



National Technical University of Athens
School of Electrical and Computer Engineering

Action Localization and Recognition in Videos

DIPLOMA PROJECT

EFSTATIOS GALANAKIS

Supervisor :

Athens, December 2019



National Technical University of Athens
School of Electrical and Computer Engineering

Action Localization and Recognition in Videos

DIPLOMA PROJECT

EFSTATIOS GALANAKIS

Supervisor :

Approved by the examining committee on the December -1, 2019.

.....

Athens, December 2019

.....
Efstathios Galanakis

Electrical and Computer Engineer

Copyright © Efstathios Galanakis, 2019.
All rights reserved.

This work is copyright and may not be reproduced, stored nor distributed in whole or in part for commercial purposes. Permission is hereby granted to reproduce, store and distribute this work for non-profit, educational and research purposes, provided that the source is acknowledged and the present copyright message is retained. Enquiries regarding use for profit should be directed to the author.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the National Technical University of Athens.

Abstract

The purpose of this diploma dissertation is on one hand the design of a simple high-level language that supports programming with proofs, and on the other hand the implementation of a compiler for this language. This compiler will produce code for an intermediate-level language suitable for creating certified binaries.

The need for reliable and certifiably secure code is even more pressing today than it was in the past. In many cases, security and software compatibility issues put in danger the operation of large systems, with substantial financial consequences. The lack of a formal way of specifying and proving the correctness of programs that characterizes current programming languages is one of the main reasons why these issues exist. In order to address this problem, a number of frameworks with support for certified binaries have recently been proposed. These frameworks offer the possibility of specifying and providing a formal proof of the correctness of programs. Such a proof can easily be checked for validity before running the program.

The frameworks that have been proposed are intermediate-level in nature, thus the process of programming in these is rather cumbersome. The high-level languages that accompany some of these frameworks, while very expressive, are hard to use. A simpler high-level language, like the one proposed in this dissertation, would enable further use of this programming idiom.

In the language we propose, the programmer specifies the partial correctness of a program by annotating function definitions with pre- and post-conditions that must hold for their parameters and results. The programmer also provides a set of theorems, based on which proofs of the proper implementation and use of the functions are constructed. An implementation in OCaml of a compiler from this language to the NFLINT certified binaries framework was also completed as part of this dissertation.

We managed to keep the language close to the feel of the current widespread functional languages, and also to fully separate the programming stage from the correctness-proving stage. Thus an average programmer can program in a familiar way in our language, and later an expert on formal logic can prove the semi-correctness of a program. As evidence of the practicality of our design, we provide a number of examples in our language with full semi-correctness proofs.

Key words

Programming languages, Programming with proofs, Secure programming languages, Certified code.

Contents

Abstract	5
Contents	7
List of Tables	9
List of Figures	11
1. Introduction	13
1.1 Problem statement	13
1.2 Motivation	13
1.3 Contributions	13
1.4 Thesis structure	13
2. Background Theory	15
2.1 Machine Learning	15
2.1.1 Introduction	15
2.1.2 Neural Networks	17
2.1.3 A single Neuron	17
2.1.4 2D Convolutional Neural Network	20
2.1.5 3D Convolutional Neural Network	20
2.2 Object Detection	21
2.2.1 Region Proposal Network	22
2.3 Action Recognition and Localization	22
2.3.1 Action Recognition	22
2.3.2 Action Localization	22
3. Related work	25
4. Tube Proposal Network	27
4.1 3D anchors as 6-dim vector	27
4.1.1 First Description	27
4.1.2 Training	28
4.1.3 Adding regressor	29
4.1.4 Validation	30
4.1.5 Improving TPN score	30
4.1.6 Changing Regressor	30
4.1.7 Changing training procedure	31
4.2 3D anchors as 4k-dim vector	31

5. Connecting Tubes	35
5.1 Description	35
5.2 Some results	36
5.2.1 JHMDB dataset	36
5.2.2 UCF dataset	36
6. Classification stage	37
6.1 Description	37
6.1.1 Linear Classifier	37
6.1.2 RNN Classifier	37
7. Conclusion	39
Bibliography	41

List of Tables

List of Figures

2.1	Example of supervised Learning	15
2.2	Example of unsupervised Learning	16
2.3	Example of Reinforcement Learning	17
2.4	An example of a single Neuron	18
2.5	Plots of Activation functions	19
2.6	An example of a Feedforward Neural Network	19
2.7	Typical structure of a ConvNet	20
2.8	Convolution with kernel size:3, stride:2, padding:1	21
2.9	3D Convolution operation	21
2.10	Region Proposal Network's structure	22
2.11	Anchors for pixel (320,320) of an image (600,800)	23
4.1	An example of the anchor $(x_1, y_1, t_1, x_2, y_2, t_2)$	27
4.2	Structure of TPN	28
4.3	Groundtruth tube is coloured with blue and groundtruth rois with colour green	28
4.4	Structure of Regressor	29
4.5	TPN structure after adding 2 new layers, where $k = 5n$	31
4.6	An example of the anchor $(x_1, y_1, x'_1, y'_1, x_2, y_2, \dots)$	32
4.7	The structure of TPN according to new approach	33
5.1	An example of calculating connection score for 3 random TOIs	35

Chapter 1

Introduction

Nowadays, the enormous increase of computing power help us deal with a lot of difficult situations appeared in our daily life. A lot of areas of science have managed to tackle with problems, which were considered non trivial 20 years ago. One of these area is Computer Vision and an import problem is human action recognition and localization.

1.1 Problem statement

The area of human action recognition and locatization has 2 main goals:

1. Automatically detect and classify any human activity, which appears in a video
2. Automatically locate in the video, where the previous action is performed.

1.2 Motivation

1.3 Contributions

1.4 Thesis structure

Chapter 2

Background Theory

2.1 Machine Learning

2.1.1 Introduction

Machine Learning (ML) is a field which is raised out of Artificial Intelligence (AI). Applying AI, we wanted to build better and intelligent machines. But except for few mere tasks such as finding the shortest path between point A and B, we were unable to program more complex and constantly evolving challenges. There was a realisation that the only way to be able to achieve this task was to let machine learn from itself. This sounds similar to a child learning from its self. So machine learning was developed as a new capability for computers. And now machine learning is present in so many segments of technology, that we don't even realise it while using it.

Finding patterns in data on planet earth is possible only for human brains. The data being very massive, the time taken to compute is increased, and this is where Machine Learning comes into action, to help people with large data in minimum time.

There are three kinds of Machine Learning Algorithms :

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

Supervised Learning

A majority of practical machine learning uses supervised learning. In supervised learning, the system tries to learn from the previous examples that are given. Speaking mathematically, supervised learning is where you have both input variables (x) and output variables (Y) and can use an algorithm to derive the mapping function from the input to the output. The mapping function is expressed as $Y = f(x)$.

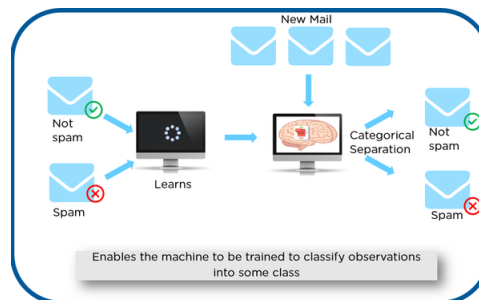


Figure 2.1: Example of supervised Learning

As shown in Figure 2.1, we have initially taken some data and marked them as 'Spam' or 'Not Spam'. This labeled data is used by the training supervised model, in order to train the model. Once

it is trained, we can test our model by testing it with some test new mails and checking of the model is able to predict the right output.

Supervised learning problems can be further divided into two parts, namely **classification**, and **regression**.

Classification : A classification problem is when the output variable is a category or a group, such as “black” or “white” or “spam” and “no spam”.

Regression : A regression problem is when the output variable is a real value, such as “Rupees” or “height.”

Some Supervised learning algorithms include:

- Decision trees
- Support-vector machine
- Naive Bayes classifier
- k-nearest neighbors
- linear regression

Unsupervised Learning

In unsupervised learning, the algorithms are left to themselves to discover interesting structures in the data. Mathematically, unsupervised learning is when you only have input data (X) and no corresponding output variables. This is called unsupervised learning because unlike supervised learning above, there are no given correct answers and the machine itself finds the answers. In Figure 2.2, we

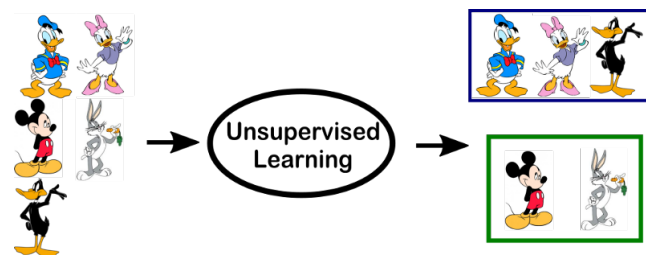


Figure 2.2: Example of unsupervised Learning

have given some characters to our model which are ‘Ducks’ and ‘Not Ducks’. In our training data, we don’t provide any label to the corresponding data. The unsupervised model is able to separate both the characters by looking at the type of data and models the underlying structure or distribution in the data in order to learn more about it. Unsupervised learning problems can be further divided into **association** and **clustering** problems.

Association : An association rule learning problem is where you want to discover rules that describe large portions of your data, such as “people that buy X also tend to buy Y”.

Clustering : A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behaviour.

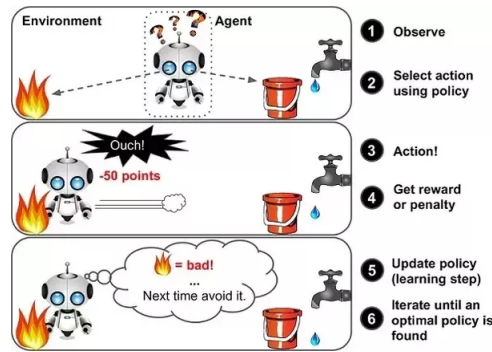


Figure 2.3: Example of Reinforcement Learning

Reinforcement Learning

A computer program will interact with a dynamic environment in which it must perform a particular goal (such as playing a game with an opponent or driving a car). The program is provided feedback in terms of rewards and punishments as it navigates its problem space. Using this algorithm, the machine is trained to make specific decisions. It works this way: the machine is exposed to an environment where it continuously trains itself using trial and error method. In Figure 2.3, we can see that the agent is given 2 options i.e. a path with water or a path with fire. A reinforcement algorithm works on reward a system i.e. if the agent uses the fire path then the rewards are subtracted and agent tries to learn that it should avoid the fire path. If it had chosen the water path or the safe path then some points would have been added to the reward points, the agent then would try to learn what path is safe and what path isn't

2.1.2 Neural Networks

Neural Networks are a class of models within the general machine learning literature. Neural networks are a specific set of algorithms that have revolutionized the field of machine learning. They are inspired by biological neural networks and the current so called deep neural networks have proven to work quite very well. Neural Networks are themselves general function approximations, that is why they can be applied to literally almost any machine learning problem where the problem is about learning a complex mapping from the input to the output space.

2.1.3 A single Neuron

The basic unit of computation in a neural network is the neuron, often called a **node** or **unit**. It receives input from some other nodes, or from an external source and computes an output. In purely mathematical terms, a neuron in the machine learning world is a placeholder for a mathematical function, and its only job is to provide an output by applying the function on the inputs provided. Each input has an associated weight (w), which is assigned on the basis of its relative importance to other inputs. The node applies a function f (defined below) to the weighted sum of its inputs as shown in Figure 2.4. The network takes numerical inputs $X1$ and $X2$ and has weights $w1$ and $w2$ associated with those inputs. Additionally, there is another input I with weight b (called *Bias*) associated with it. The main function of Bias is to provide every node with a trainable constant value (in addition to the normal inputs that the node receives). The output Y from the neuron is computed as shown in the Figure 2.4. The function f is non-linear and is called **Activation Function**. The purpose of the activation function is to introduce non-linearity into the output of a neuron. This is important because most real world data are non linear and we want neurons to learn these non-linear representations.

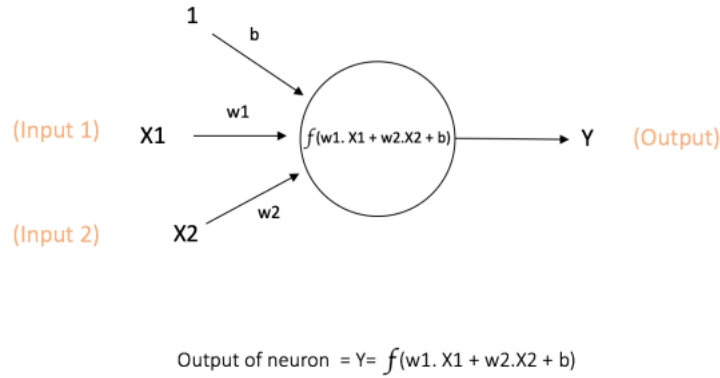


Figure 2.4: An example of a single Neuron

Activation Functions

Every activation function (or non-linearity) takes a single number and performs a certain fixed mathematical operation on it. There are several activation functions:

Sigmoid : takes a real-valued input and squashes it to range between 0 and 1. Its formula is:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

It is easy to understand and apply but it has major reasons which have made it fall out of popularity:

- Vanishing gradient problem
- Its output isn't zero centered. It makes the gradient updates go too far in different directions.
- Sigmoids saturate and kill gradients.
- Sigmoids have slow convergence.

Tanh : takes a real-valued input and squashes it to the range [-1, 1]. Its formula is:

$$\tanh(x) = 2\sigma(2x) - 1$$

Now its output is zero centered because its range is between -1 to 1. Hence optimization is easier in this method and in practice it is always preferred over Sigmoid function. But still it suffers from Vanishing gradient problem.

ReLU : ReLU stands for *Rectified Linear Unit*. It takes a real-valued input and thresholds it at zero (replaces negative values with zero). So its formula is:

$$f(x) = \max(0, x)$$

It has become very popular in the past couple of years. It was recently proved that it had 6 times improvement in convergence from Tanh function. Seeing the mathematical form of this function we can see that it is very simple and efficient. A lot of times in Machine learning and computer science we notice that most simple and consistent techniques and methods are only preferred and are best. Hence it avoids and rectifies vanishing gradient problem. Almost all deep learning Models use ReLU nowadays.

Figure 2.5 shows each of the above activation functions.

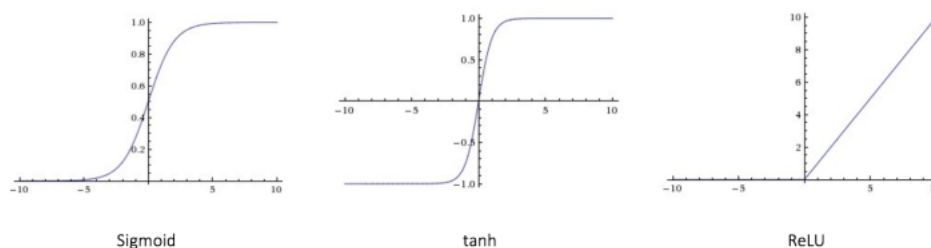


Figure 2.5: Plots of Activation functions

Feedforward Neural Network

Till now we have covered neuron and activation functions which together form the basic building blocks of any neural network. The feedforward neural network was the first and simplest type of artificial neural network devised. It contains multiple neurons (nodes) arranged in layers. A layer is nothing but a collection of neurons which take in an input and provide an output. Inputs to each of these neurons are processed through the activation functions assigned to the neurons. Nodes from adjacent layers have connections or edges between them. All these connections have weights associated with them. An example of a feedforward neural network is shown in Figure 2.6. A feedforward neural

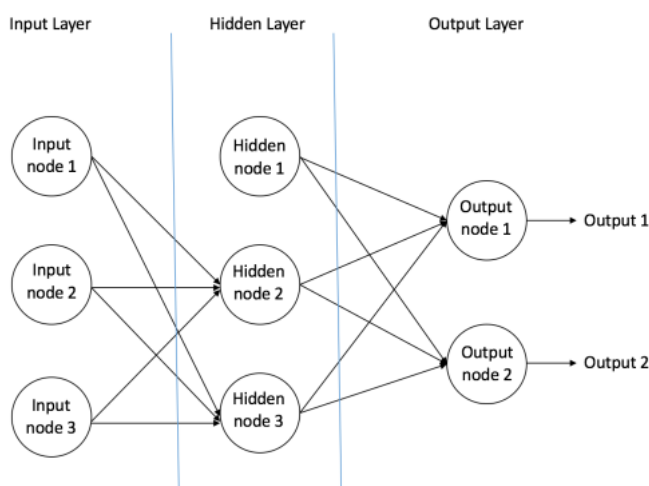


Figure 2.6: An example of a Feedforward Neural Network

network can consist of three types of nodes:

Input Nodes The Input nodes provide information from the outside world to the network and are together referred to as the “Input Layer”. No computation is performed in any of the Input nodes – they just pass on the information to the hidden nodes.

Hidden Nodes The Hidden nodes have no direct connection with the outside world (hence the name “hidden”). They perform computations and transfer information from the input nodes to the output nodes. A collection of hidden nodes forms a “Hidden Layer”. While a feedforward network will only have a single input layer and a single output layer, it can have zero or multiple Hidden Layers.

Output Nodes The Output nodes are collectively referred to as the “Output Layer” and are responsible for computations and transferring information from the network to the outside world.

In a feedforward network, the information moves in only one direction – forward – from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in

the network (this property of feed forward networks is different from Recurrent Neural Networks in which the connections between the nodes form a cycle). Another important point to note here is that each of the hidden layers can have a different activation function, for instance, hidden layer1 may use a sigmoid function and hidden layer2 may use a ReLU, followed by a Tanh in hidden layer3 all in the same neural network. Choice of the activation function to be used again depends on the problem in question and the type of data being used.

2.1.4 2D Convolutional Neural Network

A Convolutional Neural Network (ConvNet/CNN) is one of the variants of neural networks used heavily in the field of Computer Vision. It derives its name from the type of hidden layers it consists of. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers, and normalization layers. Here it simply means that instead of using the normal activation functions defined above, convolution and pooling functions are used as activation functions. It can take in an input image, assigning importance (learning weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to the other classification algorithms. While in primitive method filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the structure of the Visual Cortex. However, most ConvNets consist mainly in 2 parts:

- **Feature extractor :**

This part of the network takes as input the image and extract features that are meaningful for its classification. It amplifies aspects of the input that are important for discrimination and suppresses irrelevant variations. Usually, the feature extractor consists of several layers. For instance, an image which could be seen as an array of pixel values. The first layer often learns representations that represent the presence or absence of edges at particular orientations and locations in the image. The second layer typically detects motifs by spotting particular arrangements of edges, regardless of small variations in the edge positions. Finally, the third may assemble motifs into larger combinations that correspond to paths of familiar objects, and subsequent layers would detect objects as combinations of these parts.

- **Classifier :**

This part of the network takes as input the previously computed features and use them to predict the correct label.

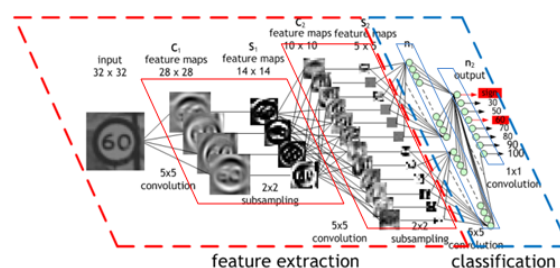


Figure 2.7: Typical structure of a ConvNet

In order to extract such features, ConvNets use 2D convolution operations.

2.1.5 3D Convolutional Neural Network

Traditionally, ConvNets are targeting RGB images (3 channels). The goal of 3D CNN is to take as input a video and extract features from it. When ConvNets extract the graphical characteristics



Figure 2.8: Convolution with kernel size:3, stride:2, padding:1

of a single image and put them in a vector (a low-level representation), 3D ConvNets extract the graphical characteristics of a set of images. 3D CNNs takes in to account a temporal dimension (the order of the images in the video). From a set of images, 3D CNNs find a low-level representation of a set of images, and this representation is useful to find the right label of the video (a given action is performed).

In order to extract such features, 3D ConvNets use 3D convolution operations.

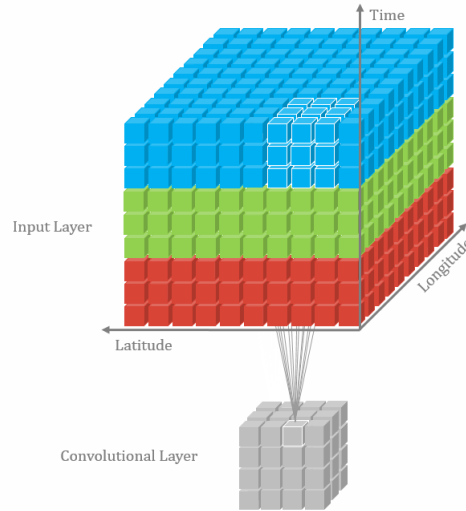


Figure 2.9: 3D Convolution operation

There are several existing approaches to tackle the video classification. This is a nonexhaustive list of existing approaches:

- **ConvNets + LSTM cell** : Extract features from each frame with a ConvNet, passing the sequence to an RNN
- **Temporal Relation Networks** : Extract features from each frame with a ConvNet and pass the sequence to an MLP
- **Two-Stream Convolutional Networks** : Use 2 CNN, 1 spatial stream ConvNet which process one single frame at a time, and 1 Temporal stream ConvNet which process multi-frame optical flow

2.2 Object Detection

Within the field of Deep Learning, the sub-discipline called “Object Detection” involves processes such as identifying the objects through a picture, video or a webcam feed. Object Detection is used

almost everywhere these days. The use cases are endless such as Tracking objects, Video surveillance, Pedestrian detection etc. An object detection model is trained to detect the presence and location of multiple classes of objects. For example, a model might be trained with images that contain various pieces of fruit, along with a label that specifies the class of fruit they represent (e.g. an apple, a banana, or a strawberry), and data specifying where each object appears in the image.

The main process followed by most of CNN for Object Detection is:

1. Firstly, we do feature extraction using as backbone network, the first Convolutional Layers of a known pre-trained CNN such as AlexNet, VGG, ResNet etc.
2. Then, we propose regions of interest (ROI) in the image. These regions contain possibly an object, which we are looking for.
3. Finally, we classify each proposed ROI.

2.2.1 Region Proposal Network

From the 3 above steps, the 2nd step is considered to be very important. That is because, in this step, we should choose regions of the image, which will be classified. Poor choice of ROIs means that the CNN will pass by some object that are located in the image, because, they were not be proposed to be classified.

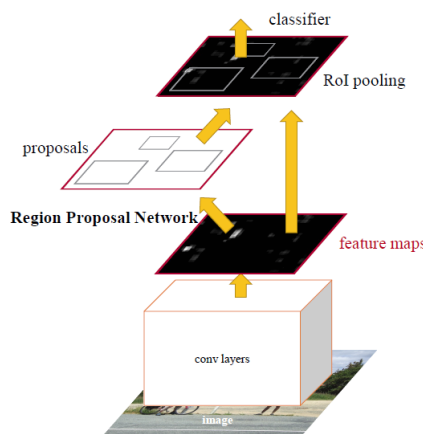


Figure 2.10: Region Proposal Network's structure

The first Object-Detection CNNs use several algorithms for proposing ROIs. For example, R-CNN[Girs13], and Fast R-CNN[Girs15] used Selective Search Algorithm for extracting ROIs. One of novelties introduced by the Faster R-CNN[Ren15] is **Region Proposal Network** (RPN). Its Function is to propose ROIs and its structure can be shown in 2.10. As we can see, RPN is consisted of:

- 1 2D Convolutional Layer
- 1 score layer
- 1 regression layer

Another basic element of RPN is the **anchors**. Anchors are predefined boxes used for extracting ROIs. In figure 2.11 is depicted an exaple of some anchors

2.3 Action Recognition and Localization

2.3.1 Action Recognition

2.3.2 Action Localization

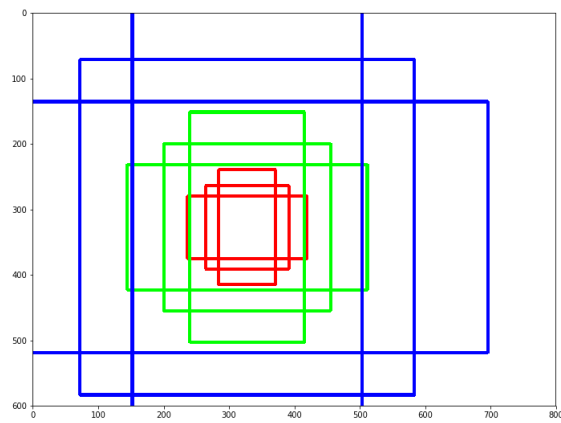


Figure 2.11: Anchors for pixel (320,320) of an image (600,800)

Chapter 3

Related work

Chapter 4

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 but for videos instead of images. In full correspondence with RPN, the structure of our TPN is similar to RPN. The only difference is that we use 3D convolutional layers and 3D anchors.

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

4.1 3D anchors as 6-dim vector

4.1.1 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 4.1.

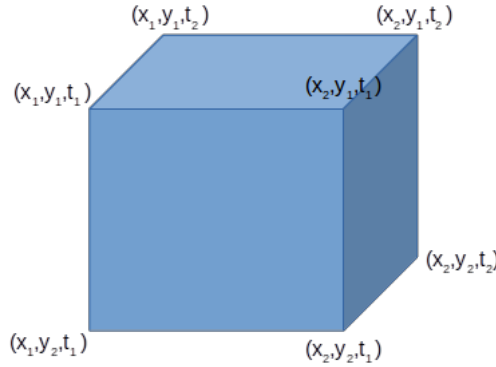


Figure 4.1: An example of the anchor $(x_1, y_1, t_1, x_2, y_2, t_2)$

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). So the structure of TPN is:

- 1 3D Convolutional Layer
- 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 4.2

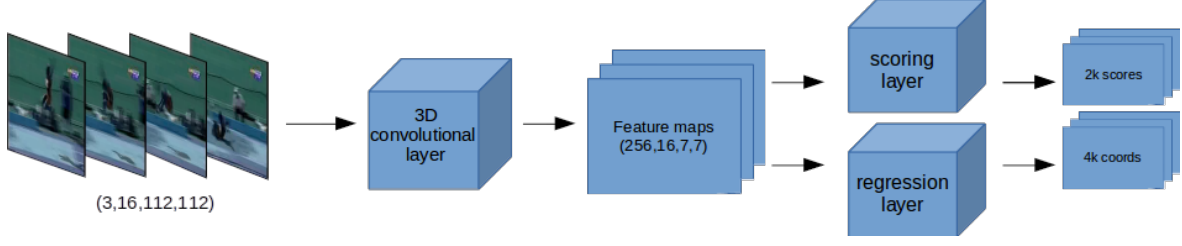


Figure 4.2: Structure of TPN

4.1.2 Training

As mentioned before, TPN extracts TOIs as 6-dim vectors. For that reason, we modify our 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 4.3, these tubes are 3D boxes which include all the groundtruth rois, which are different for each frame.



Figure 4.3: 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 (which is explained in next paragraph) and we use Cross Entropy Loss as a loss function. For regression, we use smooth-L1 loss. For regression targets, we use the same implementation as Faster R-CNN does, but for 3 domains again. So we have:

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

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

Modified Intersection over Union(mIoU) 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. We need a criterion for evaluating them if we know the groundtruth tubes. We can see an extend of Intersection over Union criterion, which is used in Object Detection algorithms. So, we will consider as foreground those which have Intersection Over Union ≥ 0.5 .

One first approach would be to consider instead of areas, the volume of 2 candidate tubes. So IoU would be:

$$IoU = \frac{\text{Volume of Overlap}}{\text{Volume of Union}}$$

Intuitively, the above criterion is good for evaluating 2 tubes if they overlap but it has one big drawback: it considers x-y dimenions to have same importance with time dimension, which we do not desire. That's because 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.

4.1.3 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[1]. 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, we get as output a feature map with size (64, 16, 7, 7).

On top of that, for each proposed TOI we give its feature map as input to a regressor. This regressor, returns $16 \cdot 4$ predicted translations, 4 for each frame. We keep only the predicted translations, for the frames that are $\geq t_1$ and $< t_2$.

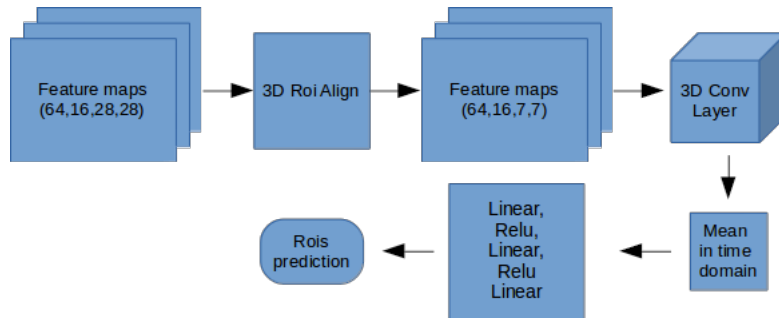


Figure 4.4: Structure of Regressor

Finally, for the frames, contained by the anchors, we set a 2D box (x_1, y_1, x_2, y_2) where x_1, y_1, x_2, y_2 are the regressed values from the anchor, and the frames which are not contained, we set a zero-ed

2D box. The previous regressor is also trainable. It is consisted of 1 2D convolutional layer followed by a Relu function and another Linear Layer as shown in figure 4.4. 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.

4.1.4 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 the groundtruth tubes. If there is, we consider this tube as True Positive (TP) else as False Negative (FN).

After we run one epochfor the whole validation dataset we calculate the **recall** metric which is:

$$recall = \frac{TP}{TP + FN}$$

In order to count overlaps we use 2 kinds of Iou metrics.

1. 3D IoU which was mentioned before, which counts tubes' Volumes' Intersection of Union (without the modification we made during training).
2. the mean ROIs' IoU for each frame.

Respectively, we calculate 2 kinds of recall. The first one tells as how good were our proposed TOIs. The second one tells us how many tubes, we managed to detect during our proposals. **We count another one recall, which tells us how many from the good Proposed TOIs managed to correspond to an actual tube. In other words, this metric shows us the performance of the regressor.**

In order to get good proposal, after TPN we use Non-Maximum Suppresion (NMS) algorith. This algorithms removes at the proposed TOIs which have overlap > 0.7 with the high-scoring proposed TOIs. As we can see in table , we get better recall scores when using NMS algorithm.

4.1.5 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 4.5. These two layers have anchors with fixed time duration. Their purpose is to be trained only in x-y domain, keeping time duration steady.

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. The regressor does not change at all.

4.1.6 Changing Regressor

As the above results show, when we translate a TOI into a sequence of ROIs, recall reduces about 20-30%. As a result, we should find a solution, in order to deal with this proble.

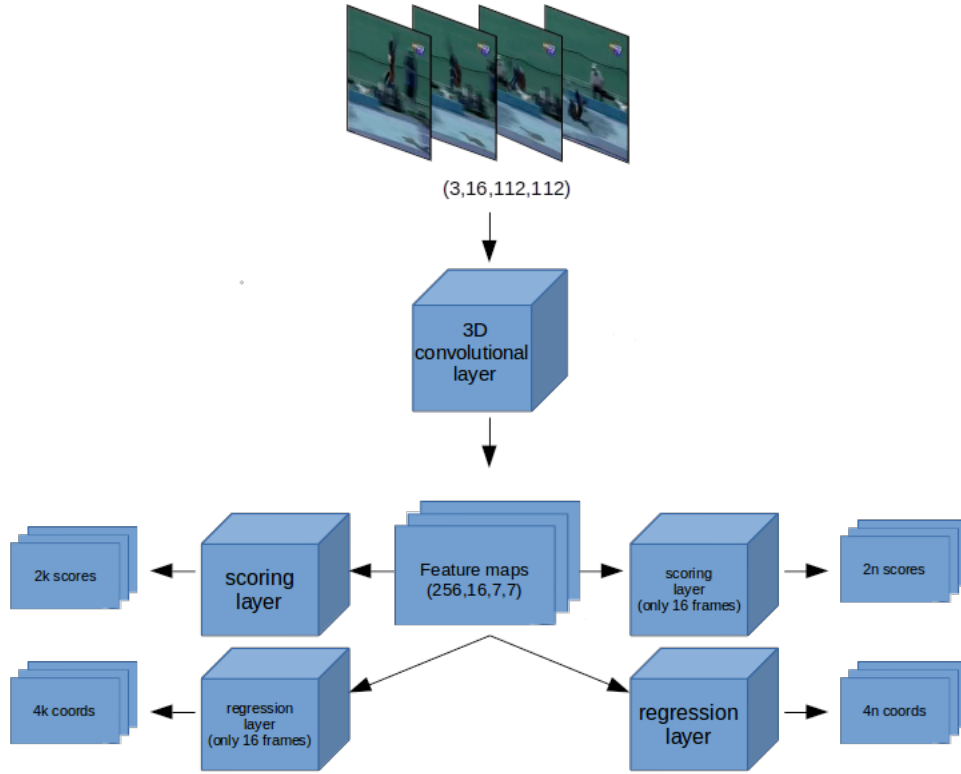


Figure 4.5: TPN structure after adding 2 new layers, where $k = 5n$.

From 3D to 2d The first idea we thought, was to change the first Convolutional layer from 3D to 2D. This means that we consider features not to have temporal dependencies for each fra. As we can see in the figure ??, we got worse results, so, we rejected this idea.

Remove time-mean layer The second idea was to remove the mean layer in time dimension. This means that, the input of the first linear layer after the 3D convolutional layer gets as input all the features outputted from 3D convolution.

Use max pooling instead of mean layer As we noticed from the above figures, our system has difficulty in translating 3D boxes into 2D sequence of ROIs. So, that makes us rethink the way we designed our TPN.

4.1.7 Changing training procedure

Until now, we trained TPN and regressor together, using one total loss, which was the sum of all the sublosses. Now we use a new approach. At first, we train TPN for 40 epochs. Then, we freeze TPN and we train the regressor for 20 epochs.

4.2 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 $(x_1, y_1, x'_1, y'_1, x_2, y_2, \dots)$ 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 4.7 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

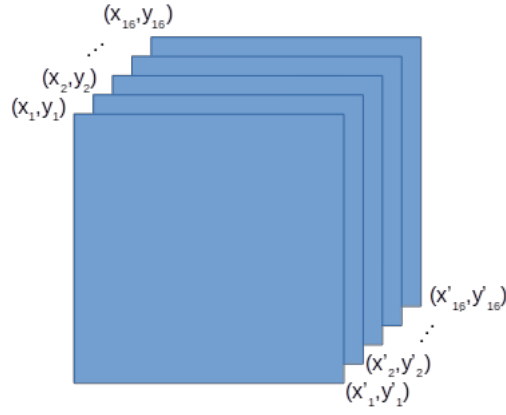


Figure 4.6: An example of the anchor $(x_1, y_1, x'_1, y'_1, x_2, y_2, \dots)$

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 coordinates 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.

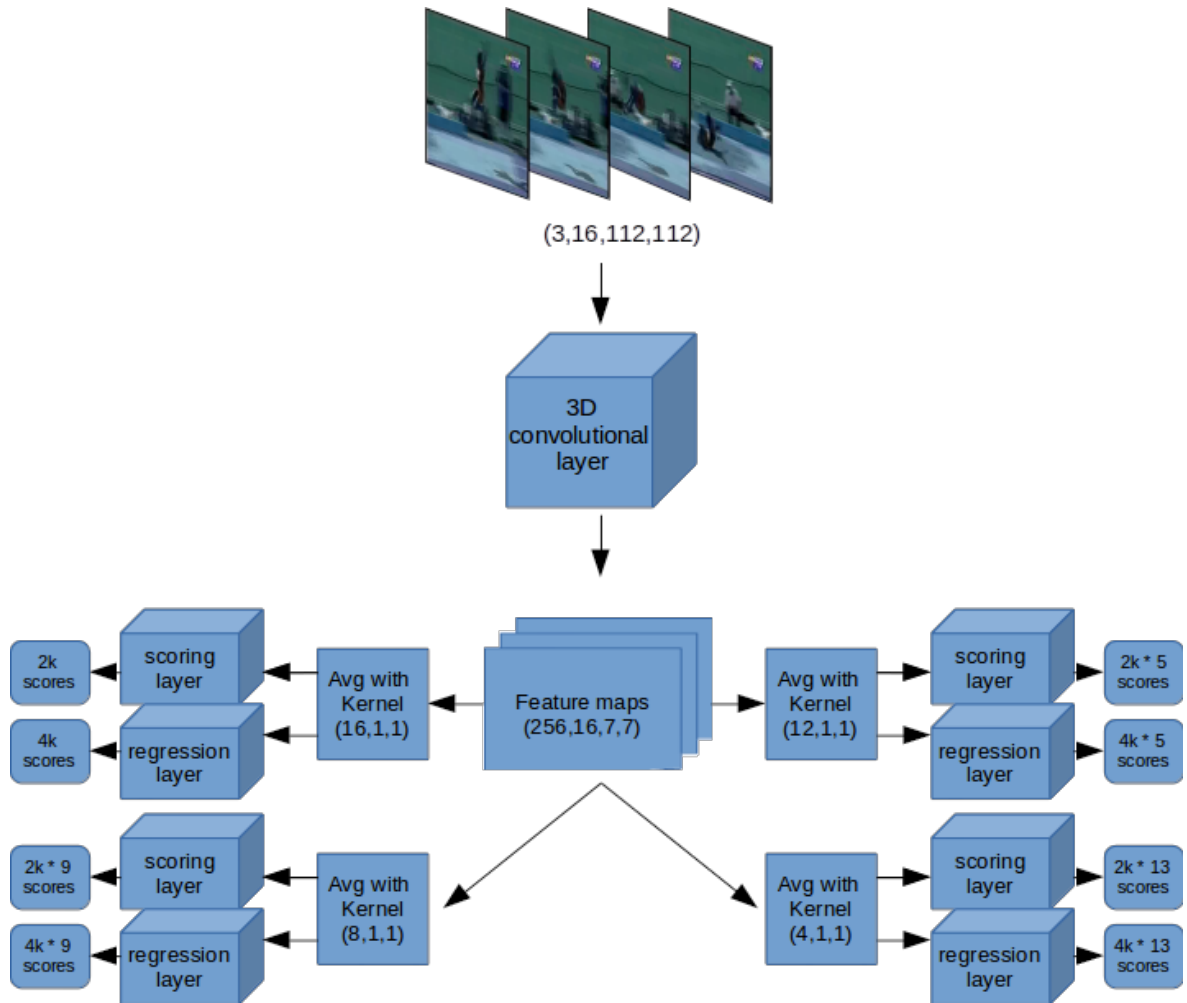


Figure 4.7: The structure of TPN according to new approach

Chapter 5

Connecting Tubes

5.1 Description

As mentioned before, TPN gets as input a sequence of 16 frames and proposes TOIs. However, most actions in videos lasts more that 16 frames. This means that, in overlapping video clips, there will be consecutive TOIs that represent the entire action. So, it is essential to create an algorithm for finding and connecting these TOIs. Our algorithm is inspired by [], and uses a score in order to decide if a sequence of TOIs is possible to contain an action. This score is a combination of 2 metrics:

Actionness, which is the TOI's possibility to contain an action. This score is produced by TPN's scoring layers.

TOIs' overlapping, which is the IoU of the last frames of the first TOI and the first frames of the second TOI.

The above scoring policy can be described by the following formula:

$$S = \frac{1}{m} \sum_{i=1}^m Actionness_i + \frac{1}{m-1} \sum_{j=1}^{m-1} Overlap_{j,j+1}$$

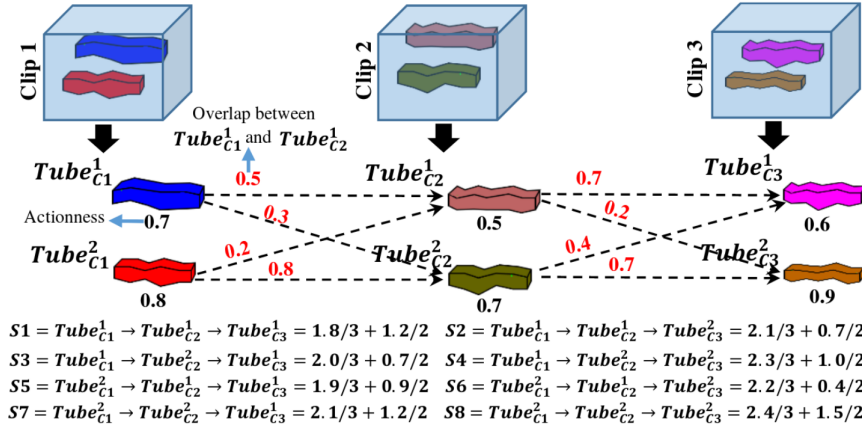


Figure 5.1: An example of calculating connection score for 3 random TOIs

For every possible combination of TOIs we calculate their score as show in figure 5.1. The above approach, however, needs too much memory for all needed calculations, so a memory usage problem is appeared. The reason is, for every new video segments we propose k TOIs (16 during training and 150 during validation). As a result, for a small video seperated in **5 segments**, we need to calculate **16^5 combinations at least**.

In order to deal with this problem, we create a greedy algorithm in order to find the candidates tubes. Intuitively, this algorithm after a new video segment keeps tubes with score higher than a threshold, and deletes the rest. So, we don't need to calculate combinations with very low score. This algorithm is described below:

1. Firstly, initialize empty lists for the final tubes, their scores, active tubes, their overlapping sum and actionness sum where:
 - Final tubes list contains all tubes which are the most possible to contain an action, and their score list contains their corresponding scores.
 - Active tubes list contains all tubes that will be match with the new TOIs. Their overlapping sum list and actionness sum list contain their sums in order to avoid calculating then for each loop.

Also, we initialize threshold equal to 0.5 .

2. For the first video segment, we add all the TOIs to both active tubes and final tubes. Their scores are only their actionness because there are no tubes for calculating their overlapping score. So, we set their overlapping sum equal to 0.
3. For each next video, firstly we calculate their overlapping score with each active tube. Then, we empty active tubes, overlapping sum and actionness score lists. For each new tube that has score higher than the threshold we add to final tubes and to active tubes.
4. If the number of active tubes is higher than a threshold (1000 in our situation), we set the threshold equal to the score of the 100th higher score. On top of that, we update the final tubes list, removing all tubes that have score lower than the threshold.
5. After that, we add in active tubes, the current video segment's proposed TOIs. Also their actionness scores in actionness sum list and zero values in corresponding positions in overlaps sum list (such as in the 1st step).
6. We repeat the previous 3 steps until there is no video segment left.

5.2 Some results

For both TPN approaches we kept the same connecting algorithm. We experiment mainly in the value of the updating threshold.

5.2.1 JHMDB dataset

As we know, this dataset uses trimmed videos. So alongside with the previous algorithm, we experiment using the same approach without step 5.

5.2.2 UCF dataset

Chapter 6

Classification stage

6.1 Description

After getting all proposed tubes, it's time to do classification. As classifiers we use 2 approaches

- A Linear Classifier
- A RNN Classifier

6.1.1 Linear Classifier

6.1.2 RNN Classifier

Chapter 7

Conclusion

Bibliography

- [Girs13] Ross B. Girshick, Jeff Donahue, Trevor Darrell and Jitendra Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation”, *CoRR*, vol. abs/1311.2524, 2013.
- [Girs15] Ross Girshick, “Fast R-CNN”, in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV ’15, pp. 1440–1448, Washington, DC, USA, 2015, IEEE Computer Society.
- [Ren15] Shaoqing Ren, Kaiming He, Ross Girshick and Jian Sun, “Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks”, in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS’15, pp. 91–99, Cambridge, MA, USA, 2015, MIT Press.