CS-512 Computer Vision Spring 2023

Assignment-2

Sai Manohar Vemuri - A20514848

1.

Problem Statement:

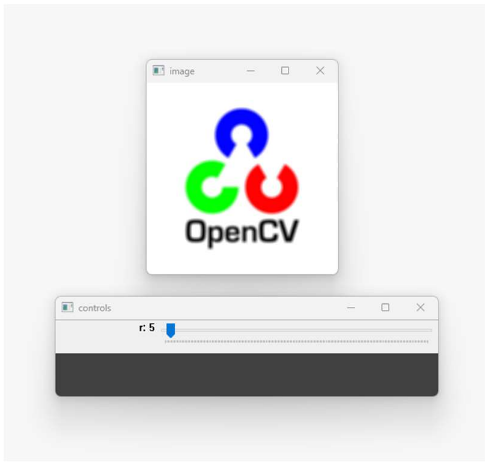
Smooth an image using the OpenCV filter2D function and show the resulting image. Use a trackbar to control the amount of smoothing.

Proposed Solution and Implementation:

Used OpenCV filter2D function to smooth the image with the kernel size taken from the track bar.

Implemented trackbar to control the kernel size of the filter.

Results:



2.

Problem Statement:

Repeat the previous step using your own implementation of a convolution function with a suitable filter. In this question (only) you must implement the convolution operation yourself. Make sure to use Cython to speed up your implementation.

Proposed Solution and Implementation:

We used the trackbar to control the kernel size for the smoothing operation.

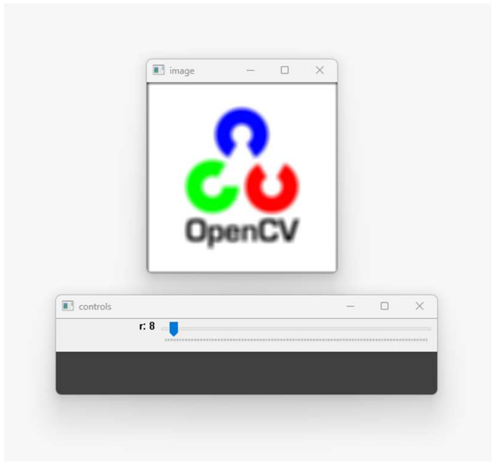
We used 3 loops to iterate through 3 channel image and applied the convolution operation.

Without using the OpenCV filter2D function we implemented the convolution operation function from scratch.

We padded the image in order to overcome the down sampling after the convolution operation.

We implemented the same code in Cython to reduce the computation time.

Results:



3.

Problem Statement:

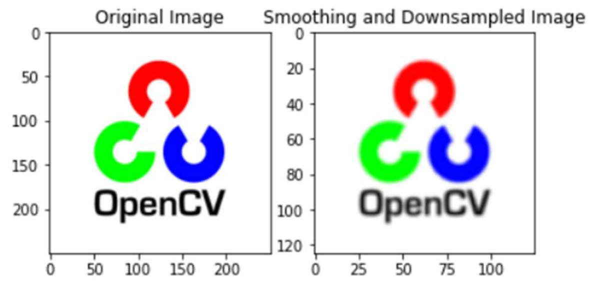
Down-sample the image by a factor of 2 with smoothing (before down-sampling) and show it.

Proposed Solution and Implementation:

We used OpenCV filter2D function to smoothen the image.

We divided the smoothened image by 2 to down sample the image by the factor of 2.

Results:



```
Original Image Shape: (250, 250, 3)
Downsampled Image Shape: (125, 125, 3)
```

4.

Problem Statement:

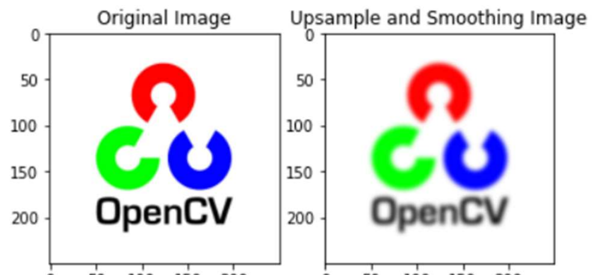
Up-sample the image from the previous step by a factor of 2 with smoothing (after up-sampling). Show the resulting image and the difference from the original image. When showing the difference, normalize difference values if necessary, so that they are visible.

Proposed Solution and Implementation:

We first up sampled the image by the factor of 2 by multiplying the array with 2.

Then we smoothened the image using the OpenCV filter2D.

Result:



```
Original Image Shape: (250, 250, 3)
Upsampled Image Shape: (250, 250, 3)
```

5.

Problem Statement:

Compute the x- and y-derivatives of an image and display the resulting images. When displaying the derivative images, normalize the obtained values if necessary, so that derivative values are visible.

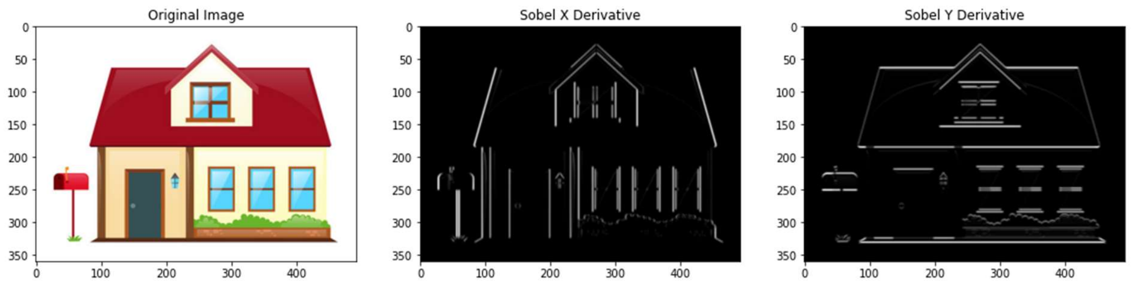
Proposed Solution and Implementation:

We first converted the image to grayscale to reduce the computation time although we can find the gradients using sobelx and sobely with 3 channel images.

Then using cv2.Sobel filter from OpenCV library we computed Sobel X and Y derivatives.

Then we plotted both X and Y derivatives.

Results:



6.

Problem Statement:

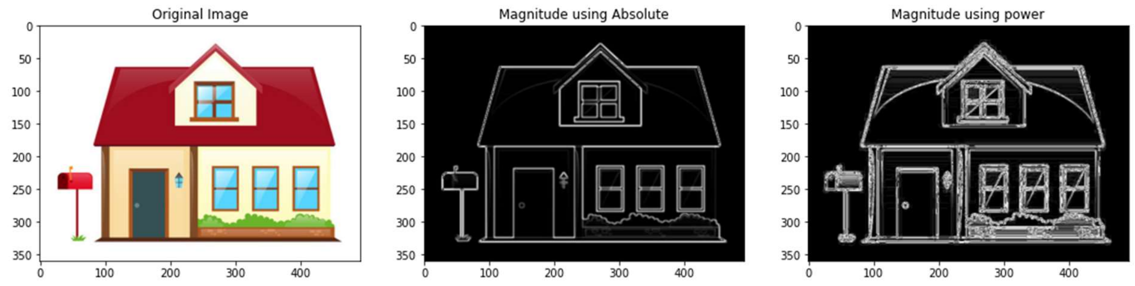
Compute the magnitude of the image gradient you computed in the previous step, and display it. Normalize the displayed values as needed.

Proposed Solution and Implementation:

We computed the magnitude of the image gradients by using Sobel X and y derivates we found in the previous question. And applied the following formula: np.sqrt(sobelX**2 + sobelY**2)

Then we normalized the array using the MinMax Normalization.

Results:



7.

Problem Statement:

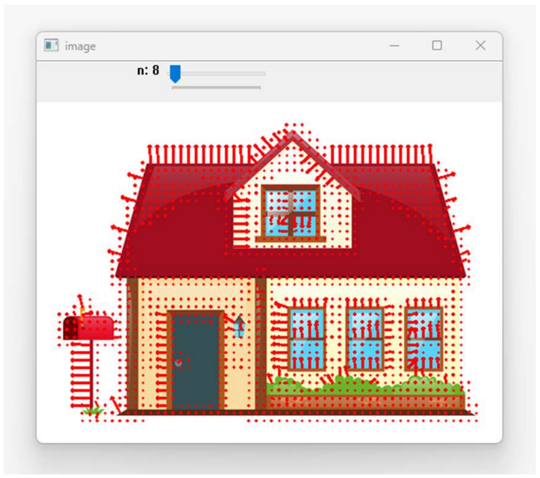
Plot the image gradient vectors on top of the original image every N pixel (e.g. using short red lines segments of length K) and display the resulting image. Use a track bar to control N.

Proposed Solution and Implementation:

Use Sobel X and Y derivatives to calculate the magnitude

Use trackbar to control N value which plots the gradient vectors on the image for every N pixels using the red lines with the length k.

Results:



8.

Problem Statement:

Detect and display corners in an image using the OpenCV Harris corner detection function (cornerHarris). Show the results using red dots. Use a track bar to control the number of control points shown.

Proposed Solution and Implementation:

Converted the original image into grayscale and applied the corner harris method on this image.

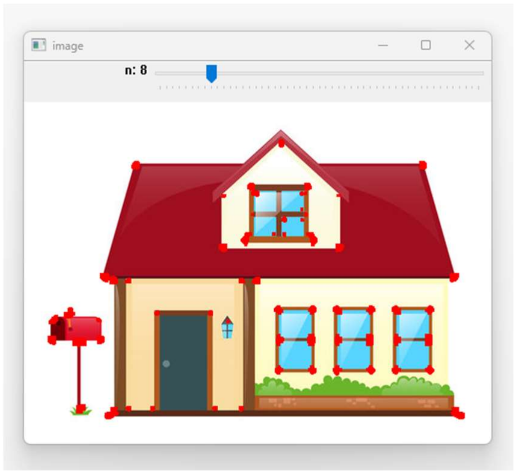
Used the trackbar to control the n value.

We used cv2.cornerHarris function to find the corners.

Used the 0.1 threshold to detect the corners.

Plotted the corners using the red points on the original image.

Results:



9.

Problem Statement:

Scan the image with a window. At each window location, compute the correlation matrix and its eigen values. Multiply the eigenvalues you computed to get a "cornerness" measure. Display the "cornerness" measure making sure to normalize values as needed so that the measure.

Proposed Solution and Implementation:

We first computed X and Y derivates with Sobel Filter.

We iterated the window of size 7 through the image to computed Correlation matrix and eigenvalues.

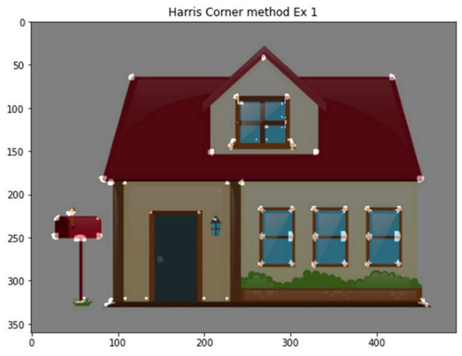
From these values we computed the corner values using the following formula:

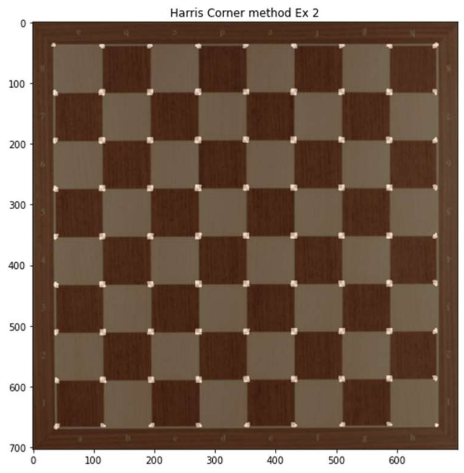
Formula: eigenvector.prod()-k*eigenvector.sum()**2

Then we normalized the corner values and plotted the corners on the original image using the cv2. addWeighted function.

We tested this on two different images.

Results:





10.

Problem Statement:

Train a simple convolutional network (code provided) to perform MNIST digit classification and report the accuracy you receive on a validation set. Smooth the images (thus degrading them), train the network, evaluate results, and report accuracy. Repeat the smoothing several times (thus increasingly degrading the images) and report the accuracy.

Proposed Solution and Implementation:

Here we first smoothened the image using the cv2. GaussianBlur function from the OpenCV library.

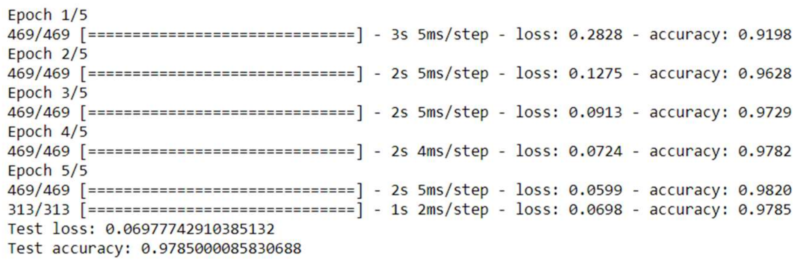
And then we preprocessed the dataset.

We used simple neural network architecture attached as a reference to the assignment.

We just used the one Dense layer with 784(28*28 image size) nodes and at the output layer we used 10 nodes to classify the image into numbers ranging from 0-9.

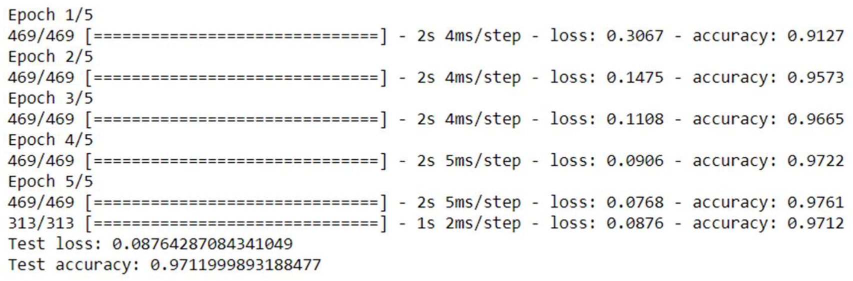
Results:

Model trained with image smoothing once:



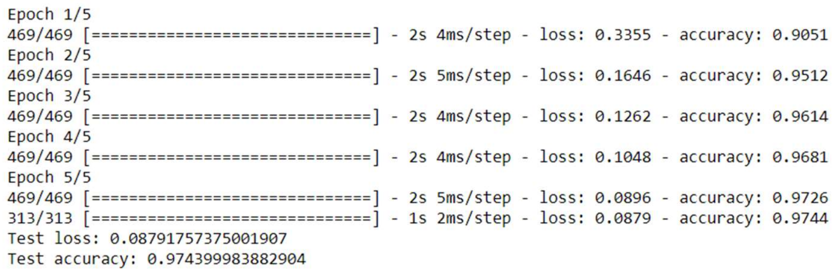
```
Epoch 1/5
469/469 [==============================] - 3s 5ms/step - loss: 0.2828 - accuracy: 0.9198
Epoch 2/5
469/469 [==============================] - 2s 5ms/step - loss: 0.1275 - accuracy: 0.9628
Epoch 3/5
469/469 [==============================] - 2s 5ms/step - loss: 0.0913 - accuracy: 0.9729
Epoch 4/5
469/469 [==============================] - 2s 4ms/step - loss: 0.0724 - accuracy: 0.9782
Epoch 5/5
469/469 [==============================] - 2s 5ms/step - loss: 0.0599 - accuracy: 0.9820
313/313 [==============================] - 1s 2ms/step - loss: 0.0698 - accuracy: 0.9785
Test loss: 0.06977742910385132
Test accuracy: 0.9785000085830688
```

Model trained with image smoothing twice:



```
Epoch 1/5
469/469 [==============================] - 2s 4ms/step - loss: 0.3067 - accuracy: 0.9127
Epoch 2/5
469/469 [==============================] - 2s 4ms/step - loss: 0.1475 - accuracy: 0.9573
Epoch 3/5
469/469 [==============================] - 2s 4ms/step - loss: 0.1108 - accuracy: 0.9665
Epoch 4/5
469/469 [==============================] - 2s 5ms/step - loss: 0.0906 - accuracy: 0.9722
Epoch 5/5
469/469 [==============================] - 2s 5ms/step - loss: 0.0768 - accuracy: 0.9761
313/313 [==============================] - 1s 2ms/step - loss: 0.0876 - accuracy: 0.9712
Test loss: 0.08764287084341049
Test accuracy: 0.9711999893188477
```

Model trained with image smoothing thrice:



```
Epoch 1/5
469/469 [==============================] - 2s 4ms/step - loss: 0.3355 - accuracy: 0.9051
Epoch 2/5
469/469 [==============================] - 2s 5ms/step - loss: 0.1646 - accuracy: 0.9512
Epoch 3/5
469/469 [==============================] - 2s 4ms/step - loss: 0.1262 - accuracy: 0.9614
Epoch 4/5
469/469 [==============================] - 2s 4ms/step - loss: 0.1048 - accuracy: 0.9681
Epoch 5/5
469/469 [==============================] - 2s 5ms/step - loss: 0.0896 - accuracy: 0.9726
313/313 [==============================] - 1s 2ms/step - loss: 0.0879 - accuracy: 0.9744
Test loss: 0.08791757375001907
Test accuracy: 0.974399983882904
```

	Test Accuracy
One time smoothing	97.84%
Two times smoothing	97.11%
Three times smoothing	97.43%

11.

<u>Problem Statement:</u>

Repeat the previous step, except that now you compute the magnitude of image gradients before feeding them into the network. Compare the results in this step to the results from the previous step.

<u>Proposed Solution and Implementation:</u>

Here we first smoothened the image using the cv2. GaussianBlur function from the OpenCV library.

And computed the magnitudes of the image gradients using the sobel filter and then fed the data to the neural network.

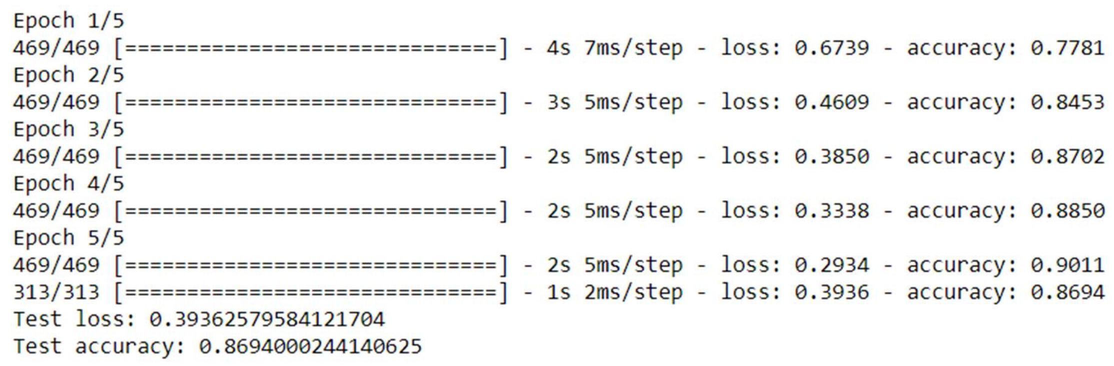
And then we preprocessed the dataset.

We used simple neural network architecture attached as a reference to the assignment.

We just used the one Dense layer with 784(28*28 image size) nodes and at the output layer we used 10 nodes to classify the image into numbers ranging from 0-9.

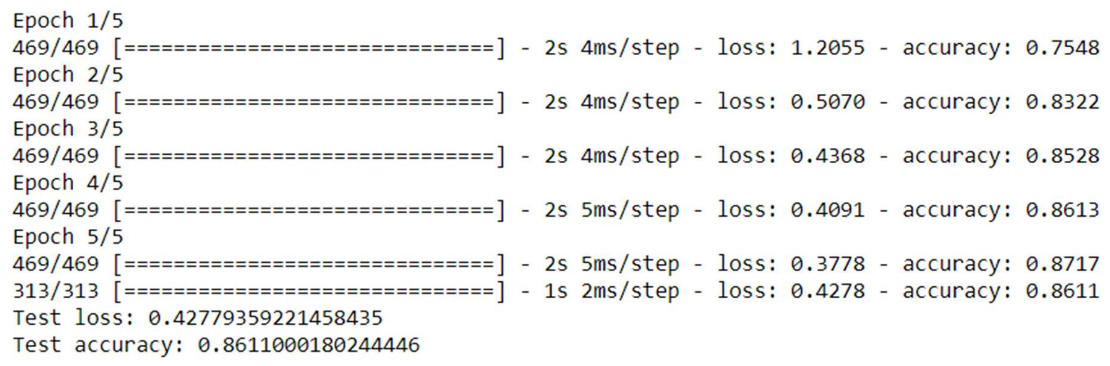
<u>Results:</u>

Model trained with image smoothing once:



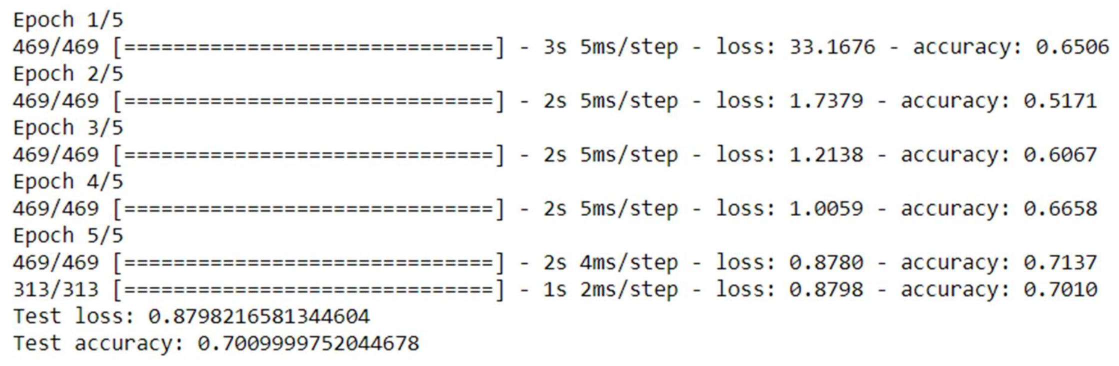
```
Epoch 1/5
469/469 [==============================] - 4s 7ms/step - loss: 0.6739 - accuracy: 0.7781
Epoch 2/5
469/469 [==============================] - 3s 5ms/step - loss: 0.4609 - accuracy: 0.8453
Epoch 3/5
469/469 [==============================] - 2s 5ms/step - loss: 0.3850 - accuracy: 0.8702
Epoch 4/5
469/469 [==============================] - 2s 5ms/step - loss: 0.3338 - accuracy: 0.8850
Epoch 5/5
469/469 [==============================] - 2s 5ms/step - loss: 0.2934 - accuracy: 0.9011
313/313 [==============================] - 1s 2ms/step - loss: 0.3936 - accuracy: 0.8694
Test loss: 0.39362579584121704
Test accuracy: 0.8694000244140625
```

Model trained with image smoothing twice:



```
Epoch 1/5
469/469 [==============================] - 2s 4ms/step - loss: 1.2055 - accuracy: 0.7548
Epoch 2/5
469/469 [==============================] - 2s 4ms/step - loss: 0.5070 - accuracy: 0.8322
Epoch 3/5
469/469 [==============================] - 2s 4ms/step - loss: 0.4368 - accuracy: 0.8528
Epoch 4/5
469/469 [==============================] - 2s 5ms/step - loss: 0.4091 - accuracy: 0.8613
Epoch 5/5
469/469 [==============================] - 2s 5ms/step - loss: 0.3778 - accuracy: 0.8717
313/313 [==============================] - 1s 2ms/step - loss: 0.4278 - accuracy: 0.8611
Test loss: 0.42779359221458435
Test accuracy: 0.8611000180244446
```

Model trained with image smoothing thrice:



```
Epoch 1/5
469/469 [==============================] - 3s 5ms/step - loss: 33.1676 - accuracy: 0.6506
Epoch 2/5
469/469 [==============================] - 2s 5ms/step - loss: 1.7379 - accuracy: 0.5171
Epoch 3/5
469/469 [==============================] - 2s 5ms/step - loss: 1.2138 - accuracy: 0.6067
Epoch 4/5
469/469 [==============================] - 2s 5ms/step - loss: 1.0059 - accuracy: 0.6658
Epoch 5/5
469/469 [==============================] - 2s 4ms/step - loss: 0.8780 - accuracy: 0.7137
313/313 [==============================] - 1s 2ms/step - loss: 0.8798 - accuracy: 0.7010
Test loss: 0.8798216581344604
Test accuracy: 0.7009999752044678
```

	Test Accuracy
One time smoothing	86.94%
Two times smoothing	86.11%
Three times smoothing	70.00%

After finding the magnitude of Image gradients and feeding it to a neural network the performance is degraded when compared to smoothed images.

References:

https://numpy.org/doc/1.24/user/index.html#user

https://docs.opencv.org/4.x/d2/d96/tutorial_py_table_of_contents_imgproc.html

https://docs.google.com/document/d/1_3fhHXzX4ngyj-UaXy6CEX8yteLYcBhFg3fFfZ0TnEE/edit