# 3. Triển khai (Implementation)

## 3.1. Cấu trúc tổng thể dự án

Project:

* Lpr: Package lõi chứa pipeline và các thuật toán
  + Algorithms.py: Thuật toán xử lý ảnh thủ công (vectorized)
  + Pipeline.py: Quy trình phát hiện biển số
  + Utils.py: Các hàm tiện ích đọc, pad, xoay ảnh
* App.py: Giao diện web Streamlit (hiển thị từng bước)
* Ui\_helpers.py: Hỗ trợ chuyển ảnh upload từ Streamlit
* Run\_batch.py: Script chạy hàng loạt ảnh (batch processing)

## **3.2. Môi trường lập trình**

| **Hạng mục** | | **Thông tin** | |
| --- | --- | --- | --- |
| **Ngôn ngữ** | | Python 3.10 | |
| **Thư viện chính** | | OpenCV, NumPy, Streamlit | |
| **Môi trường IDE** | | Visual Studio Code | |
| **Hệ điều hành** | | Windows 10 x64 | |
| **Dataset** | | Ảnh xe thực tế | |
|  |  | |

## 3.3. Mô tả các thành phần trong file lpr

**(1) utils.py – các hàm tiện ích ảnh**

* imread\_color(path) – đọc ảnh màu từ ổ đĩa, trả về mảng BGR.
* imshow(title, img) – hiển thị ảnh (phục vụ debug).
* rotate\_image(img, angle\_deg) – xoay ảnh theo góc bất kỳ (Affine transform).
* pad\_image(img, pad\_h, pad\_w) – thêm viền phản chiếu, dùng trong phép chập vectorized.

=> Các hàm này là nền tảng cho phần **vector hóa convolution và morphology** trong algorithms.py.

**(2) algorithms.py – các thuật toán xử lý ảnh cơ bản**

Tất cả các bước xử lý ảnh đều được **tự cài đặt bằng NumPy thuần** (không dùng sẵn hàm cv2.GaussianBlur hay cv2.Sobel), giúp minh bạch toàn bộ pipeline.

| **Nhóm thuật toán** | **Chức năng chính** |
| --- | --- |
| **Chuyển đổi ảnh** | rgb2gray() – chuyển RGB → grayscale. |
| **Convolution vectorized** | convolve2d() – cài đặt phép chập bằng sliding\_window\_view. |
| **Gaussian Blur** | gaussian\_kernel(), gaussian\_blur() – làm mờ giảm nhiễu. |
| **Phát hiện biên Sobel** | sobel\_filters() – tính gradient theo 2 hướng, chuẩn hóa 0–255. |
| **Ngưỡng Otsu** | otsu\_threshold() – xác định ngưỡng tự động. |
| **Morphology** | erosion(), dilation(), opening(), closing() – cài thủ công bằng NumPy. |
| **Connected Components** | connected\_components() – thuật toán hai lượt (two-pass). |
| **Grayscale morphology** | gray\_dilate(), gray\_erode() – dùng cho phép Blackhat. |
| **Bounding Box PCA** | oriented\_bbox\_from\_points() – tính hộp bao nghiêng qua PCA. |

**(3) pipeline.py – quy trình phát hiện biển số**

Có hai hàm chính:

1. **detect\_plate\_manual()** – pipeline cho batch processing.
2. **detect\_plate\_manual\_steps()** – phiên bản dành cho Streamlit (trả về từng bước trung gian).

**Các bước xử lý:**

| **Bước** | **Mô tả** | **Hàm sử dụng** |
| --- | --- | --- |
| 0 | Resize ảnh về 900 px | cv2.resize() |
| 1 | Chuyển grayscale | alg.rgb2gray() |
| 2 | Làm mờ Gaussian | alg.gaussian\_blur() |
| 3 | Blackhat (grayscale morphology) | alg.gray\_dilate(), alg.gray\_erode() |
| 4 | Phát hiện biên Sobel | alg.sobel\_filters() |
| 5 | Kết hợp biên + blackhat | np.maximum() |
| 6 | Phân ngưỡng tự động | alg.otsu\_threshold() |
| 7 | Morphology Opening | alg.opening() |
| 8 | Morphology Closing | alg.closing() |
| 9 | Connected Components | alg.connected\_components() |
| 10 | Chọn vùng biển số & crop | alg.oriented\_bbox\_from\_points() |

Kết quả cuối cùng:

* out\_vis: ảnh gốc có khung biển số vẽ màu xanh/đỏ.
* plate\_crop: vùng cắt chứa biển số.

## 3.4. Thành phần giao diện và batch

* **app.py (Giao diện trực quan Streamlit):**
  + Giao diện web cho phép **upload ảnh**, chạy pipeline và hiển thị **mọi bước xử lý**.
  + Gọi hàm detect\_plate\_manual\_steps() và hiển thị kết quả dưới dạng lưới ảnh 4 cột:
* # Hiển thị lưới 4 cột  
  MAX\_COLS = 4  
  step\_items = list(steps\_dict.items())  
    
  for i in range(0, len(step\_items), MAX\_COLS):  
   cols = st.columns(MAX\_COLS)  
   batch = step\_items[i: i + MAX\_COLS]  
    
   for j, (step\_name, cv\_img) in enumerate(batch):  
   with cols[j]:  
   st.caption(f"<b>{step\_name}</b>", unsafe\_allow\_html=True)  
    
   # Dùng hàm từ ui\_helpers.py  
   img\_to\_show = convert\_image\_for\_streamlit(cv\_img)  
    
   if img\_to\_show is not None:  
   st.image(img\_to\_show, width='stretch')  
   else:  
   st.warning("Không có ảnh")  
  st.divider()
  + Sử dụng st.session\_state để lưu nhiều ảnh đã xử lý, không mất khi refresh.
* **ui\_helpers.py**
  + load\_image\_from\_upload() – đọc ảnh từ file upload (Streamlit).
  + convert\_image\_for\_streamlit() – chuyển BGR → RGB để hiển thị đúng màu.
* **run\_batch.py**
  + Dùng cho xử lý hàng loạt ảnh trong dataset.
  + Gọi detect\_plate\_manual() cho từng ảnh và lưu \*\_crop.jpg vào thư mục output.
  + Hỗ trợ nhiều định dạng (.jpg, .png, .bmp, .tiff).

## 3.5. Quy trình hoạt động

1. **Web Mode (Streamlit):** “streamlit run app.py”

Người dùng upload 1 ảnh → hệ thống hiển thị toàn bộ 10 bước xử lý → xem trực tiếp biển số được cắt.

1. **Batch Mode (Python Script):** “python run\_batch.py”

Chạy trên toàn bộ thư mục ảnh đầu vào → kết quả lưu trong thư mục test-out/.

# 4. KẾT QUẢ THỰC NGHIỆM

## 4.1. Dữ liệu và điều kiện

* **Nguồn dữ liệu: Ảnh xe từ camera giám sát Kztech (2025-04-27).**
* **Số lượng ảnh thử nghiệm: 20 ảnh (ô tô, xe máy, xe tải).**
* **Điều kiện chụp: đa dạng về góc, độ sáng, phản chiếu, bóng mờ.**



## 4.2. Kết quả từng bước xử lý

|  |  |  |  |
| --- | --- | --- | --- |
| **Bước** | **Ảnh minh họa** | **Nhận xét** | **Kết quả** |
| 0 | 0\_Resized | Ảnh gốc chụp được từ camera |  |
| 1 | 1\_Gray | Ảnh xám loại bỏ màu nền, giữ chi tiết ký tự. |  |
| 2 | 2\_Blurred | Làm mờ giảm nhiễu nhỏ. |  |
| 3 | 3\_Blackhat | Làm nổi vùng sáng (biển số). |  |
| 4 | 4\_Sobel\_Mag | Phát hiện biên ký tự rõ. |  |
| 5 | 5\_Combined | Vùng biển số nổi bật nhất. |  |
| 6 | 6\_Binary | Ảnh nhị phân hóa rõ vùng biển. |  |
| 7 | 7\_Opened | Nhiễu bị loại bỏ. |  |
| 8 | 8\_Closed | Các ký tự biển nối liền nhau. |  |
| 9 | 9\_Contours | Biển số được vẽ khung xanh (ROI). |  |
| 10 | 10\_Crop | Biển số cắt riêng. |  |

## 4.3. Kết quả tổng hợp

|  |  |  |
| --- | --- | --- |
| **Ảnh** | **Phát hiện** | **Biển số** |
|  | **Nhận diện được** |  |
|  | **Nhận diện được** |  |
|  | **Nhận diện được** |  |
|  | **Nhận diện được** |  |
|  | **Nhận diện được** |  |
|  | **Nhận diện được** |  |
|  | **Nhận diện được** |  |
|  | **Nhận diện được** |  |
|  | **Nhận diện được** |  |
|  | **Nhận diện được** |  |
|  | **Nhận diện được** |  |
|  | **Nhận diện được** |  |
|  | **Không nhận diện được** | **Lý do:** biển số xe quá nhỏ trong khung hình => số pixel ít => Kĩ thuật lọc không bị giảm . |
|  | **Không nhận diện được** | **Lý do:** Ánh sáng tổng thể yếu + mất chi tiết kí tự nhỏ  => Sobel edge yếu |

## Các bước xảy ra với các ảnh bị lỗi

|  |  |  |
| --- | --- | --- |
| **Bước** | **Ảnh** | **Nguyên nhân** |
| 0\_Resized |  |  |
| 1\_Gray |  |  |
| 2\_Blurred |  | Làm mờ ký tự nhỏ |
| 3\_Blackhat |  | KHÔNG highlight biển số |
| 4\_Sobel\_Mag |  | Không bắt cạnh ký tự, chỉ bắt nền |
| 5\_Combined |  | Không có vùng biển số |
| 6\_Binary |  | Nhiễu nhiều, không tách được số |
| 7\_Opened |  | Xóa ký tự vì ký tự quá nhỏ |
| 8\_Closed |  | Không nối được blob |
| 9\_Contours |  | |  | | --- | |  |  |  | | --- | | Không đủ area, AR, extent | |
| 10\_Crop | **Không có ảnh** |  |

## 4.4. Giao diện người dùng (Streamlit UI)

Giao diện người dùng được xây dựng bằng thư viện **Streamlit**, giúp việc kiểm thử từng bước xử lý của pipeline trở nên trực quan, dễ theo dõi và dễ điều chỉnh tham số. Giao diện gồm hai khu vực chính: **Sidebar điều khiển** và **khu vực hiển thị kết quả**.

### 4.4.1. Cấu trúc tổng thể của giao diện

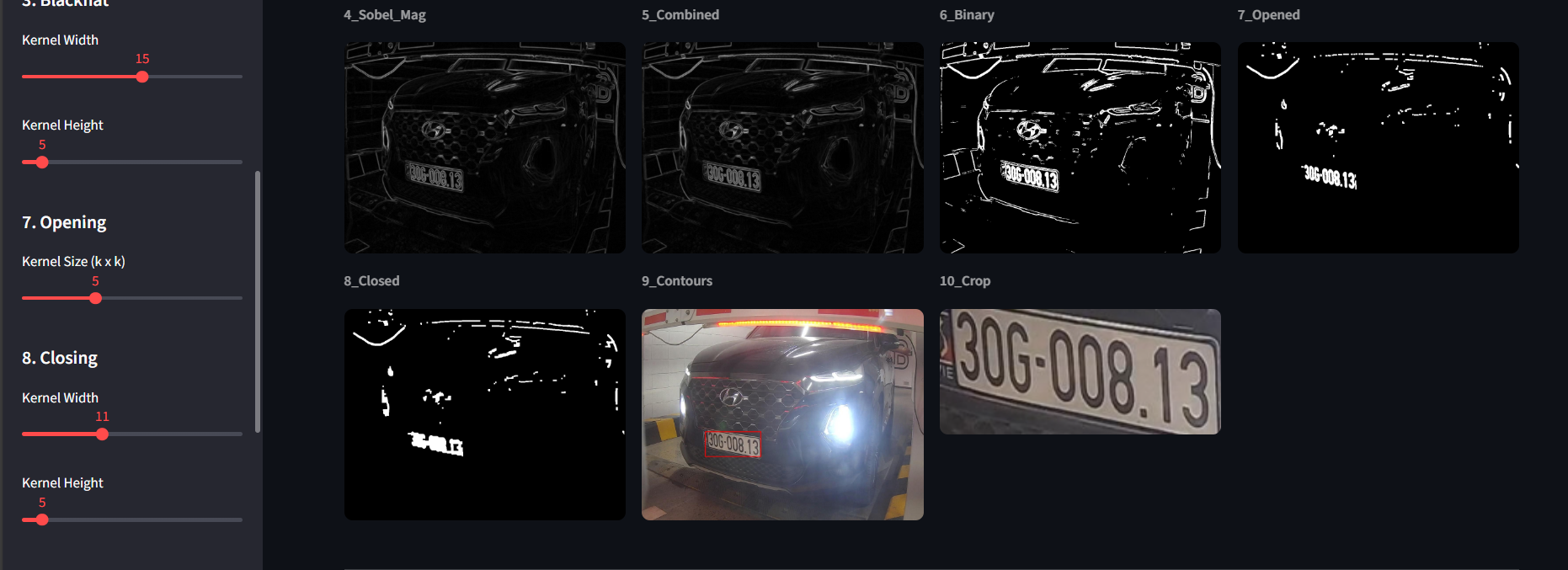
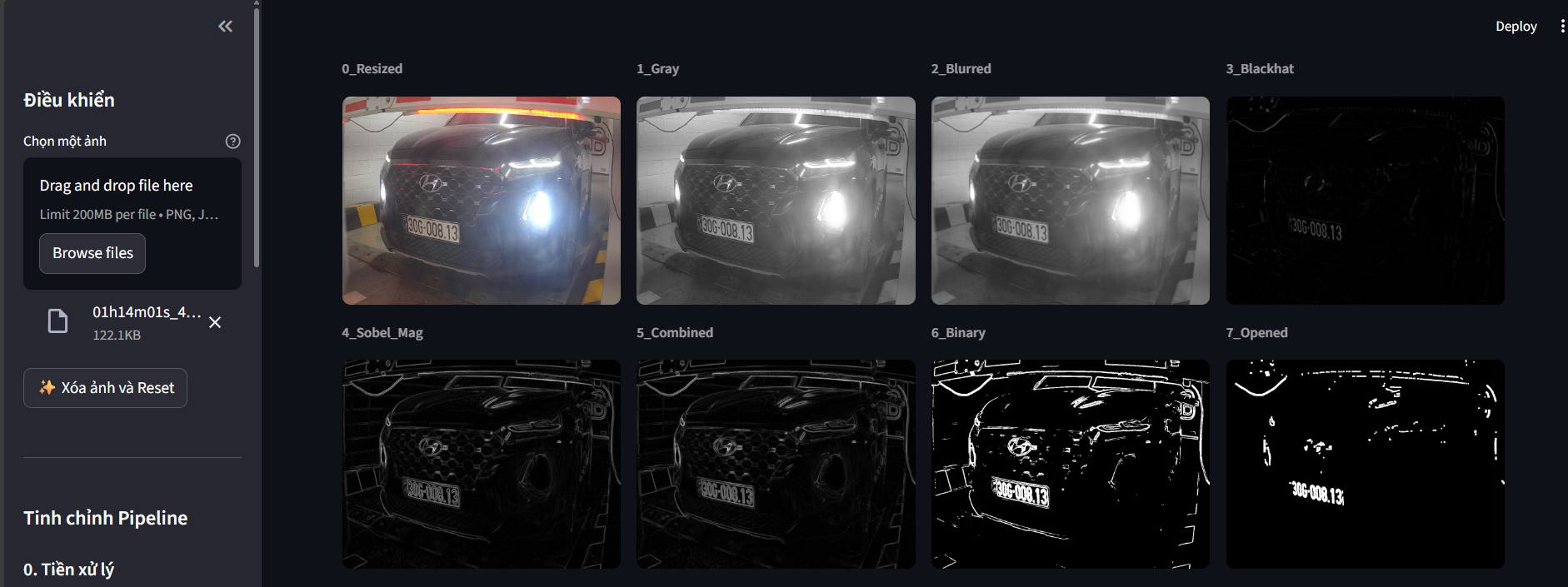
Giao diện chia thành hai phần:

* **Sidebar**: chứa toàn bộ các tham số người dùng có thể điều chỉnh và chức năng upload ảnh.
* **Main Area**: hiển thị ảnh gốc và tất cả các bước trung gian của pipeline theo dạng lưới 4 cột để quan sát dễ dàng.

Cách tổ chức này giúp người dùng vừa xem được thay đổi trực tiếp vừa điều chỉnh các tham số kiểm thử theo thời gian thực.



### 4.4.2. Kết quả hiển thị từ Sidebar và Main Area



#### 4.4.2.1 Sidebar – Khu vực điều khiển

Sidebar bao gồm:

* **Upload ảnh** (cho phép nhận file PNG, JPG, JPEG, BMP).
* **Điều chỉnh tham số pipeline**, bao gồm:
  + Resize width
  + Gaussian blur kernel và sigma
  + Blackhat kernel size
  + Opening kernel size
  + Closing kernel size
  + Các ngưỡng lọc blob:
    - min\_area, max\_area
    - dải tỉ lệ chiều rộng/chiều cao (aspect ratio)
    - extent tối thiểu

Trong quá trình thực nghiệm, việc thay đổi các tham số này dẫn đến những khác biệt rõ rệt trong hình thái các bước xử lý trung gian. Streamlit tự động cập nhật kết quả ngay lập tức, giúp nhóm dễ dàng đánh giá ảnh hưởng của từng tham số lên pipeline.

#### 4.4.2.2 Main Area – Khu vực hiển thị kết quả xử lý

Sau khi người dùng upload ảnh, tất cả các kết quả từ các bước trong pipeline sẽ được hiển thị theo thứ tự:

1. Ảnh đã resize
2. Ảnh grayscale
3. Gaussian blur
4. Blackhat
5. Sobel
6. Combined
7. Binary
8. Opended
9. Closed
10. Contours
11. Crop vùng biển số

Việc trình bày tuần tự này giúp người đọc và người thực nghiệm dễ dàng theo dõi logic hoạt động của hệ thống qua từng bước.

### 4.4.3. Quy trình hoạt động của giao diện

Quy trình hoạt động liên kết giữa giao diện và pipeline bao gồm:

1. Người dùng upload ảnh → ảnh được đọc dưới dạng byte.
2. Giao diện chuyển đổi ảnh sang định dạng xử lý và gửi vào hàm detect\_plate\_manual\_steps().
3. Pipeline thực hiện lần lượt các bước xử lý (blur, blackhat, threshold, morphology, lọc blob...).
4. Mỗi bước trả về ảnh trung gian → giao diện hiển thị ngay trên Main Area.
5. Khi người dùng điều chỉnh tham số, giao diện tự rerun và cập nhật toàn bộ kết quả.

Nhờ cơ chế này, giao diện đóng vai trò là **công cụ kiểm thử trực quan** cho toàn bộ pipeline.

### 4.4.4. Quản lý trạng thái bằng session\_state

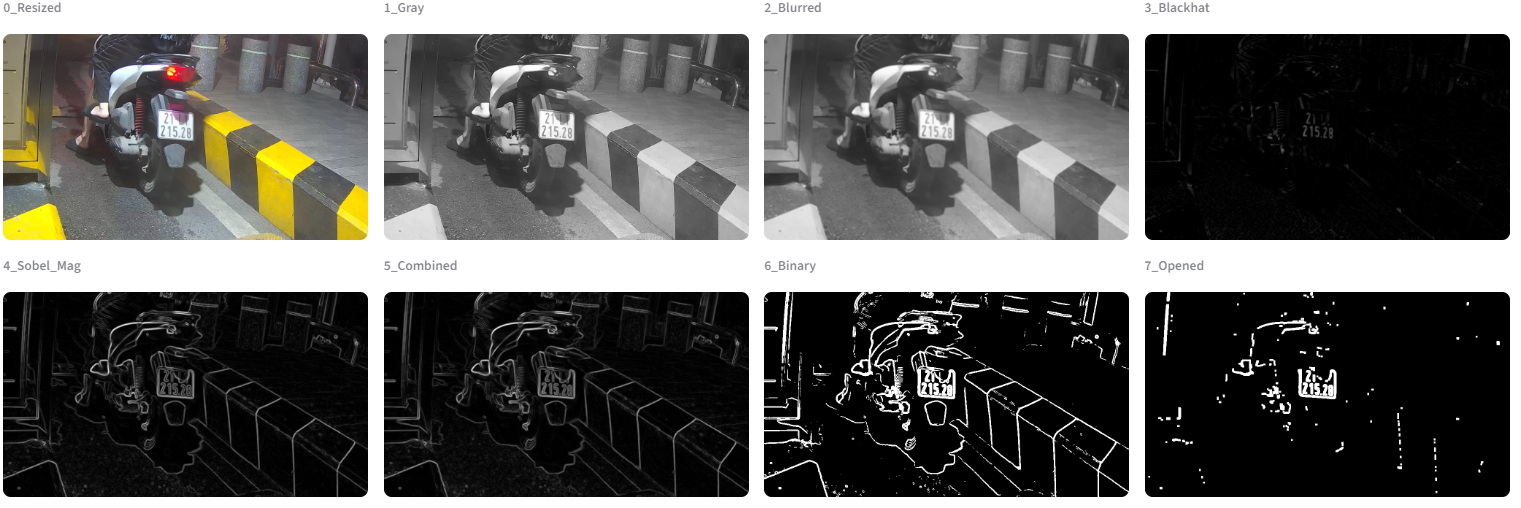
Trong Streamlit, mỗi lần người dùng thay đổi tham số, ứng dụng sẽ rerun. Để hạn chế mất dữ liệu, giao diện sử dụng:

* st.session\_state['image\_bytes']: lưu ảnh đang được xử lý
* st.session\_state['file\_name']: lưu tên file
* Các giá trị tham số hiện tại của người dùng

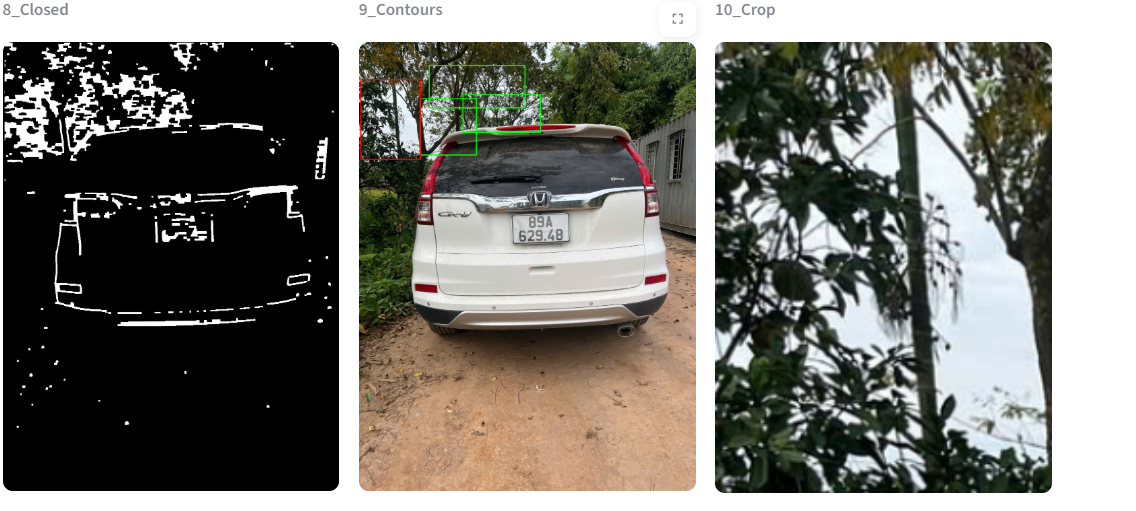
Cách quản lý này giúp:

* Không bị mất ảnh mỗi khi rerun
* Chỉ xử lý lại ảnh khi người dùng upload ảnh mới
* Tránh pipeline chạy lại không cần thiết
* Giữ giao diện ổn định và thân thiện

### 4.4.5. Minh họa giao diện với các trường hợp thành công và thất bại

Trường hợp thành công:  


Trường hợp thất bại:



# PHỤ LỤC: CHI TIẾT KẾT QUẢ THỬ NGHIỆM TRÊN TẬP DỮ LIỆU

|  |  |
| --- | --- |
| Ảnh gốc | Biển số |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | Không phát hiện biển số xe |
|  | Không phát hiện biển số xe |
|  | Không phát hiện biển số xe |
|  | Không phát hiện biển số xe |
|  | Không phát hiện biển số xe |
|  |  |
|  |  |
|  | Phát hiện nhầm biển số xe |
|  |  |
|  | Không phát hiện biển số xe |
|  | Không phát hiện biển số xe |
|  |  |
|  |  |
|  |  |
|  |  |
|  | Không phát hiện biển số xe |
|  |  |
|  |  |
|  | Không phát hiện biển số xe |
|  | Không phát hiện biển số xe |
|  |  |
|  | Phát hiện nhầm biển số xe |
|  | Không phát hiện biển số xe |
|  |  |
|  |  |
|  |  |
|  |  |
|  | Không phát hiện biển số xe |
|  |  |
|  | Không phát hiện biển số xe |
|  |  |
|  |  |
|  |  |
|  | Không phát hiện biển số xe |
|  |  |
|  | Không phát hiện biển số xe |
|  | Không phát hiện biển số xe |
|  |  |
|  | Không phát hiện biển số xe |
|  |  |
|  | Không phát hiện biển số xe |
|  | Không phát hiện biển số xe |
|  | Không phát hiện biển số xe |
|  |  |