Chapter 1

Classification stage

1.1 Description

After getting all proposed tubes, it's time to do classification. As classifiers we use several approaches includingn 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 previous 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 for classification - RNN Classifier

(Pending.. also an image of RNN classifier) Recurrent neural networks, or RNNs for short, are a type of neural network that was designed to

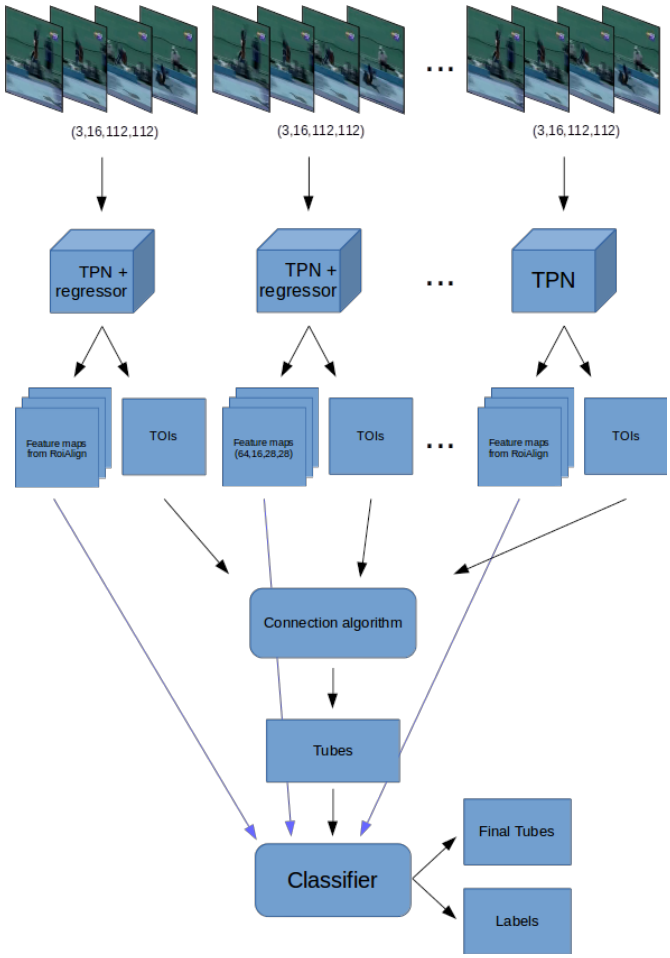

Figure 1.1: Structure of the whole network

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.

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

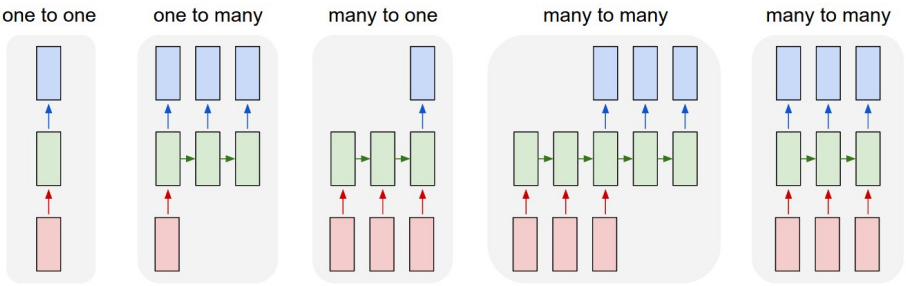

Figure 1.2: Types of RNN

video tubes and negative video tubes. Then we feed those features to our classifier and we train it in order to disciminate their classes. 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.

Then, 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 mean pooling as show in the results below. For the RNN classifier, we do not use any pooling function before feeding it. For both classifiers, at first, we didn't considered a fixed threshold for confidence score.

(Pending results in Linear... Table)

The results are disappointing. As we can see in the table, RNN classifier cannot classify very well because, probably, the duration of the videos are so small so we stopped using it in jHMDB dataset. In the Linear classifier, we noticed that every tube is considered as background tube. That means that Linear classifier gets overfitted with trained data and cannot handle unknown data. So, we thought that we need a classifier which can l̈earnv̈ery easily, with little data. So we chose to try a support vector machine classifier.

1.3 Support Vector Machine (SVM)

1.3.1 First steps

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. **write more introduction and a pic, Pending...**

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 1.1 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			0.3	0.4	0.5
(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	1.04	1.04	1.04
			4	TODO	TODO	TODO
			5	TODO	TODO	TODO
(k,64,8,7,7)	(1,64,8,7,7)	max	1	TODO	TODO	TODO
			2	TODO	TODO	TODO
			3	TODO	TODO	TODO
			4	TODO	TODO	TODO
			5	TODO	TODO	TODO
(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	TODO	TODO	TODO
			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 1.1: 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

1.3.2 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 1.2

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	TODO	TODO	TODO
			2	TODO	TODO	TODO
			3	0.96	2.11	2.9
			4	2.4	4.53	5.16
			5	0.6	0.76	0.93
(k,64,8,7,7)	(1,64,8,7,7)	max	1	1.07	1.85	2.26
			2	3.2	3.39	3.66
			3	0.86	1.98	1.98
			4	1.24	2.68	3.09
			5	0.21	0.27	0.32
(k,256,8,7,7)	(1,256,8,7,7)	mean	1	10.62	10.94	10.94
			2	9.09	10.02	10.83
			3	9.05	9.65	9.69
			4	11.19	11.51	11.51
			5	4.84	6.13	6.13
(k,256,8,7,7)	(1,256,8,7,7)	max	1	**20.94**	**24.96**	**26.46**
			2	14.94	17.78	19.38
			3	14.9	17.39	19.88
			4	19.43	23.91	25.31
			5	10.41	10.46	11.29

Table 1.2: Our architecture's performance using 5 different policies and 2 different feature maps extracted using modified Roi Align.

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.

1.3.3 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.

Dimensions		**Pooling**	**Type**	**mAP precision**		
before	after			0.3	0.4	0.5
k,256,8,7,7	2,256,8,7,7	mean	1	TODO	TODO	TODO
			2	TODO	TODO	TODO
			3	TODO	TODO	TODO
			4	TODO	TODO	TODO
			5	TODO	TODO	TODO
k,256,8,7,7	2,256,8,7,7	max	1	25.07	26.91	29.11
			2	TODO	TODO	TODO
			3	TODO	TODO	TODO
			4	TODO	TODO	TODO
			5	TODO	TODO	TODO

Table 1.3: mAP results for second approach using temporal pooling

1.3.4 Increasing sample duration to 16 frames

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

Table 1.4: mAP results for

1.3.5 Adding more groundtruth tubes

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 8. Table 1.5

Dimensions		**Pooling**	**Type**	**mAP precision**		
before	after			0.3	0.4	0.5
k,64,8,7,7	2,64,7,7	max	1	TODO	TODO	TODO
			4	TODO	TODO	TODO
k,256,8,7,7	2,256,7,7	max	1	TODO	TODO	TODO
			4	TODO	TODO	TODO

Table 1.5: Results after increasing the number of total tubes

1.4 MultiLayer Perceptron (MLP)

Last but not least approach is a Multilayer perceptorn (MLP). More specifically, we extract the 3 last residulal layers of 3D ResNet34 and we add a classification layer.

1.4.1 Regular training

tpn - freeze features, don't know

1.4.2 Extract features

1.5 Classifying dataset UCF

As mentioned before, all the above results are from JHMDB dataset.

1.6 Final Improvements

After classification, we relize that a lot of classified tubes overlap and represent the same action. So, we use again NMS algorithm in order to remove unnecessary tubes. The new model can be seen in figure 1.3.

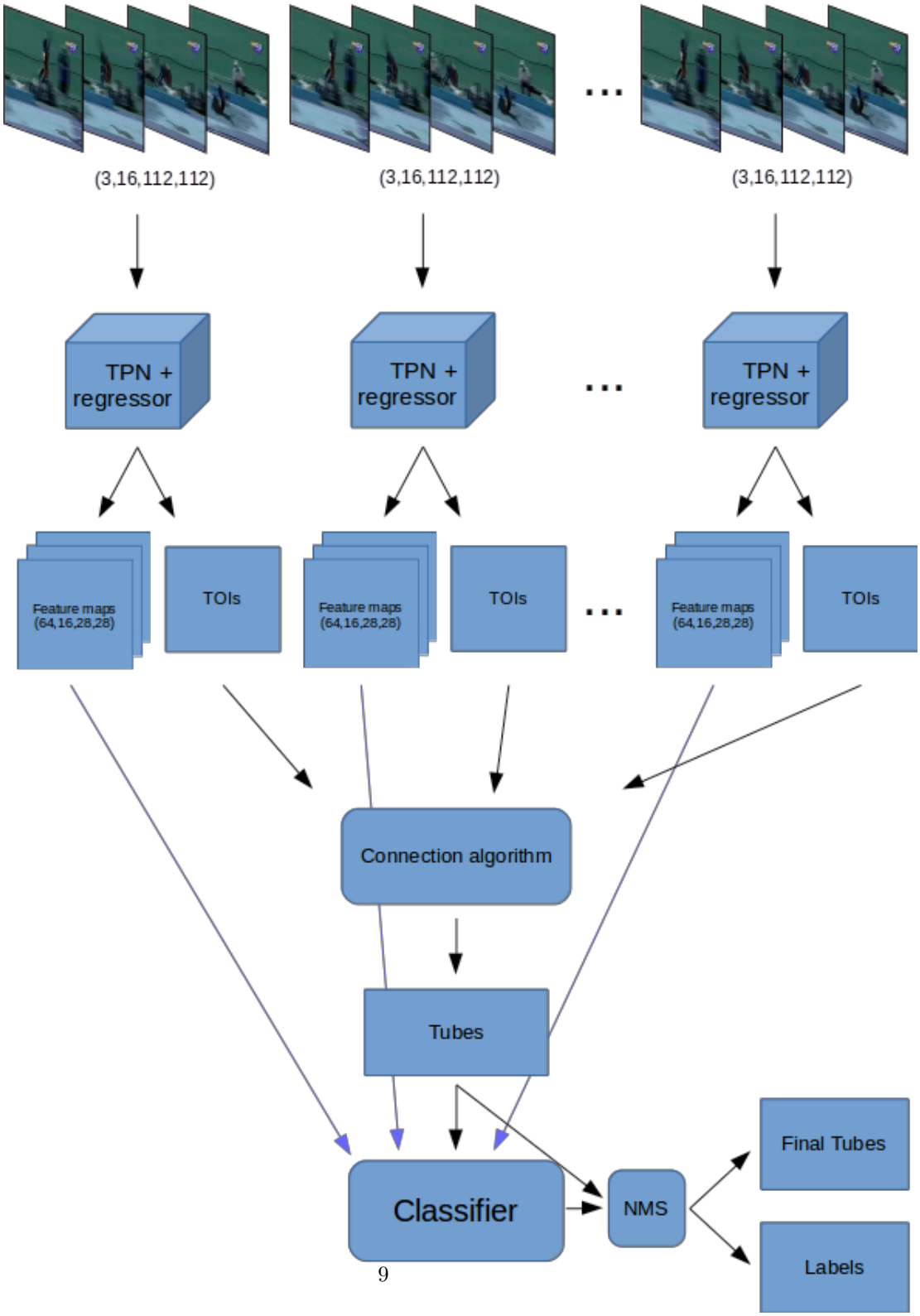

Figure 1.3: Structure of the network with NMS