

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 consecutive TOIs that represent the entire action. So, it is essential to create an algorithm for finding and connecting these TOIs.

1.2 First approach: combine overlap and actionness

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:

Actionness, 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

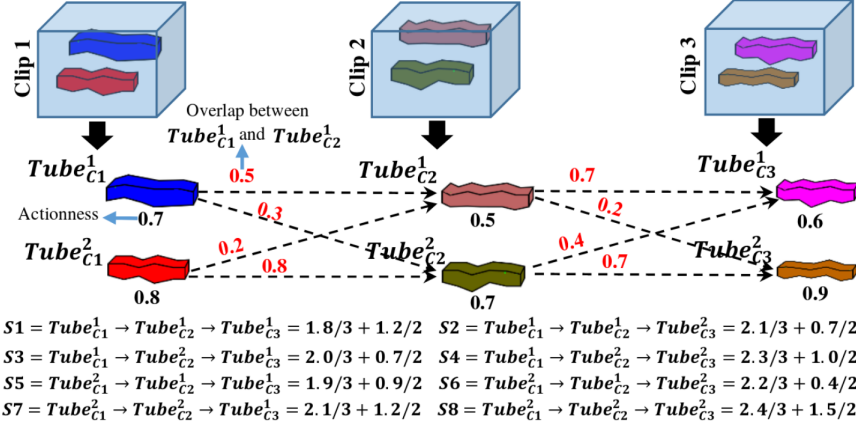


Figure 1.1: An example of calculating connection score for 3 random TOIs

new video segments we propose k *TOIs* (16 during training and 150 during validation). As a result, for a small video separated in **10 segments**, we need to calculate **150¹⁰ 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. Intuitively, 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' duration, their scores, active tubes, their corresponding duration, active tubes' overlapping sum and actionness 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 actionness 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 actionness because there are no tubes for calculating their overlapping score. So, we set their overlapping 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 actionness 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 actionness scores in actionness sum list and zero values in corrensponding 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 exactly the same bounding boxes in the frames that overlap. For example, ToIs from the 1st 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) .

1.2.1 JHMDB Dataset

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		
	0.5	0.4	0.3
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		
	0.5	0.4	0.3
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 1.3

combination	overlap thresh		
	0.5	0.4	0.3
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		
	0.5	0.4	0.3

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		
	0.5	0.4	0.3
0,1,2,3,4,5{6,7} {6,7},8,9,10,11,12,13	0.3134	0.7015	0.8619
0,1,2,3,4,5,{6},7 {6},7,8,9,10,11,12,13	0.3209	0.6679	0.847
0,1,2,3,4,5,6,{7} 6,{7},8,9,10,11,12,13	0.3172	0.6567	0.8507
0,1,2,3,4,5,6{7} {7},8,9,10,11,12,13,14	0.5597	0.7687	0.903
0,1,2,3,4,5,6{7} {8},9,10,11,12,13,14,15	0.653	0.8396	0.9179

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

According to Tables 1.4 and 1.5, it is clearly shown that, we achieve best results, for *step = sampleduration* and overlapping scores is calculated between the last box of the current tubes and the first box of next tubes.

1.2.2 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 algorithm, 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.

The first experiments we performed were related with the number of the final tubes, our network proposes alongside with TPN’s proposed tubes’ number. We experiment for cases, in which TPN proposes 30, 100 and 150, our final network proposes 500, 2000 and 4000 for sample durations equal with 8 and 16 frames. For sample duration equal with 8 we return 100 proposed action tubes because, when we tried to return 150 proposed ToIs, we got OutOfMemory error. Table 1.6 show the spatio-temporal recall and MABO performance of those approaches. Futhermore, Table 1.7 show their tempolar recall and MABO performance. We are interested in temporal performance, because UCF is consisted of untrimmed videos, unlike JHMDB which has only trimmed videos. So, we want to know how well our network is able to propose action tubes that overlap temporally with the groundtruth action tubes over a “big” threshold. For temporal localization, we don’t use 0.5, 0.4 and 0.3 overlapping threshold, but instead, we use 0.9, 0.8 and 0.7, because it is very important our network to be able to propose tubes that contain an action. In order to calculate the temporal overlap, we use IoU for 1D as described before.

combination	TPN tubes	Final tubes	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
		4000	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

According to Table 1.6, we achieve better recall and MABO performance when we set sample duration equal with 16. In all cases, recall performace of simulations with sample duration equal with 16 outweigh the corresponding with 8, with the difference varying from 2% to 8%. In addition, we get best recall and MABO performance when our system proposes 4000 tubes. As we can see, the ratio of good proposals increases about 5%-7% when we change number of proposed tubes from 500 to 2000. This ratio increases more when we double

combination	TPN tubes	Final tubes	overlap thresh			MABO
			0.9	0.8	0.7	
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

returned action tubes, from 2000 to 4000. However, this increase is only about 1%-2%, which make us rethink if this increase is worth to be performed. That's because, this modification increases memory usage, because of 4000 proposed action tubes, instead of 2000. Finally, Table 1.6 shows that, for sample duration = 8, changing the number of ToIs produced by TPN, slightly helps our network to achieve better results. This contribution is measured about 1%-2%. On the contrary, when we set sample duration equal with 16, it slightly reduces network's performance. Taking all the aforementioned results into account, we think that the most suitable choices for connection approaches are, for sample duration equal with 8, the one in which TPN returns 100 ToIs and our network proposes 4000 action tubes, and for sample duration equal with 16, the one in which, TPN returns 30 ToIs and the network 4000 action tubes.

Additionally, Table 1.7 shows some interesting facts, too. At first, it confirms that increasing the number of proposed action tubes, from 500 to 4000, increases recall and MABO performance. Also, we get better result when network has 16 frames as sample duration, too. However, unlike Table 1.6, Table 1.7 shows that when we increase TPN's number of proposed ToIs, it increases performances for both sample durations. For sample duration equal with 8, this increases results in Improving recall performances by 2% and MABO performance by 1% like spatio-temporal recall and MABO. For sample duration equal with 16, recall performance is increasing by about 8% and MABO by 1%-2%.

Taking both tables into consideration, we think that the best approach is TPN returning 30 proposed ToIs, network returning 4000 proposed action tubes and sample duration equals with 16. We didn't choose TPN returning 150 proposed ToIs because, based on MABO performances, they different only by 1%, difference which is insignificant.

Adding NMS algorithm

Previous section describe the performances of network’s proposals for variations in the returnig number of TPN’s ToIs, number of returned proposed aciton tubes and sample duration. For each situation, we choose the k-best scoring action tubes, without taking into account any relation between these aciton tubes, like their Spatio-temporal overlap. So, like TPN’s approach, we thought that we should apply nms algorithm before choosing k-best scoring tubes, in order to further improve spatio-temporal and temporal, recall and MABO performance. We experiment using again two sample durations, 16 and 8 frames per video segment, number or TPN’s returning tubes equal with 30 and number of final picked action tubes equal with 4000. NMS algorithm uses a threshold in order to choose if 2 action tubes overlap enough. We experiment setting this threshold equal with 0.7 and 0.8 and results are shown at Table 1.8 for Spatio-temporal perfomance and at Table 1.9 for temporal performance.

combination	NMS thresh	PreNMS tubes	overlap thresh			MABO
			0.9	0.8	0.7	
0,1,...,6,{ 7 ,} { 8 ,}9,...,15	0.7	20000	0.346	0.5202	0.657	0.3824685269
	0.8		0.3643	0.5392	0.6578	0.3904727407
	0.9		0.397	0.5574	0.6677	0.4031543642
0,1,...,14,{ 15 ,} { 16 ,}17,...,23	0.7	20000	0.3939	0.5559	0.6882	0.404689056
	0.8		0.4259	0.5764	0.6981	0.419487652
	0.9		0.4494	0.5856	0.7019	0.4302611039

Table 1.8: Spatio-temporal Recall results for UCF dataset

combination	NMS thresh	PreNMS tubes	overlap thresh			MABO
			0.9	0.8	0.7	
0,1,...,6,{ 7 ,} { 8 ,}9,...,15	0.7	20000	0.6281	0.8251	0.9027	0.8885141223
	0.8		0.7369	0.8616	0.9148	0.9106069806
	0.9		0.7787	0.8753	0.9209	0.9212593589
0,1,...,14,{ 15 ,} { 16 ,}17,...,23	0.7	20000	0.7452	0.8920	0.9361	0.920331595
	0.8		0.8160	0.9278	0.9506	0.93612757
	0.9		0.854	0.9346	0.9529	0.9434986107

Table 1.9: Temporal Recall results for UCF dataset

Comparing Table 1.8 with Table 1.6, we notice that NMS algorithm improves recall and MABO performance when NMS threshold is equal with 0.9. When we set it equal with 0.7 or 0.8, we get worse results. This happens, probably,

because nms algorithm removes some good proposals. Comparing these results with results presented at Tables 1.6 and 1.7 it becomes clear that using NMS algorithm is very usefull. That's because, even though we get the same number of proposed action tubes, these tubes are not very close spatiotemporally, so this makes proposed action tubes more likely to contain actual foreground action tubes.

Stop updating threshold

In previous approaches, scoring threshold was updated each time our algorithm gathered a significant number of "active" tubes in order not to add action tubes with score below this score. However, after serious cosideration we came with the conclusion that some times, the updated threshold leads to not detecting action tubes that start after some frames. That's because, until then, linking threshold may be too big that won't let new action tubes to be created. So, we came with the modification of not updating linking threshold but just filtering proposed tubes, by keeping k-best scoring each time their number is bigger that the a specific number. The rest algorithm remains the same. Tables 1.10 and 1.11 show spatio-temporal and temporal, recall and MABO performance respectively. We experiment for cases in which either we don't use NMS algorithm at all, either we set overlap threshold equal with 0.7 and 0.9 as shown below.

combination	NMS thresh	PreNMS tubes	overlap thresh			MABO
			0.9	0.8	0.7	
0,1,...,6,{7, 8,}9,...,15	-	20000	0.3779	0.5316	0.6471	0.393082961
	0.7		0.3483	0.5194	0.6471	0.3783524086
	0.9		0.416	0.5605	0.6722	0.4074053106
0,1,...,14,{15, 16,}17,...,23	-	20000	0.438	0.5635	0.6829	0.4231788
	0.7		0.4525	0.5848	0.7034	0.429747438
	0.9		0.3802	0.5133	0.6068	0.3862278851848662

Table 1.10: Spatio-temporal Recall results for UCF dataset

combination	NMS thresh	PreNMS tubes	overlap thresh			MABO
			0.9	0.8	0.7	
0,1,...,6,{7, 8,}9,...,15	-	20000	0.7087	0.8281	0.8913	0.899210587
	0.7		0.6586	0.854	0.9278	0.903373468
	0.9		0.8137	0.8973	0.9361	0.9333068498
0,1,...,14,{15, 16,}17,...,23	-	20000	0.8327	0.9156	0.9399	0.940143272
	0.7		0.8646	0.9369	0.9567	0.946701832
	0.9		0.6183	0.7696	0.8388	0.8628507037919737

Table 1.11: Temporal Recall results for UCF dataset

Comparing recall and MABO performances shown at Table 1.10 with those included at Tables 1.8 and 1.6, we deduce that for sample duration equal with 8 stopping updating linking threshold results in worse performance when we set nms threshold equal with 0.7 but it achieves the best performances for nms threshold equal with 0.9 . Furthermore, for sample duration equal with 16, we get, now, best performance performance when we set nms threshold equal with 0.7 and worse performance for nms threshold equal with 0.9 .

Soft-nms instead of nms

After widely experiment using NMS algorithm, we thought that we should try to use Soft-NMS algorithm, introduced by [?] and described in chapter 2. We implement our own soft-nms algorithm modifying it in order to calculate spatiotemporal overlapping scores, and not just spatial, like the one implemented by [?]. As mentioned before, instead of removing action tubes, Soft-NMS algorithm just reduces their score for those which overlap over a predefined threshold, We experiment for sample duration equal with 8 and thresholds equal with 0.7 and 0.9, because, our implementation ran out of memory for sample duration equal with 16. Recall and MABO performance are presented at Tables 1.12 and 1.13

combination	NMS thresh	PreNMS tubes	overlap thresh			MABO
			0.9	0.8	0.7	
0,1,...,6,{7,}	0.7	20000	0.3916	0.5384	0.6464	0.3964639
{8,}9,...,15	0.9		0.4023	0.5430	0.6502	0.398845313

Table 1.12: Spatio-temporal Recall results for UCF dataset using Soft-NMS

combination	NMS thresh	PreNMS tubes	overlap thresh			MABO
			0.9	0.8	0.7	
0,1,...,6,{7,}	0.7	20000	0.7521	0.8586	0.9110	0.915746097
{8,}9,...,15	0.9		0.7741	0.8768	0.9255	0.922677864

Table 1.13: Temporal Recall results for UCF dataset using SoftNMS

Taking results at Tables 1.12 and 1.13 into consideration, alongside with those at Tables 1.10 and 1.11 for sample duration equal with 8, we notice that using softNMS results in slightly better results. This happens when we set overlapping threshold equal with 0.9, otherwise, for overlapping threshold 0.7, we get worst performance. Despite the fact that softnms results in better recall

and MABO performance, our implementation is very slow, which means that for a 201-frame video, softNMS part lasts about 32 seconds on the contrary with standard NMS algorithm without updating linking threshold, in which this part last only 2 seconds. So, we choose to use standard NMS algorithm without updating linking threshold as an algorithm for removing overlapping action tubes.

1.3 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 actionness 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 performed.

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

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

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. $overlapscore(at_i^{(k)}, t_j^{(k)}) < \theta, at \in AT^{(k)}, t \in TOIs^{(k)}$
 - 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 confidence 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 groundtruth over the predefined threshold.

combination		overlap thresh		
sample dur	step	0.5	0.4	0.3
8	6	0.3284	0.5	0.6082
8	7	0.209	0.459	0.6119
8	8	0.3060	0.5672	0.6866
16	8	0.194	0.4366	0.7164
16	12	0.3358	0.5336	0.7537
16	16	0.2649	0.4664	0.709

Table 1.14: Recall results for second approach with step = 8, 16 and their corresponding steps

According to 1.14, we get best performance when we set sample duration equal with 16 and overlap step equal with 12. Comparing this performance with first approach, for both sample durations equal with 8 and 16, we notice that second approach falls short comparing to the first one.

1.4 Third approach : use naive algorithm - only for JHMDB

As mention in first approach, [?] calculates all possible sequences of ToIs in order to find the best candidates. We rethought about this approach and we concluded that it could be implemented 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)** . We run case for sample duration equal with 8 and 16 frames and we modify the video step each time. For sample duration = 8, we return only 15 ToIs and for sample duratio = 16, we return 30 because, if we return more, we get “out of memory error”. In the follown table, we can see the recall results.

combination		overlap thresh		
sample dur	step	0.5	0.4	0.3
8	6	0.7873	0.8657	0.9366
8	7	0.7836	0.8731	0.9366
8	8	0.7910	0.8806	0.9515
16	8	0.7873	0.8843	0.9291
16	12	0.7948	0.8881	0.9403
16	16	0.7985	0.8918	0.9515

Table 1.15: Recall results for second approach with

From the above table, firstly, we confirmed that overlap = sample duration gives us the best recall results. we notice that sample duration = 16 is slightly better than the 8. However, using sample 16 increases the memory usage even though it reduces the number of video segments. So for classification stage we will experiment using mostly sample duration = 8.