BLG 453E Computer Vision (Fall 2018)
Homework-4

Alperen Kantarcı - 150140140
December 24, 2018

1 Application Usage

Note: There is two different apps for parts A and B which are located in two different directories but usage of these programs are similar. In order to use both apps you need "Mainapp.py ui.py popup.py"" and if you have these python files and the requirements that written on the Github site then you can use this command to run the application: python Mainapp.py
Github page of this application is here.

2 Corner Detection Application

Application has two screens for input and result. You can see the initial and final states of the applications. First you should import image from file menubar. After you add the image if you click to the detect then after a few seconds application will show you the result which is the corners of the image. In below i will explain each step of the corner detection. In Figure 3,4 you can see the initial and final images.

Figure 1: Initial view of the corner detection

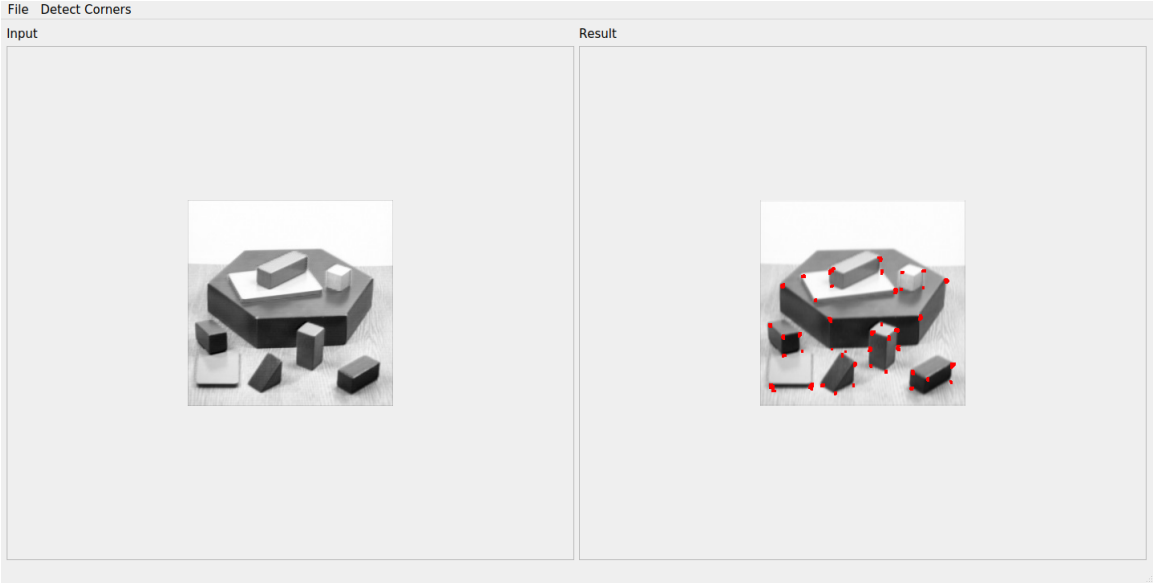

Figure 2: Final view of the program

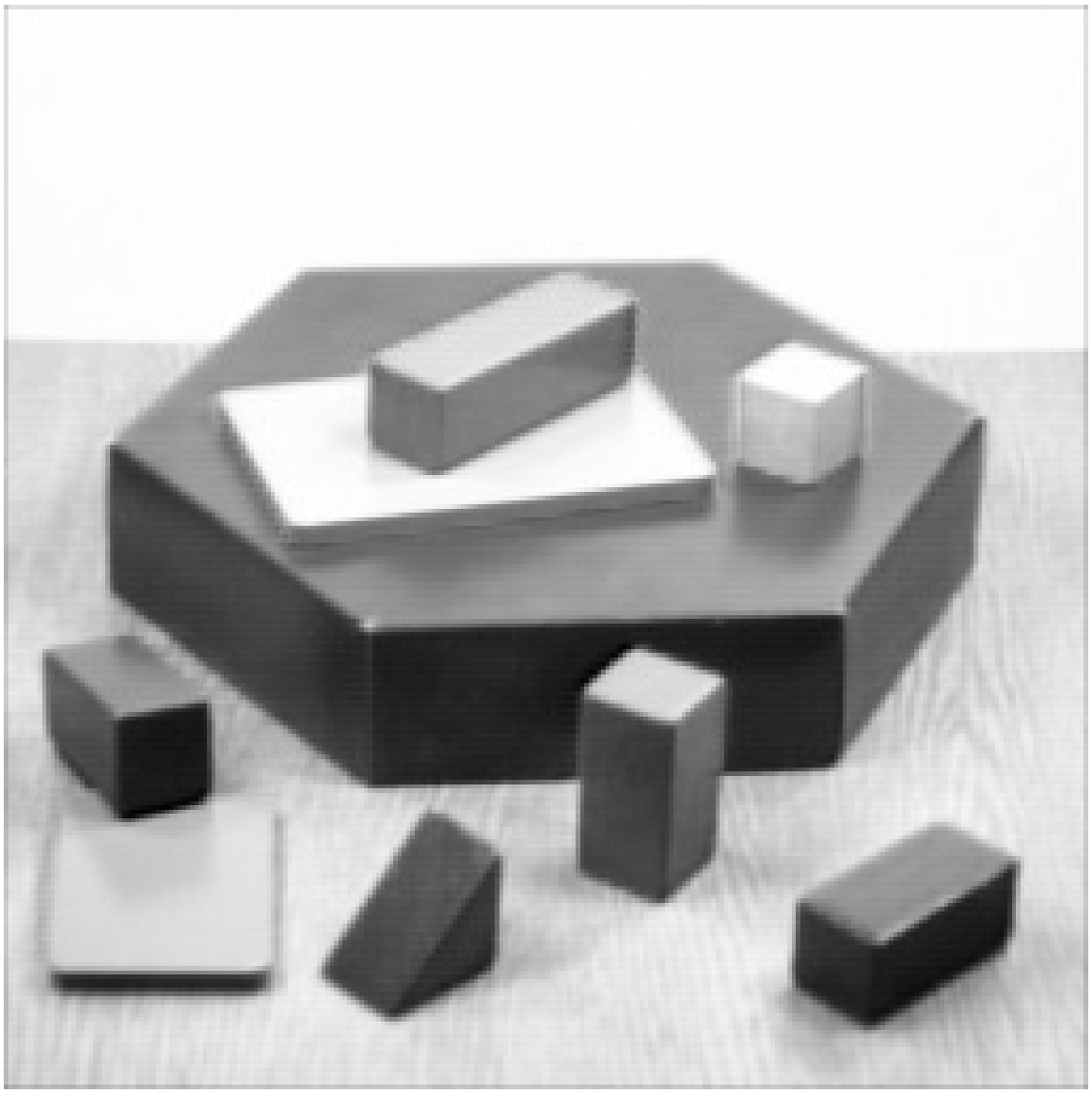

Figure 3: Blocks image

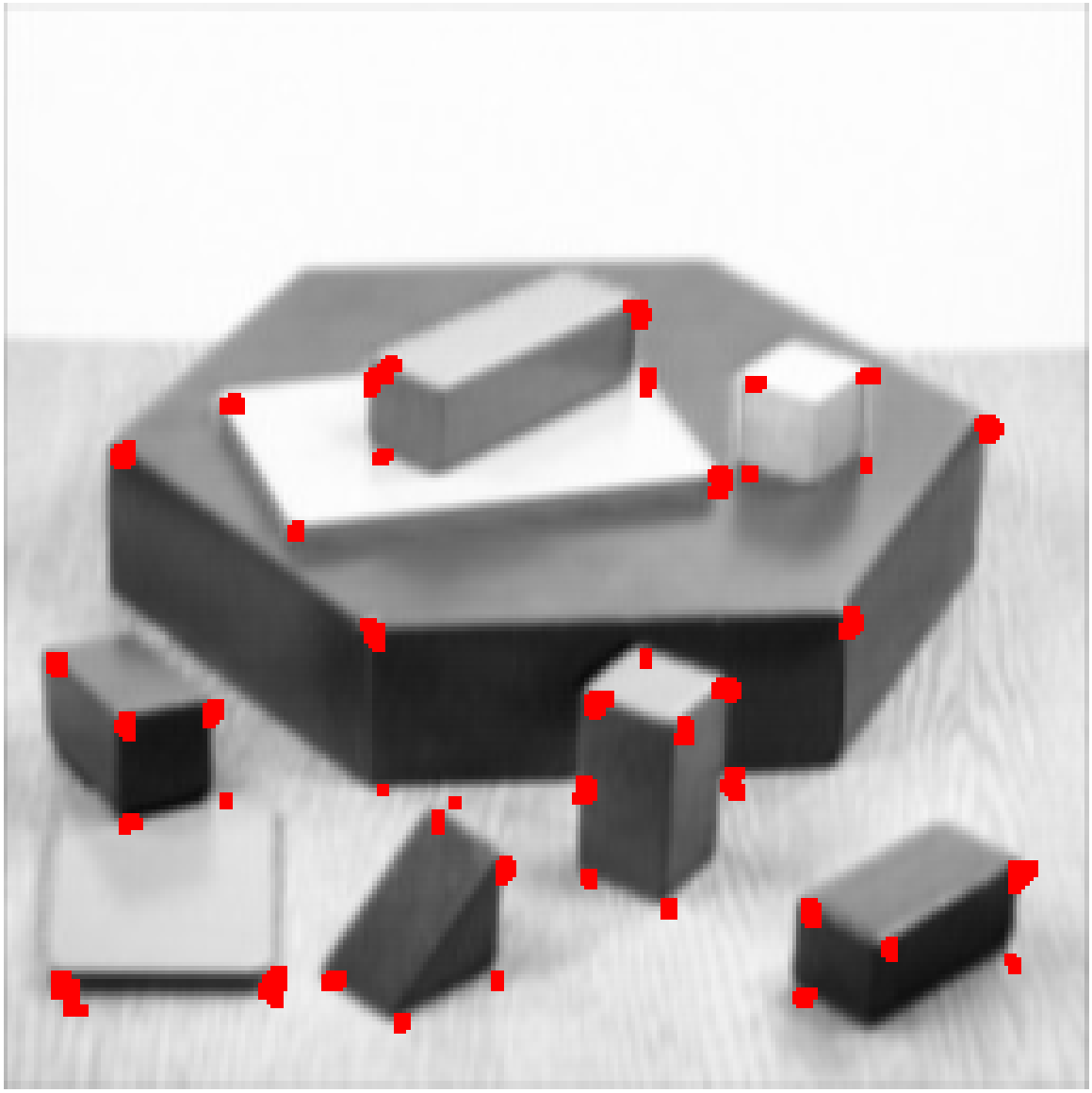

Figure 4: Corners of the blocks

3 Python code

After importing image first we smooth the image with 0.0125 sigma and 3x3 filter. Then i add padding to the image because i don't want to encounter with edge overflows.

```
smoothed_image = cv2.GaussianBlur(self.input_image_arr,(3,3),0.0125,0,
    cv2.BORDER_DEFAULT)
smoothed_image = np.pad(smoothed_image,(  (pad_s,pad_s),(pad_s,pad_s)
    ,(0,0)  ),mode='edge')
```

Then for each pixel i calculate x and y derivatives using derivative_x , derivative_y functions which are calculate centered discrete derivatives.

```
def derivative_x(self,arr,x,y):
  # Centered x derivative
  return (int(arr[x+1,y]) - int(arr[x-1,y])) / 2

def derivative_y(self,arr,x,y):
  # Centered y derivative
  return (int(arr[x,y+1]) - int(arr[x,y-1])) / 2

# Calculate x and y derivatives
for i in range(pad_s,height-pad_s):
  for j in range(pad_s,width-pad_s):
    padded_derivative[i,j] = (self.derivative_x(one_channel,i,j),self.
        derivative_y(one_channel,i,j))
```

Then by using windowsize=3 i calculate G image structure tensor with the functions that i will explain below. Then if pixel response is higher than treshold then paint that pixel to red

```
w_size = 3
# Treshold for accepting a corner
treshold = 1000000
# If pixel is above the treshold than mark as corner point
for i in range(pad_s,height-pad_s):
  for j in range(pad_s,width-pad_s):
    if self.isCorner(treshold,self.calculateG(i,j,padded_derivative,
        w_size)):
      # Paint the pixels to red
      smoothed_image[i-2:i+1,j-2:j+1,:] = [0,0,255]


def calculateG(self,x,y,derivatives,w_size):
                # Usin x and y derivatives create image structure kernel
                xsquare = 0
                xy = 0
                ysquare = 0
                w = w_size//2
                for i in range(x-w,x+w+1):
                        for j in range(y-w,y+w+1):
                                xsquare += derivatives[i,j][0]**2
                                xy += derivatives[i,j][0]*derivatives[i,
                                    j][1]
                                ysquare += derivatives[i,j][1]**2

                G = np.array([[xsquare,xy],[xy,ysquare]])
                return G

def isCorner(self,treshold,G):
                # Calculate eigen values
                eigvals = np.linalg.eigvals(G)
                # Small k value
                k = 0.04
```

```
# Calculate R
R = (eigvals[0]*eigvals[1] - k * (eigvals[0]+eigvals[1])
    **2)
# If R is above the treshold then it is a corner
if R > treshold:
        return True
return False
```

4 Tumor Segmentation Application

Application has two screens for input and results. You can see the initial and final states of the applications. First you should import image from file menubar. After you add the image if you click to the segment then after a few seconds application will show you the results which is the a) Tresholded mask b) Brain mask c) Segmentated tumor result. You can see the resulting images in Figure 8,9,10 in the same order. In below i will explain each step of the segmentation. In Figure 6,7 you can see the initial and final views of the program.

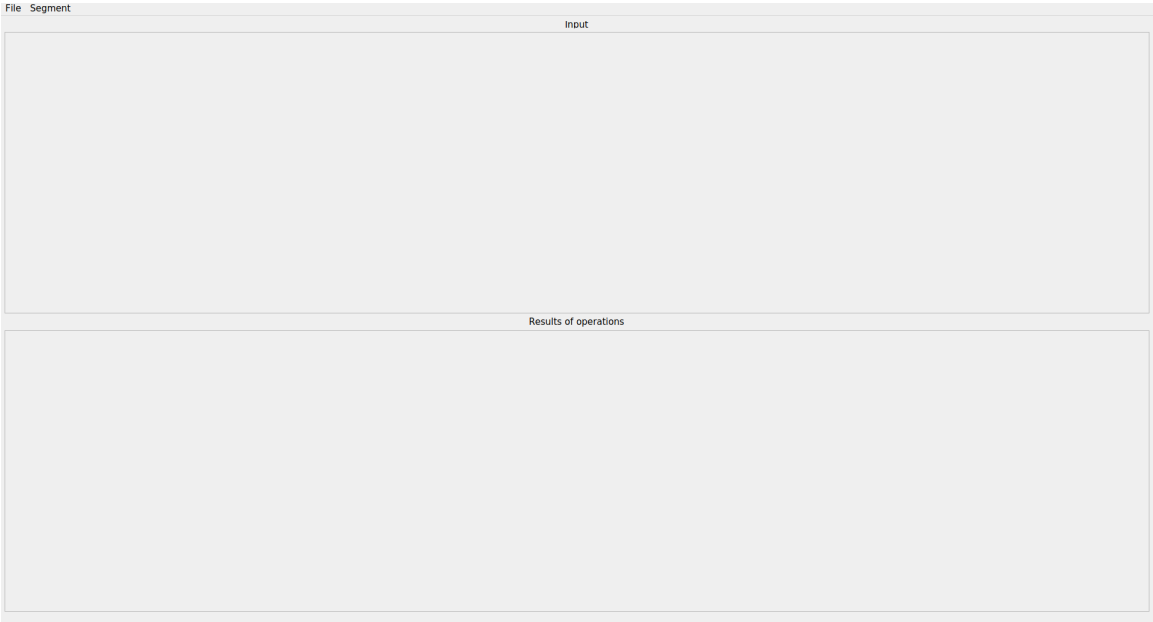

Figure 5: Initial view of the segmentation program

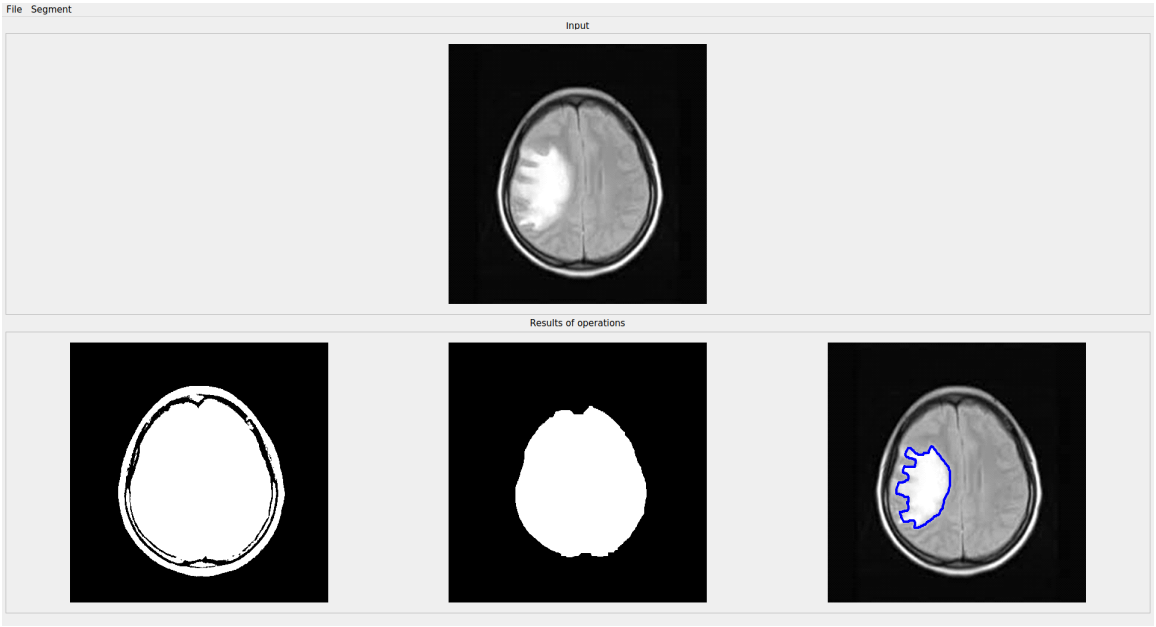

Figure 6: Final view of the program

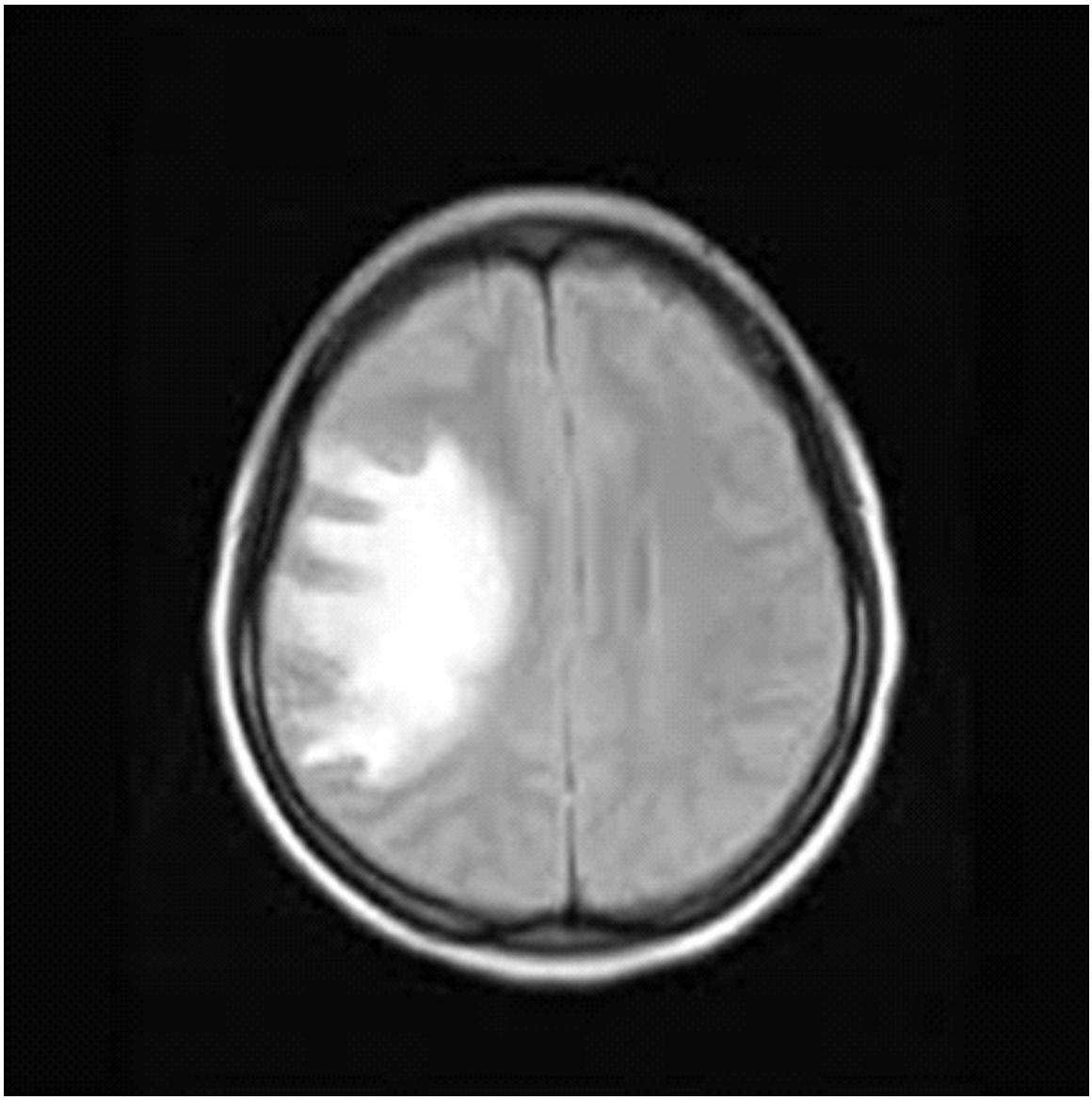

Figure 7: Input MR image

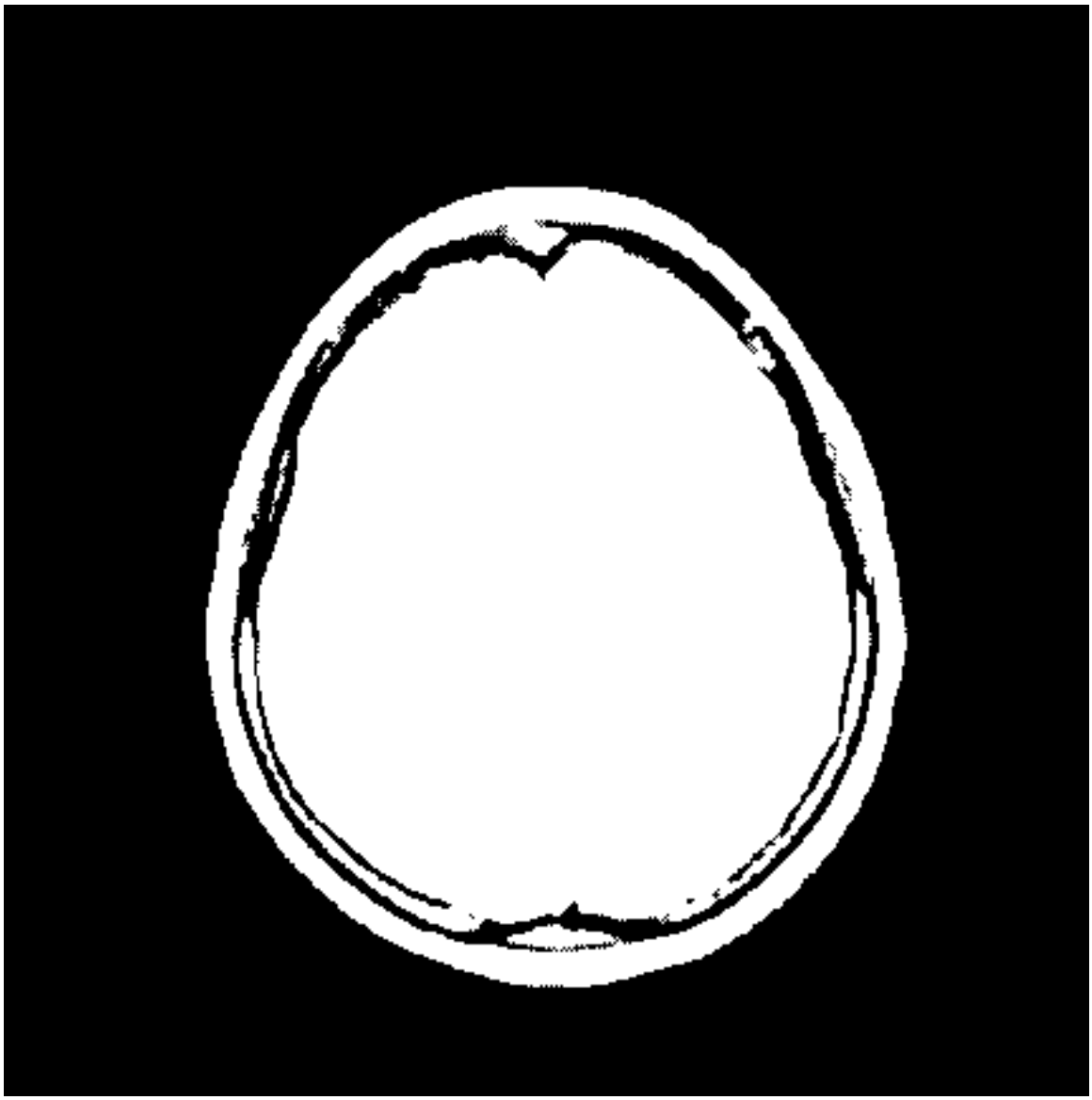

Figure 8: a) Tresholded mask image

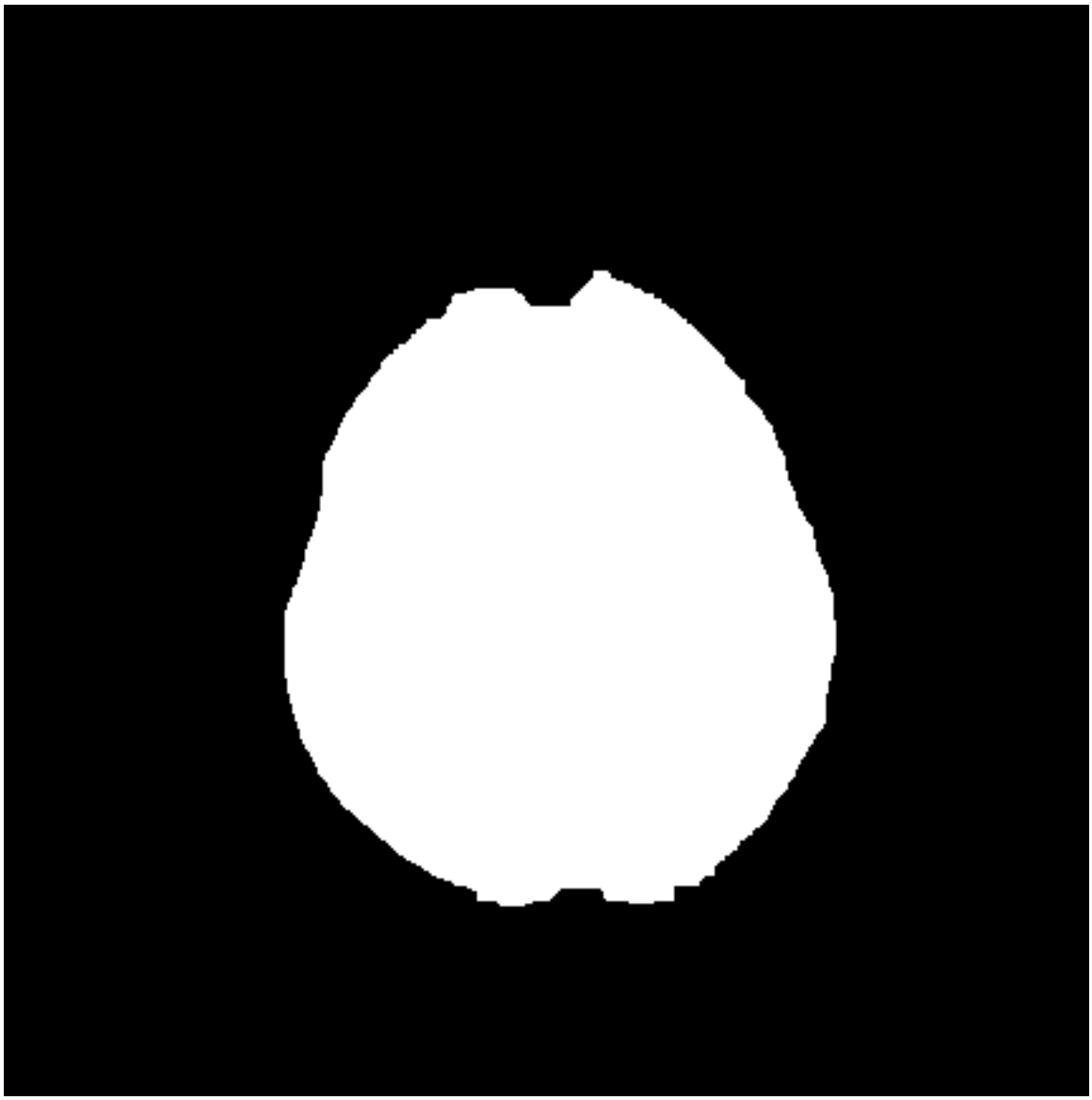

Figure 9: b) Brain mask

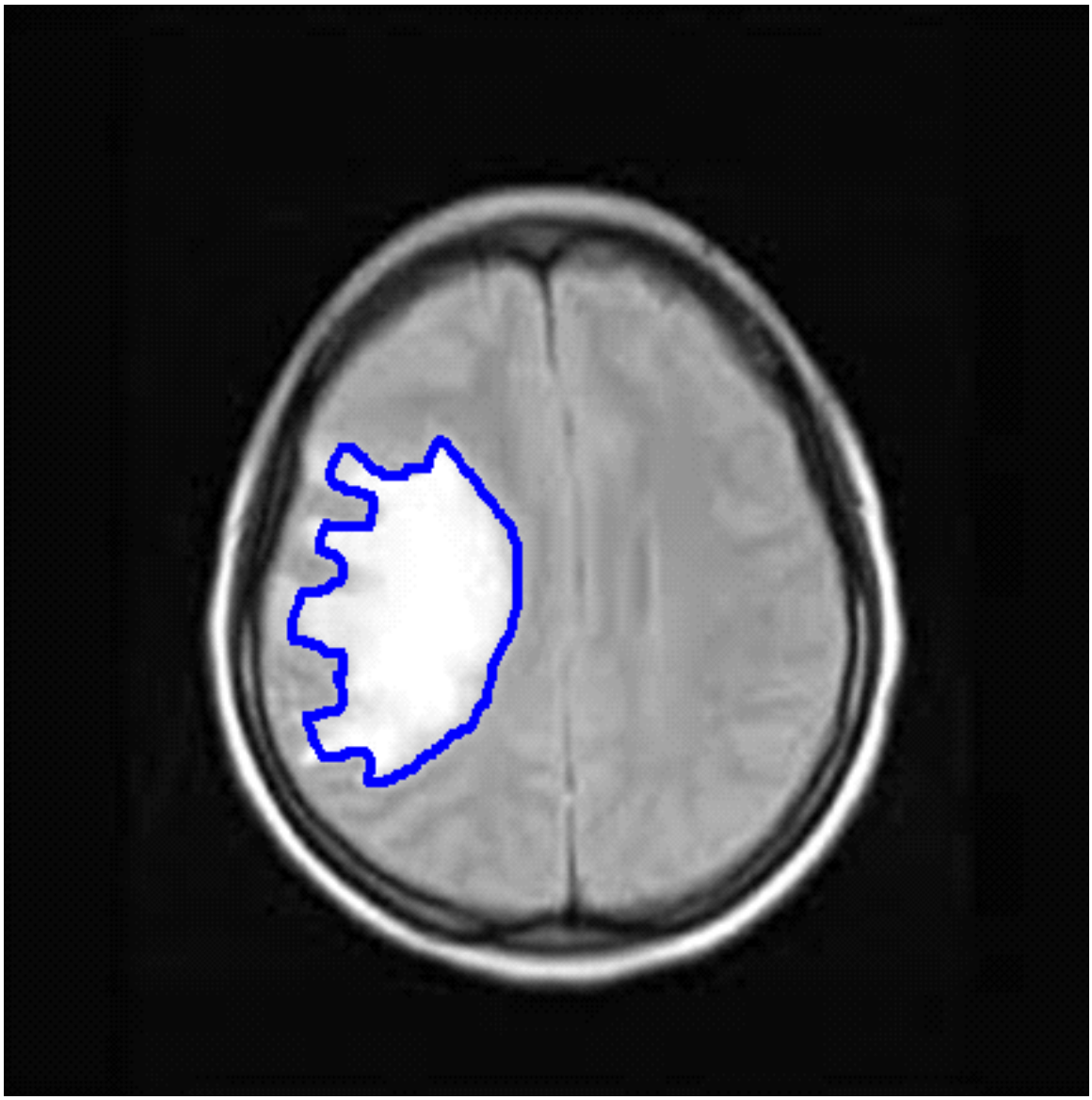

Figure 10: c) Segmentated image

5 Python Code

First i tresholded the image using treshold=50

```
treshold = 50
height, width = self.input_image_arr.shape
tresholded_image = np.copy(self.input_image_arr)
brain_region = np.copy(self.input_image_arr)
for i in range(height):
  for j in range(width):
    if tresholded_image[i,j] > treshold:
      tresholded_image[i,j] = 255
```

```
    else:
      tresholded_image[i,j] = 0
```

Then in order to remove skull parts from the brain i chose erotion operation as morphological operation with 4x4 structuring element.

```
kernel = np.ones((4,4),np.uint8) # Structuring element
tresholded_image = cv2.erode(tresholded_image,kernel,iterations = 5)
```

Then i implemented kMeans algorithm which calculates k different clusters according to image intensities and returns cluster centers and assigned pixels.

```
# This function finds closest cluster point to an intensity
def closest_cluster(self,intensity,clusters):
  counter = len(clusters)
  min = float('inf')
  assigned_cluster = 0
  for i in range(counter):
    difference = abs(clusters[i]-intensity)
    if difference < min:
      min = difference
      assigned_cluster = i

  return assigned_cluster

# Take the mean value of the assigned points for new cluster means
def calculate_new_clusters(self,clusters,assigned_points,intensities):
  new_clusters = np.zeros(clusters.shape).astype('float')
  counter = np.zeros(new_clusters.shape).astype('int')
  for j,val in enumerate(assigned_points):
    new_clusters[val] += intensities[j]
    counter[val] += 1
  new_clusters /= counter
  return new_clusters

def converge(self,old_clusters,clusters):
  return True if np.array_equal(old_clusters,clusters) else False

def kMeans(self,intensities,k):
  clusters = np.random.randint(0,256,size=k,).astype('float')
  old_clusters = np.random.randint(0,256,size=k,).astype('float')
  assigned_points = np.zeros(len(intensities),dtype=int)

  # While clusters are moving to new points
  while not self.converge(clusters,old_clusters):
    for i,val in enumerate(intensities):
      # Assign the point to the closest cluster
      assigned_points[i] = self.closest_cluster(val,clusters)

    old_clusters = np.copy(clusters)
    # Calculate new cluster centers
```

```
    clusters = self.calculate_new_clusters(clusters,assigned_points,
        intensities)

  return clusters,assigned_points
```

According to the cluster intensities from this algorithm i set a treshold for tumor region and make non tumor areas dark.

```
# After findind tumor and other parts only highlight the tumor region
    which has more intensity than other brain parts
treshold = 0
clusters = np.uint8(clusters)
treshold =  clusters[0] if clusters[0] > clusters[1] else clusters[1]
brain_region[ brain_region >= treshold ] = 255
brain_region[ brain_region < treshold ] = 0
```

After this step we have tumor and some irrevelant parts of the brain which is detached from the tumor. In order to remove that parts, i applied a closing operation with 3x3 structuring element.

```
kernel = np.ones((3,3),np.uint8)
brain_region = cv2.erode(brain_region,kernel,iterations = 3)
```

Then we get only tumor. In order to find only the edges of the tumor, first i blurred image with gaussian blur and then i removed the blurred image from the original image. This operation gives me only the edges of the tumor. This technique is part of the unsharp operation. After finding edges of the tumor i paint them to blue and add them to the original image to show the edges of the tumor in real app.

```
# In order to find the edge of the tumor blur the image and remove the
    blurred one from original
smoothed_region = cv2.GaussianBlur(brain_region,(5,5),0.6,0,cv2.
    BORDER_DEFAULT)
brain_region -= smoothed_region
# We get only edge of the tumor

colored_brain = cv2.cvtColor(brain_region, cv2.COLOR_GRAY2RGB)
# Make the tumor edges blue
colored_brain[brain_region > 0] = 255
colored_brain[:,:,1] = 0
colored_brain[:,:,2] = 0
self.input_image_arr = cv2.cvtColor(self.input_image_arr, cv2.
    COLOR_GRAY2RGB)
self.input_image_arr[colored_brain[:,:,0] > 0] = [255,0,0]
```