ISTANBUL TECHNICAL UNIVERSITY

Faculty of Computer Science and Informatics

BLG453E

Computer Vision Homework-5 Report

Author

Cansu Yanık -150170704

1. **Part 1: A (rag)doll with a soul**

First step: Frames are obtained from the videos and saved.

```
walker = mpy.VideoFileClip("walker.avi")
walker_hand = mpy.VideoFileClip("walker_hand.avi")

frame_count = walker_hand.reader.nframes
video_fps = walker_hand.fps


walker_frames = []
walker_hand_frames = []

for i in range(frame_count):
    walker_frame = walker.get_frame(i*1.0/video_fps)
    walker_hand_frame = walker_hand.get_frame(i*1.0/video_fps)

    #walker_hand_frame = (walker_hand_frame > 127)

    if(i%2==0):
        walker_frames.append(walker_frame)
        walker_hand_frames.append(walker_hand_frame)
```

2nd Step: I applied Canny edge detection to find hand position on the image.

1. Gaussian blurring is applied,
2. max and low thresholds are found,
3. Canny is done,
4. Contours are found and drew,
5. White pixels are found,
6. A bounding box is created that contains hand and box area is made bigger,
7. Rectangle drew
8. Image is saved.

```
images = []
hand_area = []
hand_area_top = []
hand_area_bottom = []

for i in range(len(walker_frames)):

    x = walker_frames[i]
    y = walker_hand_frames[i]

    #Apply Canny to hand image to detect hand edges
    gray = cv2.cvtColor(y, cv2.COLOR_BGR2GRAY)
    image = cv2.GaussianBlur(gray,(5,5),0)
    max_treshold,img = cv2.threshold(image,0,255,cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    low_treshold = max_treshold/3
    image = cv2.Canny(image,low_treshold,max_treshold)

    #Apply contour
    contours, hierarchy = cv2.findContours(image.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
    image = cv2.drawContours(image, contours, -1, (255,255,255), 1)

    #Find white pixels
    b, a = np.where(image == 255)

    #Create a bounding box that contains hand
    topy, topx = (np.min(b), np.min(a))
    bottomy, bottomx = (np.max(b), np.max(a))
```

```
    #To make bigger of the box area
    bottomx = topx+50
    bottomy = topy+50

    topx = topx-50
    topy = topy-50

    hand_area_top.append([topx,topy])
    hand_area_bottom.append([bottomx,bottomy])

    #Extract hand area from the image and saves it
    hand = x[topy:bottomy+1, topx:bottomx+1]
    hand_area.append(hand)

    #Draw area to original frame image
    img = cv2.rectangle(img=x, pt1=(topx, topy), pt2=(bottomx, bottomy),
                        color=(0, 0, 255),
                        thickness=1)

    #Saves frame
    images.append(img);
```

I wrote 2 functions. These are *detectMotion* to obtain [u v] and *drawArrows* to draw arrows on images.

- *detectMotion(image1, image2, windowSize):*

1. Kernels are determined to find gradient of image.
2. Image pixels are normalized.
3. Derivatives (Ix, Iy, It) are found by using kernels.
4. Linear system equation is build. b matrix is found.
5. A matrix is formed by using Ix and Iy.
6. [u v] is found. Operations, are given below, is performed.

$$\underset{25\times 2}{\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix}} \underset{2\times 1}{\begin{bmatrix} u \\ v \end{bmatrix}} = -\underset{25\times 1}{\begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}}$$

$$A \qquad V \qquad b$$

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

$$(A^T A) \qquad A^T b$$

$$\underset{2\times 2}{(A^T A)} \; \underset{2\times 1}{V} = \underset{2\times 1}{A^T b}$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = -(A^T A)^{-1} A^T b \quad \bigstar$$

(Above equations were taking from lecture notes.)

```
def detectMotion(image1, image2, windowSize):

    threshold=0.003
    mode = 'same'
    boundary = 'symm'
    u = np.zeros(image1.shape)
    v = np.zeros(image1.shape)

    #Kernels to find gradients (derivatives)
    Kx = np.array([[-1., 1.], [-1., 1.]])
    Ky = np.array([[-1., -1.], [1., 1.]])
    Kt = np.array([[1., 1.], [1., 1.]])

    w = int(windowSize/2)

    #normalize pixels
    image1 = image1 / 255.
    image2 = image2 / 255.

    #Find derivatives
    fx = signal.convolve2d(image1, Kx, boundary=boundary, mode=mode)
    fy = signal.convolve2d(image1, Ky, boundary=boundary, mode=mode)
    ft = signal.convolve2d(image2, Kt, boundary=boundary, mode=mode)
    + signal.convolve2d(image1, -Kt, boundary=boundary, mode=mode)
```

```
    # within window window_size * window_size
    for i in range(image1.shape[0]):
        for j in range(image1.shape[1]):

            Ix = fx[i-w:i+w+1, j-w:j+w+1].flatten()
            Iy = fy[i-w:i+w+1, j-w:j+w+1].flatten()
            It = ft[i-w:i+w+1, j-w:j+w+1].flatten()
            #Find b
            b = np.reshape(It, (It.shape[0],1))

            A = np.vstack((Ix, Iy)).T   #A = [Ix Iy]
            X = np.matmul(A.T, A)
            # the eigenvalues of A.TA should not be too small
            if np.min(abs(np.linalg.eigvals(X))) >= threshold:
                X = np.linalg.pinv(X)
                X = -1 * X
                B = np.matmul(A.T, b)
                result = np.matmul(X, B)
                u[i,j]=result[0]
                v[i,j]=result[1]

    return (u,v)
```

- *drawArrows(image, flag, u=None, v=None, flowVector=None)*

Function draws arrows on the image for interested area of hand.

```
def drawArrows(image, flag, u=None, v=None, flowVector=None):

    step=12
    row, col, h = image.shape

    #Separate area into steps to show arrows
    y, x = np.mgrid[step/2:row:step, step/2:col:step].reshape(2,-1).astype(int)

    if (flag==1):
        fx, fy = flowVector[y,x].T
        fx = fx*2
        fy = fy*2
    else:
        fx = u[y,x].T
        fy = v[y,x].T

    #Find starting and end points of arrow
    arrows = np.vstack([x, y, x+fx, y+fy]).T.reshape(-1, 2, 2)
    arrows = np.int32(arrows + 0.5)

    color = (0, 255, 0)
    thickness = 2
    length = 0.5

    #Draw arrows
    for (x1, y1), (x2, y2) in arrows:
        image = cv2.arrowedLine(image, (x1, y1), (x2, y2), color, thickness, tipLength = length)
    return image
```

Then, for every frame of the video, I also took next frame to detect motion between them.

Note: In addition, I also wanted to use existing motion detection function to be able to see proper result and compare it with my implementation result. I also uploaded this function's motion flow result. Windowsize = 15

```
frames = []
j = 0
for i in range(len(walker_frames)-1):

    print("Step: ",i+1)
    j = i + 1

    image = images[i]
    image1c = hand_area[i]
    image2c = hand_area[j]

    #Convert consecutive images to gray scale
    image1 = cv2.cvtColor(image1c, cv2.COLOR_BGR2GRAY)
    image2 = cv2.cvtColor(image2c, cv2.COLOR_BGR2GRAY)

    #Image1 bounding box coordinates
    topx_1 = hand_area_top[i][0]
    topy_1 = hand_area_top[i][1]

    bottomx_1 = hand_area_bottom[i][0]
    bottomy_1 = hand_area_bottom[i][1]


    #Find the motion field for only hand area
    u1, v1 = detectMotion(image1, image2, 15)
    out = drawArrows(image1c, 0, u1, v1, None)

    #Put result on original image
    image = images[i].copy()
    image[topy_1:bottomy_1+1, topx_1:bottomx_1+1] = out
    frames.append(image)
```

```
'''
#Here, I used existing function to see proper result, I also uploaded this result
flowVector = cv2.calcOpticalFlowFarneback(image1, image2, None, 0.5, 3, 15, 3, 5, 1.2, 0)
out = drawArrows(image1c, 1, None, None, flowVector)

image = images[i].copy()
image[topy_1:bottomy_1+1, topx_1:bottomx_1+1] = out
frames.append(image)
'''
```

2. Part 2: Part 2: "I'm Nobody! Who are you? / Are you -Nobody-too?"

I wrote 5 functions:

- *image_to_vector(image):*

  It takes image as parameter. It applies Principle Component Analysis (PCA) that does projection of high dimensional data to low dimensional space.

  I chose 3 components. Then I obtained a 1d vector for the image.

```
#Function apply dimension reduction and convert it to 1d vector
def image_to_vector(image):

    image = PCA(n_components=3).fit_transform(image)
    size = image.shape
    image_vector = cv2.resize(image, size).flatten()
    return image_vector
```

- *_prepareData(train_data_dir):*

  Function traverses all folders and reads all images inside folders. It saves images into an array and in another array the names of folders are saved to be used as group labels.

```
#Function traverses all folders and reads all images inside folders
#Prepare train data and group labels
def _prepareData(train_data_dir):

    X = []
    y = []

    for class_dir in os.listdir(train_data_dir):
        if not os.path.isdir(os.path.join(train_data_dir, class_dir)):
            continue

        for image_path in os.listdir(os.path.join(train_data_dir, class_dir)):
            path = os.path.join(train_data_dir, class_dir)

            image = cv2.imread(os.path.join(path, image_path))
            image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            image_vector = image_to_vector(image)

            X.append(image_vector)
            y.append(class_dir)

    return X, y
```

- *_train(X, y, model_save_path=None, trainAlgo='auto'):*

Function creates a model to predict the group of new input image. K-nearest neighbor classification is used. K = 10.

```
#Train data by using k nearest neighbors
def _train(X, y, model_save_path=None, trainAlgo='auto'):

    model = neighbors.KNeighborsClassifier(n_neighbors=10, algorithm=trainAlgo, weights='distance')
    model.fit(X, y)

    #Model can be saved for reusing
    if model_save_path is not None:
        with open(model_save_path, 'wb') as file:
            pickle.dump(model, file)

    return model
```

- *_predict(image_path, model=None, model_save_path=None):*

Function takes image path as parameter, reads image, converts to gray scale, calls *image_to_vector* function to get 1d vector representation then reshapes it. After preparation of image, prediction to find the group of image is made.

```
#Predict an input image
def _predict(image_path, model=None, model_save_path=None):

    #If model is saved before, it can be taken
    if model is None and model_save_path is None:
        raise Exception("There is no model or model path")

    if model is None:
        with open(model_save_path, 'rb') as file:
            model = pickle.load(file)

    #Convert image to gray scale and apply dimension reduction
    my_image = cv2.imread(image_path)
    my_image = cv2.cvtColor(my_image, cv2.COLOR_BGR2GRAY)
    my_image_vector = image_to_vector(my_image)
    my_image_vector = my_image_vector.reshape(1, -1)

    predict = model.predict(my_image_vector)
    return predict
```

- *_show_who_are_you(my_image_path, main_dir, result):*

Also, I wrote an additional function to show result in a form of image. Function takes input image path, main directory where train data is located and result of the prediction (name of the folder) as parameters. Then function finds the folder of result and takes first image of the result person. Input image and prediction result will be shown.

```python
#I used this method to show result as an image
#It takes first image of the result person from its folder
def _show_who_are_you(my_image_path, main_dir, result):

    out_path = os.path.join(main_dir, result[0])
    out_path =  os.path.join(out_path, os.listdir(os.path.join(main_dir, result[0]))[0])

    input_image = cv2.imread(my_image_path)
    output_image = cv2.imread(out_path)

    input_image = cv2.cvtColor(input_image, cv2.COLOR_BGR2GRAY)
    output_image = cv2.cvtColor(output_image, cv2.COLOR_BGR2GRAY)

    #Makes bigger of images
    input_image = cv2.resize(input_image,(150,200))
    output_image = cv2.resize(output_image,(150,200))

    horizontal_concat = np.concatenate((input_image, output_image), axis=1)

    cv2.imshow('Who are you?: ' + result[0], horizontal_concat)
    cv2.waitKey()
```

Then initial variables are prepared and functions are called in proper order.

```python
main_dir = './VGGFace-subset'
my_image_path = './cansu.jpg'

X, y = _prepareData(main_dir)
model = _train(X, y)
result = _predict(my_image_path, model)
print(result[0])

_show_who_are_you(my_image_path, main_dir, result)
```

An example result is given below.

>> Data is preparing...

Model is preparing...

Prediction is doing...

Result is: Wentworth_Miller