# Line and Circle Detection

Armita Ashabyamin

# How did we start?

- We first start by importing numpy and opencv  ,and also matplot in order to show our results:

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
```

we then imported our input picture, converted it into a binary picture using opencv's thresh_binary built-in function.
We then imported our circle kernel which was built through trying out different sizes of circles and this one worked the best.
We also turned it into a binary picture for good measures.

```
#reading our input image:
input_image = cv.imread(r'F:\cv\Line-circle.png',0)

#converting our input image into binary
binr = cv.threshold(input_image, 0, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)[1]

#reading our circles kernel
input_kernel = cv.imread(r"F:\cv\kernel.jpg",0)
kernel = cv.threshold(input_kernel, 0, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)[1]
```

# This is the circle kernel we used:

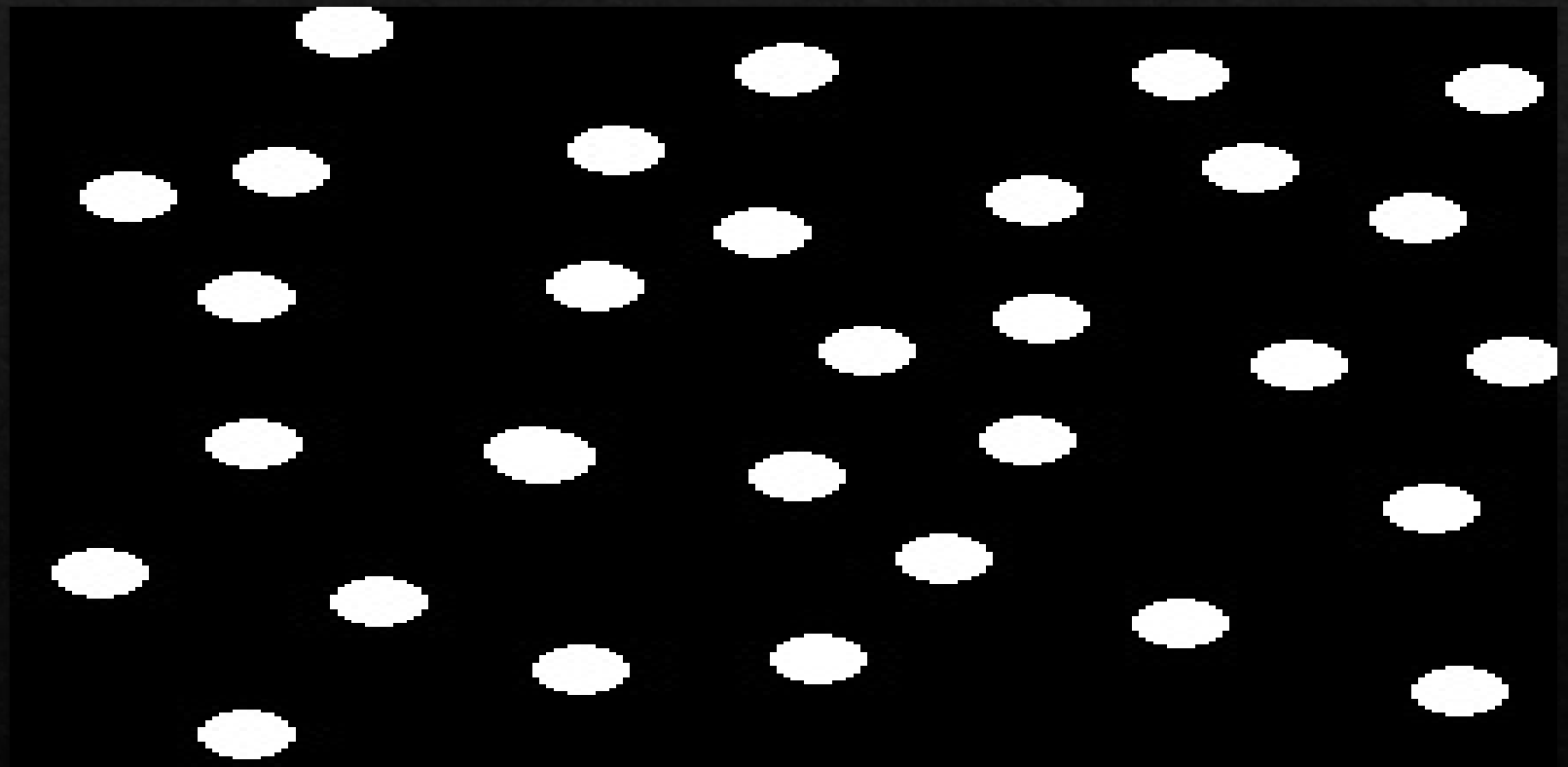
it's a 15*15 pixels kernel

We then found the circles using the opening function and just to make sure we also tried doing an erosion + dialation to test if they really are the same . we then saved the results.

```
#detecting the circles :
erosion = cv.erode(binr, kernel, iterations=1)

dilation = cv.dilate(erosion, kernel, iterations=2)

opening = cv.morphologyEx(binr, cv.MORPH_OPEN, kernel, iterations=1)

#saving the circles pic
cv.imwrite('circles.jpg', opening)
circles = cv.imread(r"F:\cv\circles.jpg",0)
```

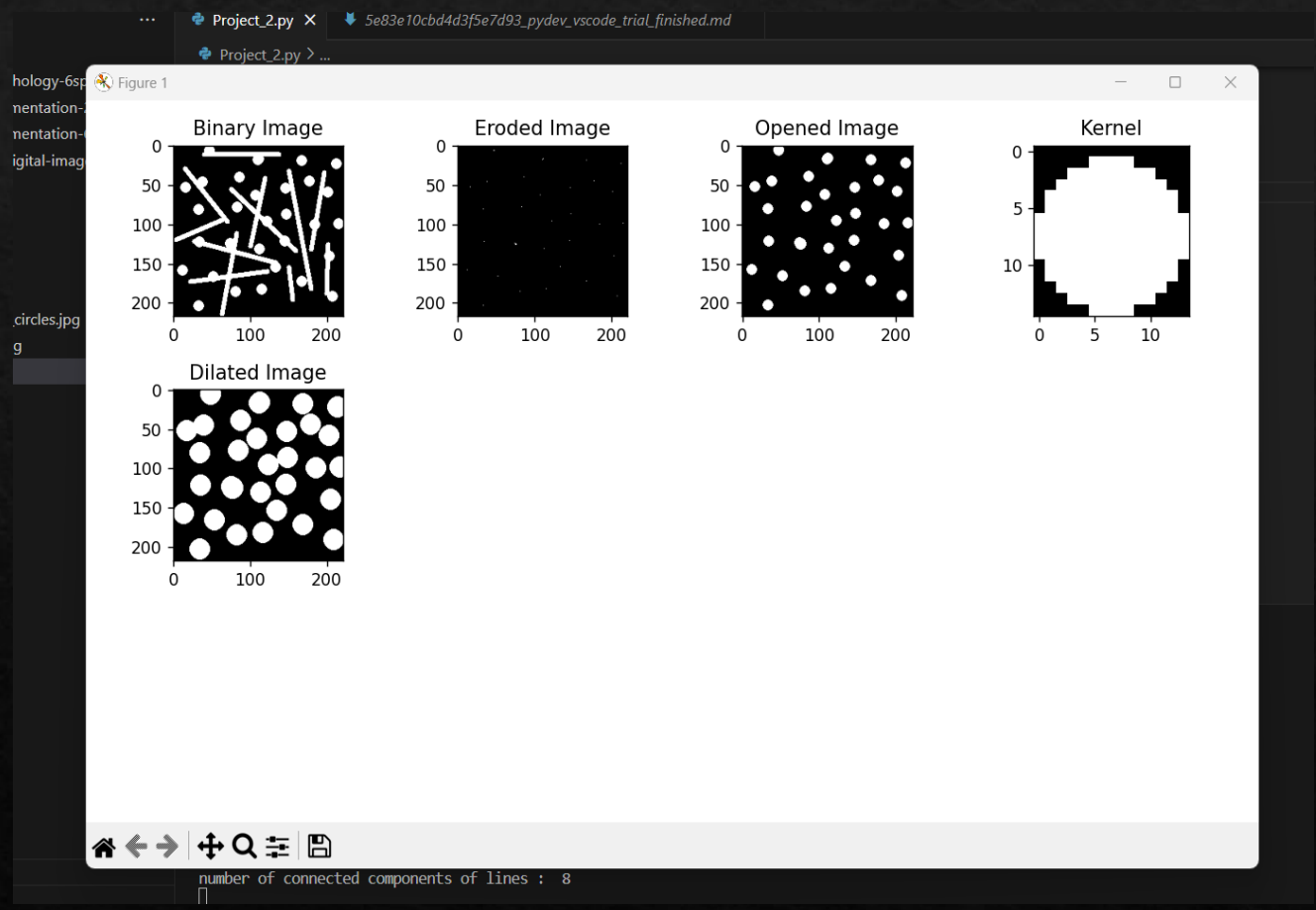
Note that the difference between our opened image and our erdoded + dilated image is due the iterations of dilation that we did to make the circles bigger.

# This is our circles result:



# Results shown in matplot:

```
plt.figure(figsize=(10,6))
plt.subplot(3,4,1)
plt.imshow(binr, cmap="gray")
plt.title('Binary Image')

plt.subplot(3,4,2)
plt.imshow(erosion, cmap="gray")
plt.title('Eroded Image')

plt.subplot(3,4,3)
plt.imshow(opening, cmap="gray")
plt.title('Opened Image')

plt.subplot(3,4,4)
plt.imshow(kernel , cmap="gray")
plt.title('Kernel')
```



# For finding our lines, we use 12 different kernels that are described as followed:

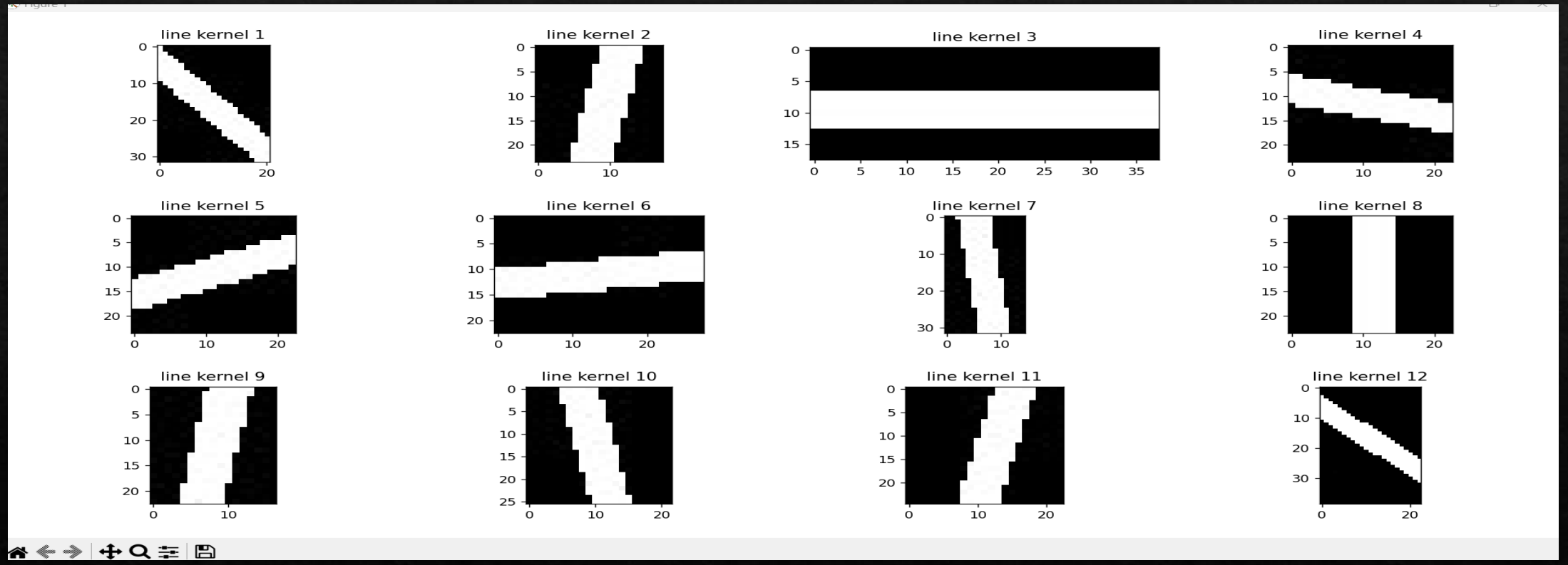
```
#finding our lines :

#defining our line kernels:
Line_kernel1 = cv.imread(r"F:\cv\Line_kernel1.jpg",0)
line_kernel1 = cv.threshold(Line_kernel1, 0, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)[1]
Line_kernel2 = cv.imread(r"F:\cv\Line_kernel2.jpg",0)
line_kernel2 = cv.threshold(Line_kernel2, 0, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)[1]
Line_kernel3 = cv.imread(r"F:\cv\Line_kernel3.jpg",0)
line_kernel3 = cv.threshold(Line_kernel3, 0, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)[1]
Line_kernel4 = cv.imread(r"F:\cv\Line_kernel4.jpg",0)
line_kernel4 = cv.threshold(Line_kernel4, 0, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)[1]
Line_kernel5 = cv.imread(r"F:\cv\Line_kernel5.jpg",0)
line_kernel5 = cv.threshold(Line_kernel5, 0, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)[1]
Line_kernel6 = cv.imread(r"F:\cv\Line_kernel6.jpg",0)
line_kernel6 = cv.threshold(Line_kernel6, 0, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)[1]
Line_kernel7 = cv.imread(r"F:\cv\Line_kernel7.jpg",0)
line_kernel7 = cv.threshold(Line_kernel7, 0, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)[1]
Line_kernel8 = cv.imread(r"F:\cv\Line_kernel8.jpg",0)
line_kernel8 = cv.threshold(Line_kernel8, 0, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)[1]
Line_kernel9 = cv.imread(r"F:\cv\Line_kernel9.jpg",0)
line_kernel9 = cv.threshold(Line_kernel9, 0, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)[1]
Line_kernel10 = cv.imread(r"F:\cv\Line_kernel10.jpg",0)
line_kernel10 = cv.threshold(Line_kernel10, 0, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)[1]
Line_kernel11 = cv.imread(r"F:\cv\Line_kernel11.jpg",0)
line_kernel11 = cv.threshold(Line_kernel11, 0, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)[1]
Line_kernel12 = cv.imread(r"F:\cv\Line_kernel12.png",0)
line_kernel12 = cv.threshold(Line_kernel12, 0, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)[1]
```

# To demonstrate the different kernels we use matplot :

```
plt.subplot(3,4,1)
plt.imshow(Line_kernel1,cmap="gray")
plt.title("line kernel 1")


plt.subplot(3,4,2)
plt.imshow(Line_kernel2,cmap="gray")
plt.title("line kernel 2")


plt.subplot(3,4,3)
plt.imshow(Line_kernel3,cmap="gray")
plt.title("line kernel 3")


plt.subplot(3,4,4)
plt.imshow(Line_kernel4,cmap="gray")
plt.title("line kernel 4")


plt.subplot(3,4,5)
plt.imshow(Line_kernel5,cmap="gray")
plt.title("line kernel 5")


plt.subplot(3,4,6)
plt.imshow(Line_kernel6,cmap="gray")
plt.title("line kernel 6")


plt.subplot(3,4,7)
plt.imshow(Line_kernel7,cmap="gray")
plt.title("line kernel 7")
```

```
plt.subplot(3,4,7)
plt.imshow(Line_kernel7,cmap="gray")
plt.title("line kernel 7")


plt.subplot(3,4,8)
plt.imshow(Line_kernel8,cmap="gray")
plt.title("line kernel 8")


plt.subplot(3,4,9)
plt.imshow(Line_kernel9,cmap="gray")
plt.title("line kernel 9")


plt.subplot(3,4,10)
plt.imshow(Line_kernel10,cmap="gray")
plt.title("line kernel 10")


plt.subplot(3,4,11)
plt.imshow(Line_kernel11,cmap="gray")
plt.title("line kernel 11")


plt.subplot(3,4,12)
plt.imshow(Line_kernel12,cmap="gray")
plt.title("line kernel 12")


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

# These are our kernels:



We then use our kernels on our input picture's binary form(binr) , we first perform an opening and then a closing on our image with each kernel and then in the end, we use logical OR to combine all of our results with different kernels and form our lines.

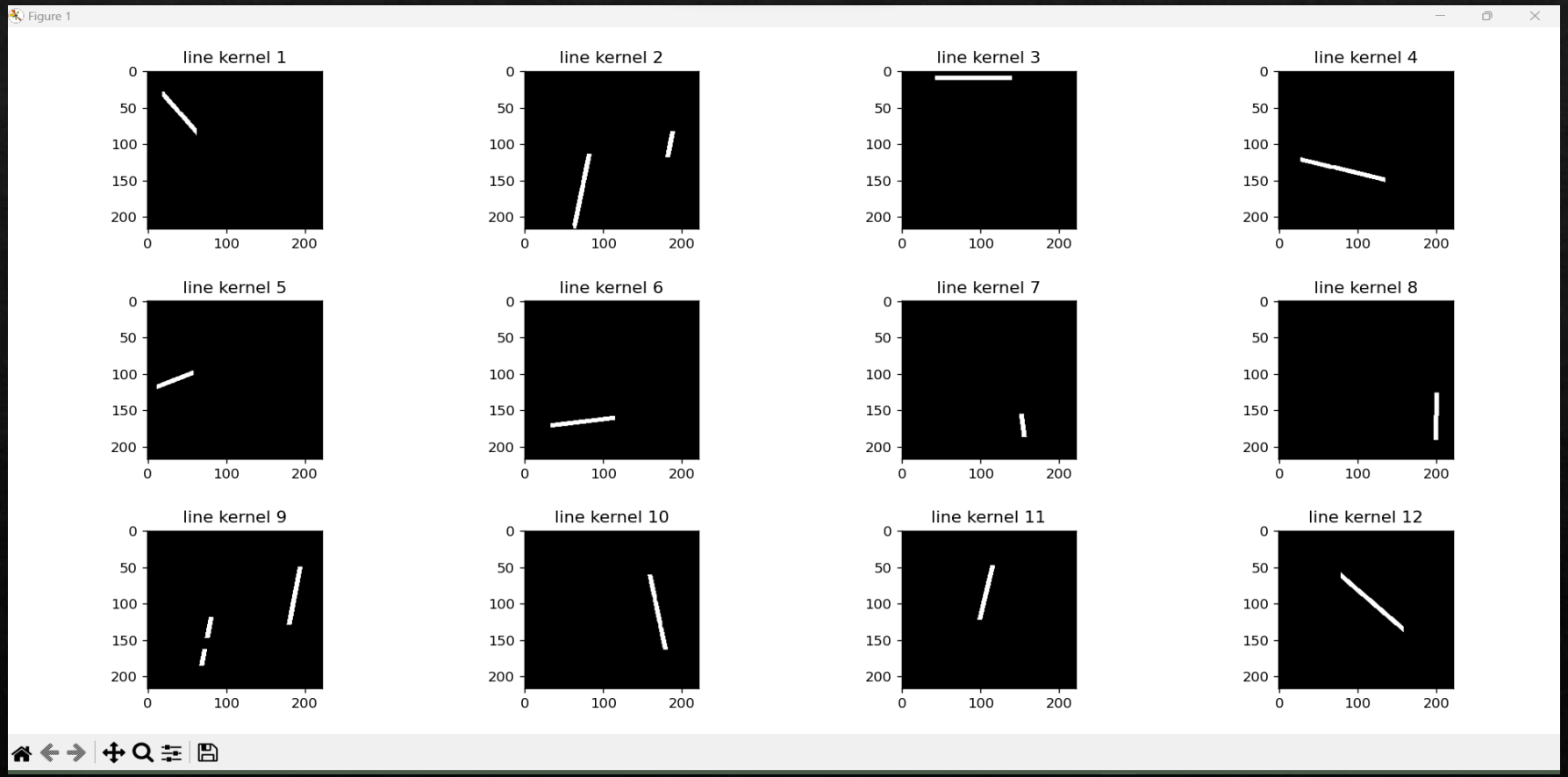
```
#using our kernels to find the lines:

opening_kernel1 = cv.morphologyEx(binr, cv.MORPH_OPEN, line_kernel1, iterations=1)
closing_kernel1 = cv.morphologyEx(opening_kernel1, cv.MORPH_CLOSE, kernel, iterations=1)
opening_kernel2 = cv.morphologyEx(binr, cv.MORPH_OPEN, line_kernel2, iterations=1)
closing_kernel2 = cv.morphologyEx(opening_kernel2, cv.MORPH_CLOSE, kernel, iterations=1)
opening_kernel3 = cv.morphologyEx(binr, cv.MORPH_OPEN, line_kernel3, iterations=1)
closing_kernel3 = cv.morphologyEx(opening_kernel3, cv.MORPH_CLOSE, kernel, iterations=1)
opening_kernel4 = cv.morphologyEx(binr, cv.MORPH_OPEN, line_kernel4, iterations=1)
closing_kernel4 = cv.morphologyEx(opening_kernel4, cv.MORPH_CLOSE, kernel, iterations=1)
opening_kernel5 = cv.morphologyEx(binr, cv.MORPH_OPEN, line_kernel5, iterations=1)
closing_kernel5 = cv.morphologyEx(opening_kernel5, cv.MORPH_CLOSE, kernel, iterations=1)
opening_kernel6 = cv.morphologyEx(binr, cv.MORPH_OPEN, line_kernel6, iterations=1)
closing_kernel6 = cv.morphologyEx(opening_kernel6, cv.MORPH_CLOSE, kernel, iterations=1)
opening_kernel7 = cv.morphologyEx(binr, cv.MORPH_OPEN, line_kernel7, iterations=1)
closing_kernel7 = cv.morphologyEx(opening_kernel7, cv.MORPH_CLOSE, kernel, iterations=1)
opening_kernel8 = cv.morphologyEx(binr, cv.MORPH_OPEN, line_kernel8, iterations=1)
closing_kernel8 = cv.morphologyEx(opening_kernel8, cv.MORPH_CLOSE, kernel, iterations=1)
opening_kernel9 = cv.morphologyEx(binr, cv.MORPH_OPEN, line_kernel9, iterations=1)
closing_kernel9 = cv.morphologyEx(opening_kernel9, cv.MORPH_CLOSE, kernel, iterations=1)
opening_kernel10 = cv.morphologyEx(binr, cv.MORPH_OPEN, line_kernel10, iterations=1)
closing_kernel10 = cv.morphologyEx(opening_kernel10, cv.MORPH_CLOSE, kernel, iterations=1)
opening_kernel11 = cv.morphologyEx(binr, cv.MORPH_OPEN, line_kernel11, iterations=1)
closing_kernel11 = cv.morphologyEx(opening_kernel11, cv.MORPH_CLOSE, kernel, iterations=1)
opening_kernel12 = cv.morphologyEx(binr, cv.MORPH_OPEN, line_kernel12, iterations=1)
closing_kernel12 = cv.morphologyEx(opening_kernel12, cv.MORPH_CLOSE, kernel, iterations=1)

Lines = (closing_kernel1 + closing_kernel2 + closing_kernel3 + closing_kernel4 + closing_kernel5 + closing_kernel6 +
closing_kernel7 + closing_kernel8 + closing_kernel9 + closing_kernel10 + closing_kernel11 + closing_kernel12)
```

# The code to show each kernels result in matplot

```
#showing the result of each line kernel:

plt.subplot(3,4,1)
plt.imshow(closing_kernel1,cmap="gray")
plt.title("line kernel 1")


plt.subplot(3,4,2)
plt.imshow(closing_kernel2,cmap="gray")
plt.title("line kernel 2")


plt.subplot(3,4,3)
plt.imshow(closing_kernel3,cmap="gray")
plt.title("line kernel 3")


plt.subplot(3,4,4)
plt.imshow(closing_kernel4,cmap="gray")
plt.title("line kernel 4")


plt.subplot(3,4,5)
plt.imshow(closing_kernel5,cmap="gray")
plt.title("line kernel 5")


plt.subplot(3,4,6)
plt.imshow(closing_kernel6,cmap="gray")
plt.title("line kernel 6")


plt.subplot(3,4,7)
plt.imshow(closing_kernel7,cmap="gray")
plt.title("line kernel 7")
```

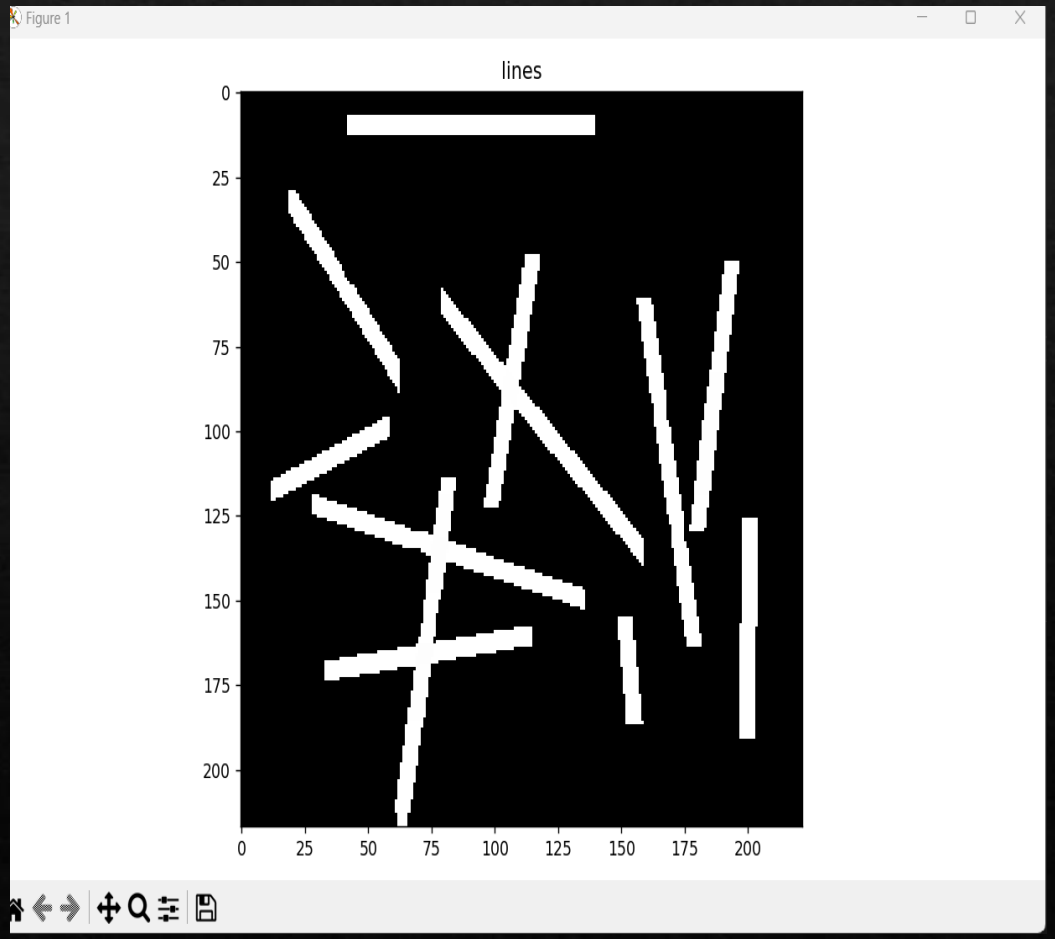
```
plt.subplot(3,4,8)
plt.imshow(closing_kernel8,cmap="gray")
plt.title("line kernel 8")


plt.subplot(3,4,9)
plt.imshow(closing_kernel9,cmap="gray")
plt.title("line kernel 9")


plt.subplot(3,4,10)
plt.imshow(closing_kernel10,cmap="gray")
plt.title("line kernel 10")


plt.subplot(3,4,11)
plt.imshow(closing_kernel11,cmap="gray")
plt.title("line kernel 11")


plt.subplot(3,4,12)
plt.imshow(closing_kernel12,cmap="gray")
plt.title("line kernel 12")
```

# The result of each kernel :



# The line that was combined:

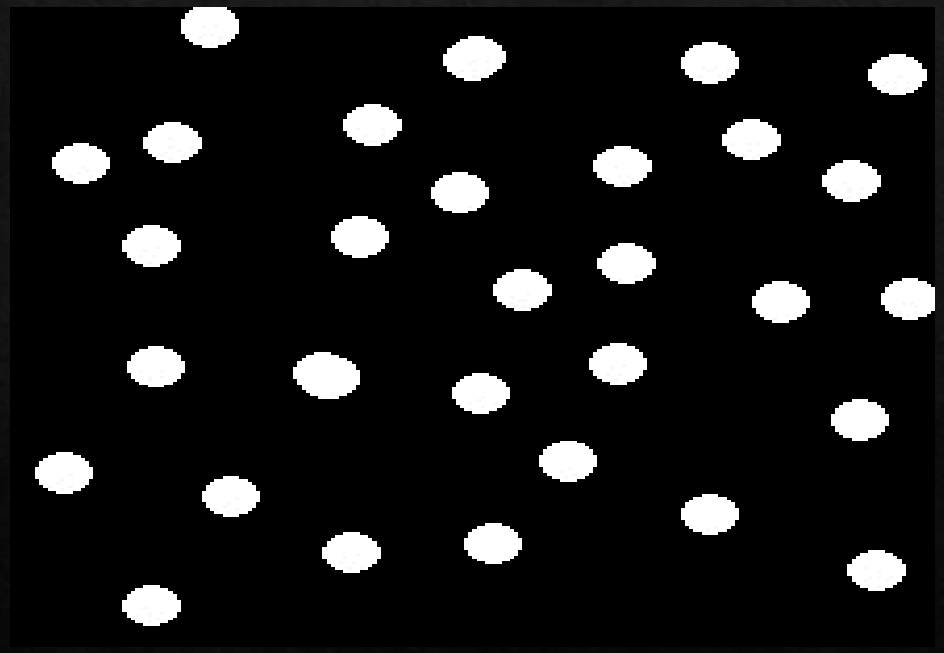
```
#showing the results
plt.figure(figsize=(10,6))
plt.subplot(1,1,1)
plt.imshow(Lines,cmap="gray")
plt.title("lines")
```

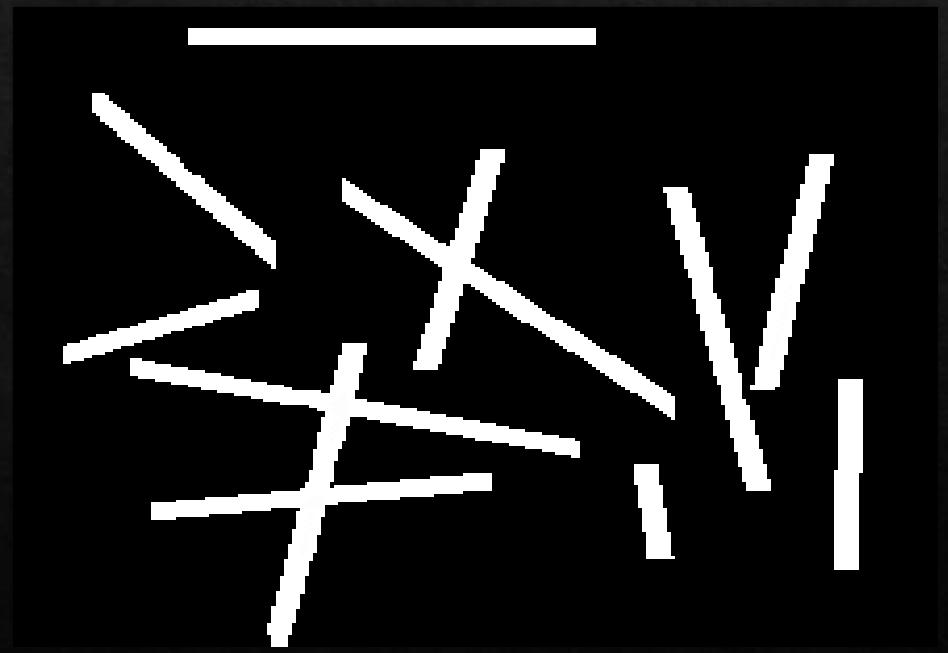


We save our results as line.bmp and circle.bmp as instructed using the following code :

```
#saving the results:
cv.imwrite('line.bmp',Lines)
cv.imwrite('circle.bmp',circles)
```

# Here is our overall result:

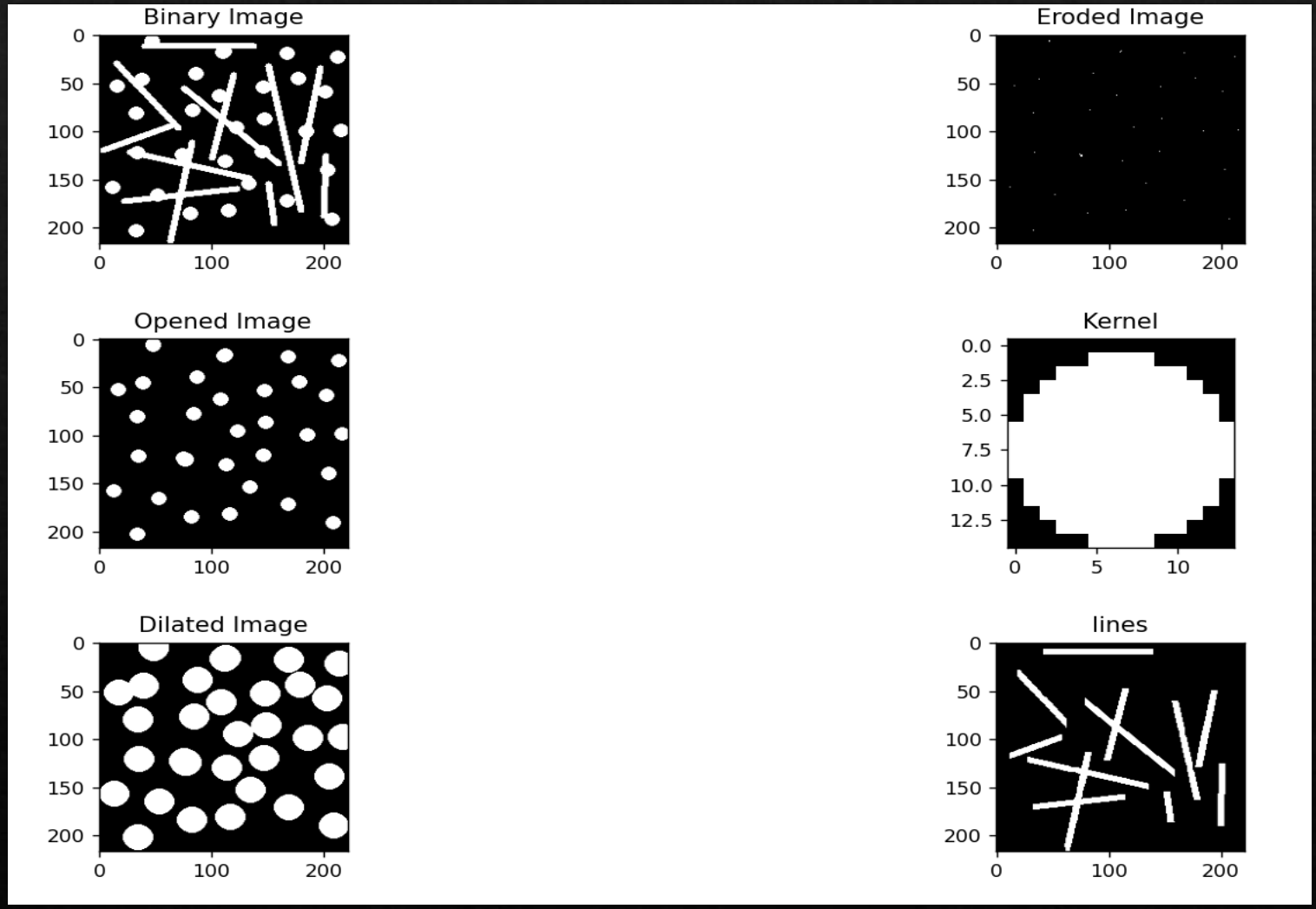




# To show the result in matplot:

```
#showing the results
plt.figure(figsize=(10,6))

plt.subplot(3,2,1)
plt.imshow(binr, cmap="gray")
plt.title('Binary Image')

plt.subplot(3,2,2)
plt.imshow(erosion, cmap="gray")
plt.title('Eroded Image')

plt.subplot(3,2,3)
plt.imshow(opening, cmap="gray")
plt.title('Opened Image')

plt.subplot(3,2,4)
plt.imshow(kernel , cmap="gray")
plt.title('Kernel')

plt.subplot(3,2,5)
plt.imshow(dilation, cmap="gray")
plt.title('Dilated Image')

plt.subplot(3,2,6)
plt.imshow(Lines,cmap="gray")
plt.title("lines")
```

# Our Result


Binary Image
Eroded Image
Opened Image
Kernel
Dilated Image
lines

Counting the circles and lines:

For counting the circles we use the built-in function of opencv . In order to use the built-in function, we first need to convert our image into a binary picture.

```python
#our first attempt to count the lines:

line_image = cv.imread('line.bmp', cv.IMREAD_GRAYSCALE)
ret_line, thresh_line = cv.threshold(line_image,127,255,cv.THRESH_BINARY)
num_labels_line,lables_line= cv.connectedComponents(thresh_line)
print("number of connected components of lines : ",num_labels_line-1)
```

```
PS F:\cv> & C:/Users/SamRayaneh/AppData/Local/Programs/Python/Python312/python.exe f:/cv/Project_2.py
number of connected components of circles :  30
```

For counting the lines we faced a few difficulties. One being that by using the connected components algorithm, any lines that had an intersection would count as one big component and that would disrupt our count

```
line_image = cv.imread('line.bmp', cv.IMREAD_GRAYSCALE)
ret_line, thresh_line = cv.threshold(line_image,127,255,cv.THRESH_BINARY)
num_labels_line,lables_line= cv.connectedComponents(thresh_line)
print("number of connected components of lines : ",num_labels_line-1)
```

```
number of connected components of lines :  9
```

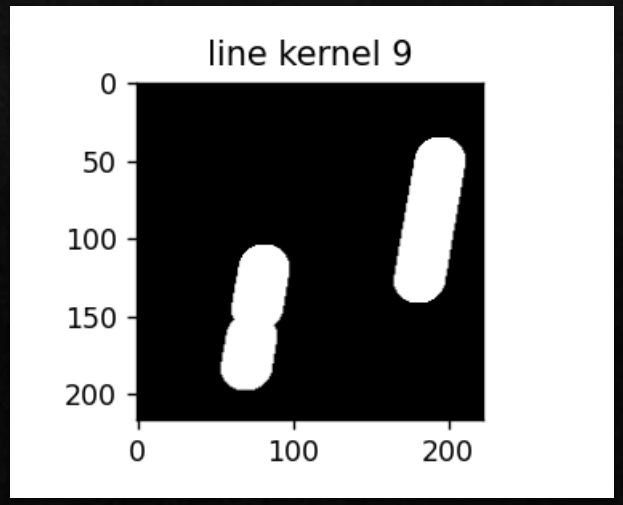
In order to fix that we decided to perform the connected component connection for each of our kernels’results and with that we were able to have an accurate answer

So using the same counting algorithm, we do the following set of instructions for each line kernel result:

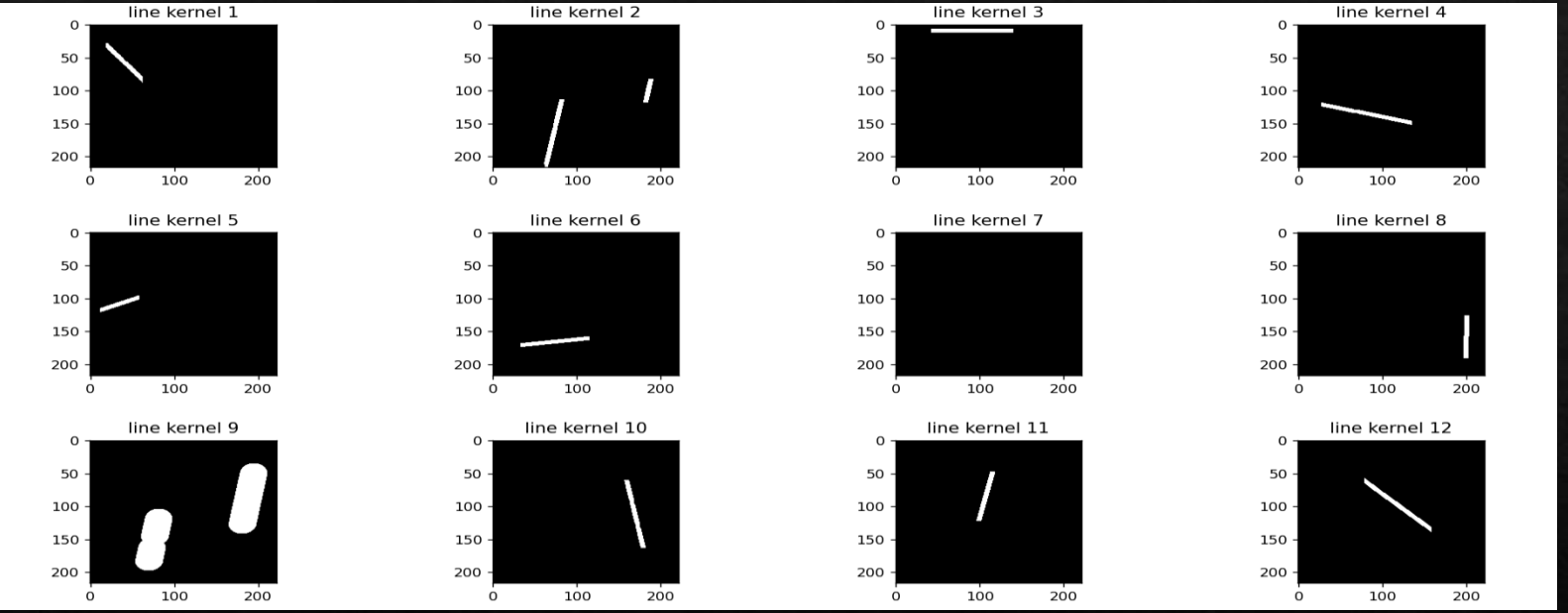
```
#counting the lines:

total_line_count = 0

line_kernel_1 = cv.imread("closing_kernel1.png", cv.IMREAD_GRAYSCALE)
ret1, thresh1 = cv.threshold(line_kernel_1,127,255,cv.THRESH_BINARY)
num_labels1,lables1 = cv.connectedComponents(thresh1)
total_line_count = total_line_count + num_labels1 - 1
#-1 is for deducting the background as a component
```

For us to have a united line, we then perform a dilation with iteration=2 to have a more accurate count on the lines.

```
closing_kernel9 = cv.dilate(closing_kernel9,kernel,iterations=2)
```



# So the overall kernels look like this:




Note: We could do the dilation for each kernel but we thought it was unnecessary here because we knew that only this kernel needed it.

# Part of our code:

```
#counting the lines:

total_line_count = 0

line_kernel_1 = cv.imread("closing_kernel1.png", cv.IMREAD_GRAYSCALE)
ret1, thresh1 = cv.threshold(line_kernel_1,127,255,cv.THRESH_BINARY)
num_labels1,lables1 = cv.connectedComponents(thresh1)
total_line_count = total_line_count + num_labels1 - 1
#-1 is for deducting the background as a component


line_kernel_2 = cv.imread("closing_kernel2.png", cv.IMREAD_GRAYSCALE)
ret2, thresh2 = cv.threshold(line_kernel_2,127,255,cv.THRESH_BINARY)
num_labels2,lables2 = cv.connectedComponents(thresh2)
total_line_count = total_line_count + num_labels2 - 2
#the other -1 is because of the little line that appeared but is not a complete one in this kernel


line_kernel_3 = cv.imread("closing_kernel3.png", cv.IMREAD_GRAYSCALE)
ret3, thresh3 = cv.threshold(line_kernel_3,127,255,cv.THRESH_BINARY)
num_labels3,lables3 = cv.connectedComponents(thresh3)
total_line_count = total_line_count + num_labels3 - 1

line_kernel_4 = cv.imread("closing_kernel4.png", cv.IMREAD_GRAYSCALE)
ret4, thresh4 = cv.threshold(line_kernel_4,127,255,cv.THRESH_BINARY)
num_labels4,lables4 = cv.connectedComponents(thresh4)
total_line_count = total_line_count + num_labels4 - 1


line_kernel_5 = cv.imread("closing_kernel5.png", cv.IMREAD_GRAYSCALE)
ret5, thresh5 = cv.threshold(line_kernel_5,127,255,cv.THRESH_BINARY)
num_labels5,lables5 = cv.connectedComponents(thresh5)
total_line_count = total_line_count + num_labels5 - 1
```

# The result :

```
183   #-1 is for deducting the background as a component
184
185
186   line_kernel_2 = cv.imread("closing_kernel2.png", cv.IMREAD_GRAYSCALE)
187   ret2, thresh2 = cv.threshold(line_kernel_2,127,255,cv.THRESH_BINARY)
188   num_labels2,lables2 = cv.connectedComponents(thresh2)
189   total_line_count = total_line_count + num_labels2 - 2
190   #the other -1 is because of the little line that appeared but is not a complete one in this kernel
191
192
193   line_kernel_3 = cv.imread("closing_kernel3.png", cv.IMREAD_GRAYSCALE)
194   ret3, thresh3 = cv.threshold(line_kernel_3,127,255,cv.THRESH_BINARY)
195   num_labels3,lables3 = cv.connectedComponents(thresh3)
196   total_line_count = total_line_count + num_labels3 - 1
```

PROBLEMS 42 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS F:\cv> & C:/Users/SamRayaneh/AppData/Local/Programs/Python/Python312/python.exe f:/cv/Project_2.py
number of connected components of circles :  30
number of lines :  12
```

# The End

author:
Armita Ashabyamin
under the surveillance of :
Professor Khosravi