Chapter 1

Classification stage

1.1 Description

After getting all proposed tubes, it's time to do classification. As classifiers we use several approaches including a Linear Classifier, a Recursive Neural Network (RNN) Classifier, a Support Vector Machine (SVM) Classifier and a Multilayer perceptron (MLP).

The whole procedure of classification is consisted from the following steps:

1. Seperate video into small video clips. Feed TPN network those video clips and get as output k-proposed ToIs and their corresponding features for each video clip.

2. Connect the proposed ToIs in order to get video tubes which may contain an action.

3. For each candidate video tube, which is a sequence of ToIs, feed it into the classifier for verification.

The general structure of the whole network is depicted in figure 1.1, in which we can see the aforementioned steps if we follow the arrows.

In first steps of classification stage we refer only to JHMDB dataset because it has smaller number of video than UCF dataset which helped us save a lot of time and resources. That's because we performed most experiments only JHMDB and after we found the optimal situation, we implemented to UCF-dataset, too.

1.2 Preparing data and first classification results

For carrying out classification stage, we use, at first, a Linear classifier and a RNN classifier.

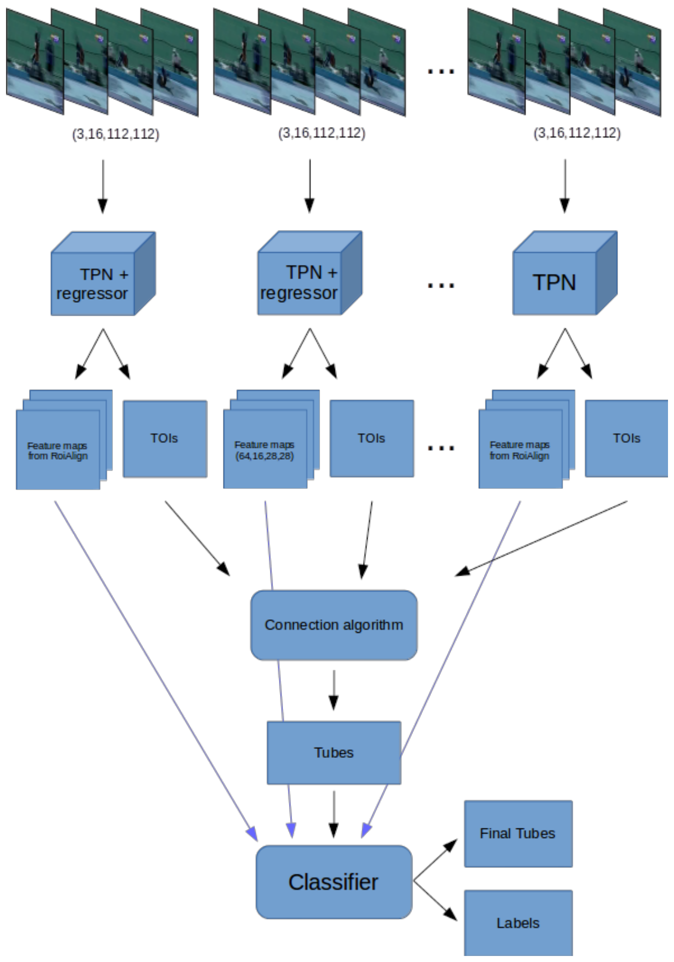

Figure 1.1: Structure of the whole network

Linear Classifier Linear classifier is a type of classifier which is able to discriminate objects and predict their class based on the value of a *linear combination* of object's feature values, which usually are presented in a feature vector. If the input feature vector to the classifier is a real vector $\vec{x}$, then the output score is :

$$y = f(\vec{w} \cdot \vec{x}) = f\left(\sum_j w_j x_i\right)$$

RNN Recurrent neural networks, or RNNs for short, are a type of neural network that was designed to learn from sequence data, such as sequences of observations over time, or a sequence of words in a sentence. RNN takes many input vectors to process them and output other vectors. It can be roughly pictured like in the Figure 1.2 below, imagining each rectangle has a vectorial depth and other special hidden quirks in the image below. For our case, we choose **many to one** approach, because we want only one prediction, at the end of the action tube.

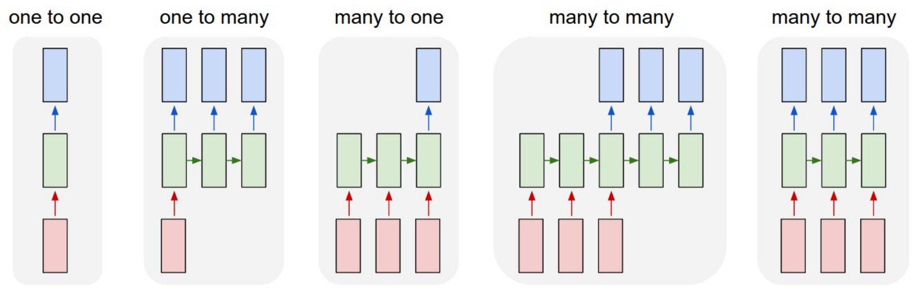

Figure 1.2: Types of RNN

Training In order to train our classifier, we have to execute the previous steps for each video. However, each video has different number of frames and reserves too much memory in the GPU. In order to deal with this situation, we give as input one video per GPU. So we can handle 4 videos simultaneously. This means that a regular training session takes too much time for just 1 epoch.

The solution we came with, is to precompute the features for both positive video tubes and negative video tubes and then to feed those features to our classifier for training it in order to discriminate classes. This solution includes the following steps:

1. At first, we extract only groundtruth video tubes' features and the double number of background video tubes. We chose this ratio between positive and negative tubes inspired by [**?**], in which it has 0.25 ratio between foreground and background rois and chooses 128 roi in total. Respectively, we chose a little bigger ratio because we have only 1 groundtruth video tube in each video. So, for each video we got 3 video tubes in total, 1 for positive and 2 for background. We considered background tubes those whose overlap scores with groundtruth tubes are ≥ 0.1 and ≤ 0.3. Of course, we use a pre-trained TPN in order to get those action tubes.

2. After extracting those features, we trained both Linear and RNN classifiers. The Linear classifier needs a fixed input size, so we used a pooling function in the dimension of the videos. So, at first we had a feature map of *3,512,16* dimensions and then we get as output a feature maps of *512,16* dimensions. We used both max and avg pooling as shown at Table 1.1. For the RNN classifier, we do not use any pooling function before feeding it.

In order to train our classifiers, we use Cross-Entropy Loss as training loss function.

Validation Validation stage includes using both pre-trained TPN and classifier. So, for each video, we get classification scores for proposed action tubes. Most approaches usually consider a confidence score for considering an action tube as foreground. However, we don't use any confidence score. On the contrary, because we know that JHMDB has trimmed videos with only 1 performed action, we just consider the best-scoring tube as our prediction.

Classifier	**Pooling**	**mAP**		
		0.5	0.4	0.3
Linear	mean	14.18	19.81	20.02
	max	13.67	16.46	17.02
RNN	-	11.3	14.14	14.84

Table 1.1

Table 1.1 shows first classification results, which are not very good. The only useful deduction that we can come with, using above results is that, avg pooling method outclass max pooling. So, for all the rest classifications using Linear classifier, we use avg pooling before classification stage.

1.3 Support Vector Machine (SVM)

SVMs are classifiers defined by a separating hyperplane between trained data in a N-dimensional space. The main advantage of using a SVM is that can get very good classification results when we have few data available.

The use of SVM is inspired from [?] and it is trained using hard negative mining. This means that we have 1 classifier per class which has only 2 labels, positive and negative. We mark as positive the feature maps of the groundtruth action, and as negative groundtruth actions from other classes, and feature maps from background classes. As we know, SVM is driven by small number of examples near decision boundary. Our goal is to find a set of negatives that are the closest to the seperating hyperplane. So in each iteration, we update this set of negatives adding those which our SVM didn't perform very well. Each SVM is trained independently.

SVM code is take from Microsoft's Azure github page in which there is an implementation of Fast RCNN using a SVM classifier. We didn't modify its parameters which means that it has a linear kernelr, uses L2-norm as penalty and L1-norm as loss during training. Also, we consider as hard-negatives the tubes that got score > -1.0 during classification.

This whole process makes the choise of the negatives a crutial factor. In order to find the best policy, we came with 5 different cases to consider as negatives:

1. Negatives are other classes's positives and all the background tubes
2. Negatives are only all the background videos
3. Negatives are only other classes's positives
4. Negatives are other classes's positives and background tubes taken only from videos that contain a positive tube
5. Negatives are only background tubes taken from videos that contain a positive tube

On top of that, we use 2 pooling functions in order to have a fixed input size.

In the next tables, we show our architecture's mAP performance when we follow each one of the above policies. Also, we experimented for 2 feature maps, *(64,8,7,7)* and *(256,8,7,7)* where 8 equals with the sample duration. Both feature maps were extracted by using 3D RoiAlign procedure from feature maps with dimensions *(64,8,28,28)* and *(256,8,7,7)* respectively (in the second case, we just add zeros in the feature map outsize from the bounding boxes for each frame). Table 1.2 contains the first classification results. At first column we have the dimensions of feature maps before pooling fuction, where k = 1,2,..5 . At second column we have feature maps' dimensions after pooling, and at the next 2 column, the type of pooling function and the policy we followed. Finally in the last 3 collumns we have the mAP performance when we have threshold equal with 0.3, 0.4 and 0.5 respectively. During validation, we keep only the best scoring tube because we know that we have only 1 action per video.

Dimensions		**Pooling**	**Type**	**mAP precision**		
before	after			0.5	0.4	0.3
(k,64,8,7,7)	(1,64,8,7,7)	mean	1	3.16	4.2	4.4
			2	2.29	2.68	2.86
			3	1.63	3.16	4
			4	2.42	4.83	5.46
			5	0.89	1.12	1.21
(k,64,8,7,7)	(1,64,8,7,7)	max	1	1.11	2.35	2.71
			2	2.31	2.62	2.64
			3	1.11	2.35	2.71
			4	1.41	2.76	3.84
			5	0.33	0.51	0.58
(k,256,8,7,7)	(1,256,8,7,7)	mean	1	11.41	11.73	11.73
			2	10.35	10.92	11.89
			3	8.93	9.64	9.94
			4	12.1	13.04	13.04
			5	5.92	6.92	7.79

(k,256,8,7,7)	(1,256,8,7,7)	max	1	22.07	24.4	25.77
			2	14.07	16.56	17.74
			3	14.22	18.94	21.6
			4	21.05	24.63	25.93
			5	11.6	13.92	15.81

Table 1.2: Our architecture's performance using 5 different policies and 2 different feature maps while pooling in tubes' dimension. With bold is the best scoring case

From the above results we notice that features map with dimension (256,8,7,7) outperform in all cases, both for mean and max pooling and for all the policies. Also, we can see that max pooling outperforms mean pooling in all cases, too. Last but not least, we notice that policies 2, 3 and 5 give us the worst results which means that svm needs both data from other classes positives and from background tubes.

1.3.1 Modifying 3D Roi Align

As we mentioned before, we extract from each tube its activation maps using 3D Roi Align procedure and we set equal to zero the pixels outside of bounding boxes for each frame. We came with the idea that the enviroment surrounding the actor sometimes help us determine the class of the action which is performed. This is base in the idea that 3D Convolutional Networks use the whole scene in order to classify the action that is performed. We thought to extend a little each bounding box both in width and height. So, during Roi Align procedure, after resizing the bounding box into the desired spatial scale (in our case 1/16 because original sample size = 112 and resized sample size = 7) we increase by 1 both width and height. According to that if we have a resized bounding box (x_1, y_1, x_2, y_2) our new bounding box becomes $(max(0, x_1 - 0.5), max(0, y_1 - 0.5), min(7, x_2 + 0.5), min(7, y_2 + 0.5))$ (we use *min* and *max* functions in order to avoid exceeding feature maps' limits). We just experiment in policies 1 and 4 for both (256,8,7,7) and (64,8,7,7) feature maps as show in Table 1.3

Dimensions		**Pooling**	**Type**	**mAP precision**		
before	after			0.3	0.4	0.5
(k,64,8,7,7)	(1,64,8,7,7)	mean	1	9.75	11.92	13.34
			4	5.74	6.62	7.59
(k,64,8,7,7)	(1,64,8,7,7)	max	1	6.46	10.26	10.83
			4	4.19	6.27	7.52

Table 1.3: Our architecture's performance using 2 different policies and 2 different pooling methods using modified Roi Align.

According to Table 1.3, modified Roi Align doesn't improve mAP performace. On the contrary, it reduces it. However, the gap between those 2 approaches is small, so we don't abandon this idea, becuase, for different approaches, modified Roi Align may outclass regular Roi Align.

1.3.2 Temporal pooling

After getting first results, we implement a temporal pooling function inspired from [?]. We need a fixed input size for the SVM. However, our tubes' temporal stride varies from 2 to 5. So we use as fixed temporal pooling equal with 2. As pooling function we use 3D max pooling, one for each filter of the feature map. So for example, for an action tube with 4 consecutive ToIs, we have 4,256,8,7,7 as feature size. We seperate the feature map into 2 groups using *linspace* function and we reshape the feature map into 256,k,8,7,7 where k is the size of each group, After using 3D max pooling, we get a feature map 256,8,7,7 so finally we concat them and get 2,256,8,7,7. In this case we didn't experiment with (64,8,7,7) feature maps because it wouldn't performed better that (256,8,7,7) ferature maps as noticed from the previous section.

We experiment using a SVM classifier for training policies 1 and 4 and using both regular and modifier Roi Align. The perfomance results are presented at Table 1.4.

Dimensions		**Pooling**	**Type**	**mAP precision**		
before	after			0.5	0.4	0.3
k,256,8,7,7	2,256,8,7,7	RoiAlign	1	25.07	26.91	29.11
			4	23.27	25.96	28.25
		mod RoiAlign	1	7.01	9.69	10.52
			4	5.5	7.25	8.99

Table 1.4: mAP results using temporal pooling for both RoiAlign approaches

Comparing Tables 1.3 and 1.4, we clearly notice that we get better results when using temporal pooling. Also, the difference between regular Roi Align and modified Roi Align become much bigger than previously, so this makes us abandon the idea of modified Roi Align. So, the rest section, we only experiment using regular Roi Align.

1.4 Increasing sample duration to 16 frames

Next, we though that a good idea would be to increase the sample duration from 8 frames to 16 frames. We experiment both using and not using temporal pooling, again for policies 1 and 4. Results are included at table 1.5.

Dimensions before	after	**Temporal Pooling**	**Type**	**mAP precision** 0.5	0.4	0.3
k,256,16,7,7	1,256,16,7,7	No	1	23.4	27.57	28.65
			4	22.7	26.95	28.05
k,256,16,7,7	2,256,16,7,7	Yes	1	21.12	24.07	24.36
			4	18.36	23.09	23.75

Table 1.5: mAP results for policies 1,4 for sample duration = 16

As shown at Table 1.5, we get better performance when we don't use temporal pooling, fact that is unexpected. However, the difference between those performaces is about 2%. Probably, this is caused by the fact that, in the temporal pooling approach, SVM classifier has to train too many parameters when it uses temporal pooling, on the contrary with the approach not using temporal pooling, in which SVM has to train half the number of parameters. Futhermore, comparing above results with results shown at Table 1.3, we can see that we get about the same results for both approaches. So, we choose to keep using approach with sample duration equal with 8. That's because, we don't have to use too much memory during training and validation.

1.5 Adding more groundtruth tubes

Pending more comments...
From above results, we notice that SVM improve a lot the perfomance of our model. In order to futher improve our results, we will add more groundtruth action tubes. We consider as groundtruth action tubes all the tubes whose overlap score with a groundtruth tube is greater that 0.7 . Also, we increase the total number of tube to from 1 to 2, 8. Table 1.8

F. map	**FG tubes**	**Total tubes**	**mAP** 0.5	0.4	0.3
(k,256,8,7,7)	1	3	11.3	14.14	14.84
	2	3	1.96	5.07	7.27
		4	3	5.03	5.77
		6	1.34	3.89	4.49
		8	0.77	1.51	2.72
	4	6	13.23	21.74	25.4
		8	20.73	28.25	29.50
		12	16.55	24.35	25.22
		16	20.11	25.50	27.62

	8	12	13.82	19.93	22.80
		16	15.47	23.08	24.19
		24	15.88	23.44	24.48
		32	12.66	23.50	25.61

Table 1.6: RNN results

F. map	**FG tubes**	**Total tubes**	**mAP**		
			0.5	0.4	0.3
(k,256,8,7,7)	1	3	14.18	19.81	20.02
	2	3	12.68	13.38	15.14
		4	11.5	14.95	16.22
		6	10.74	13.36	15.18
		8	8.00	9.83	11.17
	4	6	15	17.55	19.39
		8	17.04	20.12	22.07
		12	17.57	19.9	21.88
		16	14.24	17.24	17.95
	8	12	17.91	22.51	24.62
		16	16.76	20.34	22.72
		24	17.61	19.12	24.48
		32	14.45	18.07	19.14

Table 1.7: Linear results

F. map	**FG tubes**	**Total tubes**	**Policy**	**mAP**		
				0.5	0.4	0.3
(2,256,8,7,7)	1	3	1	25.07	26.91	29.11
			4	23.27	25.96	28.25
	3	8	1	24.38	25.97	26.4
			4	Pending...		
	4	8	1	Pending...		
			4	Pending...		
		12	1	Pending...		
			4	Pending...		
		16	1	Pending...		
			4	Pending...		
	8	16	1	Pending...		
			4	Pending...		
		24	1	Pending...		

			4	Pending...
		32	1	Pending...
			4	Pending...

Table 1.8: SVM results

1.5.1 Increasing again sample duration (only for RNN and Linear)

Table 1.5 showed that SVM classifier gets about the same performance for both sample durations 8 and 16 frames. Triggered by this fact, we trained RNN and Linear classifiers for sample duration equal with 16 frames. Table 1.9 shows RNN's mAP performance and Table 1.10 Linear's mAP performance.

FG tubes	**Total tubes**	**mAP**		
		0.5	0.4	0.3
4	8	Pending...		
	12	Pending...		
	16	Pending...		
8	16	12.29	19.51	23.11
	24	12.85	18.35	20.00
	32	9.38	14.33	16

Table 1.9: RNN results for sample duration equal with 16

FG tubes	**Total tubes**	**mAP**		
		0.5	0.4	0.3
4	8	15.11	19.78	21.14
	12	11.39	15.74	18.15
	16	13.62	16.11	18.15
8	16	12.98	17.52	19.10
	24	12.92	17.64	19.95
	32	11.51	13.98	14.82

Table 1.10: Linear results for sample duration equal with 16

Pending... commentary

1.6 MultiLayer Perceptron (MLP)

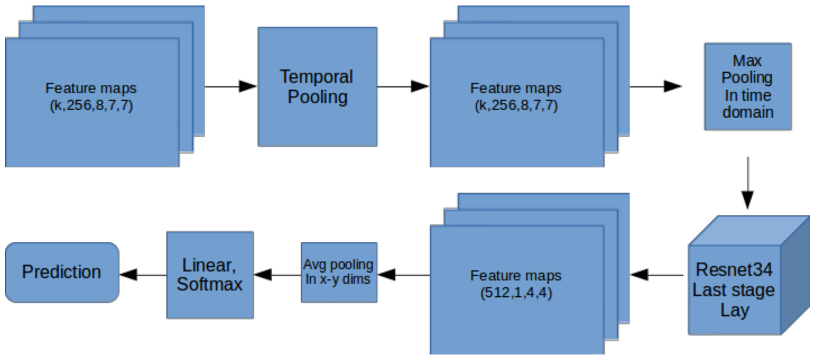

Figure 1.3: Structure of the MLP classifier

In previous sections we used classic classifiers like Linear, RNN and SVM. Last but not least approach, another widely category of classifiers is Multilayer Perceptorn (MLP) classifiers. MLP is a class of feedforward Neural Network, so its function is described in chapter 2. So, we design a MLP which is shown in Figure 1.3 for sample duration equal with 8, and is described below:

- At first, after 3D Roi align and for sample duration = 8, we get an activation map of $(k, 256, 8, 7, 7)$ where k is the number of linked ToIs. Inspired by previous sections, we perform temporal pooling followed by a max pooling operation in sample duration's dimension. So, we now have an activation maps with dimensions equal with $(2, 256, 7, 7)$, which we reshape it into $(256, 2, 7, 7)$. we extracted layers from the last stage of ResNet34. This stages includes 3 Residual Layers with stride equal with 2 in all 3 dimensions and output number of filters equal with 512.
- After Residual Layers, we perfom temporal pooling for x-y dimensions. So we get as output activation maps with dimension size equal with (512,). Finally, we feed these feature maps to a linear layer in order to get class condifence score, after applying soft-max function.

1.6.1 Regular training

According to figure 1.1, the trainable parts of our network is TPN and the classifier. As mentioned before, training code requires running only one video per GPU, because, videos have different duration. For previous approaches we came with the idea of pre-calculating video features and then training only the classifier. However, for this step, we normally trained our in order to get classification results. Of course, we used a pre-trained TPN, whose layers were freezed in order not to be trained. We tried to explore different ratios between the number of foreground tubes and the total number of tubes per video. First 3 simulations included fixed number of total tubes and variable ratio between the number of foreground and background tubes. We started using only foreground tubes, which means 32 out 32 tubes are foreground, then half of the proposed tubes aka 16 out of 32 and finally less than half, namely 14 out of 32. After that, we experiment using a fixed number of foreground tubes and variable number of total tubes, which are 16, 24 and 32. The performance results are presented at Table 1.11.

FG tubes	**Total tubes**	**mAP**		
		0.5	0.4	0.3
32	32	1.28	1.73	1.87
16		3.98	4.38	4.38
14		0.40	0.40	0.40
8	16	9.41	12.59	14.61
	24	12.32	15.53	18.57
	32	7.16	10.92	13.00

Table 1.11: MLP'smAP performance for regular training procedure

The results show that when first 3 approaches give us very bad results. Comparing them with the rest 3, we came with the conclusion that we need at the most 8 foreground tubes, even thought the ration between the number of foreground and background is in favor the second one. Probably, too many foreground action tubes make our architecture overfitted so unable to generalize.

1.6.2 Extract features

Pending... As previously performed, we trained MLP classifier using pre-computed feature maps. These feature maps include both foreground and background action tubes. Base on the conclusion made in previous sections, we will train our classifier only for number of foreground tubes equal with 4 and 8. Futhermore, we will train it for 3 different ratios between the number of foreground and background action tubes, which are 1:1, 1:2 and 1:3. Table 1.12 shows these cases and their respective mAP performance during validation step.

FG tubes	**Total tubes**	**mAP**		
		0.5	0.4	0.3
4	8	5.89	9.54	13.61
	12	9.51	12.8	14.6
	16	Pending...		
8	16	Pending...		
	24	Pending...		
	32	Pending...		

Table 1.12: mAP results for MLP trained using extracted features

Pending.. commentary

1.7 Adding nms algorithm

Pending... Introduction

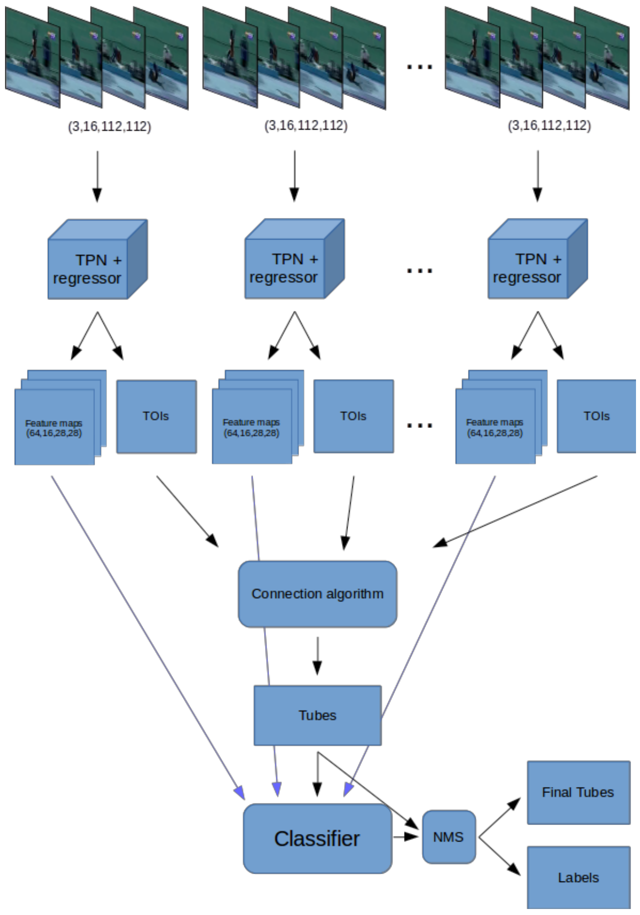

Figure 1.4: Structure of the network with NMS

FG tubes	**Total tubes**	**mAP** 0.5	0.4	0.3
4	8	Pending...		
	12	Pending...		
	16	Pending...		
8	16	Pending...		
	24	Pending...		
	32	Pending...		

Table 1.13: mAP results for SVM classifier after adding NMS algorithm

FG tubes	**Total tubes**	**mAP** 0.5	0.4	0.3
4	8	Pending...		
	12	Pending...		
	16	Pending...		
8	16	Pending...		
	24	Pending...		
	32	Pending...		

Table 1.14: mAP results for SVM classifier after adding NMS algorithm

FG tubes	**Total tubes**	**mAP** 0.5	0.4	0.3
4	8	Pending...		
	12	Pending...		
	16	Pending...		
8	16	Pending...		
	24	Pending...		
	32	Pending...		

Table 1.15: mAP results for SVM classifier after adding NMS algorithm

FG tubes	**Total tubes**	**mAP** 0.5	0.4	0.3
4	8	Pending...		
	12	Pending...		

	16	Pending...		
8	16	Pending...		
	24	Pending...		
	32	Pending...		

Table 1.16: mAP results for SVM classifier after adding NMS algorithm

Pending... Commentary

1.8 Classifying dataset UCF

1.8.1 Using RNN, Linear and MLP classifiers

At first, we use the same approach we did for classifying JHMDB dataset. However, we don't use a SVM classifier because its training requires too much resources, which we don't have available. That's because, during training, it loads every feature map and keeps it for future training of the SVM classifier, according to hard mining negative training procedure. So, we again extract foreground and background action tubes and save them in order to use them during classifiers' training phase. The only difference with JHMDB's approach is that before saving them, we perform max pooling in sample duration's dimension because saving those feature maps requires too much memory, and by doing this, we reduce significantly.

FG tubes	**Total tubes**	**mAP**		
		0.5	0.4	0.3
4	8	Pending...		
	12	Pending...		
	16	Pending...		
8	16	Pending...		
	24	Pending...		
	32	Pending...		

Table 1.17: RNN results

FG tubes	**Total tubes**	**mAP**		
		0.5	0.4	0.3
4	8	Pending...		
	12	Pending...		
	16	Pending...		

8	16	Pending...
	24	Pending...
	32	Pending...

Table 1.18: Linear results

Pending... commentary

1.8.2 Only temporal classification

As presented in chapter 5, our connection algorithm is able to get good temporal recall and MABO performance. In most cases, MABO performance got score about 92-94%. So, we came with idea of just performing temporal localization, instead of spatio-temporal localization.

In order to temporally localize action in videos, we use only the temporal information containing in the proposed action tubes, which means the first and the last frame of the action tube. After that, we will classify the proposed action tubes without performing spatiotemporal localization, but only temporal. Although we don't use the extracted bounding boxes for classification, we take advantage of the spatial information in order to perform better temporal localization. Intuitively,that's because, in order to extract the action tubes, we consider the spatial overlap between the connected ToIs . This aformentioned approach includes the following steps:

1. Fist, we use TPN in order to propose spatio-temporal ToIs, just like we did in previous approaches. Then, we link those ToIs based on the proposed algorithm in the chapter 5, using spatiotemporal NMS algorithm with threshold equal with 0.9, for removing overlapping action tubes.

2. Previous steps is exactly the same as previous classification approaches. However, in this approach, we don't use any kind of Roi Align in order to extract action tubes' feature maps. On the contrary, for all the proposed action tubes, we find their duration, aka their first and their last frame. After that, we perform temporal nms in order to remove overlapping action tubes. The only difference between spatio-temporal and temporal nms is the overlapping criterion, which is used. For spatio-temporal nms, we use spatiotemporal IoU and respectively, for temporal we use temporal IoU as presented in chapter 2.

3. Of course, the proposed action tubes last more that 16 frames, which we set as sample duration. So, we seperate action tubes into video clips lasting 16 frames (like our sample duration). These video segments are fed, again at a 3D resNet34 ([?]), but this time, we don't use it only for feature extraction but, also for classification for each video segment.

4. So, for each video clip, for each class we get a confidence score after performing softmax operation. Finally, we get average confidence score for each class, and we consider the best-scoring class as the class label of each actio tube. Of course, some action tubes may not contain any action, so we set a confidence score for Seperate foreground action tubes with background.

Training The only trainable part of this architecture is the ResNet34. We use a pre-trained TPN as presented in chapter 4. ResNet34 training procedure is based on the code given by [**?**]. We modified it in order to be able to be trained for dataset UCF-101, only for the 24 classes, for which there are spatiotemporal notations and our TPN is trained.

Validation Based on the aforementioned steps, it is clear that the parameters that can be modified are temporal NMS' threshold and confidence threshold for deciding if an action is contained or not. All the different combinations used during validation are presented at Table 1.19.

NMS thresh	**Conf thresh**	**mAP**		
		0.5	0.4	0.3
0.9	0.75	0.20	0.25	0.30
	0.	0.38	0.45	0.54
	0.85	0.49	0.55	0.64
0.7	0.75	Pending...		
	0.8	Pending...		
	0.9	Pending...		
0.5	0.75	Pending...		
	0.8	Pending...		
	0.9	Pending...		

Table 1.19: UCF's temporal localization mAP perfomance

Pending... commentary