**Use Case Title: Colour Detection from Images**

**Student Name: DIVYASRI.S**

**Register number: 510923104015**

**Institution: GLOBAL INSTITUTE OF ENGINEERING AND TECHNOLOGY**

**Department: COMPUTER SCIENCE AND ENGINEERING**

**Date of Submission: 14.05.2025**

**Github reposiratory link:** **<https://github.com/saravanan1970/colourdetectionfromimage.git>**

**Colour Detection from Images**

1. **Problem Statement:**

**Colour plays a vital role in various domains such as digital design, manufacturing, medical imaging, and agriculture. However, accurately identifying and extracting colours from images remains a challenge due to factors like lighting conditions, image quality, and background noise. Many existing solutions either lack precision or require extensive manual processing. This project aims to develop an automated system for Colour Detection from Images that can accurately identify dominant and specific colours within an image, regardless of environmental variations.**

**solution is needed to streamline workflows in industries that rely on colour-based decisions, including e-commerce (e.g., identifying product colours), agriculture (e.g., detecting ripeness or disease in crops), and accessibility technology (e.g., assisting colour-blind individuals).**

**2.Proposed Solution:**

**The proposed solution is a Colour Detection System that processes digital images to accurately identify, extract, and classify colours present in the image. This system leverages image processing and computer vision techniques to analyze image pixels and detect both dominant and specific colour values (e.g., RGB, HEX, HSV formats).**

* **Key Features of the Solution:**
* **Image Upload and Input Handling:**

**Users can upload images in common formats (JPG, PNG, etc.).**

**The system supports multiple resolutions and automatically adjusts for optimal processing.**

* **Colour Extraction and Classification:**

**The system uses clustering algorithms (e.g., K-Means) to identify dominant colours in the image.**

**It maps the detected colours to standard colour names using a colour database or API.**

* **Noise Reduction and Preprocessing:**

**Filters are applied to eliminate noise, shadows, or background variations that can affect colour accuracy.**

**Image resizing and colour space conversions (e.g., RGB to LAB or HSV) are used for more robust detection.**

* **User Interaction and Visualization:**

**The extracted colours are displayed along with their HEX/RGB codes and corresponding names.**

**Users can click on a point in the image to get the exact colour at that location.**

* **How It Solves the Problem:**

**This solution automates the complex task of identifying accurate colours in images, overcoming common challenges like inconsistent lighting or mixed backgrounds. It provides reliable colour data quickly and efficiently, supporting diverse applications in design, agriculture, e-commerce, and accessibility. By reducing the need for manual effort and increasing precision, the system enhances productivity and decision-making wherever colour recognition is critical**

**3.Technologies & Tools Considered:-**

**For a Colour Detection from Images project, a variety of technologies and tools can be used depending on the complexity and goals of the solution. Here’s a structured list of technologies, programming languages, frameworks, and APIs that could be considered:**

* **Programming Languages:**

**1.Python – Widely used for image processing and computer vision due to rich libraries.**

**2.JavaScript – Useful for web-based colour detection tools.**

**3.C++ – For high-performance applications (commonly with OpenCV).**

**4.Java – Alternative for mobile (Android) or desktop applications.**

* **Frameworks & Libraries:**

1. **Python Libraries**

**1.OpenCV – Core library for image processing and colour detection.**

**2.NumPy – For matrix and pixel-level operations.**

**3.Matplotlib – For visualizing colour distribution and histograms.**

**Pillow (PIL) – Image loading and basic manipulation.**

**5.scikit-image – Advanced image analysis.**

1. **JavaScript Libraries**

**1.p5.js – For image analysis and creative coding.**

**2.fabric.js / Konva.js – For canvas manipulation.**

**3.Colour-thief – Extracts prominent colours from an image.**

* **APIs & Tools:**

**1.ColourThief API – Extract dominant colour from images.**

**2.Google Vision API – Offers colour properties as part of its image analysis.**

**3.Adobe Colour API – For colour extraction and palette generation.**

**4.Cloudinary – Image processing API with colour extraction capabilities.**

* **Development Tools:**

1. **Jupyter Notebook – Ideal for prototyping and visualizing Python-based image processing.**

**2.VS Code / PyCharm – General-purpose IDEs for coding.**

**3.Google Colab – Cloud-based Python execution, great for demos and sharing.**

* **Colour Detection Algorithms:**

**1.K-Means Clustering – Groups similar colours for dominant colour extraction.**

**2.DBSCAN – Identifies colour regions with density-based clustering.**

**3.Histogram Analysis – Helps in analysing pixel distribution to extract colour patterns.**

* **Additional Tools:**

**1.Matplotlib & Seaborn – For visualizing colour distributions.**

**2.NumPy – Essential for image data manipulation and numerical computations.**

**3.Streamlit – Helps in building intuitive interfaces for colour detection applications**

**With these technologies, an accurate and scalable colour detection system can be developed, ensuring robust performance across different applications.**

**4. Solution Architecture & Workflow:-**

**This AI-powered colour detection system operates through a structured pipeline consisting of key components designed to ensure accurate colour extraction and classification. Below is a high-level breakdown of the solution architecture and its workflow.**

* **Image Acquisition & Preprocessing:**

**Users can upload an image or capture one in real-time using a webcam.**

**The image undergoes preprocessing, including noise reduction, contrast enhancement, and resizing to optimize it for analysis.**

**Conversion to HSV or LAB colour space ensures better segmentation of colour regions.**

* **Feature Extraction & Colour Analysis:**

**K-Means Clustering identifies dominant colours by grouping similar pixels.**

**Histogram Analysis quantifies pixel distribution for accurate colour representation.**

**Mapping detected colours to a predefined colour dictionary ensures consistency**

* **Machine Learning & AI Processing:**

**Pre-trained models (e.g., CNN-based colour classification) refine colour accuracy.**

**Integration of deep learning techniques enhances detection in complex images.**

**Model fine-tuning ensures robustness across different lighting conditions**

* **User Interface & Interaction:**

**The processed image, along with extracted colour details, is displayed on a web or mobile interface.**

**Users receive colour codes (Hex, RGB, HSV) for identified colours.**

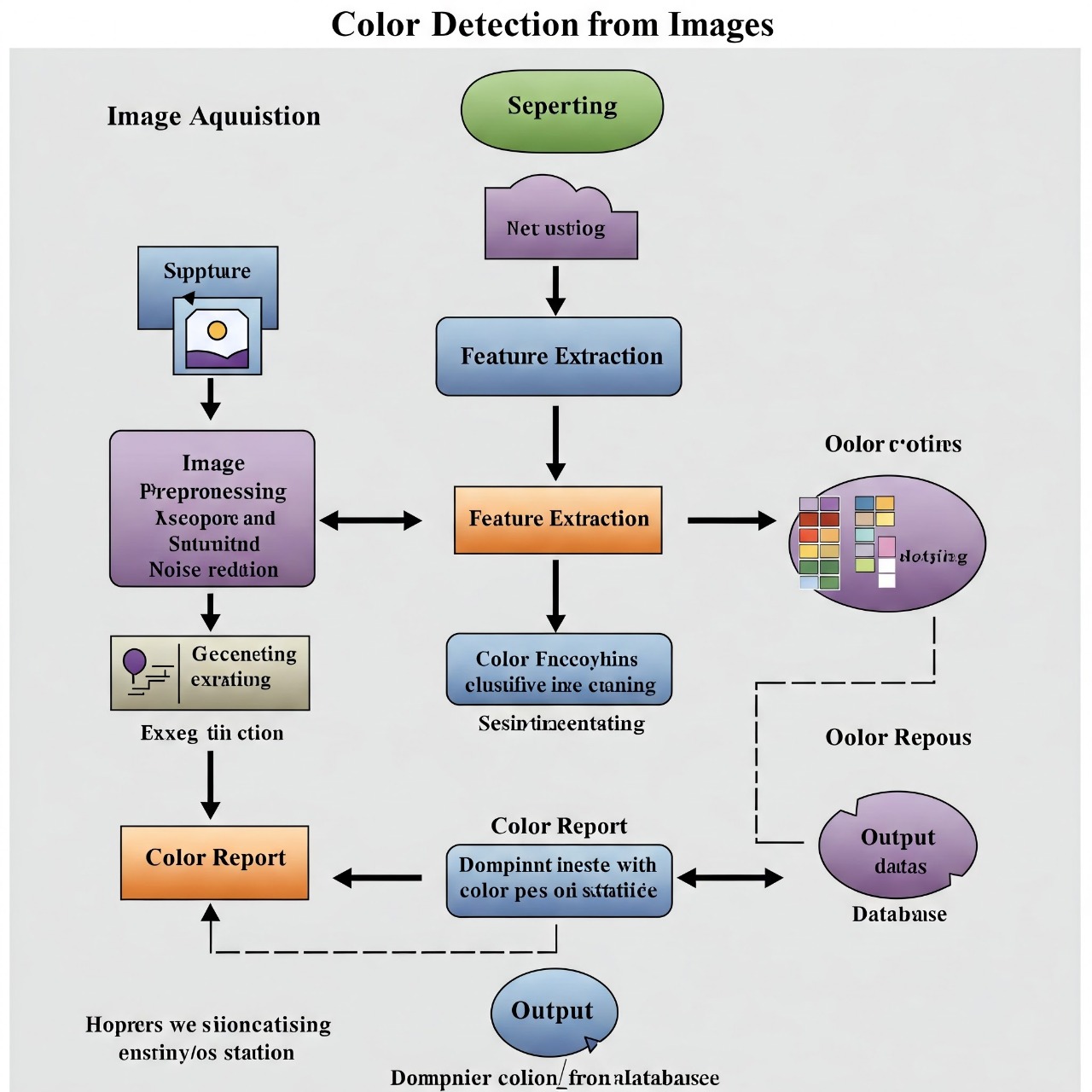
**Option to download results or apply detected colours to industry-specific applications.**

* **Deployment & Optimization:**

**The system is deployed using Flask, FastAPI, or Streamlit for web-based interaction.**

**Cloud-based model hosting (AWS, Azure, GCP) ensures scalability.**

**Flowchart:**



**Source code:**

**from graphviz import Digraph**

**# Create a new directed graph**

**dot = Digraph(comment='Colour Detection Flowchart')**

**dot.attr(rankdir='TB') # TB = Top to Bottom, LR = Left to Right**

**# Define nodes (shapes can be specified)**

**# Oval for Start/End**

**dot.node('A', 'Start', shape='oval')**

**dot.node('J', 'End', shape='oval')**

**# Parallelogram for Input/Output**

**dot.node('B', 'Input Image', shape='parallelogram')**

**dot.node('I', 'Output Results', shape='parallelogram')**

**# Rectangle for Processes**

**dot.node('C', 'Pre-processing\n(Resize, Noise Reduction, Colour Space Conversion)', shape='box')**

**dot.node('D', 'Select Region of Interest (ROI)\n(Optional)', shape='box')**

**dot.node('F', 'Extract Colour Information', shape='box')**

**dot.node('G', 'Process/Analyze Colour Data\n(Averaging, Counting, Mapping to Names)', shape='box')**

**# Diamond for Decision/Method Choice**

**dot.node('E', 'Colour Identification/Quantization Method\n(Pixel Iteration, Masking, Clustering, Histogram)', shape='diamond')**

**# Define edges (connections between nodes)**

**dot.edge('A', 'B') # Start -> Input Image**

**dot.edge('B', 'C') # Input Image -> Pre-processing**

**dot.edge('C', 'D') # Pre-processing -> Select ROI**

**dot.edge('D', 'E') # Select ROI -> Colour Identification Method**

**# If ROI is skipped, Pre-processing can go directly to Colour Identification**

**# You could add an alternative path or just keep it sequential for simplicity here.**

**# For a more complex flowchart, you'd add more decision logic.**

**dot.edge('E', 'F') # Colour Identification Method -> Extract Colour Info**

**dot.edge('F', 'G') # Extract Colour Info -> Process/Analyze Data**

**dot.edge('G', 'I') # Process/Analyze Data -> Output Results**

**dot.edge('I', 'J') # Output Results -> End**

**# --- Optional: Add a "bypass" for the ROI step if it's truly optional ---**

**# This makes the graph slightly more complex to draw simply.**

**# For this example, we'll keep the main flow.**

**# An alternative is to have an edge from 'C' directly to 'E' labeled "Skip ROI"**

**# Specify output format and render the graph**

**# You can render to PDF, PNG, SVG, etc.**

**try:**

**dot.render('colour\_detection\_flowchart', view=True, format='png')**

**print("Flowchart 'colour\_detection\_flowchart.png' generated successfully.")**

**print("It should open automatically if view=True and a viewer is available.")**

**except Exception as e:**

**print(f"Error rendering flowchart: {e}")**

**print("Make sure Graphviz is installed and in your system's PATH.")**

**5.Feasibility & Challenges:-**

**✅ Feasibility:**

**The solution is highly practical and implementable, especially given the wide availability of open-source tools and libraries for image processing. Here's why:**

**1.Technological Maturity: Libraries like OpenCV, Pillow, and scikit-image in Python provide all necessary functions for image reading, colour space conversion, and pixel-level operations.**

**2.Lightweight Processing: For basic colour detection (e.g., dominant colour extraction), the computational load is low, allowing it to run on mobile devices or in-browser with JavaScript.**

**3.Cross-Platform Options: Can be developed as a desktop app, mobile app, or web app using common tech stacks (Python, JavaScript, etc.).**

**4.Availability of Cloud Resources: If required, APIs like Google Vision or Azure Computer Vision can offer advanced capabilities without building everything from scratch**

**⚠️ Challenges & Suggested Solutions:**

* **Colour Accuracy & Variability:**

**Challenge: Lighting, shadows, and image quality can affect colour accuracy.**

**Solution: Normalize lighting conditions using preprocessing techniques like histogram equalization, or apply colour constancy algorithms**

* **Real-Time Performance (if needed):**

**Challenge: Real-time colour detection in video streams can be computationally intensive.**

**Solution: Optimize using low-resolution frames, region-of-interest analysis, or use C++/OpenCV for performance-critical applications.**

* **Multiple Colour Detection & Clustering Accuracy:**

**Challenge: Clustering (e.g., K-means) can give inconsistent results depending on the image.**

**Solution: Use adaptive clustering (e.g., Elbow method to determine optimal K) and post-processing filters to refine results.**

* **User Interface Complexity:**

**Challenge: Designing an intuitive UI that visually represents colours in an informative and appealing way.**

**Solution: Use colour palettes, visual bars, and live previews; frontend libraries like React and Chart.js can help**

* **Browser and Device Compatibility (Web/Mobile):**

**Challenge: Not all browsers or devices handle image files and canvas operations consistently.**

**Solution: Use polyfills, responsive frameworks, and thorough cross-browser testing.**

* **Colour Naming and Human Interpretation:**

**Challenge: RGB values don’t always map neatly to known colour names.**

**Solution: Use predefined colour maps (e.g., XKCD colour survey dataset or CSS3 named colours) for approximate human-friendly naming**

* **Large Image Sizes:**

**Challenge: High-resolution images may slow down processing.**

**Solution: Down sample images before processing or limit the upload size**

**6. Expected Outcome & Impact:-**

**✅ Expected Outcomes:**

* **Accurate Colour Identification:**

**The system will reliably detect dominant and specific colours in an image, presenting them in easily interpretable formats such as HEX, RGB, and colour names.**

* **User-Friendly Interface:**

**Users will be able to upload an image and receive immediate visual and textual feedback about its colour content, improving accessibility and efficiency.**

* **Automated and Scalable Process:**

**Manual colour picking and guesswork will be replaced by automated, consistent detection, saving time and reducing human error.**

* **Cross-Platform Usability:**

**The system can be adapted for use in web, desktop, or mobile environments, depending on user needs.**

**🌍 Impact & Benefits:**

**1.Designers & Developers:**

**Benefit: Quickly extract colour palettes for UI/UX design, branding, or digital art.**

**Impact: Streamlines the design workflow and enhances visual consistency across products.**

**2. E-commerce Platforms:**

**Benefit: Automatically tag product images with accurate colour information.**

**Impact: Improves searchability, customer experience, and product filtering options.**

**3. Accessibility for Colour-Blind Users:**

**Benefit: Provides labeled colour information, potentially with audio or text feedback.**

**Impact: Makes visual content more accessible to those with colour vision deficiencies.**

**4. Agriculture & Medical Imaging:**

**Benefit: Detects subtle colour changes in plants, skin, or medical scans.**

**Impact: Supports early detection of issues such as disease, decay, or infection.**

**5. Education & Research:**

**Benefit: A tool for learning about colour theory, computer vision, or digital imaging.**

**Impact: Facilitates hands-on learning and experimentation.**

**7.Future Enhancements:-**

**Here’s a detailed section on Future Enhancements for a Colour Detection from Images solution:**

**🔮 Future Enhancements:**

**1. Real-Time Colour Detection:**

**Description: Enable live colour detection from video streams (e.g., webcam or phone camera).**

**Benefit: Useful for stylists, designers, or users testing colour combinations on the fly.**

**2. Colour Name Suggestions Using AI:**

**Description: Integrate machine learning to suggest human-friendly or brand-specific colour names (e.g., “Sunset Coral” instead of just “#FF6E6E”).**

**Benefit: Makes colour data more intuitive and marketing-ready.**

**3. Colour Harmonization Tools:**

**Description: Suggest complementary, analogous, or triadic colour palettes based on detected colours.**

**Benefit: Supports design and fashion use cases by aiding colour-matching decisions.**

**4. Skin Tone Detection & Matching:**

**Description: Add modules to detect skin tones in fashion photos and suggest colours that complement them.**

**Benefit: Personalized fashion advice and better styling for different audiences.**

**5. Integration with E-commerce APIs:**

**Description: Connect to platforms like Shopify, Amazon, or Zalando to auto-tag and filter clothing based on detected colours.**

**Benefit: Enhances product discoverability and improves visual search features.**

**6. Colour Trend Analysis:**

**Description: Use colour detection across datasets (e.g., social media, fashion week photos) to track emerging colour trends.**

**Benefit: Valuable for fashion forecasting and inventory planning.**

**7. Multi-language Colour Descriptions:**

**Description: Translate colour names and descriptions into multiple languages.**

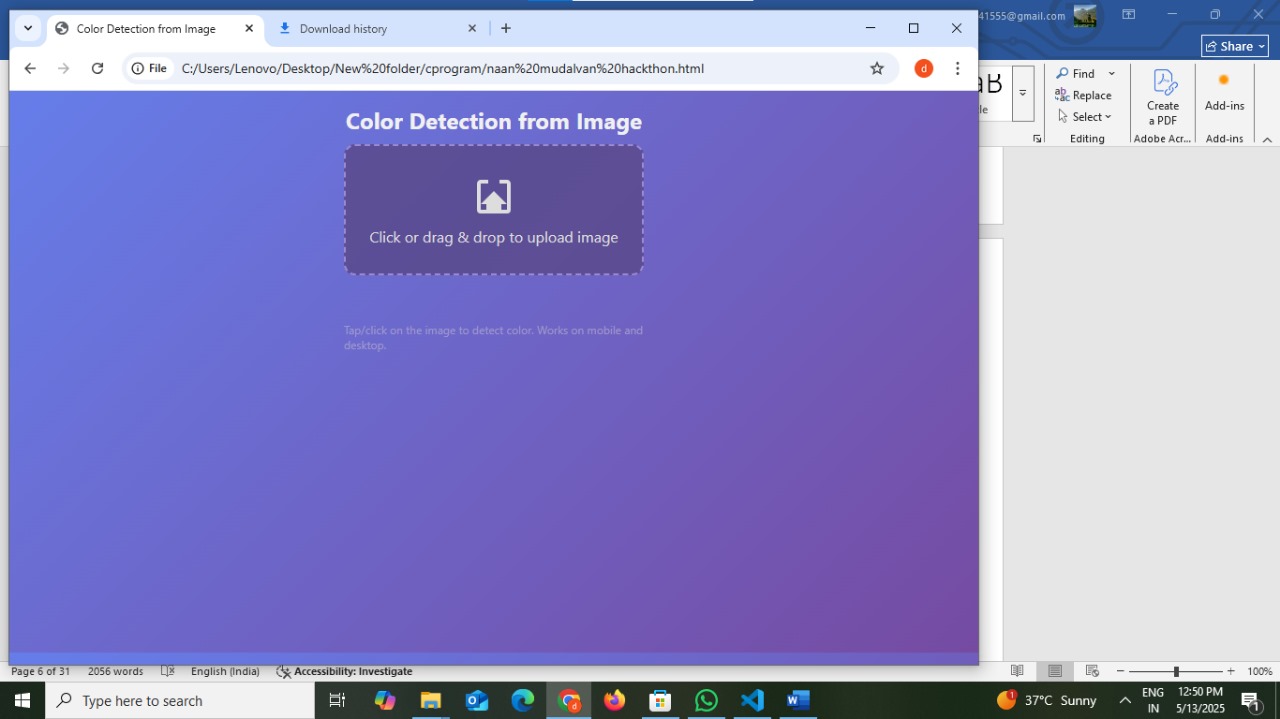
**Benefit: Useful for global users and international fashion platforms.**

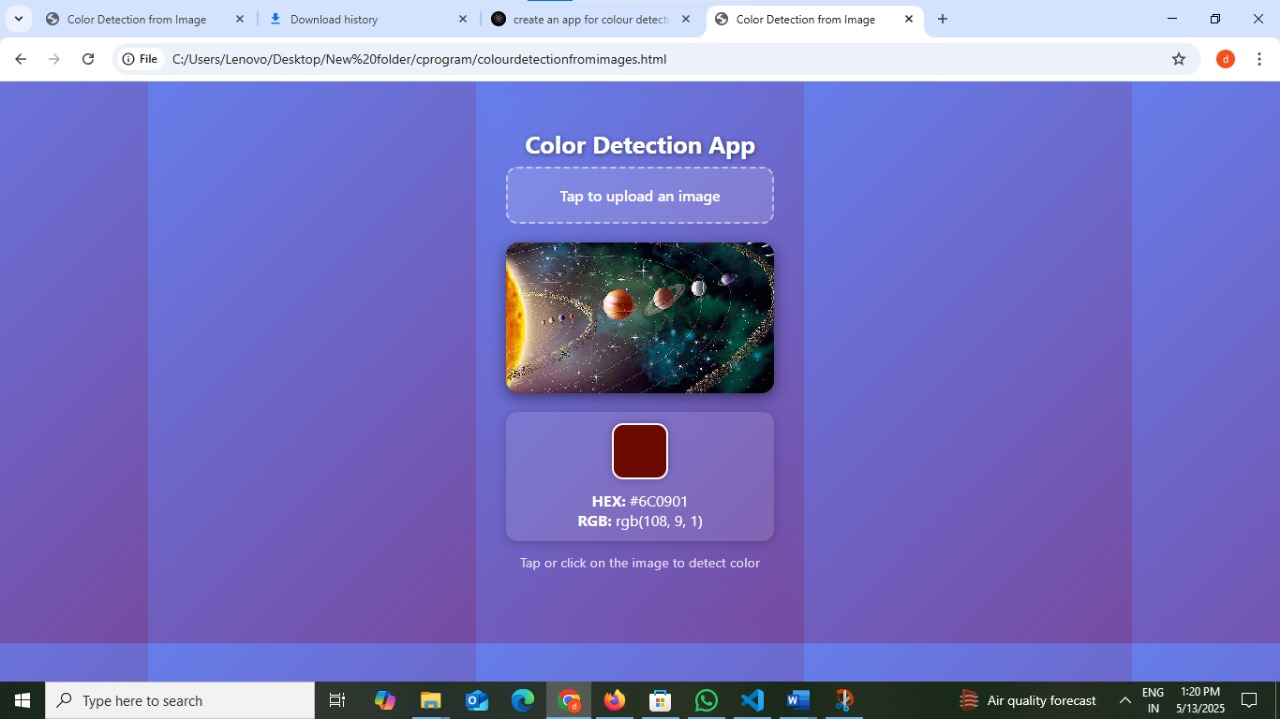
**8. Accessibility Enhancements:**

**Description: Offer features like colour-blind-safe palettes or auditory descriptions of colours.**

**Benefit: Makes the tool inclusive for users with visual impairments.**

**PUBLIC LINK:** [**C:\Users\Lenovo\Desktop\New folder\cprogram\colourdetectionfromimages.html**](file:///C:\Users\Lenovo\Desktop\New%20folder\cprogram\colourdetectionfromimages.html)





**SOURCE CODE:**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8" />**

**<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no" />**

**<title>Color Detection from Image</title>**

**<style>**

**/\* Reset and base \*/**

**\* {**

**box-sizing: border-box;**

**}**

**body, html {**

**margin: 0;**

**padding: 0;**

**height: 100%;**

**font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;**

**background: linear-gradient(135deg, #667eea, #764ba2);**

**display: flex;**

**flex-direction: column;**

**align-items: center;**

**justify-content: flex-start;**

**color: #fff;**

**overflow: hidden;**

**max-height: 600px;**

**max-width: 350px;**

**margin: auto;**

**padding: 16px;**

**user-select: none;**

**}**

**h1 {**

**margin-bottom: 8px;**

**font-weight: 700;**

**font-size: 1.6rem;**

**text-align: center;**

**text-shadow: 1px 1px 5px rgba(0,0,0,0.5);**

**}**

**#upload-label {**

**background: rgba(255 255 255 / 0.15);**

**border: 2px dashed rgba(255 255 255 / 0.5);**

**border-radius: 12px;**

**padding: 18px;**

**cursor: pointer;**

**font-weight: 600;**

**font-size: 1rem;**

**width: 100%;**

**max-width: 320px;**

**margin-bottom: 20px;**

**transition: background-color 0.3s, border-color 0.3s;**

**display: flex;**

**flex-direction: column;**

**align-items: center;**

**justify-content: center;**

**}**

**#upload-label:hover, #upload-label:focus {**

**background: rgba(255 255 255 / 0.3);**

**border-color: #fff;**

**outline: none;**

**}**

**#file-input {**

**display: none;**

**}**

**#image-container {**

**position: relative;**

**width: 100%;**

**max-width: 320px;**

**max-height: 360px;**

**border-radius: 14px;**

**overflow: hidden;**

**box-shadow: 0 4px 15px rgba(0,0,0,0.5);**

**background: #222;**

**flex-shrink: 0;**

**}**

**#uploaded-image {**

**max-width: 100%;**

**max-height: 360px;**

**display: block;**

**margin: 0 auto;**

**user-select: none;**

**-webkit-user-drag: none;**

**touch-action: none;**

**cursor: crosshair;**

**border-radius: 14px;**

**image-rendering: pixelated;**

**}**

**#color-info {**

**margin-top: 20px;**

**width: 100%;**

**max-width: 320px;**

**background: rgba(255 255 255 / 0.1);**

**border-radius: 12px;**

**padding: 12px 20px;**

**box-shadow: 0 2px 10px rgba(0,0,0,0.2);**

**color: #fff;**

**text-align: center;**

**user-select: text;**

**}**

**#color-swatch {**

**width: 60px;**

**height: 60px;**

**border-radius: 12px;**

**margin: 0 auto 12px auto;**

**box-shadow: 0 0 8px rgba(0,0,0,0.3);**

**border: 2px solid #fff;**

**}**

**#instructions {**

**font-size: 0.9rem;**

**color: rgba(255 255 255 / 0.8);**

**margin-top: 12px;**

**user-select: none;**

**}**

**/\* Responsive text for smaller width \*/**

**@media (max-width: 350px) {**

**h1 {**

**font-size: 1.4rem;**

**}**

**#color-info {**

**font-size: 0.9rem;**

**}**

**}**

**</style>**

**</head>**

**<body>**

**<h1>Color Detection App</h1>**

**<label id="upload-label" for="file-input" tabindex="0">**

**Tap to upload an image**

**<input type="file" accept="image/\*" id="file-input" />**

**</label>**

**<div id="image-container" aria-label="Uploaded image area where color detection can be performed">**

**<canvas id="image-canvas" style="display:none;"></canvas>**

**<img id="uploaded-image" alt="Uploaded image for color detection" />**

**</div>**

**<div id="color-info" aria-live="polite" aria-atomic="true" style="display:none;">**

**<div id="color-swatch" aria-hidden="true"></div>**

**<div id="color-values">**

**<div><strong>HEX:</strong> <span id="hex-value"></span></div>**

**<div><strong>RGB:</strong> <span id="rgb-value"></span></div>**

**</div>**

**</div>**

**<div id="instructions">Tap or click on the image to detect color</div>**

**<script>**

**(function() {**

**const fileInput = document.getElementById('file-input');**

**const uploadedImage = document.getElementById('uploaded-image');**

**const imageContainer = document.getElementById('image-container');**

**const colorInfo = document.getElementById('color-info');**

**const colorSwatch = document.getElementById('color-swatch');**

**const hexValue = document.getElementById('hex-value');**

**const rgbValue = document.getElementById('rgb-value');**

**const imageCanvas = document.getElementById('image-canvas');**

**const ctx = imageCanvas.getContext('2d');**

**function rgbToHex(r, g, b) {**

**return "#" + [r, g, b].map(x => {**

**const hex = x.toString(16);**

**return hex.length === 1 ? "0" + hex : hex;**

**}).join('');**

**}**

**function showColorInfo(r, g, b) {**

**const hex = rgbToHex(r, g, b);**

**colorSwatch.style.backgroundColor = hex;**

**hexValue.textContent = hex.toUpperCase();**

**rgbValue.textContent = `rgb(${r}, ${g}, ${b})`;**

**colorInfo.style.display = 'block';**

**}**

**function getColorAtPoint(x, y) {**

**try {**

**const pixel = ctx.getImageData(x, y, 1, 1).data;**

**return {r: pixel[0], g: pixel[1], b: pixel[2]};**

**} catch(e) {**

**// If out of bounds or not ready**

**return null;**

**}**

**}**

**// On file upload, read the image and display**

**fileInput.addEventListener('change', function(event) {**

**const file = event.target.files[0];**

**if (!file) return;**

**const reader = new FileReader();**

**reader.onload = function(e) {**

**const img = new Image();**

**img.onload = function() {**

**// Resize canvas to image size**

**imageCanvas.width = img.naturalWidth;**

**imageCanvas.height = img.naturalHeight;**

**ctx.drawImage(img, 0, 0);**

**// Show the image element for interaction & set src**

**uploadedImage.src = e.target.result;**

**uploadedImage.style.display = 'block';**

**// Clear any previous color info**

**colorInfo.style.display = 'none';**

**colorSwatch.style.backgroundColor = 'transparent';**

**// Adjust styles for the image container's scroll if needed**

**imageContainer.scrollTop = 0;**

**imageContainer.scrollLeft = 0;**

**};**

**img.src = e.target.result;**

**};**

**reader.readAsDataURL(file);**

**});**

**// Handle clicks/taps on the image to detect color**

**function handleClickOrTouch(evt) {**

**if (!uploadedImage.src) return;**

**evt.preventDefault();**

**let rect = uploadedImage.getBoundingClientRect();**

**let clientX, clientY;**

**if (evt.touches && evt.touches.length > 0) {**

**clientX = evt.touches[0].clientX;**

**clientY = evt.touches[0].clientY;**

**} else {**

**clientX = evt.clientX;**

**clientY = evt.clientY;**

**}**

**// Calculate scale factor if image is scaled**

**const scaleX = uploadedImage.naturalWidth / uploadedImage.clientWidth;**

**const scaleY = uploadedImage.naturalHeight / uploadedImage.clientHeight;**

**// Coordinates relative to image**

**const x = Math.floor((clientX - rect.left) \* scaleX);**

**const y = Math.floor((clientY - rect.top) \* scaleY);**

**if (x < 0 || y < 0 || x >= imageCanvas.width || y >= imageCanvas.height) {**

**return;**

**}**

**const color = getColorAtPoint(x, y);**

**if (color) {**

**showColorInfo(color.r, color.g, color.b);**

**}**

**}**

**uploadedImage.addEventListener('click', handleClickOrTouch);**

**uploadedImage.addEventListener('touchstart', handleClickOrTouch);**

**// Accessibility: Trigger file input on keyboard enter or space on label**

**const uploadLabel = document.getElementById('upload-label');**

**uploadLabel.addEventListener('keydown', e => {**

**if(e.key === 'Enter' || e.key === ' ') {**

**e.preventDefault();**

**fileInput.click();**

**}**

**});**

**})();**

**</script>**

**</body>**

**</html>**

