# FINAL PROJECT

کیان مجلسی

۹۹۳۶۱۳۰۵۱

**جبرخطی کاربردی**

دکتر پیمان ادیبی

## کتابخانه‌های استفاده شده:

- **کتابخانه numpy:** از توابع این کتابخانه جهت انجام عملیات‌های ریاضی استفاده شده است.
- **کتابخانه Pillow:** از توابع این کتابخانه جهت خواندن، نوشتن و نمایش دادن عکس‌ها استفاده شده است.
- **کتابخانه os:** از توابع این کتابخانه جهت کار با فایل‌ها استفاده شده است.

---

## توضیح قسمت‌های کد:

ابتدا به کمک بسته Kaggle و API دریافت شده از سایت آن، اقدام به دانلود دیتاست duttadebadri/image-classification می‌کنیم. فایل zip شده دیتاست را از حالت آرشیو خارج کرده و پس از آن عملیات‌های موردنظرمان را بر روی عکس‌ها انجام می‌دهیم.

- تابع **generate_gaussian**: در این تابع، برای تولید یک عدد تصادفی بر روی توزیع نرمال از روش تبدیل باکس-مولر استفاده کرده‌ایم. برای انجام این کار ابتدا دو عدد تصادفی با توزیع یکنواخت در بازه ۰ تا ۱ انتخاب کرده ($u_1$, $u_2$) و آن‌ها را در فرمول زیر قرار می‌دهیم:

$$Z = \sqrt{-2\ln{(U_1)}}\cos(2\pi U_2)$$

سپس $Z$ را با میانگین جمع و در انحراف معیار ضرب می‌کنیم تا عدد تصادفی بر روی توزیع گاوسی با میانگین و انحراف معیار داده شده به‌دست آید.

- تابع **add_gaussian_noise_to_image**: در این تابع، عملیات اضافه کردن نویز به تصویر انجام می‌شود. ابتدا به ازای هر پیکسل در کانال‌های عکس (R, G, B)، یک عدد رندوم بر روی توزیع نرمال و با پارامترهای داده شده ایجاد می‌شود. سپس با توجه به فرمول Additive Noise Model این نویز با تصویر جمع می‌شود. همچنین در اینجا لازم است تا مطمئن شویم مقدار هیچ یک از درایه‌های ماتریس از ۲۵۵ بیشتر و از ۰ کمتر نشود. هرچه میزان واریانس بیشتر باشد، شدت نویز بالاتر است و هرچه مقدار میانگین بالاتر رود، شدت روشنایی تصویر بالاتر می‌رود.

$$w(x,y) = s(x,y) + n(x,y)$$

در این رابطه، *w* مقدار نهایی هر پیکسل، *s* مقدار اصلی پیکسل و *n* میزان نویز را مشخص می‌کند.

- تابع **power_iteration**: در این تابع، از روش *power iteration* جهت محاسبه بزرگ‌ترین بردار و مقدار ویژه استفاده کرده‌ایم. برای این کار، ابتدا یک بردار *n*تایی را به صورت رندوم ایجاد می‌کنیم (*b*). سپس با روش تکرار و تا زمان همگرا شدن بردار *b*، حاصل ضرب داخلی Ab را حساب کرده و با تقسیم بر نرمش آن را نرمال می‌کنیم و در هر تکرار مقدار *b* را برابر با این عبارت قرار می‌دهیم. اثبات همگرا شدن این روش در لینکی که در منابع قرار داده شده، آمده است.

$$b_k = \lambda b_k \rightarrow A b_k {b_k}^T = \lambda b_k {b_k}^T \rightarrow \lambda = \frac{A b_k {b_k}^T}{b_k {b_k}^T}$$

- تابع **power_iteration_deflation**: با توجه به اینکه تابع *power_iteration* در هر مرحله تنها بزرگترین مقدار و بردار ویژه را به ما می‌داد، برای محاسبه تمامی بردارها و مقدارهای ویژه یک ماتریس باید از تکنیکی به نام *deflation* استفاده کنیم. ابتدا یک بردار *n*تایی و یک ماتریس مربعی *n*n* جهت ذخیره

مقادیر و بردارهای ویژه تعریف می‌کنیم. سپس با حلقه‌ای به طول *n*، هر مرتبه تابع *power_iteration* را فراخوانی می‌کنیم و در متغیرهای مربوطه آن را ذخیره می‌کنیم. در نهایت ماتریس را با کم کردن ماتریس رتبه یک تشکیل شده توسط حاصل‌ضرب بیرونی بردار ویژه و خودش کاهش می‌دهیم.

$$A = A - \lambda \times (VV^T)$$

- تابع **svd**: در این تابع، به کمک *power_iteration_deflation*، بردارها و مقدارهای ویژه ماتریس $A^TA$ را محاسبه و سپس مقدارهای ویژه را به صورت نزولی مرتب می‌کنیم. بردارهای ویژه را نیز داخل ماتریس و متناظر با مقدارهای ویژه مرتب شده تغییر مکان می‌دهیم. در نهایت با توجه به فرمول‌های گفته شده در درس، مقدارهای *U, Sigma, V* را محاسبه و برمی‌گردانیم.
- تابع **denoise_image_per_channel**: این تابع با دریافت یک ماتریس و مقدار *k* (تعداد مقادیر منفردی که قرار است نگه داشته شوند)، ابتدا حاصل *SVD* ماتریس داده شده را حساب می‌کند، سپس *k* مقدار آن نگه داشته می‌شوند و با ضرب $U\Sigma V$ به ماتریس حذف نویز شده می‌رسیم. مجددا اینجا اطمینان حاصل می‌کنیم که تمامی درایه‌های ماتریس مقداری بین ۰ تا ۲۵۵ داشته و عددی صحیح باشند.

$$image_{m*n} = U_{m*k}S_{k*k}V^T_{k*n}$$

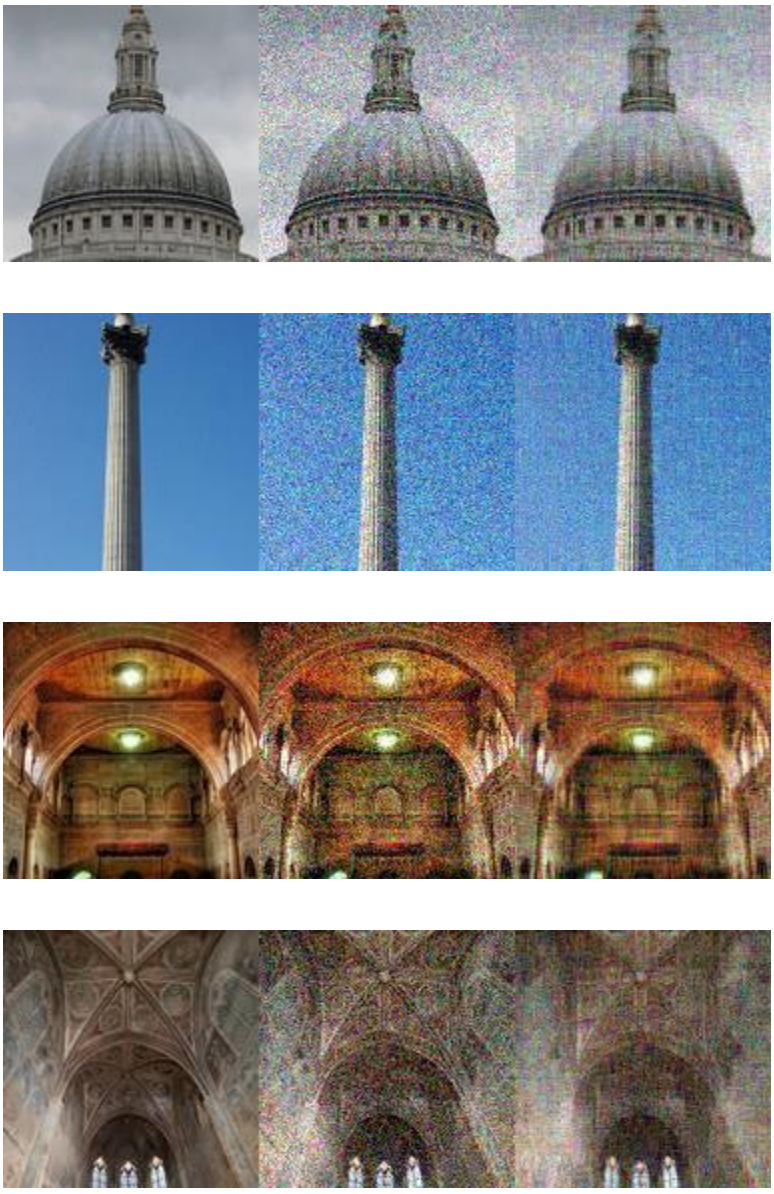
- تابع **denoise_rgb_image**: این تابع پس از جداسازی سه کانال R و G و B، هر کانال را به صورت جداگانه به تابع بالاتر داده تا عملیات حذف نویز انجام شود. در نهایت خروجی هر سه کانال را به کمک تابع merge کتابخانه Pillow با یکدیگر ترکیب می‌کنیم تا به عکس رنگی حذف نویز شده برسیم.
- تابع **get_concat_h**: این تابع با دریافت سه تصویر، آن‌ها را درکنار هم و به‌صورت افقی نمایش می‌دهد.

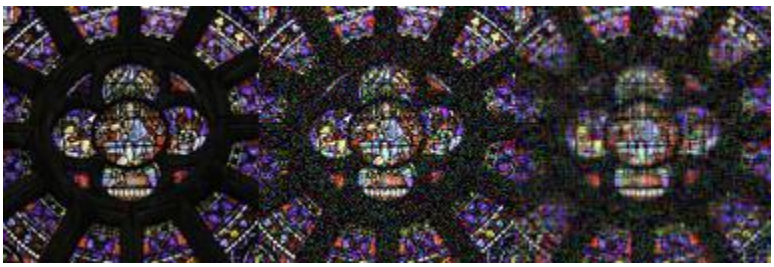
## اجرای الگوریتم:

نمونه‌ای از اجرای الگوریتم بر روی ۱۰ عکس تصادفی از دیتاست با پارامترهای زیر:

**mean=10, standard_deviation=30, k=20**

* به ترتیب از چپ به راست: تصویر اصلی – تصویر دارای نویز گاوسی – تصویر حذف نویز شده





## تاثیر پارامترهای نویز گاوسی:

- **تاثیر میانگین**: مشاهده می‌شود که هرچه پارامتر میانگین در اضافه کردن نویز گاوسی بیشتر شود، شدت روشنایی تصویر بالاتر می‌رود.

  در عکس‌های زیر، پارامتر میانگین از بالا به پایین برابر است با : 20-، 0، 20، 40

  پارامتر انحراف معیار در تمامی تصاویر زیر ثابت و برابر با ۳۰ می‌باشد.

- **تاثیر انحراف معیار**: مشاهده می‌شود که هرچه پارامتر انحراف معیار در اضافه کردن نویز گاوسی بیشتر شود، میزان نویز تصویر بالاتر می‌رود.

در عکس‌های زیر، پارامتر انحراف معیار از بالا به پایین برابر است با : 0، 30، 50، 100

پارامتر میانگین در تمامی تصاویر زیر ثابت و برابر با ۰ می‌باشد.

- **تاثیر تعداد مقادیر منفرد نگه‌داشته شده**: مشاهده می‌شود که هرچه تعداد مقادیر منفرد نگه‌داشته شده بیشتر باشد، میزان حذف نویز کمتر است و جزییات بیشتری از تصویر باقی‌مانده است. و هرچه تعداد مقادیر منفرد نگه‌داشته شده کمتر باشد، میزان حذف نویز بیشتر بوده اما جزییات کمتری از تصویر باقی می‌ماند.

در عکس‌های زیر، پارامتر k از بالا به پایین برابر است با : 1، 5، 20، 50

پارامتر میانگین و انحراف معیار در تمامی تصاویر زیر ثابت و به‌ترتیب برابر با ۰ و ۳۰ می‌باشند.

## منابع جهت ایده:


- Concatenate images with Python, Pillow | note.nkmk.me
- Gaussian Noise | Hasty.ai
- Andy Jones (andrewcharlesjones.github.io)
- تبدیل باکس-مولر - ویکی‌پدیا، دانشنامهٔ آزاد (wikipedia.org)

## لینک نوت‌بوک در کولب:

https://colab.research.google.com/drive/1YR-FJOTUiSpeB9cT81VdlYSDKgTrrVRG?usp=sharing

تیر ۱۴۰۲