

**Introduction to Computer Vision**

**Coursework**

**Submission**

**Kiana Hadysadegh**

**230632224**

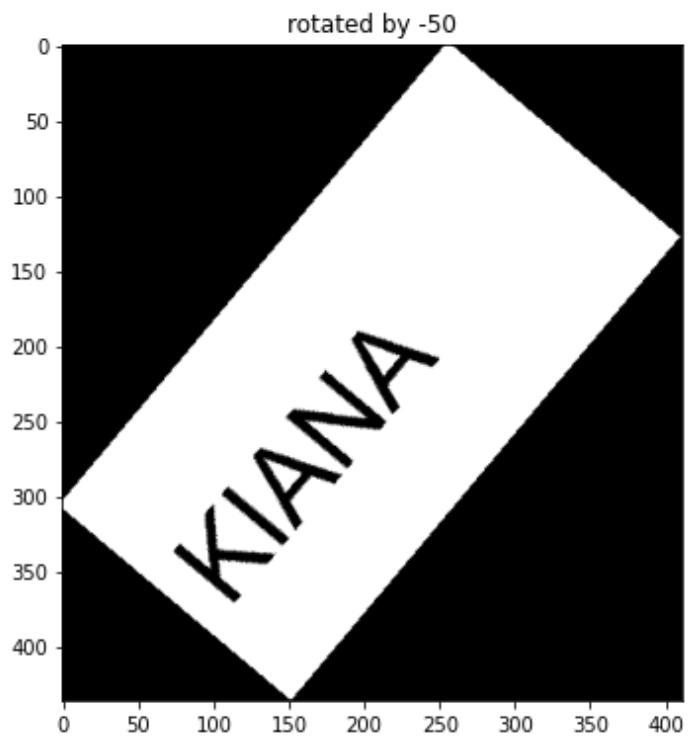
**Transformations**

**Question 1(b):**

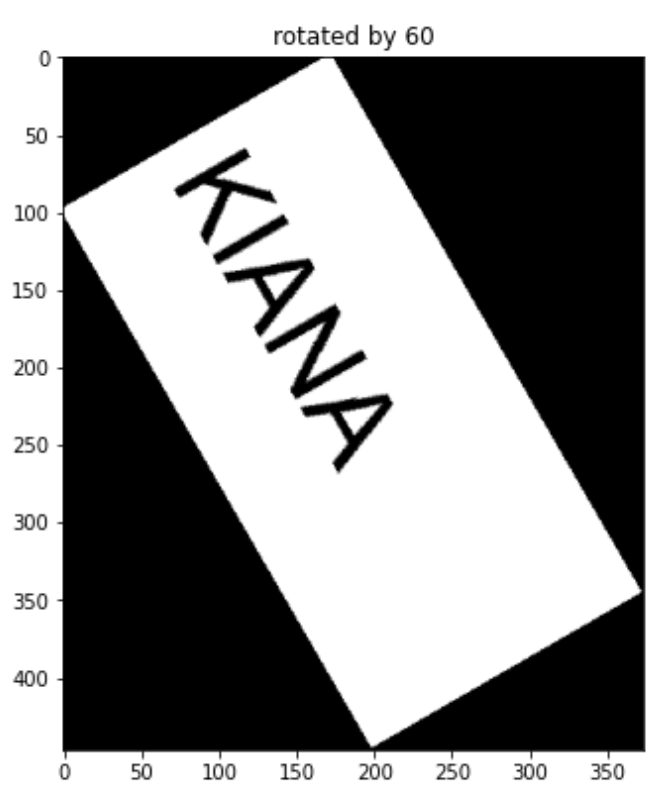


**Rotated images:**

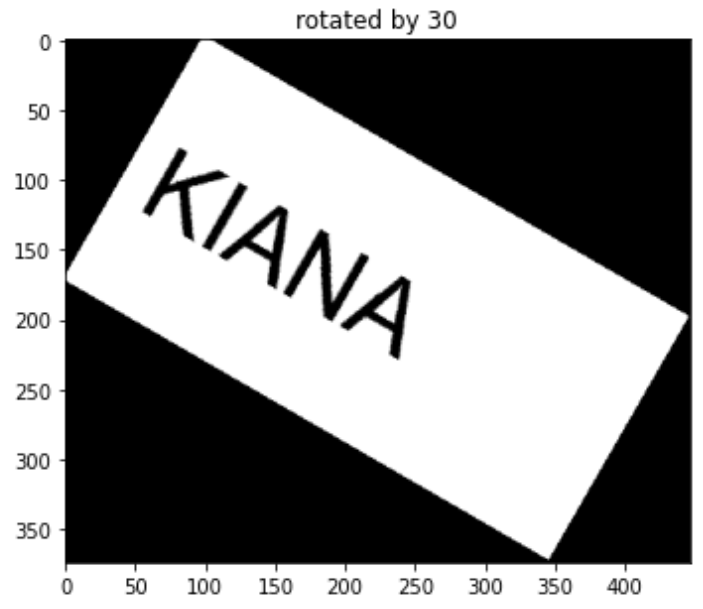
θ = -50 deg



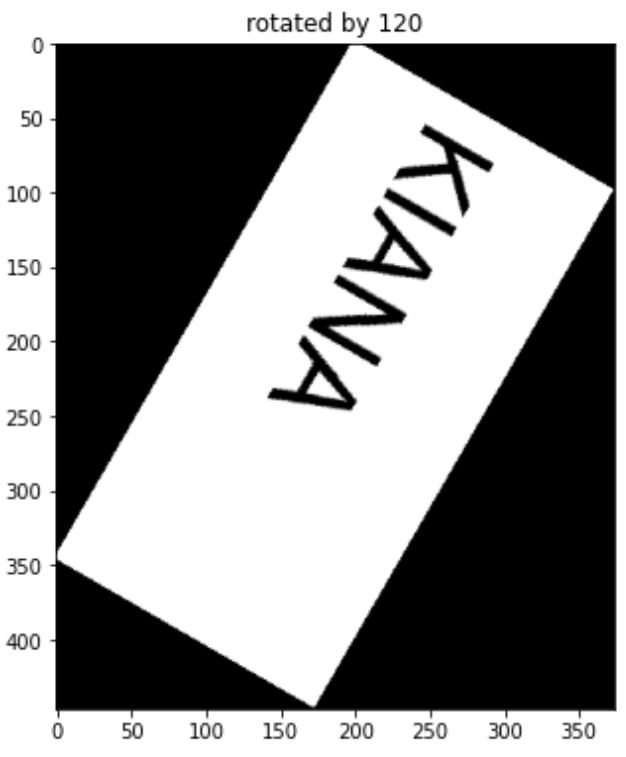
θ = 60 deg



θ = 30

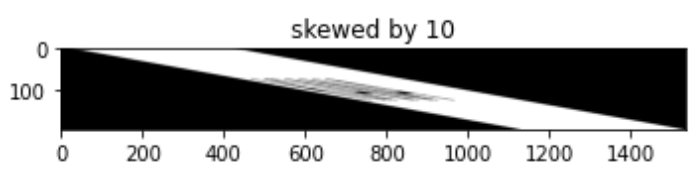


θ = 120 deg

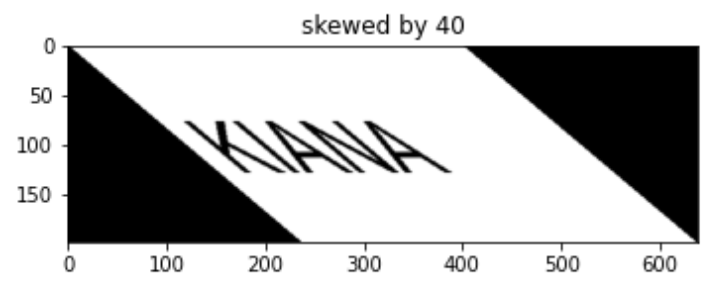


**Skewed images:**

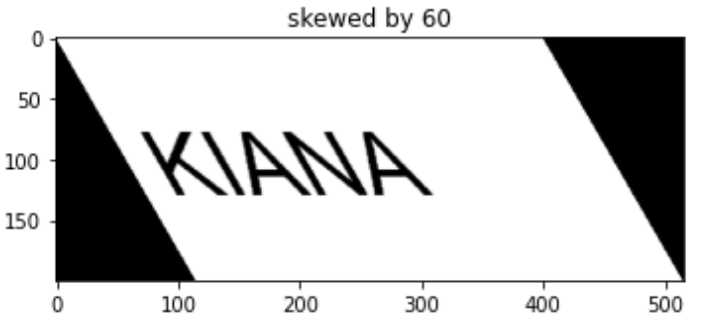
θ = 10 deg



θ = 40 deg



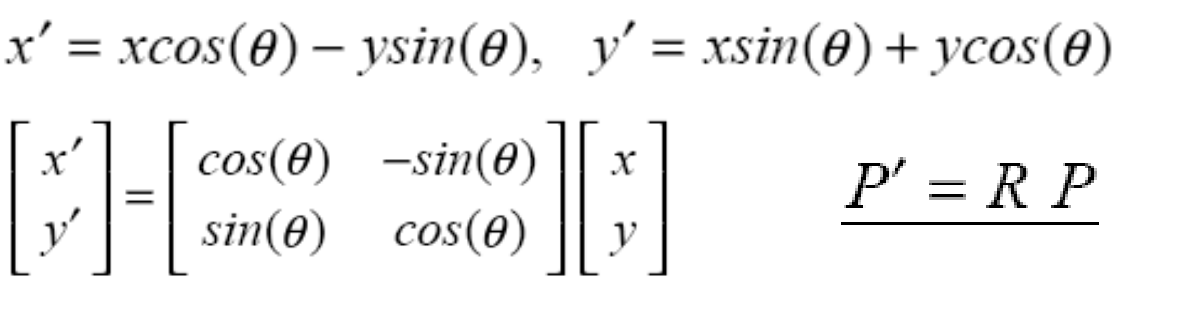
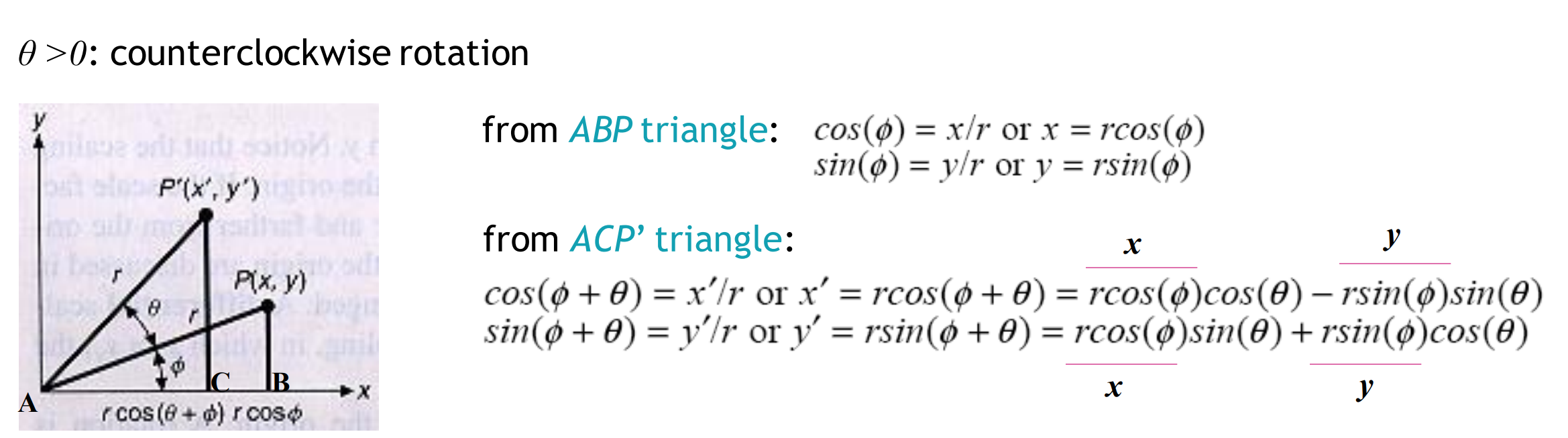
θ = 60 deg



**Your comments:**

At first we open our image as an numpy array

Rotation Formula:



Since in the rotation technique the dimensions of the original image changes, we first calculate the dimensions of the new grid (which basically follows the original formula of rotation):

new\_height = round(abs(y \* cos) + abs(x \* sin))+1

new\_width = round(abs(x \* cos) + abs(y \* sin))+1

then we define our output grid: output = np.zeros((new\_height, new\_width, channels))

Since the only point that its coordinates doesn’t change in rotation is the center, we calculate the center of the both grids. Also to use our formulas we should find all points coordinates with respect to the center point.

If we iterate through original grid and copy its pixels values into the corresponding coordinates in the new grid, we face some black dots in the output. This is because we should use round operator in calculating coordinates for the new grid, So some pixel values are copied in the same point in the new grid, and some coordinates in the new grid missed because of the round operation. As a result we can see black dots in our rotated image which means some spots were missed in the new grid. To avoid this problem we use inverse mapping technique, which iterates through new grid points and find their coordinates and pixel values from the original grid. By this method, none of the pixels of the new grid miss:

We transpose our rotation matrix and multiply it by coordinate of the point in the new grid to find the coordinate of the original point.

original\_x = (int(inverse\_rotation\_matrix[0,0] \* x + inverse\_rotation\_matrix[0,1] \* y))

original\_y = (int(inverse\_rotation\_matrix[1,0] \* x + inverse\_rotation\_matrix[1,1] \* y))

So, we first change the origin of the new grid to its center and calculate the points coordinates with respect to the center.

Then we find the corresponding points coordinates in the original grid using inverse mapping formula. The points in the original grid also were calculated with respect to its center point, so we convert them with respect to the normal origin( array[0,0] ).

In the end we check if our calculated coordinate is in the grid or not since indexes of the points in the borders become out of bound. After that we copy their value into our new grid.

For horiziontal skewness: (shear along x-axis)



new\_x = round(x + (y/tan(theta))

new\_y = y

Since it is horizontal skewness only x-values (width) of the points changes. So we create a new grid for output with the same height and new width which is calculated based on the skewness formula:

new\_width = round(abs(old\_width)+abs(old\_height/tan))+1

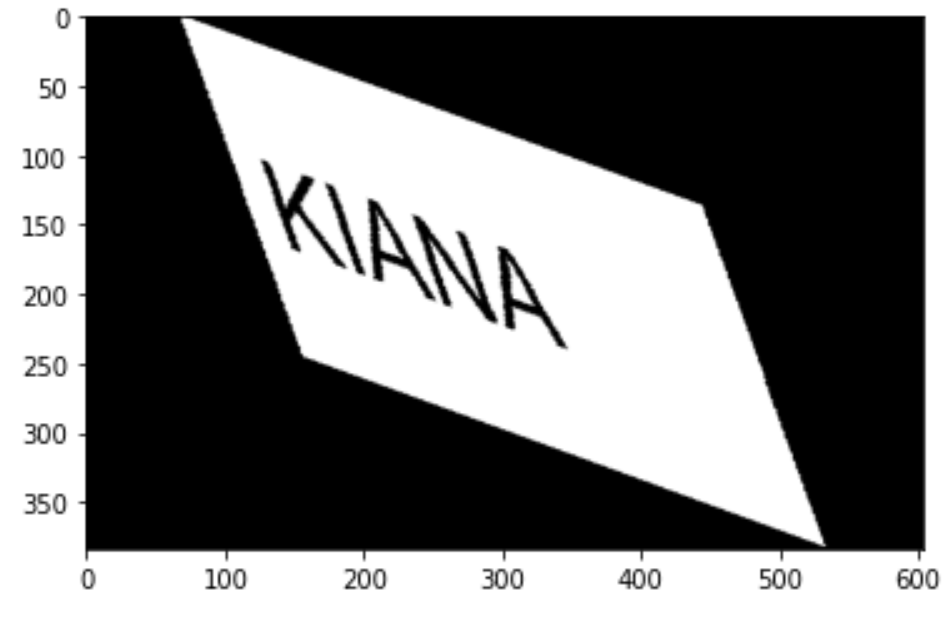
we iterate through the image and calculate the coordinates in the new grid using the above formula. In the end we copy the pixel value from original image into the output grid.

SHOULD I USE INVERSE METHOD HERE ALSO? SHOULD I CHANGE THE COUNTER CLOCKWISE SKEW?

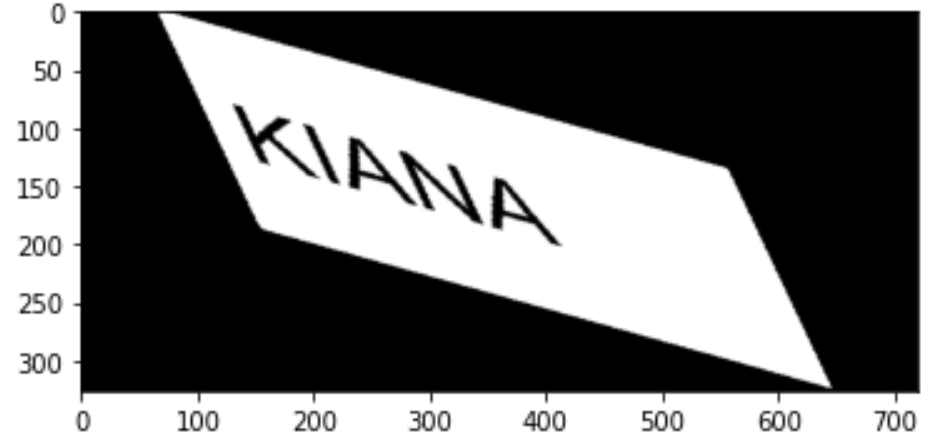
**Discuss in the report the advantages and disadvantages of different approaches.**

**Question 1(c):**

θ2=50 and θ1=20 clockwise



θ1=20 clockwise and θ2=50



**Your comments:**

**Analyse the results when you change the order of the two operators: R(S(I)) and S(R(I)).**

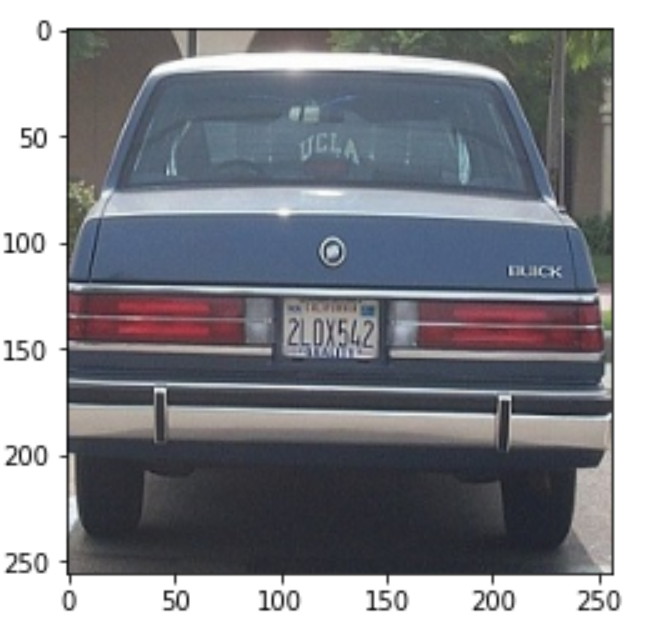
**Are the results of (i) and (ii) the same? Why?**

**CHECK COMMENTS!!!!!**

**Convolution**

**Question 2(b)**:

**Designed kernel:**



**Your comments:**

Convolution:

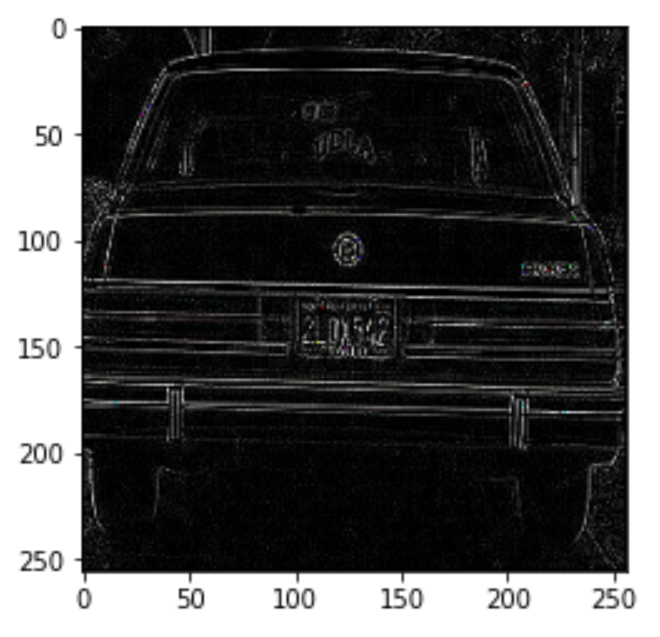
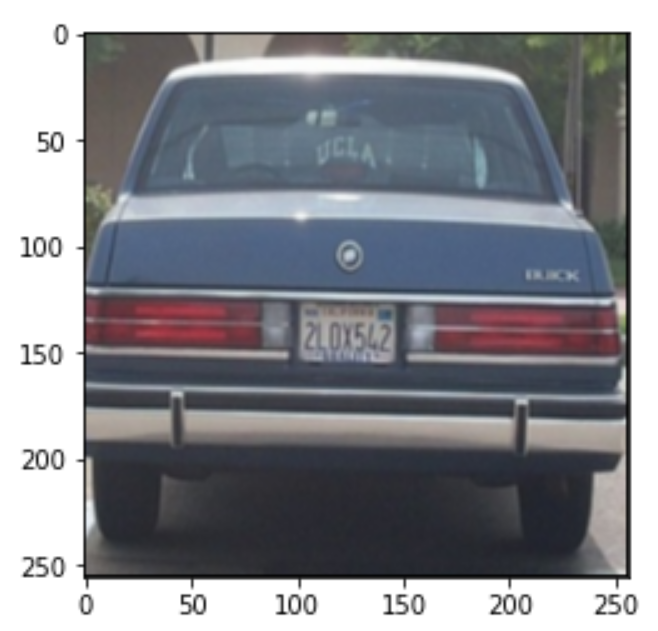
To calculate of the convolution of the given image and given kernel, we place the kernel on each pixel of the image so that the center element of the kernel is on that pixel. Then we perform element-wise multiplication between pixels of the kernel and pixels of the image which the kernel is on them. After that we calculate the summation of all multiplied elements and divide it by sum of the values in the kernel. We repeat this process for all pixels of the image and save their convolution value in another grid (with the same size as image) and return it. If the sum of products value is negative we assume it is zero and if the sum of the elements of the kernel is 0 or negative we do not perform the division to handel negative values and divide by zero problem.

The problem is that pixels from the border of the image up to kernel\_size/2 from it do not participate in the convolution operation and this is the border problem. To solve this problem I use the zero padding method. I create a new grid which its dimensions is kernel\_size bigger than the old grid and fill it with zeros. Then I copy the image in the center of the new grid in a way that the old border is in the kernel\_size/2 distance from the new grid border. By this way all of the pixels in the border to kernel\_size/2 from it also participate in the convolution operation. In the code I iterate through output grid and for each point I calculate the range of the image in the new grid that should participate in the convolution and calculate the convolution of it with the kernel and fill the point in the output grid with calculated convolution value.

**CHECK COMMENTS!!!!!**

**Design a convolution kernel that computes, for each pixel, the average intensity value in a 3x3 region.**

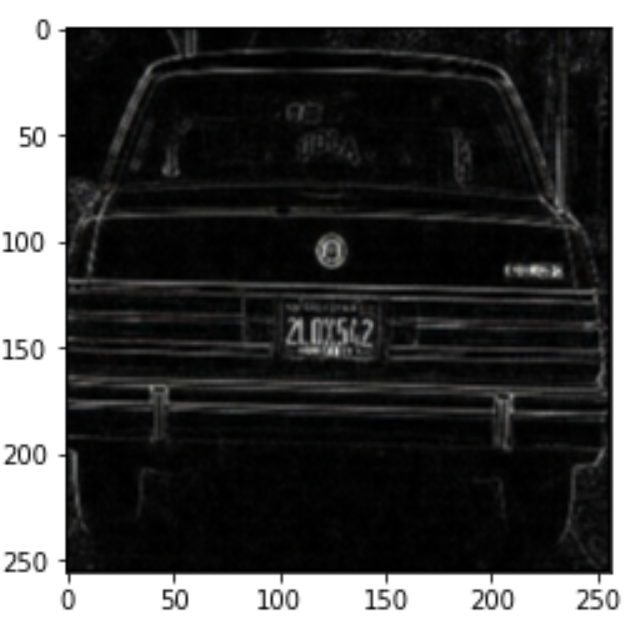
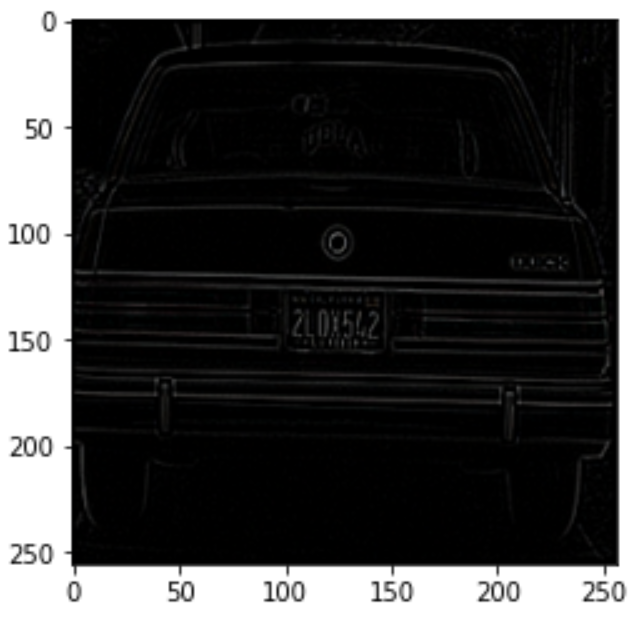
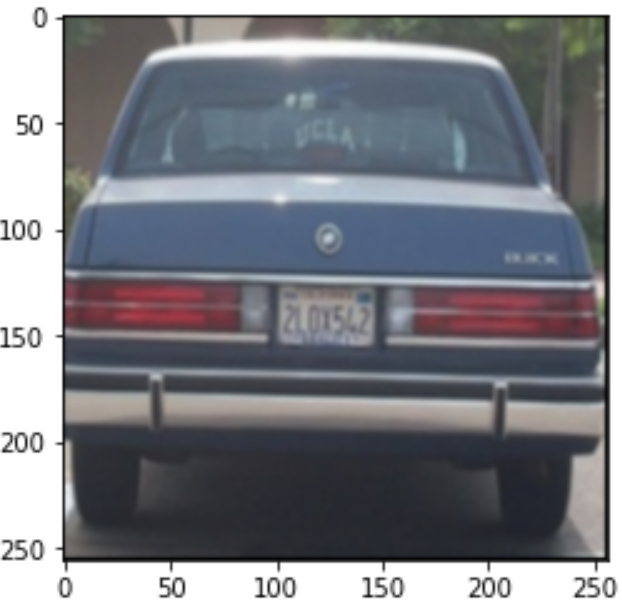
**Question 2(c):**



**Your comments:**

**Comment on the effect of each kernel.**

**Question 2(d):**



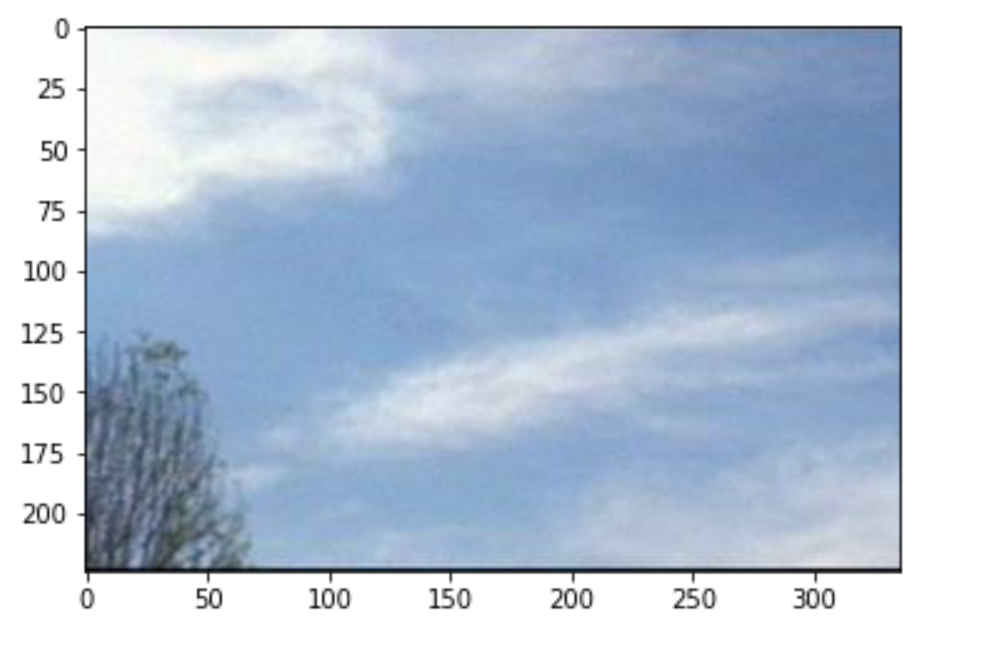
**Your comments:**

**Comment the results.**

**Histograms**

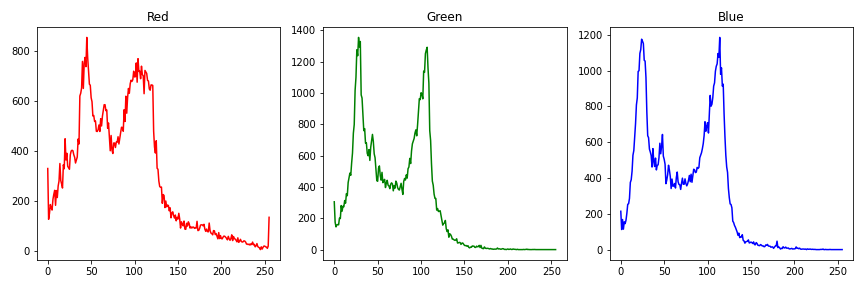
**Question 3(a):**

**Two non-consecutive frames:**

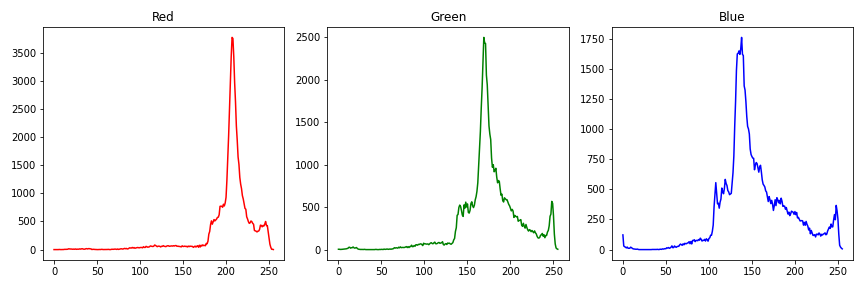


**Corresponding colour histograms:**

Histogram 9



Histogram 0

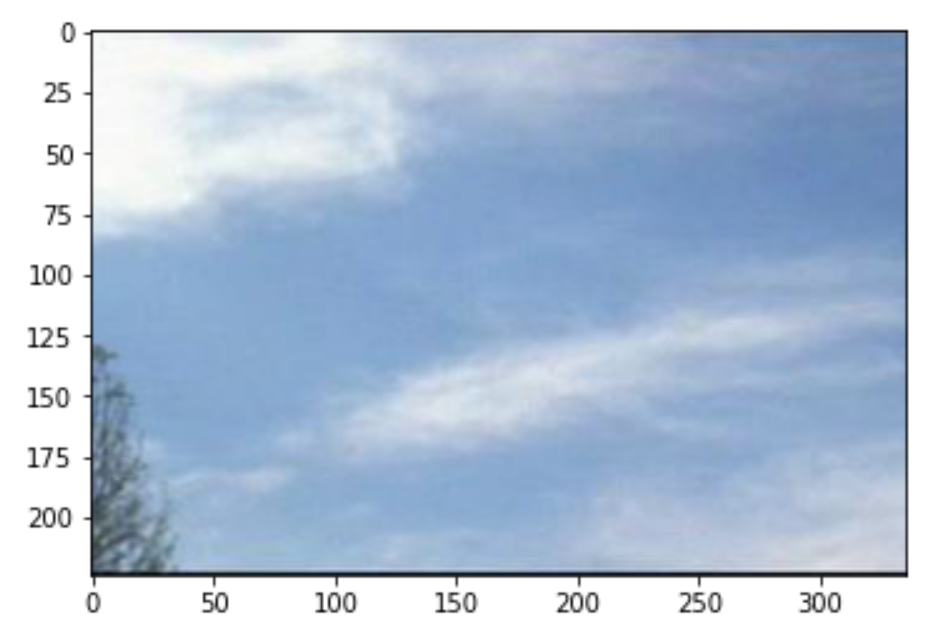


**Your comments:**

**Question 3(b):**

**Two consecutive frames:**

I8

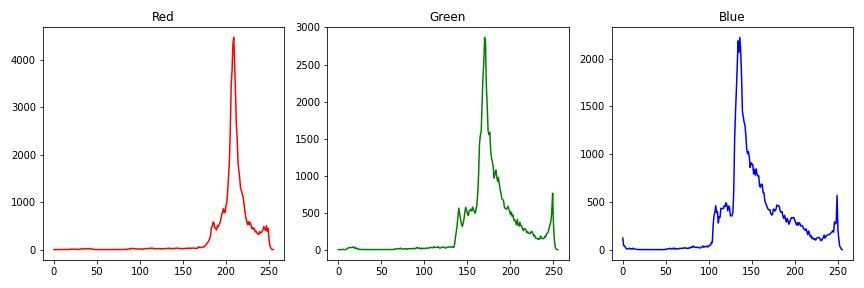


I9

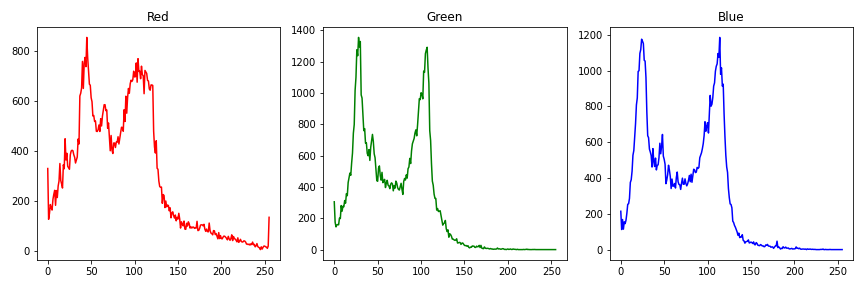


**Histograms:**

Histogram of I8



Histogram of I9

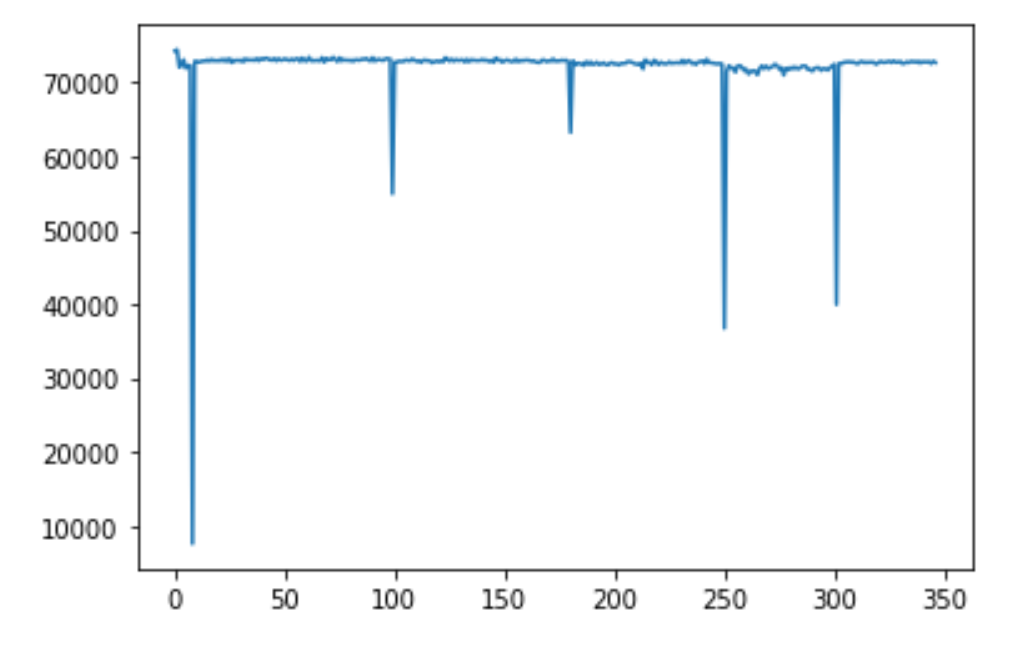
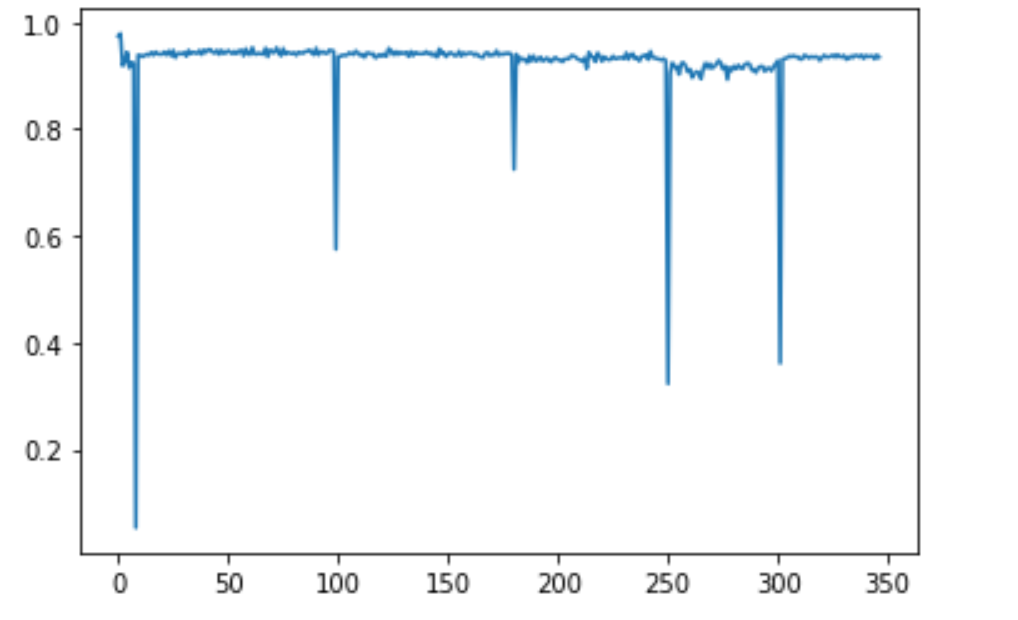


Intersection result

7562.0

**Question 3(b):**

**Intersection result for a video sequence:**



**Your Comments:**

**Does that change the results? Plot the intersection values over time and the normalised intersection values, and save the corresponding figures. Show and comment the figures in the report.**

**Question 3(c):**

**Comments:**

**Discuss in the report the following: What does the intersection value represent for a given input video? Can you use it to make a decision about scene changes? How robust to changes in the video is the histogram intersection? When does it fail?**

**Texture Descriptors and Classification**

**Question 4(a)**

**Three non-consecutive windows**

W1

W3

W2

**LBP of windows**

LBP1

LBP3

LBP2

**Histograms of LBPs**

H1

H3

H2

**Question 4(b)**

**Two example images:**

Face image

Car image

Face descriptor

Car descriptor

**Descriptors:**

**Your comments:**

**Question 4(b)**

**Block diagram of classification process**

**Your comments:**

**Question 4(c)**

**Your comments:**

**Question 4(d)**

**Your comments:**

**Question 4(e)**

**Your comments:**

**Object Segmentation and Counting**

**Question 5(a)**

**Original frames:**

Reference frame

Selected frame 1

Selected frame 2

**Frame differencing:**

**Threshold results:**

**Question 5(b)**

**Original frame:**

Selected frame 1

Selected frame 2

**Frame differencing:**

**Threshold results:**

**Your comments for 5a,5b:**

**Question 5(c)**

Generated background

**Your comments:**

**Question 5(d)**

Bar plot

**Your comments:**