**فصل 8. کار با تصاویر**

**8.0 مقدمه**

دسته‌بندی تصاویر یکی از هیجان‌انگیزترین بخش‌های علم یادگیری ماشین است. توانایی کامپیوترها در تشخیص الگوها و اشیاء از تصاویر، یک ابزار فوق‌العاده قدرتمند در پایتون است. با این حال، پیش از اینکه بتوانیم یادگیری ماشینی را روی تصاویر اعمال کنیم، در اغلب موارد ابتدا نیاز داریم که تصاویر خام را به ویژگی‌های قابل استفاده توسط الگوریتم‌های یادگیری خود تبدیل کنیم.

برای کار با تصاویر ازOpenCV[[1]](#footnote-1) استفاده می‌کنیم. علی‌رغم وجود کتابخانه‌های دیگر، OpenCV محبوب‌ترین و مستندترین کتابخانه برای مدیریت تصاویر است. یکی از بزرگ‌ترین موانع استفاده از OpenCV نصب آن است. با این حال، خوشبختانه اگر از Python 3 استفاده کنیم، (در زمان انتشار کتاب، OpenCV با Python 3.6 کار نمی‌کند)، می‌توانیم از ابزار Anaconda، کوندا، برای نصب OpenCV در یک خط کد در ترمینال خود استفاده کنیم:

conda install --channel <https://conda.anaconda.org/menpo> opencv3

پس از آن، می‌توانیم با باز کردن یک نوت بوک، وارد کردن OpenCV و بررسی شماره نسخه (3.1.0)، نصب را بررسی کنیم:

import cv2

cv2.\_\_version\_\_

اگر نصب OpenCV با استفاده از conda کار نمی‌کند، راهنماهای زیادی به صورت آنلاین وجود دارد.

در نهایت، در طول این فصل از مجموعه‌ای از تصاویر به عنوان نمونه استفاده خواهیم کرد که در [GitHub](https://github.com/chrisalbon/sim_data/tree/main/images) جهت دانلود در دسترس هستند.

**8.1 وارد کردن تصاویر**

**مسئله**

شما می‌خواهید یک تصویر را برای پیش پردازش بارگذاری کنید.

**راه حل**

از imread در کتابخانه‌ی OpenCV استفاده کنید:



می توان برای نمایش تصویر از کتابحانه Matplotlib در پایتون بهره گرفت.





**بحث**

می توان گفت، تصاویر، داده‌ها هستند و وقتی از imread استفاده می‌کنیم، آن داده‌ها را به یک نوع داده تبدیل می‌کنیم که خیلی با آن آشنا هستیم - یک آرایه NumPy:



ما تصویر را به ماتریسی تبدیل کرده‌ایم که عناصر آن به پیکسل‌های جداگانه مطابقت دارند. حتی می‌توانیم به مقادیر واقعی ماتریس نگاهی بیندازیم:



وضوح تصویر ما 3600 × 2270 بود، ابعاد دقیق ماتریس ما:



هر عنصر در ماتریس در واقع چه چیزی را نشان می‌دهد؟ در تصاویر با مقیاس خاکستری، مقدار یک عنصر منفرد، شدت پیکسل است. مقادیر شدت از سیاه (0) تا سفید (255) متغیر است. به عنوان مثال، شدت بالای سمت راست‌ترین پیکسل در تصویر ما دارای مقدار 140 است:



در ماتریس، هر عنصر شامل سه مقدار مربوط به مقادیر آبی، سبز، قرمز (BGR) است:



یک نکته کوچک: به طور پیش‌فرض OpenCV از BGR استفاده می‌کند، اما بسیاری از برنامه‌های تصویری - از جمله Matplotlib - از قرمز، سبز، آبی (RGB) استفاده می‌کنند، به این معنی که مقادیر قرمز و آبی با هم عوض می‌شوند. برای نمایش صحیح تصاویر رنگی OpenCV در Matplotlib، ابتدا باید رنگ را به RGB تبدیل کنیم (با عرض پوزش از خوانندگان نسخه چاپی که تصاویر رنگی را نمی‌توانند ببینند):



****

**منابع بیشتر**

[Difference between RGB and BGR](https://oreil.ly/N1Ub6)

[RGB color model](https://en.wikipedia.org/wiki/RGB_color_model)

**8.2 ذخیره‌سازی تصویر**

**مسئله**

شما می‌خواهید یک تصویر را برای پیش پردازش ذخیره کنید.

**راه حل**

از imwrite در OpenCV استفاده کنید.



**بحث**

دستور imwrite در OpenCV تصاویر را در مسیر فایل مشخص شده ذخیره می‌کند. فرمت تصویر با پسوند نام فایل (jpg، png و غیره) تعریف می‌شود. یک رفتار که باید مراقب آن بود: imwrite فایل‌های موجود را بدون خروجی خطا یا درخواست تایید بازنویسی می‌کند.

**8.3 تغییر ابعاد تصویر**

**مسئله**

شما می‌خواهید اندازه یک تصویر را برای پیش‌پردازش بیشتر تغییر دهید.

**راه حل**

برای تغییر اندازه تصویر از تابع resize در OpenCV استفاده کنید:



**

**بحث**

تغییر اندازه تصاویر به دو دلیل یک کار رایج در پیش پردازش تصویر است. اولاً، تصاویر در همه اشکال و اندازه‌ها هستند و برای اینکه به عنوان ویژگی قابل استفاده باشند، تصاویر باید ابعاد یکسانی داشته باشند. با این حال، این استانداردسازی اندازه‌ی تصویر با هزینه‌هایی همراه است. تصاویر ماتریس‌های اطلاعات هستند و وقتی اندازه تصویر را کاهش می‌دهیم، اندازه آن ماتریس و اطلاعات موجود در آن را کاهش می‌دهیم. علم یادگیری ماشین می‌تواند به هزاران یا صدها هزار تصویر نیاز داشته باشد. هنگامی که آن تصاویر، بسیار بزرگ هستند، می‌توانند حافظه زیادی را اشغال کنند، و با تغییر اندازه آنها می‌توانیم به طور چشمگیری استفاده از حافظه را کاهش دهیم. برخی از اندازه‌های رایج تصویر برای یادگیری ماشینی 32 × 32، 64 × 64، 96 × 96 و 256 × 256 هستند. در اصل روشی که ما برای تغییر اندازه استفاده می‌کنیم تعادلی بین عملکرد آماری مدل و هزینه محاسباتی آموزش آن خواهد بود. کتابخانه [Pillow](https://pillow.readthedocs.io/en/stable/reference/Image.html#PIL.Image.Image.resize) انتخاب‌های زیادی برای تغییر اندازه تصاویر در اختیار ما قرار می‌دهد.

**8.4 برش تصویر**

**مسئله**

شما می‌خواهید قسمت بیرونی تصویر را برش دهید و بردارید تا ابعاد آن را تغییر دهید.

**راه حل**

تصویر به یک آرایه دو بعدی numpy تبدیل شده و می‌توان آن را به راحتی به دو یا چند بخش تقسیم کرد.





**بحث**

از آنجایی که OpenCV تصاویر را به عنوان ماتریسی از عناصر نشان می‌دهد، با انتخاب سطرها و ستون‌هایی که می‌خواهیم نگه داریم می‌توانیم به راحتی تصویر را برش دهیم. برش می‌تواند به ویژه مفید باشد اگر بدانیم که فقط می‌خواهیم قسمت خاصی از هر تصویر را حفظ کنیم. به عنوان مثال، اگر تصاویر ما از یک دوربین امنیتی ثابت تهیه شده باشند، می‌توانیم تمام تصاویر را برش دهیم تا فقط ناحیه مورد نظر را در بر داشته باشند.

**منابع بیشتر**

[Slicing NumPy Arrays](https://numpy.org/doc/stable/user/basics.indexing.html#slicing-and-striding)

**8.5 شفاف‌سازی تصویر**

**مسئله**

شما می‌خواهید یک تصویر را نرم(محو) کنید.

**راه حل**

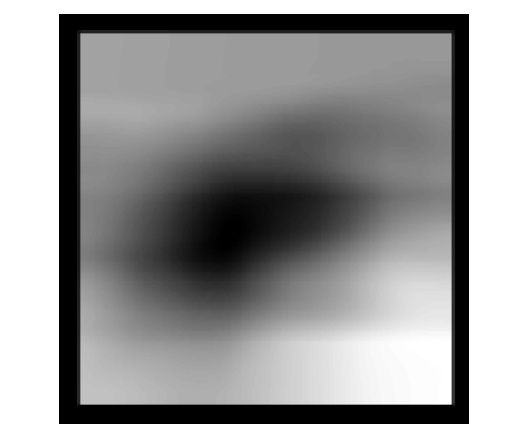
برای محو کردن یک تصویر، هر پیکسل به مقدار متوسط همسایگان خود تبدیل می‌شود. این همسایه و عملیات انجام شده از نظر ریاضی به عنوان یک کرنل نمایش داده می‌شوند (اگر نمی‌دانید کرنل چیست نگران نباشید). اندازه این کرنل میزان تاری را تعیین می‌کند و کرنل‌های بزرگتر، تصاویر صاف‌تری تولید می‌کنند. در اینجا ما یک تصویر را با میانگین مقادیر یک کرنل 5 × 5 در اطراف هر پیکسل محو می‌کنیم:



****

برای برجسته کردن اثر اندازه‌ی کرنل، در اینجا همان محو کردن با یک کرنل 100 × 100 وجود دارد.





**بحث**

کرنل‌ها به طور گسترده‌ای در پردازش تصویر برای انجام همه چیز از تیز کردن لبه تا تشخیص لبه استفاده می‌شوند و در این فصل بارها و بارها مطرح خواهند شد. کرنل محو‌کننده‌ای که استفاده کردیم به این صورت است:



عنصر مرکزی در کرنل پیکسل مورد بررسی است، در حالی که عناصر باقی مانده همسایه‌های آن هستند. از آنجایی که همه عناصر دارای مقادیر یکسانی هستند (نرمال شده که حاصل مجموع کرنل برابر 1 باشد)، هر یک در مقدار حاصل از پیکسل مورد نظر تاثیر یکسانی دارند. ما می‌توانیم به صورت دستی یک کرنل را با استفاده از filter2D روی یک تصویر اعمال کنیم تا جلوه محو‌کننده‌ی مشابه ایجاد کنیم:





**منابع بیشتر**

[Image Kernels Explained Visually](https://setosa.io/ev/image-kernels/)

[Kernel (image processing), Wikipedia](https://en.wikipedia.org/wiki/Kernel_(image_processing))

**8.6 تیز کردن لبه‌های تصویر**

**مسئله**

شما می‌خواهید یک تصویر را واضح کنید.

**راه حل**

کرنلی ایجاد کنید که پیکسل هدف را برجسته کند. سپس با استفاده از filter2D آن را روی تصویر اعمال کنید:





**بحث**

تیز کردن مانند محو کردن کار می‌کند، با این تفاوت که به جای استفاده از یک کرنل برای میانگین مقادیر همسایه، یک کرنل برای برجسته کردن خود پیکسل ایجاد کردیم. اثر به دست آمده باعث می‌شود تضادهای لبه‌ها در تصویر بیشتر برجسته شود.

**8.7 بهبود کنتراست و تضاد تصویر**

**مسئله**

ما می‌خواهیم کنتراست (تضاد یا تفاوت) بین پیکسل‌های یک تصویر را افزایش دهیم.

**راه حل**

یکسان سازی هیستوگرام ابزاری برای پردازش تصویر است که می‌تواند اشیا و اشکال را برجسته کند. وقتی تصویری در مقیاس خاکستری داریم، می‌توانیم تابع equalizeHist را مستقیماً روی تصویر اعمال کنیم: 



با این حال، زمانی که یک تصویر رنگی داریم، ابتدا باید تصویر را به فرمت رنگی YUV تبدیل کنیم. Y لوما یا روشنایی است و U و V رنگ را نشان می‌دهند. پس از تبدیل، می‌توانیم EqualizeHist را روی تصویر اعمال کنیم و سپس آن را به BGR یا RGB تبدیل کنیم:





**بحث**

در حالی که توضیح دقیق نحوه‌ی کار یکسان‌سازی هیستوگرام فراتر از محدوده این کتاب است، توضیح کوتاه، این است که تصویر را به گونه‌ای تبدیل می‌کند که از طیف وسیع‌تری از شدت پیکسل استفاده می‌کند. در حالی که تصویر حاصل اغلب "واقع گرایانه" به نظر نمی‌رسد، باید به خاطر داشته باشیم که یک تصویر فقط یک نمایش بصری از داده‌های اساسی است. اگر تساوی هیستوگرام بتواند اشیاء مورد علاقه را از دیگر اشیا یا پس‌زمینه‌ها متمایزتر کند (که همیشه اینطور نیست)، آنگاه می‌تواند افزوده‌ای ارزشمند به روند پیش‌پردازش تصویر ما باشد.

**6.8 تفکیک رنگ‌ها**

**مسئله**

شما می‌خواهید یک رنگ را در یک تصویر جدا کنید.

**راه حل**

طیف رنگی را تعریف کنید و سپس یک ماسک روی تصویر اعمال کنید:





**بحث**

جداسازی رنگ‌ها در OpenCV ساده است. ابتدا یک تصویر را به HSV (رنگ، اشباع و مقدار) تبدیل می‌کنیم. سپس، محدوده‌ای از مقادیر را تعریف می‌کنیم که می‌خواهیم جدا کنیم، که احتمالاً سخت‌ترین و زمان‌برترین بخش پردازش همین بخش است. در نهایت، یک ماسک برای تصویر ایجاد می‌کنیم (فقط مناطق سفید را حفظ می‌کنیم):





در نهایت ماسک (کرنل) را با استفاده از bitwise\_and روی تصویر اعمال می‌کنیم و به فرمت خروجی دلخواه خود تبدیل می‌کنیم

**8.9 باینری کردن تصویر**

**مسئله**

با توجه به یک تصویر، می‌خواهید یک نسخه ساده شده را خروجی بگیرید.

**راه حل**

آستانه‌گذاری فرآیندی است که در آن پیکسل‌هایی با شدت بیشتر از مقداری به رنگ سفید و کمتر از مقدار سیاه هستند. یک تکنیک پیشرفته‌تر، آستانه تطبیقی است که در آن، مقدار آستانه برای یک پیکسل با شدت پیکسل‌های همسایگان آن تعیین می‌شود. هنگامی که شرایط نور در مناطق مختلف یک تصویر تغییر می‌کند، می‌تواند مفید باشد:

**بحث**

راه حل ما دارای چهار آرگومان مهم در AdaptiveThreshold است. max\_output\_value به سادگی حداکثر شدت پیکسل‌های خروجی را تعیین می‌کند. cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C آستانه پیکسل را به صورت مجموع وزنی از شدت پیکسل‌های مجاور تعیین می‌کند. وزن‌ها توسط یک پنجره گاوسی تعیین می‌شوند. یا می‌توانیم آستانه را فقط روی میانگین پیکسل‌های همسایه با متغیر ADAPTIVE\_THRESH\_MEAN\_C در cv2 تنظیم کنیم:





دو پارامتر آخر عبارتند از اندازه بلوک (اندازه همسایگی که برای تعیین آستانه پیکسل استفاده می‌شود) و مقدار ثابتی که از آستانه محاسبه شده کم می‌شود (برای تنظیم دقیق آستانه به صورت دستی استفاده می‌شود). یکی از مزایای اصلی آستانه‌گذاری، حذف نویز از یک تصویر است - فقط حفظ مهمترین عناصر. به عنوان مثال، آستانه‌گذاری اغلب برای عکس‌های متن چاپ شده اعمال می‌شود تا حروف را از صفحه جدا کند.

**8.10 حذف پس‌زمینه**

**مسئله**

شما می‌خواهید پیش‌زمینه یک تصویر را جدا کنید.

**راه حل**

یک مستطیل در اطراف پیش زمینه مورد نظر علامت‌گذاری کنید، سپس الگوریتم GrabCut را اجرا کنید:



. 

**بحث**

اولین چیزی که متوجه می‌شویم این است که اگرچه GrabCut کار بسیار خوبی انجام داده است، اما هنوز قسمت‌هایی از پس‌زمینه در تصویر باقی مانده است. می‌توانیم به عقب برگردیم و آن مناطق را به صورت دستی به عنوان پس‌زمینه علامت‌گذاری کنیم، اما در دنیای واقعی هزاران تصویر داریم و تعمیر دستی آنها به صورت جداگانه امکان‌پذیر نیست. بنابراین، با پذیرش اینکه داده‌های تصویر همچنان دارای مقداری نویز پس‌زمینه هستند، خوب عمل می‌کنیم. در این راه حل، با علامت‌گذاری یک مستطیل در اطراف ناحیه‌ای که شامل پیش زمینه است، شروع می‌کنیم. GrabCut همه چیز خارج از این مستطیل را به عنوان پس‌زمینه فرض می‌کند و از آن اطلاعات برای تعیین پس‌زمینه احتمالی داخل مربع استفاده می‌کند. سپس ماسکی ایجاد می‌شود که مناطق مختلف قطعاً/احتمالی پس زمینه/پیش زمینه را نشان می‌دهد:





ناحیه سیاه ناحیه‌ای خارج از مستطیل ما است که به طور قطع پس زمینه فرض می‌شود. ناحیه خاکستری همان چیزی است که GrabCut پس‌زمینه محتمل در نظر می‌گیرد، در حالی که ناحیه سفید احتمالاً پیش‌زمینه است. سپس از این ماسک برای ایجاد یک ماسک دوم استفاده می‌شود که مناطق سیاه و خاکستری را ادغام می‌کند.





سپس ماسک دوم روی تصویر اعمال می‌شود تا فقط پیش زمینه باقی بماند.

**6.11 لبه‌یابی**

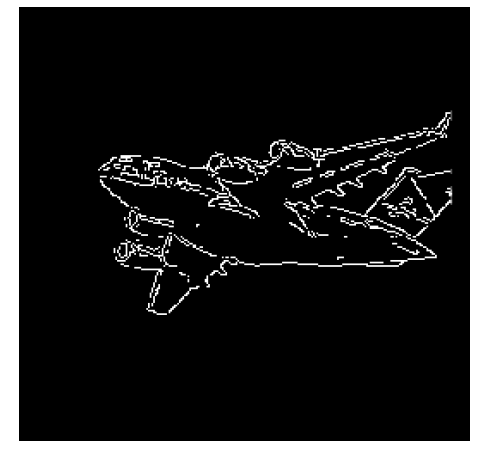
**مسئله**

شما می‌خواهید لبه‌های یک تصویر را پیدا کنید.

**راه حل**

از یک تکنیک تشخیص لبه مانند آشکارساز لبه Canny استفاده کنید:



. 

**بحث**

تشخیص لبه موضوع اصلی مورد علاقه در بینایی کامپیوتر است. لبه‌ها مهم هستند زیرا مناطقی با اطلاعات بالا هستند. به عنوان مثال، در تصویر ما یک قسمت از آسمان بسیار شبیه به دیگری است و بعید است که حاوی اطلاعات منحصر به فرد یا جالب باشد. با این حال، تکه‌هایی که آسمان پس‌زمینه با هواپیما برخورد می‌کند حاوی اطلاعات زیادی هستند (مثلاً شکل یک شیء). تشخیص لبه به ما این امکان را می‌دهد که مناطق کم اطلاعات را حذف کنیم و مناطقی از تصاویر را که حاوی بیشترین اطلاعات هستند را جدا کنیم. تکنیک‌های زیادی برای تشخیص لبه وجود دارد (فیلترهای سوبل، آشکارساز لبه لاپلاس و غیره). با این حال، راه حل ما از آشکارساز لبه Canny استفاده می‌کند. نحوه عملکرد آشکارساز Canny برای این کتاب بسیار مفصل است، اما یک نکته وجود دارد که باید به آن بپردازیم. آشکارساز Canny به دو پارامتر نیاز دارد که مقادیر آستانه گرادیان کم و زیاد را نشان می‌دهد. پیکسل‌های لبه بالقوه بین آستانه‌های پایین و بالا، پیکسل‌های لبه ضعیف در نظر گرفته می‌شوند، در حالی که پیکسل‌های بالای آستانه، پیکسل‌های لبه قوی در نظر گرفته می‌شوند. روش Canny در کتابخانه‌ی OpenCV شامل آستانه‌های پایین و بالا به عنوان پارامترهای مورد نیاز است. در راه حل خود، آستانه‌های پایین و بالایی را یک انحراف استاندارد زیر و بالاتر از شدت پیکسل متوسط تصویر قرار می‌دهیم. با این حال، اغلب مواردی وجود دارد که اگر قبل از اجرای Canny روی کل مجموعه تصاویرمان، از یک جفت مقادیر آستانه پایین و بالا از طریق آزمون و خطای دستی با استفاده از چند تصویر استفاده کنیم، ممکن است نتایج بهتری به دست آوریم.

**منابع بیشتر**

* [Canny Edge Detector](https://en.wikipedia.org/wiki/Canny_edge_detector)
* [Canny Edge Detection Auto Thresholding](http://www.kerrywong.com/2009/05/07/canny-edge-detection-auto-thresholding/)

**6.12 گوشه‌یابی**

**مسئله**

شما می‌خواهید گوشه‌ها را در یک تصویر تشخیص دهید.

**راه حل**

از آشکارساز گوشه‌ی هریس (cornerHarris) در OpenCV استفاده کنید:



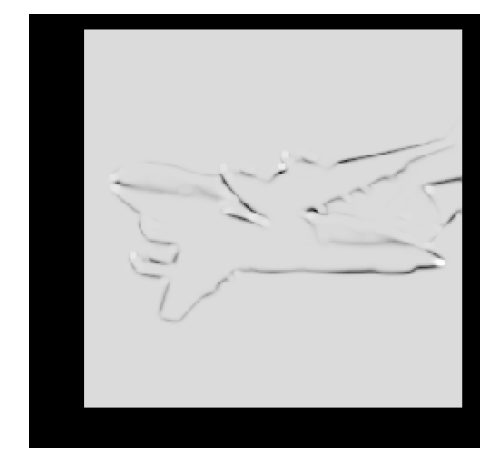


**بحث**

آشکارساز گوشه هریس یک روش رایج برای تشخیص تقاطع دو لبه است. علاقه ما به تشخیص گوشه‌ها به همان دلیلی است که برای حذف لبه‌ها وجود دارد: گوشه‌ها نقاط اطلاعات بالایی هستند. توضیح کامل آشکارساز گوشه هریس در منابع خارجی در انتهای این بخش موجود است، اما توضیح ساده ‌شده این است که به دنبال پنجره‌هایی می‌گردد (که محله‌ها یا تکه‌هایی نیز نامیده می‌شوند) که در آن حرکات کوچک پنجره (تصور کنید پنجره را تکان دهید) تغییرات بزرگی در محتوای پیکسل‌های داخل پنجره ایجاد می‌کند. cornerHarris شامل سه پارامتر مهم است که می‌توانیم از آنها برای کنترل لبه‌های شناسایی شده استفاده کنیم. اول، block\_size اندازه همسایه‌ی اطراف هر پیکسل است که برای تشخیص گوشه استفاده می‌شود. دوم، دیافراگم اندازه کرنل Sobel استفاده شده است (اگر نمی‌دانید چیست، نگران نباشید)، و در نهایت یک پارامتر آزاد وجود دارد که مقادیر بزرگتر مربوط به شناسایی گوشه‌های نرم‌تر است.

خروجی یک تصویر در مقیاس خاکستری است که گوشه‌های بالقوه را نشان می‌دهد:





سپس آستانه‌گذاری را اعمال می‌کنیم تا فقط محتمل ترین گوشه‌ها را حفظ کنیم. از طرف دیگر، می‌توانیم از یک آشکارساز مشابه، آشکارساز گوشه Shi-Tomasi استفاده کنیم، که به روشی مشابه آشکارساز هریس (goodFeaturesToTrack) برای شناسایی تعداد ثابتی از گوشه‌های قوی کار می‌کند. آشکارساز goodFeaturesToTrack سه پارامتر اصلی را در نظر می‌گیرد که آن تعداد گوشه‌های شناسایی است. حداقل کیفیت گوشه (0 تا 1)، و حداقل فاصله اقلیدسی بین گوشه‌ها:





**منابع بیشتر**

* [OpenCV’s cornerHarris](https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=goodfeaturestotrack#cornerharris)
* [OpenCV’s goodFeaturesToTrack](https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html#goodfeaturestotrack)

**8.13 ایجاد ویژگی‌ها برای یادگیری ماشین**

**مسئله**

شما می‌خواهید یک تصویر را به یک نمونه آموزشی یا مشاهده برای یادگیری ماشین تبدیل کنید.

**راه حل**

از flatten در کتابخانه Numpy برای تبدیل آرایه چند بعدی حاوی داده‌های یک تصویر به بردار حاوی مقادیر مشاهده استفاده کنید:



**بحث**

تصاویر به صورت شبکه‌ای از پیکسل‌ها ارائه می‌شوند. اگر تصویری در مقیاس خاکستری باشد، هر پیکسل با یک مقدار نمایش داده می‌شود (یعنی شدت پیکسل: 1 اگر سفید، 0 اگر سیاه است). به عنوان مثال، تصور کنید یک تصویر 10 × 10 پیکسل داریم:





در این حالت ابعاد داده‌های تصاویر 10 × 10 خواهد بود:



و اگر آرایه را flatten کنیم، برداری به طول 100 به دست می‌آید (10 ضربدر 10):



این داده‌های ویژگی تصویر ما است که می‌توان آن‌ها را با بردارهای تصاویر دیگر پیوند داد تا داده‌هایی را ایجاد کند که ما به الگوریتم‌های یادگیری ماشین خود تغذیه می‌کنیم. اگر تصویر رنگی است، به جای اینکه هر پیکسل با یک مقدار نشان داده شود، با چندین مقدار (اغلب سه مقدار) نشان داده می‌شود که نشان دهنده کانال‌هایی (قرمز، سبز، آبی و غیره) است که برای ایجاد رنگ نهایی آن، ترکیب می‌شوند. به همین دلیل، اگر تصویر 10×10 ما رنگی باشد، برای هر مشاهده 300 مقدار ویژگی خواهیم داشت:



یکی از چالش‌های اصلی پردازش تصویر و بینایی کامپیوتری این است که از آنجایی که هر مکان پیکسل در مجموعه‌ای از تصاویر یک ویژگی است، با بزرگتر شدن تصاویر، تعداد ویژگی‌ها به صورت انفجاری زیاد می‌شود:



و تعداد ویژگی‌ها حتی بیشتر هم می‌شود اگر تصویر رنگی باشد:



همانطور که خروجی نشان می‌دهد، حتی یک تصویر رنگی کوچک تقریباً 200000 ویژگی دارد، که می‌تواند هنگام آموزش مدل‌های خود مشکل ایجاد کند زیرا تعداد ویژگی‌ها ممکن است بسیار بیشتر از تعداد مشاهدات باشد. این مشکل استراتژی‌های ابعادی را که در فصل بعد مورد بحث قرار می‌گیرد پیش می‌آورد، که تلاش می‌کنند تعداد ویژگی‌ها را کاهش دهند در حالی که مقدار زیادی از اطلاعات موجود در داده‌ها را از دست ندهیم.

**8.14 استخراج هیستوگرام رنگ به عنوان ویژگی**

**مسئله**

شما می‌خواهید مجموعه‌ای از ویژگی‌ها را ایجاد کنید که رنگ‌های ظاهر شده در یک تصویر را نشان می‌دهد.

**راه حل**

هیستوگرام‌ها را برای هر کانال رنگی محاسبه کنید:



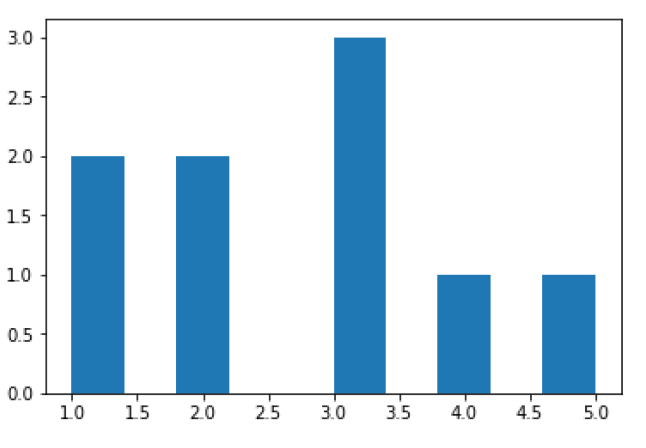
**بحث**

در مدل رنگی RGB، هر رنگ ترکیبی از سه کانال رنگ (یعنی قرمز، سبز، آبی) است. به نوبه خود، هر کانال می‌تواند یکی از 256 مقدار (که با یک عدد صحیح بین 0 تا 255 نمایش داده می‌شود) را دریافت کند. به عنوان مثال، پیکسل بالای سمت چپ در تصویر ما دارای مقادیر کانال زیر است:



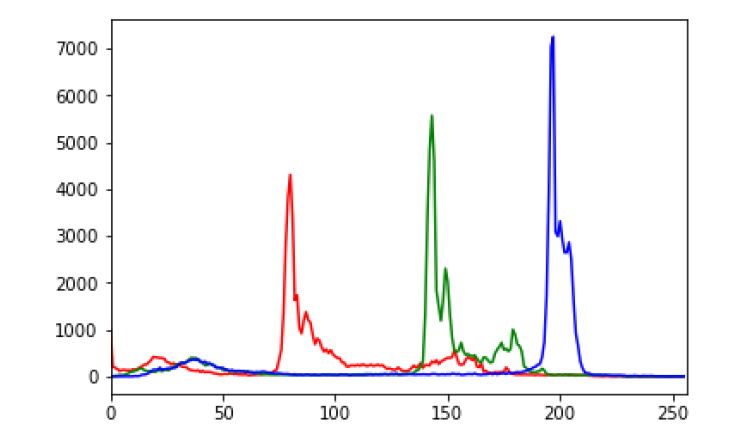
هیستوگرام نمایشی از توزیع مقادیر در داده‌ها است. در اینجا یک مثال ساده است:



****

در این مثال، ما تعدادی داده با دو 1، دو 2، سه 3، یکی 4 و یکی 5 داریم. در هیستوگرام، هر نوار نشان دهنده تعداد دفعاتی است که هر مقدار (1، 2 و غیره) در داده‌های ما ظاهر می‌شود. ما می‌توانیم همین تکنیک را برای هر یک از کانال‌های رنگی اعمال کنیم، اما به جای پنج مقدار ممکن، 256 (محدوده مقادیر ممکن برای یک مقدار کانال) داریم. محورx تعداد 256 مقدار کانال ممکن را نشان می‌دهد و محور y تعداد دفعاتی را نشان می‌دهد که یک مقدار کانال خاص در تمام پیکسل‌های یک تصویر ظاهر می‌شود:



****

همانطور که در هیستوگرام می‌بینیم، تقریباً هیچ پیکسلی حاوی مقادیر کانال آبی بین 0 تا 180 نیست، در حالی که بسیاری از پیکسل‌ها حاوی مقادیر کانال آبی بین 190 و 210 هستند. این توزیع مقادیر کانال برای هر سه کانال نشان داده شده است. با این حال، هیستوگرام صرفاً یک تجسم نیست. این دارای 256 ویژگی برای هر کانال رنگی است که مجموع 768 ویژگی را نشان می‌دهد که توزیع رنگ‌ها در یک تصویر را نشان می‌دهد.

**منابع بیشتر**

* [Histogram](https://en.wikipedia.org/wiki/Histogram)
* [pandas Histogram documentation](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.hist.html)
* [OpenCV Histogram tutorial](https://docs.opencv.org/3.2.0/d1/db7/tutorial_py_histogram_begins.html)

**8.15 استفاده از جاسازی‌های[[2]](#footnote-2) از پیش آموزش دیده به عنوان ویژگی‌ها**

**مسئله**

می‌خواهید جاسازی‌های پیش‌آموزش‌شده را از یک مدل موجود در PyTorch بارگیری کنید و از آنها به عنوان ورودی یکی از مدل‌های خود استفاده کنید.

**راه‌حل**

از torchvision.models برای انتخاب یک مدل و سپس بازیابی یک جاسازی از آن، برای یک تصویر معین، استفاده کنید:



**بحث**

در فضای ML، یادگیری انتقال اغلب به عنوان گرفتن اطلاعات آموخته شده از یک کار و استفاده از آن به عنوان ورودی برای کار دیگر تعریف می‌شود. به‌جای شروع از صفر، می‌توانیم از نمایش‌هایی[[3]](#footnote-3) استفاده کنیم که قبلاً از مدل‌های تصویری بزرگ از پیش آموزش دیده (مانند ResNet) آموخته‌ایم تا در مدل‌های یادگیری ماشین خودمان شروع به کار کنیم. به‌طور شهودی‌تر، می‌توانید درک کنید که چگونه می‌توانیم از وزن‌های یک مدل آموزش دیده برای تشخیص گربه‌ها به عنوان شروع خوبی برای مدلی که می‌خواهیم برای تشخیص سگ‌ها آموزش دهیم، استفاده کنیم. با به اشتراک گذاشتن اطلاعات از یک مدل به مدل دیگر، می‌توانیم اطلاعاتی را که از سایر مجموعه داده‌ها و معماری‌های مدل به دست می‌آیند، بدون هزینه‌های سربار آموزش یک مدل از ابتدا به کار ببریم.

کل کاربرد یادگیری انتقالی در بینایی کامپیوتر خارج از محدوده این کتاب است. با این حال، راه‌های مختلفی وجود دارد که بتوانیم نمایش‌های مبتنی بر جاسازی‌ها را از تصاویر خارج از PyTorch استخراج کنیم. در TensorFlow، یکی دیگر از کتابخانه‌های رایج برای یادگیری عمیق، می‌توانیم از tensorflow\_hub استفاده کنیم:



**همچنین ببینید:**

* [آموزش PyTorch: آموزش انتقال برای بینایی کامپیوتر](https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html#sphx-glr-beginner-transfer-learning-tutorial-py)
* [TensorFlow Hub](https://www.tensorflow.org/hub)

**8.16 تشخیص اشیا با OpenCV**

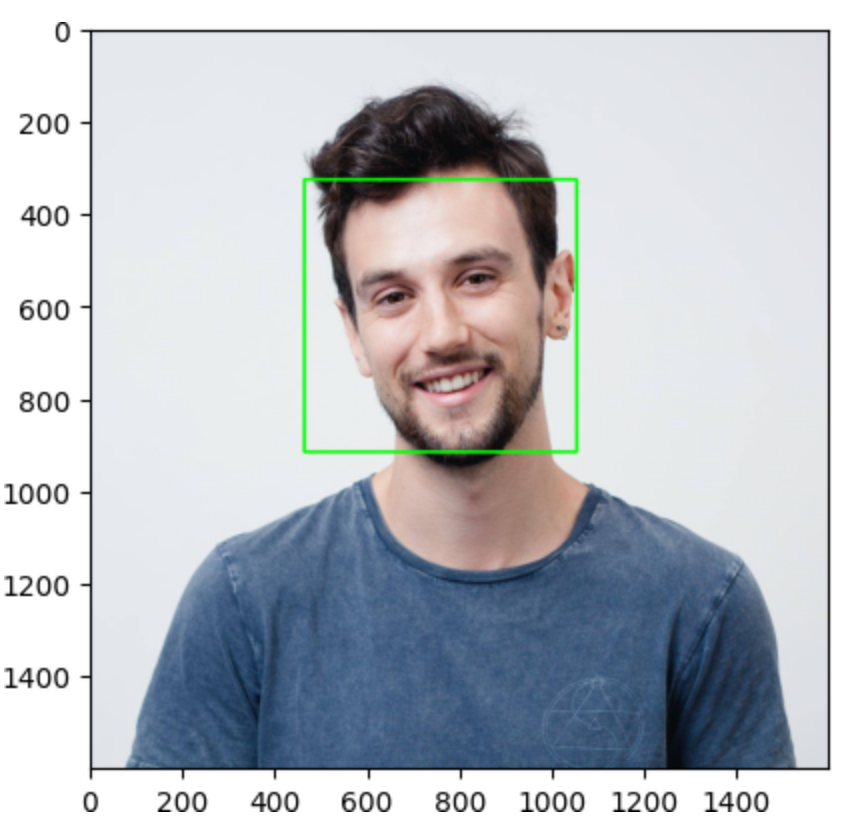
**مسئله**

شما می‌خواهید اشیاء را در تصاویر با استفاده از طبقه بندی کننده‌های آبشاریی از پیش آموزش دیده شده با OpenCV شناسایی کنید.

**راه‌حل**

یکی از طبقه‌بندی‌کننده‌های آبشاری [Haar OpenCV](https://github.com/opencv/opencv/tree/4.x/data/haarcascades) را دانلود و اجرا کنید. در این مورد، ما از یک مدل تشخیص چهره از پیش آموزش دیده شده برای تشخیص و ترسیم مستطیل دور یک چهره در یک تصویر استفاده می‌کنیم:



****

**بحث**

طبقه‌بندی‌کننده‌های آبشاری Haar مدل‌های یادگیری ماشینی هستند که برای یادگیری مجموعه‌ای از ویژگی‌های تصویر (به‌ویژه ویژگی‌های Haar) استفاده می‌شوند که می‌توانند برای شناسایی اشیاء در تصاویر استفاده شوند. خود ویژگی‌ها، ویژگی‌های مستطیلی ساده ای هستند که با محاسبه تفاوت در مجموع بین مناطق مستطیلی تعیین می‌شوند. پس از آن، یک الگوریتم تقویت گرادیان برای یادگیری مهم ترین ویژگی‌ها و در نهایت ایجاد یک مدل نسبتا قوی با استفاده از طبقه بندی کننده‌های آبشاریی اعمال می‌شود.

در حالی که جزئیات این فرآیند خارج از محدوده این کتاب است، قابل توجه است که این مدل‌های از پیش آموزش دیده شده را می‌توان به راحتی از مکان‌هایی مانند [OpenCV GitHub](https://github.com/opencv/opencv/tree/4.x/data/haarcascades) به عنوان فایل XML دانلود کرد و بدون آموزش مدلی، روی تصاویر اعمال کرد. این در مواردی مفید است که می‌خواهید ویژگی‌های ساده تصویر باینری مانند contain\_face (یا هر شیء دیگری) را به داده‌های خود اضافه کنید.

**همچنین ببینید:**

* [آموزش OpenCV: طبقه بندی آبشاری](https://docs.opencv.org/4.x/db/d28/tutorial_cascade_classifier.html)

**8.17 طبقه بندی تصاویر با Pytorch**

**مسئله**

شما می‌خواهید تصاویر را با استفاده از مدل‌های آموزش عمیق از پیش آموزش دیده شده در Pytorch طبقه بندی کنید.

**راه‌حل**

از torchvision.models برای انتخاب یک مدل طبقه بندی تصویر از پیش آموزش دیده شده و تغذیه تصویر از طریق آن استفاده کنید:



**بحث**

بسیاری از مدل‌های آموزش عمیق از پیش آموزش دیده شده، برای طبقه بندی تصاویر به راحتی از طریق PyTorch و TensorFlow در دسترس هستند. در این مثال، ما از ResNet18، یک معماری شبکه عصبی عمیق استفاده کردیم که بر روی مجموعه داده ImageNet با عمق 18 لایه آموزش داده شده است. مدل‌های عمیق‌تر ResNet، مانند ResNet101 و ResNet152 نیز در Pytorch در دسترس هستند - و فراتر از آن، مدل‌های تصویری زیادی برای انتخاب وجود دارند. مدل‌های از پیش آموزش‌دیده شده، بر روی مجموعه داده ImageNet می‌توانند احتمالات پیش‌بینی‌شده را برای همه کلاس‌های تعریف‌شده در متغیر imagenet\_class\_index در قطعه کد قبلی، که ما از GitHub دانلود کرده‌ایم، به صورت خروجی نمایش دهند.

به طور مثال تشخیص چهره در OpenCV (نگاه کنید به دستور العمل 8.16)، که ما می‌توانیم از کلاس‌های تصویر پیش‌بینی‌شده به‌عنوان ویژگی‌های پایین‌رونده[[4]](#footnote-4) برای مدل‌های آینده ML یا تگ‌های فراداده مفیدی که اطلاعات بیشتری را به تصاویر ما اضافه می‌کنند، استفاده کنیم.

**همچنین ببینید:**

* [مستندات PyTorch: مدل‌ها و وزن‌های از پیش آموزش دیده](https://pytorch.org/vision/stable/models.html)

1. Open Source Computer Vision Library [↑](#footnote-ref-1)
2. - Embeddings [↑](#footnote-ref-2)
3. - representations [↑](#footnote-ref-3)
4. - downstream [↑](#footnote-ref-4)