Chapter 1

Classification stage

1.1 Introduction

In previous 2 chapters, we introduced the procedure we used to create candidate action tubes, which probably contain some action or may not. Most of the times, the proposed action tubes belong to the background, and for that reason, as mentioned and in the previous chapter, it is important to choose a good linking algorithm that generates good sequences of bounding boxes. However, it is quite important to choose the appropriate classifier who will be able very accurately predict whether a candidate action tube belongs to a known category of actions or it belongs to the background. This is because we may produce good proposals for candidate actions, but if our classier is unable to distinguish them, our system will again fail to recognize the actions.

The right choice of a classifier is a big dilemma we are facing and we need to answer. However, this classifier will get as input some activation maps in order to be classified. Therefore, apart from the good selection of classifier, equally, good choice of features is important, too. Finally, the training process of the classifier plays a major role in order to be able to generalize and to avoid overfitting situations.

For our implementation, we implement approaches including a Linear Classifier, a Recursive Neural Network (RNN) Classifier, a Support Vector Machine (SVM) Classifier and a Multilayer perceptron (MLP). Additionally, we experiment using feature maps obtained from 3D RoiAlign using, also, avg or max pooling. Last but not least, we try to find the right ratio between foreground and background action tubes including their total number needed during training stage in order our classifier to perform efficiently.

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

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

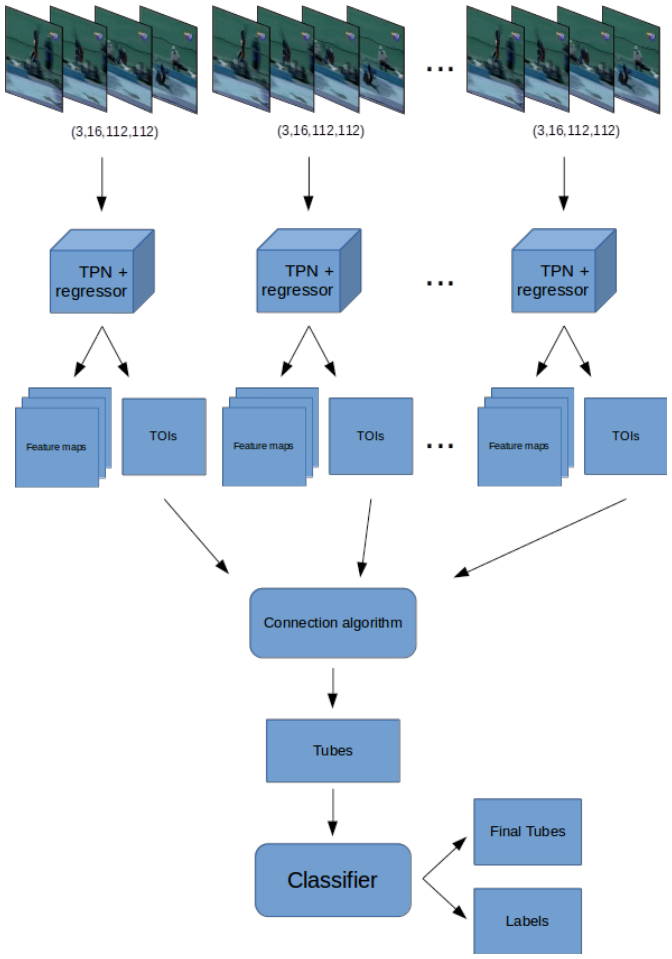

Figure 1.1: Structure of the whole network

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 its feature maps into the classifier in order to perform classification.

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.

We treated each dataset seperately, because we didn't manage to achieve good recall performance for UCF-101 dataset, it is preferable to experiment mainly at JHMDB dataset for spatiotemporal localization. Then, we performed temporal localization for UCF-101, because we achieve good temporal recall performance during connection stage.

1.2 Preparing data and first classification results

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

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.

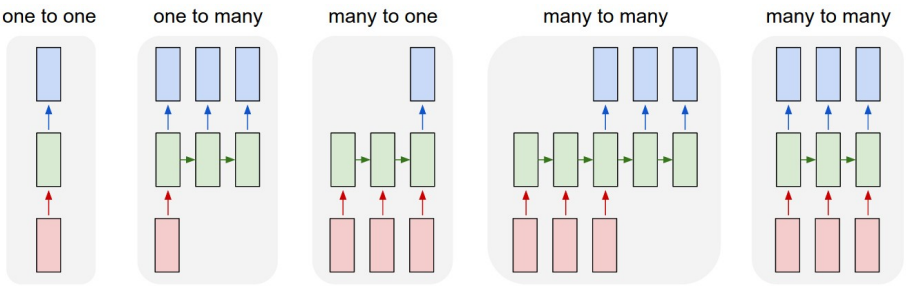

Figure 1.2: Types of RNN

Training In order to train our classifier, we have to execute the steps, presented in the first section, 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 and considering there are 4 GPUs available, 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 foreground and background 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 action tubes' features. Also we extract feature maps from background action tubes, which are double the number of groundtruth action tubes. We chose this ratio between the number of positive and negative tubes inspired by [**?**], in which it has 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 action 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, in order to get those action tubes,we use a pre-trained TPN to generate ToIs for each video segment and then our proposed connection algorithm for linking proposed ToIs. Finally, we get each action tube's corresponding feature map using 3D RoiAlign

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 number of 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 experimented using 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 threshold 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: First classification results using Linear and RNN classifiers

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 hyper-plane 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 separating hyper-plane. 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 kernel, uses L2-norm as penalty and L1-norm as loss during training. Training procedure starts with randomly picking 100 videos in order to calculate feature's scale. After that, each svm is provided by positive samples' feature maps and network looks for hard negatives for each class' svm. We consider as hard-negatives the tubes that got confidence score > -1.0 during classification, and we add them to svm's samples. When we gather hard-negatives whose number is bigger than 500 or 1000 (depending on the approach) we retrain the class' SVM and remove samples with new score < -1.0.

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

1. Negatives are other classes' positives and all the background tubes
2. Negatives are only all the background videos
3. Negatives are only other classes' positives
4. Negatives are other classes' 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 function, 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 columns 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 environment 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 performance. On the contrary, it reduces it. However, the gap between those 2 approaches is small, so we don't abandon this idea, because, 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, because a video lasting 15 frames is consisted of 2 ToIs and a video of 40 frames is consisted of 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 separate 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 with dimensions $(256, 8, 7, 7)$, so we concat them and finally get feature maps with size of $(2, 256, 8, 7, 7)$. In this case we didn't experiment with feature maps with size $(64, 8, 7, 7)$ because they wouldn't performed better than feature maps with size $(256, 8, 7, 7)$ 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 performance 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	24.97	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		**Temporal**	**Type**	**mAP precision**		
before	after	**Pooling**		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

performances 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. Furthermore, 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

The previous results came from when we train classifiers using only 1 groundtruth action tube and 2 background. We thought that we should experiment with the number of foreground action tubes and the ratio between foreground and background tubes because in previous approaches these numbers were a little arbitrary. So, we choose to train our previous classifiers using 2, 4 and 8 foreground tubes and a ratio of 2:3, 1:2, 1:3, and 1:4 between the number of foreground tubes and the total number of both foreground and background tubes.

Firstly, we train the RNN classifier using feature maps with dimensions *(256,8,7,7)* and mAP performance is presented at Table 1.6 for overlap threshold equal with 0.5, 0.4 and 0.3 .

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

According to Table 1.6, firstly we can see that increasing the number of foreground tubes from 1 to 2 leads to reduce rapidly mAP performance. But,

when we set foreground tubes equal to 4 we get better results. On top of that, we get best performance when the ratio is equal with 1:2 and 1:4. Finally, when we set the number of foreground tubes equal with 8, performance gets slightly better comparing with the initial conditions (1 foreground action tube and 3 in total) but, this situation doesn't get us the best results.

Next, it's time to experiment using the Linear classifier. We use again the same cases like we did for RNN classification. As mentioned before, we need a pooling method before classification step. According to Table 1.1, avg pooling method results in better mAP performance than max pooling, so we use avg pooling for all following cases. Results are included at Table 1.7.

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

First of all, after considering results presented at both Tables 1.6 and 1.7, it becomes clear that when we set the number of foreground tubes equal with 2, for both case, we get worse results that the initial. This probably is due to the fact that we increase also the number of background tubes for cases when ratio is 1:2, 1:3 and 1:4 resulting in considering more proposed tubes as background tubes. On the other hand when we set ratio equal with 2:3, instead of considering most proposed action tubes as background, classifiers classify them as a specific action class, which means there is an overfitting situation. So, although we think that we shouldn't investigate any more for cases with 2 foreground action tubes, we will train our SVM classifier using 2 foreground tubes and all the aforementioned ratio because we want to be sure about our assumption. On the other hand we notice that using 4 or 8 foreground tubes both get us better results than the initial results. The best results come when ratio between foreground and total tubes is 1:3 for both cases. Furthermore,

we get good results for ratios 2:3 and 1:2, and we get the worst while using ratio 1:4. This is caused probably from the big number of background tubes comparing with the one of foreground tubes.

As mentioned before, we train SVM classifier using aforementioned cases only for policy 1 because it gives us the best results for all previous cases. Classification performance using mAP metric is shown at Table 1.8.

F. map	**FG tubes**	**Total tubes**	**mAP**		
			0.5	0.4	0.3
(2,256,8,7,7)	1	3	24.97	26.91	29.11
	2	3	13.87	18.74	21.29
		4	14.21	19.67	21.75
		6	12.88	18.62	21.59
		8	12.66	18.7	21.97
	4	6	25.04	26.91	27.82
		8	24.34	25.67	26.34
		12	23.47	25.31	25.9
		16	21.94	23.55	24.23
	8	12	24.83	27.13	27.46
		16	23.97	26.38	26.94
		24	24.17	26.24	26.76
		32	24.17	26.24	26.76

Table 1.8: SVM results

Results shows us some interesting facts. Firstly, it confirms our assumption that our network is unable to train well with only 2 foreground tubes, so from now on, we will investigate training situations using 4 or 8 foreground action tube during training. Also, a strange fact occurs, which is that we get almost the same results with the results obtained for using policy 1, only one foreground tube, 3 in total and temporal pooling. This is probably because during calculation of feature scale, during training stage, we don't get such good sample set of video like we did during aforementioned situation. But we think that it is better to keep testing using 4 or 8 foreground action tubes. Last but not least, it is clear that we get the best result when we have ratio 2:3 between number of foreground and total action tubes. Also, it is more preferable to have 4 foreground action tubes instead of 8. This means that given too many action tubes confuse SVM classifier, so it fails to perform well.

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. We started experimenting with RNN classifier because it performed better than Linear classifier previously. As mentioned before, we experiment using 4 or 8 foreground tubes and ratios 2:3, 1:2, 1:3 and 1:4 between the number of foreground and total action tubes provided.

FG tubes	**Total tubes**	**mAP**		
		0.5	0.4	0.3
4	6	7.94	13.51	14.95
	8	10.88	14.02	14.74
	12	14.05	19.23	20.99
	16	11.69	15.77	16.87
8	12	10.47	15.06	19.93
	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

Comparing results from Table 1.9 and previous table 1.6 one by one, it is clear that RNN outperforms when sample duration equals with 8 frames. This results was expected, because, the increase of sample duration reduce the number of video segments needed for each video. So, this means that RNN has to classify sequences with 3 video segments at the most, which is more difficult that classifying bigger sequence like previously.

Next, we experiment using Linear classifier for the same cases like RNN's.

FG tubes	**Total tubes**	**mAP**		
		0.5	0.4	0.3
4	6	13.79	19.75	23.63
	8	15.11	19.78	21.14
	12	11.39	15.74	18.15
	16	13.62	16.11	18.15
8	12	10.63	19.37	21.65
	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

In this case, results from table 1.10 are worse than those presented at Table 1.7, with the difference being about 2%. So, after considering both cases, it becomes clear that it is preferable to experiment using sample duration equal with 8 and not to increase it to 16 frames.

1.6 MultiLayer Perceptron (MLP)

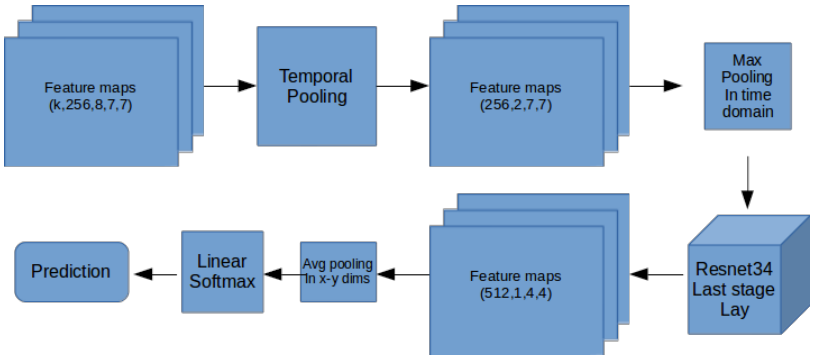

Figure 1.3: Structure of the MLP classifier

In previous sections we used classic classifiers like Linear, RNN and SVM. Last but not least, another widely category of classifiers is Multilayer Perceptron (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 $(256, 2, 7, 7)$ which we feed to layers extracted 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 perform avg 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 confidence 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 frozen 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's mAP 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 ratio 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

As previously performed, we trained MLP classifier using pre-computed feature maps. These feature maps include both foreground and background action tubes. Based on the conclusions made in previous sections, we will train our classifier only for number of foreground tubes equal with 4 and 8. Furthermore, 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	6	4,37	8,54	10,12
	8	5.89	9.54	13.61
	12	9.51	12.8	14.6
	16	6.80	13.17	14.67
8	12	8,62	12,32	14,74
	16	8.49	13.94	15.09
	24	6.72	12.17	15.30
	32	13.27	17.64	18.97

Table 1.12: mAP results for MLP trained using extracted features

Comparing results from Tables 1.12 and 1.11, it is clear we need 8 foreground tubes in order MLP classifier to perform well. However, it isn't very clear which of these two proposed training approaches is better, but if we have to decide one method, we would choose using pre-extracted features training. This approach manages to achieve the best results, and especially when we have 8 foreground action tubes and 32 in total. Also, comparing methods using 4 or 8 positive action tubes, it is clear that we would prefer using 8 generally. However, it's not clear which ratio is better because, we get best results when we have 8 foreground action tubes and ratio 1:4 while we get best results when ratio is 1:3 having 4 positive action tubes.

1.7 Adding nms algorithm

After getting previous classification results, we came with the idea that a lot of proposed action tubes overlap spatiotemporally like presented in chapter 4, for the first linking algorithm. On top of that, even though, at last, we will keep only the best scoring action tube, maybe, our Network sometimes doesn't score the best overlapping action tube but a neighbor, which should have been removed. So, similar with chapter 4, we added NMS algorithm before classification stage in order to remove unnecessary overlapping action tubes. The structure of this approach is presented at Figure 1.4. So, we run again validation stage for our classifiers and results are presented at Tables 1.13, 1.14, 1.15 and 1.16 for SVM, RNN, Linear and MLP classifiers respectively.

We start experimenting using SVM classifier, whose mAP performance is presented at Table 1.13 when we use NMS algorithm.

FG tubes	**Total tubes**	**mAP**		
		0.5	0.4	0.3
4	6	21.42	24.3	25.11
	8	21.3	24.06	24.85
	12	21.73	24.19	25.04
	16	21.11	23.98	24.84

8	12	20.47	22.55	23.41
	16	21.21	23.89	24.97
	24	21.71	23.8	24.82
	32	21.43	23.5	24.52

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

According to Table 1.13 and taking Table's 1.8 results into consideration, it is clear that mAP performance decreases while using NMS algorithm. We run only for case with foreground tubes equal with 4 or 8 and we didn't experiment using initial ratio, because we think that we are going to get the same attitude. This is probably because NMS algorithm removes some overlapping action tubes, so SVM classifier doesn't get the right action tubes for classifying them.

Next, we experiment using RNN classifier for sample duration equal with 8 frames per video segment. Results are included at Table 1.14

FG tubes	**Total tubes**	**mAP**		
		0.5	0.4	0.3
4	6	13.27	26.12	30.69
	8	16.06	25.81	27.63
	12	13.93	22.48	23.99
	16	17.24	26.44	28.36
8	12	2	5.11	7.75
	16	11.11	20.98	23.97
	24	13.84	22.77	24.31
	32	12.74	21.49	25.39

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

Alongside with SVM's mAP performances, RNN doesn't achieve any improvement in its mAP performance when adding NMS algorithm. On the contrary, it reduce about 5% in some cases, so it is more preferable to use ActionNet's architecture without NMS algorithm included while using RNN as classifier.

Following RNN, we investigate the same situations with Linear classify instead of RNN. Its performance is presented at Table 1.15.

FG tubes	**Total tubes**	**mAP**		
		0.5	0.4	0.3

4	6	12.59	14.91	17.79
	8	15.89	21.92	23.28
	12	13.23	19.72	23.17
	16	15.07	17.38	18.27
8	12	18.88	24.37	26.72
	16	15.42	22.31	24.70
	24	15.60	19.71	21.08
	32	16.1	19.99	21.47

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

On the contrary with SVM and RNN's performances, Linear's mAP performance is getting better when adding NMS algorithm, comparing results presented at Tables 1.15 and 1.7. This is probably because Linear classifier gets now less proposed action tubes comparing with previous approach. So, now it is less likely to misclassify an action tube, probably considering it as foreground when actually is a background action tube. Even though its performance increased, it doesn't achieve our best results, fact which may corroborate our previous claim. Last but not least, we need to experiment using MLP classifier in order to determine if NMS algorithm is needed during classification. The results are presented at Table 1.16.

FG tubes	**Total tubes**	**mAP**		
		0.5	0.4	0.3
4	6	3.66	7.23	9.43
	8	3.43	8.17	12.77
	12	6.32	11.26	16.15
	16	4.82	11.38	15.85
8	12	5.92	12.42	15.81
	16	6	12.55	14.66
	24	4.73	11.33	15.25
	32	9.67	14.82	16.74

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

Finally, comparing results from Tables 1.16 and 1.12, again adding NMS algorithm approach decreases mAP performance. Consequently, after considering all situations it is clear that NMS algorithm doesn't contribute at the improvement of Network's mAP performance, but, on the contrary, it reduces it. So, as a final comment, it is more preferable not to uses it, except when we use Linear classifier for classification.

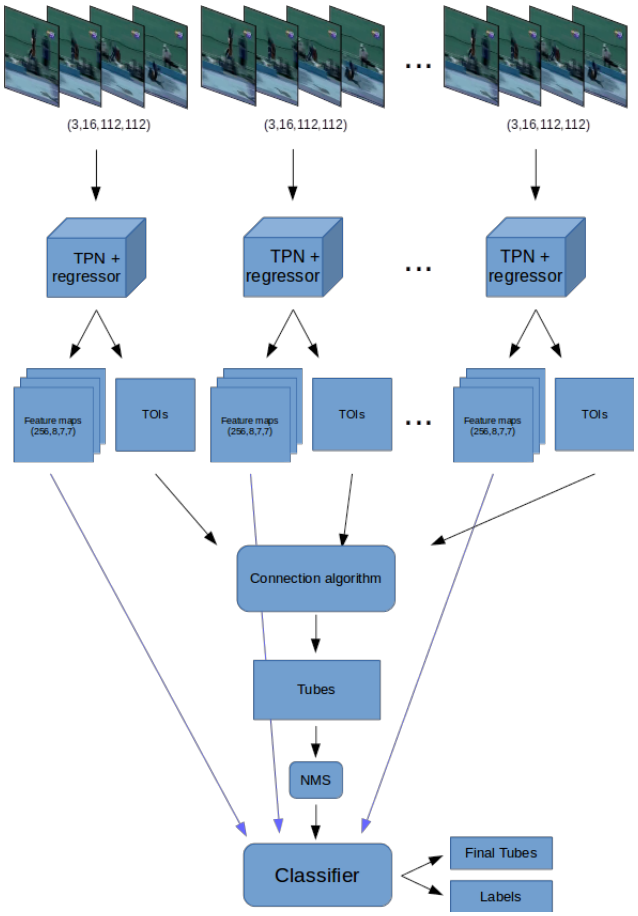

Figure 1.4: Structure of the network with NMS

1.7.1 General comments

To sum up, in this section we tried to find the most suitable classifier for achieve good classification results based on mAP metric. It is clear that when using a SVM classifier, which uses temporal pooling in order to have a fix-size input, we get the best results. The most suitable method for training is using about 4 foreground action tubes and only 2 background which using both background action tubes and action tubes from other classes as hard-negatives.

1.8 Classifying dataset UCF

1.8.1 Introduction

During previous section we explore different classification methods using several classifiers. On top of that, it became clear how important is generating good proposal considering the situation where we added NMS algorithm. NMS algorithm reduced the number of proposed action tubes, so most of classifiers failed to improve their performance. Considering recall and MABO performances presented in chapter 4, it is clear that our network would fail recognizing most of the groundtruth spatiotemporal action tubes and correctly classifying them. However, in most cases, MABO performance got score about 92-94%. So, we came with the idea of rather than performing spatiotemporal localization and classification, and getting very bad classification results, for dataset UCF, we performed only temporal localization and explored our network's potential. Temporal localization means that our network tries to detect the video segments in which an action is performed, and simultaneously determine the class of this performed action.

1.8.2 Only temporal classification

As presented in chapter 4, our connection algorithm is able to get good temporal recall and MABO performance. 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. 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 aforementioned approach includes the following steps:

1. Fist, we use TPN in order to propose spatiotemporal ToIs, just like we did in previous approaches. Then, we link those ToIs based on the proposed algorithm in the chapter 4, using spatiotemporal NMS algorithm with threshold equal with 0.9, for removing overlapping action tubes.

2. Nexts steps are 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 spatiotemporal and temporal nms is the overlapping criterion, which is used. For spatiotemporal 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 separate action tubes into video clips lasting 16 frames (like our sample duration). These video segments are fed, again at a 3D ResNet34 ([**?**]), which this time, we don't use 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 action tube. Of course, some action tubes may not contain any action, so we set a confidence score for separate 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 annotations 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 1.17.

NMS thresh	**Conf thresh**	**mAP**		
		0.5	0.4	0.3
0.9	0.6	0.3	0.54	0.64
	0.75	0.25	0.45	0.55
	0.85	0.2	0.38	0.49
0.7	0.6	0.63	1.02	1.27
	0.75	0.5	0.84	1.05
	0.85	0.4	0.68	0.89
0.5	0.6	0.96	1.21	1.75
	0.75	0.63	0.93	1.38
	0.85	0.57	0.72	1.03
0.4	0.6	1.07	1.52	2.03
	0.75	0.79	1.18	1.63
	0.85	0.71	0.98	1.33
0.3	0.6	1.1	1.66	2.53
	0.75	0.93	1.39	2.08
	0.85	0.81	1.12	1.6
0.2	0.6	0.84	1.38	2.17
	0.75	0.73	1.13	1.78
	0.85	0.65	0.81	1.31

Table 1.17: UCF's temporal localization mAP performance

NMS thresh	Recall			MABO
	0.9	0.8	0.7	
0.9	0.7361	0.8935	0.9422	0.9138130172
0.7	0.3194	0.6875	0.9293	0.8412186326
0.5	0.1757	0.3331	0.6281	0.7471525429
0.4	0.1483	0.2829	0.4707	0.6986400756
0.3	0.111	0.2038	0.3848	0.6429232202
0.2	0.1217	0.2000	0.3163	0.5769587340803654

Table 1.18: UCF's temporal localization recall and MABO performances

According to Table 1.17, mAP performance for temporal localization and classification is very bad. The best performance is about 2%, score which is very low. Comparing these results with results shown at Table 1.18, we deduce that our method is not at all efficient. Even though mAP results increase slightly, recall and MABO performance decrease rapidly. Of course, this result is anticipated because, by decreasing NMS threshold, the number of rejected action tubes is increased.

We tried another approach, which applies NMS algorithm after classification stage, and not before it like we did previously. Also, we noticed in previous results that in most cases, we get these low performances because of the number of false positives, which don't get removed during NMS procedure. To be more specific, Table 1.19 shows all the detected true and false positives when we set NMS threshold equal with 0.2, mAP overlap threshold equal with 0.3 and confidence threshold equal with 0.6 for both aforementioned approaches.

Class	Appr 1		Appr 2		Class	Appr 1		Appr 2	
	TP	FP	TP	FP		TP	FP	TP	FP
Basketball	5	279	6	403	BasketballDunk	7	7	12	13
Biking	0	3	0	5	CliffDiving	11	55	1	1
CricketBowling	0	0	10	75	Diving	20	189	23	272
Fencing	11	222	25	336	FloorGymnastics	2	86	6	131
GolfSwing	4	51	6	78	Riding	0	33	4	58
IceDancing	8	29	6	38	LongJump	1	24	6	43
PoleVault	0	202	9	296	RopeClimbing	1	24	4	43
SalsaSpin	3	158	5	237	SkateBoarding	0	10	0	13
Skiing	0	0	0	0	Skijet	1	27	6	43
SoccerJuggling	3	94	1	153	Surfing	11	102	23	159
TennisSwing	0	125	0	166	TrampolineJumping	4	18	4	32
VolleyballSpiking	20	704	20	1044	WalkingWithDog	0	5	0	9

Table 1.19: Comparing TP and FP for both approaches

Considering those two facts we came with the following solution. In previous approach, we used NMS algorithm based on the connection scores obtained from linking algorithm, so we removed those which overlap with high-scoring tubes. In our new approach, we firstly remove action tubes with same temporal limits, in order to get unique temporal action tubes. Then, we classify all the proposed action tubes exactly like we did in step 3 previously. After that, we perform temporal NMS using the confidence scores extracted by the last layer of the 3D ResNet34 and finally we keep those which their confidence score is over a predefined threshold.

NMS thresh	**Conf thresh**	**mAP**			**MABO**
		0.5	0.4	0.3	
0.9	0.6	0.31	0.54	0.65	0.9073251461121519
	0.75	0.26	0.46	0.55	
	0.85	0.2	0.39	0.49	
0.7	0.6	0.66	0.95	1.22	0.833992951972403
	0.75	0.55	0.80	1.01	
	0.85	0.41	0.67	0.87	
0.5	0.6	0.98	1.43	1.63	0.7404971333964243
	0.75	0.75	1.14	1.29	
	0.85	0.64	0.92	1.04	
0.4	0.6	1.19	1.73	2.15	0.6823923696583215
	0.75	0.9	1.35	1.63	
	0.85	0.79	1.16	1.38	
0.3	0.6	1.12	1.85	2.23	0.6068219177169945
	0.75	0.96	1.54	1.7	
	0.85	0.83	1.28	1.43	
0.2	0.6	2.05	2.68	3.7	0.5243533655334142
	0.75	1.61	2.17	3	
	0.85	1.51	1.88	2.54	

Table 1.20: UCF's temporal localization mAP performance

Comparing Tables 1.20 and 1.18, we get about the same results for overlap thresholds 0.9, 0.7, 0.5, 0.4 and 0.3 . But for overlap threshold 0.2 we notice that mAP performance is improved only about 1%. Alongside with that, we noticed that recall performance is lower now than previously, fact that is undesirable. Those two facts made us think classification procedure.

After looking more carefully the results, we noticed that using average score obtained from each video segment for final classification score is a very bad choice. That's because, this approach firstly doen't highlight the differences

between classes in each video segment and secondly, gives bigger score for action tubes that have small duration. That's the reason, using confidence score reduces more temporall recall and MABO performance. As a result, we use the sum of confidence scores per action class obtained from each video segment.

Considering aforementioned results, we experiment using only NMS threshold equal with 0.2 and 0.1 because, mAP performance was very low in higher NMS threshold's values so these cases are not worth being considered. Table 1.21 shows mAP results and MABO performance for this method.

NMS thresh	**Conf thresh**	**mAP**			**MABO**
		0.5	0.4	0.3	
0.2	0.6	5.57	6.16	7.21	0.7455353939
	0.75	4.76	5.23	6.12	
	0.85	4.3	4.76	5.61	
0.1	0.6	5.51	6.11	7.22	0.6954217834
	0.75	4.76	5.23	6.13	
	0.85	4.3	4.76	5.61	

Table 1.21: UCF's temporal localization mAP performance calculating sum of confidence scores

Results appearing in 1.21 are very encouranging. Even though we got about the same MABO performance, mAP performance has increased significantly. We changed more our approach using softmax function before calculating confidence scores' sum, in order to reduce the differences along class scores. We thought that if our classifier misclassify a video segment, the whole classification prediction is more affected when there in no softmax operation before calculating the sums. Results from this method are shown in Table 1.22.

NMS thresh	**Conf thresh**	**mAP**			**MABO**
		0.5	0.4	0.3	
0.2	0.6	8.37	9.76	11.29	0.7455353939
	0.75	8.31	9.68	11.1	
	0.85	8.14	9.3	10.68	
0.1	0.6	8.37	9.69	11.49	0.6954217834
	0.75	8.27	9.53	11.27	
	0.85	7.92	9.03	10.69	

Table 1.22: UCF's temporal localization mAP performance after adding softmax before calculation the sum

Comparing Tables 1.22 and 1.21, we noticed that our assumption is correct. Results are improved using a softmax operation for each video segment's confi-

dence scores. On top of that, we thought to change the used classifier in order to achieve better results. We use again a 3D ResNet34 classifier, pretrained to Kinetices dataset. We "freeze" the first 3 groups of Layers, so we train the last group including 3 Residual Layers plus the final classification Layer. We use Cross-Entropy loss as loss function and we experiment for both aforementioned approaches. On top of that, for the second approach, we included cases in which NMS threshold is equal with 0.3 and 0.4. Performances of mAP and MABO are presented in Tables 1.23 and 1.24. Table 1.23 contationes mAP and MABO performances when not using softmax before calculating sums and table 1.24 contained these performances when using it.

NMS thresh	**Conf thresh**	**mAP**			**MABO**
		0.5	0.4	0.3	
0.2	0.6	10.47	18.05	24	0.558332305
	0.75	10.46	18.01	23.95	
	0.85	10.41	17.95	23.88	
0.1	0.6	8.86	17.42	25.81	0.4977558269
	0.75	8.86	17.42	25.81	
	0.85	8.81	17.36	25.74	

Table 1.23: UCF's temporal localization mAP performance using new classifier without softmax before the calculation of the sums

NMS thresh	**Conf thresh**	**mAP**			**MABO**
		0.5	0.4	0.3	
0.4	0.6	49.28	54.34	58.89	0.8138344861
	0.75	46.43	50.31	54.28	
	0.85	45.2	48.77	52.39	
0.3	0.6	50.01	54.29	59.3	0.7888980887
	0.75	46.79	49.98	54.15	
	0.85	45.43	48.33	52.06	
0.2	0.6	48.78	53.11	57.93	0.7602580204
	0.75	45.93	49.29	53.38	
	0.85	44.44	47.48	51.2	
0.1	0.6	48.49	52.67	57.4	0.7050217081
	0.75	45.97	49.32	53.37	
	0.85	44.46	47.48	51.2	
0.01	0.6	48.58	52.66	57.28	0.675086919
	0.75	46.06	49.31	53.34	
	0.85	44.54	47.53	51.25	

Table 1.24: UCF's temporal localization mAP performance using new classifier with softmax before the calculation of the sums

Comparing Tables 1.24, 1.23 with 1.22 and 1.21, it is clear that the new classifier outperforms from the previous one. That means that it is more preferable to train only some layers of a pretrained classifier instead of training it from scratch. Comparing now, tables 1.24 with 1.23, we confirm that using softmax operation before calculating sums results in better performance. On top of that, we get better mAP and MABO performance when using bigger threshold than 0.2, as we did previously. Our best results, which are 50%, are when NMS threshold is 0.3 and confidence score is 0.6.