Texture Synthesis and Image Inpainting

Python Modules used:

```
skimage
NumPy
PIL
IPython
scipy
time
```

Folder Structure:

⇒ ***Assignment 2:***
- Assignment2_Q1.ipynb
- Assignment2_Q2.ipynb
- Assignment2_Q3_Efros.ipynb
- Assignment2_Q3.ipynb
- ⇒ **Assignment-II-images-1**
  - *All Images*
- ⇒ **synImages**
  - All synthesized images

To reduce the computational overhead, all image pixels are brought down between 0 and 1 by multiplying with 255.

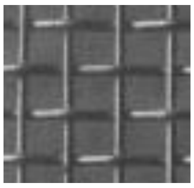
1. Texture Synthesis:

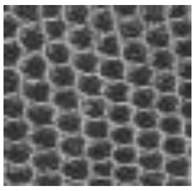
This is an implementation of Efros and Leung's approach for texture synthesis. We are given 5 sample texture (T1.gif, T2.gif, T3.gif, T4.gif, T5.gif) and we need to synthesize a 200x200 pixel image for each of the given texture.

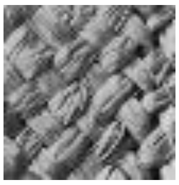
Code file: Assignment2_Q1.ipynb

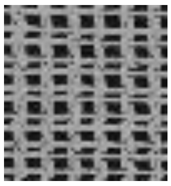
Assignment2_Q1.ipynb implements the algorithm suggested in Efros and Leung paper. The code is run for different kernelSize [5, 9, 11]

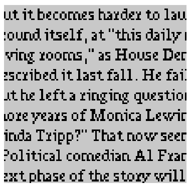
It is observed that as the window size increases, the runtime also increases because of computational complexity. However, the algorithm is able to catch the texture more prominently with increasing windowSize.


T1.gif


T2.gif


T3.gif

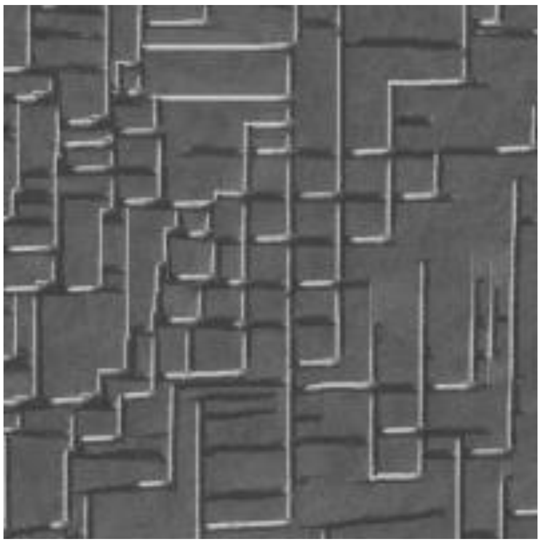

T4.gif

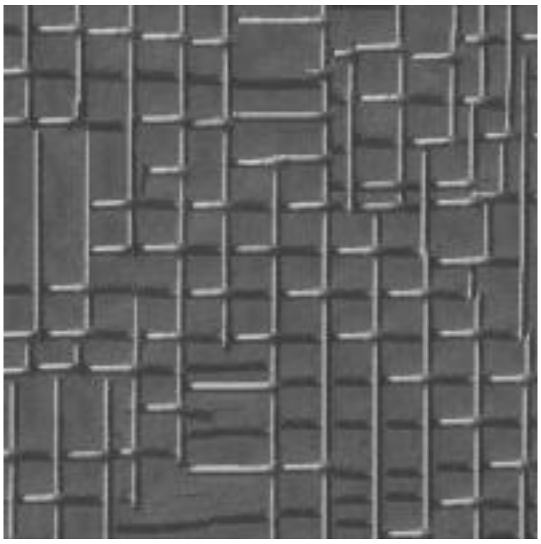

ut it becomes harder to lau
:ound itself, at "this daily r
ving rooms," as House Dec
escribed it last fall. He fai
ut he left a ringing questio
ore years of Monica Lewir
inda Tripp?" That now seer
Political comedian Al Frar
ext phase of the story will

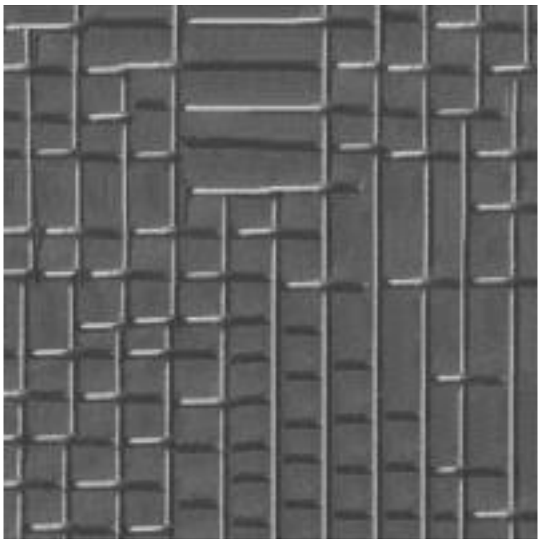
T5.gif

Fig 1.1 Original Texture Image

Synthesized Texture (200 x 200 pixel): T1.gif

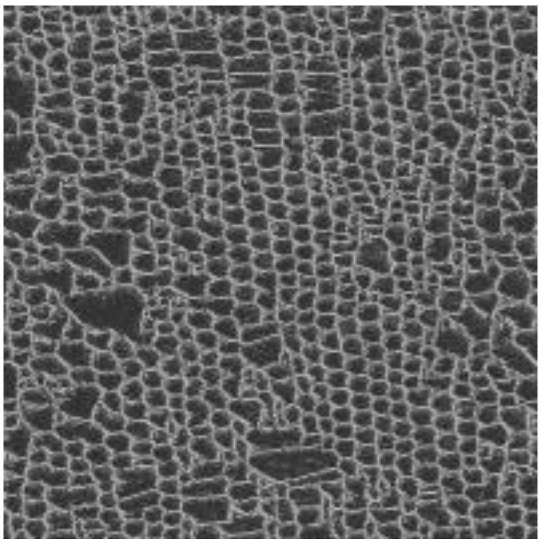

kernelSize: 5

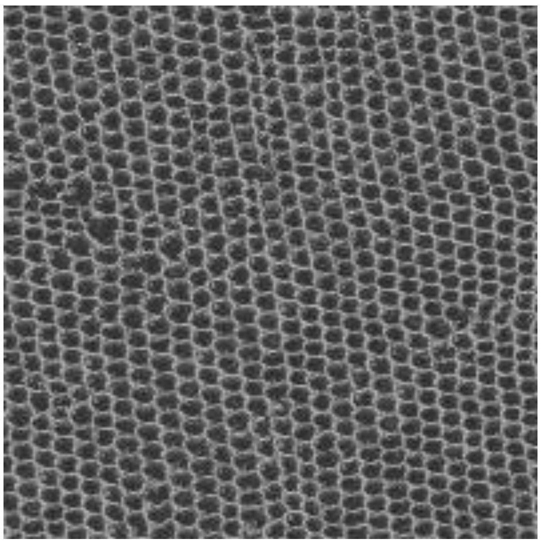

kernelSize: 9

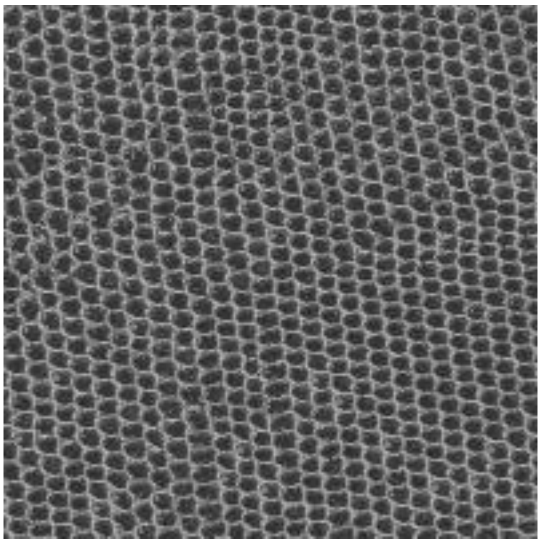

kernelSize: 11

As we increase the window size, the algorithm is better able to capture the texture pattern for synthesis.

Synthesized Texture (200 x 200 pixel): T2.gif

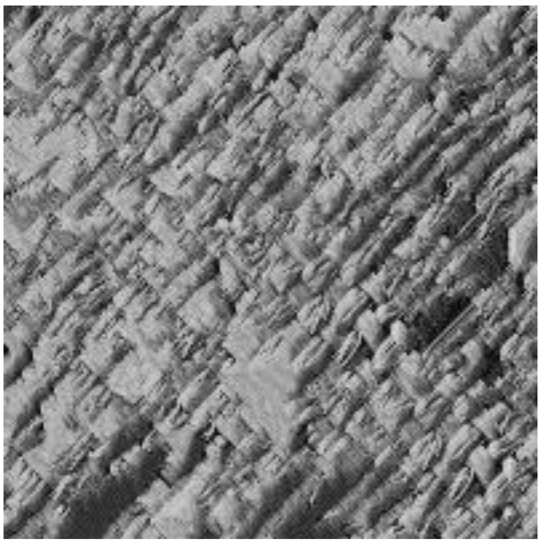

kernelSize: 5

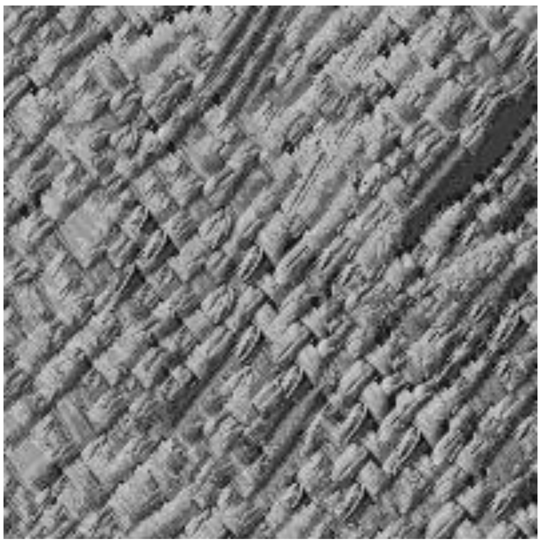

kernelSize: 9

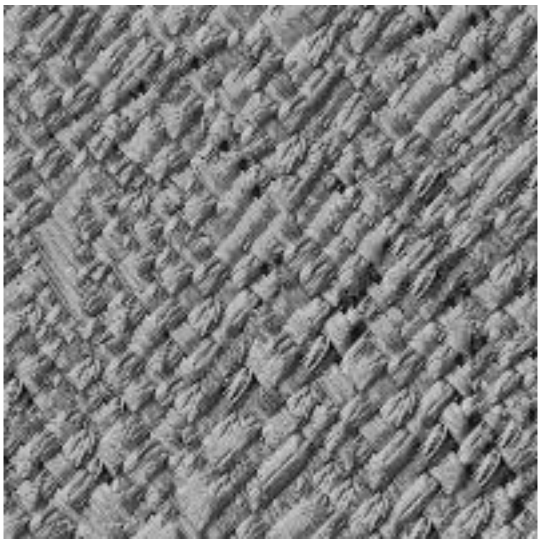

kernelSize: 11

kernelSize 9 is able to capture the relevant pattern, as the kenelSize is increased further, the synthesis is more plausible

Synthesized Texture (200 x 200 pixel): T3.gif

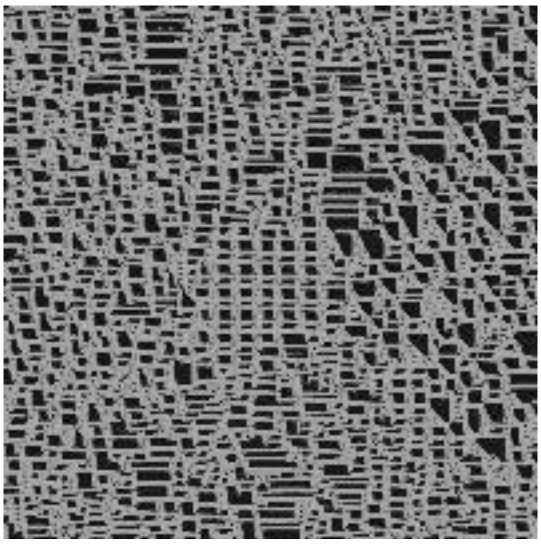

kernelSize: 5

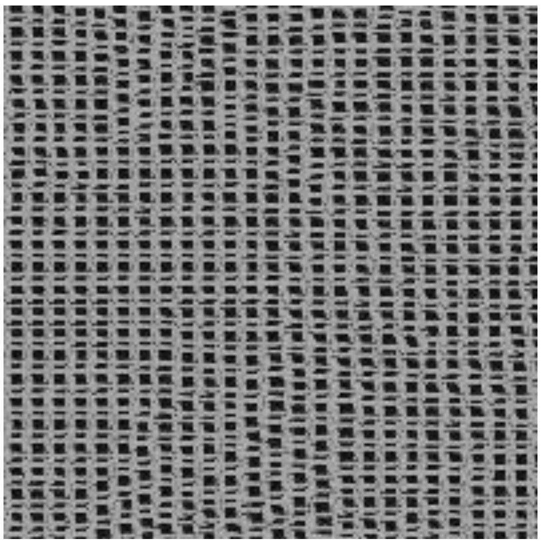

kernelSize: 9

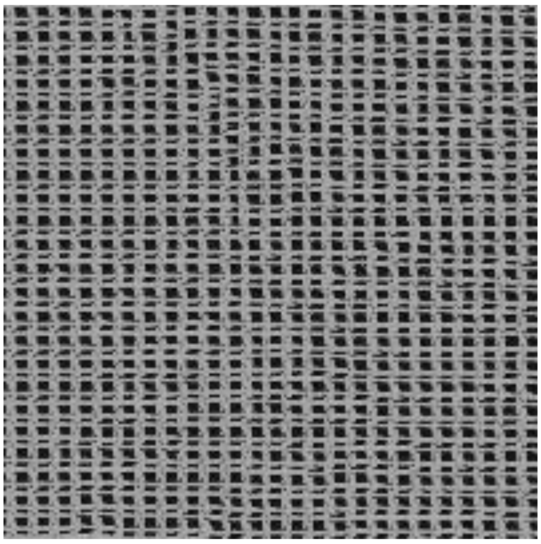

kernelSize: 11

We can clearly observe the mismatched pattern towards the right in kernelSize: 9, which is resolved in texture synthesis with kernelSize: 11

Synthesized Texture (200 x 200 pixel): T4.gif

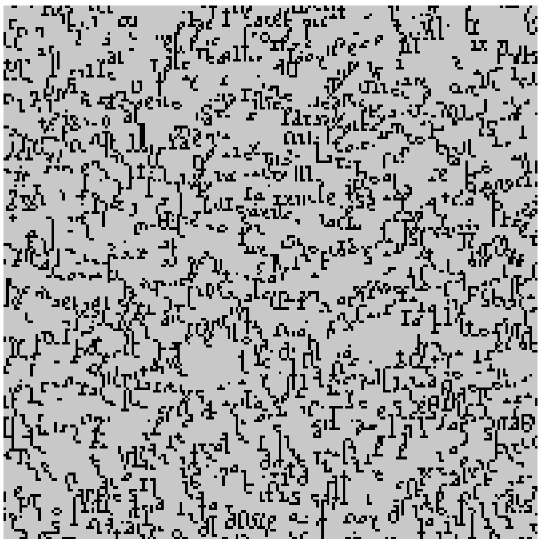

kernelSize: 5

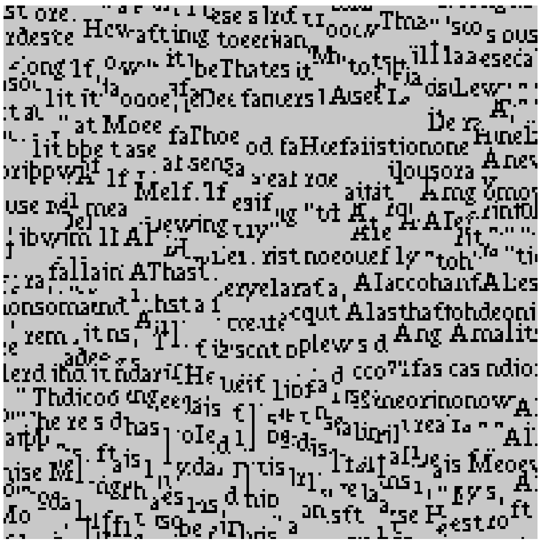

kernelSize: 9

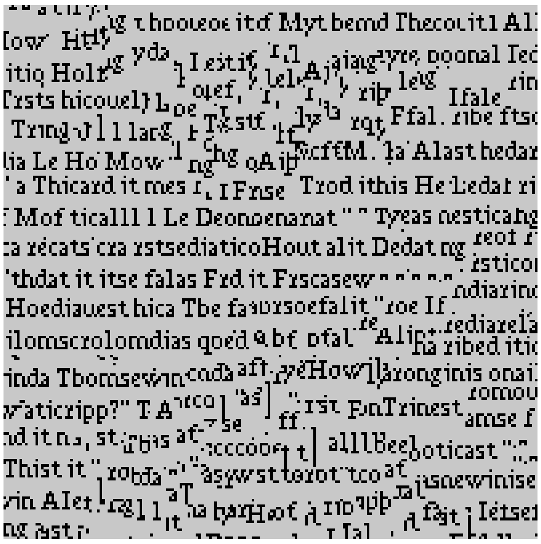

kernelSize: 11

As with the other example images, kernelSize:9 synthesizes plausible pattern for the given texture.

Synthesized Texture (200 x 200 pixel): T5.gif


kernelSize: 5


kernelSize: 9


kernelSize:11

With kernelSize:11, we are starting to get a good result, as we increase the window/kernel size, we will see even better results.

However, the kernel/window size should not go beyond a limit that it cannot differentiate between different patterns within one given texture.

Time Summary for Texture Synthesis:

```
File: T1.gif, Kernel Size:5, Time Taken:40.7599 seconds
File: T1.gif, Kernel Size:9, Time Taken:58.3465 seconds
File: T1.gif, Kernel Size:11, Time Taken:65.914 seconds
File: T2.gif, Kernel Size:5, Time Taken:28.6464 seconds
File: T2.gif, Kernel Size:9, Time Taken:35.1098 seconds
File: T2.gif, Kernel Size:11, Time Taken:48.5099 seconds
File: T3.gif, Kernel Size:5, Time Taken:28.3562 seconds
File: T3.gif, Kernel Size:9, Time Taken:36.9076 seconds
File: T3.gif, Kernel Size:11, Time Taken:47.5528 seconds
File: T4.gif, Kernel Size:5, Time Taken:23.6185 seconds
File: T4.gif, Kernel Size:9, Time Taken:26.979 seconds
File: T4.gif, Kernel Size:11, Time Taken:39.3306 seconds
File: T5.gif, Kernel Size:5, Time Taken:123.8438 seconds
File: T5.gif, Kernel Size:9, Time Taken:173.1245 seconds
File: T5.gif, Kernel Size:11, Time Taken:170.7191 seconds
```

2. Image Inpainting

With slight modifications in the code, to accommodate for sample image and target to be synthesized, this implementation is also based on Efros and Leung paper. In this implementation, the seed is considered to be the given image and we extract sample image (or texture) as portion that are filled. We then extract patches of given kernelSize from the sample image. Using gaussian weighted d_{SSD}, we then find the best match for unfilled image (black portion of the image) pixel by pixel and inpaint it.

Two images are given for image painting: test_im1.bmp, test_im2.bmp
The code is run for different kernelSize [5, 9, 11, 15, 17, 21]

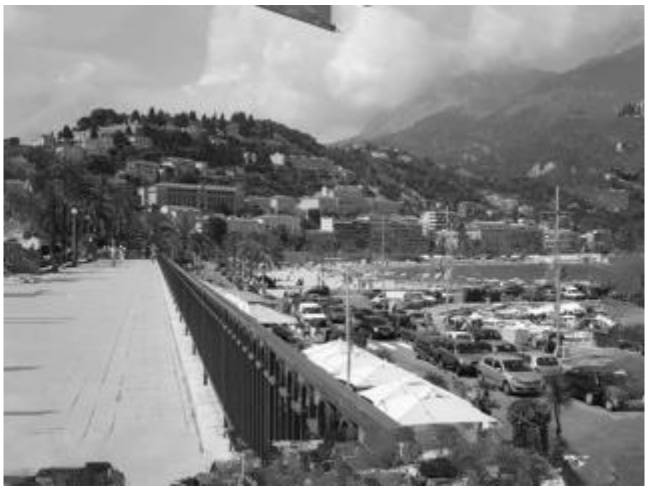
We will observe that with increase in kernelSize, the quality of image inpainting gets better.

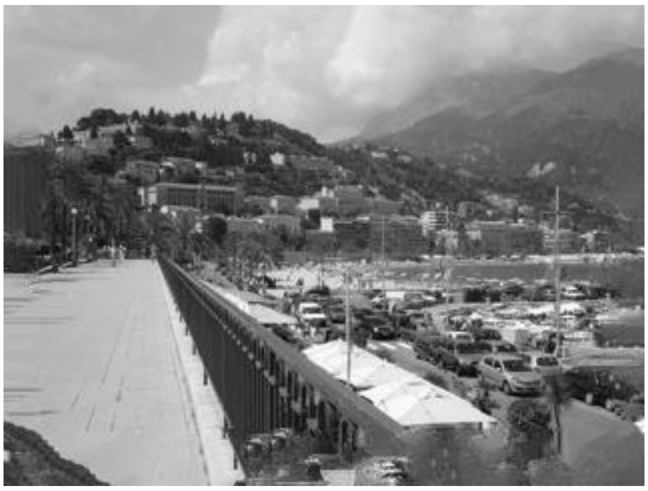
Time Summary for Image Inpainting:

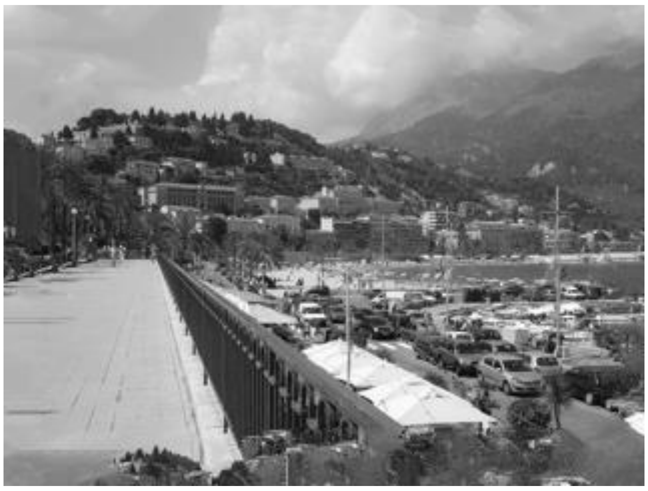
```
File: test_im1.bmp, Kernel Size:5, Time Taken:419.4565 seconds
File: test_im1.bmp, Kernel Size:9, Time Taken:740.3836 seconds
File: test_im1.bmp, Kernel Size:11, Time Taken:1071.2232 seconds
File: test_im1.bmp, Kernel Size:15, Time Taken:1631.4592 seconds
File: test_im1.bmp, Kernel Size:17, Time Taken:2286.8628 seconds
File: test_im1.bmp, Kernel Size:21, Time Taken:3204.4843 seconds
File: test_im2.bmp, Kernel Size:5, Time Taken:184.6985 seconds
File: test_im2.bmp, Kernel Size:9, Time Taken:322.5908 seconds
File: test_im2.bmp, Kernel Size:11, Time Taken:425.2403 seconds
File: test_im2.bmp, Kernel Size:15, Time Taken:657.1364 seconds
File: test_im2.bmp, Kernel Size:17, Time Taken:979.1365 seconds
File: test_im2.bmp, Kernel Size:21, Time Taken:1452.2824 seconds
```

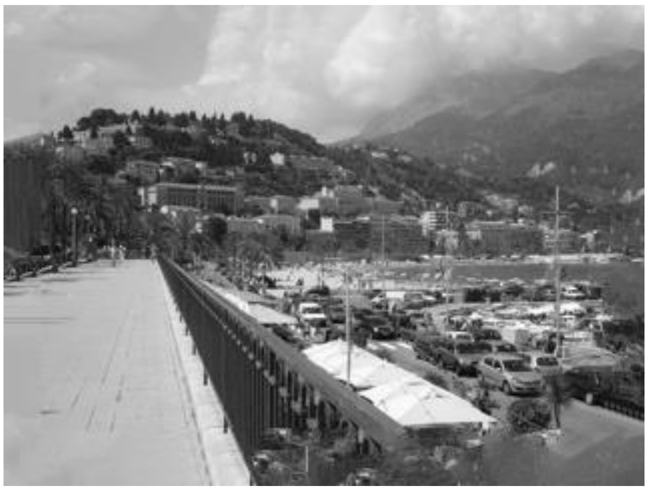
Test_im1.bmp

Original Image


kernelSize: 5


kernelSize: 9

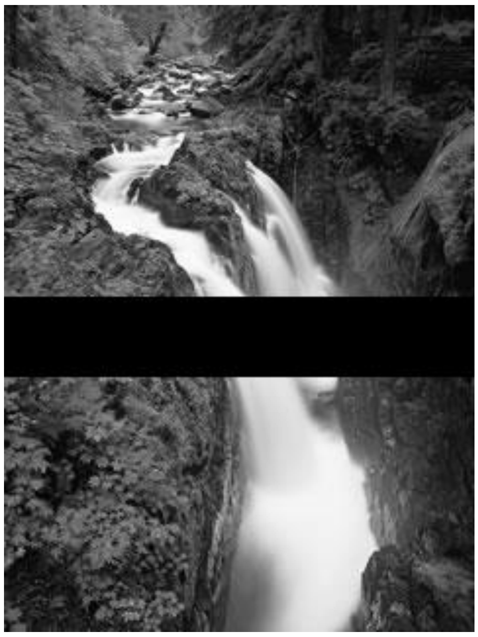

kernelSize:11


kernelSize: 15

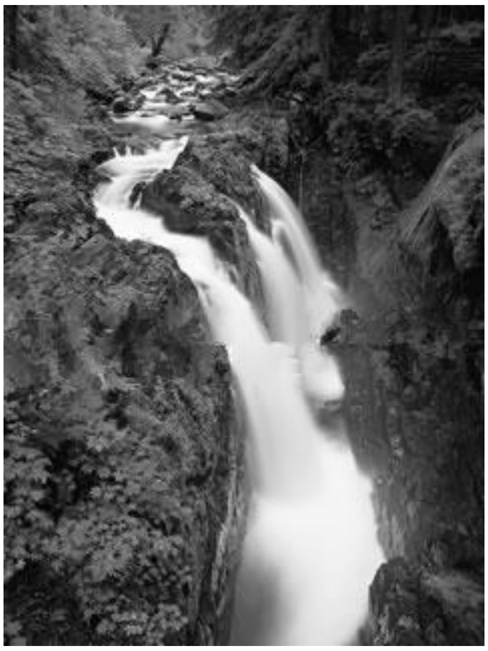
kernelSize: 17

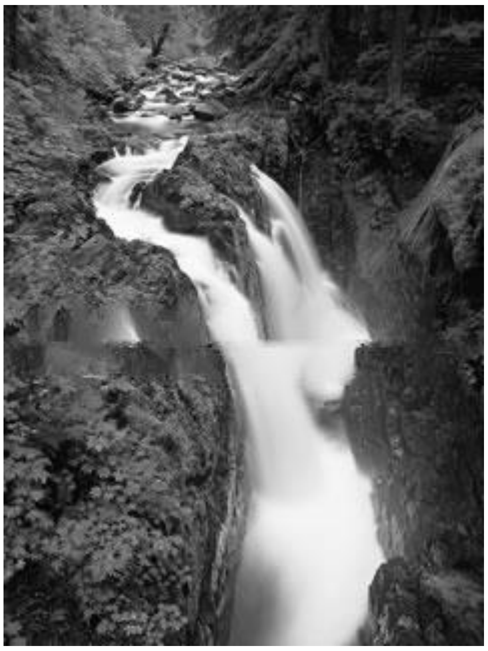
kernelSize: 21

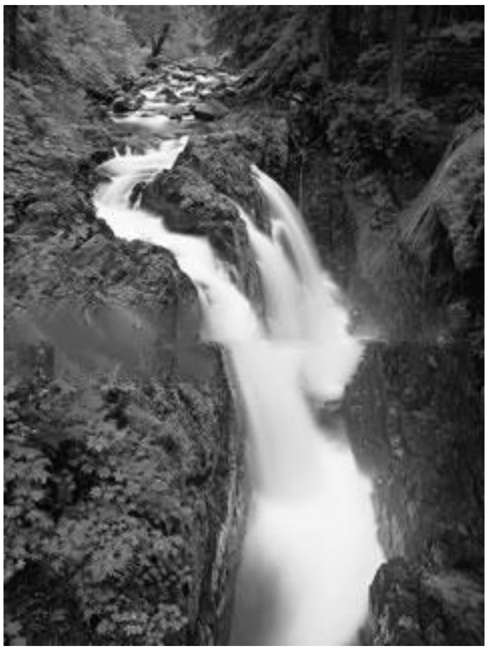
Test_im2.bmp

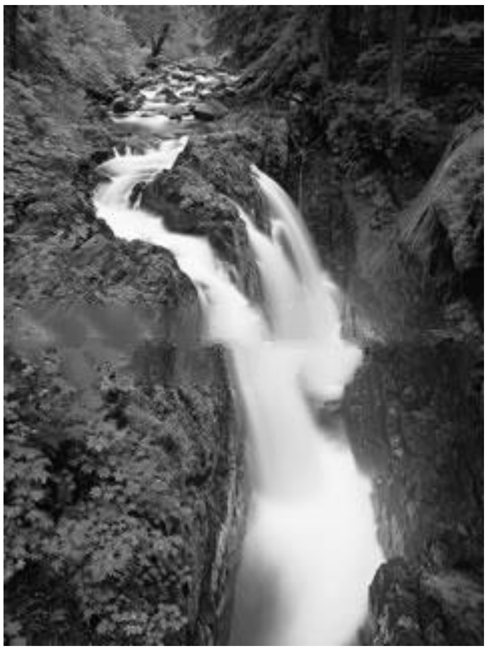


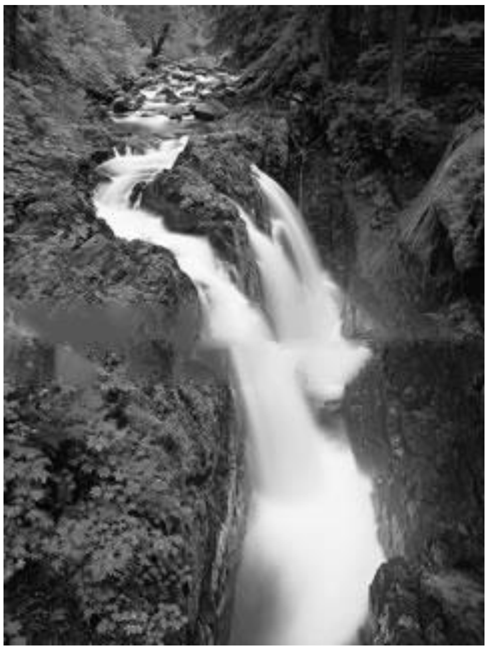
Original Image

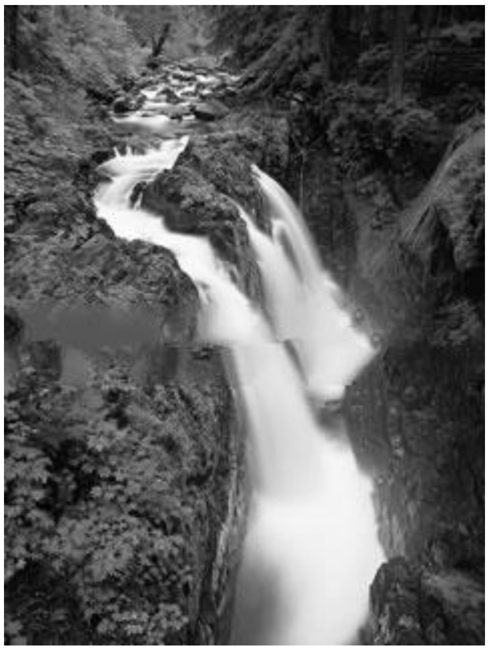

kernelSize: 5


kernelSize: 9


kernelSize: 11


kernelSize: 15


kernelSize: 17


kernelSize: 21

3. Object Removal

We are given an image to remove three objects from it and then inpaint the removed region. Three objects are: the man on the left, the sign and pole on the bottom right, and the oversaturated area on the ground.

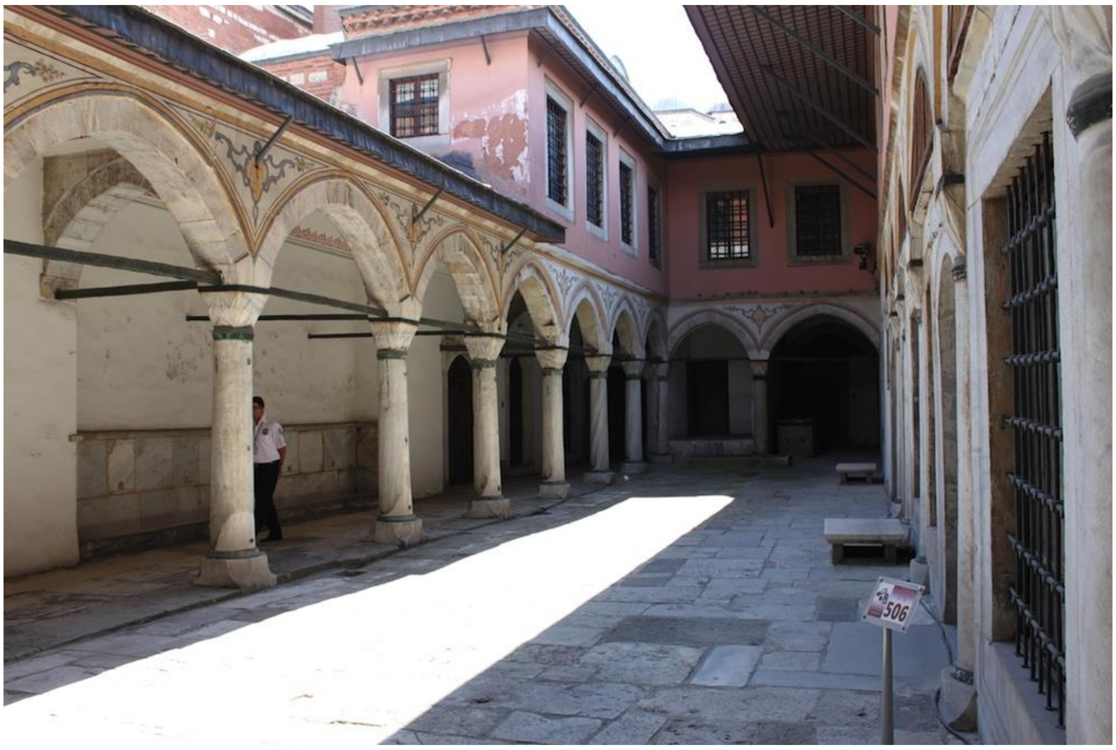
Outputs from two approaches are compared:

1) Criminisi, Perez and Toyama, "Region Filling and Object Removal by Exemplar-Based Image Inpainting"
2) Alexei A. Efros and Thomas K. Leung, "Texture Synthesis by Non-Parametric Sampling"

It is run for kernelSize: 9

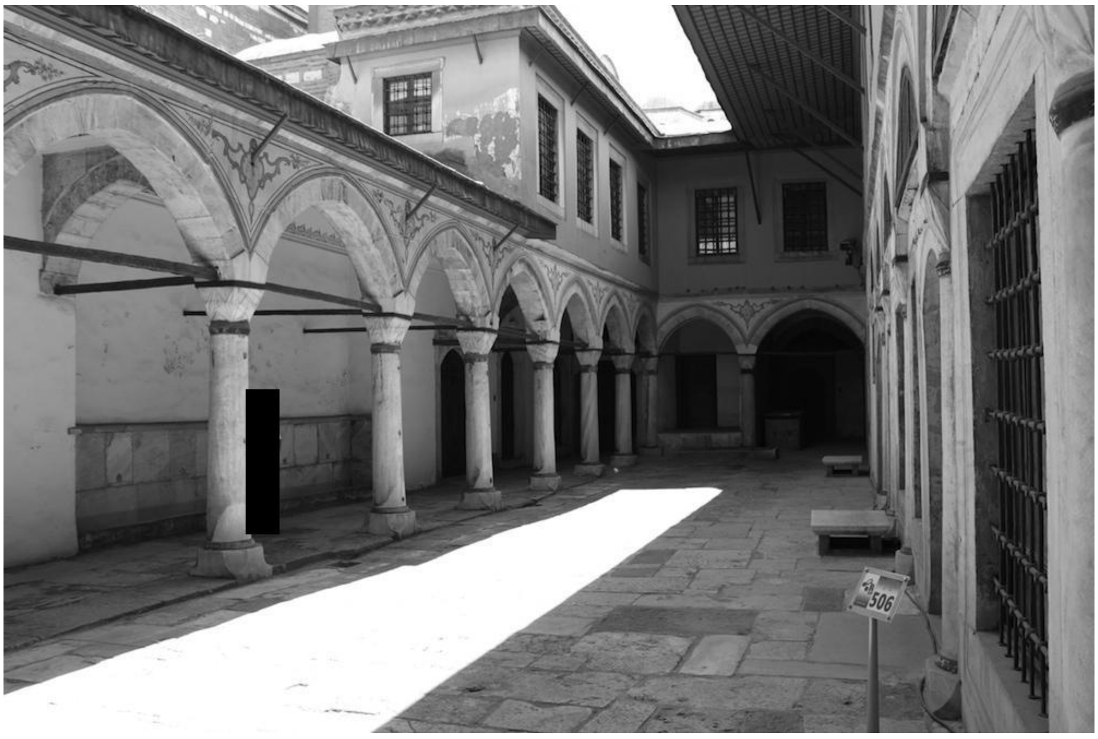
We can clearly observe that Efros and Leung's approach take a lot of time to run as compared to Criminisi, Perez and Toyama's approach. For smaller objects, Criminisi et al. approach works much better, both in terms of efficiency and plausibility. However, for bigger objects, Efros and Leung's approach give a slightly better result, although it takes a long time to run.

Time Summary:
Criminisi, Perez and Toyama's
Person=> File: test_im3.jpg, Kernel Size:9, Time Taken:160.7148 seconds
Sign=> File: test_im3.jpg, Kernel Size:9, Time Taken:185.6749 seconds
Ground=> File: test_im3.jpg, Kernel Size:9, Time Taken:2253.5974 seconds
Efros and Leung's
Person=> File: test_im3.jpg, Kernel Size:9, Time Taken:1665.5445 seconds
Sign => File: test_im3.jpg, Kernel Size:9, Time Taken:2543.6681 seconds
Ground=> File: test_im3.jpg, Kernel Size:9, Time Taken:24407.1807 seconds

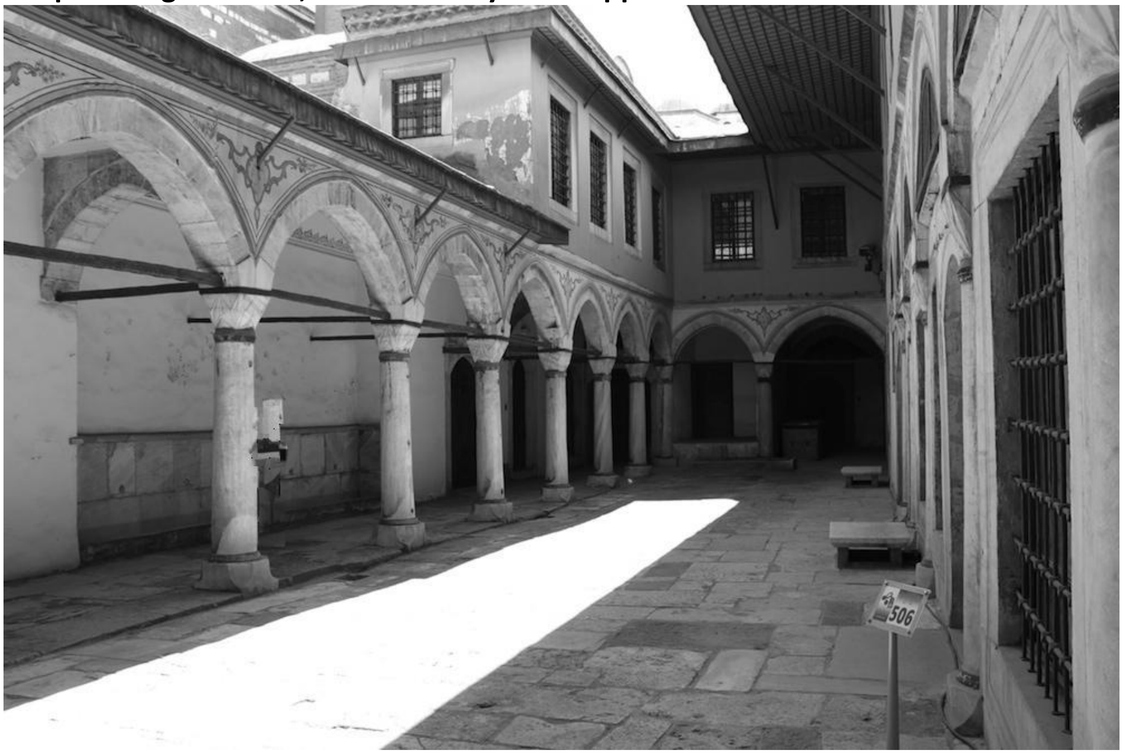


Original Image: Objects to be removed, a) the man on the left, b) the sign and the pole on the bottom right, c) oversaturated area on the ground
We get the coordinates for object to be removed:
person_coord = [(352,485,222,253)]
sign_coord = [(513,567,770,830), (566,664,788, 803)]
ground_coord = [(630, 664,3,390), (600, 630, 3, 435), (570,600,95,490), (540,570, 190,530), (510,540, 283, 570), (480, 510, 390, 610), (465,480, 465, 630), (440,465, 515,660)]

Person Masked:

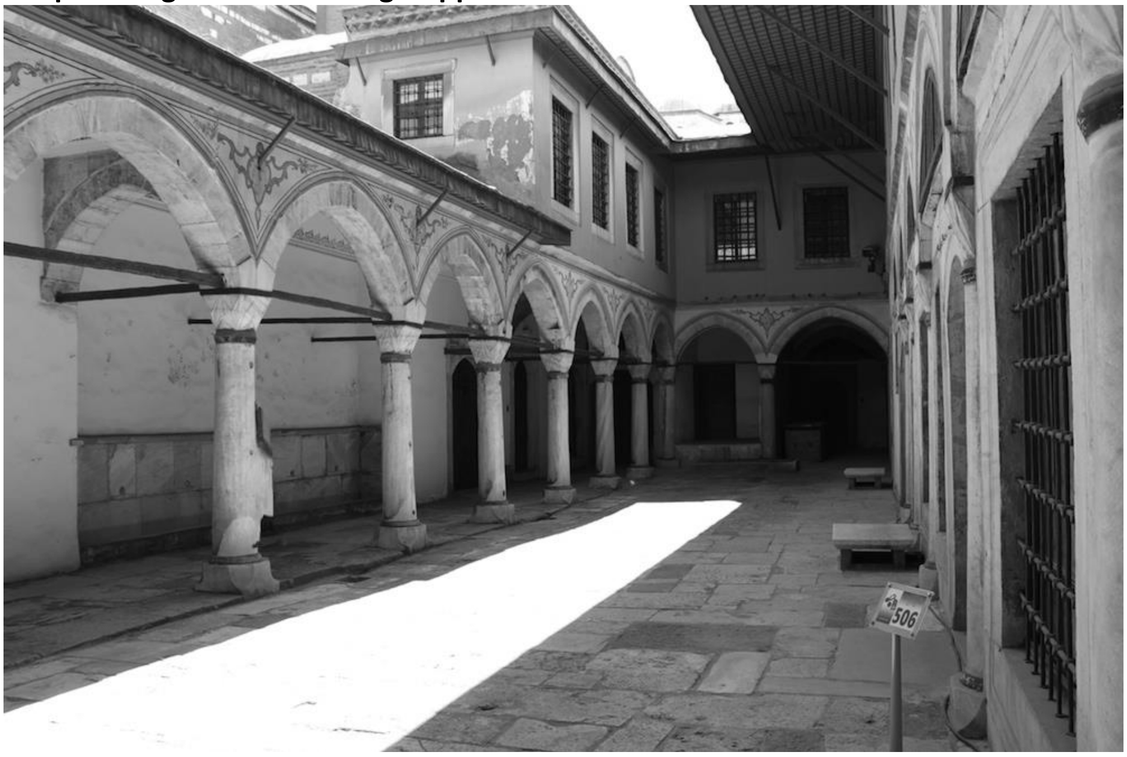


Output using Criminisi, Perez and Toyama's approach:



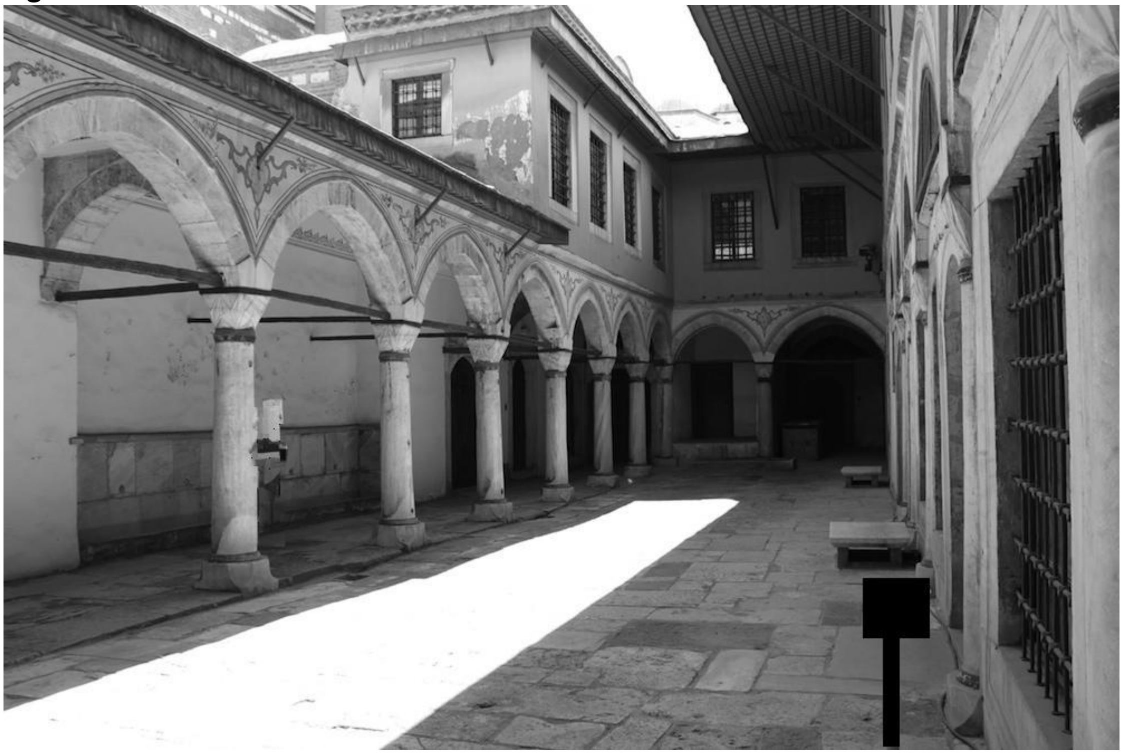
Person Removed, kernelSize: 9

Output using Efros and Leung's approach:

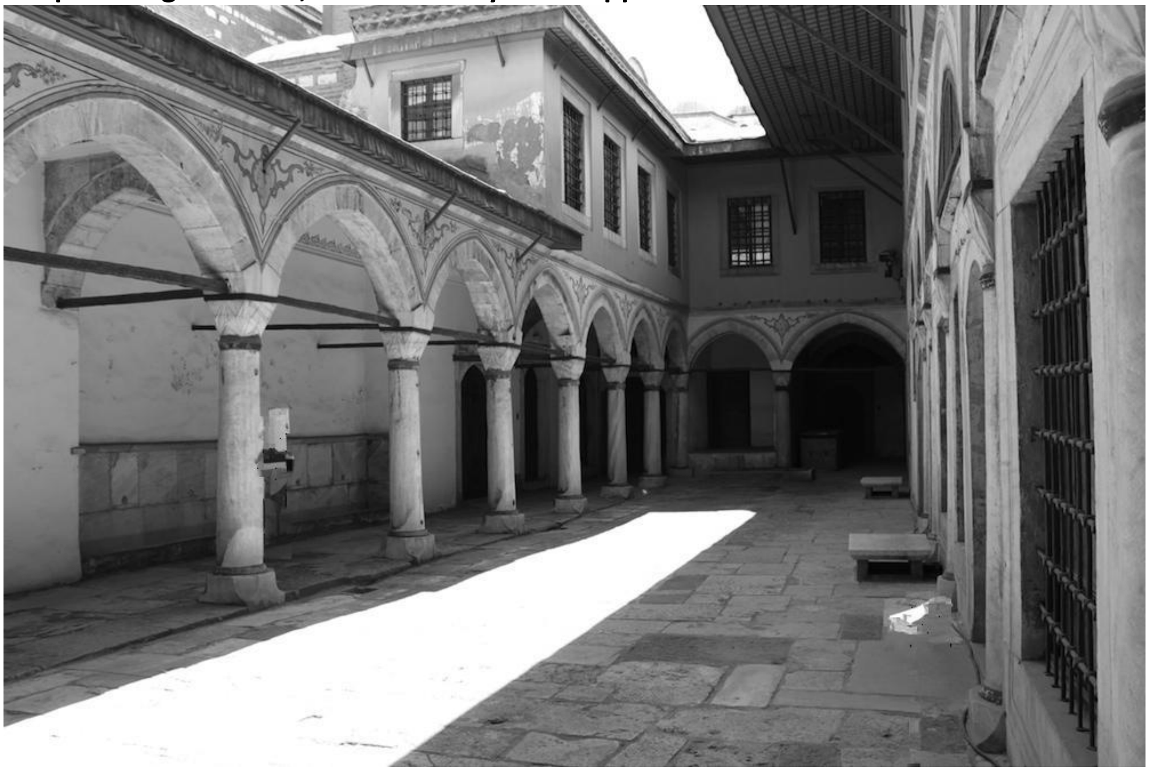


Person Removed, kernelSize: 9

Sign Masked:

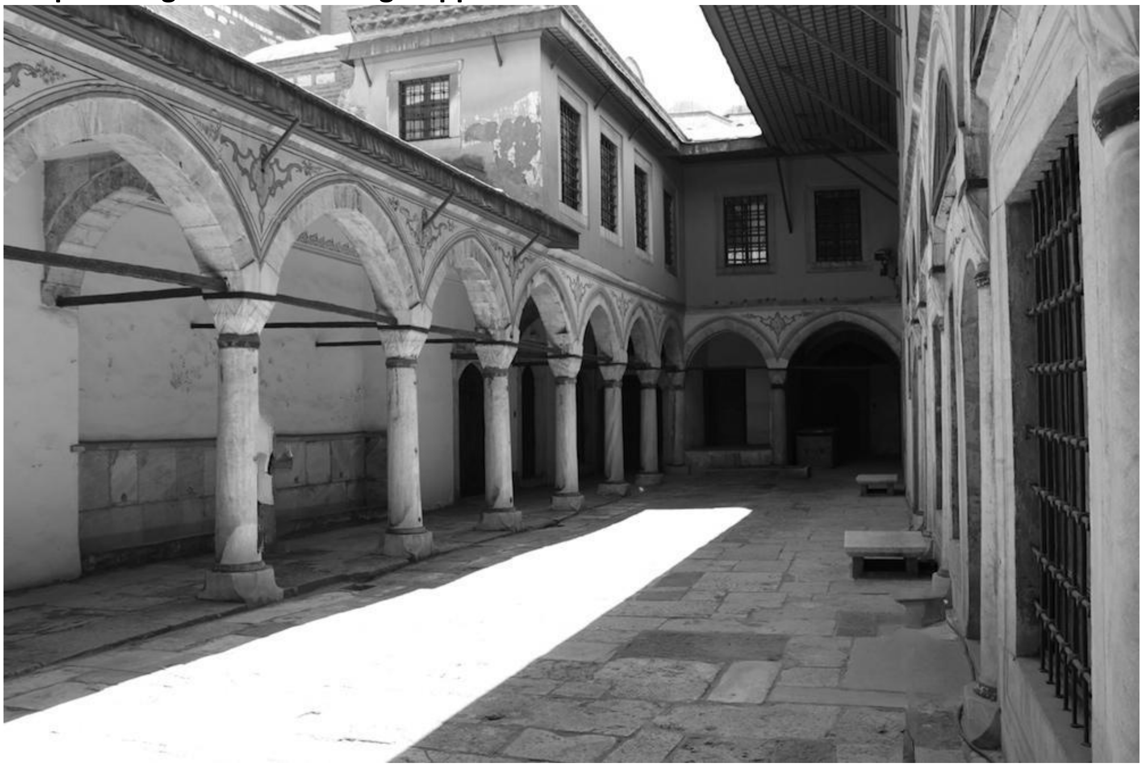


Output using Criminisi, Perez and Toyama's approach:



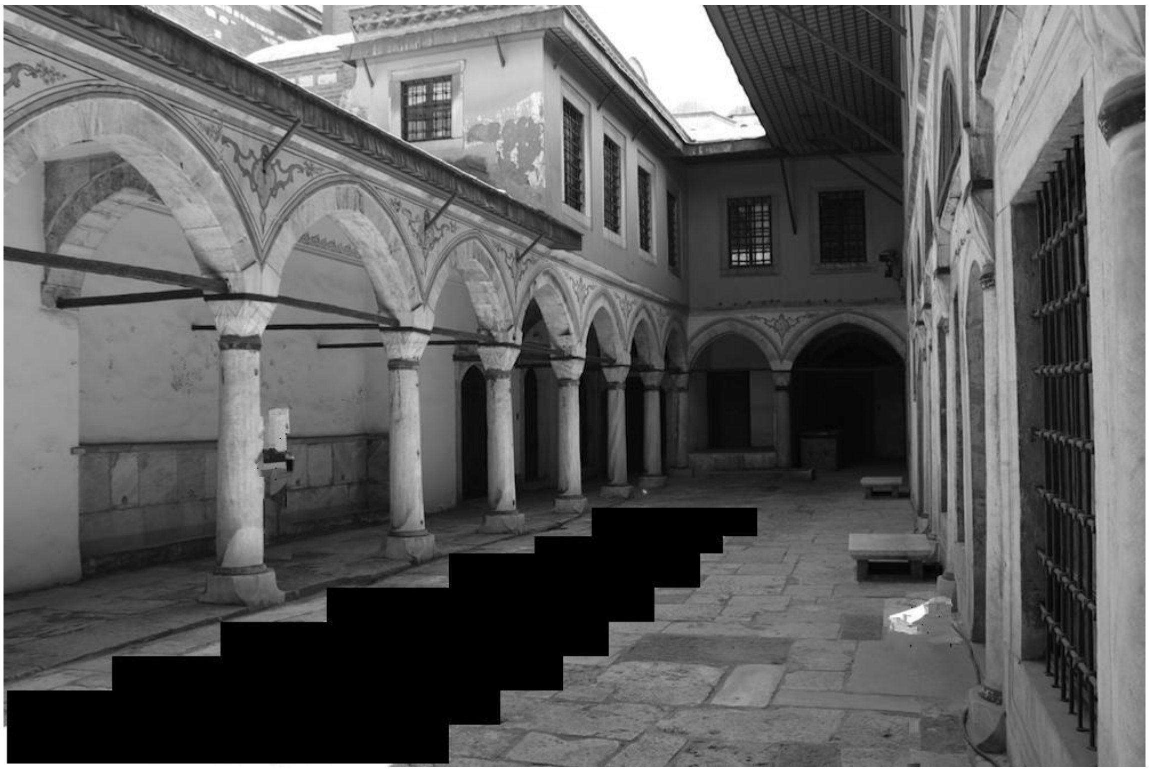
Sign Removed, kernelSize: 9

Output using Efros and Leung's approach:

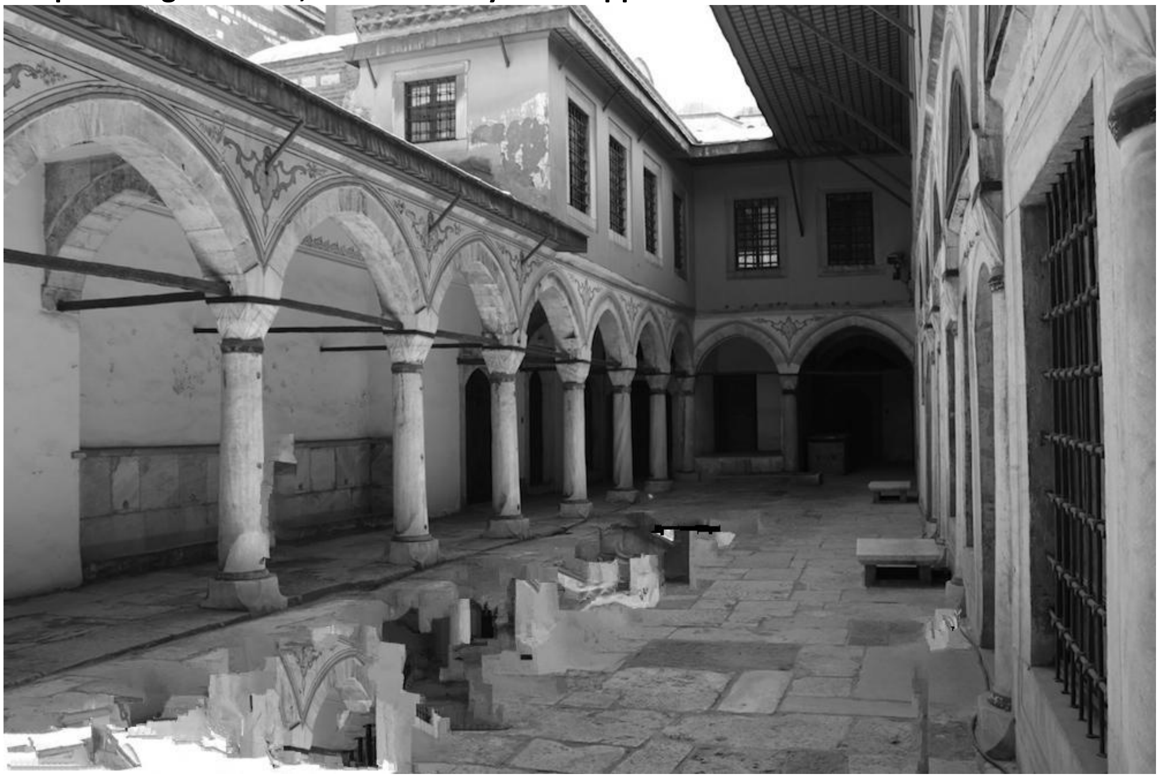


Sign Removed, kernelSize: 9

Ground Masked:

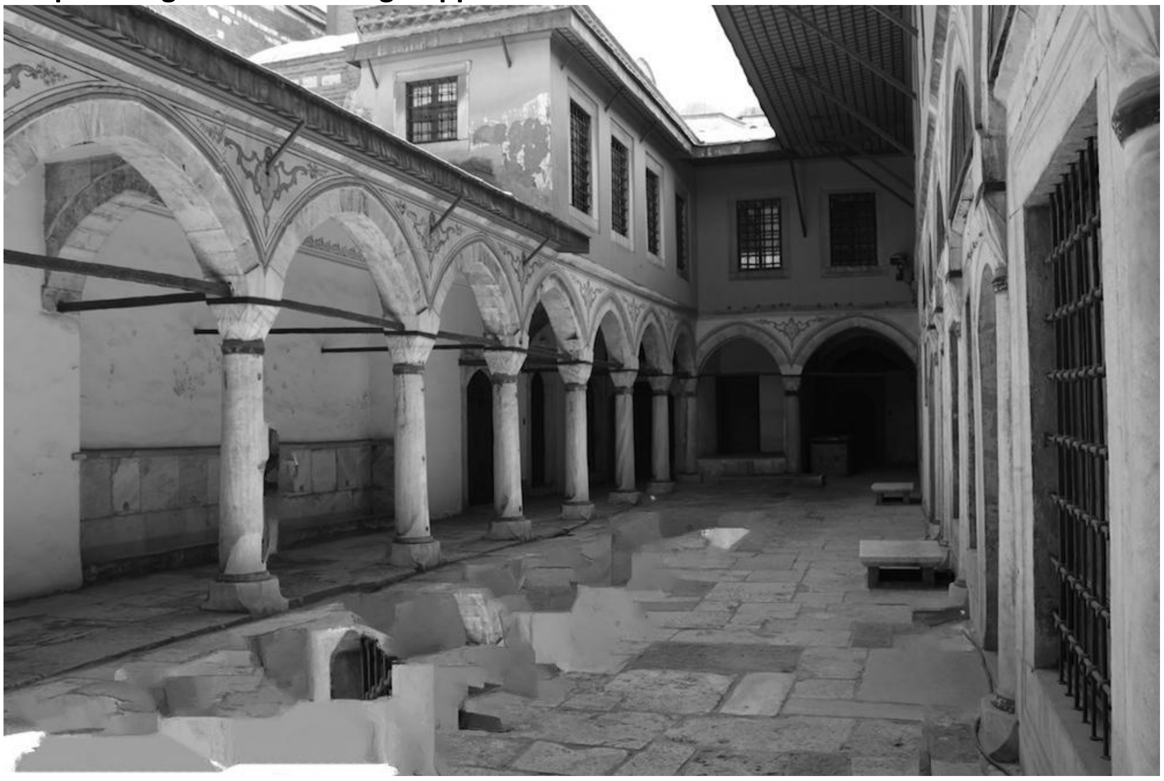


Output using Criminisi, Perez and Toyama's approach:



Ground Removed, kernelSize: 9

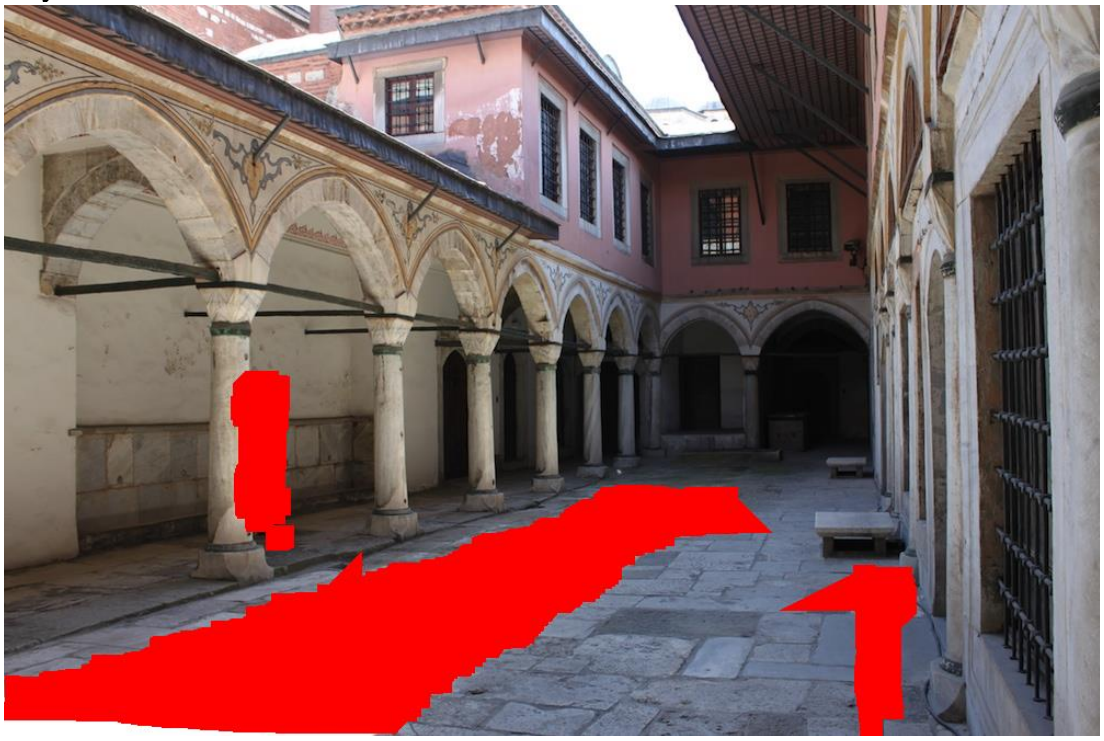
Output using Efros and Leung's approach:



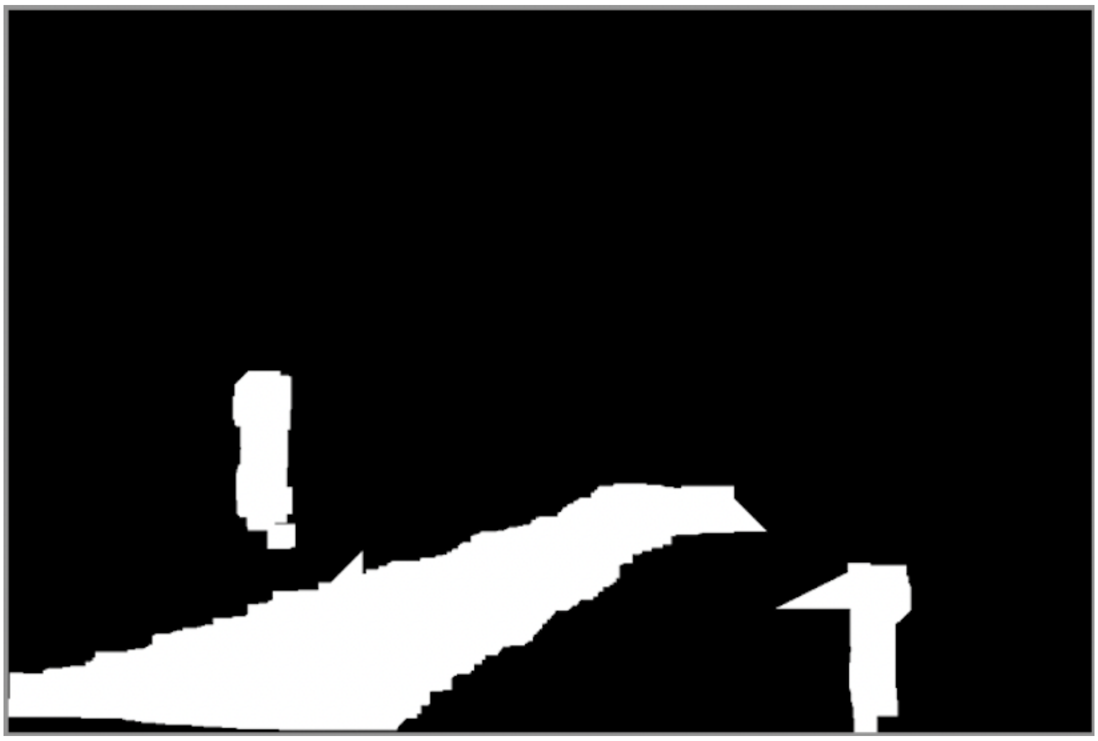
Ground Removed, kernelSize: 9

4. **Object Removal using Lama:**

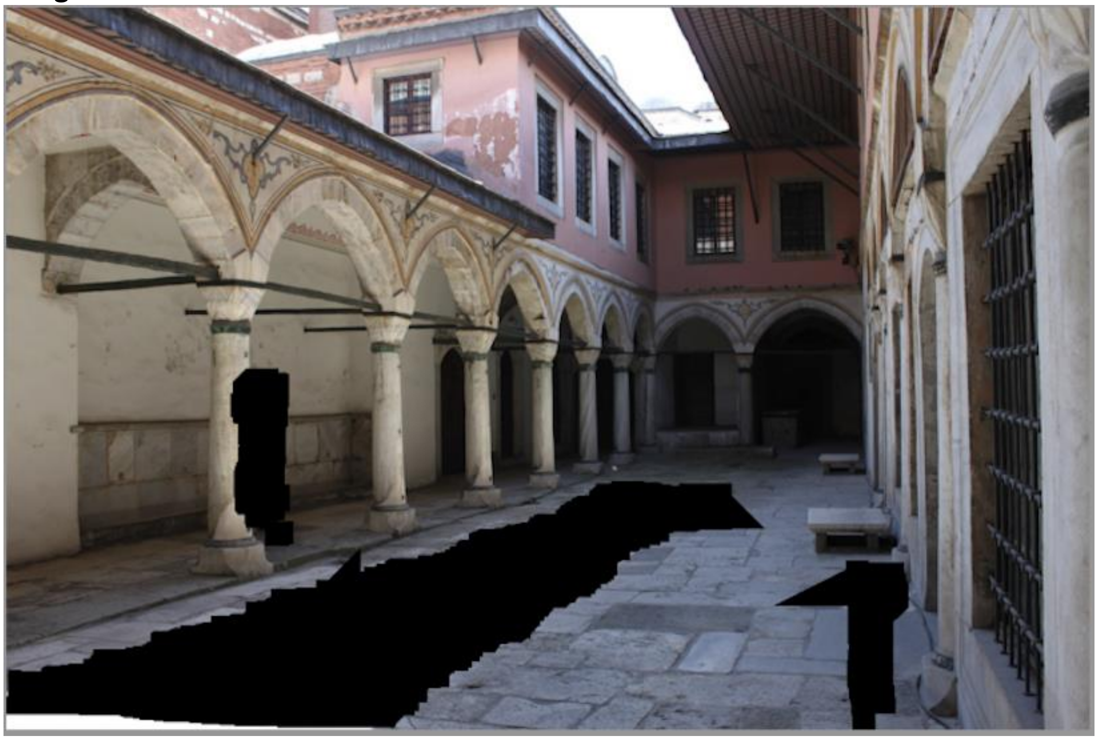
Object Mask:



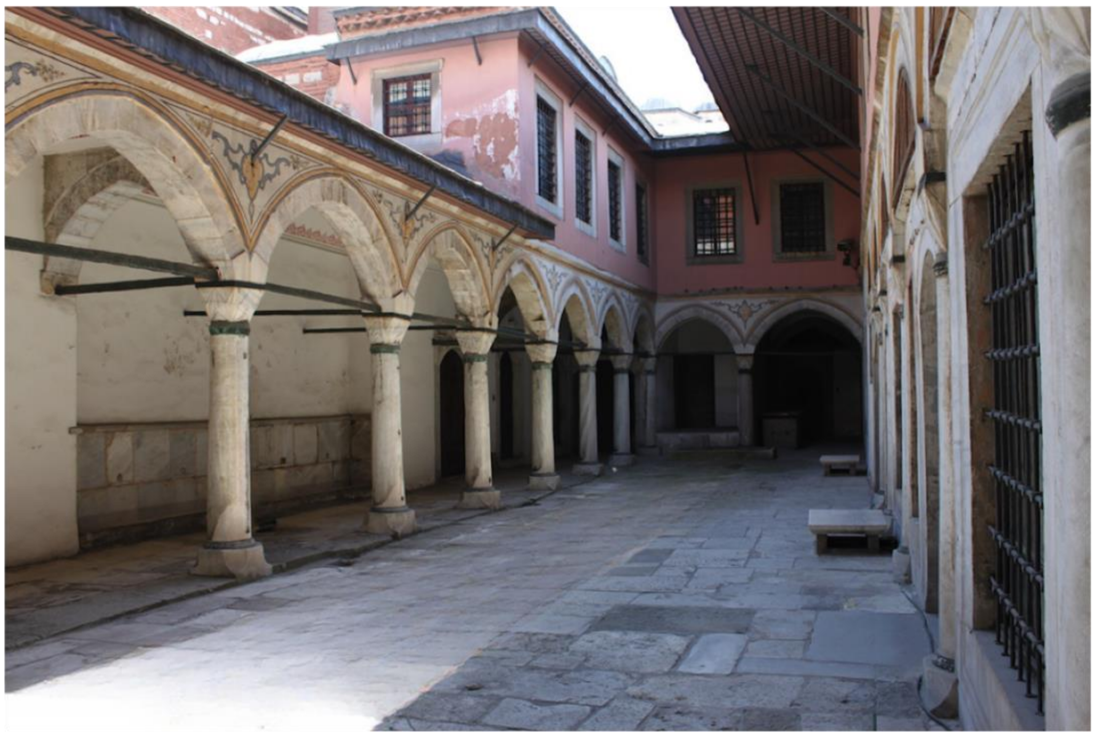
Mask:



Image*Masked:



inpainting result



The result of Lama Project is astonishing. It is successfully able to apply mask and remove object from the image. Post which it is also able to perform realistic image inpainting.

4 A) Fourier Convolutions

Fourier Transforms are used in image context to decompose image signal into functions based on spatial or temporal components. Fourier transforms represents the image in their frequency domain which provides a global context about the image. Fast Fourier Convolution is based on channel-wise fast Fourier transforms and thus has a receptive field that covers the entire image. Having a global context allows the network to achieve better propagation of both structure and texture information. FFCs splits network channels into two parallel branches, local branch that uses conventional convolutions and global branch that uses real fast Fourier Transforms allowing the network to have a perceptual confidence in local and global information of the image. FFCs also has the advantage of reducing computational complexity, especially in the context of larger images, and thus increasing network efficiency significantly. It is interesting to note that convolution in the image domain is equivalent to multiplication in the frequency domain, thus simplifying the computation to a large extent during sampling. Also, as the Fourier spectrum is uniquely defined for a given function and de-convolution from frequency domain to image domain is significantly easier task to perform, there is effectively no loss of information during this process.

Images consists of pixels that are all represented in real-valued system which further add to the computational boost involving image convolutions as Real FFT uses only half of the spectrum compared to FFT. On one hand where conventional convolutional models struggle to learn global contexts in early layers of their networks, and waste computations and parameters to create one; FFCs allow for a receptive field that covers an entire image even in the early layers of the network. This not only improves perceptual quality but also parameter efficiency of the network, and thus is able to generalize to higher resolutions and unseen patterns apart from training dataset.

Fast Fourier Convolutions proceeds by applying a Real 2D Fast Fourier Transforms to an input image. It then concatenates its real and imaginary part to apply a convolution block coupled with ReLU activation in the frequency domain. Inverse Transform is then applied to recover the image spatial structure.

FFCs, thus avoid a significant computational overhead providing close to real synthesis of masked portions on the images.