تمرین سری 2 بینایی ماشین **زهرا عبادی**

.1

: Alpha Blending

آلفا بلندینگ برای ترکیب دو تصویر یا شیء به کار می‌رود و امکان شفافیت و ترکیب صاف(smooth blending) عناصر همپوشانی را فراهم می کند. این تکنیک شامل اختصاص یک مقدار آلفا (شفافیت) به هر پیکسل است.

آلفا اغلب کانال چهارم یک تصویر است (به عنوان مثال در یکtransparent PNG) اما همچنین می تواند یک تصویر جداگانه باشد. این ماسک شفاف اغلب ماسک آلفا یا آلفا مات نامیده می شود.

در تصویر بالا، تصویر پیش‌زمینه در بالا سمت چپ، ماسک آلفای خاکستری در بالا سمت راست، تصویر پس‌زمینه در پایین سمت چپ نشان داده شده است، و تصویر ترکیبی به دست آمده از ترکیب تصویر پیش زمینه و تصویر پس زمینه با استفاده از ماسک آلفا در پایین سمت راست نشان داده شده است.

در هر پیکسل از تصویر، باید رنگ تصویر پیش زمینه **(F)** و رنگ تصویر پس زمینه **(B)** را با استفاده از ماسک آلفا ترکیب کنیم.

مقدار **α** استفاده شده در معادله مقدار پیکسل در ماسک آلفا تقسیم بر **255** است. بنابراین، در معادله زیر :$0 \leq \alpha \leq 1$

$$I = \alpha F + (1 - \alpha)B$$

وقتی $\alpha = 0$ ، رنگ پیکسل خروجی پس زمینه است.

وقتی $\alpha = 1$ ، رنگ پیکسل خروجی پیش زمینه است.

وقتی $0 < \alpha < 1$ باشد، رنگ پیکسل خروجی ترکیبی از پس زمینه و پیش زمینه است. برای ترکیب واقع گرایانه، مرز ماسک آلفا معمولا دارای پیکسل هایی بین **0** تا **1** است.

نمونه کد Alpha Blending :

```python
import cv2
from google.colab.patches import cv2_imshow

# Read the images
apple = cv2.imread('drive/MyDrive/apple.jpg')
orange = cv2.imread('drive/MyDrive/orange.jpg')

# Resize the 'apple' image to have the same dimensions as the 'orange' image
apple = cv2.resize(apple, orange.shape[1::-1])

# Display the resized 'apple' and 'orange' images
cv2_imshow(apple)
cv2_imshow(orange)

# Take user input for the alpha value (weight of blending)
alpha = float(input("alpha value : "))

# Perform alpha blending using the cv2.addWeighted function
# dst = alpha * apple + (1 - alpha) * orange + 0 (constant)
dst = cv2.addWeighted(apple, alpha , orange, 1-alpha, 0)

# Write the result of alpha blending to a file named 'alpha-blended.png'
cv2.imwrite('alpha_blended.png', dst)

# Read the alpha-blended image and display it
alpha_blended = cv2.imread('alpha_blended.png')
cv2_imshow(alpha_blended)
```

خروجی کد بالا با مقادیر مختلف آلفا :

alpha value : 0

alpha value : 0.3

alpha value : 0.5

alpha value : 0.7

Green Screen Composing:

ترکیب صفحه سبز، همچنین به نام کروما کی(Chroma key) ، یک تکنیک جلوه‌های بصری و پس از تولید برای ترکیب (لایه‌بندی) دو یا چند تصویر یا جریان ویدئو با هم بر اساس رنگ‌های رنگی (محدوده کروم) است. این تکنیک در بسیاری از زمینه‌ها برای حذف پس‌زمینه از موضوع عکس یا ویدیو استفاده شده است – به‌ویژه صنایع پخش اخبار، تصاویر متحرک و بازی‌های ویدیویی. یک محدوده رنگی در فیلم پیش زمینه شفاف می شود، که اجازه می دهد فیلم پس زمینه جداگانه فیلمبرداری شده یا یک تصویر ثابت در صحنه قرار گیرد.

Chroma key را می توان با پس زمینه های هر رنگی که یکنواخت و متمایز هستند انجام داد، اما پس زمینه های سبز و آبی بیشتر مورد استفاده قرار می گیرند، زیرا از نظر رنگ با هر رنگ پوست انسان متفاوت تر هستند. هیچ بخشی از موضوعی که از آن فیلم گرفته می شود یا عکس گرفته می شود، نباید رنگ مورد استفاده به عنوان پشتیبان را تکرار کند، یا ممکن است بخشی به اشتباه به عنوان بخشی از پشتیبان شناسایی شود. Chroma key اغلب شامل تنظیم شدگی رنگ، استفاده از الگوریتم کی و افکت‌های مختلف دیگر در پوشش فاکتور بر روی پس‌زمینه جدید می‌شود.

چندین تکنیک مختلف با کیفیت و سرعت بهینه برای اجرای Chroma keys وجود دارد :

در اکثر نسخه ها، یک تابع $f(r, g, b) \rightarrow \alpha$ برای هر پیکسل در تصویر اعمال می شود. **α** (آلفا) معنایی مشابه با تکنیک های ترکیب آلفا دارد. $\alpha \leq 0$ به این معنی است که پیکسل به طور کامل در صفحه سبز قرار دارد، $\alpha \geq 1$ به این معنی است که پیکسل به طور کامل در شی پیش زمینه قرار دارد و مقادیر میانی نشان می دهد که پیکسل تا حدی توسط شی پیش زمینه (یا شفاف) پوشیده شده است. یک تابع دیگر $g(r, g, b) \rightarrow (r, g, b)$ برای حذف نشت سبز روی اشیاء پیش زمینه لازم است.

یک تابع بسیار ساده **f** برای صفحه سبز $A(r+b) - Bg$ است که در آن **A** و **B** ثابت های قابل تنظیم کاربر با مقدار پیش فرض **1.0** هستند. یک **g** بسیار ساده $(r, \min(g,b),b)$ است. این تقریباً نزدیک به قابلیت های آنالوگ و کشیدن صفحه نمایش مبتنی بر فیلم است.

نمونه های مدرن از این توابع به بهترین وجه توسط دو سطح تو در تو بسته در فضای سه بعدی **RGB**، اغلب کاملاً پیچیده توصیف می شوند. رنگ های داخل سطح داخلی صفحه سبز در نظر گرفته می شوند. رنگ های خارج از سطح بیرونی پیش زمینه مات هستند. رنگ‌های بین سطوح تا حدی پوشیده شده‌اند، هر چه به سطح بیرونی نزدیک‌تر باشند مات‌تر می‌شوند. گاهی اوقات از سطوح بسته بیشتری برای تعیین نحوه حذف نشت سبز استفاده می شود. همچنین بسیار متداول است که **f** به چیزی بیش از رنگ پیکسل فعلی بستگی دارد، همچنین ممکن است از موقعیت **(x, y)**، مقادیر پیکسل های نزدیک، مقدار تصاویر مرجع یا مدل رنگی آماری استفاده کند. صحنه، و مقادیر از ماسک های ترسیم شده توسط کاربر. اینها سطوح بسته را در فضای بیش از سه بعدی تولید می کنند.

تفاوت کاربرد ها :

Alpha Blending :

به طور عمده در گرافیک کامپیوتری، ویرایش تصاویر، و تولید ویدئوها مورد استفاده قرار می‌گیرد.

در تولید انیمیشن، افکت‌های ویژه، و ترکیب صحنه‌های مختلف با یکدیگر از این تکنیک استفاده می‌شود.

Laplacian Pyramid :

در زمینه‌های فشرده‌سازی تصویر، سنتز تصویر، و افزایش کیفیت تصاویر استفاده می‌شود.

معمولاً در الگوریتم‌های پردازش تصویر برای افزایش تفاوت‌های جزئیات و افزایش کارایی به‌کار می‌رود.

Green Screen Composing (Chroma Keying) :

تولید فیلم و تلویزیون به‌کار می‌رود تا فاکتور یا شخص را از پس‌زمینه جدا کرده و به پس‌زمینه دلخواه جایگزین کند.

این تکنیک برای ساخت صحنه‌های مجازی، افکت‌های ویژه، و ترکیب فیلم‌های مختلف با یکدیگر استفاده می‌شود.

به طور کلی :

- Alpha Blending: برای ترکیب دو تصویر با هم استفاده می‌شود و در مواردی که نیاز به انتقالات نرم بین المان‌های همپوشان دارید.
- Laplacian Pyramid: برای ترکیب جزئیات تصویر در مقیاس‌های مختلف و افزایش کیفیت تصویر استفاده می‌شود.
- Green Screen Composing: برای جدا کردن یک شیء از پس‌زمینه و جایگزینی آن با پس‌زمینه دلخواه استفاده می‌شود، معمولاً در صنعت فیلم و تولید تلویزیون.

تفاوت خروجی ها :

Alpha Blending :

خروجی نهایی یک تصویر است که المان‌های دو تصویر با توجه به مقادیر آلفا به‌طور نرم و ترکیب شده‌اند.

این روش باعث ایجاد انتقالات آرام و نرم بین المان‌های همپوشان می‌شود.

Laplacian Pyramid :

خروجی این فرآیند یک سری تصاویر است که هرکدام جزئیات تصویر را در یک مقیاس خاص نشان می‌دهند.

با ترکیب این تصاویر، تصویر نهایی با جزئیات حاصل از هر مقیاس به دست می‌آید.

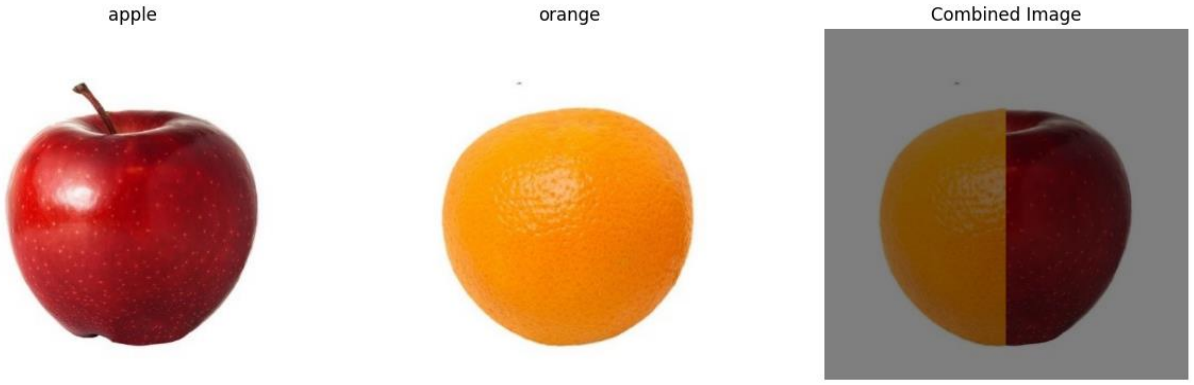
Green Screen Composing (Chroma Keying):

خروجی این روش یک تصویر یا ویدئو است که فاکتور در مقابل یک پس‌زمینه جدید قرار گرفته و رنگ مشخصی با تصویر جایگزین شده است.

به طور کلی :

- Alpha Blendingترکیب نرم و پیوسته دو تصویر را ایجاد می‌کند و به تناسب با مقادیر آلفا انتقالات نرم بین المان‌های همپوشان فراهم می‌کند.
- Laplacian Pyramidجزئیات تصویر را در مقیاس‌های مختلف نشان می‌دهد و تصویر نهایی با توجه به این جزئیات ترکیب می‌شود.
- Green Screen Composingبا جایگزینی یک رنگ خاص در تصویر با یک پس‌زمینه موردنظر، امکان ترکیب فاکتور با پس‌زمینه مختلف را فراهم می‌کند.

خروجی ترکیب دو تصویر سیب و پرتقال با روش Alpha Blending : (alpha = 0.5)



خروجی ترکیب دو تصویر سیب و پرتقال با روش Laplacian Pyramid : (n = 3)

.2

Image retexturing به معنای تغییر بافت یک تصویر با یک بافت جدید است. این فرآیند در زمینه‌های مختلفی از جمله گرافیک کامپیوتری، واقعیت مجازی، بازی‌های ویدئویی، و ویرایش تصاویر استفاده می‌شود. در ادامه، به دو دسته اصلی از روش‌های image retexturingیعنی روش‌های سنتی و روش‌های مبتنی بر یادگیری عمیق می‌پردازیم:

1. **روش‌های سنتی:**

- **الگوریتم‌های تطبیق بافت:**

الگوریتم‌هایی وجود دارند که از تطبیق بافت استفاده می‌کنند تا بافت جدید را روی تصویر اعمال کنند. این الگوریتم‌ها ممکن است بر اساس انتقال فرکانس، ماتریس کواریانس یا معیارهای دیگر بافت انتخاب شده عمل کنند.

Abstarct : ما یک روش ساده مبتنی بر تصویر را ارائه می‌دهیم که به منظور تولید ظاهر تصویر جدید استفاده می‌شود، که در آن یک تصویر جدید با دوختن به هم قطعات کوچک تصاویر موجود ساخته می‌شود. این فرآیند را به نام "image quilting" یا دوخت تصویر می‌نامیم. در ابتدا، از دوخت تصویر به عنوان یک الگوریتم سریع و بسیار ساده برای سنتز بافت استفاده می‌کنیم که نتایج قابل تعجبی برای مجموعه گسترده‌ای از بافت‌ها تولید می‌کند. در دومین مرحله، این الگوریتم را گسترش داده و به عنوان یک الگوریتم انتقال بافت به کار می‌بریم – به وجود آوردن یک شیء با یک بافت از یک شیء دیگر. به طور کلی، نشان می‌دهیم چگونه یک تصویر می‌تواند به سبک یک تصویر دیگر دوباره تجسم شود. این روش به طور مستقیم بر روی تصاویر عمل می‌کند و به اطلاعات سه‌بعدی نیاز ندارد.

مرجع:

Efros, A. A., & Freeman, W. T. (2001). Image quilting for texture synthesis and transfer. ACM SIGGRAPH.

- **روش‌های برداری:**

در این روش‌ها، ویژگی‌های وابسته به بافت مثل ساختار و رنگ تصویر تحلیل می‌شوند و بافت جدید بر اساس این ویژگی‌ها به دست می‌آید.

Abstarct :در این مقاله، یک الگوریتم کارآمد برای سنتز واقع‌گرایانه بافت ارائه می‌شود. این الگوریتم آسان در استفاده است و تنها یک بافت نمونه را به عنوان ورودی می‌پذیرد. این الگوریتم توانایی تولید بافت‌های با کیفیت ادراک شده معادل یا بهتر از نتایج تولید شده توسط تکنیک‌های گذشته را دارد، اما دو سفارش سرعت بیشتر اجرایی دارد. این امکان را فراهم می‌کند که ما به سنتز بافت در مسائلی که به طور سنتی غیرممکن می‌نموده استفاده کنیم. به ویژه، ما آن را در سنتز محدود برای ویرایش تصویر و تولید

بافت زمانی اعمال کرده‌ایم. الگوریتم ما از مدل‌های بافت میدان تصادفی مارکوف گرفته شده است و بافت‌ها را از طریق یک فرآیند جستجوی قطعی تولید می‌کند. ما این فرآیند سنتز را با استفاده از کوانتیزاسیون برداری ساختار درختی سرعت می‌بخشیم.

مرجع:

Wei, L. Y., Levoy, M., & Pulli, K. (2000). Fast texture synthesis using tree-structured vector quantization. Proceedings of the 27th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co

2. روش‌های مبتنی بر یادگیری عمیق:

- **شبکه‌های مولد معکوس:**

در این روش‌ها، از شبکه‌های مولد معکوس (GANs) استفاده می‌شود تا بافت جدیدی را برای تصویر ایجاد کنند.

Abstarct : انجام عملیات پردازش تصویر بر روی محتوای معنایی یک تصویر به سبک‌های مختلف یک وظیفه دشوار است. ممکن است بر اساس استدلال، یکی از محدودکننده‌های اصلی روش‌های گذشته، عدم وجود نمایش‌های تصویری بوده باشد که به وضوح اطلاعات معنایی را نمایش دهند و از این رو اجازه جدا کردن محتوای تصویر از سبک را فراهم کنند. در اینجا، از نمایش‌های تصویری که از شبکه‌های عصبی کانولوشنی بهینه‌سازی شده برای تشخیص اشیاء مشتق شده‌اند، استفاده می‌شود و که اطلاعات تصویر بالا را به وضوح نمایش می‌دهند. ما یک الگوریتم عصبی برای سبک هنری معرفی می‌کنیم که قادر است محتوا و سبک تصویرهای طبیعی را جدا کرده و مجدداً ترکیب کند. این الگوریتم به ما این امکان را می‌دهد که تصاویر جدیدی با کیفیت ادراکی بالا تولید کنیم که محتوای یک عکس دلخواه را با ظاهر چندین اثر هنری مشهور ترکیب می‌کنند. نتایج ما به ما بینش‌های جدیدی درباره نمایش‌های تصویر عمیقی که توسط شبکه‌های عصبی کانولوشنی یاد گرفته شده‌اند، ارائه می‌دهد و قابلیت آنها را برای سنتز و تلاش تصویر با سطح بالا نشان می‌دهد.

مرجع:

Gatys, L. A., Ecker, A. S., & Bethge, M. (2016). Image style transfer using convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition

- **شبکه‌های عصبی کانولوشنی:**

شبکه‌های کانولوشنی می‌توانند بافت‌های جدید را از تصاویر مرجع یاد بگیرند و آن را به تصاویر مورد نظر اعمال کنند.

Abstarct : در این مقاله، ما به روش سریع استایل‌سازی که در Ulyanov و همکاران (2016) معرفی شده است، مجدداً مرور می‌کنیم. ما نشان می‌دهیم که چگونه یک تغییر کوچک در معماری استایل‌سازی منجر به بهبود چشم‌گیر در کیفیت تصاویر تولید شده می‌شود. این تغییر به جابجایی نرمال‌سازی دسته با نرمال‌سازی نمونه محدود است و اینکه نرمال‌سازی نمونه را هم در زمان

آموزش و هم در زمان آزمایش اعمال کنیم. روش حاصل می‌تواند برای آموزش معماری‌های با عملکرد بالا برای تولید تصویر به صورت زمان واقعی استفاده شود.

مرجع:

Ulyanov, D., Vedaldi, A., & Lempitsky, V. (2016). Instance normalization: The missing ingredient for fast stylization. arXiv preprint arXiv:1607.08022.

: Abstarct

در پردازش تصویر، می‌توان بافت را به عنوان یک تابع از تغییرات فضایی شدت روشنایی پیکسل‌ها تعریف کرد. بافت اصطلاح اصلی برای تعریف اشیاء یا مفاهیم یک تصویر است. تجزیه و تحلیل بافت نقش مهمی در موارد دید کامپیوتری دارد، مانند شناسایی اشیاء، شناسایی عیوب سطح، تشخیص الگو، تجزیه و تحلیل تصاویر پزشکی و غیره. از آنجا که تاکنون بسیاری از روش‌ها برای دقیق توصیف تصاویر بافت ارائه شده‌اند. معمولاً روش‌های تجزیه و تحلیل بافت به چهار دسته تقسیم می‌شوند: روش‌های آماری، ساختاری، مبتنی بر مدل و مبتنی بر تبدیل. این مقاله به بررسی جزئیات روش‌های مختلف مورد استفاده برای تجزیه و تحلیل بافت می‌پردازد. تحقیقات جدید نشان می‌دهد که ترکیبی از روش‌ها برای تجزیه و تحلیل بافت، نمی‌تواند در دسته خاصی قرار گیرد. این مقاله یک مرور بر روی روش‌های ترکیبی معروف با جزئیات در یک بخش خاص ارائه می‌دهد. این مقاله مزایا و معایب توصیف‌گرهای معروف تصویر بافت را در قسمت نتایج شمرده می‌کند. تمرکز اصلی در تمام روش‌های باقی‌مانده بر عملکرد تشخیص، پیچیدگی محاسباتی و مقاومت در برابر چالش‌هایی مانند نویز، چرخش و غیره است. همچنین، یک بررسی مختصر انجام می‌شود درباره‌ی طبقه‌بندی تصویر بافت از طریق طبقه‌بندی‌های معمول استفاده شده. همچنین، یک بررسی از مجموعه داده‌های بنچمارک تصویر بافت ارائه شده است.

Row	Author	Paper subject		Dataset		Type classifier	Accuracy
1	Ahmadvand and Daliri [24]	Rotation invariant texture classification using extended wavelet channel combining and LL channel filter bank		VisTex		KNN	97.60
				Outex - CT10			99.24
				Brodatz			98.44
		Disadvantages	Higher complexity than LBP in terms of number of extracted features – net presented for color texture images				
		Advantages	Rotation and illumination invariant				
2	Du and Yan[44]	Local spiking pattern and its application		Outex	Outex 10	neural network	86.12
					Outex 12		66.62
		Disadvantages	Don't extended for color texture images – many input parameters – Don't use color information				
		Advantages	rotation invariant – Impulse noise Resistant – Illumination invariant				
3	Suresh and Shunmuganathan [45]	Gray Level Co-Occurrence Matrix Based Statistical Features		Brodatz		KNN	99.04
		Disadvantages	Don't provide a significant reason for selected statistical features – Noise sensitivity – sensitivity to texture database				
		Advantages	Better results than PSWT,TSWT and Linear regression – rotation invariant				

4	Mehta and Egiazarian[65]	Dominant Rotated Local Binary Patterns	Outex	Outex 10	K-NN	96.26
				Outex 12		99.19
			KTH-TIPS.			96.78
		Disadvantages	Impulse noise sensitive – Don't consider color information and global features			
		Advantages	Complete structural information extracted by LBP- complete structural information extracted by LBP - Lower complexity than some modified LBP versions – Rotation Invariant – Scale resistant			
5	Siqueira and et.al[42]	Multi-scale gray level co-occurrence matrices for texture description	UMD		KNN	95.2
			UIUC			81.7
			Brodatz			87.2
			VisTex			96.0
			Outex			89.4
		Disadvantages	noise sensitive – sensitive to the number of histogram bins – Nit adoptable with some classifiers			
		Advantages	Multi scale – consider color and texture features together			
6	Fekri-Ershad[43]	Combination of Edge& Co-occurrence and Local Binary Pattern	Stone		Naive Bayes	92
					KNN	3-NN 93.3
						5-NN 93.6
					LADTree	90

Row	Author	Paper subject	Dataset		Type classifier	Accuracy
		Disadvantages	High computational complexity			
		Advantages	Better accuracy than LBP or GLCM			
7	Guo and et.al[66]	LBP variance (LBPV) with global matching	Outex	Outex10	KNN	89.63
				Outex12		85.23
			CUReT			
		Disadvantages	Don't extended for color textures – Noise sensitivity			
		Advantages	Better Results than many modified LBP versions			
8	Nguyen and et.al[67]	Statistical binary patterns	KTH-TIPS		KNN	97.73
			KTH-TIPS 2b			71.59
			CUReT			98.73
			UIUC			97.4
			DTD			74
		Disadvantages	Resolution sensitive – Don't provide global features – High computational complexity			
		Advantages	rotational invariant – Noise invariant			
9	Chang and et.al[25]	SVM-PSO based rotation-invariant image texture classification in SVD and DWT domains	VisTex		SVM	100
			Sun Harvest			100
			Brodatz			99.55
			USC			100
			Coffee beans			98.22
		Disadvantages	Noise sensitivity – High computational complexities than methods in spatial domain			
		Advantages	Rotation Invariant – Extract features in spatial and frequency domain jointly			
10	Junior and Backes[68]	ELM based Signature for texture classification	VisTex		KNN	99.83
			Outex			97.59
			Brodatz			99.42
		Disadvantages	Many input parameters for neural network ELM – High computational complexity in terms of number of descriptors			
		Advantages	combination of statistical and structural features			
11	Hao and et.al[71]	Evaluation of ground distances and features in EMD-based GMM matching	KTH-TIPS 2b		KNN	78.6
			FMD			81.7
			ULUC			84
		Disadvantages	High computational complexity – Don't provide reasons for choose distance			
		Advantages	color and texture analysis jointly – Better Results than GMM based methods			

در این مطالعه، تلاش کرده‌ایم تا تقریباً تمامی مقالاتی که روش‌هایی برای تجزیه و تحلیل بافت در حوزه طبقه‌بندی بافت ارائه کرده‌اند را مورد نظر قرار دهیم. همانطور که چهار دسته اصلی برای روش‌های طبقه‌بندی بافت تعریف شده‌اند، برخی از روش‌های ابتدایی در یک دسته یکتا هستند. با این حال، از طریق گسترش روش‌ها و نوآوری در روش‌های ترکیبی، روش‌های تحلیل بافت جدید به بیش از یک دسته تخصیص داده می‌شوند. در دسته آماری، ماتریس هم‌آیندی و روش‌های Local Binary Patten محبوب‌تر هستند و در دسته مبتنی بر مدل، مدل‌های فراکتال معروف‌تر هستند. همچنین، در دسته مبتنی بر تبدیل، روش‌های Gabor و Wavelet بیشتر به‌کار رفته‌اند.

اغلب روش‌ها در دسته‌های آماری و مبتنی بر تبدیل یا ترکیبی از این دو دسته قرار دارند. یکی از دلایل اصلی تنوع روش‌ها، تغییر در روش تجزیه تصویر بافت است (مثل نویز، چرخش، مقیاس، روشنایی، دیدگاه). به عبارت دیگر، هر روش جدید به دنبال پاسخ به برخی از چالش‌هاست. تقریباً همه روش‌ها دارای تغییرات مقاوم به چرخش هستند؛ با این حال، اکثر روش‌ها حساس به نویز هستند. به عنوان مثال، روش‌های DRLBP و LBP variance LBP را گسترش داده و نتایج بهتری را به دست می‌آورند اما همچنان حساس به نویز هستند. بیشتر روش‌ها می‌توانند برای تصاویر بافت خاکستری استفاده شوند و Fekri-Ershad، Haon و Siqueira روش‌های خود را برای تصاویر بافت رنگی استفاده کرده‌اند. برخی از روش‌ها دلیل قاطعی برای انتخاب‌های خود ارائه نداده‌اند، به عنوان مثال "ویژگی‌های آماری مبتنی بر ماتریس هم‌آیندی سطح خاکستری".

مرجع:

Armi, Laleh & Fekri Ershad, Shervan. (2019). Texture image analysis and texture classification methods - A Review. 2. 1-29.

3. با استفاده از laplacian pyramid میخواهیم دو تصویر سیب و پرتقال را با هم ترکیب کنیم. (یک مرتبه برای تصویر سطوح خاکستری و یکبار هم رنگی) .

توضیحات کد برای انجام این کار :

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

# Read the color images
apple_color = cv.imread('drive/MyDrive/apple.jpg')
orange_color = cv.imread('drive/MyDrive/orange.jpg')

# Convert color images to grayscale
apple_gray = cv.cvtColor(apple_color, cv.COLOR_BGR2GRAY)
orange_gray = cv.cvtColor(orange_color, cv.COLOR_BGR2GRAY)

# Initialize lists to store image pyramids
A_color = [apple_color]
B_color = [orange_color]
A_gray = [apple_gray]
B_gray = [orange_gray]

LA_color, LB_color, L_color = [], [], []
LA_gray, LB_gray, L_gray = [], [], []
```

ابتدا تصاویر رنگی سیب و پرتقال را از فایل میخوانیم.

تصاویر رنگی را با استفاده از cv.cvtColor به تصاویر خاکستری تبدیل میکنیم.

لیست‌هایA_color ، B_color، A_gray، B_grayرا برای ذخیره پیرامون تصویر رنگی و خاکستری سیب و پرتقال ایجاد میکنیم.

A_colorو B_color پیرامون‌های گوسی برای تصاویر رنگی را ذخیره می‌کنند.

A_gray و B_gray پیرامون‌های گوسی برای تصاویر خاکستری را ذخیره می‌کنند.

لیست‌های(LA_color، LB_color، L_color، LA_gray، LB_gray، L_gray) برای ذخیره لاپلاسین پیرامیدهای تصویر رنگی و خاکستری تعریف شده‌اند.

```
n = 3

# Generate Gaussian and Laplacian pyramids for color Images
for i in range(1, n + 1):
    A_color.append(cv.pyrDown(A_color[-1]))
    B_color.append(cv.pyrDown(B_color[-1]))

for i in range(n):
    # Resize color images to match dimensions before subtraction
    h, w, _ = A_color[i].shape
    A_resized_color = cv.resize(A_color[i + 1], (w, h))

    h, w, _ = B_color[i].shape
    B_resized_color = cv.resize(B_color[i + 1], (w, h))

    # Calculate Laplacian pyramids for color
    LA_color.append(cv.subtract(A_color[i], A_resized_color))
    LB_color.append(cv.subtract(B_color[i], B_resized_color))

# Append the last level of the Gaussian pyramid to complete the Laplacian pyramid for color
LA_color.append(A_color[n])
LB_color.append(B_color[n])
```

تعداد سطوح مورد نظر (**n**) را تکرار میکنیم و برای هر سطح:

این حلقه‌ها پیرامیدهای گوسی (A_color و B_color) و لاپلاسین (LA_color و LB_color) تصویر رنگی را ایجاد می‌کنند. پیرامید گوسی با کم کردن نمونه‌ها (cv.pyrDown()) ایجاد می‌شود و پیرامید لاپلاسین با کم کردن تصویر تغییر اندازه یافته از تصویر اصلی در هر سطح ایجاد می‌شود.

پس از حلقه، آخرین سطح پیرامون گوسی به پیرامون لاپلاسیان برای هر دو تصویر اضافه میکنیم.

```
# Combine Laplacian pyramids to reconstruct the blended image for color
for la_color, lb_color in zip(LA_color, LB_color):
    min_height = min(la_color.shape[0], lb_color.shape[0])
    la_color = la_color[:min_height, :]
    lb_color = lb_color[:min_height, :]
    h, w, c = la_color.shape
    L_color.append(np.hstack([lb_color[:, :w // 2], la_color[:, w // 2:]]))

B_color = L_color[n]
for i in range(n, 0, -1):
    h, w, _ = L_color[i - 1].shape
    B_color = cv.add(cv.pyrUp(B_color, dstsize=(w, h)), L_color[i - 1])
```

این بخش پیرامیدهای لاپلاسین را ترکیب می‌کند تا تصویر رنگی بازسازی شود.

برای هر سطح از پیرامید لاپلاسین ، ابعاد تصویر تنظیم میکنیم و سپس آن‌ها با استفاده از np.hstack() و cv.add() ترکیب میکنیم. ([:, 2 // w:]برای این است که در ترکیب نصف سیب و نصف پرتقال را داشته باشیم)

```
# Generate Gaussian and Laplacian pyramids for Grayscale Images
for i in range(1, n + 1):
    A_gray.append(cv.pyrDown(A_gray[-1]))
    B_gray.append(cv.pyrDown(B_gray[-1]))

for i in range(n):
    # Resize grayscale images to match dimensions before subtraction
    h, w = A_gray[i].shape
    A_resized_gray = cv.resize(A_gray[i + 1], (w, h))

    h, w = B_gray[i].shape
    B_resized_gray = cv.resize(B_gray[i + 1], (w, h))

    # Calculate Laplacian pyramids for grayscale
    LA_gray.append(cv.subtract(A_gray[i], A_resized_gray))
    LB_gray.append(cv.subtract(B_gray[i], B_resized_gray))

# Append the last level of the Gaussian pyramid to complete the Laplacian pyramid for grayscale
LA_gray.append(A_gray[n])
LB_gray.append(B_gray[n])
```

مانند تصاویر رنگی، این بخش پیرامیدهای گوسی(A_gray و B_ gray) و لاپلاسین(LA_ gray و LB_ gray) تصویر خاکستری را ایجاد میکنیم.

پس از حلقه، آخرین سطح پیرامون گوسی به پیرامون لاپلاسیان برای هر دو تصویر اضافه میکنیم.

```
# Combine Laplacian pyramids to reconstruct the blended image for grayscale
for la_gray, lb_gray in zip(LA_gray, LB_gray):
    min_height = min(la_gray.shape[0], lb_gray.shape[0])
    la_gray = la_gray[:min_height, :]
    lb_gray = lb_gray[:min_height, :]
    h, w = la_gray.shape
    L_gray.append(np.hstack([lb_gray[:, :w // 2], la_gray[:, w // 2:]]))

B_gray = L_gray[n]
for i in range(n, 0, -1):
    h, w = L_gray[i - 1].shape
    B_gray = cv.add(cv.pyrUp(B_gray, dstsize=(w, h)), L_gray[i - 1])
```

مشابه تصاویر رنگی، پیرامون‌های لاپلاسیان را ادغام کرده و تصویر ترکیب شده خاکستری را بازسازی میکنیم.

```
# Display the images
plt.figure(figsize=(15, 8))

# Color Images
plt.subplot(231), plt.imshow(cv.cvtColor(apple_color, cv.COLOR_BGR2RGB)), plt.title('RGB: apple'), plt.axis('off')
plt.subplot(232), plt.imshow(cv.cvtColor(orange_color, cv.COLOR_BGR2RGB)), plt.title('RGB: orange'), plt.axis('off')
plt.subplot(233), plt.imshow(cv.cvtColor(B_color, cv.COLOR_BGR2RGB)), plt.title('RGB: Blended Image'), plt.axis('off')

# Grayscale Images
plt.subplot(234), plt.imshow(apple_gray, cmap='gray'), plt.title('Grayscale: apple'), plt.axis('off')
plt.subplot(235), plt.imshow(orange_gray, cmap='gray'), plt.title('Grayscale: orange'), plt.axis('off')
plt.subplot(236), plt.imshow(B_gray, cmap='gray'), plt.title('Grayscale: Blended Image'), plt.axis('off')

# Show the plot
plt.show()
```

از Matplotlib برای نمایش تصویر اصلی سیب رنگی، تصویر اصلی پرتقال رنگی، تصویر ترکیب شده رنگی، تصویر اصلی سیب خاکستری، تصویر اصلی پرتقال خاکستری و تصویر ترکیب شده خاکستری در یک شبکه plt.subplot استفاده میکنیم.

کد کامل :

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

# Read the color images
apple_color = cv.imread('drive/MyDrive/apple.jpg')
orange_color = cv.imread('drive/MyDrive/orange.jpg')

# Convert color images to grayscale
apple_gray = cv.cvtColor(apple_color, cv.COLOR_BGR2GRAY)
orange_gray = cv.cvtColor(orange_color, cv.COLOR_BGR2GRAY)

# Initialize lists to store image pyramids
A_color = [apple_color]
B_color = [orange_color]
A_gray = [apple_gray]
B_gray = [orange_gray]

LA_color, LB_color, L_color = [], [], []
LA_gray, LB_gray, L_gray = [], [], []

n = 3

# Generate Gaussian and Laplacian pyramids for color Images
for i in range(1, n + 1):
    A_color.append(cv.pyrDown(A_color[-1]))
    B_color.append(cv.pyrDown(B_color[-1]))

for i in range(n):
    # Resize color images to match dimensions before subtraction
    h, w, _ = A_color[i].shape
    A_resized_color = cv.resize(A_color[i + 1], (w, h))

    h, w, _ = B_color[i].shape
    B_resized_color = cv.resize(B_color[i + 1], (w, h))

    # Calculate Laplacian pyramids for color
    LA_color.append(cv.subtract(A_color[i], A_resized_color))
    LB_color.append(cv.subtract(B_color[i], B_resized_color))

# Append the last level of the Gaussian pyramid to complete the Laplacian pyramid for color
LA_color.append(A_color[n])
LB_color.append(B_color[n])
```

```
# Combine Laplacian pyramids to reconstruct the blended image for color
for la_color, lb_color in zip(LA_color, LB_color):
    min_height = min(la_color.shape[0], lb_color.shape[0])
    la_color = la_color[:min_height, :]
    lb_color = lb_color[:min_height, :]
    h, w, c = la_color.shape
    L_color.append(np.hstack([lb_color[:, :w // 2], la_color[:, w // 2:]]))

B_color = L_color[n]
for i in range(n, 0, -1):
    h, w, _ = L_color[i - 1].shape
    B_color = cv.add(cv.pyrUp(B_color, dstsize=(w, h)), L_color[i - 1])

# Generate Gaussian and Laplacian pyramids for Grayscale Images
for i in range(1, n + 1):
    A_gray.append(cv.pyrDown(A_gray[-1]))
    B_gray.append(cv.pyrDown(B_gray[-1]))

for i in range(n):
    # Resize grayscale images to match dimensions before subtraction
    h, w = A_gray[i].shape
    A_resized_gray = cv.resize(A_gray[i + 1], (w, h))

    h, w = B_gray[i].shape
    B_resized_gray = cv.resize(B_gray[i + 1], (w, h))

    # Calculate Laplacian pyramids for grayscale
    LA_gray.append(cv.subtract(A_gray[i], A_resized_gray))
    LB_gray.append(cv.subtract(B_gray[i], B_resized_gray))

# Append the last level of the Gaussian pyramid to complete the Laplacian pyramid for grayscale
LA_gray.append(A_gray[n])
LB_gray.append(B_gray[n])

# Combine Laplacian pyramids to reconstruct the blended image for grayscale
for la_gray, lb_gray in zip(LA_gray, LB_gray):
    min_height = min(la_gray.shape[0], lb_gray.shape[0])
    la_gray = la_gray[:min_height, :]
    lb_gray = lb_gray[:min_height, :]
    h, w = la_gray.shape
    L_gray.append(np.hstack([lb_gray[:, :w // 2], la_gray[:, w // 2:]]))

B_gray = L_gray[n]
for i in range(n, 0, -1):
    h, w = L_gray[i - 1].shape
    B_gray = cv.add(cv.pyrUp(B_gray, dstsize=(w, h)), L_gray[i - 1])
```

خروجی کد بالا برای تصاویر رنگی و خاکستری : (n = 3)