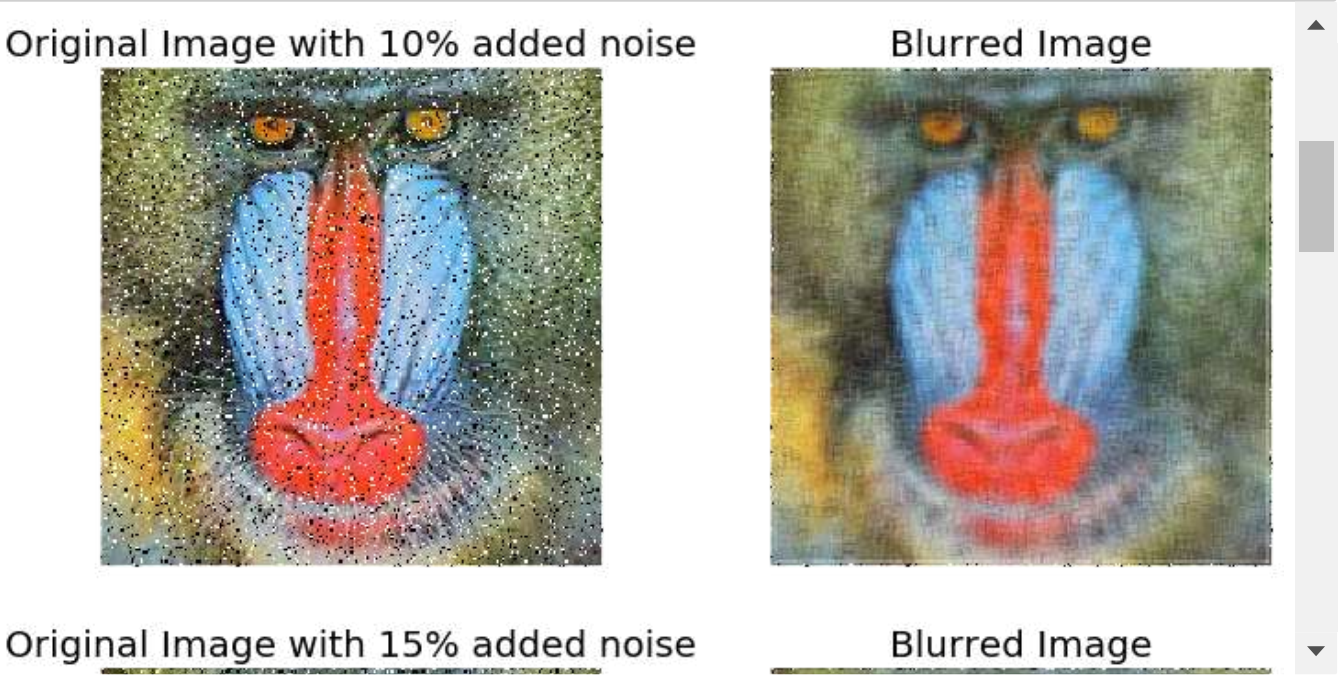
Chapter 5: Image Enhancement

Problems

1. Image Enhancement Filters with PIL for noise removal and smoothing

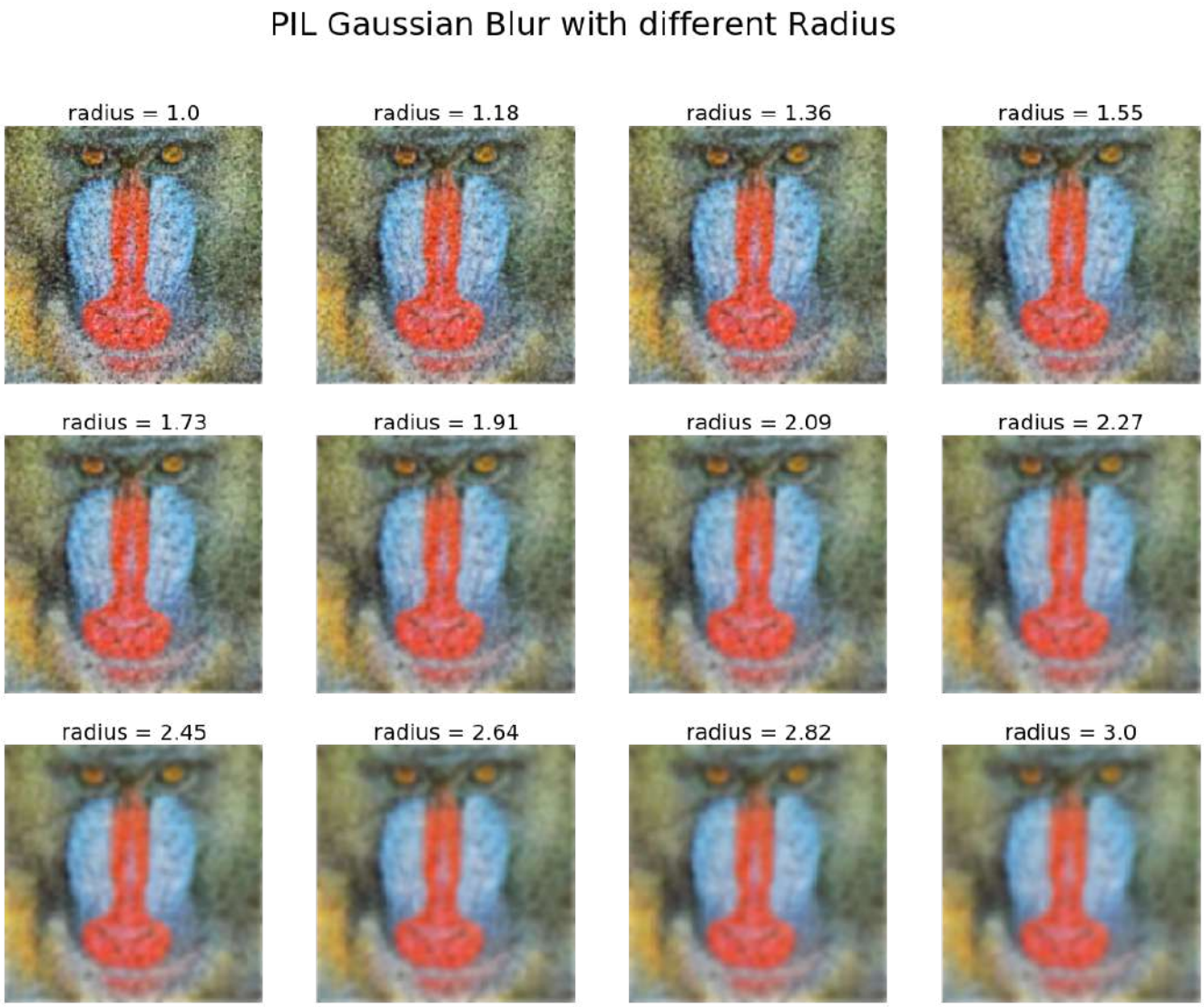
In [9]:
```
orig = Image.open('images/Img_05_01.jpg')
i = 1
plt.figure(figsize=(12,35))
for prop_noise in np.linspace(0.05,0.3,6):
 # choose random locations inside image
 im = add_noise(orig, prop_noise)
 plt.subplot(6,2,i), plot_image(im, 'Original Image with ' + str(int(100*prop_noise)) +
 im1 = im.filter(ImageFilter.BLUR)
 plt.subplot(6,2,i+1), plot_image(im1, 'Blurred Image')
 i += 2

plt.show()
```



1.2 Gaussian BLUR Filter to remove Salt & Pepper Noise

In [11]:
```
plt.figure(figsize=(20,15))
i = 1
for radius in np.linspace(1, 3, 12):
    im1 = im.filter(ImageFilter.GaussianBlur(radius))
    plt.subplot(3,4,i)
    plot_image(im1, 'radius = ' + str(round(radius,2)))
    i += 1
plt.suptitle('PIL Gaussian Blur with different Radius', size=30)
plt.show()
```



1.3 Median Filter to remove Salt & Pepper Noise

In [20]:
```
plt.figure(figsize=(20,10))
plt.subplot(1,4,1)
plot_image(im, 'Input noisy image')
i = 2
for sz in [3,7,11]:
 im1 = im.filter(ImageFilter.MedianFilter(size=sz))
 plt.subplot(1,4,i), plot_image(im1, 'Output (Filter size=' + str(sz) + ')', 20)
 i += 1
plt.tight_layout()
plt.show()
```

1.4 Max, Min and Mode filters to remove outliers from image

Min filter

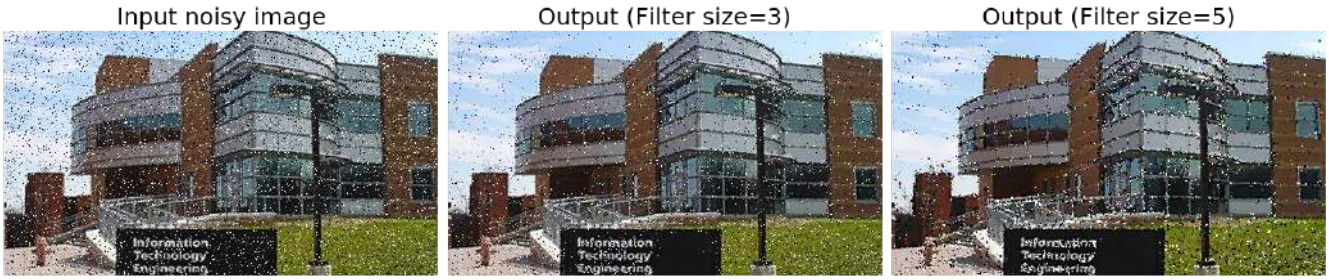
In [22]:
```
plt.figure(figsize=(20,10))
plt.subplot(1,4,1)
plot_image(im, 'Input noisy image')
i = 2
for sz in [3,7,11]:
 im1 = im.filter(ImageFilter.MinFilter(size=sz))
 plt.subplot(1,4,i), plot_image(im1, 'Output (Filter size=' + str(sz) + ')')
 i += 1
plt.tight_layout()
plt.show()
```

Max filter

In [187]:
```
plt.figure(figsize=(20,10))
plt.subplot(1,4,1)
plot_image(im, 'Input noisy image')
i = 2
for sz in [3,7,11]:
 im1 = im.filter(ImageFilter.MaxFilter(size=sz))
 plt.subplot(1,4,i), plot_image(im1, 'Output (Filter size=' + str(sz) + ')')
 i += 1
plt.show()
```

Mode filter

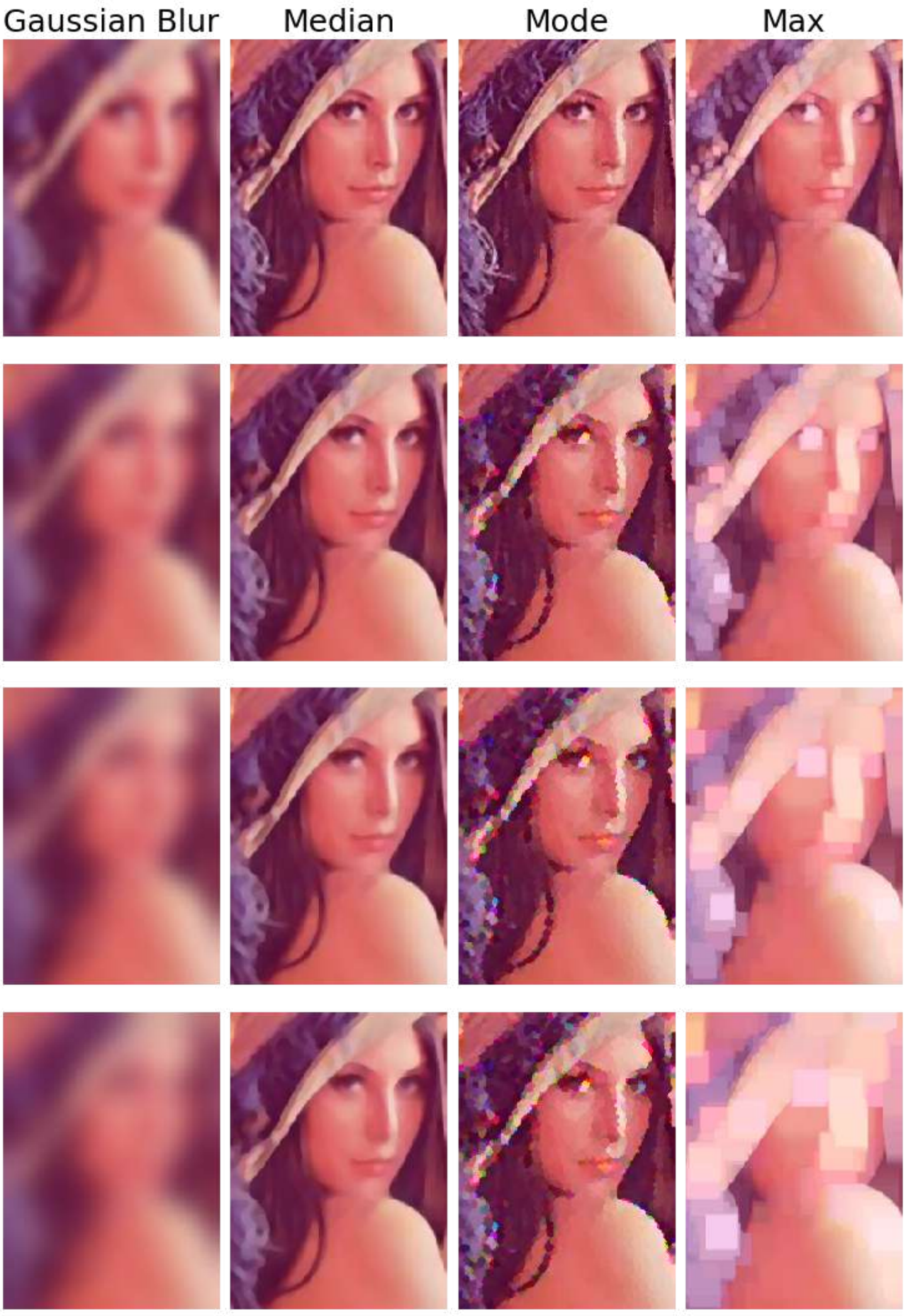
In [205]:
```
plt.figure(figsize=(20,20))
plt.subplot(1,3,1)
plot_image(im, 'Input noisy image', 25)
i = 2
for sz in [3,5]:
 im1 = im.filter(ImageFilter.ModeFilter(size=sz))
 plt.subplot(1,3,i), plot_image(im1, 'Output (Filter size=' + str(sz) + ')', 25)
 i += 1
plt.tight_layout()
plt.show()
```



1.5 Progressive Application of Gaussian Blur, Median, Mode and Max Filters on an image

In [26]:

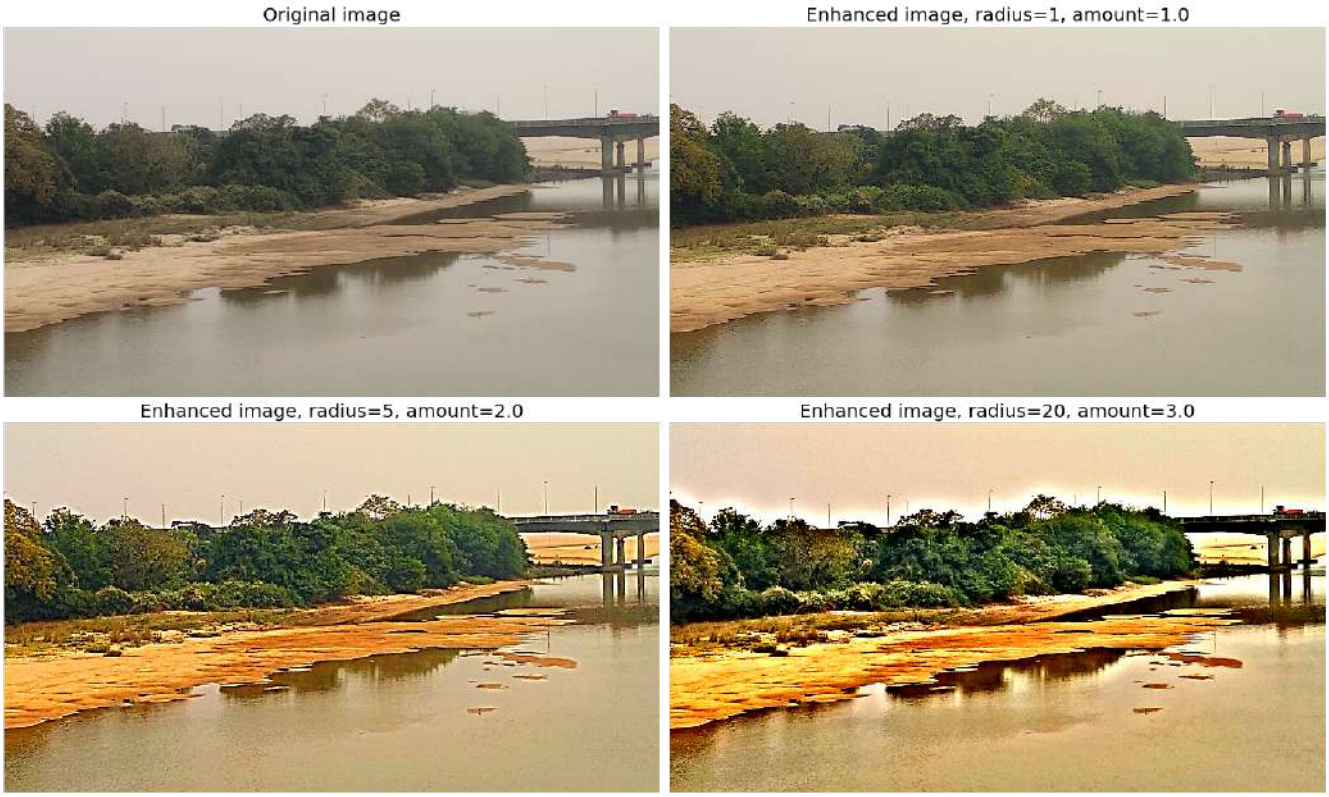
```
im = Image.open('images/Img_05_02.jpg')
plt.figure(figsize=(10,15))
plt.subplots_adjust(0,0,1,0.95,0.05,0.05)
im1 = im.copy()
sz = 5
for i in range(8):
 im1 = im1.filter(ImageFilter.GaussianBlur(radius=sz))
 if i % 2 == 0:
    plt.subplot(4,4,4*i//2+1), plot_image(im1, 'Gaussian Blur' if i == 0 else None, 25)
im1 = im.copy()
for i in range(8):
 im1 = im1.filter(ImageFilter.MedianFilter(size=sz))
 if i % 2 == 0:
    plt.subplot(4,4,4*i//2+2), plot_image(im1, 'Median' if i == 0 else None, 25)
im1 = im.copy()
for i in range(8):
 im1 = im1.filter(ImageFilter.ModeFilter(size=sz))
 if i % 2 == 0:
    plt.subplot(4,4,4*i//2+3), plot_image(im1, 'Mode' if i == 0 else None, 25)
im1 = im.copy()
for i in range(8):
 im1 = im1.filter(ImageFilter.MaxFilter(size=sz))
 if i % 2 == 0:
    plt.subplot(4,4,4*i//2+4), plot_image(im1, 'Max' if i == 0 else None, 25)
plt.show()
```



2. Unsharp masking to Sharpen an Image

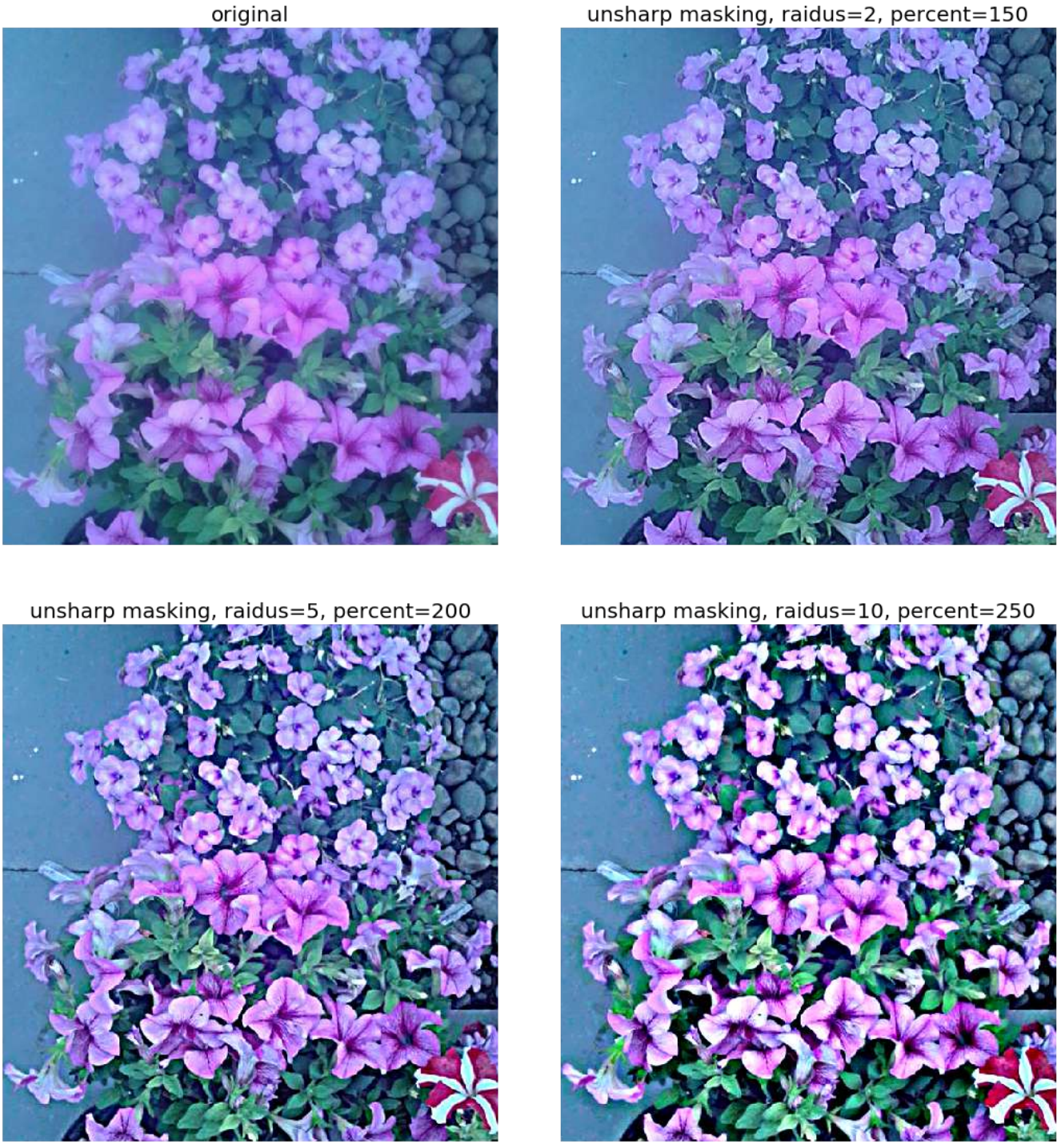
2.1 With scikit-image filters module

In [210]:
```
fig, axes = plt.subplots(nrows=2, ncols=2, sharex=True, sharey=True, figsize=(20, 12))
axes = axes.ravel()
axes[0].set_title('Original image', size=20), axes[0].imshow(im)
axes[1].set_title('Enhanced image, radius=1, amount=1.0', size=20), axes[1].imshow(im1)
axes[2].set_title('Enhanced image, radius=5, amount=2.0', size=20), axes[2].imshow(im2)
axes[3].set_title('Enhanced image, radius=20, amount=3.0', size=20), axes[3].imshow(im3)
for ax in axes:
 ax.axis('off')
fig.tight_layout()
plt.show()
```



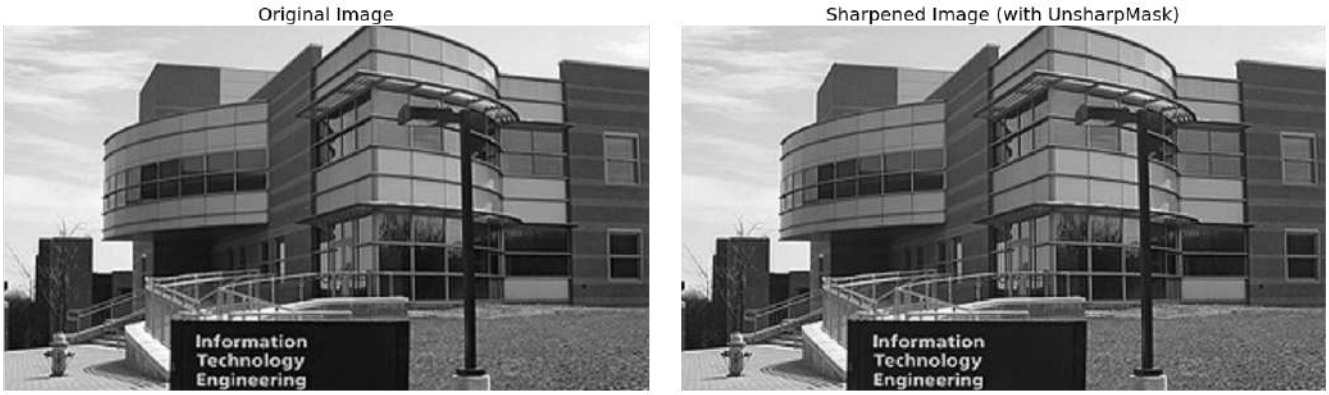
2.2 With PIL ImageFilter module

In [154]:
```
plt.figure(figsize=(15,16))
plt.subplot(221), plot_image(im, 'original')
im1 = im.filter(ImageFilter.UnsharpMask(radius=2, percent=150))
plt.subplot(222), plot_image(im1, 'unsharp masking, radius=2, percent=150')
im1 = im.filter(ImageFilter.UnsharpMask(radius=5, percent=200))
plt.subplot(223), plot_image(im1, 'unsharp masking, radius=5, percent=200')
im1 = im.filter(ImageFilter.UnsharpMask(radius=10, percent=250))
plt.subplot(224), plot_image(im1, 'unsharp masking, radius=10, percent=250')
plt.tight_layout()
plt.show()
```



2.3 Laplacian Sharpening with SimpleITK

In [108]:
```
plt.figure(figsize=(20,10))
plt.gray()
plt.subplots_adjust(0,0,1,1,0.05,0.05)
plt.subplot(121), plot_image(np_image, 'Original Image')
plt.subplot(122), plot_image(np_sharpened, 'Sharpened Image (with UnsharpMask)')
plt.show()
```

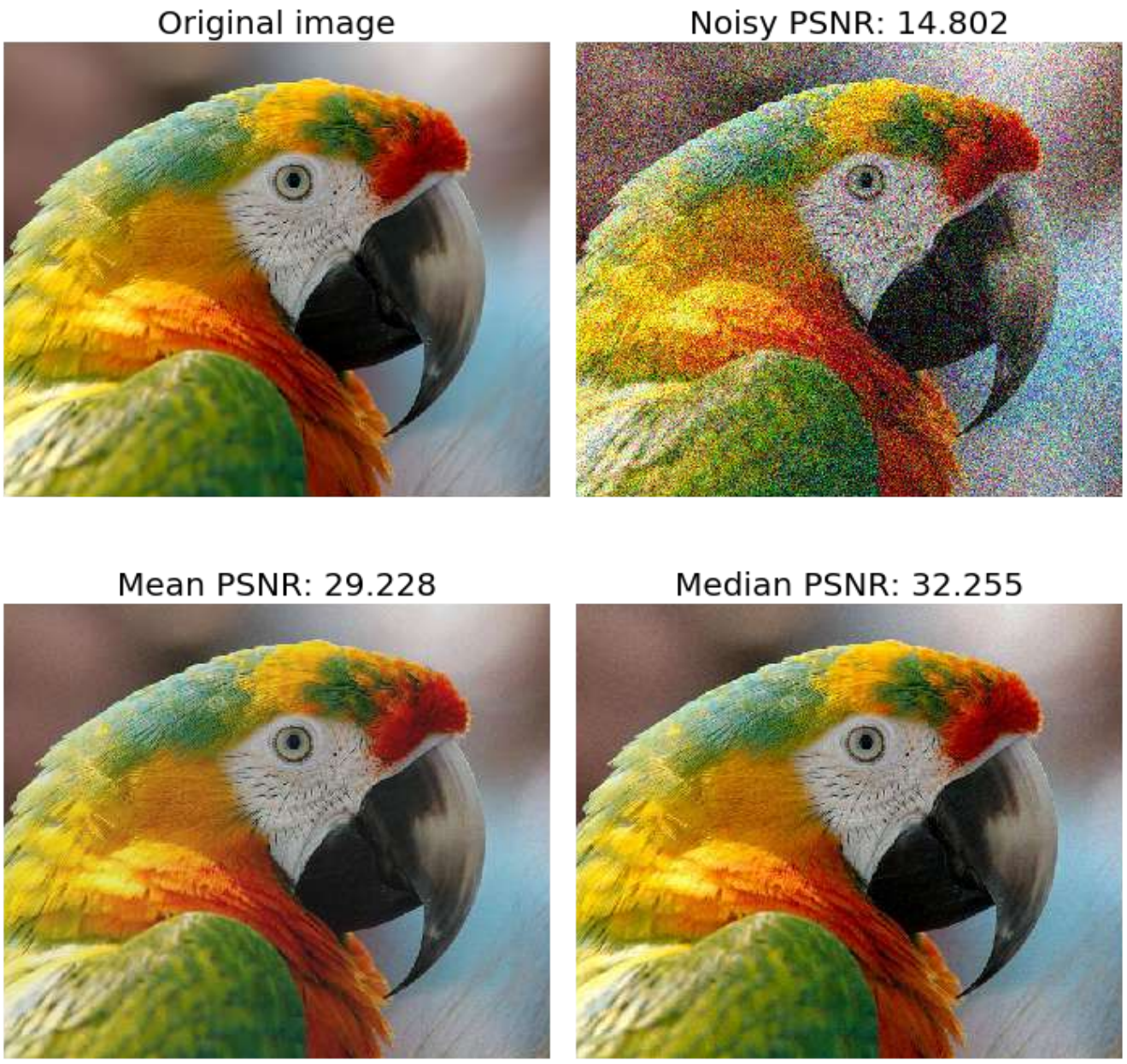


2.4 Implementing Unsharp Mask with opencv-python

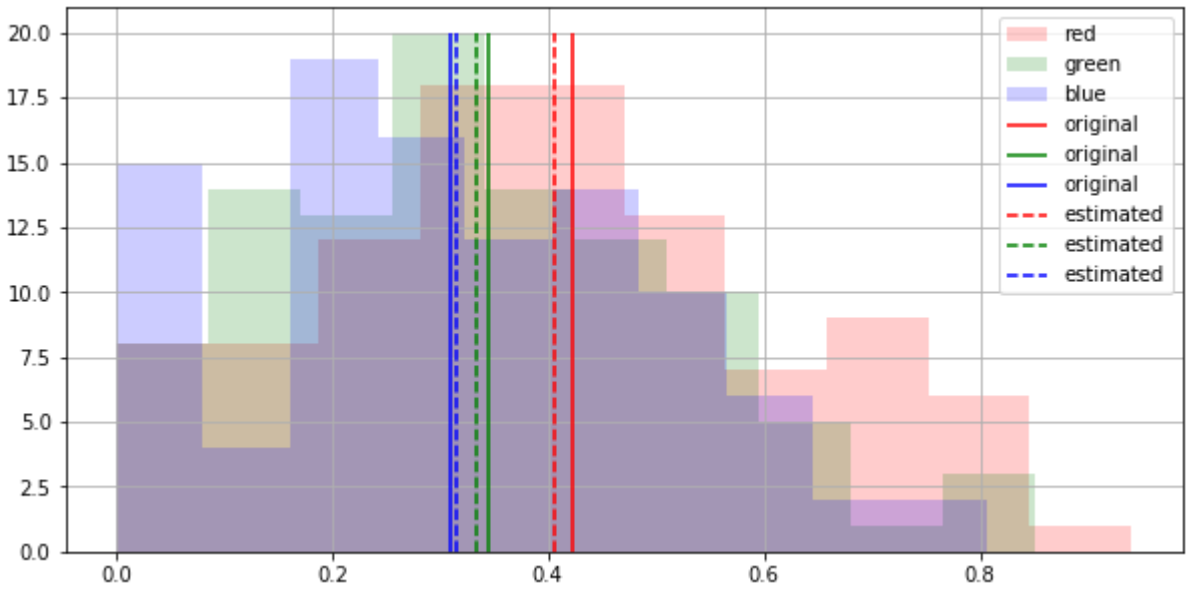
In [115]:
```
plt.figure(figsize=(20,25))
plt.subplots_adjust(0,0,1,0.95,0.05,0.05)
plt.subplot(211), plot_image(cv2.cvtColor(im, cv2.COLOR_BGR2RGB), 'Original Image')
plt.subplot(212), plot_image(cv2.cvtColor(im1, cv2.COLOR_BGR2RGB), 'Sharpened Image')
plt.show()
```

3. Averaging of Images to remove Random Noise

In [147]:
```
plt.figure(figsize=(10,10))
plt.subplots_adjust(left=.02, right=.98, bottom=.001, top=.96, wspace=.05, hspace=.01)
plt.subplot(221), plot_image(im, 'Original image')
plt.subplot(222), plot_image(images[0], 'Noisy PSNR: ' + str(round(peak_signal_noise_rat
plt.subplot(223), plot_image(im_mean, 'Mean PSNR: ' + str(round(peak_signal_noise_ratio(
plt.subplot(224), plot_image(im_median, 'Median PSNR: ' + str(round(peak_signal_noise_ra
plt.show()
```



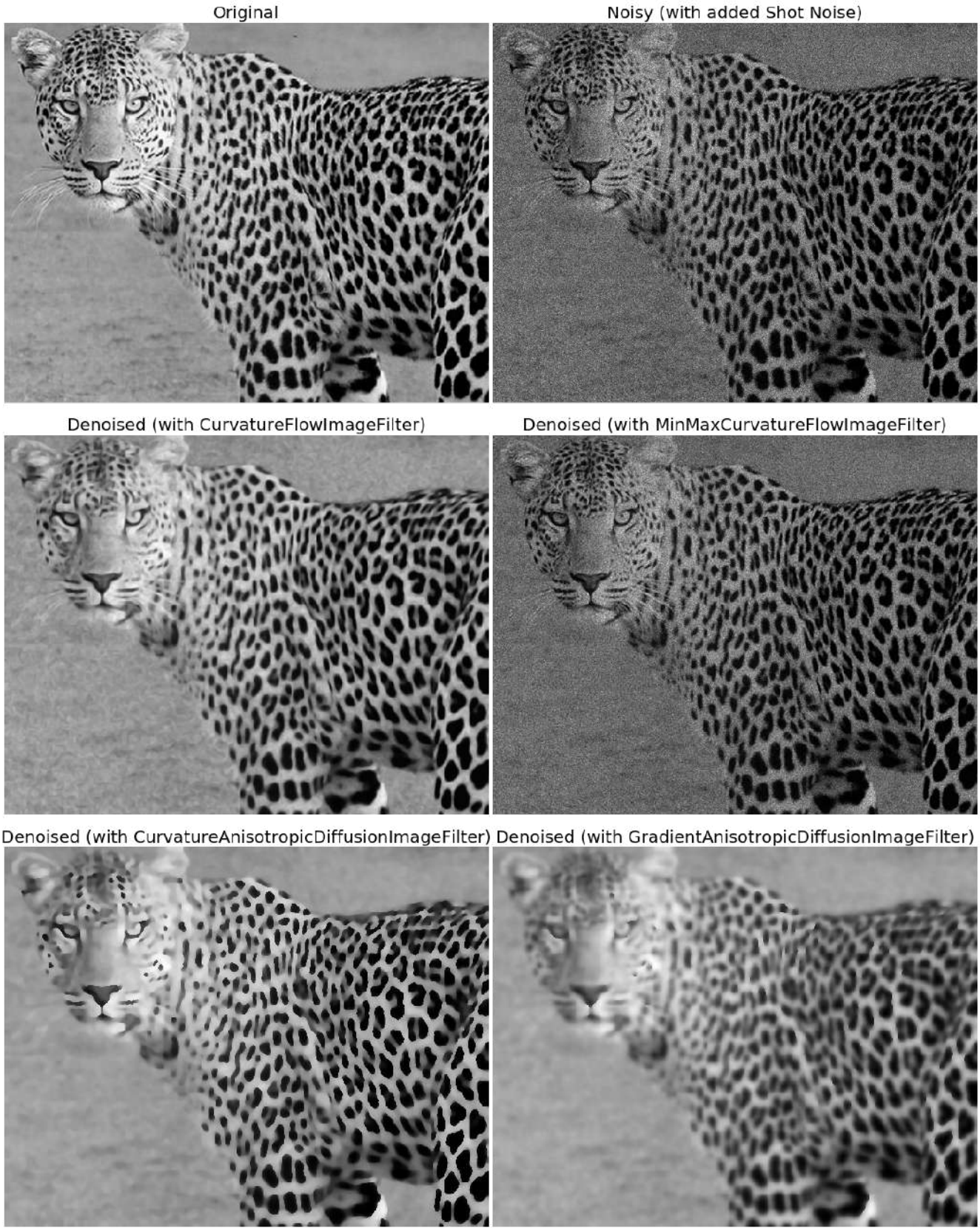
In [151]:
```
plt.figure(figsize=(10,5))
plt.hist(images[:,100,100,0], color='red', alpha=0.2, label='red')
plt.hist(images[:,100,100,1], color='green', alpha=0.2, label='green')
plt.hist(images[:,100,100,2], color='blue', alpha=0.2, label='blue')
plt.vlines(im[100,100,0], 0, 20, color='red', label='original')
plt.vlines(im[100,100,1], 0, 20, color='green', label='original')
plt.vlines(im[100,100,2], 0, 20, color='blue', label='original')
plt.vlines(im_mean[100,100,0], 0, 20, color='red', linestyles='dashed', label='estimated
plt.vlines(im_mean[100,100,1], 0, 20, color='green', linestyles='dashed', label='estimat
plt.vlines(im_mean[100,100,2], 0, 20, color='blue', linestyles='dashed', label='estimate
plt.legend()
plt.grid()
plt.show()
```



4. Image Denoising with Curvature-Driven Algorithms

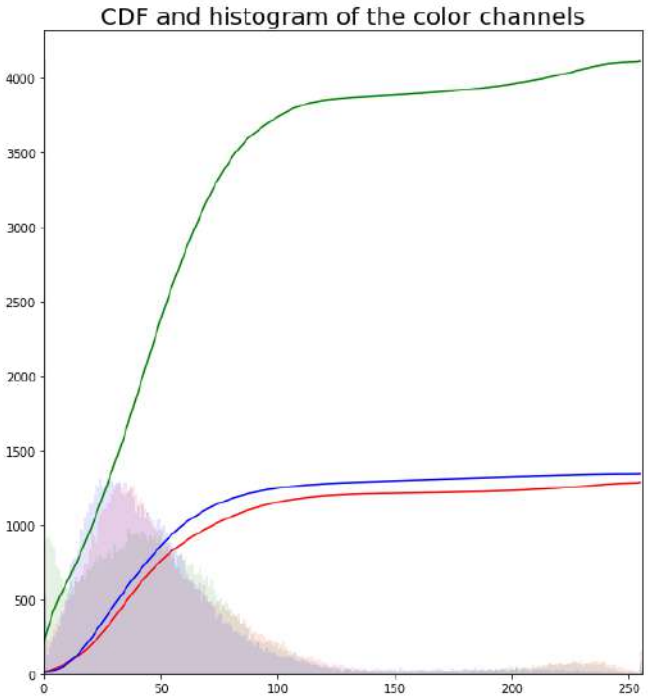
Anisotropic Diffusion

In [142]:
```
plt.figure(figsize=(16,20))
plt.gray()
plt.subplots_adjust(0,0,1,1,0.01,0.05)
plt.subplot(321), plt.imshow(sitk.GetArrayFromImage(img)), plt.axis('off'), plt.title('(
plt.subplot(322), plt.imshow(sitk.GetArrayFromImage(img_noisy)), plt.axis('off'), plt.ti
plt.subplot(323), plt.imshow(sitk.GetArrayFromImage(img_res_TK)), plt.axis('off'), plt.t
plt.subplot(324), plt.imshow(sitk.GetArrayFromImage(img_res_TK1)), plt.axis('off'), plt
plt.subplot(325), plt.imshow(sitk.GetArrayFromImage(img_res_TK2)), plt.axis('off'), plt
plt.subplot(326), plt.imshow(sitk.GetArrayFromImage(img_res_TK3)), plt.axis('off'), plt
plt.show()
```

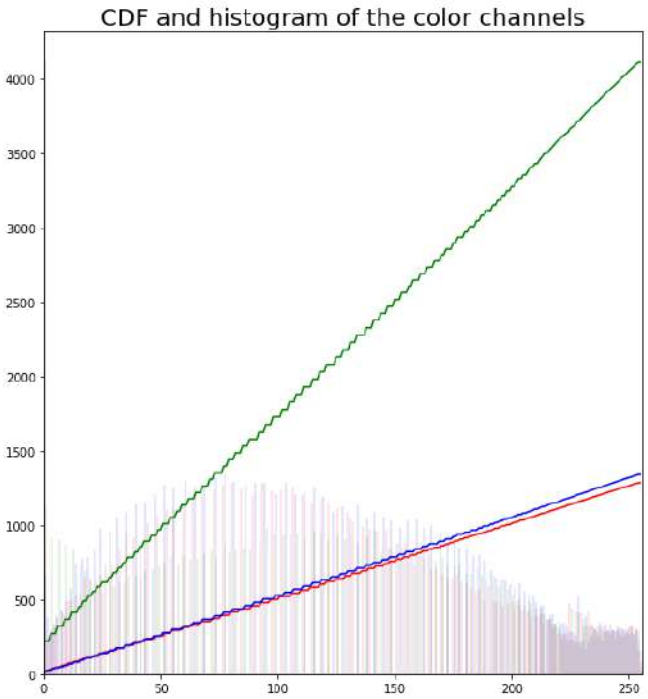


5. Contrast Strectching / Histogram Equalization with opencv-python

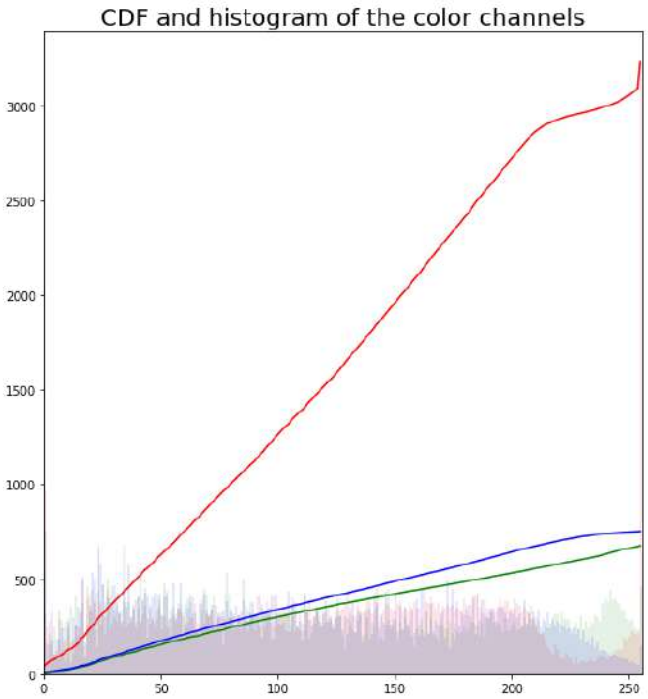
In [55]:
```
plot_img_hist(img, 'Original Image')
```



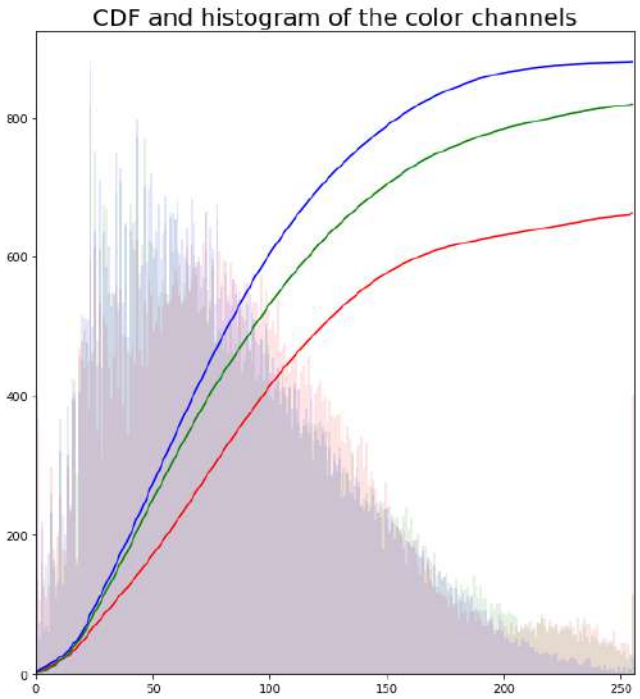
In [56]:
```
plot_img_hist(img2, 'Hist. Equalized')
```



In [57]:
```
plot_img_hist(equ, 'Hist. Equalized (LAB space)')
```



In [58]:
```
plot_img_hist(cl, 'Adaptive Hist. Equalized (LAB space)')
```



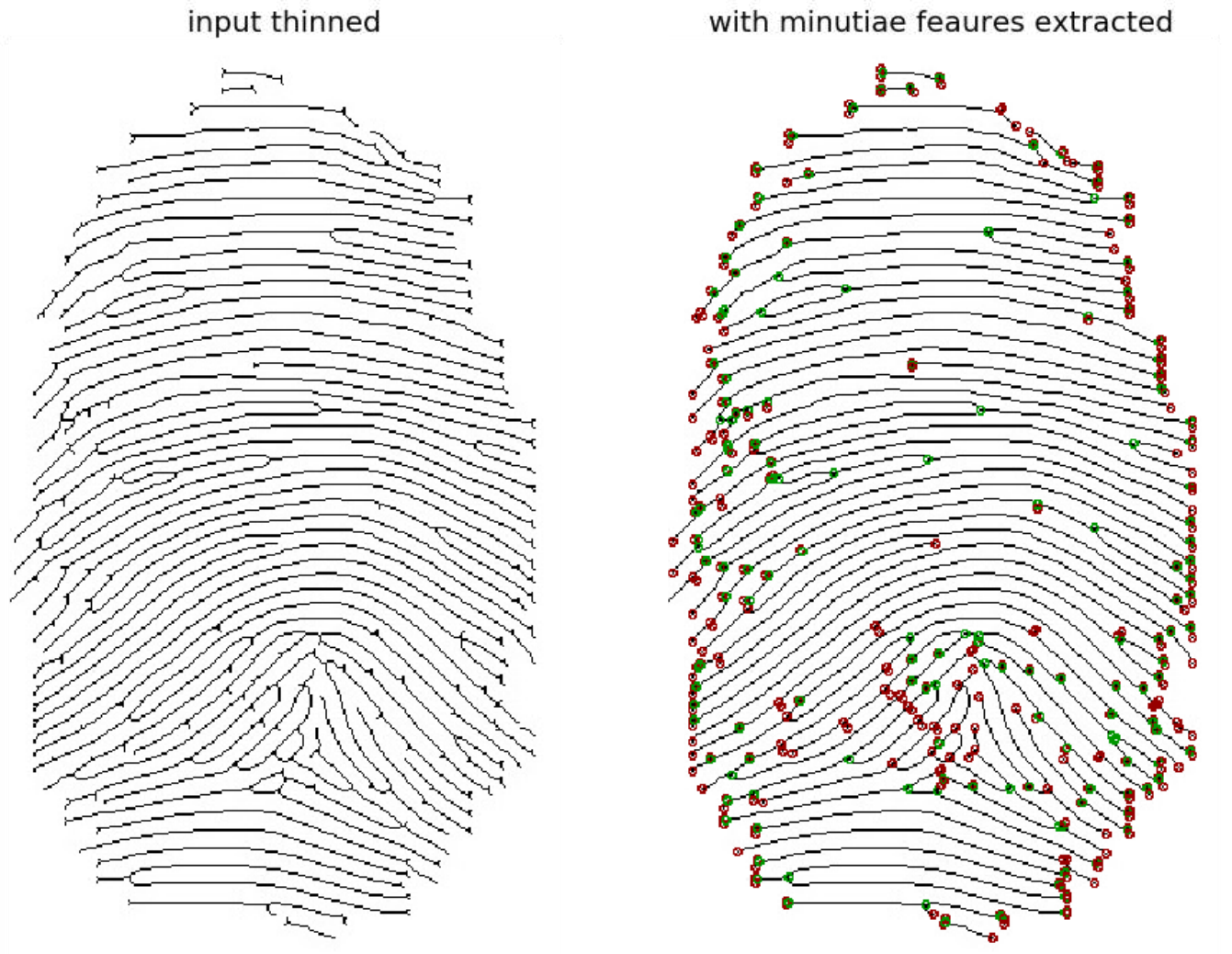
6. Fingerprint Cleaning and Minutiae feature extraction

6.1 Fingerprint Cleaning with Morphological operations

In [9]:
```
plt.figure(figsize=(20,12))
plt.gray()
plt.subplot(231), plot_image(im, 'original')
plt.subplot(232), plot_image(im_o, 'opening')
plt.subplot(233), plot_image(im_c, 'closing')
plt.subplot(234), plot_image(im_oc, 'opening + closing')
plt.subplot(235), plot_image(im_s, 'skeletonizing')
plt.subplot(236), plot_image(im_g, 'morphological gradient')
plt.show()
```

6.2 Feature (Minutiae) extraction from an enhanced fingerprint

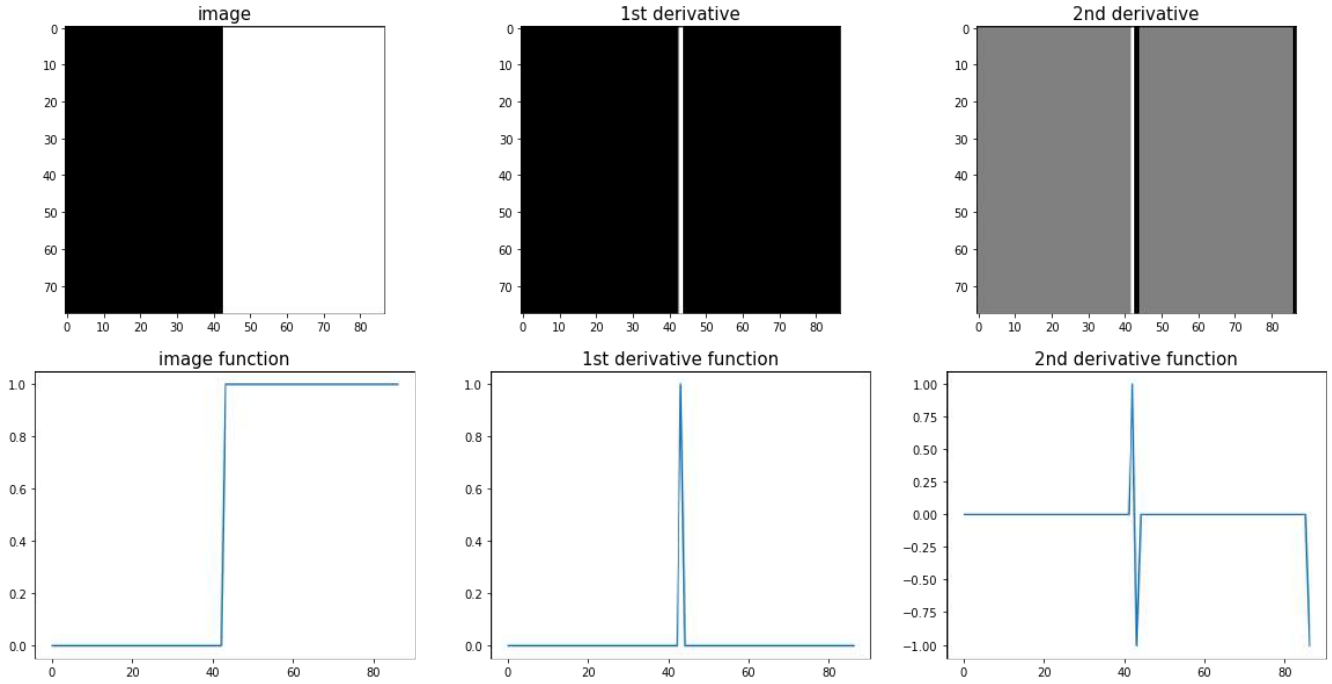
In [13]:
```
im = Image.open('images/Img_05_10.jpg').convert("L") # covert to grayscale
out = calculate_minutiae(im)
plt.figure(figsize=(15,12))
plt.gray()
plt.subplot(121), plot_image(im, 'input thinned')
plt.subplot(122), plot_image(out, 'with minutiae feaures extracted')
plt.show()
```



7. Edge Detection with LOG / Zero-Crossing, Canny vs. Holistically-Nested

7.0 Computing the Image Derivatives

In [18]:
```
plt.figure(figsize=(20,10))
plt.gray()
plt.subplot(231), plt.imshow(img), plt.title('image', size=15)
plt.subplot(232), plt.imshow(imgd1), plt.title('1st derivative', size=15)
plt.subplot(233), plt.imshow(imgd2), plt.title('2nd derivative', size=15)
plt.subplot(234), plt.plot(range(w), img[0,:]), plt.title('image function', size=15)
plt.subplot(235), plt.plot(range(w), imgd1[0,:]), plt.title('1st derivative function', s
plt.subplot(236), plt.plot(range(w), imgd2[0,:]), plt.title('2nd derivative function', s
plt.show()
```

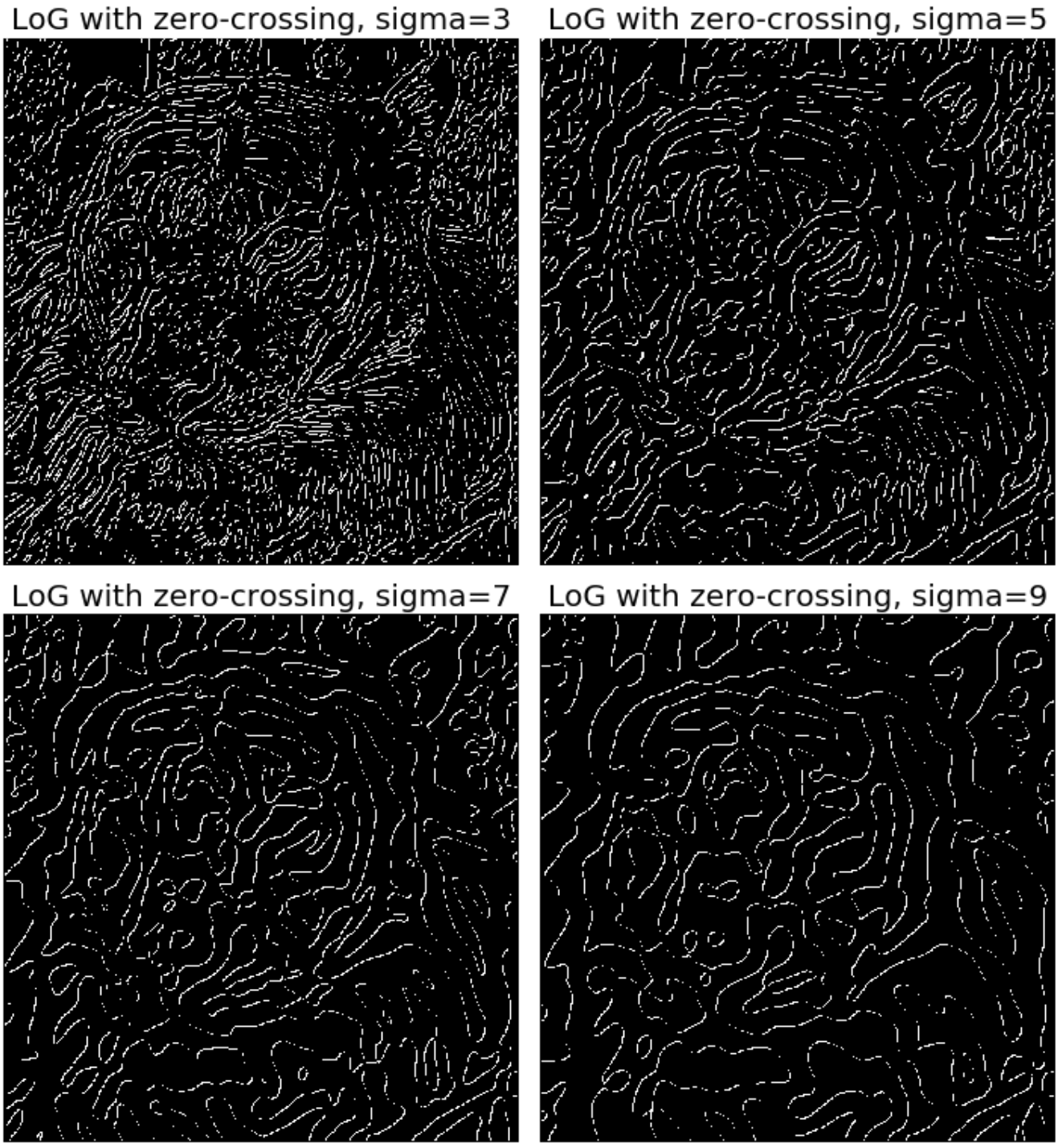


7.1 With LoG / Zero-Crossing

In [33]:
```
img = rgb2gray(imread('images/Img_05_18.jpg'))
#img = misc.imread('../new images/tagore.png')[...,3]
print(np.max(img))
fig = plt.figure(figsize=(10,16))
plt.subplots_adjust(0,0,1,0.95,0.05,0.05)
plt.gray() # show the filtered result in grayscale
for sigma, thres in zip(range(3,10,2), [1e-3, 1e-4, 1e-5, 1e-6]):
    plt.subplot(3,2,sigma//2)
    result = ndimage.gaussian_laplace(img, sigma=sigma)
    result = zero_crossing(result, thres)
    plt.imshow(result)
    plt.axis('off')
    plt.title('LoG with zero-crossing, sigma=' + str(sigma), size=20)

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

```
1.0
```



7.2 With Canny and Holistically-nested (deep learning model based)

In [38]:
```
plt.figure(figsize=(20, 8))
plt.gray()
plt.subplots_adjust(0,0,1,0.975,0.05,0.05)
plt.subplot(131), plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)), plt.axis('off'), p
plt.subplot(132), plt.imshow(canny), plt.axis('off'), plt.title('canny', size=20)
plt.subplot(133), plt.imshow(hed), plt.axis('off'), plt.title('holistically-nested', si
plt.show()
```