Pop-effect :

(Pop Effect) به فرآیندی گفته می‌شود که طی آن یک بخش خاص از تصویر (مثلاً یک چهره، یک شیء یا سوژه‌ی اصلی) **با استفاده از تکنیک‌هایی مانند افزایش کنتراست، اشباع رنگ، یا تار کردن پس‌زمینه** برجسته شود، تا بیننده بیشتر روی آن تمرکز کند.

روش‌های متداول ایجاد pop-effect درOpenCV

1. **تار کردن پس‌زمینه(Background Blur)**

با استفاده از تشخیص سوژه مثلاً با ماسک یاsegment ، می‌توان پس‌زمینه را تار کرد تا سوژه جلوه‌ی بیشتری داشته باشد.

2. **تقویت رنگ یا اشباع (Color Pop)**

تصویر به خاکستری تبدیل می‌شود و فقط بخش خاصی رنگی باقی می‌ماند. این باعث می‌شود که آن بخش "بپرد" یا برجسته شود.

3. **افزایش کنتراست و روشنایی موضعی:**

تنظیم کنتراست یا روشنایی فقط در ناحیه‌ی موردنظر، مثلاً با استفاده از ماسک.

4. **استفاده از فیلترهای هنری یا حاشیه‌ی واضح:**

به‌کارگیری فیلترهایی که مرزهای سوژه را مشخص‌تر می‌کنند یا جلوه‌های گرافیکی خاصی اضافه می‌کنند.

یکی از مشهورترین روش های pop effevt در تصاویر color pop نام دارد. در ادامه می خواهیم با کد های زیر این روش را پیاده کنیم.

ابتدا بیایید بررسی کنیم که چگونه می توان یک یا چند پیکسل از عکس را انتخاب نمود. احتمالا کد زیر یک کد بهینه برای این کار است.

```
# mouse callback function
def draw_circle(event,x,y,flags,param):
    global img
    if event == cv2.EVENT_LBUTTONDOWN:
        bgr = img[x:x+1,y:y+1,:]
        print("BGR: ", bgr)
        hsv = cv2.cvtColor(bgr, cv2.COLOR_BGR2HSV)
        print("HSV: ", hsv)
        cv2.circle(img,(x,y),5,(0,255,0),+2)

# Create a black image, a window and bind the function to window
img = cv2.imread("images/PopEffects.jpeg")

cv2.namedWindow('image')
cv2.setMouseCallback('image',draw_circle)

while(1):
    cv2.imshow('image',img)
    if cv2.waitKey(1) & 0xFF == 27:
        break
cv2.destroyAllWindows()
```

در کد فوق ابتدا یک تابع برای واکنش به کلیک ماوس می سازیم. این تابع زمانی فراخوانی می‌شود که کاربر روی پنجره‌ی تصویر کلیک کند. در این تابع ابتدا یک متغیر سراسری تعریف می کنیم سپس با استفاده از `[x:x+1,y:y+1,:]` و عملیات ایندکس گذاری به پیکسل مورد نظر می رسیم. چون در numpy یک بازه شامل شروع آن بازه می شود اما پایان را در نظر نمی گیرد بنابرای ما هر پیکسل را با عدد 1 جمع می کنیم تا به خود پیکسل مورد نظر برسیم. در نهایت مقادیر bgr و HSV را چاپ می کنیم. در نهایت یک رویداد `callback` را فراخوانی کرده و تنظیمات مورد نظر را به آن پاس میدهیم. در نهایت در یک حلقه `while` عکس را دائم به کاربر نمایش می دهیم تازمانی که پیکسل مورد نظر را انتخاب نماید.

اکنون که توانستیم مقادیر مورد نظر را از عکس استخراج کنیم وقت آن است تا یک تابع جدید برای رویداد موس نوشته و از آن در کد اصلی استفاده کنیم:

```
def pop_efect(image, lower, upper):
    img = image.copy()
    #convert the BGR image to HSV colour space
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    #obtain the grayscale image of the original image
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    #create a mask using the bounds set
    mask = cv2.inRange(hsv, lower, upper)
    #create an inverse of the mask
```

```
    mask_inv = cv2.bitwise_not(mask)
    #Take only region of image from the mask
    res = cv2.bitwise_and(img, img, mask=mask)
    #Filter the regions containing colors other than masked region from the grayscale image
    background = cv2.bitwise_and(gray, gray, mask = mask_inv)
    #convert the one channelled grayscale background to a three channelled image
    background = cv2.merge([background]*3)
    #add the color region to gray image
    return cv2.add(res, background)
```

این تابع اصلی برای رویداد موس است. در این تابع میخواهیم بخش از تصویر که رنگی است را به شکل رنگی نگه داریم و و بقیه تصویر به صورت خاکستری باقی بماند.

ابتدا تابعی با 3 پارامتر تعریف میکنیم:

`image` تصویر ورودی به فرمت BGR رنگی معمولیOpenCV

`lower, upper` محدوده رنگ در فضای HSV برای رنگ مورد نظر که می‌خواهیم "رنگی باقی بماند".

و سپس یک کپی از روی آن تهیه می کنیم.

در مرحله بعد یکبار تصویر را به HSV و یکبار آن را به gray تبدیل می کنیم.

```
mask = cv2.inRange(hsv, lower, upper)
```

- یک **ماسک باینری (سیاه و سفید)** می‌سازیم که:
  - در نواحی با رنگ در محدوده `[lower, upper]` مقدار 255 (سفید) دارد.
  - بقیه نواحی مقدار 0 (سیاه) دارند.
- این ماسک همان بخش‌هایی است که باید **رنگی بمانند**.

```
mask_inv = cv2.bitwise_not(mask)
```

این کد ماسک را **معکوس** می‌کند:

- حالا نواحی خارج از بازه رنگی سفید هستند.

- از این ماسک برای بخش‌هایی که باید **خاکستری شوند** استفاده می‌کنیم.

```
res = cv2.bitwise_and(img, img, mask=mask)
```

- فقط بخش‌هایی از تصویر را نگه می‌دارد که در ماسک (یعنی رنگ مورد نظر) مشخص شده‌اند.
- یعنی فقط **رنگ دلخواه کاربر** باقی می‌ماند.

```
background = cv2.bitwise_and(gray, gray, mask = mask_inv)
```

- بخش‌هایی از تصویر خاکستری را که **در ماسک معکوس هستند** نگه می‌دارد.
- یعنی تمام قسمت‌هایی که **نباید رنگی باشند**.

```
background = cv2.merge([background]*3)
```

- چون `background` فقط یک کانال (خاکستری) دارد و باید با تصویر رنگی ترکیب شود،
- آن را به ۳ کانال تبدیل می‌کند R ، G، Bکه همه‌شان یکسان‌اند، اما تصویر ۳ کاناله می‌شود.

```
return cv2.add(res, background)
```

- نهایتاً بخش رنگی (`res`) را با پس‌زمینه‌ی خاکستری (`background`) ترکیب می‌کند.

- خروجی، تصویری است که در آن:

- فقط یک رنگ خاص باقی مانده
- بقیه‌ی تصویر سیاه و سفید است.

بعد از نوشت این تابع نوبت به نوشتن کد اصلی است:

```
import matplotlib.pyplot as plt
import numpy as np
import cv2

# mouse callback function
def apply_pop_efect(event,x,y,flags,param, crange=10):
    global image, new_image
    if event == cv2.EVENT_LBUTTONDOWN:
        hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        h, s, v = hsv[x,y,:]
        lower_h = max(0, h-crange)
        upper_h = min(179, h+crange)
        #set the bounds for the red hue
        lower = np.array([lower_h,50,50])
        upper = np.array([upper_h,255,255])
        new_image = pop_efect(image, lower, upper)
        cv2.imshow("result", new_image)
```

```
# Create a black image, a window and bind the function to window
image = cv2.imread("images/PopEffects.jpeg")
new_image = image.copy()

cv2.namedWindow('image')
cv2.setMouseCallback('image',apply_pop_efect)

while True:
    cv2.imshow('image',image)
    if cv2.waitKey(1) & 0xFF == 27:
        break
cv2.destroyAllWindows()

plt.figure(figsize=[10,15])
plt.subplot(121);plt.imshow(image[...,::-1]);plt.title("Original");
plt.subplot(122);plt.imshow(new_image[...,::-1]);plt.title("Output");
```

این کد یک نسخه تعاملی از **"Color Pop Effect"** هست که با کلیک روی تصویر، رنگی که کلیک کرده‌ای را **خودکار انتخاب می‌کند** و فقط آن رنگ را رنگی نگه می‌دارد، بقیه را سیاه و سفید می‌کند.

در این کد ابتدا یک تابع برای اعمال pop effect ایجاد می کنیم:

```
def apply_pop_efect(event,x,y,flags,param, crange=10)
```

به آخرین ورودی این متد دقت کنید. پارامتر `crange` مقدار ±10 حول رنگ انتخاب‌شده را در نظر می‌گیرد (برای محدوده رنگی). این پارامتر مشخص می‌کند که وقتی روی یک پیکسل از تصویر کلیک می‌کنی، **چه محدوده‌ای از رنگ (Hue) "H"** در فضای رنگی HSV به‌عنوان "رنگ انتخابی" در نظر گرفته شود.

```
h, s, v = hsv[x,y,:]
```

با استفاده از این خط کد به برنامه میگوییم تمام کانال ها در سطر و ستونی که بر روی آن کلیک شده را بخوان .

```
lower_h = max(0, h-crange)
upper_h = min(179, h+crange)
```

این دو خط کد:

رنگ hue در OpenCV بین 0 تا 179 است.

این خط یک محدوده ±10 از رنگی که کلیک کردی می‌سازد.

```
lower = np.array([lower_h,50,50])
upper = np.array([upper_h,255,255])
new_image = pop_efect(image, lower, upper)
cv2.imshow("result", new_image)
```

و در نهایت فقط رنگی که hue آن در محدوده تعیین شده است در نظر گرفته می شود.