**Optical Flow of Blocks**

import cv2

import numpy as np

import math

def calcOptFlowOfBlocks(mag,angle,grayImg):

'''Takes an image (gray scale) and a flow matrix as input. Divides image into blocks and calculates Optical Flow of each block '''

'''calculate number of rows and columns in the matrix of the image'''

rows = grayImg.shape[0]

cols = grayImg.shape[1]

noOfRowInBlock = 20

noOfColInBlock = 20

''' calculate the number of rows of blocks and columns of blocks in the frame '''

xBlockSize = rows // noOfRowInBlock

yBlockSize = cols // noOfColInBlock

'''To calculate the optical flow of each block'''

'''declare an array initialized to 0 of the size of the number of blocks'''

opFlowOfBlocks = np.zeros((xBlockSize,yBlockSize,2))

for index,value in np.ndenumerate(mag):

opFlowOfBlocks[index[0]//noOfRowInBlock][index[1]//noOfColInBlock][0] += mag[index[0]][index[1]]

opFlowOfBlocks[index[0]//noOfRowInBlock][index[1]//noOfColInBlock][1] += angle[index[0]][index[1]]

centreOfBlocks = np.zeros((xBlockSize,yBlockSize,2))

for index,value in np.ndenumerate(opFlowOfBlocks):

opFlowOfBlocks[index[0]][index[1]][index[2]] = float(value)/(noOfRowInBlock\*noOfColInBlock)

val = opFlowOfBlocks[index[0]][index[1]][index[2]]

if(index[2] == 1):

angInDeg = math.degrees(val)

if(angInDeg > 337.5):

k = 0

else:

q = angInDeg//22.5

a1 = q\*22.5

q1 = angInDeg - a1

a2 = (q+2)\*22.5

q2 = a2 - angInDeg

if(q1 < q2):

k = int(round(a1/45))

else:

k = int(round(a2/45))

opFlowOfBlocks[index[0]][index[1]][index[2]] = k

theta = val

if(index[2] == 0):

r = val

x = ((index[0] + 1)\*noOfRowInBlock)-(noOfRowInBlock/2)

y = ((index[1] + 1)\*noOfColInBlock)-(noOfColInBlock/2)

centreOfBlocks[index[0]][index[1]][0] = x

centreOfBlocks[index[0]][index[1]][1] = y

return opFlowOfBlocks,noOfRowInBlock,noOfColInBlock,noOfRowInBlock\*noOfColInBlock,centreOfBlocks,xBlockSize,yBlockSize

**Motion Influence Generator**

import cv2

import numpy as np

import opFlowOfBlocks as roi

import math

def getThresholdDistance(mag,blockSize):

return mag\*blockSize

def getThresholdAngle(ang):

tAngle = float(math.pi)/2

return ang+tAngle,ang-tAngle

def getCentreOfBlock(blck1Indx,blck2Indx,centreOfBlocks):

x1 = centreOfBlocks[blck1Indx[0]][blck1Indx[1]][0]

y1 = centreOfBlocks[blck1Indx[0]][blck1Indx[1]][1]

x2 = centreOfBlocks[blck2Indx[0]][blck2Indx[1]][0]

y2 = centreOfBlocks[blck2Indx[0]][blck2Indx[1]][1]

slope = float(y2-y1)/(x2-x1) if (x1 != x2) else float("inf")

return (x1,y1),(x2,y2),slope

def calcEuclideanDist(x1,y1,x2,y2):

dist = float(((x2-x1)\*\*2 + (y2-y1)\*\*2)\*\*0.5)

return dist

def angleBtw2Blocks(ang1,ang2):

if(ang1-ang2 < 0):

ang1InDeg = math.degrees(ang1)

ang2InDeg = math.degrees(ang2)

return math.radians(360 - (ang1InDeg-ang2InDeg))

return ang1 - ang2

def motionInMapGenerator(opFlowOfBlocks,blockSize,centreOfBlocks,xBlockSize,yBlockSize):

global frameNo

motionInfVal = np.zeros((xBlockSize,yBlockSize,8))

for index,value in np.ndenumerate(opFlowOfBlocks[...,0]):

Td = getThresholdDistance(opFlowOfBlocks[index[0]][index[1]][0],blockSize)

k = opFlowOfBlocks[index[0]][index[1]][1]

posFi, negFi = getThresholdAngle(math.radians(45\*(k)))

for ind,val in np.ndenumerate(opFlowOfBlocks[...,0]):

if(index != ind):

(x1,y1),(x2,y2), slope = getCentreOfBlock(index,ind,centreOfBlocks)

euclideanDist = calcEuclideanDist(x1,y1,x2,y2)

if(euclideanDist < Td):

angWithXAxis = math.atan(slope)

angBtwTwoBlocks = angleBtw2Blocks(math.radians(45\*(k)),angWithXAxis)

if(negFi < angBtwTwoBlocks and angBtwTwoBlocks < posFi):

motionInfVal[ind[0]][ind[1]][int(opFlowOfBlocks[index[0]][index[1]][1])] += math.exp(-1\*(float(euclideanDist)/opFlowOfBlocks[index[0]][index[1]][0]))

print("Frame number ", frameNo)

frameNo += 1

return motionInfVal

def getMotionInfuenceMap(vid):

global frameNo

frameNo = 0

cap = cv2.VideoCapture(vid)

ret, frame1 = cap.read()

rows, cols = frame1.shape[0], frame1.shape[1]

prvs = cv2.cvtColor(frame1, cv2.COLOR\_BGR2GRAY)

motionInfOfFrames = []

count = 0

while 1:

'''

#if(count <= 475 or (count > 623 and count <= 1300)):

if(count < 475):

ret, frame2 = cap.read()

prvs = cv2.cvtColor(frame2,cv2.COLOR\_BGR2GRAY)

count += 1

continue

'''

#if((count < 1451 and count <= 623)):

'''

if(count < 475):

ret, frame2 = cap.read()

prvs = cv2.cvtColor(frame2,cv2.COLOR\_BGR2GRAY)

count += 1

continue

'''

print(count)

ret, frame2 = cap.read()

if (ret == False):

break

next = cv2.cvtColor(frame2, cv2.COLOR\_BGR2GRAY)

flow = cv2.calcOpticalFlowFarneback(prvs, next, None, 0.5, 3, 15, 3, 5, 1.2, 0)

mag, ang = cv2.cartToPolar(flow[...,0], flow[...,1])

prvs = next

opFlowOfBlocks,noOfRowInBlock,noOfColInBlock,blockSize,centreOfBlocks,xBlockSize,yBlockSize = roi.calcOptFlowOfBlocks(mag,ang,next)

motionInfVal = motionInMapGenerator(opFlowOfBlocks,blockSize,centreOfBlocks,xBlockSize,yBlockSize)

motionInfOfFrames.append(motionInfVal)

#if(count == 622):

# break

count += 1

return motionInfOfFrames, xBlockSize,yBlockSize

**Creating mega blocks**

import cv2

import numpy as np

import math

import itertools

def createMegaBlocks(motionInfoOfFrames,noOfRows,noOfCols):

n = 2

megaBlockMotInfVal = np.zeros(((noOfRows/n),(noOfCols/n),len(motionInfoOfFrames),8))

frameCounter = 0

for frame in motionInfoOfFrames:

for index,val in np.ndenumerate(frame[...,0]):

temp = [list(megaBlockMotInfVal[index[0]/n][index[1]/n][frameCounter]),list(frame[index[0]][index[1]])]

megaBlockMotInfVal[index[0]/n][index[1]/n][frameCounter] = np.array(map(sum, zip(\*temp)))

frameCounter += 1

print(((noOfRows/n),(noOfCols/n),len(motionInfoOfFrames)))

return megaBlockMotInfVal

def kmeans(megaBlockMotInfVal):

#k-means

cluster\_n = 5

criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 10, 1.0)

flags = cv2.KMEANS\_RANDOM\_CENTERS

codewords = np.zeros((len(megaBlockMotInfVal),len(megaBlockMotInfVal[0]),cluster\_n,8))

#codewords = []

#print("Mega blocks ",megaBlockMotInfVal)

for row in range(len(megaBlockMotInfVal)):

for col in range(len(megaBlockMotInfVal[row])):

#print("megaBlockMotInfVal ",(row,col),"/n/n",megaBlockMotInfVal[row][col])

ret, labels, cw = cv2.kmeans(np.float32(megaBlockMotInfVal[row][col]), cluster\_n, None, criteria,10,flags)

#print(ret)

#if(ret == False):

# print("K-means failed. Please try again")

codewords[row][col] = cw

return(codewords)

**Training**

import numpy as np

import motionInfuenceGenerator as mig

import createMegaBlocks as cmb

def reject\_outliers(data, m=2):

return data[abs(data - np.mean(data)) < m \* np.std(data)]

def train\_from\_video(vid):

'''

calls all methods to train from the given video

May return codewords or store them.

'''

print ("Training From ", vid)

MotionInfOfFrames, rows, cols = mig.getMotionInfuenceMap(vid)

print ("Motion Inf Map", len(MotionInfOfFrames))

#numpy.save("MotionInfluenceMaps", np.array(MotionInfOfFrames), allow\_pickle=True, fix\_imports=True)

megaBlockMotInfVal = cmb.createMegaBlocks(MotionInfOfFrames, rows, cols)

np.save("videos/scene1/megaBlockMotInfVal\_set1\_p1\_train\_40-40\_k5.npy",megaBlockMotInfVal)

print(np.amax(megaBlockMotInfVal))

print(np.amax(reject\_outliers(megaBlockMotInfVal)))

codewords = cmb.kmeans(megaBlockMotInfVal)

np.save("videos/scene1/codewords\_set1\_p1\_train\_40-40\_k5.npy",codewords)

print(codewords)

return

if \_\_name\_\_ == '\_\_main\_\_':

'''

defines training set and calls trainFromVideo for every vid

'''

trainingSet = [r"videos/scene1/train1.avi"]

for video in trainingSet:

train\_from\_video(video)

print("Done")