Chapter 1

Tube Proposal Network

One of the basic elements of ActionNet is **Tube Proposal Network**(TPN). The main purpose of this network is to propose **Tube of Interest**(TOIs). These tubes are likely to contain an known action and are consisted of some 2D boxes (1 for each frame). TPN is inspired from RPN introduced by FasterRCNN ([**?**]), but instead of images, TPN is used in videos as show in [**?**]. In full correspondence with RPN, the structure of TPN is similar to RPN. The only difference, is that TPN uses 3D Convolutional Layers and 3D anchors instead of 2D.

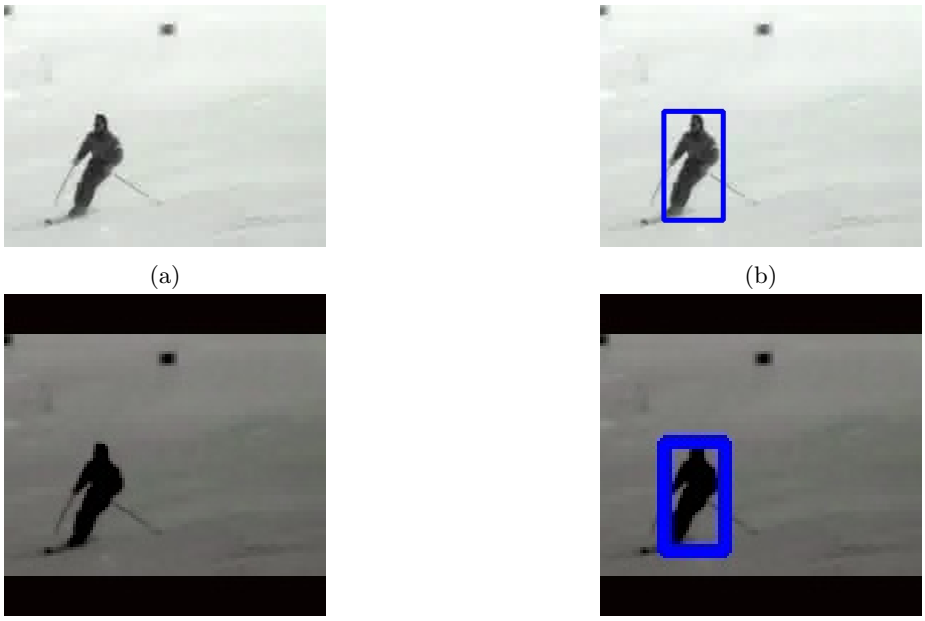
We designed 2 main structures for TPN. Each approach has a different definition of the used 3D anchors. The rest structure of the TPN is mainly the same with some little differences in the regression layer.

Before describing TPN, we present the preprocess procedure which is the same for both approaches.

1.1 Preparation for TPN

1.1.1 Preparing data

Our architecture gets as input a sequnece of frames which has a fixed size in widht, height and duration. However, each video has different resolution. That's creates the need to resize each frame before. As mentioned in previous chapter, the first element of our network is a 3D RenNet taken from [**?**]. This network is designed to get images with dimensions (112,112). As a result, we resize each frame from datasets' videos into (112,112) frames. In order to keep aspect ratio, we pad each frame either left and right, either above and bellow depending which dimension is bigger. In figure 1.1 we can see the original frame and the resize and padded one. In full correspondance, we resize the groundtruth bounding boxes for each frame (figure 1.1b and 1.1d show that).



(a) (b)

(c) (d)

Figure 1.1: At (a), (b) frame is its original size and at (c), (d) same frame after preprocessing part

1.1.2 3D ResNet

Before using Tube Proposal Network, we spatio-temporal features from the video. In order to do so, we extract the 3 first Layers of a pretrained 3D ResNet. It is pretrained in Kinetics dataset [**?**] for sample duration = 16 and sample size = (112,122).

This network normally is used for classifying the whole video, so some of its layers use temporal stride = 2. We set their temporal stride equal to 1 because we don't want to miss any temporal information during the process. So, the output of the third layer is a feature maps with dimesions (256,16,7,7). We feed this feature map to TPN, which is described in following sections.

1.2 3D anchors as 6-dim vector

1.2.1 First Description

We started desinging our TPN inspired by [**?**]. We consider each anchor as a 3D bounding box written as $(x_1, y_1, t_1, x_2, y_2, t_2)$ where x_1, y_1, t_1 are the upper front left coordinates of the 3D and x_2, y_2, t_2 are the lower back left as shown in figure 1.2.

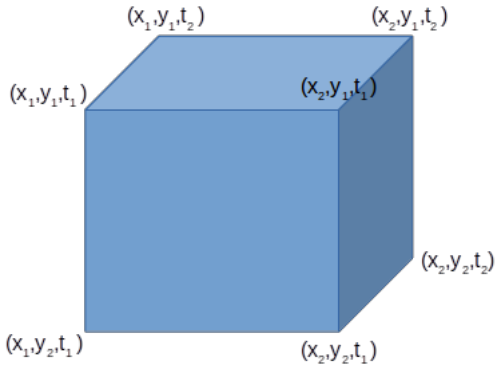


Figure 1.2: An example of the anchor $(x_1, y_1, t_1, x_2, y_2, t_2)$

The main advantage of this approach is that except from x-y dims, dimension of time is mutable. As a result, the proposed TOIs have no fixed time duration. This will help us deal with untrimmed videos, because proposed TOIs would exclude background frames. For this approach, we use **n=4k=60** anchors for each pixel in the feature map of TPN. We have k anchors for each sample duration(5 scales of 1, 2, 4, 8, 16, 3 aspect ratios of 1:1, 1:2, 2:1 and 4 durations of 16,12,8,4 frames). In [**?**], network's anchors are defined according to the dataset most common anchors. This, however, creates the need to redesign the network for each dataset. In our approach, we use the same anchors for both datasets, because we want our network not to be dataset-specific but to be able to generalize for several datasets. As sample duration, we chose 16 frames per video segment because our pre-trained ResNet is trained for video clips with that duration. So the structure of TPN is:

- 1 3D Convolutional Layer with kernel size = 3, stride = 3 and padding = 1
- 1 classification layer outputs *2n scores* whether there is an action or not for *n tubes*.
- 1 regression layer outputs *6n coordinates* $(x_1, y_1, t_1, x_2, y_2, t_2)$ for *n tubes*.

which is shown in figure 1.3

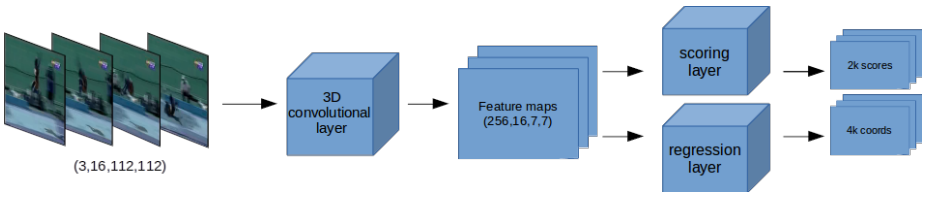


Figure 1.3: Structure of TPN

The output of TPN is the k-best scoring cuboid, in which it is likely to contain an action.

1.2.2 Training

As mentioned before, TPN extracts TOIs as 6-dim vectors. For that reason, we modify out groundtruth ROIs to groundtruth Tubes. We take for granted that the actor cannot move a lot during 16 frames, so that's why we use this kind of tubes. As shown in figure 1.4, these tubes are 3D boxes which include all the groundtruth rois, which are different for each frame.

Figure 1.4: Groundtruth tube is coloured with blue and groundtruth rois with colour green

For training procedure, for each video, we randomly select a part of it which has duration 16 frames. For each video, we train TPN in order to score all the anchors using IoU criterion (as explained in chapter 2 when 3D boxes are cubboids) and we use Cross Entropy Loss as a loss function. For regression, we use smooth-L1 loss. For regression targets, we use pytorch FasterRCNN implementation ([**?**]) for bounding box regression. We modified the code in order to extend it for 3 dimensions. **TODO more details** So we have:

$$
\begin{array}{ccc}
t_x = (x - x_a)/w_a, & t_y = (y - y_a)/h_a, & t_z = (z - z_a)/d_a, \\
t_w = log(w/w_a), & t_h = log(h/h_a), & t_d = log(d/d_a), \\
t_x^* = (x^* - x_a)/w_a, & t_y^* = (y^* - y_a)/h_a, & t_z^* = (z^* - z_a)/d_a, \\
t_w^* = log(w^*/w_a), & t_h^* = log(h^*/h_a), & t_d^* = log(d^*/d_a),
\end{array}
$$

where x, y, z, w, h, d denote the 3D box's center coordinates and its widht, height and duration. Variables $x, x_a,$ and x^* are for the predicted box, anchor box, and groundthruth box respectively (likewise for y, z, w, h, d).

TODO Training Loss formula

1.2.3 Validation

Validation procedure is a bit similar to training procedure. We randomly select 16 frames from a validation video and we examine if there is at least 1 proposed TOI which overlaps ≥ 0.5 with each groundtruth action tube and

we get recall score. In order to get good proposals, after TPN we use Non-Maximum Suppresion (NMS) algorith. We set NMS threshold equal with 0.7, so we reject thems with overlapping score ¿ 0.7 .

1.2.4 Modified Intersection over Union(mIoU) TODO check again

During training, we get numerous anchors. We have to classify them as foreground anchors or background anchors. Foreground anchors are those which contain some action, and, respectively, background don't. As presented before, IoU for cuboids calculates the ratio between volume of overlap and volume of union. Intuitively, this criterion is good for evaluating 2 tubes if they overlap but it has one big drawback: it considers x-y dimesions to have same importance with time dimension, which we do not desire. That's becase firstly we care to be accurate in time dimension, and then we can fix x-y domain. As a result, we change the way we calculate the Intesection Over Union. We calculate seperately the IoU in x-y domain (IoU-xy) and in t-domain (IoU-t). Finally, we multiply them in order to get the final IoU. So the formula for 2 tubes $(x_1, y_1, t_1, x_2, y_2, t_2)$ and $(x'_1, y'_1, t'_1, x'_2, y'_2, t'_2)$ is:

$$IoU_{xy} = \frac{\text{Area of Overlap in x-y}}{\text{Area of Union in x-y}}$$

$$IoU_t = \frac{max(t_1, t'_1) - min(t_2, t'_2)}{min(t_1, t'_1) - max(t_2, t'_2)}$$

$$IoU = IoU_{xy} \cdot IoU_t$$

The above criterion help us balance the impact of time domain in IoU. For example, let us consider 2 anchors: a = (22, 41, 1, 34, 70, 5) and b = (20, 45, 2, 32, 72, 5). These 2 anchors in x-y domain have IoU score equal to 0.61. But they are not exactly overlaped in time dim. Using the first approach we get 0.5057 IoU score and using the second approach we get 0.4889. So, the second criterion would reject this anchor, because there is a difference in time duration.

In order to verify our idea, we train TPN using both IoU and mIoU criterion for tube-overlapping. At Table 1.1 we can see the performance in each case for both datasets, JHMDB and UCF. Recall threshold for this case is 0.5 and during validation, we use regular IoU for defining if 2 tubes overlap.

Dataset	**Criterion**	**Recall(0.5)**
JHMDB	IoU	TODO
	mIoU	TODO
UCF	IoU	TODO
	mIoU	TODO

Table 1.1: Recall results for both datasets using IoU and mIoU metrics

Table 1.1 shows that modified-IoU give us slightly better recall performance, so that's the overlapping scoring policy for now on, during Training mode.

1.2.5 Improving TPN score

After first test, we came with the idea that in a video lasting 16 frames, in time domain, all kind of actions can be seperated in the following categories:

1. Action starts in the n-th frame and finishes after the 16th frame of the sampled video.
2. Action has already begun before the 1st frame of the video and ends in the n-th frame.
3. Action has already begun before the 1st frame of the video and finishes after the 16th video frame.
4. Action starts and ends in that 16 frames of the video.

On top of that, we noticed that most of actions, in our datasets, last more that 16 frame. So, we added 1 scoring layer and 1 reggression layer as shown in figure 1.5. These two layers have anchors with fixed time duration. Their purpose is to be trained only in x-y domain, keeping time duration steady.

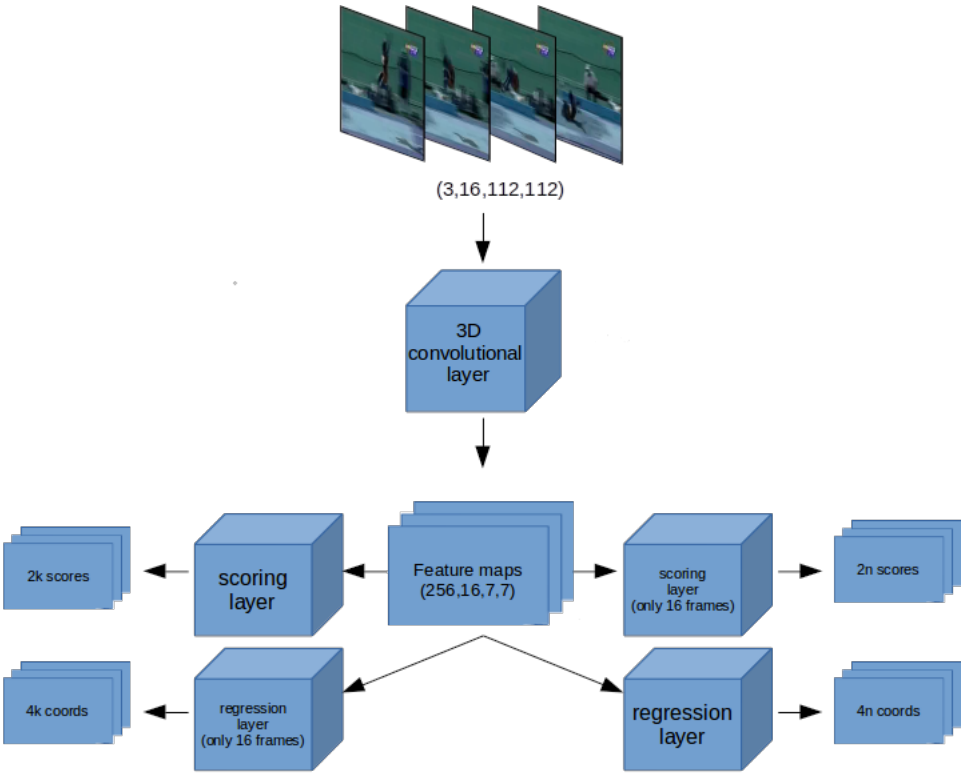


Figure 1.5: TPN structure after adding 2 new layers, where k = 5n.

Training and Validation procedures remain the same. The only big difference is that now we have from 2 difference system proposed TOIs. So, we first concate them and, then, we follow the same procedure. For training loss, we have 2 different cross-entropy losses and 2 different smooth-L1 losses, each for every layer correspondly.

TODO Training Loss formula

2 approaches Kernel vs mean

Dataset	Fix-time anchors	Type	Recall(0.5)
JHMDB	No	-	TODO
	Yes	Kernel	TODO
		Mean	TODO
UCF	No	-	TODO
	Yes	Kernel	TODO
		Mean	TODO

Table 1.2: Recall results after adding fixed time duration anchors

AS we can see from the previous results, the new layers increased recall performance significantly.

1.2.6 Adding regressor

The output of TPN is α-highest scoring anchors moved according to their regression prediction. After that, we have to translate the anchor into tubes. In order to do so, we add a regressor system which gets as input TOIs' feature maps and returns a sequence of 2D boxes, each for every frame. The only problem is that the regressor needs a fixed input size of featuremaps. This problem is already solven by R-CNNs which use roi pooling and roi align in order to get fixed size feature maps from ROIs with changing sizes. In our situation, we extend roi align operation, presented by Mask R-CNN, and we call it **3D Roi Align**.

3D Roi Align 3D Roi align is a modification of roi align presented by Mask R-CNN ([**?**]). The main difference between those two is that Mask R-CNN's roi align uses bilinear interpolation for extracting ROI's features and ours 3D roi align uses trilinear interpolation for the same reason. Again, the 3rd dimension is time. So, we have as input a feature map extracted from ResNet34 with dimensions (64,16,28,28) and a tensor containing the proposed TOIs. For each TOI whose actiovation map whose size is (64,16,7,7), we get as output a feature map with size (64, 16, 7, 7).

Regression procedure

At first, for each proposed ToI, we get its corresponding activation maps using 3D Roi Align. These features are given as input to a regressor. This regressor returns $16 \cdot 4$ predicted transforms $(\delta_x, \delta_y, \delta_w, \delta_h)$, 4 for each frame, where δ_x, δ_y specify the coordinates of proposal's center and δ_w, δ_h its width and height, as specified in [**?**]. We keep only the predicted translations, for the frames that are $\geq t_1$ and $< t_2$ and for the other frames, we set a zero-ed 2D box. After that, we modify each anchor from a cuboid written like $(x_1, y_1, t_1, x_2, y_2, t_2)$ to a sequence of 2D boxes, like:
$(0, 0, 0, 0, ..., x_{T_1}, y_{T_1}, x'_{T_1}, y'_{T_1}, ..., x_i, y_i, x'_i, ..., x_{T_2}, y_{T_2}, x'_{T_2}, y'_{T_2}, 0, 0, 0, 0, ...)$, where

- $T_1 \leq i \leq T_2$, for $T_1 < t_1 + 1, T_2 < t_2$ and $T_1, T_2 \in \mathbb{Z}$
- $x_i = x_1, y_i = y_1, x'_i = x_2, y'_i = y_2$.

Training mIou, 16mean After getting proposed TOIs from TPN, we pick, randomly, 16 tubes which will be input in the regressor. Finally, we find the traslation for each rois and, again, we use smooth-L1 loss for loss function.

TODO Training Loss formula

First regression Network

Na pw gia to normalize kai to pooling kernel

The architecture of reggression network is show in Figure 1.6, and it is described below:

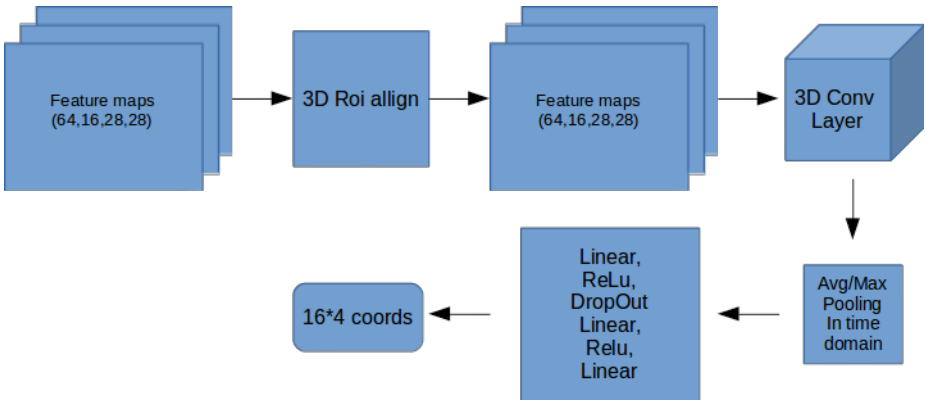


Figure 1.6: Structure of Regressor

1. Regressor is consisted, at first, with a 3D convolutional layer with kernel = 1, stride = 1 and no padding. This layer gets as input ToI's activation map extracted by 3D Roi Align.

2. After that, we calculate the average value in time domain, so from a feature map with dimensions (64,16,7,7), we get as output a feature map (64,7,7).

3. These feature maps are given as input to a Linear layer followerd by a Relu Layer, a Dropout Layer, another Linear Layer and Relu Layer and a final Linear.

NA DW TA FEATURE MAPS POIA XRISIMOPOIISA

Dataset	**Pooling**	Recall
JHMDB	avg	TODO
	mean	TODO
UCF	avg	TODO
	mean	TODO

Table 1.3

TODO sxoliasmos apotelesmatwn As the above results show, when we translate a TOI into a sequence of ROIs, recall reduces about 20-30%, which is a big problem.

1.2.7 Changing Regressor - from 3D to 2d

After getting first recall results, we experiment using another architecture for the regressor network. Instead of having a 3D Convolutional Layer, we will use a 2D Convolutional Layer in order to treat the whole time dimension as one during convolution operation. So, as shown in Figure **??**, the 2^{nd} Regression Network is about the same with first one, with 2 big differences:

1. We performing a pooling operation at the feature maps extracted by 3D Roi Align operation

2. Instead of a 3D Convolutional Layer, we have a 2D Convolutional Layer with kernel size = 1, stride = 1 and no padding.

On top of that, we tried to determine which feature map is the most suitable for getting best-scorig recall performance. This feature map will be given as input to Roi Algin operation. At Table 1.4, we can see the recall performance for different feature maps and different pooling methods.

TODO add more comment As we noticed from the above results, our system has difficulty in translating 3D boxes into 2D sequence of ROIs. So, that makes us rethink the way we designed our TPN.

1.3 3D anchors as 4k-dim vector

In this approach, we set 3D anchors as 4k coordinates (k = 16 frames = sample duration). So a typical anchor is written as $(x_1, y_1, x'_1, y'_1, x_2, y_2, ...)$

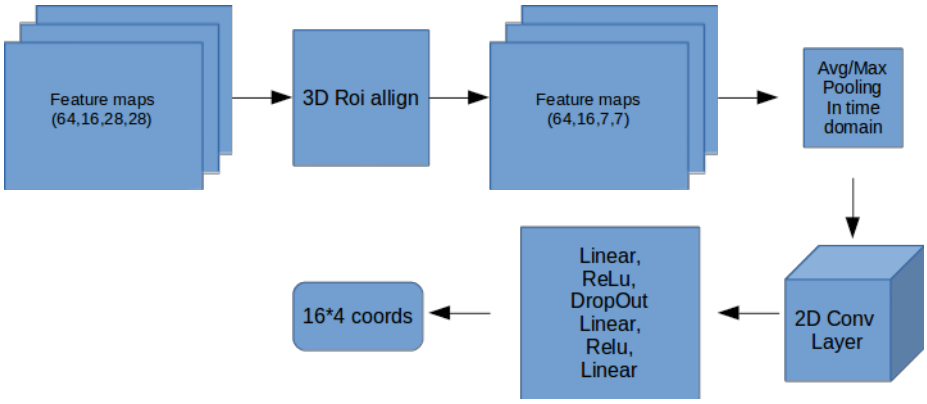


Figure 1.7: Structure of Regressor

Dataset	**Pooling**	**Feature Map**	**Recall (0.5)**
JHMDB	mean	64	TODO
		128	TODO
		256	TODO
	max	64	TODO
		128	TODO
		256	TODO
UCF	mean	64	TODO
		128	TODO
		256	TODO
	max	64	TODO
		128	TODO
		256	TODO

Table 1.4

where x_1, y_1, x'_1, y'_1 are the coordinates for the 1st frame, x_2, y_2, x'_2, y'_2 are the coordinates for the 2nd frame etc as presented in [**?**]. In figure 1.8 we can an example of this type of anchor.

The main advantage of this approach is that we don't need to translate the 3D anchors into 2D boxes. However, it has a big drawback, which is the fact that this type of anchors has fixed time duration. In order to deal with this problem, we set anchors with different time durations, which are 16, 12, 8 and 4. Anchors with duration $<$ sample duration (16 frames) can be written as 4k vector with zeroed coordinateds in the frames bigger that the time duration. For example, an anchor with 2 frames duration, starting from the 2nd frame and ending at the 3rd can be written as (0, 0, 0, 0, $x_1, y_1, x'_1, y'_1, x_2, y_2, x'_2, y'_2$, 0, 0, 0, 0) if sample duration is 4 frames.

This new approach led us to change the structure of TPN. The new one can is presented in figure 1.9. As we can see, we added scoring and regression

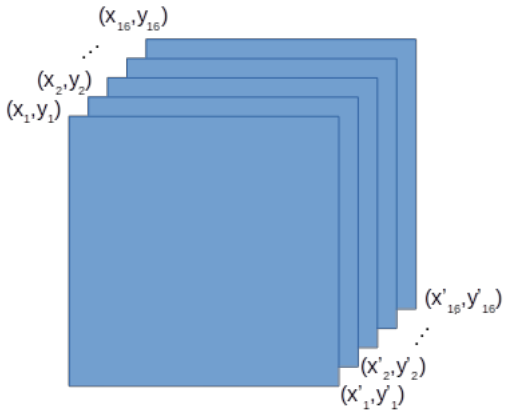


Figure 1.8: An example of the anchor $(x_1, y_1, x'_1, y'_1, x_2, y_2, ...)$

layers for each duration.

1.3.1 Training

In following figures we can see recall performance for sample duration = 16 when using max or avg pooling. From the above results, we can see that using max pooling achieves better results.

Dataset	Pooling	Recall(0.5)
JHMDB	mean	TODO
	max	TODO
UCF	mean	TODO
	max	TODO

Table 1.5: Recall results after adding fixed time duration anchors

1.3.2 Adding regressor

In full correspondance with the previous approach, we added an regressor for trying to find better results. We

1.3.3 Changing sample duration

After trying all the previous version, we noticed that we get about the same recall performances. So, we thought that we could try to reduce the sample duration. On top of that, we trained our network for sample duration = 8 and 4 frames.

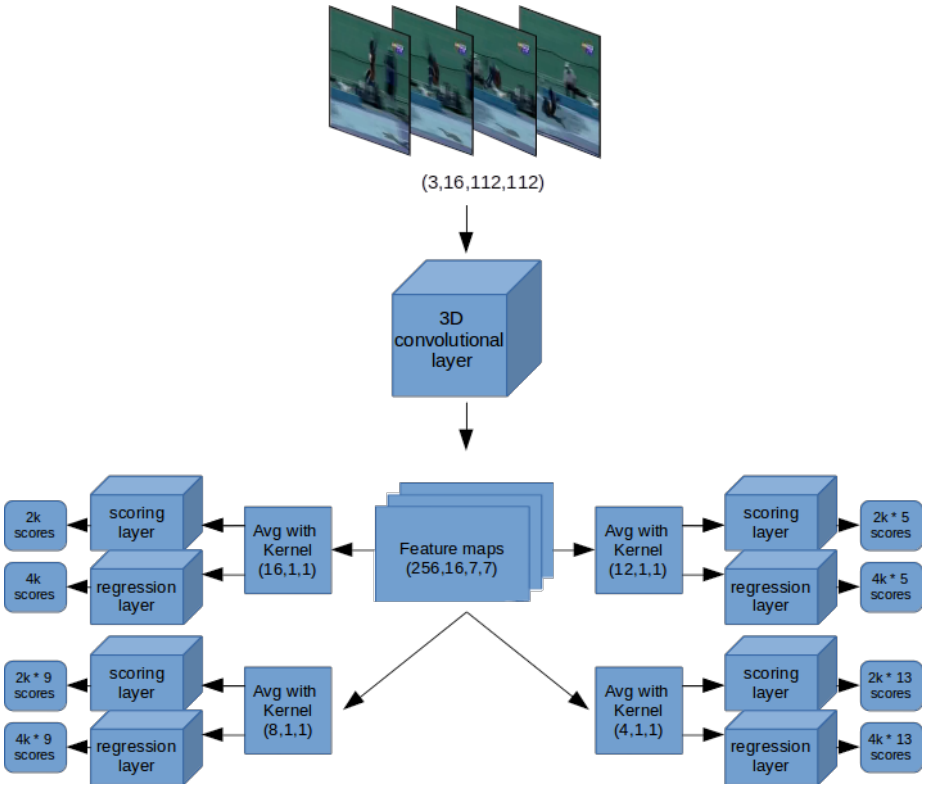


Figure 1.9: The structure of TPN according to new approach

Dataset	**Pooling**	**Feature Map**	**Recall (0.5)**
JHMDB	mean	64	TODO
		128	TODO
		256	TODO
	max	64	TODO
		128	TODO
		256	TODO
UCF	mean	64	TODO
		128	TODO
		256	TODO
	max	64	TODO
		128	TODO
		256	TODO

Table 1.6

Dataset	**Sample dur**	**Recall**
JHMDB	8	TODO
	4	TODO
UCF	8	TODO
	4	TODO

Table 1.7