Hackathon Level 1 Solution (Question 1)

Use Case Title: Multiple Color Detection Application

Student Name: Swetha K

Register Number: 732323106049

Institution: SSM College of Engineering

Department: Electronics and Communication Engineering

Date of Submission: May 10, 2025

Github Link:https://github.com/Swetha704/color-detection-using-python/upload

Public Link:https://color-detection-using-python-erd8y2f36nzmnoowjsaq2z.streamlit.app/

1. **Problem Statement**

Identifying colors in images or videos is a common need in fields like education, design, and accessibility, but manual identification is time-consuming and error-prone, especially for large datasets or real-time applications. Additionally, individuals with visual impairments may struggle to discern colors, and automated systems often fail to detect colors accurately due to lighting variations or limited color ranges. The challenge is to create a robust application that can detect multiple colors in images or videos, label them accurately, and provide detailed color information in an accessible format, even for edge cases like solid color images or small colored regions.

2. **Proposed Solution**

The Multiple Color Detection Application is a web-based tool that processes images or videos to detect regions of red, green, and blue colors, labels them with their closest matching color names from a dataset, and displays the color names and RGB values below the visual output. Key features include:

- Support for image (PNG, JPG, JPEG) and video (MP4, AVI) uploads.
- Detection of red, green, and blue regions using HSV color space, with fallback logic for solid color images.
- Contour-based region identification with a lowered area threshold (100 pixels) to detect smaller regions.
- Use of a color dataset (colours_rgb_shades.csv) to map detected RGB values to precise color names.
- Display of detected colors and their RGB values below the image/video, along with debug information to diagnose detection issues. This solution addresses the problem by automating color detection, improving accuracy through wider HSV ranges and fallback mechanisms, and providing an accessible interface for users to analyze colors in visual media.

3. **Technologies & Tools**

Considered The following technologies and tools were used to develop the solution:

- Python: The primary programming language for development.
- OpenCV: For image and video processing, including color space conversion, mask creation, and contour detection.
- NumPy: For numerical operations, such as calculating average colors in regions.
- Pandas: For loading and processing the color dataset.
- Streamlit: For creating an interactive web interface to upload files and display results.
- Pillow: For handling image file inputs.

4. Solution Architecture & Workflow The system follows a modular workflow:
5. Input: Users upload an image or video via the Streamlit interface.
6. Preprocessing: The input is converted to HSV color space for color detection.
7. Color Detection: HSV masks are created for red, green, and blue, using widened ranges to detect more shades.
8. Fallback Check: The average HSV of the image is checked to detect solid colors.
9. Region Identification: Contours are detected in each mask, with a threshold of 100 pixels for area.
10. Color Naming: The average RGB of each region (or the whole image for solid colors) is matched against the dataset.
11. Output: Bounding boxes with labels are drawn on the image, and detected colors with RGB values are displayed below, along with debug information.

4. **Solution Architecture**: The system is structured into three main layers:

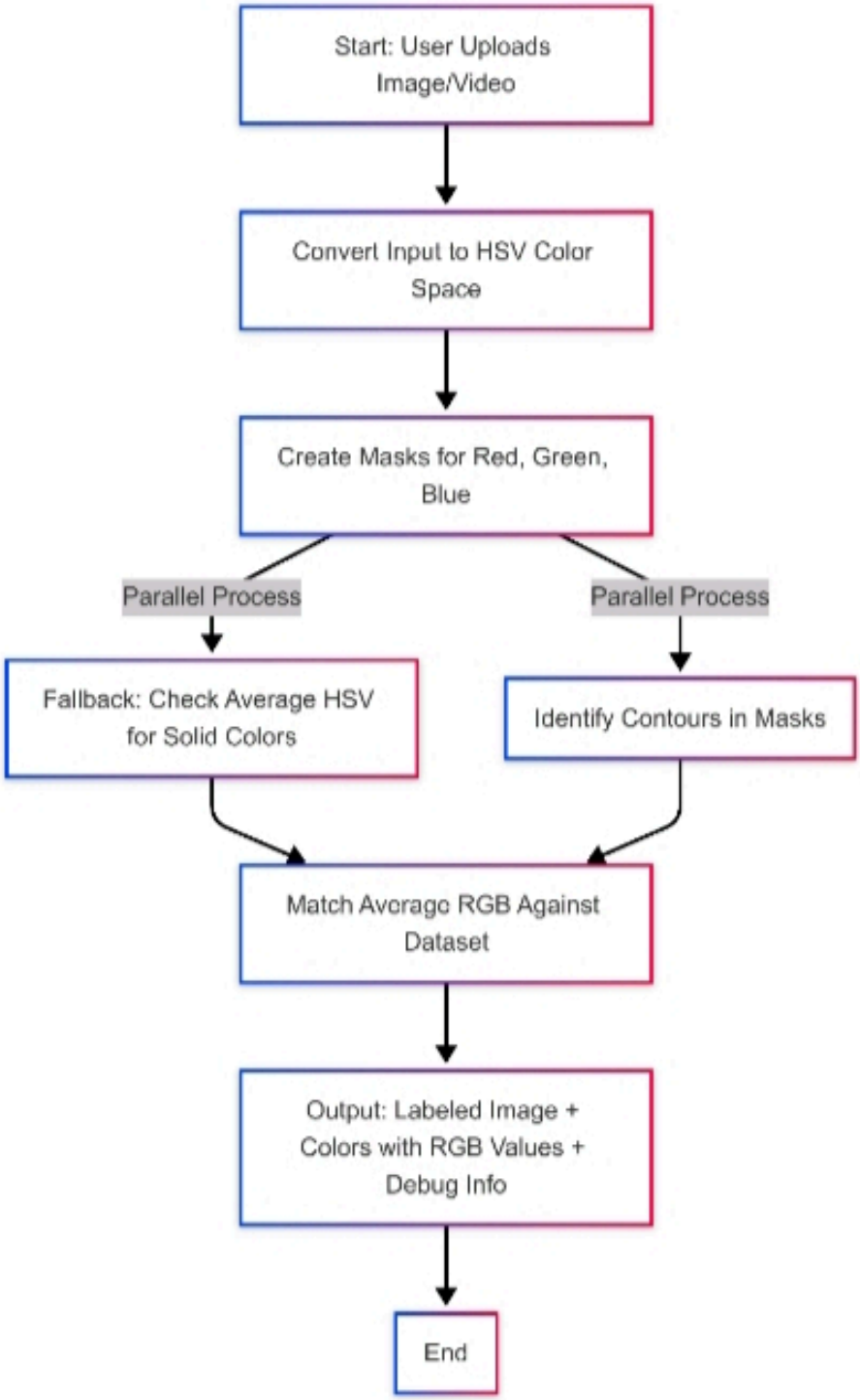
➤ Frontend: Streamlit provides an interactive web interface where users can upload images or videos and view the processed output. It handles user input and displays the results, including labeled images/videos, detected colors with RGB values, and debug information.

- Backend Processing: Python scripts using OpenCV perform the core color detection tasks. This layer converts inputs to HSV color space, creates masks for color detection, identifies regions via contours, and matches colors against the dataset. NumPy and Pandas assist with numerical computations and dataset handling, respectively.
- Data Layer: The color dataset (colours_rgb_shades.csv) serves as a reference for mapping RGB values to color names, enabling precise labeling of detected colors.

Interactions: The frontend sends the uploaded file to the backend for processing. The backend accesses the data layer to match colors, processes the input, and returns the labeled output to the frontend for display. Debug information is generated during processing and also sent to the frontend for user feedback.

- **Workflow**: The system follows a modular workflow:
- Input: Users upload an image or video via the Streamlit interface.
- Preprocessing: The input is converted to HSV color space for color detection.
- Color Detection: HSV masks are created for red, green, and blue, using widened ranges to detect more shades.
- Fallback Check: The average HSV of the image is checked to detect solid colors.
- Region Identification: Contours are detected in each mask, with a threshold of 100 pixels for area.
- Color Naming: The average RGB of each region (or the whole image for solid colors) is matched against the dataset.
- Output: Bounding boxes with labels are drawn on the image, and detected colors with RGB values are displayed below, along with debug information.

Flowchart :



5. Feasibility & Challenges

- Feasibility: The solution is highly feasible as it uses mature libraries like OpenCV and Streamlit, which are well-documented and widely adopted. Deployment on Streamlit Cloud eliminates the need for complex server management, making it accessible to users without technical expertise. The use of a pre-existing color dataset simplifies color identification.
- Challenges:
  - ✓ Color Detection Accuracy: Some images initially failed to detect colors due to narrow HSV ranges or small regions. This was addressed by widening HSV ranges, lowering the area threshold to 100 pixels, and adding a fallback for solid colors.
  - ✓ Performance with Videos: Real-time video processing can be slow due to frame-by-frame analysis. Future optimizations, such as processing every nth frame, could improve performance.
  - ✓ Lighting Variations: Changes in lighting can affect HSV values. Adaptive thresholding or machine learning-based color detection could mitigate this.
  - ✓ Expected Outcome & Impact The application will provide an efficient tool for color detection, benefiting:
  - ✓ Educational Users: Students can use it to learn about colors and computer vision techniques.

- ✓ Designers: Professionals can analyze color compositions in images or videos.
- ✓ Accessibility: Visually impaired users can benefit from automated color identification. By automating color detection and providing debug information, the solution enhances accuracy and usability, making color analysis more accessible and reliable across various domains.

7. **Future Enhancements Potential Improvements include:**

- Support for More Colors: Add detection for colors like yellow, orange, and purple by defining additional HSV ranges.
- Machine Learning Integration: Use a trained model for color classification, improving accuracy over static HSV ranges.
- Enhanced Debug Tools: Allow users to toggle debug information or visualize HSV masks.
- Real-Time Webcam Support: Integrate webcam streaming for local deployments, enabling live color detection.