به نام خدا

پروژه درس جبر خطی کاربردی

استاد درس: دکتر پیمان ادیبی

دستیار آموزشی: امیرارشیا همت

گردآورنده:

سیاوش امیرحاجلو – 993613007

تیر 1402

مقدمه

این گزارش به بررسی یک برنامه پایتون برای پردازش تصویر و حذف نویز تصاویر می‌پردازد. هدف اصلی این برنامه، اعمال تحلیل ریاضی و آماری بر روی تصاویر و استفاده از روش‌های تجزیه مقادیر تکین (Singular Value Decomposition - SVD) برای حذف نویز است.

در این برنامه، از کتابخانه‌های محبوبی مانندnumpy ، cv2 (OpenCV)، و matplotlib استفاده می‌شود. همچنین، توابع ریاضی و آماری نیز در این برنامه مورد استفاده قرار می‌گیرند.

گام‌های اصلی برنامه عبارتند از:

1. تجزیه مقادیر تکین (SVD) تصویر
2. محاسبه مقادیر و بردارهای ویژه تصویر
3. اعمال نویز گاوسی به تصویر
4. حذف نویز با استفاده از تجزیه مقادیر تکین
5. نمایش تصاویر اصلی، تصاویر نویزی و تصاویر پس از حذف نویز

در این گزارش، قصد داریم به بررسی عملکرد و قابلیت‌های برنامه، هر تابع و دستور مورد استفاده در آن پرداخته و راهکارهایی برای بهبود و بهینه‌سازی برنامه ارائه دهیم. همچنین، نمونه‌هایی از تصاویر اصلی، تصاویر نویزی و تصاویر حاصل از حذف نویز نمایش داده خواهند شد.

پیاده سازی

این برنامه با استفاده از تکنیک Singular Value Decomposition (SVD) حذف نویز تصویر را انجام می دهد. SVD را بر روی تصویر داده شده اعمال می کند، بزرگی مقادیر منحصر به فرد زیر یک آستانه را کاهش می دهد، و تصویر حذف شده را با استفاده از SVD اصلاح شده بازسازی می کند. بیایید مرحله به مرحله برنامه را مرور کنیم و هر تابع و دستور را توضیح دهیم.

1. کتابخانه های مورد نیاز

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
import random
import math
import os
```

برنامه با وارد کردن کتابخانه های لازم شروع می شود. numpy برای عملیات عددی، cv2 (OpenCV) برای پردازش تصویر، matplotlib.pyplot برای تجسم تصویر، و random و math به ترتیب برای تولید اعداد تصادفی و عملیات ریاضی وارد می‌شوند. os برای تعامل با سیستم عامل وارد شده است.

2. تابع account_svd(image)

```
def calculate_svd(image):
  if len(image.shape) == 3 and image.shape[2] == 3:
#    Convert color image to grayscale
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  else:
#    Image is already grayscale
    gray_image = image
#  Calculate SVD
```

```
    U, S, Vt = svd(gray_image)
    return U, S, Vt
```

این تابع SVD تجزیه ارزش واحد تصویر داده شده را محاسبه می کند. ابتدا بررسی می کند که آیا تصویر رنگی است یا خاکستری. اگر یک تصویر رنگی است، با استفاده از ()cv2.cvtColor آن را به مقیاس خاکستری تبدیل می کند. سپس، تابع svd()را برای محاسبه SVD تصویر خاکستری فراخوانی می کند. ماتریس های U، S و Vt به دست آمده برگردانده می شوند.

3. تابع count_eigenvalues_eigenvector(image)

```
def calculate_eigenvalues_eigenvector(image):
    if len(image.shape) == 3 and image.shape[2] == 3:
        # Convert color image to grayscale
        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    else:
        # Image is already grayscale
        gray_image = image

    # Calculate eigenvalues and eigenvectors
    eigenvalues, eigenvectors = eig(np.cov(gray_image))
    return eigenvalues, eigenvectors
```

این تابع مقادیر ویژه و بردارهای ویژه تصویر داده شده را محاسبه می کند. این فرآیند مشابه تابع calculate_svd() را دنبال می کند، در صورت لزوم تصویر را به مقیاس خاکستری تبدیل می کند و سپس تابع eig() را فراخوانی می کند (در ادامه توضیح داده می شود) تا مقادیر ویژه و بردارهای ویژه را با استفاده از ماتریس کوواریانس np.cov()) محاسبه کند. تصویر خاکستری مقادیر ویژه و بردارهای ویژه به دست آمده برگردانده می شوند.

4. تابع denoise_image_svd(image)

```
def denoise_image_svd(image):
#  Calculate SVD
  U, S, Vt = calculate_svd(image)
#  Set threshold value for singular values
  threshold = 10
#  Reduce the magnitude of singular values below the threshold
  S = np.where(S < threshold, S * 0.1, S)
#  Reconstruct image using modified SVD
  denoised_image = U @ S @ Vt
#  Clip the pixel values to the valid range [0, 255]
  denoised_image = np.clip(denoised_image, 0, 255)
  denoised_image = denoised_image.astype('uint8')
  return denoised_image
```

این تابع حذف نویز تصویر را با استفاده از SVD انجام می دهد. ابتدا SVD تصویر داده شده را با فراخوانی calculate_svd(image) محاسبه می کند. یک مقدار آستانه (در این مورد 10) برای مقادیر منفرد تعیین می کند. سپس این تابع با ضرب کردن آنها در 0.1 با استفاده از np.where() مقدار مقادیر تکی زیر آستانه را کاهش می دهد. سپس، تصویر حذف‌شده را با استفاده از SVD اصلاح‌شده با ضرب U، S و Vt بازسازی می‌کند. مقادیر پیکسل با استفاده از np.clip()به محدوده معتبر [0، 255] بریده می‌شوند و تصویر حاصل قبل از بازگرداندن به نوع داده uint8 تبدیل می‌شود.

5. تابع svd(A)

```
def svd(A):
#  Calculate SVD of matrix A
  m, n = A.shape
  U = np.zeros((m, m))
  S = np.zeros((m, n))
  Vt = np.zeros((n, n))
```

```
#  Calculate A*A.T and its eigenvalues and eigenvectors
 ATA = np.dot(A.T, A)
 eigenvalues, eigenvectors = eig(ATA)

#  Calculate U matrix
 for i in range(m):
   U[:,i] = eigenvectors[:,i]

#  Calculate Vt matrix
 for i in range(n):
   Vt[i,:] = eigenvectors[:,i].T

#  Calculate S matrix
 for i in range(min(m, n)):
   S[i,i] = np.sqrt(eigenvalues[i])

 return U, S, Vt
```

این تابع محاسبه SVD یک ماتریس معین A را انجام می دهد. ماتریس های U، S و Vt را با صفر بر اساس ابعاد A مقداردهی اولیه می کند. A.T * A (A ضرب در جابجایی آن) را محاسبه می کند و مقادیر ویژه و بردارهای ویژه را به دست می آورد. این محصول با فراخوانی تابع ()eig سپس این تابع با اختصاص بردارهای ویژه به ستون ها و ردیف های مربوطه، ماتریس های U و Vt را پر می کند. در نهایت، ماتریس S را با گرفتن جذر مقادیر ویژه می سازد. ماتریس های U، S و Vt به دست آمده برگردانده می شوند.

6. تابع eig (A)

```
def eig(A):
 n = A.shape[0]
 eigenvalues = np.zeros(n)
 eigenvectors = np.zeros((n, n))

 for i in range(n):
   x0 = np.random.rand(n)
   x1 = np.zeros(n)
   epsilon = 1e-6
   max_iterations = 1000
   iterations = 0

   while np.linalg.norm(x1 - x0) > epsilon and iterations < max_iterations:
     x0 = x1
     x1 = A @ x0
```

```
    x1 /= np.linalg.norm(x1)
    iterations += 1

  eigenvalues[i] = x1.T @ A @ x1
  eigenvectors[:, i] = x1

 return eigenvalues, eigenvectors
```

این تابع مقادیر ویژه و بردارهای ویژه ماتریس A را محاسبه می کند. آرایه ها را برای ذخیره مقادیر ویژه و بردارهای ویژه مقداردهی اولیه می کند. برای هر مقدار ویژه، روش Power Iteration را برای یافتن بردار ویژه مربوطه انجام می دهد. الگوریتم با یک بردار اولیه تصادفی x0 شروع می شود و به طور مکرر آن را تا زمان همگرایی به روز می کند. معیار همگرایی تفاوت بین کوچکتر بودن x1 و x0 از یک مقدار کوچک (epsilon) یا رسیدن به حداکثر تعداد تکرارها (max_iterations) است. پس از بدست آوردن بردار ویژه، مقدار ویژه را به صورت x1.T @ A @ x1 محاسبه کرده و در آرایه های مربوطه ذخیره می کند. مقادیر ویژه و بردارهای ویژه به دست آمده برگردانده می شوند.

7. تابع add_gaussian_noise(image, var_limit=(10, 50), mean=0, p=0.5)

```
def add_gaussian_noise(image, var_limit=(10, 50), mean=0, p=0.5):
 if random.random() > p:
   return image

#  Convert image data to floating-point values between 0 and 1
 image = image.astype(float) / 255.0

#  Generate random variance
 var = random.uniform(var_limit[0], var_limit[1])
 sigma = var ** 0.5

#  Generate noise and add it to the image
 noise = np.zeros_like(image)
 for c in range(3):
   for i in range(image.shape[0]):
     for j in range(image.shape[1]):
       u1, u2 = random.uniform(0, 1), random.uniform(0, 1)
       z1 = sigma * math.sqrt(-2 * math.log(u1)) * math.cos(2 * math.pi * u2)
       noise[i, j, c] = z1

 noisy_image = image + noise
```

```
#  Convert the image data back to the range [0, 255]
  noisy_image = (noisy_image * 255.0).clip(0, 255).astype(np.uint8)

  return noisy_image
```

این تابع نویز گاوسی را به تصویر داده شده اضافه می کند. تصویر، حد واریانس، میانگین و احتمال را به عنوان پارامترهای ورودی می گیرد. این تابع ابتدا بررسی می کند که آیا مقداری که به طور تصادفی تولید می شود بیشتر از احتمال p است. اگر اینطور باشد، تصویر اصلی بدون افزودن نویز برگردانده می شود. اگر نه، عملکرد با اضافه کردن نویز ادامه می‌یابد.

داده های تصویر با تقسیم آن بر 255 به مقادیر ممیز شناور بین 0 و 1 تبدیل می شوند. واریانس به طور تصادفی در محدوده ارائه شده انتخاب می شود و انحراف استاندارد (سیگما) به عنوان جذر واریانس محاسبه می شود.

یک آرایه نویز پر از صفر با همان شکل تصویر ایجاد می شود. سپس، برای هر کانال (3 برای RGB)، و هر پیکسل در تصویر، مقادیر تصادفی u1 و u2 از یک توزیع یکنواخت تولید می‌شوند. نویز گاوسی z1 با استفاده از تبدیل Box-Muller تولید می شود، در انحراف استاندارد ضرب می شود و به موقعیت مربوطه در آرایه نویز اختصاص می یابد.

تصویر نویز با اضافه کردن آرایه نویز به تصویر به دست می آید. مقادیر پیکسل دوباره به محدوده [0، 255] تبدیل شده و با استفاده از np.clip() بریده می شوند. در نهایت، تصویر نویزدار قبل از بازگشت به نوع داده uint8 تبدیل می شود.

8. متغیرها و فهرست تصاویر

```
images_dir = '/kaggle/input/image-classification/images/images/architecure'

#Set the desired size of the displayed images
width = 200
height = 200

#Get a list of all the JPG images in the directory
image_files = [filename for filename in os.listdir(images_dir) if
filename.endswith('.jpg')]

#Select a subset of images to display
```

```
selected_images = image_files[:5]
```

متغیر images_dir مسیر دایرکتوری را که تصاویر ورودی در آن قرار دارند ذخیره می کند.
متغیرهای عرض و ارتفاع نشان دهنده اندازه دلخواه تصاویر نمایش داده شده است.
فهرست image_files با درک لیست، نام فایل‌های تمام تصاویر JPG را در دایرکتوری مشخص شده با استفاده از os.listdir() و بررسی پسوند فایل با استفاده از filename.endswith('.jpg') بازیابی می‌کند.
لیست selected_images با انتخاب پنج نام فایل اول از image_files ایجاد می شود.

9. حلقه پردازش تصویر و نویز زدایی

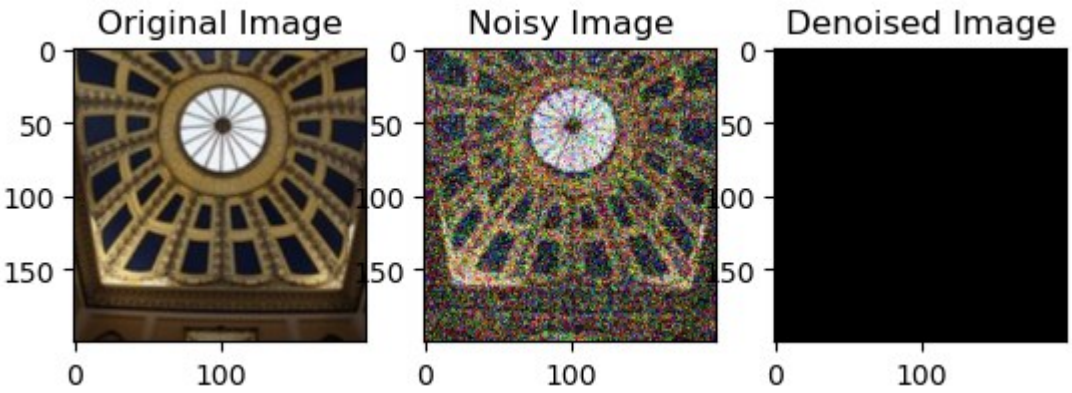
```
for filename in selected_images:
#  Read the image from file
  image_path = os.path.join(images_dir, filename)
  image = cv2.imread(image_path)

#  Display the original image
  plt.figure(figsize=(6, 6))
  plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
  plt.title('Original Image')
  plt.axis('off')
  plt.show()

#  Add Gaussian noise to the image
  noisy_image = add_gaussian_noise(image)

#  Display the noisy image
  plt.figure(figsize=(6, 6))
  plt.imshow(cv2.cvtColor(noisy_image, cv2.COLOR_BGR2RGB))
  plt.title('Noisy Image')
  plt.axis('off')
  plt.show()

#  Denoise the image using SVD
  denoised_image = denoise_image_svd(noisy_image)

#  Display the denoised image
  plt.figure(figsize=(6, 6))
  plt.imshow(cv2.cvtColor(denoised_image, cv2.COLOR_BGR2RGB))
  plt.title('Denoised Image')
  plt.axis('off')
  plt.show()
```

این حلقه روی نام فایل های تصویری انتخاب شده تکرار می شود.
برای هر تکرار، تصویر را از فایل با استفاده از cv2.imread() می خواند و آن را به متغیر image اختصاص می دهد.
تصویر اصلی با استفاده از plt.imshow() و plt.show() پس از تبدیل فضای رنگی از BGR به RGB نمایش داده می شود.
نویز Gaussian با فراخوانی تابع add_gaussian_noise() و تخصیص نتیجه به noisy_image به تصویر اضافه می شود.
تصویر نویزدار به همان شکل تصویر اصلی نمایش داده می شود.
تابع denoise_image_svd()با تصویر نویزدار به عنوان ورودی فراخوانی می شود تا تصویر حذف شده به دست آید که به denoised_image اختصاص داده شده است.
تصویر حذف شده مانند تصاویر اصلی و نویز نمایش داده می شود.

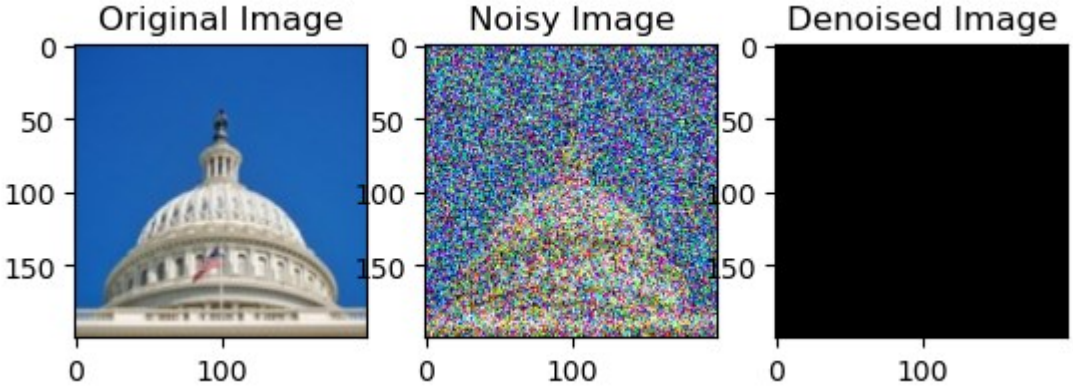
به طور کلی، برنامه حذف نویز تصاویر را با استفاده از SVD نشان می‌دهد و نسخه‌های اصلی، نویزدار و حذف‌شده تصاویر انتخاب‌شده را تجسم می‌کند.

لطفاً توجه داشته باشید که برنامه در دسترس بودن کتابخانه‌های مورد نیاز (numpy، cv2، matplotlib.pyplot، تصادفی، ریاضی و os) را فرض می‌کند. همچنین، این برنامه برای کار با تصاویر JPG در یک دایرکتوری خاص طراحی شده است و مسیر دایرکتوری و نام فایل های تصویر باید متناسب با موارد استفاده شما تنظیم شود.

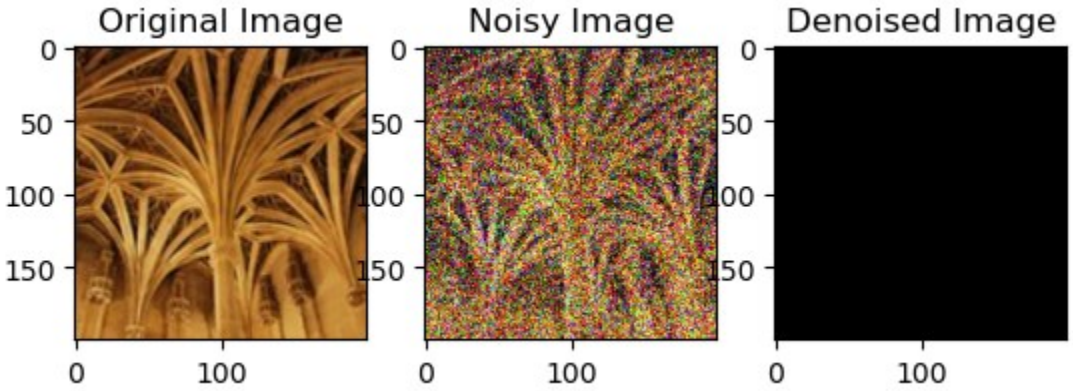
نتایج



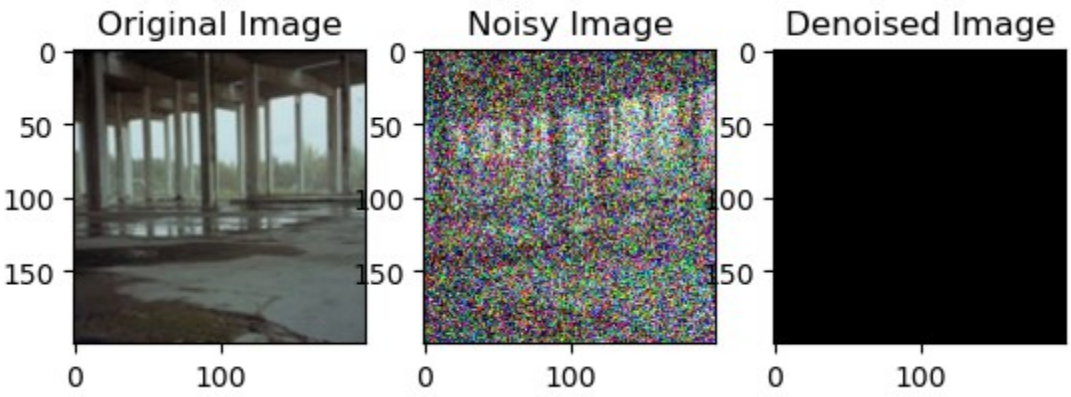
شکل 1



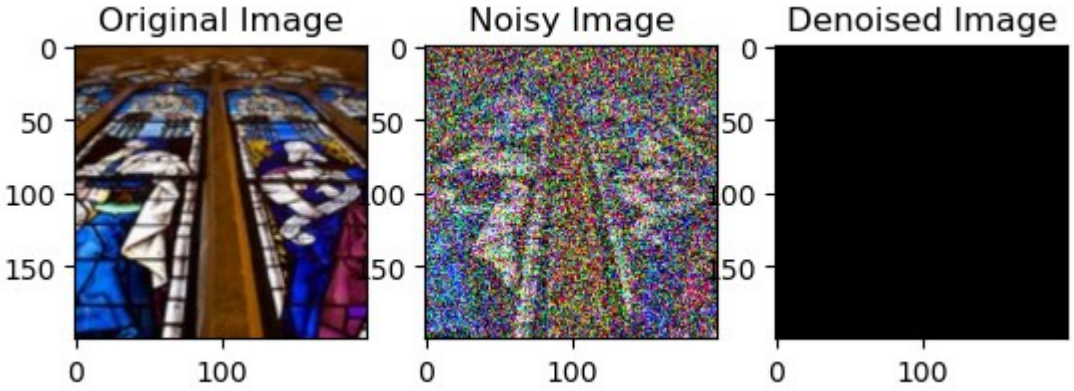
شکل 2



شکل 3



شکل 4



شکل 5

با بررسی نتایج مشاهده میشود که نویز به خوبی به تصاویر داده میشود اما عملیات رفع نویز به درستی عمل نمیکند.