به نام خدا

استاد : دکتر امینی

دانشجو : عرفان میرزازاده

https://github.com/ahuraERFAN

عنوان پروژه : اپلیکیشن تشخیص پالت رنگی از روی عکس.

**طبق هر بخش دیگه ای از تمرین پروژه هم کتابخانه شاملش میشه کتابخانه های انتخاب شده برای پروژه در ذیل آمده اند.**

    # tkinter: کتابخانه استاندارد پایتون برای ایجاد رابط کاربری گرافیکی (GUI)

import tkinter as tk

# filedialog از tkinter: برای ایجاد دیالوگ‌های انتخاب فایل

from tkinter import filedialog

# ttk از tkinter: ویجت‌های پیشرفته‌تر با ظاهر بهتر

from tkinter import ttk

# PIL (Python Imaging Library): برای کار با تصاویر و پردازش آن‌ها

from PIL import Image  # کلاس اصلی برای کار با تصاویر

from PIL import ImageTk  # برای نمایش تصاویر در رابط tkinter

# numpy: برای محاسبات عددی و کار با آرایه‌های چندبعدی

import numpy as np

# matplotlib.pyplot: برای ایجاد نمودارها و نمایش گرافیکی داده‌ها

import matplotlib.pyplot as plt

# KMeans از sklearn: الگوریتم خوشه‌بندی برای گروه‌بندی رنگ‌های مشابه

from sklearn.cluster import KMeans

# shuffle از sklearn: برای تصادفی کردن ترتیب داده‌ها

from sklearn.utils import shuffle

# Counter از collections: برای شمارش تعداد عناصر مشابه

from collections import Counter

# BytesIO از io: برای کار با داده‌های باینری در حافظه

from io import BytesIO

# colorsys: برای تبدیل بین مدل‌های رنگی مختلف (RGB به HSV و بالعکس)

import colorsys

# KDTree از scipy.spatial: برای جستجوی کارآمد نزدیک‌ترین همسایه‌ها در فضای چندبعدی

from scipy.spatial import KDTree

**بعد از کتابخوانه حال نوبت اولین کلاس ، کلاس استخراج پالت هاست:**

class ColorPaletteExtractor:

    def \_\_init\_\_(self):

        self.COLOR\_THRESHOLD = 45

        self.MIN\_SATURATION = 0.15

        self.MIN\_VALUE = 0.15

        self.PALETTE\_SIZE = 8

    def extract\_dominant\_colors(self, image, k=16):

        """استخراج رنگ‌های غالب با K-Means++"""

        image = image.resize((400, 400))

        pixels = np.array(image).reshape(-1, 3)

        sample\_size = 2000

        if len(pixels) > sample\_size:

            pixels = shuffle(pixels, random\_state=42)[:sample\_size]

        kmeans = KMeans(

            n\_clusters=k,

            init='k-means++',

            n\_init=20,

            max\_iter=300,

            tol=1e-4,

            random\_state=42

        )

        kmeans.fit(pixels)

        counts = Counter(kmeans.labels\_)

        total = sum(counts.values())

        return [

            (kmeans.cluster\_centers\_[i].astype(int), counts[i]/total)

            for i in range(k)

        ]

    def get\_hsv\_ranges(self):

        """محدوده‌های HSV برای رنگ‌های کلیدی"""

        return {

            'red': [

                {'lower': (0, self.MIN\_SATURATION, self.MIN\_VALUE), 'upper': (0.05, 1.0, 1.0)},

                {'lower': (0.9, self.MIN\_SATURATION, self.MIN\_VALUE), 'upper': (1.0, 1.0, 1.0)}

            ],

            'green': [{'lower': (0.25, self.MIN\_SATURATION, self.MIN\_VALUE), 'upper': (0.45, 1.0, 1.0)}],

            'blue': [{'lower': (0.55, self.MIN\_SATURATION, self.MIN\_VALUE), 'upper': (0.75, 1.0, 1.0)}],

            'yellow': [{'lower': (0.10, self.MIN\_SATURATION, self.MIN\_VALUE), 'upper': (0.20, 1.0, 1.0)}],

            'purple': [{'lower': (0.75, self.MIN\_SATURATION, self.MIN\_VALUE), 'upper': (0.85, 1.0, 1.0)}],

            'orange': [{'lower': (0.05, self.MIN\_SATURATION, self.MIN\_VALUE), 'upper': (0.10, 1.0, 1.0)}],

            'black': [{'lower': (0, 0, 0), 'upper': (1.0, 1.0, 0.20)}],

            'white': [{'lower': (0, 0, 0.85), 'upper': (1.0, 0.25, 1.0)}]

        }

    def detect\_special\_colors(self, pixels):

        """تشخیص رنگ‌های خاص"""

        hsv\_pixels = np.array([colorsys.rgb\_to\_hsv(\*pixel) for pixel in pixels/255.0])

        detected = set()

        for color\_name, ranges in self.get\_hsv\_ranges().items():

            for r in ranges:

                lower = np.array(r['lower'])

                upper = np.array(r['upper'])

                mask = np.ones(len(hsv\_pixels), dtype=bool)

                for i in range(3):

                    if lower[i] <= upper[i]:

                        mask &= (hsv\_pixels[:,i] >= lower[i]) & (hsv\_pixels[:,i] <= upper[i])

                    else:

                        mask &= (hsv\_pixels[:,i] >= lower[i]) | (hsv\_pixels[:,i] <= upper[i])

                if np.any(mask):

                    detected.add(color\_name)

                    break

        return detected

    def merge\_similar\_colors(self, colors):

        """ادغام رنگ‌های مشابه"""

        if colors.size == 0:

            return []

        kdtree = KDTree(colors)

        clusters = []

        used = set()

        for i, color in enumerate(colors):

            if i not in used:

                neighbors = kdtree.query\_ball\_point(color, r=self.COLOR\_THRESHOLD)

                cluster = [colors[j] for j in neighbors]

                clusters.append(np.mean(cluster, axis=0).astype(int))

                used.update(neighbors)

        return clusters

    def generate\_palette(self, image):

        """تولید پالت نهایی"""

        dominant\_colors = self.extract\_dominant\_colors(image)

        dominant\_colors.sort(key=lambda x: -x[1])

        raw\_colors = np.array([c[0] for c in dominant\_colors])

        pixels = np.array(image.resize((100, 100))).reshape(-1, 3)

        special\_colors = self.detect\_special\_colors(pixels)

        color\_map = {

            'red': (255, 0, 0),

            'green': (0, 255, 0),

            'blue': (0, 0, 255),

            'yellow': (255, 255, 0),

            'purple': (128, 0, 128),

            'orange': (255, 165, 0),

            'black': (0, 0, 0),

            'white': (255, 255, 255)

        }

        special\_rgb = [color\_map[c] for c in special\_colors if c in color\_map]

        if special\_rgb:

            all\_colors = np.vstack([raw\_colors, special\_rgb])

        else:

            all\_colors = raw\_colors

        merged\_colors = self.merge\_similar\_colors(all\_colors)

        final\_palette = []

        for color in merged\_colors:

            if len(final\_palette) >= self.PALETTE\_SIZE:

                break

            if not any(np.linalg.norm(color - c) < self.COLOR\_THRESHOLD for c in final\_palette):

                final\_palette.append(tuple(color))

        return final\_palette[:self.PALETTE\_SIZE]

* **مقادیر اولیه**:
  + تنظیم پارامترهای پایه مانند حداقل فاصله رنگ‌ها (COLOR\_THRESHOLD)، حداقل اشباع و روشنایی برای تشخیص رنگ‌های خاص
* **استخراج رنگ‌های غالب**:
  + با الگوریتم K-Means رنگ‌های اصلی تصویر را شناسایی می‌کند
  + از نمونه‌گیری تصادفی برای بهینه‌سازی سرعت استفاده می‌کند
* **تشخیص رنگ‌های خاص**:
  + با تبدیل RGB به HSV، رنگ‌های کلیدی مانند قرمز، سبز، آبی و... را تشخیص می‌دهد
  + از محدوده‌های از پیش تعریف شده در فضای HSV استفاده می‌کند
* **ادغام رنگ‌های مشابه**:
  + با درخت KD، رنگ‌های نزدیک به هم را ادغام می‌کند
  + از فاصله اقلیدسی برای سنجش شباهت رنگ‌ها استفاده می‌کند
* **تولید پالت نهایی**:
  + ترکیب هوشمندانه رنگ‌های غالب و رنگ‌های خاص
  + انتخاب نهایی ۸ رنگ متمایز و مهم تصویر

این کلاس با ترکیب هوشمندانه تکنیک‌های پردازش تصویر و یادگیری ماشین، پالت‌های رنگی دقیقی تولید می‌کند.

پس یکی از کاربرد های اصلی داده کاوی در تشخیص پالت های رنگی است به طور مثال الگوریتم کی مینز.

**در بخش بعدی کلاسی ایجاد میکنیم که یک رابط کاربری گرافیکی با UI/UX فوق العاده ساده باشد :**

class AppGUI:

    def \_\_init\_\_(self, root):

        self.root = root

        self.extractor = ColorPaletteExtractor()

        self.setup\_ui()

    def setup\_ui(self):

        self.root.title("Professional Color Palette Extractor")

        self.root.geometry("1200x800")

        self.root.configure(bg="#333")

        style = ttk.Style()

        style.theme\_use("clam")

        style.configure("TButton", font=("Helvetica", 12), padding=6)

        style.map("TButton", background=[("active", "#666"), ("!disabled", "#444")])

        main\_frame = ttk.Frame(self.root)

        main\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

        control\_frame = ttk.Frame(main\_frame)

        control\_frame.pack(fill=tk.X, pady=10)

        self.open\_btn = ttk.Button(control\_frame, text="Open Image", command=self.load\_image)

        self.open\_btn.pack(side=tk.LEFT)

        display\_frame = ttk.Frame(main\_frame)

        display\_frame.pack(fill=tk.BOTH, expand=True)

        self.image\_panel = ttk.Label(display\_frame, background="#444")

        self.image\_panel.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=10)

        self.palette\_panel = ttk.Label(display\_frame, background="#fff")

        self.palette\_panel.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=10)

    def load\_image(self):

        file\_path = filedialog.askopenfilename(filetypes=[

            ("Image Files", "\*.jpg \*.jpeg \*.png \*.bmp \*.tiff")

        ])

        if not file\_path:

            return

        try:

            image = Image.open(file\_path).convert("RGB")

            self.display\_image(image)

            palette = self.extractor.generate\_palette(image)

            self.display\_palette(palette)

        except Exception as e:

            print(f"Error processing image: {e}")

    def display\_image(self, image):

        image.thumbnail((600, 600))

        img\_tk = ImageTk.PhotoImage(image)

        self.image\_panel.config(image=img\_tk)

        self.image\_panel.image = img\_tk

    def display\_palette(self, colors):

        fig, ax = plt.subplots(figsize=(10, 2))

        ax.set\_title("Color Palette", pad=20, fontsize=14)

        for i, color in enumerate(colors):

            ax.add\_patch(plt.Rectangle((i, 0), 1, 1, color=np.array(color)/255))

            hex\_code = '#%02x%02x%02x' % color

            ax.text(i + 0.5, -0.3, hex\_code, ha='center', va='top', fontsize=10)

        ax.set\_xlim(0, len(colors))

        ax.set\_ylim(-0.5, 1)

        ax.axis("off")

        buf = BytesIO()

        plt.savefig(buf, format='png', bbox\_inches='tight', dpi=100)

        plt.close(fig)

        buf.seek(0)

        palette\_img = Image.open(buf)

        palette\_tk = ImageTk.PhotoImage(palette\_img)

        self.palette\_panel.config(image=palette\_tk)

        self.palette\_panel.image = palette\_tk

1. **بخش تنظیمات اولیه**:
   * ایجاد پنجره اصلی با ابعاد 1200x800 پیکسل
   * تنظیم تم و استایل دکمه‌ها (رنگ، فونت و اندازه)
   * ساخت چارچوب‌های اصلی برای نمایش تصویر و پالت
2. **وظایف اصلی**:
   * دکمه Open Image برای انتخاب فایل تصویری
   * نمایش تصویر ورودی در سمت چپ پنجره (با اندازه حداکثر 600x600)
   * نمایش پالت رنگی استخراج شده در سمت راست
3. **پردازش تصویر**:
   * فراخوانی کلاس ColorPaletteExtractor برای تحلیل تصویر
   * نمایش خطاها در صورت بروز مشکل
4. **نمایش پالت**:
   * ایجاد گرافیک حرفه‌ای با matplotlib
   * نمایش مستطیل‌های رنگی به همراه کد HEX هر رنگ
   * تبدیل نمودار به تصویر برای نمایش در رابط کاربری
5. **مدیریت حافظه**:
   * ذخیره صحیح تصاویر برای جلوگیری از مشکلات مرجع‌دهی
   * استفاده از حافظه موقت (BytesIO) برای پردازش سریع‌تر

این کلاس با ترکیب tkinter برای GUI و matplotlib برای نمایش حرفه‌ای، تجربه کاربری ساده و موثری ایجاد می‌کند.

**در ادامه برنامه را اجرا میکنیم :**

if \_\_name\_\_ == "\_\_main\_\_":

    root = tk.Tk()

    app = AppGUI(root)

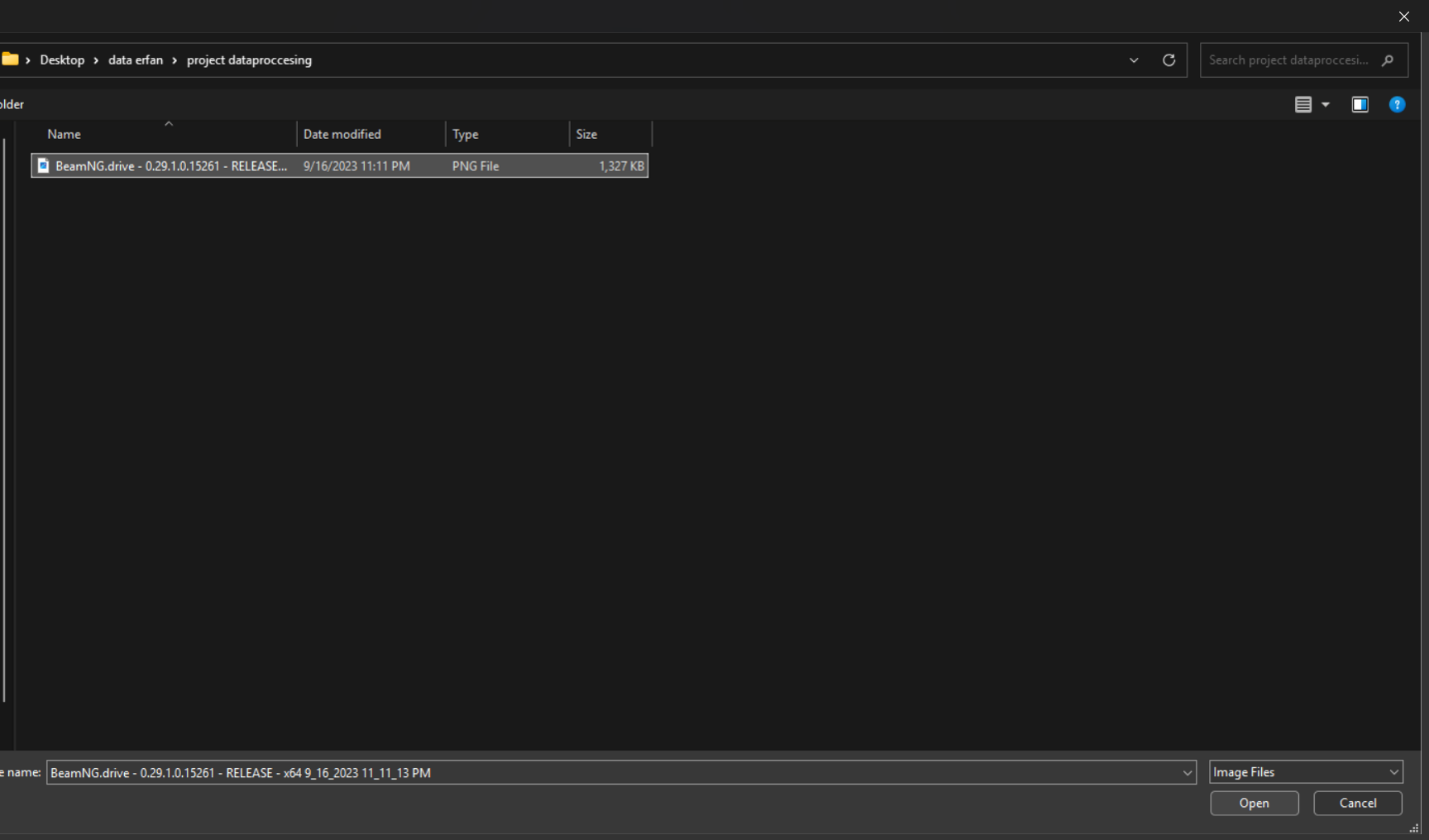
    root.mainloop()

بخش جذاب اجرای برنامه :

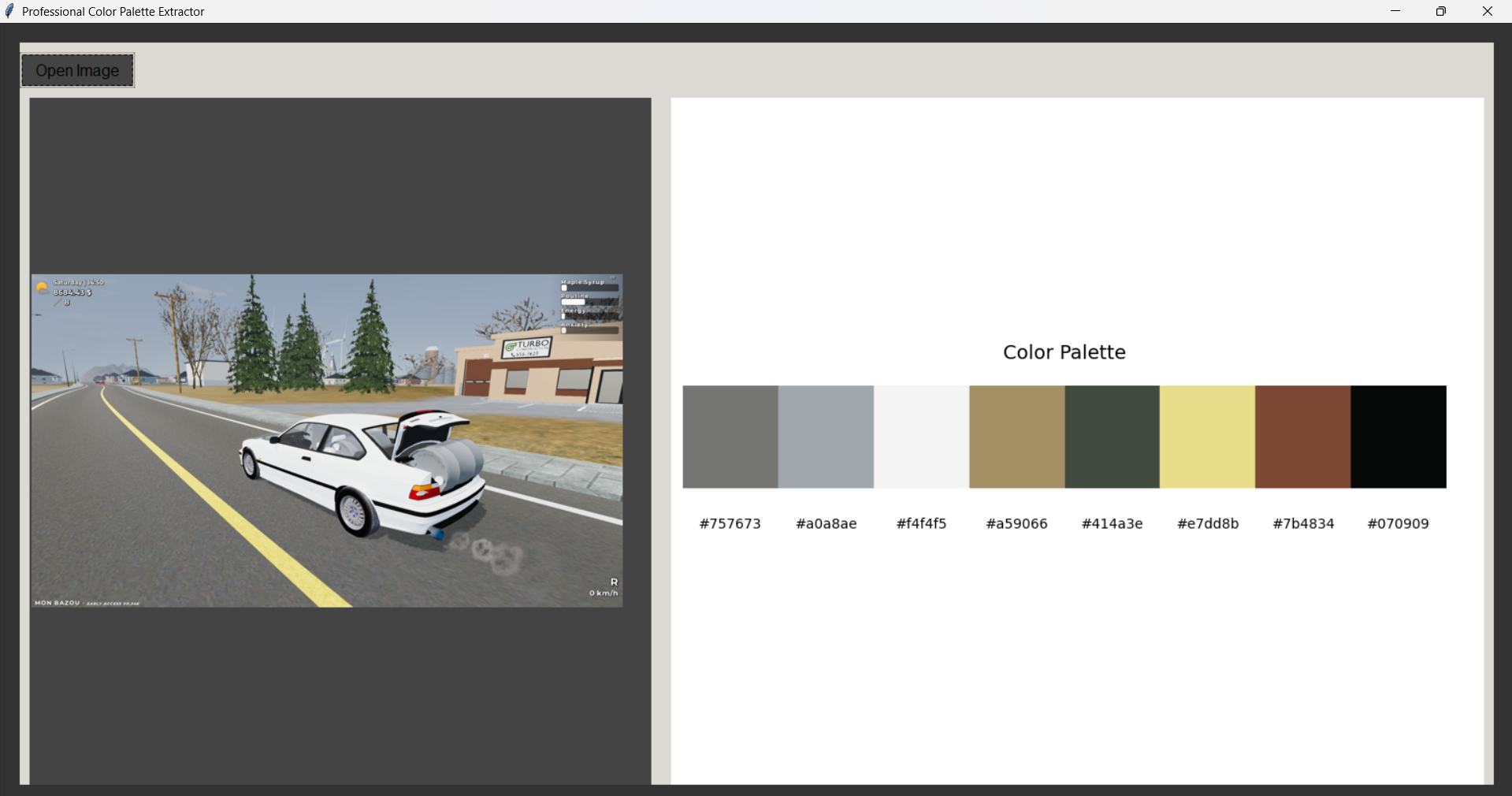
محیط کاری برنامه :



طریقه اجرای برنامه:

1. کاربر عکس خود را آپلود میکند.
2. پالت رنگی به همراه عکس با کد هگز مربوطه نوشته میشود.
3. کاربر میتواند دوباره عکس خود را آپلود کند.
4. آپلود:

نمایش پالت :



**تمامی الگوریتم های استفاده شده :**

1. **الگوریتم K-Means++ برای خوشه‌بندی رنگ‌ها**:
   * نسخه بهبودیافته K-Means با مقداردهی اولیه هوشمند
   * پارامترها: n\_clusters=16, n\_init=20, max\_iter=300
   * وظیفه: گروه‌بندی پیکسل‌های مشابه و یافتن رنگ‌های غالب
2. **تبدیل فضای رنگی RGB به HSV**:
   * استفاده از تابع colorsys.rgb\_to\_hsv()
   * مزیت: تشخیص بهتر رنگ‌های خاص با جدا کردن Hue از Saturation و Value
3. **الگوریتم جستجوی KD-Tree**:
   * scipy.spatial.KDTree برای یافتن نزدیک‌ترین همسایه‌ها
   * کاربرد: ادغام رنگ‌های مشابه با فاصله اقلیدسی کمتر از 45
4. **نمونه‌گیری تصادفی (Random Sampling)**:
   * استفاده از sklearn.utils.shuffle
   * کاهش حجم داده‌ها از ~160,000 به 2,000 نمونه برای پردازش سریع‌تر
5. **الگوریتم‌های پردازش تصویر**:
   * تغییر اندازه هوشمند با resize( Pillow و thumbnai)
   * تبدیل تصویر به آرایه numpy برای پردازش ماتریسی
6. **الگوریتم تشخیص رنگ‌های خاص**:
   * بررسی محدوده‌های HSV برای رنگ‌های کلیدی (قرمز، سبز، آبی و...)
   * استفاده از ماسک‌های منطقی برای شناسایی پیکسل‌های هر محدوده
7. **الگوریتم ادغام رنگ‌ها**:
   * محاسبه میانگین رنگ‌های مشابه
   * حذف تکراری‌ها با فاصله اقلیدسی
8. **الگوریتم انتخاب رنگ‌های نهایی**:
   * ترکیب رنگ‌های غالب و رنگ‌های خاص
   * انتخاب 8 رنگ متمایز با بیشترین فراوانی
9. **الگوریتم‌های نمایش گرافیکی**:
   * ساخت نمودار پالت با matplotlib
   * تبدیل نمودار به تصویر در حافظه (BytesIO)

تمامی این الگوریتم‌ها با همکاری یکدیگر:

1. تصویر ورودی را تحلیل می‌کنند.
2. رنگ‌های اصلی را شناسایی می‌کنند.
3. نتایج را به صورت بصری نمایش می‌دهند.

 مراحل اجرا

1. دریافت تصویر ورودی
2. تغییر اندازه تصویر به ابعاد بهینه (400×400 پیکسل)
3. استخراج رنگ‌های غالب با الگوریتم K-Means++
4. تشخیص رنگ‌های خاص در فضای HSV
5. ادغام رنگ‌های مشابه
6. انتخاب نهایی ۸ رنگ برتر
7. نمایش نتایج

محدودیت های بالا همگی بخاطر حجم سخت افزار موجود انتخاب شده است برای مثال میتوان گزینه ای اضافه کرد که نسبت به سخت افزار کاربر محدودیت هارا کاهش دهد.