**داکیومنت تکلیف شماره 2**

* **سوال 1**

**الف)**

import numpy as np

# Define the RGB color

rgb\_color = np.array([50, 70, 130], dtype=np.uint8)

def RGB2CMYK(rgb\_color):

    R, G, B = rgb\_color / 255

    K = 1 - max(R, G, B)

    C = (1 - R - K) / (1 - K)

    M = (1 - G - K) / (1 - K)

    Y = (1 - B - K) / (1 - K)

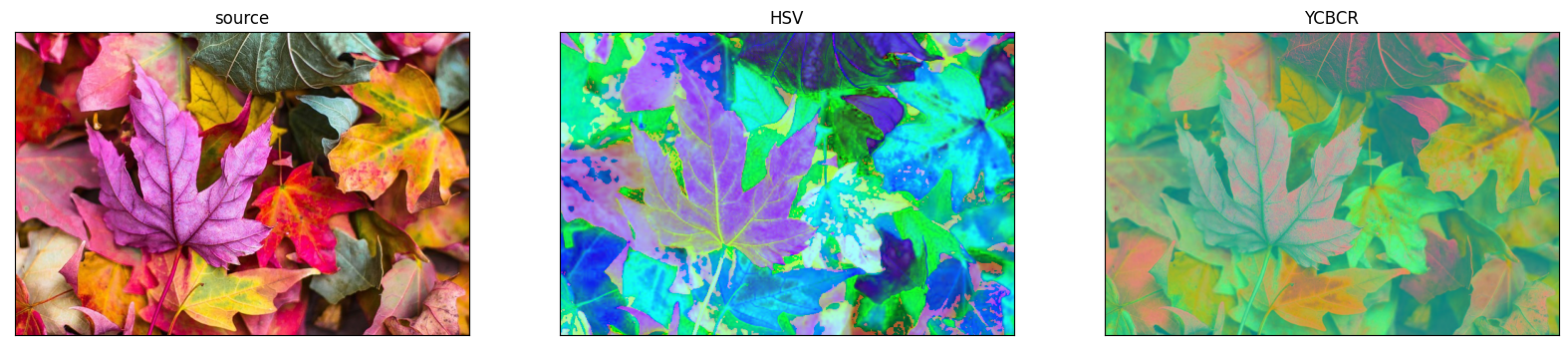
    return np.array([C, M, Y, K])

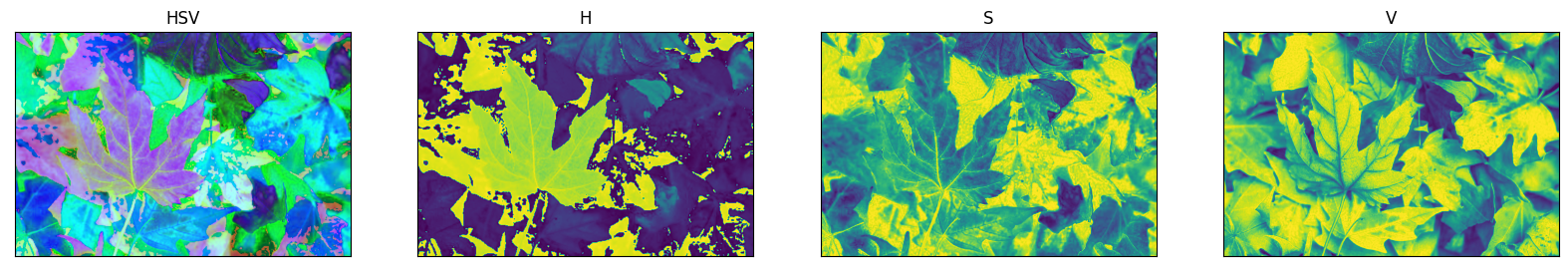
CMYK\_color = RGB2CMYK(rgb\_color)

print(CMYK\_color)

**این کد یک تابع با نام RGB2CMYK است که مقداری رنگ RGB به عنوان ورودی دریافت می کند. در این تابع، رنگ RGB در ابتدا به مقادیر معادل در بازه 0 تا 1 تقسیم می شود. سپس مقدار کلیدی K که برای CMYK استفاده می شود، به عنوان ماکسیمم مقدار از مقادیر R، G و B محاسبه می شود. سپس مقادیر C، M و Y با توجه به مقادیر R، G، B و K محاسبه می شوند. در نهایت، یک آرایه چهارتایی شامل مقادیر CMYK به عنوان خروجی تابع بازگردانده می شود.**

**در این کد، یک رنگ RGB با مقادیر (50، 70، 130) تعریف شده و سپس تابع RGB2CMYK با این مقدار به عنوان ورودی فراخوانی شده است. نتیجه به دست آمده برای CMYK رنگ با استفاده از مقدار ورودی RGB، در خط آخر با استفاده از تابع پرینت چاپ می شود.**

**ب)** ****

**ج)** ****

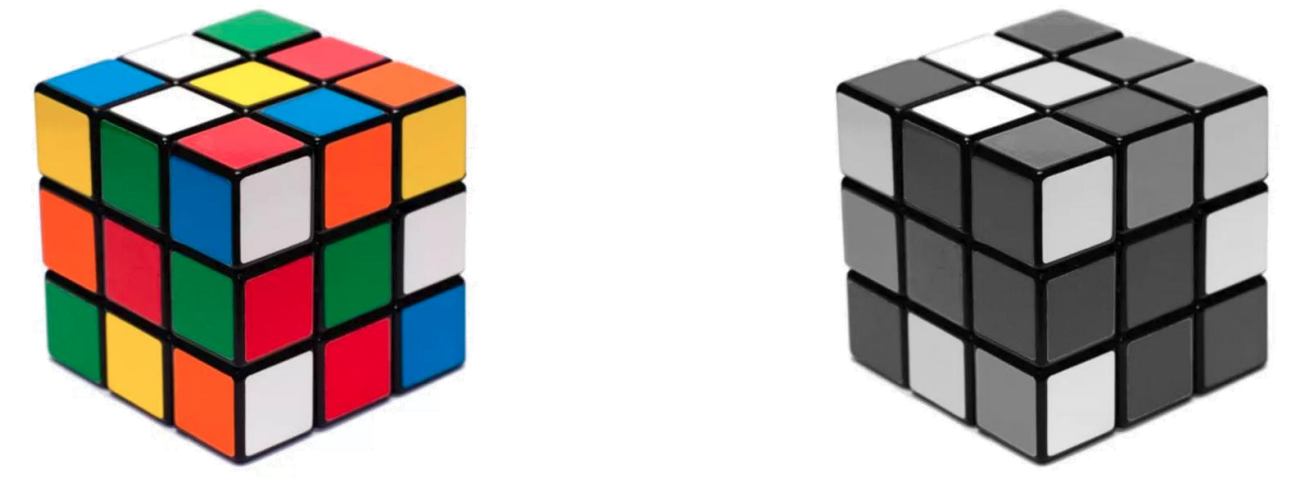
**د) این کد یک تابع به نام dif دارد که دو تصویر ورودی را گرفته، آن‌ها را با یکدیگر مقایسه کرده و تصویری را بازگردانده که تفاوت بین آن دو را نشان می‌دهد. برای این کار، این تابع ابتدا تصاویر ورودی را از فضای رنگی BGR به RGB تغییر می‌دهد و سپس تصویر دوم را با استفاده از تابع resize تغییر اندازه می‌دهد تا اندازه آن با تصویر اول یکسان شود. سپس تصویری جدید با ابعاد مشابه تصویر اول ایجاد شده و همه پیکسل‌های آن به مقدار صفر تنظیم می‌شوند. در نهایت، کانال‌های قرمز و سبز تصویر اول و دوم به ترتیب در تصویر جدید کپی شده و کانال آبی تصویر دوم نیز در تصویر جدید کپی می‌شود. سپس تصویر نهایی با تفاوت بین دو تصویر، تابع خروجی داده می‌شود.**

****

**ه) علاوه بر اینکه فضای رنگی gray اطلاعات کمتری در مورد هر پیکسل در اختیار ما قرار می‌دهد، چندین دلیل دیگر نیز وجود دارد که استفاده از فضای رنگی مناسب در پردازش تصویر بسیار مهم است:**

1. **تفاوت رنگ‌ها: در فضای رنگی gray، رنگ‌ها تنها با سطح خاکستری پیکسل‌ها مشخص می‌شوند و اگر بخواهیم تفاوت رنگ‌ها را تفسیر کنیم، نیازمند محاسبات پیچیده‌تری هستیم. از طرفی در فضای رنگی RGB، هر رنگ با سه مقدار قرمز، سبز و آبی به صورت مجزا مشخص می‌شود و تفاوت رنگ‌ها بسیار واضح است.**
2. **تحلیل روشنایی: در برخی از کاربردهای پردازش تصویر مثل تشخیص اجسام در تصاویر، تحلیل روشنایی بسیار مهم است. در فضای رنگی gray، این تحلیل بسیار محدود است و ممکن است اطلاعات مهمی را از دست بدهیم. در فضای رنگی HSV، می‌توانیم روشنایی را با سه مقدار سطوح روشنایی، اشباع و رنگ تفکیک کنیم و برای برخی از کاربردها مانند تشخیص رنگ‌های خاص، این فضای رنگی بسیار مناسب است.**
3. **تفسیر رنگ‌ها: در برخی از کاربردهای پردازش تصویر، تفسیر رنگ‌ها بسیار مهم است. در فضای رنگی gray، تفسیر رنگ‌ها ناممکن است ولی در فضای رنگی CMYK، می‌توانیم رنگ‌های چاپی را تفسیر کنیم و برای برخی از کاربردها مانند چاپ و تولید محصولات چاپی، این فضای رنگی بسیار مهم است.**

**برای مثال به دو تصویر زیر نگاه کنید. در حالت gray، رنگ های آبی و سبز و قرمز تقریبا یک رنگ می شوند و تشخیص آن ها از هم دشوار است و ممکن است در برخی از تصاویر این رنگ ها با یکدیگر ادغام شوند و مرز آن ها قابل تشخیص نباشند.**

****

* **سوال 2**

**این کد برای ادغام چند تصویر با یکدیگر به کار می‌رود. ابتدا یک شی از کلاس Stitcher در OpenCV ایجاد می‌شود و سپس این شی با استفاده از تصاویر ورودی، تصویر جدیدی را با استفاده از روش ادغام تصاویر ایجاد می‌کند. اگر مقدار نسخه OpenCV برابر با 3 باشد، تابع createStitcher() استفاده می‌شود و در غیر این صورت تابع Stitcher\_create() به کار می‌رود. پس از اجرای این کد، تصویر جدید ادغام شده در متغیر stitched\_image ذخیره می‌شود و وضعیت عملیات ادغام تصاویر نیز در متغیر status بازگردانده می‌شود.**

****

* **سوال 3**

**الف) این کد یک تابع با نام put\_mask است که عکس چهره و عکس ماسک را دریافت می‌کند و ماسک را روی چهره اعمال می‌کند و عکس حاصل را برمی‌گرداند.**

**اولین قدم این است که با استفاده از کتابخانه dlib و فایل shape\_predictor\_68\_face\_landmarks.dat چهره در تصویر تشخیص داده می‌شود. سپس با استفاده از نقاط landmark چهره، نقاط متناظر با بینی، پوزخند چپ، پوزخند راست و دُهن را انتخاب می‌کنیم.**

**سپس بردارهایی به نام face\_points و mask\_points تعریف می‌شوند که نقاط متناظر با چهره در تصویر و نقاط متناظر با ماسک را دربرمی‌گیرند.**

**سپس با استفاده از تابع getPerspectiveTransform، ماتریس تبدیلی برای تبدیل نقاط متناظر با ماسک به نقاط متناظر با چهره به دست می‌آید.**

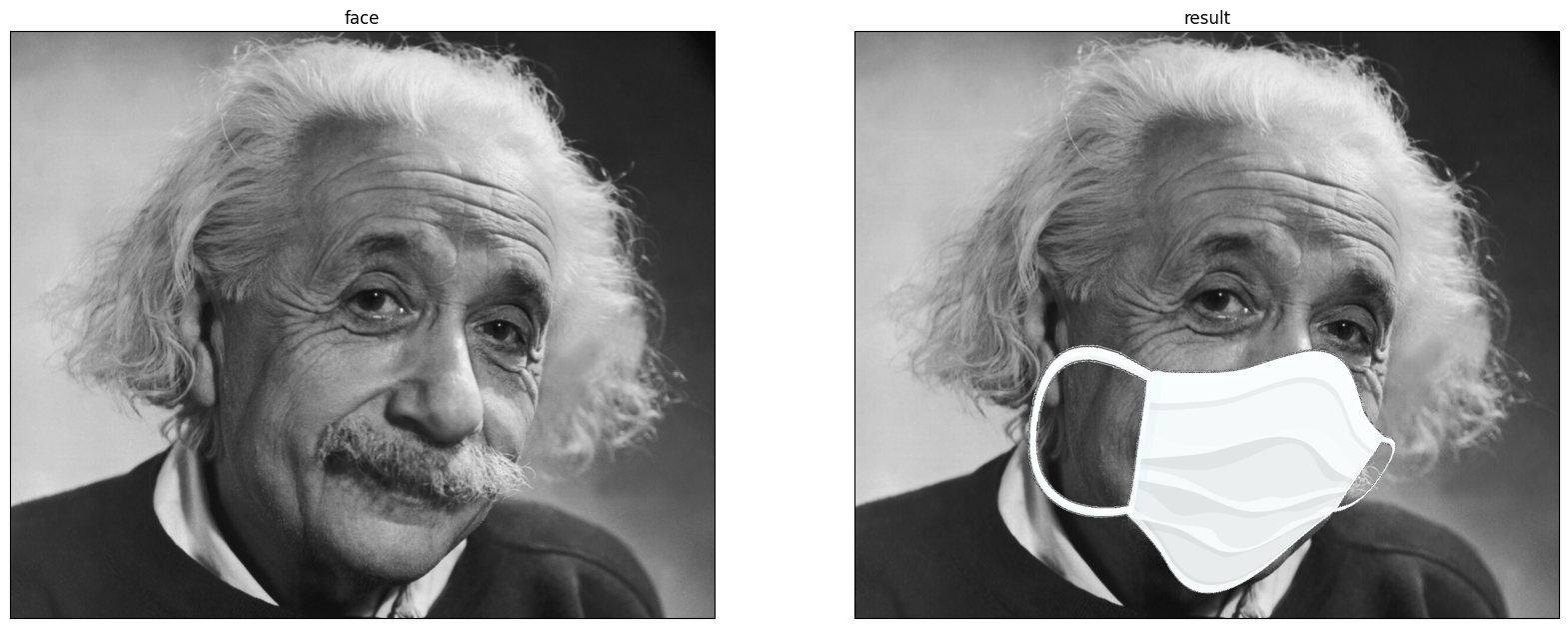
**سپس با استفاده از تابع warpPerspective، ماسک به صورت projective بر روی چهره اعمال می‌شود و تصویر جدیدی به نام mask\_warped به دست می‌آید.**

**سپس با استفاده از تابع cvtColor، تصویر mask\_warped به یک تصویر خاکستری تبدیل می‌شود. سپس با استفاده از تابع threshold، تصویر خاکستری به یک تصویر دودویی تبدیل می‌شود و تصویر نهایی ماسک به دست می‌آید.**

**در مرحله بعد، تصویر نهایی چهره به دست می‌آید. در این بخش از کد، ابتدا با استفاده از تابع cv2.bitwise\_not، ماسک را به‌صورت عکسِ تهیه می‌کنیم. سپس با استفاده از تابع cv2.bitwise\_and، عکس چهره (face\_img) را با نقاطی که در ماسک خالی نیستند (mask\_rgb) یا به عبارتی بخشی از ماسک که سفید است، عمل AND می‌کنیم. این کار باعث می‌شود که بخش‌های چهره که مشابه بخش‌های ماسک خالی نیستند، قابل دید نباشند.**

**سپس با تابع cv2.bitwise\_and، ماسک ماسک وارپ شده (mask\_warped) را با ماسک خالی نیست (mask\_rgb) عمل AND می‌کنیم. این کار باعث می‌شود که بخش‌های ماسک که سفید هستند، بر روی بخش‌های مشابه در عکس چهره نمایش داده شوند.**

**در نهایت با استفاده از تابع cv2.add، عکس چهره که در آن بخش‌های ماسک با بخش‌های مشابه در عکس چهره جایگزین شده‌اند (result) را با ماسکِ وارپ شده (mask\_warped\_masked) جمع می‌کنیم و نتیجه را برمی‌گردانیم.**

****

**ب) تشخیص landmark یا نقاط قابل توجه روی چهره، یکی از مهمترین و پرکاربردترین مسائل در بینایی کامپیوتر و تشخیص چهره است. Landmark ها به عنوان نقاط مشخصی که بر روی چهره وجود دارند، مانند چشم، بینی، لب، گوش و... تعریف می شوند. تشخیص این landmark ها، به صورت خودکار، به دلیل دقت بالا و کاربردهای گسترده، از اهمیت بسیاری برخوردار است. برای تشخیص landmark های چهره، می توان از چند روش استفاده کرد که در ادامه به توضیح یکی از این روش ها می پردازم.**

**یکی از روش های تشخیص landmark چهره، استفاده از شبکه های عصبی کانولوشنی (CNN) است. این روش در چند مرحله انجام می شود:**

1. **تشخیص چهره: ابتدا باید چهره در تصویر شناسایی شود. برای این کار می توان از الگوریتم های تشخیص چهره مانند Haar Cascade و یا شبکه های عصبی مانند SSD استفاده کرد.**
2. **پیش پردازش: در این مرحله، تصویر چهره به صورت استاندارد سازی شده و اندازه آن کوچک شده تا سرعت پردازش افزایش یابد.**
3. **استخراج ویژگی: در این مرحله، شبکه عصبی کانولوشنی بر روی تصویر استفاده می شود تا ویژگی های مختلفی از تصویر استخراج شوند. به عبارت دیگر، شبکه عصبی با عبور تصویر از آن، به صورت خودکار، اجزای مختلف چهره را شناسایی می کند.**
4. **در این مرحله، با استفاده از اطلاعات به دست آمده در مرحله قبلی، به طور دقیق landmark های مختلف روی چهره تشخیص داده می شوند. این کار با استفاده از یک شبکه عصبی کانولوشنی و یا یک شبکه عصبی مولد (Generative Adversarial Network) انجام می شود. در نهایت، مختصات landmark های تشخیص داده شده بر روی تصویر چهره قرار داده می شوند تا بتوان از آنها برای کاربردهای مختلفی مانند تشخیص احساسات صورت، تشخیص جنسیت، تشخیص سن و... استفاده کرد.**