Denoising with SVD

Applied Linear Algebra

Course Instructor: Peyman Adibi
Alireza Dastmalchi Saei
(993613026)

Introduction

In this project the aim is to explore the effectiveness of Singular Value Decomposition (SVD) for denoising images. We will add noise to an image dataset from Kaggle (architecture) and then apply SVD to denoise it with different number of singular values (K).

Code Explanation

First, we import the necessary libraries for this project:

Required libraries

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

Then we use a function to read images from the architecture folder of the dataset in Kaggle:

Function to read images from given path

+ Code + Markdown

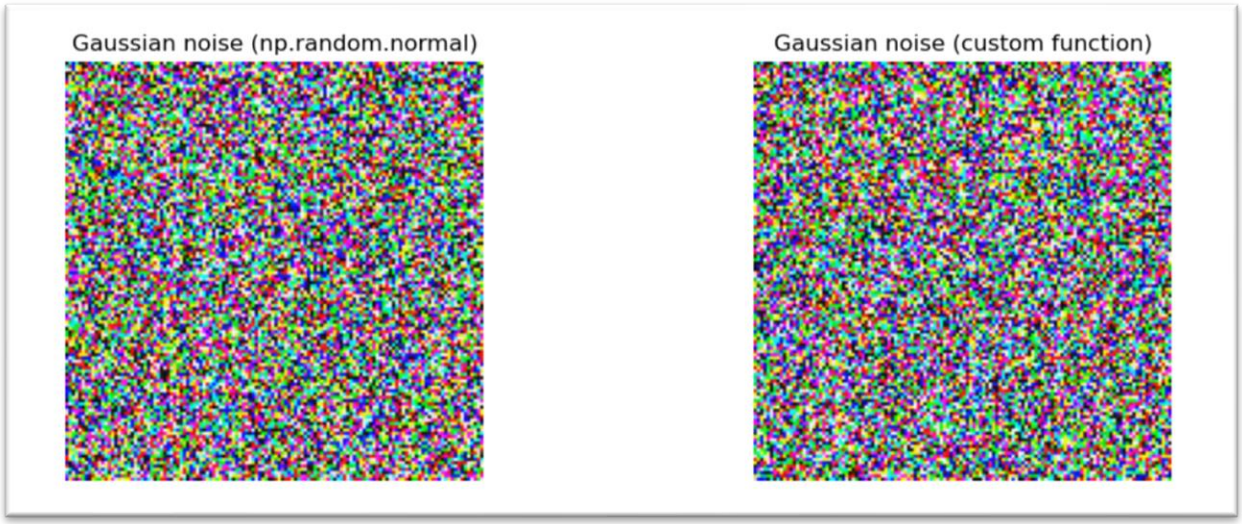
```
def read_images(directory):
    image_data = []
    for file in os.listdir(directory):
        if file.endswith(".jpg"):
            image_path = os.path.join(directory, file)
            image = Image.open(image_path)
            pixel_data = np.array(image)
            image_data.append(pixel_data)
    return image_data

# 128 * 128 images
directory_path = "/kaggle/input/image-classification/images/images/architecure"
image_data_list = read_images(directory_path)
print(f'Number of images: {len(image_data_list)}')
```

```
Number of images: 8763
```

In the next part, a gaussian noise must be added to the picture before denoising it:

```
def gaussian_noise(shape, mean=10, std_dev=10):
    size = np.prod(shape)

    u1 = np.random.uniform(0, 1, size)
    u2 = np.random.uniform(0, 1, size)

    # Apply the Box-Muller transform to generate Gaussian noise
    z = np.sqrt(-2.0 * np.log(u1)) * np.cos(2.0 * np.pi * u2)

    # Reshape the noise array to the desired shape
    z = np.reshape(z, shape)

    # Use standard deviation and mean
    noise = std_dev * z + mean

    return noise.astype(np.uint8)
```

To see gaussian noise works fine we can compare it with "*np.random.normal*":



In the next step, the noise is added to the RGB values of the pictures and it is also checked that the RGB values must in range [0, 255] making the image noisy.

```
def add_gaussian_noise(image):
    mean = random.randint(30, 50)
    std_dev = random.randint(10, 25)
    noise = gaussian_noise(image.shape[:2], mean=mean, std_dev=std_dev)
    noisy_image = np.clip(image + noise[..., np.newaxis], 0, 255).astype(np.uint8)
    return noisy_image
```

Calculating SVD

Two approaches have been chosen to calculate the SVD:

1. QR Decomposition

For this method, first we need to calculate Q function using "*Gram Schmidt*" algorithm.

```
def gram_schmidt(A):
    Q = []
    for j in range(A.shape[1]):
        v = A[:,j]
        for i in range(len(Q)):
            q = Q[i]
            v = v - np.dot(v, q) * q
        if not np.allclose(v, 0):
            v = v / np.linalg.norm(v)
            Q.append(v)
    return np.column_stack(Q)
```

Then, we apply QR decomposition and return Q and R:

```
def QR(M):
    Q = gram_schmidt(M)
    R = np.dot(Q.T, M)
    return Q, R
```

In the next step, we calculate eigen values and eigen vectors by initializing them and then updating in a for loop after "max_iteration". Both variables are returned after loop.

At last, in the "*svd*" function U and Sigma and V are calculated using eigen function. These variables are later used in denoising images.

```
def svd(A):
    # Compute eigenvalues and eigenvectors of A^T.A
    ATA = np.dot(A, A.T)
    eigenvalues, V = eig(ATA)

    # Sort eigenvalues in descending order
    sort_indices = np.argsort(eigenvalues)[::-1]
    eigenvalues = eigenvalues[sort_indices]
    V = V[:, sort_indices]

    # Compute singular values and U
    S = np.sqrt(np.abs(eigenvalues))
    U = np.dot(A, V) / S

    return U, S, V.T
```

To check if svd function works well, we can check with the below code:

```
# Implemented SVD
U, SV, V = svd(noisy_images[0][:,:,0])
reconstructed_A = U @ np.diag(SV) @ V
# print(f'U:\n{U}\nSV:\n{SV}\nEigen:\n{Eigen}')

# SVD using np.linalg.svd
U, SV, V = np.linalg.svd(noisy_images[0][:,:,0])
reconstructed_A_with_np = U @ np.diag(SV) @ V

# Check if implemented SVD works fine
print(np.allclose(reconstructed_A, reconstructed_A_with_np))
```

```
True
```

2. **Power Iteration**

   The second method to calculate SVD is the “Power Iteration” method that the code of this part has been gathered from the website: https://stevealbertwong.github.io/2016/12/23/SVD/

Denoising Images

After implementing required function, some functions are defined to get a noisy image and denoise it. First one, is a function that takes an image and a parameter k as input, and applies a denoising algorithm to each of the RGB color channels of the image. Once all of the color channels have been denoised, the function merges them back together into an RGB image and then the image is clipped to the valid range of pixel values (between 0 and 255) and converted to the uint8 data type before being returned:

```python
def denoise_image(image, k):
    # Split the image into RGB channels
    red_channel, green_channel, blue_channel = split_channels(image)

    # Denoise each channel
    denoised_red = denoise_channel(red_channel, k)
    denoised_green = denoise_channel(green_channel, k)
    denoised_blue = denoise_channel(blue_channel, k)

    # Combine the denoised channels into an RGB image
    denoised_image = cv2.merge((denoised_red, denoised_green, denoised_blue))

    # Clip the values to the valid range [0, 255] and convert to uint8
    denoised_image = np.clip(denoised_image, 0, 255).astype(np.uint8)

    return denoised_image
```

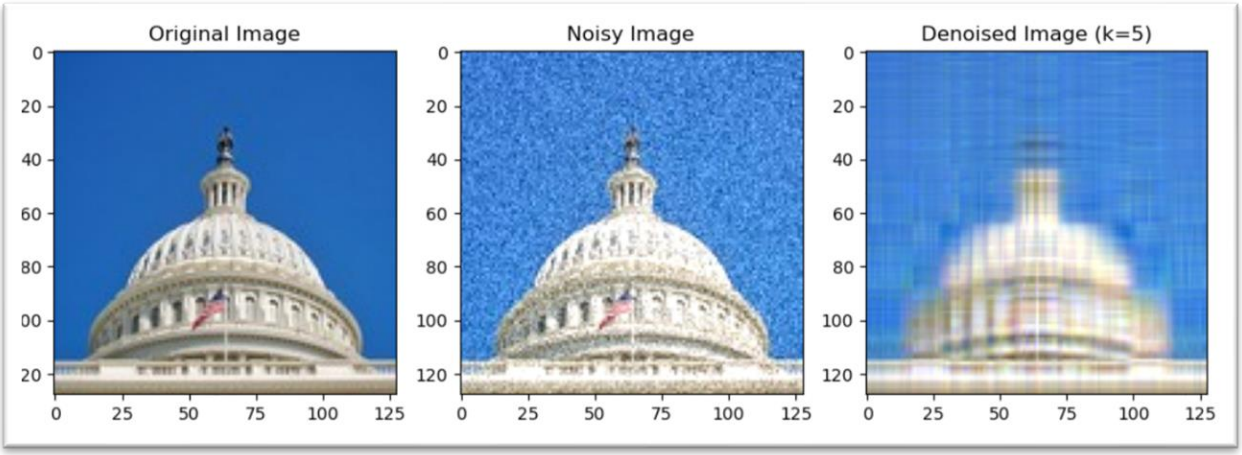
In the previous function, "*Denoise_channel*" function is called for each channel of image (can be called with any of implemented SVDs). This function reconstructs channels only with first k singular values:

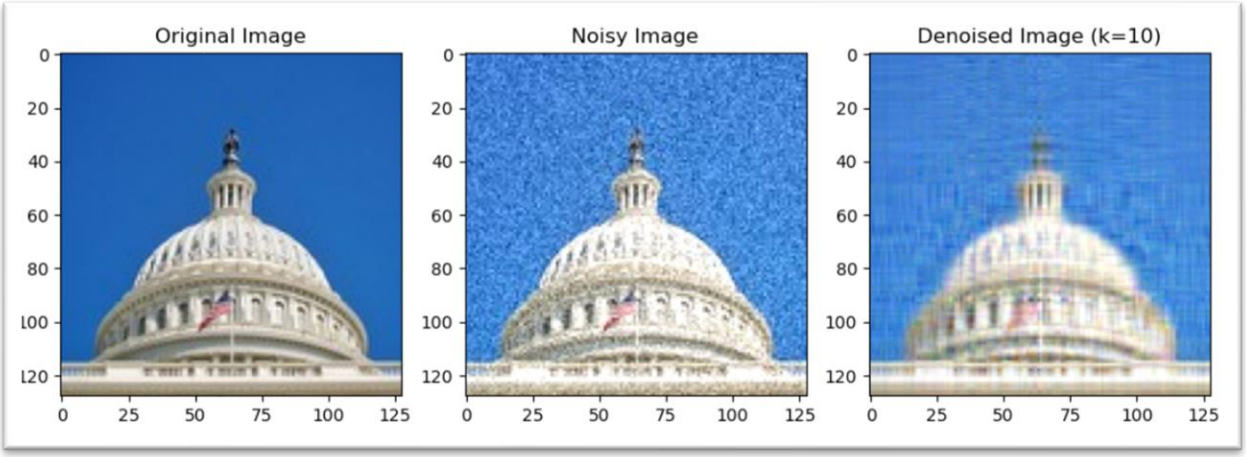
```python
def denoise_channel(channel, k):
    channel = channel.astype(float)
    # U, s, V = svD(channel)

    U, s, V = svd(channel)
    S = np.zeros((channel.shape[0], channel.shape[1]))
    S[:channel.shape[0], :channel.shape[0]] = np.diag(s)
    A = U.dot(S[:, :k].dot(V[:k, :]))
    return A
```

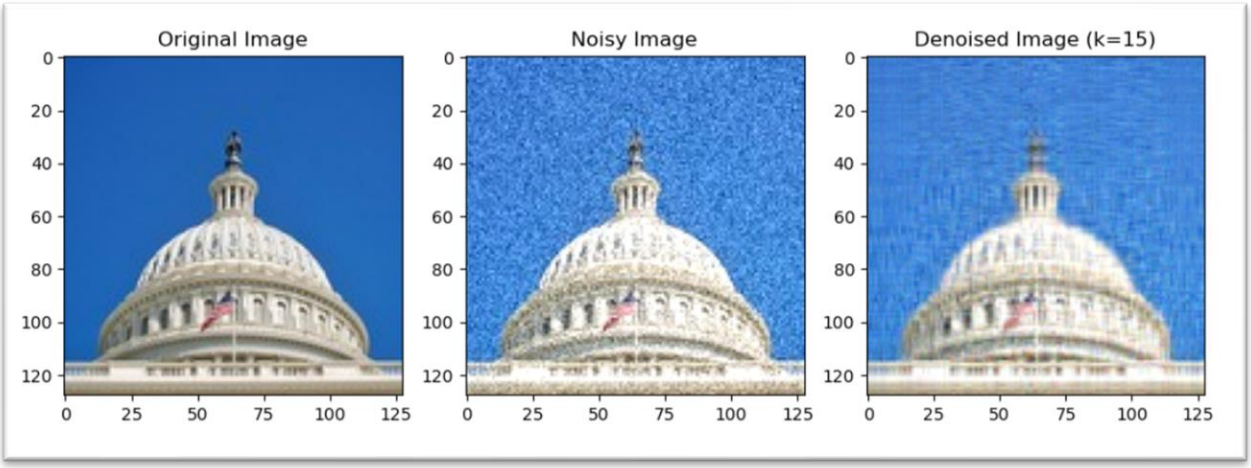
In the final step, we must apply these functions on the images with different values of K to see the effect of it on denoising:

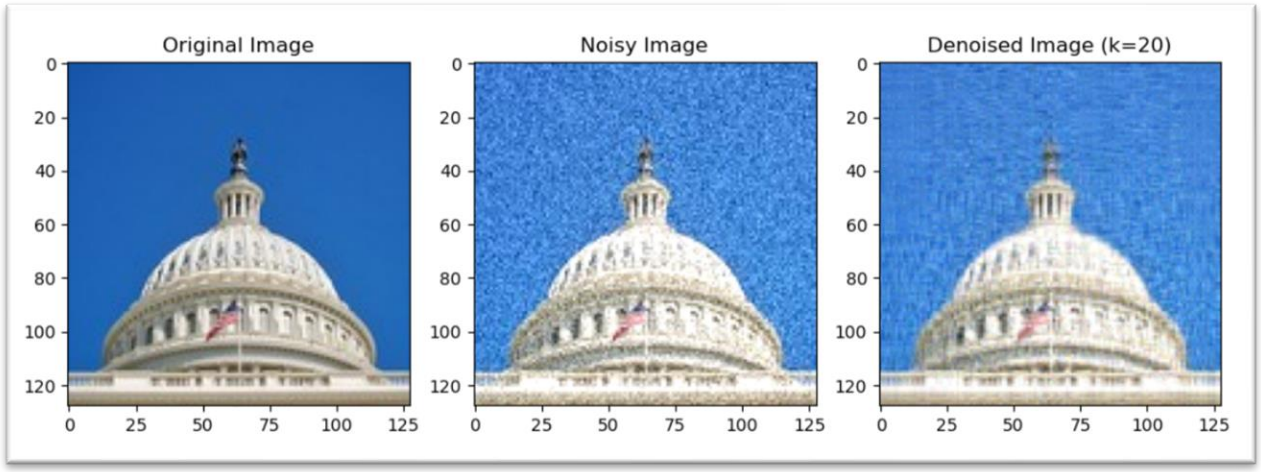
```
# Value of k
k = 20
img_number = 0

# Plot the denoised images for each k value
fig, axs = plt.subplots(1, 3, figsize=(10, 10))

# Choose noisy image
noisy_image = noisy_images[img_number]

# Denoise the image using SVD
denoised_image = denoise_image(noisy_image, k)

# Plot the images
axs[0].imshow(first_10_images[img_number])
axs[0].set_title("Original Image")
axs[1].imshow(noisy_image)
axs[1].set_title("Noisy Image")
axs[2].imshow(denoised_image)
axs[2].set_title(f"Denoised Image (k={k})")

plt.tight_layout()
plt.show()
```

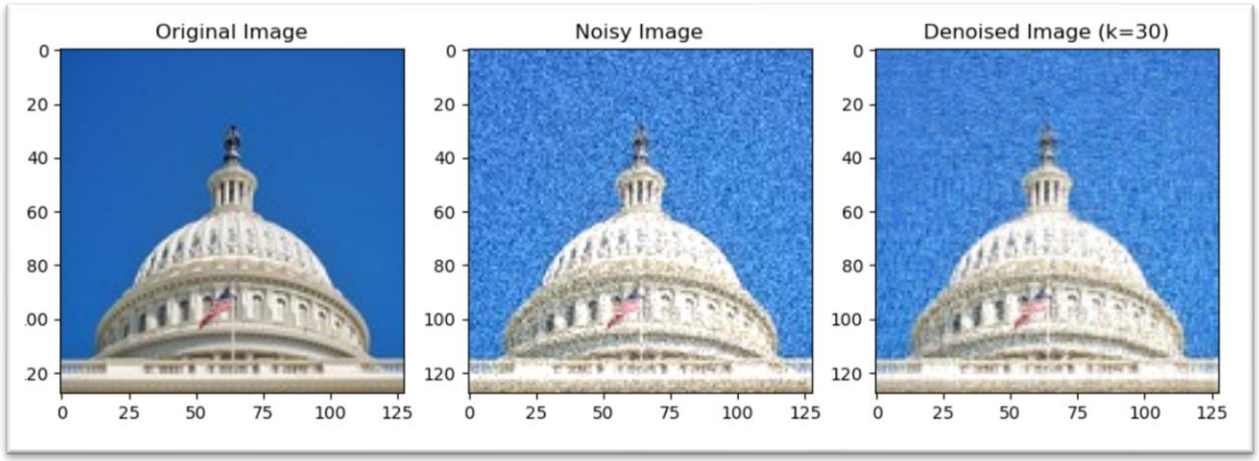
Results for different values of K

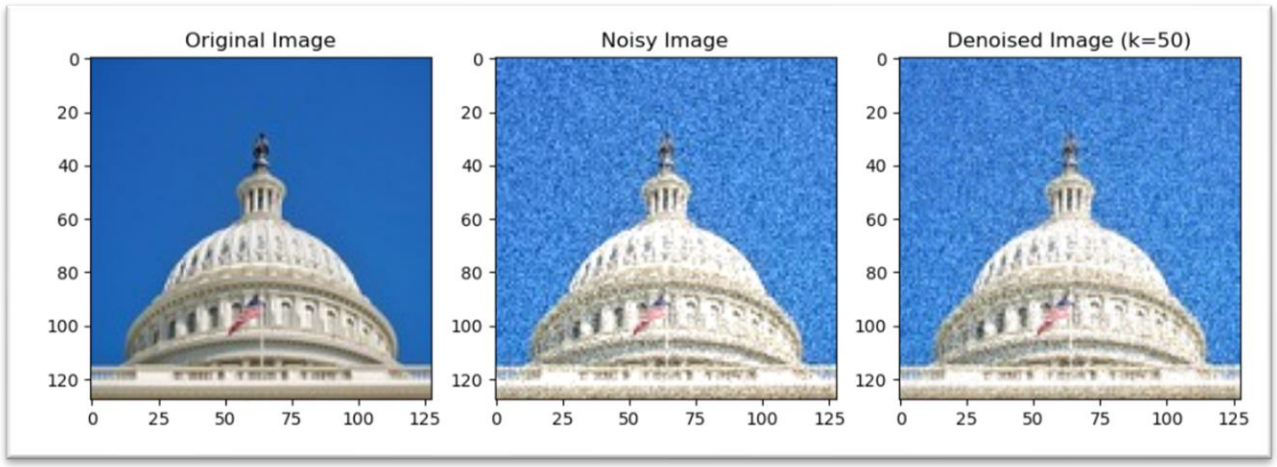




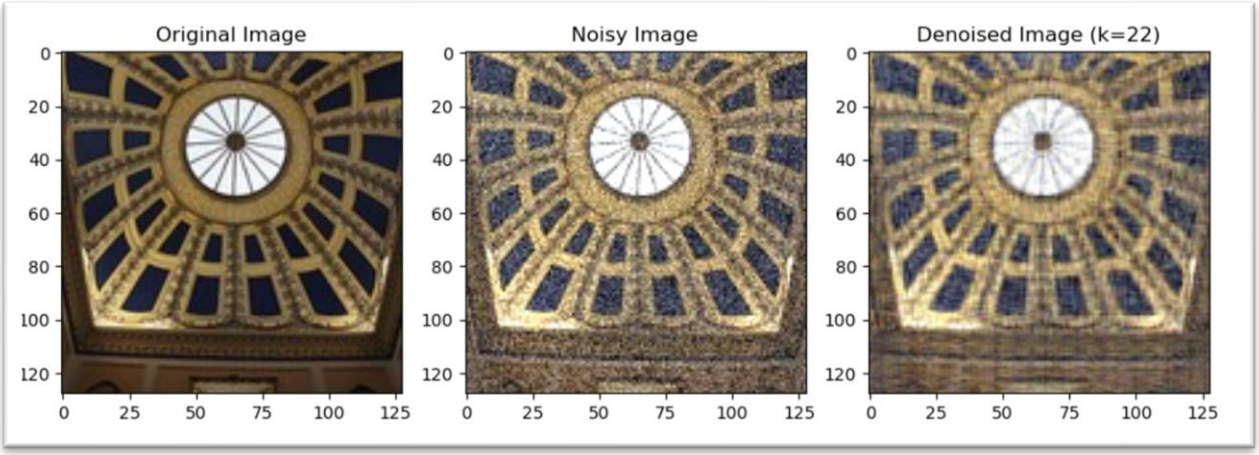


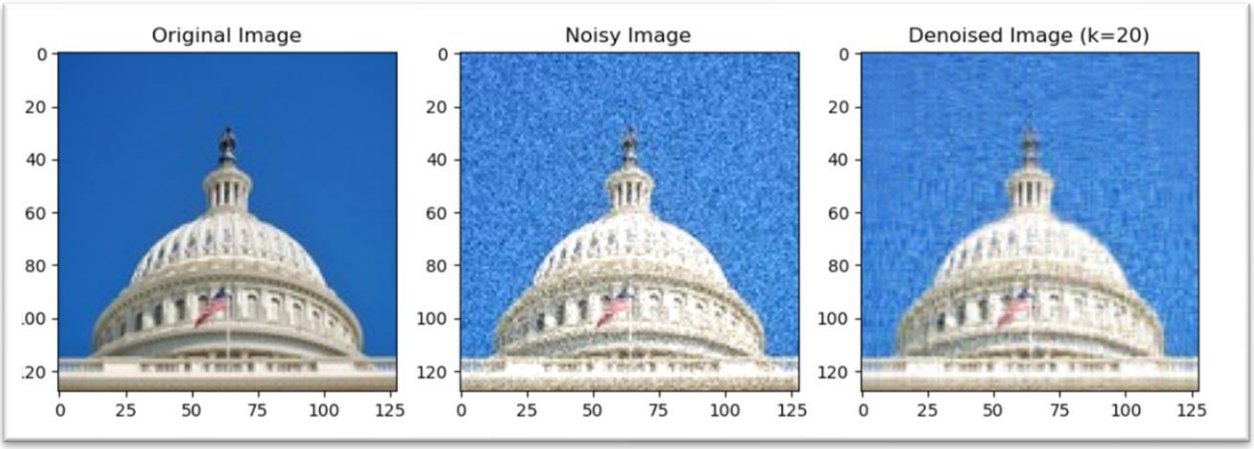

Original Image
Noisy Image
Denoised Image (k=20)

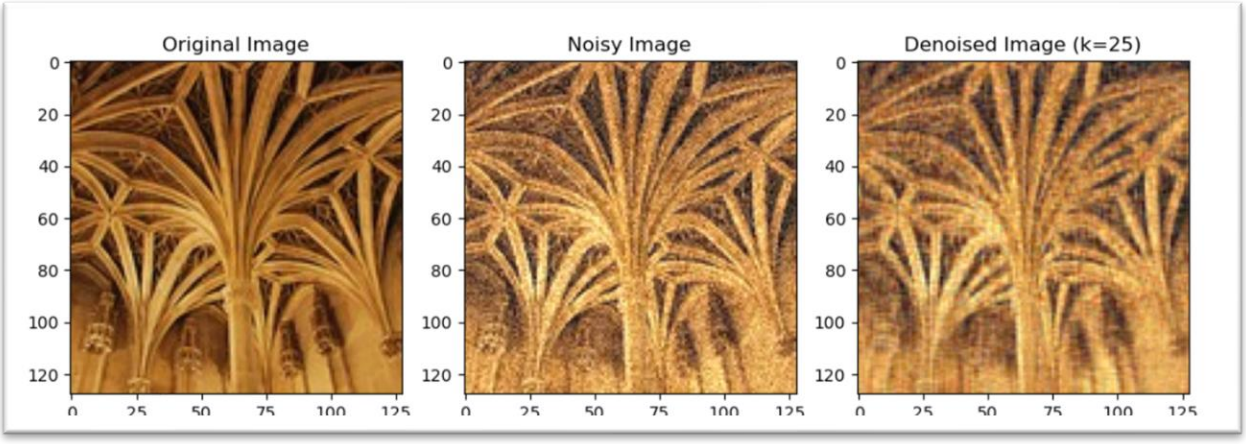

Original Image
Noisy Image
Denoised Image (k=30)

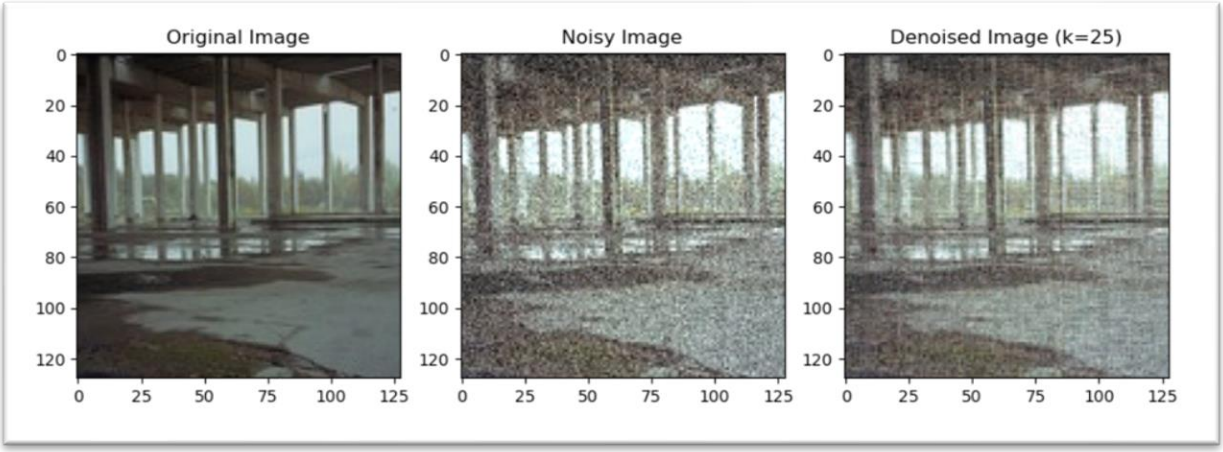

Original Image
Noisy Image
Denoised Image (k=50)

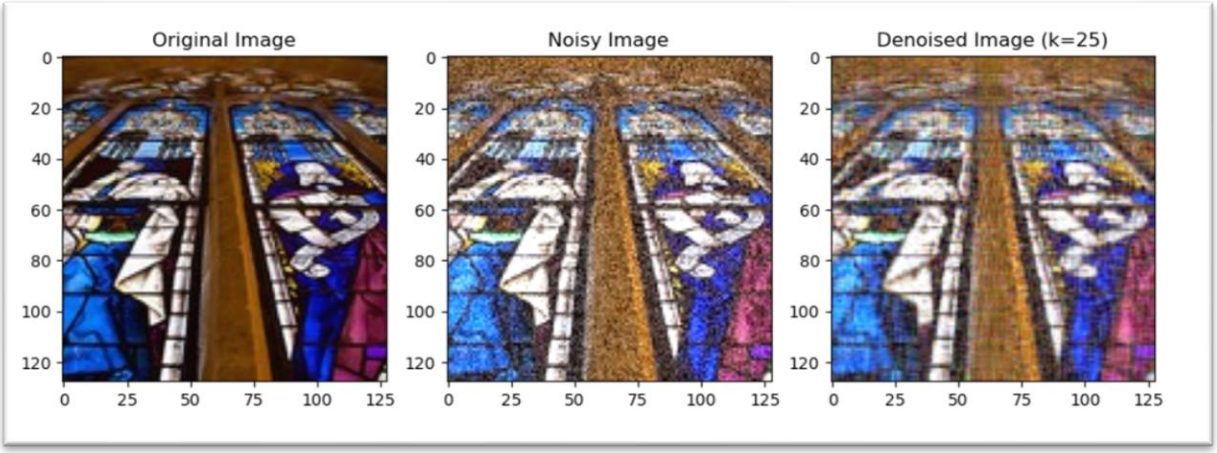
Denoising result for 10 images

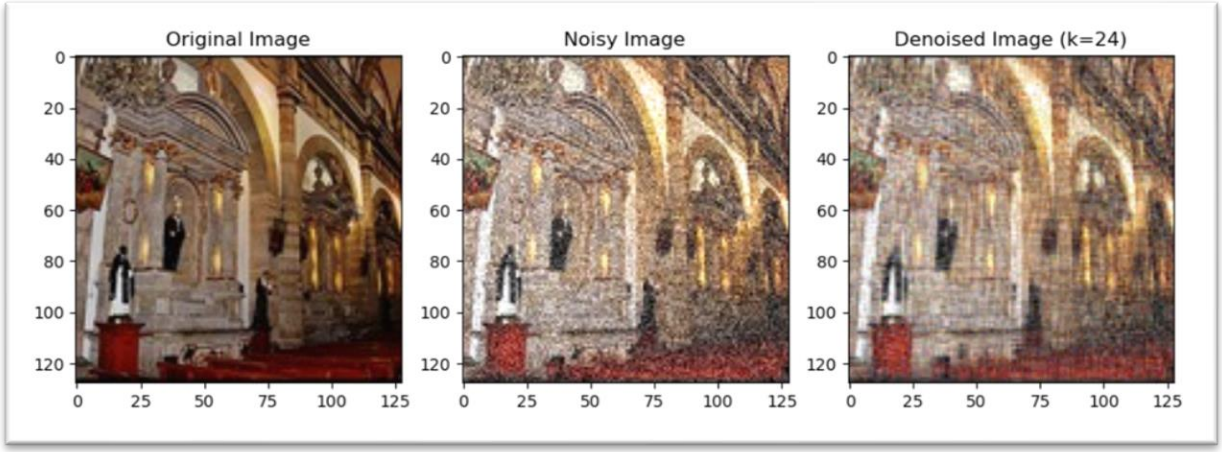


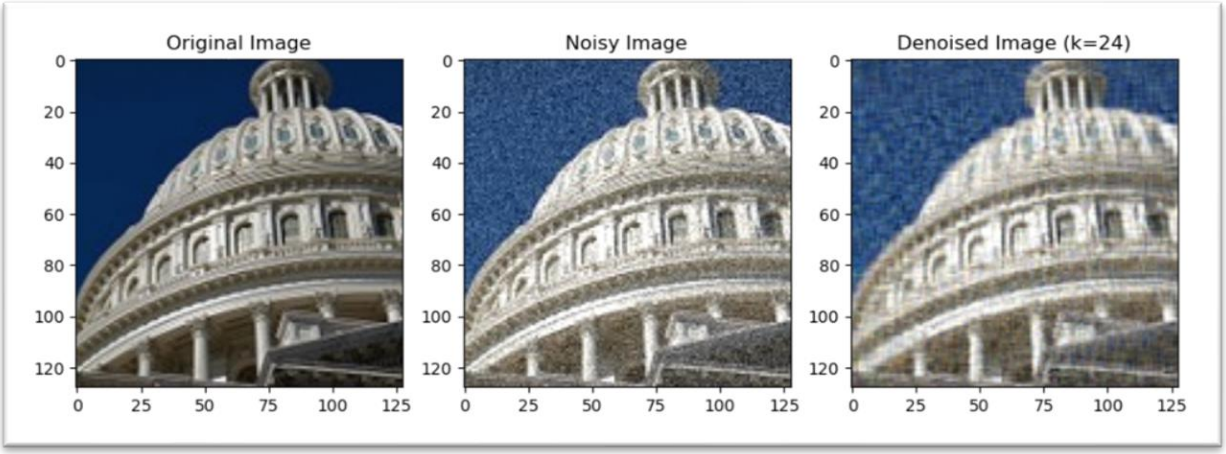


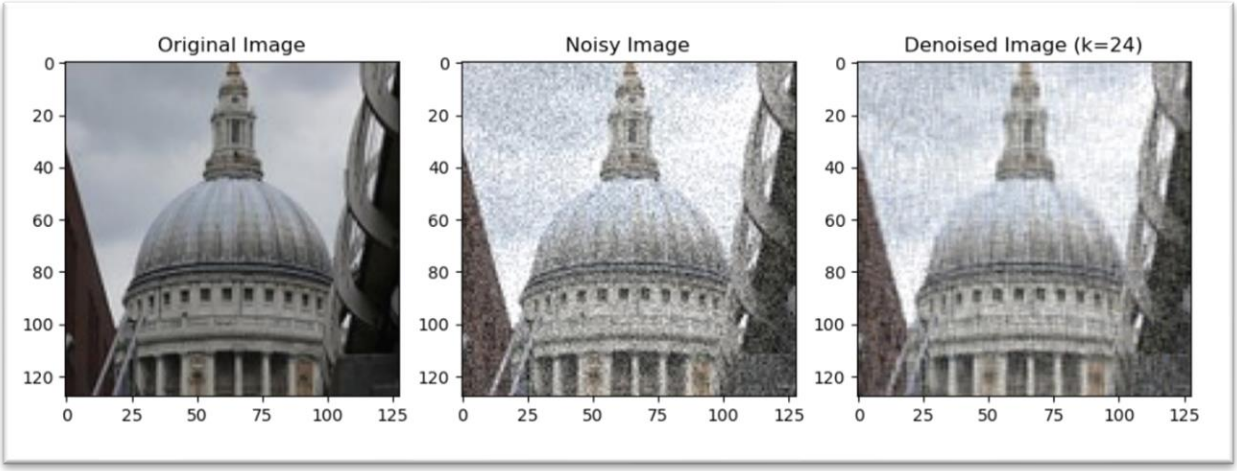


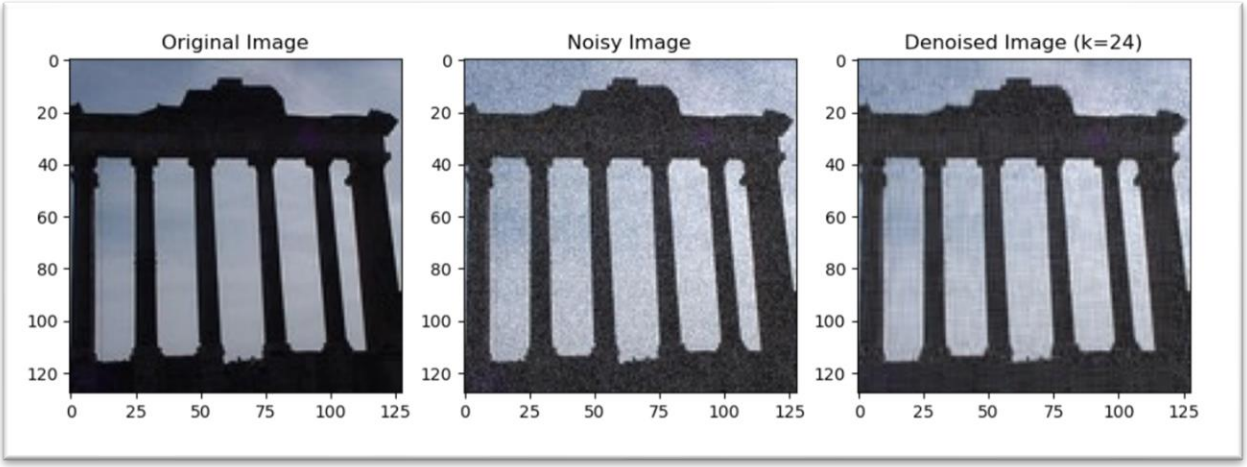

Original Image
Noisy Image
Denoised Image (k=25)

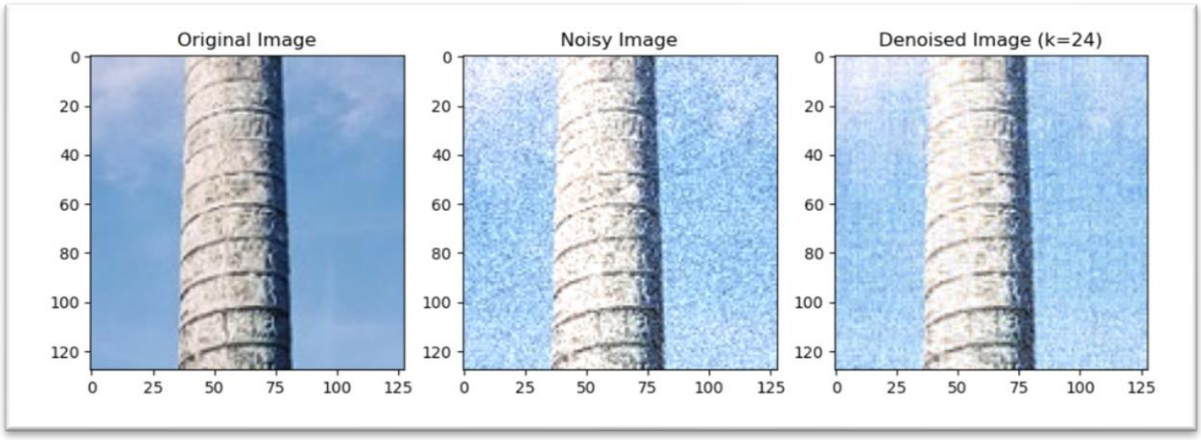

Original Image
Noisy Image
Denoised Image (k=25)


Original Image
Noisy Image
Denoised Image (k=24)


Original Image
Noisy Image
Denoised Image (k=24)


Original Image
Noisy Image
Denoised Image (k=24)


Original Image
Noisy Image
Denoised Image (k=24)


Original Image
Noisy Image
Denoised Image (k=24)
0
20
40
60
80
100
120
0
25
50
75
100
125

Analyzing results

Based on the results, it can be concluded that:

- **Reducing 'k'**: When we reduce 'k', we are keeping fewer components. This means we are discarding the components with smaller singular values, which are often associated with noise or less important features.
- **Increasing 'k'**: When we increase 'k', we are keeping more components, including those with smaller singular values. This might retain more detailed information but may keep noise, too.
- If **'k' is increased too much,** (especially to the point where it equals the total number of components), we're not simplifying or denoising the data at all.
- If **'k' is reduced too much**, important information might be lost, leading to a loss in data accuracy.

Generally, the choice of 'k' is a trade-off between noise reduction (denoising) and preserving important information.

Why SVD?

Singular Value Decomposition (SVD) can help in denoising images because it allows us to break down an image matrix into its singular values and vectors. These singular values represent the amount of variation within each component of the image.

In singular values, the first few values are much larger than the rest. With this characteristic, can be used to reduce the dimensionality of the image while keeping most of the important information. When we select only the largest singular values and corresponding singular vectors, we can reconstruct a "cleaner" version of the image that is less affected by noise.