**REGION FILTERING**

**INTRODUCTION**

Filtering is a technique used for modifying or enhancing an image like highlighting certain features or remove other features. Image filtering involves application of window operations that perform useful function such as noise removal and image enhancement that include smoothing, sharpening, edge enhancement. Filtering a region is the process of applying a filter to a region of interest in an image. In this project we have applied the following 6 filters which are pencil sketch, sepia, paint, emboss, cartoon, overlay to a live video feed which can be accessed with the help of the app created with the help of tkinter library in python.

**FUNCTIONS**

**Pencil Sketch:**

Most smoothing filters (e.g. a Gaussian or a Box filter) in image processing and computer vision have a parameter called sigma\_s (for Sigma\_Spatial) that determines the amount of smoothing. A typical smoothing filter replaces the value of a pixel by the weighted sum of its neighbours. The bigger the neighbourhood, the smoother the filtered image looks. The size of the neighbourhood is directly proportional to the parameter sigma\_s.

In edge preserving filters there are two competing objectives — a) smooth the image b) don’t smooth the edges / colour boundaries. In other words we cannot simply replace the colour of a pixel by the weighted sum of its neighbours. Instead, we want to replace the colour value at a pixel by the average of pixels in the neighbourhood which also have colour similar to the pixel. So we have two parameters: sigma\_s and sigma\_r. Just like other smoothing filters sigma\_s controls the size of the neighbourhood, and sigma\_r (for sigma\_range) controls the how dissimilar colours within the neighbourhood will be averaged. A larger sigma\_r results in large regions of constant colour.

**cv2.pencilSketch:**

A predefined function in OpenCV module which performs pencil sketching in both colour and grayscale mode.

* sigma\_s controls how much the image is smoothed - the larger its value, the more smoothed the image gets, but it's also slower to compute. Range 1 - 200
* sigma\_r is important if you want to preserve edges while smoothing the image. Small sigma\_r results in only very similar colours to be averaged (i.e. smoothed), while colours that differ much will stay intact. . Range 0 – 1
* shade\_factor is a simple scaling of the output image intensity. The higher the value, the brighter is the result. Range 0 - 0.1

**Sepia:**

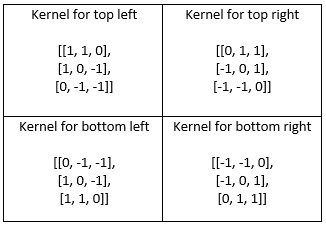
Sepia filter is one of the most popular tools for editing images. The word “sepia” refers to the name of the rich brown pigment that was widely used by photographers back in the early days of photography. Sepiaeffect gives your images a warm brownish tone. Sepia filter improves the general look and feel of your image. We can also create very interesting filter effects using a little bit different image processing. We perform a transformation of colours. Process for creating sepia effect include following steps.  Red, green and blue channels are multiplied with certain coefficients and they created new values for the red, green and blue. Then we split our channels. After we processed them, we obtained new channels for red, green and blue. Once this is finished, we create our final output image by merging channels in new red, new green and new blue. Finally, we plot our image and we can see sepia effect that we are created. It resembles old brown pigment that was widely used by photographers back in the early days of photography. Other way to achieve same effect is to apply the function cv2.transform (). First we create our sepia filter by adding identical coefficients from previous example to it and then we process our image with a sepia filter with the function cv2.transform ().

**Paint:**

Paint basically highlights the finer details of an image by enhancing the hue, tone & saturation values. In the oil painting effect some morphology open is applied for smoothing the outline of the image with the help of cv2.GetStructuringElement function. This function returns a structuring element of the specified size and shape for morphological operations. The function constructs and returns the structuring element that can be further passed to erode, dilate or morphology. Then the dark regions of the image are brightened using cv2.erode function.

**Emboss:**

Image embossing is a computer graphics technique in which each pixel of an image is replaced either by a highlight or a shadow, depending on light/dark boundaries on the original image. Low contrast areas are replaced by a gray background. The meaning of Emboss is to make a mould, a 3d- mould that stands out from the surface. When an emboss filter is applied on an image, the resultant image looks like an emboss - a paper or metal emboss of the source Image where the features stand out in high relief (more prominently) or low relief(less prominently).An emboss filter in Pillow, the Python Image Processing library is provided through applying a convolution of specific 3x3 matrix to the Image.



**Cartoon:**

A cartoon has 2 important characteristics:

a) Really clear edges

b) Homogeneous Colours

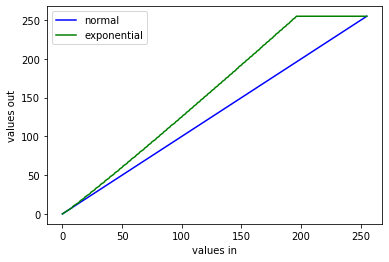
Here stylization filter is used over both edge preserving filter and bilateral filter since bilateral filter is slow and the clarity of clear edges is found to be more in this filter as compared to edge preserving filter. The stylization filter produces an output that looks like the image was painted using water colour. The stylization filter produces an output that looks like the image was painted using water colour. Stylization aims to produce digital imagery with a wide variety of effects not focused on photorealism. Stylization can abstract regions of low contrast while preserving, or enhancing, high-contrast features.

Basically we are going to use a series of filters and image conversions.

* First we downscale the image and then apply bilateral filter to get a cartoon flavour. Then again we upscale the image.
* Next step is getting a blurred version of the original image. Now, we don’t want the colours to interfere in this process. We only want the blurring of the boundaries. For this, we first convert the image to gray – scale and then we apply the median blur filter.
* Next step is to identify the edges in the image and then add this to the previously modified images to get a sketch pen effect. For this first we create an edge mask from the grayscale image using adaptive thresholding and finally cv2.stylization function is used to obtain the desired effect.

**Overlay:**

In the following filter, a screen of a particular colour is added giving the effect that the image was clicked with a paper of that colour against the lens. For this a exponential function was used on a particular channel and the values of other channels were set to zero. In the exponential function, the change in values is less at first and more later on just like exponents. For this we use cv2.LUT after creating a lookup table. The values of the lookup table are shown in the graph below.



**CODE**

import cv2

import numpy as np

import tkinter as tk

from tkinter import \*

from tkinter import filedialog,Text

from tkinter import \*

from PIL import Image,ImageTk

import tkinter.font as tkFont

def blurring(img):

blur=cv2.GaussianBlur(img, (51,51), 0)

return blur

def pencil\_sketch(img):

pencil\_img\_gray,pencil\_img=cv2.pencilSketch(img, sigma\_s=65, sigma\_r=0.1, shade\_factor=0.03)

pencil=cv2.cvtColor(pencil\_img\_gray,cv2.COLOR\_GRAY2BGR)

roi1 = np.array([[(0,0), (224,168),(276,0),(640,0),(640,480),(0,480) ]], dtype=np.int32)

cv2.fillPoly(pencil, roi1, (0,0,0))

return pencil

def sepia\_filter(img):

kernel = np.array([[0.272, 0.534, 0.131],

[0.349, 0.686, 0.168],

[0.393, 0.769, 0.189]])

return cv2.filter2D(img, -1, kernel)

def cartoon(img):

cartoon\_image = cv2.stylization(img, sigma\_s=160, sigma\_r=0.25)

return cartoon\_image

def green\_overlay(img):

def exponential\_function(channel, exp):

table = np.array([min((i\*\*exp), 255) for i in np.arange(0, 256)]).astype("uint8")

channel = cv2.LUT(channel, table)

return channel

def overlay(img, number):

for i in range(3):

if i == number:

img[:, :, i] = exponential\_function(img[:, :, i], 1.02)

else:

img[:, :, i] = 0

return img

img = overlay(img, 1) #(0 = blue, 1 = green and 2 = red)

return img

def emboss\_filter(img):

kernel = np.array([[0,-1,-1],

[1,0,-1],

[1,1,0]])

return cv2.filter2D(img, -1, kernel\*5)

def paint(frame):

kern = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (15,15))

frame7 = cv2.erode(frame.copy(), kern)

return frame7

def webcam\_btn\_clicked():

cap = cv2.VideoCapture(0)

while True:

x,frame=cap.read()

cv2.imshow('Frame',frame)

f1=frame.copy()

f2=frame.copy()

f3=frame.copy()

f4=frame.copy()

f5=frame.copy()

f6=frame.copy()

mask\_colour = (255,255,255)

mask1 = np.zeros(f1.shape, dtype=np.uint8)

roi1 = np.array([[(0,0), (276,0), (224,168)]], dtype=np.int32)

cv2.fillPoly(mask1, roi1, mask\_colour)

masked\_image1 = cv2.bitwise\_and(f1,mask1)

masked\_image1 = pencil\_sketch(masked\_image1)

mask2 = np.zeros(f2.shape, dtype=np.uint8)

roi2 = np.array([[(276,0), (640,0), (340,255),(224,168)]], dtype=np.int32)

cv2.fillPoly(mask2, roi2, mask\_colour)

masked\_image2 = cv2.bitwise\_and(f2,mask2)

masked\_image2=sepia\_filter(masked\_image2)

mask3 = np.zeros(f3.shape, dtype=np.uint8)

roi3 = np.array([[(640,0), (640,480),(340,255)]], dtype=np.int32)

cv2.fillPoly(mask3, roi3, mask\_colour)

masked\_image3 = cv2.bitwise\_and(f3,mask3)

masked\_image3=paint(masked\_image3)

mask4 = np.zeros(f4.shape, dtype=np.uint8)

roi4 = np.array([[(380,285), (640,480),(320,480)]], dtype=np.int32)

cv2.fillPoly(mask4, roi4, mask\_colour)

masked\_image4 = cv2.bitwise\_and(f4,mask4)

masked\_image4=emboss\_filter(masked\_image4)

mask5 = np.zeros(f5.shape, dtype=np.uint8)

roi5 = np.array([[(284,213), (380,285),(320,480),(0,480),(0,290)]], dtype=np.int32)

cv2.fillPoly(mask5, roi5, mask\_colour)

masked\_image5 = cv2.bitwise\_and(f5,mask5)

masked\_image5 = cartoon(masked\_image5)

mask6 = np.zeros(f6.shape, dtype=np.uint8)

roi6 = np.array([[(0,0), (284,213),(0,290)]], dtype=np.int32)

cv2.fillPoly(mask6, roi6, mask\_colour)

masked\_image6 = cv2.bitwise\_and(f6,mask6)

masked\_image6=green\_overlay(masked\_image6)

final1=cv2.bitwise\_or(masked\_image1,masked\_image2)

final2=cv2.bitwise\_or(final1,masked\_image3)

final3=cv2.bitwise\_or(final2,masked\_image4)

final4=cv2.bitwise\_or(final3,masked\_image5)

final5=cv2.bitwise\_or(final4,masked\_image6)

cv2.imshow('Final',final5)

if cv2.waitKey(1) & 0xFF==ord('q'):

break

def exit\_btn\_clicked():

cv2.destroyAllWindows()

exit()

root= tk.Tk()

root.title("Welcome to Region Filtering Application")

fontStyle = tkFont.Font(family="Lucida Grande", size=30)

canvas=tk.Canvas(root, height=550, width=500, bg="#101820")

frame=tk.Frame(canvas, bg="#101820")

frame.place(relx=0.01, rely=0.05, relwidth=0.9, relheight=0.8)

title=tk.Label(frame,text='Region Filtering', fg= '#FEE715', width=15, bg="#101820", font=fontStyle)

title.pack()

title.place(relx=0.15, rely=0.05)

descrption='Description of each filter:\n\n1. Pencil Sketch: This filter turns your webcam feed into beautiful hand-drawn pencil sketches.\n\n2. Sepia: Sepia effect gives your images a warm brownish tone. Sepia filter improves the general look and feel of your image.\n\n3. Paint: It creates an effect of brush strokes, which paint the image.\n\n4. Emboss: The emboss filter, also called a directional difference filter, will enhance edges in the direction of the selected convolution mask(s).\n\n5. Cartoon: It transforms the frame into a cartoon-like picture\n\n6. Green Overlay: Adds a green tint on the frame to produce a green virtual distortion'

desc=tk.Label(frame, text=descrption, bg="#101820", fg='#FEE715', justify=LEFT, wraplength=400)

desc.pack()

desc.place(relx=0.1, rely=0.25)

webcam\_btn=tk.Button(frame,text='Webcam', bg='#101820', fg='#FEE715', padx=3, pady=2, command=webcam\_btn\_clicked, width=20)

webcam\_btn.pack()

webcam\_btn.place(relx=0.1, rely=0.9)

exit\_btn=tk.Button(frame,text='Exit', bg='#101820', fg='#FEE715', padx=3, pady=2, command=exit\_btn\_clicked, width=20)

exit\_btn.pack()

exit\_btn.place(relx=0.65, rely=0.9)

canvas.pack()

root.mainloop()

**CONCLUSION**

From this project we have learnt about various filters, techniques to implement it simultaneously and divide a frame into various segments. The project gives the user to view the provided filters all together in the same frame and experience the difference in the filters. The user can also compare the output frame with the normal frame and notice the differences.

**OUTPUT RESULT**

