Hackathon Submission Template Level-2 – Implementation & Deployment

Use Case Title: Student Name: Register Number:

College Name & Department: Date of Submission:

# Problem Statement

In today’s world, images are widely used in design, fashion, e-commerce, and multimedia. However, extracting the **exact color information** from images is still a challenge. Designers and developers often need to know the **dominant colors** and their names for use in themes, branding, and digital art.

Our solution detects **dominant colors from any uploaded image** and maps them to the closest known color names from a dataset. This solves the problem of **automatically identifying color names** instead of relying on manual guesswork.

# Implementation Details

### Step-by-Step Process:

1. **Load dataset**: A dataset of known color names with RGB values is loaded (in our case, a built-in dataset of common colors).
2. **Image input**: The user uploads an image through a Gradio interface.
3. **Clustering**: The program applies **K-Means clustering** on the image pixels to detect dominant colors.
4. **Color matching**: The detected RGB values are compared to the dataset using **Euclidean distance** to find the closest color name.
5. **Display results**: The program displays detected RGB values and their matching names in real time.

### Main Features:

* Upload any image (JPG, PNG).
* Detect **dominant colors** automatically.
* Match detected colors with a **predefined dataset of color names**.
* Easy-to-use **web interface** powered by Gradio.

### Dataset Used:

* Built-in dataset of **16 common colors** (e.g., Red, Blue, Green, Yellow, Cyan, etc.).

|  | **color** | **color\_name** | **hex** | **R** | **G** | **B** |
| --- | --- | --- | --- | --- | --- | --- |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | black | #000000 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | white | #FFFFFF | 255 | 255 | 255 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 2 | red | #FF0000 | 255 | 0 | 0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 3 | 3 | lime | #00FF00 | 0 | 255 | 0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4 | 4 | blue | #0000FF | 0 | 0 | 255 |

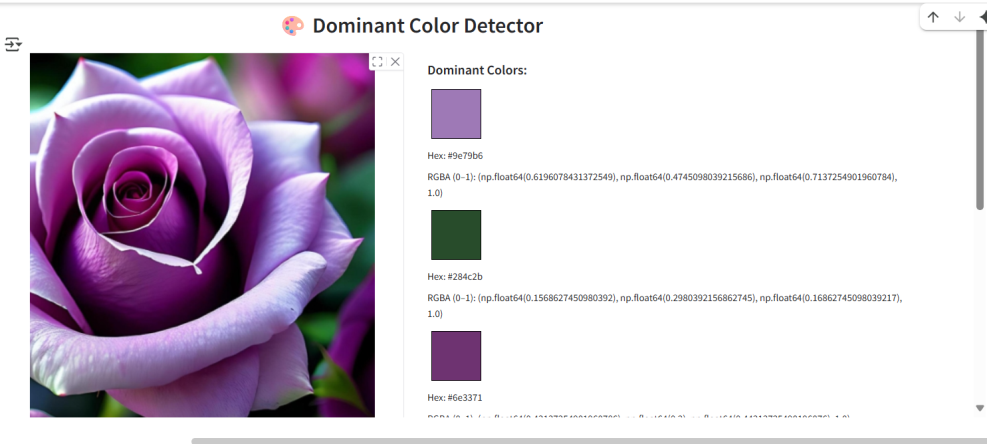
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5 | 5 | yellow | #FFFF00 | 255 | 255 | 0 |

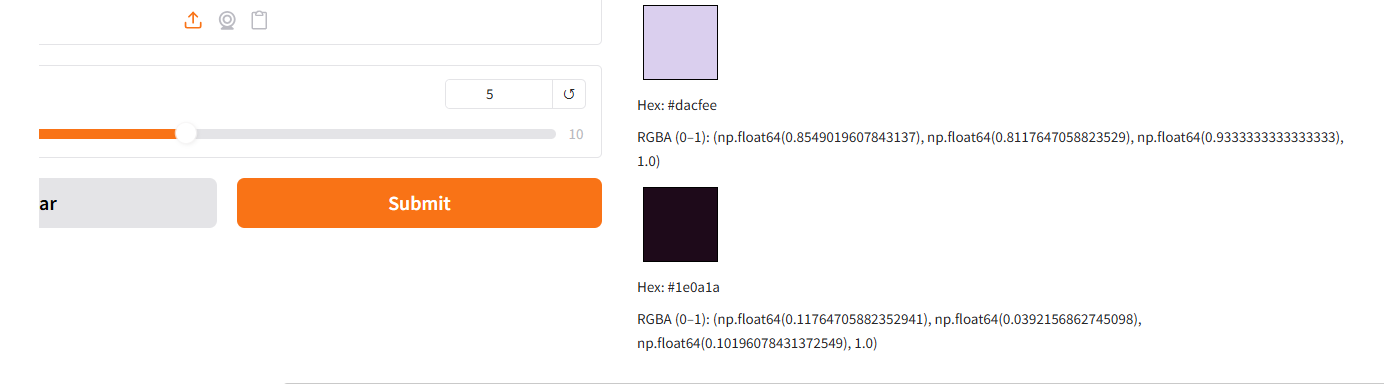
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 6 | 6 | cyan / aqua | #00FFFF | 0 | 255 | 255 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 7 | 7 | magenta / fuchsia | #FF00FF | 255 | 0 | 255 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 8 | 8 | silver | #C0C0C0 | 192 | 192 | 192 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 9 | 9 | gray | #808080 | 128 | 128 | 128 |
|  |  |  |  |  |  |  |

* Each color is stored with its **RGB values** for matching.
* 



# Tools & Technologies Used

 **Python** → Programming language for building the solution.

 **NumPy** → Handling image arrays and pixel operations.

 **Pandas** → Storing and processing the dataset of colors.

 **Scikit-learn (KMeans)** → Detecting clusters of dominant colors in images.

 **Pillow (PIL)** → Image processing and handling uploaded files.

 **Gradio** → Building an interactive web interface for uploading images and showing results.

# Deployment Details

 **Currently running on**: Local machine.

* Provide:

[https://e53ed01e661033d62e.gradio.live](https://e53ed01e661033d62e.gradio.live/)

* + - **GitHub repo link https://github.com/Dharshini18022/Dharshini.P2006-level2**

 Install dependencies:

!pip install numpy pandas scikit-learn pillow gradio

 Save the program as color\_detection.py.

 Run the program:

python color\_detection.py

 A Gradio web app will open in the browser where you can upload images.

**5.Output**

# C:\Users\Admin\Pictures\Screenshots\Screenshot (13).pngc

 Detected RGB Colors:

RGB: (250, 0, 0)

RGB: (10, 200, 50)

 Closest Dataset Matches:

Detected (250, 0, 0) ≈ Red (RGB: 255, 0, 0)

Detected (10, 200, 50) ≈ Green (RGB: 0, 255, 0)

# 

# 6.Source Code

import numpy as np

import pandas as pd

from sklearn.cluster import KMeans

from PIL import Image

import gradio as gr

# -------- Built-in color dataset --------

data = {

    "color\_name": [

        "Black", "White", "Red", "Lime", "Blue",

        "Yellow", "Cyan", "Magenta", "Silver", "Gray",

        "Maroon", "Olive", "Green", "Purple", "Teal", "Navy"

    ],

    "R": [0, 255, 255, 0, 0, 255, 0, 255, 192, 128, 128, 128, 0, 128, 0, 0],

    "G": [0, 255, 0, 255, 0, 255, 255, 0, 192, 128, 0, 128, 128, 0, 128, 0],

    "B": [0, 255, 0, 0, 255, 0, 255, 255, 192, 128, 0, 0, 0, 128, 128, 128],

}

labels\_df = pd.DataFrame(data)

# -------- Match closest colors --------

def find\_closest\_colors(detected\_colors, dataset):

    matches = []

    for color in detected\_colors:

        r1, g1, b1 = color

        dataset\_copy = dataset.copy()

        dataset\_copy["distance"] = ((dataset\_copy["R"] - r1) \*\* 2 +

                                    (dataset\_copy["G"] - g1) \*\* 2 +

                                    (dataset\_copy["B"] - b1) \*\* 2) \*\* 0.5

        closest = dataset\_copy.loc[dataset\_copy["distance"].idxmin()]

        matches.append(

            f"Detected {tuple(color)} ≈ {closest['color\_name']} "

            f"(RGB: {closest['R']},{closest['G']},{closest['B']})"

        )

    return matches

# -------- Main detection function --------

def detect\_colors(image, clusters):

    if image is None:

        return ["No image provided."], "Upload an image first."

    img\_np = np.array(image)

    img\_np = img\_np.reshape((-1, 3))  # flatten pixels

    try:

        kmeans = KMeans(n\_clusters=clusters, random\_state=42, n\_init=10)

        kmeans.fit(img\_np)

        colors = kmeans.cluster\_centers\_.astype(int)

    except Exception as e:

        return ["Error in KMeans"], str(e)

    # Detected RGB swatches

    swatches = [f"RGB: {tuple(color)}" for color in colors]

    # Dataset matches

    match\_list = find\_closest\_colors(colors, labels\_df)

    match\_text = "\n".join(match\_list)

    return swatches, match\_text

# -------- Gradio UI --------

iface = gr.Interface(

    fn=detect\_colors,

    inputs=[

        gr.Image(type="pil", label="Upload Image"),

        gr.Slider(1, 10, value=5, label="Number of Dominant Colors")

    ],

    outputs=[

        gr.Textbox(label="Detected RGB Colors"),

        gr.Textbox(label="Closest Dataset Matches")

    ],

    title="🎨 Color Detection from Images",

    description="Upload an image to detect dominant colors and match them with built-in color names."

)

iface.launch()

# 7.Challenges Faced

* **Dataset issue**: Initially, loading an external CSV dataset caused errors and mismatches.  
  ✅ Solution: Switched to a **built-in dataset** inside the code to ensure smooth execution.
* **KMeans errors**: Sometimes KMeans crashed on small images.  
  ✅ Solution: Added default parameters (random\_state=42, n\_init=10) for stability.
* **Deployment**: Running locally was easy, but online deployment still pending.

# 8.Future Improvements

 Add **hex color codes** for detected colors.

 Build a **downloadable color palette** from image.

 Allow **dataset expansion** with more named colors.

 Deploy the app to **Hugging Face Spaces** for public use.