Introduction to OpenCV

Install

- Install Numpy, opencv, matplotlib
  - conda install numpy matplotlib
  - conda install -c menpo opencv
  - pip install numpy opencv-python==3.4.2.16 opencv-contrib-python==3.4.2.16 matplotlib
- Install jupyter notebook
  - conda install -c conda-forge jupyterlab
  - pip install jupyterlab
- Open jupyter notebook in you desired folder
  - jupyter lab

Getting Started

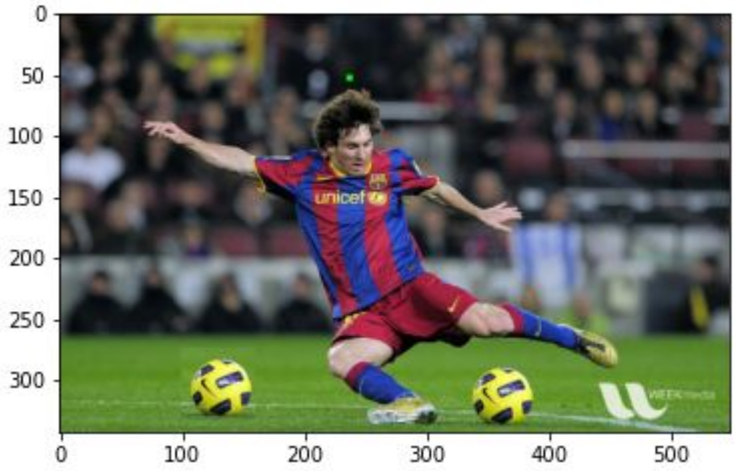
- Import numpy, opencv, and matplotlib.pyplot
- Print their versions
- Using cv2.imread, read one of the images
- Make sure the image is opened by using (is None)
- Print the image type, the shape, and a small portion of the image
- Using matplotlib.pyplot.imshow, show the image

Operations on images

- Create a black image and try out all of these commands
  - cv2.line(image, startPoint, endPoint, rgb, thinkness)
  - cv2.rectangle(image, topLeft, bottomRight, rgb, thinkness)
  - cv2.circle(image, center, radius, rgb, thinkness)
  - cv2.ellipse(image, center, axes, angle, startAngle, endAngle, rgb, thinkness)
  - cv2.polylines(image, points, isClosed, rgb, thinkness, lineType, shift)
  - cv2.putText(image, text, bottomLeft, fontType, fontScale, rgb, thinkness, lineType)
- Show the final image

Modify pixels

- Load an image
- Change the color of a 5 by 5 square to green
- Copy a 60 by 60 square from one location to another
- Show the changed image

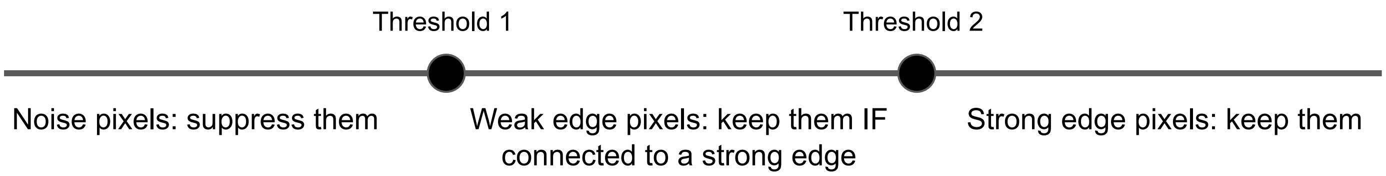


Edge detection - Sobel

- Load an image
- Using cv2.Sobel(src, ddepth, dx, dy, ksize=3, scale=1.0),
  - src: input image
  - ddepth: output image depth
  - dx: order of the derivative x
  - dy: order of the derivative y
  - ksize: size of the extended Sobel kernel; it must be 1, 3, 5, or 7
  - scale: optional scale factor for the computed derivative values
- Apply sobel filter in x direction to the image
- Apply sobel filter in y direction to the image
- Find the intensity
- Show the result (use cmap='gray')

Edge detection - Canny

- Load an image
- Using cv2.Canny(image, threshold1, threshold2, apatureSize=3, L2gradient=False)
    - image: 8-bit grayscale input image
    - threshold1/threshold2: thresholds for the hysteresis procedure
    - apertureSize: aperture size for the Sobel() operator
    - L2gradient: A flag. True to use $L2$ -norm of gradients. $L1$ -norm for False
- Apply canny edge detector with a few different thresholds
- Show the result (use cmap='gray')



Feature detection

- Read an image
- Create a SIFT feature detector with cv2.xfeatures2d.SIFT_create()
- Detect features using detect(image, section) on your detector
  - If its named st, it would be: st.detect()
  - 2nd pos argument is a mask indicating a part of image to be searched in, you don’t need to use it, just put None
- Draw the detected keypoints using cv2.drawKeypoints(image, keyPoints, section)

Feature detection - Continued

- Extract the SIFT descriptor using sift.compute(image, keyPoints)
- You can detect features AND extract descriptor at the same time using sift.detectAndCompute(image, section)

Feature matching - Try 1

- Read 2 images
  - 'images/box.png', 'images/box_in_scene.png'
- Detect features and extract descriptors for both
- Create a bruteforce matcher using cv2.BFMatcher(cv2.NORM_L2)
- Use knnMatch(des1, des2, k) on your brute force matcher to get matches
  - This will find k matches for each keypoint in des1
  - Set k to 2 for now
- We should only keep good matches
  - One way is to do a ratio test: if distance of best match/distance of second best match < threshold => keep the best match
  - Use 0.75 threshold for now
- Use cv2.drawMatches(img1, kp1, img2, kp2, matches, None) to draw your matches

Feature matching - Try 2

- Read 2 images
  - 'images/box.png', 'images/box_in_scene.png'
- Detect features and extract descriptors for both
- Normalize the descriptors
- Compute Hellinger Distance for all pairs of descriptors in img1 and img2
  - $\sqrt{1 - \sqrt{des1.des2}}$
  - Try to vectorize this
- Use the ratio test to keep good matches
  - You can match 2 features yourself by using cv2.DMatch(queryIdx, trainIdx, imgIdx, distance)
  - For example match = cv2.DMatch(queryIdx, trainIdx, imgIdx, distance)
- Use cv2.drawMatches(img1, kp1, img2, kp2, matches, None) to draw your matches

Face detection

- Using cv2.CascadeClassifier(), create a cascade classifier with the file 'detect/haarcascade_frontalface_alt.xml'
  - https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_alt.xml
  - https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf
  - This file has a cascade of simple face features
- Write a function that takes an image path
  - Convert it to grayscale
  - Use detectMultiscale on your cascade classifier to detect faces
    - scaleFactor: How much the image size is reduced at each image scale=1.2
    - minNeighbors: How many neighbors each candidate rectangle should have to retain it=5
    - minSize: Minimum possible object size=(30,30)
  - Draw a rectangle around each face
  - Show the image
- Run you function on a couple of images that have faces