آموزش کـــــاربردی پردازش تصویر با پایتون

امیرحسین ملک زاده

سخن نویسنده

من فارق التحصیل رشته برق – کنترل دانشگاه سراسری کاشان هستم. بخاطر فعالیتم در زمینه رباتیک، پردازش تصویر و هوش مصنوعی برای من جذاب بودند و همیشه علاقه‌مند بودم در این زمینه‌ها مطالعه داشته باشم تا در نهایت در سال ۱۳۹۵ این اتفاق برای من رقم خورد. کتابی که شما در حال خواندن آن هستید در حقیت یادداشت های من در زمان یادگیری پردازش تصویر بود که پس از کمی اصلاحات تصمیم گرفتم تا آن را منتشر کنم تا افراد دیگری بتوانند از تجربه‌های هرچند اندک من استفاده نمایند. این کتاب براساس اسناد ارائه شده توسط انجمن opencv و چندین سایت دیگر جمع آوری شده. باتوجه به تغییرات در نسخه های مختلف کتابخانه opencv ممکن است در برخی توابع تغییراتی بووجود آمده باشد که با جست و جو در اینترنت میتوانید این مشکل را حل نمایید. این کتاب در سال ۱۳۹۵ – ۱۳۹۶ یادداشت شده است. از آنجا که این کتاب نسخه نخستی است که منتشر می‌شود ممکن است دارای برخی اشکالات نگارشی باشد که پیشاپیش از شما عذرخواهی میکنم. امیدوارم خواندن این کتاب برای شما مفید باشد و دریچه‌ای باشد برای ورود شما به دنیای عظیم پردازش تصویر. در آخر تشکر میکنم از همه دوستانی که مرا در انتشار این کتاب تشویق نمودند.

فهرست

پردازش تصویر چیست؟

پردازش تصویر یعنی انجام عملیاتی بر روی تصویر برای تولید یک تصویر مطلوب جدید یا استخراج اطلاعات لازم. هنگامی که شما با گوشی هوشمند خود عکس برداری می‌کنید. گوشی شما با استفاده از پردازش تصویر، عکس شما را از نظر رنگ اصلاح می‌کند و نویزهای آن‌را حذف می‌کند تا شما تصویر مطلوب تری بدست اورید. هنگامی که شما در اینستاگرام خود رنگ‌بندی یک تصویر را تغییر می‌دهد، به عکس خود استیکر و متن اضافه می‌کند درحقیقت بر روی تصویر پردازش انجام می‌دهید تا تصویر جدید مطلوب خود را بدست آورید. از سوی دیگر پردازش تصویر می‌تواند اطلاعات مورد نیاز ما را از تصویر استخراج نماید. برای مثال هنگام عکس برداری با گوشی هوشمند خود متوجه شده اید که گوشی شما در تصویر برای فوکوس بهتر، چهره های موجود در تصویر را شناسایی می‌کند یا هنگامی که در جاده با سرعت غیر مجاز رانندگی می‌کنید، دوربین‌های نظارتی پلاک شما را از تصویر استخراج می‌نمایند. پردازش تصویر در زمینه‌های مختلفی مانند، نظامی، کشاورزی، خدماتی و... کاربرد دارد. و با پیشرفت روز به روز تکنولوژی، کامپیوترهایی که قدرت دیدن دارند، بخشی از زندگی ما هستند و خواهند بود.

کتابخانه OpenCV

این کتابخانه ابتدا در سال ۱۹۹۹ توسط اینتل ساخته شد.OpenCV یک کتابخانه متن باز بوده و هر روز در حال گسترش است. این کتابخانه از طیف گسترده‌ای از زبان‌های برنامه‌نویسی نظیر java , c++, python و غیره پشتیبانی می‌کند و در سیستم عامل‌های مختلف نظیر ویندوز، اندروید، لینوکس، ios پیاده سازی می‌شود. شاید بتوان مهم ترین ویژگی این کتابخانه پردازش تصویر را در متن باز و رایگان آن دانست که این کتابخانه را تبدیل به پر طرفدارترین کتابخانه پردازش تصویر کرده است.

openCV-Pthton : یک API برای پایتون است که ترکیب قوی از زبان پایتون و API زبان c++ می‌باشد. در این کتابخانه از ماژول numpy که یک کتابخانه قوی برای عملیات عددی استفاده شده است. تمام ساختارهای آرایه‌ای OpenCV به آرایه‌های numpy تبدیل می‌شوند

آموزش نصب opencv

برای نصب ابتدا لازم است که پایتون ۳ را بر بروی سیستم خود نصب نمایید. برای نصب OpenCV کافی لست در پنجره cmd دستور زیر را وارد نمایید، درصورتی که اتصال شما به اینترنت برقرار باشد، این پکیج به صورت خودکار بر روی سیستم شما دانلود و نصب خواهد شد.

```
pip install opencv-python
```

مبانی پردازش تصویر

تصویر چیست

تصویر از تعدادی پیکسل تشکیل شده است. پیکسل کوچکترین واحد یک تصویر است که از یک رنگ تشکیل شده است در کامپیوتر و دیجیتال تصاویر در حقیقت ارایه ای از اعداد هستند. که ابعاد این آرایه برابر با ابعاد تصویر ما می‌باشد. برای مثال برای نمایش یک تصویر سیاه و سفید ۴۸۰ * ۶۴۰ از یک آرایه با ۶۴۰ ستون و ۴۸۰ سطر استفاده می‌شود. ارایه تصویر علاوه بر داشتن سطر و ستون دارای پارامتر دیگری با نام عمق می‌باشد که این پارامتر بر اساس فضای رنگی تصویر متفاوت است

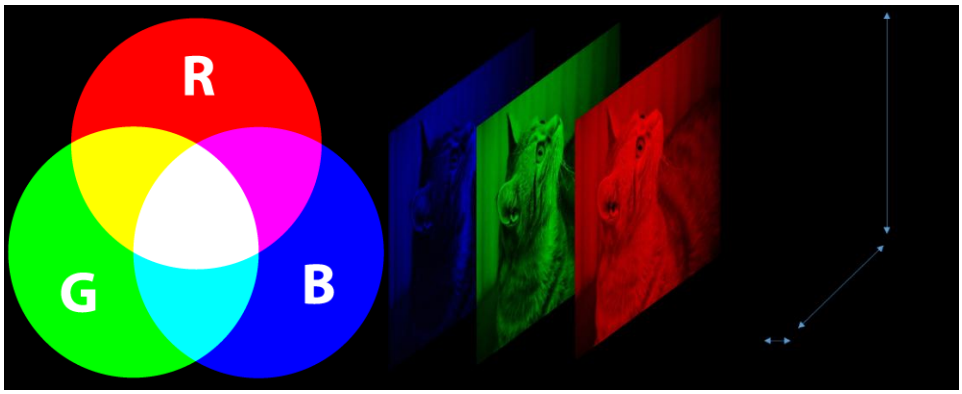
0	1	1	2	2	2	1	1	0
1	2	4	5	5	5	4	2	1
1	4	5	3	0	3	5	4	1
2	5	3	-12	-24	-12	3	5	2
2	5	0	-24	-40	-24	0	5	2
2	5	3	-12	-24	-12	3	5	2
1	4	5	3	0	3	5	4	1
1	2	4	5	5	5	4	2	1
0	1	1	2	2	2	1	1	0

فضاهای رنگی

فضای رنگی یک تصویر قرار دادی است که نمایش و ذخیره رنگ‌ها در کامپیوتر را مشخص می‌کند که دارای انواع مختلفی می‌باشد. در ادامه تعدادی از پرکاربردترین فضاهای رنگی را توضیح خواهیم داد.

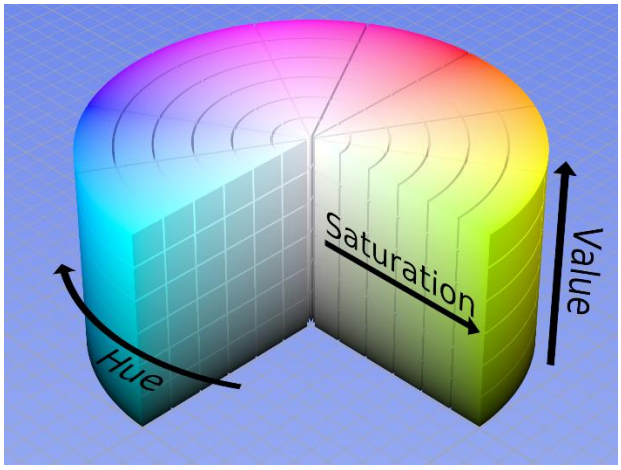
فضای رنگی خاکستری : همانگونه که از نام این فضای رنگی مشخص است، این فضا برای ذخیره و نمایش تصاویر سیاه و سفید کاربرد دارد. در این فضا هر پیکسل با یک عدد مشخص می‌شود، که در سیستم ۸ بیتیُ این عدد مقداری بین ۰ تا ۲۵۵ می‌باشد که ۰ به معنای سیاه مطلق و ۲۵۵ به معنای سفید است. ارایه متناظر با یک تصویر سیاه و سفید، یک ارایه دو بعدی می‌باشد که ابعاد آن برابر است با : عرض تصویر * طول تصویر

فضای رنگی RGB : این فضای رنگ برای نمایش تصاویر رنگی کاربرد دارد و نزدیک ترین فضای رنگی به درک انسان می‌باشد. در این فضا کلیه رنگ ها، از ترکیب سه رنگ اصلی قرمز، سبز و آبی تشکیل می‌شوند. در این فضای رنگی هر پیکسل با سه عدد مشخص می‌شود که این اعداد متناسب با مقدار سه رنگ اصلی می‌باشند و در سیستم ۸ بیتی، مقداری بین ۰ تا ۲۵۵ دارند که ۰ به معنای نبودن آن رنگ و ۲۵۵ به معنای کامل بودن آن است. برای مثال پیکسل [۰ , ۰ ,۲۵۵] نشان دهنده قرمز خالص و پیکسل [۰,۱۰۰,۰] نشان دهنده آبی کم رنگ می‌باشد. از این رو در این فضای رنگی، آرایه متناظر با تصویر سه بعدی بوده و دارای ابعاد زیر است ۳ * W * H می‌باشد. در حقیقت تصویر در این فضای رنگی، از قرار گرفتن سه تصویر با رنگ‌های اصلی بر روی هم ایجاد می‌شود



فضای رنگی CMYK : این فضای رنگ در چاپ تصاویر رنگی کاربرد دارد و از چهار رنگ سیاه، زرد، آبی کمرنگ و صورتی تشکیل می‌شود. این رنگ‌ها در حقیقت همان فضاهای اشتراکی در فضای رنگی RGB هستند.

فضای رنگی HSV : این فضای رنگی از سه پارامتر رنگ (hue) ، میزان اشباع رنگ (saturation) و میزان نور (value) تشکیل شده است. این فضای رنگ در پردازش تصاویر رنگی بسیار کاربرد دارد که علت آن در فصل‌های بعدی توضیح داده می‌شود. در این فضای رنگ هر پیکسل با سه مقدار مشخص می‌شود که متناظر با سه پارامتر گفته شده می‌باشند. با استفاده از پارامتر hue ما رنگ نورد نظر خود را می‌توانیم از حلقه رنگی برگزنیم. پارامتر saturation میزان اشباع رنگ را مشخص میکند. شاید اگر تاکنون دست به نقاشی با رنگ هایی مثل ابرنگ یا اسپری و... شده باید متوجه اشباع رنگ شدید. فرض کنید می‌خواید یک اسمان را در نقاشی خود رنگ کنید. ابتدا قلموی خود را به رنگ ابی اغشته می‌کنید و شروع به رنگ زدن اسمان می‌کنید اما ممکن است در بار نخست اسمان ابی شما کم رنگ باشد یا به اصطلاح میزان اشباع آن کم باشد. پس بار دگیر قلموی خود را به رنگ آبی اغشته می‌کنید و اسمان را مجدد رنگ امیزی می‌کنید. اینبار بدون آنکه رنگ خود را تغییر دهید اسمان شما پررنگ تر می‌شود. در حقیقت شما رنگ آبی خود را تغییر نمی‌دهد بلکه میزان اشباع آن را تغییر می‌دهید تا رنگی با اشباع بیشتر، داشته باشید. اما در آخر پارامتر value میزان تاثیر نور بر رنگ شما را مشخص می‌کند. فرض کنید یک جسم قرمز در دست دارید اگر شما در مکانی با نور مناسب قرار بگیرید شما جسم خود را قرمز روشنی می‌بینید. اما اگر حال چراغ هارا خاموش کنید شما جسم خود را تیره تر می‌بینید. اما در واقع رنگ جسم شما تغییر نکرده است تنها میزان نور آن تغییر کرده است و چه بسا در تاریکی مطلق شما جسم خود را کاملا سیاه ببینید. در فضای رنگی RGB با تغییر نور، هر سه پارامتر RGB ممکن است تغییر کنند و اصالت رنگ جسم شما از بین برود. اما ر فضای رنگی HSV تغییر نور تنها بر روی پارامتر V موثر است و Hue و saturation رنگ شما را تغییر نمی‌دهد. پس به راحتی می‌توان در فضای رنگی HSV تاثیر نور را تحت کنترل داشته باشید. از این رو همانند فضای رنگ RGB، تصاویر در این فضای رنگ با یک آرایه سه بعدی نمایشه داده میشوند



سخنی با خوانندگان

در این کتاب هدف اصلی پیاده سازی الگوریتم‌های پردازش تصویر با کتابخانه opencv و زبان python می‌باشد. ما در این کتاب به بررسی تئوری‌های پردازش تصویر، جزئیات الگوریتم‌ها نخواهیم پرداخت و مباحث تا حد نیاز برای بکارگیری کاربردی پردازش تصویر ارائه شده است. شما می‌توانید در صورت علاقه به کتاب مبانی پردازش تصویر گنزالس مراجعه نمایید. این کتاب به زبان فارسی نیز ترجمه گردیده است. توصیه اینجانب آن است که در ابتدا الگوریتم‌های پردازش تصویر را در عمل فراگرفته و در نهایت برای درک بهتر، جزئیات الگوریتم‌های ارائه شده را مطالعه نمایید.

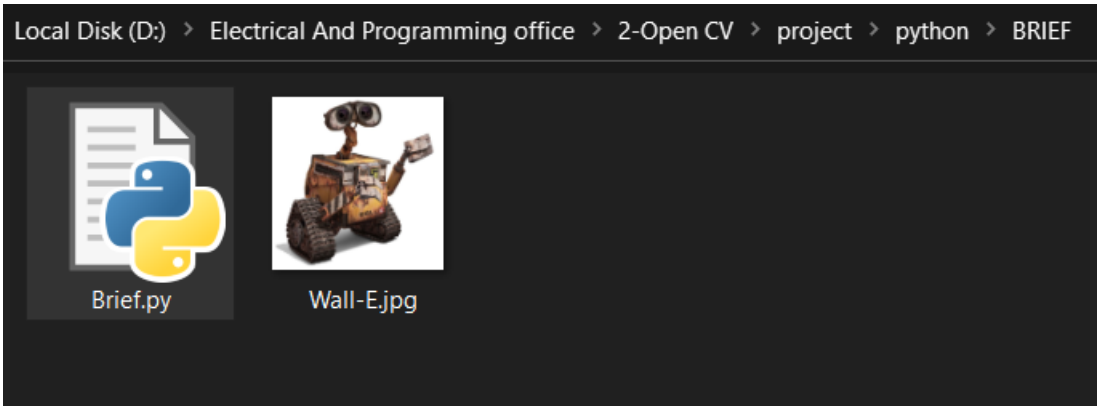
شروع کار با کتابخانه OpenCV

خواندن و نوشتن تصویر

پیش از هر ار برای شروع پردازش تصویر ما لازم داریم تا یک تصویر را از روی حافظه بخوانیم یا تصاویر دوربینمان را دریافت نماییم. برای خواندن یک تصویر از تابع imread() استفاده می‌کنیم. پارامترهای این تابع به صورت زیر می‌باشند.

Imread(image Address , flags) -> dst

Image Address : آدرس و نام تصویر همراه با پسوندش مورد نظر در این بخش در بین " " وارد می‌گردد. اگر سورس کد و تصویر ما در یک فایل قرار داشته باشند وارد کردن نام تصویر همراه با پسوندش کفایت می‌کند. پس به شما توصیه می‌کنیم که عکس مورد نظر خود را در یک پوشه کنار کد خود قرار دهید تا نیازی به وارد کردن ادرس نداشته باشید و در صورت جابه جایی فایل های خود برنامه شما بدون مشسکل اجرا گردد.



Brief.py - D:\Electrical And Programming office\2-Open CV\project\python\BRIEF\Bri...

File Edit Format Run Options Window Help

```
import cv2

img=cv2.imread ("Wall-E.jpg" , 0)
```

تذکر : در آدرس تصویر به‌جای وارد کردن / باید // را وارد نمایید

Flags : این پارامتر میتواند یکی از سه پرچم زیر باشد. همچنین می‌توان به‌جای این پرچم ها مقادیر عددی متناظرشان که در داخل پرانتز نوشته شده اند را به کار ببریم

۱- IMREAD_COLOR : تصویر را به عنوان یک تصویر رنگی لود می‌کند (می‌توان از مقدار عددی ۱ نیز به جای پرچم فوق استفاده نمود)
۲- IMREAD_GRAYSCALE : تصویر را به عنوان یک تصویر خاکستری لود میکند (می‌توان از مقدار عددی ۰ نیز به جای پرچم فوق استفاده نمود)
۳- (
۴- IMREAD_UNCHANGED : تصویر را همراه کانال آلفا لود می‌کند. (می‌توان از مقدار عددی -۱ نیز به جای پرچم فوق استفاده نمود)
۵- (

ودرصورتی که مقدار این flag را مشخص نکنید، قدار پیشفرض آن یعنی ۱ در نظر گرفته می‌شود.

کانال آلفا چیست؟

شاید پیش از این با تصاویری مانند آیکون ها، مواجه شده باشید که بخش هایی از این تصاویر حذف شده باشند. اگر با فتوشاپ کار می‌کنید قطعا خودتان این کار را انجام داده اید. به چنین تصاویری کانال آلفا می‌گویند.

مثال :

```
img0 = cv2.imread ("cat.JPG")
img1 = cv2.imread ("C:\\Users\\Amir_PC\\Desktop\\image.jpg" )
img2 = cv2.imread ("face.jpg" , 0)
cv2.imshow(“image1” , mat)
```

حالا که توانستیم یک تصویر را اصطلاحا لود نماییم، قدمی بعدی نمایش تصویر است تا از اجرای درست مرحله قبل مطمئن شویم و همچنین تغییراتی را که در ادامه بر روی تصویر ایجاد می‌کنیم را بتوانیم به صورت بصری مشاهده نماییم. برای نمایش یک تصویر از تابع imShow() استفاده می‌کنیم که پارامتر های آن به صورت زیر می‌باشد.

Imshow(" Windows name " , image)

Windows name : نام پنجره‌ای که تصویر در آن نمایش داده می‌شود. این نام می‌تواند کاملا دلخواه باشد.

Image : نام تصویری که میخواهیم آن را نمایش دهیم.

مثال:

```
>>> mat=cv2.imread(“cat.JPG”)
>>> cv2.imshow(“image1” , mat)
```

در گام اخر می‌خواهیم نحوه ذخیره ی یک تصویر در opencv را توضیح دهیم. گاهی لازم است تا ما پس از انجام پردازش بر روی تصویر، تصویر نهایی را ذخیر نماییم. برای ذخیره یک تصویر از تابع imwrite() استفاده می‌کنیم.

Imwrite (" image Name & Address" , imag) -> retval (bool)

image Name & Address : آدرس و نامی که میخواهیم تصویر را با آن ذخیره کنیم. اگر آدزسی برای تصویر وارد نکنیم و تنها نام آن را بنویسیم، تصویر در فایل سورس کد ذخیره می‌شود.

Imag : نام تصویری که میخواهیم ذخیره کنیم.

مثال :

```
Cv2.imwrite ("C:\\Users\\Amir_PC\\Desktop\\my.jpg", img )
```

در کد بالا تصویر img در ادرس فوق و با نام my و فرمت jpg ذخیر می‌شود.

برای یک درک بهتر از توابعی که تاکنون آموزش دیده ایم. یک مثال جامع بیان می‌کنیم. در مثال زیر ما یک تصویر را به طور خاکستری لود می‌کنیم و آن را نمایش می‌دهیم سپس اگر کلید s را فشار دهیم تصویر ذخیره می‌شود و اگر esc را فشار دهیم کلیه پنجره ها بسته می‌شوند

```
import cv2

img = cv2.imread('messi5.jpg',0)
cv2.imshow('image',img)

k = cv2.waitKey(0)

if k == 27:         # wait for ESC key to exit
    cv2.destroyAllWindows()
elif k == ord('s'):
    cv2.imwrite('messigray.png',img)
    cv2.destroyAllWindows()
```

در opencv لازم است تا پس از نمایش یک تصویر، تابع cv۲.waitKey() را فرخوانی کنیم. اگر ورودی این تابع مقدار صفر باشد، برنامه منتظر زدن کلید از سوی کاربر می‌شود و اگر عددی غیر از صفر باشد، تابع به میزان عدد وارد شده بر حسب میلی ثانیه منتظر فشرده شدن کلید از سوی کاربر می‌شود و اگر کاربر در این زمان، کلیدی را از صفحه کیبورد فشار ندهد، برنامه به کار خود ادامه می‌دهد. خروجی این تابع یک مقدار عددی است که نشان دهنده کد کلیدی است که کاربر فشار داده. اگر این مقدار -۱ باشد یعنی کاربردی کلیدی را فشار نداده است. برای مثال کد ۲۷ نشان دهنده کلید Esc کیبورد است که در مثال بالا استفاده شده یا خروجی تابع ord('s') مقدار کد کلید s از صفحه کلید را باز می‌گرداند. استفاده از تابع waikey به ویژه برای نمایش تصاویر در حلقه ها امری ضروری است. نمایش یک تصویر، امری زمان بر است و این تابع درحقیقت فرصتی برای اجرای آن فراهم می‌سازد. بنابراین اگر از این تابع استفاده ننماییم، دستور نمایش تصویر به CPU ارسال می‌شود و بیش از آنکه cpu فرصت نمایش تصویر را پیداکند، دستور نمایش تصویر بعدی ارسال میگردد و بدین ترتیب تصویر ما به درستی به نمایش در نمی‌آید

نمایش تصویر با matplotlib

کتابخانه matplot یک از کاربردی ترین کتابخانه های پایتون بوده و این کتابخانه برای ترسیم و نمایش انواع تصاویر، آرایه‌ها است و انواع نمودار ها می‌باشد. نمایش تصویر با matplot امکانات ویژه‌ای در اختیار ما قرار می‌دهد مانند ذخیره عکس، زوم کردن و...

برای نصب matplotlib دستور مقابل را در CMD وارد کنید : pip install matplotlib

این کتابخانه دارای ماژول های بسیاری است. اما ماژول مورد استفاده ما pyplot می‌باشد. از این رو ماژول pyplot را از این کتابخانه لود میکنیم. برای نمایش یک تصویر در pyplot از تابع ()imshow استفاده می‌کنیم. پارامتر نخست این تابع تصویر مورد نظر است. این تابع پارامتر های زیادی دارد که ما به طرح یک مثال بسنده می‌کنیم. پس از فراخوانی تابع فوق برای مشخص کردن عنوان های محور های X و Y از توابع ()xticks و yticks از ماژول ()pyplot استفاده می‌کنیم. ورودی این توابع رشته ای است که روی محور ها نشان داده می‌شود. در مثال زیر از آنجا که نمی‌خواهیم برچسبی بر روی محور های x,y نمایش داده شوند، مقدار وردی این توابع را خالی می‌گذاریم در نهایت با استفاده از تابع ()show از همین ماژول تصویر را نمایش می‌دهیم.

مثال

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

img = cv2.imread('messi5.jpg',0)
plt.imshow(img, cmap = 'gray', interpolation = 'bicubic')
plt.xticks([]), plt.yticks([])  # to hide tick values on X and Y axis
plt.show()
```

خواندن یک دنباله ویدویی

در کتابخانه opencv علاوه بر آنکه ما می‌توانیم بر روی یک تصویر پردازش کنیم، می‌توانیم بر روی یک دنباله ویدیویی نیز پردازش انجام دهیم. یک دنباله ویدویی از تصاویری به نام فریم در فواصل زمانی یکسان و کوتاهی تشکیل شده است. عملیات پردازش بر روی تصویر و ویدیو یکسان است. در دنباله ویدیویی تنها به جای پردازش بر روی یک تصویر، ما بر روی فریم های یک فیلم به صورت تصاویر جداگانه پردازش انجام می‌دهیم

برای خواندن یک دنباله ویدیویی در opencv ابتدا لازم است تا یک شی از کلاس VideoCapture بسازیم. ورودی متد سازنده این کلاس می‌تواند نام فایل ویدیویی مورد نظر و یا id یک دوربین برای دریافت تصاویر زنده از آن باشد. آیدی دوربین پیشفرض یک سخت افزار برای مثال وب کم لبتاب شما یا دوربین برد های رزبری پای برابر با مقدار ۰ است و آیدی دوربین دوم ۱ و با همین روند ادامه دارد. پس از تعریف آبجکت از کلاس VideoCapture ما می‌توانیم با متد های این کلاس کار با دنباله ویدیویی را آغاز نماییم. در زیر به معرفی متدهای این کلاس می‌پردازیم.

read() -> retVal (bool) , frame

این متد برای خواندن فریم های دنباله ویدیویی به کار می‌رود. این تابع پارامتر ورودی ندارد. خروجی دوم این تابع (frame) فریم جاری و خروجی اول (ret) یک مقدار بولین است و اگر خواندن فریم بدون مشکل انجام شود مقدار true و در غیر این صورت مقدار false را باز می‌گرداند. از این خروجی میتوان برای تشخیص پایان ویدیو بهره برد.

isOpened() -> retVal (bool)

این تابع مشخص می‌کند که آیا فایل ویدیویی یا دوربین لود شده است یا خیر. خروجی این تابع یک مقدار bool است. اگر true باشد یعنی یک فایل باز شده و اگر false باشد یعنی فایل ویدیویی باز نشده است.

get(probId) -> retVal / set(probId, value) -> -> retVal (bool)

```
>>>cap.get(3)
640
>>>cap.get(4)
480
>>>cap.set(3,320)
>>>cap.set(4,240)
```

تابع get() می‌تواند برخی از ویژگی های فریم ها را دریافت کرد. ورودی این تابع عددی بین ۰ تا ۱۸ است که هر مقدار عددی یک از خصوصیات فایل ویدویی را بر میگرداند. همچنین می‌توان توسط تابع set مقدار هر کدام از این خصوصیات را به مقدار مورد نظر تغییر داد. برای مثال ویژگی‌های ۳ و ۴ عرض و طول فریم ها را نشان میدهند. که می‌توان آنرا به مقدار مورد نظر نیز تغییر داد.

معرفی تابع release

گاهی نیاز است پس از دسترسی به سخت افزار دوربین و دریافت فریم، سخت افزار را رهاسازی نماییم تا اجازه دسترسی آن را به بخش‌های دیگر و نرم افزارهای دیگر بدهیم. برای مثال هنگامی که شما درحال خواندن از وب کم لبتاب خود هستید، امکان استفاده نرم افزار های دیگر از وب کم شما نیست و تا زمانی که شما سخت افزار دوربین خود را آزاد ننمایید این دسترسی برای بخش های دیگر بسته خواهد بود. تابع فوق وظیفه رهاسازی دوربین را دارد.

Open(Video Address OR Camera ID) -> retVal (bool)

این تابع برای باز کردن یک دنباله ویدیویی جدید مورد استفاده قرار می‌گیرد. ورودی این تابع آدرس فایل و یا آیدی دوربین مورد نظر می‌باشد.

مثال : در مثال زیر یک دنباله ویدیویی را از دوربین با آیدی ۰ (دوربین پیشفرض که در لب تاب همان وب کم شما است) میخوانیم و پس از تبدیل هر فریم آن به فضای رنگ خاکستری آن را نمایش می‌دهیم.

```
import numpy as np
import cv2
cap=cv2.VideoCapture (0)

while (True):

    ret,frame=cap.read()
    gray= cv2.cvtColor (frame , cv2.COLOR_RGB2GRAY )
    cv2.imshow (" WebCam", gray)
```

ذخیره دنباله ویدیویی : برای ذخیره یک دنباله ویدیویی ابتدا باید یک شی از کلاس VideoWriter تعریف کنیم. متد سازنده این کلاس شامل چهار پارامتر به صورت زیر است.

VideoWriter ("Out Video Name" , fourCC , frame per second , video size) -> retVal (bool)

Out Video Name : در این پارامتر نام فایل خروجی و فرمت آن را مشخص می‌کنیم.

fourCC : این پارامتر یک کد چهار بیتی است که کدک ویدیویی را مشخص می‌کند. و بستگی به پلتفورمی که روی آن پیاده سازی میکنیم دارد.

۱- برای ویندوز -> DIVX و...
۲- برای لینوکس -> DIVX, XVID, MJPG , WMV۱ , WMV۲, X۲۶۴ و ...

Frame per second : نرخ فریم فایل خروجی را مشخص میکند.

Vide size : سایز فایل خروجی را بر اساس طول و عرض فریم ها به صورت یک دوتایی دریافت می‌کند

مثال : در مثال زیر ما یک دنباله ویدیویی را از یک آدرس میخوانیم و با تغییری در آن ، آن را به عنوان یک فایل جدید ذخیره می‌نماییم

```
import numpy as np
import cv2

cap = cv2.VideoCapture('vist.avi ')

# Define the codec and create VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi',fourcc, 20.0, (640,480))

while(cap.isOpened()):

    ret, frame = cap.read()

    if ret==True:
        frame= cv2.cvtColor (frame , cv2.COLOR_RGB2GRAY )

        # write the flipped frame
        out.write(frame)
        cv2.imshow('frame',frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        break


cap.release()
out.release()
```

دسترسی به پیکسل های یک تصویر

همان طور که پیش از این گفته شد در کامپیوتر هر تصویر در حقیقت یک ارایه است. که هر درایه آن متعلق به یک ویژگی رنگی یک پیکسل است. برنابراین همانطور که به درایه‌های یک آرایه دسترسی مستقیم داریم. می‌توانیم به پیکسل های یک تصویر نیز دسترسی داشته باشیم. برای دسترسی به یک پیکسل مشخص کافی است شماره سطر و ستون آن پیکسل را بدانیم و در تصاویر رنگی شماره کانالی از پیکسل را که می‌خواهیم تغییر دهیم. نکته مهم آن است که در opencv نقطه بالا و سمت چپ تصویر مختصات (۰,۰) است و با حرکت در راستای طول تصویر به سمت راست مختصات x یا شماره ستون افزوده می‌شود و با حرکت به سمت پایین مختصات y یا شماره سطر افزایش می‌یابد.

Img[row number , column number]

Img[row number , column number , channel number]

نکته : ترتیب کانال ها در opencv در کانال رنگی RGB به صورت BGR است.

بیایید با یک مثال درک بهتری نسبت به این موضوع پیدا کنیم

```
>>> import cv2
>>> import numpy as np

>>> img = cv2.imread('messi5.jpg')

>>> px = img[100,110]
>>> print(px)
[157 166 200]

# accessing only blue pixel
>>> blue = img[100,110,0]
>>> print blue
157
>>> img[100,100] = [255,255,255]
>>> print img[100,110]
[255 255 255]
```

در ابتدا ما کتابخانه opencv و کتابخانه numpy را فرخوانی می‌کنیم. کتابخانه numpy را می‌توان به جرئت کاربردی ترین کتابخانه پایتون دانست که رد پای آن در بسیاری از کتابخانه های پایتون و برنامه‌های مختلف می‌توان دید. در opencv نیز کتابخانه numpy یکی از پایه ها و ستون های اصلی این کتابخانه است. این کتابخانه قابلیت های بسیاری برای کار با آرایه‌ها در پایتون به ما می‌دهد از این رو استفاده از این کتابخانه در پردازش تصویر که هر عکس در حقیقت یک آرایه است، دور از ذهن نیست. در گام بعدی ما یک تصویر با نام 'messi۵.jpg' را لود کرده در در متغییر img قرار می‌دهیم. Img در اینجا در حقیقت تصویر ما یک تصویر رنگی است، بنابراین داده‌های آن به شکل یک آرایه سه‌بعدی ذخیره می‌شوند. ابعاد این آرایه به صورت (عرض تصویر, طول تصویر, ۳) هستند که بعد سوم نشان‌دهنده کانال‌های رنگی B، G و R است

برای مثال، اگر بخواهیم مقدار پیکسلی که در سطر ۱۰۰ و ستون ۱۱۰ قرار دارد را بخوانیم، می‌توانیم آن را در متغیری به نام px قرار دهیم. با چاپ px، مقدار [۱۵۷, ۱۶۶, ۲۰۰] نمایش داده می‌شود.

چون تصویر رنگی است، هر پیکسل دارای سه مقدار است:

مقدار ۱۵۷ مربوط به کانال آبی (B)

مقدار ۱۶۶ مربوط به کانال سبز (G)

مقدار ۲۰۰ مربوط به کانال قرمز (R)

این مقادیر با هم ترکیب شده و رنگ نهایی پیکسل را در تصویر ایجاد می‌کنند. لازم به ذکر است که هر درایه در تصویر از نوع unsigned char است و بنابراین مقداری بین ۰ تا ۲۵۵ دارد.

برای مثال، اگر پیکسل مورد نظر دارای رنگ سبز خالص باشد، مقدار آن برابر [۰, ۲۵۵, ۰] خواهد بود؛ یعنی تنها کانال سبز فعال است و به مقدار کامل خود (۲۵۵) تنظیم شده است.

همچنین می‌توانیم مقادیر هر کانال را به صورت جداگانه بخوانیم. برای مثال، جمله زیر مقدار کانال آبی پیکسل در سطر ۱۰۰ و ستون ۱۱۰ را برمی‌گرداند که در مثال ما برابر ۱۵۷ است.

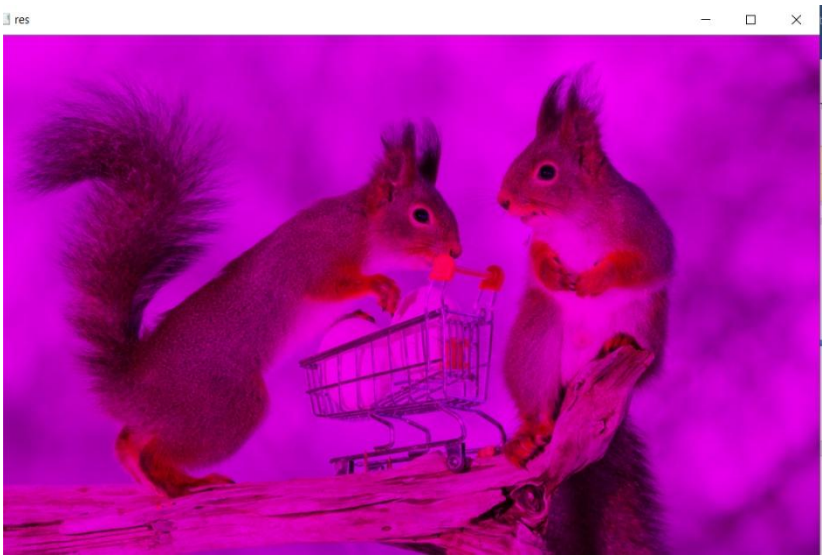
Img[100,110,0]

```
import cv2
img=cv2.imread ("img.jpg")

for i in range(200,800 ):
    img[20,i] = [0,0,255]

cv2.imshow("res", img)
cv2.waitKey(0)
```

بیایید یک مثال دیگر را بررسی کنیم. اینبار قصد داریم با دسترسی مستقیم به پیکسل های یک تصویر یک قرمز بر روی تصویر رسم نماییم. برای این کار کافی است پیکسل هایی را که خط مورد نظر ما از آن ها عبور میکند را مقدارشان را برابر [۰،۰،۲۵۵] قرار دهیم.

حال بیایید و یک مثال دیگر را بررسی کنیم. در این مثال می‌خواهیم مقدار سبز کلیه پیکسل های یک تصویر را حذف کنیم. برای این کار کافی است تا مقدار کانال یکم را برای همه پیکسل های تصویر برابر ۰ قرار دهیم.

```
import cv2

img=cv2.imread ("img.jpg")

for i in range(536):
    for j in range(858):
        img[i,j,1] = 0

cv2.imshow("res", img)
cv2.waitKey(0)
```



همان طور که مشاهده می‌کنید رنگ سبز در تصویر فوق فیلتر شده و کل تصویر تنها از ترکیب رنگ های قرمز و آبی حاصل گردیده و از این رو ما آن را به صورت طیفی از بنفش مشاهده می‌کنیم.

حال اگر تصویر ما خاکستری باشد اوضاع چگونه خواهد شد؟ در این صورت از آنجا که کانال رنگی نداریم آرایه تصویر ما یک آرایه دو بعدی خواهد بود و هر پیکسل تنها دارای یک مقدار می‌باشد که میزان روشنایی پیکسل را مشخص می‌کند. این مقدار اگر برابر ۰ باشد پیکسل مد نظر ما سیاه رنگ، اگر برابر ۲۵۵ باشد پیکسل ما سفید و هر مقداری بین این دو یک طیف خاکستری خواهد بود. همانند قبل می‌توانیم به راحتی به پیکسل های تصویر دسترسی داشته ، مقدار آن ها را بخوانیم و یا تغییر دهیم

```
>>> img=cv2.imread ("img.jpg",0)
>>> img[100,200]
172
>>> img[100,200] = 255
>>> img[100,200]
255
>>> img[100,200,0]
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    img[100,200,0]
IndexError: too many indices for array: array is 2-dimensional, but 3 were
indexed
```

همچنین دسترسی و مقدار دهی پیکسل های یک تصویر توسط توابع item() و itemset() نیز امکان پذیر می‌باشد

Item(row number , column number , channel number)

Itemset((row number, column number , channel number) , pixel new value)

```
# accessing RED value
>>> img.item(10,10,2)
59

# modifying RED value
>>> img.itemset((10,10,2),100)
>>> img.item(10,10,2)
100
```

خواص یک تصویر

هر تصویر دارای ویژگی‌های منحصر به فرد خود است، مانند ابعاد، تعداد پیکسل‌ها و نوع داده هر پیکسل. دانستن این ویژگی‌ها می‌تواند در بسیاری از موارد به ما کمک کند. در ادامه، برخی از این ویژگی‌ها معرفی شده‌اند.

خاصیت shape ابعاد و تعداد کانال‌های یک تصویر را برمی‌گرداند.

- مقدار اول تعداد سطرهای تصویر را نشان می‌دهد، یا به عبارت دیگر، تعداد پیکسل‌های موجود در ارتفاع تصویر.
- مقدار دوم تعداد ستون‌های تصویر را مشخص می‌کند، یعنی تعداد پیکسل‌های موجود در عرض تصویر.
- اگر تصویر رنگی باشد، مقدار سوم تعداد کانال‌های تصویر را نشان می‌دهد.

اگر تصویر خاکستری باشد، تابع shape تنها دو مقدار برمی‌گرداند که سطر و ستون تصویر را مشخص می‌کنند و تعداد کانال‌ها در خروجی دیده نمی‌شود.(rows , columns , channels) → img.shap

```
>>> print img.shape
(342, 548, 3)
rows,cols,channels = img2.shape
```

خاصیت size تعداد پیکسل های یک تصویر را بر میگرداند. این مقدار می‌تواند از حاصل ضرب طول تصویر در عرض آن که از خاصیت Shape بدست می‌ایند نیز حاصل شود

```
>>> print img.size
562248
```

خاصیت dtype نوع پیکلسل یک تصویر را برمیگرداند. که به صورت پیشفرض ۸ بیتی صحیح بدون علامت می‌باشد. یعنی هر درایه آرایه مقداری بین ۰ تا ۲۵۵ خواهد داشت

```
>>> print img.dtype
uint8
```

تمرین: برنامه ای بنویسید که یک تصویر را دریافت کند و تنها بخش هایی از تصویر که دارای رنگ قرمز هستند را نمایش دهد.

راهنما: با بررسی مقدار پیکسل ها در یک حلقه تو در تو، پیکسل هایی که اختلاف کانال قرمز آن ها از هر دو کانال سبز و آبی بیشتر هستند را بدون تغییر و برای سایر پیکسل ها مقدار هر سه کانال پیکسل را برابر ۰ قرار دهید.

توابع رسم شکل روی تصویر

یکی از قابلیت های opencv امکان رسم اشکال هندسی بر روی یک تصویر است. گاهی این رسم شکل به عنوان هدف یک برنامه می‌باشد مانند نرم افزار paint ویندوز. گاهی ممکن است برای نمایشر بصری یک فرایند پردازشی مورد استفاده قرار می‌گیرد مانند مستطیلی که نرم افزار دوربین گوشی شما اطراف صورت های شناسایی شده رسم می‌کند تا به کاربر اطلاعاتی که را پردازش نموده است به صورت گرافیکی نشان دهد. در زیر به معرفی توابع مختلف برای رسم اشکال مختلف بر روی تصویر می‌پردازیم. از آنجا که این بخش دارای فرآیند های واضحی بوده از توضیحات بیشتر خودداری کرده و تنها این توابع را معرفی می‌نماییم.

تابع () line : این تابع برای رسم یک خط روی شکل استفاده می‌شود.

line (img , pt۱ , pt۲ , Color [, thickness , line type]]) ->dst

- img : تصویری که میخواهیم روی آن خط رسم کنیم
- pt۱ , pt۲ : به ترتیب مختصات ابتدا و انتهای خط
- Color : رنگ خط مورد نظر که به صورت یک سه‌تایی وارد میشود (BGR)
- Thickness : ذخامت خط مورد نظر بر اساس پیکسل. اگر این پارامتر برای خطوط بسته مانند دایره ۱- داده شود. شکل مورد نظر به صورت fill تو پر رسم میگردد

- Line type : نوع خط نظیر نقطه چین بودن و... را مشخص می‌کند. مانند LINE_AA که برای خطوط منحنی مناسب است.

مثال :

```
import numpy as np
import cv2

img = cv2.imread("image.jpg")

cv2.line(img , (50,50 ) , (200,200) , (255,0,0) , 5)

cv2.imshow( "out" , img)
```

تابع rectangle()

این تابع برای رسم یک مستطیل روی تصویر مورد استفاده قرار می‌گیرد.

- rectangle (img , pt۱ , pt۲ , Color [, thickness , line type]]) ->dst
- pt۱,pt۲ : به ترتیب مختصات راس بالا سمت چپ و پایین سمت راست

مثال :

```
cv2.rectangle(img,(384,0),(510,128),(0,255,0),3)
```

تابع circle() :

برای رسم دایره روی تصویر کاربرد دارد

- circle (img , center , radius , Color [, thickness , line type]]) ->dst

مثال :

```
cv2.circle(img,(447,63), 63, (0,0,255), -1)
```

تابع ellipse

این تابع برای رسم بیضی کاربرد دارد

ellipse (img , center , axes , angle , start angle , end angle [, Color , thickness]]) ->dst

- center : مختصات مرکز بیضی
- Axes : طول محور اصلی و فرعی بیضی که به صورت یک دوتایی وارد می‌شود.
- Angle : زاویه چرخش بیضی در جهت عقربه های ساعت
- Start/end angle : زاویه شروع و پایان قوس بیضی . برای مثال مقادیر ۰ , ۳۶۰ یک بیضی کامل رسم می‌کند

مثال :

```
cv2.ellipse(img,(256,256),(100,50),0,0,180,255,-1)
```

تابع polylines : برای رسم یک چند ضلعی و یا چند خط جدا مورد استفاده قرار می‌گیرد.

Polylines (img , pts , isClosed , Color [, thickness , line type]]) ->dst

- Pts : این پارامتر آرایه‌ای از مختصات رئوس را تشکیل می‌دهد. هر دو مختصات متوالی در این آرایه تشکیل یک خط می‌دهند
- isClosed : اگر این پارامتر True باشد، خطوط ترسیمی به هم پیوسته و در غیر این صورت جداگانه ترسیم می‌شوند

مثال :

```
Pts=Np.array( [ [10,5] , [20,30] , [70,20] , [50,10] ] , np.uint16)

Pts.reshape((-1,1,2))

Cv2.polylines( img , pts , True , (0 , 255 , 255 )
```

نوشتن متن روی تصویر

برای نوشتن متن روی تصویر از تابع putText() استفاده می‌کنیم.

- putText(img , text , org , font , Scale , color [, thickness , line type]]) ->dst
- text : متن مورد نظر که میخواهیم روی عکس بنویسیم.
- Org : مختصات شروع متن
- Font : فونت متن مورد نظر که از پرچم های پیشفرض نظیر (FONT_HERSHEY_SIMPLEX استفاده می‌شود
- Scale : اندازه متن مورد نظر

مثال:

```
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(img,'OpenCV',(10,500), font, 4,(255,255,255),2,cv2.LINE_AA)
```

ایجاد کنترل کننده Trackbar (اختیاری)

گاهی لازم است برای تنظیم یک ویژگی در تصویر از کنترل‌کننده‌ای به نام trackBar یا همان نوار لغزان استفاده کنیم؛ همان ابزاری که با آن نور صفحه یا میزان صدای گوشی هوشمند خود را تنظیم می‌کنیم. برای ایجاد یک trackBar از تابع createTrackbar() استفاده می‌کنیم.

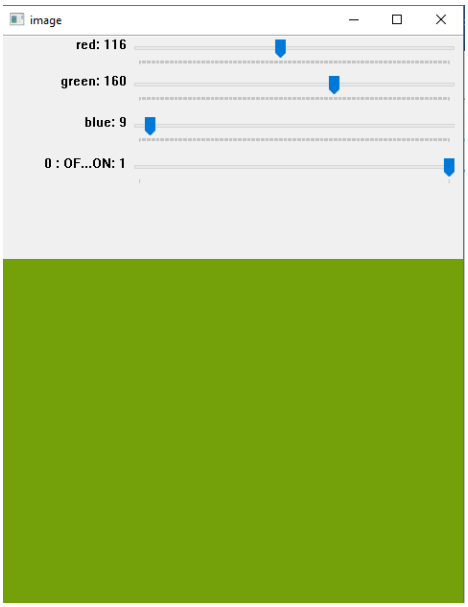
creatTrackbar("Trackbar name" , " windows name" , value , count , onChange)

- Trckbar name : نام نوار مورد نظر که در تصویر نیز به همین نام نشان داده می‌شود.
- Windows name : پنجره‌ای که منوی Trackbar روی آن نمایش داده می‌شود.
- Value : مقدار پیشفرض نوار لغزان
- Count : تعداد پله های نوار لغزان (ماکزیمم مقدار)
- onChange : نام تابعی دلخواه که به محض تغییر Trackbar فراخوانی می‌شود. این تابع باید یک پارامتر ورودی داشته باشد.

برای دریافت مقدار جاری Trackbar از تابع getTrackbarPos() استفاده میکنیم. این تابع نام Trackbar و windows را به عنوان پارامتر دریافت می‌کند

توجه شود که نام پنجره ای که track bar را به آن نسبت می‌دهید و نام پنجره ای که تصویر خود را در آن به نمایش می‌گذارید باید یکسان باشد.

```
import cv2
import numpy as np

def nothing(x):
    pass

mat = np.zeros ((640,480,3) , np.uint8 )
cv2.namedWindow ("image")

cv2.createTrackbar ("red","image",100,255,nothing)
cv2.createTrackbar ("green","image",100,255,nothing)
cv2.createTrackbar ("blue","image",100,255,nothing)


switch = '0 : OFF / 1 : ON'
cv2.createTrackbar (switch,"image" , 0,1,nothing)


while (True):

    key=cv2.waitKey(1)
    #print(key)
    if key==113:
        break
    ######################################
    r=cv2.getTrackbarPos ("red" , "image")
    g=cv2.getTrackbarPos ("green" , "image")
    b=cv2.getTrackbarPos ("blue" , "image")
    sw=cv2.getTrackbarPos (switch , "image")
    ######################################
    if sw==1:
        mat[::]=[b,g,r]

    cv2.imshow ("image" , mat)


pass
cv2.destroyAllWindows()
```



Imag ROI

به هر برشی از تصویر ROI میگویند. گاهی لازم است تا ما تنها بر روی بخش خاصی از تصویر پردازش کنیم برای اینکار ابتدا بخش مورد نظر از تصویر را برش داده و بخش برش داده شده را به عنوان یک تصویر جداگانه در نظر بگیریم و برش دهیم. برای برش یک تصویر ما از عملگر : در پایتون استفاده می‌کنیم. با استفاده از این عملگر می‌توانیم بخش خاصی از یک آرایه را برش دهیم. در صورتی که با این عملگر آشنا نیستید به لینک زیر مراجعه نمایید.

https://www.w۳schools.com/python/numpy_array_slicing.asp

در مثال زیر ما تصویر توپ را در عکس برش داده و در متغییر ball قرار می‌دهیم سپس مقدار ball را برابر یک برش دیگر از تصویر قرار می‌دهیم و به این صورت ما یک کپی از توپ در تصویر ایجاد نمودیم

```
>>> ball = img[280:340, 330:390]
>>> img[273:333, 100:160] = ball
```

Split and merg

گاهی در پردازش تصویر لازم است تا ما تنها بر روی یک کانال رنگی از تصویر پردازش انجام دهیم. برای مثال فرض کنید می‌خواهیم رنگ سبز را در یک تصویر اصلاح کنیم برای اینکار کافی است تا پردازش خود را بر روی کانال سبز معطوف کنیم یا برای مثال میخواهیم. مستطیل های قرمز رنگ در یک تصویر راجداسازی برای این کار می‌توانیم در فضای رنگی HSV پردازش خود را تنها بر روی کانال های h و s انجام دهیم تا تاثیر نور بر روی رنگ را در نظر نگیریم. از این رو جداسازی و ادغام کانال های رنگی یکی از عملیات های کاربردی در پردازش تصویر است. برای جداسازی کانال های یک تصویر از تابع split استفاده می‌کنیم. ورودی این تابع تصویر ما و خروجی آن کانال های رنگی تصویر است که متناسب با فضای رنگی تصویر می‌توانند متفاوت باشند. در مثال زیر ما کانال های یک تصویر رنگی در فضای رنگی BGR را جدا سازی می‌کنیم. هر کدام از این کانال ها خود یک تصویر جداگانه می‌باشند.

یک اشتباه رایج: با جداسازی کانال های یک تصویر یک تصور رایج آن است که ما انتظار داریم که برای مثال کانال رنگی r قرمز رنگ باشد یا کانال b ابی رنگ باشد اما هر کانال رنگی یک آرایه دوبعدی است و از این رو یک تصویر خاکستری می‌باشد. پس با این حساب این تصویر سیاه و سفید چه درکی به ما می‌دهد. بیایید کانال رنگی قرمز را در نظر بگیریم. این کانال یک نسخه سیاه و سفید از تصویر اصلی ما است. اما با این خاصیت که میزان روشنایی هر پیکسل نشان دهنده میزان رنگ قرمز آن پیکسل است. برای مثال اگر در ای تصویر خاکستری یک پیکسل سفید رنگ باشد یا به عبارت دیگر مقدار آن برابر ۲۵۵ باشد، نشان دهنده آن است که رنگ قرمز این پیکسل کامل است یا اگر مقدار آن برابر ۰ یا سیاه رنگ باشد نشان دهنده آن است که پیکسل فوق دارای هیچ رنگ قرمزی نیست. برای درک بهتر بیاید و مثال زیر را در نظر بگیرید

ما می‌خواهیم کانال‌های رنگی تصویر فوق را از یک دیگر جدا کرده و به صورت جداگانه نمایش دهیم. کد پروژه به صورت زیر می‌باشد

```
import cv2
img=cv2.imread ("sp.png")
b,g,r = cv2.split(img)

cv2.imshow('blue channel', b )
cv2.imshow('red channel', r )
cv2.imshow('green channel', g )
cv2.waitKey(0)
```


blue channel


green channel


red channel

بیایید چند نوار رنگی را بررسی کنیم. نوار اول یک نوار قرمز رنگ است. همانطور که مشاهده می‌کنید این نوار در کانال های سبز و آبی به صورت تیره در آمده اما در کانال قرمز نسبتا روشن بوده و به سفید میل می‌کند. و این نشان دهنده آن است که مقدار رنگ سبز و آبی نوار اول بسیار کم و مقدار رنگ قرمز آن زیاد است و ترکیب این سه همان رنگ قرمز خروجی را می‌سازند. نوار سوم و چهارم که به ترتیب سبز و آبی هستند در کانال های رنگی خود روشن و در دو کانال دیگر تیره می‌باشند. حال بیایید و نوار مشکلی رنگ را بررسی کنیم. رنگ مشکی در حقیقت از نبود رنگ های اصلی به دست می‌آید از این رو این نوار در هر سه کانال سبز، قرمز و آبی کاملا سیاه رنگ هستند و این نشان دهنده آن است که این نوار هیچگونه رنگ آبی، قرمز و سبزی ندارد. در مقابل رنگ سفید از ترکیب کامل سه رنگ اصلی بدست می‌آیند. بنابرین مقدار این نوار در هر سه کانال رنگی کامل سفید هستند. حال به سراغ رنگ های ترکیبی می‌رویم. نوار بنفش رنگ را در نظر بگیرید. رنگ بنفش از ترکیب رو رنگ اصلی آبی و قرمز به دست می‌آید. مطابق انتظار در تصاویر بالا مقدار این نوار در کانال های آبی و قرمز نسبتا روشن و در کانال سبز تیره رنگ هستند و از سوی دیگر چون کانال آبی این نوار روشن تر از کانال قرمز است نشان دهنده آن است که مقدار رنگ آبی این نوار بنفش بیشتر از قرمز آن است. و در آخر نوار زرد رنگ، رنگ زرد از ترکیب دو رنک سبز و قرمز تشکیل شده است. بنابراین مطابق انتظار در تصاویر بالا این نوار در کانال های قرمز و سبز روشن و در کانال آبی تیره رنگ است.

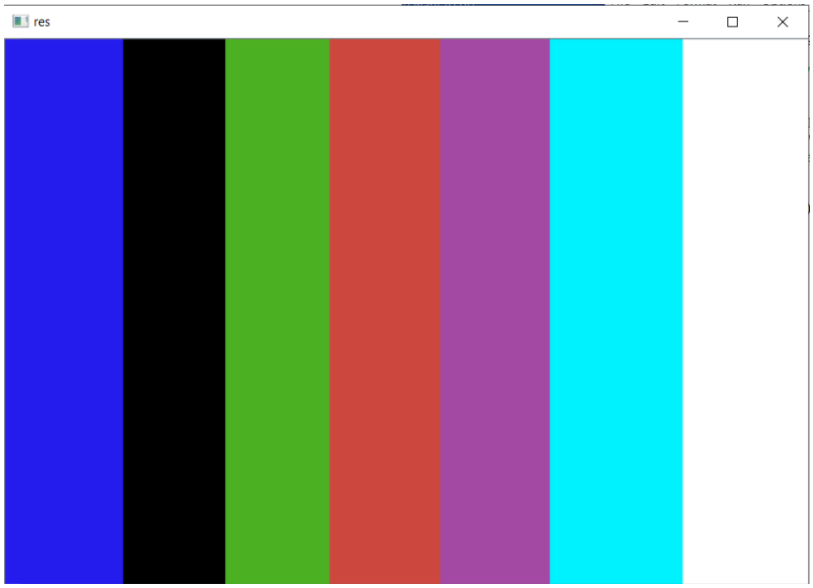
برای جداسازی کانال های رنگی میتوانیم از عملگر : پایتون نیز به صورت زیر استفاده کنیم. در مثال زیر ما کلیه سطر و ستون های تصویر به ازای کانال ۰ که همان کانال آبی است را استخراج نمودیم.

```
>>> b = img[:,:,0]
```

در مقابل ما میتوانیم کانال های رنگی را به یک دیگر متصل نماییم و تصویر نهایی را ایجاد نماییم. اینکار با استفاده از تابع merge در opencv انجام می‌گیرد. ورودی این تابع یک چندتایی (tupel) از کانال های رنگی است و خروجی آن ترکیب کانال ها است. در مثال قبلی ما پس از جدا سازی کانال ها آن ها را مجددا به یک دیگر متصل می‌نماییم . تصویر نهایی همان تصویر ورودی ما خواهد بود

```
import cv2
img=cv2.imread ("sp.png")
b,g,r = cv2.split(img)

res = cv2.merge((b,g,r ))
cv2.imshow('res', res)

cv2.waitKey(0)
```

توجه داشته باشید که ترتیب کانال های رنگی در ورودی تابع merge دارای اهمیت است. برای مثال اگر ما جای کانال قرمز و آبی را جابه جا کنیم تصویر ما به صورت زیر خواهد بود.



نتیجه تا حدی قابل پیشبینی بود. این طور نیست؟

تمرین : کانال های یک تصویر را جدا کرده و سپس مقدار درایه های کانال های سبز و آبی بر ۱۰ تقسیم کنید. و دوباره سه کانال را با هم ترکیب کنید. نتیجه چیست؟

توجه: به یاد داشته باشید که با تقسیم کانال بر ۱۰ نوع درایه های آن از unsigned char به نوع float تغییر پیدا می‌کند و لازم است تا مجدد با استفاده از تابع asType() نوع آن را به np.uint۸ باز گردانید.

ایجاد قاب برای تصاویر(اختیاری)

در opencv قابلیت ایجاد قاب در اطراف تصویر فراهم شده است . برای اینکار میتوان از تابع copyMakeBorder() استفاده کنید. این تابع کاربرد های بسیار دیگر در عملیات کانولوشن ، صفر کردن و... دارد.

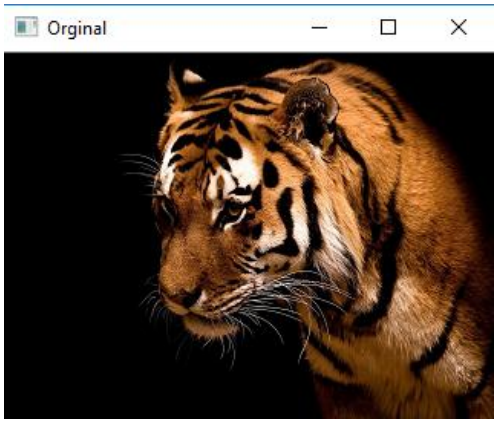
copyMakeBorder(image , top , bottom , left , right , borderType [,value])

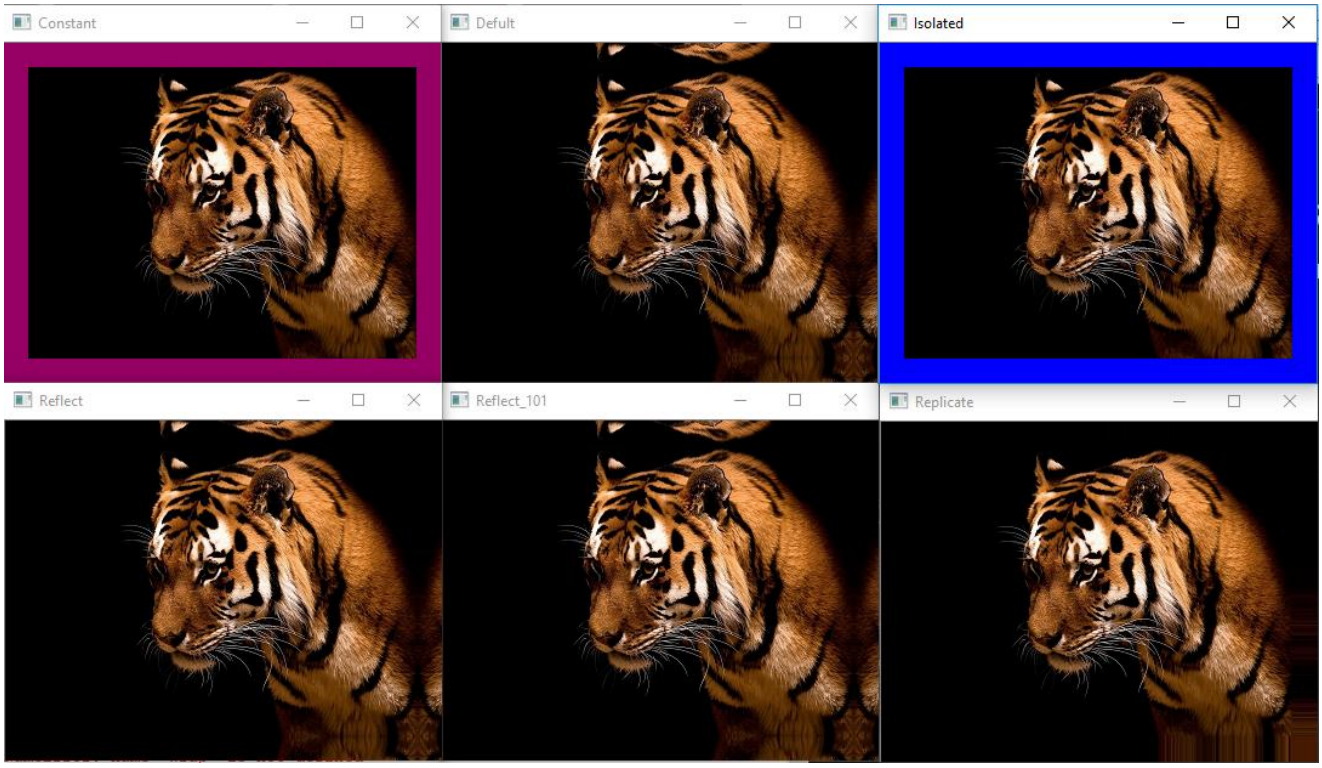
- image : تصویر مورد نظر
- Top,bottom,left,right : حاشیه قاب از چهار طرف تصویر بر حسب پیکسل با این پارامتر ها مشخص می‌شود.
- Border type : نوع قاب را مشخص می‌کند که میتواند یکی از حالات زیر باشد
  - BORDER_RFLECT : این قاب به حالت آینه‌ای بوده و عناصر مرزی را بازتاب می‌کند
  - BORDER_REFLECT_DEFUALT) BORDER_RFLECT_۱۰۱: مانند حالت بالا بوده با کمی تفاوت
  - BORDER_REPLICATE : عناصر حاشیه به صورت سراسری تکرار می‌شوند
  - BORDER_WRAP : برای درک بهتر به مثال زیر توجه فرمایید
  - BORDER_CONSTANT : یک قاب با یک رنگ ثابت ایجاد می‌کند. رنگ این قاب در پارامتر Value مشخص می‌شود.

مثال

```
import cv2
import numpy as np


img=cv2.imread("image.jpg")

cons=cv2.copyMakeBorder(img , 20,20,20,20,cv2.BORDER_CONSTANT , value=(100,0,150))
Def=cv2.copyMakeBorder (img , 20,20,20,20, cv2.BORDER_DEFAULT )
iso=cv2.copyMakeBorder (img , 20,20,20,20, cv2.BORDER_ISOLATED,value=(255,0,0))
ref=cv2.copyMakeBorder (img , 20,20,20,20, cv2.BORDER_REFLECT)
ref101=cv2.copyMakeBorder (img , 20,20,20,20, cv2.BORDER_REFLECT_101 )
rep=cv2.copyMakeBorder (img , 20,20,20,20, cv2.BORDER_REPLICATE )
wrp=cv2.copyMakeBorder (img , 20,20,20,20, cv2.BORDER_WRAP )



cv2.imshow ("Orginal",img)
cv2.imshow ("Constant",cons)
cv2.imshow ("Defult",Def)
cv2.imshow ("Isolated",iso)
cv2.imshow ("Reflect",ref)
cv2.imshow ("Reflect_101",ref101)
cv2.imshow ("Replicate",rep)
cv2.imshow ("Wrap",Wrap)
```

نتایج به صورت زیر می‌باشد:





تذکر : با اضافه شدن قاب به تصویر، اندازه تصویر به میزان حاشیه قاب افزایش می‌یابد

تکنیک های اندازه گیری و بهبود عملکرد

در پردازش تصویر به ویژه به صورت Real Time سرعت پردازش از اهمیت ویژه‌ای برخورد دار است و بهینه سازی الگوریتم یک فرایند معمول در پردازش های سنگین تصویر است. برای بهینه سازی یک الگوریتم داشتن درک صحیحی از سرعت اجرا و حجم پردازشی آن یک عامل خوب برای مقایسه است تا با بررسی این مقادیر از بهینه تر شدن یا کندتر شدن سرعت اجرای الگوریتم خود اطمینان حاصل نماییم. برای این کار می‌توانیم از توابع زیر در openv استفاده نماییم.

تابع getTickCount() تعداد کلاک ها را پس از یک کد نشان میدهد. بنابراین با فراخوانی این تابع قبل و بعد از یک کد، میتوان فهمید چند کلاک cpu برای اجرای آن کد ، اختصاص یافته.

تابع getTickFrequncy مقدار فرکانس را بر میگرداند.

```
e1 = cv2.getTickCount()
# your code execution
e2 = cv2.getTickCount()
time = (e2 - e1)/ cv2.getTickFrequency()
```

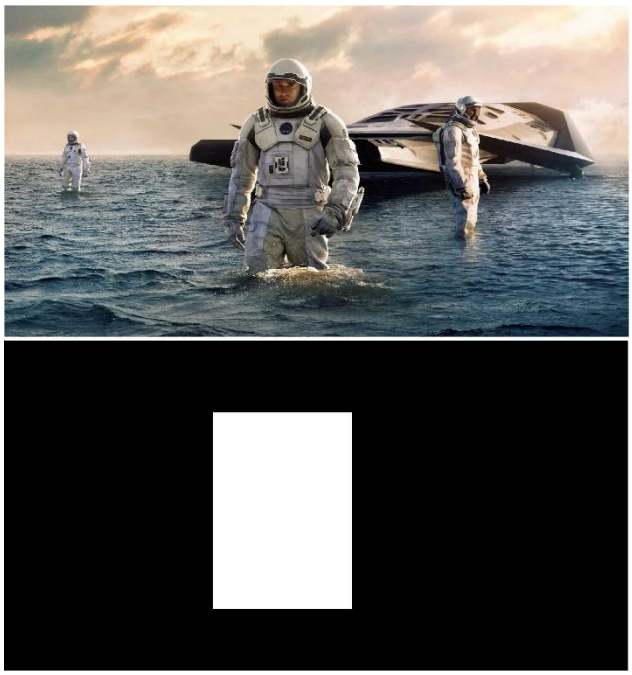
نکته: با ترکیب توابع بالا میتوان فهمید که برنامه برای اجرا چقدر زمان برده است

```
img1 = cv2.imread('messi5.jpg')

e1 = cv2.getTickCount()
for i in xrange(5,49,2):
    img1 = cv2.medianBlur(img1,i)
e2 = cv2.getTickCount()
t = (e2 - e1)/cv2.getTickFrequency()
print t
```

عملیات محاسباتی

همانطور که پیش از گفته شد هر تصویر یک آرایه است و مطابق هر آرایه دیگر می‌توان بر روی آن عملیات های حسابی مانند جمع و تفریق و یا عملیات های منطقی مانند and و or انجام داد. برای مثال فرض کنید ما می‌خواهیم یک لوگو بر روی یک تصویر قرار دهیم. به راحتی می‌توانیم تصویر لوگو را با تصویر مورد نظر جمع کنیم. البته لازم به ذکر است که ابعاد هر دو تصویر باید یکسان باشند. یا برای مثال فرض کنید فریم های متوالی یک دوربین را از یک دیگر تفریق کنیم در این صورت میتوان حرکت های انجام شده در دوربین را تشخیص دهیم . این دقیقا همان کاری است که دوربین های مدار بسته با قابلیت motion detection انجام می‌دهند تا تنها هنگامی فیلم را ذخیره نمایند که در تصویر تحرکی رخ داده است تا در مصرف حافظه بهینه سازی شود. یا برای مثال فرض کنید می‌خواهید در یک تصویر رنگ قرمز را تشخیص دهید. از انجا که بخش هایی از تصویر قرمز هستند که در کانال سبز و آبی مقدار کمی داشته و در کانال قرمز مقدار بالا دارند از این رو برای این کار کافی است تا کانال های ابی و سبز را از

کانال قرمز تفریق کنید و در تصویر حاصل پیکسل هایی را که دارای مقدار بالا مثلا بیشتر 100 هستند را به عنوان رنگ قرمز تشخیص دهید. یا فرض کنید یک تصویر مانند زیر و یک ماسک برای آن مطابق زیر داریم



با AND کردن دو تصویر بالا تنها بخش هایی از تصویر باقی می‌مانند که در ماسک مقدار سفید دارند و سایر بخش ها حذف شده و به رنگ مشکی در می‌آیند. بنابراین با توجه به مثال های بالا انجام عملیات های ریاضی بر روی یک تصویر می‌توانند کاربردی های زیادی در پردازش تصاویر داشته باشند.

عملیات جمع

برای جمع دو تصویر میتوان از تابع ()add و یا عملگر جمع استفاده کرد. این دو راه در هنگام سر ریزی، نتایج متفاوتی دارند. در تابع add اگر نتیجه محاسبات بیشتر از ۲۵۵ باشند (سرریز رخ دهد) مقدار پیکسل برابر ۲۵۵ خواهد شد اما در استفاده از عمل جمع سریز رخ می‌دهد. . برای مثال اگر حاصل جمع دو پیکسل برابر ۲۶۰ شود، مقدار حاصل برابر ۴ خواهد شد. (اعداد ذکر شده برای حالتی است که نوع یک تصویر uint۸ باشد)

مثال:

```
>>> print cv2.add(img1,img2) #250+10 = 260 => 255
[[255]]

>>> print x+y          # 250+10 = 260 % 256 = 4
[4]
```

همچنین برای جمع وزن دار دو تصویر از تابع addweighted() استفاده میشود که ضریبی از تصویر اول را با ضریبی از تصویر دوم جمع میکند

```
img1 = cv2.imread('ml.png')
img2 = cv2.imread('opencv_logo.jpg')

dst = cv2.addWeighted(img1,0.7,img2,0.3,0)

cv2.imshow('dst',dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

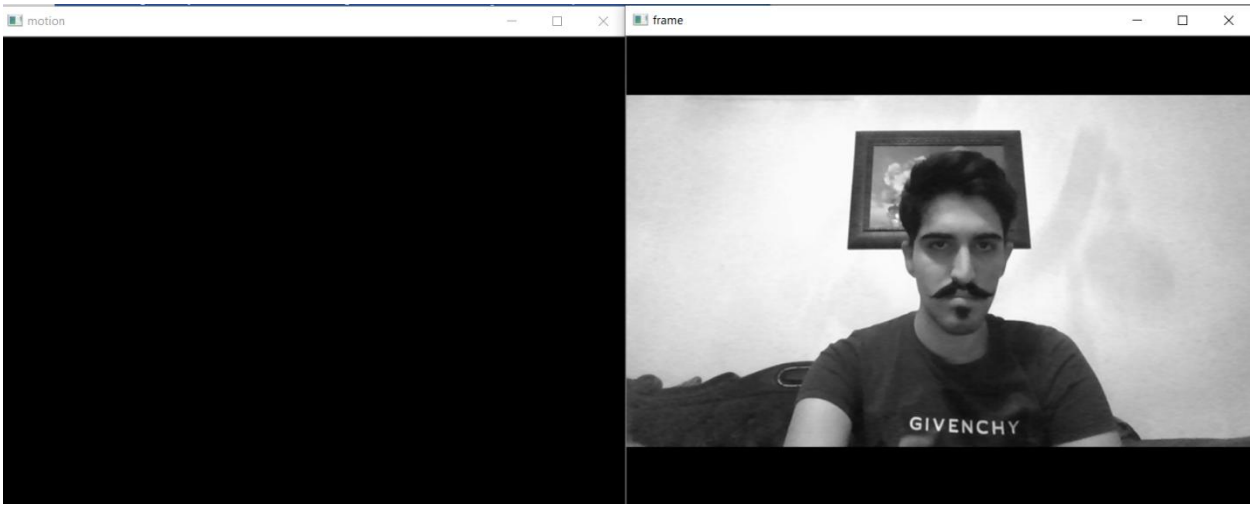
عملگر تفریق

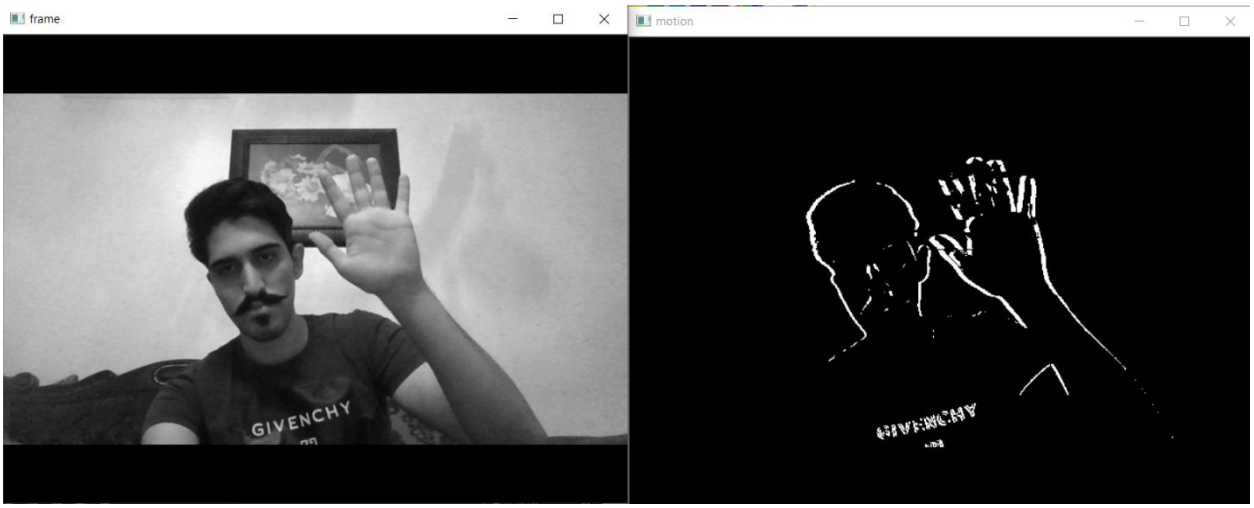
برای تفریق دو تصویر از تابع absdiff استفاده می‌کنیم این تابع دو تصویر را به عنوان ورودی دریافت و قدر مطلق اختلاف آن ها را به عنوان خروجی باز میگرداند.

مثال :

در زیر ما یک برنامه برای نشان دادن حرکت در دوربین نمایش دادیم. در حلقه اصلی برنامه ابتدا یک فریم را از دوربین دریافت و سپس آن را به فضای رنگی سیاه و سفید می‌بریم. سپس اختلاف این فریم را از فریم قبلی حساب کرده و حاصل را در diff قرار می‌دهیم. سپس در یک حلقه پیکسل ها را بررسی می‌کنیم اگر مقدار اختلاف دو فریم متوالی در هر پیکسل بیشتر از ۵۰ باشد مقدار آن را برابر ۲۵۵ و در غیر این صورت برابر ۰ قرار می‌دهیم. این کار به این دلیل است که در فریم های متوالی همواره تغیراتی کمی در شدت نور هر پیکسل وجود دارد و ما با این کار از این تغییرات کم صرف نظر می‌کنیم. در نهایت پس از نمایش خروجی، مقدار فریم فعلی را در متغییر فریم قبلی کپی می‌کنیم و حلقه مجدد تکرار می‌شود. اجرای این برنامه به دلیل حلقه‌های for کمی زمان بر بوده و از این رو فریم ریت خروجی کم می‌باشد. پس از یادگیری توابع threshold می‌توانید به جای این حلقه های for تو در تو از این تابع استفاده کنید و سرعت اجرای برنامه خود را چند برابر کنید

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)
_,frame_prev = cap.read()
frame_prev = cv2.cvtColor(frame_prev, cv2.COLOR_BGR2GRAY )

while True :
    _,frame = cap.read()
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY )
    diff = cv2.absdiff( frame, frame_prev )
    h,w = frame.shape
    for i in range(h):
        for j in range(w):
            if diff[i,j] > 50 :
                diff[i,j] = 255

            else :
                diff[i,j] = 0


    cv2.imshow('motion', diff)
    cv2.imshow('frame', frame )
    cv2.waitKey(30)

    frame_prev= np.copy(frame)
```

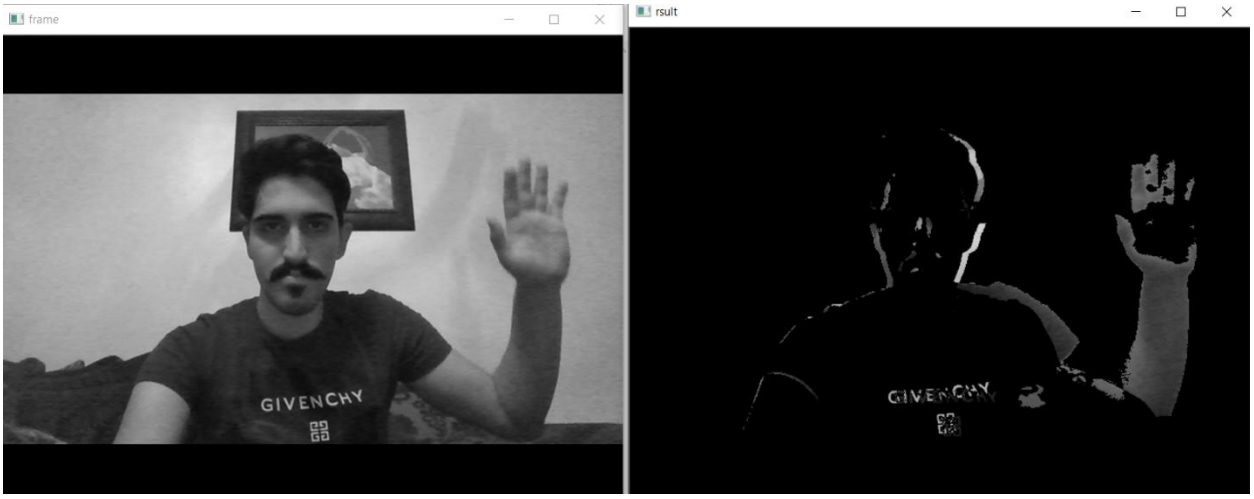




عملگر های منطقی

عملگرهای منطقی عملگرهایی هستند که به صورت بیتی بر روی پیکسل های یک تصویر اعمال می‌شوند و عبارتند از and , or , not و...

معمولا از عملگرهای منطقی برای انجام عملیات بر روی ماسک ها استفاده می‌شوند. همانطور که پیش از این گفته شده ماسک ها تصاویر هستند که تنها دارای دو مقدار سیاه (۰) و سفید (۲۵۵) هستند. ماسک ها برای استخراج محتویات خاصی از تصویر کاربرد دارند. در بخش های بعدی ما با استفاده از عملیات هایی نظیر threshold و تشخیص لبه ماسک های مورد نظر خود را ایجاد می‌کنیم و در فرآیند هایی نظیر تشخیص کانتورها از این ماسک ها برای استخراج ویژگی های هندسی object مورد نظر استفاده می‌نماییم. پیش از این لازم است تا شما با انجام عملیات های منطقی آشنا باشید. برای نمونه در مثال قبل که یک برنامه برای تشخیص حرکت نوشتیم تصویر خروجی ما در حقیقت یک ماسک بود چرا که تنها از پیکسل های ۰ و ۲۵۵ تشکیل شده بود. برای نمونه با استفاده از عملیات منطقی and بر روی ماسک مثال قبل (diff) و تصویر ورودی (frame) می‌توانیم بخش هایی را که در آن حرکت رخ داده است را استخرج نماییم.




```
res = cv2.bitwise_and( frame, diff )
cv2.imshow('motion', diff)
cv2.imshow('frame', frame )
cv2.imshow('rsult', res )
```

توابع عملگر های منطقی به صورت زیر می‌باشند.

- bitwise_and(src۱ , src۲ [, dst [, mask]]) -> dst
- bitwise_and(src۱ , src۲ [, dst [, mask]]) -> dst
- bitwise_not(src۱ [, dst [, mask]]) -> dst
- bitwise_xor(src۱ , src۲ [, dst [, mask]]) -> dst

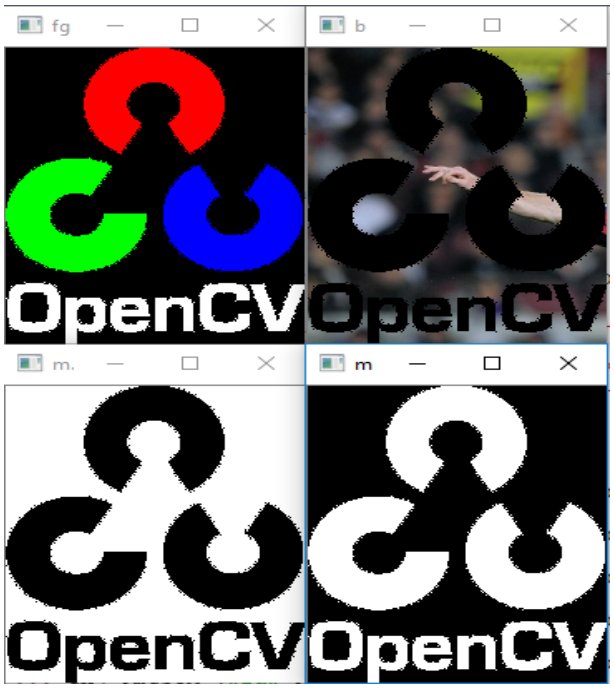
ورودی‌های این توابع عبارتند از:

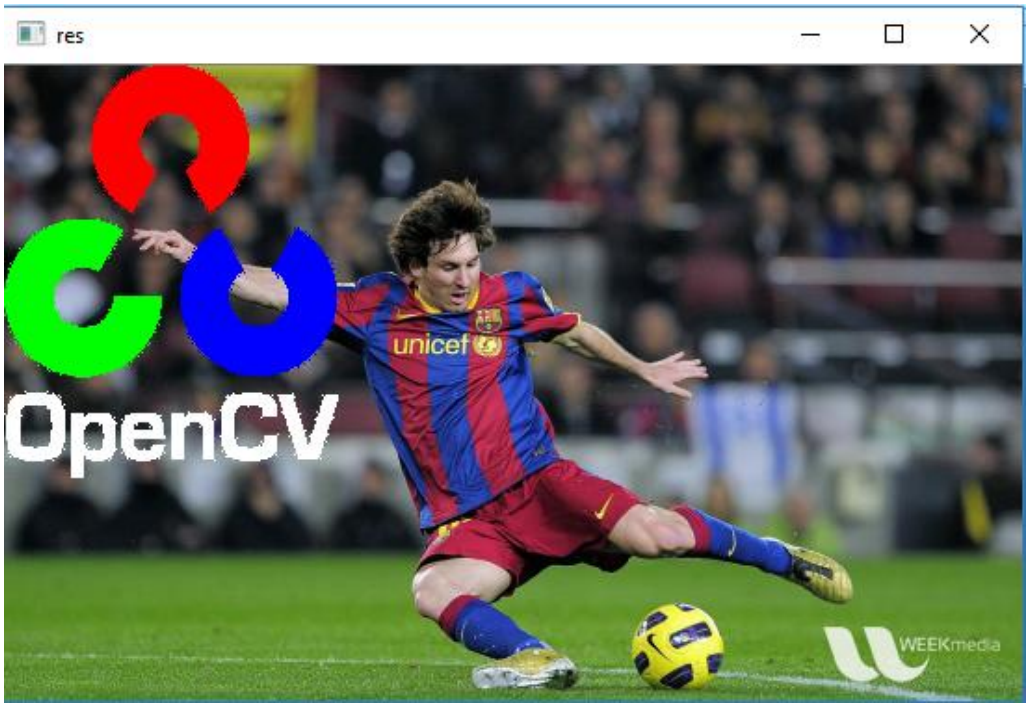
- src۱ : تصویری ورودی اول
- Src۲ : تصویر ورودی دوم
- Dst : تصویر خروجی

مثال :

```
import cv2
# Load two images
img1 = cv2.imread('messi.jpg')
img2 = cv2.imread('open_cv.png')
# I want to put logo on top-left corner, So I create a ROI
rows,cols,channels = img2.shape
roi = img1[0:rows, 0:cols ]
# Now create a mask of logo and create its inverse mask also
img2gray = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)
ret, mask = cv2.threshold(img2gray, 10, 255, cv2.THRESH_BINARY)
mask_inv = cv2.bitwise_not(mask)

# Now black-out the area of logo in ROI
img1_bg = cv2.bitwise_and(roi,roi,mask = mask_inv)

# Take only region of logo from logo image.
img2_fg = cv2.bitwise_and(img2,img2,mask = mask)

# Put logo in ROI and modify the main image
dst = cv2.add(img1_bg,img2_fg)
img1[0:rows, 0:cols ] = dst

cv2.imshow('res',img1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

در مثال بالا میخواهیم یک لوگو را بر روی یک تصویر قرار دهیم. ابتدا تصویر اصلی و تصویر لوگوی خود را لود می‌کنیم. سپس یک برش از بخشی از تصویر که میخواهیم لوگو روی آن قرار گیرد می‌دهیم (roi) . سپس تصویر لوگو را با تبدیل به فضای رنگی خاکستری و اعمال یک حد آستانه مناسب، به یک تصویر باینری تبدیل می‌کنیم (mask) در این تصویر لوگو سفید و بکگراند سیاه می‌باشد. سپس با اعمال یک عملگر not روی آن یک ماسک بر عکس ماسک قبلی ایجاد میکنیم.(mask_inv) . سپس mask را روی تصویر لوگو اعمال میکنیم تا بگراند آن حذف و تنها لوگو باقی بماند (img۱_fg). همچنین mask_inv را روی تصویر برش خورده اعمال میکنیم تا محل حذف (سیاه)شود. (img۱_bg). سپس این دو تصویر را باهم جمع میکنیم (dst) و حاصل را روی تصویر اصلی قرار میدهیم. عملگر های and در اینجا برای اعمال ماسک مورد استفاده قرار گرفته است.


fg
b
m.
m
OpenCV
OpenCV
OpenCV
OpenCV


res
OpenCV
unicef
WEEKmedia

تبدیل فضای رنگ تصاویر

فضاهای رنگی متفاوت دارای کاربردهای خاصی هستند. برای مثال فرض کنید می‌خواهید یک صورت را در تصویر شناسایی کنید. برای شناسایی یک صورت ما نیازی به تصویر رنگی نداریم چرا که هنگامی که شما به یک تصویر سیاه سفید از یک انسان نگاه کنید باز هم می‌توانید صورت فرد را شناسایی کنید یا مثلا در فیلم آواتار شما برای شناسایی صورت ها قطعا مشکلی نداشتید با اینکه رنگ صورت ها کاملا آبی رنگ و متفاوت بود. بنابراین شما برای شناسایی یک صورت به فضای رنگی RGB یا HSV نیازی ندارید بنابرین با انجام پردازش بر روی تصویر خود در فضای رنگی خاکستری می‌توانید حجم پردازشی را تا حد زیادی بکاهید. یا حتی در مثال شناسایی حرکت که پیش از این داشتیم نیازی نبود تا ما بر روی تصاویر رنگی پردازش کنیم چرا که پردازش بر روی تصاویر خاکستری علاوه بر کافی بودن اطلاعات، پردازش مارا سبک تر و برنامه ما را آسان تر کرده است. یا برای مثال فرض کنید می‌خواهید در تصاویر هوایی که از یک پهپاد بدست می‌آید بخش های سرسبز را شناسایی کنید. احتمالا فضای رنگی HSV انتخاب بهتری از RGB است چرا که با حذف کانال V می‌توانید تاثیر نور را بر روی رنگ در نظر نگیرید و برنامه شما بتواند در شرایط نوری متفاوت بهتر کار کند. مگر آنکه قصد داشته باشید برنامه ای بنویسید که تنها در روز های افتابی کار کند. بنابراین تبدیل تصاویر به فضاهای رنگ متفاوت و مناسب برای پروژه شما برای شروع پردازش انتخاب مناسبی است.

برای تبدیل یک تصویر از یک فضای رنگی به یک فضای دیگر از تابع cvtColor() استفاده میکنیم. این تابع بصورت زیر است:

cvtColorI (image , Code)

- image : تصویر ورودی
- Code : پارامتری که فضای رنگی مبدا و مقصد را مشخص می‌کند که بیش از ۱۵۰ پرچم برای این پارامتر تعریف شده است . نظیر: COLOR_HSV۲BGR COLOR_BGR۲GRAY COLOR_BGR۲HSV و...

در hsv محدوده رنگ یا h [۰,۱۷۹] محدوده اشباع رنگ یا s [۰,۲۵۵] و محدوده روشنی و تیرگی یا v [۰,۲۵۵] است

مثال: در

```
Cv2.cvtColor(img , cv2.COLOR_BGR2GRAY)
#convert from BGR to Gray
```

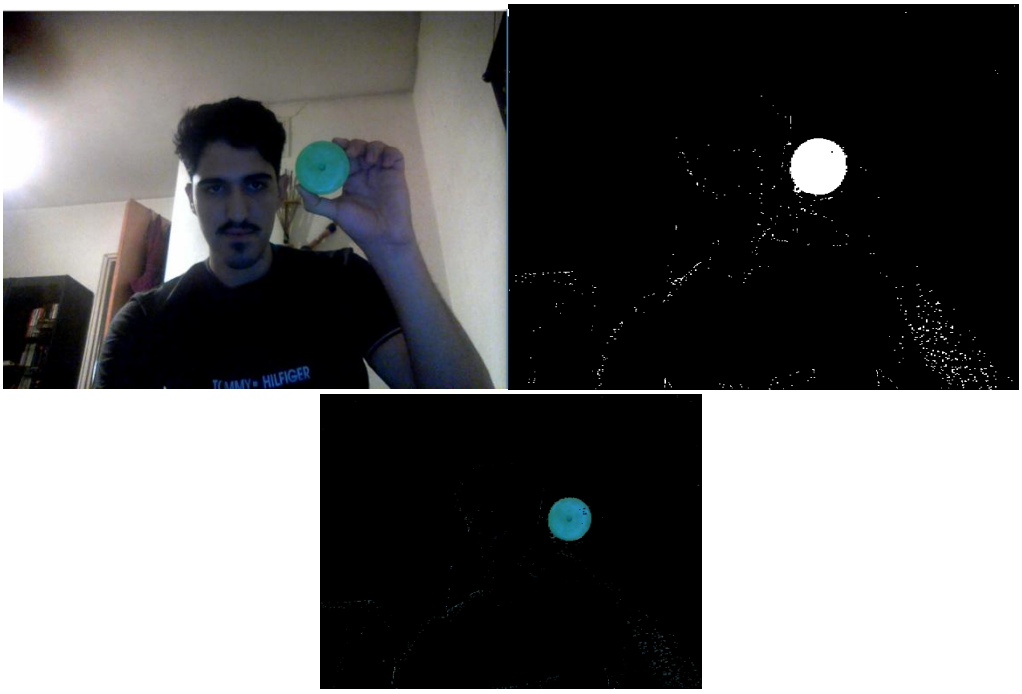
مثال زیر ما یک تصویر را از فضای رنگ BGR به Gray تبدیل میکنیم

نکته مهم : تابع imshow تصاویر را در فرمت BGR نمایش می‌دهد. بنابراین هنگامی که شما یک تصویر را به فضای رنگی دیگری مانند HSV می‌برید. در هنگام نمایش، رنگ های تصویر به هم ریخته خواهد بود. توجه داشته باشید که بهم ریختگی در رنگ تنها در نمایش تصویر در opencv است و تصویر شما در حقیقت همان رنگبندی اصلی را دارد اما در یک فضای رنگی دیگر

مثال : در مثال زیر ما قصد داریم تا یک شی سبز رنگ را دریک تصویر پیدا کنیم. در این مثال ما ابتدا تصویر را به فضای رنگ hsv می‌بریم تا نور تاثیر کمتری داشته باشد. سپس دو سطح رنگ سبز تعریف میکنیم. برای بدست آوردن مقادیر مناسب برای این رنگ ها میتوان از نرم افزار های طراحی نظیر فتوشاب استفاده کرد. سپس از طریق تابع inRange() یک ماسک برای تصویر می‌سازیم. این تابع پیکسل هایی که بین سبز کمرنگ و پرنگ هستند را سفید و سایر پیکسل هارا سیاه می‌کند. سپس این ماسک را روی تصویر اعمال می‌کنیم.

- برای بدست آوردن مقادیر سبز در HSV علاوه بر استفاده از نرم افزار های مختلف، میتوان از خود تبدیل رنگ پایتون استفاده کرد و رنگ مورد نظر را از BGR به HSV تبدیل کرده و مقادیر آن را بدست آورد

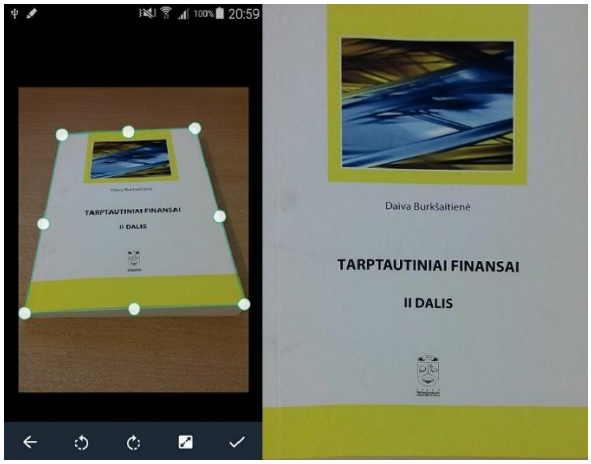
```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)
while(True):
    _, frame = cap.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    # define range of blue color in HSV
    lower_blue = np.array([80,50,50])
    upper_blue = np.array([100,255,255])
    # Threshold the HSV image to get only blue colors
    mask = cv2.inRange(hsv, lower_blue, upper_blue
    # Bitwise-AND mask and original image
    res = cv2.bitwise_and(frame,frame, mask= mask)

    cv2.imshow('frame',frame)
    cv2.imshow('mask',mask)
    cv2.imshow('res',res)
    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break

cv2.destroyAllWindows()
```



تبدیل هندسی تصاویر

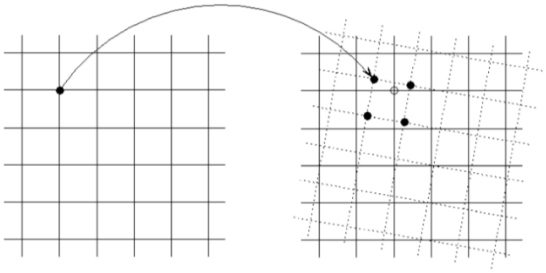
تبدیل هندسی در تصاویر به عملیاتی مانند چرخش، انتقال، زوم، تغییر پرسپکتیو و غیره گفته می‌شود.هر بار که در اینستاگرام می‌خواهید یک استوری بگذارید و تصویر خود را با چرخش و زوم به حالت ایده‌آل درمی‌آورید، در واقع یک تبدیل هندسی روی تصویر انجام می‌دهید.

همچنین احتمالا با نرم‌افزارهای اسکنر گوشی‌های هوشمند آشنا هستید؛ هنگامی که از یک صفحه جزوه عکس می‌گیرید، این نرم‌افزارها علاوه بر اصلاح رنگ، پرسپکتیو تصویر را نیز به گونه‌ای تنظیم می‌کنند که تصویر بدون زاویه به نظر برسد، انگار کاملا از رو به رو و به صورت تمام صفحه عکس‌برداری شده است..



در تبدیل هندسی هر پیکسل با مختصات (x,y) به یک پیکسل با مختصات (u,v) تصویر می‌شود.

$$u = f_1(x, y)$$
$$v = f_2(x, y)$$



Affine Transorm

نوعی انتقال هندسی است که در آن همبستگی تصویر حفظ می‌شود. برای مثال اگر پیکسل هایی از تصویر بر روی یک خط قرار داشته باشند، در تصویر خروجی نیز این پیکسل ها همچنان بر روی یک خط قرار خواهند داشت. این تبدیل هندسی ، نسبت مسافت را نیز در تبدیل حفظ می‌نماید. برای مثال اگر یک پیکسل بر روی یک خط در ¼ آن قرار داشته باشد، پس از تبدیل باز هم در موقعیت ¼ آن خط قرار خواهد داشت. این تبدیل هندسی به طور کلی شامل عملیات های انتقال، چرخش، کوچک نمایی و بزرگ نمایی تصویر و برش یک عکس می‌شود. بنابراین در انجام این عملیات ها از این تبدیل هندسی استفاده می‌شود. این تابع تبدیل بر اساس یک ماتریس تبدیل با ابعاد ۳×۲ به صورت زیر کار می‌کند

$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \texttt{map\_matrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

Xi , yi مختصات های پیکسل ورودی می‌باشند که پس از ضرب در ماتریس تبدیل مختصات های جدید این پیکسل را پس از تبدیل بدست می‌آوریم. بیایید کمی موضوع را باز تر کنیم. فرض کنید ماتریس تبدیل ما به صورت زیر باشد.

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}$$

بر اساس رابطه بالا خواهیم داشت.

$$u = c_{11}x + c_{12}y + c_{13}$$
$$v = c_{21}x + c_{22}y + c_{23}$$

پارامترهای c۱۳ و c۲۳ پارامتر های انتقال هستند برای مثال فرض کنید ماتریس تبدیل ما به صورت زیر باشد

$$M = \begin{bmatrix} 1 & 0 & 10 \\ 0 & 1 & 20 \end{bmatrix}$$

آنگاه خواهیم داشت:

$$u = x + 10$$
$$v = y + 20$$

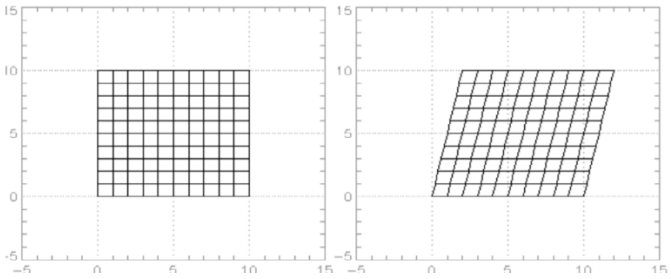
این یک ماتریس برای انتقال تصویر است. در این مختصات x به اندازه c۱۳ و مختصات y هر پیکسل به اندازه c۲۳ جابه جا می‌شوند.

این بار ماتریس تبدیل را به صورت زیر در نظر بگیرید

$$\begin{vmatrix} 1 & 0.2 & 0 \\ 1 & 0 & 0 \end{vmatrix}$$

بنابر این طبق تبدیل affine یک برش در محور x خواهیم داشت:

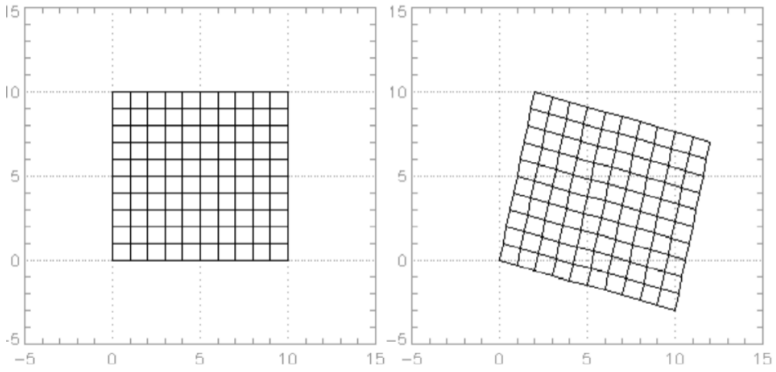
$$u = x + 0.2y$$
$$v = y$$



یا برای ماتریس تبدیل زیر چرخش خواهیم داشت

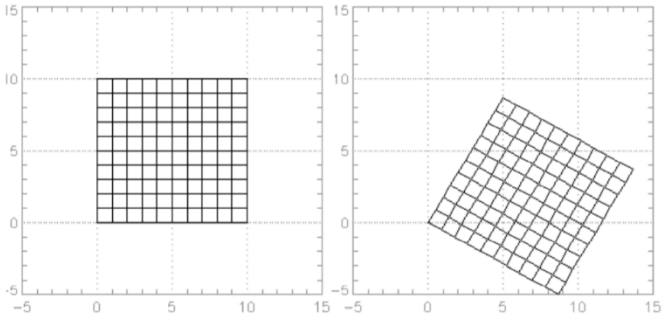
$$\begin{vmatrix} 1 & 0.2 & 0 \\ 0.3- & 1 & 0 \end{vmatrix}$$

$$u = x + 0.2y$$
$$v = -0.3x + y$$



برای ایجاد یک چرخش در تصویر به اندازه زاویه تتا بر اساس مثلثات روابط زیر بدست می‌آید

$$u = x\cos\theta + y\sin\theta$$
$$v = -x\sin\theta + y\cos\theta$$



بنابراین ماتریس انتقال بصورت مقابل خواهد

$$\begin{vmatrix} \cos\theta & \sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \end{vmatrix}$$

بنابراین با تعیین صحیح ماتریس تبدیل می‌توان تبدیلات هندسی مختلف را پیاده سازی کرد. برای پیاده سازی تبدیل هندسی affine در opencv از تابع ()wrapAffine استفاده می‌شود. در ادامه پیاده سازی چند نمونه تبدیل هندسی را درکتابخانه opencv آورده ایم. یک تبدیل هندسی خود می‌تواند شامل چند تبدیل هندسی باشد. برای مثال علاوه بر چرخش، انتقال نیز صورت بگیرد. نکته حائز اهمیت آن است که بر اساس ابعاد تصویر خروجی ممکن است برخی پیکسل‌های تصویر، در این دامنه قرار نگیرند و حذف شوند. از این رو انتخاب ابعاد مناسب برای تصویر خروجی می‌تواند حائز اهمیت باشد. برای محاسبه ابعاد تصویر خروجی می‌توانید مختصات نقاط چهار گوشه تصویر را با استفاده از تابع تبدیل محاسبه نمایید یا با تعیین عددس ابعاد تصویر خروجی و با آزمون و خطا به ابعاد مورد نظر خود برسید. شاید استفاده از آزمون و خطا روش مناسبی به نظر نرسد اما لازم به ذکر است که در بسیاری ها بخش های پردازش تصویر، یکی از رایج ترین و مرسوم ترین روش های رسیدن به پارامتر های مطلوب، آزمون خطا می‌باشد.

انتقال

انتقال (Translation) یکی از ساده‌ترین تبدیل‌های هندسی در تصاویر است که تصویر را در راستای محورهای x و y جابه‌جا می‌کند بدون اینکه شکل یا اندازه پیکسل‌ها تغییر کند. در عمل، برای هر پیکسل تصویر، موقعیت جدید آن با اضافه کردن مقادیر دلخواه tx و ty به مختصات x و y تعیین می‌شود. برای اعمال انتقال در OpenCV، ابتدا ماتریس انتقال ۲×۳ ساخته می‌شود و سپس با تابع warpAffine روی تصویر اعمال می‌گردد.

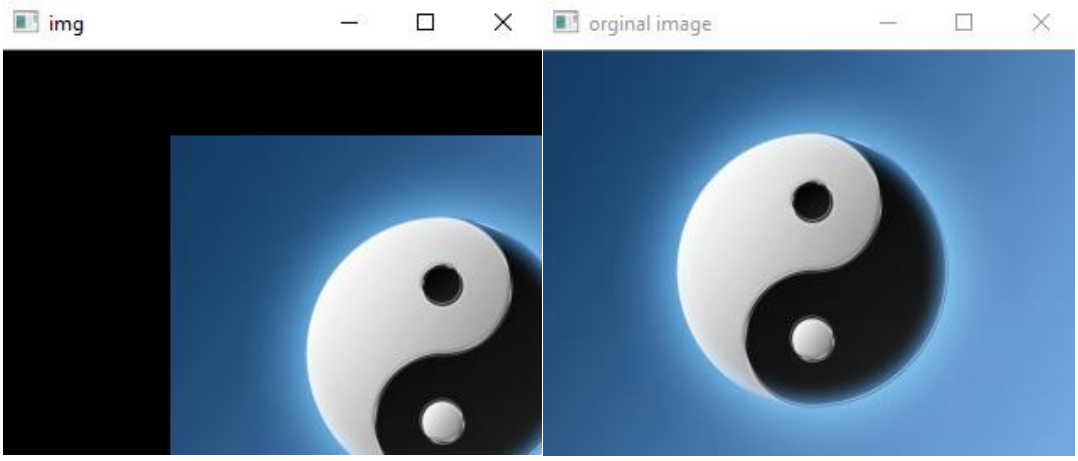
warpAffine(image , M , dsize)

- image : تصویر ورودی
- M : ماتریس تبدیلی که می‌خواهیم روی تصویر اعمال کنیم این ماتریس یک ماتریس ۲×۳ است. برای انجام عمل جابه‌جایی باید ماتریس تبدیل را به صورت زیر تعریف کنیم که در آن tx و ty میزان جابه‌جایی برحسب پیکسل در جهت های xوy را نشان می‌دهند

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

- Dsize : اندازه تصویر خروجی بر اساس عرض و ارتفاع می‌باشد که به صورت یک دوتایی به تابع داده می‌شود.

```
import cv2
import numpy as np

img=cv2.imread ("img.jpg")

rows,cols,_ = img.shape
M = np.float32([[1,0,100],[0,1,50]])
dst = cv2.warpAffine(img,M,(cols,rows))

cv2.imshow('img',dst)cv2.imshow ("orginal image",img)
```



چرخش

چرخش (Rotation) یکی از تبدیلات هندسی در تصاویر است که تصویر را حول یک نقطه مشخص می‌چرخاند. این چرخش می‌تواند در جهت عقربه‌های ساعت یا خلاف آن انجام شود و زاویه چرخش با درجه یا رادیان مشخص می‌گردد. هنگام چرخش، شکل و اندازه پیکسل‌ها حفظ می‌شوند، اما موقعیت آن‌ها تغییر می‌کند.

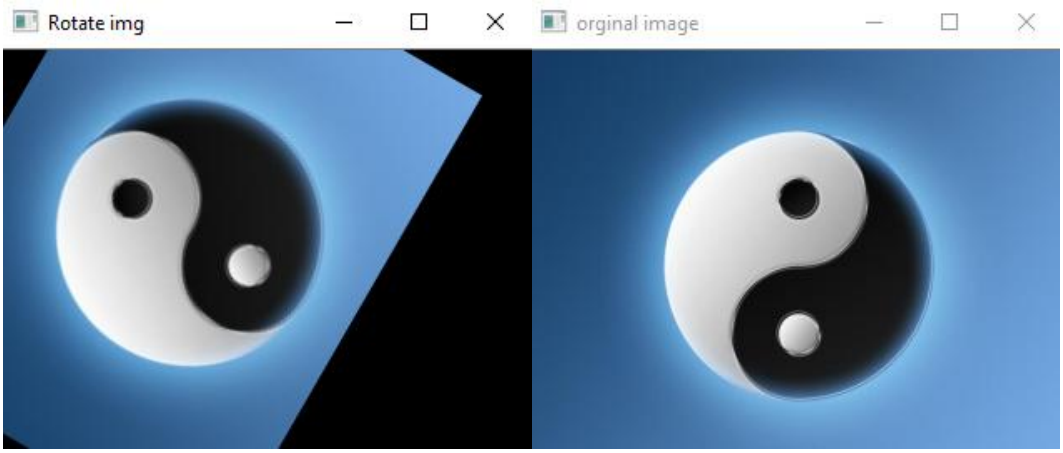
برای چرخاندن تصویر در OpenCV می‌توان از تابع warpAffine() استفاده کرد. برای این کار، ابتدا باید ماتریس تبدیل چرخش تعریف شود. این ماتریس می‌تواند به صورت دستی ساخته شود یا با استفاده از تابع آماده getRotationMatrix۲D() در OpenCV ایجاد گردد.

getRotationMatrix۲D(center , angle , scale)

- center : مرکز چرخش که تصویر حول آن دوران میکند و مختصات آن به صورت یک دوتایی می‌دهیم
- Angle : زاویه دوران تصویر
- Scale : میزان بزرگ نمایی تصویر

مثال :

```
import cv2
import numpy as np

img=cv2.imread ("img.jpg")
cv2.imshow ("orginal image",img)
rows,cols,_ = img.shape

M=cv2.getRotationMatrix2D ( (rows/2 , cols/2) , 60 ,1)
rotImg=cv2.warpAffine ( img, M , (cols , rows))

cv2.imshow('Rotate img',rotImg)
```



محاسبه تابع تبدیل

در بسیاری از عملیات پردازش تصویر، ما ممکن است مستقیماً ماتریس تبدیل را نداشته باشیم، اما مختصات چند نقطه در تصویر مبدا و مختصات متناظر آن‌ها در تصویر مقصد را می‌دانیم. با استفاده از این نقاط، می‌توان روابط خطی یا آفاین بین موقعیت‌های مبدا و مقصد را برقرار کرد. سپس با حل این معادلات، ماتریس تبدیل بدست می‌آید و می‌توان آن را برای اعمال تبدیل روی تصویر استفاده کرد.

نرم‌افزارهای اسکنر گوشی‌های هوشمند هنگام عکس‌برداری از یک صفحه جزوه، سعی می‌کنند پرسپکتیو تصویر را اصلاح کنند. برای این کار، نرم‌افزار چهار گوشه صفحه را شناسایی می‌کند و می‌خواهد این چهار نقطه در تصویر نهایی به چهار گوشه یک مستطیل دقیق تبدیل شوند. با داشتن مختصات این چهار نقطه در تصویر اصلی و مختصات متناظر آن‌ها در تصویر مستطیل شده، نرم‌افزار می‌تواند ماتریس تبدیل هوموگرافی را محاسبه کرده و تصویر را اصلاح کند.

ماتریس تبدیل، از حل معادله زیر حاصل می‌شود

$$u = c_{11}x + c_{12}y + c_{13}$$
$$v = c_{21}x + c_{22}y + c_{23}$$

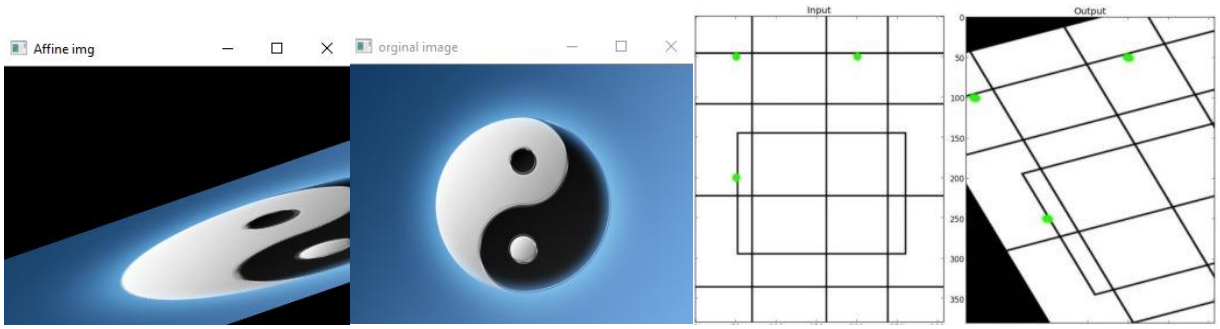
برای حل این معادله ما به سه نقطه از تصویر ورودی و مختصات تبدیل شده آن‌ها در تصویر خروجی نیاز داریم. در کتابخانه opencv تابع getAffineTransform() برای انجام این کار در نظر گرفته شده است. که بر اساس مختصات مبدا و مقصد نقاط مدنظر، ماتریس تبدیل را بدست می‌آورد. و سپس می‌توان با استفاده از تابع wrapAffine از این ماتریس برای تبدیل هندسی تصویر بهره ببریم.

getAffineTransform(pts۱ , pts۲)

- pts۱ : آرایه‌ی سه مختصات مبدا نقاط انتخابی
- Pts۲ : آرایه‌ی سه مختصات مقصد نقاط انتخابی

مثال :

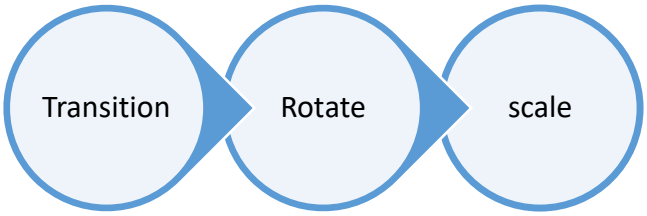
```
import cv2
import numpy as np

img=cv2.imread ("img.jpg")
cv2.imshow ("orginal image",img)
rows,cols,_ = img.shape

pts1 = np.float32([[50,50],[200,50],[50,200]])
pts2 = np.float32([[10,200],[300,100],[100,250]])

M = cv2.getAffineTransform(pts1,pts2)
affine = cv2.warpAffine(img,M,(cols,rows))

cv2.imshow('Affine img',affine)
```



تبدیلات هندسی پیچیده تر

تبدیلات هندسی پیچیده‌تر را می‌توان با ترکیب چند تبدیل آفاین (Affine) به دست آورد. برای مثال، فرض کنید می‌خواهیم یک تصویر را ابتدا انتقال دهیم، سپس آن را به اندازه یک زاویه α چرخانده و در نهایت مقیاس (scale) تصویر را تغییر دهیم.

برای انجام این کار، می‌توان این تبدیلات را به صورت دنباله‌ای روی تصویر اعمال کرد، به طوری که خروجی هر مرحله به عنوان ورودی مرحله بعدی استفاده شود.



اما در مقابل می‌توان بر اساس قواعد جبر خطی و ماتریس ها به این واقعیت دست یافت که با استفاده از ضرب ماتریس‌ها تبدیل در یک دیگر، می‌توان ماتریس تبدیلی تولید کرد که کلیه این تبدیلات را انجام دهد. اما از آنجا که ابعاد این ماتریس های تبدیل ۲×۳ می‌باشند. امکان ضرب این ماتریس ها در یک دیگر وجود ندارد. یک راهکار بسیار جالب تغییر شکل ماتریس تبدیل به یک ماتریس با ابعاد ۳×۳ می‌باشد. به گونه‌ای که معادلات تغییری نکند. این ماتریس تبدیل به صورت زیر خواهد بود

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

به سادگی با ضرب ماتریس های بالا، متوجه خواهید شد که معادلات همانند قبل خواهد با این تفاوت که ماتریس تبدیل ما اکنون یک ماتریس تبدیل ۳×۳ است.

U = ax + by + c v = dx + ey + f ۱ = ۰x + ۰y + ۱

با این تغییر اکنون قادر خواهیم بود تا ماتریس های تبدیل مختلف را در یک دیگر ضرب کرده و یک تبدیل هندسی پیچیده تر داشته باشیم

ماتریس تبدیل انتقال × ماتریس تبدیل چرخش × ماتریس تغییر ابعاد = ماتریس تبدیل نهایی

با تبدیل هندسی بر اساس ماتریس نهایی ما به خواسته خود در مثالی که در ابتدا گفتیم خواهیم رسید. اما با پردازش کمتر چرا که تنا با یکبار تبدیل هندسی به تصویر مطلوب خود دست پیدا می‌کنیم

بیایید بر اساس مطالب گفته شده، به یک ماتریس تبدیل جامع برای چرخش، بزرگ نمایی یا کوچک نمایی و انتقال دست پیدا کنیم. براساس آنچه پیش از این گفتیم ما سه ماتریس تبدیل برای عملیات های فوق به صورت زیر خواهیم داشته که به صورت ماتریس های ۳×۳ در آمده اند

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{Translation by } (x_0, y_0)$$

$$\mathbf{T} = \begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{Scale by } s_1 \text{ and } s_2$$

$$\mathbf{T} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{Rotate by } \theta$$

با ضرب سه ماتریس فوق و در نهایت استخراج ماتریس تبدیل ۳×۲ نهایی می‌توانیم به ماتریس تبدیلی دست پیدا کنیم که این سه تبدیل دنباله‌ای را در دل خود دارد. در شکل زیر مقدار این ماتریس تبدیل محاسبه شده است

$$H = RST = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_0 & 0 & 0 \\ 0 & s_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & x_1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} s_0\cos\theta & s_1\sin\theta & s_0x_0\cos\theta + s_1x_1\sin\theta \\ -s_0\sin\theta & s_1\cos\theta & s_1x_1\cos\theta - s_0x_0\sin\theta \end{bmatrix}$$

بنابراین به راحتی می‌توانید با تبدیل ماتریس‌های تبدیل خود به فرم مربعی و ضرب آن ها در یک دیگر و در نهایت استخراج ماتریس تبدیل ۳×۲ تبدیلات پیچیده تر را پیاده سازی نمایید.

تبدیل پرسپکتیو

تبدیل پرسپکتیو (Perspective Transformation) به تغییر ظاهر تصویر اشاره دارد که شبیه تغییر زاویه یا موقعیت دوربین نسبت به صحنه است. به عنوان مثال، وقتی از یک صفحه جزوه از زاویه گرفته می‌شود، لبه‌های صفحه به صورت مایل دیده

می‌شوند. تبدیل پرسپکتیو می‌تواند تصویر را طوری اصلاح کند که این لبه‌ها مستقیم و مستطیلی به نظر برسند، گویی عکس از روبه‌رو و بدون زاویه گرفته شده است.

ماتریس تبدیل پرسپکتیو یک ماتریس ۳×۳ است که اجازه می‌دهد چرخش، انتقال و تغییر پرسپکتیو به صورت همزمان روی تصویر اعمال شود. برای اعمال این تبدیل در OpenCV از تابع warpPerspective() استفاده می‌کنیم. پارامترها و عملکرد این تابع شباهت زیادی به warpAffine() دارد، با این تفاوت که ماتریس ورودی آن ۳×۳ است و می‌تواند تغییرات پرسپکتیو را نیز اعمال کند.

برای محاسبه ماتریس تبدیل مناسب، از تابع getPerspectiveTransform() استفاده می‌شود. همانند getAffineTransform()، این تابع دو آرایه نقطه دریافت می‌کند:

- نقاط مبدا: مختصات چهار نقطه در تصویر اصلی
- نقاط مقصد: مختصات متناظر این چهار نقطه در تصویر مقصد

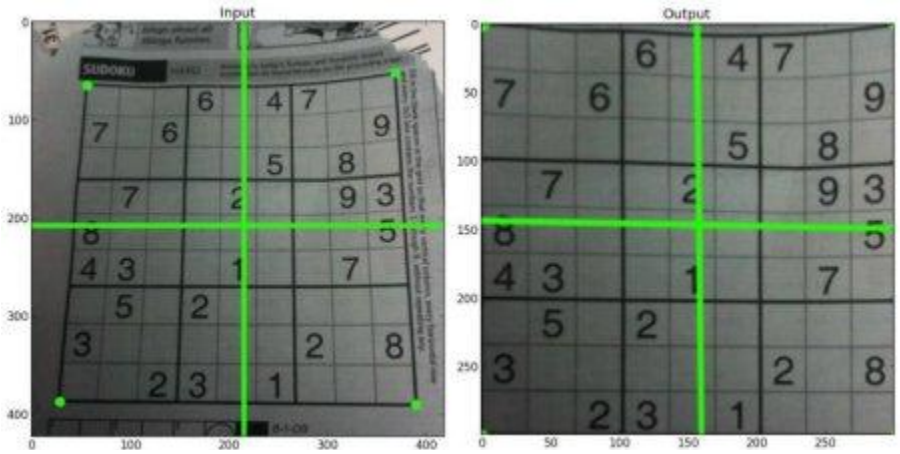
با داشتن این چهار نقطه، تابع می‌تواند ماتریس پرسپکتیو ۳×۳ را محاسبه کند و تصویر را مطابق پرسپکتیو جدید اصلاح نماید. این روش معمولاً در اصلاح تصاویر زاویه‌دار اسکن شده یا عکس‌های گرفته شده با دوربین از زاویه نامناسب استفاده می‌شود.

warpPerspective(image , M , size)

getPerspectiveTransform(pts۱, pts۲)

مثال :

```
import cv2
import numpy as np

img=cv2.imread ("img.jpg")
cv2.imshow ("orginal image",img)
rows,cols,_ = img.shape

pts1 = np.float32([[56,65],[368,52],[28,387],[389,390]])
pts2 = np.float32([[0,0],[300,0],[0,300],[300,300]])

M = cv2.getPerspectiveTransform(pts1,pts2)
prstc = cv2.warpPerspective(img,M,(300,300))

cv2.imshow('Perspective img',prstc)
```



تغییر مقیاس

Scale تصویر : یکی از تبدیلات هندسی که به آن اشاره شد بزرگ نمایی یا کوچک نمایی تصویر است. از آن جا که این تبدیل هندسی بسیار کاربردی است در کتابخانه opencv یک تابع جداگانه برای آن پیشبینی شده . تابع مذکور ()resize می‌باشد.

Resize (img , dsize [, dst [, fx [, fy [, interpolation]]]])

Img : تصویر ورودی

Dsize : ابعاد تصویر جدید بر اساس تعداد ستون و ردیف که به صورت یک دوتایی به تابع داده می‌شود.

Fx,fy : ضریب افزایشی-کاهشی طول تصویر جدید است که یک عدد اعشاری بوده و میتواند کوچکتر یا بزرگتر از ۱ باشد.

Interpolation : برای تغییر ابعاد تصویر، نیازمند درون یابی برای بدست آوردن مقدارِ پیکسل های جدید است . این پارمتر مشخص کننده روش درون یابی است که یکی از مقادیر زیر می‌تواند باشد

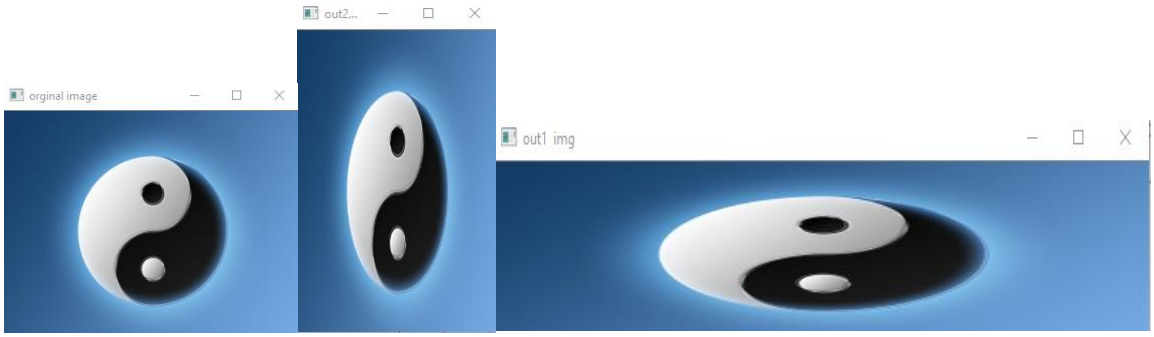
۱- INTER_LINEAR : درون یابی برای مواقع زوم کردن یا کوچک کردن تصویر
۲- INTER_CUBIC : درون یابی برای زوم کردن
۳- INTER_AREA : دورن یابی برای کوچک کردن تصویر

از این تابع میتوان به دو صورت استفاده کرد. روش اول تعیین ابعاد خروجی به صورت مستقیم است. برای منظور ابعاد خروجی مطلوب را به عنوان پارامتر Dsize به توابع می‌دهیم. روش دیگر استفاده از ضرایب کاهشی fx و fy است که در این روش مقدار پارامتر Dsize را برابر None قرار داده و نسبت تبدیل مورد نظر برای محور x و y را به صورت جداگانه به پارامترهای fy و fx می‌دهیم. در مثال زیر هر دوی این روش ها نشان داده شده است.

Cv۲.resize(img, (۲*h, ۲/w)) = cv۲.resize(img, None, fx=۲, fy = ۲)

مثال :

```
import cv2

img=cv2.imread ("img.jpg")

out1=cv2.resize(img ,None, fx=2, fy=0.5 , interpolation=cv2.INTER_CUBIC)
out2=cv2.resize(img ,(200,300), interpolation=cv2.INTER_CUBIC)

cv2.imshow ("out1  img" , out1)
cv2.imshow ("out2  img" , out2)
cv2.imshow ("orginal image",img)
```



اعمال threshold روی تصویر

ترشولد (Thresholding) یکی از پایه‌ای‌ترین و پرکاربردترین عملیات در پردازش تصویر است که معمولاً بر روی تصاویر خاکستری اعمال می‌شود. هدف اصلی آن، تبدیل تصویر خاکستری به تصویر باینری است، یعنی تصویری که هر پیکسل فقط دو مقدار ممکن دارد: روشن یا تاریک (معمولاً سفید یا مشکی).

مفهوم اصلی ترشولد به این صورت است که یک مقدار آستانه (threshold value) برای تصویر مشخص می‌کنیم. سپس بسته به نوع ترشولد:

- پیکسل‌هایی که مقدار روشنایی آن‌ها بیشتر از آستانه است، به یک مقدار ثابت (مثلاً سفید) تبدیل می‌شوند.
- پیکسل‌هایی که کمتر از آستانه هستند، به مقدار دیگر (مثلاً مشکی) تبدیل می‌شوند.

چرا ترشولد لازم است؟

ترشولد باعث ساده شدن تصویر و کاهش پیچیدگی آن می‌شود، به طوری که ویژگی‌های مهم تصویر (مانند اجسام یا اعداد) از پس‌زمینه متمایز می‌شوند. این مرحله برای بسیاری از پردازش‌های بعدی، مانند موارد زیر، ضروری است:

- تشخیص اجسام یا کانتورها (مفهوم کانتور را در آینده بررسی خواهیم کرد)

- شناسایی متن یا اعداد روی پلاک‌ها
- تفکیک اشیاء از پس‌زمینه

مثال کاربردی: فرض کنید می‌خواهیم اعداد روی یک پلاک خودرو را استخراج کنیم. در اینجا می‌توانیم با اعمال ترشولد مناسب: بخش‌های روشن پلاک (پس‌زمینه) را به رنگ مشکی بخش‌های تیره (اعداد) را به رنگ سفید در نتیجه، اعداد پلاک به صورت واضح از پس‌زمینه جدا می‌شوند و آماده انجام پردازش‌های بعدی مانند خواندن اعداد توسط الگوریتم‌های OCR می‌شوند.

در OpenCV، این عمل با تابع threshold() انجام می‌شود و پارامترهای آن شامل تصویر ورودی، مقدار آستانه، مقدار جایگزین برای پیکسل‌ها و نوع ترشولد است..

```
import cv2
import numpy as np

img = cv2.imread('pela.jpg',0 )
roi = img[ 220:350, 100:500]
_,thresh = cv2.threshold( roi, 100, 255, cv2.THRESH_BINARY_INV )

cv2.imshow('img', img )
cv2.imshow('thresh', thresh )
cv2.waitKey(0)
```

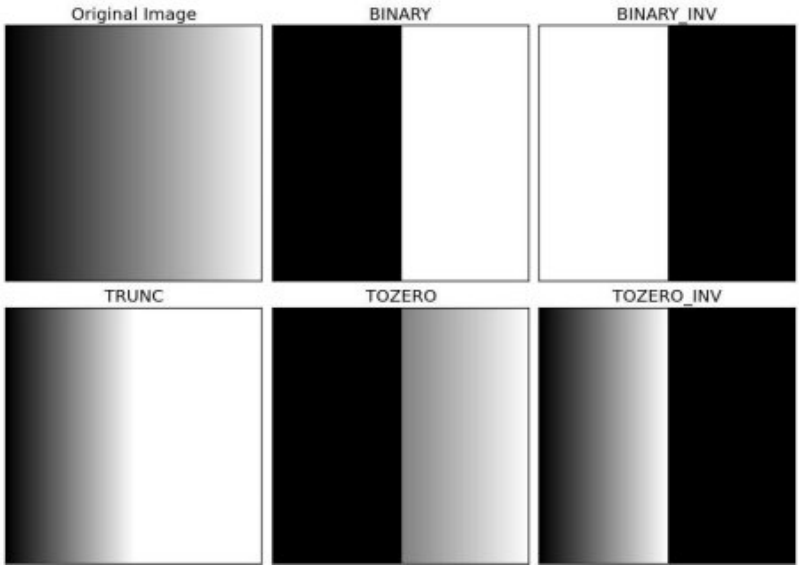
در ادامه به بررسی کامل تابع threshold می‌پردازیم.

Threshold (image , thresh , maxVal , type) ->retVal , dst

- Image : تصویر ورودی که عموما در فضای رنگی Gray است.
- Thresh : مشخص کردن حد آستانه (مقداریست که پیکسل های بالاتر و پایین تر آن از هم جدا میشوند)
- maxVal : حداکثر مقدار عددی که میتوان به آن رسید. مثلا در یک تصویر ۸ بیتی برای رسیدن به حداکثر مقدار مجاز، این مقدار ۲۵۵ خواهد بود.
- Type : این پارامتر مشخص میکند که threshold چه نوعی باشد که یکی از مقادیر زیر میتواند باشد
  - THRESH_BINARY : مقادیر بیشتر از حد آستانه برابر MaxVal و مقادیر کمتر صفر می‌شوند
  - THRESH_BINARY_INV : مقادیر بیشتر از حد آستانه برابر صفر و مقادیر کمتر برابر maxVal می‌شوند
  - THRESH_TRUNC : مقادیر بیشتر از حد آستانه برابر maxVal و مقادیر کمتر بدون تغیر باقی می‌مانند
  - THRESH_TOZERO : مقادیر کمتر از حد آستانه برابر صفر و مقادیر بیشتر بدون تغییر باقی می‌مانند
  - THRESH_TOZERO_INV : مقادیر بیشتر از حد آستانه برابر صفر و مقادیر کمتر بدون تغییر باقی می‌مانند

مثال :

```
ret,thresh1_img = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
```



Adaptiv Threshold

در تابع threshold که تا کنون بررسی کردیم، ما یک حد آستانه عمومی برای کل تصویر در نظر می‌گیریم اما در مواردی، این حد آستانه عمومی، کارامد نیست به‌خصوص در تصاویری که روشنایی در بخش‌های مختلف آن متفاوت است. در این تصاویر راهکار استفاده از حد آستانه محلی است، برای انجام این کار یک پنجره به صورت کشویی بر روی هر بخش تصویر قرار می‌دهیم. سپس مقدار آستانه برای هر پنجره برابر خواهد بود با میانگین مقدار پیکسل ها یا میانگین گاوسی آن ها به این صورت برای هر بخش از تصویر یک حد آستانه مناسب انتخاب می‌شود. در کتابخانه opencv تابع adaptiveThreshold() برای همین منظور ایجاد شده است.

adaptiveThreshold(imag , maxval , adaptiveMethod , thresholdType , blockSize , C)

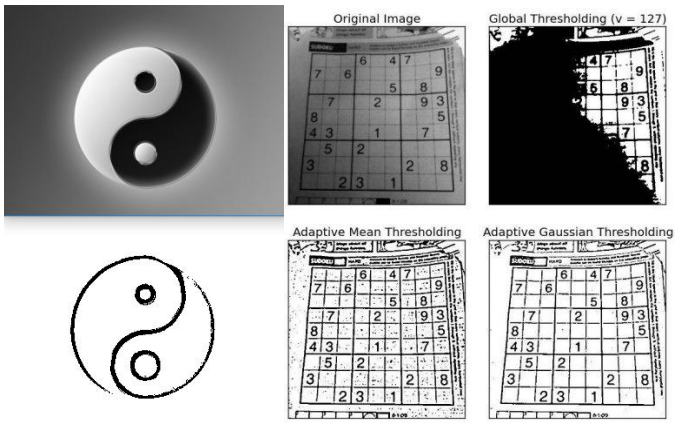
- image : تصویر ورودی
- Maxval : حداکثر مقداری که می‌توان به خروجی اختصاص داد
- adaptiveMethod : در این پارامتر مشخص می‌کنیم از چه متد و روشی برای ترشولد انطباقی، استفاده میکنیم. برای اینکار دور روش زیر تعریف شده است.
  - ADAPTIVE_THRESH_MEAN_C : محاسبه حد آستانه یا روش میانگین
  - ADAPTIVE_THRESH_GAUSSIAN_C : محاسبه حد آستانه یا روش گاوسی

- thresholdType : نوع ترشولد را مشخص می‌کند که یکی از ۵ نوع threshold است که پیش از این به آن اشاره شده بود.
- blockSize : یک عدد صحیح بوده که مشخص می‌کند تا چه محدوده‌ای از پیکسل ها برای محاسبه حد آستانه استفاده گردد.
- C : یک مقدار ثابت که از میانگین و یا میانگین وزن دار محاسبه شده است.

مثال :

```
img=cv2.imread ("img.jpg",0)
th2=cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,11,10)
cv2.imshow('Perspective ',th2)
```

همانطور که در مثال زیر مشاهده می‌کنید تابع ترشولد عادی خروجی که تصویر جدول بالا سمت راست است خروجی مطلوبی نداشته اما در مقابل تابع adaptiveThreshold به خوبی تصویر را باینری کرده



بدست آوردن حد آستانه با OTSU

هنگام اعمال ترشولد (Threshold) روی یک تصویر با یک حد آستانه ثابت، معمولاً نیاز است تا یک مقدار مناسب برای کل تصویر پیدا کنیم. روش رایج این است که با آزمون و خطا، مقدار آستانه‌ای انتخاب شود که بهترین نتیجه را بدهد.

روش دیگری که دقیق‌تر است، استفاده از هیستوگرام تصویر است (هیستوگرام در فصل‌های بعدی آموزش داده می‌شود). در این روش، ابتدا هیستوگرام تصویر محاسبه می‌شود و سپس مقدار آستانه بین دو قله هیستوگرام انتخاب می‌شود تا پس‌زمینه و اجسام مورد نظر به خوبی از هم جدا شوند.

در OpenCV، برای انجام این کار از همان تابع threshold() استفاده می‌کنیم، با این تفاوت که در پارامتر type علاوه بر نوع ترشولد، پرچم THRESH_OTSU را نیز به تابع می‌دهیم. در این حالت، تابع به صورت خودکار حد آستانه مناسب را برای تصویر محاسبه می‌کند.

تابع threshold() دو خروجی دارد:

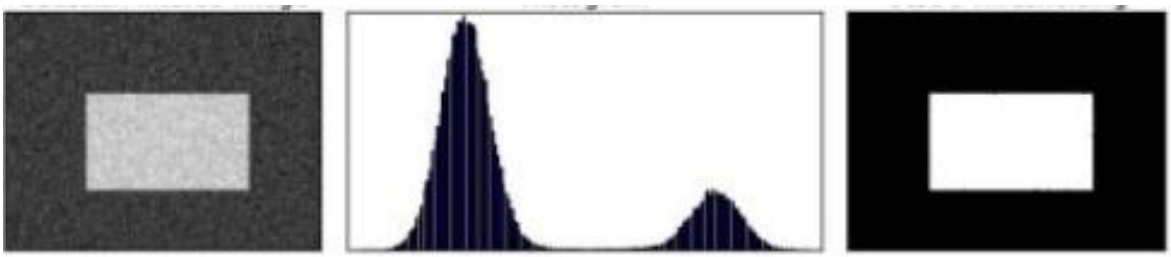
- تصویر ترشولد شده
- retVal که مقدار آستانه واقعی است

وقتی حد آستانه را خودمان مشخص می‌کنیم، retVal همان مقداری است که به تابع داده‌ایم. وقتی از پرچم OTSU استفاده می‌کنیم، retVal همان حد آستانه محاسبه شده توسط تابع است و مقدار ورودی آستانه را می‌توان به راحتی صفر قرار داد.

این روش باعث می‌شود نیاز به انتخاب دستی آستانه کاهش یابد و نتیجه دقیق‌تر شود، به خصوص در تصاویری که نور و روشنایی یکنواخت نیست.

مثال :

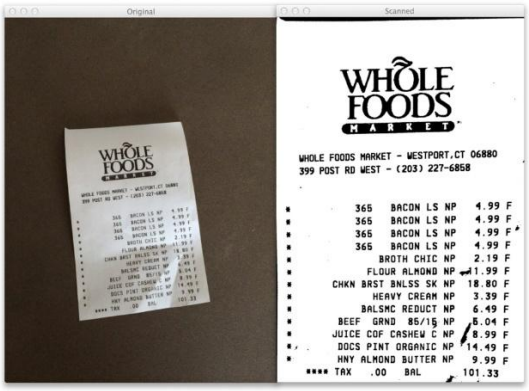
```
>>> img=cv2.imread ("img.jpg",0)
>>> ret2,th2 = cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
>>> ret2
112.0
>>> cv2.imshow('Perspective ',th2)
```



image

تمرین : با استفاده از ترشولد و تبدیل هندسی یک برنامه اسکنر برای اسکن سیاه و سفید بنویسید.

راهنمایی : در ابتدا باید یک ترشولد مناسب بر روی تصویر اعمال کنید تا رنگ بندی تصویر اصلاح شود. سپس با انتخاب مختصات چهار گوشه برگه به عنوان نقاط ورودی و دادن مختصات های پیشفرض براساس نسبت ابعاد برگه، ماتریس تبدیل را بدست آورید و بر روی تصویر اعمال کنید.

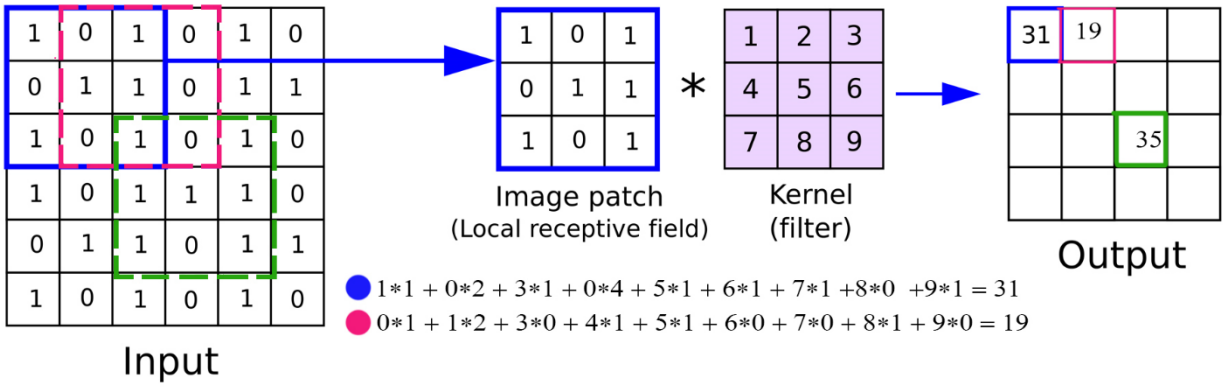


اعمال فیلتر بر روی تصویر

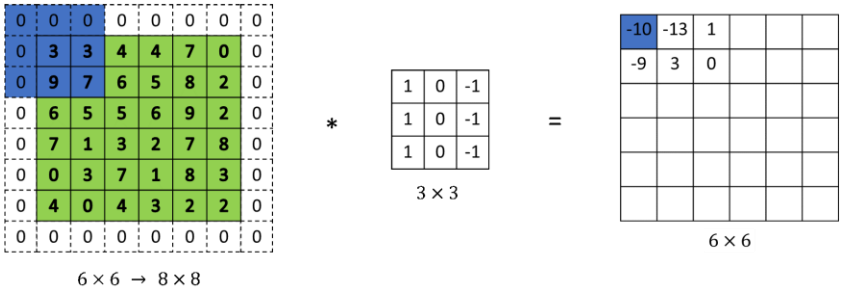
فیلتر چیست

اعمال فیلتر بر روی تصاویر یکی از پایه ای ترین و مهم ترین عملیات‌های پردازش تصویر است و در مواردی نظیر یافته لبه‌ها، حذف نویز، تار کردن تصویر و... کاربرد دارد. هر فیلتر از یک هسته تشکیل شده است که این

هسته آرایه ای از اعداد است که این اعداد وزن نام دارند. همچنین هر هسته دارای یک نقطه لنگرگاه است که عموما این نقطه، دارایه مرکزی در یک هسته می‌باشد. برای مثال در یک هسته با ابعاد ۳*۳ نقطه لنگرگاه به صورت پیشفرض درایه (۲,۲) می‌باشد. برای اعمال یک فیلتر بر روی یک تصویر کافی است است که فیلتر را بر روی تصویر به صورت کشویی حرکت دهیم و در لحظه مقدار هر پیکسل تصویر در مقدار وزن نظیرش در هسته ضرب می‌شود. سپس مجموع این حاصل ضرب ها را حساب می‌کنیم و این حاصل به جای مقدار پیکسلی که لنگرگاه هسته روی آن قرار دارد، قرار می‌گیرد. برای درک بهتر به تصویر زیر نگاه نمایید.



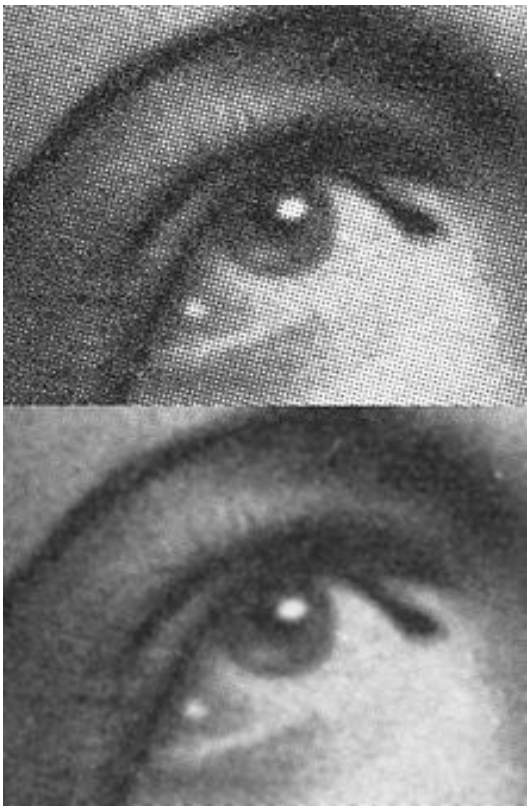
این عملیات کانوولوشن نیز نام دارد. همان گونه که از تصویر فوق مشخص است، ابعاد تصویر خروجی، از تصویر ورودی کوچک تر است. Opencv برای جلوگیری از این اتفاق، بصورت پیشفرض در حاشیه تصویر ورودی از پیکسل‌هایی با مقدار ایجاد می‌کند. به گونه‌ای که ابعاد تصویر ورودی و خروجی برابر باشند. این عملیت padding نام درد



فیلتر BLURING

فیلتر Blur یک فیلتر پایین گذر (LPF) بوده برای نرم کردن تغیرات یک تصویر استفاده میشود. این فیلتر توسط تابع blur() اعمال میشود. این فیلتر با یکدست کردن تصویر، نویز های تصویر را حذف می‌کند. برای درک بهتر تصویر زیر را در نظر بگیرید.

تصویری بالایی ، تصویر اصلی ما می‌باشد که دارای نویز بوده. در حقیقت در این تصویر پیکسل ها یکدست نیستند و دارای تغییرات زیادی می‌باشند یا به اصطلاح این تصویر دارای فرکانس بالایی است. زیرا با حرکت از یک نقطه به نقطه دیگر در آن، تغییرات شدت روشنایی در هر گام زیاد می‌باشد. اما با اعمال یک فیلتر پایین گذر مانند blur می‌توانید تاثیر آن را مشاهده نمایید که یک تصویر یکدست تر و ملایم تر برای ما فراهم می‌سازد. از این رو به این فیلتر یک فیلتر پایین گذر می‌گویند. زیرا تغیرات فرکانس بالا را حذف و یک تصویر فرکانس پایین به ما ارائه می‌دهد و این تصویر را برای انجام سایر عملیات های پردازشی که در این کتاب به آن می‌پردازیم مناسب می‌سازد. یکی از کاربردهای مهم این عملگر پیش پردازش تصویر برای تشخیص لبه است.



Blur(image , kernel size)

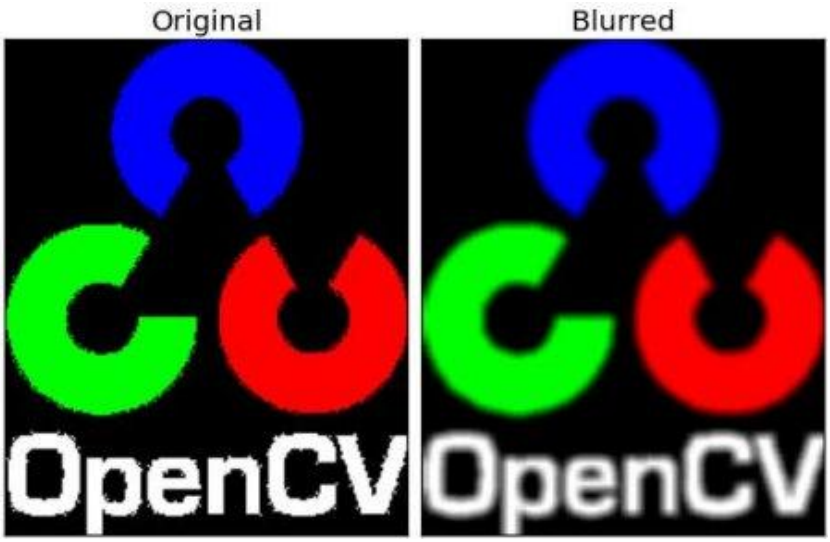
مثال: هسته این فیلتر به صورت زیر است:

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

همانگونه که از هسته این فیلتر پیدا است، در این فیلتر مقدار هر پیکسل که مرکز هسته روی آن قرار دارد با میانگین n پیکسل اطراف خود جایگزین میشود که مقدار n به اندازه هسته بستگی دارد از این رو مقدار هر پیکسل ها به یک مقدار میانگین نزدیک می‌شوند و تصویر تقریبا یکدست می‌شود.

مثال:

```
import cv2
import numpy as np

img = cv2.imread('opencv_logo.png')

blur = cv2.blur(img,(5,5))
```



فیلتر Gaussian

فیلتر گاوسی مانند فیلتر blur یک فیلتر پایین گذر (LPF) بوده و از نظر بصری تصویر را تار می‌کند اما با این تفاوت که بر عکس فیلتر blur که وزن کلیه پیکسل های همسایه یکسان بود و با یک ضریب یکسان میانگین گیری میشد، در این فیلتر ضرایب بر اساس فاصله از مرکز و با عملگر گاوسی محاسبه میشوند. از این رو پیکسل های نزدیک تر به پیکسل مرکزی، وزن و تاثیری بیشتری در میانگین گیری دارند. تابع این فیلتر()GaussianBlur بوده که در آن باید علاوه بر معرفی ابعاد هسته، باید سیگما در جهت x و y را برای محاسبه ضرایب معرفی کرد. اگر فقط سیگما x مقدار دهی شود، سیگما y برابر با آن در نظر گرفته میشود . اگر هردو صفر در نظر گرفته شوند، مقدارشان بر اساس ابعاد هسته، محاسبه میشود. این فیلتر برای رفع نویز های گاوسی بسیار مناسب است.

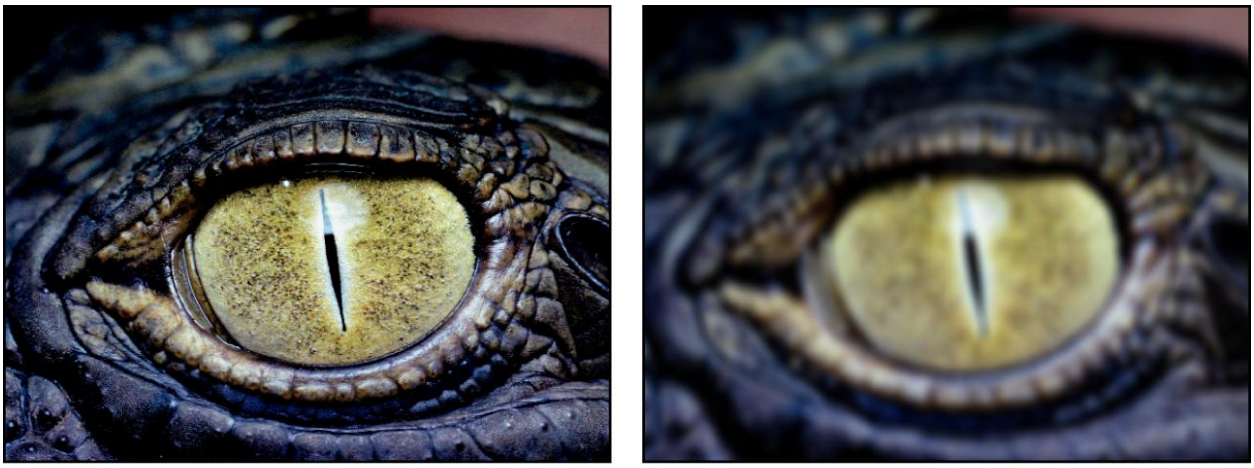
GaussianBlur(image, kernel size , sigma x , sigmay)

مثال هسته یک فیلتر gaussain

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

مثال :

```
blur = cv2.GaussianBlur(img,(5,5),0)
```



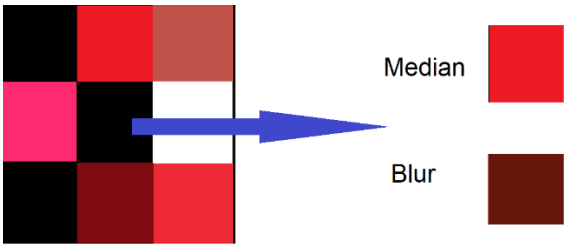
Original image

Gaussian Blur filter applied

فیلتر median

این فیلتر نیز یک فیلتر پایین گذر (LPF) می‌باشد. در این فیلتر median مقدار هر پیکسل با مقدار میانه پیکسل های اطرافش که در همسایگی معینی قرار دارند، جایگزین میشود. برای مثال فرض کنید پیکسل های همسایه یک پیکسل مشخص دارای مقادیر ۱۳۰, ۴۰, ۵۰, ۲۰۰, ۱۰۰ باشند. در این صورت مقدار پیکسل مدنظر برابر مقدار میانه این مقادیر یعنی ۱۳۰ خواهد شد. این فیلتر برای رفع نویز هایی نظیر نویز فلفل- نمکی بسیار مناسب می‌باشد (نویز هایی که پیکسل های سفید و سیاه پراکنده روی تصویر به وجود می‌آورند). فرض کنید چنین تصاویری را بخواهیم با یک فیلتر blur فیلتر نماییم در این صورت خروجی مطلوبی نخواهیم داشت چراکه پیکسل هایی سفید یا مشکی در میانگین گیری تاثیر میگذارند اما با اعمال فیلتر median ما

مقدار میانه پیکسل های همسایه را انتخاب خواهیم کرد که این مقدار قطعا پیکسل های سفید یا مشکی نخواهند بود بلکه یک مقدار پیکسل از همسایگی است که مقدارش در میانه قرار دارد . به این صورت پیکسل های سفید و مشکی تاثیری در مقدار جدید پیکسل ما نخواهند داشت. این موضوع برای تصویر رنگی نیز صادق است چرا که هر تصویر رنگی خود از سه کانال رنگی تشکیل شده است که هر کانال مانند یک تصویر سیاه و سفید است. در تصویر زیر مقدار نتیجه برای پیکسل مرکزی به ازای هر دو فیلتر را مشاهده می‌نمایید. مشخصا نتیجه فسلتر median بسیار مطلوب تر است.

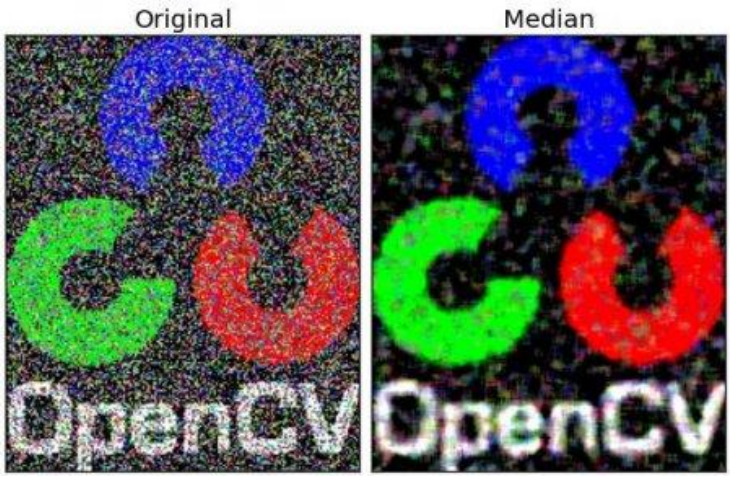


تابع این فیلتر در opencv، medianBlur می‌باشد.

medianBlure(image , kernel size)

مثال:

```
median = cv2.medianBlur(img,5)
```



فیلتر Bilateral

این فیلتر مانند فیلتر گاوسی عمل می‌کند با این تفاوت که با اعمال این فیلتر لبه های تصویر حفظ خواهند شد. در فیلتر گاوسی هنگام میانگین گرفتن بین پیکسل های همسایه، توجه نمی‌شود که آیا این پیکسل های همسایه متعلق به همان محدوده بوده و همان شدت را دارند یا خیر . از این رو پیکسل های مرزی شدتشان کاهش یافته و لبه‌ها محو می‌شوند. اما در فیلتر دوطرفه فوق، هنگام میانگین گیری، همواره برآورد میشود که پیکسل های همسایه که در میانگین گیری گاوسی مورد استفاده قرار میگیرند، متعلق به همان ناحیه باشند که پیکسل مرکزی در آن قرار دارد در غیر این صورت در میانگیری محاسبه نخواهند شد. از این رو با حفظ شدن لبه ها، فیلتر گاوسی روی تصویر اعمال می‌شود. تابع پیاده سازی این فیلتر bilateralFilter() می‌باشد

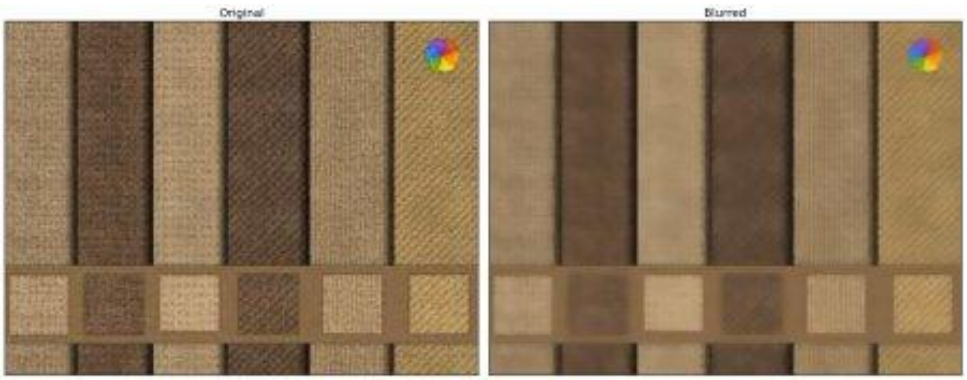
bilateralFilter(img , d , sigma Color , sigma Space)

img : تصویر ورودی

- D : این پارامتر قطر محدوده هر پیکسل که در هنگام فیلترینگ استفاده می‌شود را برحسب پیکسل مشخص می‌کند.
- Sima Color : انحراف معیار در فضای رنگی
- Sigma Space : انحراف معیار در فضا (فاصله)

مثال :

```
blur = cv2.bilateralFilter(img,9,75,75)
```



همانگونه که در شکل مشخص است ، با وجود تار شدن تصویر، لبه ها کاملا حفظ شده اند

فیلتر با هسته دلخواه

در بعضی مواقع نیاز است تا ما یک فیلتر با یک هسته دلخواه روی تصویر اعمال کنیم. برای استفاده از این فیلتر ابتدا لازم است یک هسته تعریف نماییم. برای اینکار ابتدا ماژول numpay را به برنامه import میکنیم. سپس به کمک تابع ones() به صورت زیر یک هسته تعریف مینماییم. که یک هسته با درایه های ۱ میسازد. تابع zeros نیز یک هسته با درایه های صفر می‌سازد.

Ones((rows number , column number) , kernel type)

یا تولید یک با تابع زیر

array(array values , kernel type)

سپس با استفاده از تابع filter۲D() این هسته را روی تصویر اعمال میکنیم.

Filter۲D(image , out type , kernel)

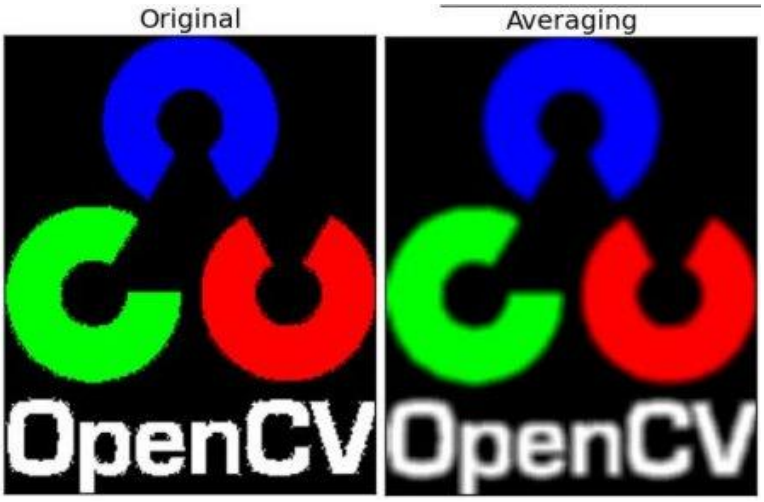
نکته:در این فیلتر مقدار هر پیکسل با مقدار مجموع مقدار پیکسل های همسایه ضرب در وزنشان

نکته : این هسته در واقع از نوع ارایه کتابخانه numpy بوده و میتوان آن را مانند یک لیست مقدار دهی کرد

مثال: در برنامه زیر ما هسته زیر را تولید و روی تصویر اعمال میکنیم

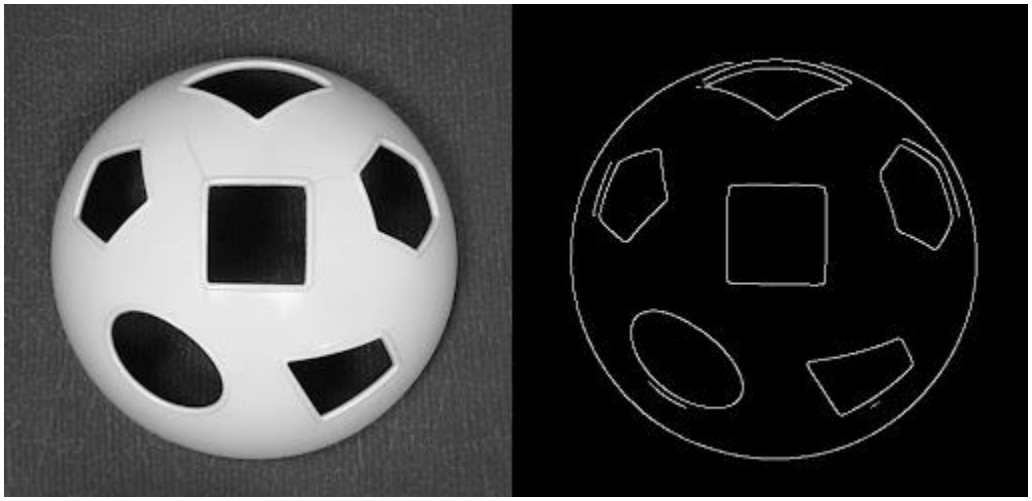
$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

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

img = cv2.imread('opencv_logo.png')

kernel = np.ones((5,5),np.float32)/25
kernel2= np.array( [[0,1,0] , [1,-4,1] , [0,1,0]] , np.int8)

dst = cv2.filter2D(img,-1,kernel)
kernel[3,3]=2
```



تشخیص لبه

لبه‌های تصویر خطوطی در تصویر هستند که نواحی مختلف تصویر را از یکدیگر جدا می‌کنند. تشخیص لبه می‌تواند درک مناسب از اشکال موجود در تصویر برای ما ایجاد نماید. در گام‌های بعدی ما شناسایی کانتورها بر روی لبه‌های یک تصویر می‌توانیم خصوصیات هندسی هر شی در تصویر نظیر مساحت، زاویه، فرم و... را استخراج نماییم الگوریتم های تشخیص لبه بر اساس تضاد رنگ نواحی مختلف در تصویر کار می‌کنند از این رو برای آنکه در تشخیص لبه نویزها و لبه های ناخواسته نداشته باشیم، اعمال یک فیلتر پایین گذر مانند blur پیش از تشخیص لبه می‌تواند گزینه مناسبی باشد.



فیلتر Laplacian

این فیلتر بر خلاف فیلتر های قبلی، یک فیلتر بالا گذر (HPF) بوده و از آن برای تشخیص لبه های یک تصویر میتوان استفاده نمود.. همچنین کنتراست یک تصویر را میتوان با تفریق لاپلاس یک تصویر از خودش، بالا برد . تابع این فیلتر Laplacian() میباشد

Laplacian(imag , out type , out image_dst , kernel size)

مثال: هسته یک فیلتر لاپلاسین به صورت زیر میباشد:

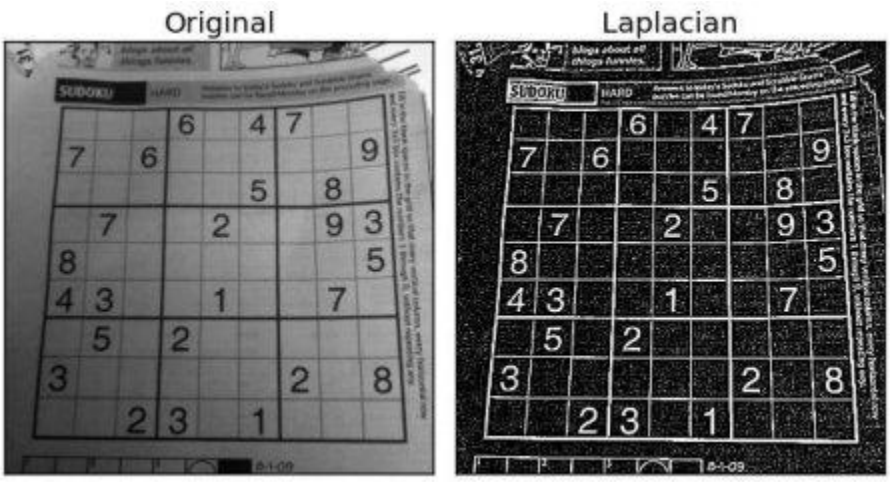
$$\text{kernel} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

با توجه به هسته فوق، مشخص است در این فیلتر مقدار هر پیکسل از اختلاف چهار برابر هر پیکسل از چهار همسایه اطراف خود بدست می‌آید که در لبه ها مقداری غیر صفر و در سایر بخش ها تقریبا صفر میباشد.

نکته : این فیلتر عموما برای تصاویری خاکستری کاربرد دارد

مثال:

```
img = cv2.imread('dave.jpg',0)

laplacian = cv2.Laplacian(img,cv2.CV_8U)
```



عملگر Sobel

عملگر سوبل (Sobel Operator) یک فیلتر بالاگذر (High-pass Filter) است که برای تشخیص لبه‌ها در تصاویر استفاده می‌شود. این عملگر نسبت به عملگر لاپلاس (Laplacian) دقت بیشتری در تشخیص لبه‌ها دارد و حساسیت آن به نویز کمتر است.

سوبل دارای دو هسته (Kernel) مجزا است:

- یکی برای تشخیص لبه‌ها در جهت x
- دیگری برای تشخیص لبه‌ها در جهت y

با استفاده از تابع add() می‌توان نتایج این دو هسته را با هم ترکیب کرد تا لبه‌های کامل تصویر به دست آید.

به طور کلی، عملگر سوبل برای استخراج جزئیات و مرزهای اشیاء در تصویر بسیار مفید است و نسبت به لاپلاس حساسیت کمتری به نویز دارد و بنابراین اغلب در کاربردهای واقعی ترجیح داده می‌شود.

Sobel(image , out type , x_bool , y-bool , kernel siza)

مثال: هسته یک عملگر سوبل به صورت زیر است:

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

همان طول که از هسته های این عملگر مشخص است . اگر این هسته بر روی یک ناحیه یک دست مثلا پیشانی یک فرد در عکس قرار گیرد، مقدار حاصل عددی کوچک خواهد شد زیرا مقدار پیکسل ها تقریبا یکسان بوده و با ضرب آن ها در وزن های بالا که مجموعشان برابر صفر است، حاصل و جمعشان با یک دیگر عددی کوچکی خواعد بئد اما اگر در لبه قرار بگیرد مقدار این فیلتر محالف صفر می‌شود

-1	0	+1
-2	0	+2
-1	0	+1

Gx

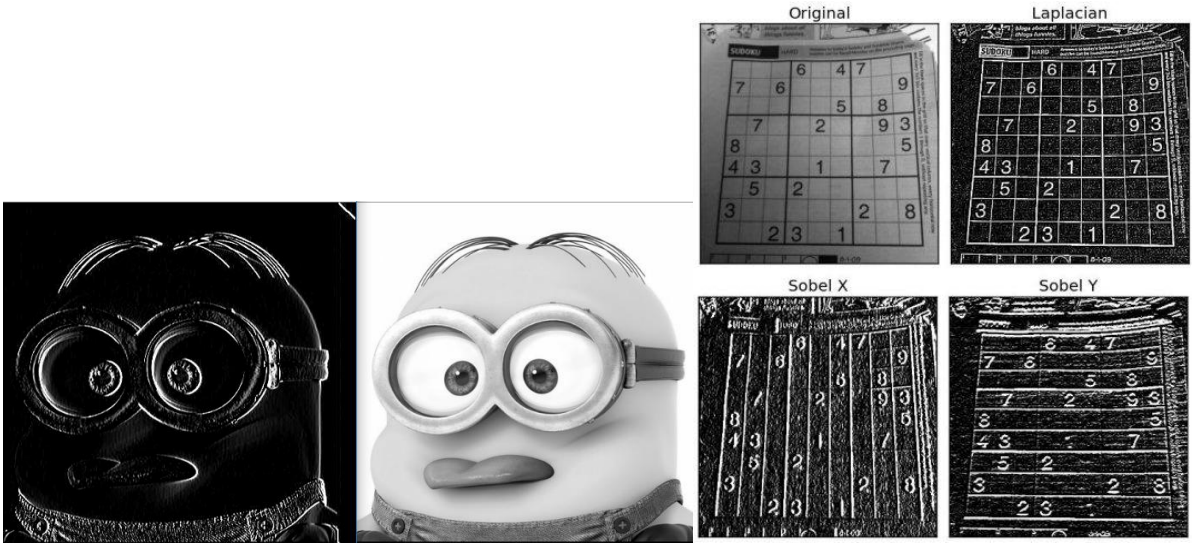
برای مثال در تصویر بالا که فیلتر بر روی یک لبه قرار گرفته است، مقدار حاصل بصورت زیر خواهد بود (رنگ خاکستری عکس برابر ۱۰۰ می‌باشد)

(-۱×۲۵۵ - ۲×۲۵۵ — ۱×۲۵۵) + (۱×۱۰۰ + ۲×۱۰۰ + ۱×۱۰۰) = -۶۲۰

اما اگر این فیلتر بر روی یک محیط کاملا سفید یا کاملا خاکستری بود حاصل برابر ۰ می‌شد. یا حتی اگر تصویر یک دست نباشد مقدار عددی کوچکی خواهد شد

```
img = cv2.imread('dave.jpg',0)

sobelx = cv2.Sobel(img,cv2.CV_8U,1,0)
sobely = cv2.Sobel(img,cv2.CV_8U,0,1)
sobxy=cv2.add(sobelx , sobely)
```

مثال:در زیر تصویری از عملگر سوبل و مقایسه آن با لاپلاسین را مشاهده می‌نمایید



فیلتر Scharr

نوعی فیلتر یا عملگر لبه یاب دیگری است که هرگاه نیازمند تخمین دقیقتر در جهت گرادیان باشد استفاده می‌شود. این عملگر مانند عملگر سوبل، لبه هارا در دو جهت x و y بدست می‌آورد. که میتوان با تابع add() آنها را باهم جمع نمود. تابع این عملگر Scharr() می‌باشد.

Scharr(imag , output type , dx , dy)

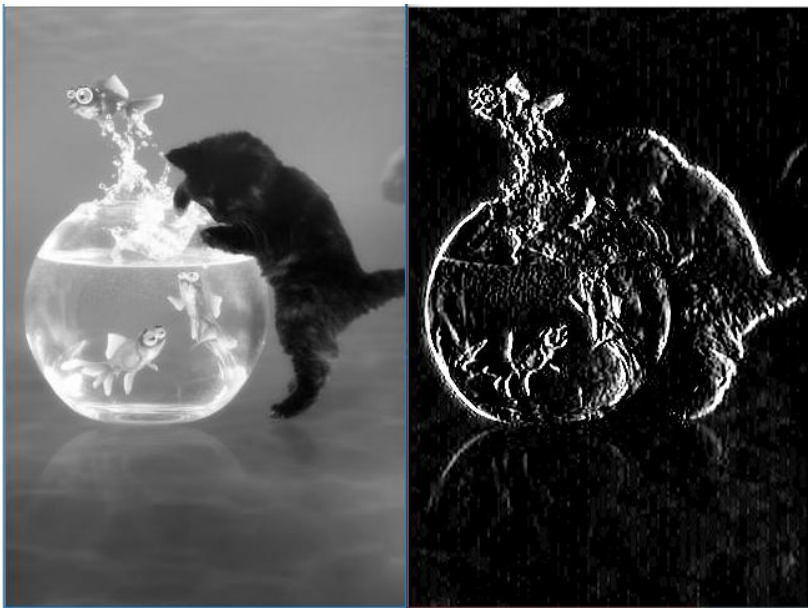
هسته‌های این فیلتر به صورت زیر می‌باشند

$$\begin{bmatrix} +3 & 0 & -3 \\ +10 & 0 & -10 \\ +3 & 0 & -3 \end{bmatrix} \qquad \begin{bmatrix} +3 & +10 & +3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix}$$

Scharr-x **Scharr-y**

مثال :

```
img = cv2.imread('dave.jpg',0)

scharrx = cv2.Scharr(img,cv2.CV_8U,1,0)
scharry = cv2.Scharr(img,cv2.CV_8U,0,1)
scharrxy=cv2.add(scharrx , scharry)
```



تذکر مهم : هنگام استفاده از عملگرهای Sobel و Scharr، ما در واقع تغییرات شدت روشنایی پیکسل‌ها را محاسبه می‌کنیم. این تغییرات می‌توانند مثبت یا منفی باشند:

- مثبت: تغییر روشنایی از تیره به روشن (سیاه به سفید)
- منفی: تغییر روشنایی از روشن به تیره (سفید به سیاه)

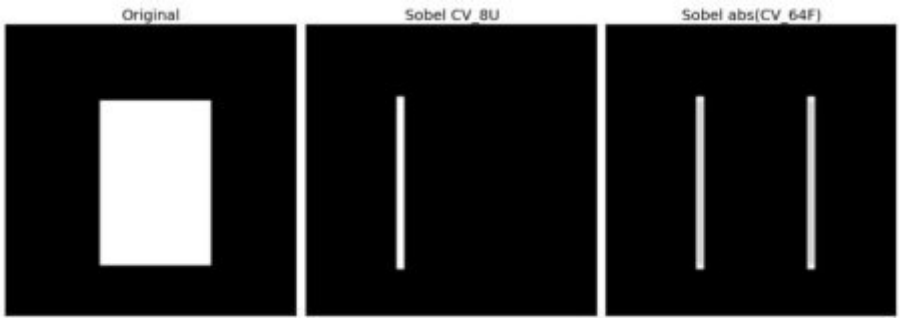
اگر تصویر خروجی را با نوع داده CV_8U ذخیره کنیم، مقادیر منفی به صفر تبدیل می‌شوند. نتیجه این است که لبه‌هایی که در جهت منفی قرار دارند، از تصویر حذف می‌شوند و بخشی از جزئیات تصویر از دست می‌رود.

راهکار استاندارد برای حفظ تمام لبه‌ها این است که ابتدا از نوع داده‌ای با قابلیت ذخیره مقادیر منفی مانند CV_۱۶S یا CV_۶۴F استفاده کنیم. سپس با اعمال عملگر قدر مطلق (absolute)، تمام مقادیر منفی و مثبت به مقادیر مثبت تبدیل می‌شوند و در نهایت می‌توان تصویر را به CV_۸U برگرداند تا قابل نمایش و پردازش باشد.

به عبارت ساده‌تر، این روش باعث می‌شود هیچ لبه‌ای از تصویر به دلیل منفی بودن از بین نرود و تصویر خروجی کامل‌تر و دقیق‌تر باشد.

مثال:

```
import cv2
import numpy as np

img = cv2.imread('image.jpg',0)

img=cv2.bilateralFilter(img , 10,50,50)
sobelx8u = cv2.Sobel(img,cv2.CV_8U,1,0,ksize=5)

#Output dtype = cv2.CV_64F. Then take its absolute and convert to cv2.CV_8U
sobelx16s = cv2.Sobel(img,cv2.CV_16S,1 ,0,ksize=5)
abs_sobel16s = np.absolute(sobelx16s)

sobel_8u = np.uint8(abs_sobel16s)
#improve image
_,sobel_8u=cv2.threshold (sobel_8u,120,255,cv2.THRESH_BINARY )
cv2.imshow ("out" , sobel_8u)
```



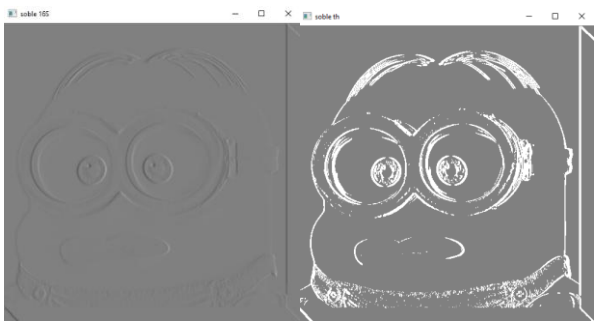
همانگونه که مشخص است در تصویری که نوع خروجی CV-۸U انتخاب شده، با حرکت از چپ به راست، لبه اول که از سیاه به سفید است، تشخیص داده شده اما لبه دوم که از سفید به سیاه است، به خاطر مقدار منفی، حذف شده است.

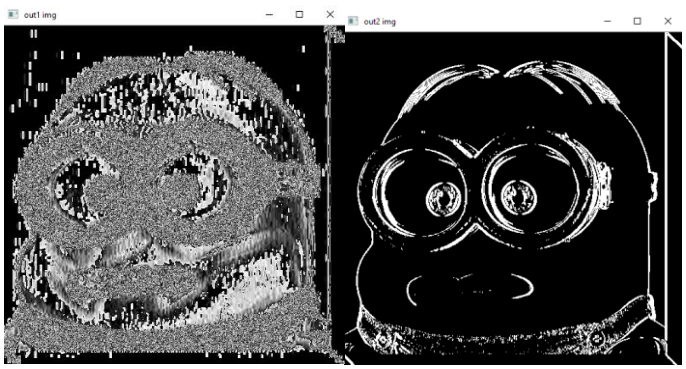
البته در موارد تبدیل تصویر به CV_8U باعث خراب شدن تصویر لبه‌ها می‌شود و بهتر است ابتدا یک ترشولد روی تصویر نوع CV_16S اعمال کرد. و سپس آن را به نوع ۸ بیتی تبدیل کرد. یا حتی میتوان از همان تصویر با نوع CV_16S استفاده کرد برای درک بهتر به مثال زیر توجه فرمایید.

```
import cv2
import numpy as np

img = cv2.imread('min.jpg',0)

sobelx16s = cv2.Sobel(img,cv2.CV_16S,1,0,ksize=5)

abs_sobelx16s = np.absolute(sobelx16s)
abs_sobelx6u=np.int16 (abs_sobelx16s)
_,abs_sobelxth=cv2.threshold (abs_sobelx6u,800,65535,cv2.THRESH_BINARY )


sobel1_8u = np.uint8(sobelx16s)
sobel2_8u = np.uint8(abs_sobelxth)

cv2.imshow (" out1 img" , sobel1_8u )
cv2.imshow (" out2 img" , sobel2_8u )
cv2.imshow ("soble 16S", sobelx16s)
cv2.imshow ("soble th", abs_sobelxth)
```

مثال :

همانطور که مشخص است در تصاویر زیر خروجی اول که مستقیم با تبدیل ۱۶s به ۸u ساخته شده مناسب نیست. اما تصویر که از تبدیل تصویر ترشولد یافته ۱۶s به ۸u به وجود آمده قابل قبول است. همچنین خود تصویر ۱۶s نیز برای تشخیص لبه ها مناسب است





عملگر Canny

این عملگر برای تشخیص لبه‌ها مورد استفاده قرار می‌گیرد و محبوبیت بالایی دارد. این عملگر در مراحل زیر

1- حذف نویز با اعمال یک فیلتر گاوسی با یک هسته 5×5

2- یافتن لبه های تصویر فیلتر شده در دو جهت YوX با عملگر سوبل

3- حذف پیکسل‌هایی که احتمالا جزو لبه‌ها نیستند.

4- اعمال آستانه‌های بالا و پایین: این بخش به این صورت است که دو آستانه بالا و پایین روی تصویر لبه‌ها اعمال می‌شود . بخش‌هایی که زیر آستانه پایین قرار دارند مطمئنا جزو لبه‌ها نیستند و بخش‌هایی که بالاتر از آستانه بالا قرار دارند مطمئنا جزو لبه‌ها هستند. اما بخش‌هایی که بین سطح آستانه بالا و پایین قرار دارند بر اساس ارتباطشان با سایر لبه‌های قطعی یا جزو لبه‌ها انتخاب می‌شوند یا حذف می‌شوند.

این عملگر توسط تابع Canny() اعمال می‌شود.

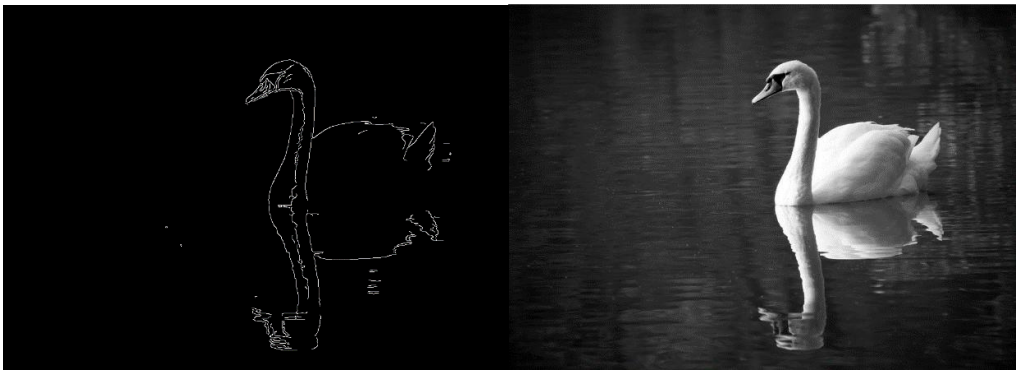
Canny(img , min threshold , max threshold)

Img : تصویر ورودی

Min thrshold : سطح آستانه پایین

Max threshold : سطح آستانه بالا

```
import cv2
import numpy as np
img=cv2.imread("imag.jpg",0)
blurImg=cv2.blur (img , (3,3))
imgCanny=cv2.Canny(blurImg,100,150)
cv2.imshow("canny image", imgCanny)
cv2.imshow(" image", img)
```

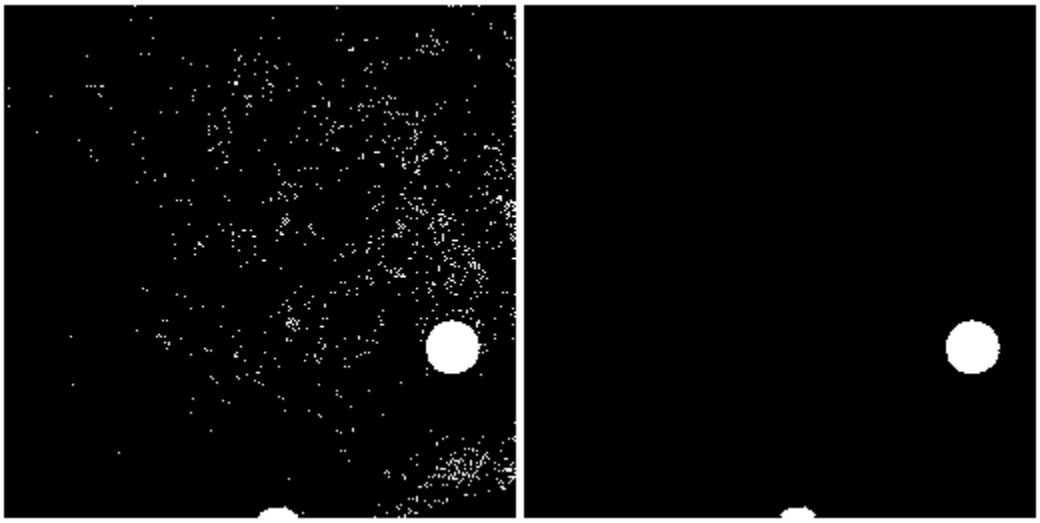


ریخت شناسی یا Morphology

تصاویر باینری

تصاویری هستند که از تنها از پیکسل های سیاه و سفید تشکیل شده اند. این تصاویر اطلاعاتی خوبی از شکل object مورد نظر نمایش می‌دهند. این تصاویر معمولا با توابع threshold و تشخیص لبه بدست می‌آیند. این تصاویر همچنین به عنوان ماسک کاربرد زیادی دارند. برای مثال اگر ما بخواهیم پردازش خود را به بخش خاصی از یک تصویر محدود نماییم میتوانیم یک ماسک را به گونه‌ای ایجاد کنیم که مقدار آن ناحیه مورد نظر ما سفید و سایر نواحی سیاه باشند. گاهی برای رسیدن به تصاویر باینری، تصویر ممکن است دچار نویز شود یا بخشی از اطلاعات از بین روند. در این مواقع ما به کمک ریخت شناسی تصویر را اطلاح مینماییم و تصاویر مطلوب خود را بدست می‌آوریم.

برای مثال در تصویر زیر هدف ما استخراج ماسک یک سکه موجود در تصویر است اما تصویر باینتری ما دارای نویز بسیار زیادی است (نقاط سفید منفرد موجود در عکس سمت چپ) . ما می‌توانیم با استفاده از عملیات ریخت شناسی این تصویر را اصلاح و به تصیر بدون نویز سمت راست دست یابیم

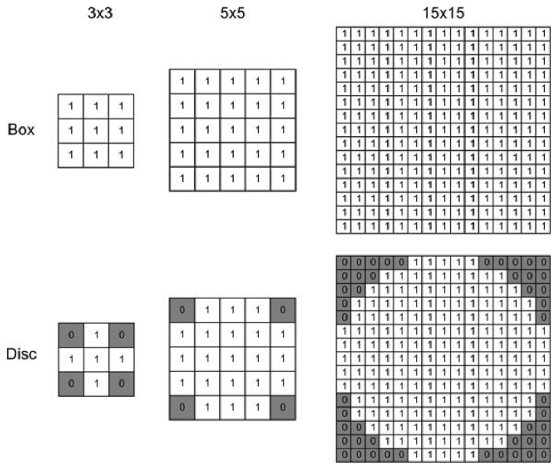


ریخت شناسی

ریخت شناسی مجموعه‌ای از الگوریتم‌هاست که برای اصلاح و بهبود تصویر که عموما برای تصاویر باینری مورد استفاده قرار می‌گیرد. ریخت شناسی برپایه دو عمل Erosion (فرسایش) و Dilation (انحطاط) است و تعمیم این دو عمل، سایر الگوریتم ها و عملیات‌های ریخت‌شناسی نظیر opening , closing و... را تولید می‌کنند. ریخت شناسی شامل دو پارامتر زیر است:

۱-تصویر: یک تصویر باینری که عملیات ریخشناسی روی آن اعمال می‌شود

۲-هسته: یک هسته در ریخت شناسی یک آرایه دو بعدی است که از درایه های صفر و یک تشکیل شده است و یک نقطه لنگر دارد. نقطه لنگر درایه‌ای از این ماتریس است که پیکسلی که تحت پوشش آن قرار میگیرد را تحت تاثیر قرار می‌دهد. درایه لنگر معمولا در مرکز ماتریس قرار دارد



عملیات های ریخت شناسی

Erosion

این الگوریتم موجب فرسایش تصویر پیش زمینه (هنگامی که پیش زمینه سفید و پس زمینه سیاه باشد) می‌شود. روش این الگوریتم به این صورت است که اگر تمام پیکسل هایی که تحت پوشش درایه‌های یک قرار دارند به رنگ سفید باشند، پیکسلی که تحت پوشش درایه لنگر است، سفید و در غیر این صورت سیاه می‌شود. از این رو با اعمال ان، پیکسل های مرکزی سفید می‌مانند و پیکسل های که در نزدیکی لبه‌ها قرار دارند سیاه می‌شوند. در نتیجه لبه‌های تصویر پس‌زمینه فرسایش یافته و پس‌زمینه کوچک‌تر می‌شود. این عملگر برای حذف نویز از تصویر بسیار کاربرد دارد و نقاط مفرد در تصویر را حذف می‌مند

تابع این عملگر erdode() می‌باشد و به صورت زیر است:

erode(image , Kernel , anchor , iteration)

- image : تصویر ورودی
- Kernal : ماتریس هسته
- Anchor : شماره درایه‌ی نقطه لنگری

Iteration : تعداد تکرار الگوریتم را مشخص میکند. مشخصا هرچه تعداد تکرار بیشتر باشد فرسایش نیز بیشتر خواهد بود.

مثال:

```
import cv2
import numpy as np

#step_1 convert image
img=cv2.imread("imag.JPG",0)

kernel = np.ones((5,5),np.uint8)
erosin=cv2.erode(img , kernel , iterations=1)  # nitration write for inter face
cv2.imshow("erod" , erosin)
```

Dilation

این عملگر بر عکس عملگر erode بوده و نحوه کار این عملگر به اینصورت است که اگر حداقل یک پیکسل تحت پوشش درایه های یک، سفید باشد، پیکسل تحت پوشش درایه لنگر سفید می‌شود و اگر همه پیکسل های تحت پوشش درایه های یک، سیاه باشند، پیکسل تحت پوشش درایه لنگری سیاه می‌شود. از این رو پیکسل های سیاه پس زمینه که در مجاورت پیکسل های سفید هستند، سفید می‌شوند و سایر پیکسل ها بدون تغییر باقی می‌مانند. در نتیجه تصویر پیش زمینه گسترش می‌یابد. این فیلتر در ترکیب با erode عملگر های پرکاربرد opening و closing را ایجاد می‌کنند.

تابع این عملگر ()dilate است که پارامتر های آن مانند تابع erode است.

```
erosin=cv2.dilate(img , kernel , iteration=1)
```

عملگر opening :

این عملگر از اعمالی متوالی dilate و erode تشکیل شده است. در ابتدا با اعمال یک erode تصویر دچار فرسایش شده و نویز های تصویر (پیکسل ها و لکه های سفید) حذف می‌شوند. سپس با اعمال یک dilate تصویر اصلی که لبه هایش فرسایش یافته تا حد زیادی ترمیم می‌شود در نتیجه با اعمال opening تنها لکه های سفید از تصویر حذف می‌شوند. این عملگر با تابع morphologyEX به این صورت که در پارامتر دوم نوع عملگر که در اینجا opening است،پیاده سازی می‌شود.

morphologyEX(img , opration , kernel)

operation : نوع عملگر که در اینجا MORPH_OPEN می‌باشد.

مثال :

```
Op=cv2.morphplogyEX( img ,cv2.MORPH_OPRN ,kernel)
```

عملگر Closing :

این عملگر مانند عملگر opening است با این تفاوت که ابتدا عملگر dilateو سپس erode روی تصویر اعمال می‌شود. این عملگر برای تصاویری که بخشی هایی از اطلاعات آن حذف شده‌اند بسیار کاربردی است. در ابتدا با اعمال dilate روی تصویر، ضمن گسترش پیش زمینه، نواحی سیاه درون پیش زمینه را که سیاه هستند، سفید می‌کند تا تصویر پیش زمینه کاملا پر شود و بخش های حذف شده جبران گردند. سپس با اعمال عملگرerode تصویر پیش زمینه که گسترش یافته دچار فرسایش می‌شود تا به حالت نرمال خود باز گردد. این عملگر نیز با تابع morphologyEX() پیاده سازی می‌شود

morphologyEX(img , opration , kernel)

```
Cl=cv2.morphplogyEX( img ,cv2.MORPH_CLOSE ,kernel)
```

operation : نوع عملگر که در اینجا MORPH_CLOSE می‌باشد.

در ادامه تنها به معرفی برخی دیگر از عملگرهای ریخت شناسی می‌پردازیم و توضیح درباره آن‌ها خارج از محدوده این کتاب است

عملگر Morphological Gradiant

ین عملگر تصویر را به حالت outline در می‌آورد. این عملگر درحقیقت از اختلاف dilate و erode تصویر بدست می‌اید این عملگر نیز با تابع ()morphologyEX پیاده سازی می‌شود.

morphologyEX(img , opration , kernel)

operation : نوع عملگر که در اینجا MORPH_GRADIENT می‌باشد.

```
Gr=cv2.morphplogyEX( img ,cv2.MORPH_GRADIENT ,kernel)
```

عملگر Top Hat

این عملگر نتیجه‌ای به صورت زیر دارد و توسط تابع morphologyEX پیاده سازی می‌شود.

morphologyEX(img , opration , kernel)

operation : نوع عملگر که در اینجا MORPH_TOPHAT می‌باشد.

```
th=cv2.morphplogyEX( img ,cv2.MORPH_TOPHAT ,kernel)
```

عملگر Black Hat

این عملگر نتیجه‌ای به صورت زیر دارد و توسط تابع morphologyEX پیاده سازی می‌شود.

morphologyEX(img , opration , kernel)

operation : نوع عملگر که در اینجا MORPH_BLACKHAT می‌باشد.

```
th=cv2.morphplogyEX( img ,cv2.MORPH_BLACKHAT ,kernel)
```

ساخت یک Kernel متفاوت

گاهی لازم است ما یک kernel با شکلی خاص، مثلا دایره ایجاد کنیم یک روش ساخت یک آرایه و مقدار دهی درایه به درایه آن است اما این روش کمی زمان بر است. اما روش دیگر که ساده تر است، بهره بردن از تابع getStructuringElement() است که تنها دو پارامتر برای ساخت kernel دریافت می‌کند.

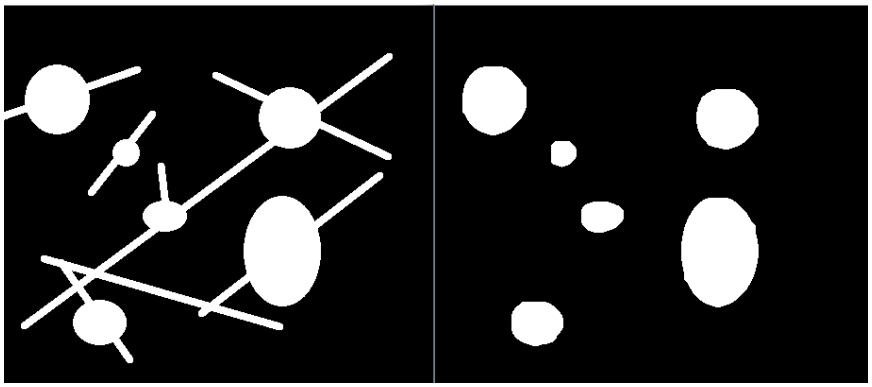
getStructuringElement(kernel shape , size)

- kernel shape : شکل هسته را مشخص می‌کند که برای اینکار از keyword های پیشقرض نظیر MORPH_RECT
- size : ابعاد هسته را به صورت یک دوتایی مشخص می‌کند.

```
>>> cv2.getStructuringElement(cv2.MORPH_RECT,(5,5))
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]], dtype=uint8)

# Elliptical Kernel
>>> cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
array([[0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0]], dtype=uint8)

# Cross-shaped Kernel
>>> cv2.getStructuringElement(cv2.MORPH_CROSS,(5,5))
array([[0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0]], dtype=uint8)
```

مثال :

نکته : این عملگرها برای شناسی در تصاویر نیز کاربرد دارند برای مثال در شکل زیر اگر یک هسته دایره ای شکل با اندازه دایره های تصویر را روی تصویر با عملگر open اعمل میکینیم و خطوط مانند نویز حذف می‌شوند. زیر تنها دوایر در این هسته مطابقت دارند و باقی می‌مانند و سایر اشکل حذف می‌شوند. توجه داشته باشید که ابعاد فیلتر تاثیر مهمی در این عملیات دارند. برای مثال اگر فیلتر ما از ذخامت خط ها کوچکتر باشد، خط ها به خوبی حذف نمی‌شوند.



کانتورها

کانتور چیست

کانتور (Contour) منحنی‌هایی هستند که از اتصال مجموعه‌ای از نقاط با ویژگی‌های مشابه شکل می‌گیرند و اغلب دور لبه‌های اشیاء در تصویر رسم می‌شوند. به عبارت ساده‌تر، کانتور مرز یا خطوطی است که شکل یک شیء را در تصویر مشخص می‌کند.

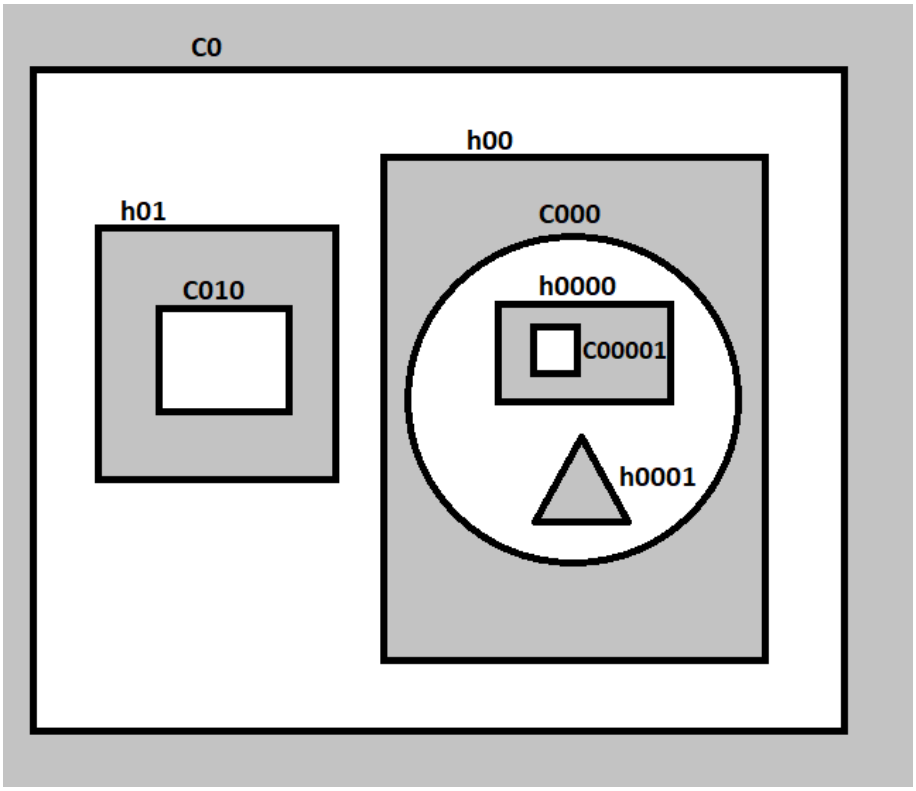
کانتورها کاربردهای زیادی دارند، از جمله:

- شناسایی و جداسازی اشیاء در تصویر
- محاسبه خصوصیات هندسی مانند مساحت، محیط و مرکز اشیاء
- ردیابی و تشخیص حرکت در تصاویر متوالی
- تشخیص شکل و اندازه اشیاء برای پردازش‌های بعدی

کانتورها معمولاً پس از اعمال فیلترهای لبه یا باینری کردن تصویر استخراج می‌شوند و کمک می‌کنند تا ساختار و شکل دقیق اشیاء به صورت عددی و تصویری تحلیل شود.در کانتور ها ما دو بخش داریم:

- Hole : hole یا همان چاله، پیرامون یک فضای خالی (فضای بین اشیا) را نشان می‌دهد. در تصاویر باینری چاله ها همان بخش های سیاه تصویر هستند. چاله ها در واقع مرزی داخلی اشیا هستند.
- Curve : curve یا همان منحنی پیرامون یک شی را توصیف میکنند. در تصاویر باینری، شی ها همان بخش‌های سفید تصویر هستند.

در تصویر زیر منحنی ها با حرف c و چاله ها با حرف h مشخص شده‌اند.



استخراج کانتورها

برای یافتن کانتورها لازم است تا از تصاویر باینری استفاده شود از این رو لازم است ابتدا یک threshod و یا یک عملگر تشخیص لبه مانند Canny روی تصویر اعمال کرد. باید دقت کرد که در تصویر باینری، شی مورد نظر باید سفید و پس زمینه آن سیاه باشد. برای یافتن کانتور های یک تصویر از تابع findContours() استفاده می‌شود.

تذکر: در نسخه‌های جدید openv، این تابع دارای تغییراتی بوده است.

findContours (img , mode , method) ->img۲ , contours , hierarchy

- img : تصویر ورودی
- Mode : این پارامتر مشخص میکند که تابع چه کانتورهایی را یافته و به چه صورت آنهارا ذخیره نماید. این پارامتر یکی از چهار حالت زیر است (در بخش آخر این فصل این موارد و عملکرد آن ها بیشتر پرداخته می‌شود.)
  - CV_RETR_EXTERNAL : در این حالت تابع تنها بیرونی ترین کانتور ها را پیدا می‌کند
  - CV_RETR_LIST : همه کانتور هارا یافته و آن‌ها را در یک لیست قرار می‌دهد. در تصویر مثال قبل در این حالت
  - CV_RETR_CCOMP : در این حالت تابع همه منحنی هارا به همراه مرز داخلی‌شان (حفره های درونشان) در یک سلسه مراتب دوسطحی نشان می‌دهد.

- CV_RERT_TREE : در این حالت تابع همه منحنی‌ها را به صورت سلسه مراتبی، در یک ساختار درختی نشان می‌دهد.

ی

- Method : این پارامتر چگونگی ذخیره کانتور را معرفی می‌کند و می‌تواند یکی از گزینه‌های زیر باشد.
  - CHAIN_APPROX_NONE : در این حالت تابع مختصات تمام نقاط مرزی (تشکیل دهنده کانتور) را ذخیره می‌کند. این کار حافظه زیادی اشغال می‌کند که معمولا در بسیار از موارد نیازی به این کار نیست.
  - CHAIN_APPROX_SIMPLE : در این حالت تابع تنها نقاط ابتدایی و انتهایی یک خط را ذخیره میکند و در واقع نقاط اضافی را حذف و کانتور را با تعدادی خط توصیف می‌کند از این‌رو در حافظه صرفه‌جویی می‌نماید.
  - CHAIN_APPROX_CODE : کانتور را به صورت کد زنجیری فریمن توصیف می‌کند

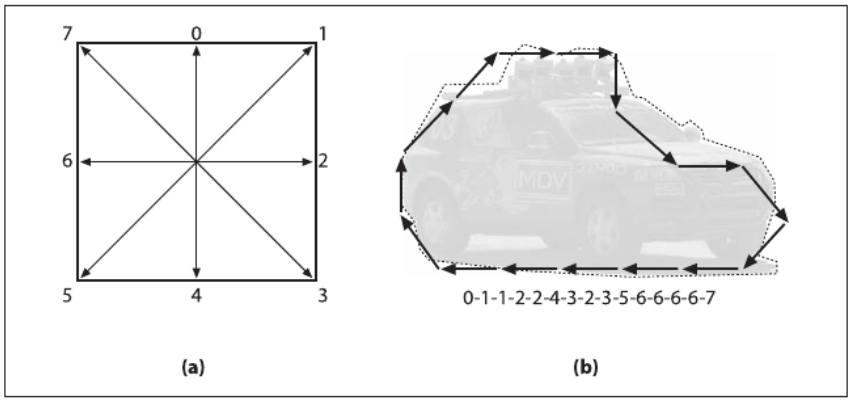
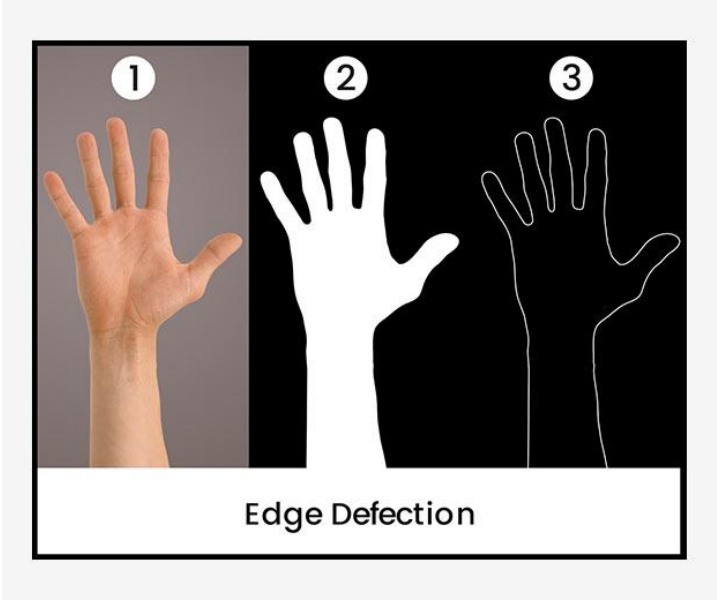


Figure 8-4. Panel a, Freeman chain moves are numbered 0–7; panel b, contour converted to a Freeman chain-code representation starting from the back bumper

در کد فریمن هر عدد نشان دهنده یک جهت بردار است و کانتور با این بردار ها توصیف می‌شود و خروجی آن دنباله‌ای از اعداد است که شکل شی را توصیف می‌کنند

خروجی های این تابع نیز عبارتند از :

- Img۲ : تصویر باینری که جست‌وجوی کانتورها روی آن انجام گرفته و در مواقعی که تصویر ورودی باینری نشده باشد، کاربرد دارد. (همواره برای رسیدن به نتیجه مطلوب، تصویر ورودی تابع بهتر است باینری باشد)
- Contours : کانتور های پیدا شده که ممکن است به صورت مجموعه از نقاط تشکیل دهنده کانتور یا مجموعه از خطوط تشکیل دهنده کانتور (مختصات نقاط ابتدا و انتهایی خطوط را باز میگرداند) و یا کد فریمن باشد
- Hierarchy : سلسله مراتب (توضیحات بیشتر در بخش های بعدی)



مثال :

```
import numpy as np
import cv2 as cv

im = cv.imread('test.jpg')
imgray = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
ret, thresh = cv.threshold(imgray, 127, 255, 0)

im2, contours, hierarchy =
 cv.findContours(thresh, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
```

رسم کانتور ها

برای رسم کانتورها از تابع drawContours() استفاده می‌شود. در واقع از این تابع میتوان برای رسم هر شکلی که مختصات نقاط مرزی آن‌را داریم بهره ببریم.

drawContours(img , contours , ContourIdx , color , thickness)

img : تصویر ورودی که می‌خواهیم کانتورها روی آن رسم شوند

Contours : لیست کانتور ها که از تابع findContours() بدست آوردیم و میخواهیم آن را رسم کنیم.

ContourIdx : پارامتر دوم یا Contours در واقع لیستی از چند منحنی است و هر درایه آن ، نشان دهنده یک منحنی می‌باشد. در این پارامتر با دادن شماره درایه، مشخص میکنیم کدام منحنی را تابع رسم کند. اگر این پارامتر ۱- داده شود ،تابع کلیه کانتورها را رسم می‌کند

Color : رنگ کانتور را به صورت BGR مشخص می‌کند

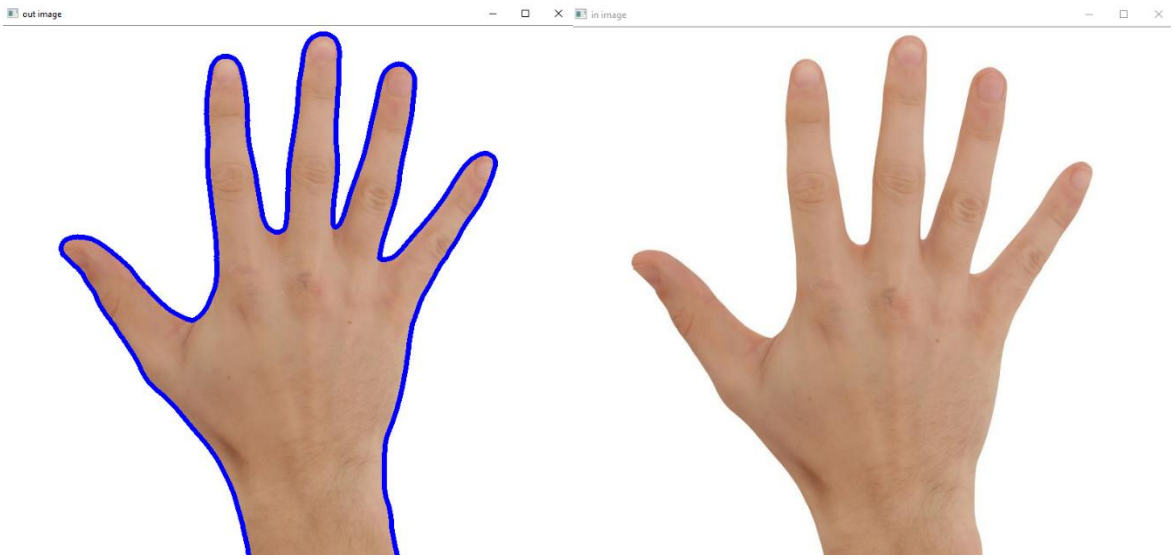
Thinkness : ذخامت کانتور را موقع رسم مشخص میکند. اگر این پارامتر ۱- داده شود، تابع کانتور را به صورت تو پر رسم می‌کند.

مثال :

```
#draw all contours
cv2.drawContours(img, contours, -1, (0,255,0), 3)

#draw 3th contour
cv2.drawContours(img, contours, 2, (0,255,0), 3)

#draw 3th contour
cnt=countours[2]
cv2.drawContours(img, [cnt], 0, (0,255,0), 3)
```

در مثال بالا دو روش برای رسم یک منحنی خاص که در این مثال منحنی سوم است، ارایه شده، خروجی هردوی این روش ها یکسان است اما در ادامه خواهیم فهمید روش دوم کارامد تر است

```
import numpy as np
import cv2


img = cv2.imread('images.jpg')
img2=img.copy ()

#Convert to gray ang thresh
imgry=cv2.cvtColor (img ,cv2.COLOR_BGR2GRAY )
ret,thresh = cv2.threshold (imgry,0,255,cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

#find and draw contours
im2, contours, hierarchy = cv2.findContours(thresh,cv2.RETR_CCOMP
,cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours (img2 , contours ,-1, (255,0,0) , thickness=5)

cv2.imshow ("out image", img2 )
cv2.imshow ("in image", img )
```



توابع کانتور

مساحت کانتور

برای بدست آوردن مساحت یک منحنی یا کانتور، از تابع contourArea() استفاده میکنیم که مساحت کانتور را بر اساس پیکسل محاسبه میکند.

contourArea(contour) -> Area

تذکر : دقت شود که خروجی تابع findContour() لیستی از کانتور ها است و نه یک کانتور

مثال :

```
cnt=contours[0]
area=cv2.contourArea (cnt)
```

طول کانتور

برای محاسبه طول یک کانتور یا منحنی از تابع arcLength() استفاده می‌شود. خروجی تابع برحسب پیکسل است

arcLength(countour , closed) -> length

تابع closed

این پارامتر یک مشخص میکند که آیا منحنی یک منحنی بسته است یا خیر. اگر منحنی بسته باشد مقدار True در غیر این صورت مقدار False به تابع ارسال می‌شود

مثال :

```
cnt=contours[0]
primeter=cv2.arcLength (cnt ,True)
```

Approximates Contour

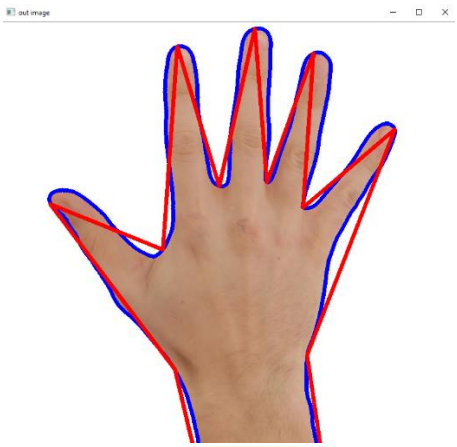
به معنی تقریب یک کانتور بوده که شکل کانتور را ساده تر می‌کند و از خطوط کمتری استفاده میکند. برای مثال فرض کنید شی مورد نظر شما یک مستطیل است اما به دلیل نویز، کانتور آن دقیقا شکل یک مستطیل نیست و دارای ناهنجاری است. در اینجا تقریب کانتور بسیار مفید خواهد بود. برای اینکار از تابع approxPolyDP() استفاده می‌شود

approxPolyDP(curves , epsilon , closed) -> approx. curve

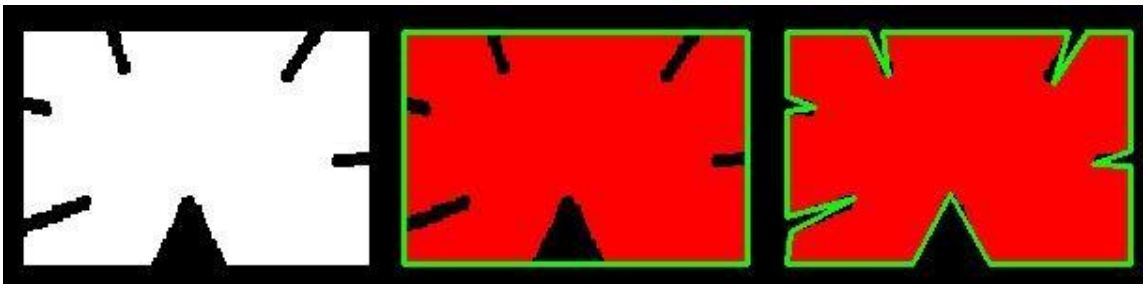
- curves : منحنی یا کانتور مورد نظر
- Epsilon : حداکثر اختلاف کانتور اصلی با کانتور تقریبی را بر اساس پیکسل مشخص می‌کند. در واقع این پارامتر دقت تخمین را مشخص میکند
- Closed : این پارامتر یک پارامتر Boolean بوده مشخص میکند که آیا منحنی بسته است یا خیر

مثال :

```
cntlen=cv2.arcLength (cnt ,True)
approxCnt=cv2.approxPolyDP(cnt , cntlen*0.01 , True)

cv2.drawContours (img2 , [approxCnt] ,0, (0,0,255) , thickness=5)
```



در تصویر مربع های زیر کانتور سمت راست با اپسیلون lenCnt * ۰.۰۱ تخمین زده شده که نتیجه مطلوب ما را ندارد اما منحنی وسطی با اپسیلون lenCnt*۰.۱ تخمین زده شده و کانتور تقریبی به شکل مستطیل درآمده و مارا به هدفمان رسانده است



Hull Contour

هال در واقع یک منحنی محدب است که تقریبی از کانتور بوده و آنرا محاط میکند. Hull و approx متقاوتند اما ممکن است در برخی سوژه ها Hull یک کانتور شبیه approx آن باشد . برای بدست آوردن hull از تابع ()convexHull استفاده می‌شود

hull convexHull (points [, hull [, clock wise , return point) ->

points : نقاط تشکیل دهنده منحنی که میخواهیم hull آن را رسم کنیم که در اینجا همان کانتور مد نظر می‌باشد

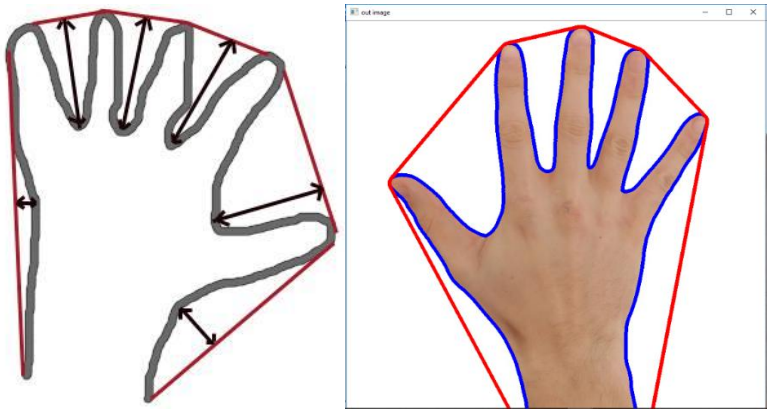
Hull : منحنی خروجی که معمولا به جای استفاده از این پارامتر از مقدار بازگشتی تابع استفاده می‌کنیم

Clock wise : اگر این پارامتر True باشد، منحنی hull در جهت عقربه های ساعت و اگر false باشد منحنی در جهت خلاف عقربه های ساعت رسم می‌شود.

Return Points : این پارامتر یک مقدار ، True و False است. اگر این پارامتر True باشد(حالت پیشفرض) تابع مختصات نقاط hull را باز میگرداند. اما اگر این پارامتر false باشد، شاخصه (شماره درایه) نقاط hull در کانتور را باز میگرداند. برای مثال برای یک hull مستطیلی در حالت true خروجی تابع به صورت [(۱۳۰,۴۰) , (۱۰۰,۹۰) , (۱۴۰,۷۰) , (۱۲۰,۶۰)] است که مختصات چهار نقطه HULL را نشان می‌دهد. اما اگر false باشد ، خروجی شاخصه نقاط بالا در کانتور مرج است و مثلا به صورت [۱۲۰, ۲۱۰,۳۵۰,۳۶۰] است که هر کدام از این اعداد شاخصه یکی از نقاط هستند و مثلا cnt[۱۲۷] همان نقطه (۱۲۰,۶۰) است پس (۱۲۰,۶۰)= cnt[۱۲۷] کاربرد حالت دوم در بدست آوردن convexity defects یا نقص تخلخل‌ها کاربرد دارد که در تصویر دست زیر با فلش مشخص شده اند.

مثال : در تصویر زیر منحنی آبی کانتور ورودی و منحنی قرمز رنگ hull آن است

```
hull=cv2.convexHull (cnt)
cv2.drawContours (img2 , [hull] ,0, (0,0,255) , thickness=5)
cv2.imshow ("out image", img2 )
```



برای چک کردن این موضوع که یک کانتور محدب است یا خیر، از تابع isContourConvex() استفاده می‌شود که خروجی آن یک مقدار True یا false است.

```
k = cv2.isContourConvex(cnt)
```

مستطیل محاطی

مستطیلی است که یک کانتور را محاط می‌کند. در opencv دو نوع مستطیل محاطی برای کانتور ها تعریف شده است. مستطیل نوع straight نام دارد که مستطیل افقی و بدون زاویه است که چرخش شی را در نظر نمیگرد و مساحت آن حداقل مساحت ممکن نیست. برای پیاده سازی این مستطیل محاطی از تابع boundingRect() استفاده می‌شود.

boundingRect (contour) -> x,y,w,h

مثال :

```
cnt=contour[0]
x,y,w,h = cv2.boundingRect(cnt)
cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
```

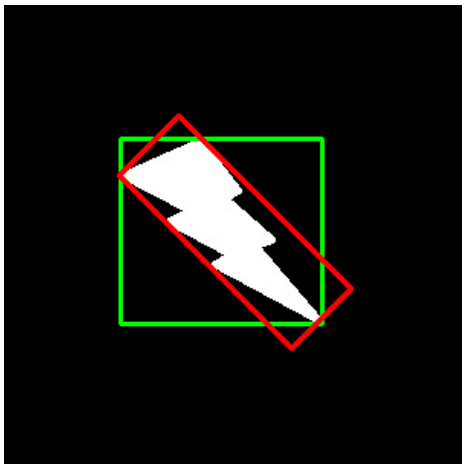
مستطیل محاطی دیگر rotated نام دارد که با حداقل مساحت رسم می‌شود از این رو زاویه دار بود و چرخش شی را در نظر می‌گیرد. برای بدست آوردن این مستطیل از تابع minAreaRect() استفاده می‌شود

minAreaRect (cnt) -> center(x,y) , (w,h) , angle of Rotation

تذکر : برای رسم یک مستطیل ما نیاز به مختصات چهار گوشه آن داریم اما خروجی این تابع به این صورن نیست از این رو از تابع boxPoints() استفاده میکنیم. برای یادگیری روش رسم این مربع به مثال زیر توجه فرمایید

مثال :

```
rect = cv2.minAreaRect(cnt)
box = cv2.boxPoints(rect)
box = np.int0(box)
cv2.drawContours(img,[box],0,(0,0,255),2)
```



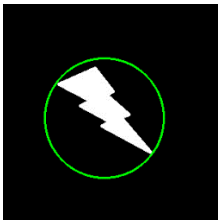
در تصویر بالا مستطیل سبز رنگ straight و مستطیل قرمز رنگ rotated است که حداقل مساحت ممکن را دارد.

دایره محاطی : برای بدست آوردن کوچکترین دایره محاطی از تابع minEncloseingCircle() استفاده می‌کنیم.

minEncloseingCircle(contour Or points) -> center(x,y) , radius

مثال

```
(x,y),radius = cv2.minEnclosingCircle(cnt)
center = (int(x),int(y))
radius = int(radius)
cv2.circle(img,center,radius,(0,255,0),2)
```



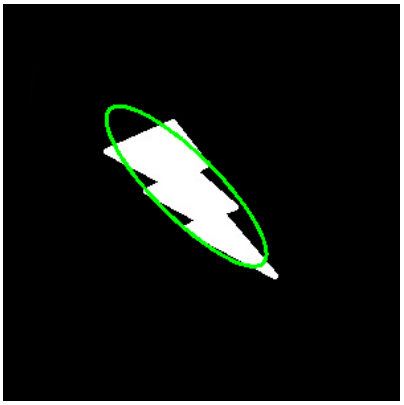
Fit Ellpse

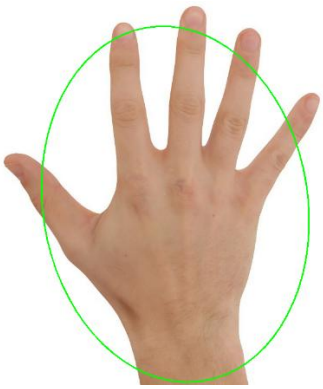
تابع مربوطه یک بیضی تقریبی از کانتور ایجاد میکند. تابع این عملیات fitEllipse() می‌باشد.

fitEllipse (contour) -> center(x,y) , axel(a,b) , angle

مثال :

```
ellipse = cv2.fitEllipse(cnt)
cv2.ellipse(img,ellipse,(0,255,0),2)
```





Moment : moment یک تصویر یا منحنی، خصوصیات مهمی از آنرا ارایه میدهد. خصوصیاتی نظیر مرکز جرم ، مرکز و... تابع بدست آوردن آن ()memnt است

Moment (array) -> moment

- Array : ارایه ورودی که در اینجا کانتور ما است.

خروجی این تابع یک دیکشنری است که ویژگی های مختلف تصویر را از آن استخراج میکنیم در مثال زیر ما مختصات مرکز جرم کانتور را بر اساس moment محاسبه میکنیم. فرمول استفاده شده در این مثال ثابت بوده و بسیار کاربردی است

```
import cv2
import numpy as np

img = cv2.imread('star.jpg',0)
ret,thresh = cv2.threshold(img,127,255,0)
contours,hierarchy = cv2.findContours(thresh, 1, 2)
cnt = contours[0]
M = cv2.moments(cnt)
print M
################################################
cx = int(M['m10']/M['m00'])
cy = int(M['m01']/M['m00'])
################################################
```

مثال :

```
================ RESTART: C:/Users/Amir_PC/Desktop/contour.py ================
{'m00': 202458.5, 'm10': 83242258.33333333, 'm01': 87696183.66666666, 'm20': 366
45025510.916664, 'm11': 36044150715.95833, 'm02': 45739380900.25, 'm30': 1695556
7422548.8, 'm21': 15727765277362.117, 'm12': 18990730401792.35, 'm03': 266035372
72980.6, 'mu20': 2419376439.9544563, 'mu11': -12761567.932510376, 'mu02': 775322
2602.634773, 'mu30': -100784360418.34375, 'mu21': -134765978208.18762, 'mu12': 1
95712759684.0923, 'mu03': 74519153383.89453, 'nu20': 0.05902437776019 33, 'nu11':
 -0.00031133791088543497, 'nu02': 0.18915168892253847, 'nu30': -0.00546453367878
758, 'nu21': -0.007307018902700266, 'nu12': 0.010611556815193388, 'nu03': 0.0040
4043267914525}
>>> cy
433
>>> cx
411
>>>
```

خصوصیات هندسی یک کانتور

Aspect Ratio

این پارامتر یا همان نسبت ابعاد، برابر است با نسبت $\frac{w}{h}$ مستطیل محاطی

```
x,y,w,h = cv2.boundingRect(cnt)
aspect_ratio = float(w)/h
```

Extent

این پارامتر برابر است با نسبت مساحت منحنی (کانتور) بر مساحت مستطیل محاطی .مساحت یک کانتور یا منحنی با تابع contourArea() محاسبه می‌شود. $extent = \frac{contour\ area}{rect\ area}$

```
x,y,w,h = cv2.boundingRect(cnt)
rect_area=w*h
area=cv2.contourArea( cnt)
extent=float (area) rect_area
```

Solidity

این پارامتر برابر است با نسب مساحت یک کانتور بر مساحت کانتور hull آن، تابع به دست آوردن hull یک منحنی convexHull() است. که ورودی آن منحنی مورد نظر و خروجی آن چندضلعی hull است. برای به دست آوردن مساحت یک hull از همان تابع contourArea() استفاده می‌شود. $solidity = \frac{contour\ area}{hull\ area}$

```
Hull =cv2.convexHull(cnt)

area=cv2.contourArea( cnt)
hull area=cv2.contourArea( hull)

solidity=float (area) rect_area
```

Equivalent Diameter

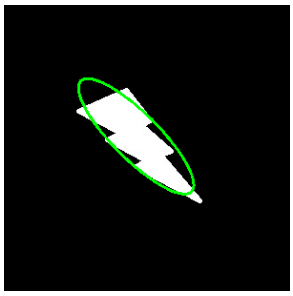
این پارامتر معادل است با قطر دایره‌ای که مساحتش با مساحت کانتور مورد نظر برابر باشد و مقدارش از فرمول ساده‌ی زیر بدست می‌آید $equivalent\ Diameter = 2 \times \sqrt{\frac{area}{\pi}}$

```
area=cv2.contourArea( cnt)
equi_diameter = 2*np.sqrt(area/np.pi)
```

Orientation

این پارامتر جهت گیری کانتور را بررسی می‌کند. که میتوان آنرا از تابع fitEllipse() برآورد کرد.

```
(x,y),(MA,ma),angle = cv2.fitEllipse(cnt)
ellipse = cv.fitEllipse(cnt)
cv.ellipse(img,ellipse,(0,255,0),2)
```



سایر توابع کانتور

```
(x,y),(MA,ma),angle = cv2.fitEllipse(cnt)
ellipse = cv.fitEllipse(cnt)
cv.ellipse(img,ellipse,(0,255,0),2)
```

Mask کانتور

گاهی نیاز است تا ما از کانتور یک ماسک ایجاد کنیم. به این صورت که پیکسل های تشکیل دهنده کانتور در ماسک سفید و سایر بخش ها سیاه شوند. برای ساخت یک ماسک از کانتور کافی است ابتدا یک آرایه با ابعاد تصویر با درایه‌های صفر به عنوان ماسک ایجاد کنیم سپس با تابع drawContour() تصویر کانتور را روی این آرایه، با رنگ سفید رسم میکنیم. همچنین برای بدست آوردن مختصات پیکسل‌های کانتور از تابع findNoneZero() استفاده میکنیم. ورودی این تابع تصویر مورد نظر که در اینجا ماسک ما است و خروجی آن لیستی از مختصات پیکسل های غیر صفر می‌باشد. برای درک بهتر به مثال زیر توجه نمایید.

مثال : در کد زیر ، در کامنت خط آخر روش دیگر برای بدست آوردن مختصات پیکسل های سفید ماسک داده شده است

```
mask = np.zeros(imgray.shape,np.uint8)
cv2.drawContours(mask,[cnt],0,255,-1)
pixelpoints = cv2.findNonZero(mask)
#pixelpoints = np.transpose(np.nonzero(mask))
```

Max min Value & Location

ما به استفاده از ماسکی که در قسمت قبل بدست آوردیم میتوانیم مختصات پیکسل‌هایی که متعلق به آن شی هستند و مقدارشان ماکزیمم و مینمم است را مکان و مقدارشان را بدست آوریم. این کار با تابع minMaxLoc() انجام میگرد. ورودی این تابع تصویر اصلی به همراه ماسک مذکور می‌باشد و خروجی آن مختصات پیکسل های مذکور است

```
min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(imgray,mask = mask)
```

Mean Color

توسط ماسک بخش قبل و با کمک تابع mean() میتوان شدت رنگ متوسط شی مورد نظر را بدست آورد. تصویر اصلی و ماسک مذکور به عنوان پارامتر به تابع ارسال می‌شوند و تابع، شدت متوسط رنگ شی را بر میگرداند.

```
min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(imgray,mask = mask)
```

Extreme Point

با روش زیر میتوان مختصات نقاطی را که در بالاتر ، پایین ترین، راستی ترین و چپ ترین بخش شی قرار دارند را بدست آورد

```
leftmost = tuple(cnt[cnt[:,:,0].argmin()][0])
rightmost = tuple(cnt[cnt[:,:,0].argmax()][0])
topmost = tuple(cnt[cnt[:,:,1].argmin()][0])
bottommost = tuple(cnt[cnt[:,:,1].argmax()][0])
```

convexity defect

برای یافتن convexity defects از تابع convexityDefects() استفاده میکنیم

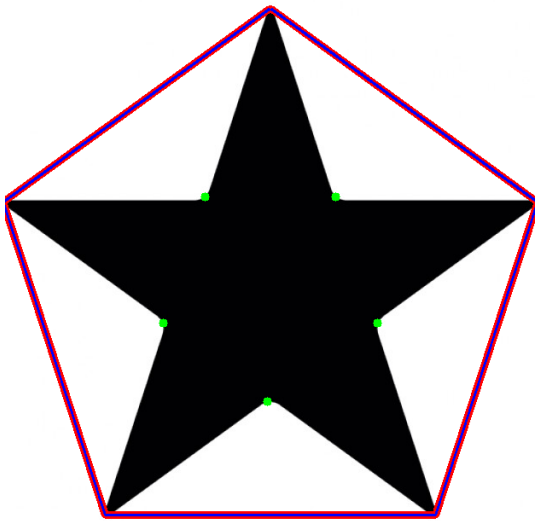
convexityDefects (contour , convex hull) -> convexityDefects

- contour : کانتور شی مورد نظر
- Hull : hull کانتور

خروجی این این تابع یک آرایه است که هر سطر آن یک به صورت [نقطه شروع یک خط hull، نقطه پایان یک خط hull، دورترین نقطه از کانتور نسبت به آن خط ، فاصله دورترین نقطه کانتور تا آن خط]

مثال : در مثال زیر ما بین دو نقطه شروع و پایان یک خط با رنگ آبی و مرکزیت دورترین نقطه از خط یک دایره سبز رسم می‌کنیم. دقت داشته باشید که مقدار سه پارامتر اول، شاخص های آن نقاط در کانتور هستند و نه خود نقاط (منحنی hull در تصویر با رنگ قرمز مشخص شده است)

```
_, cnts, _=cv2.findContours( thresh  ,cv2.RETR_CCOMP , cv2.CHAIN_APPROX_SIMPLE )

hullA = cv2.convexHull(cnt )
hullB = cv2.convexHull(cnts[0],returnPoints = False)
cv2.drawContours (img , [hullA] ,0, (0,0,255) , thickness=8)

convexDFS=cv2.convexityDefects (cnts[0] , hullB)

cnt=cnts[0]
for i in range(convexDFS.shape[0]) :
    s,e,f,d=convexDFS[i,0]
    start=tuple(cnt[s,0])
    end=tuple(cnt[e,0])
    far=tuple (cnt[f,0])
    cv2.line (img , start , end ,(255,0,0),2)
    cv2.circle (img , far , 5, (0,255,0), -1)
cv2.imshow ("out image", img )
```



وضعیت یک نقطه نسبت به کانتور

تابع pointPolygonTest() کوتاه ترین فاصله یک نقطه دلخواه نسبت به یک کانتور را بدست می‌آورد. و اگر فاصله مثبت باشد، نقطه درون کانتور و اگر منفی باشد، نقطه خارج از کانتور قرار دارد، فاصله صفر یعنی نقطه روی کانتور قرار دارد.

PointPolygonTest (contour , point , measure Dist) -> distance

Contour : کانتوری که میخواهیم وضعیت نقطه را تسبت به آن بررسی کنیم

Point : مختصات نقطه مورد نظر

Measure Dist :اگر این پارامتر True باشد، تابع کمترین فاصله نقطه از کانتور را باز میگرداند، اما اگر این پارامتر False باشد، تابع تنها با مقادیر (outside) ۱-, (on) ۰ , (inside)۱ وضعیت نقطه نیست به کانتور را مشخص میکند

مثال :

```
dist = cv2.pointPolygonTest(cnt,(50,50),True)
```

تشخیص شباهت دو کانتور

در بسیاری از کاربردهای پردازش تصویر، ممکن است بخواهیم تشابه شکل دو شیء را بررسی کنیم، بدون توجه به اندازه، موقعیت یا چرخش آن‌ها. یکی از روش‌های مؤثر برای انجام این کار، استفاده از کانتورها (Contours) است، زیرا کانتورها مرز و شکل دقیق اشیاء را نمایش می‌دهند.

با استخراج کانتورها از دو شیء، می‌توان شکل آن‌ها را به صورت عددی و قابل مقایسه نمایش داد. برای این منظور، OpenCV تابع matchShapes()را فراهم کرده است که شباهت دو شیء را بر اساس کانتورهایشان اندازه‌گیری می‌کند .این تابع در واقع بر پایه محاسبه Hu Momentsاست و معیارهای خاصی را برای مقایسه هندسی اشیاء استفاده می‌کند، به طوری که حتی اگر اشیاء چرخیده، مقیاس شده یا جابه‌جا شده باشند، می‌توان میزان شباهت آن‌ها را تعیین کرد.

matchShapes (cotour۱ , contour۲ , method , parameter)

- contour۲ / contour۱ : کانتور هایی که میخواهیم باهم مقایسه کنیم.
- Method : این پارامتر روش مقایسه دو کانتور را معین میکند که یکی از سه حالت زیر است
  - CV_CONTOURS_MATCH-I۱
  - CV_CONTOURS_MATCH_I۲
  - CV_CONTOURS_MATCH_I۳
- Parameter پارامتر روش خاص (فعلا ساپورت نمیکند ، ۰ داده شود)

تذکر : هرچه مقدار خروجی تابع کمتر باشد، شباهت دو کانتور بیشتر است.

مثال :

```
import cv2
import numpy as np

img1 = cv2.imread('star.jpg',0)
img2 = cv2.imread('star2.jpg',0)

ret, thresh = cv2.threshold(img1, 127, 255,0)
ret, thresh2 = cv2.threshold(img2, 127, 255,0)

contours,hierarchy = cv2.findContours(thresh,2,1)
cnt1 = contours[0]
contours,hierarchy = cv2.findContours(thresh2,2,1)
cnt2 = contours[0]

ret = cv2.matchShapes(cnt1,cnt2,1,0.0)
print ret
```

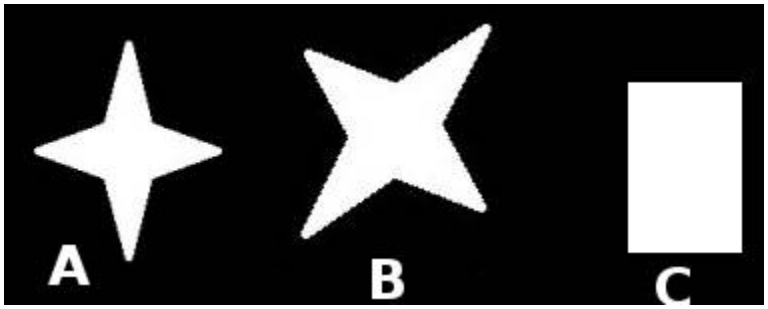
$$I_1(A,B) = \sum_{i=1\ldots7} \left| \frac{1}{m_i^A} - \frac{1}{m_i^B} \right|$$

$$I_2(A,B) = \sum_{i=1\ldots7} \left| m_i^A - m_i^B \right|$$

$$I_3(A,B) = \max_{i=1\ldots7} \frac{\left| m_i^A - m_i^B \right|}{\left| m_i^A \right|}$$

$$m_i^A = \mathrm{sign}(h_i^A) \cdot \log h_i^A$$
$$m_i^B = \mathrm{sign}(h_i^B) \cdot \log h_i^B$$

مثال : در مقایسه ای که میان سه تصویر زیر صورت گرفت است، نتایج به صورت زیر است.



تطابق A با خودش = ۰.۰

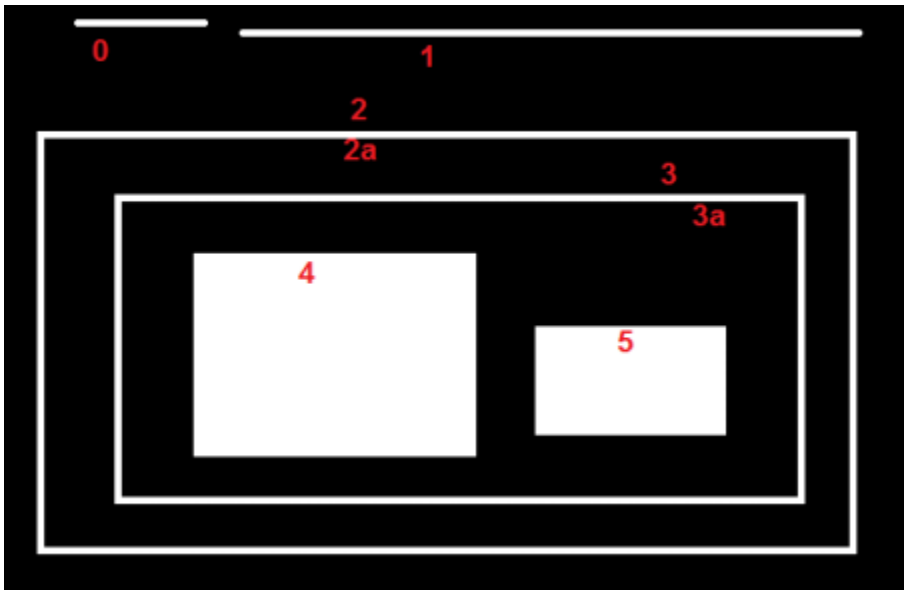
تطابق A با B = ۰.۰۰۱۹۴۶

تطابق A با C = ۰.۳۲۶۹۱۱

همانطور که مشخص است اگر رزولوشن، ابعاد و زاویه دو تصویر یکسان نباشد، تاثیری در مقایسه کانتورها ندارد. زیرا این عملگر مقایسه شکل را مستقل از این پارامترها محاسبه می‌نماید.

سلسله مراتب یا hierarchy

در مباحث قبلی با تابع findContours() آشنا شدید. همانطور که مشاهده کردید این تابع سه خروجی داشت که شامل تصویر، کانتورها و سلسله مراتب می‌شد. اما سلسله مراتب واقعا به چه معنایی است. همچنین تاثیر پارامتر دوم یعنی mode به چه صورت است. ما از کانتورها برای تشخیص اشیا در تصویر استفاده میکنیم . اما در برخی تصاویر ، اشیا سلسله مراتبی هستند و یک شی ممکن است در داخل شی دیگری باشد. برای مثال صورت انسان که خود شامل اشیایی دیگر نظیر چشم ها است. ما به کمک خروجی سوم یعنی hierarchy ارتباط کانتور ها با یکدیگر را در میابیم . اینکه یک کانتور یک پدر برای کانتوری دیگر باشد و یا برعکس یک کانتور، زیرمجموعه یا فرزندی از یک کارتون دیگر باشد . هر کانتور شامل چهار پارامتر به صورت مقابل است که این ارتباطات را توصیف میکند [کانتور بعدی، کانتور قبلی ، فرزند اول ، پدر] که این مقادیر در hierarchy ذخیره میشوند.
[next , previous , First child , parent]



در تصویر بالا ما ۵ شی داریم، که از ۰ تا ۵ نام گذاری شده اند، کانتور ۲ و ۲a به ترتیب به مرز خارجی و داخلی یک شی تعلق دارند. کانتور های ۰،۱،۲ هم سطح هستند و در بیرونی تری سطح قرار دارند که آنرا سطح صفر می‌نامیم. بعد از کانتور ۲a است که فرزند کانتور ۲ است(کانتور ۲ نیز پدر کانتور ۲a است) و در سطح یک قرار دارد. کانتور ۳ فزند کانتور ۲a است. و به همین

صورت تا آخرین سطح که کانتور ۴ و ۵ در آن قرار دارند . این کانتور ها هردو فرزند کانتور ۳a هستند و کانتور ۴ اولین فرزند کانتور ۳a است (ممکن است کانتور ۵ فرزند اول باشد) این ها سطح بندی و سلسه مراتب کانتور ها هستند حال به پارامتر های hierarvhy میپردازیم.

کانتور بعدی : کانتوریست هم سطح با کانتور مورد نظر که بعد از آن است. برای مثال در تصویر بالا کانتور ۰ را درنظر بگیرید، کانتور هم سطح بعدی این کانتور کانتور ۱ است. پس مقدار next برای کانتور ۰ برابر ۱ است. همچنین برای کانتور ۱ برابر ۲ می‌باشد. اما برای کانتور ۲، چون هیچ کانتور همسطح دیگری پس از آن وجود ندارد، مقدار پارامتر next= -۱ است.

کانتور قبلی : کانتوریست در همان سلسله مرتبه که قبل از کانتور مورد نظر قرار دارد. برای مثال برای کانتور ۲، کانتور همسطح قبلی ، کانتور ۱ است، در نتیجه برای کانتور ۲ مقدار پارامتر previous برابر ۱ است و برای کانتور ۱ ، previous=۰ می‌باشد. اما برای کانتور ۰ چون کانتور همسطی قبل از آن قرار ندارد این مقدار ۱- است.

اولین فرزند : همانگونه که از نامش مشخص است، اولین فرزند کانتور مورد نظر را تعیین میکند. برای مثال اولین فرزند کانتور ۲، کانتور ۲a است که تنها فرزند آن می‌باشد . همچنین برای کانتور ۳a اولین فرزند کانتور ۴ است. مشخصا اگر کانتور فرزندی نباشد مقدار این پارامتر ۱- خواهد بوذ

پدر : کانتور پدر کانتور فعلی را مشخص می‌کند. برای کانتور های ۴ و۵ ، کانتور پدر کانتور ۳a است و برای کانتور ۳a مقدار parent برابر ۳ است و همین طور الی آخر. اگر کانتوری پدر نداشته باشد، مقدار این پارامتر ۱- است.

حالا که با سلسه مراتب کانتور ها آشنا شده‌ایم به سراغ پارامتر mode در تابع ()findContour می‌رویم تا بررسی کنیم که پرچم های RETR_CCOMP, RETR_LIST و غیر واقعا به چه مفهومی هستند

پرچم‌های کانتورها

۱-پرچم RETR_LIST

این پرچم ساده ترین حالت است که در آن تابع کلیه کانتور هارا بدون در نظر گرفتن سلسه مرتبه ذخیره میکند. از این رو پارامتر های Firs-Child و Parent در این حالت برای هر کانتور برابر ۱- است. اما کانتور های قبلی و بعدی در این حالت مشخص هستند. در زیر ما hierarchy یک تصویر را نشان دادیم، سطر اول این آرایه مربوط به کانتور ۰ و سطر دوم مربوط به کانتور یک و الی آخر. برای مثال برای کانتور ۰ ام، طبق hierarchy کانتور بعدی کانتور ۱ (پارامتر اول) است و کانتور قبلی آن ۱- (پارامتر دوم) است که یعنی کانتوری قبل از آن وجود ندارد

```
>>> hierarchy
array([[[ 1, -1, -1, -1],
        [ 2,  0, -1, -1],
        [ 3,  1, -1, -1],
        [ 4,  2, -1, -1],
        [ 5,  3, -1, -1],
        [ 6,  4, -1, -1],
        [ 7,  5, -1, -1],
        [-1,  6, -1, -1]]])
```

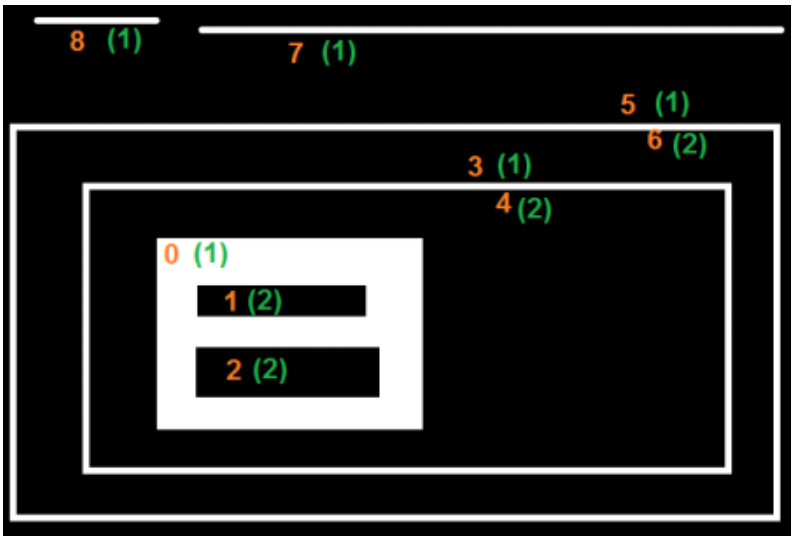
```
>>> hierarchy
array([[[ 1, -1, -1, -1],      #contour 0
        [ 2,  0, -1, -1],      #contour 1
        [-1,  1, -1, -1]]])    #contour 2
```

پرچم RETR_EXTERNAL

با استفاده از این پرچم، تابع تنها کانتور هایی را که در بالاترین سطح قرار دارند پیدا می‌کند. در واقع در این حالت، تنها کانتور هایی که در سطح صفر هستند بررسی می‌شوند و سایر کانتور ها از دید تابع حذف می‌شوند. برای تصویر بالا نتیجه به صورت زیر است و تنها کانتور های ۰,۱,۲ در خروجی دیده می‌شوند.

پرچم RETR_CCOMP

در این حالت تابع کلیه کانتور ها را استخراج میکند و آنها را در دو سطح، سطح بندی میکند، به این صورت که کانتور های متعلق به اشیا (کانتورهای مرز خارجی) را در سطح ۱ و کانتور های متعلق به حفره ها (کانتور مرز داخلی) را در سطح ۲ قرار می‌دهد البته در اصل کانتور حفره ها بستگی به کانتوری که در آن قرار دارند سطح بندی می‌شوند. برای درک بهتر به تصویر زیر توجه قرمایید



در تصویر بالا اعداد نارنجی رنگ شماره کانتور و اعداد سبز نشان دهنده سطح آن کانتور می‌باشند.

در تصویر بالا کانتور ۰ را در نظر بگیرید. در این کانتور کانتور قبلی وجود ندارد. کانتور بعدی آن که هم سطح آن باشد(سطح ۱) کانتور ۳ است. از طرفی این کانتور چون در سطح ۱ قرار دارد هیچ پدری ندارد و فرزند اول آن کانتور ۱ است. بنابراین آرایه سلسه مرتبه این کانتور به صورت مقابل است : [۱- , ۱ , ۱- , ۳]

همچنین برای کانتور ۱ ، کانتور بعدی آن در همان سطح، کانتور ۲ است و کانتور قبل از آن وجود ندارد. همچنین این کانتور چون در سطح ۲ است، هیچ فرزندی ندارد و پدر آن کانتور ۰ میباشد. پس آرایه سلسه مرتبه ان برابر است با [۰ , ۱- , ۱- , ۲]

برای کانتور ۲ نیز، کانتور بعدی هم سطحش باسد وجود ندارد و کانتور قبلی آن کانتور ۱، مشابها این کانتور فرزندی ندارد و کانتور پدر آن کانتور ۰ است [۰ , ۱- , ۱, ۱-]

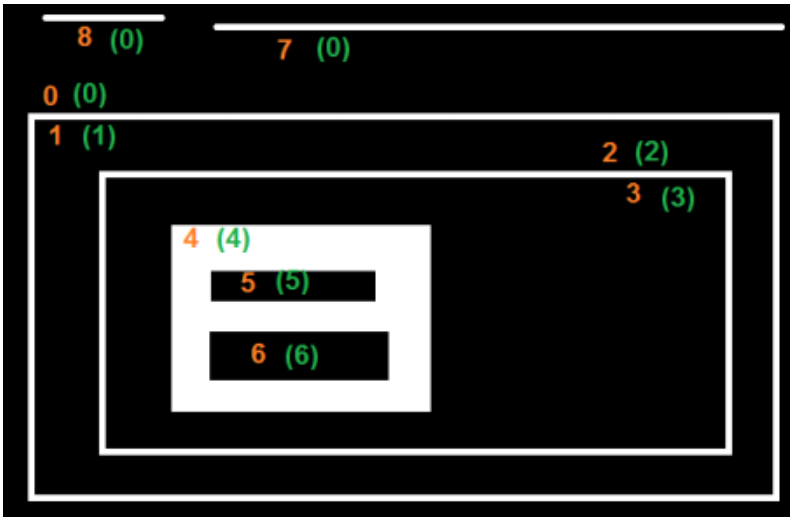
برای کانتور ۳ نیز کانتور بعد اش ۵، کانتور قبلی کانتور ۰ ، کانتور فرزند آن کانتور ۴ و کانتور پدر ندارد [۱- , ۴ , ۰ ,۵]

همچنین برای کانتور ۴، کانتور بعدی و قبلی در همان سطح و مشخصا فرزندی ندارد همچنین کانتور پدر آن کانتور ۳ است

```
>>> hierarchy
array([[[ 3, -1,  1, -1],
        [ 2, -1, -1,  0],
        [-1,  1, -1,  0],
        [ 5,  0,  4, -1],
        [-1, -1, -1,  3],
        [ 7,  3,  6, -1],
        [-1, -1, -1,  5],
        [ 8,  5, -1, -1],
        [-1,  7, -1, -1]]])
```

پرچم RETR_TREE

در این حالت پرچم کلیه کانتور ها را یافته و یک سلسله مراتب کامل برای آن ایجاد میکند به طوری که در آن کانتور پدر، فرزند، نوه و... قابل تشخیص می‌باشد . برای درک بهتر این موضوع تصویر زیر را درنظر بگیرید.



در تصویر بالا اعداد نارنجی رنگ شماره کانتور و اعداد سبز سطح آنان است

برای مثال برای کانتور ۰، کانتوری قبل از آن در همان سطح وجود ندارد اما کانتور بعدی آن کانتور ۷ است، فرزند این کانتور کانتور ۱ است و هیچ پدری ندارد در نتیجه آرایه سلسله مرتبه آن به صورت [۱- , ۱ , ۱- , ۷] است

همچنین برای کانتور ۲، هیچ کانتور همسطحی در قبل و بعد آن نیست. کانتور پدر آن ۱ است و فرزند اول آن کانتور ۳ است پس آرایه آن به صورت مقابل خواهد بود [۱ , ۳ , ۱- , ۱-]

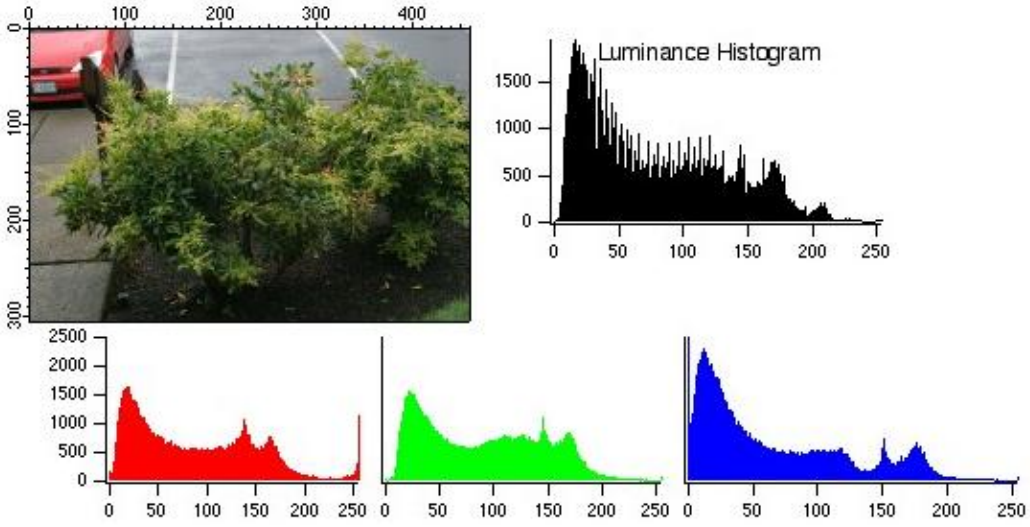
کانتور ۳ نیز هیچ کانتور همسطحی ندارد، کانتور پدر آن ۲ و فرزند اول آن کانتور ۴ است. [۲ , ۴ , ۱- , ۱-]

```
>>> hierarchy
array([[[ 7, -1,  1, -1],
        [-1, -1,  2,  0],
        [-1, -1,  3,  1],
        [-1, -1,  4,  2],
        [-1, -1,  5,  3],
        [ 6, -1, -1,  4],
        [-1,  5, -1,  4],
        [ 8,  0, -1, -1],
        [-1,  7, -1, -1]]])
```

هیستوگرام

هیستوگرام چیست

هیستوگرام در واقع یک نمودار میله ای است که محور افقی آن شامل داده ها و محور عمودی فراوانی داده ها در یک مجموعه را نشان می‌دهد. پس هیستوگرام یک تصویر، فراوانی(تعداد) پیکسل های بازه های رنگی مختلف را نشان می‌دهد. مثلا در یک تصویر خاکستری می‌تواند نشان دهد چه تعداد پیکسل ها در بازه (۰,۱۰) و (۱۰,۲۰) و... را نشان دهد. در تصویر زیر هیستوگرام کانال های سبز، قرمز و آبی را مشاهده میکند. هر بخش از این نمودار ها نشان میدهد که از چه شدت پیکسلی چه میزان استفاده شده برای مثال در هیستوگرام کانال آبی مشخص است که تعداد بیشتر پیسکل ها دارای شدت آبی بین ۰ تا ۵۰ هستند.



هیستوگرام کاربردهای زیادی در پردازش تصویر دارد و اطلاعات مهمی از تصویر را در خود ذخیره کرده. نظیر ماهیت تصویر، تیرگی یا روشنی تصویر و...

محاسبه هیستوگرام تصویر

هیستوگرام یک تصویر را میتوان با تابع calcHist() بدست آورد

- calcHist(imag , channels num , mask , hist Size , ranges)
- imag : تصویری که میخواهیم هیستوگرامش را بدست آوریم.
- Channels num : شماره کانال‌هایی از فضای رنگ تصویر که میخواهیم هیستوگرامشان را در تصویر را بدست آوریم.

- Mask: در صورتی که میخواهیم هیستوگرام بخشی از تصویر را بدست آوریم، ماسک مورد نظر را در این آرگومان وارد میکنیم در غیر این صورت، برای کل عکس، عبارت None را وارد می‌کنیم.
- Hist Size: تعداد کل ستون های هیستوگرام (میله های نمودار میله) را مشخص میکند مشخصا هرچه تعداد ستون ها بیشتر باشد، پردازش سنگین‌تر خواهد بود.
- Ranges : بازه محور افقی هیستوگرام را مشخص میکند مثلا برای بررسی پیکسل های روشن در یک تصویر خاکستری بازه ۱۰۰ تا ۲۵۵ را انتخاب میکنیم . مشخصا برای بررسی کل بازه محور در یک تصویر ۸ بیتی این بازه ۰ تا ۲۵۵ خواهد بود

نکته: مقادیر imag, channels num , Hist size , Ranges به صورت لیست باید وارد شوند

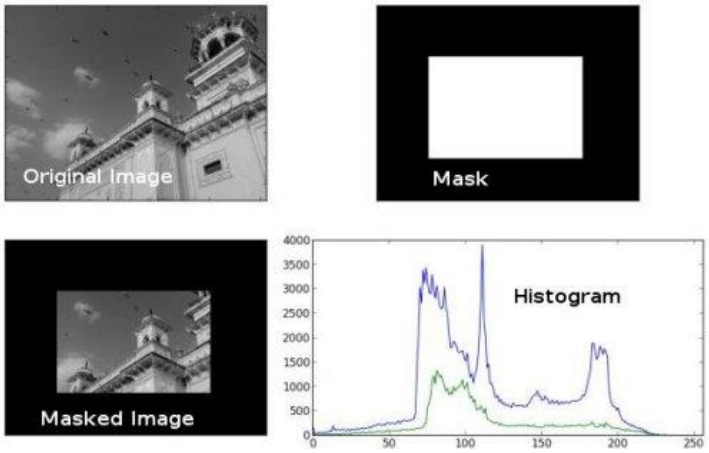
مثال : در این مثال هیستوگرام کانال سبز تصویر img را در بازه ۰ تا ۲۵۵ (مقادیر پیکسلی) با ۲۵۶ ستون بدست آوردیم.

```
import cv2
img = cv2.imread('opencv_logo.png')

hist = cv2.calcHist([img] , [0] , None , [256] , [0,255])
```

برای تعریف یک ماسک کافی است، با کتابخانه numpy و از طریق توابع مربوطه (array() یا zeros() و..) یک آرایه با ابعاد تصویر ایجاد نماییم و پیکسل هایی که میخواهیم ماسک (محافظت) شوند، درایه نظیرشان در آرایه را ۰ (سیاه) و پیکسل مورد هدف را درایه نظیرسان را ۲۵۵ (سفید) مقدار می‌دهیم و نهایت در تابع calcHist آن را وارد می‌کنیم.

مثال : در مثال زیر ما میخواهیم تنها هیستوگرام یک بخش مربعی از تصویر را محاسبه کنیم.

```
import cv2
img = cv2.imread('home.png')

# crat mask
mask= np.zeros(img.shap[:2] , np.uint8 )
mask[100:300 , 100:400]=255
masked_img=cv2.bitwise_and(img , img , mask=mask)

#calc Hist
Hist_full = cv2.calcHist([img] , [0] , None , [256] , [0,255])
Hist_mask = cv2.calcHist([img] , [0] , mask , [256] , [0,255])
```



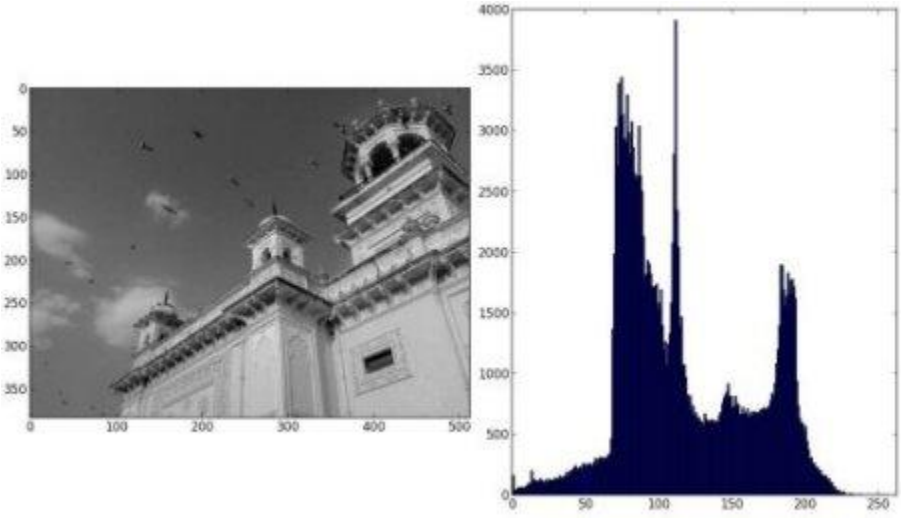
برای نمایش هیستوگرام تصویر می‌توان از کتابخانه matplotlib و ماژول pyplot استفاده کرد. با اد کردن این ماژول به برنامه، با استفاده از تابع ()plot و ()show هیستوگرام را نمایش داد.

Plot (hist)

Show()

مثال :

```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread('home.png')

#calc Hist
Hist = cv2.calcHist([img] , [0] , None , [256] , [0,255])
#show hist
Plt.plot(hist)
Plt.show()
```



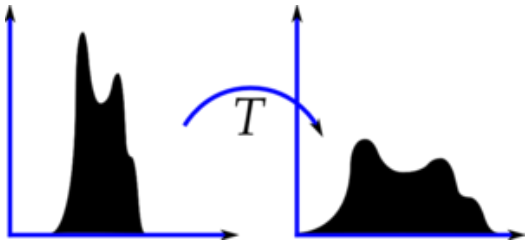
بهبود کنتراست با Equalization Histogram

یکی از روش‌های بهبود کنتراست تصویر، یکنواخت‌سازی هیستوگرام (Histogram Equalization) است.

با این روش، به جای اینکه تصویر تنها از بازه محدودی از مقادیر رنگی استفاده کند، پیکسل‌ها از کل بازه مقادیر ممکن بهره می‌برند.

به عنوان مثال، در تصویر سمت چپ، هیستوگرام نشان می‌دهد که پیکسل‌ها تنها در یک بازه محدود از شدت‌ها قرار دارند، در حالی که تصویر با کنتراست مناسب، پیکسل‌های آن در تمام بازه شدت‌ها پخش شده‌اند، مشابه هیستوگرام سمت راست.

در واقع، فرآیند یکنواخت‌سازی هیستوگرام باعث می‌شود که هیستوگرام در محور x کشیده شود و تمامی شدت‌ها به طور یکنواخت در تصویر توزیع شوند، که نتیجه آن افزایش وضوح و کنتراست تصویر است.



با یکنواخت سازی هیستوگرام ما میتوانیم تصاویر با شرایط نوری متفاوت را باهم مقایسه کنیم و مثلا در تشخیص چهره بسیار کاربردی خواهد بود. برای یکنواخت سازی هیستوگرام از تابع equalizeHist() استفاده می‌کنیم

equalizeHist(one channel img) ->img

تذکر: همانطور که مشخص است، ورودی این تابع باید یک تصویر تک کاناله باشد. مانند یک تصویر خاکستری.

تذکر: برای تصاویر رنگی لازم است ابتدا به کمک تابع split() کانال های تصویر را جدا کرده و هر کانال را به صورت جداگانه equalize کرده و سپس تصاویر بدست آمده را با تابع merg() ادغام مینماییم

```
import cv2
import matplotlib.pyplot as plt
img = cv2.imread('wiki.jpg', 0)
eqH = cv2.equalizeHist(img)
#show
cv2.imshow(“ eq hist”, eqH)
cv2.imshow(“ org”, img)
```

در مثال قبل ما با استفاده از یکنواخت سازی هیستوگرام کنتراست تصویر را بهبود بخشیدیم. در تابع equalizeHist() ما کنتراست کل تصویر را به صورت کلی و یکجا یکسان سازی کردیم. اما این کار ممکن است در بعضی موارد نتیجه خوبی نداشته باشد

در تصاویر بالا تصویر سمت راست هیستوگرامش یکنواخت شده است. با وجود آنکه کنتراست کتابخانه بهتر شده است اما صورت مجسمه مطلوب نیست و اطلاعات زیادی از دست رفته. علت به این خاطر است که هیستوگرام این تصویر به محدوده خاصی از مقادیر، محدود نبوده . برای بهبود کنتراست این تصویر، ابتدا باید تصویر به بلوک های کوچکی مثلا ۸×۸ ، تقسیم شود که این بلوک هارا کاشی می‌نامیم سپس هیستوگرام این کاشی هارا یکی یکی یکنواخت می‌کنیم. اما اگر تصویر دچار نویز باشد، نویز آن

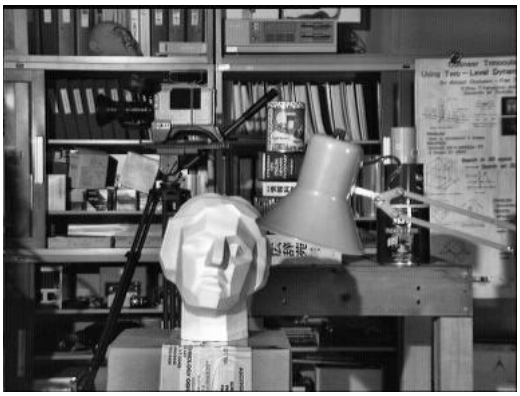
تقویت خواهد شد، برای حل این موضوع، یک حد مجاز برای bin یا میله هیستوگرام در نظر گرفته می‌شود و اگر طول bin از این حد مجاز بیشتر شود، پیکسل های متعلق به آن مقدار همگی حذف و بین سایر مقادیر تقسیم می‌شوند سپس عمل یکنواخت سازی هیستوگرام انجم می‌پذیرد.

برای پیاده سازی در openCV باید یک CLAHE ایجاد کرد و آن را به کمک تابع apply() روی تصویر پیاده سازی کرد. تابع ایجاد CLAHE ، createCLAHE() می‌باشد

createCLAHE ([, clipLimit [, titleGrid Size)

- clipLimit : حد مجاز برای طول bin ها
- titleGrid Size : اندازه کاشی ها برای تقسیم بندی عکس

```
import numpy as np
import cv2

img = cv2.imread('tsukuba_l.png',0)

# create a CLAHE object (Arguments are optional).
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
cl1 = clahe.apply(img)
cv2.imwrite('clahe_2.jpg',cl1)
```



هیستوگرام دو بعدی

در بخش قبل ما هیستوگرام یک کانال را بدست آوردیم . این هیستوگرام یک بعدی بود و تنها شدت رنگ یک کانال را یا تصویر خاکستری را بررسی می‌کرد. اما هیستوگرام دو بعدی هیستوگرامی است که ویژگی رنگ و میزان اشباع را به تصویر بکشد و کاربرد اصلی آن برای تصاویر رنگی است. برای بدست آوردن هیستوگرام دو بعدی از همان تابع calcHist() استفاده می‌کنیم اما پیش از آن باید تصویر را به فضای رنگی HSV ببریم. مشخصات این هیستوگرام به صورت زیر است

Channels : به صورت [۰,۱] است زیر میخواهیم هیستوگرام کانال های h , s را بدست آوریم.

Bins : به صورت [۲۵۶ , ۱۸۰] است . مقدار ۱۸۰ برای کانال h و مقدار ۲۵۶ برای s

Ranges : به صورت [۰,۲۵۶ , ۰,۱۸۰] می‌باشد.

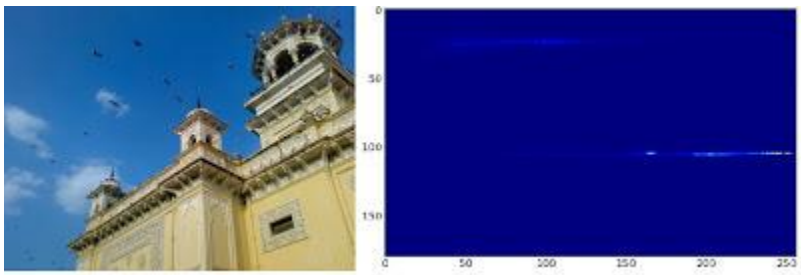
مثال :

```
import cv2
import numpy as np
img = cv2.imread('home.jpg')
hsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
hist = cv2.calcHist([hsv], [0, 1], None, [180, 256], [0, 180, 0, 256])
```

برای نمایش هیستوگرام دوبعدی دو راه وجود دارد. راه اول استفاده از تابع imshow() است. زیر هیستوگرام ما دو بعدی بوده و میتوان آنرا به عنوان یک تصویر نشان داد. اما تصویر خروجی تصویری خاکستری بوده و درکی از رنگ ها در آن وجود ندارد. راه دیگر استفاده از توابع کتابخانه matPlotlib است که درک بهتری از تراکم پیکسلی برای ما فراهم میکند. اما شاید باز هم مانند بالا رنگ ها مشکل باشد مگر با دانستن مقادیر H به ازای رنگ های مختلف.
تذکر : هنگام استفاده از تابع imshow در کتابخانه matplotlib مقدار پارامتر interpolation را برابر 'nearest' قرار دهید.

مثال : در هیستوگرام تصویر زیر ، محور افقی مربوط به پارامتر H و محور عمودی پارامتر S را نشان می‌دهد

```
hist = cv2.calcHist( [hsv], [0, 1], None, [180, 256], [0, 180, 0, 256] )

plt.imshow(hist,interpolation = 'nearest')
plt.show()
```



در هیستوگرام بالا یک در h=۱۰۰ و s=۲۰۰ یک تراکم پیکسلی مشاهد می‌شود که متعلق به آبی آسمان است. همچنین در H=۲۵ و S=۱۰۰ نیز تراکم پیکسل دیگری به چشم می‌خود که متعلق به رنگ زرد قصر است.

تشخیص محتوای خاصی از تصویر

گاهی در پردازش تصویر نیاز داریم تا یک محتوای خاص را در تصویر جست‌وجو کنیم و برای هر پیکسل، احتمال تعلق آن به آن محتوا را مشخص نماییم.

به عنوان مثال، فرض کنید می‌خواهیم در یک تصویر، احتمال تعلق هر پیکسل به پوست بدن را بدست آوریم.

برای این کار می‌توان از پس‌افکنش هیستوگرام (Histogram Backprojection) استفاده کرد. روش کار به این صورت است که ابتدا هیستوگرام نمونه‌ای از محتوای مورد نظر (مثلاً یک بخش از پوست) محاسبه می‌شود. سپس این هیستوگرام را به عنوان یک تابع احتمال در نظر می‌گیریم، به طوری که برای هر پیکسل تصویر، احتمال تعلق آن به محتوا مشخص شود.

خروجی پس‌افکنش هیستوگرام، یک تصویر خاکستری از احتمال‌ها است:

- پیکسل‌های سفید (مقدار ۲۵۵) نشان‌دهنده احتمال ۱۰۰ درصد تعلق به محتوا هستند.
- پیکسل‌های مشکی (مقدار ۰) نشان‌دهنده احتمال ۰ درصد تعلق هستند.

به این ترتیب می‌توان مناطق تصویر که بیشترین شباهت به نمونه دارند را تشخیص داد و برای پردازش‌های بعدی مانند شناسایی، جداسازی یا فیلتر کردن آن محتوا استفاده کرد.

تصویر پس افکنش هیستوگرام پوست انسان را روی یک عکس

برای پس افکنش یک هیستوگرام از تابع calcBackProject() استفاده میکینم.

calcBackProject([imag] , [channel numbers] , hist ,[ranges] , scale)

- imag : تصویری که میخواهیم بررسی کینم
- Channel number : شماره کانالی مورد نظر از تصویر برای پس افکنش. ۰ (آبی یا خاکستری یا hue) ، ۱ (سبز یا saturation) ، ۲(قرمز یا vale) برای فضاهای رنگی RGB ، gray و HSV باشد.

- Hist : هیستوگرامی که میخواهیم بر اساس آن پس افکنش کنیم به عبارت دیگر، هیستوکرامی ناحیه ای که میخواهیم در تصویر جست و جو کنیم و احتمال تعلق هر پیکسل به آن را بدست آوریم.
- Ranges : بازه ی پیکسل هایی که میخواهیم بررسی کنیم
- Scale : یک عدد است که به عنوان ضریب در تصویر خروجی ضرب میشود تا در مواردی تصویر بصری بهتری تولید شود.

نکته: توصیه می‌شود تصویر نمونه که هیستوگرام آن محاسبه می‌شود و تصویر مورد بررسی که پس افکنش می‌شود؛ در فضای رنگی HSV مورد استفاده قرار گیرند تا با حذف پارامتر V ، تاثیرات روشنایی روی رنگ ها از بین برود.

پیدا کردن یک محتوا در یک تصویر:

۱- تبدیل عکس محتوا و عکس مورد بررسی به HSV
۲- بدست آوردن هیستوگرام یک نمونه از محتوا
۳- بهتر است در این مرحله هیستوگرام را توسط تابع ()normalize نرمال کنیم (مراجعه به مثال بعد)

normalize(input, output , min , max , thresh type)

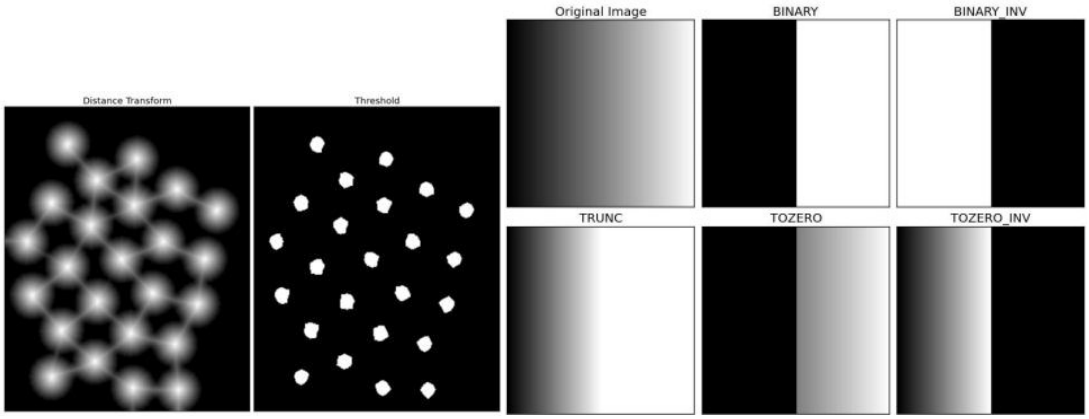
۴- فراکنش هیستوگرام روی تصویر و به دست آوردن تصویر احتمال

*مراحل۵ به بعد اضافی می‌باشد و برای نتیجه بهتر ارایه شده‌اند.

۵- ترشولد کردن تصویر احتمال برای بدست آوردن تصویر بصری بهتر. با استفاده از تابع ()Threshold

Threshold(imag , tresh , max , thresh type)

Imag : تصویر ورودی
Thresh: مقداری که اعداد بیشتر آن به بالا و اعداد کمتر به پایین منتقل می‌شوند.
Max: ماکزیزمم مقداری عددی برای thresh شدن.
Thresh thpy : روش مورد نظر برای ترش کردن که در شکل زیر آموده شده اند.

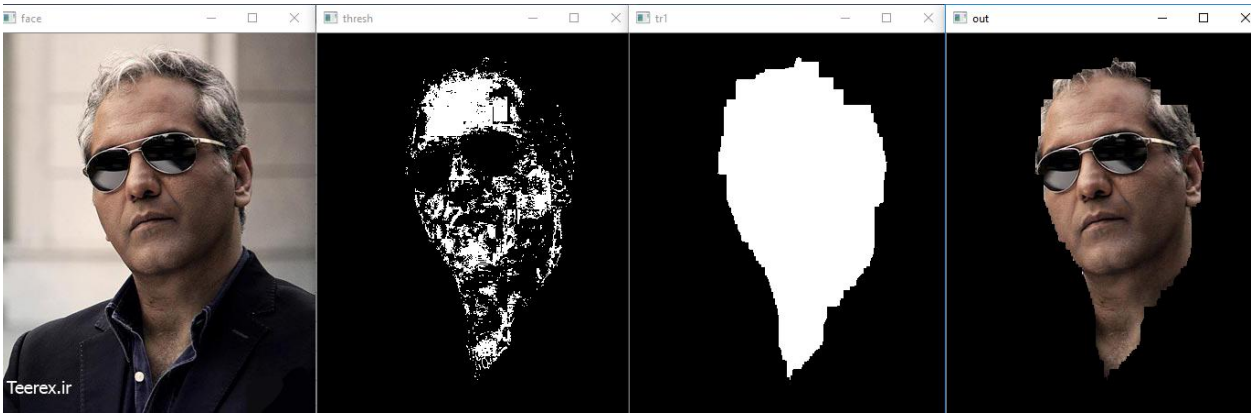


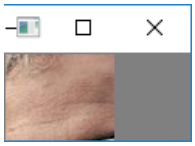
یک نمونه**threshold**

۶- ساخت یک تصویر سه بعدی (آرایه سه بعدی) و and کردن آن با تصویر اصلی برای حذف بخش های سیاه. با استفاده از تابع bitwise_and()

در مثال زیر ما هیستوگرام یک عکس نمونه از پوست را بدست آورده و تصویر احتمال صورت را بدست می‌آوریم.

```
import cv2
import numpy as np

face=cv2.imread ("face.jpg")
skin = cv2.imread ("skin.png")
skinhsv=cv2.cvtColor (skin,cv2.COLOR_BGR2HSV )
facehsv=cv2.cvtColor (face,cv2.COLOR_BGR2HSV )


hist=cv2.calcHist ([skinhsv],[0,1], None , [180,256],[0,180,0,256])
cv2.normalize (hist,hist,0,255 ,cv2.NORM_MINMAX )
bp=cv2.calcBackProject ([facehsv],[0,1], hist,[0,180,0,255],1)

cv2.imshow("face", face)
cv2.imshow("skin", skin)
cv2.imshow("back prj", bp)

_,tr=cv2.threshold(bp, 10,255,cv2.THRESH_BINARY )
cv2.imshow("thresh", tr)

kernel1=np.ones((7,7), np.uint8 )
kernel2=np.ones((3,3), np.uint8 )
tr1=cv2.morphologyEx (tr,cv2.MORPH_OPEN, kernel2 , iterations=1)
tr1=cv2.morphologyEx (tr,cv2.MORPH_CLOSE, kernel1 , iterations=3)
cv2.imshow("tr1", tr1)

tr1=cv2.merge ( (tr1,tr1,tr1))
out=cv2.bitwise_and (face , tr1)

cv2.imshow("out", out)
```





مقایسه محتوای دو تصویر با هیستوگرام

برای مقایسه محتوای دو تصویر، می‌توان از مقایسه هیستوگرام آن دو تصویر بهره بود. هرچه هیستوگرام دو تصویر به هم نزدیک تر باشند. مثلا هیستوگرام تصویر دو صورت انسان بسیار شبیه تر است تا هیستوگرام صورت یک انسان و یک ببر. برای مقایسه هیستوگرام دو تصویر از تابع comparHist() استفاده می‌شود. خروجی این تابع یک عدد است . هرچه این عدد بزرگتر باشد، دو هیستوگرام و دو تصویر به هم شبیه ترند و مقدار صفر به معنای عدم هیچگونه شباهت است:

compareHis (Hist۱ , Hist ۲ , Compare method)

Hist۱,Hist۲ : هیستوگرام هایی که قرارد باهم مقایسه شوند.

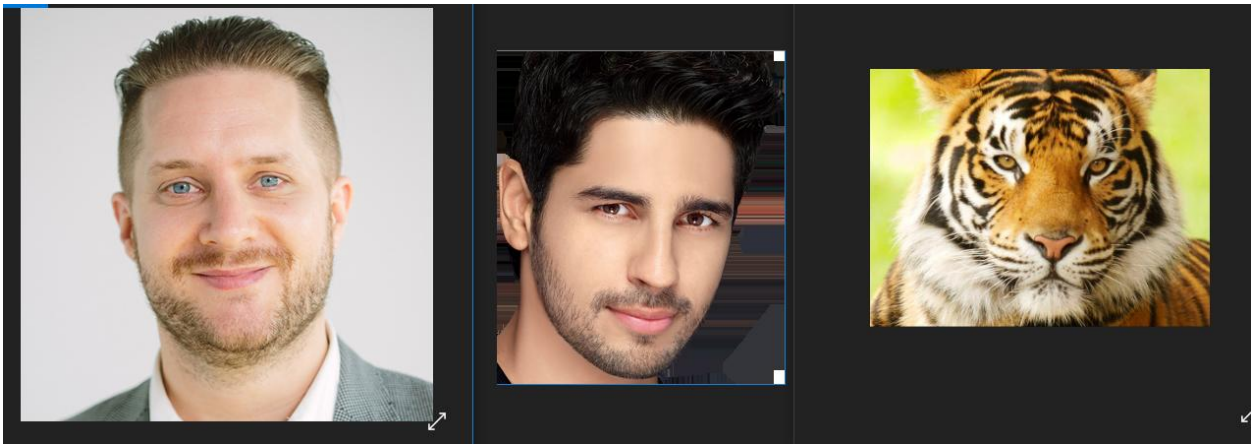
Compare method : روش مقایسه دو هیستوگرام که به صورت های زیر است:

۱- HISTCMP_CHISQR : روشی که تفاضل مربعی بین bin (میله‌های نموار میله‌ای) ها را بدست می‌آورد
۲- HISTCMP _CORREL : روشی مبتی بر عملگر همبستگی نرمال که از آن برای اندازه گیری شباهت بین دو سیگنال استفاده می‌شود.
۳- HISTCMP _BHATTACHARYYA: برای تخمین شباهت بین دو توزیع احتمال مورد استفاده قرار می‌گیرد
۴- HISTCMP _INTERSECT : مبتی بر تقاطع دو هیستوگرام
۵- HISTCMP _HELLINGER: همانند روش شماره ۳ است.

تذکر:کلیه این روش ها مقایسه را bin به bin انجام می‌دهند

مثال:

```
Import cv2

face1=cv2.imread("face1.JPG")
face2=cv2.imread("face2.JPG")
face3=cv2.imread("face3.JPG")

hsv1=cv2.cvtColor (face1,cv2.COLOR_BGR2HSV)
hsv2=cv2.cvtColor (face2,cv2.COLOR_BGR2HSV)
hsv3=cv2.cvtColor (face3,cv2.COLOR_BGR2HSV)

hist1=cv2.calcHist([hsv1],[0,1] , None , [180,255],[0,180,0,255],1)
hist2=cv2.calcHist([hsv2],[0,1] , None , [180,255],[0,180,0,255],1)
hist3=cv2.calcHist([hsv3],[0,1] , None , [180,255],[0,180,0,255],1)

com12=cv2.compareHist(hist1 , hist2 , cv2.HISTCMP_INTERSECT)
com13=cv2.compareHist(hist1 , hist3 , cv2.HISTCMP_INTERSECT)
com23=cv2.compareHist(hist2 , hist3 , cv2.HISTCMP_INTERSECT)

cv2.imshow("face1",face1)
cv2.imshow("face2",face2)
cv2.imshow("face3",face3)
```



تصاویر به ترتیب از سمت چپ به راست face۳ , face۲ , face۱

```
================ RESTART: C:\Users\Amir_PC\Desktop\opencv.py ================
>>> com12
51845.0
>>> com13
36233.0
>>> com23
8255.0
>>>
```

همان گونه که از نتایج پیداست بیشترین شباهت میان تصاویر ۱ و ۲ است که هردو صورت انسان را تشکیل می‌دهند و تصویر ۱ و۳ که یکی متعلق به ببر و دیگری متعلق به انسان است ، کمترین شباهت را دارند.

در این مثال در صورتی که با بر افکنش هیستوگرام پوست انسان و با الگوریتم meamShift بخشی از هر تصویر که بیشترین شباهت به صورت را دارند جدا کرده و آن بخش ها را با هم مقایسه کنیم، نتایج بهتری را شاهد خواهیم بود. در این مثال با تست متد HISTCMP_CORREL، نتایج زیر دریافت شد. که قابل قبول تر است

```
================ RESTART: C:\Users\Amir_PC\Desktop\opencv.py ================
>>> com12
0.2178886735836143
>>> com13
0.0018759967839581284
>>> com23
-0.0011413661431428628
>>>
```

تطابق الگو (TempLate Machining)

هدف از تطابق الگو، پیدا کردن یک تصویر الگو یا object در یک تصویر می‌باشد. مثلا پیدا کردن صورت در یک تصویر. تابع matchTemplate() در openCV برای همین منظور ایجاد شده‌است. در این تابع تصویر الگو روی تصویر اصلی حرکت می‌دهد و در هر مرحله، میزان شباهت تصویر الگو با بخشی از تصویر که روی آن قرار دارد را بررسی می‌کند. خروجی این تابع یک تصویر خاکستری است که هر پیکسل آن میزان شباهت پیکسل های محدوده خود را نشان می‌دهد. اگر ابعاد تصویر اصلی W,H و ابعاد تصویر الگو w,h باشد، ابعاد تصویر خروجی (W-w+۱ , H-h+۱) می‌باشد. در نهایت با تابع minMaxLoc() میتوان پیکسلی که بیشترین شدت را دارد پیدا نموده و به مرکزیت آن، یک مستطیل با ابعاد w,h رسم نمود. لازم به ذکر است که اندازه تصویر template شی مورد نظر باید مطابق با اندازه همان شی در تصویر اصلی باشد.

matchTemplate (img , templ , method)

img : تصویر ورودی که می‌خواهیم عملیات جست و جو را روی آن انجام دهیم

Templ : تصویر الگویی که میخواهیم آن را در تصویر اصلی جست و جو کنیم، تصویر شی مورد نظر

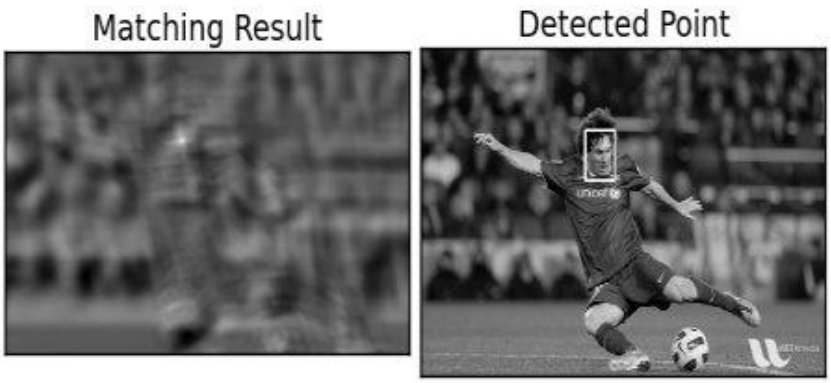
Method : روش جست جو که یکی از گزینه های زیر میتواند باشد :

۱- پرچم های TM_CCOEFF, TM_CCOEFF_NORMED, TM_CCORR, YM_CCORR_NORMED که در تصویر خروجی آن‌ها، پیکسل هایی که بیشترین شدت را دارند (نزدیک به سفید هستند) محل هدف است
۲- پرچم‌های TM_SQDIFF, TM_SQDIFF_NORMED که در تصویر خروجی آن‌ها، پیکسل هایی که کمترین شدت (نزدیک به سیاه هستند) را دارند محل هدف هستند

مثال :

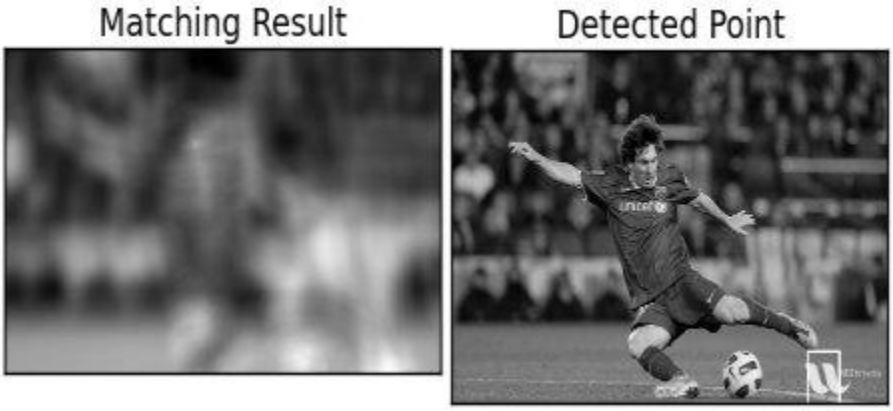
```
import cv2

img = cv2.imread(messi.jpg',0)
img2 = img.copy()
template = cv2.imread('templ.jpg',0)
w, h = template.shape[::-1]

methods = ['cv2.TM_CCOEFF', 'cv2.TM_CCOEFF_NORMED', 'cv2.TM_CCORR',
            'cv2.TM_CCORR_NORMED', 'cv2.TM_SQDIFF', 'cv2.TM_SQDIFF_NORMED']

for meth in methods:
    img = img2.copy()
    method = eval(meth)
    res = cv2.matchTemplate(img,template,method)
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
    if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
        top_left = min_loc
    else:
        top_left = max_loc
    bottom_right = (top_left[0] + w, top_left[1] + h)

    cv2.rectangle(img,top_left, bottom_right, 255, 2)
    cv2.imshow (meth , img)
    cv2.imshow (meth + " res" , res)
```

: TM_CCOEFF



: TM_CCOEFF_NORMED

: TM_CCORR



: TM_CORR_NORMED

: TM_SQDIFF

: TM_SQDIFF_NORMED

در مثال بالا تنها یک عدد از سوژه مورد نظر ما در تصویر وجود داشت اما اگر از سوژه ما چند عدد در تصویر باشد و ما بخواهیم همه آن‌ها را پیدا کنیم، استفاده از تابع minMaxLoc مناسب نخواهد بود و از تابع np.where() استفاده میکنیم. با توجه به مثال زیر، این تابع پیکسل هایی که مقدارشان از ۰.۰۵ کمتر است را پیدا میکند (دقت داشته باشید با توجه به متد TM_SQDIFF_NORMED مقدار پیکسل های تصویر خروجی بین ۰و۱ است).

مثال :

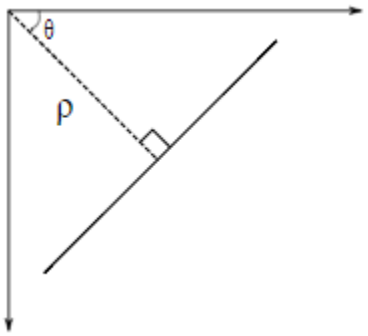
```
import cv2
import numpy as np

img=cv2.imread ( "image.jpg")
tmpl=cv2.imread ("template.jpg")

img_gray=cv2.cvtColor ( img , cv2.COLOR_BGR2GRAY )
tmpl_gray=cv2.cvtColor ( tmpl , cv2.COLOR_BGR2GRAY )

w,h,_=tmpl .shape

res=cv2.matchTemplate( img_gray , tmpl_gray , cv2.TM_SQDIFF_NORMED  )

loc=np.where ( res<0.05)
cv2.imshow ("t" , res )

for px in zip(*loc):

    y,x=px
    px=x,y
    cv2.rectangle( img , px , (px[0]+h , px[1]+w) , (0,180,200), thickness=3)

cv2.imshow ("result" , img)
```

تشخیص دایره و خط

تشخیص خطوط

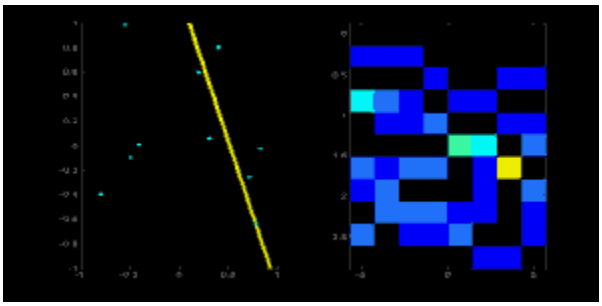
تبدیل هاف الگوریتمی است برای تشخیص اشکال در تصویر. اشکالی در تبدیل یاف یافت می‌شوند که دارای معادلات ریاضی باشند. در این بخش به توضیح تبدیل هاف برای یافتن خطوط می‌پردازیم

Hough Line Transform : در این تبدیل خطوط به فرم $p=x\cos\Theta+y\sin\Theta$ نمایش داده می‌شوند. پارامتر p فاصله عمودی خط از مبدا و پارامتر Θ زاویه است که خط گذرنده از مبدا و عمود بر خط مورد نظر، با محور افقی تشکیل می‌دهد. مبدا در تصویر گوشه بالا سمت چپ می‌باشد



در یک تصویر مقدار Θ بین ۰ تا ۱۸۰ و اندازه p از ۰ تا نهایت قطر تصویر می‌تواند باشد. اگر خط واسط بین مبدا و خط مورد نظر، در زیر مبدا قرار بگیرد، مقدار p یک عدد مثبت و مقدار Θ بین ۰ تا ۱۸۰ می‌باشد اما اگر خط واسط در بالا مبدا قرار گیرد، مقدار p یک عدد منفی و مقدار Θ باز هم بین ۰ تا ۱۸۰ خواهد بود

نحوره کار این الگوریتم به این صورت است که ابتدا یک آرایه دو بعدی ایجاد میکند که یک بعد آن نشانگر زاویه بودی و نهایتا طولی معادل ۱۸۰ دارد و بعد دیگر نشانگر پارامتر p بوده که طولش نهایتا به اندازه قطر تصویر می‌باشد. در نتیجه هر درایه این آرایه نشانگر یک خط است، سپس تابع به ازای هر نقطه بررسی میکند که چه خطوطی از آن نقطه عبور میکنند و به هر خطی که از آن نقطه بگذرد یک امتیاز می‌دهد و اینکار نقطه به نقطه انجام می‌شود در نهایت هر خط به اندازه نقاطی که از آنها عبور میکند دارای امتیاز و اهمیت است. در آخر تابع خطوطی را که دارای امتیاز پایینی هستند را حذف، و سایر خطوط را به عنوان خط در نظر می‌گیرد.



تابع یافتن خطوط در تصویر، تابع HoughLines() می‌باشد

HoughLines(img , rho , theta , threshold) -> lines

Img : تصویر ورودی که بایستی باینری باشد

rho : دقت یا رزولوشن فاصله از مبدا

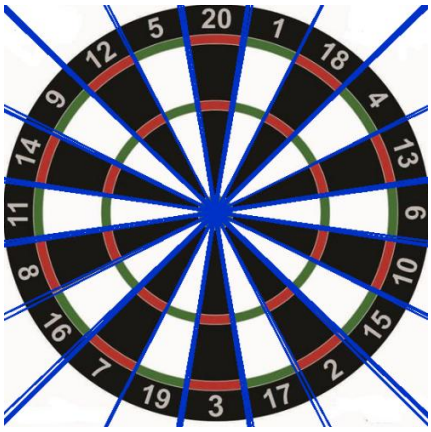
Theta : دقت یا رزولوشن زاویه خط عمود

Threshold : این پارامتر حداقل امتیاز خط برای آنکه یک خط در نظر گفته شود را مشخص می‌کند. از نگاهی دیگر این پارامتر حداقل طول یک خط را مشخص میکند زیرا امتیاز یک خط همان تعداد پیکسل هایی است که از آنها میگذرد

خروجی این تابع یک لیست از دوتایی (rho , theta) می‌باشد که rho بر حسب پیکسل و theta بر حسب رادیان است

```
import cv2
import numpy as np

img=cv2.imread ("img.jpeg")
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
gray=cv2.blur( gray , (3,3) )
edges = cv2.Canny(gray,50,150,apertureSize = 3)
lines=cv2.HoughLines (edges , 1 , np.pi/180 , 110)

for line in lines:
    rho,theta = line[0]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    cv2.line(img,(x1,y1),(x2,y2),(200,50,0),2)
cv2.imshow ( "result" , img)
```

مثال :



تشخیص دوایر در تصویر

Probabilidtic Hough Line یا تبدیل هاف احتمالی، بهینه شده تبدیل هاف معمولی است که در آن به جای بررسی همه نقاط، بخشی از آنها را به‌صورت رندوم انتخاب و بررسی می‌کند. از طرفی هنگامی که امتیاز یک خط به مقدار قابل قبولی رسید، کلیه خطوط گذرنده از آن، از تصویر حذف می‌شوند. با توجه به موارد گفته شده، واضح است که حجم محاسبات در تبدیل هاف احتمالی کمتر است. لازم به ذکر است از آن جا که در تبدیل هاف احتمالی، همه نقاط بررسی نمی‌شوند، امتیاز خطوط نسبت به تبدیل هاف معمولی کمتر است و این موضوع باید در پارامتر دهی تابع مربوطه درنظر گرفته شود. تابع تشخیص خط با تبدیل هاف احتمالی HoughLinesP() می‌باشد.

HoughLinesP (img , rho , theta ,threshold [, Lines [, minLineLength [, maxLineGap]]]) -> Lines

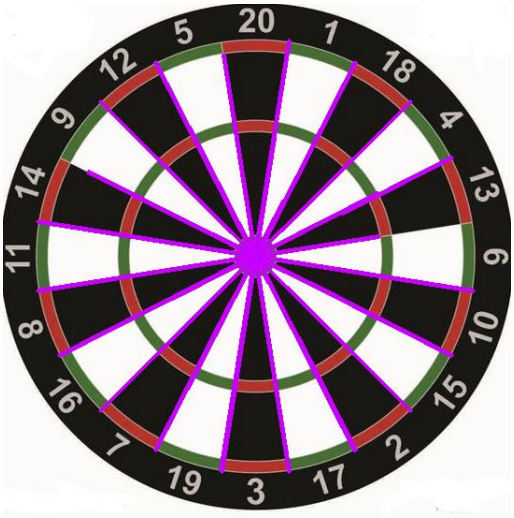
چهار پارامتر اول مشابه تبدیل هاف معمولی می‌باشد

Min Line Lenght : حداقل طول خط را بر حسب پیکسل مشخص می‌کند

Max Line Gap : حداکثر فاصله قابل قبول بین نقاط یک خط را مشخص می‌کند.

خروجی این تابع مختصات نقاط ابتدا و انتهایی خط می‌باشد (x۱,y۱,x۲,y۲)

```
img=cv2.imread ("img.jpeg")
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
gray=cv2.blur( gray , (3,3) )
edges = cv2.Canny(gray,50,150,apertureSize = 3)
lines=cv2.HoughLinesP ( edges , 1 , np.pi/180 , 50 , minLineLength=110 ,
maxLineGap=15)

for line in lines:
    x1,y1,x2,y2=line[0]
    cv2.line(img,(x1,y1),(x2,y2),(255,0,200),3)
cv2.imshow ( "HoghP result" , img)
```



تشخیص دوایر با Hough Circle : دوایر را با معرفی مرکز و شعاع دایره پیدا می‌کند. پس برای هر دایره سه پارامتر وجود دارد. تابع تشخیص دوایر ()HoughCircles است

HoughCircles (img , method , dp , minDist [, circles [, param۱ [, param۲ [,min R [, max R]]]]])

- Img : تصویر باینری ورودی
- Method : روش جست و جو را مشخص میکند و در حال حاضر تنها روش تعریف شده HOUGH_GRADIENT می‌باشد
- Dp : رزولوشن accumulator را مشخص میکند. برای مقدار ۱ به معنی برابر بودن رزولوشن آکومولاتور با تصویر اصلی است.
- minDist : حداقل فاصله قابل قبول بین مراکز دایره های شناسایی شده.
- Param۱ : این تابع در حالت HOUGH_GRADIENT دارای عملگر canny داخلی است و مقدار این پارامتر درواقع همان حد آستانه بالا برای عملگر canny است. حد آستانه پایین نیز به طور پیشفرض، نصف حد آستانه بالا است.
- Param۲ : مشابه پارامتر حداقل امتیاز در تابع ()HoighLine است
- Min R / Max R : به ترتیب حداقل و حداکثر مقدار شعاع را مشخص می‌کنند

```
img=cv2.imread ("img.jpeg")
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY )
gray=cv2.GaussianBlur ( gray , (9,9) , 0)
circles = cv2.HoughCircles(gray,cv2.HOUGH_GRADIENT,1,8,
param1=150,param2=150,minRadius=50,maxRadius=230)

circles = np.uint16(np.around(circles))
for circ in circles:
     x,y,r=circ[0]
     cv2.circle(img,(x,y),r,(200,0,0),5)
     cv2.circle(img,(x,y),2,(0,0,255),5)

cv2.imshow('detected circles',img)
```

مثال :

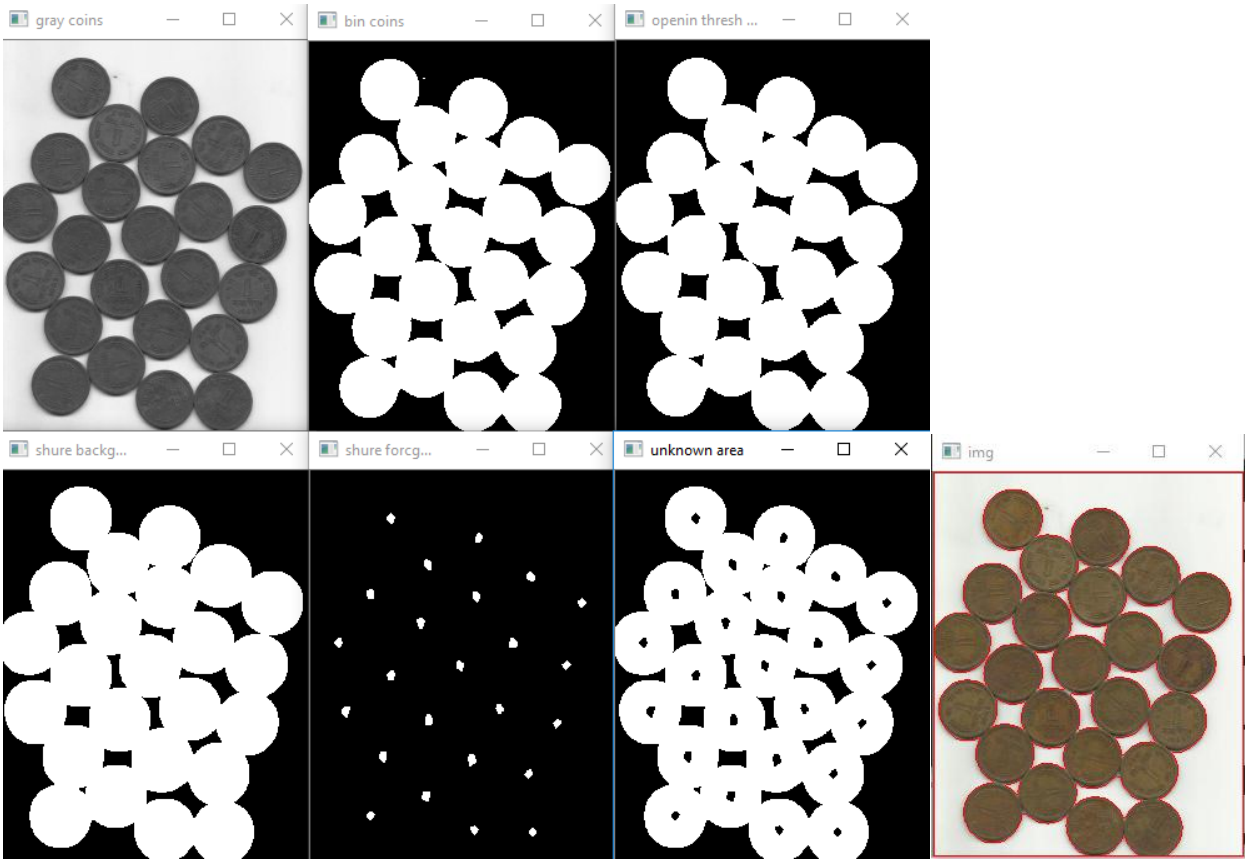
الگوریتم آبگیر یا watershed

این الگوریتم برای جدا سازی نواحی مختلف همگن از یک‌دیگر است. برای مثال در تصویر سکه روی یک میز، ناحیه های همگن مختلف که در این‌جا سکه و میز است را قطعه بندی می‌کند. این الگوریتم بر اساس ریخت‌شناسی است. این الگوریتم بر پایه یک تصویر marker کار می‌کند. تصویر مارکر یک تصویر 32 بیتی علامت دار و هم انداز ه تصویر اصلی می‌باشد. در این تصویر به پیکسل هایی که مطمئنیم به یک ناحیه تعلق دارند یک مقدار یکسان غیر صفر (مثلا 2000) میدهیم و برای نواحی مختلف ، مقادیر مختلف در نظر می‌گیریم. مثلا در مثال بالا به نواحی که مطمئنیم متغلق به سکه هستند مقدار 255 و به نواحی که مطمئنیم متعلق به میز هستند مقدار 127 میدهیم. سپس با دادن مارکر و تصویر اصلی به این الگوریتم، این الگوریتم درباره سایر نواحی تصمیم گیری میکند که متعلق به کدام ناحیه هستند. تابع این الگوریتم watershed() است. خروجی این تابع مارکر جدیدی است که همه نواحی را پوشش می‌دهد و همچنین به لبه‌ها برچسب -1 اختصاص می‌دهد

Watershed(img , markers)

مثال:

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

img = cv2.imread('coins.jpg')

gray = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY )
_,thresh = cv2.threshold(gray,150,255,cv2.THRESH_OTSU + cv2.THRESH_BINARY_INV )

cv2.imshow ("gray coins" , gray)
cv2.imshow ("bin coins" , thresh)

# noise removal
kernel = np.ones((3,3),np.uint8)
opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 2)

cv2.imshow ("openin thresh coins" , opening )

# sure background area
sure_bg = cv2.dilate(opening,kernel,iterations=1)
cv2.imshow ("shure background" , sure_bg)

# Finding sure foreground area
kernel =cv2.getStructuringElement (cv2.MORPH_ELLIPSE ,(9,9) )
sure_fg = cv2.erode (opening,kernel,iterations=5)
cv2.imshow ("shure forcground" , sure_fg)

# Finding unknown region
unknown = cv2.subtract(sure_bg,sure_fg)
cv2.imshow ("unknown area" , unknown)

#markers=np.zeros ((312,252),np.int32)
#markers [sure_fg==255]=5
ret, markers = cv2.connectedComponents(sure_fg)
markers = markers+1
markers[unknown==255] = 0

markers = cv2.watershed(img ,markers)
img[markers == -1] = [255,0,0]
cv2.imshow ("img " , img)
```

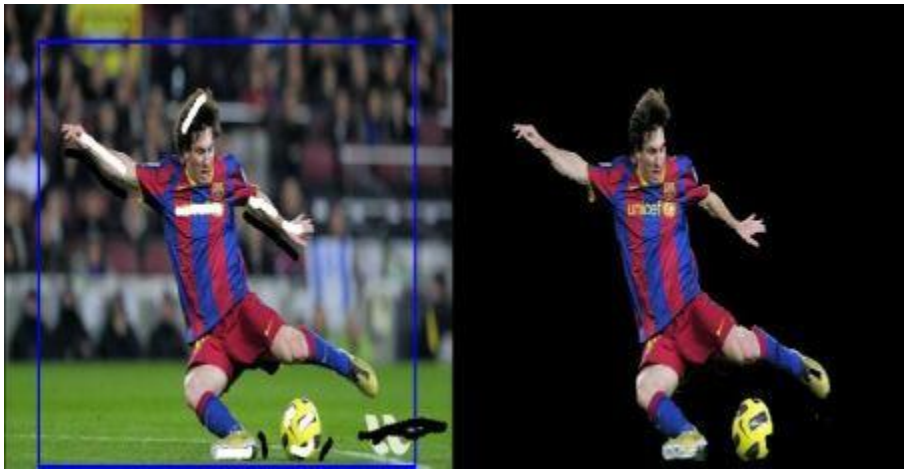


در مثال بالا ابتدا تصویر را به فضای رنگ خاکستری برده‌ایم. سپس یک threshold روی آن اعمال کردیم تا یک تصویر باینری بدست آید. سپس برای رفع نویز احتمالی، یک عملگر opening روی آن اعمال کردیم. سپس برای بدست آوردن ناحیه پس‌زمینه و اطمینان از اینکه هیچ ناحیه‌ای از سکه ها، جز پس زمینه نباشند، با عکلگر dilate آن را گسترش دادیم. سپس برای بدست آوردن ناحیه سکه ها و اطمینان از اینکه هیچ ناحیه‌ای از پس زمینه، جز سکه ها نباشد به کمک عملگر erode آنرا فرسایش دادیم. سپس برای بدست آوردن ناحیه نامعلوم، این دو تصویر را از هم کم کردیم. حال نوبت به تعریف marker می‌رسد. در اینجا تابع connectedCoponents() بر اساس ورودی آن که تصویر sure_fg است یک مارکر تعریف می‌کند. خروجی دیگر این تابع که ret است، تعداد نواحی را بر میگرداند. (در اینجا می‌توان به جای تعریف تابع یک آرایه تعریف کرد و درایه های آن‌را مقدار دهی کرد که این دوخط در بالای فراخوانی تابع، به صورت کامنت درآمده‌اند) . از طرفی میدانیم پیکسل هایی که برچسبشان صفر است ، ناحیه نامعلوم به حساب می‌آیند. از این رو کل markers را با مقدار 1 جمع میکنیم و پیکسل هایی که در تصویر unknow میدانیم جز ناحیه نامعلوم هستند. مقدار برچسبشان را صفر می‌کنیم. و سپس الگوریتم آبگیر را اجرا میکنیم. در نهایت پیکسل هایی از تصویر که برچسبشان 1- است(لبه‌ها) را قرمز میکنیم تا نواحی معلوم گردند.

الگوریتم Grab Cut

هدف از این الگوریتم جداسازی تصویر پیش زمینه از پس زمینه می‌باشد. روش کارکرد این الگوریتم به این صورت است که در ابتدا یک مستطیل اطراف تصویر پیش زمینه رسم می‌کنیم تا تصویر پیش زمینه به طور کامل درون آن قرار گیرد. سپس با اجرای الگوریتم با یک تعداد تکرار معین تصویر پیش زمینه شناسایی می‌گردد. اما در برخی موارد ممکن است که بخش هایی از پس زمینه به عنوان پیش زمینه انتخاب شده‌باشند یا بر عکس. در این موارد کاربر باید یک تصویر ماسک ایجاد کند و در آن بخش‌هایی از نواحی که باید به عنوان پس زمینه تشخیص داده می‌شدند اما به عنوان پیش زمینه شناسایی شدند را با رنگ مشکی و بخش‌هایی از نواحی که به عنوان پیش زمینه باید تشخیص داده می‌شدند اما به عنوان پس زمینه شناسایی شدند را با پیکسل سفید نشانه گذاری نماید تا الگوریتم آن نواحی را تصحیح نماید.

برای درک بهتر به تصویر زیر توجه نمایید که در تصویر سمت چپ با رسم یک مستطیل اطراف پیش‌زمینه و نشانه گذاری در نواحی که دچار تشخیص اشتباه شده‌اند؛ تصویر راست را خروجی می‌دهد که در آن پیش زمینه به طور کامل جدا شده است.



تابع پیاده سازی این الگوریتم grabCut() است.

grabCut (imag , mask , rect, bgd model , fgd mode, itration , type)

image : تصویر ورودی

Mask : ماسکی که در آن مشخص میکنیم کدام نواحی به طور حتم جزو پیش‌زمینه یا پس‌زمینه هستند و یا نواحی که احتمالا جزو پیش زمینه یا پس زمینه هستند و... . که این موارد را با پرچم های زیر و یا اعداد 0و1و2و3 مشخص می‌کنیم.

cv.GC_BGD (0) , cv.GC_FGD (1), cv.GC_PR_BGD (2) , cv.GC_PR_FGD (3)

rect : مستطیلی که دور پیش‌زمینه رسم می‌شود و با فرمت (x,y,w,h) می‌باشد.

Bgd model-fgd model: دو آرایه که در درون الگوریتم مورد استفاده قرار می‌گیرند و شما آنهارا به طور پیشفرض با نوع np.float64 و ابعاد (1,65) و درایه ای صفر، تعریف کنید.

Iteration : تعداد تکرار الگوریتم را مشخص می‌نماید

Type : این پارامتر مشخص می‌کند که اجرای این الگوریتم مبتنی بر قاب مستطیلی، ماسک نشانه گذاری شده یا هردو باشد. این موارد با پرچم‌های زیر مشخص می‌شود.

cv.GC_INIT_WITH_RECT , cv.GC_INIT_WITH_MASK , Or Both of them

این تابع پس از اجرا mask ورودی را به عنوان خروجی اصلاح می‌کند و در ماسک خروجی به هر پیکسل یک عدد با پرچم‌هایی که پیش از در پارامتر mask توضیح داده شده است می‌دهد

مثال: در مثال زیر ما ابتدا یک مستطیل اطراف پیش‌زمینه ایجاد می‌کنیم به طوریکه پیش‌زمینه را به‌طور کامل در بر بگیرد. سپس ایم مستطیل را به عنوان پارامتر را به تابع می‌دهیم و پارامتر نوع عملکرد الگوریتم را مبتنی بر قاب مستطیلی می‌دهیم. سپس این الگوریتم mask ورودی را اصلاح میکند. سپس ما به پیکسل هایی که مقدار 0 و 2 دارند مقدار 0 (رنگ سیاه) و به پیکسل هایی که مقدار 1و3 دارند مقدار 255 (رنگ سفید) اختصاص می‌دهیم و یک ماسک برای تصویر پیش‌زمینه می‌سازیم و با and کردن این ماسک با تصویر اصلی، پیش‌زمینه را جدا می‌کنیم.

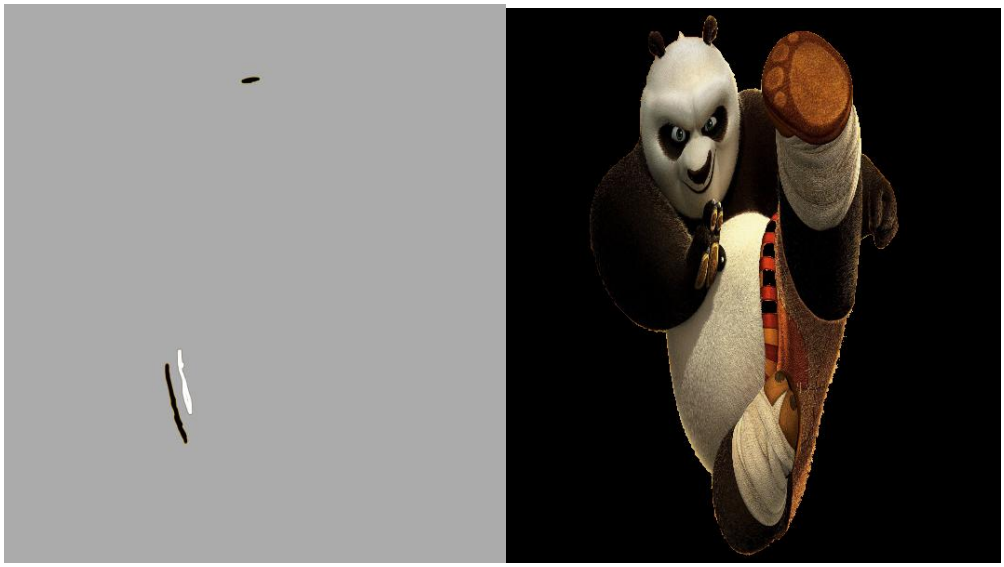
```
import cv2
import numpy as np

img=cv2.imread("panda1.jpg")

mask = np.zeros(img.shape[:2],np.uint8)
bgdModel = np.zeros((1,65),np.float64)
fgdModel = np.zeros((1,65),np.float64)
rect = (70,5 , 380 , 585)


img_rec=img.copy ()
img_rec=cv2.rectangle (img_rec, (70,5), (450,590) , (255,0,0))
cv2.imshow ("img whit rect", img_rec)


cv2.grabCut ( img , mask , rect , bgdModel , fgdModel , 5 , cv2.GC_INIT_WITH_RECT)


#mask2 = np.zeros(img.shape[:2],np.uint8)
#mask2[ mask== 1 ]=255
#mask2[ mask==3]=255
mask2 = np.where((mask==2)|(mask==0),0,255).astype('uint8')

img_cut=cv2.bitwise_and (img , img , mask=mask2)
cv2.imshow ("image Cut", img_cut)
```

در تصویر بالا نتیجه تا حد زیادی قابل قبول است اما در برخی قسمت ها تشخیص درست صورت نگرفته است و نیاز است که با استفاده از یک ماسک آن‌ها را اصلاح کنیم. ما برای ساخت ماسک از فتوشاپ کمک گرفته ایم. به این صورت که بخش هایی که متعلق به پیش زمیته بودند اما حذف شده اند را سفید، بخش های متعلق به پس زمینه که حذف نشده اند را سباه و سایری بخش ها را خاکستری میکنیم سپس با استفاده از این تصویر یک ماسک برای استفاده در grabCut() مطابق مثال زیر ایجاد میکنیم.

تذکر : در مثال بالا با افزایش تعداد تکرار تابع به ۱۰ باز، به نتیجه مطلوب رسیدیم که البته حجم پردازش بسیار زیاد گشت

```
import cv2
import numpy as np

img=cv2.imread("panda1.jpg")

mask = np.zeros(img.shape[:2],np.uint8)
bgdModel = np.zeros((1,65),np.float64)
fgdModel = np.zeros((1,65),np.float64)

rect = (70,5 , 380 , 585)

cv2.grabCut ( img , mask , rect , bgdModel , fgdModel , 5 , cv2.GC_INIT_WITH_RECT)

mask2 = np.where((mask==2)|(mask==0),0,255).astype('uint8')

img_cut=cv2.bitwise_and (img , img , mask=mask2)

#####################################################
#                         mask                      #
#####################################################
mask_img=cv2.imread("mask.jpg",0)

mask [ mask_img==255]=1
mask [ mask_img==0]=0

cv2.grabCut ( img , mask , None , bgdModel , fgdModel , 5 , cv2.GC_INIT_WITH_MASK )

mask2 = np.where((mask==2)|(mask==0),0,255).astype('uint8')

img_cut=cv2.bitwise_and (img , img , mask=mask2)
cv2.imshow ("image Cut by Mask", img_cut)
```

مثال کد مثال قبل را به صورت زیر تکمیل نمودیم. تصویر سمت چپ، تصویر ماسکی است که با طراحی کردیم:



درک ویژگی‌های یک تصویر و تطابق دو تصویر

درک ویژگی های یک تصویر

بسیاری از شما بازی پازل را انجام داده‌اید. در پازل ما قطعه‌های کوچک یک تصویر را به یکدیگر متصل میکنیم و یک تصویر واحد می‌سازیم. اما چگونه ما اینکار را انجام میدهیم. چگونه میتوان این تئوری را روی کامپیوتر پیاده سازی کرد و مثلا تعداد تصویر از یک منطقه طبیعی را به کامپیوتر بدهیم تا یک تصویر کامل و واحد از آن منظره ایجاد کند یا تعداد تصویر از یک ساختمان به کامپیوتر بدهیم و کامپیوتر تصویر ۳D از آنها بدست آورد. ما ابتدا باید به این سوال پاسخ دهیم که چگونه ما قطعه های یک پازل را بهم متصل می‌کنیم؟

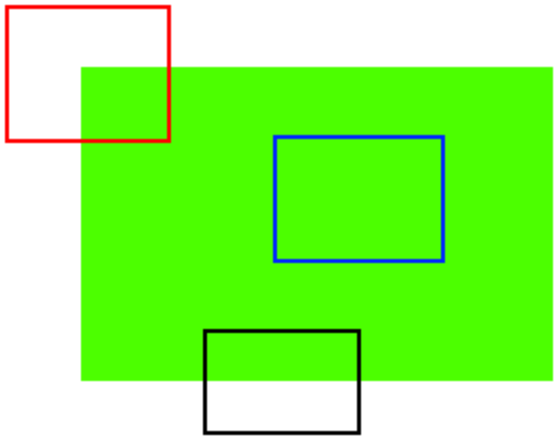
در پاسخ باید گفت که ما در تصویر به دنبال ویژگی ها و الگوهایی منحصر به فرد هستیم که به راحتی بتوان آنها را ردیابی و مقایسه کرد. ما به راحتی این ویژگی ها را درک میکنیم با اینکه ممکن است توصیف آنها در کلمات برایمان دشوار باشد اگر از ما بخواند تا در بین چند تصویر، ویژگی بیایم که بتوان به کمک آن، آنها را باهم مقایسه کرد، به راحتی اینکار را انجام میدهیم. در واقع ما در یک پازل ویژگی های یک قطعه را شناسایی میکنیم و ویژگی های مشابهی را در تصویر دیگر پیدا می‌کنیم و بر این اساس آن قطعات را در کنار هم قرار می‌دهیم. توضیح اینکه ما چگونه این ویژگی ها را شناسایی میکنیم سخت است زیر این ویژگی ذاتی ما است. اما اگر ما به طور عمیق به تعداد تصویر نگاه کنیم و برای آنها به دنبال الگو باشیم به نتیجه جالبی خواهیم رسید.

برای مثال به تصویر زیر نگاه کنید و سعی کنید محل دقیق ۶ تکه مشخص شده را در تصویر بیابید

قطعات A و B متعلق به یک سطح هستند و در نواحی زیادی گسترش یافته اند. از این رو پیدا کردن محل دقیق آنها مشکل است.

پیدا کردن قطعات C و D بسیار ساده است. این قطعات متعلق به لبه‌های ساختمان هستند اما باز هم پیدا کردن محل دقیق آنها مشکل است زیر در امتداد لبه این تکرار شده اند. پس آن‌ها را می‌توان به عنوان یک ویژگی خوب در نظر گرفت اما باز هم به اندازه کافی خوب نیستند

در نهایت قطعات E و F متعلق به گوشه ها هستند و به راحتی می‌توان محل دقیق آن ها را مشخص کرد. زیرا منحصر به فرد هستند. از این رو آنها را میتوان به عنوان یک ویژگی خوب در نظر گرفت .



به تصویر بالا نگاه کنید. قطعه آبی متعلق به سطح است و یافتن محل دقیق آن دشوار است. زیرا هرچه آنرا حرکت دهیم به نظر همان است. همچنین قطعه مشکی متعلق به لبه ها است و یافتن آن ساده تر است. این قطعه را اگر به صورت عمودی حرکت دهیم به راحتی تغیر را میتوان فهمید اما با این حال با حرکت افقی این قطعه تغییری حس نخواهد شد. اما قطعه قرمز متعلق به گوشه است و با هر حرکتی میتوان تغییر آن را متوجه شد و منحصر به فرد است. از این رو گوشه های یک تصویر ویژگی خوبی هستند. پس ما توانستیم ویژگی خوبی برای شناسایی بیابیم.

پس ما ویژگی های خوب را درک کردیم. بخش هایی از تصویر، نظیر گوشه‌ها که بتوان با کوچکترین حرکت، تغییر در آن‌ها را حس کرد. هنگامی که ما ویژگی های یک تصویر را درک کردیم باید به دنبال ویژگی های مشابه در تصاویر دیگر باشیم. در واقع ما یک ناحیه اطراف آن ویژگی را در نظر میگیریم و آن را توصیف میکنیم. مثلا " قسمت بالا آبی آسمان است، قسمت پایینی ساختمان است که دارای نمای شیشه ای بوده و... " و ما همان منطقه را در تصاویر دیگر جست و جو می‌کنیم. این همان کاری است که کامپیوتر باید انجام دهد. یعنی توصیف منطقه اطراف یک ویژگی و جست و جو آن در تصاویر دیگر. اینکار شرح ویژگی یا description feature نام دارد.

پس هدف از این بخش، یافتن ویژگی های یک تصویر، توصیف آنها، مقایسه آن ها در تصاویر دیگر و ... است. تا بر اساس آن بتوان کارهایی از قبیل یافتن یک شی در یک تصویر را نسبت به هم، تناظر یابی دو تصویر و... را انجام داد

تشخیص گوشه‌های تصویر با الگوریتم هریس

برای تشخیص گوشه از تشخیص گوشه هریس استفاده می‌شود. در این الگوریتم با حرکت یک قاب کوچک، میانگین شدت روشنایی پیکسل های درون آن اندازه گیری می‌شود. اگر این مقدار با حرکت جزیی، تغیرات شدید داشته باشد متعلق به لبه یا گوشه هستند. حال اگر این تغییرات در بیش از یک جهت(عمودی و افقی) شدید باشند، آن ناحیه متعلق به یک گوشه است. تابع تشخیص گوشه هریس cornerHarris() می‌باشد.

cornerHarris(img , block size , ksize , k) ->dst

img : تصویر ورودی که باید خاکستری بوده و نوع درایه های آن باید float۳۲ باشد

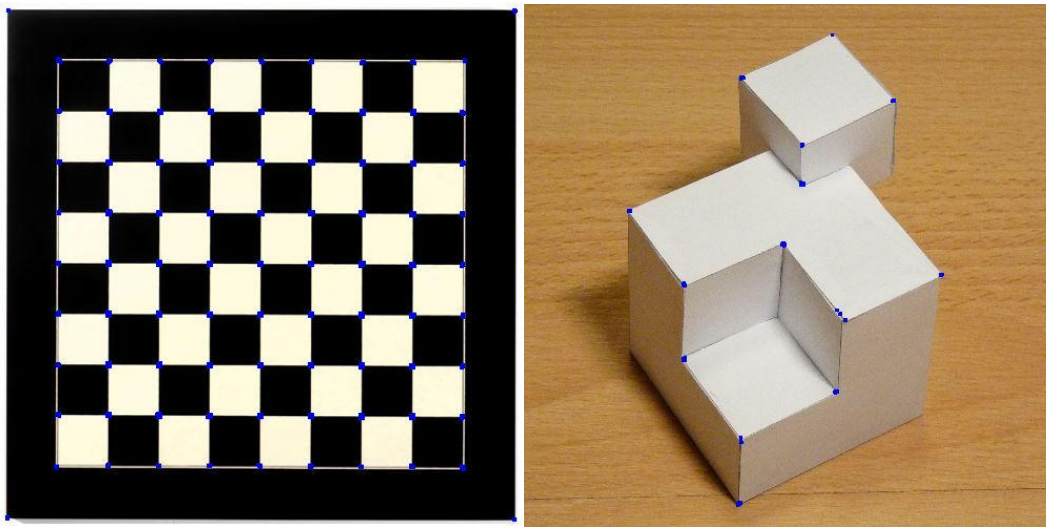
Block Size : محدوده پیکسل های همسایه گوشه را مشخص می‌کند

Ksize : پارامتری برای مشتقات سوبل

K : یک پارامتر آزاد در معادله تشخیص هریس . مقدار آن معمولا بین ۰ تا ۰.۵ است

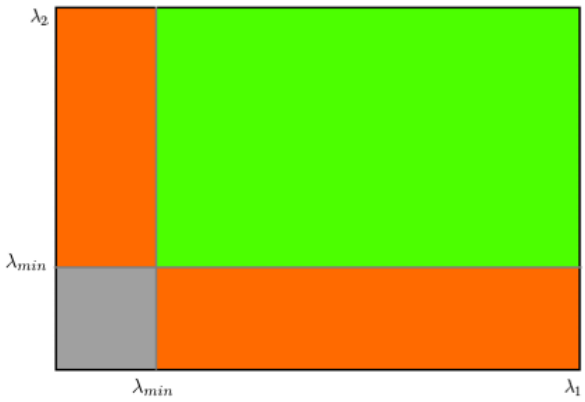
تصویر خروجی یک تصویر خاکستری از محل گوشه های تصویر ورودی است.

```
import cv2
import numpy as np

img=cv2.imread ( "chess.jpg")
gray=cv2.cvtColor (img , cv2.COLOR_BGR2GRAY)
gray= np.float32 (gray)
dst=cv2.cornerHarris( gray , 3 , 3, 0.04)

#for increas corner Area
dst = cv2.dilate(dst,None)

img[  dst > 0.1 * dst.max()] = [255,0,0]
cv2.imshow( " detect Corner " , img)
```



یافتن گوشه ها با الگوریتم shi_tomasi

این الگوریتم با ایجاد تغیراتی در الگوریتم هریس، به وجود آمده است و بهبود یافته الگوریتم هریس می‌باشد.



با توجه به تصویر بالا اگر landa۱ و landa۲ (تغیرات میانگین روشنایی با جابه جایی تصویر در جهت عمودی وافقی) هر دو ار landa min بیشتر شوند آن قسمت یک گوشه است (بخش سبز) و اگر تنها یکی :از آنها از لاندا مینیمم بیشتر باشد آن بخش متعلق به لبه است (بخش نارنجی) و اگر هر دو کوچک تر باشند آن بخش متعلق به یک سطح است (بخش خاکستری)

تابع مربوطه در open cv تابع goodFeaturesToTrack() است

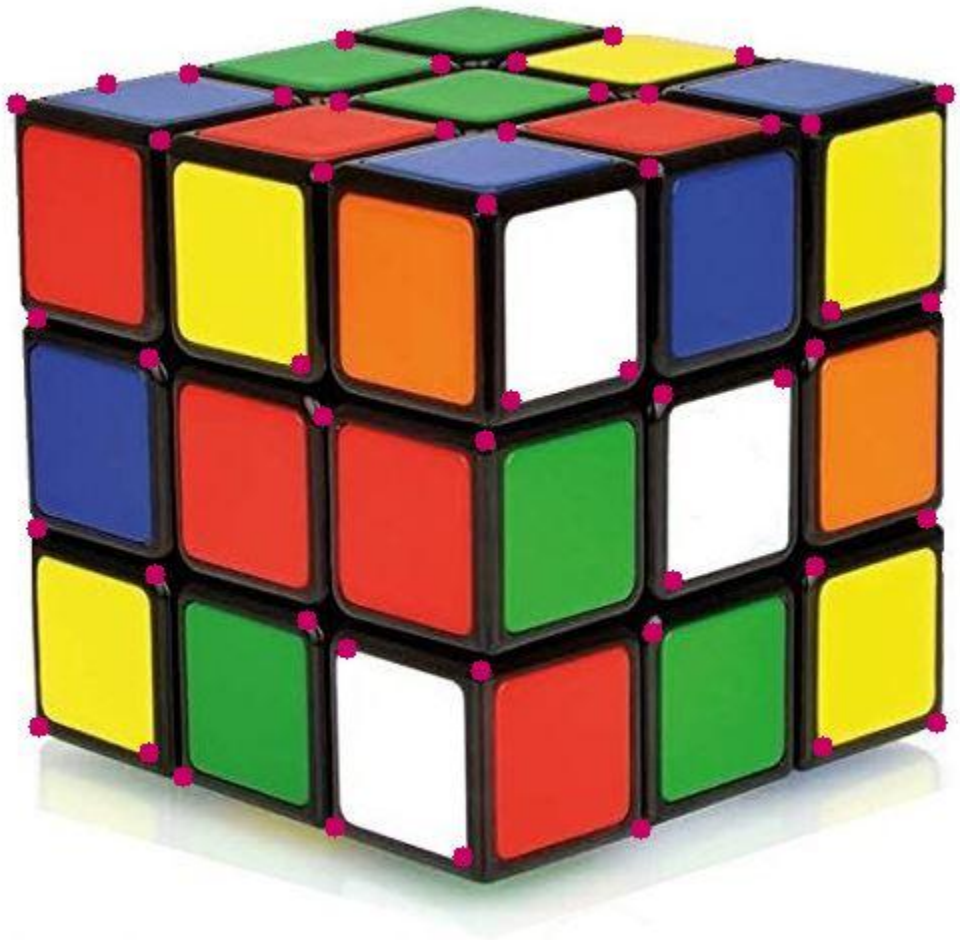
goodFeaturesToTrack(img ,max corners , qualityLevel , minDistance) -> cornerrs

- img : تصویر ورودی که باید خاکستری باشد و نوع آن ۸uint و یا ۳۲float باشد
- Max coners : حداکثر تعداد گوشه ها که تابع باز میگرداند را مشخص میکند. اگر تعداد گوشه های تصویر بیشتر از این مقدار باشد، تابع تنها گوشه های با کیفیت و اهمیت بالا را باز می‌گرداند
- qualityLevel : حداقل کیفیت قابل قبول برای آنکه یک نقطه به عنوان گوشه در نظر گرفته شود را مشخص می‌کند. این پارامتر عددی است بین ۰ و ۱ که این مقدار در ماکزیمم کیفیت موجود ضرب می‌شود و حداقل کیفیت قابل قبول را تولید می‌کند. برای مثال اگر ماکزیمم کیفیت ۲۰۰۰ و مقدار پارامتر۰.۰۱ باشد، نقاطی که کیفیتشان کمتر از ۲۰ باشد حذف می‌شوند.
- MinDistance : حداقل فاصله قابل قبول بین گوشه

خروجی این تابع لیستی از مختصات گوشه های تصویر است که هر سطر شامل یک آرایه با دو درایه x,y است

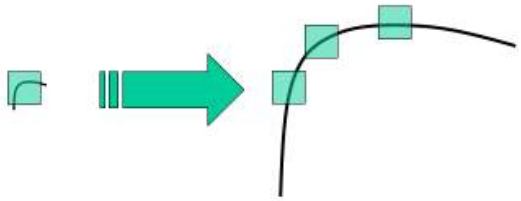
```
import cv2
import numpy as np


img=cv2.imread("cube.jpg")
gray=cv2.cvtColor( img , cv2.COLOR_BGR2GRAY)

#delete noise
gray =cv2.GaussianBlur (gray , (3,3) , 0)

corners=cv2.goodFeaturesToTrack( gray , 60 , 0.05 , 20 )

for corner in corners:
    x,y=corner[0]
    img2=cv2.circle ( img , (x,y) , 5 , (100,0,200), thickness=-1 )

cv2.imshow( "corners image " , img2)
```

مثال :



الگوریتم SIFT

در بخش قبل آموختیم که چگونه گوشه‌های یک تصویر را استخراج نماییم. هدف از یافتن گوشه ها بدست آوردن ویژگی از تصویر بود تا از طریق آن بتوانیم تصاویر مختلف را تطبیق دهیم و یا عملیات های دیگر. گوشه ها مستقل از چرخش تصویرند به این مفهوم که اگر یک تصویر را بچرخانیم باز همان گوشه‌هایی را خواهیم یافت که در تصویر بدون چرخش پیدا نموده ایم و این مطلوب ما است. اما ویژگی گوشه‌ها نسبت به تغییر ابعاد مستقل نیست و اگر روی یک تصویر بزرگ نمایی یا کوچک نمایی اعمال کنیم ممکن است گوشه‌های یافت شده نسبت به تصویر اصلی متفاوت باشند. برای درک بهتر این موضوع به تصویر زیر توجه فرمایید



در تصویر سمت چپ ما یک گوشه تشخیص میدهیم اما در تصویر سمت راست که بزرگ‌نمایی شده است، ما تنها لبه تشخیص میدهیم و گوشه یافت نمی‌شود زیرا لازم است تا ابعاد قاب جست و جو نیز بزرگنمایی شوند. تا ویژگی ها همسان درک شوند.

پس ما نیازمند الگوریتمی هستیم که ویژگی‌های یک تصویر یا دقیق تر بگوییم ، توصیف ویژگی های یک تصویر را مستقل از اندازه، چرخش، زاویه دید، روشنایی و... استخراج نماید. الگوریتم SIFT نیز به همین منظور طراحی شده است. کارکرد این الگوریتم استخراج ویژگی های تصویر می‌باشد که علاوه بر یافتن نقاط کلیدی یا همانkeypoint های تصویر، یک بردار توصیف یا Descriptor برای هر کدام از این نقاط تعریف میکند. که میتوان به کمک آن، نقاط کلیدی دو تصویر را یک به یک نسبت به هم بررسی نمود و اگر بردار توصیف آنها به هم نزدیک بود این دو نقطه را تطبیق داد. برای درک بهتر به تصویر زیر توجه فرمایید

در تصویر بالا نقاط کلیدی هر دو عکس بررسی شده و نقاطی را که با هم تطبیق داشتند (بردار توصیفشان تقریبا برابر بوده)، با یک خط به هم وصل شدند.

الگوریتم SIFT شامل ۵ مرحله به صورت زیر است

۱- پیدا کرد keypoint ها بر اساس مقیاس بندی های مختلف
۲- پیداکردن محل دقیق این نقاط کلیدی
۳- تعریف یک بردار جهت برای هر نقطه کلیدی
۴- تعریف یک بردار توصیف برای هر نقطه کلید بر اساس بردار جهت مورد ۳
۵- تطبیق نقاط کلیدی

برای استفاده از ماژول مربوطه باید ماژول opencv-contrib نصب باشد این ماژول از نسخه ۳ دیگر همراه opencv ارایه نمیشود. برای نصب آن کافی است یکی از دو دستور زیر را در CMD ویندوز وارد کنید.

Pip install opencv-contrib-python

Pip install opencv-contrib-python-headless

در صورتی که خطای permition داد، کافی است عبارت user-- را پس از کلمه install در دستورات بالا قرار دهیم.

تذکر : نکته مهمی که باید بدان توجه داشت غیر رایگان بودن الگوریتم های SURF و SIFT است و در کاربرد های تجاری استفاده از آن‌ها نیازمند مجوز می‌باشد

برای استفاده از الگوریتم SIFT ابتدا باید یک شی SIFT از کلاس SIFT_creat از ماژول xfeatures۲d ایجاد نمود. سپس با کمک تابع ()detect از همان کلاس keypoint های تصویر را استخراج می‌نماییم. همچنین تابع دیگری نیز در این ماژول با نام ()detectAndCompppute موجود است که علاوه بر یافتن نقاط کلیدی، توصیفگر نیز برای آنها باز میگرداند.

detect (image [, mask]) -> keypoints

image : تصویر ورودی که باید از نوع خاکستری باشد.

Mask : پارامتر ماسک در صورتی که بخواهیم بخشی از تصویر را بررسی کنیم

خروجی این تابع نقاط کلید تصویر است. هر نقطه کلیدی یا keyPoint دارای یک ساختار خاص است که ویژگی های بسیاری نظیر مختصات نقطه ، زاویه جهت گیری، محدوده پیکسل های همسایه و... را مشخص میکند.

برای نمایش keypoint های یافته شده از تابع ()drawKeypoints استفاده می‌شود

drawKeypoint (image , KeyPoins , out imge[, color [, flags]]) -> out image

image : تصویر ورودی

Keypoint : نقاط کلیدی مورد نظر که همان خروجی تابع ()detect است

Out image : تصویر خروجی

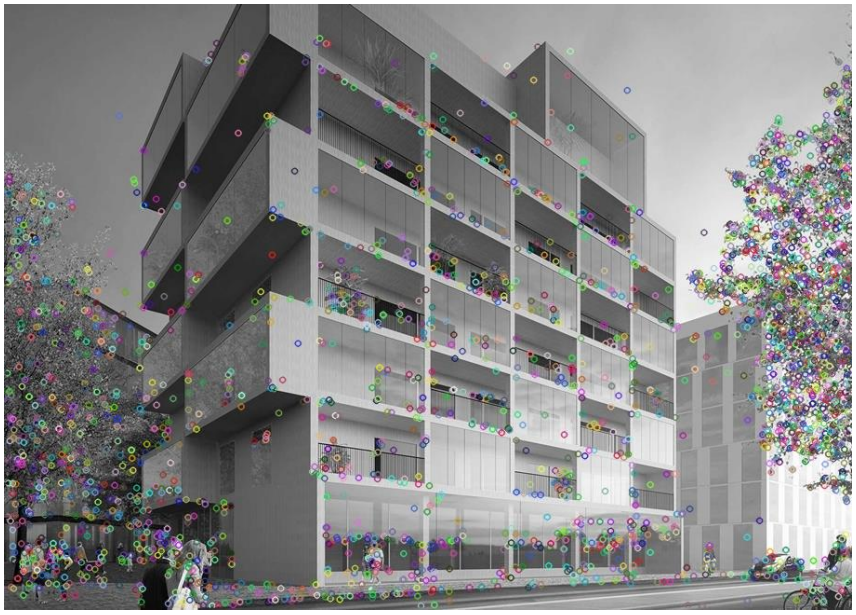
Color : رنگ keypoint ها را مشخص میکند و اگر ۱- باشد برای هر keypoint یک رنگ رندوم در نظر می‌گیرد.

Flags : این پارامتر میتواند دارای مقادیر زیر باشد

۱- DRAW_MATCHES_FLAGS_DRAW_DEFAULT : حالت پیشفرض بوده و با فراخوانی این پرچم برای تصویر خروجی (مقدار بازگشتی) یک ماتریس جدید تعریف می‌شود و تصویر نهایی در آن ریخته می‌شود
۲- DRAW_MATCHES_FLAGS_DRAW_OVER_OUTIMG: با فراخوانی این پرچم، تصویر خروجی در همان پارامتر ورودی out image ذخیره می‌شود. در این حالت اگر پارامتر وردی مذکور تصویری رنگی باشد، خروجی نیز رنگی خواهد بود
۳- DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS : با فراخوانی این پرچم، نقاط کلیدی را با یک دایره به اندازه محدوده همسایگی و جهتشان رسم میکند. در غیر این صورت keypoint ها تنها با یک دایره تو خالی نمایش داده می‌شوند
۴- DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS : با این پرچم تابع نقاط کلیدی تکی را رسم نمی‌کند. کاربرد اصلی آن در تابع رسم تطابق ها است که فقط نقاطی را که باهم تطابق دارند را نشان می‌دهد

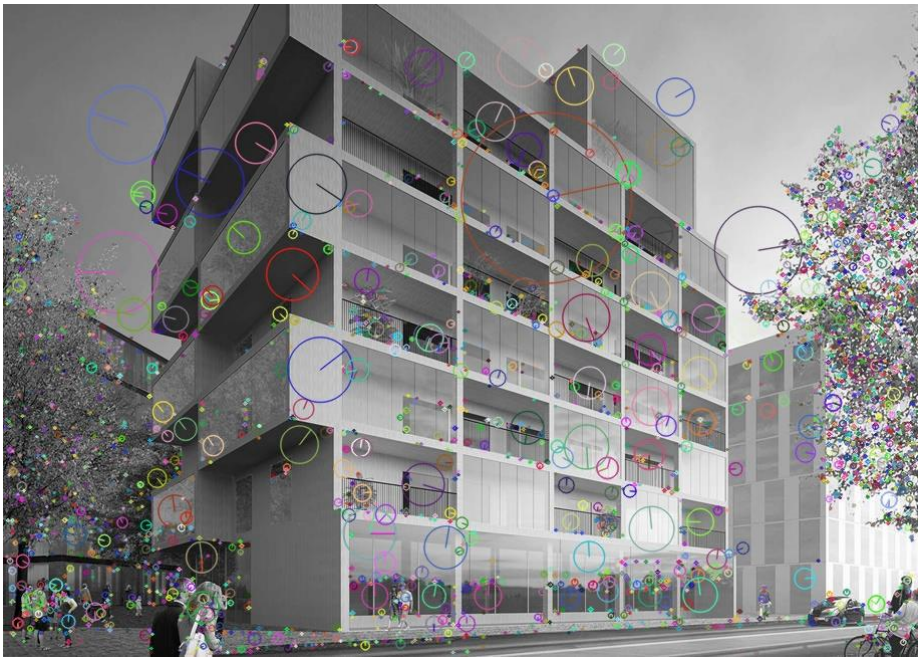
```
import cv2

img=cv2.imread("bulding.jpg")
gray=cv2.cvtColor( img , cv2.COLOR_BGR2GRAY)

sift=cv2.xfeatures2d .SIFT_create ()
keyPoint=sift.detect ( gray , None)
outimg=cv2.drawKeypoints(gray,keyPoint,img)

cv2.imwrite('sift_keypoints.jpg',outimg)
```

مثال :



مثال :

```
import cv2

img=cv2.imread("bulding.jpg")
gray=cv2.cvtColor( img , cv2.COLOR_BGR2GRAY)

sift=cv2.xfeatures2d .SIFT_create ()
keyPoint=sift.detect ( gray , None)
outimg=cv2.drawKeypoints(gray,keyPoint,img ,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

cv2.imwrite('sift_keypoints.jpg',outimg)
```



الگوریتم SURF

الگوریتم SIFT زمان زیادی برای اجرا صرف میکند. در سال ۲۰۰۶ الگوریتم دیگری با نام SURF با همان عملکرد SIFT ولی با سرعتی به مراتب بیشتر طراحی شد. این الگوریتم همانند الگوریتم SIFT می‌باشد که سرعت اجرای آن بالاتر می‌باشد هزینه این سرعت بالا در کاهش دقت می‌باشد که عموما قابل چشم پوشی است. الگوریتم SIFT همانند یک آشکار ساز لکه است. برای استفاده از این الگوریتم ابتدا باید یک شی از کلاس SURF از ماژول xfeatures۲d ایجاد نمود. ورودی های متد سازنده این کلاس به صورت زیر است

SURF_creat ([hessianThreshold [, nOctaves [, nOctaveLayers [, extended [, upRights]]]])

hessianThreshold : پارامتر مقدار آستانه که حداقل میزان اهمیت را برای اینکه یک نقطه به عنوان keyPoint در نظر گرفته شود را مشخص می‌کند و هرچه این عدد افزایش یابد تعداد نقاط کلیدی کم خواهد شد. مقداری بین ۳۰۰ تا ۵۰۰ می‌تواند مقدار پیشقرض خوبی باشد.

nOctaves : مقدار این پارامتر به طور پیشفرض برابر ۴ است. اگر مقدار آن افزایش یابد، الگوریتم ویژگی های بزرگتر را می‌یابد و اگر کوچک تر باشد، الگوریتم تنها ویژگی های کوچک را می‌یابد

nOctaveLayers : تعداد لایه های در هر هرم گاوسی را مشخص میکند و مقدار پیشفرض آن ۲ است.

Extended : اگر این پارامتر ۰ باشد توصیف گر برای هر نقطه کلیدی به صورت ۶۴ عنصری و اگر ۱ باشد ۱۲۸ عنصری خواهد بود

upRights : اگر این پارامتر ۰ باشد الگوریتم keyPoint ها را به همراه زاویه جهت گیرشان مشخص میکند و اگر ۱ باشد آنها را بدون جهت گیری می‌یابد. در تصاویری که تغییر زاویه کم است با ۱ کردن این پارامتر می‌توان سرعت اجرا را تا حد زیادی افزایش داد

معرفی توابع کلاس SURF_creat :

getExtended() ->bool retval / setExtended(bool extend)

getHessianThreshold() -> doubl retVal / setHessainThreshold(double thresh)

getNOctaveLayers() -> int retVal / setNOctaveLayers(int nOctaveLayers)

getNOctaves() -> int retVal / setNOctaves(int nOctaves)

getUpright() -> bool retVal / setUpright(bool upright)

کلیه توابع بالا به ترتیب برای دریافت و تغییر پارامتر مربوطه مورد استفاده قرار می‌گیرند

برای تشخیص keyPoint ها و ارایه توصیفگر برای آن‌ها از تابع ()detectAndComput استفاده می‌کنیم. تفاوت این تابع با تابع ()detect آن است که علاوه بر یافتن نقاط کلیدی، توصیف گر نیز برای آنها ارایه می‌دهد

detectAndComput (image , mask) -> keyPoints , Descriptors

- image : تصویر ورودی که باید یک تصویر خاکستری باشد.
- Mask : ماسک در صورتی که بخواهیم تنها بخشی از عکس را بررسی کنیم.

این تابع دارای دو خروجی است. خروجی اول keyPoint ها و خروجی دوم Descriptor یا توصیفگر آن‌ها می‌باشد.

برای رسم keyPoint ها از همان تابع ()drawKeypoints که در بخش الگوریتم SIFT معرفی کردیم، استفاده می‌کنیم.

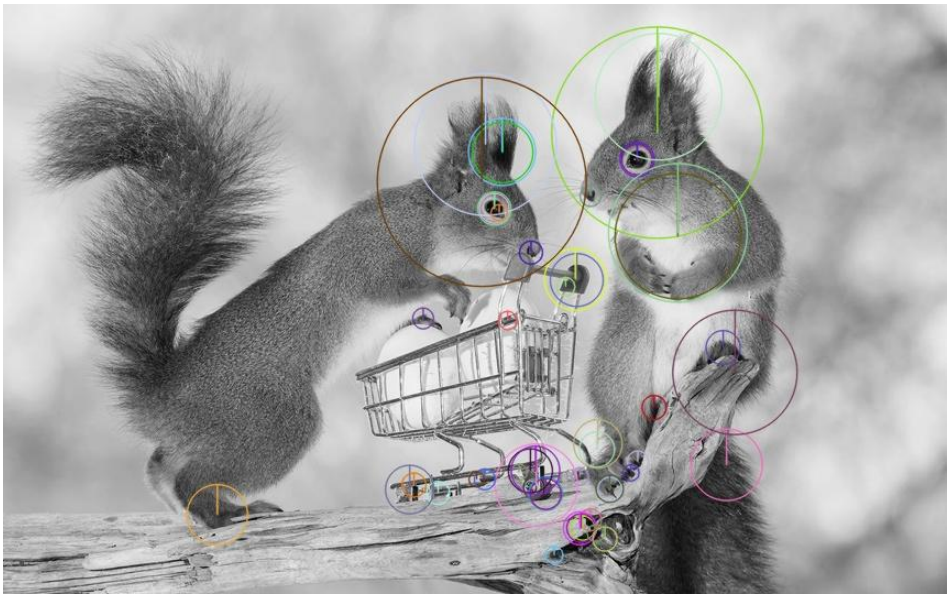
مثال :

```
import cv2
from matplotlib import pyplot as plt

img=cv2.imread ("img.jpg" , 0)
outimg=img.copy ()

surf=cv2.xfeatures2d .SURF_create (6000)
kp , des =surf.detectAndCompute( img , None)

outimg=cv2.drawKeypoints ( img , kp , outimg , flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

plt.imshow( outimg)
plt.show()
```

در مثال بالا ابتدا حد آستانه SURF را روی ۴۰۰ تنظیم کردیم و تعداد نقاط (با تابع len()) ۱۶۰۰ عدد بدست آمد.از این رو تصویر خروجی بشدت شلوغ بود و برای استفاده در این آموزش نا مناسب بود. از این رو مقدار حد آستانه را به ۶۰۰۰ افزایش دادیم و تعداد keypoint ها به ۷۹ عدد کاهش یافت.

مثال :

```
import cv2
from matplotlib import pyplot as plt

img=cv2.imread ("img.jpg" , 0)
outimg=img.copy ()

surf=cv2.xfeatures2d .SURF_create (7000)
surf.setUpright (True)

kp , des =surf.detectAndCompute( img , None)
outimg=cv2.drawKeypoints ( img , kp , outimg , flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

plt.imshow( outimg)
plt.show()

print (“ kp len=” , len(kp))
print (“des shape=” , des.shap)
```



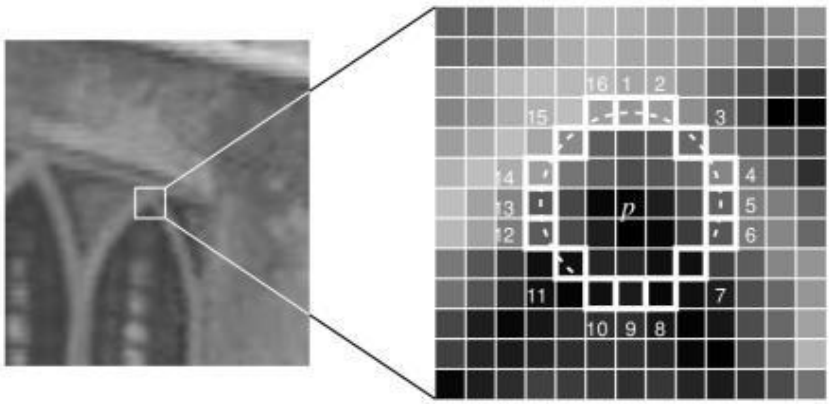
همان گونه که مشخص است، در مثال بالا با true کردن upright، نقاط کلیدی بدون جهت گیری یافت شدند.

```
>>> kp len = 50
>>> des shape = (50 , 64)     # 64 elements
```

الگوریتم FAST

الگوریتم های ارایه شده در بخش های قبلی بسیار مفید و کارامد بودند اما به اندازه کافی سریع نبودند . در کاربرد هایی که سرعت محاسبات محدود است، میتواند ما را با مشکل مواجه کند. در سال ۲۰۰۶ الگوریتمی برای تشخیص گوشه ها با نام FAST ارایه

شد که دارای سرعت بالا است. این الگوریتم در سال ۲۰۱۰ تا حدی اصلاح شد. در این الگوریتم برای تشخیص یک نقطه به عنوان گوشه ابتدا یک دایره با شعاع مشخص به مرکز نقطه مذکور رسم می‌کنیم. حال اگر کمانی به طول حداقل ¾ متشکل از پیکسل های روی محیط این دایره وجود داشته باشد به طوریکه همه آنها به مقدار زیادی از پیکسل مرکزی بیشتر یا کمتر باشند، آن نقطه به عنوان گوشه یا نقطه کلیدی در نظر گرفته می‌شود. برای سرعت بخشیدن به این عملیات ابتدا چهار پیکسل با زاویه ۹۰ درجه روی محیط دایره انتخاب می‌شوند و اگر حداقل سه تای آن ها اختلافشان با پیکسل مرکزی زیاد نباشد آن نقطه گوشه نبوده و به سراغ پیکسل بعدی می‌رویم. برای درک بهتر موضوع به تصویر زیر توجه نمایید.



در تصویر بالا پیکسل p، نقطه مورد بررسی ما است و ما یک دایره به طول ۱۶ پیکسل حول آن در نظر می‌گیریم . مقدار آستانه را t و مقدار روشنایی پیکسل مرکزی را ip در نظر بگیرید. حال اگر ¾ این ۱۶ پیکسل ، یعنی ۱۲ پیکسل، مقدار روشناییشان همه بیشتر از ip+t یا کمتر از ip-t باشد، این نقطه به عنوان گوشه در نظر گرفته می‌شود. در آزمون سرعتی، پیکسل های ۱،۵،۹،۱۳ را بررسی میکنیم اگر حداقل ۳ تا از آن ها روشن تر ip+t یا تیره تر از ip-t نباشند، آن نقطه گوشه نیست.

برای پیاده سازی این الگوریتم مانند الگوریتم های قبل ابتدا یک شیFAST ایجاد می‌کنیم . پارامترهای متد سازنده این کلاس به صورت زیر است

FastFeatureDetector_create([, threshold [, nonmaxSuppression [,type]]])

Threshold : آستانه تفاوت میان شدت شدت روشنایی پیکسل مرکزی و پیکسل های روی محیط دایره. مقدار پیشفرض برای پارامتر برابر ۱۰ است

nonmaxSuppression : یک مقدار منطقی است برای سرکوب غیر حداکثر، که اگر True باشد، تابع از میان نقاط تشخیص داده شده برای هر گوشه، بهترین آن‌ها را انتخاب و بقیه را حذف میکندو اگر false باشد همه آن نقاط را باز می‌گرداند . مقدار این تابع در حالت پیشفرض True است

type : یکی از سه حالت زیر می‌تواند باشد

۱- FAST_FEATURE_DETECTOR_TYPE_۵_۸

۲- FAST_FEATURE_DETECTOR_TYPE_۷_۱۲

۳- FAST_FEATURE_DETECTOR_TYPE_۹_۱۶

معرفی برخی توابع کلاس FAST:

۱-getThreshod() / setThreshold(int threshold)

۲-getNonmaxSuppression() / setNonmaxSuppression(bool NonmaxSuppression)

۳-getType() / setType()

```
import cv2
from matplotlib import pyplot as plt

img=cv2.imread ( "shape.jpg")
gray=cv2.cvtColor( img , cv2.COLOR_BGR2GRAY)

fast=cv2.FastFeatureDetector_create()
kp=fast .detect( gray , None)
gray=cv2.drawKeypoints(gray , kp  , None , (0,0,255))

plt.imshow ( gray)
plt .show ()

print("point number" , len(kp))
print(" fast Th" , fast.getThreshold () )
print(" fast NonmaxSupprestion " , fast.getNonmaxSuppression () )
print(" fast Type" , fast.getType () )
```

این توابع به ترتیب برای دریافت و یا تغییر مقدار پارامتر مربوطه است.

BRIEF

همان طور که در بخش های قبلی ملاحضه نمودید الگوریتم SIFT از یک توصیفگر ۱۲۸ تایی استفاده می‌کند. هر کدام از این مقادیر یک داده float است از این رو حجم توصیفگر برای هر نقطه ۵۱۲ بایت خواهد بود و این می‌تواند برای هزاران ویژگی، حجم زیادی اشغال کند. الگوریتم SURF نیز حداقل از توصیفگر ۶۴ تایی استفاده میکند که حافظه‌ای معادل با ۲۵۶ بایت برای هر یک اشغال می‌کند. مشخصا بررسی این حجم بالا از اطلاعات، زمانبر نیز خواهد بود. ما می‌دانیم که همه این ابعاد برای تطبیق لازم نیستند و میتوان این توصیفگر ها را با روش هایی فشرده کرد. برای مثال می‌توان به جای استفاده از داده های اعشاری از رشته های باینری بهره برد. الگوریتم BRIEF برای همین موضوع طراحی شده است. این الگوریتم برای نقاط کلیدی تصویر، توصیفگری به صورت رشته‌های باینری ارائه می‌دهد. برای بهره برد از این الگوریتم ابتدا باید نقاط کلیدی را توسط یک الگوریتم دیگر بیابیم و نقاط پیدا شده را به BRIEF دهیم تا برای آن یک توصیفگر فشرده ارایه دهد. توصیف گر BRIEF میتواند ۱۲۸، ۲۵۶ یا ۵۱۲ تایی باشد که حافظه‌هایی معادل ۳۲، ۶۴ و ۱۲۸ بیتی اشغال می‌کند.

برای استفاده از این الگوریتم ابتدا باید یک شی brief از کلاس briefDescriptorExtractor_creat() از ماژول xfeatures۲d ایجاد نمود. سپس توسط یک الگوریتم keypoint ها را بیابیم. متد سازنده این کلاس به صورت زیر است

briefDescriptorExtractor_creat([,bytes])

bytes : چند بایتی بودن توصیف گر را مشخص می‌کند که می‌تواند ۳۲،۶۴ یا ۱۲۸ باشد.

در نهایت توسط تابع compute() از کلاس فوق، برای keypoint ها، توصیفگر ایجاد کنیم.

Compute (image , keypoints) -> keypoints , descriptors

Image : تصویر ورودی

Keypoints : نقاط کلیدی که توسط یک الگوریتم ویژگی یاب نظیر SIFT یافت شده

این تابع دو خروجی دارد. خروجی اول نقاط کلیدی و خروجی دوم توصیفگر این نقاط

تذکر: الگوریتم BRIEF در برابر چرخش های زیاد ، ضعیف عمل می‌کند و ممکن است توصیفگر با چرخش تصویر دچار تغییر شود.

مثال :

```
import cv2
img=cv2.imread ("Wall-E.jpg" , 0)

brief=cv2.xfeatures2d .BriefDescriptorExtractor_create()
surf=cv2.xfeatures2d .SURF_create()

kp=surf .detect (img , None)
kp,ds = brief .compute (img , kp)

print ( "brief size : ",brief.descriptorSize())
print ( "ds shape : " ,ds.shape)
```

خروجی به صورت زیر خواهد بود :

```
>>>
brief size :  32
ds shape :  (4485, 32)
>>>
```

الگوریتم ORB

این الگوریتم در سال ۲۰۱۱ به عنوان یک جایگزین برای SURF و SIFT، مطرح شد. این الگوریتم بر خلاف الگوریتم‌های SIFT و SURF رایگان است. این الگوریتم را میتوان ترکیب از آشکار ساز FAST و توصیف کننده BRIEF دانست که بهبود یافته است. در ابتدا این الگوریتم با استفاده از FAST نقاط کلیدی تصویر را پیدا می‌کند. سپس با الگوریتم Hariss تعداد n نقطه را از بین آن‌ها انتخاب می‌کند. با توجه به این که هر دوی این الگوریتم ها نسبت به چرخش تصویر مستقل نیستند، برای حل این مشکل راهکاری

ارایه شده است. در نهایت این الگوریتم با استفاده از BRIEF برای نقاط ویژگی descriptor ارایه می‌دهد . همچنین مشکل حساسیت BRIEF به تغییر زاویه نیز در این الگوریتم اصلاح شده است.

برای استفاده از این الگوریتم مانند قبل ابتدا یک شی orb ایجاد میکنیم. متد سازنده این کلاس به صورت زیر است

ORB_creat ([,nF [, typeScore [, WTA_K [,...]]]])

nFeaturs : تعداد نقاط ویژگی را مشخص می‌کند. مقدار پیشفرض آن ۵۰۰ است.

TypeScor : این پارامتر مشخص می‌کند که رتبه بندی بر اساس FAST صورت گیرد و یا Harris که می‌تواند به صورت یکی از مقادیر ORB_FAST_SCORE یا ORB_HARRIS_SCORE باشد که به طور پیشفرض، هریس است

WTA_K : این پارامتر نیز می‌تواند NORM_HAMMING یا ۲NORM_HAMMING باشد

پس از ایجاد یک شی orb، برای بدست آوردن keypoints ها از تابع ()detect استفاده می‌کنیم و سپس برای بدست آوردن descriptors از تابع ()compute بهره می‌بریم.

detect (image , mask) -> keypoints

- compute(image , keypoints) -> keypoints , descriptor

- image : تصویر ورودی
- Mask : پارامتر ماسک در صورتی که بخواهیم بخشی از تصویر را بررسی کنیم.
- Keypoints : نقاط ویژگی که از تابع ()detect بدست آوردیم

تطبیق ویژگی ها با Brue_Force

هدف در این بخش تطبیق ویژگی های دو تصویر است. در اینجا یک توصیفگر از مجموعه اول انتخاب و با کلیه توصیفگرهای مجموعه دوم مقایسه می‌شود. و فاصله آن ها دو به دو محاسبه می‌شود . سپس تابع توصیفگری که کمترین فاصله را با توصیفگر دوم داشته باشد را باز می‌گرداند. که در واقع این دو توصیف گر نشان‌دهنده دو نقطه همسان در تصویر هستند. برای استفاده از تطبیق دهنده BF، ابتدا باید یک شی از کلاس ()BFMatcher_creat تعریف می‌کنیم. متد سازنده این کلاس به صورت زیر است

cv۲.BFMatcher_create ([,normType [, crossCheck]])

normType : این پارامتر روش اندازه گیری فاصله را مشخص می‌کند. که می‌تواند به صورت های زیر باشد

۱- NORM_L۲ یا NORM_L۱ : این روش برای توصیفگر الگوریتم های SURF و SIFT مناسب است
۲- NORM_HAMMING : برای توصیفگر الگوریتم های ORB، BRIEF و BRISK مناسب است. اگر در الگوریتم ORB پارامتر WTA_K برابر ۳ یا ۴ باشد باید از NORM_HAMMING۲ استفاده کنیم.

crossCheck : این پارامتر یک مقدار Boolean است که در حالت پیشفرض false است. اگر این پارامتر true باشد تابع تطابق ها را به صورت (i , j) باز میگرداند که i نشان دهنده درایه توصیفگر اول از مجوعه اول و j نشان‌دهنده توصیفگر دوم از مجموعه دوم است.

پس از ایجاد شی bf ما دو روش برای بدست آوردن تطابق‌ها داریم. روش اول استفاده از تابع ()match و روش دوم استفاده از تابع ()KnnMatch . تابع ()match تنها بهترین تطابق را باز می‌گرداند اما در تابع دوم k تا از بهترین تطابق ها را باز می‌گرداند

match (descriptors۱ , descriptors۲) -> matches

knnMatch (descriptors۱ , descriptors۲ , k) -> matches

- descriptor۱ : مجموعه توصیفگرهای اول
- descriptor۲ : مجموعه توصیفگر های دوم
- k : این پارامتر در تابع ()knnMatch مشخص می‌کند تابع برای هر ویژگی چند تطبیق باز گرداند.

برای رسم تطابق ها، از تابع ()drawMatches . این تابع دو تصویر را به صورت افقی رسم می‌کند و نقاطی را که مطابق هم هستنند با رسم یک خط از تصویر اول به تصویر دوم نشان می‌دهد

drawMatches(img۱ , keypoints۱ , img۲ , keypoints۲ , matches , out img [,color [, flag]])

- img۱ و img۲ : به ترتیب تصاویر اول و دوم که می‌خواهیم مطابقتشان را بررسی کنیم.
- Keypoints۱ و keypoints۲ : به ترتیت نقاط کلیدی تصاویر اول و دوم هستند
- Matches : م طابقت ها که همان خروجی تابع ()match می‌باشد
- Out img : تصویر خروجی که شامل دو تصویر ورودی است و نقاطی که مطابق دارند در آن با اتصال خط مشخص شده‌اند
- Color : رنگ خطوطی که نقاط را به هم متصل می‌کند را مشخص می‌نماید و اگر ۱- باشد، رنگ خطوط تصادفی انتخاب می‌شوند
- Flag : پرچم های این پارامتر درست همانند پرچم های متد ()drawKeypoints است که قبلابه آن‌ها اشاره شده.

همچنین تابع drawMatcesKnn() برای رسم k تا از بهترین ویژگی های تصویر را رسم می‌کند که برای رسم خروجی تابع knnMatck() کاربرد دارد.

مثال :

```
import cv2
from matplotlib import pyplot as plt

img1=cv2.imread("dollor.jpg",0)
img2=cv2.imread("image.jpg",0)

surf=cv2.xfeatures2d .SURF_create(1000)

kp1, des1=surf .detectAndCompute( img1 , None)
kp2, des2=surf .detectAndCompute( img2 , None)

bf=cv2.BFMatcher_create( cv2.NORM_L1)
matches=bf.match (des1 , des2)
imgOut=cv2.drawMatches( img1 , kp1 , img2 , kp2 , matches[0:20] , None ,
                        flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)

plt.imshow(imgOut),plt.show()

cv2.imwrite ( "out.jpg" , imgOut)
```

خروجی تابع match() یک لیست از Dmatch ها است. هر Dmatch شامل ویژگی های از یک تطابق است که شامل موارد زیر می‌شود

۱- Dmatch.distance : این پارامتر نشان دهنده فاصله آن تطابق است که هر چه کمتر باشد، تطابق بهتر است.

۲- Dmatch.trainIdx : درایه توصیفگر از لیست دوم

۳- Dmatch.queryIdx : درایه توصیفگر از لیست جست جو که همان لیست اول است (عموما تصویر اول شی است که میخواهیم آن را در تصویر دوم پیدا کنیم)

۴- Dmatch.imgIdx : درایه تصویر دوم

اگر از تابع knnMatch() استفاده نماییم هر سطر شامل k تا Dmatch می‌باشد همچنین keypoins ها نیز لیستی از Dkp ها است که شامل ویژگی های زیر است

۱- Dkp.angle : زاویه آن نقطه کلیدی را نشان می‌دهد

۲- Dkp.pt : مختصات نقطه کلیدی را نشان می‌دهد

۳- Dkp.size : قطر همسایگی نقطه کلیدی مربوطه را نشان می‌دهد. هر چه این پارامتر بزرگتر باشد، نقطه کلیدی اطلاعات بیشتری را پوشش می‌دهد

۴- Dkp.response : این پارامتر نشان دهنده میزان اهمیت نقطه کلیدی است و از این پارامتر میتوان برای رتبه بندی نقاط کلیدی استفاده کرد

```
import cv2
from matplotlib import pyplot as plt

img1=cv2.imread("dollor.jpg",0)
img2=cv2.imread("image.jpg",0)

surf=cv2.xfeatures2d .SURF_create(1000)

kp1, des1=surf .detectAndCompute( img1 , None)
kp2, des2=surf .detectAndCompute( img2 , None)

bf=cv2.BFMatcher_create( cv2.NORM_L1)
matches=bf.match (des1 , des2 )

Min=matches [0].distance
for i in range(0,len(matches )-1) :
    if( matches[i].distance  < Min):
        Min=matches[i].distance


best=[]
for m in matches:
    if( m.distance<= Min * 1.6):
        best.append(m)


imgOut=cv2.drawMatches( img1 , kp1 , img2 , kp2 , best[0:13] , None ,
                        flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)

plt.imshow(imgOut),plt.show()
```

مثال : در مثال زیر ما بر اساس ویژگی های ذکر شده، ۱۳ تا از بهترین تطابق‌ها را پیدا و رسم نمودیم

مثال : در مثال زیر ما بر از ()knnMatch استفاده نمودیم و k=۲ قرار دادیم. سپس برای هر توصیفگر، تطبیقی را باز می‌گردانیم که از ۰.۸ دیگری کوچک تر باشد

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
img1=cv2.imread("dollor.jpg",0)
img2=cv2.imread("image.jpg",0)

surf=cv2.xfeatures2d .SURF_create()

kp1, des1 = surf.detectAndCompute(img1,None)
kp2, des2 = surf.detectAndCompute(img2,None)

bf = cv2.BFMatcher()
matches = bf.knnMatch(des1,des2, k=2)

good = []
for m,n in matches:
    if m.distance < n.distance*0.8:
        good.append([m])
    elif n.distance < m.distance*0.8:
        good .append ([n])

imgOut = cv2.drawMatchesKnn(img1,kp1,img2,kp2,good[0:10],None,flags=2)

plt.imshow(imgOut),plt.show()
cv2.imwrite ( "outKnn.jpg" , imgOut)
```

تطبیق ویژگی با FLANN

الگوریتم FLANN همانند Bure-Force برای بررسی تطابق دو تصویر کاربرد دارد. این الگوریتم برای داده‌های بزرگ، سریع تر از الگوریتم BF عمل می‌کند.

برای استفاده از این الگوریتم ابتدا باید یک شی flann از کلاس FlannBasedMatcher() تعریف نمود. متد سازنده کلاس به صورت زیر است

FlannBasedMatcher(indexParams , searchParams)

indexParams : برای درک جزئیات و تنظیم پارامتر اول لازم است تا به مقالات ارایه شده برای این تابع مراجعه نمایید. اما به طور پیشفرض میتوان دو مقدار دهی برای این پارامتر در نظر گرفت. اگر الگوریتم ویژگی‌یاب استفاده شده، الگوریتم‌های SIFT یا SURF باشد، میتوان این پارامتر را به صورت زیر تنظیم کرد

```
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
```

```
FLANNINDEX_LSH = 6
index_params= dict(algorithm = FLANN_INDEX_LSH,
table_number = 12,
key_size = 20,
multi_probe_level = 2)
```

```
FLANNINDEX_LSH = 6
index_params= dict(algorithm = FLANN_INDEX_LSH,
table_number = 6,
key_size = 12,
multi_probe_level = 1)
```

و

برای الگوریتم‌هایی مانند ORBو BRISK برای رسیدن به نتیجه مطلوب میتوان از یکی از تنظیمات زیر بهره بود

```
search_params = dict(checks=x)
```

search : پارامتر دوم نیز به صورت زیر تنظیم می‌شود. مقدار x برای این پارامتر هرچه بیشتر باشد، دقت الگوریتم بالا رفته و زمان پردازش نیز افزایش می‌یابد.

پس از تنظیم پارامتر های فوق این دو پارامتر را به متد سازنده کلاس FLANN می‌دهیم و یک شی ایجاد می‌کنیم و در نهایت برای بدست آوردن تطابق، همانند الگوریتم BF از تابع ()knnMatch یا ()match استفاده میکنیم

مثال : در این مثال، مثال شماره ۳ در بخش الگوریتم BF را با الگوریتم FLANN پیاد سازی کردیم. تنها تفاوت این مثال با مثال قبلی در بخشی است که میان کامنت ها قرار گرفته. نکته مهم آن است که در مقایسه بین تطابق های m و n میدانیم همواره m بهتر از n است و فاصله آن کمتر است پس elif هیچگاه رخ نمی‌دهد و ما در این مثال آن را کامنت کردیم.

```
import cv2
from matplotlib import pyplot as plt
img1=cv2.imread("dollor.jpg",0)
img2=cv2.imread("image.jpg",0)
surf=cv2.xfeatures2d .SURF_create(1000)
kp1, des1=surf .detectAndCompute( img1 , None)
kp2, des2=surf .detectAndCompute( img2 , None)
#########################################################
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)
flann=cv2.FlannBasedMatcher (index_params , search_params)
matches=flann.knnMatch ( des1 , des2 , k=2 )
#########################################################
good = []
for m,n in matches:
    if m.distance < n.distance*0.8:
        good.append([m])
    #elif n.distance < m.distance*0.8:
        #good .append ([n])

imgOut = cv2.drawMatchesKnn(img1,kp1,img2,kp2,good[0:10],None,flags=2)

plt.imshow(imgOut),plt.show()
cv2.imwrite ( "outKnn.jpg" , imgOut)
```

یافتن شی با استفاده از تطابق و هوموگرافی

در بخش قبلی ما ویژگی های تصویر query را در یک تصویر train جست و جو کردیم و تطابق هایی را برای آن یافتیم و بهترین آن‌ها را انتخاب نمودیم. حال با داشتن این اطلاعات می‌توانیم به راحتی با استفاده از homography محل سوژه مورد نظر را در تصویر train بیابیم. اگر ما نقاطی از تصویر query و تطابق آنها در train را به تابع بدهیم، تابع تغییرات پرسپکتیوی سوژه را به ما خروجی می‌دهد که با استفاده از آن میتوان با کمک تابع ()perspectiveTransform سوژه مورد نظر را پیدا کنیم.

برای شروع ابتدا باید مختصات نقاط تطابق را در تصویر train و query در دو آرایه جداگانه با نوع float۳۲ قرار دهیم. سپس تابع ()findHomography را فراخوانی می‌کنیم

findHomography(SrcPoint , dstPoint , method , ransanReprojThreshold) -> homography , mask

- SrcPoint : مختصات نقاط از تصویر اصلی (تصویر سوژه)
- dstPoint : مختصات نقاط از تصویر که میخواهیم سوژه را در آن بیابیم
- Method : روش مورد استفاده برای پیداکردن پیداکردن ماتریس Homographhy یا همگرایی که می‌تواند یکی از حالات زیر باشد
  - ۰ : یک روش منظم بر اساس کلیه نقاط
  - RANSAC : یک روش قدرتمند
  - LMEDS : یک روش قدرتمند مبتنی بر کمترین میانه
- ransanReprojThreshold : این پارامتر برای روش RANSAC مورد استفاده قرار میگیرد و حداکثر خطا قابل قبول را مشخص میکند که عموما عددی بین ۱ تا ۱۰ است.

خروجی اول این تابع یک ماتریس تبدیل و یک ماسک است

پس از فراخوانی تابع فوق مختصات چهار گوشه تصویر query را که متعلق به سوژه ما است را به همراه ماتریس تبدیل حاصل از تابع بالا را به تابع ()persprctiveTransform می‌دهیم تا مختصات گوشه ها را در تصویر train پیدا کنیم.

perspectiveTransform(src , m) -> dst

- src : مختصات های اولیه

- M : ماتریس تبدیل که همان خروجی تابع findHomography() است

خروجی این تابع مختصات های انتقال یافته است

مثال :

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

img1 = cv2.imread("dollor.jpg",0)
img2 = cv2.imread("image.jpg",0)

sift = cv2.xfeatures2d.SIFT_create ()

kp1 , des1 = sift.detectAndCompute ( img1 , None)
kp2 , des2 = sift.detectAndCompute ( img2 , None)

FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)

flann = cv2.FlannBasedMatcher ( index_params , search_params)

matches=flann .knnMatch ( des1 , des2 , 2)

best=[]
for m,n in matches:

    if m.distance < n.distance * 0.6 :
        best.append( m )

###############################################################################

src_pts = np.float32 ( [kp1[ m.queryIdx ].pt for m in best ]).reshape(-1,1,2)
dst_pts = np.float32 ( [kp2[ m.trainIdx ].pt for m in best ]).reshape(-1,1,2)

#find Homography
HM, mask= cv2.findHomography ( src_pts , dst_pts , cv2.RANSAC , 5)

#find opject in train image
h,w = img1 .shape
src = np.float32 ( [[0,0] , [0,h] , [w,h] , [w,0] ]).reshape(-1,1,2)
dst=cv2.perspectiveTransform ( src , HM)

#draw dst
imgOut = cv2.polylines(img2,[np.int32(dst)],True,255,3, cv2.LINE_AA)

plt.imshow(imgOut, 'gray'),plt.show()
```

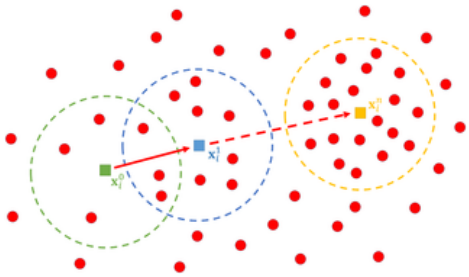
همچنین برای رسم تطابق ها می‌توان از ماسک خروجی تابع findHomography() استفاده کرد تا تنها نقاط درون محدوده جسم رسم شود. کد آن به صورت زیر است که در ادامه کد قبل می‌آید

```
matchesMask = mask.ravel().tolist()
draw_params = dict(matchColor = (0,255,0), # draw matches in green color
                   singlePointColor = None,
                   matchesMask = matchesMask, # draw only inliers
                   flags = 2)
img3 = cv.drawMatches(img1,kp1,img2,kp2,good,None,**draw_params)
plt.imshow(img3, 'gray'),plt.show()
```

پردازش دنباله‌های ویدیویی

الگوریتم meanshift

کاربرد این الگوریتم در ترکینگ و دنبال کردن یک سوژه در تصویر است تیوری این الگوریتم بسیار ساده است. در این الگوریتم ما یک مجموعه نقاط یا یک توزیع پیکسل مانند تصویر احتمال Backproject داریم و آن را به همراه یک قاب به تابع می‌دهیم.
این الگوریتم با جست‌وجو روی این نقاط یا توزیع پیکسل، قاب را به سمتی که بیشترین تراکم در آن وجود دارد می‌برد تا در نهایت به یک ناحیه همگرا شود. برای درک عملکرد این الگوریتم به تصویر زیر نگاه کنید که قاب دایره‌ای شکل به سمت تراکم بیشتر حرکت میکند.



در این الگوریتم دو معیار برای توقف جست و جو در نظر گرفته می‌شود. یکی تعداد دفعات تکرار یا به عبارت دیگر تعداد حرکات قاب جست‌وجو و دیگری، میزان حداکثر جابه‌جایی مرکز قاب یا پنجره جست و جو. میزان جابه‌جایی کمتر از یک حد را میتوان همگرایی به یک نقطه پایدار درنظر گرفت و عملیات را پایان دهیم

این الگوریتم توسط تابع meanShift() پیاده سازی می‌شود.

meanShift(src image , Windows , criteria) –> move count , widow

src image: تصویر ورودی که عموما تصویر احتمال backproject مورد بررسی (پس افکنش یک هیستوگرام) است که همان خروجی تابع calcBackprjoct() است. این تصویر می‌تواند مجموعه‌ای از نقاط و یا هر تصویر خاکستری دیگری نیز باشد.

Windows: قاب یا پنجره ای اولیه که روی تصویر برای یافتن بیشتر محل احتمال، حرکت می‌کند. برای توصیف یک پنجره چهار عدد تعریف می‌شود.عرض و طول پنجره به همراه مختصات راس بالا سمت راست مستطیل

windows = (x,y,wight,height).

Criteria: شرایط توقف جست و جو را معین می‌کند. این ورودی به صورت یک سه‌تایی مشخص میشود. ورودی اول شروط توقف ، ورودی دوم ماکزیمم تعداد حرکت و ورودی سوم حداقل میزان حرکت بر اساس پیکسل را معین می‌کند.

Criteria = (condition , Max number of move , epsilon Moving)

```
Condition= cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT
```

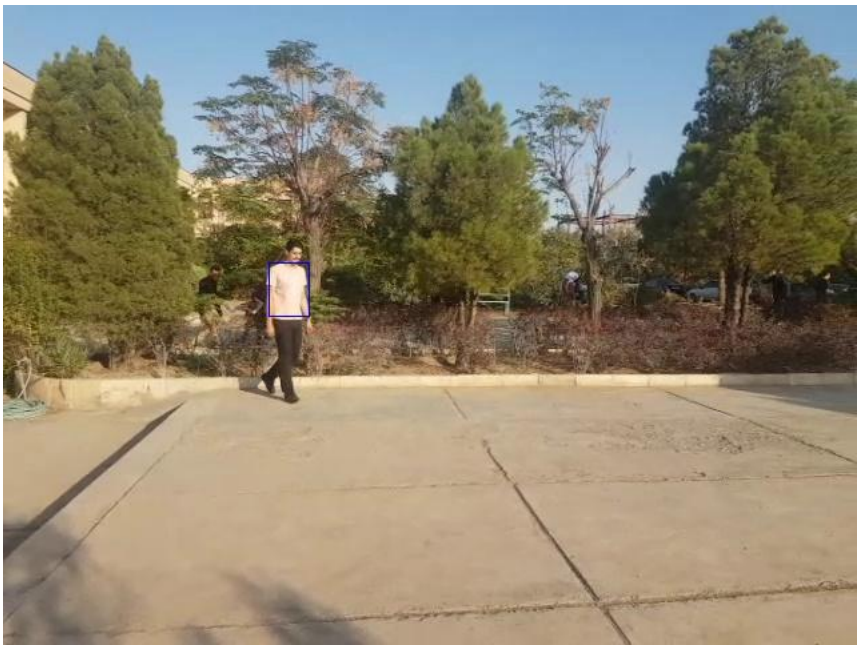
این تابع همچنین دو خروجی دارد

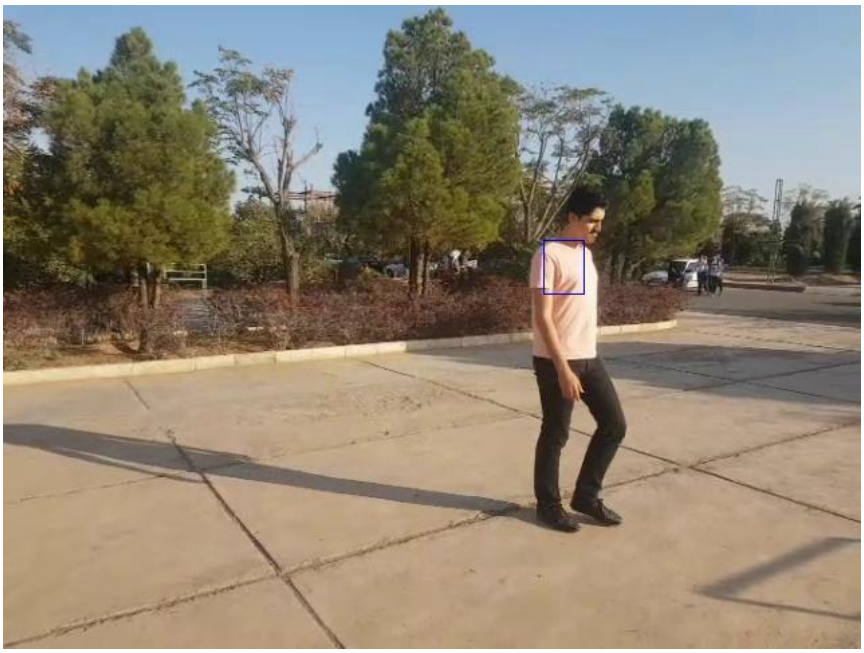
Move count : تعداد حرکات انجام شده، پنجره برای پیدا کردن محلی با بیشترین تراکم پیکسلی

Windoe : پنجره جست و جو خروجی بعد از پایان الگوریتم که پارامترهای آن همانند پنجره ورودی است

مثال : در مثال زیر ما ابتدا فریم اول فیلم مورد نظررا دریافت میکنیم. سپس بخشی از تصویر را که آبجکت مورد نظر در آن قرار دارد را جدا کرده و هیستوگرام آن را محاسبه می‌کنیم که همان هیستوگرام شی مورد نظر ما است. سپس وارد حلقه while می‌شویم و عملیات ترکینگ را آغاز میکنیم. برای اینکار ابتدا پس افکنش هیستوگرام بدست آمده را روی هر فریم فیلم بدست آورده و با اعمال تابع meanShift بر روی تصویر خروجی BackProject ، عملیات ترکینگ را انجام می‌دهیم.

```
import cv2
import numpy as np

cap = cv2.VideoCapture ( "video.mp4")

_,frame = cap.read ()


window = ( 210 , 190 , 30 , 60)

roi = frame [190 : 250 , 210 : 240 ]
cv2.imshow ( " first frame object" , roi)
hsv = cv2.cvtColor ( roi , cv2.COLOR_BGR2HSV )
hist = cv2.calcHist ( [hsv] , [0] , None , [180] , [0,180] )
cv2.normalize ( hist , hist , 0, 255, cv2.NORM_MINMAX )
criteria = ( cv2.TERM_CRITERIA_COUNT | cv2.TERM_CRITERIA_EPS , 10 , 1)
cv2.imwrite ( " object.jpj", roi)

while (cap.isOpened()) :

    _,frame = cap.read ()
    hsv = cv2.cvtColor ( frame , cv2.COLOR_BGR2HSV )
    backProjrct = cv2.calcBackProject ( [hsv] , [0] , hist, [0,180] , 1)
    _,window = cv2.meanShift ( backProjrct , window , criteria )

    x,y,w,h = window

    cv2.rectangle ( frame , (x,y) , (x+w , y+h) , (255,0,0) , thickness=1)
    cv2.imshow ( " video" , frame )
    key=cv2.waitKey (5)

    if key == 27 :
        break

    elif key!=-1 :
        cv2.imwrite ( str(key)+ ".jpg" , frame)
```





الگوریتم CamShift

الگوریتم همانند الگوریتمmeanShift می‌باشد با این تفاوت که در این الگوریتم، اندازه و زاویه پنجره جست و جو ثابت نبوده و مرتبا تغییر پیدا می‌کند و برای اهدافی که فاصله آنها از دوربین تغییر پیدا می‌کند و یا چرخش دارند مناسب است. این الگوریتم توسط تابع CamShift() پیاده سازی می‌شود . این تابع از نظر پارامتر ورودی همانند تابع meanShift() می‌باشد.

CamShift(src image , Windows , criteria) –> move count , window

src image: تصویر ورودی

Window: قاب یا پنجره ای اولیه که روی تصویر برای یافتن بیشتر محل احتمال، حرکت می‌کند.

windows = (x,y,wight,height).

Criteria: شرایط توقف جست و جو را معین می‌کند و همانند الگوریتم meanShift می‌باشد و شامل پارامتر های زیر است.

Criteria = (condition , Max number of move , epsilon Moving)

```
Condition= cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT
```

این تابع همچنین دو خروجی دارد

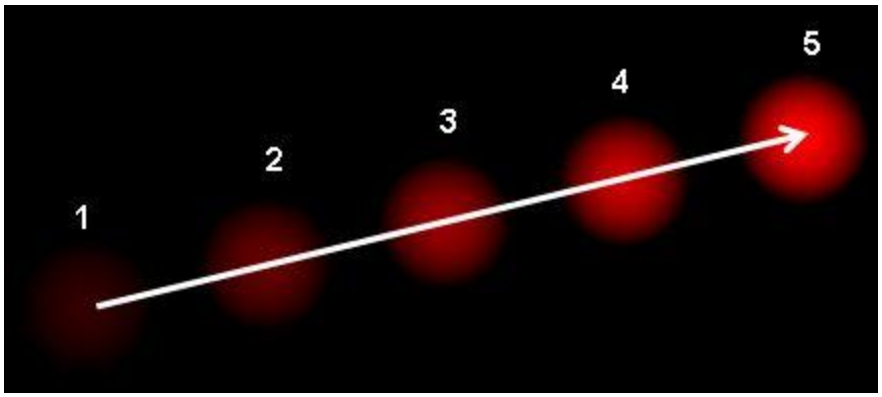
retVal : این پارامتر پنجره جست و جو پس از پایان الگوریتم، به همراه زاویه آن می‌باشد که از پارامترهای، مختصات مرکز پنجره، طول و عرض پنجره، و زاویه پنجره تشکیل شده است که با استفاده از تابع boxPoints() این پارامتر را به مختصات چهار راس چهار ضلعی تبدیل کرده و سپس با استفاده از تابع polylines() آن را رسم می‌کنیم. در الگوریتم CamShift عموما از این خروجی استفاده می‌شود. برای درک بهتر، به مثال توجه کنید.

Windoes : پنجره جست و جو خروجی بعد از پایان الگوریتم بدون زاویه، که پارامترهای آن همانند پنجره ورودی است

```
import numpy as np
import cv2

cap = cv2.VideoCapture('slow.flv')

# take first frame of the video
ret,frame = cap.read()

 # setup initial location of window
r,h,c,w = 250,90,400,125  # simply hardcoded the values
track_window = (c,r,w,h)

# set up the ROI for tracking
roi = frame[r:r+h, c:c+w]
hsv_roi =  cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)

roi_hist = cv2.calcHist([hsv_roi],[0],None,[180],[0,180])
cv2.normalize(roi_hist,roi_hist,0,255,cv2.NORM_MINMAX)

# Setup the termination criteria, either 10 iteration or move by atleast 1 pt
term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1 )

while(1):
    ret ,frame = cap.read()

    if ret == True:
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    dst = cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)
    # apply meanshift to get the new location
    ret, track_window = cv2.CamShift(dst, track_window, term_crit)
    # Draw it on image
    pts = cv2.boxPoints(ret)
    pts = np.int0(pts)
    img2 = cv2.polylines(frame,[pts],True, 255,2)
    cv2.imshow('img2',img2)

    k = cv2.waitKey(60) & 0xff
    if k == 27:
        break

    cv2.destroyAllWindows()
    cap.release()
```

الگوریتمOptical flow

یا جریان نوری الگوریتمی است برای آشکار سازی حرکت یک شی در تصویر در دو فریم متوالی که ناشی از حرکت دوربین یا شی باشد. این یک بردار دو بعدی است که هر بردار حرکت نقاط از فریم اول به فریم دوم را نشان می‌دهد. تصویر زیر را در نظر بگیرید.



در این تصویر یک توپ، در ۵ فریم متوالی نشان و فلش سفید رنگ بردار جابه‌جایی را نشان می‌دهد. Optical flow برای کار هایی نظیر ساختار حرکت، حذف لرزش دوربین، فشرده سازی فیلم و... کاربرد دارد.

Optical flow بر روی فرضیه‌های زیر کار می‌کند :

- شدت پیکسل‌های یک شی بین فریم‌های متوالی تغییر نمی‌کند.
- پیکسل‌های همسایه حرکت مشابهب دارند

پیکسل $I(x,y,t)$ را در نظر بگیرید. دقت شود که چون ما با قیلم سر و کار داریم، بعد جدید t به مشخصه‌های پیکسل اضافه شده. در فریم بعدی با گذشت dt ، این پیکسل به مقدار (dx , dy) جابه‌جا می‌شود. چون شدت این پیکسل در این دو فریم تغییر نمی‌کند، خواهیم داشت $I(x,y,t) = I(x+dx , y+dy , t+dt)$ که با استفاده از سری تیلور و تقسیم طرفین بر dt خواهیم داشت $f_x u+f_y v+f_t=0$ که

$$f_x = \frac{\partial f}{\partial x};\ f_y = \frac{\partial f}{\partial y}$$

$$u = \frac{dx}{dt};\ v = \frac{dy}{dt}$$

که با حل معادله فوق می‌توان dx , dy را بدست آورد. برای حل این معادله که دارای دو مجهول است، از روش لوکاس کاناد استفاده می‌شود. در opencv کلیه این عملیات توسط تابع calcOpticalFlowPyrLK() انجام می‌شود. این تابع به صورت زیر می‌باشد.

calcOpticalFlowPyrLK (prevImg , nextImg , prevPts , nextPts , status , err , winSize , Maxlvl , criteria) -> nextPts , status , err

- prevImg : تصویر یا فریم قبلی که باید در فضای رنگ خاکستری ۸ بیتی باشد.
- nextImg : تصویر یا فریم فعلی که باید در فضای رنگ خاکستری ۸ بیتی باشد.
- prevPts : بردار مختصات نقاط در فریم قبلی که می‌خواهیم دنبال کنیم. مختصات هر نقطه باید به صورت x,y مشخص شود.
- nextPts : مختصات جدید نقاط ورودی که در تصویر دوم، محاسبه شده. اگر پرچم OPTFLOW_USE_INITIAL_FLOW استفاده شود. تعداد نقاط ورودی و خروجی یا به عبارت دیگر، اندازه بردار ورودی و خروجی برابر خواهد بود.
- Status : بردار خروجی وضعیت است و برای هر ویژگی، عنصر یا نقطه، اگر روند ردیابی ادامه یابد و نقطه دنبال شود، مقدار ۱ و در غیر این صورت مقدار ۰ را برای آن نقطه در بردار وضعیت، بر می‌گرداند.
- Err : بردار خروجی خطاها است و برای هر ویژگی، عنصر یا نقطه، مقدار خطا را باز می‌گرداند. نوع اندازه‌گیری خطا را می‌توان در پارامتر flags تنظیم کرد. اگر برای نقطه‌ای روند ردیابی قطع شود، برای آن نقطه، خطا تعریف نمی‌شود. (از پارامتر status برای پیدا کردن چنین مواردی استفاده می‌شود.)
- WinSize : اندازه پنجره جست و جو در هر مرحله، که هر چه سرعت حرکت شی مورد نظر بیشتر باشد، پنجره جست و جو نیز باید بزرگ تر شود.
- Criteria: شرط توقف جست و جو را معین می‌کند که به صورت یک سه‌تایی مشخص میشود. ورودی اول شروط توقف ، ورودی دوم ماکزیمم تعداد حرکت و ورودی سوم حداقل میزان حرکت بر اساس پیکسل را معین می‌کند.

Criteria = (condition , Max number of move , epsilon Moving)

```
Condition= cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT
```

Flags : پرچم های عملیاتی

OPTFLOW_USE_INITIAL_FLOW : اگر از این پرچم استفاده شود، از مقدار پارامتر nextPts که تحت ورودی به تابع دادیم، به عنوان تخمین اولیه استفاده می‌شود. در غیر این صورت مقدار prevPts در nextPts کپی شده و این مقدار به عنوان تخمین اولیه استفاده می‌شود

OPTFLOW_LK_GET_MIN_EIGENVALS : اگر از این پرچم استفاده شود، برای محاسبه خطا از روش min eigen استفاده می‌شود که اگر از حداقل مقدار محاسبه شده کمتر باشد، آن نقطه به عنوان یک ویژگی ضعیف تلقی شده و حذف می‌شود و باعث بهبود عملکرد می‌شود. اما اگر این پرچم استفاده نشود، برای محاسبه خطا، از تقسیم فاصله بین نقاط ویژگی تقسیم بر تعداد پیکسل های یک پنجره، استفاده می‌شود.

مثال : در این مثال ما با استفاده از الگوریتم shi-Tomasi با تابع goodFeatursToTrack() نقاط کلیدی را در اولین فریم پس از زدن کلید space پیدا کرده و با دادن مختصات این نقاط، به همراه فریم فعلی، قبلی و سایر پارامترها به تابع calcOpticalFlowPyrLK()، نقاط ویژگی را ردیابی میکنیم و نقاطی را که پارامتر status، آن‌ها ۱ است، یا به عبارت دیگر نقاطی را که ردیابیشان موفق بوده است را رسم می‌کنیم.

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

shiTomasiParams = dict ( maxCorners = 100 , qualityLevel = 0.3 , minDistance = 5 , blockSize = 7
)
lkParams = dict( winSize  = (15,15),maxLevel = 2, criteria = (cv2.TERM_CRITERIA_EPS |
cv2.TERM_CRITERIA_COUNT, 10, 0.03))

while 1:
    ret ,frame = cap.read ()
    frameGray = cv2.cvtColor ( frame , cv2.COLOR_BGR2GRAY )

    prevPts = cv2.goodFeaturesToTrack ( frameGray , **shiTomasiParams )

    for point in prevPts :

        frame = cv2.circle ( frame , tuple(point.ravel()) , 2 , (0,0,255) , thickness=-1)

    cv2.imshow ( " WebCam " , frame )
    key = cv2.waitKey (5)

    if key == 32 :
        break
#------------------------------------------------

oldgray = frameGray.copy()

while 1 :

    ret ,newFrame = cap.read ()
    newgray = cv2.cvtColor ( newFrame , cv2.COLOR_BGR2GRAY )

    newPts, status , err = cv2.calcOpticalFlowPyrLK  ( oldgray , newgray , prevPts , None ,
**lkParams )

    correctPts = newPts [ status == 1 ]

    for point in correctPts :

        newFrame = cv2.circle ( newFrame , tuple(point.ravel()) , 2 , (0,100,255) , thickness=-1)


    cv2.imshow ( " WebCam " , newFrame )
    key = cv2.waitKey (30)

    oldgray = newgray.copy()
```

الگوریتم Background Subtraction

هدف در این بخش حذف پس‌زمینه از فیلم، می‌باشد. تشخیص پس‌زمینه یکی از مراحل اولیه در برنامه های بینایی است. حذف پس زمینه در برنامه‌هایی نظیر شمارش تعداد بازدیدکنندگان از یک محل، به وسیله شمارش تعداد افرادی که از آن وارد یا خارج می‌شوند و یا دوربین‌های ترافیکی که اطلاعاتی راجب به وسایل نقلیه بدست می‌آورند، کاربرد فراوان دارد. از نظر فنی باید بتوان پیش‌زمینه متحرک را از پس زمینه ثابت جدا کرد. اگر بتوان یک عکس خالی از پس‌زمینه مانند عکس اتاق بدون بازدید کننده یا جاده‌ی بدون وسیله نقلیه تهیه کرد، کار خیلی ساده است و کافی است که این تصویر را از هر فریم دوربین منها کنیم تا پیش‌زمینه بدست آید؛ اما در اقلب موارد ما چنین تصویری نخواهیم داشت و لازم است که پس‌زمینه را از با استفاده از همان تصویری که داریم استخراج کنیم. وجود سایه می‌تواند کار را پیچیده‌تر کند، زیرا سایه نیز حرکت می‌کند و به عنوان پیش‌زمینه تشخیص داده می‌شود. در opencv سه الگوریتم برای حذف پس زمینه ارایه شده که در زیر به معرفی آن‌ها می‌پردازیم.

الگوریتم BackGroundSubtractorMOG

این یک الگوریتم پیش‌زمینه، پس زمینه گاوسی است. این الگوریتم در سال ۲۰۰۱ معرفی شد. در این الگوریتم از مدل ترکبی توزیعی گاوسی برای مدل‌سازی هر پیکسل استفاده می‌کند. وزن ترکیب گاوسی در این الگوریتم نشان دهنده میزان زمانی است که رنگ پیکسل مذکور، ثابت است. پیکسل‌های پس زمینه آن‌هایی هستند که زمان نسبت طولانی، ثابت هستند

در opencv برای استفاده از این الگوریتم، ابتدا لازم است که از کلاس ()creatBackgrpundSubtractorMOG از ماژول cv۲.bgsegm یک شی ایجاد نمود، سپس در حلقه اجرای فیلم با استفاده از تابع apply در کلاس فوق، فریم را در هر مرحله به تابع دهیم و یک ماسک به عنوان خروجی دریافت کنیم که این ماسک، ماسک forceground یا پیش زمینه است و بخش‌های پس زمینه در آن به رنگ مشکی هستند.

creatBackgrpundSubtractorMOG (history , nmixturs , backgroundRatio , noiseSigma) -> mask

history : طول حافظه را تعیین می‌کند که تا چند فریم را به عنوان history در نظر بگیرد که مقدار پیشفرض ۲۰۰ است.

nmixtur : تعداد مخلوط گاوسی را مشخص می‌کند. مقدار پیشفرض آن برابر ۵ است

BackgroundRatio : نسبت ابعاد پس زمنیه که مقدار پیشفرضش ۰.۷ است

noiseSigma : قدرت نویز را مشخص می‌کند (انحراف استاندار در فضاهای رنگی) . مقدار ۰ به معنای انتخاب مقدار به صورت خود کار است.

مثال :

```
 import cv2

cap = cv2.VideoCapture (0)
bgs = cv2.bgsegm.createBackgroundSubtractorMOG ()

while True :

    ret,frame = cap.read ()
    forcegroundMask = bgs.apply ( frame )

    result = cv2.bitwise_and ( frame , frame , mask = forcegroundMask )
    cv2.imshow ( "mask" , forcegroundMask )
    cv2.imshow ( "forceground" , result )
    cv2.imshow ( "orginall " , frame )

    cv2.waitKey (30)
```

الگوریتم BackGroundSubtractorMOG2

این یک الگوریتم توسعه سافته الگوریت قبلی بوده. یکی از ویژگی‌های مهم این الگوریتم، انتخاب تعداد توزیعی گاوسی برای هر پیکسل به طور مجزا است که باعث بهبود تشخیص پس زمینه شده. برای استفاده از این الگوریتم، ابتدا باید از کلاس creatBackgrpundSubtractorMOG۲() یک شی ایجاد نمود و سپس در حلقه اجرای فیلم با استفاده از تابع apply در کلاس فوق، فریم را در هر مرحله به تابع دهیم و یک ماسک به عنوان خروجی دریافت کنیم که این ماسک، ماسک forceground یا پیش زمینه است

creatBackgrpundSubtractorMOG۲(history , varThreshold , detectShadow) -> mask

- history : طول حافظه را تعیین می‌کند که تا چند فریم را به عنوان history در نظر بگیرد که مقدار پیشفرض ۵۰۰ است.
- VarThreshold : یک حد آستانه برای انکه معین کند یک پیکسل آیا به درستی به عنوان background انتخاب شده است یا خیر.
- detectShow : اگر این پارامتر true باشد، الگوریتم سایه را نیز تشخیص می‌دهد و با رنگ خاکستری در mask نشان می‌دهد که این کار سرعت را کاهش می‌دهد. بنابراین در مواردی که نیازی به این مورد نیست، بهتر است false باشد که به طور پیشفرض false است.

مثال :

```
import cv2

cap = cv2.VideoCapture (0)
bgs = cv2.createBackgroundSubtractorMOG2 ()

while True :

    ret,frame = cap.read ()
    forcegroundMask = bgs.apply ( frame )

    result = cv2.bitwise_and ( frame , frame , mask = forcegroundMask )
    cv2.imshow ( "mask" , forcegroundMask )
    cv2.imshow ( "forceground" , result )
    cv2.imshow ( "orginall " , frame )

    cv2.waitKey (30)
```

الگوریتم BackGroundSubtractorGMG

این الگوریتم ترکیبی است از تخمین آماری تصویر و تقسیم‌بندی بیزی برای هر پیکسل. این الگوریتم در سال ۲۰۱۲ معرفی شد. در این الگوریتم از چند فریم اول (به طور پیشفرض ۱۲۰) برای مدل سازی بک گراند استفاده می‌کند. سپس با استفاده از الگوریتم قطعه‌بندی احتمالی پیش زمینه که اشیا پیش‌زمینه را توسط استنتاج بیزی (Bayesian) استخراج نماید. تخمین‌ها به صورت انطباقی هستند. به این صورت که مشاهدات جدید، وزن بیشتری از مشاهدات قبلی دارند تا متناسب با نور متغیر باشد.

برای استفاده از این الگوریتم باید از کلاس creatBackgrpundSubtractorMOG() از ماژول cv۲.bgsegm یک شی ایجاد نمود، سپس در حلقه اجرای فیلم با استفاده از تابع apply در کلاس فوق، فریم را در هر مرحله به تابع دهیم و یک ماسک به عنوان خروجی دریافت کنیم که این ماسک، ماسک forceground یا پیش زمینه است و بخش‌های پس زمینه در آن به رنگ مشکی هستند.

creatBackgrpundSubtractorMOG (initialzationFrames , decisionThreshold) -> mask

- initialzationFrames: تعداد فریمی که برای مدل سازی پس‌زمینه استفاده می‌شود.
- decisionThreshold : حد آستانه که برای بالاتر از آن پیش‌زمینه و کمتر از آن پس‌زمینه خواهد بود

مثال :

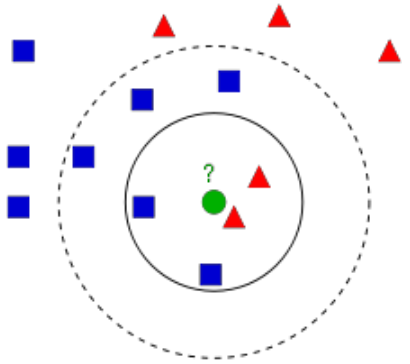
در مثالزیر برای حذف نویز عملیات morphological روی ماسک خروجی انجام می‌گیرد.

```
import cv2

cap = cv2.VideoCapture (0)

kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3))
fgbg = cv2.bgsegm.createBackgroundSubtractorGMG()

while(1):
    ret, frame = cap.read()
    fgmask = fgbg.apply(frame)
    fgmask = cv2.morphologyEx(fgmask, cv2.MORPH_OPEN, kernel)
    cv2.imshow('frame',fgmask)
    cv2.waitKey(30)
```

یادگیری ماشین

الگوریتم KNN (k nearest neighbour)

الگوریتم KNN یکی از ساده ترین الگوریتم‌های موجود برای یادگیری ماشین تحت نظارت است. ایده این الگوریتم پیدا کردن بیشترین تطابق ها در فضای ویژگی میان داده های مورد آزمون است. به تصویر زیر توجه نمایید



در تصویر بالا دو خانواده مثلث قرمز و مربع آبی وجود دارد که محل هرکدام در نقشه بالا نشان داده شده است که ما به آن فضای ویژگی می‌گوییم (فضای ویژگی در واقع یک فضایی است که تمام داده‌ها و حالات را پیشبینی کند. مثلا یک فضای ۲D را در نظر بگیرید. در این فضا هر داده دارای دو ویژگی است که همان مختصه‌های X,Y است. حال اگر ما سه ویژگی داشته باشیم، به یک فضای ۳D نیاز داریم مانند یک رنگ RGB که میتوان تمام حالات آن را در یک فضای سه بعدی رسم کرد. همچنین مشخص است که اگر ما n ویژگی داشته باشیم، به یک فضای nD نیازمندیم) در تصویر بالا یک عضو جدید که با دایره سبز نشان داده شده، به این خانواده اضافه می‌شود که با دایره‌ی سبز رنگ نشان داده شده. حال ماشین باید تصمیم بگیرد که این عضو جدید متعلق به کدام خانواده باشد. در الگوریتم KNN ماشین نزدیک ترین همسایه‌ی این عضو جدید را بررسی میکند و آن را به همان خانواده نسبت می‌دهد. همان‌طور که مشخص است در تصویر بالا مثلث قرمز به دایره سبز نزدیکتر بوده و این عضو جدید به این خانواده تعلق می‌گیرد.

اما در اینجا یک مشکل وجود دارد، شاید نزدیک ترین همسایه به عضو جدید مثلث قرمز باشد، اما چی میشه اگر این عضو همسایه‌های آبی زیادی داشته باشد.. در این صورت مربع‌های آبی قدرت بیشتر در آن منطقه دارند. پس بررسی نزدیکترین همسایه شاید خیلی مناسب نباشد. به جای آن ما K تا از نزدیکترین همسایه های عضو جدید را بررسی می‌کنیم. برای مثال فرض کنید K=۳ باشد در این صورت ۲ تا از نزدیک ترین همسایه‌ها مثلث قرمز و تنها یکی مربع آبی است. که در این صورت عضو جدید باز هم به خانواده مثلث قرمز تعلق می‌گیرد اما اگر K=۵ باشد، عضو جدید دارای ۳ همسایه مربع آبی و ۲ همسایه مثلث قرمز است که در این صورت به خانواده مربع آبی تعلق می‌گیرد. اما اگر K=۴ باشد تعداد همسایه های آبی و قرمز برابر است پس بهتر است که K همواره عددی فرد انتخاب شود. اما در این حالت یا به صورت کلی آیا صحیح است که ما دو همسایه مثلث قرمز را که خیلی

نزدیکتر هستند را هم ارزش با دو مربع آبی بدانیم؟ از این رو ما علاوه بر بررسی تعداد همسایگی، فاصله آن‌ها را هم از عضو جدید، به عنوان یک معیار در نظر میگیریم و بر اساس این فاصله ها الگوریتم تعدادی وزن به هر خانواده اختصاص می‌دهد. به این صورت که برای همسایه‌های نزدیک یک وزن بزرگ و برای همسایه‌های دورتر یک وزن کوچک در نظر می‌گیرد. سپس وزن های مختلف هر خانواده را باهم جمع می‌کنیم و خانواده‌ای که مجموع وزن هایش بیشتر باشد، عضو جدید به آن خانواده تعلق میگیرد. این تعمیم الگوریتم KNN است.

مثال :

در مثال زیر ما مانند تصویر بالا یک فضای ویژگی دوبعدی با دو خانواده ایجاد میکنیم خانواده اول را با مقدار ۰ مشخص میکنیم و آن‌ها را با مثلث قرمز و خانواده دوم را با مقدار ۱ مشخص میکنیم و آن‌ها را با مربع آبی رسم می‌کنیم.در این مثال ابتدا به کمک تابع numpy.random.randint() تعدادی داده به صورت رندوم ایجاد می‌کنیم. که ویژگی ها یا همان مختصات آن‌ها را در trainData و کلاس یا خانواده آن‌ها را در responses ذخیره میکنیم که نوع درایه های هردوی این آرایه‌ها برای استفاده در الگوریتم knn باید float۳۲ باشد. سپس مختصات داده های دو خانواده را جدا و در دو لیست جداگانه red و blue ذخیره می‌کنیم و آن ها را با تابع plt.scatter() رسم میکنیم.

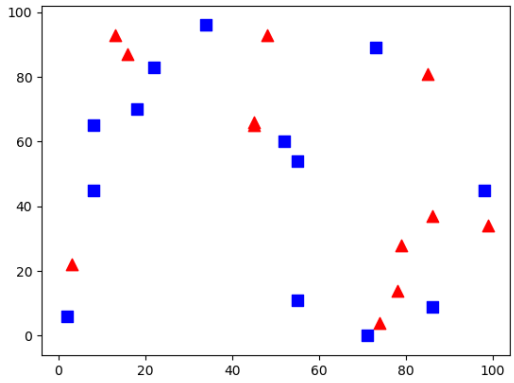
numpy.random.randint(low , high , size)

plt.scatter(x , y , Scaller , color , marker shape)

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

#set 25 numbers of (x,y)
trainData=np.random.randint (0 , 100 , (25,2)).astype (np.float32 )

#set 25 numbers of (class number)
responses=np.random.randint (0 , 2 , (25,1)).astype (np.float32 )

red=trainData [ responses.ravel()==0]
blue=trainData [ responses.ravel()==1]

plt.scatter ( red[:,0] , red[:,1], 80 , 'r' , '^')
plt.scatter ( blue[:,0] , blue[:,1], 80 , 'b' , 's')
plt.show ()
```

در

تابع scatter ، رنگ ها و اشکال دارای نماد اختصاری هستند برای مثال '^' نماد اختصاری برای مثلث است



پس از انجام مراحل فوق، یک عضو جدید ایجاد میکنیم و می‌خواهیم با استفاده از الگوریتم بررسی کنیم که این عضو متعلق به کدام خانواده است. برای استفاده از الگوریتم Knn ابتدا یک شی از کلاس cv۲.ml.KNearest_creat() تعریف میکنیم . سپس با استفاده از تابع train() داده های معلومی که داریم که در این مثال در بالا ساختیم را به شی knn می‌دهیم یا به اصطلاح به ماشین آموزش می‌دهیم.

Train (trainData , flage , responses)

TrainData : آرایه از مختصات داده ها (مقدار ویژگی های داده مربوطه)

Flages : پرچم را مشخص میکند که در اینجا پرچم ROW_SAMPLE توصیه می‌شود

Response : آرایه که خانوده یا کلاس ها را مشخص میکند و با trainData نظیر به نظیر است. یعنی برای مثال درایه سوم از آرایه‌ی TrainData مختصه داده‌ای را معین می‌کند که عضو خانواده است که در درایه سوم response معین شده

پس از یاددهی به ماشین حال میخواهیم مشخص کنیم که نقطه جدید متعلق به کدام خانواده است. برای اینکار از تابع findNearst() استفاده می‌کنیم.

findNearest (sample , k) -> retVal , result , neighbors , dist

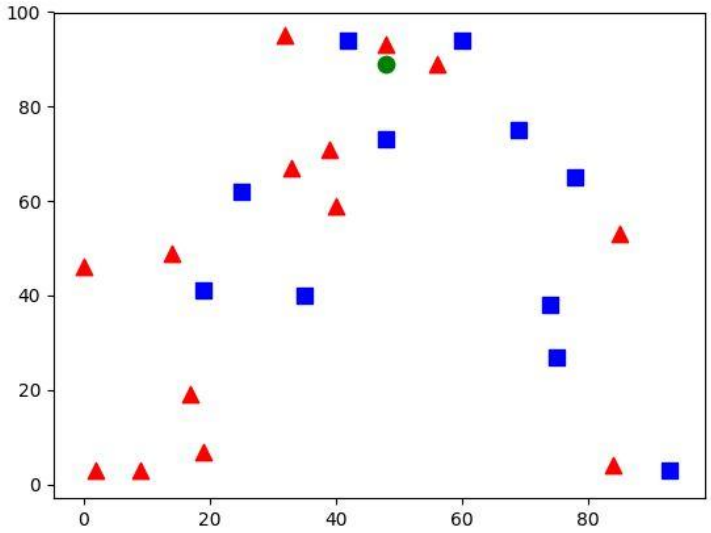
sample : داده یا داده‌های جدید که میخواهیم بررسی کنیم

K : تعداد همسایه ها را مشخص می‌کند

این تابع ۴ خروجی دارد که خروجی result مشخص می‌کند نقطه جدید متعلق به کدام کلاس است، neighbors همسایه‌های یافت شده را بر اسا کلاسشان نشان می‌دهد و خروجی dist مقدار فاصله هر کدام از این همسایگی ها را مشخص می‌کند.

پس کد بالا با کد زیر تکمیل می‌شود

```
newcomer = np.random.randint(0,100,(1,2)).astype(np.float32)
plt.scatter(newcomer[:,0],newcomer[:,1],80,'g','o')

#creat knn and train & use
knn = cv2.ml.KNearest_create()
knn.train(trainData, cv2.ml.ROW_SAMPLE, responses)
_,result,neighbors , dist =knn.findNearest ( newcomer , 3 )

print( " result : " , result )
print( " neighbors : " , neighbors )
print( " dist : " , dist )

plt.show ()
```




```
========== RESTART: C:\Users\Amir_PC\Desktop\TrV.py ==========
 result :  [[0.]]
 neighbors :  [[0. 1. 0.]]
 dist :  [[16. 61. 64.]]
```

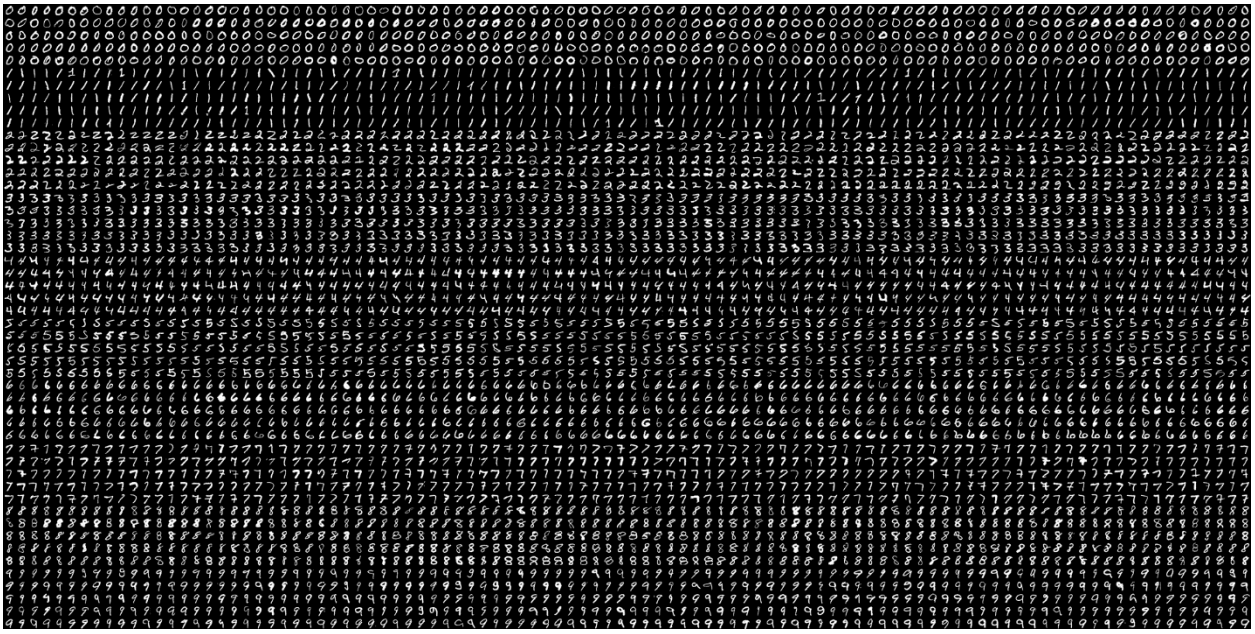
اگر sample به جای یک نقطه، آرایه ای از نقاط باشد، خروجی‌های تابع findNearest() نیز به صورت آرایه خواهند بود. برای مثال درایه اول از خروجی های neighbors , result , dist نتیجه مربوط به نقطه ای است که در درایه اول sample است

```
# 10 new comers
newcomers = np.random.randint(0,100,(10,2)).astype(np.float32)
ret, results,neighbours,dist = knn.findNearest(newcomer, 3)
# The results also will contain 10 labels.
```

```
neighbors :  [[1. 0. 1.]
[1. 0. 1.]
[0. 1. 1.]
[1. 0. 0.]
[0. 0. 0.]
[0. 1. 1.]
[1. 1. 0.]
[1. 0. 1.]
[0. 0. 1.]
[0. 1. 1.]]
```

ساخت برنامه OCR برای خواندن اعداد لاتین

در این بخش قصد داریم یک نرم افزار OCR برای خواندن دست خط حروف و اعداد انگلیسی بنویسیم. ابتدا با برنامه خواندن اعداد شروع میکنیم . در ابتدا لازم است تا ما با استفاده از تعداد داده و الگوریتم knn به ماشین آموزش دهیم برای اینکار یک تصویر آماده با نام digits.png موجود است که کلیه ارقام ۰ تا ۹ را با دست خط های مختلف دارد. این تصویر شامل ۵۰ ردیف است که هر ۵ ردیف برای یک رقم بوده. هر ردیف نیز به خودی خود دارای ۱۰۰ ستون است که در هر ستون شامل رقم مربوطه با دست خطی منحصر، می‌شود. هر کدام از ارقام 20×20 پیکسل است. برای شروع ما تصویر را به ۵۰ ردیف و ۱۰۰ ستون تقسیم میکنیم. از ۵۰۰ تصویری که برای هر رقم موجود است، ۲۵۰ عدد را برای آموزش و ۲۵۰ عدد را برای تست ماشین در نظر می‌گیریم. ویژگی استفاده شده در اینجا شدت روشنایی پیکسل‌های تشکیل دهنده است که با اندازه 20×20 ما ۴۰۰ پیکسل و در نتیجه ۴۰۰ ویژگی خواهیم داشت. برای ساخت آرایه response که برچسب و نام خانواده هر عضو را مشخص میکند، از آنجا که میدانیم برای هر ۱۰ رقم 50×5 دست خط داریم، با کمک تابع arrange() یک لیست از ۰ تا ۹ می‌سازیم و با کمک تابع repeat() هر عضو این لیست را ۲۵۰ با تکرار میکنیم و از آنجا که نام هر کلاس خود باید به صورت یک لیست باشد مثلا برای عدد یک باید برچسب به صورت [۱] باشد و نه به صورت ۱، پس از اینرو دستور [np.newaxis,:] را در ادامه آن می‌آوریم تا هر برچسب به صورت یک لیست باشد عدد داده تست برای بررسی درصد یادگیری ماشین است که هرچه این درصد بالاتر باشد بهتر است. اما ۱۰۰ درصد یادگیری ممکن است همیشه خوب نباشد و ماشین تنها داده های آموزش دیده را درست تشخیص دهد و داده ای خارج از داده های آموزشی را اشتباه تشخیص دهد پس در بسیاری مواقع، ۹۸ درصد صحت میتواند بهتر از ۱۰۰ درصد باشد. نکته دیگر آن که داده هایی که ماشین در تست درست تشخیص ندهد را میتوان به آن آموزش داد تا یادگیری ماشین بهتر شود.



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

img=cv2.imread ("digits.png")
gray=cv2.cvtColor (img,cv2.COLOR_BGR2GRAY)

#split image to 50*100 picture
cells = np.array ([np.hsplit(row , 100 ) for row in np.vsplit (gray,50)])

#divide data to train and test and reshape
train=cells[:, 0:50].reshape (-1,400).astype(np.float32)
test =cells[:,50:100].reshape (-1,400).astype(np.float32)

#creat lable for train and test data
k = np.arange(10)
train_labels = np.repeat(k,250)[:,np.newaxis]
test_labels = train_labels.copy()

#trian and test machin
knn=cv2.ml.KNearest_create()
knn.train (train ,cv2.ml.ROW_SAMPLE , train_labels)
retval , result , neighbors , dist= knn.findNearest( test , k=5)

#calc accuracy of machin learning
matches= ( result==test_labels)
correstNum=np.count_nonzero(matches)
testNum=len(test)
accuracy = (correstNum/testNum)*100
print ( " accuracy of machin learning " , accuracy)

#Accuary is 91.7%
```

در اینجا کد پایه برنامه مورد نظر ما آماده شده است. که یادگیری ۹۱.۷ درصد را نتیجه داده است. همچنین میتوان داده های Knn را با کد زیر ذخیره کرد و هرگاه نیاز بود از آن‌ها استفاده نمود

```
np.savez('knn_data.npz',train=train, train_labels=train_labels)

# Now load the data
with np.load('knn_data.npz') as data:
    print( data.files )
    train = data['train']
    train_labels = data['train_labels']
```

برای بهینه سازی میتوان داده ها را قبل از ذخیره به نوع uint۸ تبدیل کرد و سپس ذخیره نمود و برای استفاده داده هارا لود کرده و دوباره به نوع float۳۲ تبدیل کرد.

ساخت برنامه OCR برای خواندن حروف لاتین

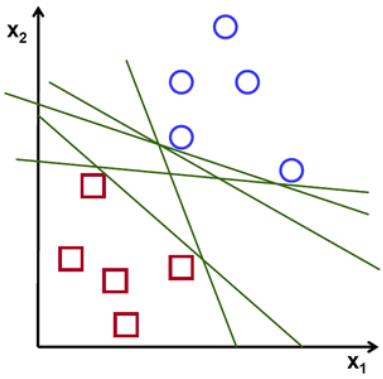
در اینجا به جای استفاده از یک تصویر، از یک فایل داده آماده به نام letter-recognition که توسط opencv ارایه شده استفاده میکنیم. اگر این فایل را به صورت متنی باز کنید ۲۰۰۰۰ خط را مشاهده خواهید کرد. در هر ردیف، ستون اول نام یک حرف لاتین است که همان برچسب ما است و بعد از آن ۱۶ شماره آمده که همان ویژگی های ما است. این ویژگی ها از یادگیری ماشین UCI بدست آمده. ما از این ۲۰۰۰۰ دستخط، ۱۰۰۰۰ تای اول را برای یادگیری و ۱۰۰۰۰ تای دوم را برای تست ماشین استفاده می‌کنیم. همچنین ستون اول را از سایر داده ها جدا می‌کنیم که برچسب های کلاس ها را می‌سازد و بقیه داده‌ها بردار ویژگی ها است. نتیجه یادگیری در این آزمون ۹۳ درصد بود. در رابطه با این دیتاست و ویژگی های استفاده شده در آن می‌توانید در لینک زیر بیشتر بخوانید http://archive.ics.uci.edu/ml/datasets/Letter+Recognition

```
T,2,8,3,5,1,8,13,0,6,6,10,8,0,8,0,8
I,5,12,3,7,2,10,5,5,4,13,3,9,2,8,4,10
D,4,11,6,8,6,10,6,2,6,10,3,7,3,7,3,9
N,7,11,6,6,3,5,9,4,6,4,4,10,6,10,2,8
G,2,1,3,1,1,8,6,6,6,6,5,9,1,7,5,10
S,4,11,5,8,3,8,8,6,9,5,6,6,0,8,9,7
B,4,2,5,4,4,8,7,6,6,7,6,6,2,8,7,10
A,1,1,3,2,1,8,2,2,2,8,2,8,1,6,2,7
J,2,2,4,4,2,10,6,2,6,12,4,8,1,6,1,7
M,11,15,13,9,7,13,2,6,2,12,1,9,8,1,1,8
.
.
```

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
# Load the data, converters convert the letter to a number
data= np.loadtxt('letter-recognition.data', dtype= 'float32', delimiter = ',',
                    converters= {0: lambda ch: ord(ch)-ord('A')})
# split the data to two, 10000 each for train and test
train, test = np.vsplit(data,2)
# split trainData and testData to features and responses
responses, trainData = np.hsplit(train,[1])
labels, testData = np.hsplit(test,[1])
# Initiate the kNN, classify, measure accuracy.
knn = cv.ml.KNearest_create()
knn.train(trainData, cv.ml.ROW_SAMPLE, responses)
ret, result, neighbours, dist = knn.findNearest(testData, k=5)
correct = np.count_nonzero(result == labels)
accuracy = correct*100.0/10000
print( accuracy )
```

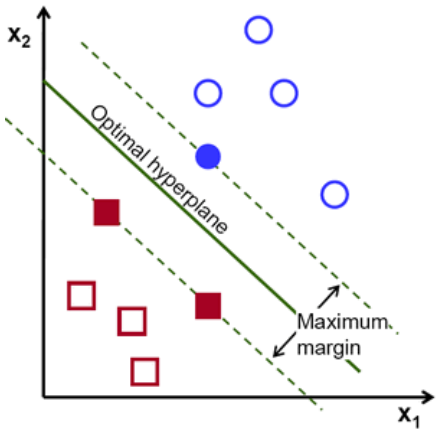
یادگیری ماشین با الگوریتم SVM

تصویر زیر دارای دو خانواده قرمز و آبی است. در الگوریتم knn برای هر داده آزمایشی، فاصله آن از همه داده ها را اندازه میگرد و حداقل فاصله را انتخاب می‌کند که اینکار زمان صرف می‌کند همچنین ذخیره همه داده های آموزشی نیز حافظه زیادی را مصرف می‌کند. اما باتوجه به اطلاعات داده شده در تصویر اینکار واقعا نیاز است؟



ایده دیگری را در نظر بگیرید. ما خطی را پیدا میکنیم $f(x) = ax_1 + bx_2 + c$ که داده ها را به دو ناحیه تقسیم کند حال داده جدید X وارد می‌شود . اگر $f(X) > 0$ باشد این داده متعلق به خانواده آبی و در غیر این صورت به خانواده قرمزها تعلق دارد . ما می‌توانیم این خط را به عنوان مرز تصمیم در نظر گرفت که بسیار ساده و کار آمد خواهد بود. داده هایی را که بتوان با یک خط(یا هایپر پلین در ابعاد بالاتر) تقسیم کرد را تفکیک پذیر خطی می‌گویند.

همانطور که از تصویر بالا مشخص است، خطوط زیادی، داده‌ها را ناحیه بندی می‌کنند اما کدام یک را باید در نظر گرفت، پاسخ خطی است که بیشترین فاصله را از همه نقاط داشته باشد. زیرا در داده های ورودی نویز وجود دارد و این نویز ها نباید دقت طبقه بندی را تحت تاثیر قرار دهند بنابراین دورترین خط ، حاشیه اطمینان بالاتری را دارد. به تصویر زیر دقت نمایید



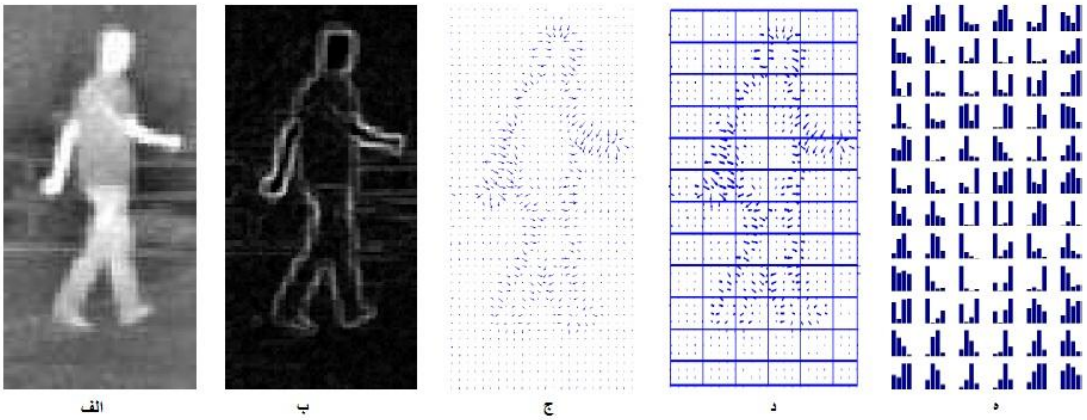
بنابراین ما برای پیداکردن این خط به داده های آموزشی نیاز داریم. اما آیا به همه آن‌ها نیازمندیم؟ خیر، ما تنها به داده‌هایی نیاز داریم که در نزدیک و همسایگی گروه مخالف هستند که در شکل بالا به صورت توپر رسم شده اند. این داده ها را ما بردار پشتیبانی و خطوط که از میان آن‌ها می‌گذرند را صفحه پشتیبانی، می‌نامیم. این موضوع در کاهش اطلاعات و حافظه بسیار موثر است.

برنامه خواندن دست خط (OCR) با الگوریتم SVM :

در بخش قبل ما از شدت پیکسل‌ها به طور مستقیم به عنوان بردار ویژگی استفاده کردیم اما در این بخش از HOG یا هیستوگرام گرادیان جهت دار استفاده میکنیم

HOG : HOG (Histogram of Oriented Gradients) یا همان هیستوگرام گرادیان جهت‌دار یک بردار ویژگی (Feature Vector) است که در پردازش تصویر برای تشخیص اشیاء مورد استفاده قرار می‌گیرد. مفهوم اصلی HOG این است که شکل یک شیء در تصویر را می‌توان با توزیع شدت گرادیان‌ها یا جهت‌گیری لبه‌ها توصیف کرد. به عبارت دیگر، با بررسی چگونگی تغییرات روشنایی و جهت لبه‌ها در بخش‌های مختلف تصویر، می‌توان مشخصه‌ای ساخت که ساختار و شکل شیء را به صورت عددی و قابل مقایسه بیان کند.

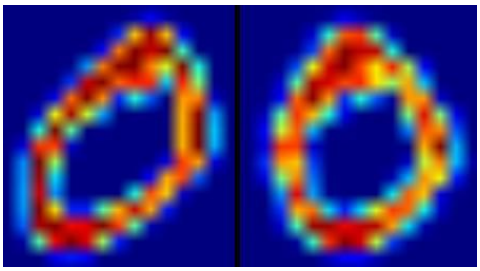
این ویژگی‌ها سپس برای شناسایی اشیاء، تشخیص انسان، یا طبقه‌بندی تصاویر در الگوریتم‌های یادگیری ماشین و بینایی ماشین استفاده می‌شوند..



برای بهره از HOG ابتدا لازم است که پیچش یا warp تصویر را تصحیح نموده و تصویر را صاف نماییم. اینکار توسط تابع زیر انجام می‌گیرد.

```
def deskew(img):
    SZ=20
    m = cv2.moments(img)
    if abs(m['mu02']) < 1e-2:
        return img.copy()
    skew = m['mu11']/m['mu02']
    M = np.float32([[1, skew, -0.5*SZ*skew], [0, 1, 0]])
    img = cv2.warpAffine(img,M,(SZ, SZ),flags=affine_flags)
    return img
```

تصویر زیر، تصویر از عدد صفر است که در آن تصویر سمت چپ تصویر ورودی ما است که دارای پیچش بوده و تصویر سمت راست تصویر خروجی است که پیچش آن رفع شده.



پس از این مرحله نوبت به محاسبه HOG برای هر سلول (تصویر هر رقم) می‌رسد که در تابع hog() انجام می‌گیرد برای درک این تابع، ابتدا به توضیح توابع استفاده شده در آن می‌پردازیم.

تابع cartToPolar() : این تابع یک بردار دوبعدی یا لیستی از آن هارا بر اساس (x,y) دریافت نموده و اندازه و زاویه آن‌ها را محاسبه می‌کند که زاویه خروجی بر اساس رادیان است

Magnitude , angle = cv۲.cartToPolar(x , y)

تابع zip() : این تابع دو لیست میگیرد و درایه های آن‌ها را نظیر به نظیر ترکیب می‌کند

List۳=zip (list۱ , list۲)

List۱= [۱ , ۳ , ۵ , ۷] list۲=[۲ , ۴ , ۶ , ۸] -> list۳= [(۱,۲) , (۳,۴) , (۵,۶) , (۷,۸)]

تابع bincount() : این تابع در حالت عادی یک آرایه با درایه های مثبت دریافت میکند و تعداد هر مقدار موجود در آرایه ورودی را می‌شمارد و تعداد آن‌ها را در یک آرایه خروجی قرار می‌دهد . به این صورت که درایه صفرم آرایه خروجی ، تعداد صفرها در آرایه ورودی ، درایه یکم تعداد یک ها در آرایه ورودی و ... را نشان می‌دهد

Bincount (array , weights , min_len)

Array : آرایه ورودی

Weight : آرایه وزن ها که طول آن باید برابر طول آرایه خروجی باشد. در این حالت به جای شمردن تعداد هر رقم، وزن های موجود در درایه‌های نظیر در آرایه weights را باهم جمع می‌کند.

Min_len : حداقل طول آرایه خروجی را مشخص می‌کند

مثال :

```
>>> arr = [ 2,1,3,1,0,2,2]
>>> Bin = np.bincount ( arr )
>>> Bin
array([1, 2, 3, 1], dtype=int32)
```

در لیست بالا ما یک صفر داریم، پس مفدار درایه صفرم برابر ۱ ، همچنین دو عدد یک داریم پس درایه یکم برابر ۲ و الی آخر

```
>>> arr = [ 1,3,4,1,3,0,3]
>>> wei = [ 5,10,8,12,9,6,11]
>>> Bin =np.bincount( arr ,wei)
>>> Bin
array([ 6., 17.,  0., 30.,  8.])
```

در لیست بالا ما علاوه بر آرایه یکسری وزن به تابع دادیم اما عملکرد به چه صورت خواهد بود در اینجا به جای تعداد، مجموع وزن های هم درایه در نظر گرفته می‌شود. برای مثال ما یک صفر داریم که درایه پنجم است که درایه پنجم در لیست wei برابر ۶ است، پس درایه صفرم Bin برابر ۶ خواهد بود. همچنین ما دو عدد ۱ در خانه‌های صفرم و سوم داریم که مقدار این خانه ها در لیست wei برابر ۵ و ۱۲ است که مجموع آن‌ها برابر ۱۷ است پس درایه یکم در bin برابر ۱۷ میگردد و الی آخر.

```
>>> arr = [ 1,3,4,1,3,0,3]
>>> wei = [ 5,10,8,12,9,6,11]
>>> Bin =np.bincount( arr ,wei , 8)
>>> Bin
array([ 6., 17.,  0., 30.,  8.,  0.,  0.,  0.])
```

در بالا چون ما حداقل طول را ۸ قرار دادیم پس تابع حداقل تا درایه هفتم را مقدار دهی میکند که چون ما ارقام ۵و۶و۷ در آرایه ورودی نداریم، مقدار آن‌ها صفر می‌باشد

پس از درک توابع بالا به سراغ تابع hog() می‌رویم. برای محاسبه هیستوگرام گرادیان جهت دار ما ابتدا مشتقات سوبل را در دوجهت x و y محاسبه می‌کنیم. سپس اندازه و زاویه گرادیان هر پیکسل را محاسبه می‌کنیم. سپس مقدار زاویه را به ۱۶ کوانتیزه می‌کنیم. یعنی زاویه، عددی بین ۰ تا ۱۶ می‌شود. سپس تصویر را به ۴ بخش تقسیم کرده و در هر بخش هیستوگرام گرادیان جهت را با وزن خودش محاسبه می‌شود. یعنی برای هر زاویه از پیکسل ها، مجموع اندازه آن پیکسل ها را در نظر می‌گیریم. برای مثال اگر ما ۴ پیکسل با زاویه ۳۰ درجه داشته باشیم که اندازه آن‌ها مثلا ۴ و ۶ و۷ و ۲ باشد، در هسیتوگرام HOG طول bin زاویه ۳۰ درجه برابر جمع این مقادیر می‌باشد که برابر ۱۹ است. به عبارتی دیگر در هیستوگرام HOG محور افقی زاویه پیکسل ها و محور عمودی مجموع اندازه پیکسل‌های موجود با آن زاویه می‌باشد. از این رو ما برای هر یک از چهار قسمت تصویر چون ۱۶ زاویه داریم در نتیجه بردار ویژگی آن ۱۶ بعدی است که یک لیست بوده و شماره درایه آن نشان دهنده زاویه و مقدار هر درایه نشان دهنده مجموع اندازه پیکسل‌های موجود با آن زاویه است که برای کل تصویر یعنی هر چهار قسمت، ۶۴ ویژگی داریم. کلیه این مراحل در تابع hog() انجام می‌شود.

```
def hog(img):
    gx = cv2.Sobel(img, cv2.CV_32F, 1, 0)
    gy = cv2.Sobel(img, cv2.CV_32F, 0, 1)
    mag, ang = cv2.cartToPolar(gx, gy)
    bins = np.int32(bin_n*ang/(2*np.pi))    # quantizing binvalues in (0...16)
    bin_cells = bins[:10,:10], bins[10:,:10], bins[:10,10:], bins[10:,10:]
    mag_cells = mag[:10,:10], mag[10:,:10], mag[:10,10:], mag[10:,10:]
    hists = [np.bincount(b.ravel(), m.ravel(), bin_n) for b, m in zip(bin_cells, mag_cells)]
    hist = np.hstack(hists)     # hist is a 64 bit vector
    return hist
```

پس از تعرف توابع deskew() و hog() همانند قبل عمل می‌کنیم. ابتدا توصویر digits.png را فرا خوانده و آن را به ۵۰۰۰ بخش تقسیم می‌کنیم که شامل ۵۰۰ نمونه برای هر رقم است که ۲۵۰ تای آن را برای آموزش و مابقی را برای تست استفاده میکنیم. برای استفاده از الگوریتم SVM ابتدا یک شی از کلاس SVM() از ماژول cv۲.ml تعریف می‌کنیم. و سپس آن را به صورت زیر تنظیم می‌کنیم

Svm.setKernel (cv۲.ml.SVM_LINEAR)

Svm.setType (cv۲.ml.SVM_C_SVC)

Svm.setC (۲.۶۷)

Svm.setGamma (۵,۳۸۳

-برای آموزش از تابع train() همانند الگوریتم Knn استفاده می‌کنیم

train(trainData , flag , response)

- TrainData : آرایه از مختصات داده ها (مقدار ویژگی های داده مربوطه)
- Flages : پرچم را مشخص میکند که در اینجا پرچم ROW_SAMPLE توصیه می‌شود
- Response : آرایه که خانوده یا کلاس ها را مشخص میکند و با trainData نظیر به نظیر است. یعنی برای مثال درایه سوم از آرایه‌ی TrainData مختصه داده‌ای را معین می‌کند که عضو خانواده است که در درایه سوم response معین شده

برای تشخیص کلاس یک عضو جدید از تابع predict() استفاده می‌کنیم

predict (test Data) -> retVal , result

- testData : بردار ویژگی داده یا داده های جدید

خروجی این تابع هم خانواده داده یا داده‌های ورودی است.

-همچنین برای ذخیره کردن میتوانیم از تابع save() استفاده کنیم و آن را با پسوند .dat و نامی دلخواه ذخیره نماییم

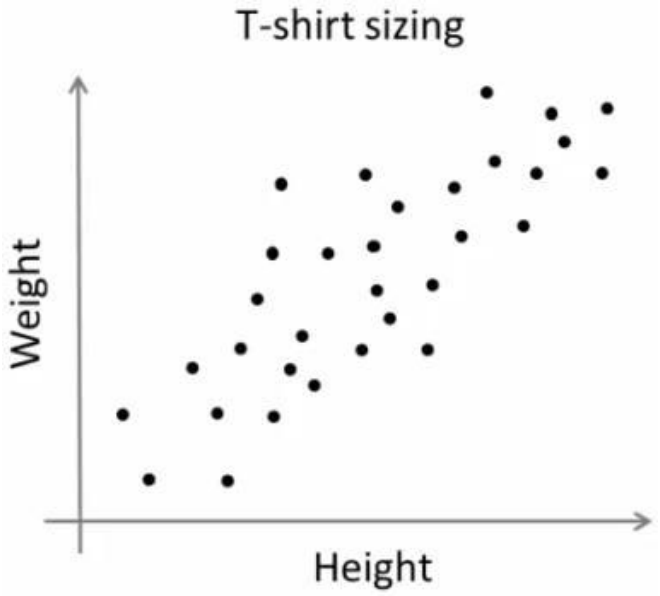
Smv.save ('myName.dat')

نتیجه ای که ما در اینجا بدست آوردیم ۹۴ درصد بود که نتیجه قابل قبولی است و می‌توان با دادن داده های آموزشی جدید آن را بهبود بخشید

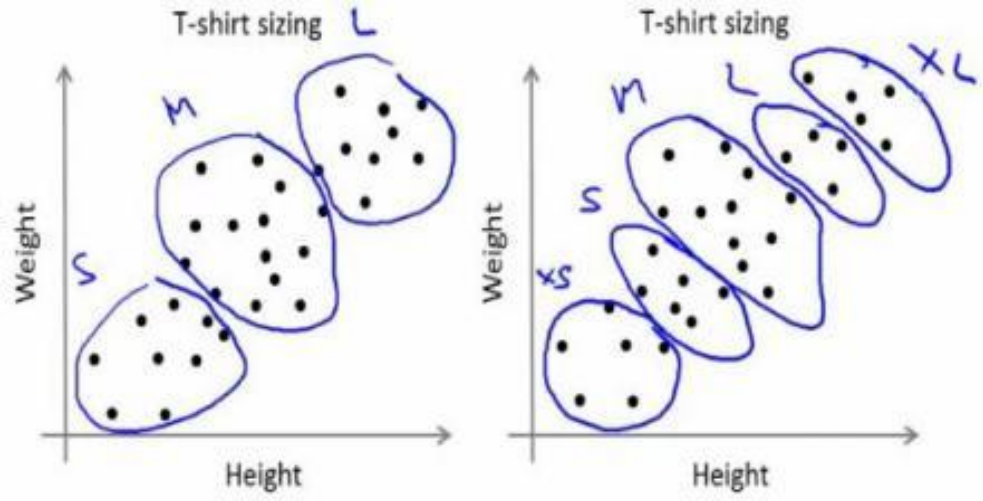
```
import cv2
import numpy as np

SZ=20
bin_n = 16

#_____________________________________________________________

def deskew(img):
    m = cv2.moments(img)
    if abs(m['mu02']) < 1e-2:
        return img.copy()
    skew = m['mu11']/m['mu02']
    M = np.float32([[1, skew, -0.5*SZ*skew], [0, 1, 0]])
    img = cv2.warpAffine(img,M,(SZ, SZ),flags=cv2.WARP_INVERSE_MAP|cv2.INTER_LINEAR)
    return img
#_____________________________________________________________

def hog(img):
    gx = cv2.Sobel(img, cv2.CV_32F , 1, 0)
    gy = cv2.Sobel(img, cv2.CV_32F , 0, 1)
    mag, ang = cv2.cartToPolar(gx, gy)
    bins = np.int32(bin_n*ang/(2*np.pi))    # quantizing binvalues in (0...16)
    bin_cells = bins[:10,:10], bins[10:,:10], bins[:10,10:], bins[10:,10:]
    mag_cells = mag[:10,:10], mag[10:,:10], mag[:10,10:], mag[10:,10:]
    hists = [np.bincount(b.ravel(), m.ravel(), bin_n) for b, m in zip(bin_cells, mag_cells)]
    hist = np.hstack(hists)     # hist is a 64 bit vector
    return hist
#_____________________________________________________________

img = cv2.imread( "digits.png" ,0)
cells = np.array([np.hsplit(row,100) for row in np.vsplit(img,50)])

train_cells=cells[:,0:50]
test_cells =cells[:,50:100]

deskeweds = [list(map(deskew,row)) for row in train_cells]
hogdata   = [list(map(hog,row)) for row in deskeweds]

trainData = np.float32(hogdata).reshape(-1,64)
responses = np.repeat(np.arange(10),250)[:,np.newaxis]
#_____________________________________________________________

svm = cv2.ml.SVM_create()
svm.setKernel(cv2.ml.SVM_LINEAR)
svm.setType(cv2.ml.SVM_C_SVC)
svm.setC(2.67)
svm.setGamma(5.383)
svm.train(trainData, cv2.ml.ROW_SAMPLE, responses)
svm.save('svm_data.dat')

#_____________________________________________________________

deskeweds = [list(map(deskew,row)) for row in test_cells]
hogdata =   [list(map(hog,row)) for row in deskeweds]
testData = np.float32(hogdata).reshape(-1,bin_n*4)

_,result = svm.predict(testData)

mask = result==responses
correct = np.count_nonzero(mask)
print(correct*100.0/result.size)
```

الگوریتم خوشه‌بندی K-Means

یک شرکت را در نظر بگیرید که قرارا است تی‌شرت جدیدی را به بازار عرضه کند. بدیهی است که این شرکت باید تی‌شرت‌هایی با سایزهای مختلف تولید کند که همه مردم با سایزهای مختلف بتوانند از آن‌ها استفاده کنند. بنابراین این شرکت اطلاعاتی از قد و وزن افرادی را بر روی گرافی، مانند زیر قرار می‌دهند.

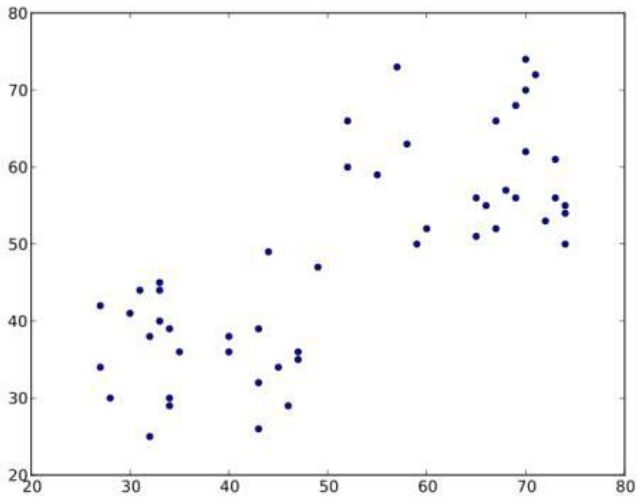


مشخصا شرکت نمی‌تواند در تمامی سایزها لباس تولید کند. در عوض داده‌ها را به چند دسته‌ی مثلا کوچک، متوسط و بزرگ تقسیم می‌کند و تنها این ۳ مدل را تولید می‌کند که برای همه مردم مناسب است. این تقسیم‌بندی افراد به سه گروه، می‌تواند با الگوریتم خوشه بندی K-Means انجام شود و الگوریتم، ۳ سایز را که برای همه مردم راضی کننده است، ارایه دهد و اگر این گونه نبود؛ شرکت می‌تواند تعداد گروه‌ها را افزایش دهد. برای درک بهتر به تصاویر زیر نگاه کنید.

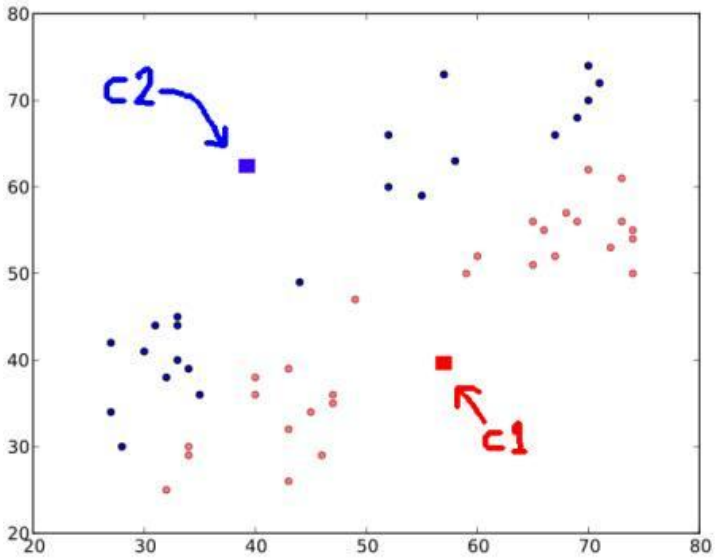


اما روش کارکرد این الگوریتم چگونه است؟

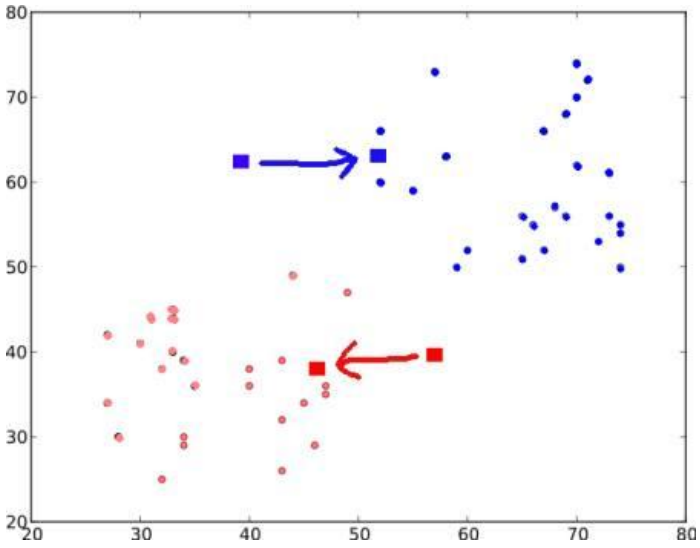
این الگوریتم، یک فرایند تکراری است. برای درک این فرایند، ابتدا داده‌های زیر را در نظر بگیرید. ما می‌خواهیم این داده ها را به دو گروه تقسیم کنیم.



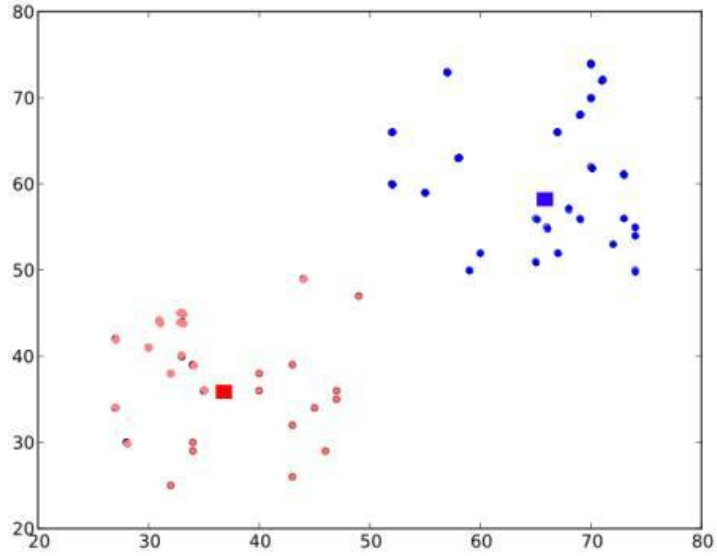
در مرحله اول، الگوریتم ابتدا دو نقطه c۲ , c۱ را به عنوان مرکز این گروه‌ها در نظر می‌گیرد که ممکن است c۱, c۲ خود جزوه داده باشند یا نباشند. سپس در مرحله دوم، الگوریتم فاصله داده‌ها را از هر دو نقطه محاسبه می‌کند. اگر داده تست به c۱ نزدیک‌تر باشد با برچسب ۰ و اگر به c۲ نزدیک‌تر باشد با برچسب ۱ مشخص می‌کند. (اگر گروه‌ها و مراکز بیشتر وجود داشته باشد، با برچسب‌های ۲، ۳ و... معین می‌شود). در تصویر زیر، داده ها با برچسب ۰ را با رنگ قرمز و داده‌ها با برچسب ۱ را با رنگ آبی مشخص نموده‌ایم.



در مرحله سوم، میانگین کلیه نقاط آبی و قرمز را به صورت جداگانه محاسبه می‌کنیم و نقاط به دست آمده را به عنوان مراکز جدید در نظر می‌گیریم که نقاط مرکزی c۱ و c۲ جدید را می‌سازند. پس از این مرحله دوباره مرحله دوم، یعنی برچسب گذاری داده‌ها نسبت به مراکز جدید تکرار می‌شود. برای درک بهتر به تصویر زیر توجه قرمایید.



در نهایت مراحل دوم و سوم را آنقدر تکرار می‌کنیم که نقاط مرکزی ثابت شوند و یا ممکن است براساس معیارهایی که ما ارائه می‌دهیم متوقف شوند که این معیارها ماکزیم تعداد حرکت و حداقل میزان حرکت برای مراکز است. این درکی کلی برای الگوریتم خوشه بندی k-means بود.


80
70
60
50
40
30
20
20
30
40
50
60
70
80

برای پیاده سازی این الگوریتم از تابع ()kmeans استفاده می‌کنیم

Kmeans (samples, nclusters, bestLables, criteria, attempts, flags)-> compactness, labels, centers

- Samples : داده‌های نمونه هستند که باید از نوع float۳۲ باشند و هر ستون متعلق به یک ویژگی باشد.
- Nclusters : تعداد خوشه‌بندی ها را مشخص می‌کند.
- bestLables : آرایه ورودی – خروجی که برچسب خوشه‌ها را برای هر داده ذخیره می‌کند.
- Criterisa : شرط متوقف شدن الگوریتم را مشخص می‌کند که به صورت یک سه‌تایی مشخص میشود. ورودی اول شروط توقف ، ورودی دوم ماکزیمم تعداد حرکت و ورودی سوم حداقل میزان حرکت بر اساس پیکسل را معین می‌کند.

Criteria = (condition , Max number of move , epsilon Moving)

```
Condition= cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT
```

- Attempts : این پارامتر، تعداد دفعاتی را که الگوریتم مراکز اولیه را محاسبه می‌کند، مشخص می‌کند. در این بین مراکزی انتخاب می‌شوند که با برچسب گذاری داده‌ها، داده‌هایی که هم گروه هستند، بیشترین فشردگی را داشته باشند. پس از انتخاب مراکز اولیه، فرایند تکرار برای رسیدن به بهترین گروه‌بندی، آغاز می‌شود.
- Flags : پارامتر پرچم روش بدست آوردن مراکز اولیه را مشخص میکند که می‌تواند یکی از حالات زیر باشد

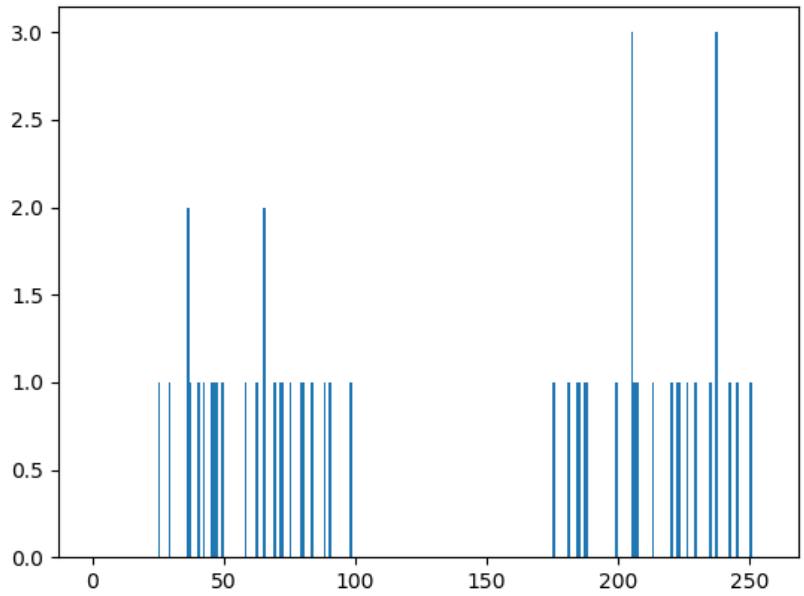
۱- KMEANS_RANDOM_CENTERS
۲- KMEANS_PP_CENTERS

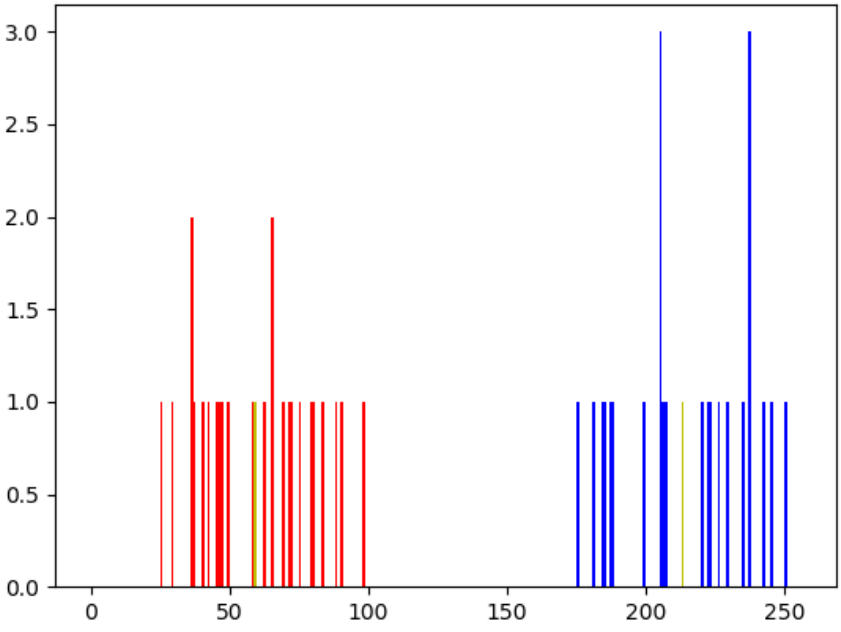
خروجی های این تابع نیز به صورت زیر می‌باشد

- Compotness : این پارامتر برابر است با مجموع مربعات فاصله نقاط هر خوشه نسبت به مرکزشان که میزان فشردگی هر خوشه را نشان می‌دهد.
- Lables : آرایه‌ای است که پرچسب داده‌ها را نشان می‌دهد. این پرچسب‌ها به صورت ۰،۱،۲ و... است
- Centers : آرایه است که مراکز خوشه‌ها را نشان می‌دهد.

مثال : در مثال زیر می‌خواهیم مجموعه‌ای از داده‌ها تنها با یک ویژگی را خوشه بندی کنیم. برای این که داده ها در دو گروه پراکنده باشند، ایتدا دو مجموعه X و Y در بازه دلخواه تولید کرده و سپس باهم ادغام می‌کنیم و مجموعه z را ایجاد می‌کنیم که داده‌های آن در بازه ۰، تا ۲۵۵ قرار دارد. در نهایت، الگوریتم kmeans را روی آن اعمال کرده و داده‌ها را خوشه بندی می‌کنیم.

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

# creat sample data----------------------------------------------------
x = np.random .randint ( 25,100 , 25)
y = np.random .randint ( 175,255 , 25)

z = np.hstack ((x,y))
z = z.reshape ( (50,1) )
z = np.float32(z)
#-------------------------------------------------------------------------

plt.hist ( z , 256 , [0,256] )
plt.show ()

# Kmean algoritm-------------------------------------------------------
criteria = ( cv2.TERM_CRITERIA_COUNT | cv2.TERM_CRITERIA_EPS , 10 , 0.1 )

compactness,labels,centers = cv2.kmeans ( z , 2 , None , criteria , 10 ,
cv2.KMEANS_RANDOM_CENTERS)
#-------------------------------------------------------------------------

A = z [ labels==0 ]
B = z [ labels==1 ]
plt.hist ( A , 256 , [0,256] , color = 'red' )
plt.hist ( B , 256 , [0,256] , color = 'blue' )
plt.hist ( centers , 256 , [0,256] , color = 'y')
plt.show ()
```


3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
50
100
150
200
250

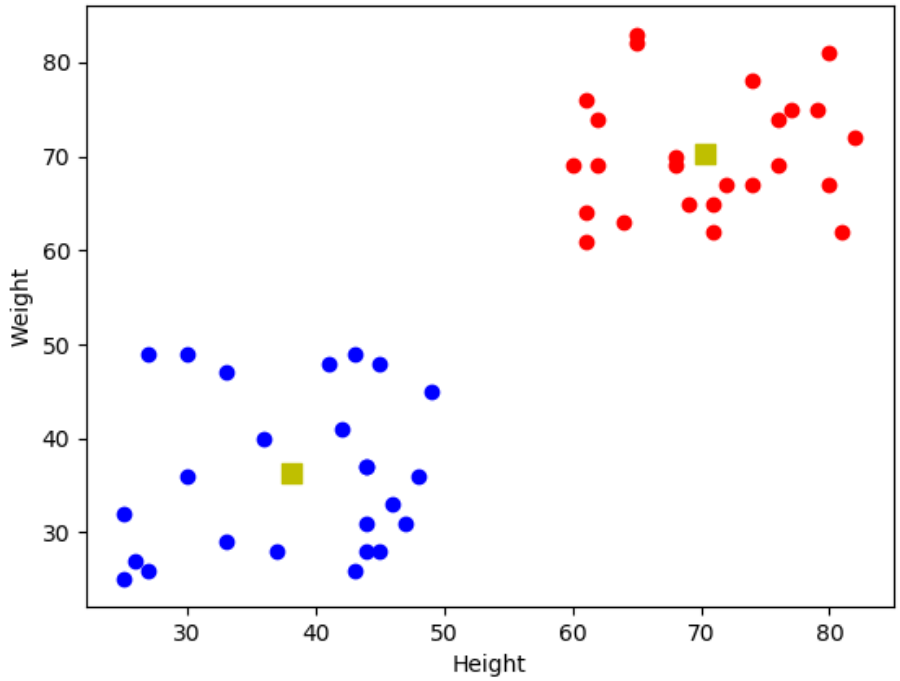

3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
50
100
150
200
250

مثال : در مثال قبل داده‌های ما تنها دارای یک ویژگی بودند اما در این مثال می‌خواهیم از داده‌ها با دو ویژگی مثلا قد و وزن استفاده کنیم. همان طور که پیش از این گفتیم هر ویژگی باید در یک ستون باشد، پس ما برای ۵۰ داده نیازمند آرایه‌ای با ابعاد 50×2 نیازمندیم که ستون اول آن متعلق به ویژگی قد و ستون دوم آن متعلق به ویژگی وزن است. و مثلا برای نفر اول، درایه (۰,۰) قد آن، و درایه (۰,۱) وزن آن را نشان می‌دهد. به شکل زیر توجه فرمایید.

Features

	Height	Weight
Person 1	H1	W1
Person 2	H2	W2
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
Person 50	H50	W50

```
import numpy as np
import cv2 as cv

from matplotlib import pyplot as plt
X = np.random.randint(25,50,(25,2))
Y = np.random.randint(60,85,(25,2))

Z = np.vstack((X,Y))
Z = np.float32(Z)

criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 10, 1.0)
ret,label,center=cv.kmeans(Z,2,None,criteria,10,cv.KMEANS_RANDOM_CENTERS)

A = Z[label.ravel()==0]
B = Z[label.ravel()==1]

plt.scatter(A[:,0],A[:,1],c = 'b')
plt.scatter(B[:,0],B[:,1],c = 'r')
plt.scatter(center[:,0],center[:,1],s = 80,c = 'y', marker = 's')
plt.xlabel('Height'),plt.ylabel('Weight')
plt.show()
```



مثال : در مثال زیر می‌خواهیم تعداد رنگ‌های یک تصویر را کاهش دهیم به این کار color Quantization یا کوانتیزاسیون رنگ می‌گویند. این کار برای موارد نظیر کاهش حجم تصویر کاربرد دارد یا در بعضی موارد بعضی دستگاه‌ها تنها می‌توانند رنگ‌های محدودی را نشان دهند. از این رو نیاز است تا رنگ‌های یک تصویر را کوانتیزه کنیم. در این مثال ما سه کانال رنگی سبز و قرمز و آبی را به عنوان سه ویژگی در نظر می‌گیریم و مانند قبل عملیات خوشه بندی را دنبال می‌کنیم.

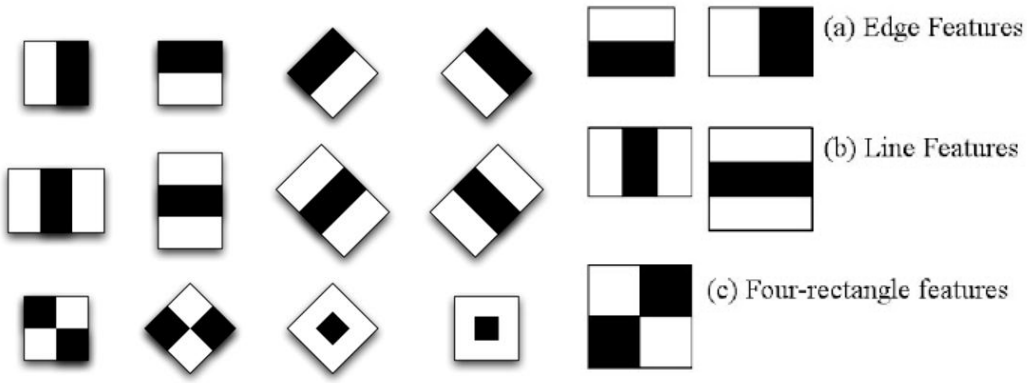
```
import numpy as np
import cv2

img = cv2.imread('dragon.jpg')
Z = img.reshape((-1,3))
Z = np.float32(Z)

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
K = 2
components,label,center=cv2.kmeans(Z,K,None,criteria,10,cv2.KMEANS_RANDOM_CENTERS)

center = np.uint8(center)

result=[]
for l in label :
    result.append (center[l])
result = np.array ( result)

result = result.reshape((img.shape))

cv2.imshow('res2',result)
cv2.waitKey(0)
cv2.imwrite ( "image" + str(K) + ".jpg" , result )
cv2.destroyAllWindows()
```

در زیر تصویر اصلی را به همراه تصاویر کوانتیزه شده با ۸ , ۳ , ۲ = k را آورده ایم.

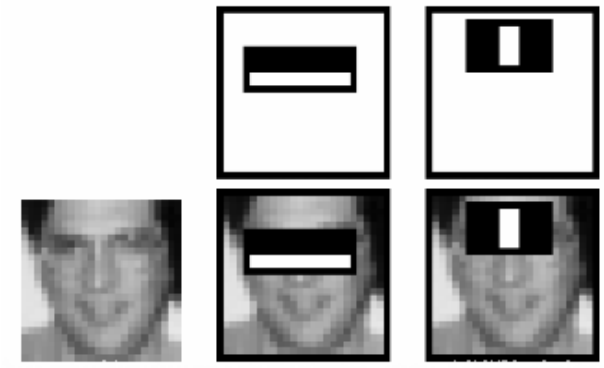
تشخیص شی در opencv (تشخیص چهره با haar–cascades)

تشخیص شی با استفاده از طبقه بندی آبشاری ها یک روش موثر برای تشخیص چهره است. این روش مبتنی بر یادگیری ماشین است که در آن یک تابع آبشاری با استفاده از تعداد زیادی تصاویر مثبت و منفی، برای تشخیص شی در تصاویر دیگر آموزش می‌بیند. در اینجا ما با تشخیص چهره کار خواهیم کرد. ابتدا الگوریتم به تعداد زیادی تصاویر مثبت (دارای چهره) و تصاویر منفی (بدون چهره) برای آموزش طبقه بندی، نیاز دارد. سپس باید ویژگی‌های آن را استخراج کنیم. برای اینکار از ویژگی های haar که برخی از آن‌ها تصویر زیر نشان داده شده‌اند استفاده میکنیم که هر ویژگی یک مقدار ثابت است و برابر است با :

مقدار Haar = مجموع پیکسل های زیر مستطیل مشکی - مجموع پیکسل های زیر مستطیل سفید



در این بخش تمام اندازه‌ها و محل‌های ممکن برای تمامی هسته‌ها برای محاسبه ویژگی‌های مورد استفاده واقع می‌شود. این موضوع حجم محاسبات را بسیار گسترده می‌کند و حتی برای پنجره‌ای 24×24 از یک تصویر بیش از ۱۶۰۰۰۰ ویژگی بدست می‌آید. و برای هر ویژگی باید مجموع پیکسل های زیر مستطیل‌های سیاه و سفید محاسبه شود. برای حل این مشکل از انتگرال تصویر بهره می‌بریم. اما از طرفی همه این ویژگی‌ها هم کارامد نیستند. برای مثال در تصویر زیر در ردیف بالا دو پنجره درنظر گرفته شده است که ویژگی های خوبی را در تصویر نشان می‌دهند. برای مثال در تصویر اول روی این ویژگی تمرکز دارد که چشم‌ها اغلب تیره از بینی هستند و یا در تصویر دوم بر این ویژگی تمرکز دارد که چشم‌ها معمولا از گونه و بینی تیره تر هستند. اما برای مثال همین هسته اگر بر روی قسمت های دیگر نظیر پیشانی یا گونه اعمال شود، بیهوده هستند. اما چگونه ویژگی‌های کارامد را پیدا کنیم



برای این موضوع ما تک تک ویژگی‌ها را روی تصاویر آموزشی اعمال می‌کنیم و برای هر ویژگی بهترین حد آستانه را که تصاویر آموزشی را به مثبت و منفی (در اینجا چهره و غیر چهره) تقسیم می‌کند را می‌یابیم. مشخصا در این بخش اشتباهات و خطاها پدیدار می‌شوند. ما ویژگی هایی را انتخاب می‌کنیم که حداقل خطا را داشته باشند یعنی ویژگی هایی که به طوز دقیق‌تر تصاویر مثبت و منفی را تقسیم بندی می‌کند. این یک فرایند پیچیده است. در نهایت ما مجموعه از ویژگی‌های ضعیف داریم. ضعیف به این منظور که هیچکدام از این ویژگی ها نمی‌توانند تقسیم بندی خوبی ارایه دهند اما این ویژگی‌ها در کنار هم یک طبقه بندی قوی ارایه می‌دهند. در یک مقاله حتی با ۲۰۰ ویژگی به دقت درصد دست یافته بوند و نسخه نهایی شامل ۶۰۰۰ ویژگی بود که نسبت به ۱۶۰۰۰۰ ویژگی یک کاهش چشمگیر است.

پس حال باید مثلا با استفاده از یک پنجره 24×24 و ۶۰۰۰ هزار ویژگی را روی یک تصویر اعمال کنیم تا ببینیم که آیا تصویر شامل شی مورد نظر هست یا خیر. اما این نیز زمان بر است. راه حل مشکل در درنظر گرفتن این موضوع است که عموما همه تصویر شامل شی مورد نظر نیست پس این ایده خوبی است که بررسی کنیم اگر یک پنجره چهره نیست، آن را کنار گذاشته و از پردازش دوباره آن پرهیز کنیم. در عوض تمرکز را بر روی بخش هایی از تصویر بگذاریم که احتمالا چهره هستند و زمان بیشتری را صرف مناطق ممکن می‌کنیم. برای این موضوع از طبقه آبشاری استفاده می‌کنیم . به این صورت که به جای آنکه هر ۶۰۰۰ ویژگی را روی یک پنجره اعمال کنیم. ویژگی ها را دسته بندی کرده و در مراحل مختلف اعمال کنیم و اگر یک پنجره در یک مرحله مردود شود در مراحل بعدی آن را در نظر نمی‌گیریم. و پنجره هایی در هر مرحله کنار گذشته می‌شوند و پنجره‌ای که در آخر می‌ماند شامل شی مورد نظر ما است. مشخصا در مراحل اولیه ویژگی‌های کمتری را بررسی می‌شود. برای مثال در مقاله مذکور، ۶۰۰۰ ویژگی در ۳۸ مرحله بررسی می‌شوند که در ۵ مرحله اول به ترتیب ۱, ۱۰ , ۲۵, ۲۵, ۵۰ ویژگی بررسی می‌گردد

کتابخانه opencv شامل یکسری پیش آموزش های آماده برای آبجکت‌هایی نظیر، صورت ، چشم، لبخند و... است که این پیش آموزش‌ها با پسوند xml. در مسیر / opencv / sources / data / haarcascades موجود است. در زیر یک مثال برای استفاده از این دیتاهای آماده آورده شده است

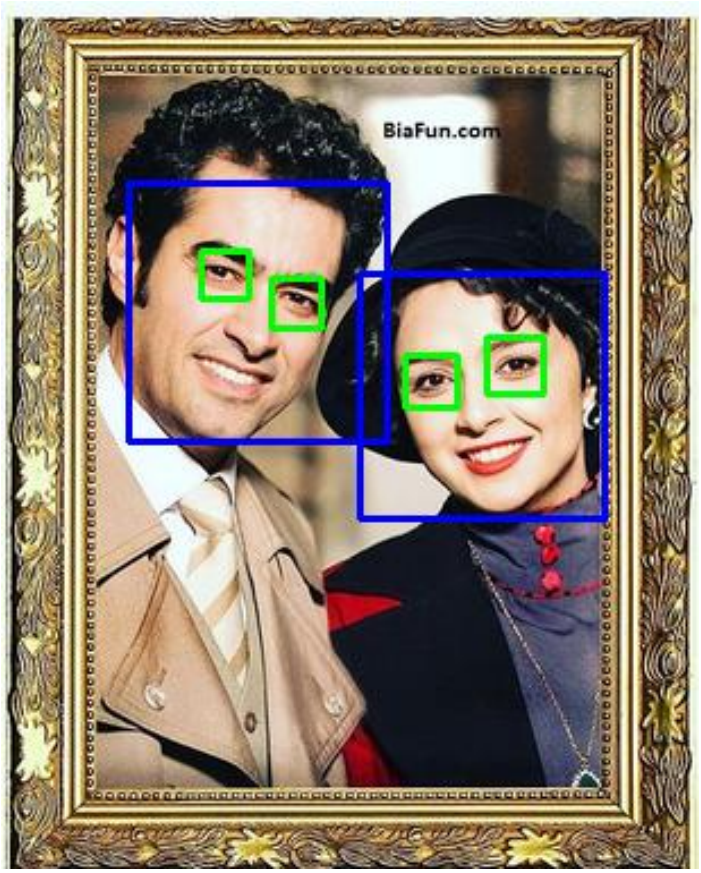
مثال : در این مثال با استفاده از پیش آموزش‌های طبقه‌بندی شدی opencv برای تشخیص چهره و چشم استفاده میک‌نیم. در ابتدا با لود کردن این قایل‌ها با استفاده از تابع detectMultiScale() چهره‌های موجود در تصویر را تشخیص می‌دهیم و در چهره‌ها با استفاده از همین تابع چشم‌ها را تشخیص می‌دهیم.

detectMultiScale (image , scaleFactor , minNeighbors) -> X, Y, W, H

- image :تصویر ورودی که می‌خواهیم شی را در آن تشخیص دهیم که نوع آن باید خاکستری باشد.
- scaleFactor : مشخص می‌کند که اندازه تصویر به چه میزان کاهش یابد.
- minNeighbors : حداقل تعداد قابل قبول همسایه‌ها را مشخص میکند.

خروجی این تابع نیز یک قاب مستطیلی است که محل شی مورد نظر را نشان می‌دهد و به فرمت x ,y ,w ,h می‌باشد.

```
import numpy as np
import cv2
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')

img = cv2.imread('sachin.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray, 1.3, 5)

for (x,y,w,h) in faces:

    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)

    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
cv2.imshow('img',img)
```



شناسایی چهره (face recognize) (اختیاری)

برای شناسایی یک چهره ما نیازمند به یادگیری متریک عمیق هستیم. در یادگیری و یادگیری عمیق ما یک تصویر به سیستم می‌دادیم و سیستم آن تصویر را کلاس‌بندی می‌کرد. اما در یادگیری متریک عمیق به جای برچسب گذاری، هدف دریافت یک بردار ویژگی قوی است. برای شبکه عصبی dlib ، این بردار ویژگی یک بردار یا ۱۲۸ مقدار عددی می‌باشد که ویژگی‌های تصویر را ارائه می‌دهد. این شبکه عصبی عمیق با معماری VAE توصیه یافته است که در آن شبکه عصبی با فشرده سازی تصویر و بازنمایی مجدد آن، آموزش می‌بینند.

برای مثال تصویر بالا را در نظر بگیرید که دو تا متعلق به ویل فرل و یکی متعلق به چاد اسمیت است. شبکه ما یک بردار شامل ۱۲۸ داده ۸ بیتی، برای هر تصویر ارایه می‌دهد و ما وزن‌ها را طوری اصلاح میکنیم که بردار ویژگی دو تصویز ویل فرل نزدیک به هم و متفاوت از بردار ویژگی تصویر چاد اسمیت باشند. لازم به ذکر است که خود این شبکه عصبی بر پایه دیتابیسی شامل ۳ میلیون عکس آموزش دیده است.

مقالات مفصلی برای نحوه شناسایی چهره در الگوریتم تشخیص چهره با کیفیت بالا با یادگیری عمیق متریک (ارایه شده توسط دیویس کینگ) و تشخیص چهره مدرن با یادگیری عمیق (ارایه شده توسط آدام گیتگی) ارائه شده است که می‌توانید برای درک بهتر به آن‌ها مراجعه کنید.

برای شروع لازم است که کتابخانه‌های face_recognition و dlib را نصب نموده. برای نصب کافی است دستورات زیر را در محیط cmd خود اجرا نمایید. لازم به ذکر است که پیش از این دو پکیج لازم است cmake را به همین روش نصب نمایید

```
pip install dlib
```

در صورتی که با دستور بالا نصب نشد دستور زیر را امتحان نمایید.

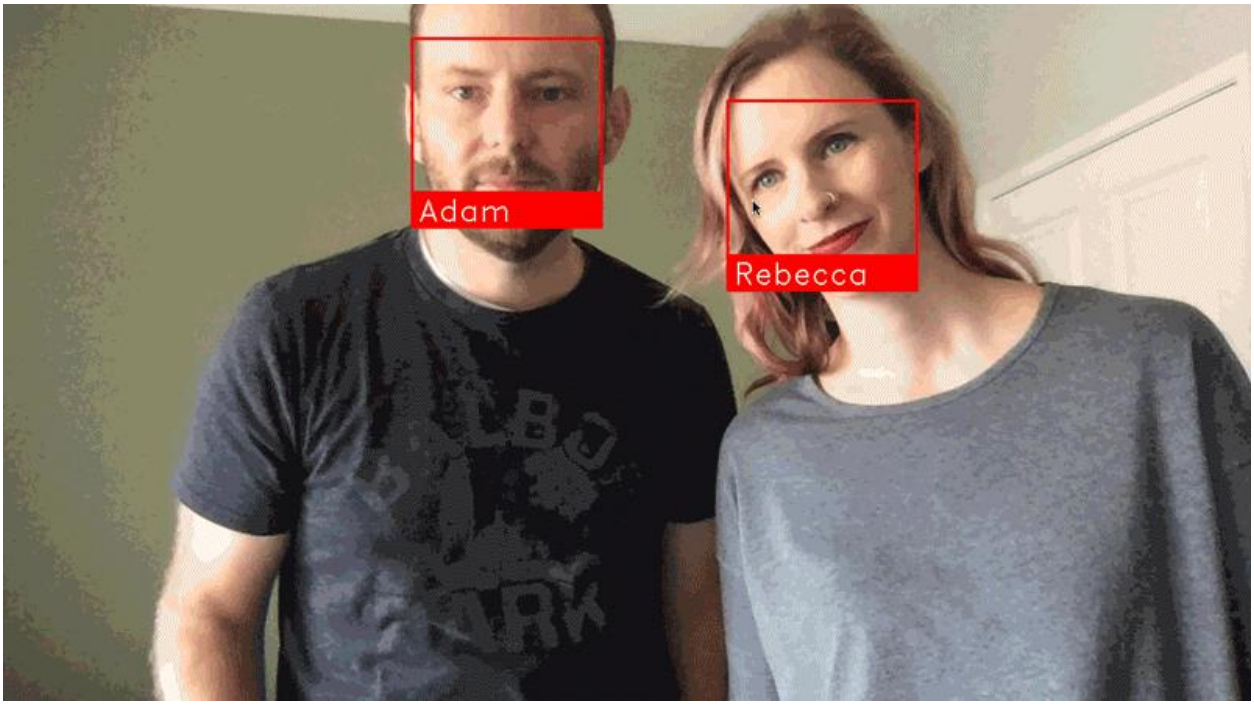
```
pip install https://pypi.python.org/packages/da/06/bd3e241c4eb0a662914b3b4875fc52dd176a9db0d4a2c915ac2ad8800e9e/dlib-19.7.0-cp36-cp36m-win_amd64.whl#md5=b7330a5b2d46420343fbed5df69e6a3f
```

```
pip install face_recognition
```

```
import face_recognition

picture_of_me = face_recognition.load_image_file("me.jpg")
my_face_encoding = face_recognition.face_encodings(picture_of_me)[0]

# my_face_encoding now contains a universal 'encoding' of my facial features that can be
compared to any other picture of a face!

unknown_picture = face_recognition.load_image_file("unknown.jpg")
unknown_face_encoding = face_recognition.face_encodings(unknown_picture)[0]

# Now we can see the two face encodings are of the same person with `compare_faces`!

results = face_recognition.compare_faces([my_face_encoding], unknown_face_encoding)

if results[0] == True:
    print("It's a picture of me!")
else:
    print("It's not a picture of me!")

#face_locations = face_recognition.face_locations(image)
# boxes = face_recognition.face_locations(rgb, model=args["detection_method"])
#encodings = face_recognition.face_encodings(rgb, boxes)
```

همانطور که مشخص است ابتدا یک تصویر از فرد مورد نظر لود میکنیم و بردار ویژگی آن را استخراج می‌کنیم، سپس یک تصویر ناشناخته را لود کرده و بردار ویژگی‌های آن را هم استخراج می‌کنیم و در نهایت بردار ویژگی دو تصویر را باهم مقایسه می‌کنیم.

بخش آخر که به صورن کامنت درآمده است برای استخراج صورت در یک تصویر است. نکته مهمی که باید بدان اشاره کرد در opencv تصاویر به صورت bgr هستند ولی در dlib تصاویر rgb بوده و باید در کار کردن با این دو کتابخانه این موارد را در نظر گرفت.


Adam
Rebecca