main

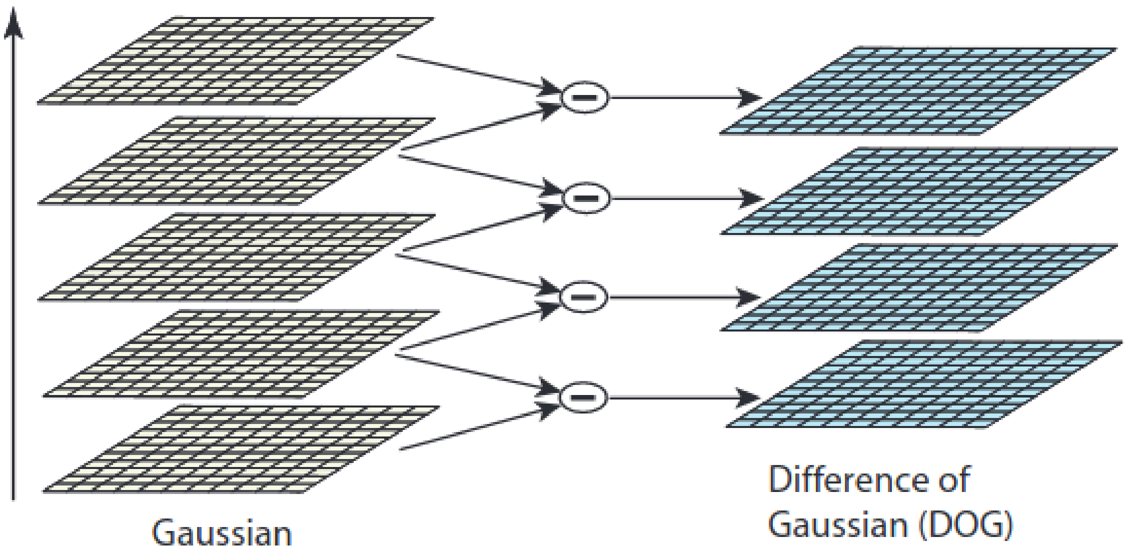
February 23, 2023

1 Laplacian of Gaussian with Difference of Gaussian

The Laplacian of Gaussian (LoG) can be used to detect blobs in images. Classic methods using the LoG blur the image with differing amounts to reduce the effect of noise, especially since the LoG requires second-order derivatives.

David Lowe found that the LoG can be approximated with a difference of Gaussians with increasing sigmas. Combining this with an image pyramid, he created the scale-invariant feature transform (SIFT).

This code shows a single octave for the SIFT algorithm.

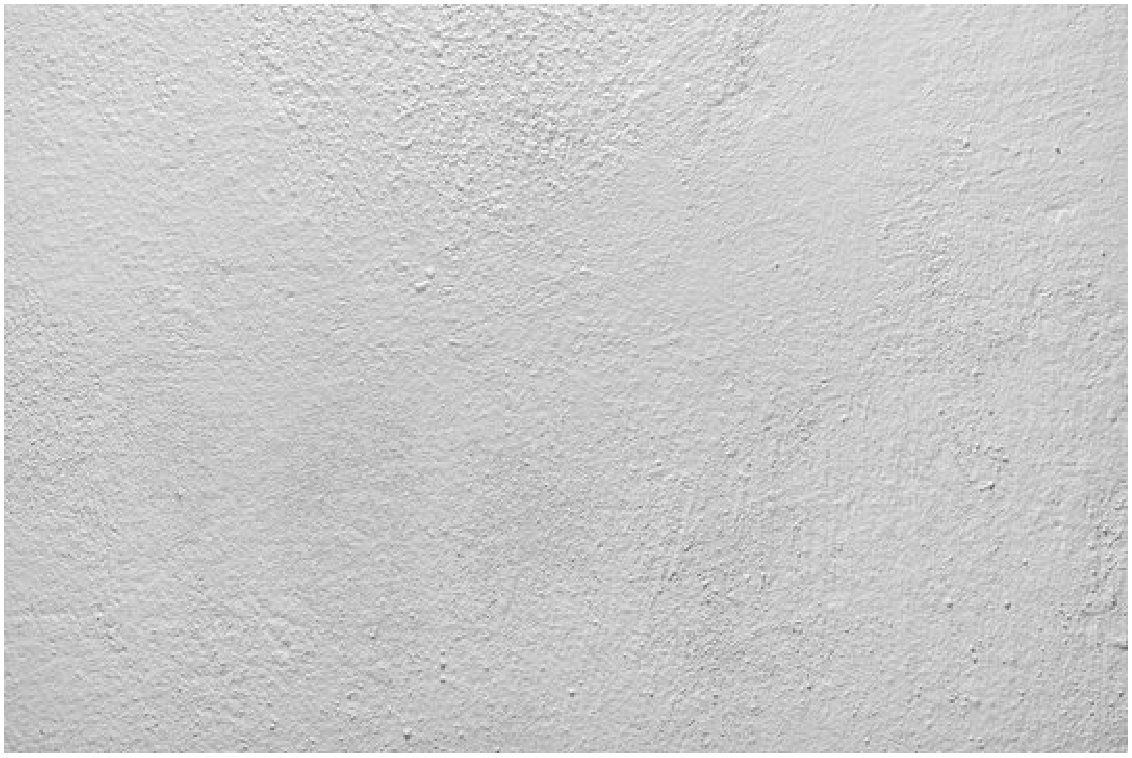


[1]:
```python
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt


image_path = 'wall.jpg'
# image_path = 'building.jpg'
# image_path = 'bigben.jpg'

resize = 1.0
```

```python
# Read in image as grayscale
image = Image.open(image_path).convert('L')

# Resize image
w = int(image.width * resize)
h = int(image.height * resize)
image = image.resize((w, h))

# Store as array
image = np.array(image)

print(f'Read in {image_path}. Shape: {image.shape}')
width = image.shape[1]
height = image.shape[0]
Image.fromarray(image)
```

```
Read in wall.jpg. Shape: (360, 540)
```

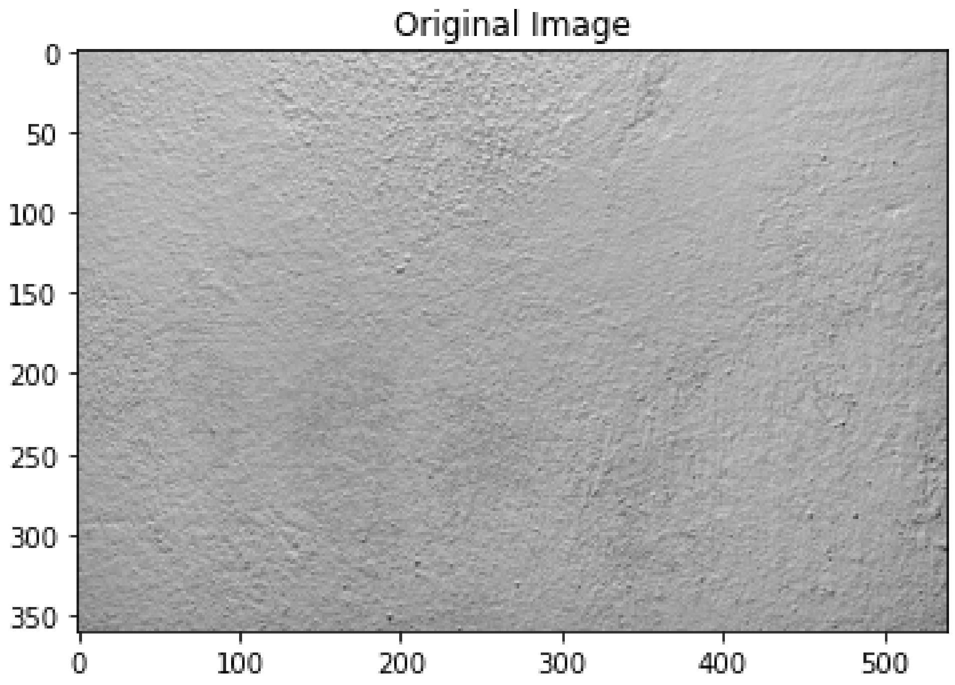
[1]:

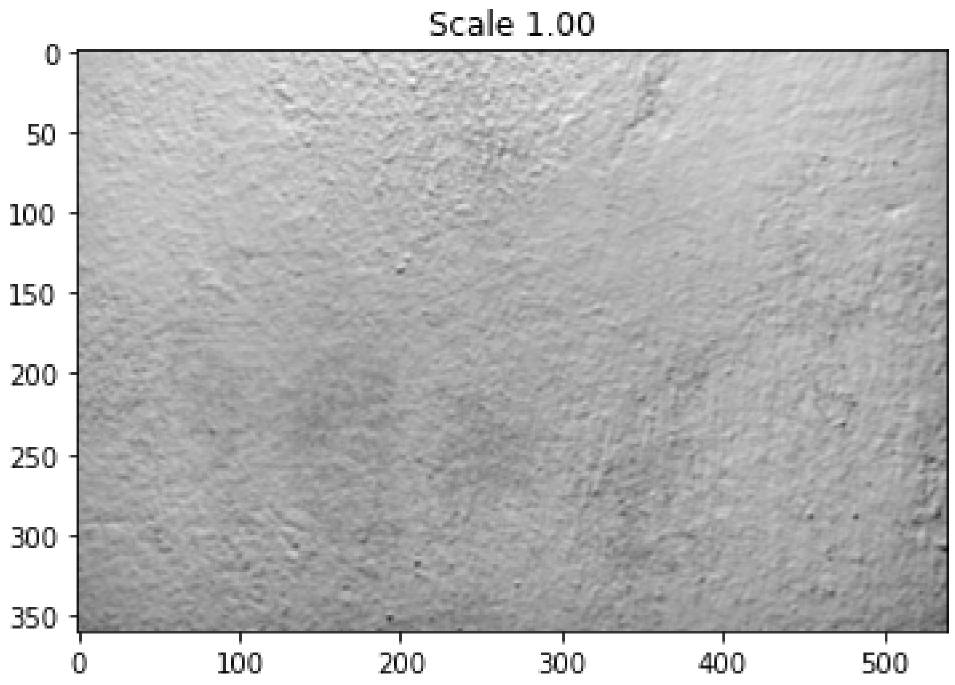


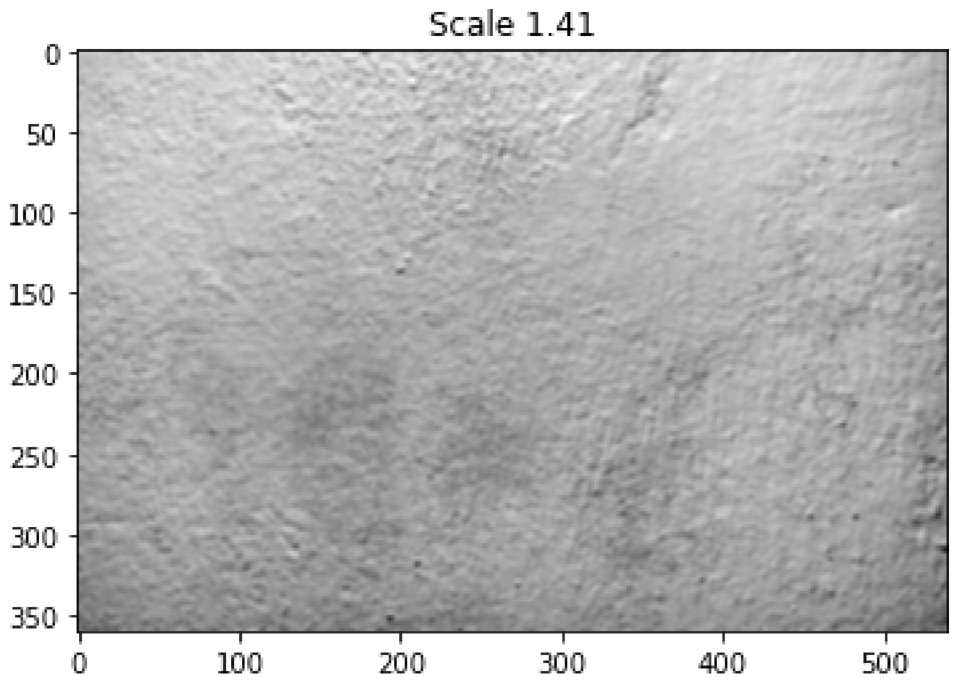
1.1 Scale space

In SIFT, an octave is defined as an image at a certain resolution, but with increasing Gaussian blurs applied. It's recommended to have at least 5 different scales within an octave.

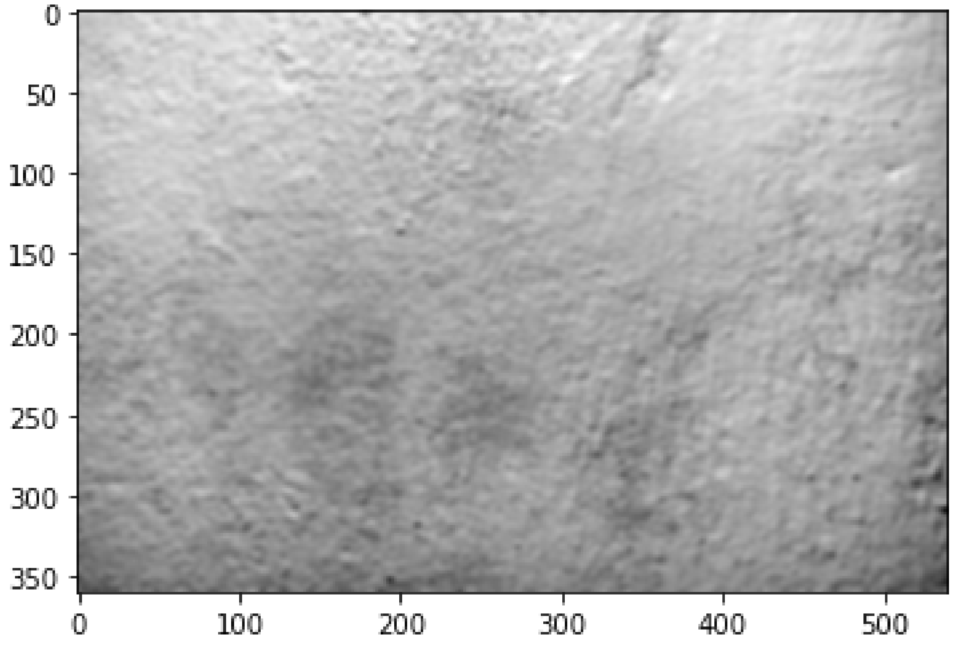
[2]:
```python
from scipy.ndimage import gaussian_filter
from matplotlib import pyplot as plt

scales = 5
scale_space = []
sigmas = [1]
multiplier = np.sqrt(2)

# Show original image
plt.figure()
plt.title('Original Image')
plt.imshow(image, cmap='gray')

for i in np.arange(scales):
    scale_space.append(gaussian_filter(image.astype(np.double), sigmas[i]))
    sigmas.append(multiplier * sigmas[i])

    # Show scale space within octave
    plt.figure()
    plt.title(f'Scale {sigmas[i]:.2f}')
    plt.imshow(scale_space[i], cmap='gray')
```



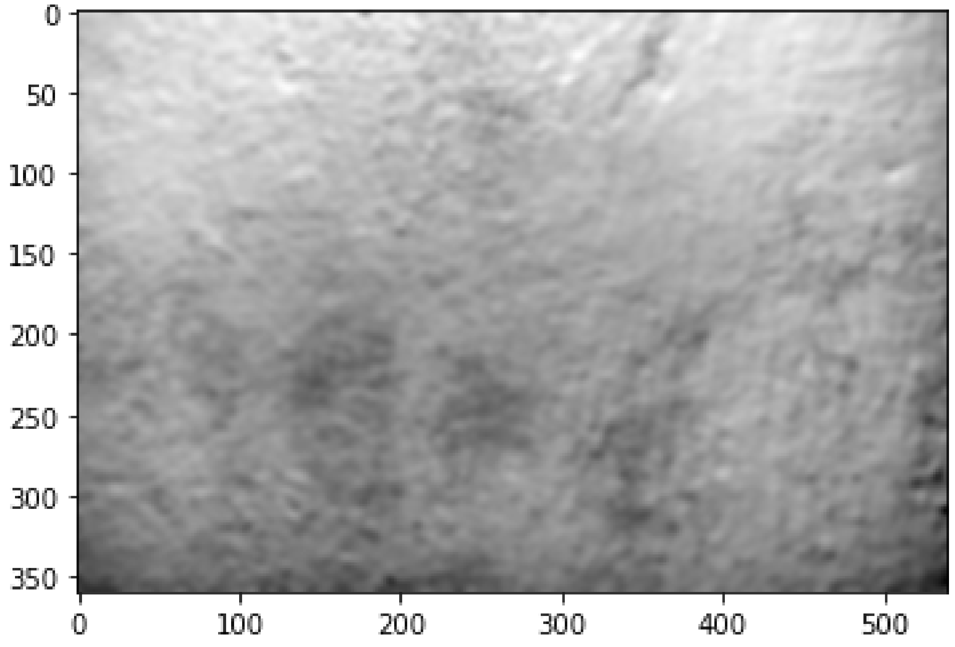

Scale 1.00
0
50
100
150
200
250
300
350
0
100
200
300
400
500

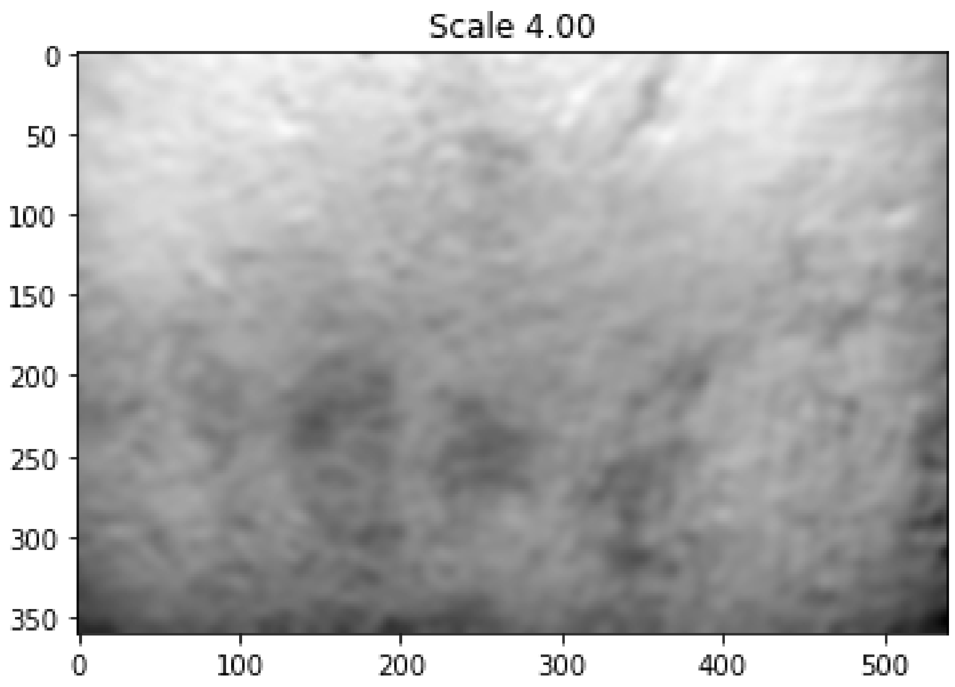

Scale 1.41
0
50
100
150
200
250
300
350
0
100
200
300
400
500

Scale 2.00



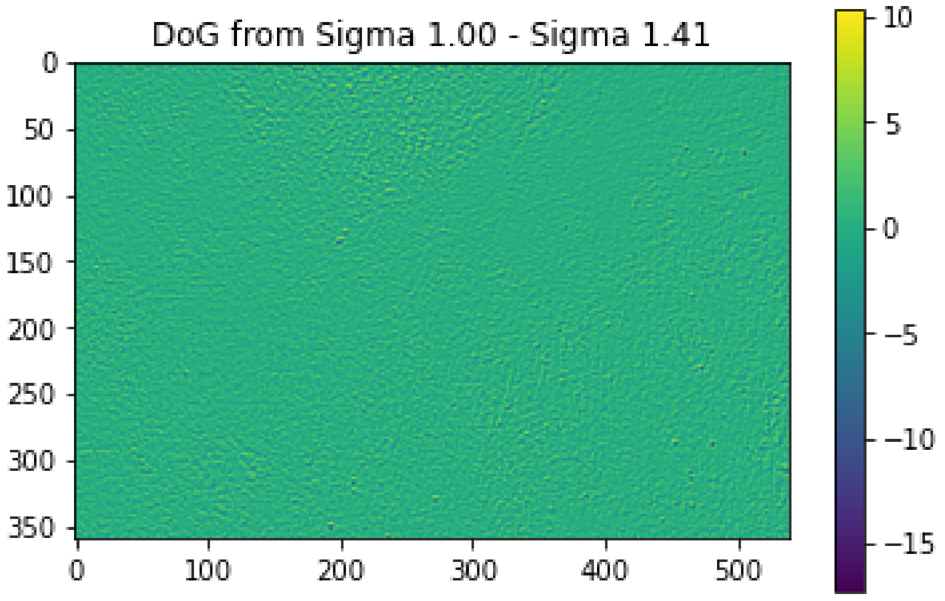
Scale 2.83

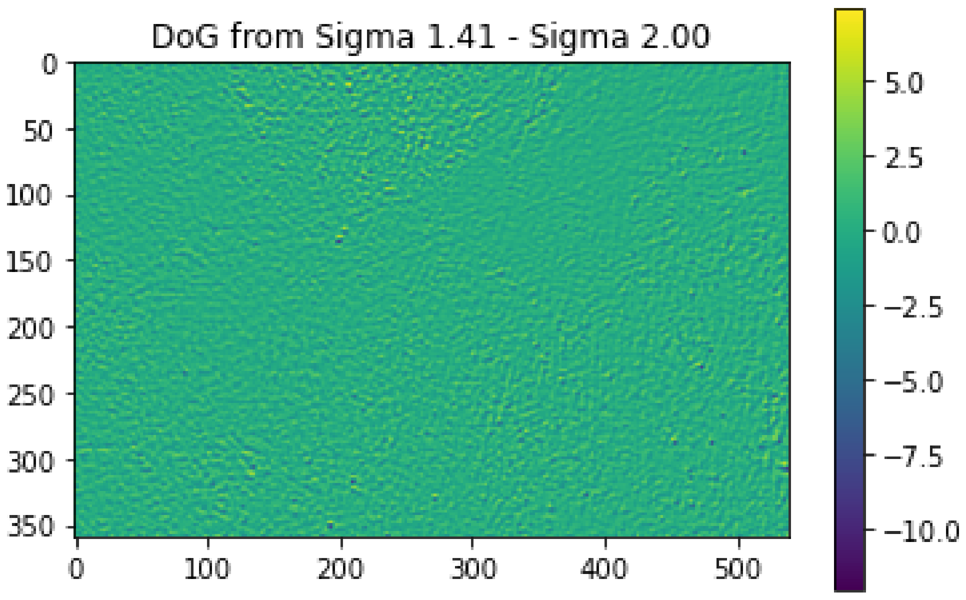


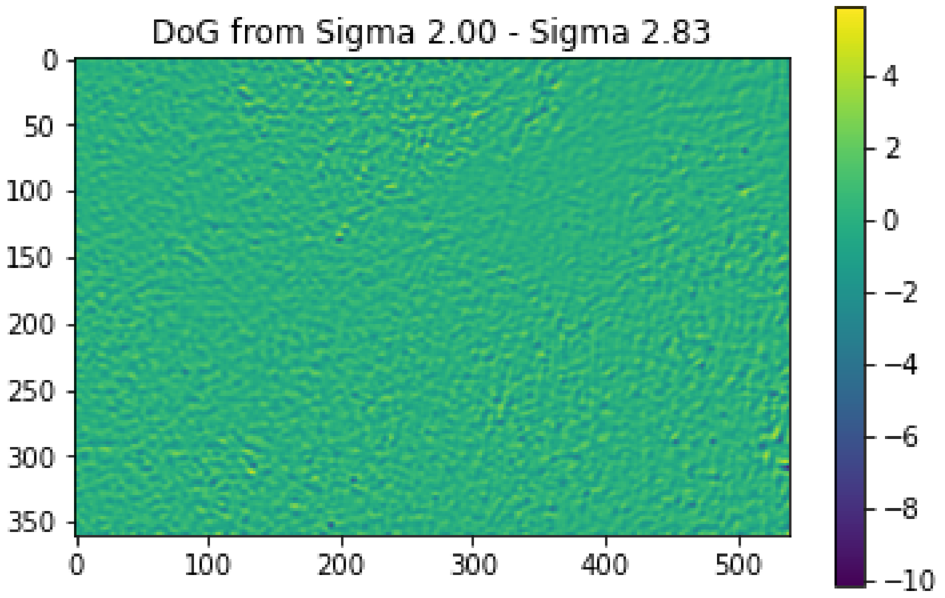


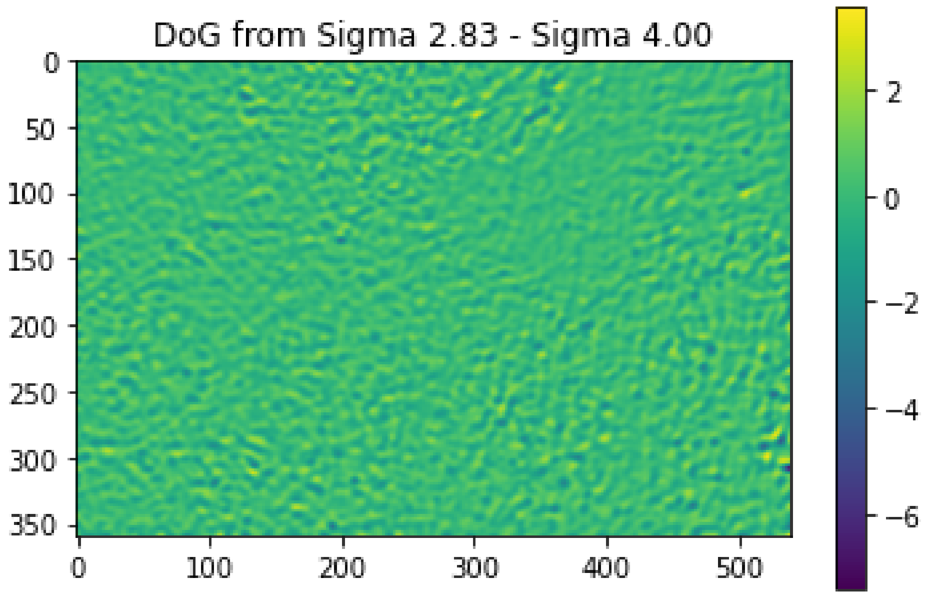
1.2 Difference of Gaussians

[3]:
```
DoGs = []

for i in np.arange(scales - 1):
    DoGs.append(scale_space[i] - scale_space[i + 1])
    plt.figure()
    plt.title(f'DoG from Sigma {sigmas[i]:.2f} - Sigma {sigmas[i + 1]:.2f}')
    plt.imshow(DoGs[i])
    plt.colorbar()
```


DoG from Sigma 1.00 - Sigma 1.41
0
50
100
150
200
250
300
350
0
100
200
300
400
500
10
5
0
−5
−10
−15

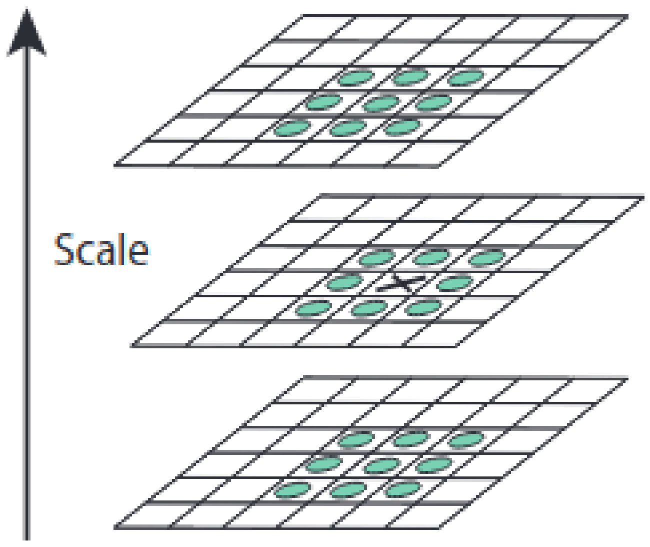

DoG from Sigma 1.41 - Sigma 2.00
0
50
100
150
200
250
300
350
0
100
200
300
400
500
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0





1.3 Optima in DoG

Find the optima across three scales. If a pixel is the maximum or minimum out of its 26 neighbors, it is marked as a keypoint. The smallest and largest scales will not be used for keypoints. Why is that?



[4]:
```python
# Turn list of numpy arrays into 3D matrix, but only if it's a list
# Rerunning this cell will permute the axes if it's already a numpy array
if type(DoGs) == list:
    DoGs = np.dstack(DoGs)

rows, cols = DoGs.shape[0:2]

keypoints = []

for i in np.arange(1, scales - 1):
    for row in np.arange(1, rows - 1):
        for col in np.arange(1, cols - 1):
            candidate = DoGs[row, col, i]

            neighborhood = DoGs[row-1:row+2, col-1:col+2, i-1:i+2]

            maxima = (candidate > neighborhood).sum()
            minima = (candidate < neighborhood).sum()

            if maxima == 26 or minima == 26:
               keypoints.append(np.array([row, col]))

print(f'{len(keypoints)} keypoints detected.')
```

```
189 keypoints detected.
```

1.4 Plot Keypoints

[5]:
```
import cv2

image_features = cv2.imread(image_path)
image_features = cv2.resize(image_features, (w, h))
image_features = cv2.cvtColor(image_features, cv2.COLOR_BGR2RGB)
for i in np.arange(len(keypoints)):
    image_features = cv2.circle(image_features,
                                (keypoints[i][1], keypoints[i][0]),
                                radius=3, color=(255,0,0), thickness=-1)

plt.figure(figsize=(15,15))
plt.xticks([])
plt.yticks([])
plt.imshow(image_features)
```

[5]:
```
<matplotlib.image.AxesImage at 0x7f17d09a2d40>
```