opencv introduction

Tutorial 1

1 loading and Displaying Images using the cv2.image

```
import argparse
import cv2

ap = argpase.ArguentParser()
ap.add_argument("-i","--image",required=True, help="path to input image")
args = vars(ap.parse_args())

# load the image from the disk via cv2.imread and then grab the spatiall
# dimensions including width heigh and number of channels

image = cv2.imread(args["image"])
(h,w,c) = image.shape[:]

# display the image width height and number of channels to our terminal

print ("width: {} pixels".format (w))
print ("height: {} pixels".format (h))
print ("channels: {} ".format (c))

# show the image and wait for a keypress
cv2.imshow("Image",image)
cv2.waitKey(0)

# save the image back to disk
cv2.imwrite ("newimage.jpg", image)
# width (no column)
# heght (no rows)
```

tutorial two

Getting and setting pixels and how to access and manuplating pixels values

```
# importing the necessary packages
import argparse
import cv2

# constrauct the argument parser and the parse the arguments
ap =argparse.ArgumentParser()
ap.add_argument("-i", ("--image", type = str, default="adrian.png",help="path to the input image" ))
args = vars(ap.parse_args())
```

```
""" load the image grab its spatial dimentions (width and height) and the display
the original image to our screen """

image = cv2.imread(args["image"])
(h,w) = image.shape[:2]
cv2.imshow("Original",image)


""" images are simply numpy arrays -- with the origin (0,0) located at  the top
left of the image """

(b,g,r) = image [0,0]
print ("Pixel at (0,0) -Red: {}, Green: {}, Blue: {}".format(r,g,b))

# access the pixel at x = 50 y = 20
(b,g,r) = image [20,50]
print ("Pixel at (50,20) -Red: {}, Green: {}, Blue: {}".format(r,g,b))

# update the pixel at (50,20) and set it to red
image [20,50] = (0,0,255)
(b,g,r) = image [0,0]
print ("Pixel at (0,0) -Red: {}, Green: {}, Blue: {}".format(r,g,b))

""" compute the center of the image which is simply the width and height divided
by two """

(cX,cY) = (w//2,h//2)

""" since we are using Numpy arrays we can apply arry slicing to grab larger
chunks/regions of interest from the image --here we grab the top-left corner of
the image """
#starty:endy

tl= image [0:cY, 0:cX]
cv2.imshow ("Top-Left Corner", tl)

""" in a similar fashion we can crop the top-right, bottom-right, and bottom-left
corners of the image and then display them to our sreen """

tr = image [0:cY, cX:w]
br = image [cY:h, cX:w]
bl = image [cY:h,0:cX]
cv2.imshow ("Top-Right Corner",tr)
cv2.imshow ("Bottom-Right Corner",br)
cv2.imshow ("Bottom-left Corner",bl)

# set the top left corner of the original image to be green
image [0:cY, 0:cX] = (0,255,0)

#show our updated image
cv2.imshow ("Updated",image)
cv2.waitKey(0)
```

Tutorial 3

Drawing with openCV

basic drawing using opencv

```
import the necessary packages
import numpy as np
import cv2

""" initialize our canvas as 300X300 pixel image with 3 channels (Red, Green,
Blue) with black backgound """
canvas = np.zeros((300,300, 3), dtype="uint8")

""" draw a green line from the top-left corner of our canvas to the bottom-right
"""
green = (0,255,0)
cv2.line(canvas, (0,0),(300,300),green)
cv2.imshow("Canvas", canvas)
cv2.waitKey(0)

# draw a 3 pixel thick line red line from the top-right corner to the
# bottom-left

red = (0,0,255)
cv2.line (canvas, (300,0),(0,300),red,3)
cv2.imshow("Canvas", canvas)
cv2.waitKey(0)

# draw a green 50x50 pixel square starting at 10,10 and ending at 60,60
cv2.rectangle(canvas,(10,10),(60,60)green)
cv2.imshow("Canvas", canvas)
cv2.waitKey(0)

#draw another rectangle this one red with 5 pixel thickness
cv2.rectangle(canvas, (50,200),(200,225),red,5)
cv2.imshow("Canvas",canvas)
cv2.waitKey(0)

# draw a final rectangle (blue and fillled in)
blue = (255, 0,0)
cv2.rectangle(canvas, (200,50),(255,125),blue, -1)
cv2.imshow("Canvas",canvas)
cv2.waitKey(0)

""" re-initialize our canvas as an empty array then compute the center (x,y)-
coorinates of the canvas """

canvas= np.zeros((300,300,3), dtype="uint8")
(centerX, centerY) = (canvas.shape[1]//2,canvas.shape[0]//2)
white = (255,255,255)
```

```
""" loop over increasing radii from 25 pixel to 150 pixel in 25 pixel increments
"""

for rin range (0,175,25):
    # drae a white cricle with the current radius size
    cv2.circle (canvas,(centerX, centerY),r,white)

# show our work of art
cv2.imshow ("Canvas",canvas)
cv2.waitKey(0)

# re-initialllize our canvas again
canvas = np.zeros((300,300,3),dtype = "uint8")

# let draw 25 random circles
for i in range (0,25):
    """ randomly generate a radius size between 5 and 200 generate a random color
and the pick a random point on our canvas where the circle will be drawn """

    radius = np.random.randint(5, high = 200)
    color = np.random.randint(0,high=256, size=(3,)).tolist()
    pt = np.random.randint(0,high=300,size=(2,))
    cv2.circle(canvas,tuple(pt),radius,color,-1)

#display our masterpiece to our screen
cv2.imshow("Canvas",canvas)
cv2.waitKey(0)
```

drwaing on top of the image

import the necessary packages

```
import argparse
import cv2

ap = argparse.ArgumentParser()
ap.add_argument("-i","--image",type=str,default="name of the image",help="path to
the input image")
args = vars(ap.parse_args())

# load the input image from disk
image = cv2.image(args["image"])

""" draw a circle around my face, two filled in circles covering my eyes and a
rectangle over top of my mouth """
red = (0,0,255)

cv2.circle(image, (168,188),90,red,2)
cv2.circle(image,(150,164),10,red,-1)
cv2.circle(image,(192,174),10,red,-1)
cv2.rectangle(image,(134,200),(186,218),red,-1)
```

```
cv2.imshow("Output",image)
cv2.waitKey(0)
```

Tutarial four

Geometrical Translation

```
import numpy as np
import argparse
import imutils
import cv2

ap =argparse.ArgumenteParser()
ap.add_argument("-i","--image",type=str, default="name of image",help="path to the
input image")
args = vars(ap.parse_args())

#load the image and display it to the screen
image = cv2.imread(args["image"])
cv2.imshow("Original",image)

# shift the image 25 pixels to the right and 50 pixels down
M = np.float32([1,0,25],[0,1,50])
"""
Translating (shifting )an image is given by a numPy matrix in the form
[
    [1,0, shiftX] -values to the left + to the right
    [0,1, shiftY] -values to the up + down

]

"""
shifted = cv2.warpAffine(image, M,(image.shape[1],image.shape[0]))
cv2.imshow ("Shifted Down and Right", shifted)

""" now lets shift the image 50pixel to the left and 90 pixels up by specifying
negative values for the x and y directions respectively """
M = npfloat32([[1,0,-50],[0,1,-90]])
shifted = cv2.warpAffine(image, M, (image.shape[1],image.shape[0]))
cv2.imshow("Shifted up and left",shifted)

""" use the imutils helper function to translate the image 100 pixels down in a
single function call """
shifted = imutils.translate(image,0,100)
cv2.imshow("Shifted Down",shifted)
cv2.waitKey(0)
```

Tutarila five

image rotation

rotation matrix

```
import argparse
import imutils
import cv2

ap = argparse.ArgumentParser()
ap.add_argument("-i","--image",typ=str,default="image location",help="path to the
input image")
args= vars(ap.parse_args())

# load the image and show it
image = cv2.imread(args["images"])
cv2.imshow("Original",image)

#grab the dimensions of the image and calculate the center of the image
(h,w)=image.shape[:2]
(cX,cY)= (w//2,h//2)


# rotate the image by 45 degrees around the center of the image
M = cv2.getRotationMatrix2D((cX,cY),45,1.0) # scale
rotaed = cv2.warpAffine(image, M, (w,h))
cv2.imshow("Rotated by 45 degree",rotated)

# rotate the image by -90 degrees around the center of the image
M = cv2.getRotationMatrix2D((cX,cY),-90,1.0) # clockwise use neative values
rotated = cv2.warpAffine(image,M,(w,h))
cv2.imshow ("Rotated by -90 Degrees",rotated)

# rotat the image around arbitrary point rather than the center
M = cv2.getRotationMatrix2D((10,10),45,1.0)
rotated = ccv2.warpAffine (image,M,(w,h))
cv2.imshow("Rotated by arbitrary point",rotated)

# using the imutils function to rotate an image 180 degrees
rotated = imutils.rotate(image,180)
cv2.imshow("Rotated by 180 degrees",rotated)

""" rotate our image by 33 degree counterclockwise ensuring the entire rotated
image still views in the viewing area """
rotated = imutils.rotate_bound(image, -33)
cv2.imshow("Rotated without Cropping",rotated)
cv2.waitKey(0)
```

Tutorial six

Resizing imaze (cv2.resize)

```
import argparse
import imutils
import cv2

ap = argparse.ArgumentParser()
ap.add_argument("-i","--image",type=str,default="imaga name",help="path to input
image")
args = vars(ap.parse_args())

#load the original image and display it on the screen
image = cv2.imread(args["image"])
cv2.imshow("Original",image)

""" lets resize our image to be 150 pixels wide but in order to prevent our
resized image from being skewed/distorted we must first calculate the ration of
the new width to the old width """

r 150.0/image.shape[1]
dim = (150,int(image.shape[0]*r))

# perform the actual resizing of the image
resized = cv2.resized(image,dim,interpolation=cv2.INTER_AREA)
cv2.imshow("Resized width",resized)


""" lets resize the image to have a width of 50 pixels again keeping in mind the
aspect ration """

r = 50.0/image.shape[0]
dim = (int(image.shape[1]*r),50)

#perform the resizing
resized = cv2.resize (image,dim,interpolation=cv2.INTER_AREA)
cv2.imshow("Resized (Height)", resized)
cv2.waitKey(0)

""" calculating the ration each and every time we want to resize an image is a
real pain so lets use the imutils convenience function which will *automatically
maintain our aspect ration for us """

resized = imutils.resize(image,width=100)
cv2.imshow("Resized via imutils",resized)
cv2.waitKey(0)

# construct the list of interpolation methods in opencv
methods =[
("cv2.INTER_NEAREST",cv2.INTER_NEAREST),
("cv2.INTER_LINEAR",cv2.INTER_LINEAR),L2S
("cv2.INTER_AREA", cv2.INTER_AREA),
("cv2.INTER_CUBIC",cv2.INTER_CUBIC),# S2L
("cv2.INTER_LANCZOS4",cv2.INTER_LANCZOS4)]

""" loop over the unterpolation methods in openCv
```

```
for (name,method) in methods: increase the size of the image by 3x using the
current interpolation method """
print ("[INFO] {}".format(name))
resized = imutils.resize(image,width=image.shape[1]*3,inter=method)

cv2.imshow("Method: {}".format(name), resized)
cv2.waitKey(0)
```

Tutorial seven

flliping images

```
# import the necessary packages
import argparse
import cv2

ap = argperse.ArgumentParser()
ap.add_argument ("-i","--image",type=str,default="image name",help="path to the
input image")
args = vars(ap.parse_args())

# load the original input image and display it to our screen
image = cv2.imread(args["image"])
cv2.imshow("Original", image)

# flip the image vertically
print ("[INFO] flipping image vertically..")
flipped = cv2.flip(image,1)
cv2.imshow("Flipped Horizontally", flipped)

# flip the image vertically
print ("[INFO] flipping the image vertically...")
flipped = cv2.flip(image, 0)
cv2.imshow("Flipped vertically",flipped)

# flip the image along both axes
print ("[INFO] flipping image horizontally and vertically ...")
flipped = cv2.flip(image,-1)
cv2.imshow("Flipped horizontally & vertically",flipped)
cv2.waitKey(0)
```

Tutorial 8

cropping image withopencv

```
# import the necessary packages
import argparse
import cv2
```

```
ap = argparse.ArgumentParser()
ap.add_argument("-i","--image",type=str,default="image location",help="path to the
input imagae")
args = vars(ap.parse_args())

# load the input image and display it to our screen
image=cv2.imread(args["image"])
cv2.imshow("Original",image)

""" cropping an image with openCV is accomplishe via simple numpy array sclices in
startY:endY, startX:endX order --here we are cropping the face from the image
(these coordinates were determined using photo editiing software such as
photoshop)"""
face = image[85:250,0:200]
cv2.imshow("FACE",face)
cv2.waitKey(0)

#apply another image crop this time extracting the body
body = image[90:450,0:290]
cv2.imshow("Body",body)
cv2.waitKey(0)
```

Tutorial nine (9)

image arithmetic opencv

```
#import numpy as np
import argparse
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i","--image",type=str,default="image path",help="path to the
input image")
args = vars(ap.parse_args())

""" images are Numpy arrays stored as unsigned 8-bit integers (unit8) with values
in the range [0,255]; when using the add/sutract functions in opencv these values
will be clipped to this range even if they fall outside the range [o,255] after
applying the  operation"""
added = cv2.add(np.uint8([200]),np.uint8([100]))
subtracted = cv2.subtract(np.uint8([50]),np.uint8([100]))
print ("max of 255: {}".format(added))
print ("min of 0: {}".format(subtracted))

""" using Numpy arithmetic operations (rather that opencv operation) will result
in a modulo ("wrap around") instead of being clipped to the range [0,255]"""

added = np.uint8([200]) + np.uint8([100])
subtracted = np.uint8([50]) - np.uint8([100])
```

```
print ("wrap around: {}".format(added))
print("wrap around: {}".format(subtracted))

# loading the original input image and display
image = cv2.imread(args["image"])
cv2.imshow("Original",image)

""" increasing the pixel intensities in our input image by 100 is accomplished by
constructing a numpy array that has the same dimensions as our input image filling
it with ones multiply i bay 100 and then adding the input image and the matrix
together """

M = np.ones(image.shape, dtype="uint8")*100
added = cv2.add(image,M)
cv2.imshow("Lighter",added)

similarly we can subtract 50 from all pixels in our image and make it darker
M = np.ones(image.shape, dtype="uint8") *50
subtracted = cv2.subtract(image,M)
cv2.imshow("Darker",subtracted)
cv2.waitKey(0)
```

tutorial 10

opencv bitwise opartion and or xor and NOT

```
# import numpy as np
import cv2



# draw a rectangle
rectangle = np.zeros((300,300),dtype="uint8")
cv2.rectangle(rectangle,(25,25),(275,275),255,-1)
cv2.imshow("Rectangle",rectangle)



# draw a circle
circle = np.zeros((300,300),dtype="uint8")
cv2.circle(circle,(150,150),150,255,-1)
cv2.imshow("Circle",circle)

""" a bitwise "AND" is only True when both inputs have a value that is "ON" -- in
this case the cv2.bitwise_and function examines every pixel in the rectangle and
circle if BOTH pixels have a value greater than zero then the pixel is turned ON
(ie 255) in the output image otherwise the output value is set to OFF (ie 0) """

bitwiseAnd = cv2.bitwise_and(rectangle,circle)
cv2.imshow("AND",bitwiseAnd)
cv2.waitKey(0)

""" a bitwise OR examines every pixel in the two inputs and if EITHER pixel in the
```

```
rectangle or circle is greater than zero then the output pixel has a value of 255,
otherwise it is 0 """

bitwiseOr = cv2.bitwise_or(rectangle,circle)
cv2.imshow("OR",bitwiseOr)
cv2.waitKey(0)

""" the bitwise XOR is identical to the OR function with one exception both the
rectangle and circle are not allowed to BOTH have values greater than 0 (only one
can be 0) """

bitwiseXor = cv2.bitwise_xor(rectangle,circle)
cv2.imshow("XOR",bitwiseXor)
cv2.waitKey(0)

""" finally the bitwise NOT inverts the values of the pixels; pixels with a value
od 255 become 0, and pixels with a value of 0 become 255 """

bitwiseNot = cv2.bitwise_not(circle)
cv2.imshow("NOT",bitwiseNot)
cv2.waitKey(0)
```

Tutorial 11 eleven

masking

```
# import the necessary packages
import numpy as np
import argparse
import cv2

# constract the argument parser and parse the arguments
ap =argparse.ArgumentParser()
ap.add_argument("-i","--image", type=str,default="image name",help="path tho the
image")
args = vars(ap.parse_args())

# load the original input image
image = cv2.imread(args["image"])
cv2.imshow("Original",image)

""" a mask is the same size as our image but has only two pixel values 0 and 255
pixels with a value of 0 (background ) are ignored in the original image while
mask pixel with a value of 255 (foreground) are allowed to be kept """

mask = np.zeros(image.shape(:2),dtype="uint8")
cv2.rectangle(mask,(0,90),(290,450),255,-1)
cv2.imshow("Rectangular Mask",mask)

#apply mask -- notice how only the person in the image is cropped out
masked = cv2.bitwise_and(image,image,mask = mask)
```

```
cv2.imshow("Mask applied to image",masked)
cv2.waitKey(0)

""" now let make a circular mask with a radius of 100 pixels and apply the mask
again """
mask = np.zeros(image.shape[:2],dtype="uint8")
cv2.circle(mask, (145,200)100,255,-1)
masked = cv2.bitwise_and(image,image,maks=mask)

cv2.imshow("circle Mask",mask)
cv2.imshow("Mask applied to image",masked)
cv2.waitKey(0)
```

Tutorial 12 twelve

splitting and merging channels

```
# import the necessary packages
import numpy as np
import argparse
import cv2

# constract the argument parser and parse the arguments
ap =argparse.ArgumentParser()
ap.add_argument("-i","--image", type=str,default="image name",help="path tho the
image")
args = vars(ap.parse_args())

""" load the input image and grab each channel --note how OpenCV represents images
as numpy arrays with channels in Blue, Green and Red odering rather than red green
blue """

image = cv2.imread(args["image"])
(B,G,R)=cv2.split(image)

# show each channel individually
cv2.imshow ("Red",R)
cv2.imshow ("Green", G)
cv2.imshow ("Blue",B)
cv2.waitKey(0)

# merg the image back together again
merged = cv2.merge([B,G,R])
cv2.imshow("Merged",merged)
cv2.waitKey(0)
cv2.destroyAllWindows()

# visualize each cahnnel in color
zeros =np.zeros(image.shape[:2],dtype="uint8")
cv2.imshow("Red",cv2.merge([zeros,zeros,R]))
cv2.imshow("Green",cv2.merge([zeros,G,zeros]))
```

```
cv2.imshow("Blue",cv2.merge([B,zeros,zeros]))
cv2.waitKey(0)
```