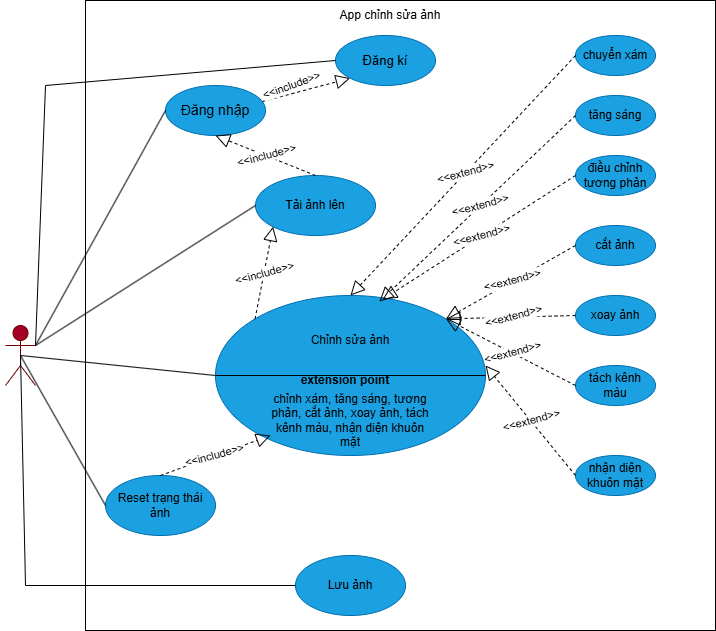
**CHƯƠNG 2: MÔ TẢ TỔNG QUAN SẢN PHẨM**

**2.1 Mục tiêu và phạm vi**

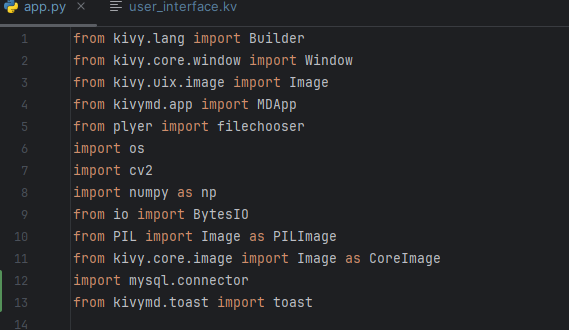
* Xây dựng một ứng dụng đơn giản chạy trên điện thoại sử dụng ở bất kì đâu cả online và offline, có khả năng xử lý ảnh cơ bản và nhận diện khuôn mặt.
* Phát triển thêm từ tính năng nhận diện khuôn mặt để khôi phục lại khuôn mặt bị mờ, anh đã cũ từ xưa thành rõ nét và dễ nhận diện hơn trong tương lai.
  1. **Người dùng và các chức năng cơ bản**
* **Người dùng sử dụng các chức năng sau:**
* Đăng kí tài khoản
* Đăng nhập
* Tải ảnh lên
* Chỉnh sửa ảnh:
  + Chuyển về ảnh xám
  + Điều chỉnh độ sáng
  + Điều chỉnh độ tương phản
  + Cắt ảnh (cắt vùng thừa ở xung quanh)
  + Xoay ảnh
  + Tách 3 kênh màu
  + Nhận diện khuôn mặt trong ảnh
* Reset trạng thái ảnh
* Lưu ảnh
  1. **Biểu đồ usecase**

****

Người dùng

Hình 2.1 biểu đồ usecase

* 1. **Môi trường và các công cụ, thư viện sử dụng:**
* Ngôn ngữ: Python
* Framework: Kivy
* Hệ quản trị cơ sở dữ liệu: MySQL
* Công cụ hỗ trợ lập trình: Visual Studio Code
* Các thư viện sử dụng:
* OpenCv(cv2): Xử lí ảnh
* KivyMD: Thiết kế giao diện người dùng
* NumPy(np): Hỗ trợ tính toán ma trận và mảng, dung cùng OpenCv
* Plyer, io, os: Quản lí tập tin và dữ liệu
* MySQL Connector: Kết nối và tương tác với cơ sở dữ liệu
* Pillow: xử lí ảnh, chuyển đổi định dạng ảnh

****

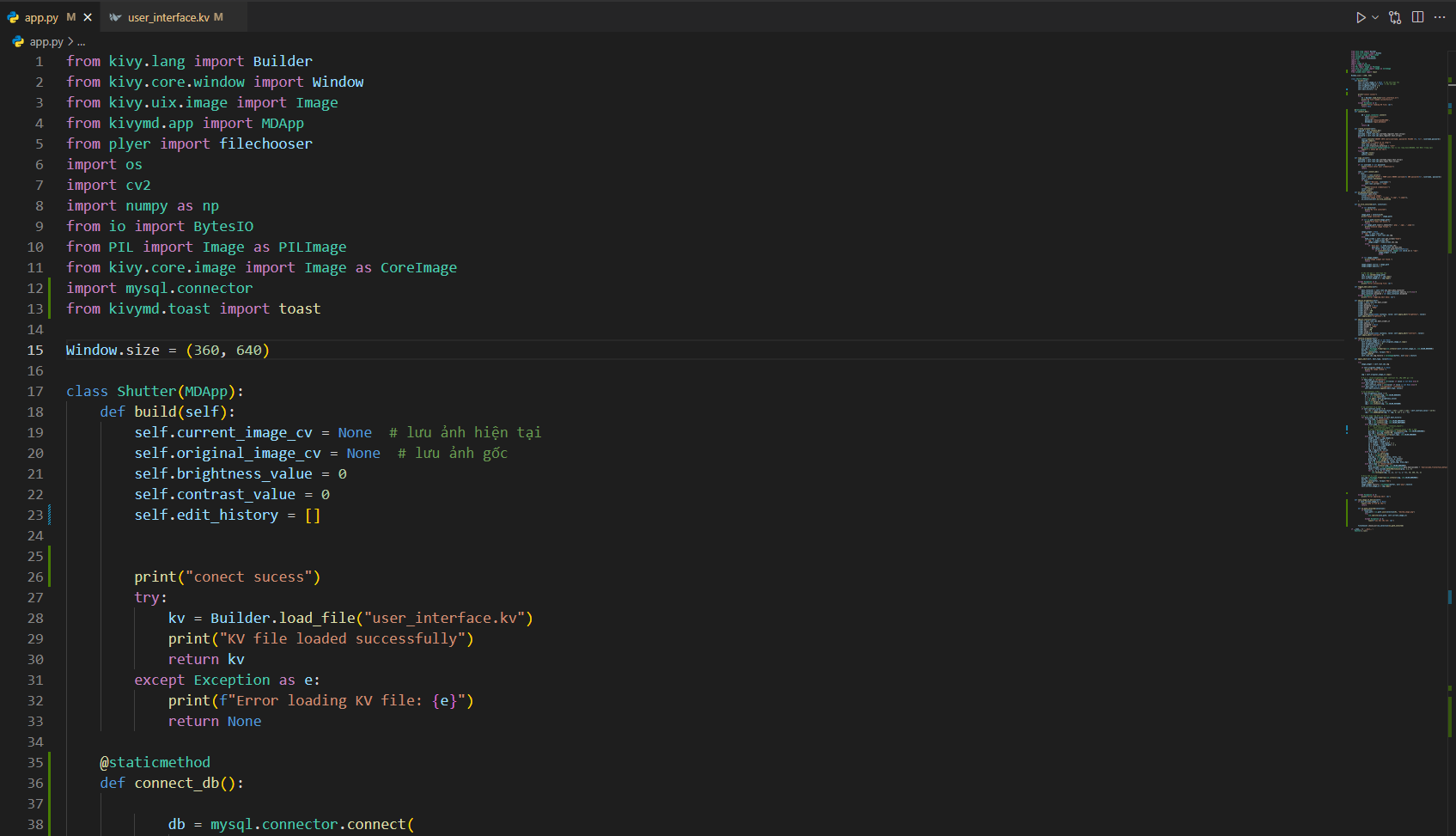
Hình 2.2 Các thư viện được import và sử dụng

* 1. **Các giao diện chính và cách tổ chức**

**2.5.1 Tổng quan về cấu trúc ứng dụng**

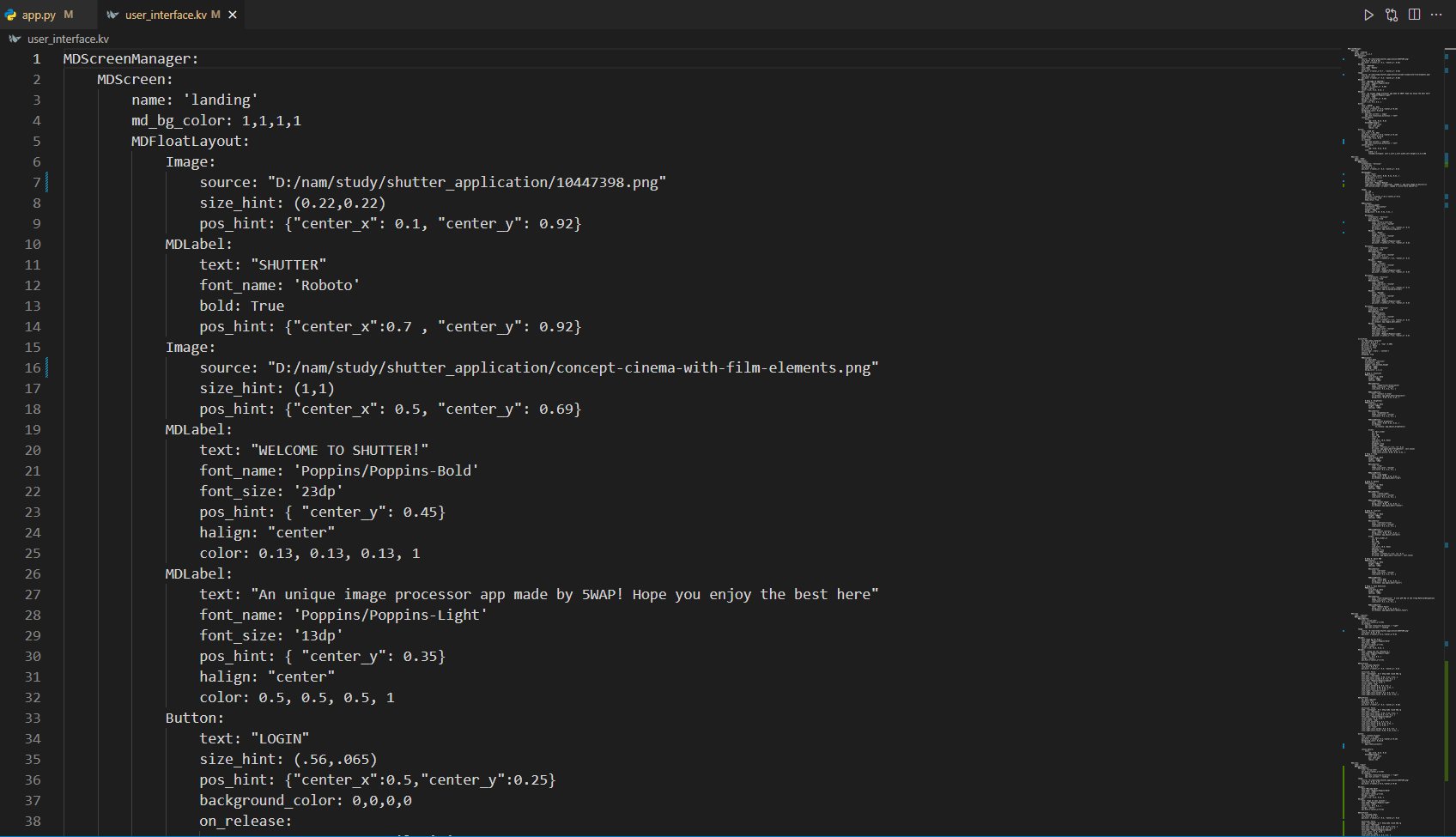
Ứng dụng chỉnh sửa ảnh “Shutter” được thiết kế với kiến trúc phân tách rõ ràng giữa logic xử lý và giao diện người dùng. Toàn bộ mã nguồn được tổ chức thành hai file chính:

* **File logic và xử lí**: File chính của ứng dụng, chịu trách nhiệm nhập các thư viện cần thiết, định nghĩa logic cốt lõi, và điều phối các chức năng như quản lý giao diện, xử lý ảnh, và tương tác với cơ sở dữ liệu MySQL.



Hình 2.3 Tổng quan cấu trúc file app.py

* **File giao diện ứng dụng**: File định nghĩa giao diện người dùng theo cú pháp Kivy, sử dụng các widget để xây dựng bố cục giao diện theo phong cách Material Design.



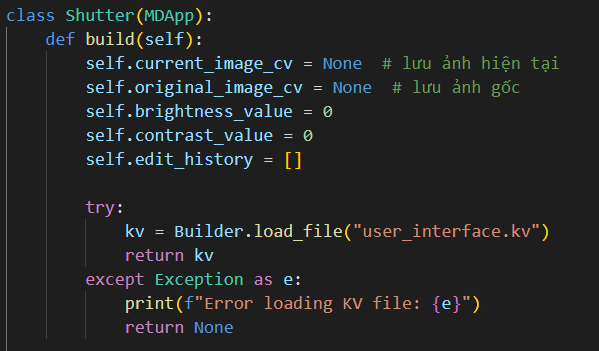
Hình 2.3 Tổng quan cấu trúc file user\_interface.kv

Sự phân tách này giúp code dễ bảo trì, mở rộng, và đảm bảo tính modular trong thiết kế ứng dụng.

**2.5.2 File logic và xử lí ảnh**

* File logic và xử lí ảnh *app.py* là **điểm khởi chạy** (entry point) của ứng dụng, chứa lớp chính Shutter (kế thừa từ MDApp của KivyMD). File này đóng vai trò trung tâm, tích hợp các thư viện và quản lý toàn bộ logic của ứng dụng. Cụ thể:

**2.5.2.1 Khởi tạo giao diện (build).**

****

Hình 2.4 Hàm khởi chạy của lớp chính

* Đầu tiên hàm def build(self) sẽ nhúng file chứa giao diện vào file chính để hiển thị thông qua phương thức:

kv = Builder.load\_file("user\_interface.kv")

* Ngoài ra hàm này còn để định nghĩa và setup các đối tượng để lưu trữ ảnh gốc lúc người dùng tải lên và ảnh hiện tại đang trong quá trình chỉnh sửa cùng lịch sử áp dụng các chức năng

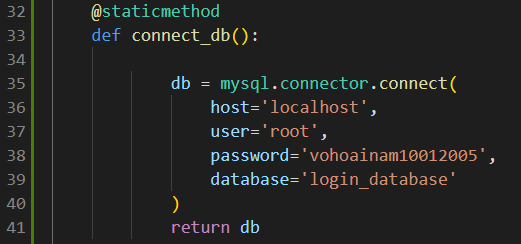
self.current\_image\_cv = None

self.original\_image\_cv = None

self.edit\_history = []

**2.5.2.2 Kết nối cơ sở dữ liệu MySQL**

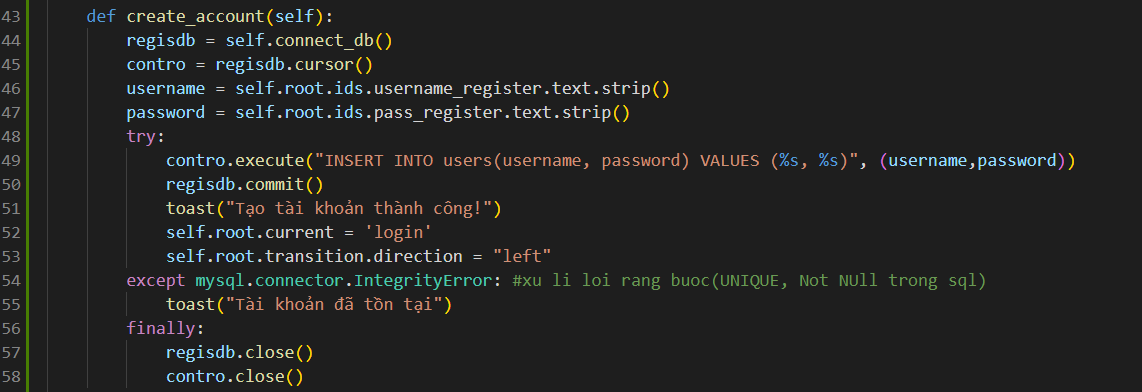
- Kêt nối ứng dụng với cơ sở dữ liệu thông qua hàm connect\_db và thư viện mysql.connector



Hình 2.5 Kết nối database

**2.5.2.3 Quản lý tài khoản người dùng**

* **Đăng kí tài khoản:**

****

Hình 2.6 Hàm đăng kí tài khoản

* Khi người dùng ấn nút Create Account trên màn hình thì hàm sẽ khởi chạy và kết nối với cơ sở dữ liệu để lưu thông tin đăng kí vào table thông qua phương thức:

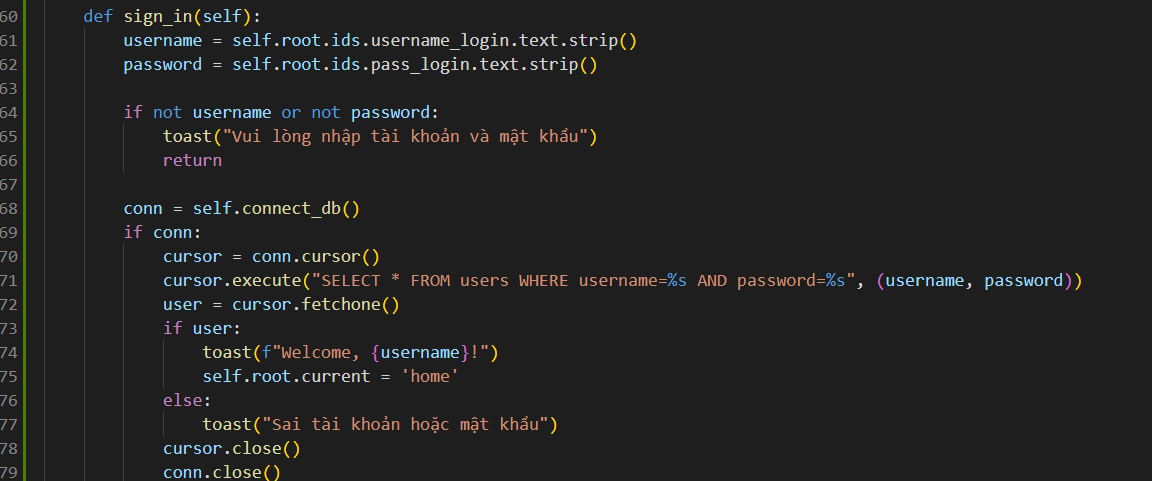
contro.execute("INSERT INTO users(username, password) VALUES (%s, %s)", (username,password))

regisdb.commit()

* Hàm sẽ tạo ra 1 biến con trỏ để thực thi lệnh query trên MySQL Workbech lưu tài khoản và mật khẩu vào table
* Dòng lệnh dưới để xử lí ngoại lệ khi người dùng để trống 1 trong 2 trường tài khoản hoặc mật khẩu và khi người dùng đăng kí trùng tài khoản

except mysql.connector.IntegrityError:

* **Đăng nhập:**



Hình 2.7 Hàm đăng nhập

* Tương tự như hàm đăng kí thì hàm đăng nhập cũng sẽ tạo ra 1 con trỏ để thực thi lệnh query để truy vấn dữ liệu từ bảng users những tài khoản trùng khớp thông qua phương thức:

cursor.execute("SELECT \* FROM users WHERE username=%s AND password=%s", (username, password))

user = cursor.fetchone()

* Nếu trùng khớp thì chuyển người dùng tới trang home để tiếp tục chỉnh sửa

**2.5.2.4 Tải và hiển thị ảnh**

* Hàm cho phép người dùng tải ảnh lên từ thiết bị với filechooser, lưu ảnh được chọn vào biến image\_path rồi đọc ảnh để chỉnh sửa bằng opencv qua lệnh:

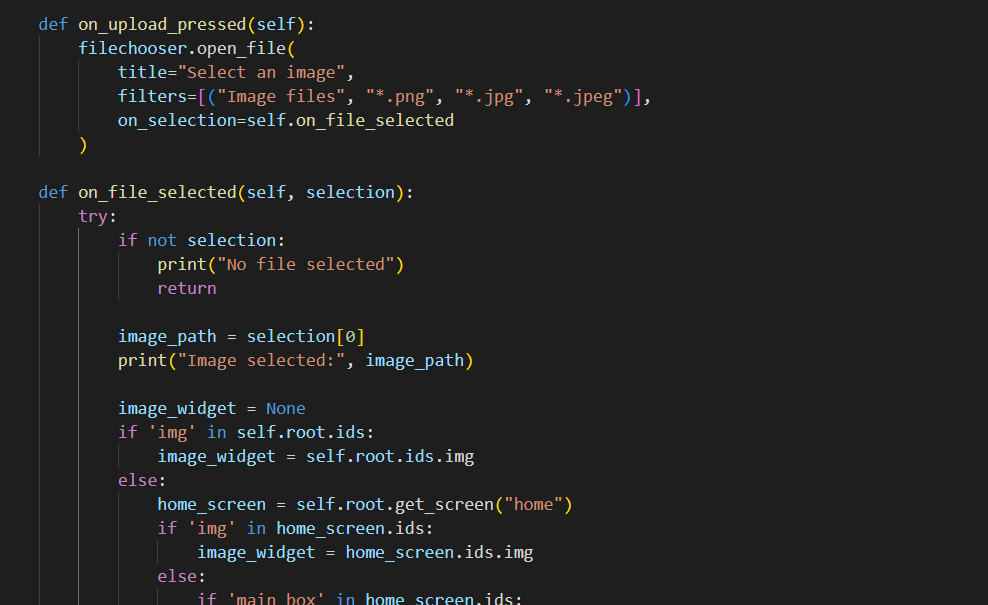
image\_widget.source = image\_path

image\_widget.opacity = 1

img = cv2.imread(image\_path)

self.original\_image\_cv = img.copy()

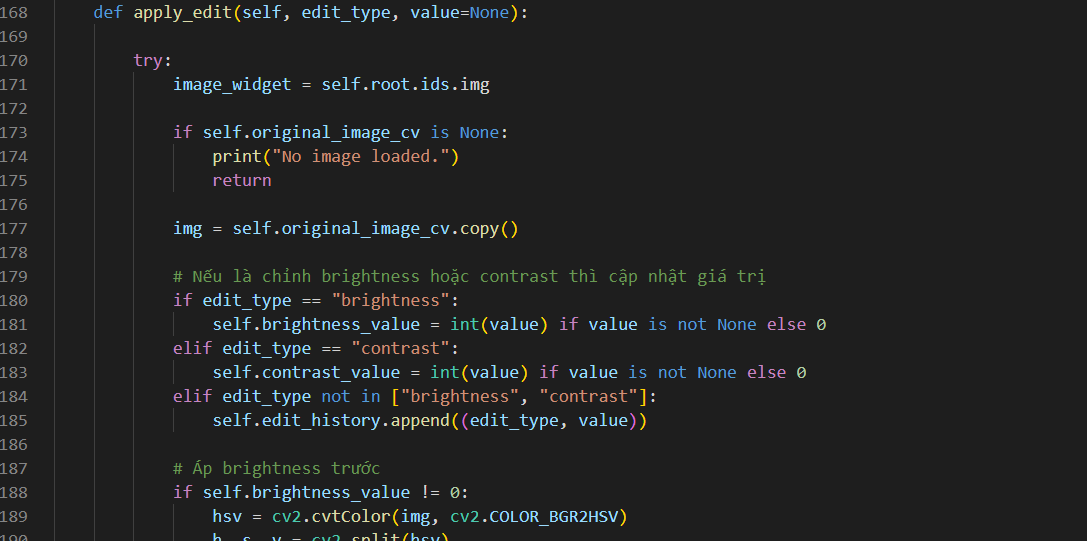
self.current\_image\_cv = img.copy()

****

Hình 2.8 Hàm đọc và hiển thị ảnh

**2.5.2.5 Chỉnh sửa ảnh với các hiệu ứng**

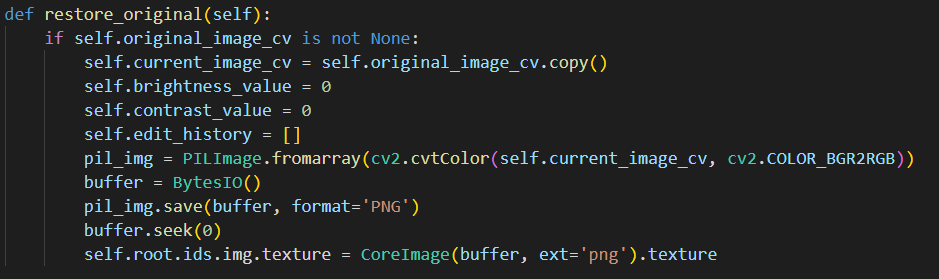
- Tất cả các hiệu ứng áp dụng vào ảnh sẽ được quản lí tronng hàm apply\_edit dựa vào parameter tương ứng được truyền vào đối với từng chức năng:



Hình 2.9 Hàm áp dụng hiệu ứng

**2.5.2.6 Lưu và khôi phục ảnh**

* **Khôi phục ảnh gốc:**



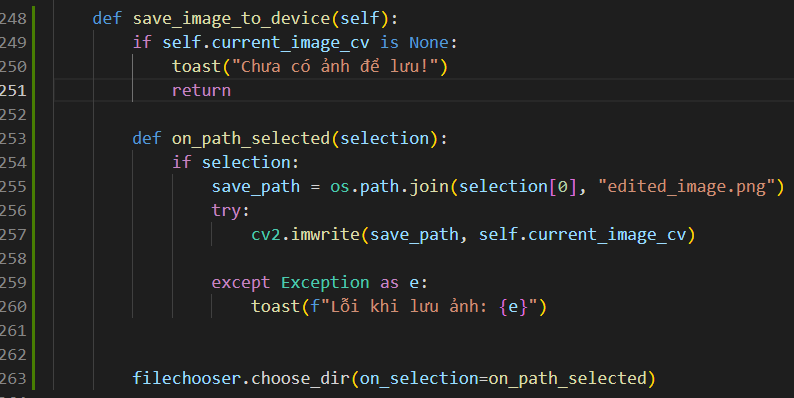
Hình 2.10 Hàm khôi phục ảnh gốc

* Khi người dùng ấn nút reset trên màn hình thì ứng dụng sẽ cập nhật lại ảnh ban đầu lúc chưa chỉnh sửa với các dòng lệnh để điều chỉnh độ sáng và độ tương phản về giá trị “0” và đồng thời cũng reset lại biến edit\_history thành mảng rỗng
* Sau đó ảnh gốc sẽ được cập nhật vào bộ nhớ dưới dạng bitstream và mã hoá theo PNG bằng:

buffer = BytesIO()

pil\_img.save(buffer, format='PNG')

* **Lưu ảnh:**

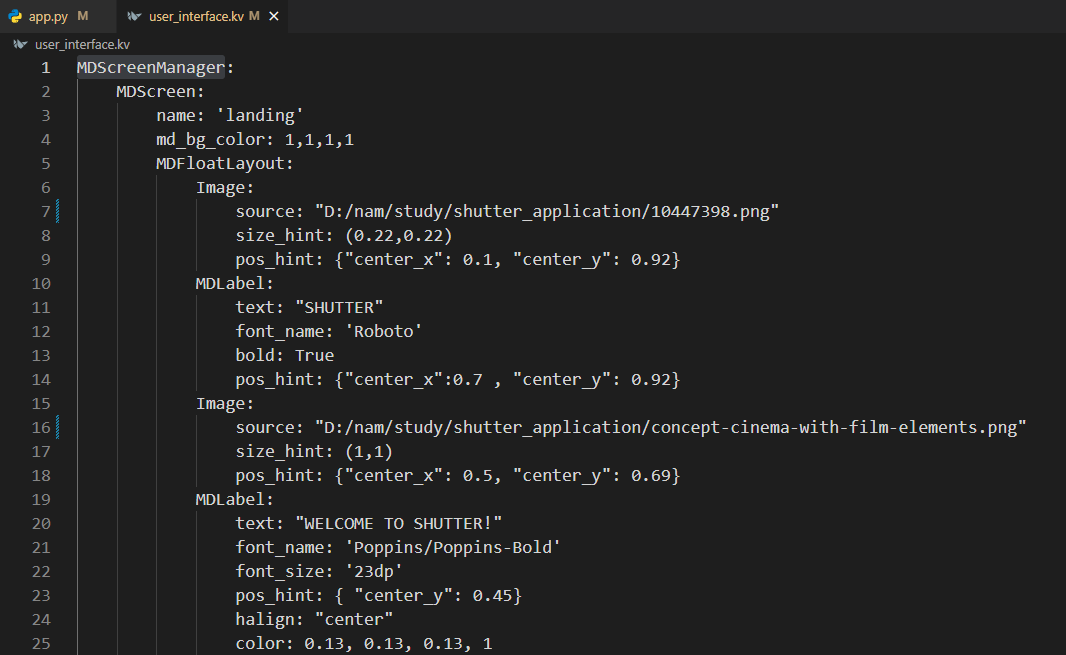


Hình 2.11 Hàm lưu ảnh về thiết bị

* Khi người dùng ấn nút lưu ảnh trên màn hình thì ứng dụng sẽ lưu trực tiếp từ OpenCv bằng mảng numpy thành các tệp ảnh mà không cần phải chuyển sang Pilow để hiển thị ảnh nên sẽ tăng hiệu suất

**2.5.3 File giao diện ứng dụng**

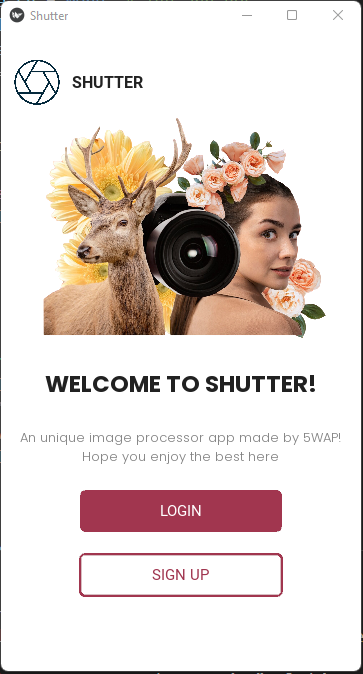
* File định nghĩa giao diện người dùng theo cú pháp Kivy, sử dụng các widget để xây dựng bố cục giao diện theo phong cách Material Design.
* Ứng dụng được chia thành 4 trang chính và được quản lí bằng MDScreenManager, mỗi trang sẽ có 1 id riêng để phân biệt cũng như thực thi logic. Mỗi trang sẽ được định nghĩa dựa trên các Block/Grid/Float Layout riêng biệt hoặc lồng nhau cùng với các widget để cấu tạo nên khung của trang:



Hình 2.12 Tổng quan cấu trúc giao diện

* **Trang chủ:**

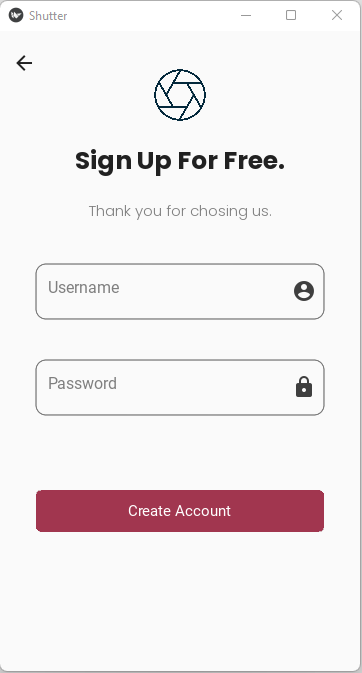
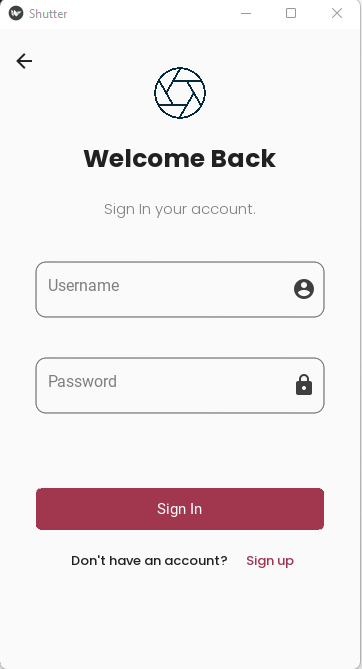
Đây là màn hình trang chủ khi người dùng khởi động ứng dụng, chứa tên và mô tả của ứng dụng và các nút để điều hướng người dùng

****

Hình 2.13 Giao diện trang chủ

* **Trang đăng nhập và đăng kí:**

Khi người dung bấm Login thì chuyển sang trang đăng nhập, có 2 trường tài khoản và mật khẩu. Nếu chưa có tài khoản thì khi click Sign up, ứng dụng điều hướng người dung sang trang đăng kí

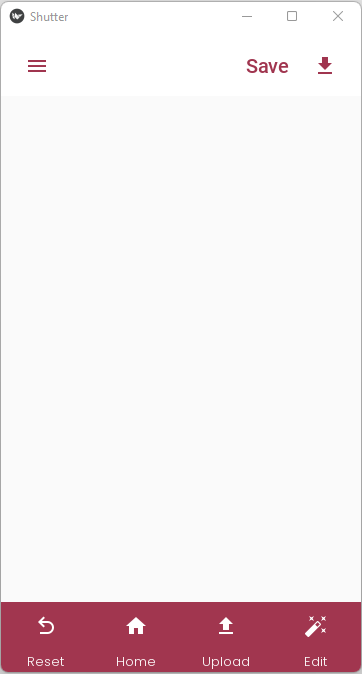


Hình 2.13 Giao diện trang đăng nhập và đăng kí

* **Trang chính:**

Giao diện trang chính gồm 2 thanh navigation bar ở trên và thanh công cụ ở dưới, ảnh người dùng tải lên sẽ được hiển thị ở khoảng trống giữa màn hình.

Nút edit giúp người dùng có thể chọn chức năng mà mình mong muốn. Khi đã chỉnh sửa xong thì nút Save ở góc trên bên phải giúp lưu ảnh về thiết bị

****

Hình 2.14 Giao diện trang chính

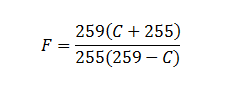
* 1. **Chi tiết các chức năng**
     1. **Điều chỉnh độ tương phản**
* Khi người dùng tải ảnh lên và chọn chức năng điều chỉnh độ tương phản, cạnh nút chức năng sẽ hiện lên 1 thanh giúp điều chỉnh chỉ số
* Chức năng sử dụng thuật toán trộn tuyến tính Linear Blending với hệ số tăng cường f qua công thức:

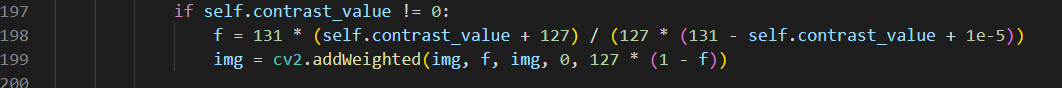
output=α⋅img+β⋅img+γ

Ở đây:

* alpha = f
* beta = 0 (ảnh thứ hai không đóng góp)
* gamma = 127 \* (1 - f) là một hằng số được cộng vào tất cả các pixel làm thay đổi độ tương phản của ảnh

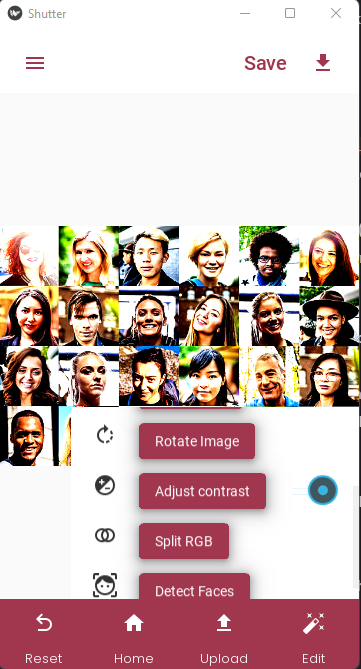
Ta tính được hệ số tăng cường f qua công thức:





* Áp dụng vào ứng dụng, ta sử dụng giá trị giảm nhẹ 131 và 127 để thay cho 259 và 255 như ở trên công thức để phù hợp hơn với công thức 8bit của OpenCV và tránh điều chỉnh độ tương phản quá gắt gao
* Biến C (contrast) được nhận trực tiếp từ người dung khi kéo thanh trượt điều chỉnh độ tương phản
* Ở đây ta cộng them với hằng số rất nhỏ là 1e-5 để tránh trường hợp mẫu là 0 trong phép chia

f = 131 \* (self.contrast\_value + 127) / (127 \* (131 - self.contrast\_value + 1e-5))

****

Hình 2.16 Ảnh sau khi chỉnh tương phản

* + 1. **Chuyển về ảnh xám**

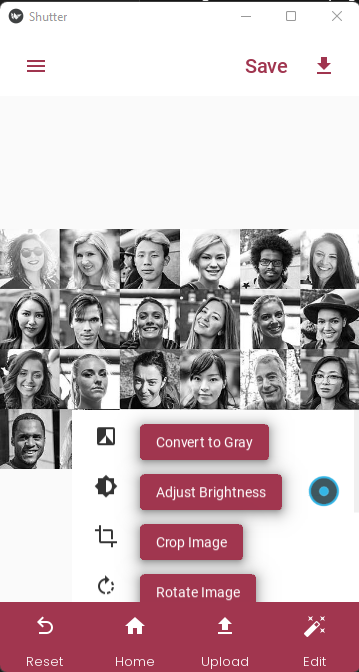
if hist\_type == "grayscale":

  img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

img = cv2.cvtColor(img, cv2.COLOR\_GRAY2BGR)

* OpenCV lưu ảnh màu ở định dạng BGR (Blue, Green, Red).
* Khi dùng cv2.COLOR\_BGR2GRAY, nó sẽ chuyển ảnh màu thành ảnh xám bằng cách áp dụng công thức sau (chuẩn quốc tế ITU-R BT.601), Tức là ảnh xám được tính bằng **trọng số ánh sáng** của từng kênh màu:

Y = 0.114 \* B + 0.587 \* G + 0.299 \* R

****

Hình 2.17 Ảnh sau khi chuyển xám

* + 1. **Điều chỉnh độ sáng**
* Hàm tăng hoặc giảm độ sáng của ảnh bằng cách tác động trực tiếp lên kênh màu V( value hay brightness) trong hệ màu HSV

if self.brightness\_value != 0:

hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

h, s, v = cv2.split(hsv)

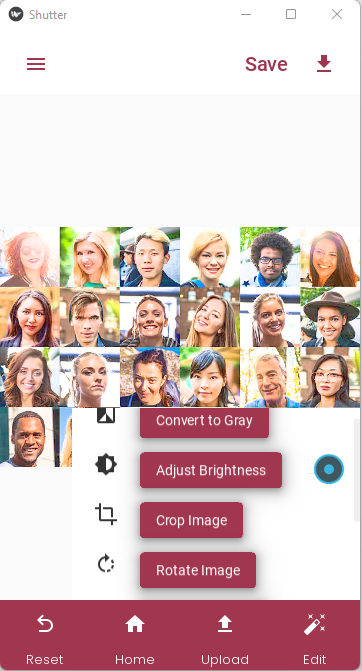
v = cv2.add(v, self.brightness\_value)

v = np.clip(v, 0, 255)

img = cv2.merge((h, s, v))

img = cv2.cvtColor(img, cv2.COLOR\_HSV2BGR)

* Đầu tiên sẽ chuyển từ BRG sang HSV bằng cv2.cvtColor rồi tách riêng 3 kênh. Sau đó cộng trực tiếp vào kênh V giá trị mà người dùng điều chỉnh trên thanh trượt. Cuối cùng kết hợp lại 3 kênh bằng cv2.merge rồi chuyên lại sang BRG để hiển thị

****

Hình 2.17 Ảnh sau khi tăng sáng

* + 1. **Cắt ảnh theo giá trị cố định**

1. elif hist\_type == "crop":
2. height, width = img.shape[:2]
3. crop\_width = int(width \* 0.9)
4. crop\_height = int(height \* 0.9)
5. x1 = (width - crop\_width) // 2
6. y1 = (height - crop\_height) // 2
7. x2 = x1 + crop\_width
8. y2 = y1 + crop\_height
9. img = img[y1:y2, x1:x2]

* Khi người dùng ấn nút Crop ảnh, ứng dụng sẽ giữ lại 90% bức ảnh và chỉ cắt xung quanh
* Biến crop\_width và crop\_height để xác định cần giữ lại bao nhiêu, x1 để xác định cắt từ vị trí x1 mỗi bên tương tự với y1. [y1:y2,x1:x2] để giữ lại vùng ảnh sau khi cắt

**2.6.5 Xoay ảnh**

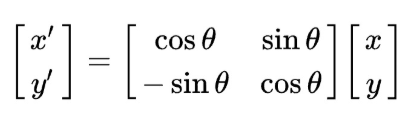
elif hist\_type == "rotate":

pil\_img=PILImage.fromarray(cv2.cvtColor(img,cv2.COLOR\_BGR2 RGB))

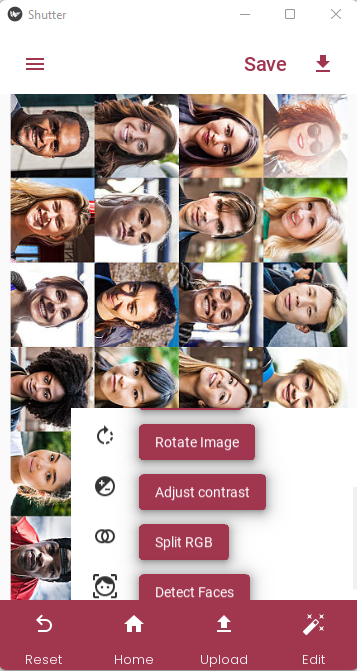
pil\_img = pil\_img.rotate(-90, expand=True)

img = cv2.cvtColor(np.array(pil\_img), cv2.COLOR\_RGB2BGR)

* Để xoay ảnh, đầu tiên ta cần chuyển từ opencv sang pillow để xoay, do đó cần đổi lại hệ màu phù hợp (opencv dung BGR còn pillow dùng RGB) bằng lệnh cv2.cvtColor
* Sau đó mỗi khi người dùng ấn nút Rotate, ảnh sẽ xoay 90o theo chiều kim đồng hồ quanh gốc (0,0) bằng lệnh pil\_img.rotate()
* Hàm sử dụng thuật toán xoay ảnh bằng ma trận chuyển với (x’,y’) là toạ độ điểm lúc sau của điểm gốc sau khi thực hiện nhân với ma trận chuyển:



* Cuối cùng chuyển về hệ màu của OpenC để tiếp tục chỉnh sửa

****

Hình 2.18 Ảnh sau khi xoay

**2.6.6 Tách 3 kênh màu**

elif hist\_type == "split":

b, g, r = cv2.split(img)

zeros = np.zeros\_like(b)

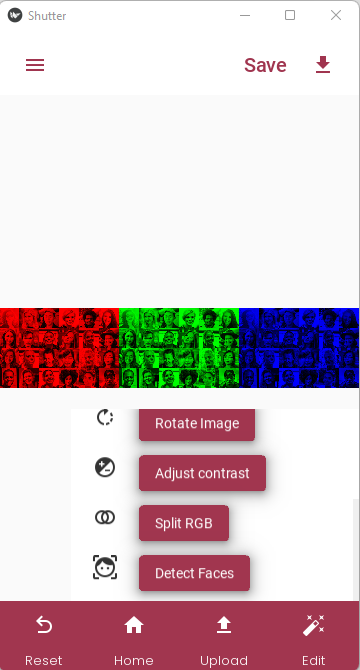
red\_img = cv2.merge([zeros, zeros, r])

green\_img = cv2.merge([zeros, g, zeros])

blue\_img = cv2.merge([b, zeros, zeros])

img = np.hstack((red\_img, green\_img, blue\_img))

* Đầu tiên hàm sẽ tách 3 giá trị tương ứng với 3 kênh màu bằng cv2.split. Định nghĩa 1 biến zeros là 0
* Sau đó tương ứng với mỗi kênh màu sẽ giữ lại giá trị tương ứng ban đầu cho mỗi kênh màu đó và cho giá trị 2 kênh màu còn lại bắng biến zeros và gộp lại bằng lệnh cv2.merge



Hình 2.19 Ảnh sau khi tách 3 kênh màu

**2.6.7 Nhận diện khuôn mặt**

elif hist\_type == "detect\_faces":

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

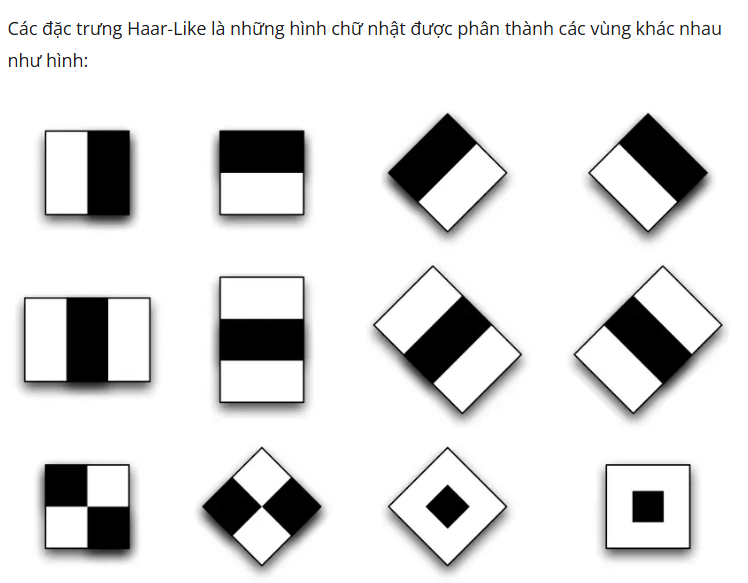
face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')

faces = face\_cascade.detectMultiScale(gray, 1.1, 4)

    for (x, y, w, h) in faces:

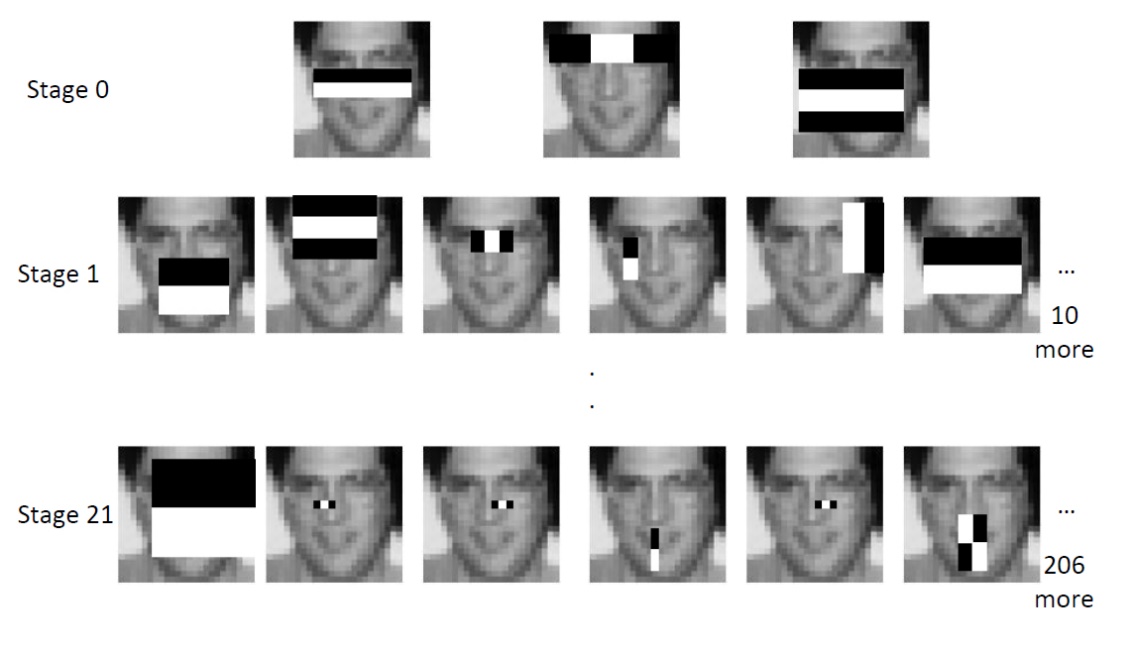
      cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)

* Hàm sử dụng thuật toán học máy truyền thống được huấn luyện sẵn bởi Paul Viola và Michael Jones để nhận biết **đặc trưng của khuôn mặt** trong ảnh đen trắng
* Do chỉ làm việc trên ảnh đen trắng nên đầu tiên ta chuyển về ảnh xám ( 1 kênh) bằng “cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)”
* Sau đó Tải mô hình Haar Cascade đã được **huấn luyện sẵn**, nằm trong thư viện của OpenCV. File XML chứa **các đặc trưng (features)** được học từ nhiều khuôn mặt người
* Haar được dựa trên nguyên lý các hình mẫu gồm vùng sáng – vùng tối tức à những vùng như mắt, mũi, miệng,.. thường có sự chênh lệch độ sáng đặc trưng và Haar tận dụng điều này để xác định khuôn mặt



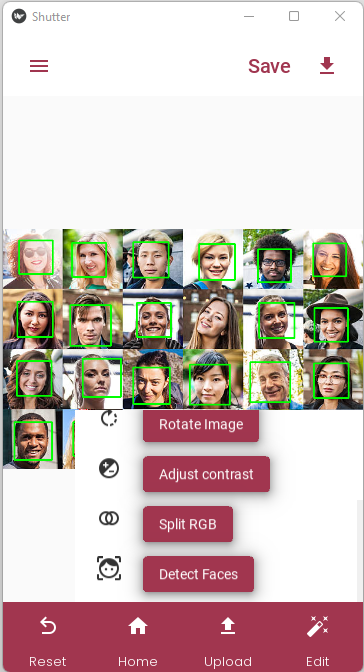
Hình 2.20 Các đặc trưng Haar cơ bản

* Khi một ảnh được đưa vào, 1 hình chữ nhật sẽ được vẽ ra và trượt qua tất cả các điểm ảnh trên ảnh (sliding window) và sử dụng các Haar Feature và  Integral Image(tính ảnh tích phân) để tính được độ trung bình độ chênh lệch giữa vùng đen và vùng trắng trong ảnh, từ đó sẽ xác định được các vùng có nghi ngờ là khuôn mặt dựa trên số liệu đã tính được và số liệu model đã được học



Hình 2.21 Cấu trúc phân tầng và cách xác định các vùng

* Dòng lệnh “faces = face\_cascade.detectMultiScale(gray, 1.1, 4)” để quét ảnh tại nhiều vị trí và qua từng khu vực dựa trên sliding window và áp dụng phân loại phân tầng để phát hiện sớm các vùng không giống khuôn mặt, tránh mất thời gian cho các lớp sau
* Cuối cùng hàm trả về danh sách các vùng chứa khuôn mặt dưới dạng hình chữ nhật bao quanh (x, y, w, h)



Hình 2.22 Chức năng nhận diện gương mặt