ASSIGNMENT 0

(COMPUTER VISION)

SUBMITTED BY:

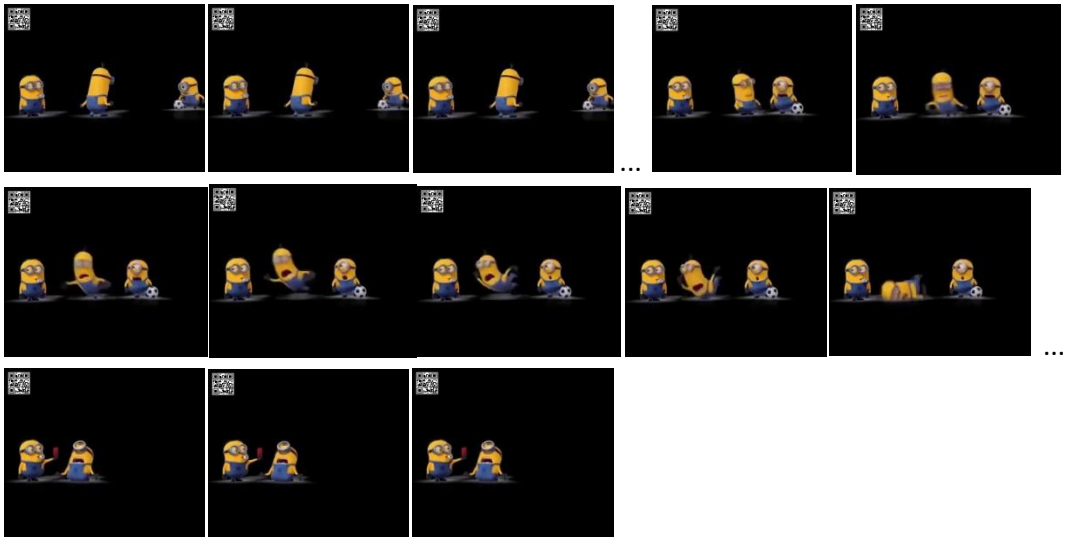
TANU SHARMA

2018702012

Problem 1-a: Converting a video into its constituent frames

Solution: Using VideoCapture class of OpenCV, the video file is read frame by frame. Each frame is displayed and saved in a specified output folder.

Results: For minions.mp4 video, 244 frames were generated. Some of them are as follows



The frame rate of the video is 24fps. Each frame is of 288X240 dimension.

Code:

```
import cv2
import os
import shutil
import numpy as np

# Function to create a folder
def createFolder(name):
    try:
        os.mkdir(name)
        print('Folder created!')
    except:
        print('Folder already exists!')

# Create output folder
def createOpFolder(vid, opFolder):
    # If the video file is not found or some other error occured in opening the file
    if not vid.isOpened():
        print('Error while opening the file')
    # If the file is opened, create an output folder to save frames of the video
    else:
        # If ouput folder exists, delete its data
        if os.path.isdir(opFolder):
```

```
        try:
            # Delete existing output folder and create a new one
            shutil.rmtree(opFolder)
            print('Old folder deleted')
            createFolder(opFolder)
        except:
            print('Old folder not deleted')
    else:
        createFolder(opFolder)

# Converts video to frames and saves them in output folder
def convertVidToIm(filename, opFolder):
    vid = cv2.VideoCapture(filename)
    createOpFolder(vid, opFolder)
    opFrames = []
    frameCount = 0
    while(vid.isOpened()):
        # Return is True if the read() returns a frame
        # frame is the frame of the video
        ret, frame = vid.read()

        # If ret is True, show the frame
        if ret:
            frameCount = frameCount + 1
            opFrames.append(frame)
            cv2.imshow('frame',frame)
            # Save frame in output folder
            fn = opFolder + '/%d.jpg'%frameCount
            cv2.imwrite(fn, frame)
            # Press 'q' to quit
            if cv2.waitKey(25) & 0xFF == ord('q'):
                break
        else:
            break
    opFrames = np.asarray(opFrames)
    vid.release()
    cv2.destroyAllWindows()
    return opFrames

# Constants
opFolder = 'output'
 videoFile = 'input/minions.mp4'
frames = convertVidToIm(videoFile, opFolder)
```

Problem 1-b: Converting a group frames of same dimension into a video.

Solution: A VideoWriter instance is formed and all the frames are written to it one by one in orderly fashion. During this process, user defined frame rate is used based on which the video can be made very fast, very slow or normal.

Results: For 244 frames that were generated in Problem 1-a, videos with different frame rates are formed using the above code. As the frame rate increases, motion in video becomes more smooth. Whereas for very small frame rate the motion is somewhat discrete.

Code:

```
# Reads image in sorted order
def getImages(inputFolder):
    fileType = ''
    imFiles = []

    for a,b,files in os.walk(inputFolder):
        for file in sorted(files):
            name =file.split('.')
            imFiles.append(name[0])

            if not fileType:
                fileType = name[1]
                frame = cv2.imread(inputFolder+'/'+file)
                frameSize = (frame.shape[1],frame.shape[0])

    ims = np.asarray(list(cv2.imread(inputFolder + '/%s'%im + '.' + fileType) for im in sorted(imFiles, key = int)))

    return ims, frameSize

# Plays video
def playVideo(filename):
    vid = cv2.VideoCapture(filename)

    while(vid.isOpened()):
        # Return is True if the read() returns a frame
        # frame is the frame of the video
        ret, frame = vid.read()

        # If ret is True, add the frame
        if ret:
            cv2.imshow('frame', frame)
            # Press 'q' to quit
```

```
            if cv2.waitKey(25) & 0xFF == ord('q'):
                break
        else:
            break

    vid.release()
    cv2.destroyAllWindows()

# Converts a list of images into a single video
def convertImToVid(images, fps, frameSize):
    name = 'outputVideo.mp4'
    opvid = cv2.VideoWriter(name, cv2.VideoWriter_fourcc('M','J','P','G'), fps, frameSize, True)

    for im in images:
        opvid.write(im)

    opvid.release()
    cv2.destroyAllWindows()
    playVideo(name)
```

Problem 2: Capturing video using webcam of the laptop and displaying the video

Solution: The approach is similar as that of Problem 1-a. Just instead of giving a video filename to VideoCapture constructor, we actually give a number i.e., 0 for it to understand that the input is from WebCam.

Result: Frames obtained from the video captured using webcam.

Code:

```
import cv2
import os
import shutil
import numpy as np

# Function to create a folder
def createFolder(name):
    try:
        os.mkdir(name)
        print('Folder created!')
    except:
        print('Folder already exists!')

# Create output folder
def createOpFolder(vid, opFolder):
    # If the video file is not found or some other error occured in opening the file
    if not vid.isOpened():
        print('Error while opening the file')
    # If the file is opened, create an output folder to save frames of the video
    else:
        # If ouput folder exists, delete its data
        if os.path.isdir(opFolder):
            try:
                # Delete existing output folder and create a new one
                shutil.rmtree(opFolder)
                print('Old folder deleted')
                createFolder(opFolder)
```

```
            except:
                print('Old folder not deleted')
        else:
            createFolder(opFolder)

# Converts video to frames and saves them in output folder
def convertVidToIm(filename, opFolder):
    vid = cv2.VideoCapture(filename)
    createOpFolder(vid, opFolder)
    opFrames = []
    frameCount = 0
    while(vid.isOpened()):
        # Return is True if the read() returns a frame
        # frame is the frame of the video
        ret, frame = vid.read()

        # If ret is True, show the frame
        if ret:
            frameCount = frameCount + 1
            opFrames.append(frame)
            cv2.imshow('frame',frame)
            # Save frame in output folder
            fn = opFolder + '/%d.jpg'%frameCount
            cv2.imwrite(fn, frame)
            # Press 'q' to quit
            if cv2.waitKey(25) & 0xFF == ord('q'):
                break
        else:
            break
    opFrames = np.asarray(opFrames)
    vid.release()
    cv2.destroyAllWindows()
    return opFrames

# Constants
webopFolder = 'weboutput'
# Capture and show webcam video and save as images
frames = convertVidToIm(0, webopFolder)
```

Problem 3: Chroma keying

Solution: The following approach works only for green keying color. The algorithm is the one mentioned in wikipedia. According to the algorithm, for each Foreground image pixel, alpha is calculated using

$$alpha = A*(R + B) -B*G$$

where,

A,B = 1 (default value)

R,G,B = RGB value of a pixel of foreground image

If alpha <= 0, then the pixel in foreground image is replaced by the corresponding pixel in background image. Then green spill correction is also done by replacing G in each pixel of resulting image with minimum of (G,B).

Result: Video obtained using snowfall video on my video captured using webcam. Other results.
The above code is based on the algorithm mentioned in Wikipedia. The results as we can see are promising for green background.

Code:

```
import cv2
import os
import shutil
import numpy as np

# Function to create a folder
def createFolder(name):
    try:
        os.mkdir(name)
        print('Folder created!')
    except:
        print('Folder already exists!')

# Create output folder
def createOpFolder(vid, opFolder):
    # If the video file is not found or some other error occured in opening the file
    if not vid.isOpened():
        print('Error while opening the file')
    # If the file is opened, create an output folder to save frames of the video
    else:
        # If ouput folder exists, delete its data
        if os.path.isdir(opFolder):
            try:
                # Delete existing output folder and create a new one
                shutil.rmtree(opFolder)
                print('Old folder deleted')
                createFolder(opFolder)
            except:
                print('Old folder not deleted')
        else:
            createFolder(opFolder)
```

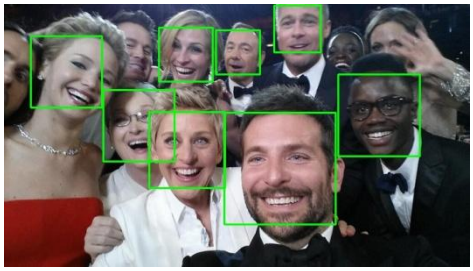
```
# Converts video to frames and saves them in output folder
def convertVidToIm(filename, opFolder):
    vid = cv2.VideoCapture(filename)
    createOpFolder(vid, opFolder)
    opFrames = []
    frameCount = 0
    while(vid.isOpened()):
        # Return is True if the read() returns a frame
        # frame is the frame of the video
        ret, frame = vid.read()

        # If ret is True, show the frame
        if ret:
            frameCount = frameCount + 1
            opFrames.append(frame)
            cv2.imshow('frame',frame)
            # Save frame in output folder
            fn = opFolder + '/%d.jpg'%frameCount
            cv2.imwrite(fn, frame)
            # Press 'q' to quit
            if cv2.waitKey(25) & 0xFF == ord('q'):
                break
        else:
            break
    opFrames = np.asarray(opFrames)
    vid.release()
    cv2.destroyAllWindows()
    return opFrames

# Plays video
def playVideo(filename):
    vid = cv2.VideoCapture(filename)

    while(vid.isOpened()):
        # Return is True if the read() returns a frame
        # frame is the frame of the video
        ret, frame = vid.read()

        # If ret is True, add the frame
        if ret:
            cv2.imshow('frame', frame)
            # Press 'q' to quit
            if cv2.waitKey(25) & 0xFF == ord('q'):
                break
        else:
            break
```

```
    vid.release()
    cv2.destroyAllWindows()

# Converts a list of images into a single video
def convertImToVid(images, fps, frameSize):
    name = 'outputVideo.mp4'
    opvid = cv2.VideoWriter(name, cv2.VideoWriter_fourcc('M','J','P','G'), fps, frameSize, True)
    for im in images:
        opvid.write(im)
    opvid.release()
    cv2.destroyAllWindows()
    playVideo(name)


# fgVid = Foreground video
# bgVid = Background video
def chromaKeying(fgVid, bgVid):
    fgFolder = 'fg'
    bgFolder = 'bg'
    fgFrames = convertVidToIm(fgVid, fgFolder)
    bgFrames = convertVidToIm(bgVid, bgFolder)

    if not ((len(fgFrames) == 0) & (len(bgFrames) == 0)):
        newFgFrames = []
        fs = fgFrames[0].shape

        for fg, bg in zip(fgFrames, bgFrames):
            # For each pixel, check whether fg or bg
            for i in range(0, fs[0]):
                for j in range(0,fs[1]):
                    fgVal = fg[i,j]
                    alpha = np.int16(fgVal[0]) + np.int16(fgVal[2]) - np.int16(fgVal[1])
                    # Replace with background pixel value if alpha is less than 0
                    if alpha <= 0:
                        fg[i,j] = bg[i,j]
                    # Remove green spill
                    fg[i,j] = [fgVal[0], min(fgVal[1], fgVal[2]), fgVal[2]]
            newFgFrames.append(fg)

        convertImToVid(newFgFrames, 25, (fs[1], fs[0]))

# Constants
fg = 'fgVid.mp4'
bg = 'bgVid.mp4'
chromaKeying(fg, bg)
```

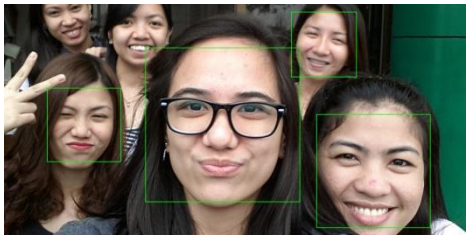
Bonus Problem: Face detection using OpenCV inbuilt function

Solution: For face detection, OpenCV's Haar face detector is used. Scale factor and min neighbours are varied to get best results. If Scale Factor is kept constant and minimum neighbours are increased, the number of faces detected decreases. For smaller values of min neighbour, there are chances of wrongly detected regions. If min neighbours is kept constant and Scale Factor is increased, again the number of faces detected decreases. The smaller the scale factor, more are the chances of small faces getting detected. But this could also lead to wrongly detected regions.

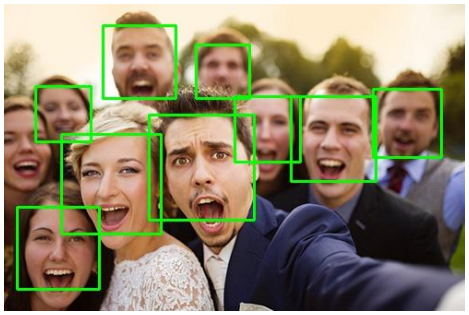
Results: Rectangles location files are also generated along with the face detected images.

Actual number of faces = 12
Faces detected for 1.1 scale factor and 2 minimum number of neighbours = 8

Actual number of faces = 6
Faces detected for 1.1 scale factor and 2 minimum number of neighbours = 4

Actual number of faces = 10
Faces detected for 1.1 scale factor and 2 minimum number of neighbours = 9

Actual number of faces = a lot!
Faces detected for 1.1 scale factor and 2 minimum number of neighbours = 18

Actual number of faces = a lot!
Faces detected for 1.03 scale factor and 5 minimum number of neighbours = 19

Code:

```
import cv

clf = cv2.CascadeClassifier("C:/Users/TANU/Anaconda/pkgs/libopencv-3.4.1-h875b8b8_3/Library/etc/haarcascades/haarcascade_frontalface_default.xml")

I = cv2.imread('faces.jpg')
I_g = cv2.cvtColor(I, cv2.COLOR_RGB2GRAY)

# Scale factor = 1.1
# Minimum neighbours = 2
faces = clf.detectMultiScale(I_g, 1.03, 2)
print('faces: ', len(faces))

opFile = open("face_loc.txt", "w+")
opFile.write("Four coordinates of the faces detected\n\n")

for (a,b,c,d) in faces:
    cv2.rectangle(I, (a,b), (a+c, b+d), (0, 250,0), 2)
    opFile.write("%d\t %d\t %d\t %d\n"%(a, b, (a+c), (b+d)))

opFile.close()
cv2.imwrite('facDet.jpg', I)
```