

بسم الله الرحمن الرحيم

گزارش کارآموزی:

پروژه PerfectShow

استاد ناظر:

امیدجعفرنژاد

محقق:

امیرحسین رستمی

دانشگاه صنعتی شریف

تابستان 99

مقدمه:

پروژه perfectShow یک پروژه‌ی openSource حوزه‌ی موبایل - بینایی ماشین است که امکان آرایش بر لحظه را در موبایل فراهم می‌کند. هدف از این پژوهش، بررسی و مطالعه جزئیات الگوریتمی بخش openCV این پروژه است. در این پروژه از تکنیک‌های زیر استفاده شده است:

1- استفاده از HaarFeatures

2- استفاده از SIFT

3- یافتن landmark های چهره

4- ... (در حال تکمیل)

مورد اول:

تکنیک اول جهت استخراج اجزای چهره (مثل گوش، چشم، بینی، دهان و...) است که برای آشنایی با نحوه‌ی عملکردی آن نیاز است که در مبحث توصیف گره‌های تصویر تعمیق کنیم تا ببینیم:

اولا چرا این روش انتخاب شده است؟ (Haar)

دوما این روش چگونه کار می‌کند و برتری‌اش بر سایر روش‌ها چیست؟

شرح مفصل این تکنیک در ادامه آورده شده است.

مورد دوم:

فرض کنید که ما به کمک تکنیک قبلی اجزای چهره شخص را استخراج کرده‌ایم، حال که روی آن‌ها تغییراتی اعمال می‌کنیم (مثلا آرایش می‌کنیم 😊)، نیاز است که به طریقی مثلا چشم آرایش شده شخص را بر چهره اصلی شخص بیاندازیم و حال چگونه این عمل امکان پذیر است؟

در پروژه (و نیز اغلب پروژه‌های بینایی ماشین) برای انجام این کار از تطبیق نقاط کلیدی چشم آرایش شده و چشم اصلی شخص استفاده می‌شود. ("چشم" صرفا یک مثال است)

که برای فهمیدن و تعمیق در الگوریتم استفاده شده نیاز است بدانیم که

اولا نقاط کلیدی چه نقاطی اند و چگونه استخراج می‌شوند؟

دوما روش SIFT چگونه کار می‌کند و چرا این روش به جای روش‌های دیگر موجود انتخاب شده است؟

برای شفاف تر شدن جزئیات این دو تکنیک، ابتدا مروری بر برخی مفاهیم بینایی ماشینی مرتبط می‌کنیم و سپس پله پله در تکنیک‌ها عمیق شده و در انتها با باقی متدهای موجود مقایسه‌شان می‌کنیم.

مورد سوم:

Landmark ها نقاطی بسیار مهم در تصویر اند که الگوی موجود در تصویر را نشان می دهند، مثلاً با مشاهده ی landmark های چهره به سرعت به الگوی چشم، بینی، ابرو و سایر اجزای چهره پی می بریم و از این الگو ها برای یافتن دقیق اجزای چهره و اعمال تبدیلات مختلف روی آنها استفاده می شود، در اصل با دانستن همین landmark ها به Mesh Mask چهره- که یک ابزار خیلی مفید جهت اعمال تغییرات روی چهره شخص است- دست می یابیم:

یک نمونه از Mesh Mask را در زیر مشاهده می کنید:



یک نمونه MeshMask چهره

مورد چهارم:

// در حال تکمیل است.

برای توضیح و تشریح عملکرد پروژه در ابتدا نیاز است با "توصیف گر ها" که از توصیف گر های معروف تصویر است آشنا شویم که به این منظور ابتدا یک مروری بر مبحث **توصیف گر های تصویر** می کنیم.

توصیف گر در واقع یک برداری است که بتواند الگوی intensity های یک تصویر را بیان کند، یک توصیف گر خوب توصیف گری است که بتواند تصاویر مربوط به یک دسته/نوع/بافت از اشیا را از سایر دسته ها برایمان متمایز کند.

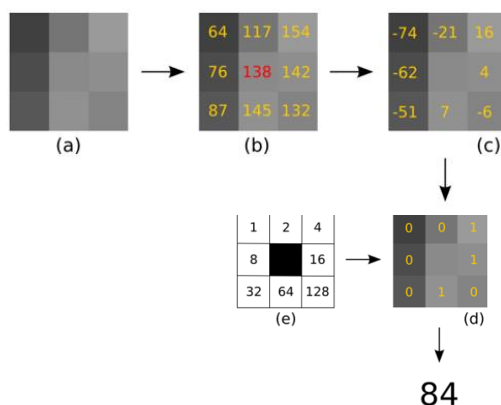
فرض کنید ما می خواهیم از این توصیف گر ها برای طبقه بندی شی مورد نظرمان استفاده می کنیم، مثلاً فرض کنید که ما می خواهیم چهره را از بقیه تصاویر جدا کنیم. ابتدا در وهله آموزش نیاز داریم که تعداد خوبی تصویر از چهره آماده کنیم (منظور از تصویر صورت مثلاً یک box ای است که درونش فقط یک چهره قرار دارد)، هم چنین در مقابل باید یکسری تصاویر غیر چهره را نیز در میان دیتاست خود داشته باشیم، حال که این تصاویر جمع آوری گردید، باید از روی آنها "توصیف گر" بسازیم و در نهایت این توصیف گر ها را به یک طبقه بند (classifier) بدهیم (همراه label آنها) تا این طبقه بند متوجه مرز بین این دو دسته اشیا بشود (در اینجا صورت و غیر صورت) و حال به کمک این مرز به دست آمده به تشخیص چهره می پردازیم، به این صورت که ابتدا تصویر داده شده را توصیف گرش را به دست می آوریم و سپس به کمک این توصیف گر و (مرز و طبقه بند موجود) به اصطلاح label تصویر ورودی را به دست می آوریم.

حال پله پله به معرفی توصیف گر های معروف می پردازیم و در انتها به توصیف گر های Haar, SIFT که در این پروژه استفاده گردیده است می پردازیم.

توصیف گر LBP:

این توصیف گر از جمله توصیف گر (descriptor) های معروف است و از لحاظ پردازشی مناسب بوده و به طریق زیر محاسبه می شود:

- هر پیکسل را با 8 پیکسل مجاورش مقایسه می کنیم، اگر یک همسایه ای intensity اش بیشتر از پیکسل مرکزی بود، به جای آن 1 می گذاریم و در غیر این صورت به جای آن صفر می گذاریم، به این صورت یک عدد 8 بیتی به ازای هر پیکسل به دست می آوریم. نمونه مثال این محاسبه در تصویر زیر موجود است:



یک نمونه مثال از محاسبه LBP یک پیکسل

یک نمونه از اثر این توصیف گر را روی تصویر زیر ملاحظه می کنیم:



یک مثال از LBP یک تصویر

در مثال بالا، LBP تصویر Gray شده ی سمت چپ را در تصویر سمت راست ملاحظه می کنید. می توانیم توصیف گر LBP را روی هر سه کانال RGB اعمال کنیم (که طبیعتا در این صورت اطلاعات بیشتری ذخیره می گردد) حال آنکه اغلب مواقع LBP را روی همان تصویر Gray اعمال می کنیم.

حال به کمک تکنیک LBP می خواهیم توصیف گر بسازیم:

یکی از روش های معروف به شرح زیر است:

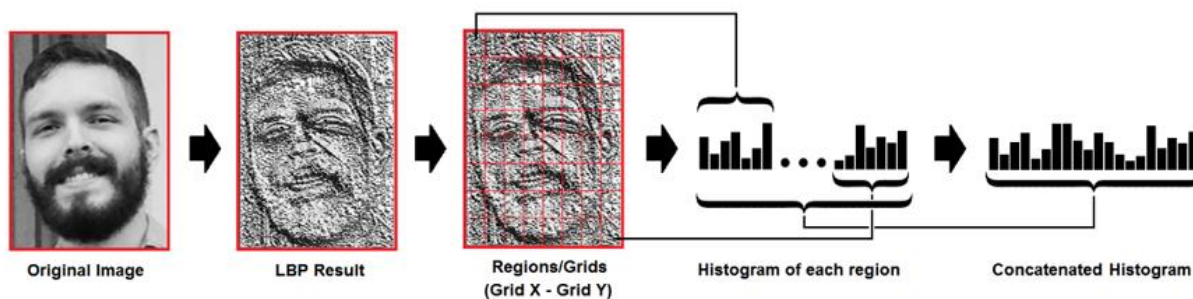
- 1- ابتدا برای کل پیکسل های تصویر LBP را محاسبه می کنیم.
- 2- حال تصویر را سلول بندی می کنیم (مثلا مستطیل های 8 پیکسل در 8 پیکسل)
- 3- حال هیستوگرام LBP هارا در این سلول ها محاسبه می کنیم.
- 4- حال این هیستوگرام هارا concat می کنیم تا یک بردار به دست بیاید.
- 5- بردار به دست آمده یک توصیف گر بر مبنای LBP است.

علت استفاده از هیستوگرام: استفاده از هیستوگرام باعث می شود که توصیف گر ما نسبت به جابه جایی و دوران مقاوم شود.

علت استفاده از سلول بندی: اگر ما از سلول بندی استفاده نمی کردیم و فقط از یک بار هیستوگرام گرفتن استفاده می کردیم باعث می شد که اگر تصویر فعلی کل پیکسل هایش شافل بخورد همچنان توصیف گری برابر با حالت اولیه اش داشته باشد اما با سلول بندی و سپس محاسبه هیستوگرام در این سلول ها این مشکل تا حد خوبی برطرف گردید.

یک نمونه از این روند در صفحه بعد توضیح داده شده است.

مثال:



یک مثال از محاسبه توصیف گر LBP

تکنیک های اعمالی روی LPF: (جهت ارتقا کیفیت)

اگر دقت کرده باشید داریم که طول توصیف گر های بر مبنای LBP برابر تعداد سلول ها ضرب در 256 (با فرض استفاده از یک کانال gray)، تکنیک زیر را اعمال می کنیم تا به دو هدف زیر برسیم:

1- کاهش حجم دیتای توصیف گر

2- robust بودن نسبت به نویز.

تکنیک اعمالی: به اعداد 8 بیتی LBP نگاه می کنیم و اگر این اعداد فقط دارای دو transient (منظور از transient تغییر دو بیت مجاور است) از 0 به 1 یا 1 به 0 باشند، به آن ها LBP های uniform می گوئیم و برای هر کدام از این LBP ها یک bin در هستوگرام در نظر میگیریم و به بیان دیگر هر LBP یونیفرم دارای یک بین در هستوگرام بوده و همه LBP های غیر یونیفرم دارای یک بین در هستوگرام نهایی است.

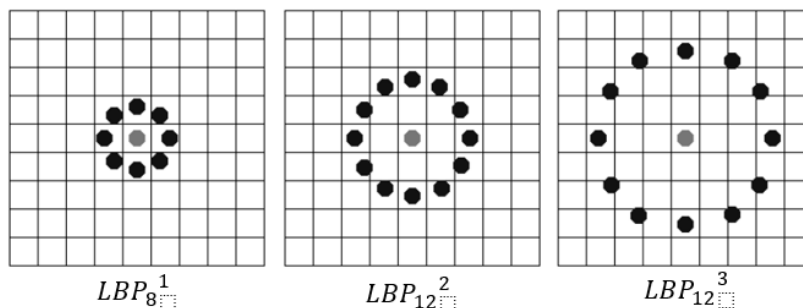
مثال: LBP های زیر همگی یونیفرم اند:

00011000	} Uniform
00111000	
10000111	
10000000	

با این تکنیک طول بردار های توصیف گر، تقریباً 1/4 (تعداد بین ها از 256 به حدوداً 59 بین تقلیل می یابد) می گردد و با توجه به تکنیک اعمالی این توصیفگر نسبت به نویز های اعمالی تا حدی مقاوم می گردد.

در ادامه یک تکنیک دیگر را توضیح می دهیم.

یک تکنیک خوب دیگر که بر LBP اعمال می کنند این است که پارامترهای شعاع و تعداد همسایه ها را هم اعمال کنیم، که نتیجه این پارامترها، دانستن رنگ نقاط بین پیکسلی است (چون دایروی محاسبه می گردد) که این رنگ ها را به کمک interpolation به دست می آوریم. برای شفاف تر شدن این تکنیک تصویر زیر را ملاحظه کنید:



تکنیک LBP دایروی

اندیس زیرنویس: تعداد همسایه ها و اندیس بالانویس: شعاع سلول دایروی.

توصیف گر HOG: (Histogram oriented Gradient)

در ادامه فرآیند محاسبه ی HOG آورده شده است:

1- ابتدا مشتق های تصویر را به کمک فیلترهای ساده ی زیر انجام می دهیم.

$$[-1 \ 0 \ 1]^T \text{ and } [-1 \ 0 \ 1]^T.$$

2- تصویر را سلول بندی می کنیم.

3- در هر سلول هیستوگرام زاویه های گرادیان ها را محاسبه می کنیم.

این هیستوگرام ها دارای تعداد بین های محدود است (مثلا اگر فاصله هر دو بین 20 درجه باشد در نهایت 9 بین خواهیم داشت).

$$9 \text{ bins among } [-90, 90]$$

هم چنین این هیستوگرام ها وزن دار اند و وزن آن ها به کمک اندازه بردار گرادیان آن زاویه وزن دار می گردد.

یعنی هر زاویه را که می شماریم این شمارش با اندازه گرادیان آن زاویه بایست وزن دار گردد.

4- در مرحله آخر، بایست block normalization انجام بدهیم، ابتدا لازم است block را تعریف کنیم، هر بلاک یک پنجره ی دارای 4 سلول به صورت 2 در 2 است.

در این مرحله هیستوگرام های هر 4 سلول موجود در یک بلاک را با یکدیگر concat می کند و در نهایت یک بردار از آنها می سازد و در نهایت آن را به نرم 2 اش normalize می کند و یک عدد ثابت کوچک نیز در نظر میگیرد که تقسیم بر صفر رخ ندهد.

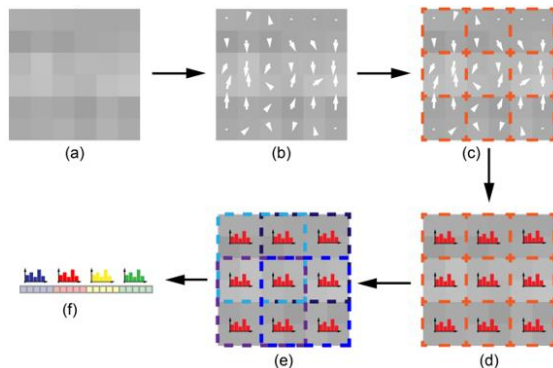
$$v' = \frac{v}{\sqrt{|v|^2 + \epsilon}}$$

علت انجام normalization:

- 1- این عمل باعث مقاوم شدن نسبت به تغییرات روشنایی می گردد.
- 2- هر کدام از این سلول ها هیستوگرامش 4 بار در توصیف گر نهایی ظاهر می شود (با 4 بار normalization متفاوت) که باعث می شود که دقت و کیفیت تمیز در توصیف گر نهایی افزایش پیدا کند، توجه کنید که منظورمان از 4 بار تکرار، برای سلول های غیر گوشه بود.

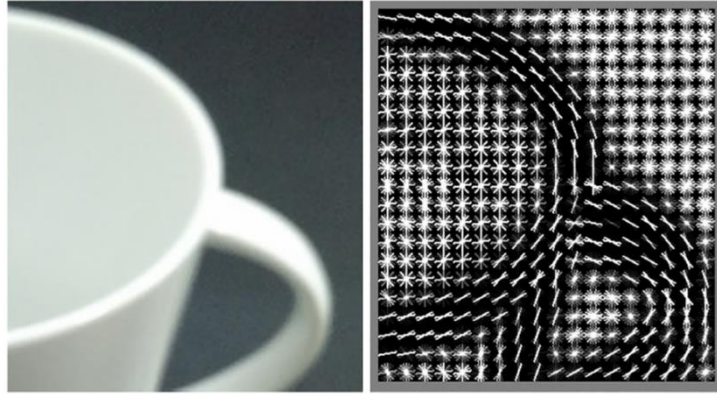
پس از آنکه بردار های هر بلاک به دست آمد، آن هارا با یکدیگر concat می کند و بردار نهایی که توصیف گر ماست ساخته می شود.

مراحل تولید توصیف گر HOG به بیان تصویری:



مراحل ساخت توصیف گر HOG

یک مثال از استخراج توصیف گر HOG روی تصویر:



تصویر سمت راست HOG تصویر سمت چپ است

در شکل سمت راست هیستوگرام های هر سلول نشان داده شده است، همانطور که گفته شد سلول های غیر حاشیه 4 بار نرمال می شود (که در این تصویر یکی از این نرمال کردن ها نشان داده شده است)، در ضمن توجه کنید که محور هیستوگرام ها افقی است (محور x به سمت پایین و محور y به سمت راست).

ذکر چند نکته:

- 1- در نواحی مرز ها جهت هیستوگرام ها در یک جهت خاص dominant اند.
- 2- در نواحی flat داریم که شکل هیستوگرام ها به صورت ستاره ای است، یعنی در تمام جهت ها مقدار خوبی دارد چون در نواحی flat، گرادیان ما یکنواخت است و طبیعی است که هیستوگرام در این نقاط متقارن ستاره ای باشد.
- یکی از کاربرد های خیلی موفق HOG در تشخیص عابر پیاده است (بدیهی است که این تشخیص، عمل خیلی پرکاربرد است) که با وجود چالش های زیر خوب عمل کرد:

1- نحوه ایستادن ها

2- تنوع background ها و لباس اشخاص

3- زاویه افراد نسبت به دوربین و ...

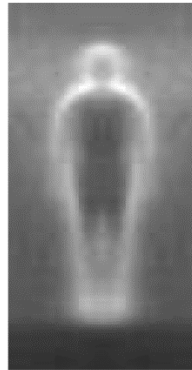
و بدیهتا اطلاعاتی مثل فقط رنگ و intensity نمی تواند به تنهایی توصیف مناسبی از عابر پیاده بدهد، اما چرا توصیف گر HOG در این تشخیص، موفق واقع شد؟

به تصاویر عابر پیاده زیر توجه کنید:



چند نمونه از دیتاست Pedestrian (عابرپیاده)

اگر اندازه گرادیان را در هر پیکسل محاسبه کنیم و بعد از این تصاویر میانگین بگیریم به تصویر زیر می رسم:



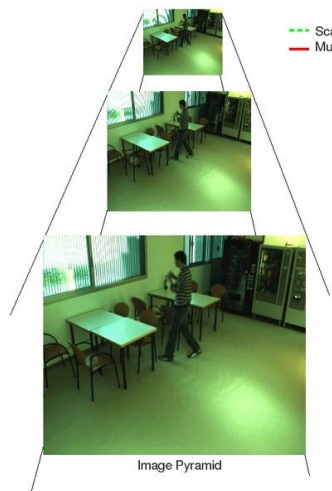
خروجی اعمال فیلتر میانگذر روی اندازه گرادیان تصاویر عابرپیاده

تصویر زیر می گوید که گرادیان در مرز شخص و محیط توصیف گر خیلی مناسبی است و به همین دلیل، توصیف گر HOG توصیف گر موفقی در این تشخیص بود.

یک نکته حیاتی: معمولا برای تشخیص یک شی، classifier ما در یک اسکیل train می شود و خب حال که چون تا به اینجای کار همه کارها در یک اسکیل انجام می گرفت، لذا این توصیف گرهای تا به اینجای کار، در مقابل scale مقاوم نبودند و اگر یک تصویر با اسکیلهایی متفاوت با اسکیل train شده به ورودی طبقه بند بدهیم، در خروجی خواهیم دید که خروجی طبقه بند در برخی اسکیلهایی که فاصله نسبتا زیادی با اسکیل train شده دارند، مطلوب نخواهد بود.

برای حل این مشکل از روش pyramid استفاده می کنیم که یعنی طبقه بند خود را نسبت به اسکیلهای مختلف ورودی train می کنیم.

یک مثال از روش pyramid:

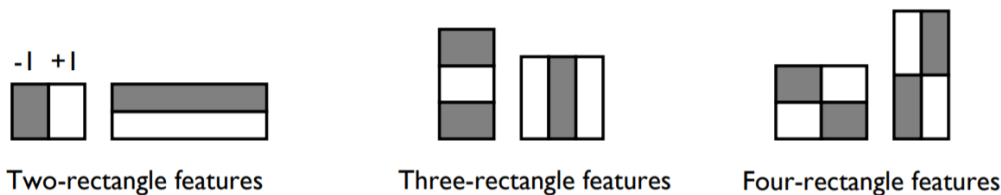


یک مثال از هرم با سه octave (سه مرحله اسکیل)

حال در ادامه به بیان دو روش دیگر به دست آوردن توصیف گر می پردازیم، لازم به ذکر است که هر دوی روش هایی که در ادامه توضیح می شود صراحتاً در پروژه استفاده می گردد.

:Haar-Like

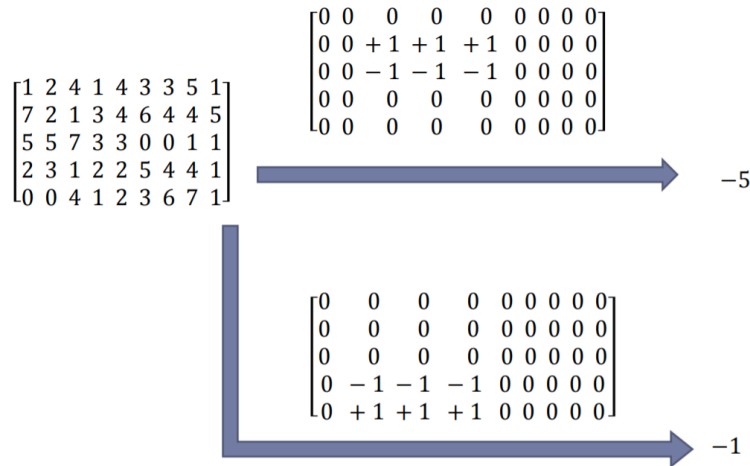
هار فیچر ها به صورت تفاضل مجموع intensity ها در مستطیل های مجاور هم است، چند نمونه از الگوهای مستطیل ها را در تصویر زیر مشاهده می کنید:



سه نمونه مثال از پترن های Haar Feature

که پیکسل های مشکی با وزن 1- و پیکسل های سفید با وزن 1 در به دست آوردن هار فیچر استفاده می شوند یعنی به عبارت دیگر تفاضل (مجموع intensity های پیکسل های سیاه) از (مجموع intensity های سفید) محاسبه می گردد.

یک نمونه از این روش محاسبه را در زیر مشاهده می کنیم:



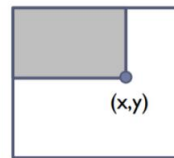
یک مثال از محاسبه HaarFeature

یک روش سریع جهت محاسبه Haar Feature به کمک تصویر انتگرال است.

نکته: تصویر انتگرال، در اصل هر پیکسلش برابر انتگرال پیکسل های زیرش است، جهت شفاف تر شدن این بیان، صراحتاً سراغ رفرنس اصلی می رویم:

The integral image at (x,y) is equal to the sum of the intensities above and to the left of (x,y),

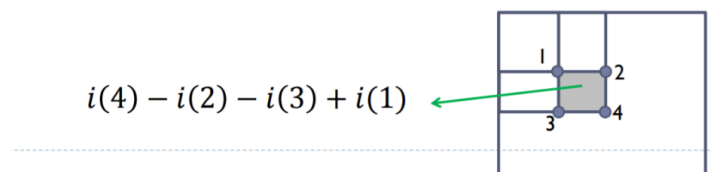
$$i(x,y) = \sum_{x' \leq x, y' \leq y} f(x',y')$$



فرمول محاسبه تصویر انتگرال

حال به کمک تصویر انتگرال (که به راحتی با پردازش های موازی محاسبه می گردد) به سرعت هار فیچر های تصویر به دست می آید:

به کمک تصویر انتگرال، مجموع intensity های یک مستطیل به سرعت با 4 جمع و تفریق به کمک تصویر انتگرال محاسبه می گردد، به شکل زیر توجه کنید:



محاسبه Haar فیچر مستطیلی به کمک integral image

یکی از روش های معروف تشخیص صورت، Viola Jones است که از هار فیچرها جهت تشخیص استفاده می کند. این روش از دو تکنیک زیر جهت ارتقا کیفیت تشخیص استفاده می کند:

1- انتخاب فیچرها:

این مرحله به صورت آفلاین در فاز training انجام می گیرد و از الگوریتم adaboost برای انجام این عمل استفاده می شود.

2- استفاده کردن از ساختار آبشاری (cascade):

حال به تشریح هردوی این تکنیک ها می پردازم:

انتخاب فیچرها: با توجه به ظرافت و روانی بیانِ رفرنس در این مورد، صریحا محتوای رفرنس را بیان می کنیم:

Adaboost method is used for selecting most discriminative features out of about 180K features.

Each of the 180K features constitutes a weak learner with two parameters: threshold and parity.

$$WL_t(x) = \begin{cases} 1, & \text{if } p_t h_t(x) < p_t \theta_t \\ 0, & \text{o. w.} \end{cases}$$

parity threshold

4916 face windows and 10,000 non-face windows are used for training.

در ادامه شرح دقیق الگوریتم را می آوریم:

Adaboost is a very powerful classification method.

In each iteration

- ▶ Best feature is chosen.
- ▶ Wrongly classified samples get larger weights.

Algorithm:

- ▶ $train = \{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$, $y^{(i)} \in \{0, 1\}$
- ▶ Initialize $w_i = \begin{cases} \frac{1}{2m}, & \text{if } y^{(i)} = 0 \\ \frac{1}{2l}, & \text{if } y^{(i)} = 1 \end{cases}$, where m is the no. of negative samples and l is the no. of positive samples.
- ▶ For $t = 1, \dots, T$
 - ▶ Normalize weights, $w_i \leftarrow \frac{w_i}{\sum_{i=1}^n w_i}$
 - ▶ For each feature h_j , train a weak classifier WL_j . Calculate error and choose the weak classifier with the lowest error.
$$err_t = \sum_{i=1}^n w_i |WL_t(x^{(i)}) - y^{(i)}|$$
 - ▶ Update the weights, $w_i \leftarrow w_i e^{\alpha_t I(WL_t(x^{(i)}) \neq y^{(i)})}$, where $\alpha_t = \log(\frac{1-err_t}{err_t})$
- ▶ Final strong classifier: $SL(x) = \begin{cases} 1, & \text{if } \sum_{t=1}^T \alpha_t WL_t(x) \geq \frac{1}{2} \sum_{t=1}^T \alpha_t \\ 0, & \text{o.w.} \end{cases}$

روند کلی الگوریتم به شرح زیر است:

1- نسبت دادن وزن های برابر اولیه به نمونه های موافق (حاوی تصویر هدف) و مخالف به صورت یک تقسیم بر تعداد نمونه ها.

2- در هر مرحله وزن ها به مجموع وزن ها نرمال می گردند.

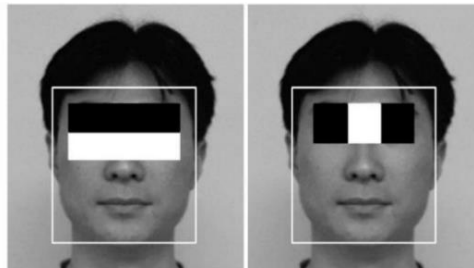
3- در هر مرحله تمامی خطا ها (طبق فرمول بیان شده در عکس بالا) محاسبه می گردد و مورد (به اصطلاح weak classifier) با خطای کمتر انتخاب می گردد.

4- وزن ها با فرمول گفته شده در خط یکی مانده به آخر عکس، به روز رسانی می شوند.

و در نهایت به کمک weak classifier ها یک strong classifier تولید می گردد.

نکته: الگوریتم adaboost یک هار فیچر هارا به صورت مرتب شده می دهد.

مثال: در شکل زیر دو نمونه از غالب ترین هارفیچر هارا مشاهده می کنیم:



Two most discriminative features.

حال به صورت گذرا به معرفی معیار کیفیت و دقت در این روش می پردازیم:

True positives: No. of face windows that are truly detected as face.

False positive: No. of non-face windows that are falsely detected as face.

Accuracy: $\frac{\text{correctly classified samples}}{\text{total samples}}$

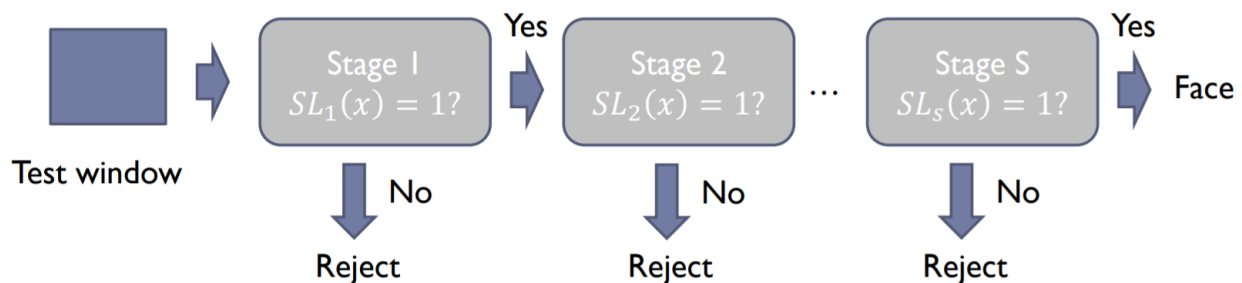
True positive rate: $\frac{TPs}{Ps}$

False positive rate: $\frac{FPs}{Ns}$

که همانطور که ملاحظه می کنید هرچقدر میزان True positive ها بیشتر باشد داریم که طبقه بند، دقت بالایی دارد.

استفاده کردن از ساختار آبشاری (cascade):

با قرار دادن چند سلسله Strong Classifier به صورت Cascade داریم که به صورت پله ای تصاویر غیرچهره را رد می کند.



تعدادی نکته ی کوتاه راجع به این ساختار های آبشاری موجود است که مستقیماً از رفرنس آن هارا در اینجا ذکر می کنم.

Each stage is a strong learner. Early stages are fast classifiers with smaller number of weak learners, higher TP rates and higher FP rates. (In Viola-Jones, the first stage has one WL, 100% TP rate and 40% FP rate).

Later stages are slow classifiers with larger number of WLs and lower FP rates (TP rate decreases very slightly). Most windows get rejected early and do not reach to the later stages.

Partial cascade classifiers are scanned over non-face images and their FPs are used to train the subsequent stages.

In Viola-Jones, there are a total of 38 stages and 6000 WLs.




On average, each window is evaluated using 10 feature evaluations.

On a challenging dataset, final classifier had a TP rate of 94% when there were only 167 FPs out of 75,000,000 non-face windows.

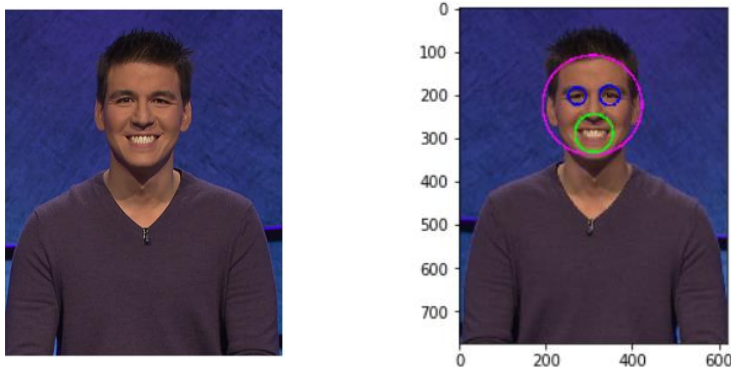
Viola-Jones is so fast that scanning a 300×400 image with all its scales takes about 0.07 of a second on a Pentium processor.

برای شفاف تر شدن نحوه کار با Haar فیچر ها قطعه کدی به زبان پایتون زده ام.

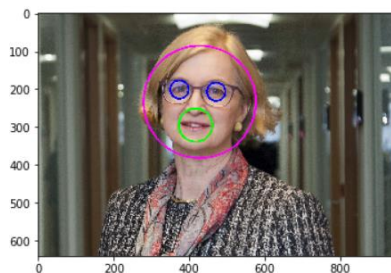
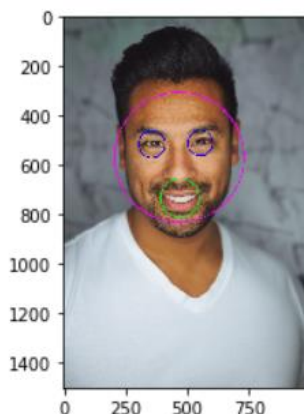
برای تشخیص اجزای چهره به کمک HaarFeature ها ابتدا نیاز است تا فایل های xml ای که به نوعی حاوی طبقه بندی های نهایی اند هستند را در دایرکتوری کد ضمیمه شده قرار دهید.

 haarcascade_eye
 haarcascade_frontalface_default
 haarcascade_smile

حال فایل با نام FaceDetectionHaarFeature را باز کرده (و کتاب خانه openCV را نیز نصب شده داشته باشید) و ران کنید، چند نمونه از خروجی کد زده شده را درعکس های زیر مشاهده می کنیم.



نمونه اول از تشخیص اجرا چهره به کمک HaarFeature



نمونه دوم و سوم از تشخیص اجرا چهره به کمک HaarFeature

همانطور که ملاحظه می کنید با کیفیت خوبی کد ما کار میکند و مشابه همین کار ما برای تشخیص اجزا صورت در پروژه اصلی استفاده شده است. (لازم به ذکر است که ما از openCV پایتون استفاده کردیم حال آنکه در پروژه اصلی از openCV به زبان c++ استفاده شده است) و تنها تفاوت موجود، تفاوت سینتکس است.

نکته مهم: چرا در روش هار مشکل اسکیل وجود ندارد؟

پاسخ: به این علت که در روش هار ما از مستطیل هایی استفاده می کنیم که می توانند کاملاً اسکیل شوند و در اصل به طور غیر مستقیم، اسکیل کردن با تغییرات ابعاد مستطیل های استخراج کننده هار فیچر انجام می گیرد و لذا داریم که روش هار نسبت به اسکیل هم مقاوم است.

از دیگر ابزارهای مهمی که پروژه از آن استفاده کرده است SIFT است که برای تشریح آن نیاز است که یک مروری بر نقاط کلیدی گوشه (interest points) بکنیم.

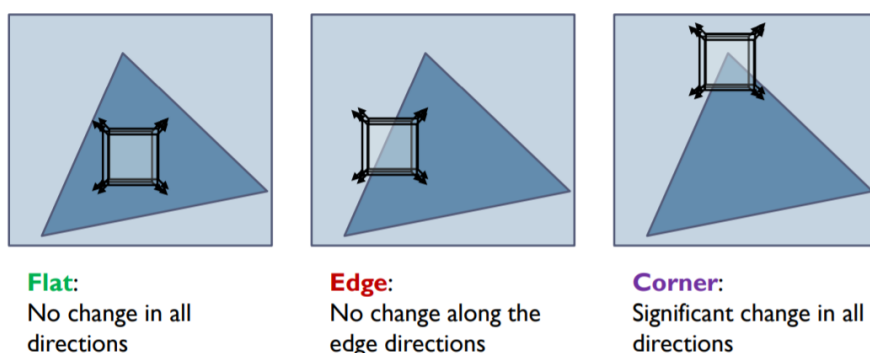
یک نقطه در صفحه را در نظر بگیرید، حال یک box حول آن نقطه را در نظر بگیرید، سه حالت ممکن است رخ بدهد:

5- Corner: بردارهای گرادیان در برخی از راستاها در این box ماکزیمم اند.

6- Edge: بردارهای گرادیان به صورت تقریبی برابر با هم در یک راستا اند.

7- Flat: بردار گرادیان در راستای خاصی dominant نیست.

برای شفاف تر شدن این حالات به شکل زیر توجه کنید:



حالات یک نقطه در صفحه.

حال ممکن است بپرسید که نقاط کلیدی چه کاربرد هایی دارند؟

ابتدا به بیان کاربرد نقاط کلیدی به طور کلی می پردازیم و سپس به بیان آن کاربردی از آن که در این پروژه استفاده شده است می پردازیم.

1- دنبال کردن اشیا (image tracking)

2- تشخیص عمق

3- 3D reconstruction

4- تصاویر پانوراما

5- تشخیص اشیا

6- Motion-based segmentation

و اما آن کاربرد کلیدی که در این پروژه از آن استفاده شده است این است که فرض کنید مثلاً ما یک چهره عادی داریم و یک چهره که روی آن آرایش انجام شده است، حال می خواهیم چهره ای که روی آن آرایش انجام شده است روی چهره اصلی قرار بگیرد و این کار به این صورت انجام میگیرد که ابتدا نقاط کلیدی چهره اصلی و نقاط کلیدی چهره آرایش شده را استخراج می کنیم و سپس نقاط کلیدی دو تصویر را به صورت متناظر در آورده (ممکن است برخی نقاط، متناظری نداشته باشند، در این صورت از آن ها صرف نظر کنید) و سپس یک تبدیل به گونه ای می زنیم که این دو تصویر را با شرط انطباق نقاط کلیدی شان روی هم قرار دهد (طراحی این ماتریس تبدیل سخت نیست).

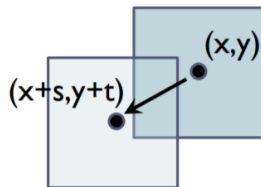
حال به بیان دو روش معروف در به دست آوردن نقاط کلیدی می پردازیم:

1- روش هریس

SIFT -2

مروری بر نقاط کلیدی:

ابتدا نیاز است تا با متریکی فاصله ی دو مستطیل (با ابعاد برابر) یک تصویر را از یکدیگر بیابیم:



برای اینکار ابتدا لازم است که بردار انتقالی که نقاط مستطیل اول را به نقاط مستطیل دوم می نگارد را محاسبه کنیم و این بردار برابر است با برداری که از مرکز مستطیل اول شروع شده و به مرکز مستطیل دوم منتهای می گردد، حال که نقاط متناظر را به کمک این بردار انتقال به دست آوردیم به محاسبه فاصله ی نقاط می پردازیم و متریک فاصله را در اینجا، تفاوت intensity های دو پیکسل در نظر میگیریم (معیار SSD - sum of squared distance)

► To measure the change we use Sum of Squared Distances (SSD).

$$SSD_{(x,y)}(s, t) = \sum_{m=-r}^r \sum_{n=-r}^r (f(x + m + s, y + n + t) - f(x + m, y + n))^2$$

بنا به دلایلی که در آینده اشاره خواهیم کرد داریم که پیکسل هارا وزن دار در فرمول بالا اعمال می کنیم به عبارتی از فرمول زیر استفاده می کنیم:

$$SSD_{(x,y)}(s, t) = \sum_{m=-r}^r \sum_{n=-r}^r w(m, n) (f(x + m + s, y + n + t) - f(x + m, y + n))^2$$

خب برگردیم به سرفصل، ما دنبال یافتن بردار انتقالی بودیم که تفاضل دو مستطیل با معیار گفته شده ماکزیمم شود، حال می خواهیم به کمک ریاضیات این بردار انتقال را بیابیم.

به کمک بسط تیلور حول مرکز مستطیل اولیه داریم:

Using Taylor series:

$$f(x+m+s, y+n+t) = f(x+m, y+n) + sf_x(x+m, y+n) + tf_y(x+m, y+n)$$

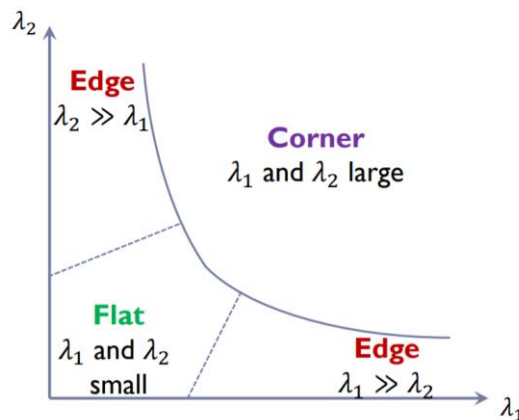
Then,

$$\begin{aligned} SSD_{(x,y)}(s, t) &= \sum_{m=-r}^r \sum_{n=-r}^r w(m, n) (sf_x(x+m, y+n) + tf_y(x+m, y+n))^2 \\ &= \sum_{m=-r}^r \sum_{n=-r}^r w(m, n) \left([s \ t] \begin{bmatrix} f_x(x+m, y+n) \\ f_y(x+m, y+n) \end{bmatrix} \right)^2 \\ &= \sum_{m=-r}^r \sum_{n=-r}^r w(m, n) \left([s \ t] \begin{bmatrix} f_x \\ f_y \end{bmatrix} [f_x \ f_y] \begin{bmatrix} s \\ t \end{bmatrix} \right) \\ &= \sum_{m=-r}^r \sum_{n=-r}^r w(m, n) \left([s \ t] \begin{bmatrix} f_x \\ f_y \end{bmatrix} [f_x \ f_y] \begin{bmatrix} s \\ t \end{bmatrix} \right) \\ &= [s \ t] \underbrace{\sum_{m=-r}^r \sum_{n=-r}^r w(m, n) \begin{bmatrix} f_x^2 & f_x f_y \\ f_x f_y & f_y^2 \end{bmatrix}}_{M_{2 \times 2}} \begin{bmatrix} s \\ t \end{bmatrix} \end{aligned}$$

همانطور که از ریاضی 2 به خاطر داریم، عبارت فوق هنگامی ماکزیمم می گردد که بردار $[s \ t]^T$ برابر بردار ویژه ی ماتریس M باشد.

پس اگر بردارمان در راستای بردار ویژه ماتریس M باشد، بیشترین میزان تغییر با معیار SSD intensity خواهیم داشت.

خب، می دانیم که مقدار ویژه های ماتریس، هویداگر شدت تغییرات در راستای بردار ویژه ها است و حال داریم که چون ماتریس M ما دودویی است داریم که دو عدد بردار ویژه خواهد داشت و حال با دسته بندی حالات ممکن برای مقدار ویژه های ماتریس M به پاسخ می رسیم و پاسخ به شرح زیر است:



حالات مختلف نقطه برحسب اندازه ی مقدار ویژه های ماتریس M

حال یک معیار cornerness معرفی می کنیم:

$$R = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2}$$

که هرچه این مقدار بزرگتر باشد، به نقطه گوشه نزدیک تر شده ایم و ترشهولد معقولی که از آن بیشتر بودن R به معنی گوشه بودن است به کمک training روی دیتاست به دست می آید.

سوال:

چرا SSD وزن دار شده را استفاده کردیم؟

پاسخ:

توجه کنید که اگر وزن برابر برای نقاط در نظر بگیریم برای تعداد خوبی از نقاط، حول نقطه گوشه داریم که معیار SSD عدد بزرگی خواهد داد و این اصلاً خوب نیست لذا مناسب است که مثلاً یک وزن گوسی روی پیکسل ها، حین محاسبه SSD اعمال کنیم تا بدین صورت مشکل ذکر شده تا حد خوبی حل شود.

حال برویم سراغ روش هار:

مراحل روش هار به شرح زیر است:

Compute f_x and f_y .

Compute images f_x^2 , f_y^2 and $f_x f_y$.

Smooth the three above images with Gaussian filter.

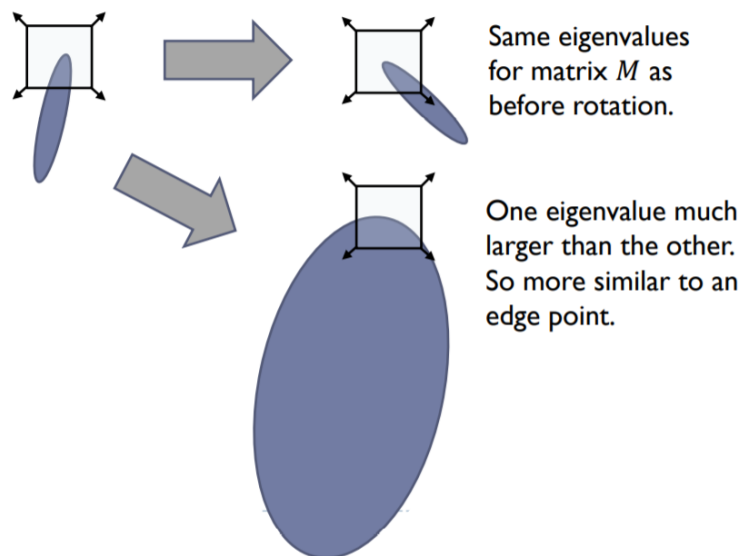
Calculate the cornerness measure R at each pixel using matrix M .

Apply a threshold to R .

Find local maxima.

توجه به این دلیل ما از ماکزیمم محلی استفاده می کنیم که خب تصویر ما شامل نقاط گوشه ای متعددی است و لازم است حول هر نقطه گوشه، آن نقطه ای که ماکزیمم R را داشته باشد جدا کنیم و اگر از ماکزیمم مطلق استفاده کنیم در خروجی یک عدد نقطه گوشه که تیز ترین گوشه است را خواهیم داشت که هدف ما نیست.

روش هریس نسبت به rotation مقاوم است ولی نسبت به scale شدن مقاوم نیست! و برای مشخص تر شدن اینکه چرا نسبت به scale مقاوم نیست به شکل زیر توجه کنید. در اصل علت استفاده کردن از روش SIFT به جای روش هریس مقاوم بودن بی نظیر آن نسبت به اسکیل است.



Robust بودن و نبودن Haar نسبت به rotation و scale.

همانطور که ملاحظه می کنید در حالتی که بیضی اسکیل نشده است یک گوشه داخل مستطیل ملاحظه می کنیم و همانطور که ملاحظه می کنید با دوران این گوشه بودن تغییر نمی کند اما اگر بیضی اسکیل شود دیگر این نقطه به صورت گوشه خود را نشان نمی دهد و بیشتر شبیه به edge می شود تا گوشه.

در ادامه روش SIFT را ذکر خواهیم کرد که یکی از موفق ترین الگوریتم های حوزه بینایی ماشین است و نسبت به اسکیل هم در تشخیص نقاط کلیدی مقاوم است.

روش SIFT:

روش SIFT شامل 6 مرحله کلی است که به شرح زیر است:

LoG scale space construction

Localization of interest points

Accurate localization

Outlier rejection

Orientation assignment

Description of the interest points

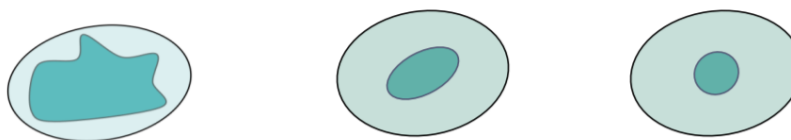
مورد سوم و چهارم جزئیات خیلی زیادی دارند و در این گزارش تشریح نمی شوند (تاثیر کمتری هم در فهم روش SIFT دارند).

شرح مرحله اول و مرحله دوم:

احتمالا اصطلاح blob را شنیده اید، blob به معنی لکه است و blob detection یکی از روش های بینایی ماشین است برای تشخیص لکه ها، اما لکه شاید واژه ی فارسی مناسبی برای توصیف blob نباشید، blob در اصل به معنی پترن هایی در تصویر است که از لحاظ اسکال در قیاس با کل تصویر کوچکتر بوده و با الگویش با باقی تصویر متفاوت است.

در اصل، blob ها دسته مهمی از نقاط کلیدی اند که رفتارشان شباهت خوبی با سایر انواع نقاط کلیدی دارد و روش SIFT از این نکته جهت یافتن نقاط کلیدی استفاده می کند.

چند نمونه از blob ها:



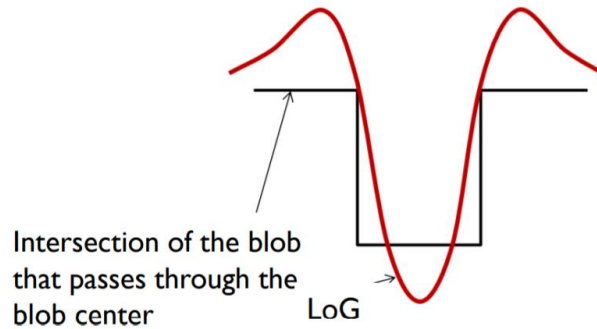
چند نمونه مثال از blob

تشخیص blob ها به کمک LoG که همان Laplacian of Gaussian است انجام میگیرد (رابطه LoG به شرح زیر است):

$$\nabla^2 G(x, y) = \left(\frac{x^2 + y^2 - 2\sigma^2}{2\pi\sigma^6} \right) e^{-\frac{x^2 + y^2}{2\sigma^2}}$$

به کمک روش LoG به خوبی می توان blob ها را تشخیص داد.

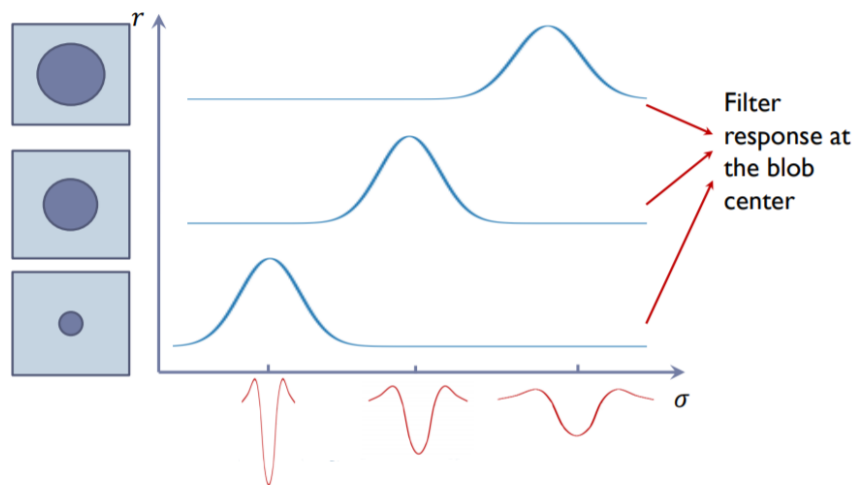
اگر LoG را روی پیکسل های blob بندازیم داریم که: (خط قرمز، نمودار blob است).



همانطور که ملاحظه می کنید اگر سیگمای LoG طوری تعیین شود که blob را همانطور که در شکل مشخص است به خوبی دربرگیرد داریم که فیلتر LoG در مرکز blob، absolute value زیادی خواهد داشت و با محاسبات ریاضی داریم که این سیگمای مطلوب، برابر رادیکال 2 برابر شعاع blob است (برای اینکه قعر LoG به خوبی مشابه شکل، blob را دربرگیرد).

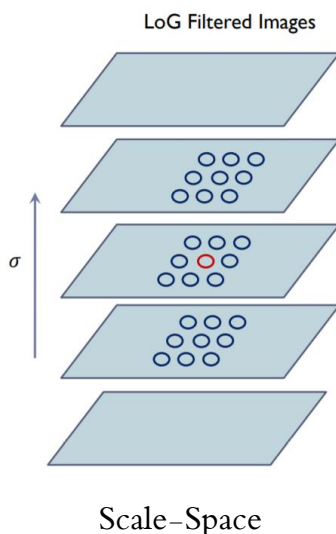
برای شفاف تر شدن عبارت گفته شده به مطلب زیر توجه کنید:

If there is a blob with arbitrary radius at any point, by changing the sigma of LoG, the filter response has an extremum at the right sigma.



حال برای مقاوم بودن نسبت به اسکیل داریم که در اندازه های مختلف از sigma فیلتر LoG را روی عکس اعمال می کنیم و به کمک ماکزیمم محلی نقاط گوشه را استخراج می کنیم، برای شفاف تر شدن این مطلب به تصویر صفحه بعد توجه کنید.

فرض کنید که طبقات زیر، خروجیِ فیلترِ اعمالیِ LoG با سیگماهای مختلف روی تصویر اولیه است:



اگر یک پیکسل در قیاس با 26 همسایه ی خود (9 همسایه بالا در سیگمای بیشتر، 9 همسایه پایین در سیگمای کمتر و 8 همسایه در سیگمای فعلی) ماکزیمم باشد، آن را به عنوان نقطه کلیدی در نظر میگیریم.

به طبقاتی همانند تصویر نشان داده شده، Scale Space می گویند و برای robust شدن نسبت به تغییرات اسکیل تصویر در نظر گرفته می شوند.

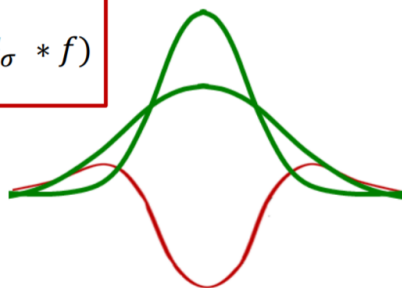
نکته اضافی: با تکنیک زیر عملیات محاسبه لاپلاسیان را سبک تر می کنیم و صرفا با دو تفاضل گوسی، آن را محاسبه می کنیم:

$$\frac{\partial G(x, y; \sigma)}{\partial \sigma} = \left(\frac{x^2 + y^2 - 2\sigma^2}{2\pi\sigma^5} \right) e^{-\frac{x^2+y^2}{2\sigma^2}} = \sigma \nabla^2 G(x, y)$$

$$\frac{\partial G(x, y; \sigma)}{\partial \sigma} \approx \frac{G(x, y; k\sigma) - G(x, y; \sigma)}{(k-1)\sigma}, k \approx 1$$

$$\nabla^2 G(x, y) \approx \frac{G(x, y; k\sigma) - G(x, y; \sigma)}{(k-1)\sigma^2}$$

$$\Rightarrow \nabla^2 G_\sigma * f \approx \frac{1}{(k-1)\sigma^2} (G_{k\sigma} * f - G_\sigma * f)$$



مرحله پنجم: Orientation Assignment

در این مرحله می خواهیم به هر کدام از interest point ها یک orientation نسبت بدهیم (در اصل برای مقاوم شدن توصیف گرمان نسبت به rotation این کار را انجام می دهد).

در این مرحله صرفاً زاویه را استخراج می کنیم.

برای اینکه orientation یک interest point شده را تشخیص بدهیم بدین ترتیب عمل می کنیم:

- 1- ابتدا blur شده ی تصویر شده در همان اسکالی که interest point را استخراج کردیم را برمی داریم.
 - 2- حول هر interest point یک پنجره در نظر بگیرید و گرادیان را برای هر پیکسل محاسبه کن.
 - 3- حال برای گرادیان های یک وزن گوسی اعمال می کنیم.
- وزن فعلی در اصل اندازه گرادیان هاست که یک وزن گوسی نیز روی آن اعمال شده است.
- 4- حال یک هیستوگرام از زاویه گرادیان ها رسم می کنیم.
- هیستوگرام ما 36 بین دارد و یعنی 10 درجه 10 درجه تقسیم بندی شده است.
- 5- حال که هیستوگرام را ساخته ایم بررسی می کنیم که کدام bin، غالب است (بیشترین اندازه را دارد) و حال این orientation را به عنوان زاویه این interest point در نظر می گیریم.

نکته: سیگمای وزن گوسی اعمال شده به اندازه ی گرادیان ها برابر 1.5 برابر سیگمای است که در آن سیگما نقطه کلیدی استخراج شده است.

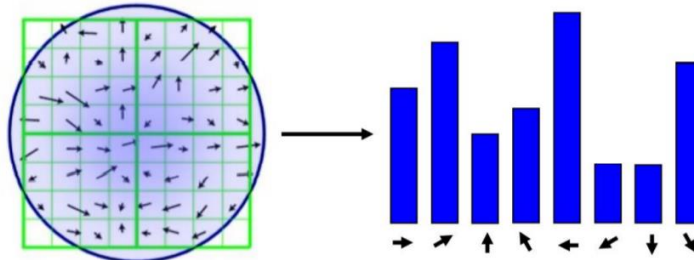
$$\text{SigmaOfGaussianWeight} = 1.5 * \text{InterestPointSigma}$$

نکته: سایز پنجره به صورت معمول به طریق زیر محاسبه می شود:

$$\text{Widow size} = 1 + 6 * \text{SigmaOfGaussianWeight} = 1 + 6 * 1.5 * \text{InterestPointSigma} = 1 + 9 * \text{InterestPointSigma}$$

سوال: چرا سایز پنجره ثابت نیست؟

پاسخ: همانطور که در روش هار مشاهده کردید داشتیم که این متد نسبت به اسکیل مقاوم نبود. چرا؟ چون سایز پنجره هایش مستقل از scale ثابت بود، به همین دلیل در روش SIFT سایز پنجره ها طبق سیگمایی که در آن سیگما، interest point ها استخراج شده است تعیین می گردد تا نسبت به اسکیل و نقاط کلیدی با سایز های مختلف مقاوم باشد.



هیستوگرام orientation ها

یک نمونه مثال که از مقاله خود SIFT استخراج شده است:



یک مثال از Orientation Assignment

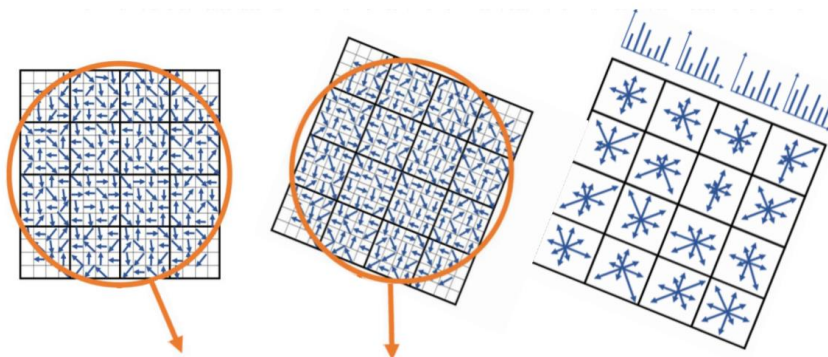
نکته خیلی مهم: جهت این فلش ها، در اصل orientation نقاط کلیدی را نشان می دهد و اندازه آن ها در اصل سیگمای interest point را نشان می دهد.

مرحله ششم: Interest Point Descriptor

- 1- ابتدا پنجره با سایز مشخصی حول interest point در نظر میگیریم (برای اینکه می خواهیم در نهایت این توصیف گر ها را با یکدیگر match کنیم)، بدین منظور یک پنجره به اسکیل 16 در 16 در نظر میگیریم.
نکته: بله این سایز ثابت گرفتن مشکلات ناچیزی ایجاد می کند اما در نهایت مشاهده می کنیم که توسط کیفیت و دقت سایر قسمت ها، اثر خطای ایجاد شده در این قسمت بسیار اندک است و در عوض ثابت فرض کردن آن به سریع تر شدن محاسبات کمک می کند.
- 2- مجدد blur شده ی تصویر شده در همان اسکیلی که interest point را استخراج کردیم را برمیداریم.
- 3- حال این پنجره 16 در 16 را در جهت منفی زاویه ی interest point می چرخانیم تا orientation نهایی این پنجره 16 در 16 صفر شود و لذا مستقل از زاویه ی interest point کار می کند و لذا در نهایت توصیف گرمان نسبت به دوران، مقاوم خواهد بود.
در اصل گرادیان ها را منهای زاویه interest point ها می کند (که به مثابه ی گفته ذکر شده است).
- 4- حال یک وزن گوسی روی این پنجره می اندازیم.
- 5- در هر کدام از بلوک های 4 در 4 از این پنجره 16 در 16 هیستوگرام درست می کنیم.
توجه: هیستوگرام های این مرحله دارای 8 بین اند.
- 6- حال هیستوگرام ها را concat و در نهایت normalize نیز می کنیم و بدین طریق توصیف گر ما به دست می آید.

نکته: سائز بردار توصیفگر ما 128 است، چرا؟ چون 16 تا بلوک داشتیم (توجه کنید که پنجره اولیه 16 در 16 بود و ما بلوک های 4 در 4 از آن استخراج کردیم و لذا 16 عدد بلاک خواهیم داشت) و در هر بلوک نیز یک هیستوگرام 8 bin داشتیم و لذا $16 \times 8 = 128$ برابر طول descriptor خواهد بود.

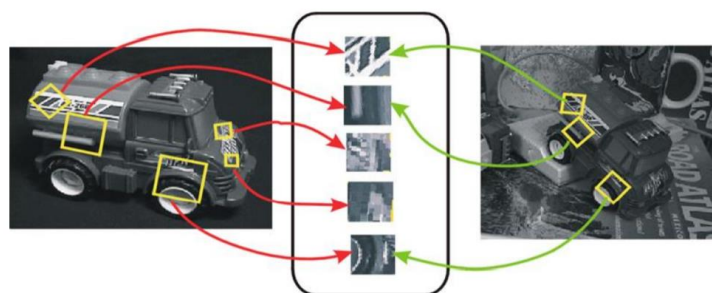
تصویر زیر روند گفته شده را نشان می دهد:



مراحل بخش interest Point Descriptor به بیان تصویر

خب حال که روش SIFT را با دقت بالا توصیف کردیم، چند نمونه مثال از نحوه یافتن نقاط کلیدی به کمک این متد را نشان می دهیم.

توصیف گر ساخته شده نسبت به تغییرات روشنایی، اسکیل و دوران بسیار مقاوم است، به تصویر زیر توجه کنید که با وجود هر سه تغییر رنگ و اسکیل و دوران، نقاط کلیدی به خوبی استخراج شده اند.



یک مثال از SIFT:

وقتی می خواهیم interest point هارا مچ کنیم چنین عمل می کنیم:

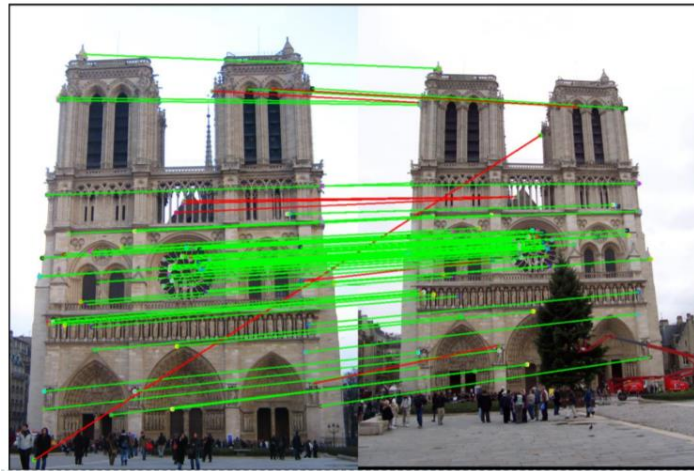
1- ابتدا interest point هارا به دست می آوریم در هر تصویر.

2- حال به ازای هر یک از interest point های تصویر اول، روی interest point های تصویر دوم جاروب می زنیم تا ببینیم کدام یک از این interest point ها از نقطه ی کلیدی تصویر اول کمترین فاصله ی گوسی را دارد. البته ما match دوم را هم پیدا می کنیم و اگر match دوم، به اندازه کافی از match اول دور بود آنگاه match اول را به عنوان نقطه متناظر در نظر میگیریم در غیر اینصورت صرف نظر میکنیم.

این تکنیک به صورت زیر انجام می شود:

- 8- نسبت فاصله ی دو match اول را از interest point، به دست می آوریم (مچ اول به مچ دوم).
- 9- حال اگر این نسبت از 0.8 کوچکتر بود آنگاه مچ اول انتخاب می شود و اگر از 0.8 بیشتر بود، آنگاه با ابهام مواجه هستیم و هیچ کدام انتخاب نمی گردند.

یک نمونه مثال از یافتن نقاط کلیدی متناظر:



خطوط قرمز بیانگر matching های نادرست است و خطوط سبز بیانگر matching های درست است.

برای شفاف تر شدن نحوه کار و استفاده از SIFT یک نمونه کد با نام `InterestPointDetection&Matching` به زبان پایتون ضمیمه شده است که به دو روش SIFT و ORB به استخراج نقاط کلیدی می پردازد (توجه کنید که به جای SIFT از روش AKAZE که احیا شده ی SIFT است و به طور معمول به جای آن استفاده می شود، استفاده شده است).

روش SIFT در میان روش های دیگر تشخیص نقاط کلیدی از قدرت بالایی برخوردار است، برای نشان دادن این موضوع یک مقایسه بین دو روش معروف ORB و SIFT نیز انجام خواهیم داد.

کد این قسمت را نیز ضمیمه کرده ام و در ادامه خروجی آن را نشان می دهم، در این کد، نقاط متناظر بین دو تصویر زیر را یافته و به هم وصل می کنیم.



و



برای شفاف تر شدنِ کد ضمیمه شده، خروجی آن را تشریح می کنیم:

1- روش AKAZE: ابتدا نقاط کلیدی سمت چپ و راست تصاویر را نشان می دهیم و سپس به رسم می پردازیم:



نقاط کلیدی سمت راست



نقاط کلیدی سمت چپ

حال به کمک Matcher به رسم خطوط بین این نقاط کلیدی می پردازیم:

نکته 1: در حالت عادی به خاطر شباهت زیاد دو تصویر تعداد نقاط کلیدی زیادی بین دو تصویر پیدا می شود و رسم کردن همه آن ها شکل نهایی را بسیار شلوغ می کند لذا بنده همانطور که در کد نیز مشخص است بخشی از این نقاط را ترسیم کرده ام:



تصویر حاصل از اتصال نقاط کلیدی تصویر چپ و راست

2- روش ORB: ابتدا نقاط کلیدی سمت چپ و راست تصاویر را نشان می دهیم و سپس به رسم می پردازیم:



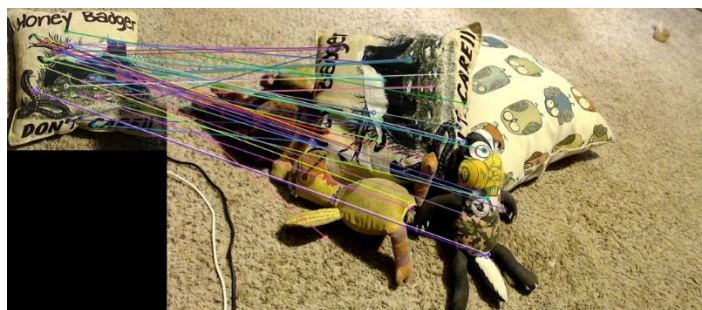
نقاط کلیدی تصویر سمت راست



نقاط کلیدی تصویر سمت چپ

حال به کمک Matcher به رسم خطوط بین این نقاط کلیدی می پردازیم:

نکته 1: در حالت عادی به خاطر شباهت زیاد دو تصویر تعداد نقاط کلیدی زیادی بین دو تصویر پیدا می شود و رسم کردن همه آن ها شکل نهایی را بسیار شلوغ می کند لذا بنده همانطور که در کد نیز مشخص است بخشی از این نقاط را ترسیم کرده ام:



تصویر حاصل از اتصال نقاط کلیدی تصویر چپ و راست

پس از انجام تست های مختلف به کمک این دو الگوریتم به نتایج زیر دست یافتیم.

به طور میانگین روش AKAZE نسبت به روش ORB از دقت بیشتری در هنگام مواجه شدن با

- چرخش
- تغییر در عمق
- تغییر در مقدار روشنایی

دارد.

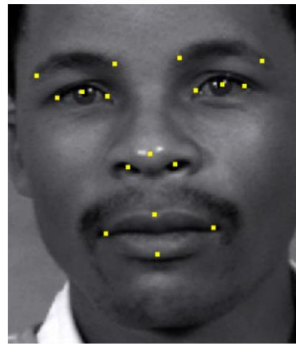
هم چنین در برخی تست ها که تعداد بیشتری از interestPoint ها را رسم و بررسی کردیم مشاهده کردیم که الگوریتم AKAZE علاوه بر بیشتر مقاوم بودن نسبت به موارد ذکر شده در تشخیص نقاط از دقت بیشتری برخوردار است و کمتر در اتصال نقاط کلیدی مختلف به یکدیگر دچار خطا می شود.

تکنیک سوم: یافتن landmark های چهره.

ابتدا نیاز است تعریفی از landmark ارایه کنیم:

A landmark represents a **distinguishable** point present in **most** of the images under **consideration**, for example the **nose tip**.

دسته ای از landmark های مرتبط با یک پدیده که به اصطلاح config (الگوی وجودی) آن پدیده را رقم می زند. مثلاً به تصویر زیر که دسته landmark های چهره است توجه کنید (همانطور که مشاهده می کنید به کمک این landmark ها ساختار چهره به خوبی در می آید).



A landmarked face

روش های مختلفی برای یافتن landmark های چهره وجود دارد و یکی از معروف ترین های آن ها روش ASM (Active Shape Model) است اما این روش مراحل و دشواری هایی دارد که پیاده سازی آن را به صورت realTime ناممکن می کند و حال به دنبال ساختن نسخه ای احیاشده از این روش هستیم تا انجام بر لحظه ای آن را ممکن سازیم.

پیش از پرداختن به جزییات عملی که منتها به نیل به هدف فوق می شود نیاز است تا با تکنیک MARS آشنا شویم. در آمار، MARS یا همان Multivariate adaptive regression splines یک تحلیل رگرشن (regression) است که برای اولین بار در تاریخ 1991 توسط آقای Jerome H. Friedman مطرح گردیده است. این تکنیک در اصل همانند یک افزوده (extension) برای مدل های خطی است که منتها به در نظر گرفتن و مدل سازی روابط غیر خطی و interaction های بین پارامترهایی که اندازه گیری می شوند، می گردد. برای تعمیق و مطالعه ی دقیق تر این تکنیک [به این لینک](#) مراجعه کنید.

حال گفتیم که به می خواهیم به کمک ترفندی، روش ASM را به یک نسخه قابل اجرا به صورت realtime تبدیل کنیم. در اصل برای نیل به هدف گفته شده، به جای قسمت 1D gradient profile از ASM عادی، از یک نسخه ساده شده از توصیف گر های SIFT به کمک MARS (جهت تطابق توصیفگر ها) استفاده می کنیم.

جهت اطلاع از کیفیت عملکردی ASM تغییر یافته، گفته صریح نویسنده مقاله (Stephen Milborrow) را ذکر می کنیم.

This modified ASM is fast and performs well against existing techniques for automatic face landmarking on frontal faces.

هم چنین در ادامه این پژوهش به معرفی تکنیک هایی جهت تسریع چشم گیر حجم محاسبات نیز می پردازیم (تکنیک هایی که منتهای اجرای بر لحظه ی مدل های از ASM که بر مبنای SIFT اند، می گردد).

حال که با کلیت landmark آشنا شده ایم به نحوه ی تشخیص این landmark ها به کمک ASM می پردازیم.

در نظر بگیرید که ما یک چهره پایه که نقاط landmark آن استخراج شده است را داریم (این چهره پایه طبق اشاره مقاله می تواند توسط میانگین گیری از چهره های landmark شده موجود در مرحله train به دست آید)، در این پژوهش نام این تصویر پایه حاوی landmark را baseLandmarkedFace می گذاریم. حال ASM به این طریق نقاط landmark یک تصویر (که حاوی چهره است) را استخراج میکند:

مراحل اولیه و پیش پردازش: (یکبار اجرا می شوند)

1- در ابتدا به کمک face detector (که پیشتر جزییات آن ها تشریح داده شد) بخش چهره تصویر را استخراج می کنیم.

2- حال که چهره ها استخراج شدند، ASM یک چهره اولیه ی حاوی landmark ها را به کمک انطباق baseLandmarkedFace بر چهره (ها) استخراج شده به دست می آورد، به عبارت دیگر در این قدم صرفاً ASM بر روی چهره شخص landmark های baseLandmarkedFace را قرار میدهد (به این طریق که ابتدا baseLandmarkedFace را به صورت مناسب (با تغییر اسکیل و orientation های لازم) روی چهره شخص قرار می دهد و سپس landmark های اولیه این چهره را همان landmark های baseLandmarkedFace در نظر میگیرد).

پس از انجام دو مرحله فوق، ASM دو مرحله زیر را آنقدر انجام می دهد که تا خطای به دست آمده از ترشهولد مورد نظر کمتر گردد (مثلاً خطای کمتر از 1 دهم درصد).

در اصل مراحل پردازشی که در صفحه بعد ذکر خواهند شد در داخل این حلقه تکرار قرار میگیرند.

```
While(currentError > targetError){
```

```
Do processing steps.
```

```
}
```

```
// now the currentError <= targetError so the current landmarks's position being
```

```
//accepted.
```

مراحل پردازشی: (این دو مرحله به صورت متناوباً آنقدر اجرا می شوند تا حد خطای به دست آمده مطلوب گردد)

1- این مرحله خیلی شبیه هسته عملکردی روش [K-means](#) است. خب تا به اینجا کار ما یک مکان اولیه از landmark ها داشتیم و به دنبال آن ایم که این مکان های اولیه را پله پله به مکان درست آن در چهره اصلی نزدیک کنیم و به این صورت انجام میگیرد که نقاط همسایه landmark فعلی را (از چهره) استخراج می کنیم، حال این بسته از نقاط را با مدل توصیف گر landmark های موجود (تولید شده در مدل های مرحله train) مقایسه می کنیم.

توجه: مدل های موجود توصیف گر landmark ها، در مرحله training استخراج شده است و مرحله training پیش از این مراحل پردازشی (که در حال ذکر کردنشان ایم) انجام میگیرد.

حال landmark فعلی را به مرکز landmark ای که مدل توصیف گرش بهترین تطابق ساختاری را با بسته نقاط فعلی دارد، انتقال می دهیم.

2- در این مرحله ساختار فعلی landmark هارا با یک ساختار کلی مطابقت می دهیم، این عمل همانند عملکرد لایه pooling در CNN هاست و در اصل این مرحله نتایج نادرستی که ممکن است توسط matcher ها به وجود بیاید را حذف می کند و نقاطی که واضحاً (با توجه به ساختار کلی مورد انتظار) نادرست اند را تصحیح می کند.

سوال: ممکن است بپرسید مرحله اول کافیت و صرفاً با کم ترکردن ترشهولد خطا می توان به هدف مورد دوم هم نایل شد؟

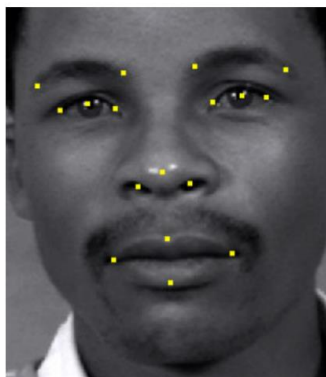
پاسخ: دقت کنید matcher هایی که در مرحله 1 جهت بررسی میزان شباهت بسته نقاط و مدل های توصیف گر landmark ها استفاده می شوند، تنها محدوده کوچکی از چهره را بررسی می کنند و نمی توانند صد رد صد مورد اطمینان باشند.

نکته: در این پژوهش، بحث را با تعمیق حول مرحله اول از مراحل پردازشی ادامه می دهیم.

نکته: همانطور که پیشتر مطرح گردید، جهت حل مشکل عدم robust بودن یک مدل نسبت به scale، نیاز است تا مدل به جای بررسی ورودی خالی، هرم (pyramid) ورودی را مورد بررسی قرار بدهند، لازم به ذکر است که در همین مراحل پردازشی هم همین نکته مورد توجه قرار گرفته و تمامی مراحل به pyramid image که حاوی اسکیل های مختلف ورودی است اعمال می گردد.

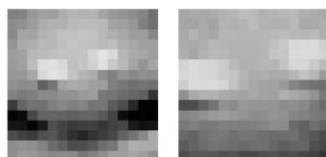
نکته: هرم نکته قبل در 4 اسکیل از درشت تا بیشترین resolution تشکیل می شود (یعنی هرم یک تصویر، شامل 4 اسکیل مختلف از آن تصویر خواهد بود).

مجدد به تصویر ابتدای این بخش توجه کنید:



A landmarked face

حال می خواهیم 2 اسکیل مختلف از 4 اسکیل موجود در هرم بینی این شخص را که همگی به صورت بسته های 15 پیکسل در 15 پیکسل اند را رسم کنیم:



تصویر سمت چپ با اسکیل 0.5

تصویر سمت راست در اسکیل اصلی.

همانطور که پیشتر گفته شد، در این پژوهش ما بحثمان را حولِ مرحله اول از مراحل پردازشی ادامه می دهیم.

ادامه بحث خود را حول زیرموضوعات زیر ادامه می دهیم:

SIFT Descriptors -1

HAT Descriptors -2

Descriptor matching methods -3

MARS -4

:SIFT & HAT descriptors – 1, 2

پیشتر توصیف گر های SIFT را با جزییات نسبتا زیادی بیان کردیم و از ذکر مجدد آنها خودداری می کنیم. توصیف گر های SIFT به عنوان پایه ای برای طراحی توصیف گر های HAT اند. همان طور که پیشتر گفته شد ما در مدل ASM تغییر یافته خود از یک نسخه ساده شده از توصیف گر های SIFT به کمک MARS (جهت تطابق توصیفگر ها) استفاده می کنیم و در اصل ما به کمک توصیف گر های SIFT، توصیف گر های HAT را میسازیم و سپس HAT را در مدل ASM خود استفاده می کنیم.

توصیف گر HAT (Histogram Array Transforms) عمدتا همان توصیف گر های SIFT دوران نیافته با اسکیل ثابت اند.

They are essentially unrotated SIFT descriptors with a fixed scale.

توصیف گر های HAT نسبت به توصیف گر های استاندارد SIFT با سرعت بالایی تولید می شوند و سرعت ترم خیلی مهمی برای ماست چراکه برای به دست آوردن محل landmark ها در چهره ما به محاسبه ای در حدود 25 هزار توصیف گر نیاز داریم.

سوال: این 25 هزار توصیف گر از کجا آمد؟

پاسخ: به طور معمول چهره آدمی 68 landmark دارد و همانطور که پیشتر گفته شد برای robust بودن نسبت به scale نیاز است تا ما هرم های 4 تایی از تصاویر ایجاد کنیم (هرم با 4 اسکیل مختلف) حال داریم:

Num of Calculated descriptors

$$= 68 (\text{landmarks}) \times 4 (\text{pyramid levels}) \times 4 (\text{model iterations})$$

$$\times 5 \times 5 (\text{search Grid}) = 27k \sim 25k$$

حال می خواهیم به جزییات توصیف گر های HAT بپردازیم که شامل 4 مورد زیر است:

- No automatic Scale Determination
- No patch Orientation
- Pre-calculation of Mapping from Patch to Histogram Array
- Descriptor Caching

حال در ادامه هر کدام از 4 مورد فوق را تشریح می کنیم.

• No automatic Scale Determination

همانطور که پیشتر در مورد توصیف گر SIFT مطرح شد، مشاهده کردیم که در به دست آوردن توصیفگر های SIFT، در ابتدا یک مرحله محاسبه ی سنگینِ scale-space analysis جهت ارزیابی اینکه چه نقاطی در تصویر keypoint اند انجام دادیم، هم چنین این مرحله scale ذاتی هرکدام از این key point ها را نیز استخراج کرد، اما در مقابل در ASM، نقاط کلیدی از پیش تعیین شده اند- این نقاط landmark های چهره اند- و همانطور که دیدیم، پیش از اینکه مراحل پردازشی آغاز شود چهره شخص به یک اسکیل ثابت prescale می گردد. (این مرحله جهت initial کردن اولیه landmark ها روی چهره شخص انجام شد)؛ لذا با توجه به مطلب گفته شده در ASM دیگر نیاز نیست که مرحله سنگین scale space انجام گیرد.

نکته مهم: از آنجا که در ASM هرم با 4 octave برای تصویر، مورد بررسی قرار میگیرد، لذا دیگر نیازی به هرم های درون مراحل SIFT نمی باشد.

• No patch orientation

همانطور که در مراحل SIFT مشاهده کردید، ما جهت تعیین orientation نقاط کلیدی از زاویه ی dominant هیستوگرام زاویه ها استفاده می کردیم در حالی که در ASM، پیش از آنکه مراحل پردازشی ما آغاز شود، تصویر اصلی آنقدر rotate می شود که چشم های شخص کاملاً افقی گردد، لذا بخش تعیین orientation در مراحل SIFT descriptors که جهت robust کردن SIFT نسبت به rotation است آنقدر الزامی نیست.

توجه: البته برخی rotational variarion هم چنان وجود خواهند داشت چرا که همه چهره ها یکسان نیستند و مثلاً گاهی وقتاً eye detector ها، مکان چشم شخص را تنیده با ابرو شخص بیان می کنند که باعث رخداد mis-positioning در شروع کار ASM می گردد و لذا هم چنان نیاز است که برخی ترم های robust کننده نسبت به rotation هم چنان باقی بمانند.

• Pre-calculation of Mapping from Patch to Histogram Array

هنگام تولید توصیف گر SIFT یا HAT نیاز است که هر پیکسل در بسته (patch) به یک مکان در آرایه ی هیستوگرام ها map شود، محاسبه این mapping زمانبر است (مخصوصاً در حالاتی که بسته های ما scale و rotate شده باشند) و توجه کنید که این تعیینِ mapping باید هر زمان که یک توصیف گر ساخته می شود انجام گیرد.

اما توصیف گر های HAT آنقدر از scaling و rotation استفاده نمی کنند و لذا تعیینِ mapping تنها به histogram array dimensions (که برابر 5×4 است) و image patch width که برابر 15 پیکسل است، وابسته خواهد بود و دو پارامتر ذکر شده حین عملیات ASM ثابت است؛ بنابراین ما می توانیم یکبار همه mapping های موجود را طی یک پیش محاسبه، فقط یک بار برای همه توصیف گر ها انجام دهیم (بهتر است بگوییم یکبار در هر مرحله از اسکیل های موجود در هرم).

نکته: بنا به گفته مقاله انجام این پیش محاسبه حدود 40 درصد مدت زمان کل محاسبات بخش پردازشی ASM را کاهش داد.

• Descriptor Caching

بنا به دلیلی که در ادامه گفته می شود با caching توصیف گر ها به میزان خوبی زمان اجرای مرحله پردازشی ASM افزایش می یابد.

1- در مرحله پردازشی ASM ما مکررا با مختصات تقریباً یکسانی از تصویر سروکار داریم (حین مراحل iterative).

2- در درشت ترین اسکیل هرم (coarse scale of pyramid) نیز محدوده مورد بررسی ASM اغلب هم پوشانی زیادی دارد.

بنا به گفته مقاله انجام این مرحله نزدیک به 70 درصد زمان اجرای ASM را ارتقا داد.

با لحاظ کردن مراحل گفته شده، توصیف گر های HAT را خواهیم داشت که هم به دلیل نکات ریز گفته شده سرعت بسیار بالایی دارند و هم چون الهام گرفته از توصیف گر های SIFT اند، دقت بالایی خواهند داشت. همانطور که پیشتر مشاهده کردید مراحل تفحص خود را روی ASM به 4 مرحله زیر شکاندیم:

1- SIFT Descriptors

2- HAT Descriptors

3- Descriptor matching methods

4- MARS

حال که موارد اول و دوم به خوبی تشریح شده اند به مراحل سوم و چهارم می پردازیم.

3- Descriptor matching methods:

همانطور که در مرحله اول از مراحل پردازشی ASM گفتیم، نیاز داریم که هنگامی که برای یک patch از تصویر، یک توصیف گر به دست آوردیم، کیفیت این توصیف گر را به طریقی محاسبه کنیم. در این بخش به بررسی معیار های رایج descriptor matching ای که تکیه بر تکنیک MARS دارند، می پردازیم.

فاصله اقلیدسی:

بدیهی ترین معیاری که به ذهن می رسد فاصله اقلیدسی است، اما این معیار بنا به این دلیل که به همه bin های هیستوگرام وزن برابری میدهد (یعنی همه پیکسل ها با وزن یکسان در مقایسه descriptor ها شرکت می کنند) معیار مناسبی نیست. تخصیص وزن برابر از دو جهت مطلوب ما نیست.

- 1- پیکسل هایی که در لبه های خاصی قرار دارند از اهمیت بیشتری نسبت به سایر پیکسل ها برخوردارند.
- 2- هم چنین پیکسل هایی که واریانس بیشتری دارند بیشتر مستعد نویز بودن اند و بهتر است وزن کمتری به آن ها تخصیص داده شود.

فاصله مویوس:

این معیار برخلاف معیار اقلیدسی به bin ها وزن یکسان نمی دهد بلکه به bin های مهم وزن بیشتر داده (این معیار bin ای را مهم می پندارد که واریانس کمتری داشته باشد لذا طبیعتا نویز ها اثر کمتری خواهند داشت) و نیز رابطه بین bin ها را نیز در نظر گرفته و برخلاف معیار قبلی به هر bin به صورت مستقل نگاه نمی کند. معیار مویوس برای پدیده هایی که از توزیع گوسی پیروی می کنند خیلی خوب پاسخ می دهد و همانطور که احتمالا می دانید توزیع رنگ در تصاویر چهره تا حد خوبی شبیه به توزیع گوسی است.

این معیار از لحاظ محاسباتی سرعت کمی دارد و از $O(n^2)$ است که n برابر تعداد descriptor هاست، همانطور که ملاحظه می کنید این order برای محاسبات سنگین مطلوب نمی باشد.

نکته: در فریم ورک SIFT، برای بررسی descriptor matching از معیار Nearest Neighbor استفاده می شود که طبیعتا در نظر گرفتن این معیار برای ASM اصلا مناسب نمی باشد چرا که هم حافظه زیاد می طلبد و هم از لحاظ محاسباتی کند است.

:Regression

یک تکنیک دیگر استفاده کردن از یک مدل رگرشن است که نمونه های توصیف گر ها را در و حول محل صحیح landmark ها در تصاویر training دریافت می کند و یک مدلی train می کند که کیفیت تطابق descriptor های ورودی را تخمین می زند.

طبیعتا معمول ترین رگرشنی که به ذهن می رسد، linear Regression است اما این رگرشن دو ضعف عمده دارد:

- 1- این رگرشن اثرات غیرخطی موجود را در نظر نمی گیرد.
- 2- این رگرشن به طور پیشفرض میپندازد که المان های ورودی بایکدیگر رابطه ای ندارند (مستقل از یکدیگر اند). لذا از آنجا که این دو فرض در matching descriptor ها اصلا فرض های معقولی نیستند لذا از معیار رگرشن خطی استفاده نمی کنیم.

استفاده کردن از SVM (Support Vector Machines) در این مرحله نیز یکی دیگر از تکنیک های رایج است، اگرچه این تکنیک از کیفیت خروجی خوبی برخوردار است اما سرعت train کردن آن با توجه به آنکه هدف اجرای بر لحظه داریم مناسب نمی باشد.

بنابراین سراغ معیار MARS (Multivariate Adaptive Regression Spline) می رویم که تا حد خوبی در حد SVM کیفیت و درستی خروجی دارد و علی ذلک از سرعت train و اجرای خوبی نیز برخوردار است. با توجه به گفته نگارنده مقاله داریم که استفاده از MARS سرعت انجام مرحله پردازش ASM را تا حد 80 درصد بهبود داد.