Chapter 1

Connecting Tubes

1.1 Description

After getting TOIs for each video segment, it is time to connect them. That's because most actions in videos lasts more that 16 frames. This means that, in overlaping video clips, there will be consequentive TOIs that represent the entire action. So, it is essential to create an algorithm for finding and connecting these TOIs.

1.1.1 First approach: combine overlap and actioness

Our algorithm is inspired by [**?**], which calculates all possible sequences of ToIs. In order find the best candidates, it uses a score which tells us how likely a sequence of TOIs is to contain an action. This score is a combination of 2 metrics:

Actioness, which is the TOI's possibility to contain an action. This score is produced by TPN's scoring layers.

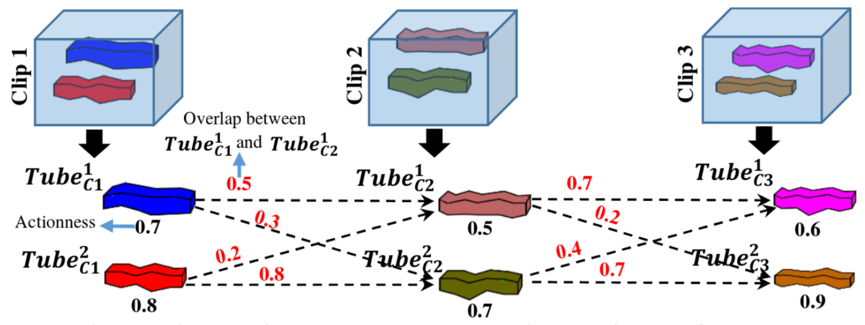
TOIs' overlapping, which is the IoU of the last frames of the first TOI and the first frames of the second TOI.

The above scoring policy can be described by the following formula:

$$S = \frac{1}{m}\sum_{i=1}^{m} Actioness_i + \frac{1}{m-1}\sum_{j=1}^{m-1} Overlap_{j,j+1}$$

For every possible combination of TOIs we calculate their score as show in figure 1.1.

The above approach, however, needs too much memory for all needed calculations, so a memory usage problem is appeared. The reason is, for every new video segments we propose *k TOIs* (16 during training and 150 during validation). As a result, for a small video seperated in **10 segments**, we need



$S1 = Tube^1_{C1} \rightarrow Tube^1_{C2} \rightarrow Tube^1_{C3} = 1.8/3 + 1.2/2$ $\quad S2 = Tube^1_{C1} \rightarrow Tube^1_{C2} \rightarrow Tube^2_{C3} = 2.1/3 + 0.7/2$

$S3 = Tube^1_{C1} \rightarrow Tube^2_{C2} \rightarrow Tube^1_{C3} = 2.0/3 + 0.7/2$ $\quad S4 = Tube^1_{C1} \rightarrow Tube^2_{C2} \rightarrow Tube^2_{C3} = 2.3/3 + 1.0/2$

$S5 = Tube^2_{C1} \rightarrow Tube^1_{C2} \rightarrow Tube^1_{C3} = 1.9/3 + 0.9/2$ $\quad S6 = Tube^2_{C1} \rightarrow Tube^1_{C2} \rightarrow Tube^2_{C3} = 2.2/3 + 0.4/2$

$S7 = Tube^2_{C1} \rightarrow Tube^2_{C2} \rightarrow Tube^1_{C3} = 2.1/3 + 1.2/2$ $\quad S8 = Tube^2_{C1} \rightarrow Tube^2_{C2} \rightarrow Tube^2_{C3} = 2.4/3 + 1.5/2$

Figure 1.1: An example of calculating connection score for 3 random TOIs

to calculate **150^{10} scores** during validation stage. This causes our system to overload and it takes too much time to process just one video.

In order to deal with this problem, we create a greedy algorithm in order to find the candidates tubes. Inituitively, this algorithm after a new video segment keeps tubes with score higher than a threshold, and deletes the rest. So, we don't need to calculate combinations with very low score. We wrote code for calculating tubes' scores in CUDA language, which has the ability to parallel process the same code using different data. Our algorithm is described below:

1. Firstly, initialize empty lists for the final tubes, final tubes' duratio, their scores, active tubes, their corrensponding duration, active tubes' overlapping sum and actioness sum where:

    - Final tubes list contains all tubes which are the most likely to contain an action, and their score list contains their corresponding scores. We refer to each tube by its index which is related a tensor, in which we saved all the ToIs proposed from TPN for each video segment.
    - Active tubes list contains all tubes that will be match with the new TOIs. Their overlapping sum list and actioness sum list contain their sums in order to avoid calculating then for each loop.

    Also, we initialize threshold equal to 0.5 .

2. For the first video segment, we add all the TOIs to both active tubes and final tubes. Their scores are only their actioness because there are no tubes for calculating their overlapping score. So, we set their overlaping sum equal to 0.

3. For each next video, after getting the proposed ToIs, firstly we calculate their overlapping score with each active tube. Then, we empty active

tubes, active tubes' duration, overlapping sum and actioness score lists. For each new tube that has score higher than the threshold we add both to final tubes and to active tubes, and we increase their duration.

4. If the number of active tubes is higher than a threshold, we set the threshold equal to the score of the 100th higher score. On top of that, we update the final tubes list, removing all tubes that have score lower than the threshold.

5. After that, we add in active tubes, the current video segment's proposed TOIs. Also their actioness scores in actioness sum list and zero values in cornensponding positions in overlaps sum list (such as in the 1st step).

6. We repeat the previous 3 steps until there is no video segment left.

7. Finally, as we mentioned before, we have a list which contains the indexes of the saved tubes. So, we modify them in order to have the final bounding boxes. However, 2 succeeding ToIs do not have extactly the same bounding boxes in the frames that overlap. For example, ToIs from the 1^{st} video segment start from frame 1 to frame 16. If we have video step equal with 8, it overlaps with the ToIs from the succeeding video segment in frames 8-16. In those frames, in final tube, we choose the area that contains both bounding boxes which is denoted as $(min(x_1, x'_1), min(y_1, y'_1), max(x_2, x'_2), max(y_2, y'_2))$ for bounding boxes (x_1, y_1, x_2, y_2) and (x'_1, y'_1, x'_2, y'_2).

1.2 Some results

In order to validate our algorithm, we firstly experiment in JHMDB dataset's videos in order to define the best overlapping policy and the video overlapping step. Again, we use recall as evaluation metrinc. A groundtruth action tube is considered to be found, as well as positive, if there is at least 1 video tube which overlaps with it over a predefined threshold, otherwise it . These thresholds are again 0.5, 0.4 and 0.3. We set TPN to return 30 ToIs per video segment. We chose to update threshold when active tubes are more than 500 and to keep the first 100 tubes as active. We did so, because, a big part of the code is performing in CPU. That's because, we use lists, which are very easy to handle for adding and removing elements. So, if we use bigger update limits, it takes much more time to process them.

sample duration = 16 At first we use as sample duration = 16 and video step = 8. As overlapping frames we count frames *(8...15)* so we have #8 frames. Also, we use only #4 frames with combinations *(8...11), (10...13) and (12...15)* and #2 frames with combinations *(8,9), (10,11), (12,13), and (14,15)*. The results are shown in Table 1.1 (in bold are the frames with which we calculate the overlap score).

combination	overlap thresh 0.3	0.4	0.5
0,1,...,**{8,...,15}** **{8,9,...,15}**,16,...,23	0.3172	0.4142	0.6418
0,1,...,**{8,...,11,}**...,14,15 **{8,...,11}**,12,...,22,23	0.3172	0.4142	0.6381
0,1,...,**{10,...,13,}**14,15, 8,9,**{10,...,13}**,14,...,22,23	0.3209	0.4179	0.6418
0,1,...,**{12,...,15,}** 8,9,...,**{12,...,15}**,16,...,23,	0.3284	0.4216	0.6381
0,1,...,**{8,...,11,}**,...,14,15, **{8,9,...,11,}**12,...,22,23	0.3172	0.4142	0.6381
0,1,...,**{10,...,13,}**14,15, **{10,...,13}**,14,...,22,23	0.3209	0.4179	0.6418
0,1,...,**{12,...,15}** 8,9,...,**{12,...,15}**,16,...	0.3284	0.4216	0.6381
0,1,...,**{8,9,}**,10,...,14,15, **{8,9,}**10,11,...,22,23	0.3134	0.4104	0.6381
0,1,...,**{10,11,}**,12,...,14,15, 8,9,**{10,11,}**12,...,22,23	0.3209	0.4216	0.6418
0,1,...,**{12,13,}**,14,15, 8,9,...,**{12,13,}**14,...,22,23	0.3246	0.4179	0.6418
0,1,...,13,**{14,15,}** 8,9,...,**{14,15,}**16,...,22,23	0.3321	0.4216	0.6306

Table 1.1: Recall results for step = 8

As we can from the above table, generally we get very bad performance and we got the best performance when we calculate the overlap between only 2 frames (either *14,15* or *12,13*). So, we thought that we should increase the video step because, probably, the connection algorithm is too strict into big movement variations during the video. As a results, we set video step = 12 which means that we have only 4 frames overlap. In this case, for #4 frames, we only have the combination *(12...15)*, for #2 frames we have *(12,13), (13,14) and (14,15)* as shown in Table 1.2.

combination	overlap thresh 0.3	0.4	0.5
0,1,...,11,**{12,...,15}** **{12,13,...,15}**,16,...,26,27	0.3769	0.4627	0.6828

0,1,...,**{12,13,}**,14,15, **{12,13,}**14,15,...,26,27	0.3694	0.4627	0.6903
0,1,...,12**{13,14,}**,15, 12,**{13,14,}**15,...,26,27	0.3843	0.4627	0.6828
0,1,...,12,13**{14,15,}** 12,13,**{14,15,}**16,...,26,27	0.3694	0.459	0.6828

Table 1.2: Recall results for step = 12

As we can see, recall performance is increase so that means that our assumption was correct. So again, we increase video step into 14, 15 and 16 frames and recall score is shown at Table **??**

combination	overlap thresh		
	0.3	0.4	0.5
0,1,...,13**{14,15}** **{14,15}**,16,...,28,29	0.3731	0.5336	0.6493
0,1,...,13,**{14,}**15, **{14,}**15,...,28,29	0.3694	0.5299	0.6455
0,1,...,14,**{15}** 14,**{15,}**16,...,28,29	0.3731	0.5187	0.6381
0,1,...,14,**{15}** **{15}**,16,...,30	0.3918	0.5187	0.6381
0,1,...,14,**{15}** **{16}**,17,...,31	0.4067	0.7313	0.8731

Table 1.3: Recall results for steps = 14, 15 and 16

The results show that we get the best recall performance when we have no overlapping steps and video step = 16 = sample duration. We try to improve more our results, using smaller duration because, as we saw from TPN recall performance, we get better results when we have sample duration = 8 or 4.

sample duration = 8 We wanted to confirm that we get the best results, when we have no overlapping frames and step = sample duration. So Table 1.4 shows recall performance for sample duration = 8 and video step = 4 and Table 1.5 for video steps = 6, 7 and 8.

combination	overlap thresh		
	0.3	0.4	0.5

0,1,2,3,13**{4,5,6,7}** **{4,5,6,7}**,8,9,10,11	0.2015	0.3582	0.5858
0,1,2,3,**{4,5,}**6,7 **{4,5,}**6,7,8,9,10,11	0.1978	0.3582	0.5933
0,1,2,3,4**{5,6,}**7 4,**{5,6,}**7,8,9,10,11	0.1978	0.3507	0.5821
0,1,2,3,4,5**{6,7}** 4,5,**{6,7,}**8,9,10,11	0.194	0.3433	0.585

Table 1.4: Recall results for step = 4

combination	overlap thresh		
	0.3	0.4	0.5
0,1,2,3,4,5**{6,7}** **{6,7}**,8,9,10,11,12,13	TODO	TODO	TODO
0,1,2,3,4,5,**{6,}**7 **{6,}**7,8,9,10,11,12,13	TODO	TODO	TODO
0,1,2,3,4,5,6,**{7}** 6,**{7}**8,9,10,11,12,13	TODO	TODO	TODO
0,1,2,3,4,5,6**{7}** **{7,}**8,9,10,11,12,13,14	TODO	TODO	TODO
0,1,2,3,4,5,6**{7}** **{8}**9,10,11,12,13,14,15	TODO	TODO	TODO

Table 1.5: Recall results for steps = 6, 7 and 8

1.2.1 UCF dataset

In previous steps, we tried to find the best overlap policy for our algorithm in JHMDB dataset. After that, it's time to apply our algorithm in UCF dataset using the best scoring overlap policy. We did some modifications in the code, in order to save memory and move most parts of the code to GPU. This happened by using tensors instead of lists for scores and most operations are, from now on, matrix operations. On top that, last step of the algorith, which is the modification from indices to actual action tubes was written in CUDA code so it takes place in GPU, too. So, we are now able to increase the number of tubes returned by TPN, the max number of active tubes before updating threshold and the max number of final tubes. In Table **??**

combination	TPN tubes	Final tubes	overlap thresh 0.5	0.4	0.3	MABO
0,1,...,6,**{7,}** **{8,}**9,...,14,15	30	500	0.2829	0.4395	0.5817	0.3501
		2000	0.3567	0.4996	0.6289	0.3815
		400	0.3749	0.5316	0.6487	0.3934
	100	500	0.2966	0.451	0.5947	0.356
		2000	0.3757	0.5163	0.6471	0.3902
		4000	0.3977	0.5506	0.6624	0.4029
0,1,...,14,**{15,}** **{16,}**17,18,...,23	30	500	0.362	0.5042	0.6243	0.3866
		2000	0.416	0.5468	0.6631	0.4108
		4000	0.4281	0.5589	0.6779	0.4182
	150	500	0.3589	0.4981	0.6198	0.3845
		2000	0.4129	0.5392	0.6563	0.4085
		4000	0.4266	0.5521	0.6722	0.4162

Table 1.6: Recall results for UCF dataset

combination	TPN tubes	Final tubes	overlap thresh 0.9	0.8	0.7	MABO
0,1,...,6,**{7,}** **{8,}**9,...,15	30	500	0.4464	0.581	0.6844	0.7787
		2000	0.635	0.7665	0.8403	0.8693
		4000	0.7034	0.8228	0.8875	0.8973
	100	500	0.454	0.5924	0.692	0.783
		2000	0.651	0.7696	0.8441	0.8734
		4000	0.7209	0.8312	0.8913	0.9026
0,1,...,14,**{15,}** **{16,}**17,18,...,23	30	500	0.6844	0.8327	0.9027	0.8992
		2000	0.7475	0.8684	0.9217	0.9175
		4000	0.7567	0.8745	0.9255	0.9211
	150	500	0.7498	0.8707	0.9171	0.9125
		2000	0.8243	0.911	0.9392	0.9342
		4000	0.8403	0.9179	0.9437	0.9389

Table 1.7: Temporal Recall results for UCF dataset

Adding NMS algorithm

combination	NMS thresh	PreNMS tubes	overlap thresh 0.9	0.8	0.7	MABO
0,1,...,6,**{7,}** **{8,}**9,...,15	0.7	10000	0.4464	0.581	0.6844	0.7787

		20000	0.635	0.7665	0.8403	0.8693
	0.5		TODO	TODO	TODO	TODO
0,1,...,14,**{15,}** **{16,}**17,18,...,23	0.7	10000	TODO	TODO	TODO	TODO
		20000	0.635	0.7665	0.8403	0.8693
	0.5		TODO	TODO	TODO	TODO

Table 1.8: Temporal Recall results for UCF dataset

1.2.2 Second approach: use progression and progress rate

As we saw before, our first connecting algorithm doesn't have very good recall results. So, we created another algorithm which is base in [**?**]. This algorithm introduces two 2 metrics according to [**?**]:

Progression, which describes the probability of a specific action being performed in the TOI. We add this factor because we have noticed that actioness is tolerant to false positives. Progression is mainly a rescoring mechanism for each class (as mentioned in [**?**])

Progress rate, which is defined as the progress proportion that each class has been perfomed.

So, each action tube is describes as a set of TOIs

$$T = \{\mathbf{t}_i^{(k)} | \mathbf{t}_i^{(k)} = (t_i^{(k)}, s_i^{(k)}, r_i^{(k)})\}_{i=1:n^{(k)}, k=1:K}$$

where $t_i^{(k)}$ contains TOI's spatiotemporal information, $s_i^{(k)}$ its confidence score and $r_i^{(k)}$ its progress rate.

In this approach, each class is handled seperately, so we discuss action tube generation for one class only. In order to link 2 TOIs, for a video with N video segments, the following steps are applied:

1. For the first video segment (k = 1), initialize an array with the M best scoring TOIs, which will be considered as active action tubes (AT). Correspondly, initialize an array with M progress rates and M confidence scores.

2. For k = 2:N, execute (a) to (c) steps:

   (a) Calculate overlaps between $AT^{(k)}$ and $TOIs^{(k)}$.

   (b) Connect all tubes which satisfy the following criterions:

      i. $overlapscore(at_i^{(k)}, t_j^{(k)}) < \theta, at \varepsilon AT^{(k)}, t \varepsilon TOIs^{(k)}$

      ii. $r(at_i^{(k)}) < r(t_j^{(k)})$ or $r(t_i^{(k)}) - r(at_i(k)) < \lambda$

   (c) For all new tubes update confidence score and progress rate as follows:

New cofidence score is the average score of all connected TOIs:

$$s_z^{(k+1)} = \frac{1}{n}\sum_{n=0}^{k} s_i^{(n)}$$

New progress rate is the highest progress rate:

$$r(at_z^{(k+1)} = max(r(at_i^{(k)}), r(t_j^{(k)}))$$

(d) Keep M best scoring action tubes as active tubes and keep K best scoring action tubes for classification.

This approach has the advantage that we don't need to perform classification again because we already know the class of each final tube. In order to validate our results, now, we calculate the recall only from the tubes which have the same class as the groundtruth tube. Again we considered as positive if there is a tube that overlaps with groudtruth over the predefined threshold.

combination		**overlap thresh**		
sample dur	step	0.3	0.4	0.5
8	6	TODO	TODO	TODO
8	7	TODO	TODO	TODO
8	8	0.3060	0.5672	0.6866
16	8	TODO	TODO	TODO
16	12	TODO	TODO	TODO
16	16	TODO	TODO	TODO

Table 1.9: Recall results for second approach with step = 8, 16 and their corresponding steps

(Pending Table...) As we can see from the table above, the results in recall are not very good either.

1.3 Third approach : use naive algorithm - only for JHMDB

As mention in first approach, [**?**] calculates all possible sequences of ToIs in order to the find the best candidates. We rethought about this approach and we concluded that it could be implement for JHMDB dataset if we reduce the number of proposed ToIs, produced by TPN, to 30 for each video clip. We exploited the fact that JHMDB dataset's videos are trimmed, so we do not need to look for action tubes starting in the second video clip which saves us a lot of memory. On top of that, we modified our code in order to be memory efficient at the most writing some parts in CUDA programming language, saving a lot of processing power, too.

So, after computing all possible combinations starting of the first video clip and ending in the last video clip, we keep only the **k-best scoring tubes (k = 500)** . In the followin table, we can see the recall results for sample durations = 8 and 16.

combination		**overlap thresh**		
sample dur	step	0.3	0.4	0.5
8	6	0.7873	0.8657	0.9366
8	7	TODO	TODO	TODO
8	8	0.7910	0.8806	0.9515
16	8	TODO	TODO	TODO
16	12	TODO	TODO	TODO
16	16	0.7910	0.8806	0.9478

Table 1.10: Recall results for second approach with

From the above table, we notice that sample duration = 8 is slightly better that the 16.