Computer Vision
Assignment-1
Abu Shahid
B20CS003

Link to colab file: click here

- Implementation can be found in the .ipynb file attached

1. Spot the difference
    1. Implementation can be found in the colab file
    2. Algorithm

```
let img1, img2
gray1 = gray( img1)
gray2 =  gray( img2)
diff = absolute_difference( gray1, gray2)
threshold -> the 'diff' is thresholded at pixel value= 25.
#This makes all the similar regions black and the difference ican be
seen as white silhouette in the thresholded image.
 contour -> the active regions in 'threshold' are then used to draw
contours
 result -> contour image is then superimposed on the original image to
get the difference
```

c. **Original Image**

Threshold of difference

Result

d. **Limitations**

- The algorithm works by calculating pixel-pixel difference. So if the image is shifted, then then difference will be shown in the entire image.
- For the same reason, the algorithm is very sensitive to noise. Anything that is not an exact copy will be shown as difference.
- Minor differences may get lost while thresholding.

Question 2- Distance in images-1

a. Implementation can be found in the .ipynb file attached

- The algorithm calculates mid-point of the bounding box given by the OCR.
- These coordinates are then used to calculate the distance.
- Each state is given a numerical code.

```
enter first code10
enter second code3
pixel distance is 67.60177512462228
```

b. Limitations

- The way algorithm calculates distance is by calculating the mid-points of the co-ordinates of the bounding box of state-names. Therefore, algorithm is sensitive to font of text and orientation chosen.

Question 3- Distance in images-2

- Implementation can be found in `.ipynb` and `P3.py` file attached.
- Algorithm works by calculating the thickness of the perimeter and number of pixels in the perimeter. This is used to then give the radius and then the perimeter, area.
- Thickness of perimeter is calculated by counting number of black pixels along the diameter.

```
circumference: 1523.0
area: 184582.25299750047
```

Question 4- Towards reading time

a. Implementation can be found in the .ipynb file attached
b. Algorithm

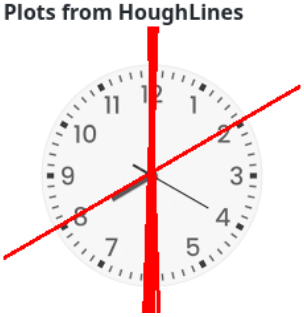
```
let clock_img
let clean_img = fine_line_purge( clock_img, thickness)
# fine_line_purge() erases all the lines whose thickness is less
# than 'thickness' pixels. This is do so as to remove the
# second's needle

let clk_gray = gray(clean_img)
binary -> clk_gray image is thresholded at pixel value 128
edge -> edge detection is applied on the binary image
let lines = HoughLines( edge, votes=45)
for all thetas in lines:
        thetas are clustered in two groups using Kmeans clustering
 angle -> tan inverse of difference of thetas
```

Original clock image

Image after fine_line_purge()

Plots from HoughLines



Result

```
print("Means of each cluster:", means)
```

```
Means of each cluster: [1.7320509241325899, 0.006984163144448549]
```

```
import math
diff= abs( means[0]- means[1])

print(f'angle between the hands is {math.degrees(math.atan(diff)) }')
```

```
angle between the hands is 59.89965755030821
```

c. Limitations

- Algorithm will fail if the design of the clock has lines in it.
- It also fails if the thickness of second's hand is comparable to other hands in the clock.
- If the color of hands and the clock background is very similar, threshold can cause problems.

Question 5- Fun with Landmarks

NOTE: Implementation in `.ipynb` file uses grayscale images Places chosen:

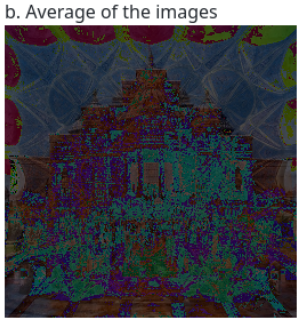
Aali-qapu

Alberquerque temple

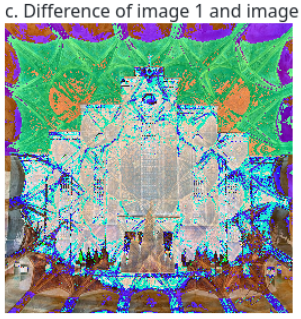
Aksharadham

a) Images were aptly resized

b. Average of the images



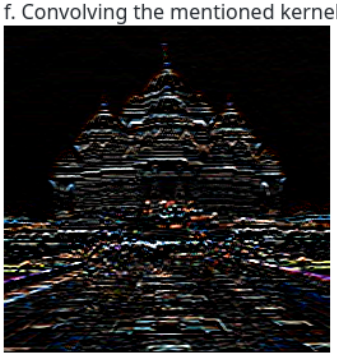
c. Difference of image 1 and image 2



d. Adding salt and pepper to image 1 with 5% probability

e. Various methods of denoising was implemented. Please check the attached `.ipynb` file

f. Convolving the mentioned kernel with image 3.



Question 6- Digit recognition

- Approach

```
image0_files = list of 0 label images
image1_files = list of 1 label images
data = [] #corresponds to our dataset
for all image0_files, image1_files:
        hppf= horizontal_projection_profile_feature( img)
        data.append( [ hppf, img.label ])

train, val, test = data.split
clf1 = SVC( train, kernel= 'linear')
clf2 = SVC( train, kernel= 'poly', degree=3)
clf3 = KNN( train, neighbours= 5)
training, validation, testing accuracies are then reported
```

- HPPF is calculated as follows

```
    height, width = img.shape

    horizontal_projection = np.zeros(height)
    for row in range(height):
        for col in range(width):
            if img[row, col] == 0:
                horizontal_projection[row] += 1
```

- SVM Model performance

```
Accuracy of linear kernel on validation data: 0.9727595736281089
Accuracy of poly kernel on validation data: 0.9846032372680615
Accuracy of linear kernel on  test data: 0.9748124753257007
Accuracy of poly kernel on  test data: 0.9901302803000395
```

Example predictions from Poly kernel model

```
cv2_imshow( sample0)
print( clf2.predict( horizontal_projection))
```

```
[0.]
```

```
cv2_imshow( sample1)
print( clf1.predict( horizontal_projection))
```

```
[1.]
```

- KNN Model performance

```
Accuracy on validation data: 0.9869719699960521
Accuracy on test data: 0.9906829846032372
```

Example predictions of KNN Model

```
cv2_imshow( sample0)
print( clf.predict( horizontal_projection))
```

```
[0.]
```

```
cv2_imshow( sample1)
print( clf.predict( horizontal_projection))
```

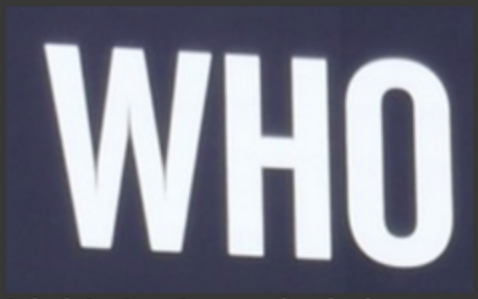
```
[1.]
```

Question 7-W on B/ B on W

Approach

- Given the image it is converted to grayscale and, the bounding box coordinates of text is extracted using EasyOCR
- Average pixel value of the pixels inside and outside the box are calculated from the grayscale image
- If the inside value is lesser than outside, it is dark text of light background, else the opposite

```
in_out(img1, result1)
```




```
Light text on dark bg
Average pixel value of text box: 141.87044
Average pixel value of remaining image: 73.62417134115248
```

```
] in_out(img2, result2)
```


```
Dark text on light bg
Average pixel value of text box: 145.2674940898345
Average pixel value of remaining image: 337.0065359477124
```

Question 8- Template Matching

Reference: [gfg](https://www.geeksforgeeks.org/template-matching-using-opencv-in-python/)

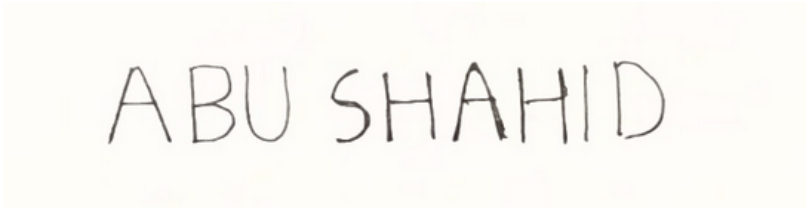
Implementation can be found in the .ipynb file

Approach

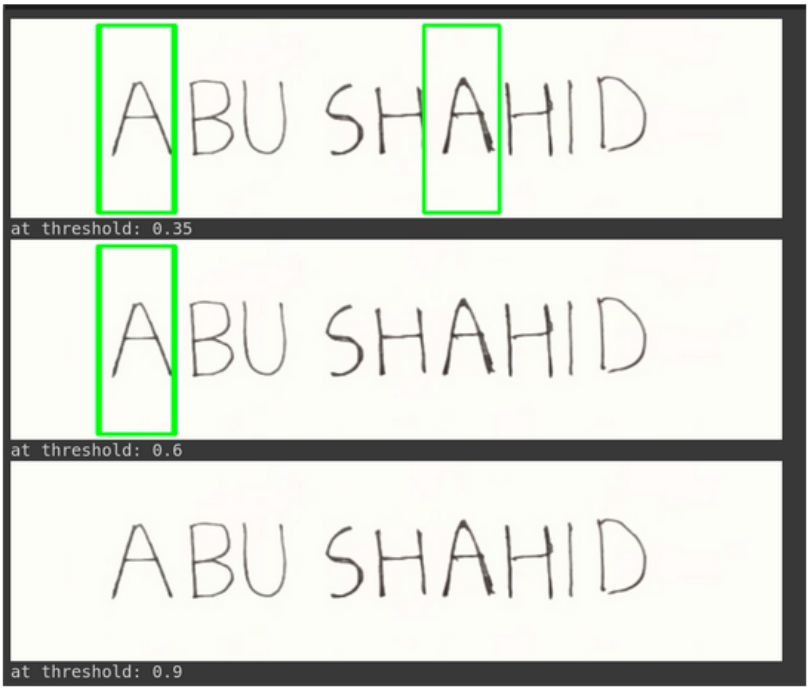
- Image is binarized and cv2.matchTemplate is used to perform the template matching.
- cv2.TM_CCOEFF_NORMED flag is used for matching which stands for Normalized Cross-Correlation Coefficient.
- A threshold is chosen to pick locations which can be considered a match.
- cv2.matchTemplate gives an array which represents the match between the template and the binary image at each location
- The value of threshold had to be cherry-picked

- Template

- Name

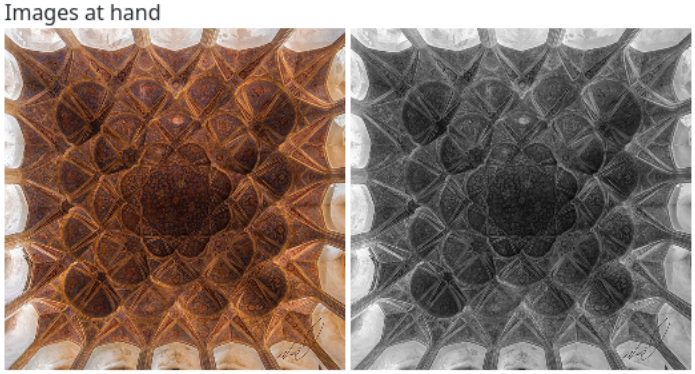


- Results

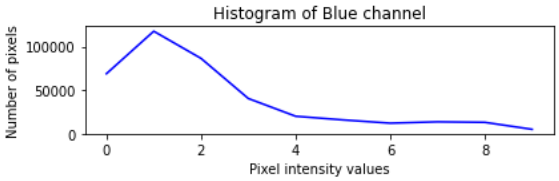


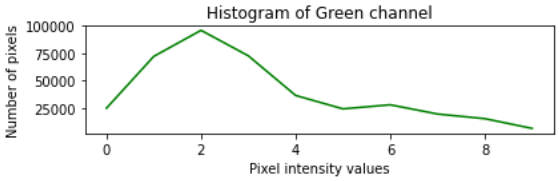
Question 9- Histogram Equalization

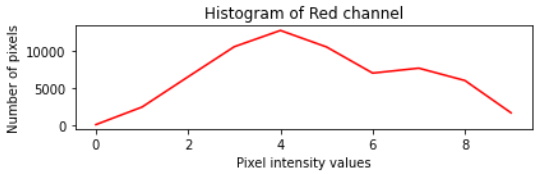
- Images at hand

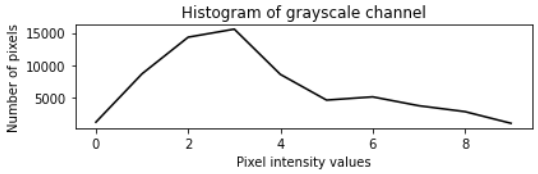


- Histogram of pixel values with bin size=10, using `cv2.calcHist()`

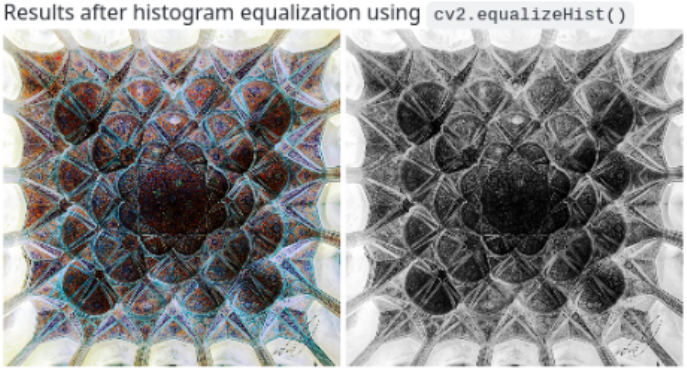




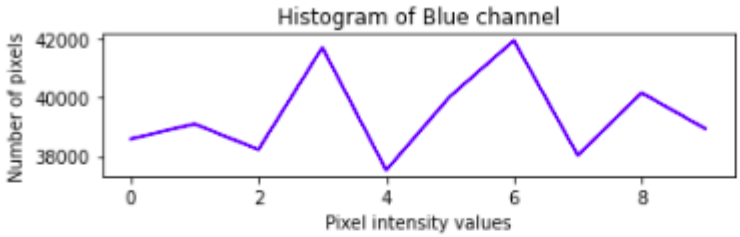


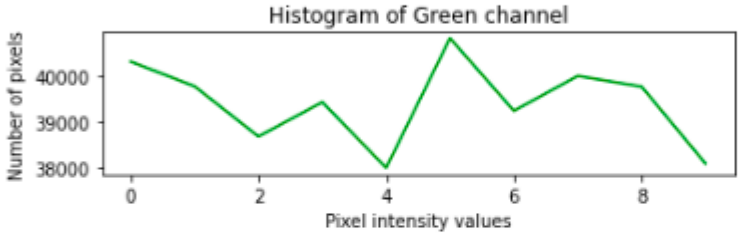


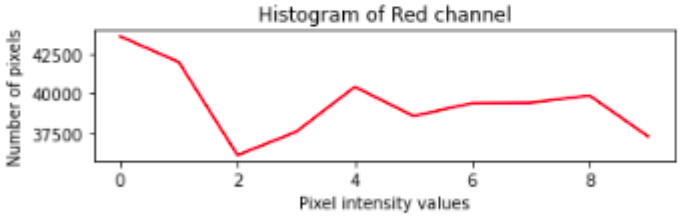
- Results after histogram equalization using `cv2.equalizeHist()`

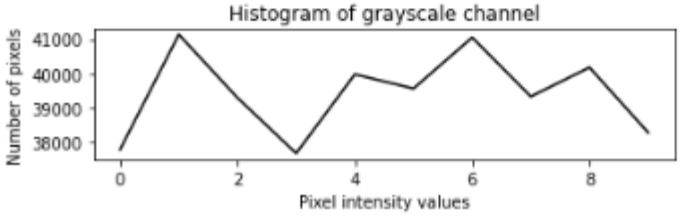


- Histogram of equalized images









Question 10- Reading Mobile Number

Approach:

- Use EasyOCR to read the text in the image
- Extract last three numbers

References

1. https://www.geeksforgeeks.org/template-matching-using-opencv-in-python/