Journal Report 4

9/21/19-9/30/19
Avyuk Dixit
Computer Systems Research Lab
Period 4, White

Daily Log

Monday September 23

Found a solution to serialization problem. JSON was technically working, but what was taking long was the conversion process from a numpy array to python lists - JSON can't serialize numpy arrays. I thought about using cPickle, but couldn't find it or install it using pip for some reason. I also explored msgpack and hdf5, but in the end, I settled for using np.save, which is not necessarily as fast, but should be good enough for now. Naturally, it works well in preserving the content of the numpy array I'm dumping, though it does take a fairly long amount of time to save. Also, instead of saving after every video, I changed it to save after every 100 videos. Though this means it takes more time to save at certain intervals, I think it helps to keep the code going and makes sure things don't get stuck. Tried to run code through entire dataset, but didn't finish because I ran out of time – will make sure to do that at home.

Tuesday September 24

Downloaded OpenPose model and converted from Caffe to Keras architecture, so that it could be run without a GPU sometimes and modified from my laptop. Went through individual lines of code in OpenPose model to better understand what statistical representation poses could be modelled by. Model operates by first generating a heat map of individual joints using a pre-trained CNN, layering those heat maps to better identify where exactly joints are positioned in relation to each other, and connecting joints to create limbs using part affinity fields. These are basically vector fields that point in the direction where the change in pixel value away from the joint is smallest.
Ran the code on a few examples, but didn't have the time to explore how specific components in the code - different lists, dictionaries, etc. - interacted and could help create the best statistical representation. Will do this next class.

Thursday September 26

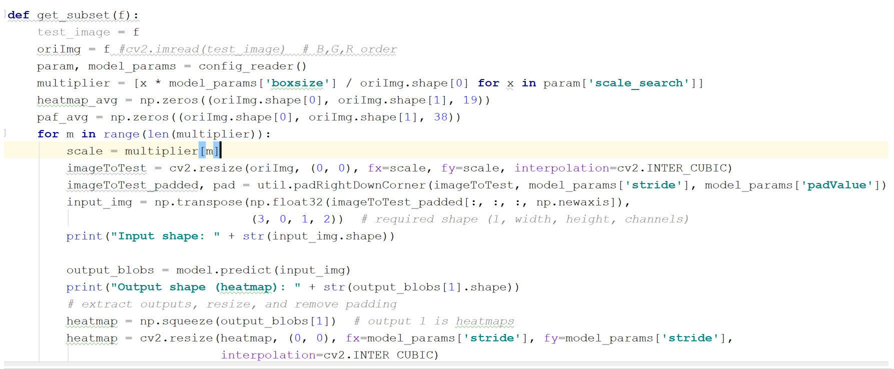
I was absent on Thursday because I was sick.

Timeline

Date	Goal	Met
Today minus 2 weeks	Run pre-processing code to decompose surveillance feed into individual frames marked as violent or non-violent through entire dataset.	A number of memory related errors made it so that I was not able to meet this goal.
Today minus 1 week	Review OpenPose code and determine best statistical representation for poses.	Was not able to achieve this - still focused on memory issues from last week.
Today	Review OpenPose code and determine best statistical representation for poses.	In progress - looking in depth at how OpenPose model works currently, and should have this done soon. Also fixed memory errors from previous week, though I need to still run the code through everything in the dataset. The code is working though, so it's just a question of running everything for a few hours or during the night maybe.
Today plus 1 week	Run OpenPose code through dataset of frames marked as violent or non-violent.	Probably need to push this back a little bit now.
Today plus 2 weeks	Write code that loads new dataset, preprocesses it, and gets it ready as input data for the neural network	

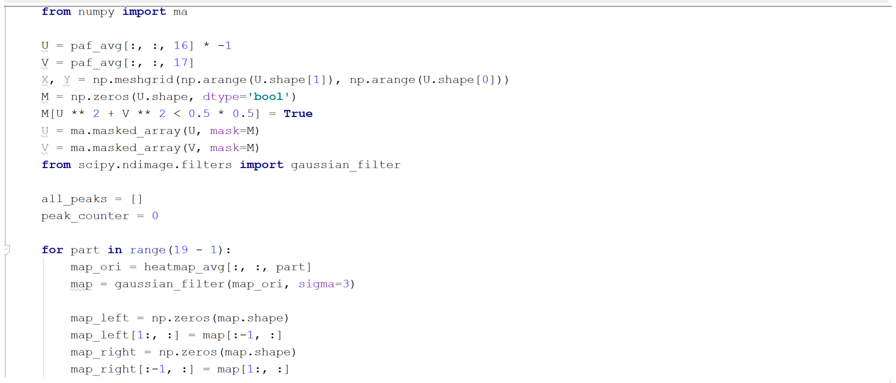
Reflection

Good thing I found np.save. It seems to work well and I should've thought of it earlier, considering all of my work had been in terms of numpy arrays and matrices. I'm still unsure how long it will take for the save method to run for a large number of matrices, but I hope its proportional to the amount of time it takes for a smaller number. Ideally, I'd expect it to take about 2 minutes total per 100 videos, which means running everything through 800 videos should probably take around 20 minutes, including processing the frames themselves.
Working through the details of the OpenPose model will be quite a task. A lot of their code is fairly dense and I've attached some snippets of it below. I think even if I don't completely understand it, I could do some sort of backwards analysis where I measure the poses from the images that the OpenPose model writes on. Really all I need is some account of where each joint is relative to the other joints, and which joints are attached to each other.



```
def get_subset(f):
    test_image = f
    oriImg = f #cv2.imread(test_image)  # B,G,R order
    param, model_params = config_reader()
    multiplier = [x * model_params['boxsize'] / oriImg.shape[0] for x in param['scale_search']]
    heatmap_avg = np.zeros((oriImg.shape[0], oriImg.shape[1], 19))
    paf_avg = np.zeros((oriImg.shape[0], oriImg.shape[1], 38))
    for m in range(len(multiplier)):
        scale = multiplier[m]
        imageToTest = cv2.resize(oriImg, (0, 0), fx=scale, fy=scale, interpolation=cv2.INTER_CUBIC)
        imageToTest_padded, pad = util.padRightDownCorner(imageToTest, model_params['stride'], model_params['padValue'])
        input_img = np.transpose(np.float32(imageToTest_padded[:, :, :, np.newaxis]),
                                 (3, 0, 1, 2))  # required shape (1, width, height, channels)
        print("Input shape: " + str(input_img.shape))

        output_blobs = model.predict(input_img)
        print("Output shape (heatmap): " + str(output_blobs[1].shape))
        # extract outputs, resize, and remove padding
        heatmap = np.squeeze(output_blobs[1])  # output 1 is heatmaps
        heatmap = cv2.resize(heatmap, (0, 0), fx=model_params['stride'], fy=model_params['stride'],
                             interpolation=cv2.INTER_CUBIC)
```



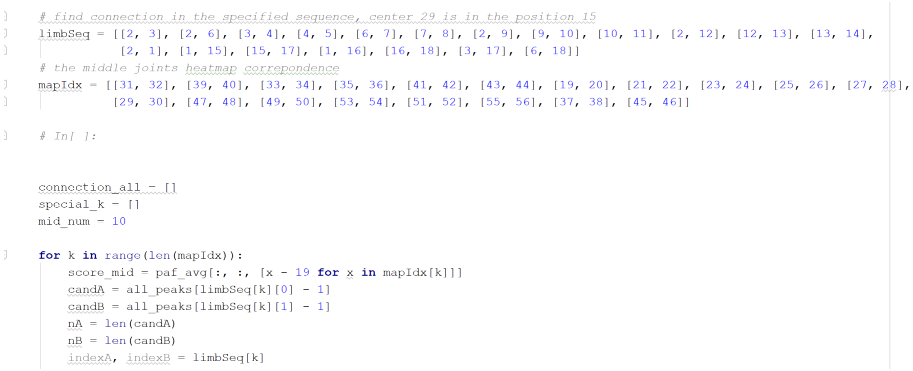
```
    from numpy import ma

    U = paf_avg[:, :, 16] * -1
    V = paf_avg[:, :, 17]
    X, Y = np.meshgrid(np.arange(U.shape[1]), np.arange(U.shape[0]))
    M = np.zeros(U.shape, dtype='bool')
    M[U ** 2 + V ** 2 < 0.5 * 0.5] = True
    U = ma.masked_array(U, mask=M)
    V = ma.masked_array(V, mask=M)
    from scipy.ndimage.filters import gaussian_filter

    all_peaks = []
    peak_counter = 0

    for part in range(19 - 1):
        map_ori = heatmap_avg[:, :, part]
        map = gaussian_filter(map_ori, sigma=3)

        map_left = np.zeros(map.shape)
        map_left[1:, :] = map[:-1, :]
        map_right = np.zeros(map.shape)
        map_right[:-1, :] = map[1:, :]
```



```
    # find connection in the specified sequence, center 29 is in the position 15
    limbSeq = [[2, 3], [2, 6], [3, 4], [4, 5], [6, 7], [7, 8], [2, 9], [9, 10], [10, 11], [2, 12], [12, 13], [13, 14],
               [2, 1], [1, 15], [15, 17], [1, 16], [16, 18], [3, 17], [6, 18]]
    # the middle joints heatmap correpondence
    mapIdx = [[31, 32], [39, 40], [33, 34], [35, 36], [41, 42], [43, 44], [19, 20], [21, 22], [23, 24], [25, 26], [27, 28],
              [29, 30], [47, 48], [49, 50], [53, 54], [51, 52], [55, 56], [37, 38], [45, 46]]

    # In[ ]:


    connection_all = []
    special_k = []
    mid_num = 10

    for k in range(len(mapIdx)):
        score_mid = paf_avg[:, :, [x - 19 for x in mapIdx[k]]]
        candA = all_peaks[limbSeq[k][0] - 1]
        candB = all_peaks[limbSeq[k][1] - 1]
        nA = len(candA)
        nB = len(candB)
        indexA, indexB = limbSeq[k]
```