



دانشگاه اصفهان

دانشکده مهندسی کامپیوتر – گروه هوش مصنوعی

جبرخطی و کاربردی – نیمسال دوم ۱۴۰۱-۰۲

پروژه پایانی درس جبرخطی و کاربردی

استفاده از تکنیک حذف نویز SVD برای دیتاست Google Scraped Image Dataset

تاریخ : ۱۴۰۲/۴/۰۲

تهیه شده توسط : علی نیک آیین

شماره دانشجویی : ۹۹۳۶۶۳۰۱۰

## خلاصه پروژه :

---

در این پروژه من ابتدا از دیتاست Google Scraped Image Dataset از عکس های داخل پوشه ی architecture تعداد ۱۰ عکس را انتخاب کردم و آن ها را در برنامه لود کردم و سپس یک نویز گوسی رندوم بر روی آن ها اعمال کردم و در نهایت با استفاده از الگوریتم SVD عکس ها را نویز زدایی کردم.

هر بار برای هر عکس سه حالت آن نشان داده میشود :

حالت اول عکس اصلی

حالت دوم عکس نویز دار شده

حالت سوم عکس نویز زدایی شده

سپس همین روند برای دیگر عکس ها انجام میشود و عکس های نویز دار شده در یک فولدر ذخیره میشوند و عکس های نویز زدایی شده نیز در فولدر دیگری ذخیره میشوند.

## برنامه به سه قسمت تقسیم میشود :

قسمت اول بارگزاری هر عکس و آماده کردن آن جهت اعمال نویز بر روی آن ها و نمایش آن

قسمت دوم اعمال نویز گوسی بر روی هر عکس با استفاده از توابع نوشته شده و نمایش آن

قسمت سوم نویز زدایی هر عکس با استفاده از توابع نوشته شده و نمایش آن

## قسمت اول : بارگزاری عکس ها

---

با استفاده از توابع نوشته شده در فایل `Data_loader.py` ما عکس ها را در برنامه بارگزاری میکنیم و در نهایت آن ها را در یک پنجره نمایش میدهیم

در تابع `main` با صدا زدن تابع `get_data()` و درخواست به دریافت سه ماتریس که در اصل سه کانال هر عکس هستند میکنیم

در تابع `get_data()` ابتدا با صدا زدن تابع `check_image_size` ابتدا بررسی میکنیم که آیا عکس به درستی بارگزاری شده است یا `None` برگردانده است.

سپس بررسی میکنیم که ابعاد عکس دریافت شده `(128,128,3)` باشد

اگر نبود ابعاد آن عکس را به ابعاد مدنظر تغییر میدهیم، سپس بررسی میکنیم که نوع عکس `uint8` باشد ، اگر نبود آن را به نوع `md` نظر تبدیل میکنیم و در نهایت عکس را برمیگردانیم . قابل توجه است که در هر کدام از این مراحل به مشکل برخوردیم پیغام خطا مناسب چاپ میشود.

در ادامه کار تابع `get_data()` ما سه ماتریس از عکس بارگزاری شده استخراج میکنیم که درایه های این سه ماتریس رنگ های قرمز و سبز و آبی پیکسل های تصویر هستند.

در نهایت تابع `get_data()` سه ماتریس `R_img,G_img,B_img` را به تابع `main` برمیگرداند

در تابع `main` یک حلقه ی اجرا داریم که به تعداد عکس های موجود در فایل دیتاست مدنظر ماست اجرا خواهد شد :

داخل حلقه برای هر عکس مقادیر ماتریس های سه کانال عکس در متغیر های `R_img,G_img,B_img` ذخیره میشوند.

## قسمت دوم : ایجاد نویز در عکس

در این بخش در تابع main سه مرتبه و در هر مرتبه تابع noise\_maker را صدا میزنیم و ماتریس یکی از کانال های عکس را برای آن ارسال میکنیم ،

```
R_img, G_img, B_img = Data_loader.get_data(input_file)
# This part is for making Noise on images :
noised_R_img=noise_maker(R_img)
noised_G_img=noise_maker(G_img)
noised_B_img=noise_maker(B_img)

# Convert the lists to NumPy arrays
noised_R_img = np.array(noised_R_img, dtype=np.uint8)
noised_G_img = np.array(noised_G_img, dtype=np.uint8)
noised_B_img = np.array(noised_B_img, dtype=np.uint8)
```

در تابع noise\_maker به صورت دستی دو مقدار mean و std\_dev را که همان سیگما میباشد را تعیین میکنیم و در یک حلقه تو در تو تمام درایه های ماتریس کانال مشخص شده را با یک نویز گاوسی رندوم جمع میکنیم . یعنی در حلقه هر بار برای هر درایه یک مقدار رندوم گاوسی تولید میشود و با مقدار قبلی درایه مد نظر از ماتریس کانال جمع میگردد.

نحوه کار تابع گاوسی :

نام این تابع در برنامه من generate\_non\_uniform\_random میباشد ، در این تابع با گرفتن مقادیر mean و std\_dev ، ابتدا دو عدد رندوم یکپارچه تولید میکند به نام u و v و در ادامه با اعمال یک رابطه ریاضی روی آن دو یک عدد رندوم توزیع غیر یکنواخت ایجاد میکنیم. سپس این مقدار را با مقدار std\_dev که همان سیگما است و نشان میدهد بازه ی اعداد تولید شده چه مقدار نزدیک به مرکز باشد و چه مقدار پراکنده تر باشد ضرب میکنیم و حاصل را با مقدار mean که همان نقطه ای است که میخواهیم عدد رندوم حول آن تولید شود جمع میکنیم . در نهایت مقدار حاصل یک عدد رندوم گاوسی است که برگردانده میشود تا با یکی از درایه های ماتریس مد نظر جمع گردد.

رابطه ریاضی استفاده شده در تابع `generate_non_uniform_random` که برای ساخت یک عدد رندوم توزیع غیر یکنواخت استفاده شده بود برگرفته از روابط ریاضی داخل سایت [https://en.wikipedia.org/wiki/Normal\\_distribution](https://en.wikipedia.org/wiki/Normal_distribution) می باشد که استفاده از آن مجاز بود.

- The **Box-Muller method** uses two independent random numbers  $U$  and  $V$  distributed **uniformly** on  $(0,1)$ . Then the two random variables  $X$  and  $Y$

$$X = \sqrt{-2 \ln U} \cos(2\pi V), \quad Y = \sqrt{-2 \ln U} \sin(2\pi V).$$

در نهایت خروجی های ماتریس های هر کانال عکس که نویز دار شده اند را در سه متغیر :

`noised_R_img`

`noised_G_img`

`noised_B_img`

ذخیره کردیم و آن ها را به یک آرایه از جنس `NumPy` تبدیل کرده و در نهایت با صدا زدن تابع `cv2.merge()` که از توابع آماده پایتون بود این سه ماتریس کانال های عکس را باهم ترکیب کرده تا عکس نویز دار شده را بتوانیم نمایش دهیم ، که با صدا زدن تابع `show_noised_image` یک پنجره به ابعاد `800*600` باز خواهد شد و عکس در این ابعاد نمایش داده خواهد شد.

## قسمت سوم : نویز زدایی یک عکس

در ادامه برنامه در تابع `main` ابتدا یک مقدار دستی به عنوان `threshold` در نظر میگیریم و هر ماتریس کانال عکس را که نویز دار شده است به تابع `svd_denoising` همراه با مقدار `threshold` ارسال میکنیم .

تابع `svd_denoising` ، برای حذف نویز تصویر با استفاده از تجزیه مقادیر منفرد عمل می کند. این تابع تصویر ورودی را گرفته و پس از انجام محاسبات مربوط به تجزیه مقادیر منفرد، تصویر را تمیز می کند و بازسازی می کند.

عملکرد تابع به این صورت است:

تابع `svd_denoising` یک تصویر و (`threshold`) را به عنوان ورودی می گیرد. تصویر ورودی را به نوع داده `float32` تبدیل می کند تا بتواند با استفاده از `SVD` محاسبه شود. با استفاده از تابع `calculate_U_s_Vt` ، تجزیه مقادیر منفرد (`SVD`) را بر روی تصویر ورودی انجام می دهد. این تابع ماتریس های `U` ، `s` و `Vt` را برمی گرداند که به ترتیب بردارهای منفرد چپ، مقادیر منفرد و ماتریس  $V^T$  را نشان می دهند.

با استفاده از تابع `np.where` ، مقادیر منفرد را با مقدار `threshold` مقایسه کرده و در صورتی که بزرگتر از آستانه باشند، اون ها رو به عنوان مقادیر منفرد تمیز شده قرار میده. در غیر این صورت، مقدار صفر رو قرار میده.

با استفاده از ضرب ماتریسی و تابع `np.diag` ، تصویر نویز زدایی شده را بازسازی می کند. این عمل ریاضی شامل ضرب ماتریس های `U` ، مقادیر منفرد تمیز شده و ماتریس  $V^T$  است.

تصویر نویز زدایی را به نوع داده uint8 تبدیل می‌کند و مقادیری که کمتر از صفر یا بیشتر از 255 باشند، را به بیشینه یا کمینه مقدار مجاز (0 و 255) محدود می‌کند.

و در نهایت هم تصویر نویز زدایی شده را برمی‌گرداند.

### نحوه کار تابع `calculate_U_s_Vt` :

اگر ماتریس ورودی را  $A$  در نظر بگیریم ، نحوه عملکرد SVD به اینصورت خواهد بود :

$$A = U * S * V^T$$

در تابع داده شده، ماتریس  $U$  بردارهای منفرد چپ ماتریس ورودی را نشان می‌دهد.

عملکرد تابع به این صورت است:

ماتریس  $A$  را با ضرب داخلی ماتریس ورودی با ترانپوز آن محاسبه می‌کند و در متغیر  $A$  ذخیره می‌کند.

با استفاده از تابع `np.linalg.eig` از کتابخانه NumPy، مقادیر و بردارهای ویژه ماتریس  $A$  را محاسبه می‌کند و در متغیرهای `eigenvalues` و  $U$  ذخیره می‌کند.

برای مرتب‌سازی مقادیر ویژه به ترتیب نزولی، اندیس‌های مرتب‌سازی شده مقادیر ویژه را با استفاده از `np.argsort(eigenvalues)[::-1]` محاسبه کرده و در متغیر `eigenvalues_sorted_index` ذخیره می‌کند.

سپس مقادیر ویژه را به ترتیب مرتب‌سازی شده در متغیر `eigenvalues_sorted` ذخیره می‌کند.

بردارهای منفرد چپ را با استفاده از اندیس‌های مرتب‌سازی شده مقادیر ویژه، از ماتریس  $U$  استخراج می‌کند و در متغیر  $U$  ذخیره می‌کند.

مقادیر منفرد را با استفاده از ریشه مربع مقادیر ویژه مرتب‌سازی شده محاسبه کرده و در متغیر `singular_values` ذخیره می‌کند.

ماتریس `Vt` را با استفاده از فرمول `(matrix.T @ U) / singular_values` محاسبه می‌کند و در متغیر `Vt` ذخیره می‌کند.

در نهایت، ماتریس `U`، مقادیر منفرد و ماتریس `Vt.T` را برگردانده و باز می‌گرداند.

من پس از اجرا کردن چندین بار کد به این نتیجه رسیدم که بهترین مقادیر متغیرهای زیر به صورت زیر می‌باشد:

---

`mean = 0`

`std_dev = 10`

`threshold = 310`

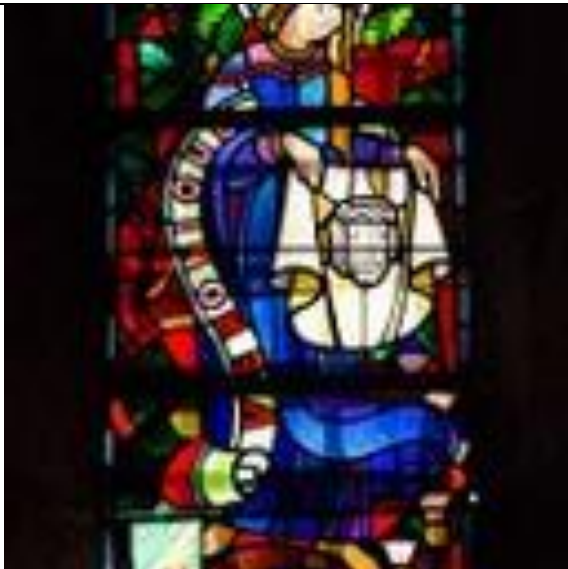
- 
- **توجه:** من یک تابع به اسم `calculate_eigenvalues` در کدم نوشتم که تمام کاری که تابع `np.linalg.eig` انجام میداد را انجام میداد، اما در مقادیر خروجی که من از این تابع می‌گرفتم خطایی وجود داشت که با بررسی‌های فراوان و دیباگ کردن نتوانستم آن را حل کنم و در نهایت برای اینکه از کد خروجی بگیرم از همان تابع `np.linalg.eig` استفاده کردم با این حال تابع جایگزین آن که همان تابع `calculate_eigenvalues` میباشد را پیاده‌سازی کردم و در کدها قابل مشاهده است منتها مقادیری که برمی‌گرداند کاملاً صحیح نیستند.
-



خروجی ده عکس :

			عکس اصلی :
			عکس نویز دار شده :
			عکس نویز زدایی شده :

عکس اصلی :



عکس نويز دار شده :





عکس نویز زدایی شده :









عکس اصلی :






عکس نویز دار شده :

	<p>عکس نويز زدایي شده :</p>
---	-----------------------------

	<p>عکس اصلي :</p>
	<p>عکس نويز دار شده :</p>
	<p>عکس نويز زدایي شده :</p>

	<p>عکس اصلی :</p>
	<p>عکس نویز دار شده :</p>
	<p>عکس نویز زدایی شده :</p>

	<p>عکس اصلی :</p>
	<p>عکس نويز دار شده :</p>
	<p>عکس نويز زدایی شده :</p>



عکس اصلی :


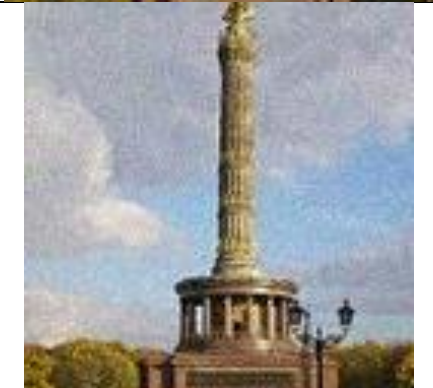
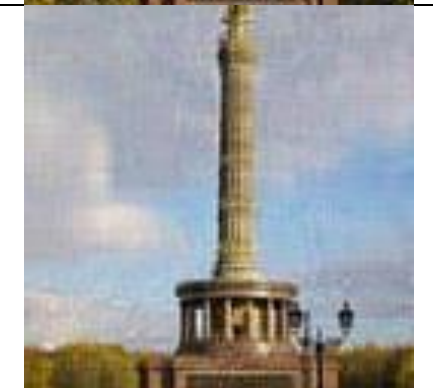


عکس نویز دار شده :



عکس نویز زدایی شده :

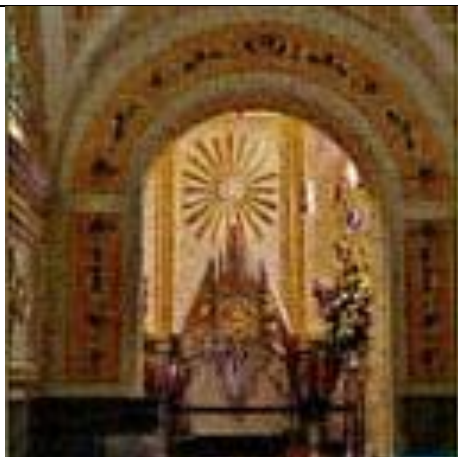


		عکس اصلی :
		عکس نویز دار شده :
		عکس نویز زدایی شده :

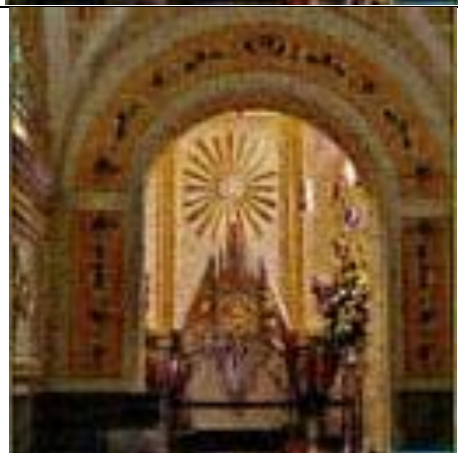
	عکس اصلی :
---	------------



عکس نويز دار شده :



عکس نويز زدایی شده :



عکس اصلی :



	<p>عکس نویز دار شده :</p>
	<p>عکس نویز زدایی شده :</p>

نمونه عکس ها در پوشه های Noised و DeNoised وجود دارند که میتوانید آن ها را بهتر بررسی کنید .  
با تشکر از شما.