Contour Detection - Solution Template

NOTE: All values and figures in this template are examples that you will need to replace with your own results

1. **Method Description.**: Describe the different mehtods and their key implementation detials.

   Warm up:

   I use the spicy.

   ```
   spicy.signal.convolve2d(I,np.array([[-1,0, 1],[-4,0,4],
   [-1,0,1]]),mode='same', boundary = 'symm').
   ```

   I changed the boundary function into "symm", which means symmetrical boundary conditions.

   Smoothing:

   We smooth the image so that we can reduce the amount of edges detected from the noise and apply a low-pass filter. In my applications, I firstly use GaussianBlur function to get the blur image.

   ```
   I = cv2.GaussianBlur(I, (7,7),2)
   ```

   Also, I use derivative of Gaussian filters to obtain more robust estimates of the gradient:

   ```
   dx = signal.convolve2d(I,np.array([[-1,0, 1],[-4,0,4],
   [-1,0,1]]),mode='same', boundary = 'symm')
   dy = signal.convolve2d(I,np.array([[-1,0, 1],[-4,0,4],
   [-1,0,1]]).T,mode='same', boundary = 'symm')
   ```

   Then, for the filter, I change the [−1, 0, 1] to np.array([[-1,0, 1],[-4,0,4],[-1,0,1]]) for better results.

   Non-maximum Suppression:

   To sharpen the edfes, we will find the local maxima in the gradient image. Here I use this strategy: A gradient is considered locally maximal if it is either greater than or equal to its neighbors in the positive and negative gradient direction. For the gradient direction, we calculate the arctangent of the gradient vector:

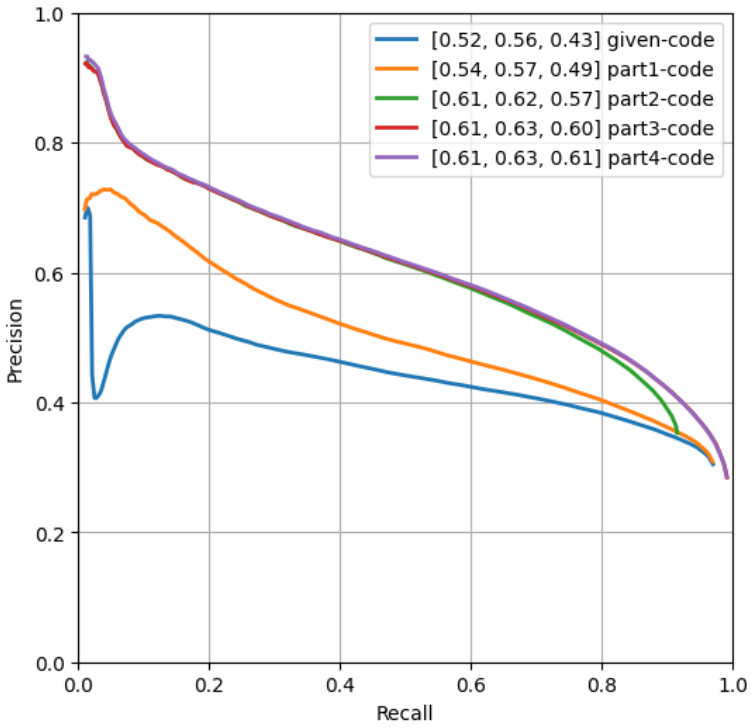
   ```
   angle = np.arctan2(dy, dx)
   ```

   Then I round it to th nearest 45 degree by the angle mod, so that I can conform to one of 8 discrete directions Also, for more accuracy, I use the alpha, which can take the combination of different

directions. After than, I compare the the neighbor results with the cernter one. If the neighbor results have a higher "edgeness", we will ignore this center or. Otherwise, we keep it.

After that, we normalize our resluts to get the better results.

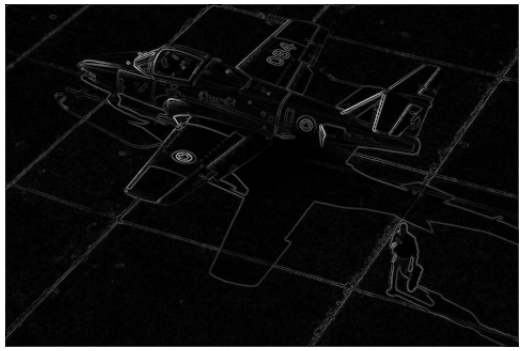
2. **Precision Recall Plot.** *TODO*: Use contour_plot.py to add curves for the different methods that you implemented into a single plot.



3. **Results Table.** : Present the performance metrics for each implementation part in a table format

Method	overall max F-score	average max F-score	AP	Runtime (seconds)
Initial implementation	0.52	0.56	0.43	0.008
Warm-up [remove boundary artifacts	0.538742	0.574959	0.491509	0.013049
Smoothing	0.605989	0.619656	0.571090	0.019512
Non-max suppression	0.610538	0.632524	0.603645	2.103400
Test set numbers of best model [From gradescope]	0.611799	0.633582	0.605676	2.106341

4. **Visualizations.** Include visualization on 3 images (before and after the contour detection). Comment on your observations, where does your contour detector work well, where it doesn't and why? you can are also add visualizations of your own images.



The contour detector work well when the color differences are obvious. For instance, the strips of the zebras, the strips of the tiger.

The reason for this is that the change in color can be catch by our [-1,0, 1],[-4,0,4],[-1,0,1]] filure.

The contour detector doesn't work well when the color differences is not obvious, such as the texture of the tree and the grass.

The reason for this is that it is even hard for the human beings to draw the contour since the noise in color are obvious.

5. **Bells and Whistles.** *TODO*: Include details of the bells and whistles that you tried here.

: Present the performance metrics for the bells and whistles in a table format

Method	overall max F-score	average max F-score	AP	Runtime (seconds)
Best base Implementation (from above)	0.611799	0.633582	0.605676	2.106341

Method	overall max F-score	average max F-score	AP	Runtime (seconds)
Bells and whistle (1) The fuunction cv2.GaussianBlur()) can help us to get better smoothing. At the beginning, I tried the convolve2d(I, gaussian_kernel, mode='same') for the blur, but the cv2.GaussianBlur() generate better results. Also, we can adjust the number in the GaussianBlur. I tried several different filture to get the best result. Finally, I choose I = cv2.GaussianBlur(I, (7,7),2) as the best Gaussian parameter	0.605989	0.619656	0.571090	0.019512
Bells and whistle (2) The improvement of NMS in all directions. Instead of comparing with the 4 neighbour pixels, or the eight neighbor pixels, I also introduce the alpha here, so that we can have the accuracy to 360 degrees rather than 45 degrees, which can generate more accurate results for comparing. For instance: it can be the combinayion of both pixel [i + 1, j + 1] and pixel [i, j + 1])	0.610538	0.632524	0.603645	2.103400
Bells and whistle (n) difference start filter. The original filter is the [−1, 0, 1] after GaussianBlur. However, we can change it into np.array([[-1,0, 1],[-4,0,4],[-1,0,1]], np.array([[-1,0, 1],[-3,0,3],[-1,0,1]]) or np.array([[-3,0, 3],[-10,0,10],[-3,0,3]], so that we can find out the differences between colors more clear. I tried several different filture to get the best result.	0.611799	0.633582	0.605676	2.106341

Corner Detection - Solution Template

NOTE: All values and figures in this template are examples that you will need to replace with your own results

1. **Method Description.** : Describe the different methods and their key implementation detials.

   The first step is to Calculating image gradients in x and y direction by the

   ```
   dx = cv2.Sobel(image_gray, cv2.CV_64F,1, 0)
   dy = cv2.Sobel(image_gray, cv2.CV_64F,0, 1)
   ```

   Then, we can use the cv2.GaussianBlur() function to get the dxx, dxy, dyy. Then, by the math, by can calculate the cornerness score value of each point by

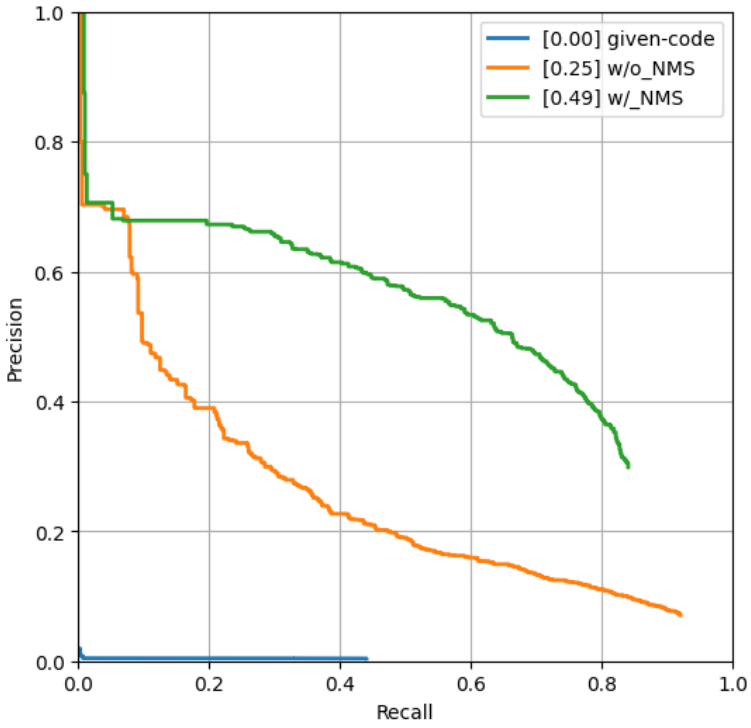
   ```
   detM = dxx * dyy - dxy**2
   traceM = (dxx+dyy)**2
   response = detM - alpha * traceM
   ```

   The alpha is between 0.04 to 0.06.

   For the Non maximum suppression. The method I use is to compare cornerness score value with the cloeset 8 pixels. If its cornerness score value is smaller than the neighbor, it will be reset to 0. Otherwise, we keep it.

   And we always nomalize our results to 0 to 255 at the end.

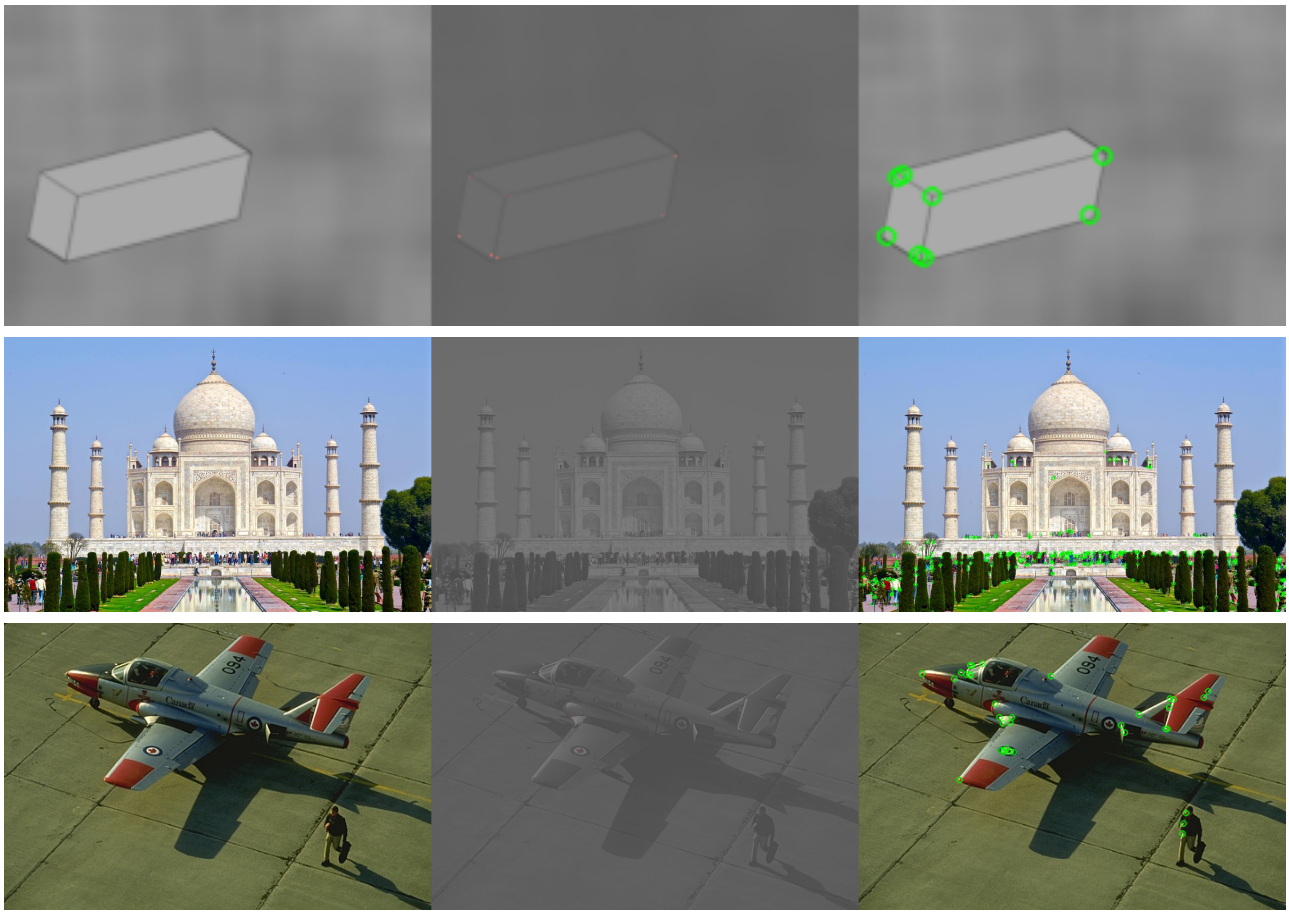
2. **Precision Recall Plot.** : Use corner_plot.py to add curves for the different methods that you implemented into a single plot.



3. **Results Table.** : Present the performance metrics for each implementation part in a table format

Method	Average Precision	Runtime
Random	0.001	0.001
Harris w/o NMS	0.445783	0.182202
Harris w/ NMS	0.491765	0.09
Hyper-parameters tried (1) alpha = 0.05;kernal_size = (5,5);sigma = 0; window = 1	0.437521	0.185428
Hyper-parameters tried (2) alpha = 0.05; kernal_size = (7,7);sigma = 1; window = 1	0.445783	0.188328
Test set numbers of best model [alpha = 0.05;kernal_size = (3,3);sigma = 0; window = 1]	0.491765	0.09

4. **Visualizations.** Include visualization on 3 images. Comment on your observations, where does your corner detector work well, where it doesn't and why? We aslo provided some images in data/vis for testing, but you are free to use your own images as well.



5. **Bells and Whistles.** : Include details of the bells and whistles that you tried here.

   TODO: Present the performance metrics for the bells and whistles in a table format

Method	**Average Precision**	**Runtime**
Best base Implementation (from above)		
Bells and whistle (1) compare with more than 8 pixels. Instead of finding the eight neibors around the pixel for NMS, we can calculate the pixels within a range and find the pixles within the radius. Although it turns out that the radius of 1 is the best results, which is similar to compare with the 8 closest pixels.	0.48257	0.10
Bells and whistle (2) [extra credit])		
Bells and whistle (n) [extra credit])		

Multi-resolution Blending - Solution Template

NOTE: All values and figures in this template are examples that you will need to replace with your own results

1. **Method Description.** *TODO*: Describe the key implementation details for blending. **Step 0**: The process the pictures into right scale. I firstly devide the picture by 255. Then I convert it into gray scale.

   Step1: Build Laplacian pyramids LA and LB from images A and B I will calcualte the image after the Gaussian, and calculate the Laplacian by using the former layer minus later layer.

```
def get_Laplacian_Stack(iimg, N):
   w = iimg.shape[0]
   h = iimg.shape[1]

   gstack = np.zeros((w,h,N))
   for x in range(0,N):
      sigma = 2**x
      img = get_gaussian_filter(iimg,sigma)
      gstack[:,:,x] = img

   w,h,n = gstack.shape
   stack = np.zeros((w,h,n))
   for x in range(0,n-1):
      img = gstack[:,:,x]-gstack[:,:,x+1]
      stack[:,:,x] = img
      if x == n-2:
            stack[:,:,x+1] = gstack[:,:,x+1]
   return stack
```

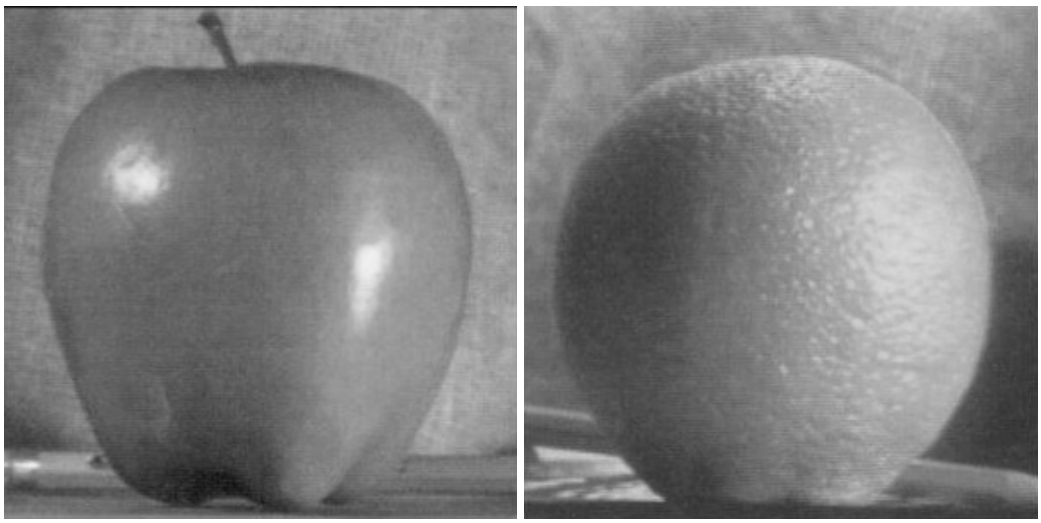
   Step2: Build a Gaussian pyramid GR from selected region R The Gaussian pyramid is get by having the filture after the Gaussian filter. The sigma will change as the exponential of 2.

```
def get_Gaussian_Stack(iimg, N, alpha):
   w, h= iimg.shape
   stack = np.zeros((w,h,N))
   for x in range(0,N):
      sigma = (2**(x*alpha))
      img = get_gaussian_filter(iimg,sigma)
      stack[:,:,x] = img
   return stack
```

   Step3: Form a combined pyramid LS from LA and LB using nodes of GR as weights and blended Collapse the LS pyramid to get the final blended • LS(i,j) = GR(I,j,)*LA(I,j) + (1-GR(I,j))*LB(I,j) And combine all the layers as a loop

```
for x in range(0,N):
      left = mask_gstack[:,:,x]*left_lstack[:,:,x]
      right = (1 - mask_gstack[:,:,x])*right_lstack[:,:,x]
      blend += left + right
```

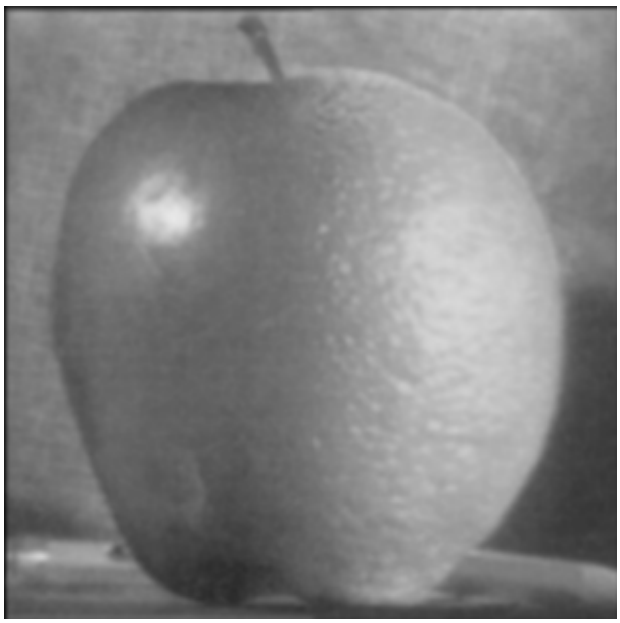
2. **Oraple.** *TODO:* Include visualizations of the Oraple blending along any variations your tried (include both original images and the blended image) **The original picture is:**



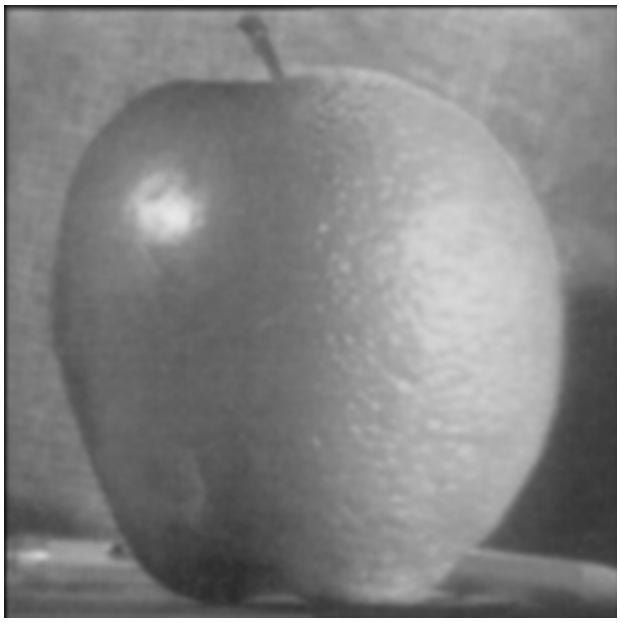
The results:

- The N is the number of total laplacian level I will have.
- The alpha will affect the simga in the gaussian filter. alpha is sigma = 2**(x*alpha)

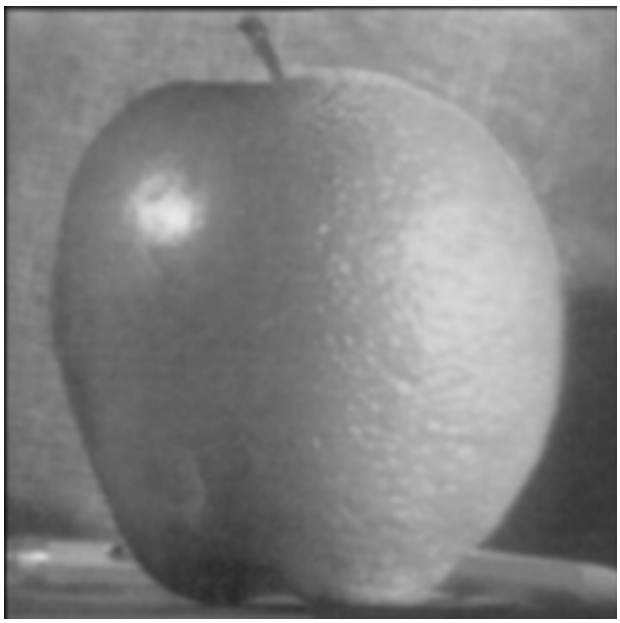
N = 5 and alpha = 4



N = 4 and alpha = 4



N = 4 and alpha = 3



3. **Blends of your choice.** Include visualizations of blends of your choice (include both original images and the blended image). Describe any modifications you made on top of what worked for the oraple.

   The original picture is:


   The results: