

INTRODUCTION TO COMPUTER VISION

PROJECT DOCUMENTATION



Ferdowsi University of Mashhad
Department of Computer Engineering

SPRING 2025

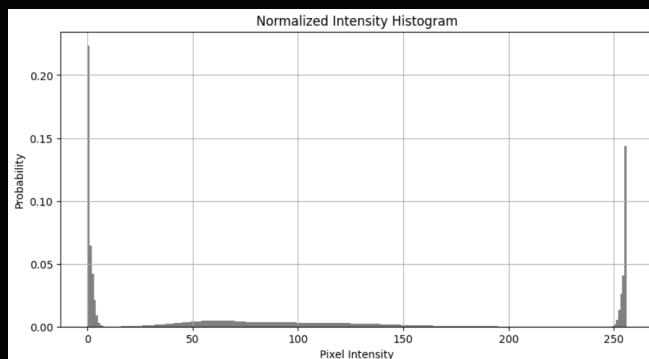
نام و نام خانوادگی	شماره دانشجویی
امیرحسین افشار	۴۰۱۲۲۶۲۱۹۶

رپوهای گیتها بی که پیاده سازی پروژه را انجام دادم:

- <https://github.com/AmirHossienAfshar/classic-vision>
- <https://github.com/AmirHossienAfshar/cv-noise-denoise>

(۱) سوال اول: معمای داوینچی

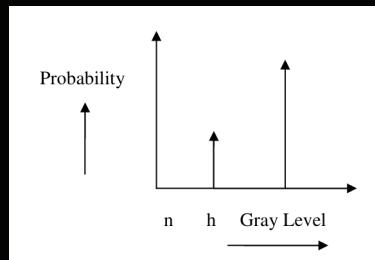
در ابتدا برای حل پازل، عملیات denoising *processed_img_part_1.jpg* را برای تصویر *denoising* انجام شد. بدین منظور، از هیستوگرام تصویر بهره گرفته شده است [۱]؛ هیستوگرام تصویر با سطوح خاکستری، بر اساس نرمال کردن میزان پیکسل های با شدت بین ۰ تا ۲۵۵ ساخته می شود تا شکل function *probability distribution* بسته باشد. پلاس زیر، بیانگر هیستوگرام برای تصویر نویزی شماره ۱ است:



شکل ۱: پلاس هیستوگرام تصویر نویزی شماره ۱

این شکل، بیان می کند که تعداد پیکسل ها با مقدار نزدیک به صفر و مقدار نزدیک به ۲۵۵ بسیار زیاد است. همچنین هیستوگرام احتمال قابل انتظار برای تصاویر نویزی با نوع نویز نمک فلفل به شکل زیر است:

[۱] Asoke Nath: Image Denoising Algorithms: A Comparative Study of Different Filtration Approaches Used in Image Restoration



شکل ۲: هیستوگرام قابل انتظار برای تصاویر نویزی نمک و فلفل

Category	Filter Name	Function
Smoothing Filters	Box Filter	denoise_box_filter
	Gaussian Filter	denoise_gaussian_filter
Statistical Filters	Median Filter	denoise_median_filter
	Max Filter	denoise_max_filter
	Min Filter	denoise_min_filter
Advanced Filters	Bilateral Filter	denoise_bilateral_filter
	Non-Local Means	denoise_nl_means
	Wavelet Filter	denoise_wavelet_filter
	Total Variation Filter	denoise_total_variation

جدول ۱: انواع فیلتر ها و توابع denoising به کار گرفته شده

با مقایسه شکل ۱ و ۲ می توان نتیجه گرفت که نویز تصویر شماره ۱ از نوع نمک و فلفل می باشد. بنابراین، برای denoise کردن تصویر، بهترین گزینه، استفاده از فیلتر median می باشد. شایان ذکر است که در عملیات denoise کردن تصویر برای اطمینان بیشتر از نوع نویز، از انواع فیلترها استفاده شد که به شرح زیر هستند:

همانطور که از شکل ۱ انتظار می رفت، بهترین عملکرد خروجی با استفاده از فیلتر میانه یا همان median بدست می آید. در نهایت، برای نهایی کردن بهترین خروجی با استفاده از فیلتر میانه، دو روش پیگیری شد:

۱. استفاده از چند مرحله فیلتر median با ابعاد یکسان

۲. استفاده از یک فیلتر median با کرنل بزرگتر

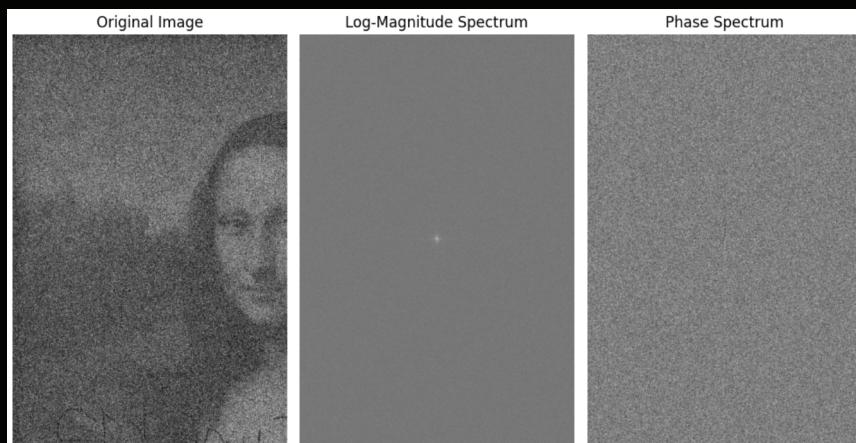
روش اول، به طور کلی برای حفظ جزئیات تصویر مناسب تر است؛ زیرا هرگونه کرنل بزرگ با ریسک از دست دادن جزئیات همراه است. از طرفی، ممکن است که همگرا شدن تصویر پس از اعمال چند بار فیلتر میانه، به کندی پیش روید و حتی برخی نویز ها در تصویر باقی بمانند که این مشکل، در روش دوم وجود ندارد.

در نهایت، با توجه به مسئله که تنها نیاز است یک متن از تصویر استخراج شود (و حفظ سایر جزئیات دارای اهمیت کمتری است)، از روش دوم استفاده شد.



شکل ۳: شکل نهایی تصویر denoise شده

همچنین شایان ذکر است که برای درک بهتر نویز، از حوزه‌ی فرکانس نیز کمک گرفته شد^[1] که بتوان نوع نویز را حدس زد و در نهایت با استفاده از فیلترهای notch و band-pass یا band-reject از نویز تصویر کاست. شکل سوم بیانگر این موضوع است.



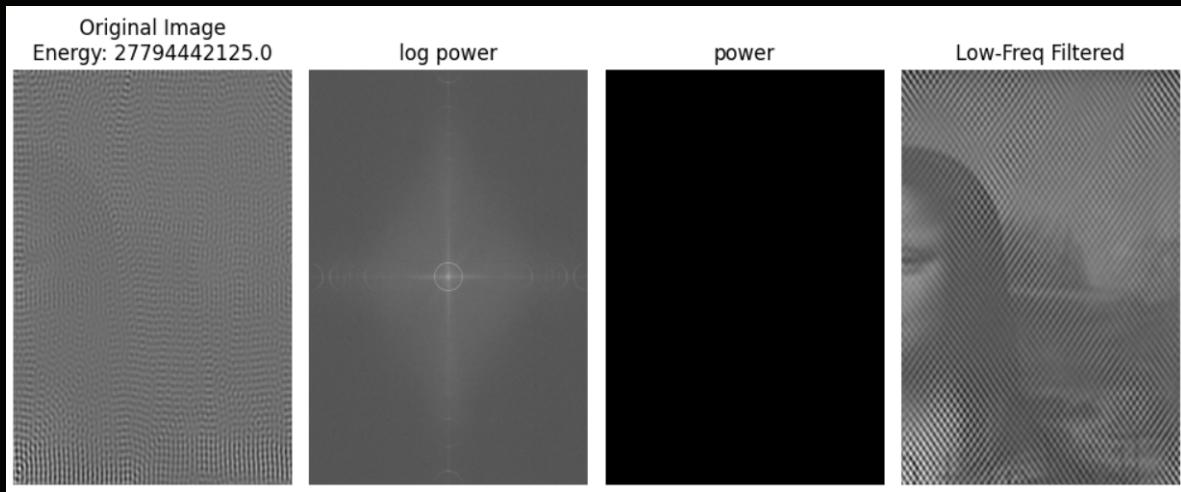
شکل ۴: تصویر شماره ۱ در حوزه فرکانس

با توجه به magnitude تصویر در حوزه فرکانس، نمی‌توان نویز خاصی را قائل شد که نهایتاً، همان فیلتر میانه که از هیستوگرام تصویر درک شده بود، استفاده شد.

^[1] Digital Image Processing By Gonzalez 2nd Edition 2002, chapter 4.2

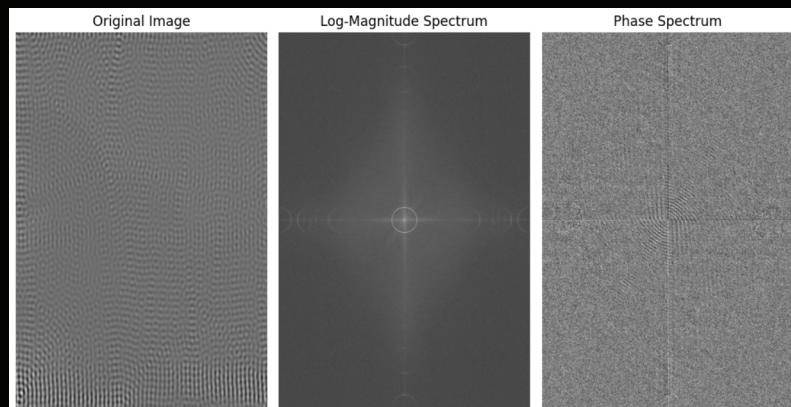
۰ بخش دوم

برای این بخش باید تصویر *processed_img_part_2.jpg* اصلاح می شد. بدین منظور و در ابتدا برای تشخیص نویز، از انرژی و *magnitude* تصویر استفاده شد؛ زیرا تصویر کاملا پوشیده از نویز بود و پیش بینی نوع نویز را سخت می کرد. همچنین برای درک بهتر کلیت تصویر، با حذف فرکانس های بالا، یک تخمین از کلیت تصویر به دست آمده است. برای این منظور، پلات زیر رسم شده است:



شکل ۵: پیش بینی نوع نویز در تصویر دوم

با دقت کردن به لگاریتم *power* در پلات (شکل شماره ۵) متوجه یک حلقه می شویم که با الگوی سینوسی نویز در تصویر در حوزه مکان تطابق دارد. برای درک بهتر این نویز، به طور مستقل برای تصویر در حوزه فرکانس *magnitude* و *phase* رسم شده است که به شکل زیر است:

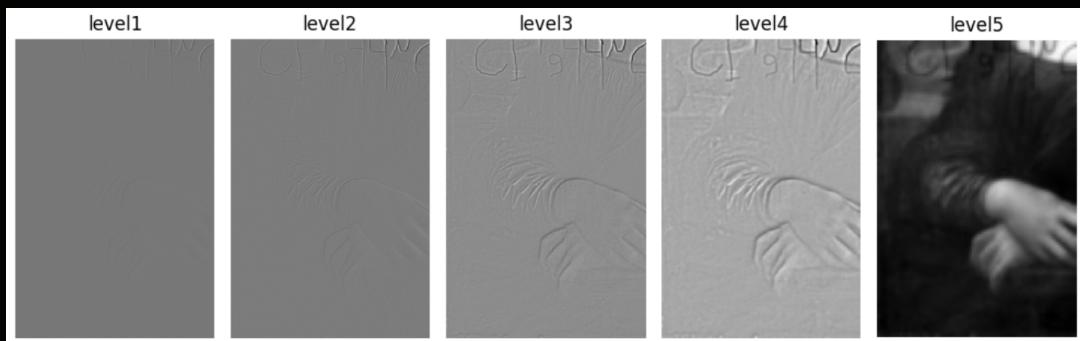


شکل ۶: تصویر دوم در حوزه فرکانس

که ایده اولیه را تایید می کند. بدین منظور، تنها و به سادگی با ساختن یک فیلتر *band-reject* در حوزه فرکانس، این مشکل حل شده و تصویر نهایی ساخته می شود.

۰ بخش سوم

برای بخش سوم از پازل، باید تصویر اصلی را از ۵ زیر تصویر reconstruct می کردیم. بدین منظور، تصاویر در ابتدا و در کنار یکدیگر plot شده اند:



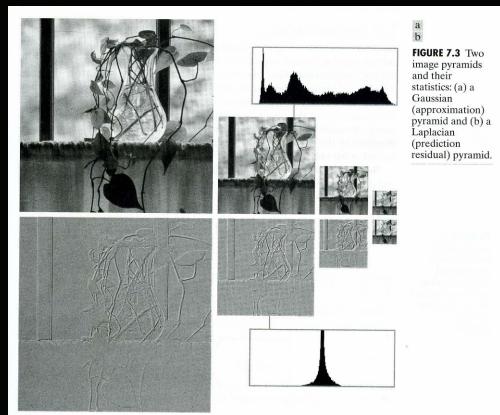
شکل ۷: زیرتصاویر بخش سوم پازل

همچنین ابعاد هر کدام از تصاویر در جدول زیر بیان شده است:

Image File	Width (px)	Height (px)
processed_img_part_3_Level_0.jpg	1796	1201
processed_img_part_3_Level_1.jpg	898	601
processed_img_part_3_Level_2.jpg	449	301
processed_img_part_3_Level_3.jpg	225	151
processed_img_part_3_Level_4.jpg	113	76

جدول ۲: ابعاد زیر تصاویر در بخش سوم پازل

با توجه به جدول شماره ۲ که بیانگر ابعاد پازل است، میتوان متوجه شد که در هر مرحله با تقریب هم طول و هم عرض تصویر نصف می شود. این موضوع، وجود نوعی pyramid را در ذهن تداعی می کند^[۱]. این موضوع، در زیربخش سوم فصل هفتم کتاب گونزالس^[۲] به این شکل آمده است:

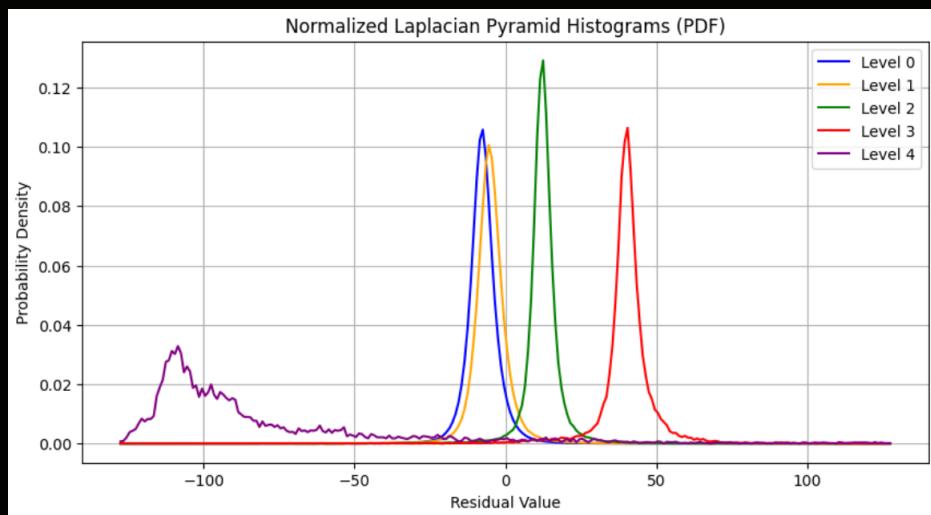


شکل ۸: انواع pyramid ها: کتاب گونزالس، زیربخش سوم فصل هفتم

[۱] medium: A Beginners Guide to Computer Vision (Part 4)- Pyramid

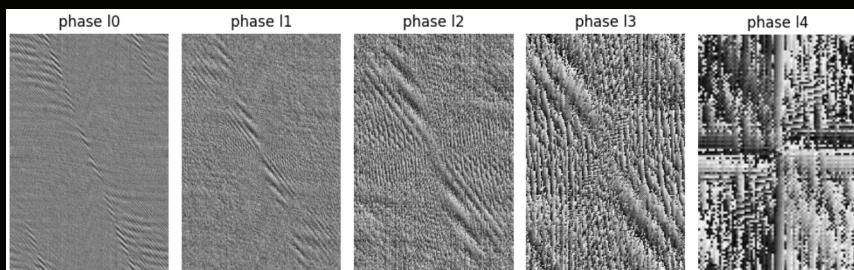
[۲] Digital Image Processing By Gonzalez 2nd Edition 2002, chapter 7.3

برای درک بهتر هرم، هیستوگرام احتمال برای هر کدام از زیر تصاویر رسم شده است:



شکل ۹: هیستوگرام در هرم

نمودار هیستوگرام نرمال شده هرم، توزیع آماری مقدارهای باقیمانده در هر سطح از هرم را نمایش می‌دهد. در فرآیند ساخت هرم، هر تصویر در یک سطح با نسخه‌ی بزرگ نمایی شده‌ی تصویر سطح بعدی کم می‌شود، و نتیجه این اختلاف، تصویر باقیمانده‌ای است که تنها حاوی اطلاعات لبه‌ها، جزئیات کوچک است. برای نمایش بهتر این اطلاعات، از یک آفست استفاده شده و همچنین نرمال سازی انجام شده است. در تایید این موضوع، از تبدیل به حوزه فوریه و نمایش فاز هر کدام از زیر تصاویر نیز استفاده شده است:



شکل ۱۰: هیستوگرام در هرم

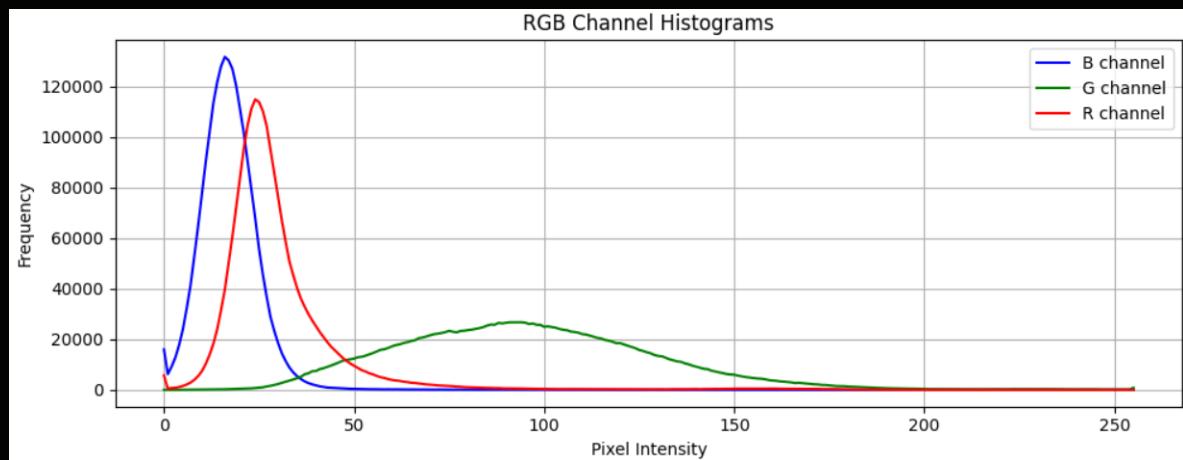
همانطور که مشخص است، لبه‌های کوچک در سطوح بالا، دارای فاز یکنواخت تری هستند و هرچه به لایه‌های پایین‌تر می‌رویم، این یکنواختی کمتر می‌شود و تغییرات فرکانس وضوح می‌یابند. با دقیقت به این هیستوگرام، (و همچنین زیرتصاویر ارائه شده در شکل پنجم) میتوان نتیجه گرفت که هرم از نوع لاپلاسی است؛ هر مرحله از *downsample* شدن تصویر و محاسبه اختلاف با لایه قبلی محاسبه می‌شود که برای ساختن تصویر اصلی، کافیست معکوس این کار را انجام دهیم.

برای درست کردن تصویر، در ابتدا تلاش شد که با طی کردن فرایند به شکل معکوس، تصویر اصلی ساخته شود، اما تصویر خروجی شامل کیفیت مناسب نبود. بدین منظور، از تابع `resize` از کتابخانه `cv2` استفاده شد که به صورت خطی تصویر را تغییر سایز می‌دهد. همچنین، برای بهبود نهایی، مقادیر `grayscale` به `uint8` تغییر یافته اند؛ به عبارتی، مقادیر قبلی سطوح خاکستری با این کار، به بازه $(0, 255)$ گسترش یافتند که باعث بهبود کنترast شد. تصویر نهایی ساخته شده نیز در شکل ۹ آمده شده است:



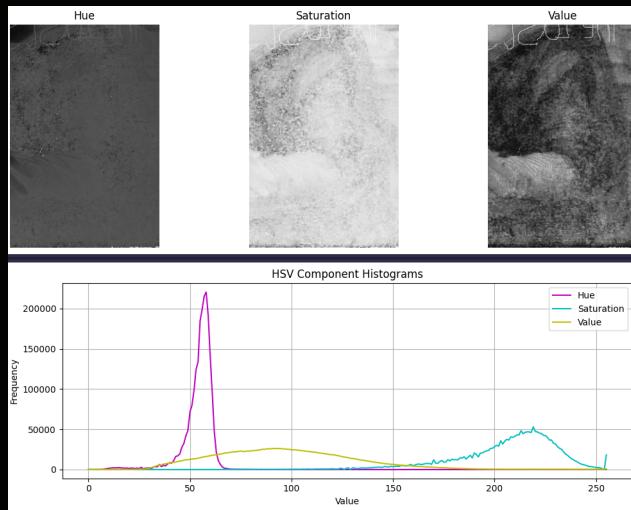
شکل ۱۱: تصویر نهایی ساخته شده از زیرتصاویر بخش سوم

در نهایت نیز به زیر تصویر چهارم میرسیم که به ظاهر رنگش به هم ریخته است. در ابتدا و در اولین گام، هیستوگرام رنگی تصویر تهییه شد:



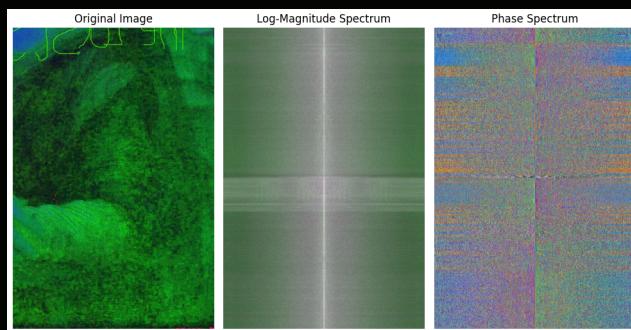
شکل ۱۲: هیستوگرام تصویر برای سه کanal رنگی

که به شکل واضحی تفاوت میان رنگ سبز و دو رنگ دیگر را نشان می‌دهد. همچنین برای درک بهتر تصویر، از حوزه `HSV` نیز استفاده شد:



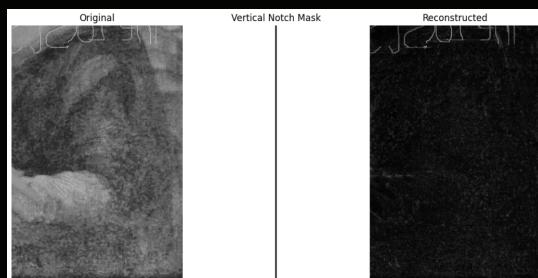
شکل ۱۳: HSV آنالیز

که البته بیانگر بایاس خاصی نمی باشد و از این حوزه نمی توان اطلاعات خاصی استخراج کرد. در نهایت، بر روی حوزه فرکانس نیز یک آنالیز کلی انجام دادم:



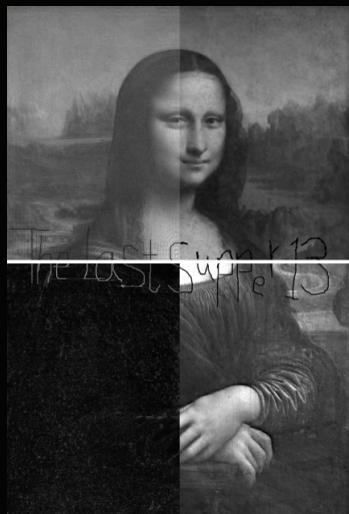
شکل ۱۴: حوزه فرکانس

که خط عمودی در بزرگی فرکانس ها توجهم را جلب کرد. بدین منظور آن را حذف کدم و به چنین خروجی ای رسیدم:



شکل ۱۵: اصلاح تصویر در حوزه فرکانس

که بخش متن را به شکل خیلی خوبی از زمینه جدا کرده و به همین خروجی بسنده کدم. در نهایت تصویر خروجی من به دست آمد که آن را قرینه کرم تا به متن *the last supper* ۱۳ رسیدم که ظاهرا به نقاشی مسیح داوینچی اشاره دارد و شخص ۱۳ ام نیز خود مسیح است!



شکل ۱۶: تصویر نهایی

(۲) پروژه دوم: تصاویرنویزی/غیرنویزی

در این پروژه، هر دو روش یادگیری عمیق و بینایی ماشین کلاسیک پیاده سازی شده است. در ابتدا به تفصیل، روش یادگیری عمیق توضیح داده می شود:

یادگیری عمیق در پروژه دوم

در ابتدا، برای پیاده سازی سیستم تشخیص نویز ها، این نیاز دیده میشد که دیتاست های در اختیار قرار گرفته، دیتاست های کوچکی هستند و برای روش های ml و یا dl مناسب نیستند. به گونه ای که حتی با augment کردن داده ها و پیاده سازی بر روی مدل های pre-trained tune fine کردن آنها نیز، کفایت نمیکرد. بنابراین، با بررسی دیتاست های قرار داده شده کشاورزی که از نوع برگ درختان هستند، دیتاست های کاندیدا از وبسایت kaggle به شرح زیر هستند:

1. <https://www.kaggle.com/datasets/ichhadhari/leaf-images>
2. <https://www.kaggle.com/datasets/mdwaquarazam/agricultural-crops-image-classification/data>
3. <https://www.kaggle.com/datasets/alexo98/leaf-detection>

که طبق بررسی انجام شده، بهترین و مشابه ترین دیتاست، مربوط به دیتاست leaf-detection است. دقت شود که این موضوع که دیتاست ها باید برای استفاده این پروژه به شکل کاملی شخصی سازی شوند (به عنوان مثال، نویز به تصاویر اضافه شود) کاملاً واضح است و از ابتدا صرفاً به دنبال دیتاست هایی صرفاً با زمینه مرتبط بودم و سپس دیتاست ها را به شکل مناسبی برای استفاده از این پروژه شخصی سازی کردم. مجدداً تاکید میشود که دیتاست انتخاب شده، صرفاً انواع برگ های گوناگون بوده است و هیچ هدفی از نویز/دینویز کردن تصاویر در این دیتاست دنبال نشده است.

بررسی دیتاست آورده شده:

با توجه به این که این دیتاست، صرفاً شامل تصاویری از برگ های ۱۰ نوع درخت هندی هستند، بدون توجه به لیبل های آن برگ ها، و صرفاً با در نظر گرفتن همه آنها از یک نوع، (همگی را به شکل

یکسان و «برگ» در نظر گرفتم) به آنها انواع گوناگونی از نویز ها پیاده سازی کردم. در ابتدا، نویزهای تکی، یعنی :

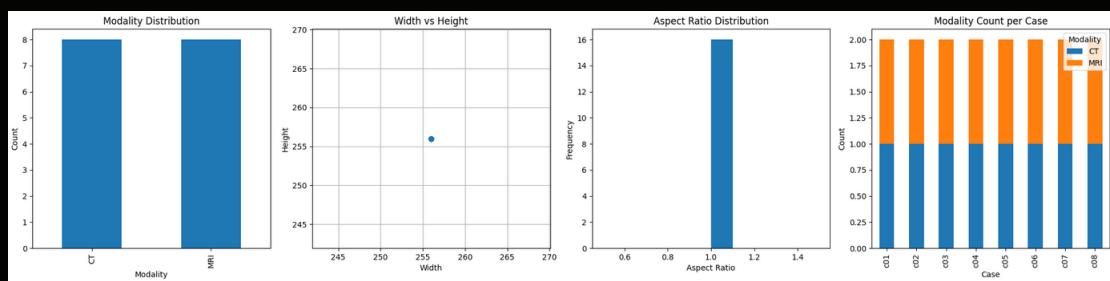
سپس انواع نویز های دو تایی (تمامی جایگشت ها لحاظ نشده است؛ به علت محدودیت پردازشی گوگل کولب به ۱۶ گیگ حافظه رم، بیش از این نوع نمیتوان جایگشت های مختلفی از نویز در نظر گرفت):

No.	Combinations	Noise Type
1	1	Gaussian
2	1	Salt & Pepper
3	1	Poisson
4	1	Speckle
5	1	Uniform
6	2	Gaussian + Salt & Pepper
7	2	Gaussian + Poisson
8	2	Gaussian + Speckle
9	2	Gaussian + Uniform
10	2	Salt & Pepper + Speckle
11	2	Salt & Pepper + Uniform
12	2	Poisson + Speckle
13	2	Poisson + Uniform
14	2	Speckle + Uniform
15	3	Gaussian + Salt & Pepper + Speckle
16	3	Gaussian + Poisson + Uniform
17	3	Salt & Pepper + Speckle + Uniform

جدول ۳: لیست انواع نویز های پیاده شده

(۳) پروژه سوم: ادغام تصاویر پزشکی

در این پروژه، ابتدا پیاده‌سازی با استفاده از روش‌های کلاسیک انجام شده و جزئیات مربوط به آن به‌طور کامل توضیح داده شده است. گام مشترک میان هر دو روش، پروفایل دیتا است. در ابتدا باید بر روی دیتاست عملیات پروفایل کردن دیتا انجام می‌شود که بررسی شود دیتاست تصاویر به شکل ct (که به شکل هستند که استخوان‌ها را به رنگ سفید نمایش می‌دهند) با تصاویر mri از نظر سایز، فرمت، طول و عرض تصویر و نسبت طول به عرض با یکدیگر تطابق دارند یا خیر. بدین منظور، نتیجه این پروفایل کردن در زیر قابل مشاهده است:

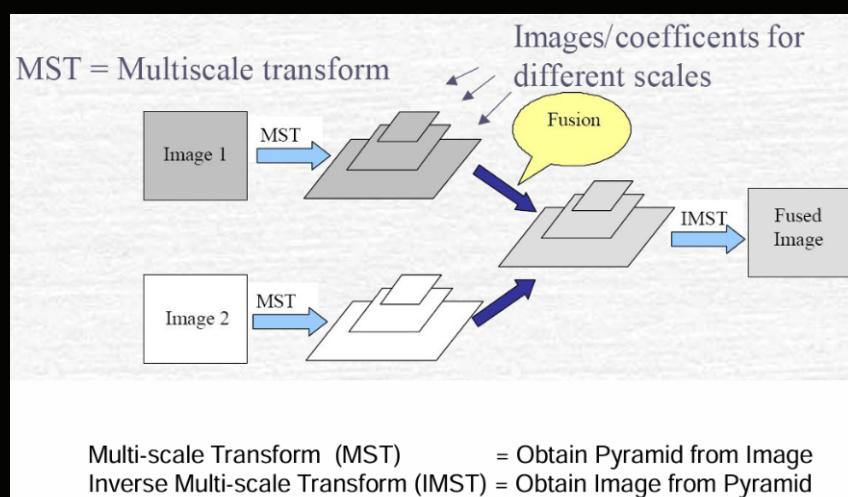


شکل ۱۷: نتیجه پروفایل دیتا

(شایان ذکر است که شاید گمان شود که برای این دیتاست کوچک، نیازی به پروفایل کردن دقیق دیتا نبوده و صرفا یک مشاهده کوچک بر روی ۸ زوج عکس، کفايت می‌کند، اما روش ارائه شده برای هر تعداد جفت عکس ارائه شده قابل اجراست و مرحله ابتدایی الزامی هر پروژه‌ای محسوب می‌شود.) همچنین شایان ذکر است که جفت تصاویری که داده شده است، از نظر مکانی با یکدیگر align هستند و بنابراین، نیازی به پیاده‌سازی matching با استفاده از روش‌های surf نمی‌باشد.

راه حل با روش‌های بینایی کلاسیک

همانطور که در اسلاید‌های درسی داشتیم:



شکل ۱۸: نحوه ادغام دو نوع تصویر مختلف

باید pyramid هایی از هر دو نوع تصویر تهیه کنیم. در ابتدا ویژگی هایی که هر سطح از این ها شامل می شوند، به شکل زیر بیان می شود: این قانون در رابطه با هر دو نوع هرم گوسی

ویژگی های استخراج شده	فرکانس	اندازه تصویر	سطح
جزئیات ریز (لبه ها، بافت، نویز)	فرکانس بالا	اندازه کامل	سطح ۰
لبه های بزرگتر، اشکال	فرکانس متوسط	نصف اندازه	سطح ۱
نواحی صاف، روشنایی کلی	فرکانس پایین	یکچهارم اندازه	سطح ۲
ساختار کلی، سایه روش نرم	فرکانس بسیار پایین	بسیار کوچک	سطح N (بالا)

جدول ۴: ساختار سطوح در هرم تصویری

و لایپلاسی صدق می کند. (البته این دو هرم در نوع نمایششان با یکدیگر متفاوت هستند). در نهایت، مورد دیگری که باید بحث شود نحوه ادغام است که به این شکل چندین روش می توان به کار برد:

ویژگی های حفظ شده	عملکرد	Rule Fusion
اطلاعات متعادل	ترکیب برابر دو تصویر	(mean) Average
ویژگی های تیره	انتخاب مقدار کمینه هر پیکسل	Min
ویژگی های روشن	انتخاب مقدار بیشینه هر پیکسل	Max
لبه های قوی با فرکانس بالا	انتخاب پیکسلی با بیشترین قدر مطلق	Max-Abs
بافت، واریانس محلی	انتخاب پیکسل از تصویری با انرژی محلی بالاتر	Energy-Max
ساختارهای برجسته	انتخاب پیکسل با لبه بیشتر	Saliency-Based
تأثیر قابل تنظیم	میانگین وزنی با وزن های دستی	(static) Weighted
ترکیب تطبیقی	وزن دهنی بر اساس انرژی	(adaptive) Weighted
ساختارهای همبسته	تقسیم تصویر به مؤلفه های اصلی	PCA-Based

جدول ۵: قوانین مختلف ادغام تصویر و ویژگی های حفظ شده

پارامتر بعدی که باید بررسی شود، عمق این pyramid ها می باشد. به طور کلی، با توجه به اینجا:

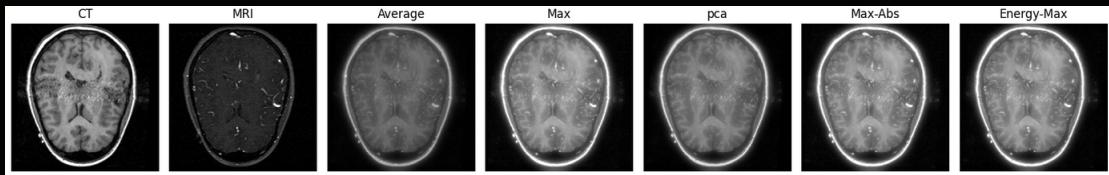
- The Laplacian Pyramid as a Compact Image Code

اشاره شده است که برای تهیه هر هرمی، میتوان از الگوی زیر تبعیت کرد:

$$\text{max_levels} = \lfloor \log_2 (\min(H, W)) \rfloor$$

که با توجه به تصویر ورودی $256 * 256$ می توان هرم را به حداقل ۸ سطح تقسیم کرد. اکنون روش های کلاسیک را نیز با دو شیوه پیاده سازی کرده ام: ابتدا با استفاده از هرم لایپلاسین، و سپس با استفاده از هرم ویولت.

در روش *fuse* کردن با استفاده از هرم لایپلاسی، در ابتدا، هرم گوسی را بدست آورده ام و سپس با استفاده آن، هرم لایپلاسین را ساختم. روش کار نیز در ابتدا نصف کردن ابعاد تصویر است (هرم گوسی) و سپس در ابعاد کوچکترین تصویر را به اندازه لایه قبلی *upsample* میکنیم و اختلاف میگیریم و به همین ترتیب تا بالاترین لایه جلو میرویم و تصاویر اختلاف گرفته شده، همان هرم لایپلاسی هستند.



شکل ۱۹: ادغام به کمک هرم لایپلاسی و با عمق ۳ (تصاویر با کفیت بالاتر در فایل جوپیتر نوتبوک قابل مشاهده هستند).

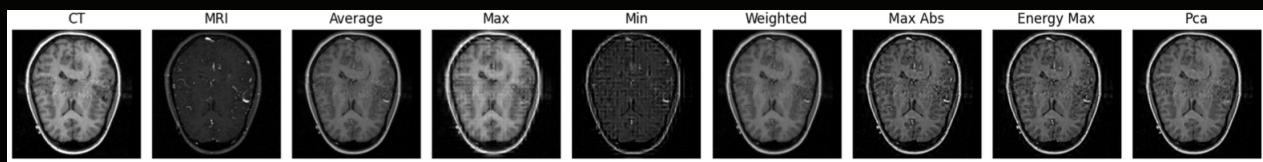
سپس، با توجه به جدولی که درباره نحوه fuse کردن اطلاعات ارائه شد، انواع روش‌ها را پیاده کردم و در کنار یکدیگر قرار دادم تا بتوان مقایسه‌ای انجام داد. خروجی برای تصویر نمونه ورودی به شکل زیر است:

همانطور که مشخص است، به شکل کلی در انواع روش‌های fuse، بافت نرم مغز متمایل به رنگ سفید می‌شود (که مربوط به استخوان هاست) و جزئیات آن نیز کم می‌شود. همچنین میان روش‌های fuse کردن، تفاوت معناداری دیده نمی‌شود.

سپس این عملیات را به ازای تعداد لایه‌ها، tune کردم. نتیجه این tuning آن بود که هرچقدر تعداد لایه‌ها را بیشتر در نظر می‌گرفتم، تفاوت معناداری در افزایش جزئیات بافت نرم مغز دیده نمی‌شد اما هاله‌ای سفید رنگ در کنار بخش‌های استخوانی قرار می‌گرفت که نشان دهنده blur بیشتر به ازای تعداد لایه بیشتر است.

در نهایت، میتوان گفت که fuse کردم این نوع تصاویر و با استفاده از هرم لایپلاسین چندان خروجی جالبی نداشت که انگیزه‌ای شد برای پیاده سازی با استفاده از هرم ویولت. هرم ویولت به عنوان پایه عملیات fuse ساخته شد.

همچنین، تابع fuse کننده تصاویر، با همان روش‌هایی که قبلاً اشاره شد (مانند ماکسیمم، میانگین، میانگین وزن دار و غیره) پیاده سازی شد. خروجی این نوع هرم در تصویر زیر قابل مشاهده است:



شکل ۲۰: ادغام به کمک هرم ویولت (تصاویر با کفیت بالاتر در فایل جوپیتر نوتبوک قابل مشاهده هستند).

همانطور که دیده می‌شود، هرم ویولت به شکل بسیار مشهودی در عملیات fuse کردن بهتر از هرم لایپلاسی عمل کرده است. در نهایت نیز، در این روش، پارامترهایی که بر روی آنها tuning را انجام دادم، تعداد لایه‌های هرم و نوع هرم ویولت بود. با توجه به تعداد زیاد آنها، یک سرج رندوم میان ۵ کانفیگ مختلف انجام دادم و خروجی را مقایسه کردم.

نتیجه خروجی آن tuning این شد که نوع هرم ویولت تاثیر چشمگیری در خروجی ندارد، اما مشابه با چیزی که در tune کردن هرم لایپلاسی داشتیم، در اینجا نیز با اضافه شدن تعداد لایه‌ها یک هاله روشن میان بخش‌های روشن تر دیده می‌شود که میتوان با blur شدن تصویر آن را توجیه کرد.

راه حل با روش های یادگیری عمیق

به عنوان مطالعه‌ی تکمیلی، روش‌های مبتنی بر یادگیری عمیق را نیز بررسی کردم. به طور خاص مقاله‌ای که مطالعه کردم در اینجا قرار دارد:

- DM-FNet: Unified multimodal medical image fusion via diffusion process-trained encoder-decoder

روش‌های کلاسیکی که پیاده سازی شده است، صرفا شامل ساختارهای هرم بودند و در نهایت در پیشرفت‌ههای ترین مدل‌های خود، از ویولت استفاده می‌کردند که مشکل بزرگی داشتند: یا تصویر نهایی به شکل زیادی smooth می‌شود و یا جزئیات هر دو منبع ورودی را نمیتوان نگه داشت. این دو مورد انگیزه‌هایی بودند که برای حل این مسئله شبکه‌های عمیق سوق داده شده‌اند.

مرحله‌ی اول بر پایه‌ی DDPM طراحی شده است؛ مدلی که هدف آن، یادگیری معکوس‌سازی یک process noise forward به صورت مرحله‌به‌مرحله است. برخلاف کاربردهای معمول که در آن‌ها تصویر feature map به عنوان ابزاری برای استخراج های با کیفیت بالا و در مقیاس‌های مختلف از هر نوع تصویر MRI و CT به کار گرفته شده است.

در این چارچوب، برای هر مدل‌الیته یک model diffusion مبتنی بر UNet به صورت جداگانه آموزش داده شده است. هدف این مرحله بازسازی تصویر نیست، بلکه embedding کردن اطلاعات هر مدل‌الیته در یک فضای بازنمایی عمیق و سلسله‌مراتبی است که بتواند ساختارهای مهم تصویر را به خوبی نمایش دهد.

ویژگی‌هایی که در این مرحله استخراج می‌شوند، نقش نوعی سوپروایزر را برای نوع تصویر خود ایفا می‌کنند؛ به این معنا که مدل MRI در شناسایی ویژگی‌های مرتبه با tissue soft تخصص دارد، در حالی که مدل CT تمرکز بیشتری روی ساختارهای استخوانی دارد.

در مجموع، این مرحله مانند یک preprocessor هوشمند عمل می‌کند که به جای استفاده از feature extractor های سطحی و سنتی، از هایی encoder بهره می‌گیرد که با فرایند diffusion آموزش دیده‌اند و می‌توانند فیچر‌هایی دقیق ارائه دهند.

پس از استخراج feature deep ها از هر دو مدل‌الیته، مرحله‌ی دوم با نام

DFFM (Diffusion Feature Fusion Model)

وظیفه‌ی ادغام واقعی اطلاعات را بر عهده می‌گیرد. برخلاف روش‌های ساده‌ای مانند میانگین‌گیری یا روی هم قرار دادن مستقیم ویژگی‌ها، در این‌جا فرایند fusion با استفاده از دو مازول مجزا و هدفمند انجام شده است.

بخشی از کدهایی که از این مقاله که شامل معماری مدل اصلی بود در فایل پایتونی ام قرار دادم. قصد داشتم از این مدل یک inference انجام دهم تا با روش‌های کلاسیک مقایسه کنم؛ اما در نهایت تنها به بیان مطالعات، درک و برداشت خود از این نوع شبکه‌های عمیق بسنده کردم؛ زیرا یتاست‌های غنی و همچنین مدل‌ها و مقالات متعددی وجود دارند که بر روی همین زمینه فعالیت کرده‌اند و منبع خوبی برای یادگیری هستند.

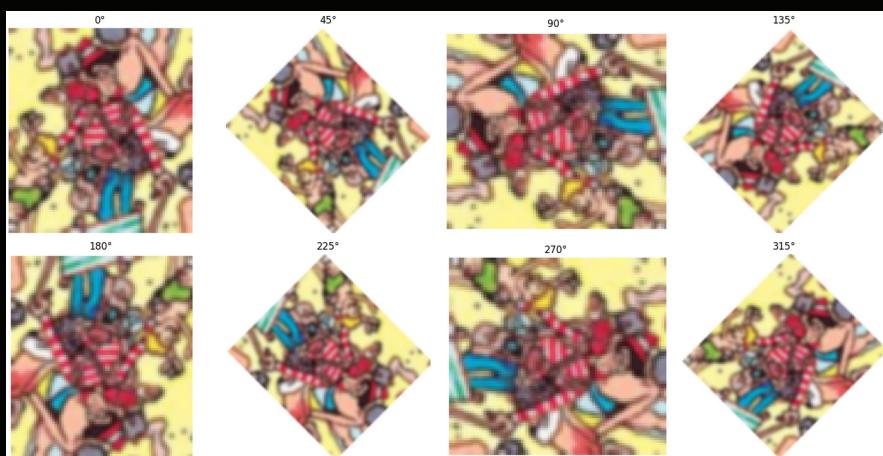
(۴) پروژه چهارم: والدو

برای این پروژه، سه شیوه اصلی پیاده شد:

۱. با استفاده از cross-correlation
۲. استفاده از هرم و پیاده سازی multi-resolution-correlation
۳. تهیه دیتاست و پیاده سازی با شبکه عصبی عمیق به تفصیل هر کدام از آنها در ادامه توضیح داده می شود:

پیاده سازی با کورولیشن

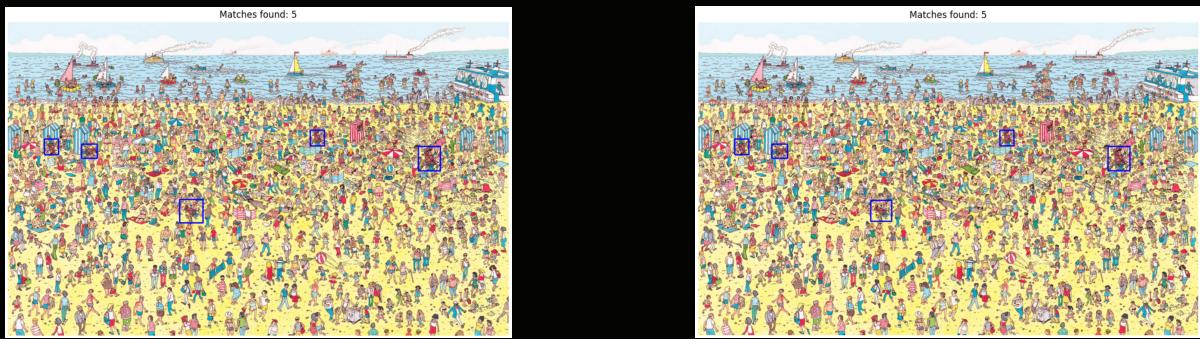
در این روش، در ابتدا کانفیگ های مختلف به ازای درجه های چرخیدن ۰°، ۹۰°، ۱۸۰°، ۲۷۰°، ۳۱۵° و ۳۶۰° درجه و به ازای میزان اسیکلیک ۰.۸، ۰.۹، ۱.۰، ۱.۱، ۱.۲، ۱.۳، ۱.۴، ۱.۵ ساخته شد که در مجموع ۶۴ کانفیگ گوناگون را می سازند.



شکل ۲۱: جهت چرخش در تصویر ورودی والدو

سپس به کمک کتابخانه cv2 و با استفاده ازتابع cross-correlation به ازای تصویر ورودی، انواع این ۶۴ کانفیگ به ورودی اعمال شد. در ابتدا، به ازای ورودی والدو و تصویر زمینه رنگی rgb ، تست گرفته شد که نتیجه خروجی مطلوب است و تمامی والدو ها به درستی تشخیص داده شده اند. (برای میزان شباهت باید بر روی متغیر ترشولد حداقل شباهت tuning میشد که پس از خروجی درست شد).

سپس برای همین متد ارائه شده، و با ورودی تصاویر زمینه و والدو grayscale شده نیز همین عملیات انجام شد و خروجی دقیقا مشابه و البته با صرف یک سوم زمان به ازای ورودی رنگی انجام شد. و اما برای مقایسه زمان پیاده سازی: که به طرز واضحی نشان دهنده سرعت خروجی دادن یک سوم شده است و همان کیفیت نیز حفظ شده است.



شکل ۲۳: به ازای ورودی تصویر خاکستریس

شکل ۲۲: به ازای ورودی تصویر رنگی

زمان مصرف شده	نوع ورودی
15.819602489471436	به ازای ورودی خاکستری
5.439622163772583	به ازای ورودی رنگی

جدول ۶: زمان صرف شده برای پیدا کردن والدو

روش دوم برای کاهش زمان مصرف شده : multi-resolution

Multi-resolution correlation

- Multi-resolution template matching
 - reduce resolution of both template and image by creating an **image pyramid**
 - match small template against small image
 - identify locations of strong matches
 - expand the image and template, and match higher resolution template selectively to higher resolution image
 - iterate on higher and higher resolution images
- Issue:
 - how to choose detection thresholds at each level
 - too low will lead to too much cost
 - too high will miss match

شکل ۲۴: پیدا کردن والدو به کمک روش چند رزولوشنی طبق اسلاید های درسی

در این روش، به جای مقایسه الگوی کامل با تمام موقعیت‌های ممکن در تصویر اصلی با وضوح کامل، از هرم‌های تصویری استفاده می‌شود تا تطبیق به صورت تدریجی و از وضوح پایین به بالا انجام شود.

فرآیند با ساخت هرم‌هایی از تصویر ورودی و الگو آغاز می‌شود. هر سطح از هرم، نسخه‌ای کوچکتر (پایین‌نمونه‌گیری شده) از سطح قبلی است که عرض و ارتفاع آن به نصف کاهش می‌یابد. برای مثال، یک تصویر 1024×1024 در سطح ۱ به 512×512 و در سطح ۲ به 256×256 تبدیل می‌شود. تطبیق از پایین‌ترین سطح هرم (مثلًاً سطح ۲) آغاز می‌شود؛ جایی که تصویر کوچکتر است و پردازش سریع‌تر انجام می‌گیرد. در این سطح، یک جستجوی کامل انجام شده و ممکن است تعداد زیادی تطبیق تقریبی پیدا شود — مثلًاً حدود ۱۰۰۰ مورد.

این تطبیق‌های تقریبی در سطح بالاتر هرم (مثلاً سطح ۱) با دقت بیشتر بررسی و اصلاح می‌شوند. اما به جای بررسی کل تصویر، فقط نواحی اطراف تطبیق‌های قبلی بررسی می‌گردند. این کار به طور چشمگیری تعداد مقایسه‌ها را کاهش می‌دهد. بسیاری از تطبیق‌های ضعیف در این مرحله حذف می‌شوند و ممکن است تعداد کاندیدها از ۱۰۰۰ به ۲۰۰ مورد کاهش یابد.

در نهایت، در بالاترین سطح هرم (سطح ۰، یعنی تصویر با وضوح کامل)، تطبیق دقیق فقط در نواحی مربوط به آن ۲۰۰ کاندیدای فیلترشده انجام می‌شود، نه در کل تصویر. در این مرحله، تنها تطبیق‌های قوی و دقیق باقی می‌مانند.

این فرآیند پالایش سلسه‌مراتبی همان چیزی است که تطبیق مبتنی بر هرم را بسیار قدرتمند می‌سازد. در واقع، بار محاسباتی اصلی به سطوح پایین‌تر منتقل می‌شود که تطبیق در آن‌ها بسیار سریع‌تر است، و تنها برای نواحی مناسب در وضوح بالا وقت صرف می‌شود. نتیجه این است که سیستمی با دقت بالا ولی سرعت بسیار بیشتر نسبت به روش تطبیق کلی در وضوح کامل خواهیم داشت.

در نهایت، استفاده از این روش نیازمند tune کردن تعداد لایه‌ها نیز می‌باشد. نتیجه تیون کردن تعداد لایه‌ها به این شکل بود که هرچه لایه‌ها را بیشتر می‌کردم، تعداد مواردی که تشخیص داده می‌شد نیز بیشتر می‌شد و باید ترشولد را نیز بالاتر می‌بردم. این بالاتر بردن ترشولد باعث می‌شود والدوهایی که روتیت شده بودند (که به علت خاصیتی که از کیفیت والدو اصلی به دلیل روتیت شدن برخوردار نبودند) قابل تشخیص نباشند. بنابراین تعداد لایه‌ها را محدود به ۳ لایه کردم.

همچنین، طبق بررسی هایم، هر چقدر تعداد لایه‌ها بیشتر باشد، سرعت خروجی دادن نیز بیشتر می‌شد. (البته مسلماً این رابطه تا بینهایت لایه تاثیر گذار نیست، اما تا ۶ لایه تاثیر خودش را نشان میداد.) اما به این دلیل که کیفیت خروجی را کاهش میداد، در نهایت، به همان سه لایه بسنده کردم. همچنین از نظر سرعت نیز، به ازای ورودی ۶۴ کانفیگ توضیح داده شده، چنین است:

زمان مصرف شده	نوع ورودی
6.921	به ازای ورودی خاکستری
19.6	به ازای ورودی رنگی

جدول ۷: زمان صرف شده برای پیدا کردن والدو با هرم رزولوشن‌های مختلف

که البته نسبت به روش اولیه افزایش یافته است. دلیل آن هم می‌توان در این یافت که تعداد لایه‌ها محدود است و پیشرفت زمانی خودش را نمی‌تواند نشان دهد. همچنین این متدهای تیونینگ بود که توضیح داده شد.

یادگیری عمیق در پیدا کردن والدو

- Finding “Waldo” using a simple Convolutional Neural Network

با توجه به این صفحه مدیوم، در ابتدا یک دیتاست ساختم. دیتاست ساخته شده ام به این شکل عمل می‌کرد که در ابتدا یک سمت پل ۲۰۰*۲۰۰ از تصویر زمینه می‌گرفت و بعد با یک درصد رندومنس والدو را بر روی آن بر روی یک نقطه رندوم و با یک کانفیگ رندوم قرار میداد.

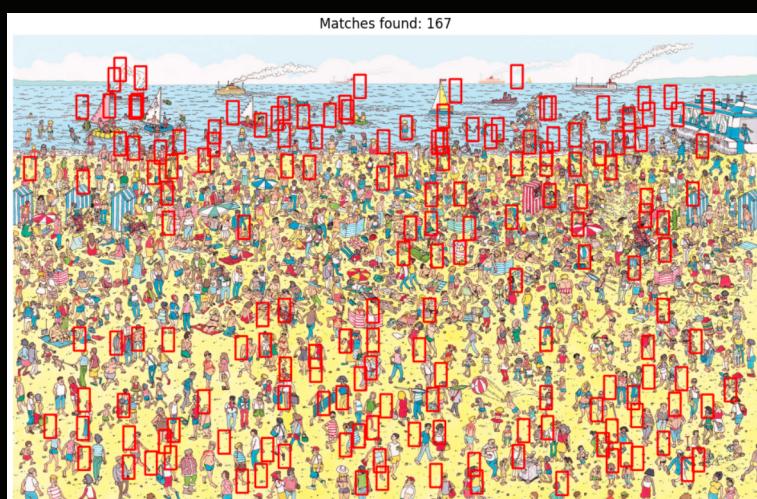
سپس بر روی یک شبکه عصبی pre-train شده از نوع mobile-net، هم classification به معنی بررسی حضور یا عدم حضور والدو و هم regression برای پیدا کردن موقعیت والدو پیاده سازی شد.

نتیجه این مدل این شد که حتی با کمترین ترشولد دقت نیز، تنها ۳۰ موقعیت تشخیص داده شده بود که همان هم خیلی رندوم بودند و اصلاً والدو نبودند.

سپس با بررسی بیشتر، دلیل این موضوع را به کم بودن دیتا والدو واقعی یافتم. بدین منظور دیتاست جدیدی ساختم که ۹۰ درصد دیتا والدو حاضر بود و در بقیه نبود و سپس فاین تیون کردن را انجام دادم. این مدل در پیدا کردن والدو، همانطور که انتظار میرفت، بیشتر ریسک میکرد، و تعداد بیشتری کاندید نشان میداد، اما باز هم والدو واقعی هیچی پیدا نکرد.

در نهایت دلیل اصلی عدم موفقیت این دو دیتاست و مدل ها را پیدا کردم: این که باید تصویر زمینه ثابت و برابر با کل تصویر اصلی باشد. باید ذکر کنم که انگیزه ام از شیوه قبلی تنظیم دیتاست، این بود که وقتی کل زمین را یکجا به مدل میدهیم، به نوعی یادگیری واقعی انجام نشده و به سمت ورودی بایاس می شویم. اما با خواندن مقاله، دیدم که ظاهرا همین نوع که کل تصویر زمینه اصلی را استفاده کنیم، روش استانداردی است. بدین منظور، دیتاست جدیدی ساختم که والدو ها را به شکل رندوم مشابه قبلی در تصویر patch میکرد، اما این تصویر، به جای سمپل های رندوم ۲۰۰۰*۲۰۰۰ از تصویر زمینه اصلی، کل تصویر زمینه اصلی بود.

نتیجه این دیتاست اخیری، این شد که گویا واقعاً یادگیری انجام میشد، شاخص loss از نزدیک به $k=1000$ به 1000 رسید، که در همینجا متوقف شد. به دلیل سبک بودن بیش از حد مدل من، و همچنین کم بودن دیتا والدو (تنها ۵۰۰ تصویر ساختم که همگی شامل تصویر اصلی بودند). به علت محدودیت حافظه گوگل کولب (همان ۵۰۰ تصویر، نزدیک به ۷ گیگابایت رم از ۱۲ گیگ را اشغال کرده بوند)، مدل همچنان loss بالای داشت که عدد 1000 همچنان نمیتوانست جای والدو را به خوبی پیدا کند. در صورتی که میتوانستم از مدلی سنگین تر و همچنین دیتاستی غنی تر استفاده کنم، مسلماً به پاسخ بهتری میرسیدم، اما با همین مدل نهایی، یک والدو درست (و تعداد زیادی موارد نامرتب) تشخیص داده شد.



شکل ۲۵: والدو با شبکه عصبی سوم

همچنین شایان ذکر است مدت زمان inference این مدل سومی، حدود ۱۷ ثانیه طول میکشد که از روش های کلاسیک نیز بیشتر است.

(۵) پروژه پنجم: استخراج ویژگی

توضیحات در رابطه با پارامتر های این سه روش استخراج ویژگی خواسته شده، به شرح زیر هستند:

۱. SIFT

این متد را به تفصیل در اسلاید های درسی داشتیم و من از توضیح مجدد آن خودداری می کنم. تنها موردی که در آن وجود دارد پارامتر آن است که تنها پارامتری که تنظیم می شود، ماکسیمم تعداد فیچر آن است.

۲. ORB

این روش را میتوان به شکل ترکیبی از هریس و سیفت دانست که از هریس دقیق تر است (هریس اصلاً توصیفگری نداشت) و از سیفت نیز سریع تر است. در این روش در ابتدا کرنر ها پیدا میشوند با کمک یک متد به نام FAST و سپس جهت اصلی هر کرنر پیدا می شود و تفاوت اصلی اش با سیفت در این است که برای توصیف هر فیچر، از BRIEF استفاده می کند که خیلی کوتاه تر از توصیفگر سیفت است. تنها پارامتری که میتوان برای این قائل شد، همین ماکسیمم تعداد فیچر است.

۳. Harris detection corner

در ابتدا مروری بر روی روش هریس و روش پیدا کردن کرنر ها به وسیله آن انجام میدهم. هریس ایده اصلی اش را از pca میگیرد، که در پنجره ای لغزنده بر روی تصویر، در راستای x و y گرادیان میگیرد و هرکجا که گرادیان در هر دو جهت مثبت شد یعنی یک کرنر داریم. اما به شکل دقیق تر، در هریس به جای گرادیان گرفتن از فیلتر های سوبل استفاده می کنیم. یعنی:

$$I_x = \text{Sobel}(I, \text{direction} = x)$$

$$I_y = \text{Sobel}(I, \text{direction} = y)$$

که در نهایت این I_x و I_y اجزای اصلی ماتریس ممان دوم می شوند:

$$M = \begin{bmatrix} \sum_{\text{window}} I_x^2 & \sum_{\text{window}} I_x I_y \\ \sum_{\text{window}} I_x I_y & \sum_{\text{window}} I_y^2 \end{bmatrix}$$

و پس از آن، با محاسبه تریس ماتریس، در رابطه با کرنر بودن یا نبودن یک موقعیت تصمیم میگیریم:

$$R = \det(M) - k \cdot (\text{trace}(M))^2$$

که در آن

$$\det(M) = (\sum I_x^2)(\sum I_y^2) - (\sum I_x I_y)^2$$

$$\text{trace}(M) = \sum I_x^2 + \sum I_y^2$$

را داریم. با اعمال threshold بر روی R میتوانیم به نقاط کرنر برسیم.

اکنون با توجه به فرمول ها و بحثی که شد، پارامتر های این روش استخراج ویژگی را جمع بندی میکنیم:

• ksize •

سایز کرنل مربوط به فیلتر سوبل. هرچقدر این کرنل سایز بیشتری داشته باشد، حساسیت به نویز آن بیشتر می شود و مناسب فیچرهای بزرگ است (چون اگر تصویر شامل فیچرهای زیاد و کوچک باشد، سیار کرنل بزرگ باعث میشود تعداد زیادی نویز هم به عنوان فیچر انتخاب شوند).

• blocksize •

سایز پنجره ای که برای آن ماتریس ممان دوم را محاسبه میکنیم و فیچرهای را پیدا میکنیم. مسلماً هرچه این سایز کوچکتر باشد، پیمایش تصویر با این نوع پنجره های کوچک نیازمند زمان بیشتر است و همچنین ممکن است که با نویز رو به رو شویم که یعنی باید این پارامتر را با پارامتر threshold به شکل مستقیم تنظیم کنیم و مستقل از هم نیستند.

• k •

این پارامتر مستقیم از همین فرمول به دست می آید:

$$R = \det(M) - k \cdot (\text{trace}(M))^2$$

هرچقدر این k کوچکتر باشد، احتمالاً لبه ها را به عنوان کرنر بر میگرداند و هرچقدر بزرگتر باشد، کرنر های مهم تر و بولد تر را بر میگرداند.

• thresh_rel •

این پارامتر را برای تعیین threshold نسبی به نسبت بیشترین مقدار h است که توسط هریس برای نقطه ویژگی انتخاب شده. مثلاً اگر یک نقطه مقدار h اش 8000 باشد و این ترشولد نیز 1 درصد باشد، همه فیچرهایی که مقدار کمتری از 80 دارند حذف می شوند. گفتنی است که این یک پارامتر مستقیم برای خود هریس نیست، اما در تابعی که پیاده کرده ام، برای حذف فیچرهای نویزی این پارامتر را هم قرار دادم.

• nfeatures_desc •

این پارامتر نیز به شکل مستقیم جزو پارامترهای خود هریس قرار نمیگیرد و وظیفه اش این است که یک مقدار ماکسیمم برای تعداد کرنر هایی که هریس پیدا کرده قرار دهد. بدین شکل که ابتدا کرنر ها بر اساس میزان h ای که دارند، از بزرگ به کوچک سورت میشوند و سپس این ترشولد بر روی آنها اعمال می شود.

• keypoint_size_harris •

این پارامتر را با مقدار 3 به شکل hardcode قرار داده ام و در پارامترهای تابعمن قرار ندادم و در ادامه نیز tune اش نکردم. هر کدام از روش های بهتر استخراج ویژگی مانند sift و غیره، همگی مقدار دیفالتی را برای بیان کردن ویژگی های خود دارند، اما هریس چون اصلاً descriptor ندارد، پس باید یک size به منظور توصیف همسایگی آن پیکسل انتخاب شود که همین پارامتر است. اکنون دلیل آن که آن را به شکل hardcode قرار داده ام، این است که بعداً از ORB برای توصیف آن نقاط استفاده می کنم که یعنی هرچه آن size را تغییر دهیم، باز هم توسط ORB اورراید می شود و تبدیل به مقدار ثابتی می شود و در نهایت هیچ تفاوتی در توصیف ویژگی وجود نخواهد داشت. شایان ذکر است که در پایان کد هایم یک تابع تست برای همین موضوع نوشته ام و این را کامل تشریح کرده ام که با بررسی آن، مواردی که گفته ام اثبات می شود.

در نهایت دلیل کندی این روش هم پیاده سازی همین ماتریس ممان دوم برای هر پنجره تصویر است؛ پنجره ها نیز stride اشان برابر با ۱ پیکسل است که مشخصا خیلی زمان برخواهد بود.

در ادامه، force brut matching پیاده سازی شده که به این شکل عمل می کند که به ازای تمامی توصیف گرهای تصویر A مقایسه انجام می شود با تمام توصیف گرهای تصویر B و فاصله میان آنها محاسبه می شود. همچنین گفتنی است که این فاصله برای SIFT از نوع اقلیدسی بوده و برای ORB و هریس (که توسط ORB توصیف می شود)، از نوع hamming می باشد. دلیل تفاوت میان این فواصل نیز نوع خروجی توصیفگرها می باشد. منبعی که برای پیاده سازی این brut matching مطالعه کردم:

- Feature Matching using Brute Force in OpenCV

سپس برای هر توصیفگر X از تصویر A دو تا توصیفگر از تصویر B انتخاب می شوند. در صورتی که این دو توصیفگر اختلاف قابل قبول داشتند، توصیفگر نزدیکتر به عنوان یک match برای X انتخاب می شود. در صورتی نیز که این دو توصیفگر تقریبا مشابه بودند، هر دوی آنها و نقطه X از تصویر اول کنار گذاشته می شوند و هیچ match ای رخ نمیدهد. این الگویی که برای mathing پیاده کردم مطابق است با مقاله اصلی

- Object Recognition from Local Scale-Invariant Features

و تحت عنوان *Lowe's ratio* بیان شده است. در نهایت و به عنوان یک خلاقيت، موقعيت مكانی توصیفگرها را نیز فیلتر کردم. بدین منظور، به این شکل عمل کرده ام که از RANSAC استفاده کردم که تعداد رندومی از match های مرحله قبل را انتخاب میکند و سپس یک transformation بر روی زوج نقاط انجام میدهد که این transformation را در اینجا به شکل homography انتخاب کرده ام. اگر این projection الگوی یکسانی با سایرین نداشت، این match کنار گذاشته می شود.

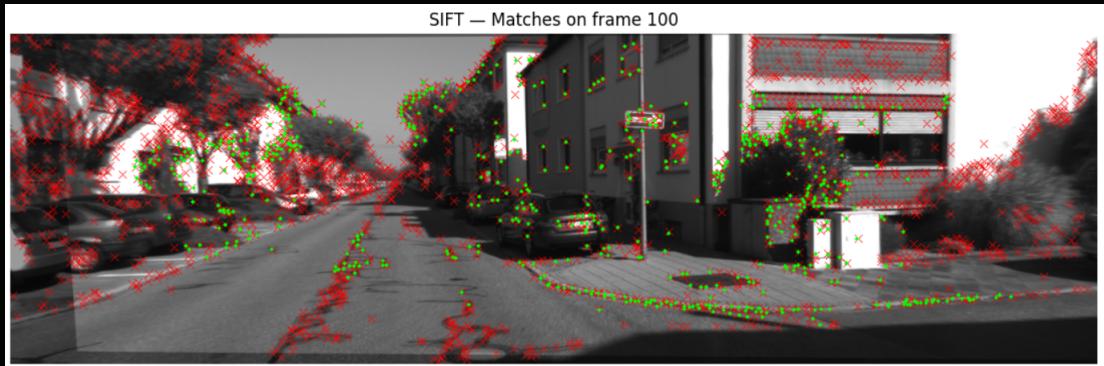
تشخيص لنمارک

برای پیدا کردن لنمارک ها، از یک ساختار داده ای به اسم tracker استفاده کردم که فیچرها را ترک میکند. مواردی که نگه داری میکند شامل یک آیدی یکتا، آخرین فریم دیده شده، و تعداد بارهای متوالی دیده شده و آخرین موقعیت آن است. در ابتدا، برای تمامی فیچرها انتخاب شده در فریم اول، یک tracker ساخته میشود و سپس، به سراغ فریم دوم میروم. در این لحظه، فیچرهای فریم دوم را با فریم اول مقایسه می کنم. اگر فیچری داشتم که با فیچری از فیچرهای فریم اول مج شده بود، برای آن یک ترکر فیچر فریم اول، آخرین فریم دیده شده را آپدیت می کنم و تعداد بارهای متوالی دیده شده را نیز یکی اضافه می کنم و موقعیت را هم آپدیت می کنم.

در نهایت برای فیچرهایی از فریم دوم که با هیچکدام از فیچرهای فریم اول مج نشدهند، یک ترکر جدید می سازم. همین الگو در یک حلقه برای تمامی فریم های متوالی اجرا می شود. این عملیات تحت عنوان پروپگیشین پیاده سازی شده است. در نهایت نیز، فیچرهایی که تعداد بارهای متوالی دیده شدنشان بیش از ۵ بار بود، به عنوان لنمارک برمیگردانم.

شایان ذکر است که در ابتدا قصد داشتم این عملیات را در دو حلقه و برای تک به تک فیچرپوینت های تک به تک فریم ها بدون پیاده سازی هیچ ساختار داده ای انجام دهم که هم خوانایی کمی داشت

و هم عملیات تشخیص لندمارک را طولانی میکرد که از هدف پروژه برای مقایسه زمان تشخیص دهنده فیچرها، دور بود و سپس این سبک را پیاده کردم.
همچنین تابعی را پیاده کردم که به شکل دقیقی، دو تصویر متوالی را روی هم با استفاده از فیچرهای مج شده قرار میداد و همچنین آن نقاط مج شده را نیز نمایش میداد:



شکل ۲۶: قرار گرفتن دو تصویر متوالی به کمک سیفت

مقایسه متدها و همچنین نتیجه تیونینگ پارامترها نیز به شرح زیر است:

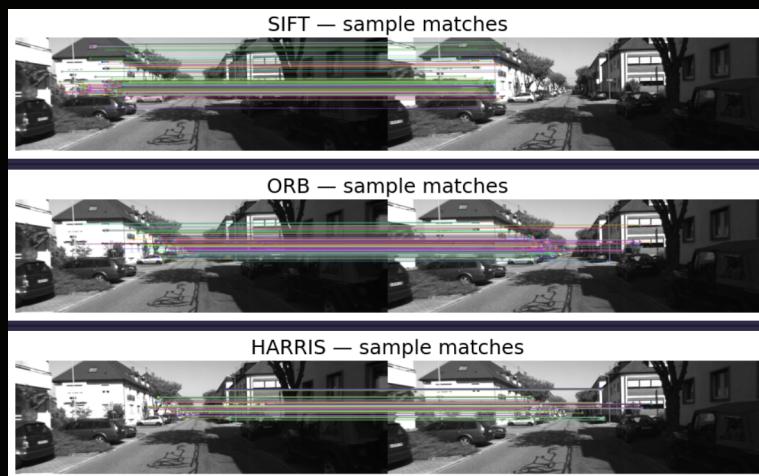
- مقایسه تعداد لندمارک‌ها:
دو متده سیفت و ORB هر دو در حدود ۱۰هزار لندمارک را پیدا کرده بودند، در حالی که متده هریس نصف آنها یعنی ۵هزار لندمارک را پیدا کرد که نشان می‌دهد هریس به دلیل خاصیت ذاتی خود در توصیف نقاط دچار مشکل می‌شود. این عدد‌ها برای کل تصاویر ذکر شده‌اند.
- بررسی تاثیر RANSAC نیز نتایج جالبی داشت. بدون استفاده از این فیلتر موقعیت مکانی، تعداد landmark‌ها در سیفت ۱۸هزار، در orb نیز ۱۵هزار و در هریس ۶هزار شد که بیان می‌کند که تاثیر فوق العاده‌ای در پیدا کردن و فیلتر کردن فیچرپوینت‌های نامطمئن (که حتی لندمارک نیز شده‌اند) را خواهیم داشت.
- تاثیر پارامتر ماکسیمم فیچر در orb و سیفت: در orb این پارامتر توسط مقدار built-in میزان ۵۰۰ واحد تنظیم می‌شد که در نتیجه، در صورتی که آن را تنظیم نمیکردم، این میزان تنظیم می‌شد. اما برای سیفت، در صورتی که این میزان را تنظیم نمیکردم، تعداد فیچرها ۳۲۰۰ تا می‌شد (برای تک تصویر و نه کل تصاویر) و این یعنی به طور کلی در صورت عدم تیون کردن این پارامتر برای سیفت، ممکن است با فیچرپوینت‌هایی نه چندان قوی نیز رو به رو شویم.
- تاثیر thresh_rel در سیفت: برای هر کدام از مقادیر به ترتیب ۱هزارم، ۵هزارم و ۱صدم، مقادیر ۴۹۱ و ۹۶۵ و ۱۹۳۴ تا فیچر (به ازای یک تصویر) استخراج شد که قابل پیش‌بینی نیز می‌باشد و کاملاً منطقیست که تعداد فیچرها به ازای ترشولد سختگیرانه‌تر، کمتر شود.
- میزان زمان استفاده شده برای پیدا کردن کل فیچرها و کل عملیات در سه روش در جدول ۸ قابل مشاهده است:

که نشان می‌دهد که orb در حالت معمول بسیار سریع‌تر از سیفت عمل میکند. همچنین سریع بودن هریس نیز به دلیل تیون کردن پارامترها و اعمال محدودیت بر روی تعداد نقاط انجام شده است.

Method	Detection	Matching	Total	Per Frame Avg	Avg Candidates	Landmarks
SIFT	13853.8	5006.4	18860.1	69.3	2540	10410
ORB	1819.8	2788.6	4608.4	9.1	2000	10362
HARRIS	2268.9	2292.3	4561.2	11.3	1869	5618

جدول ۸: مقایسه زمان مصرف شده در سه روش: زمان ها بر حسب میلی ثانیه

در نهایت، میان دو فریم تابع پلات را برای نمایش نحوه match شدن میان فیچر ها را نیز پلات کرده ام.



شکل ۲۷: نمایشی از مج شدن فیچر ها میان دو فریم

منابع

1. Image Denoising Algorithms: A Comparative Study of Different Filtration Approaches Used in Image Restoration
2. Digital Image Processing By Gonzalez, 4th Edition
3. Automatic Identification of Noise in Ice Images Using Statistical Features
4. Medium: A Beginner's Guide to Computer Vision (Part 4) - Pyramid
5. Object Recognition from Local Scale-Invariant Features
6. Feature Matching using Brute Force in OpenCV
7. Finding "Waldo" using a simple Convolutional Neural Network
8. DM-FNet: Unified multimodal medical image fusion via diffusion process-trained encoder-decoder
9. The Laplacian Pyramid as a Compact Image Code