Computer Vision – Assignment 1

Ashwin Ramanathan

Q1 Image filtering and enhancement

Code: solution-part1.ipynb

1. The convolution of the given matrix with a mean filter is

```
array([[ 1.11111112,  2.55555557,  2.66666669,  2.88888891,  1.44444446],
       [ 1.8888889 ,  4.11111114,  4.33333337,  4.77777781,  2.55555557],
       [ 1.8888889 ,  4.22222225,  4.77777781,  4.88888893,  2.55555557],
       [ 1.44444446,  3.66666669,  4.22222225,  4.6666667 ,  2.44444446],
       [ 0.66666667,  2.11111113,  2.55555557,  2.7777778 ,  1.33333334]])
```

   The convolution was performed by padding the matrix with zero around the edges. As it is a mean filter, it can be seen that the high values , especially at the edges, have decreased ,and points with low values have a higher value after convolution.

2. Convolution with median filter

```
array([[ 0.,  2.,  1.,  2.,  0.],
       [ 2.,  4.,  5.,  6.,  2.],
       [ 2.,  4.,  5.,  7.,  2.],
       [ 1.,  4.,  4.,  4.,  3.],
       [ 0.,  1.,  1.,  3.,  0.]])
```

Convolution with a median filter replaces a point's values with the median of values in the kernel window, whereas a mean filter replaces a point's value with the avg of all values in the kernel window.

3. After applying the Sobel operator
   Gradient magnitude:

```
array([[  8.1297102 ,  14.08007336,  23.05065727,  23.05065727,  11.0991354 ],
       [  0.        ,   7.14435101,   8.1297102 ,   9.11767578,
         10.10763168],
       [  2.2859962 ,   2.2859962 ,   4.21399975,  11.0991354 ,
          4.21399975],
       [  8.1297102 ,  11.0991354 ,   5.18519354,   2.2859962 ,
          5.18519354],
       [  6.16245461,  16.07094383,  19.06056023,  12.09186172,
          7.14435101]], dtype=float32)
```

Gradient direction:

```
array([[ 1.44644129,  1.49948883,  1.52734542,  1.52734542,  1.48013639],
       [ 0.        ,  1.42889929,  1.44644129,  1.46013916,  1.47112763],
       [-1.10714877,  1.10714877, -1.3258177 , -1.48013639, -1.3258177 ],
       [-1.44644129, -1.48013639, -1.37340081, -1.10714877, -1.37340081],
       [-1.40564764, -1.50837755, -1.51821327, -1.48765504, -1.42889929]], dtype=float32)
```

Each point in the gradient magnitude matrix provides the magnitude of the gradient of the corresponding pixel point. Gradient magnitude is give by *Gmag = np.sqrt(Gx*Gx + Gy*Gy)* where Gx and Gy are partial derivative equivalents (w.r.t x and y).
Gradient direction is given by *GradDir = np.arctan(Gy,Gx)*
At the center: Gradient mag: `4.472136`
`Gradient dir: -75.96375885442686`

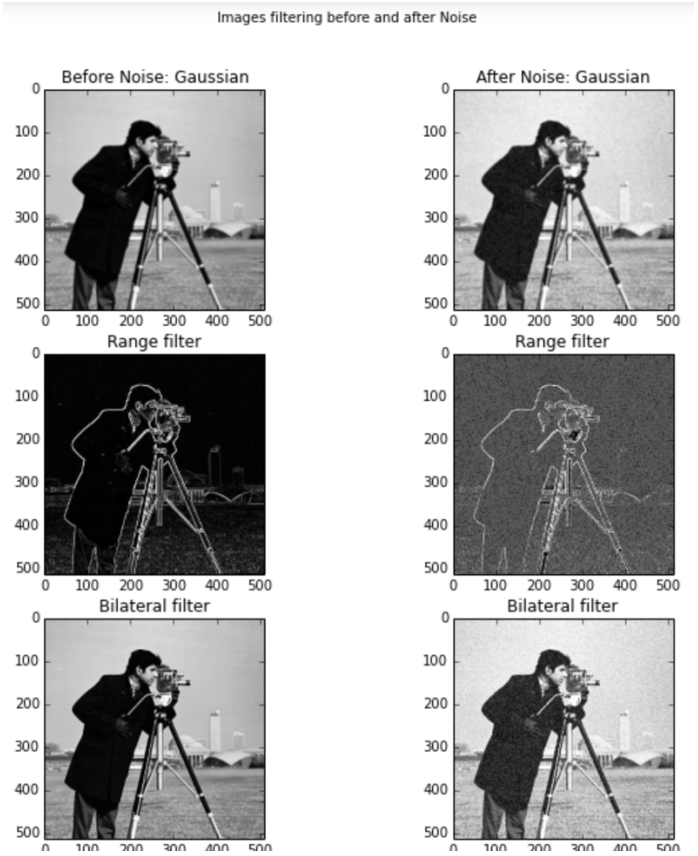
4. a. A Gaussian filter is a very good example of a distance based filter. The Gaussian kernel has peak values at the center and the values decrease gradually based on the kernel's sigma value. This can be seen in the below image.

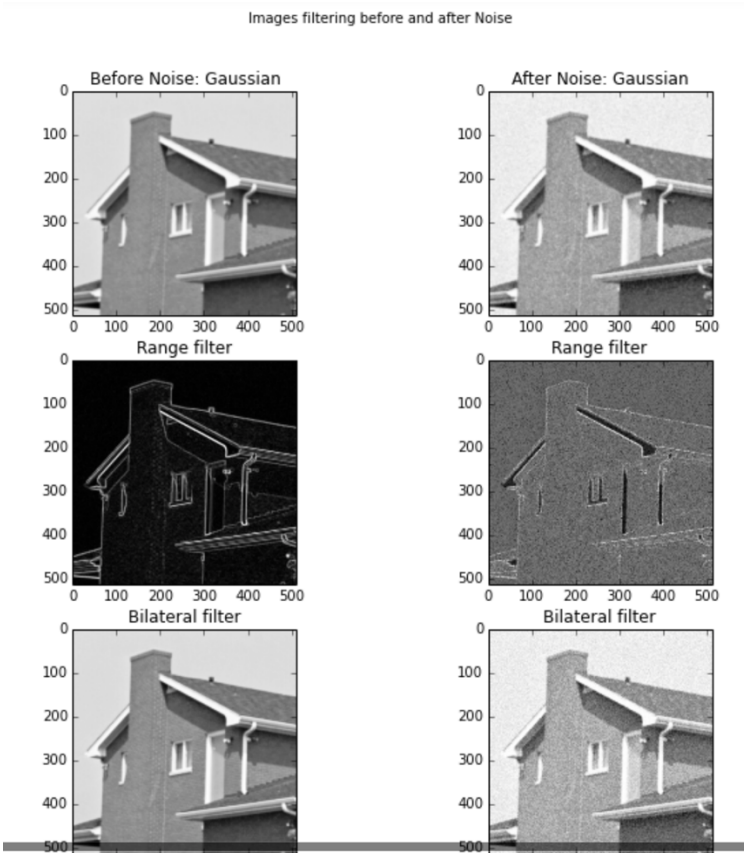
$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

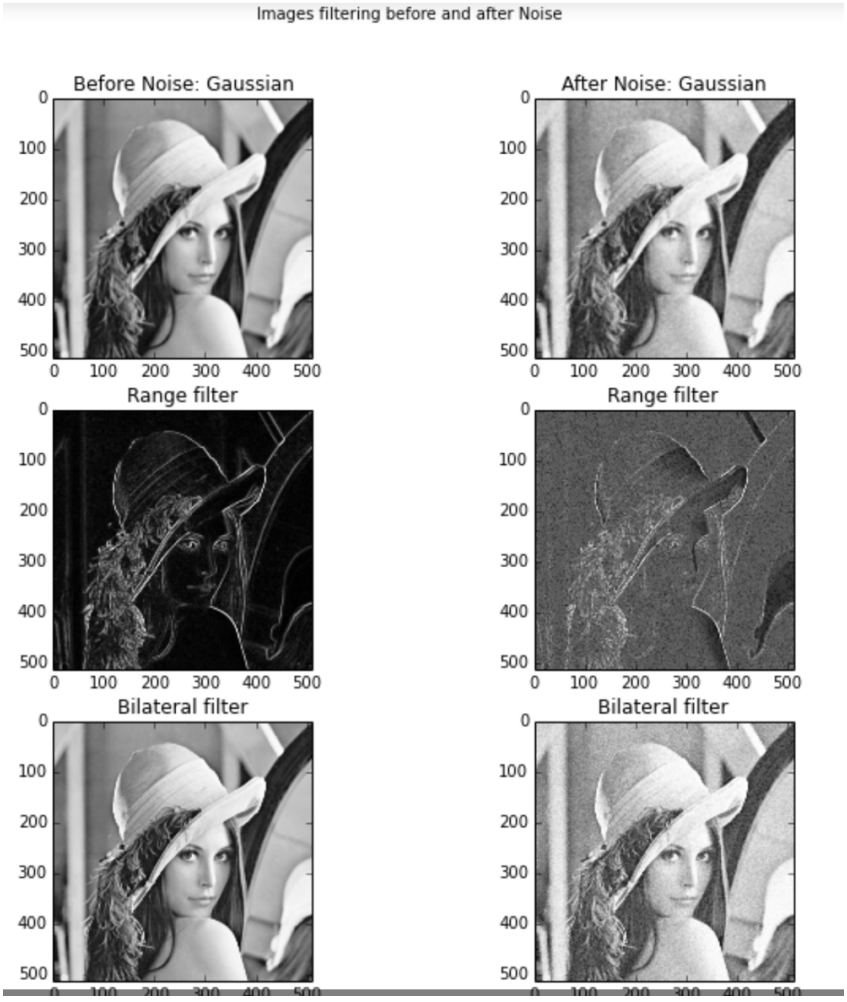
Hence when a convolution is performed, then values near to the image block's center (a *k x k* block with which convolution is performed in an iteration) are given higher weights as compared to pixels which are far away (like the corners), making this a distance based filter.

b. A range filter is an example of a filter which considers only the distance between pixel values. It replaces the pixel with the difference between maximum and minimum pixel values in its local neighborhood.

c. A bilateral filter is a good example of a filter which considers both distance as well similarity (or in a way, distance) between pixel values. Instead of taking a weighted average of pixels (like in the case of Gaussian) it considers a function of pixel value as well. Due to this, pixels which do not have similar values (for example, pixels which are on opposite sides of an edge) will have minimal impact on the pixel after convolution. This filter can be used to perform blurring while preserving edges at the same time. (Implementation used: skimage.restoration.denoise_bilateral)

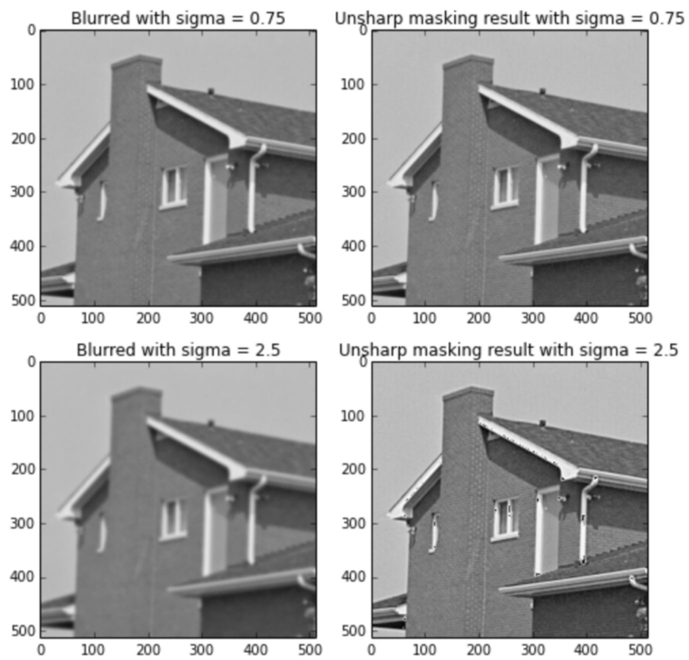

Images filtering before and after Noise
Before Noise: Gaussian
After Noise: Gaussian
Range filter
Range filter
Bilateral filter
Bilateral filter

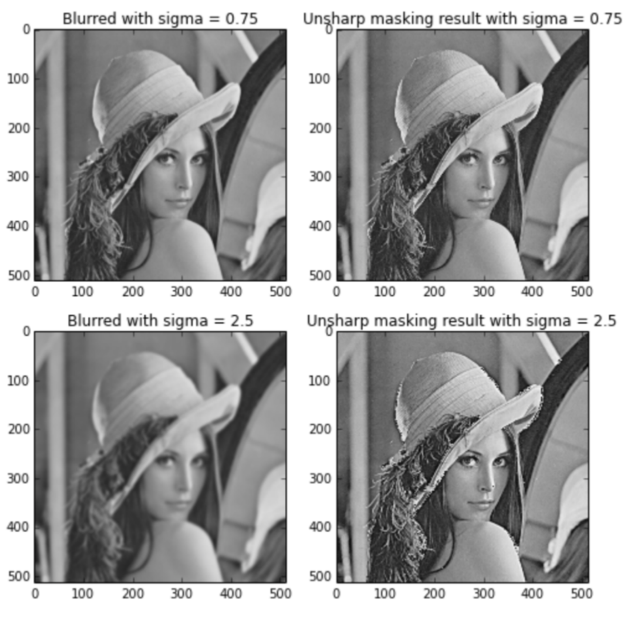

Images filtering before and after Noise
Before Noise: Gaussian
After Noise: Gaussian
Range filter
Range filter
Bilateral filter
Bilateral filter


Images filtering before and after Noise
Before Noise: Gaussian
After Noise: Gaussian
Range filter
Range filter
Bilateral filter
Bilateral filter

The images in the previous page display the effect of applying the filters to the three images. In all the three sets of images, it can be seen that

a. Gaussian and Bilateral filter are successful in decreasing noise to quite an extent. Although, images after bilateral filtering have sharper edges than Gaussian filtered images.
b. Range filter works like an edge detector, and the edge identification capabilities of this filter is not impacted by noise. However, the range filter (and by extension, any pixel value based filter) is not very capable in removing noise from an image.





5. For this part, the task was to apply unsharp masking on the image. Unsharp masking is a process which is often used to slightly sharpen images. The process:
   a. The image is blurred (usually using a Gaussian filter) and subtracted from the original image. This leaves behind those parts of the image which have comparatively higher intensity values, often, near of around the edges.
   b. The result of the previous step is added to the image again (often with a multiplication factor). This amplifies the regions with high intensity, and makes the image look sharper.

   The effect of unsharp masking can be seen in the image. The images with unsharp masking seem to have higher contrast and sharper edges.

Q2. Color quantization with k-means

Code: solution-part2.ipynb

a. Function name: quantize(img,k)
   This function takes the image as an input, converts it into an array of dimension rows*cols,3 and then performs kmeans clustering with k clusters. The result of the prediction is the cluster to which each point is assigned, so a new array is created which replaces the value of each pixel (r,g &b) with the value of the centroid of the cluster to which it belongs. This array is converted to the shape of the input image and returned.

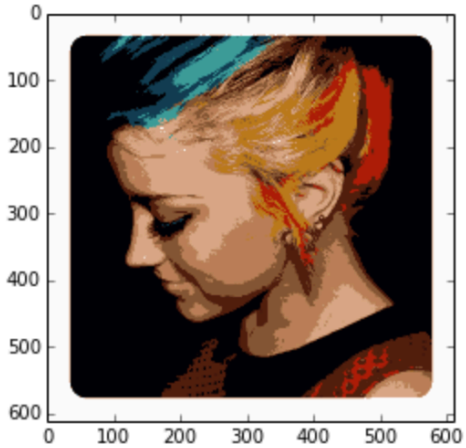


Fig 1 : quantized version of Colorful1.jpg (k=10)

After quantization, it can be seen that parts of the image with similar colors have been replaced with a single color.

b. Function name: labQuantizeAndCombine(img,k)
   The rgb image is converted to lab using skimage.rgb2lab. The image was converted to 0-1 scale before conversion.

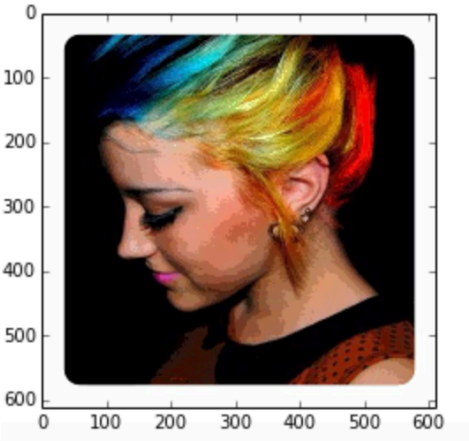


Fig2: L quantized version of Colorful1.jpg(k=10)

c. Function name: calculateSSD(im1, im2)
   This function calculates the SSD by summing the square of differences in pixel values (r,g &b) of image 1 and image2. Since both images are expected to be of the same size (this function will be used to calculate SSD between the original and quantized image), conditions of unequal sizes haven't been handled.

d. Function name: plotHists(img,k)
   This function first performs quantization of the l channel and then plots the histograms.

e. Function name: compileResults(img, k)

Differences before and after quantization:

Quantization is performed by applying KMeans to the given image. After quantization, pixels with similar values or features (rgb values in case 1, L values in case 2) are replaced with the cluster center value to which they belong. Hence, this operation, in a way, suppresses information which impacts the visual appearance of the image post operation.

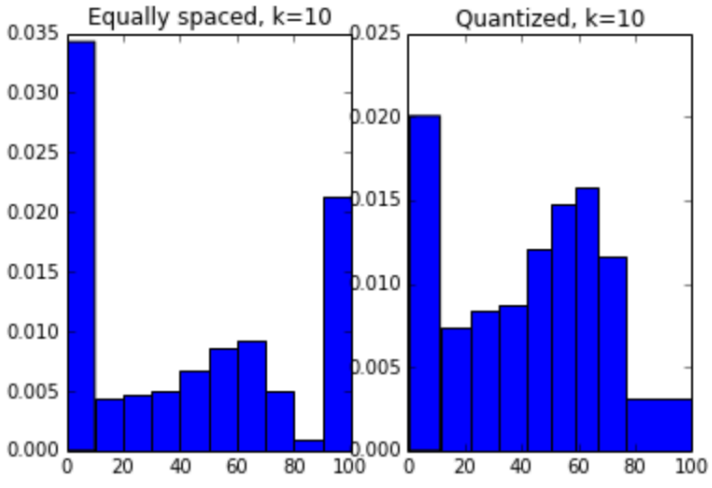
It can be seen that the Fig2 is more realistic and closer to the original image than Fig1. This could be because:

i. *labQuantizeAndCombine* quantizes only the L channel without changing a and b channels. This implies lesser impact when

compared to rgb quantization. This is also seen by comparing SSD values of rgb_quantized and l_quantized images. L_Quantized images have a much lower SSD value than RGB quantized images.

i. the L channel is mainly the indicator of lightness. Pixel with similar values in the L channel or pixels with similar luminosity according to human perception will get the same cluster. Hence, when they are converted to RGB they much closer to the original image in comparison to RGB quantized image (for the same k values).

```
ssd_rgbQuantized:: 3946.90383943
ssd_lQuantized:: 529.136391046
```




```
ssd_rgbQuantized:: 2651.98958983
ssd_lQuantized:: 243.440839976
```

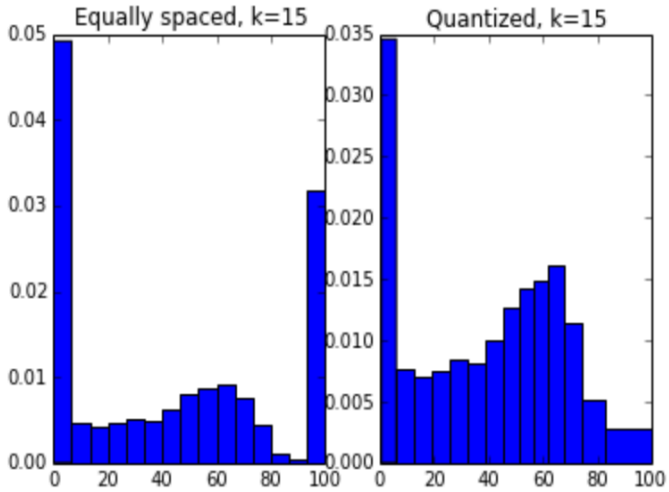


Fig 3: Histograms of L channel for colourful1.jpg for different values of k

From the above histograms it can be seen that, L channel values are more evenly distributed around cluster centers than uniform distribution, which is expected.

With increasing K:

With increase in k, the quality of image increases. This is because there are more number of values (or cluster centers) a pixel can be mapped to, and hence increasing the chances of the image resembling the original image. At high values of K, both the images will closely resemble each other, but the quantized image might look smoother, because nearby pixels might have the exact same values, which might not be the case with the original image.

Different iterations of the program can yield slightly different results because the cluster center could be different (Kmeans cluster centers depend on the initial seed values).

Part 3: Edge Detection

A.1
The underlying principle in finding edges is that the intensity varies at the edges. This variation is successfully captured by the partial derivatives (w.r.t x and y axes). All the three edge detection methods mentioned below use partial derivatives.

Sobel filtering: Sobel filter obtains partial derivatives in x and y direction and returns a value which is equal to the sum of its squares. In other words, the Sobel filter replaces every pixel value with the (approx.) partial derivative at that pixel. At the edges, intensity variation would be high, and hence partial derivatives, would have high values and other pixels would have comparatively lower values. Hence, on global thresholding, only those pixel values above a certain limit remain, making it an edge detector.

Gaussian Laplace filtering: The first step in Laplace of Gaussian edge detection is filtering the input image with the Gaussian to reduce noise. Laplace filter is then applied to obtain second order partial derivatives. Across different sides of an edge, the second order partial derivative will have different signs (as in one case intensity varies from low to high, and in the other case it is the opposite). Hence, zero order crossing and local thresholding is applied to the pixels, which return the detected edges.

Canny Edge Detection:
In this method

a. A smoothing operation is done (Gaussian filter)
b. Finding intensity gradients, through filters such as Sobel.
c. Non maximum suppression is applied to thin the edges. This is performed to obtain the exact edge and to remove thick lines from the resulting image.
d. Double thresholding is applied to identify strong edge and weak edge pixels
e. Unconnected weak edge pixels are removed (as they could present be because of noise)

A2.

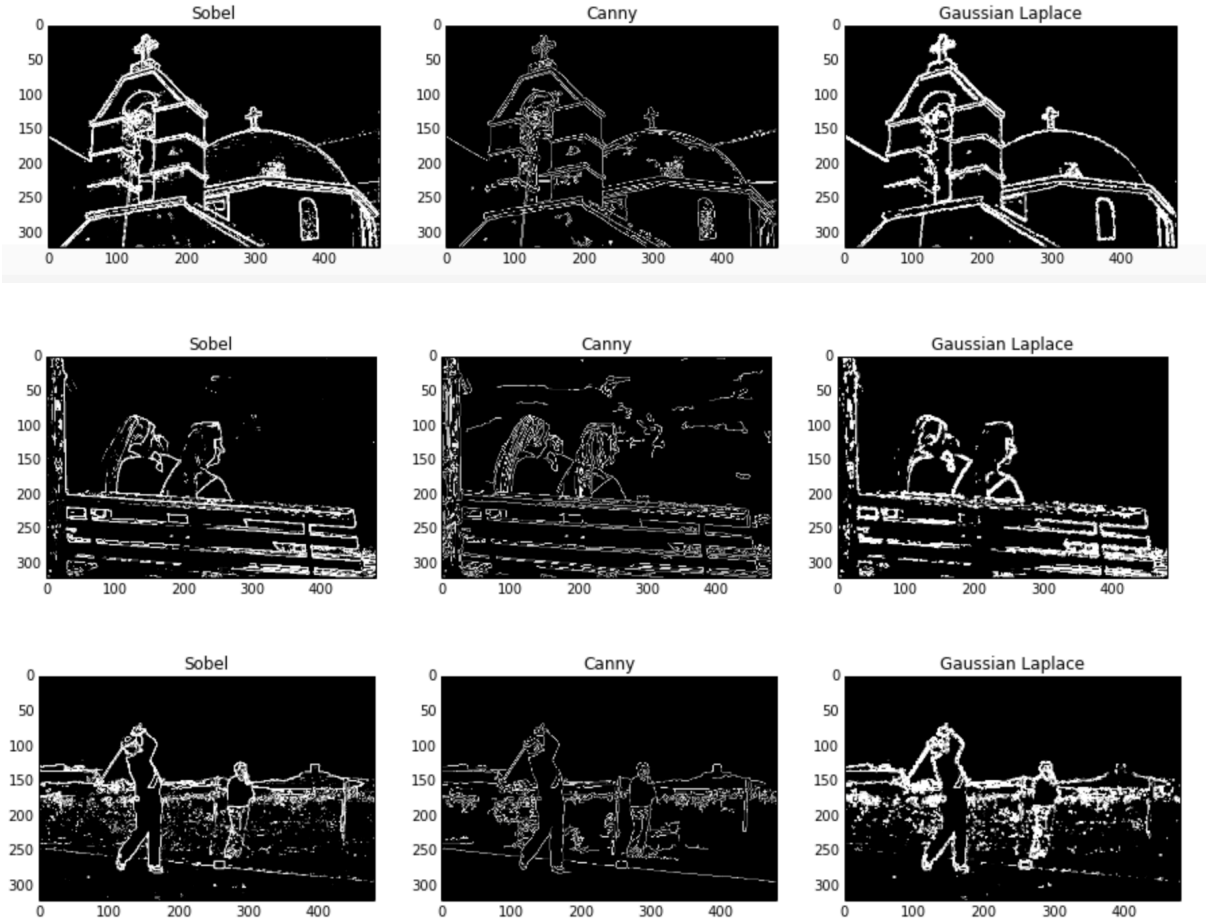
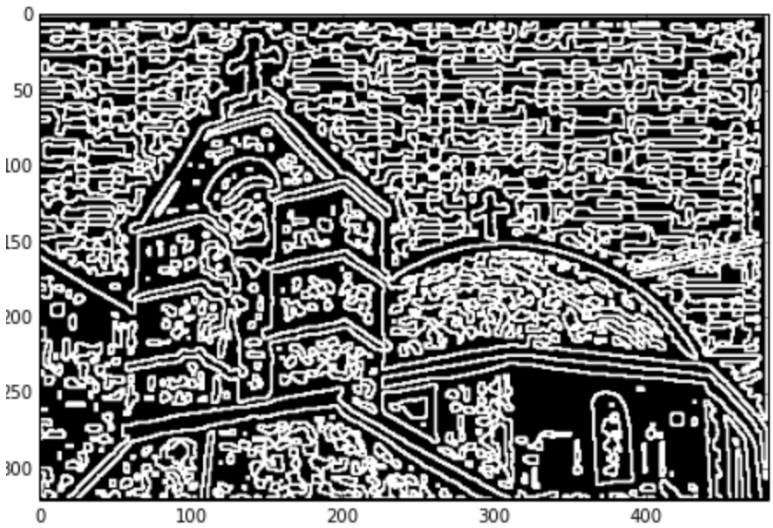


Fig 4: results of different edge detectors with manually tuned parameters

The above images show the edges detected by the three detectors with manually tuned parameters. The parameters in question are sigma and threshold. Sigma is the variance of the Gaussian kernel which is used to blur the images before performing edge detection (applicable for Canny and Gaussian Laplace). All the three detectors use threshold. Sobel filter based edge detection requires a global threshold (to remove pixels with lower partial derivatives), LoG requires a local threshold to determine if there is sufficient gap between the second order partial derivative values of pixels on opposite sides to qualify as an edge, and canny requires a threshold value to categorize strong and weak edge pixels (low threshold has been set to 30% of high threshold). Although threshold is used differently by the three detectors, it can be seen that increasing or decreasing sigma has a direct impact on the range of threshold (for detecting edges optimally). With increase in sigma, the threshold range decreases and vice versa. This occurs because high sigma implies higher smoothening making the edges weak. Hence slight modification in threshold values lead to loss of edges in the final image.
Sample values: Nuns.jpg

Sobel: Threshold=.059
Canny: Sigma = .5151, High Threshold = 0.23908, low threshold = 069 (.3*high threshold)
Laplace of Gaussian: Sigma = .8143, local threshold=.1

c.



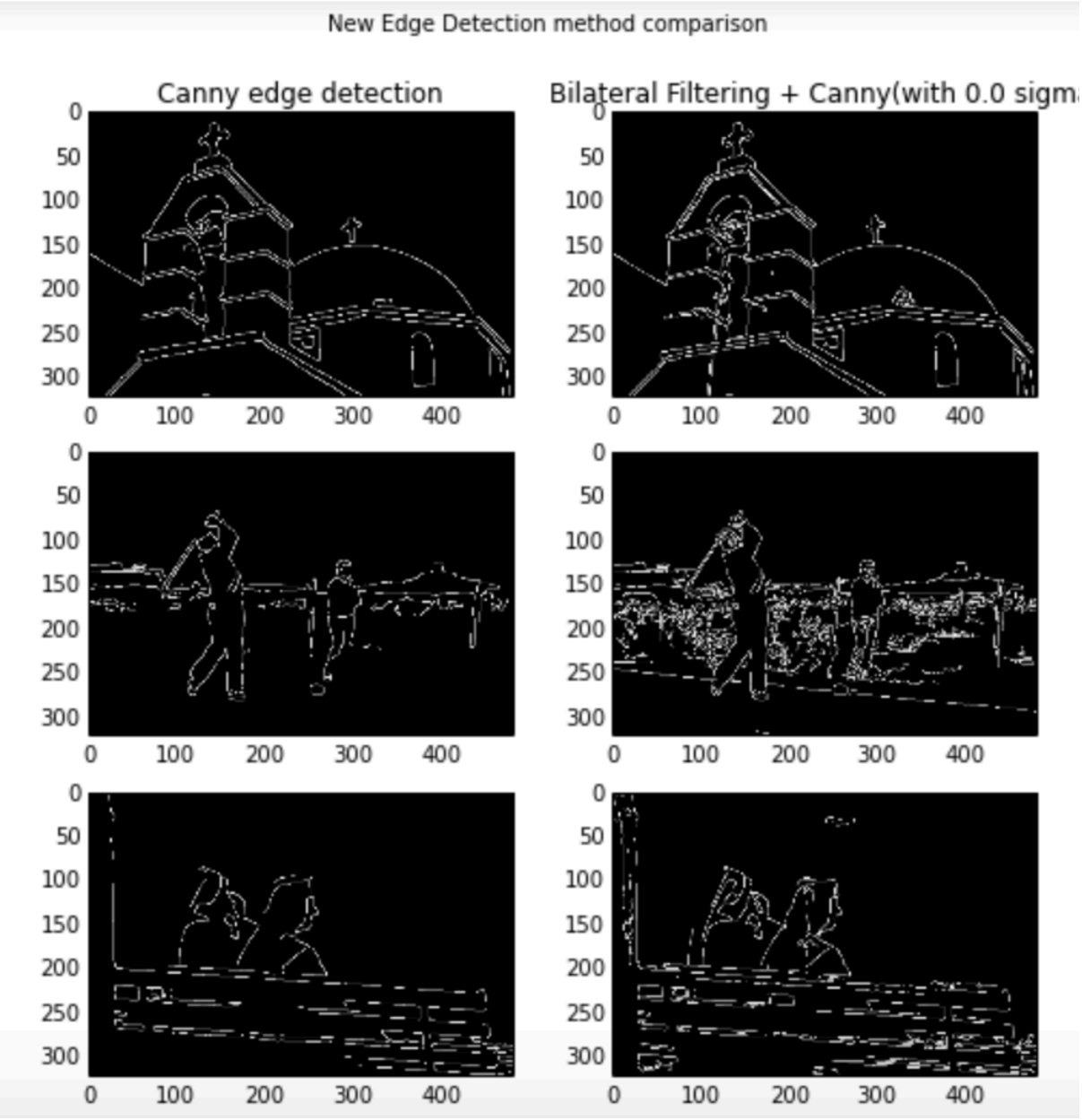
Img: church.jpg, Sigma = 2.0,

The above image displays the edges detected by Laplace of Gaussian when threshold is 0. It can be seen that all the edges are closed curves. This is so because:

a. As long as there are pixels in a patch of image with second order derivatives greater and less than zero, there will be a zero crossing. With a zero threshold, every zero crossing will be taken as an edge.

b.

A4. An issue with Canny (and Gaussian laplace) is that the blurring step decreases the sharpness of the edges as well which makes edge detection harder. My prize winning edge detection algorithm attempts to address this problem by using a bilateral filter instead of using a Gaussian filter. A Bilateral filter preserves the edges and performs de-noising as well. (For implementation canny function of skimage has been used with zero sigma value).

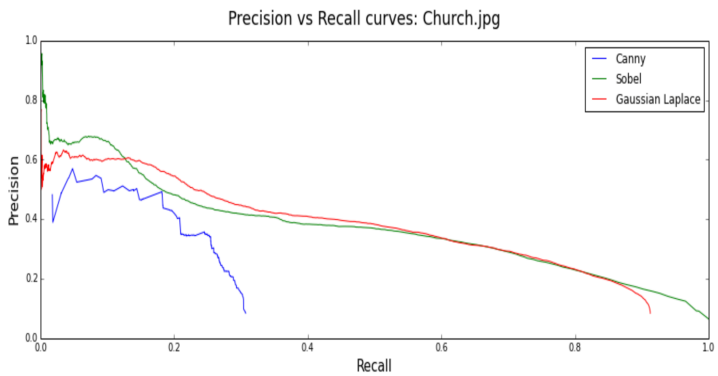


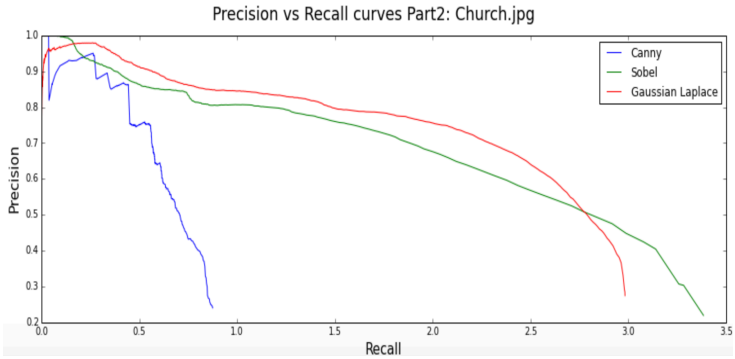
As it can be seen in the above images, there is far less information loss and clarity in edges in the second case when compared to the simple canny edge detector.

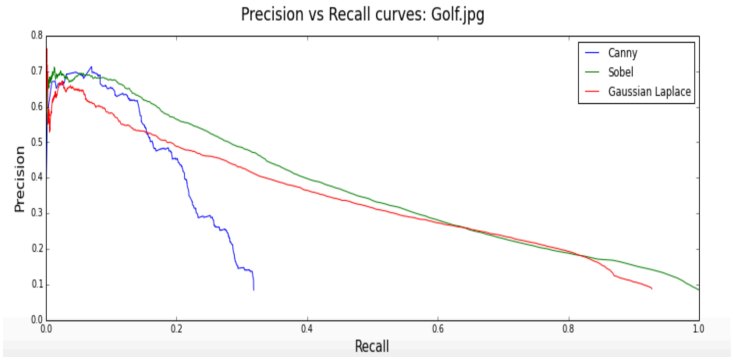
B 1. Precision and recall measure two different aspects of a predictor. Precision measures what percentage of the predictions were relevant and recall measures what percentage of the relevant edges were predicted. In mathematical terms, Precision = True positive/(true positive+false positive) and recall = True positive/(true positive+false negative). A predictor can have very good recall if it predicts everything as true, however in that case it would

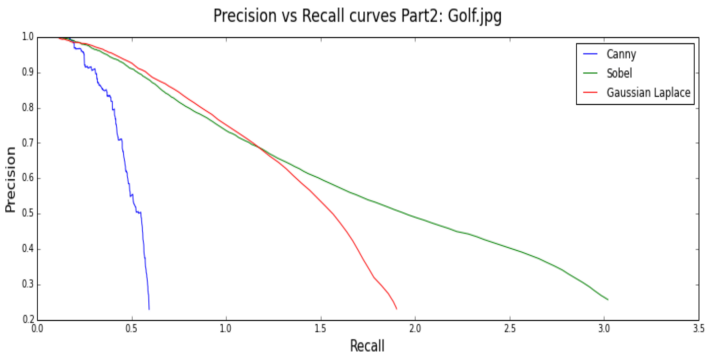
have a very bad precision. Similarly, a predictor can have high precision if it predicts a small portion of the ground truth correctly but might miss a large portion (and hence have low recall). Hence, both values are important.

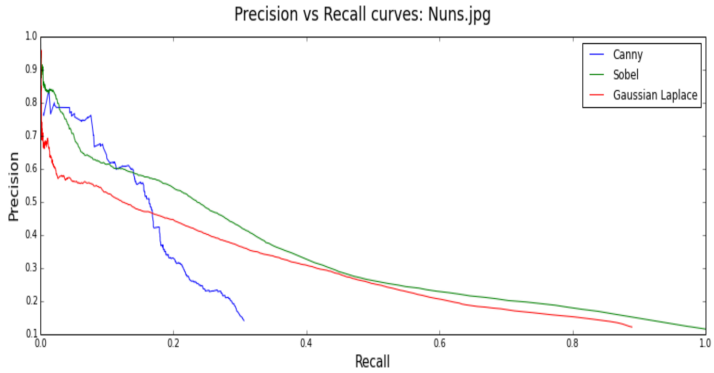
B2 & B3

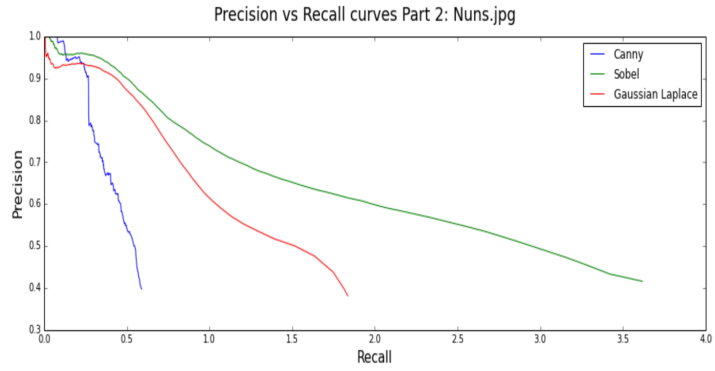












The above curves display Precision vs Recall for the three images. Images in the right allow 3px spatial errors whereas the images in the left do not. It can be seen that the general trend of the images is similar, but there is a significant increase in recall and a slight increase in precision.

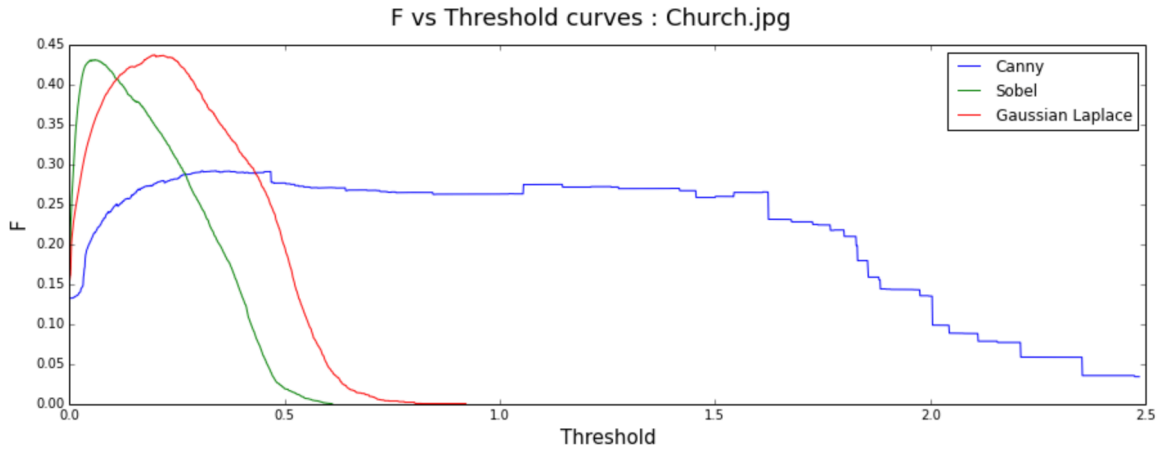
Method for accounting 3px spatial errors:

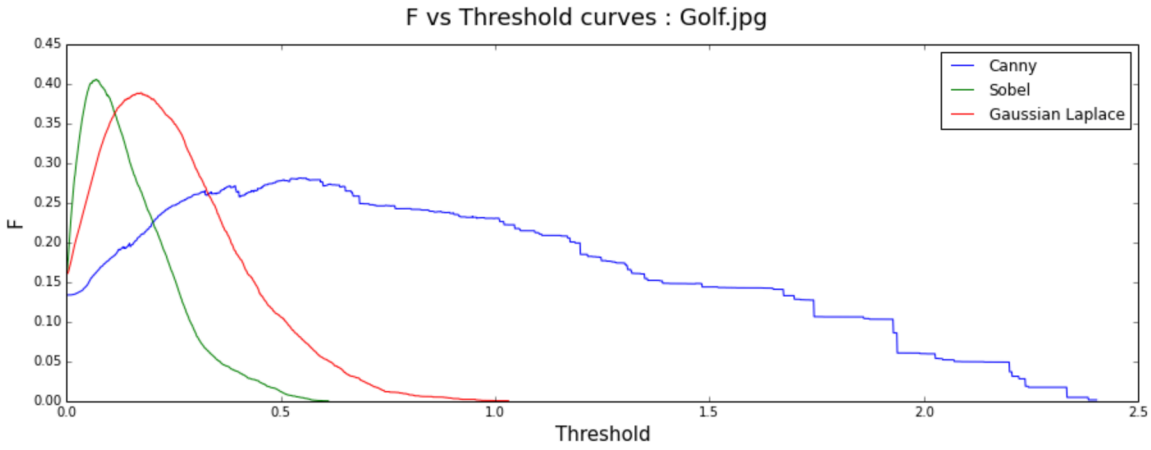
a. Coordinates of all edge points of _GT image stored in an array
b. Copy of this array created
c. Pixels with |manhattan distance| <=3 w.r.t pixels in the original array appended to the copy.

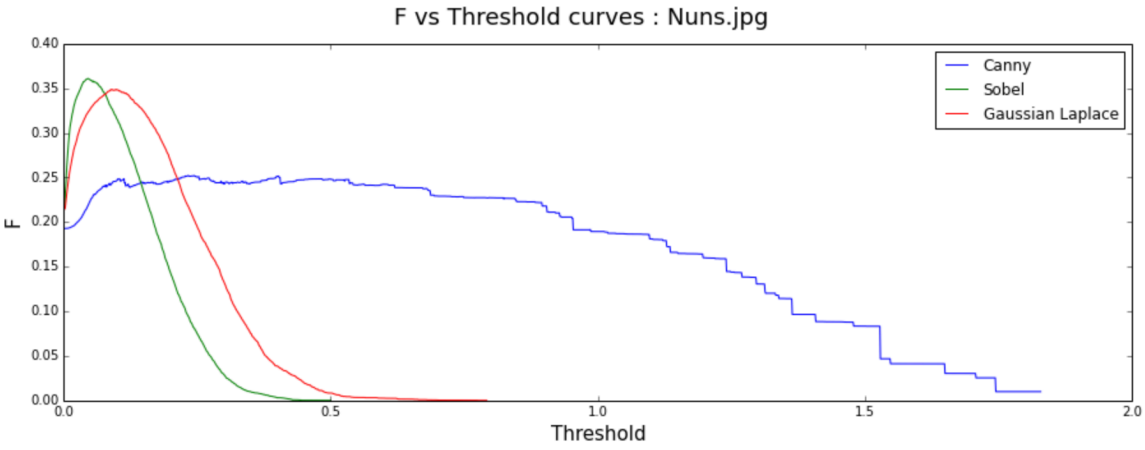
d. Set intersection with the new array gives the true positive count.

The images which allow spatial errors have both higher precision and recall range. The recall can be higher than 1 in this case because the true positive count is calculated differently and might be more in number than the denominator (number of edge pixels in GT image).

Q4.







In the above F vs Threshold curves, it can be seen that canny edge detector maintains a considerably high F value for a large range of threshold in comparison with Sobel and Laplace of Gaussian edge detectors. However, it has a lower max F value than Sobel and LoG. This can be easily understood by the precision vs recall graphs in the previous page. There exist points in the precision recall curve where both Sobel and LoG have higher precision (comparable to the max of Canny) as well as recall than Canny. Hence, their max values are expected to be higher. Canny detector has a much higher resistance to changes in threshold values, which makes it very reliable and easy to use.