

```
In [7]: %matplotlib inline
from __future__ import print_function
#import ganymede
#ganymede.configure('uav.beaver.works')
import matplotlib.pyplot as plt
import numpy as np
import cv2
import os
```

```
In [8]: def check(p): pass
check(0)
```

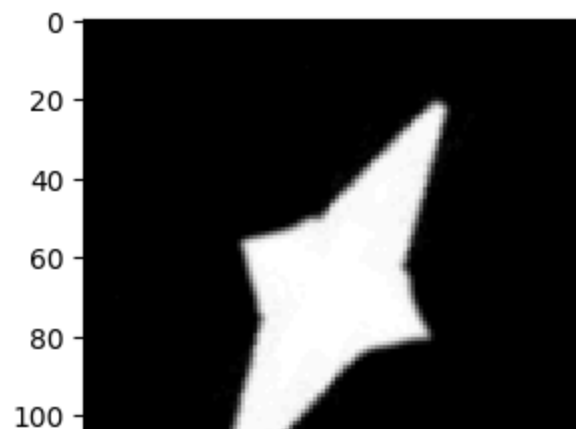
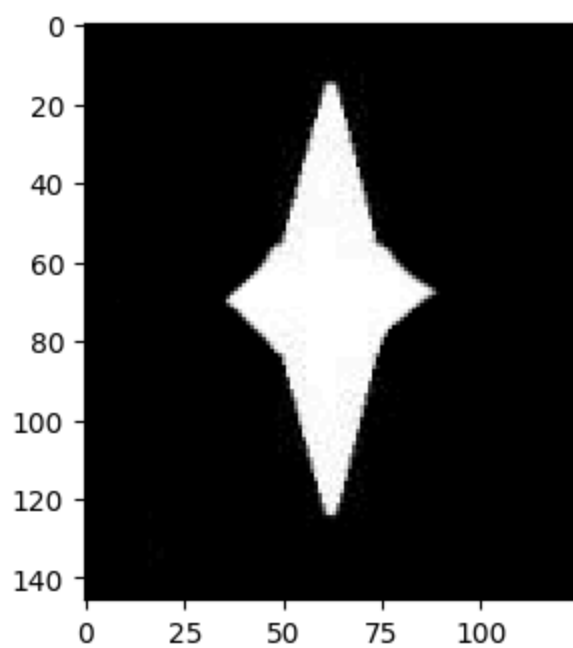
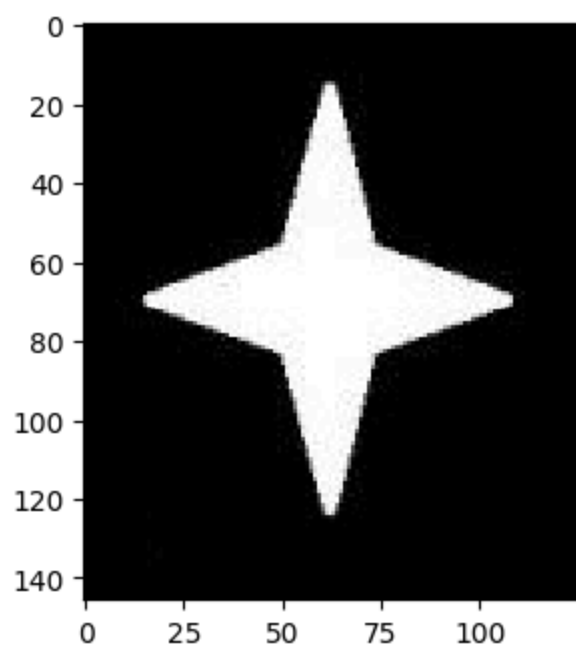
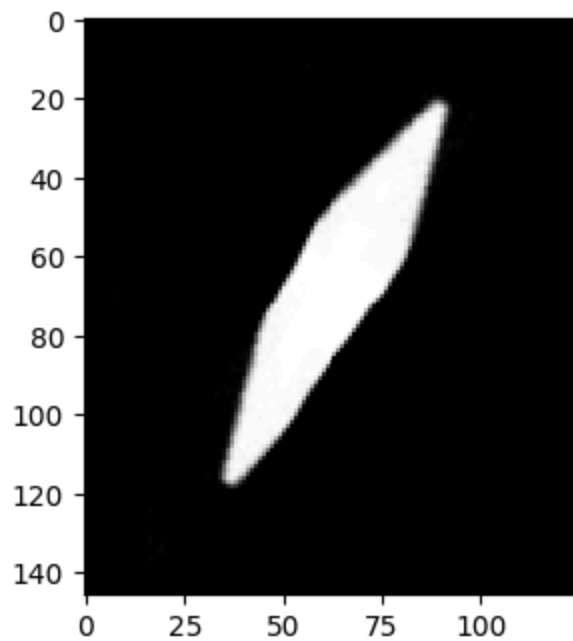
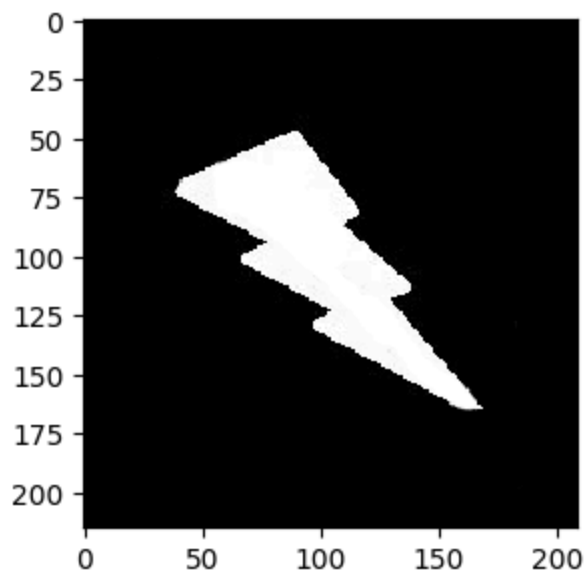
## Note

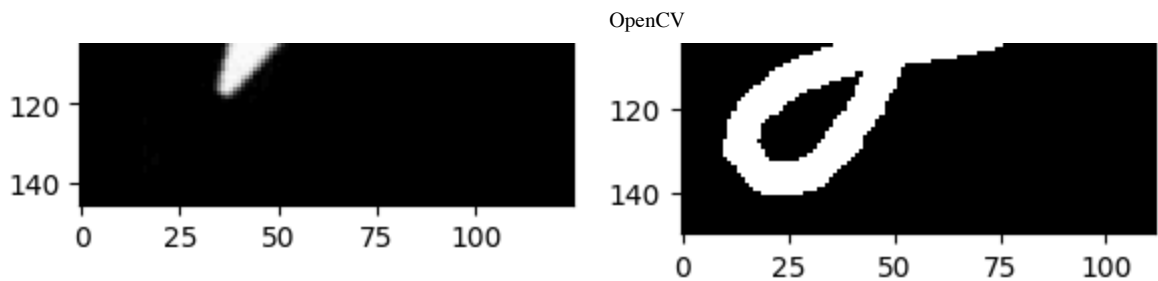
`cv2.imshow()` will not work in a notebook, even though the OpenCV tutorials use it. Instead, use `plt.imshow` and family to visualize your results.

```
In [9]: lightningbolt = cv2.imread('shapes/lightningbolt.png', cv2.IMREAD_GRAYSCALE)
blob = cv2.imread('shapes/blob.png', cv2.IMREAD_GRAYSCALE)
star = cv2.imread('shapes/star.png', cv2.IMREAD_GRAYSCALE)
squishedstar = cv2.imread('shapes/squishedstar.png', cv2.IMREAD_GRAYSCALE)
squishedturnedstar = cv2.imread('shapes/squishedturnedstar.png', cv2.IMREAD_GRAYSCALE)
letterj = cv2.imread('shapes/letterj.png', cv2.IMREAD_GRAYSCALE)

images = [lightningbolt, blob, star, squishedstar, squishedturnedstar, letterj]

fig, ax = plt.subplots(nrows=3, ncols=2)
for a, i in zip(ax.flatten(), images):
    a.imshow(i, cmap='gray', interpolation='none');
fig.set_size_inches(7, 14);
```





```
In [10]: intensity_values = set(lightningbolt.flatten())
print(len(intensity_values))
```

75

## Question:

What would you expect the value to be, visually? What explains the actual value?

```
In [11]: # TODO
# I would expect the value to be the number of unique colors present in the image
```

## Thresholding

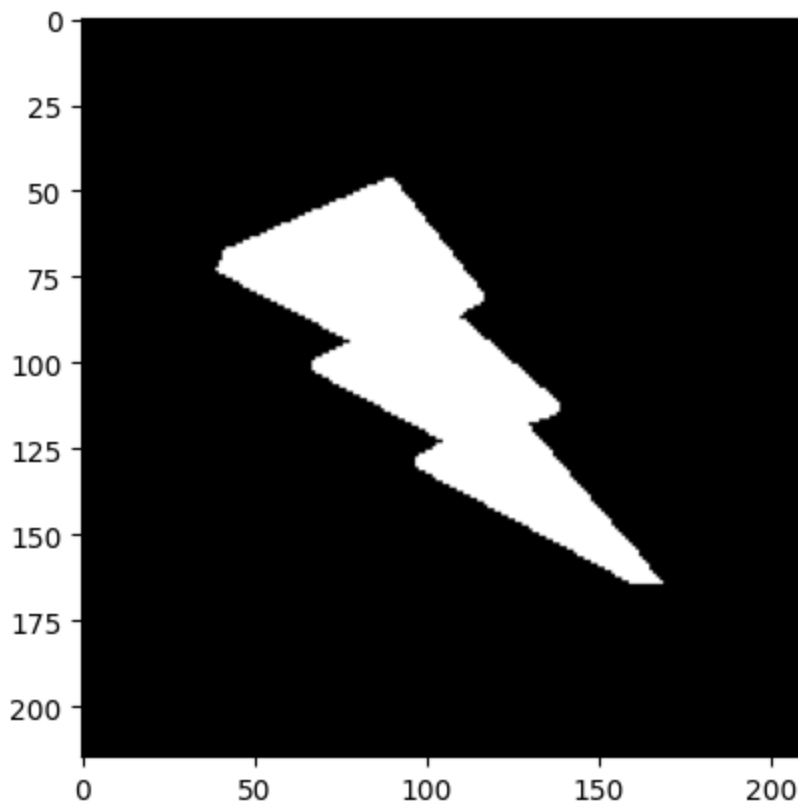
[https://docs.opencv.org/3.4.1/d7/d4d/tutorial\\_py\\_thresholding.html](https://docs.opencv.org/3.4.1/d7/d4d/tutorial_py_thresholding.html)

```
In [12]: _, lightningbolt = cv2.threshold(lightningbolt, 150, 255, cv2.THRESH_BINARY)

intensity_values = set(lightningbolt.flatten())
print(len(intensity_values))

plt.imshow(lightningbolt, cmap='gray');
```

2



## Question

What happens when the above values are used for thresholding? What is a "good" value for thresholding the above images? Why?

```
In [13]: ## TODO  
## The above values result in a very grainy/noisy image. A good value would be
```

## Exercises

### Steps

1. Read each tutorial
  - Skim all parts of each tutorial to understand what each operation does
  - Focus on the part you will need for the requested transformation
2. Apply the transformation and visualize it

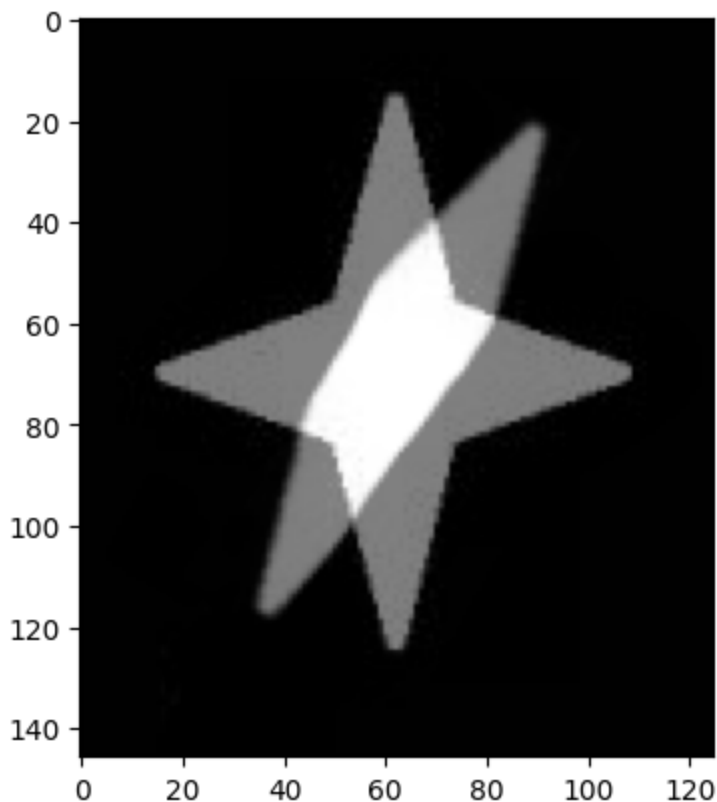
### 1. Blend lightningbolt and blob together

[https://docs.opencv.org/3.4.1/d0/d86/tutorial\\_py\\_image\\_arithmetics.html](https://docs.opencv.org/3.4.1/d0/d86/tutorial_py_image_arithmetics.html)

Remember: Don't use `imshow` from OpenCV, use `imshow` from `matplotlib`

```
In [14]: # 1. Blend  
blended=cv2.addWeighted(star,0.5,blob,0.5,0)  
plt.imshow(blended,cmap="gray")
```

```
Out[14]: <matplotlib.image.AxesImage at 0x11b13d8e0>
```

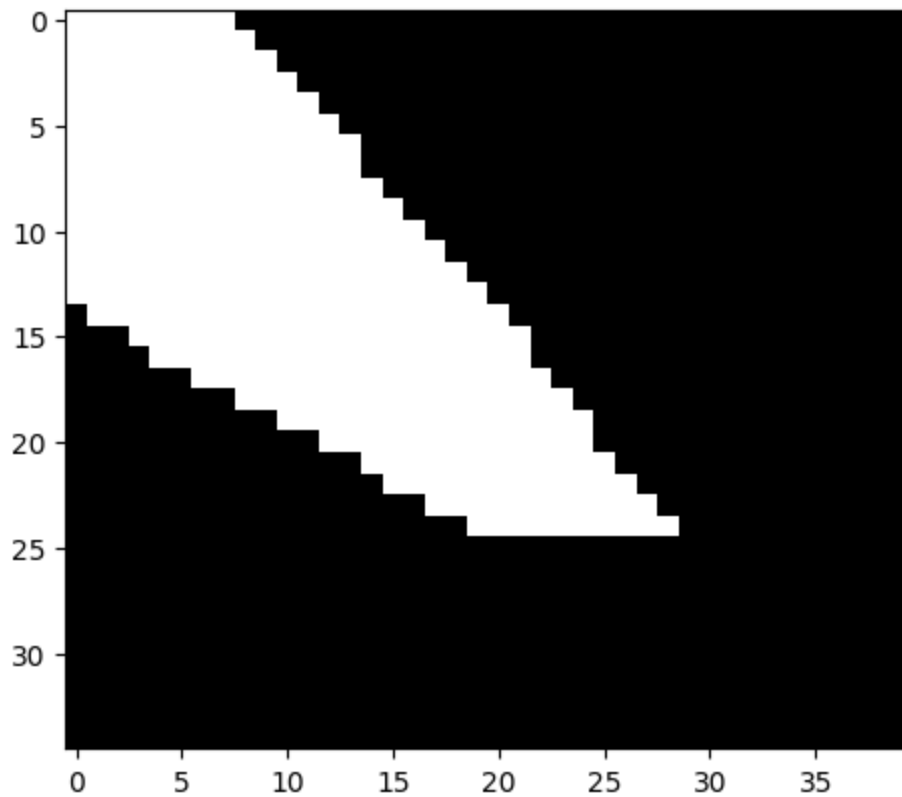


## 2. Find a ROI which contains the point of the lightning bolt

[https://docs.opencv.org/3.4.1/d3/df2/tutorial\\_py\\_basic\\_ops.html](https://docs.opencv.org/3.4.1/d3/df2/tutorial_py_basic_ops.html)

```
In [15]: # 2. ROI  
point=lightningbolt[140:175, 140:180]  
plt.imshow(point,cmap="gray")
```

```
Out[15]: <matplotlib.image.AxesImage at 0x11ad7eeb0>
```

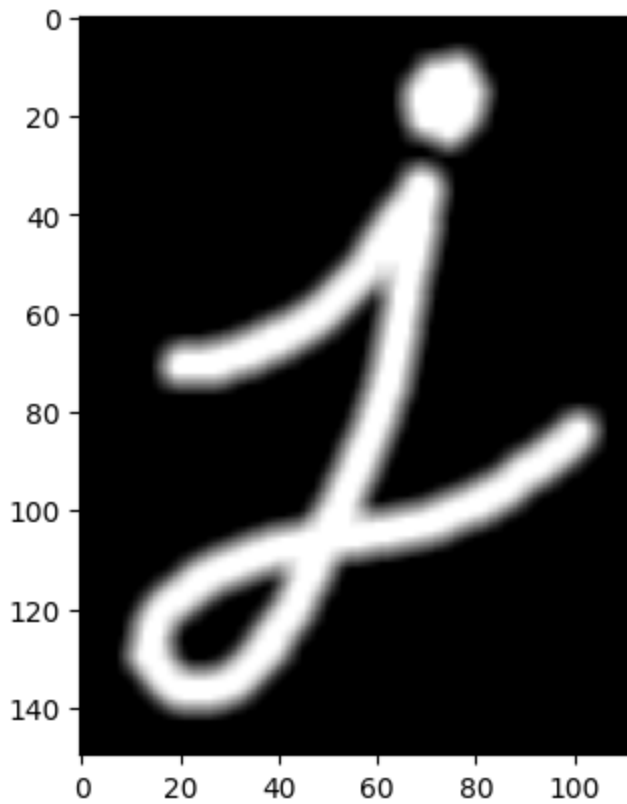


### 3. Use an averaging kernel on the letter j

[https://docs.opencv.org/3.4.1/d4/d13/tutorial\\_py\\_filtering.html](https://docs.opencv.org/3.4.1/d4/d13/tutorial_py_filtering.html)

```
In [16]: # 3.  
avg=cv2.blur(letterj,(5,5))  
plt.imshow(avg,cmap="gray")
```

```
Out[16]: <matplotlib.image.AxesImage at 0x11aa6a2b0>
```



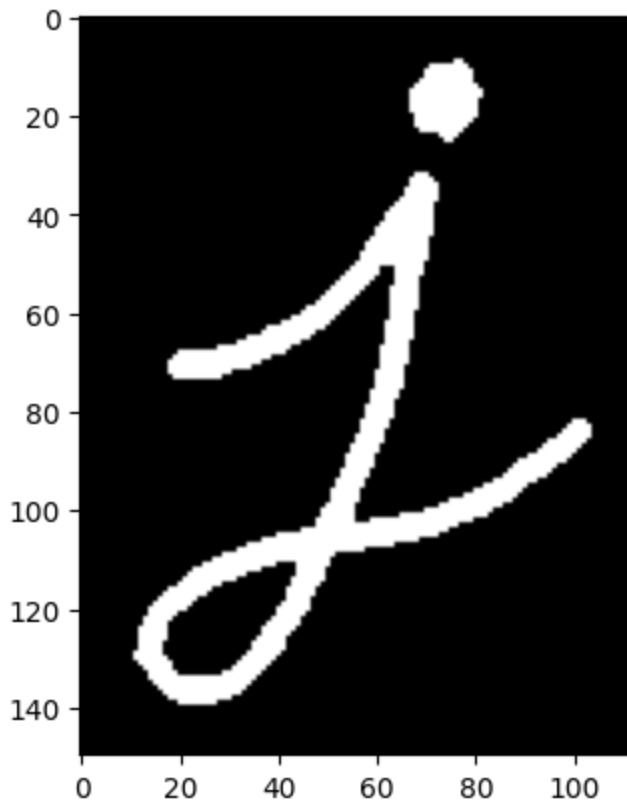
## Morphology

[https://docs.opencv.org/3.4.1/d9/d61/tutorial\\_py\\_morphological\\_ops.html](https://docs.opencv.org/3.4.1/d9/d61/tutorial_py_morphological_ops.html)

### 4. Perform erosion on j with a 3x3 kernel

```
In [17]: # 4
kernelthree=np.ones((3,3),np.uint8)
halferoded=cv2.erode(letterj,kernelthree,iterations=1)
plt.imshow(halferoded,cmap="gray")
```

```
Out[17]: <matplotlib.image.AxesImage at 0x11b414160>
```

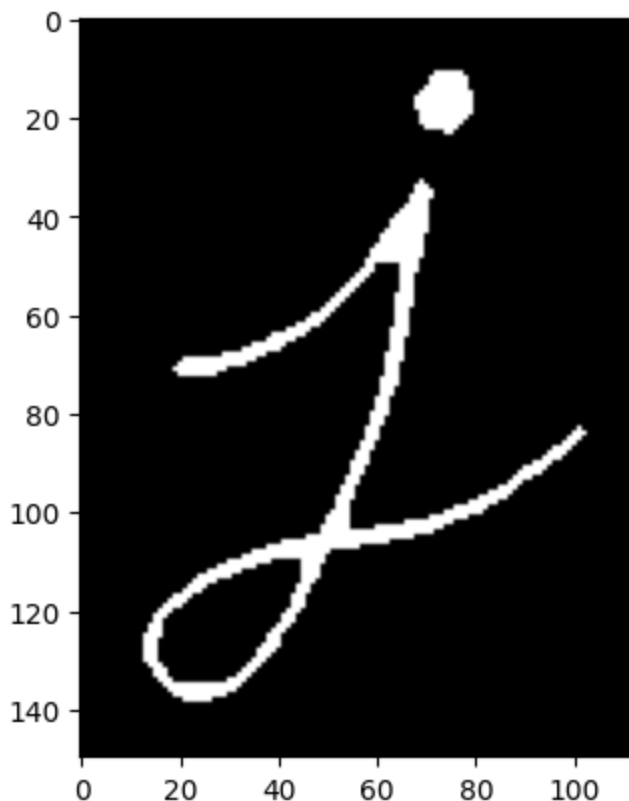


## 5. Perform erosion on j with a 5x5 kernel

```
In [18]: # 5
kernelfive=np.ones((5,5),np.uint8)
veryeroded=cv2.erode(letterj,kernelfive,iterations=1)
plt.imshow(veryeroded,cmap="gray")
```

```
Out[18]: <matplotlib.image.AxesImage at 0x11b491b20>
```





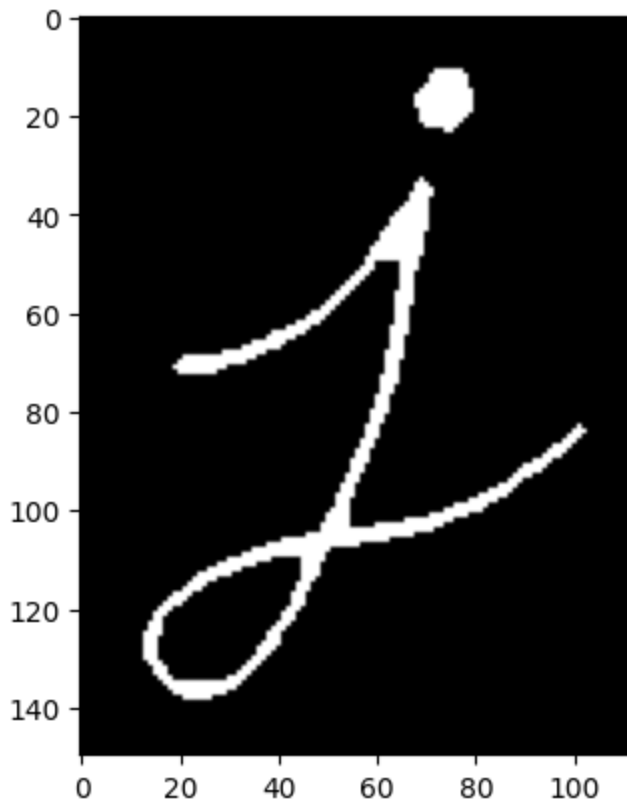
## 6. Perform erosion on j with **two** iterations, using a kernel size of your choice

Hint: look at the OpenCV API documentation. It is possible to perform two iterations of erosion in one line of Python!

[https://docs.opencv.org/3.4.1/d4/d86/group\\_\\_imgproc\\_\\_filter.html#gaeb1e0c1033e3f6b891a25](https://docs.opencv.org/3.4.1/d4/d86/group__imgproc__filter.html#gaeb1e0c1033e3f6b891a25)

```
In [19]: # 6
kernelthree=np.ones((3,3),np.uint8)
eroded=cv2.erode(letterj,kernelthree,iterations=2)
plt.imshow(eroded,cmap="gray")
```

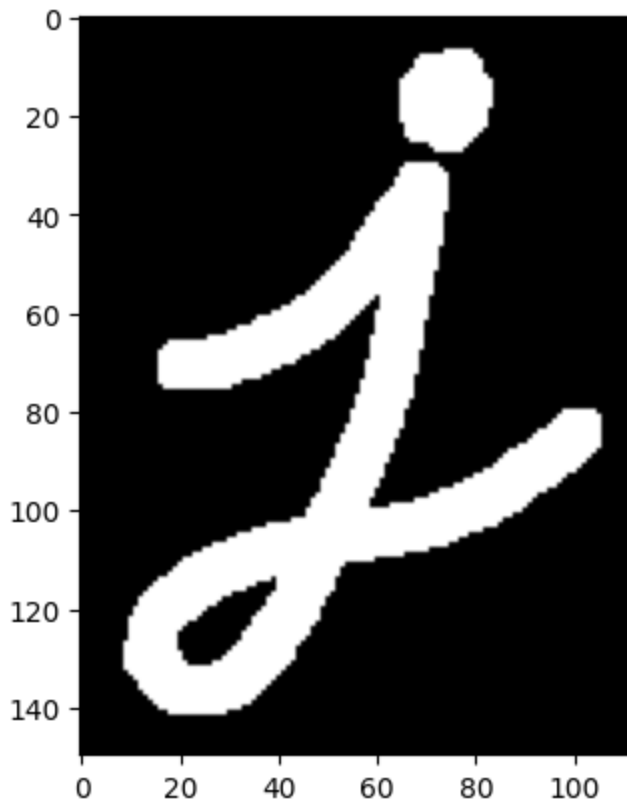
```
Out[19]: <matplotlib.image.AxesImage at 0x11b5105b0>
```



## 7. Perform dilation on j with a 3x3 kernel

```
In [20]: # 7
kernelthree=np.ones((3,3),np.uint8)
halfdilated=cv2.dilate(letterj,kernelthree,iterations=1)
plt.imshow(halfdilated,cmap="gray")
```

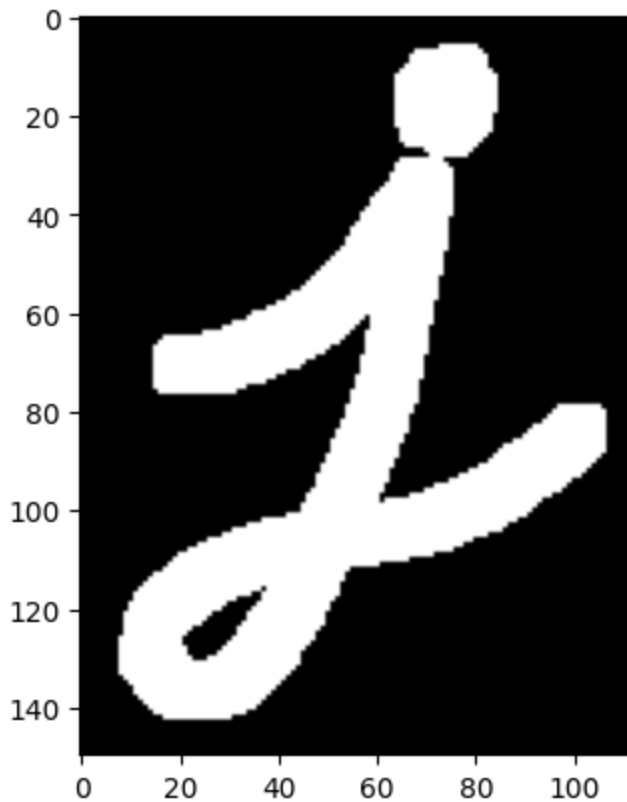
```
Out[20]: <matplotlib.image.AxesImage at 0x11b586640>
```



## 8. Perform dilation on j with a 5x5 kernel

```
In [21]: # 8
kernelfive=np.ones((5,5),np.uint8)
dilated=cv2.dilate(letterj,kernelfive,iterations=1)
plt.imshow(dilated,cmap="gray")
```

```
Out[21]: <matplotlib.image.AxesImage at 0x11b5dbe20>
```



## 9. What is the effect of kernel size on morphology operations?

In [22]: `# 9`  
*# The kernel size impacts the area of effect any operation can have. The larger the kernel, the more area it covers.*

## 10. What is the difference between repeated iterations of a morphology operation with a small kernel, versus a single iteration with a large kernel?

In [23]: `# 10`  
*# Theoretically speaking, there is no difference. Practically, repeating a small kernel operation many times is faster than a single large kernel operation.*

## 11. Rotate the lightningbolt and star by 90 degrees

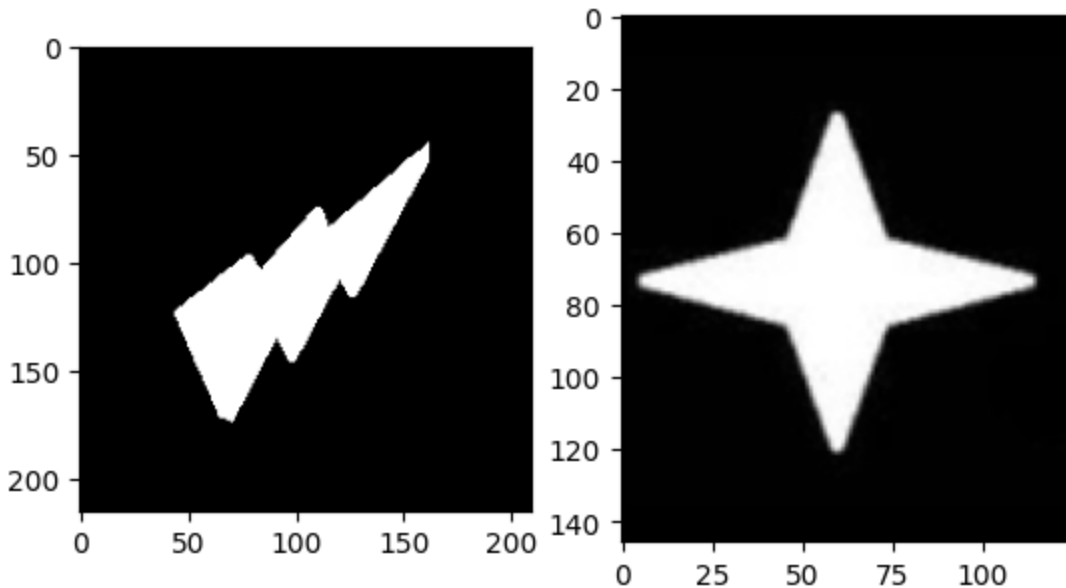
[https://docs.opencv.org/3.4.1/da/d6e/tutorial\\_py\\_geometric\\_transformations.html](https://docs.opencv.org/3.4.1/da/d6e/tutorial_py_geometric_transformations.html)

In [30]: `# 11`  
`rows1,cols1=lightningbolt.shape`  
`rows2,cols2=star.shape`  
`M1=cv2.getRotationMatrix2D((cols1/2,rows1/2),90,1)`  
`M2=cv2.getRotationMatrix2D((cols2/2,rows2/2),90,1)`  
`rotatedbolt=cv2.warpAffine(lightningbolt,M1,(cols1,rows1))`  
`rotatedstar=cv2.warpAffine(star,M2,(cols2,rows2))`  
`fig,ax = plt.subplots(nrows=1, ncols=2)`  
`for i in range(2):`

```

a=ax[i]
if (i==0):
    a.imshow(rotatedbolt,cmap="gray")
else:
    a.imshow(rotatedstar,cmap="gray")

```



## 12. STRETCH GOAL:

Visualize the result of Laplacian, Sobel X, and Sobel Y on all of the images. Also, produce a combined image of both Sobel X and Sobel Y for each image. Is Exercise 1 the best way to do this? Are there other options?

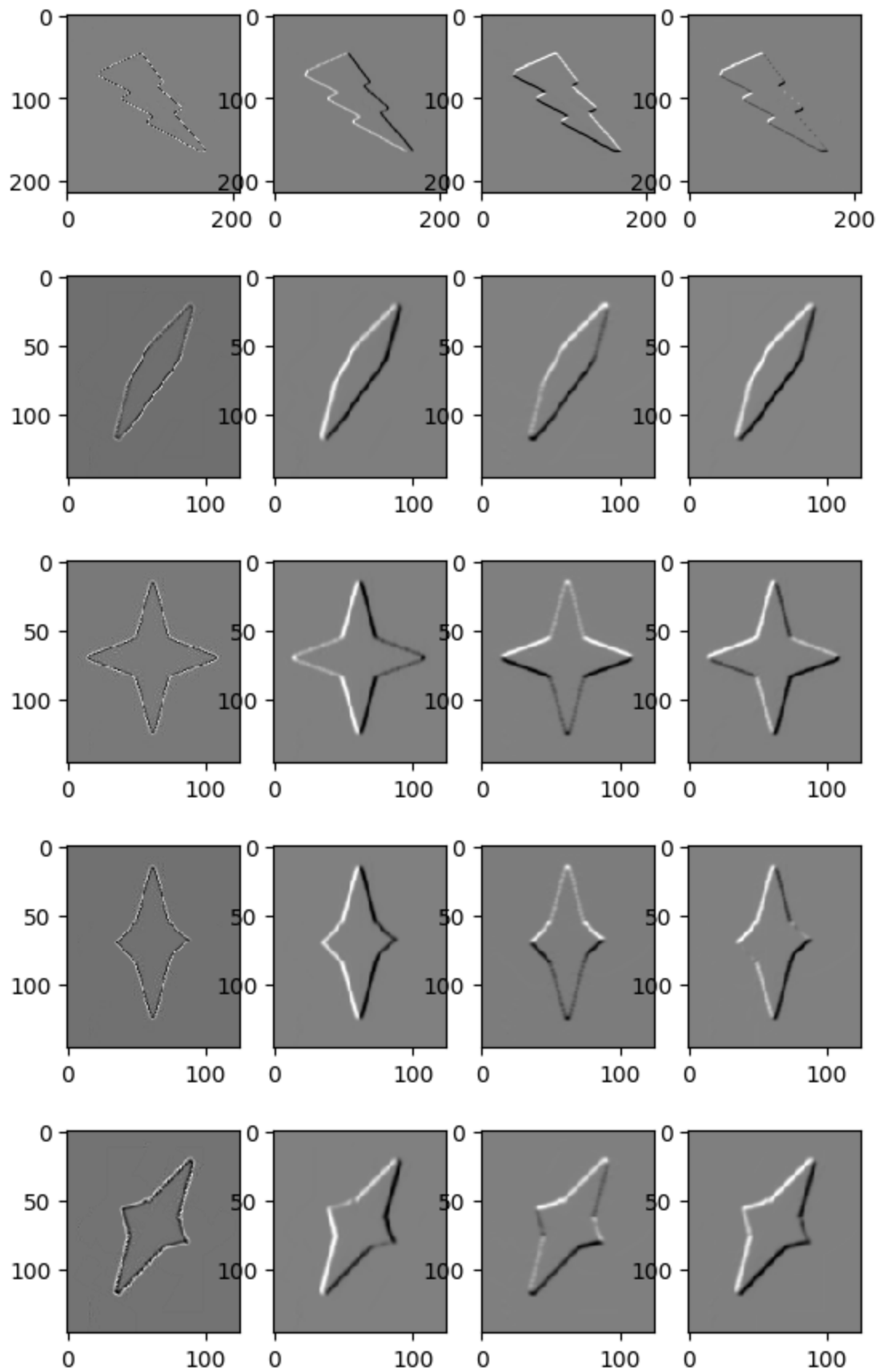
You should have 4 outputs (Laplacian, SobelX, SobelY, and the combination) for each input image visualized at the end.

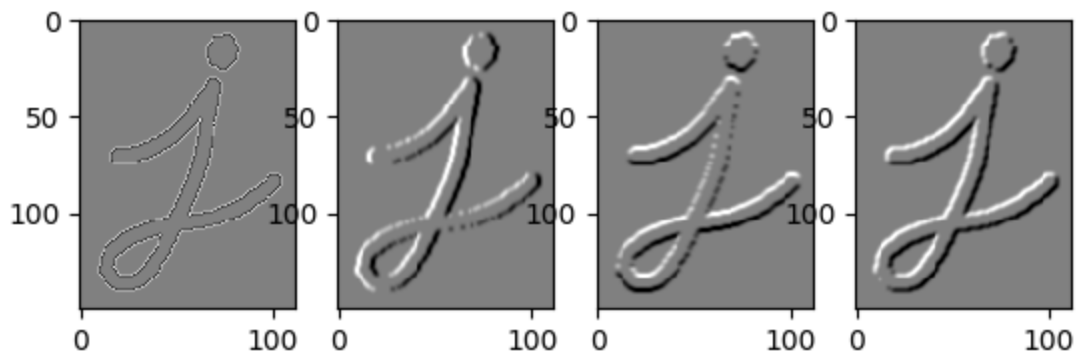
[https://docs.opencv.org/3.4.1/d5/d0f/tutorial\\_py\\_gradients.html](https://docs.opencv.org/3.4.1/d5/d0f/tutorial_py_gradients.html)

```

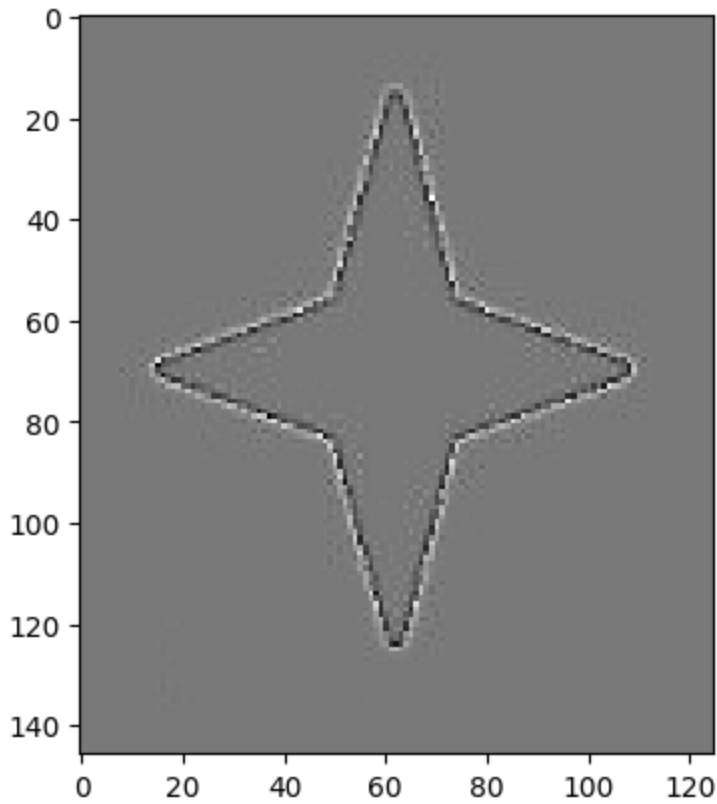
In [25]: def Laplace(img):
          return cv2.Laplacian(img,cv2.CV_64F)
          def SobelX(img):
              return cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
          def SobelY(img):
              return cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)
          def SobelSum(img):
              return cv2.addWeighted(SobelX(img),0.5,SobelY(img),0.5,0)
          transformations=[Laplace, SobelX, SobelY, SobelSum]
          for i in images:
              fig,ax = plt.subplots(nrows=1, ncols=4)
              for x in range(4):
                  a=ax.flatten()[x]
                  t=transformations[x]
                  a.imshow(t(i), cmap="gray", interpolation="none")

```



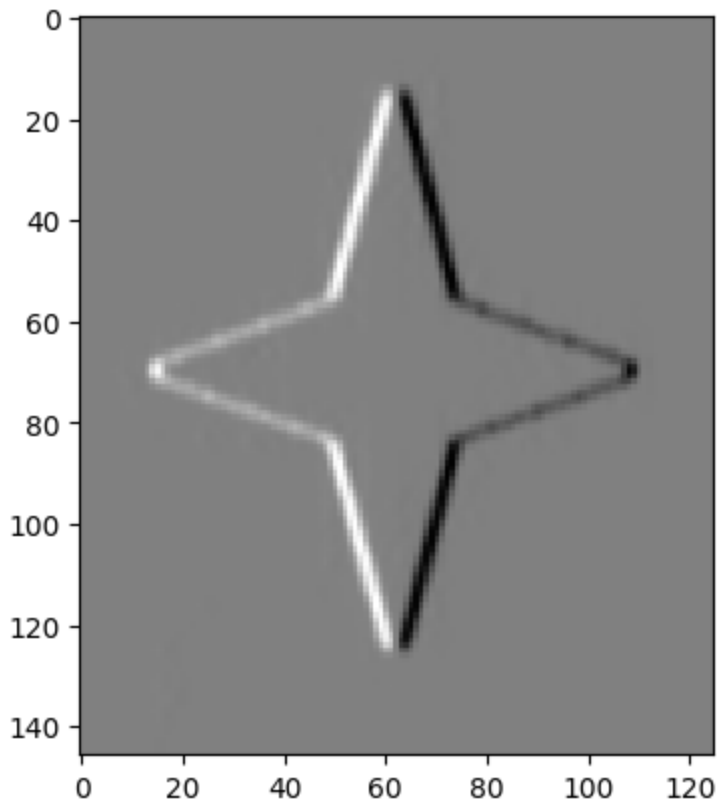


```
In [26]: plt.imshow(cv2.Laplacian(star, cv2.CV_64F), cmap="gray", interpolation="none")
Sobelx=cv2.Sobel(star, cv2.CV_64F, 1, 0, ksize=5)
Sobely=cv2.Sobel(star, cv2.CV_64F, 0, 1, ksize=5)
```



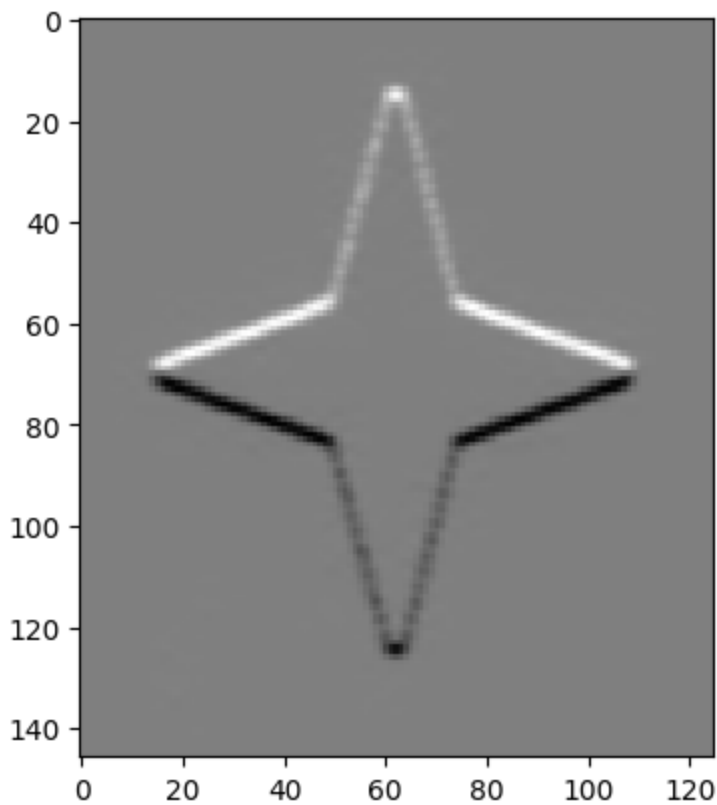
```
In [27]: plt.imshow(Sobelx, cmap="gray", interpolation="none")
```

```
Out[27]: <matplotlib.image.AxesImage at 0x11d1b4e50>
```



In [28]: `plt.imshow(Sobely,cmap="gray",interpolation="none")`

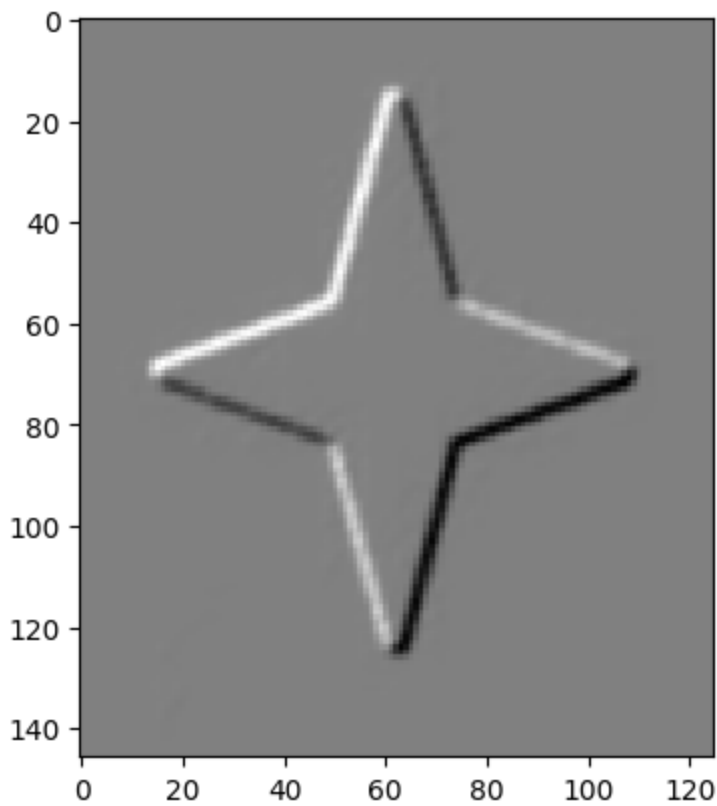
Out[28]: `<matplotlib.image.AxesImage at 0x11d1c4280>`



In [29]: `plt.imshow(cv2.addWeighted(Sobelx,0.5,Sobely,0.5,0),cmap="gray",interpolation='')`



Out[29]: <matplotlib.image.AxesImage at 0x11d241340>



## When you are done:

You should have one or more images for each exercise.

1. Double-check that you filled in your name at the top of the notebook!
2. Click **File -> Export Notebook As -> PDF**
3. Email the PDF to **YOURTEAMNAME@beaver.works**