi do không detect được mắt ở thư mục FERET database chiếm gần 70% ảnh lỗi, sỡ dĩ ở thư mục này ảnh lỗi nhiều là vì ảnh vào hầu như là ảnh bị khuất một bên mặt nên thuật toán chỉ có thể phát hiện được một mắt, vì thế thuật toán không thể xoay ảnh đươc. Kết quả được minh họa dưới hình sau:

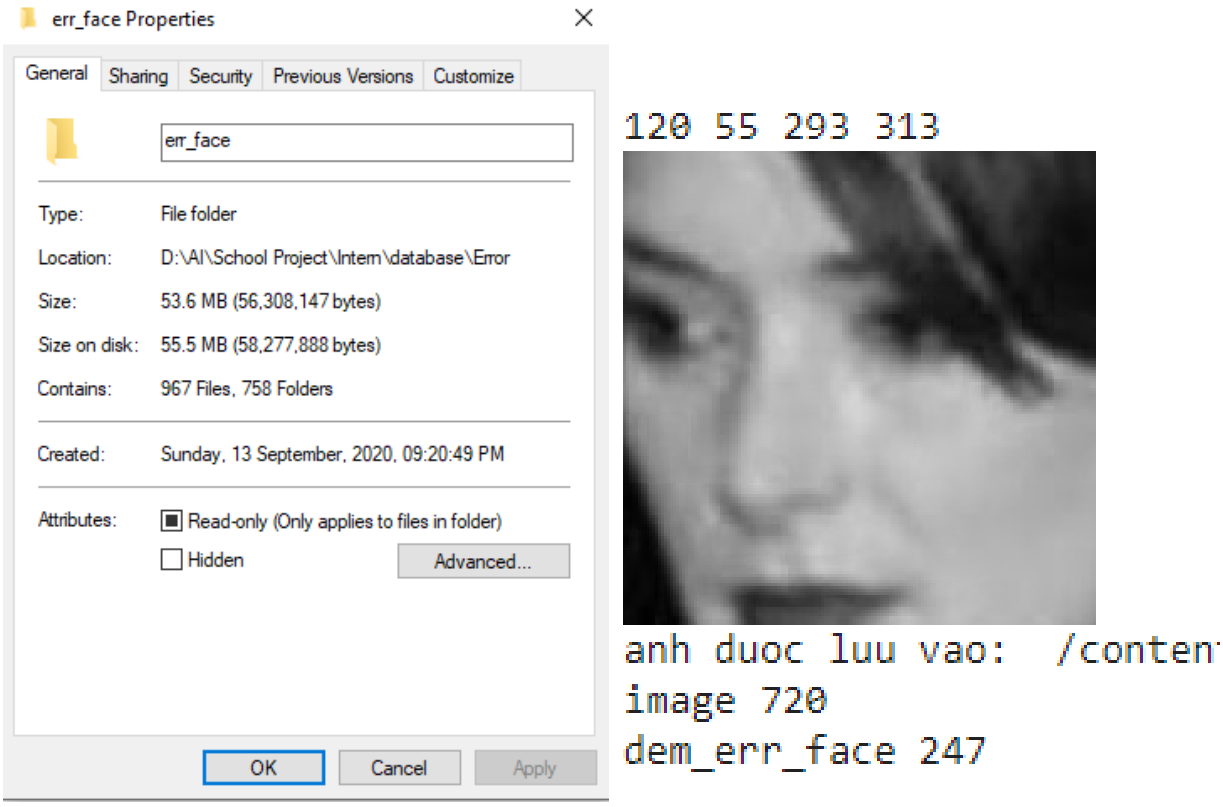


(a) Original (b) Result (c) Error

(d) Error-rects (e) Error-face (f) Error-FERET folder

Hình 3.4: So sánh kết quả

Sau khi tìm hiểu các thuật toán khác, nhận thấy có thể cải thiện những ảnh lỗi do phát hiện gương mặt có thể được cải tiến bằng cách áp dụng thuật toán Yolo-Face, vì thế em đã cho thư mục ảnh lỗi phát hiện gương mặt vào thư mục gốc và tiếp tục cải thiện kết quả.



(a) Error-face (b) Yolo-face

Hình 3.5: Kết quả sau khi dùng Yolo-Face

Với kết quả này, em thấy rằng Yolo-Face đã cải thiện được tới khoảng 66% số lượng ảnh lỗi mà MTCNN để lại, từ đó tăng tính hiệu quả hơn cho bài toán của em.

Tổng ảnh	Độ Chính xác	Ảnh Lỗi				
		Lỗi mắt		Lỗi face		
65000(100%)	60800(93,54%)	3284(5.05%)		967(1.41%)		
		FERET folder	Còn lại	MTCNN	Yolo-Facce	Cải thiện Độ chính xác
		2963(4.56%)	321(0.49%)	967(1.41%)	247(0.38%)	61520(94.57%)
				Yolo-Face đã cải thiện được tới khoảng 66% số lượng ảnh lỗi mà MTCNN để lại.		

Hình 3.6: Bang Thống Kê

Tài liệu tham khảo

[1] Tereza Soukupová, J. Cech *Real-Time Eye Blink Detection using Facial Landmarks*.
https://www.pyimagesearch.com/2017/04/24/eye-blink-detection-opencv-python-dlib

[2] Florian Schroff, Dmitry Kalenichenko, James Philbin *FaceNet: A Unified Embedding for Face Recognition and Clustering*.
https://machinelearningmastery.com/how-to-develop-a-face-recognition-system-usi

[3] Wang Yang, Zheng Jiachun *Real-time face detection based on YOLO*
https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/

[4] Trong quá trình sữa lỗi code, em có tham khảo những website sau:
https://stackoverflow.com/
https://github.com/
https://www.google.com/
https://www.kaggle.com/