National Technical University of Athens
School of Electrical and Computer Engineering

Action Localization and Recognition in Videos

DIPLOMA THESIS

EFSTATIOS GALANAKIS

Supervisor :

Athens, December 2019

National Technical University of Athens
School of Electrical and Computer Engineering

Action Localization and Recognition in Videos

DIPLOMA THESIS

EFSTATIOS GALANAKIS

Supervisor :

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

....

Athens, December 2019

Efstatios Galanakis
Electrical and Computer Engineer

Abstract

The purpose of this diploma thesis is the design of a simple network for recognizing and localition human actions in videos. This network produces a sequence of 2D boxes for each video and a classification label for each action.

The need for ...

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

List of Tables

List of Figures

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 consided 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.

1.1.1 Human Action Recognition

Considering human action recognition, a video may be consisted of only by 1 person doing something, however, this is a ideal situation. In most cases, videos contain multiple people, who perform multiple actions or may not act at all in some segments. So, our goal is not only to classify an action, but to dertemine the temporal boundaries of each action.

1.1.2 Human Action Localization

Alongside with Human Action Recognition, another problem is to present spatial boundaries of each action. Usually, this means presenting a 2D bounding box for each video frame, which contains the actor. Of course, this bounding box moves alongside with the actor.

1.2 Applications

The field of Human Action Recognition and Localization has a lot of applications which include content based video analysis,automated video segmentation, security and surveillance systems, human-computer interaction.

The huge availability of data (especially of videos) create the necessity to find ways to take advantage of them. About 2.5 billion images are uploaded in Facebook every month, more than 34K hours of video in YouTube and about 5K images every minutes. On top of that, there are about 30 million surveillance cameras in US, which means about 700K video hours per day. All those data need to be seperated in categories according to their content in order to search them more easily. This process takes places by hand, by the user who attaches keywords or tags, however, most users avoid to do so. This situation creates the need to create algorithms for automated indexing based on the content of the video.

Another application is video summury. This area take place usually in movies or sports events. In movies, video analysis algorithms can create a small video containing all the important moments

of the movie. This can be achieve by choosing video segments which an important action take place such as killing the villain of the movie. In sports events, video summury applications include creating highlight video automatically.

On top of that, human action recognition can replace human operators in surveillance systems. Until now, security systems include a system of multiple cameras handled by a human operator, who judges if a person is acting normally or not. Automatic action classification systems can act like human, and immediately judge if there is any human behavioral anomaly.

Last but not least, another field of application is related with human-computer interaction. Robotic applications help elderly people deal with their daily needs. Also, gaming applications using Kinect create new kinds of gaming experience without the need of a physical game controller.

1.3 Challenges and Datasets

The wide variety of applications creates a lot of challenges which involve with action recognition systems. The most important include large variations in appearence of the actors, camera view-point changes, occlusions non-rigid camera motions etc. On top of that, a big problem is that there are too many action classes which means that manual collection of training sample is prohibitive. Also, some times, action vocabulary is not well defined. As figure 1.1 shows, “Open” action can include a lot of kinds of actions, so we must carefully choose which granularity of the action we will consider.

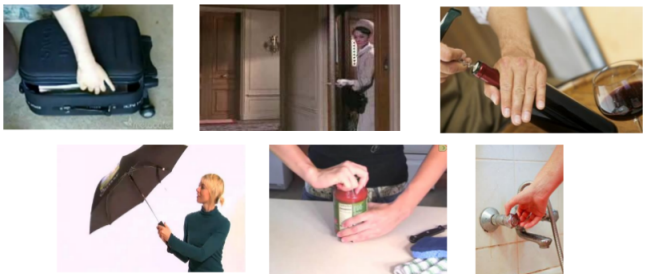


Figure 1.1: Examples of “Open” action

In order to deal with those challenges, several standard action datasets have been created in order to delevop robust human action recognition systems and detection algorithms. The first datasets like KTH include 1 actor performing using a static camera over homogeneous backgrounds. Even though, those datasets help us design the first action recognition algorithms, they were not able to deal with the above challenges. This lead us to design datasets containing more ambigious videos such as Joint-annotated Human Motion Database(JHMDB) and UCF-101.

1.3.1 JHMDB Dataset

The JHMDB dataset (Jhuang et al. 2013) is a fully annotated dataset for human actions and human poses. It is consisted of 21 action categories and 928 clips extracted from Human Motion Database (HMDB51) Kuehne et al. 2011. This dataset contains trimmed videos with duration between 15 to 40 frames. Each clip is annotated for each frame using a 2D pose and contains only 1 action. In order to train our model for action localization, we modify 2D poses into 2D boxes containing the whole pose in each frame. There are available 3 different splits for training data, proposed by the authors. We chose the first split which contains 660 videos for training set and 268 for validation .

1.3.2 UCF-101 Dataset

The UCF-101 dataset(Soomro, Zamir, and Shah 2012 contains 13320 videos from 101 action categories. From those, for 24 classes and 3194 video spatio-temporal annotations are included. This means for each frames in which an action is taking place, there is a 2D bounding box surrounding the

actor. We seperate them in 2284 videos for training set and 910 for validation test according to the first proposed training split. For training data, there are videos up to 641 frame while in validation data max number of frames is 900. Each video,both training and validation, is, of course, untrimmed, including more than 1 simultaneous actions. We took annotations from Singh et al. 2017 because the proposed, by the authors, annotations contains some mistakes.

1.4 Motivation ans Contibutions

The current achievements in Object Recognition Networks and in 3D Convolution Networks for Action Recognition have triggered us to try to combine them in order to achieve state-of-the-art results for action localization. We introduce a new network structure inspired by Hou, Chen, and Shah 2017, Girdhar et al. 2017,Ren et al. 2015 and for implementation by Yang et al. 2017.

Our contributions are the following 1) We create a new framework for action localization extending the code taken from faster RCNN, 2) We try to create a network for proposing sequences of bounding boxes in video clips which may containg an anction taking advantage of the spatio-temporal features which 3D Convolutions provide us, 3) We create a connection algorithm for linking these sequences of bounding boxes in order to propose a final action tube and 4) we try to find the most suitable feature maps for classifying them.

1.5 Thesis structure

The rest of Thesis is organized as follows. Chapter 2 provides an general introduction to Machine Learning techniques currently used. After that, we present the basic elements of object recognition systems and alongside with that, loss functions and evaluation metrics that we used. Chapter 3 presents an brief overview of literature on human action recognition and localization. Chapter 4 introduces the first basic element of our network, Tube Proposal Network (TPN) and shows all the approachs designed for this. Chapter 5 proposes an algorithm for linking the proposed TOIs from every video segment and proposal performance is presented. In Chapter 6, we present all the classification approaches we used for designing our architecture and some classification results. Chapter 7 is used for conclusions, summary of our contribution alongside with possible future work.

Chapter 2

Background

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 $Y = f(x)$.

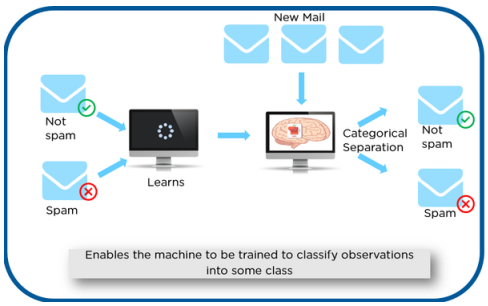


Figure 2.1: Example of supervised Learning

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

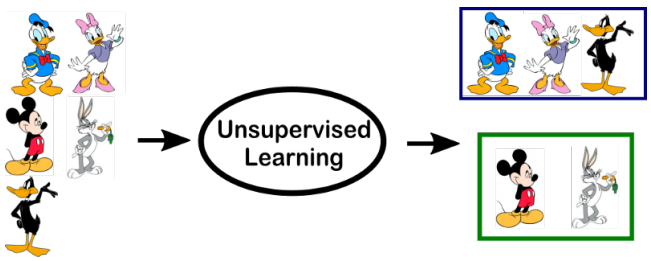


Figure 2.2: Example of unsupervised Learning

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.

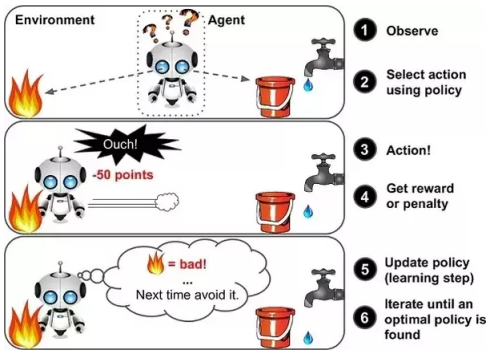


Figure 2.3: Example of Reinforcement Learning

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 1* 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.

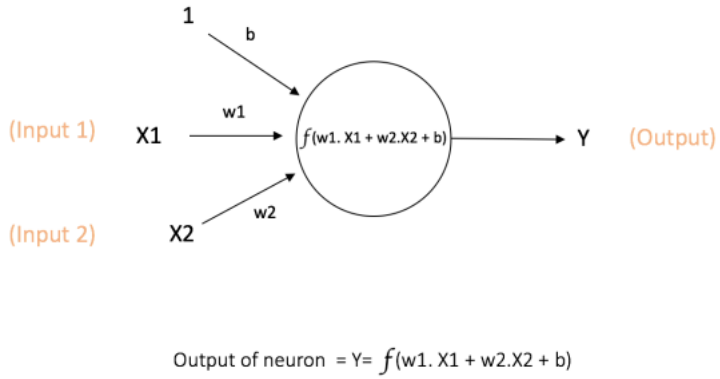


Figure 2.4: An example of a single Neuron

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 it's output is zero centered because its range in 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 mathamatical form of this function we can see that it is very simple and efficinent . 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 show each of the above activation functions.

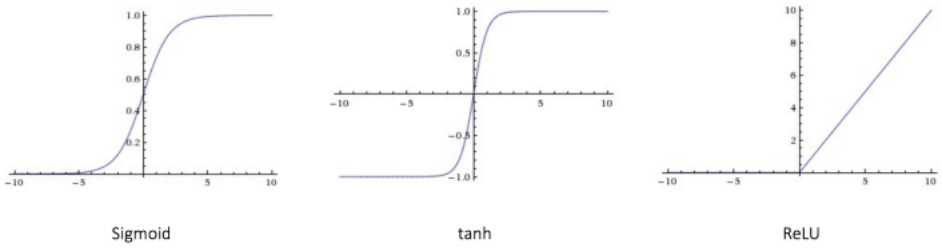


Figure 2.5: Plots of Activation functions

Feedforward Neural Network

Till now we have covered neuron and activation functions which together for 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

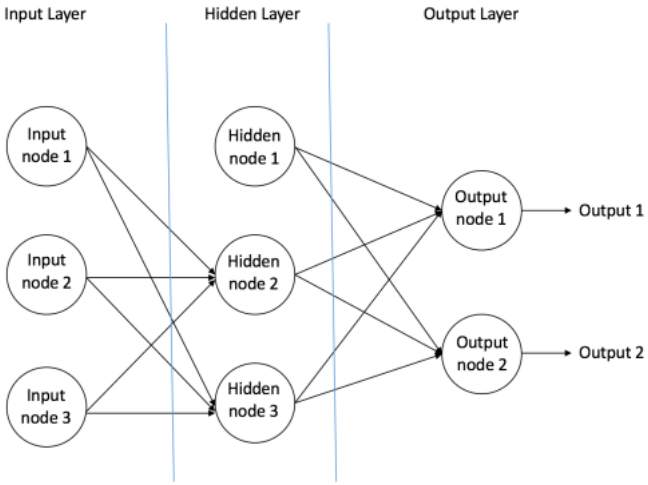


Figure 2.6: An example of a Feedforward Neural Network

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, assing 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 costist 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 cosists of several layers. For instance, an image which could be seen as an array of pixel values. The first layer often learns reprensations 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, regardeless 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.

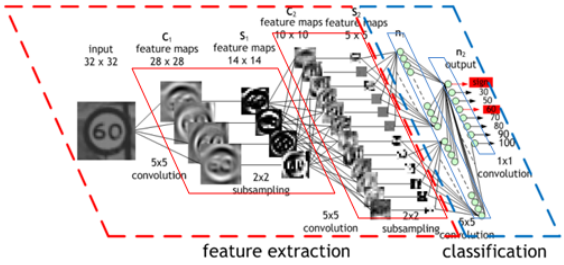


Figure 2.7: Typical structure of a ConvNet

Convolutional Layers In order to extract such features, ConvNets use 2D convolution operations. These operations take place in convolutional layers. Convolutional layers consist of a set of learnable filters. Every filter is small spatially (along widht and height), but extends through the full depth of input. During forword pass, we slide (more precisely, convolve) each filter across the width and height

of the input volume and compute dot products between the entries of the filter and the input at any position (as Figure 2.8 shows). The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.

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

Pooling Layers Pooling Layers are also refered as downsampling layers and are used to reduce the spatial dimensions, but not depth, on a convolution neural network. The intuitive reasoning behind this layer is that once we know that a specific feature is in the original input volume (there will be a high activation value), its exact location is not as important as its relative location to the other features. The main advantages of pooling layer are:

- We gain computation performance since the amount of parameters is reduce.
- Less parameters also means We deal with overfitting situations.

The pooling operation is specified, rather than learned. Two common functions used in the pooling operation are:

Average Pooling Calculate the average value for each patch on the feature map.

Maximum Pooling (or Max Pooling) Calculate the maximum value for each patch of the feature map.

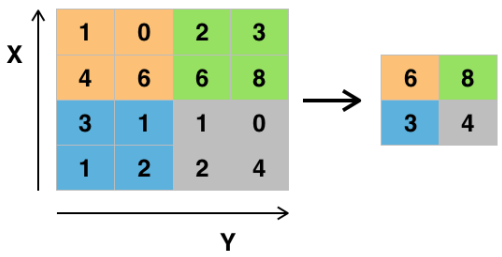


Figure 2.9: Example of Max pooling operation with a 2x2 filter and stride of 2

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 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.

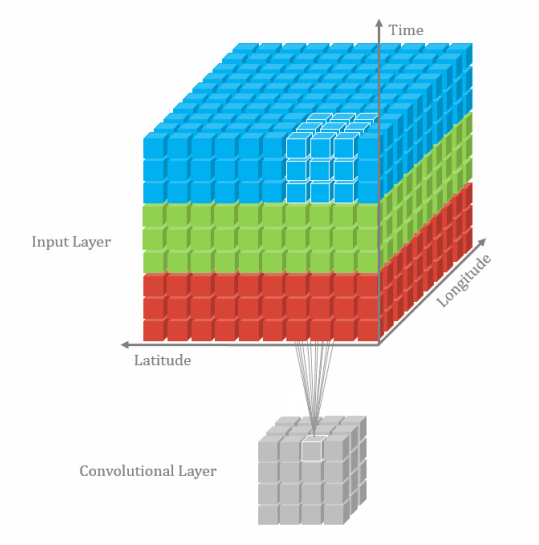


Figure 2.10: 3D Convolution operation

There are several existing approaches to tackle the video classification. This is a nonexaustive 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. Fistly, 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.

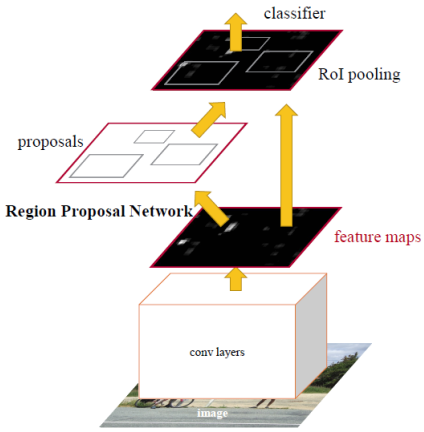


Figure 2.11: Region Proposal Network's structure

The first Object-Detection CNNs use several algorithms for proposing ROIs. For example, R-CNNGirshick et al. 2013, and Fast R-CNNGirshick 2015 used Selective Search Algorithm for extracting ROIs. One of noverlties introduced by the Faster R-CNNRen et al. 2015 is **Region Proposal Network** (RPN). Its Function is to propose ROIs and its structure can be shown in 2.11. 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 2.12 is depicted an exaple of some anchors

For each feature map's pixel corresponds **k (k=9)** anchors (3 different scales and 3 different ratios 1:1, 1:2, 2:1).

As a result, RPN gets as input feature maps extracted from the backbone CNN. Then performs 2D convolution over this input and passes the output to its scoring layer and regression layer. Those scores represent the confidence of existing an object in this specific position. On top of that, regression layer outputs 4k displacements, 4 for each anchor. Finally, we keep as output only the *n-best scoring* anchors.

2.3 Losses and Metrics

In order to train our model and check its performance, we use some known Loss functions and Metrics used in Object Detection systems.

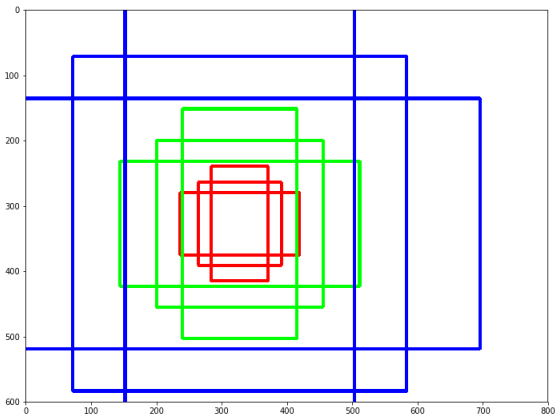


Figure 2.12: Anchors for pixel (320,320) of an image (600,800)

2.3.1 Losses

For training our network, we use **Cross Entropy Loss** for classification layers and **smooth L1-loss** for bounding box regression in each frame and their diagram is show at Figure 2.13.

Cross Entropy Loss

Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1.

Entropy is the measure of uncertainty assosiated with a given distribution *q(y)* and its formula is:

$$H = -\sum_{i=1}^{n} p_i \cdot \log p_i$$

Intuitively, entropy tells us how "surprised" we are when some event E happened. When we are sure about an event E to happened ($p_E = 1$) we have 0 entropy (we are not surprised) and vise versa.

On top of that, let's assume that we have 2 distributions, one known (our network's distribution) *p(y)* and one unknown (the actual data's distribution) *q(y)*. Cross-entropy tells us how accurate is our known distribution in predicting the unknown distribution's results. Respectively, Cross-entropy measures how accurate is our model in predicting the test data. Its formula is:

$$H_p(q) = -\sum_{c=1}^{C} q(y_c) \cdot log(p(y_c))$$

Smooth L1-loss

Smooth L1-loss can be interpreted as a combination of L1-loss and L2-loss. It behaves as L1-loss when the absolute value of the argument is high, and it behaves like L2-loss when the absolute value of the argument is close to zero. It is usually used for doing box regression on some object detection systems like Fast-RCNN(Girshick 2015), Faster-RCNN(Ren et al. 2015) and it is less sensitive to outliers according according to Girshick 2015. As showm in Girshick 2015, its formula is:

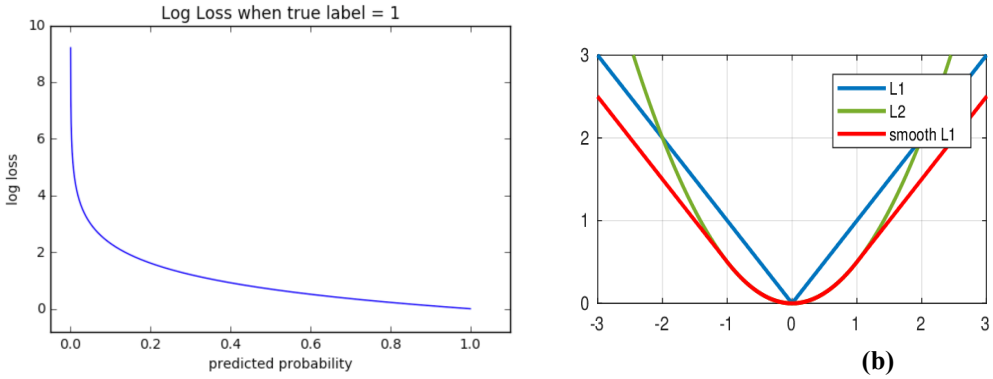
$$smooth_{L1}(x) = \begin{cases} 0.5x^2 & \text{if } x < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

It is similar to Huber loss whose formula is:

$$L_\delta(x) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \leq \delta \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise} \end{cases}$$

if we set δ parameter equal 1.

Smooth L1-loss combines the advantages of L1-loss (steady gradients for large values of x) and L2-loss (less oscillations during updates when x is small). Figure 2.13 b shows a comparison between L1-norm, L2-norm and smooth-L1 .



(a) (b)

Figure 2.13: (a) and (b) show the behavior of cross-entropy loss and smooth-L1 respectively.

2.3.2 Metrics

Evaluating our machine learning algorithm is an essential part of any project. The way we choose our metrics influences how the performance of machine learning algorithms is measured and compared. They influence how to weight the importance of different characteristics in the results and finally, the ultimate choice of which algorithm to choose. Most of the times we use classification accuracy to measure the performance of our model, however it is not enough to truly judge our model.

At first, we introduce some basic evaluation metrics in order,then, to present those we use for our assesment.

2.3.3 Intersection over Union

The first and most important metric that we use is Intersection over Union (IoU). IoU measures the overlap between 2 boundaries. It is usually used in Object Recognition Networks in order to define how good overlap a predicted bounding box with the actuall bounding box as shown in Figure 2.14. We predefine an IoU threshold (say 0.5) in classifying whether the prediction is a true positive or a false positive.

Intersection over Union is defined as:

$$IoU = \frac{\text{area of overlap}}{\text{area of union}}$$

In Figure 2.14, spatial IoU between 2 bounding boxes, (x_1, y_1, x_2, y_2) and (x_3, y_3, x_4, y_4),is presented, which means IoU metric is implemented for x-y dimensions. Area of overlap and area of union can be defined as:

$$\text{Area of overlap } = \|(min(x_2, x_4) - max(x_1, x_3), min(y_2, y_4) - max(y_1, y_3))\|$$

$$\text{Area of union } = \|(max(x_2, x_4) - min(x_1, x_3), max(y_2, y_4) - min(y_1, y_3))\|$$

On top of that, we can implement IoU for 1 dimension and for 3 dimensions.

Figure 2.14: Example of IoU scoring policy

1D IoU We can name 1D IoU as temporal overlap. Let's consider 2 temporal segments (t_1, t_2) and (t_3, t_4), between which we want to estimate their overlap score. Their IoU can be described as:

$$\text{Area of overlap } = \|(min(t_2, t_4) - max(t_1, t_3))\|$$

$$\text{Area of union } = \|(max(t_2, t_4) - min(t_1, t_3))\|$$

3D IoU 3-dimentional Intersection over Union which can, also, be named as spatiotemporal IoU, can be defined by 2 ways:

3D boxes are cuboids In this case, 3D boxes can be written as (x, y, z, x', y', z'). So, the IoU overlap between 2 boxes, $(x_1, y_1, z_1, x_2, y_2, z_2)$ and $(x_3, y_3, z_3, x_4, y_4, z_4)$, is defined as:

$$\text{Area of overlap } = \|(min(x_2, x_4) - max(x_1, x_3), min(y_2, y_4) - max(y_1, y_3), min(z_2, z_4) - max(z_1, z_3))\|$$

$$\text{Area of union } = \|(max(x_2, x_4) - min(x_1, x_3), max(y_2, y_4) - min(y_1, y_3), max(z_2, z_4) - min(z_1, z_3)$$

x-y are continuous and z discrete In this case 3D boxes is defined as a sequence of 2D boxes (x, y, x', y'). For this definition, z-dimension is discrete, and IoU can be defined with 2 ways, which both result in the same overlapping score. Let's cosider 2 sequences of boxes, with temporal limits (t_1, t_2) and (t_3, t_4). We calculate their IoU following one of the following methods:

1. IoU is the product between temporal-IoU and the average spatial-IoU between 2D boxes in the overlapping temporal area and it is described as:

$$IoU = IoU((t_1, t_2), (t_3, t_4)) \cdot \frac{1}{K_2 - K_1} \sum_{i=K_1}^{K_2} IoU(X_1^i, X_2^i)$$

   where
   - $K_1 = min(t_2, t_4)$
   - $K_2 = max(t_1, t_3)$
   - $X_1^i = (x_1^i, y_1^i, x_2^i, y_2^i)$ and $X_2^i = (x_3^i, y_3^i, x_4^i, y_4^i)$

2. IoU is the average spatial-IoU if we consider 2D boxes as $(0, 0, 0, 0)$ if $t \notin [t_{start}, t_{finish}]$ and it is written as:

$$IoU = \frac{1}{K} \sum_{i=min(t_1,t_3}^{max(t_2,t_4)} IoU(X_1^i, X_2^i)$$

   - $K = max(t_2, t_4) - min(t_1, t_3)$
   - $X_1 = (x_1, y_1, x_2, y_2)$ if $i \in [t_1, t_2]$ or $(0, 0, 0, 0)$ if $i \notin [t_1, t_2]$
   - $X_2 = (x_3, y_3, x_4, y_4)$ if $i \in [t_3, t_4]$ or $(0, 0, 0, 0)$ if $i \notin [t_3, t_4]$

From above implementations, we are involve mostly with temporal and spatiotemporal IoU.

2.3.4 Precision & Recall

In order to describe **precision** and **recall** metrics, we will use an example. Let's consider a group of people in which, some of them are sick and the others are not. We use a network which is able to predict if a person is sick or not if we give it some data as input.

Precision measures how accurate are our model's predictions. i.e. the percentage of predictions that are correct. In our case, how accurate is our model when it predicts that a person is sick.

Recall measures how good we found all the sick people. In our case, how many of the acctual sick people we managed to find.

Their definisions are:

$$Precision = \frac{TP}{TP + FP}$$

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

where

- TP = True positive, which means that we predict a person to be sick and he is actually sick.
- TN = True negative, we predict that a person isn't sick and he isn't.
- FP = False positive, we predict a person to be sick but he isn't actually.
- FN = False negative, we predict a person not to be sick but he actually is.

From these 2 metrics we use recall metric in order to evaluate our networks performance, and more specifically, its performance on finding good action tube proposals. We consider a groundtruth action as true positive when there is at least 1 proposed action tube that its IoU overlap score is bigger that a predifed threshold. If there is no such action tube, then we consider this groundtruth action tube as false negative.

2.3.5 mAP - TODO

Precision and recall are single-value metrics based on the whole list of predictions. By looking their formulas, we can see that there is a trade-off between precision and recall performance. This trade-off can be adjusted by the softmax threshold, used in model's final layer. In order to have high precision performance, we need to decrease the number of FP. But this will lead to decrease recall performance and vice-versa.

As a result, these metrics fail to determine if a model is performing well in object detection tasks as well as action detection tasks. For that reason, we use mAP metric

AP (Average precision) is a popular metric in measuring the accuracy of object detectors like Faster R-CNN, SSD, etc. Average precision computes the average precision value for recall value over 0 to 1.

In our case, we use the metrics presented in Gkioxari and Malik 2014 in order to quantify our results:

video-AP measures the area under the precision-recallcurve of the action tubes predictions. A tube is correctif the mean per frame intersection-over-union with theground truth across the frames of the video is greaterthan and the action label is correctly predicted.

2.3.6 MABO

In order to evaluate more the quality of our proposals, both during TPN and connecting tube stages, recall metric isn't enough. That's because recall metric tells us only in how many known objects there was at least 1 proposal that satified the detection criterion. However, it doen't tells us how much good. This can be better clarify if we consider figure 2.15.

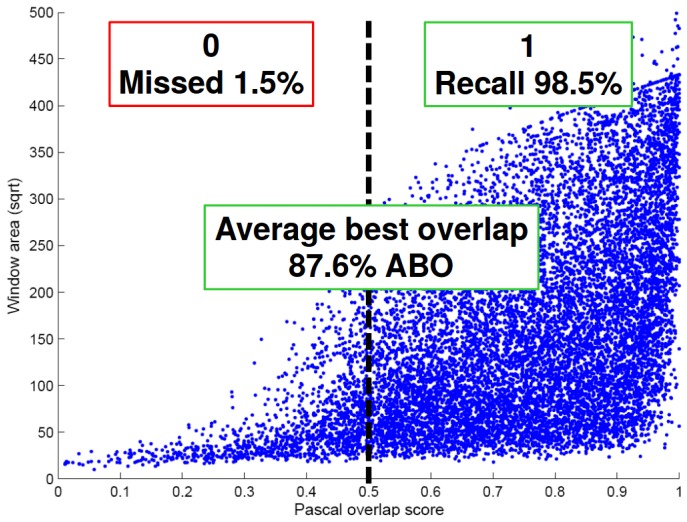


Figure 2.15: Example of Max pooling operation with a 2x2 filter and stride of 2

Chapter 3

Related work

3.1 Action Recognition

First approaches for action classification consisted of 2 steps a) compute complex handcrafted features from raw video frames such as SIFT, HOG, optical flow and b) train a classifier based on those features. These approaches made the choise of features a signifact factor for network's performance. That's because different action classes may appear dramatically different in terms of their appearences and motion patterns. Another problem was that most of those approaches take assumptions about the circumstances under which the video was taken because there was problems such as cluttered background, camera viewpoit variations etc. A review techniques used until 2011 made by Aggarwal and Ryoo 2011.

Recent results in deep architectures and especially in image classification made us attempt to train CNN networks for the task of action classification. First significant attempt made by Karpathy et al. 2014. Simonyan and Zisserman 2014 and Feichtenhofer, Pinz, and Zisserman 2016 both added optical flow in order to achieve better results. On top of that, the increase in computing performance contributed to the design more complicated architectures including 3D Convolutions as presented in Ji et al. 2013 as done by Tran et al. 2014.

R(2+1) Tran et al. 2017 **(**Pending...More before 3D ResNet) Recent day 3D ResNet has been introduced by Hara, Kataoka, and Satoh 2018a

3.2 Action Localization

As mentioned before, Action Localization can be seen as an extention of object detection problem, where the outputs are action tubes that consist of a sequence of bounding boxes. So, there are several approaches including an object-detector network for single frame action proposal and a classifier. The introduction of R-CNN (Girshick et al. 2013) achieve significant improvemet in the performance of Object Detection Networks. This architectures, firstly, proposes regions in the image which are likely to contain an object and then it classify it using a SVM classifier. Inspired by this architecture, Gkioxari and Malik 2014 desing a 2-stream RCNN network in order to generate action proposals for each frame, one stream for frame level and one for optical flow. Then they connect them using viterbi connection algorithm. Weinzaepfel, Harchaoui, and Schmid 2015 extend this approach, performing frame-level proposals and using a tracker for connecting those proposals using both spatial and optical flow features. Also it performs temporal localization using a sliding window over the tracked tubes. Peng and Schmid 2016 used Faster R-CNN (Ren et al. 2015) instead of RCNN for frame-level proposals, and they use Viterbi algorithm for linking proposals, too. For temporal localization, they use a maximum subarray method.

Jain et al. 2014 introduces the tubelets.

Singh et al. 2017 uses SSD

Some approaches include tracking Weinzaepfel, Harchaoui, and Schmid 2015. Other approaches treat a video as a sequence of frames such as in Kalogeiton et al. 2017 and in Hou, Chen, and Shah 2017.

3d-2d pose

3.3 Our implementation

We propose a network similar to Hou, Chen, and Shah 2017. Our architecture is consisted by the following basic elements:

- One 3D Convolutional Network, which is used for feature extraction. In our implementaion we use a 3D Resnet network which is taken from Hara, Kataoka, and Satoh 2018b and it is based on ResNet CNNs for Image Classification He et al. 2015.
- Tube Proposal Network for proposing action tubes (based on the idea presented in Hou, Chen, and Shah 2017).
- A classifier for classifying video tubes.

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 introduced by FasterRCNN (Ren et al. 2015), but instead of images, TPN is used in videos as show in Hou, Chen, and Shah 2017. 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.

4.1 Preparation for TPN

4.1.1 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 Hara, Kataoka, and Satoh 2018b. 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 4.1 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 4.1b and 4.1d show that).

4.1.2 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 Kay et al. 2017 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.

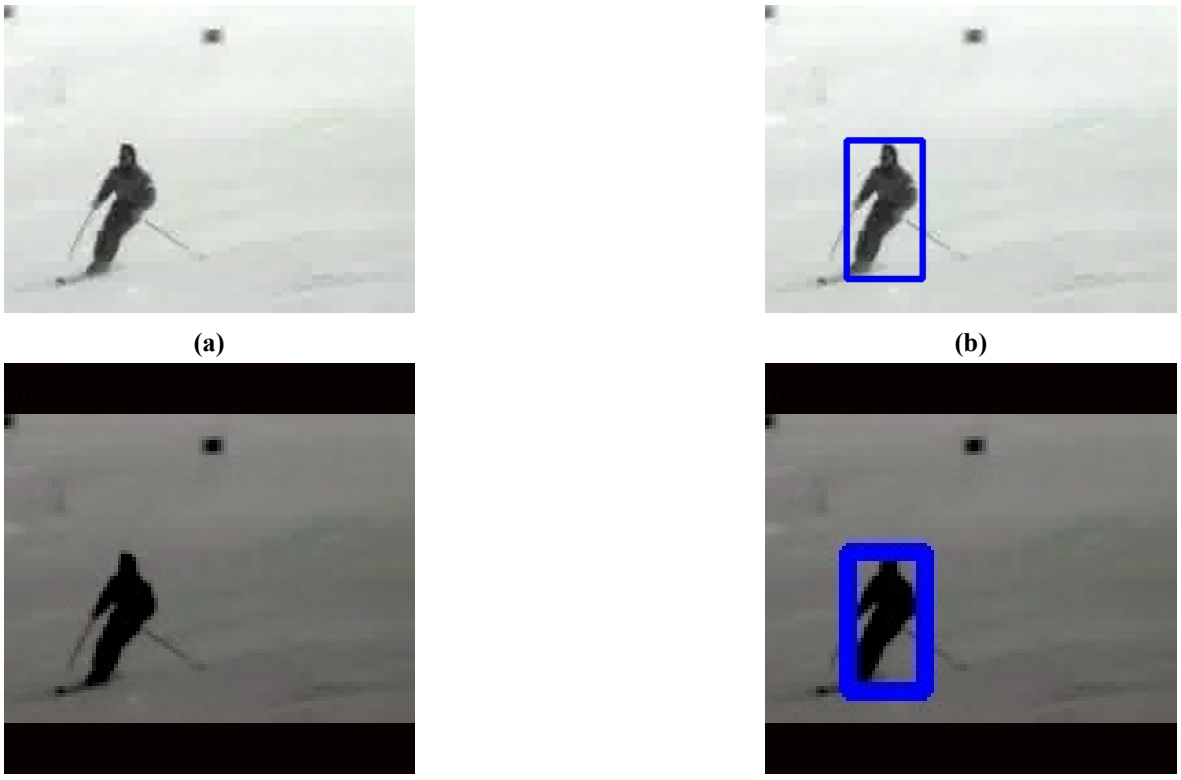


Figure 4.1: At (a), (b) frame is its original size and at (c), (d) same frame after preprocessing part

4.2 3D anchors as 6-dim vector

4.2.1 First Description

We started desinging our TPN inspired by Hou, Chen, and Shah 2017. 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 4.2.

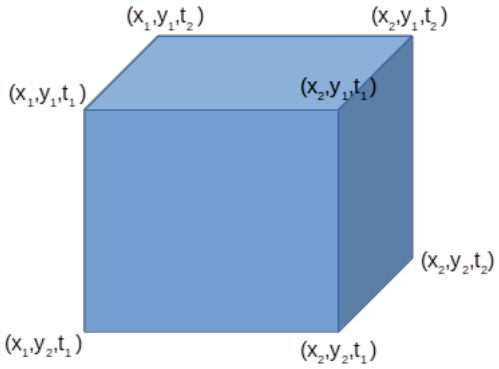


Figure 4.2: An example of the anchor $(x_1, y_1, t_1, x_2, y_2, t_2)$

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 Hou, Chen, and Shah 2017, network's anchors are defined according to the dataset most

common anchors. This, however, creates the need to redesign the network for each dataset. We use the same anchors for both datasets, trying to generalize our approach. 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 4.3

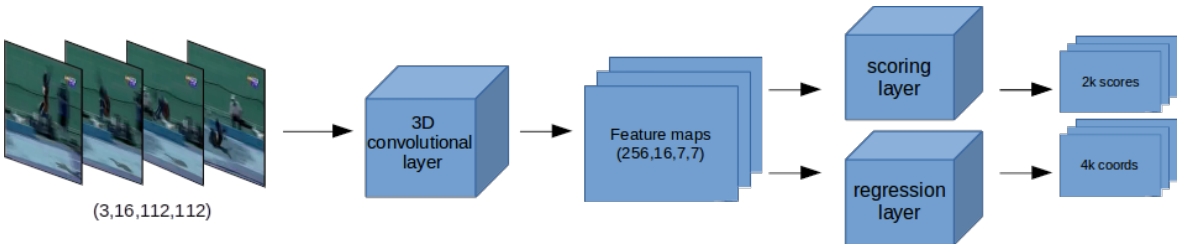


Figure 4.3: Structure of TPN

4.2.2 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 4.4, these tubes are 3D boxes which include all the groundtruth rois, which are different for each frame.

Figure 4.4: 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 (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:

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

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*).

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 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.

4.2.3 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 (He et al. 2017)o. 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 $< t_2$.

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 reggressed 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 4.5. 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.

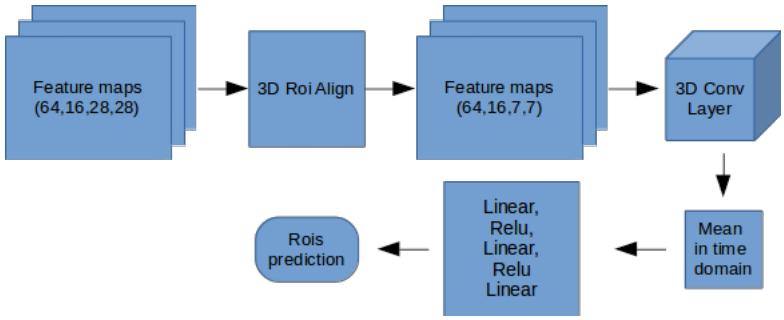


Figure 4.5: Structure of Regressor

4.2.4 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:

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

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 algoriths 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.

4.2.5 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 4.6. These two layers have anchors with fixed time duration. Their purpose is to be trained only in x-y domain, keeping time duration steady.

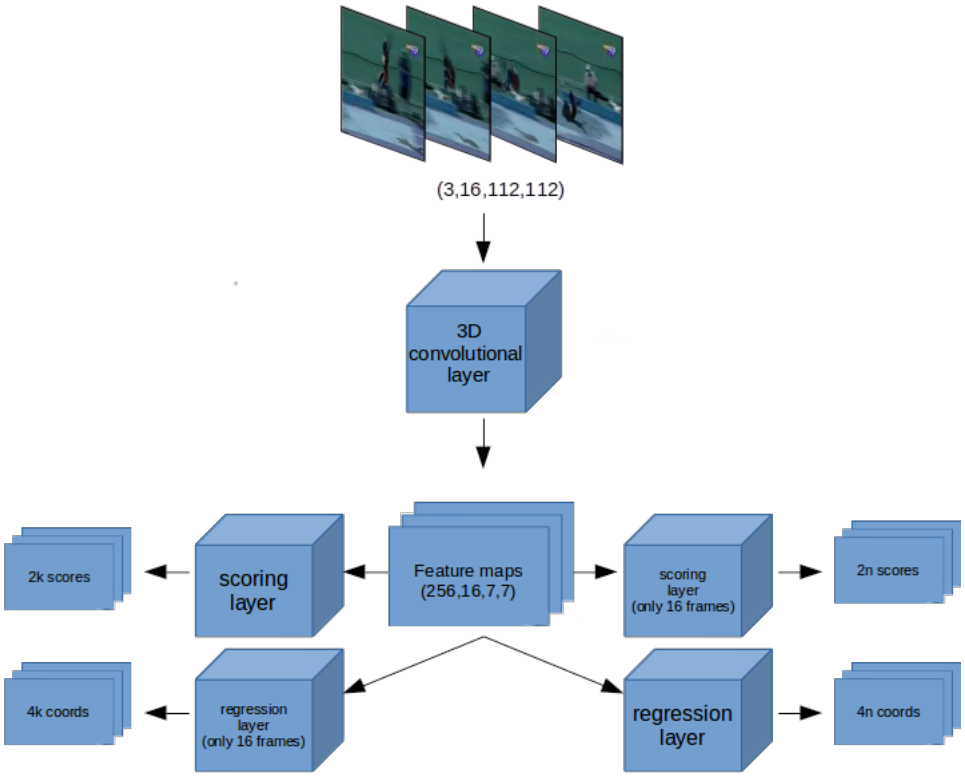


Figure 4.6: TPN structure after adding 2 new layers, where k = 5n.

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. The regressor does not change at all.

4.2.6 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.

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 frame. 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 outputed 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.

4.2.7 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.

4.3 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 $(x_1, y_1, x'_1, y'_1, x_2, y_2, ...)$ 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 Girdhar et al. 2017. In figure 4.7 we can an example of this type of anchor.

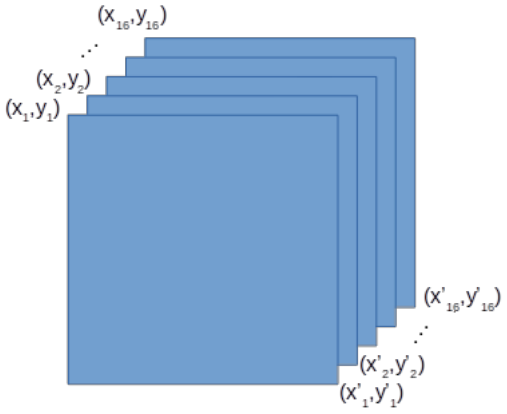


Figure 4.7: An example of the anchor $(x_1, y_1, x'_1, y'_1, x_2, y_2, ...)$

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 4.8. As we can see, we added scoring and regression layers for each duration.

4.3.1 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.

4.3.2 Adding regressor

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

4.3.3 Trying to improve performance

TODO

4.3.4 Changing training procedure

TODO

4.3.5 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

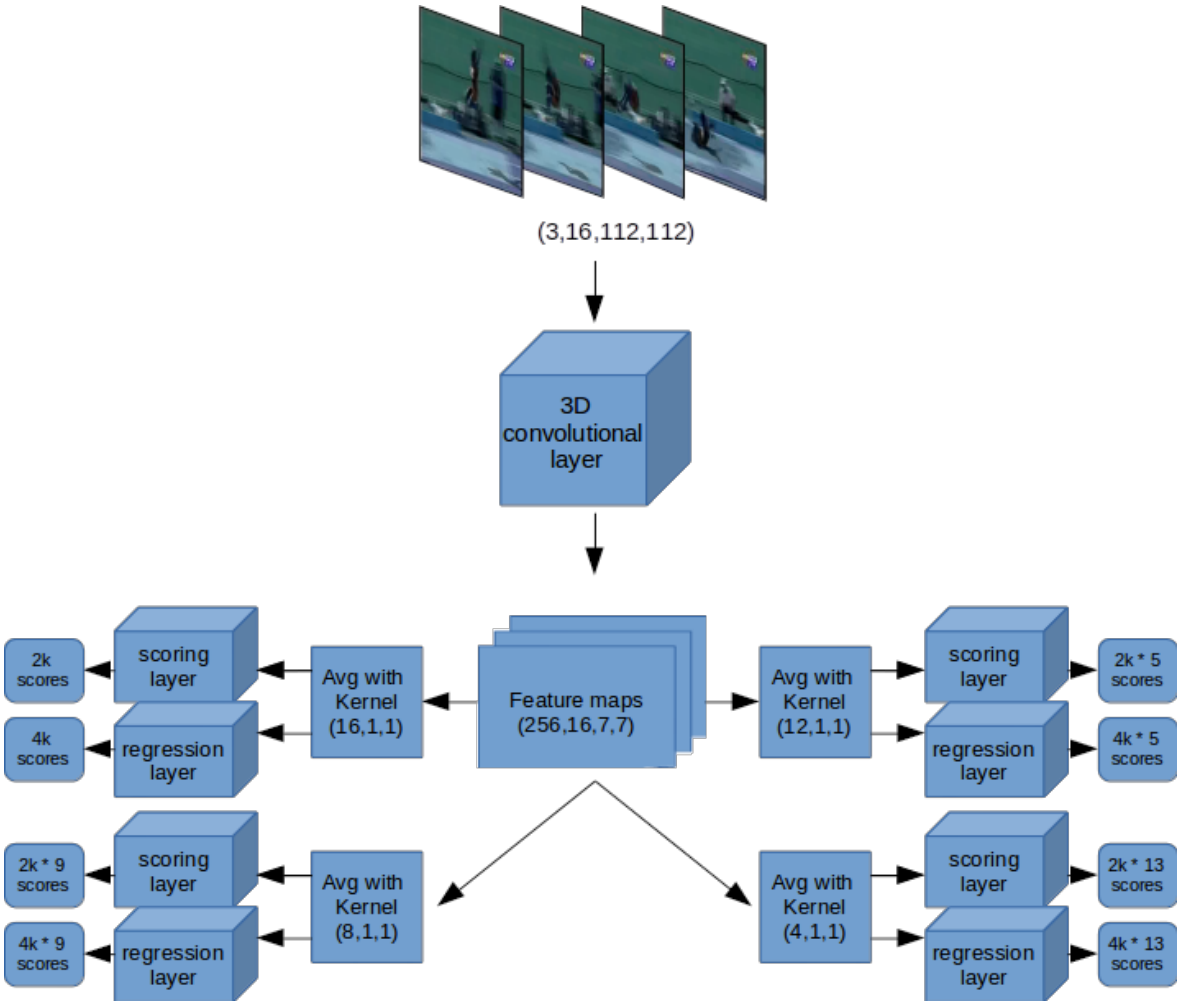


Figure 4.8: The structure of TPN according to new approach

for sample duration = 8 and 4 frames.

.....

Chapter 5

Connecting Tubes

5.1 Description

After getting TOIs for each video segment, it is time to connect them. That's because most actions in videos lasts more that 16 frames. This means that, in overlaping video clips, there will be consequentive TOIs that represent the entire action. So, it is essential to create an algorithm for finding and connecting these TOIs.

5.1.1 First approach: combine overlap and actioness

Our algorithm is inspired by Hou, Chen, and Shah 2017, which calculates all possible sequences of ToIs. In order find the best candidates, it uses a score which tells us how likely a sequence of TOIs is to contain an action. This score is a combination of 2 metrics:

Actioness, 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:

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

For every possible combination of TOIs we calculate their score as show in figure 5.1.

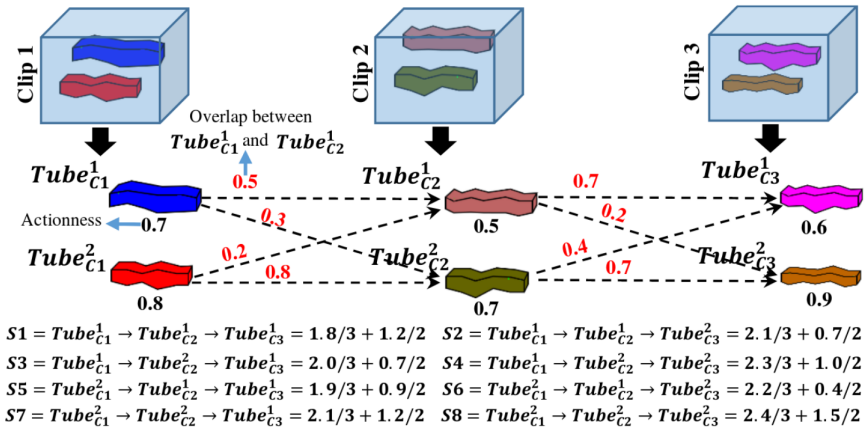


Figure 5.1: An example of calculating connection score for 3 random TOIs

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 **10 segments**, we need to calculate **150^{10} scores** during validation stage. This causes our system to overload and it takes too much time to process just one video.

In order to deal with this problem, we create a greedy algorithm in order to find the candidates tubes. Inituitively, 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. We wrote code for calculating tubes' scores in CUDA language, which has the ability to parallel process the same code using different data. Our algorithm is described below:

1. Firstly, initialize empty lists for the final tubes, final tubes' duratio, their scores, active tubes, their correnspondig duration, active tubes' overlapping sum and actioness sum where:

   - Final tubes list contains all tubes which are the most likely to contain an action, and their score list contains their corresponding scores. We refer to each tube by its index which is related a tensor, in which we saved all the ToIs proposed from TPN for each video segment.
   - Active tubes list contains all tubes that will be match with the new TOIs. Their overlapping sum list and actioness 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 actioness because there are no tubes for calculating their overlapping score. So, we set their overlaping sum equal to 0.

3. For each next video, after getting the proposed ToIs, firstly we calculate their overlapping score with each active tube. Then, we empty active tubes, active tubes' duration, overlapping sum and actioness score lists. For each new tube that has score higher than the threshold we add both to final tubes and to active tubes, and we increase their duration.

4. If the number of active tubes is higher than a threshold, 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 actioness scores in actioness sum list and zero values in correnspondig 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.

7. Finally, as we mentioned before, we have a list which contains the indexes of the saved tubes. So, we modify them in order to have the final bounding boxes. However, 2 succeeding ToIs do not have extactly the same bounding boxes in the frames that overlap. For example, ToIs from the 1^{st} video segment start from frame 1 to frame 16. If we have video step equal with 8, it overlaps with the ToIs from the succeeding video segment in frames 8-16. In those frames, in final tube, we choose the area that contains both bounding boxes which is denoted as $(min(x_1, x'_1), min(y_1, y'_1), max(x_2, x'_2), max(y_2, y'_2))$ for bounding boxes (x_1, y_1, x_2, y_2) and (x'_1, y'_1, x'_2, y'_2).

5.2 Some results

In order to validate our algorithm, we firstly experiment in JHMDB dataset's videos in order to define the best overlapping policy and the video overlapping step. Again, we use recall as evaluation

metrinc. A groundtruth action tube is considered to be found, as well as positive, if there is at least 1 video tube which overlaps with it over a predefined threshold, otherwise it . These thresholds are again 0.5, 0.4 and 0.3. We set TPN to return 30 ToIs per video segment. We chose to update threshold when active tubes are more than 500 and to keep the first 100 tubes as active. We did so, because, a big part of the code is performing in CPU. That's because, we use lists, which are very easy to handle for adding and removing elements. So, if we use bigger update limits, it takes much more time to process them.

sample duration = 16 At first we use as sample duration = 16 and video step = 8. As overlapping frames we count frames *(8...15)* so we have #8 frames. Also, we use only #4 frames with combinations *(8...11), (10...13) and (12...15)* and #2 frames with combinations *(8,9), (10,11), (12,13), and (14,15)*. The results are shown in Table 5.1 (in bold are the frames with which we calculate the overlap score).

combination	overlap thresh		
	0.3	0.4	0.5
0,1,...,**{8,...,15}** **{8,9,...,15}**,16,...,23	0.3172	0.4142	0.6418
0,1,...,**{8,...,11,}**...,14,15 **{8,...,11}**,12,...,22,23	0.3172	0.4142	0.6381
0,1,...,**{10,...,13,}**14,15, 8,9,**{10,...,13}**,14,...,22,23	0.3209	0.4179	0.6418
0,1,...,**{12,...,15,}** 8,9,...,**{12,...,15}**,16,...,23,	0.3284	0.4216	0.6381
0,1,...,**{8,...,11,}**,...,14,15, **{8,9,...,11,}**12,...,22,23	0.3172	0.4142	0.6381
0,1,...,**{10,...,13,}**14,15, **{10,...,13}**,14,...,22,23	0.3209	0.4179	0.6418
0,1,...,**{12,...,15}** 8,9,...,**{12,...,15}**,16,...	0.3284	0.4216	0.6381
0,1,...,**{8,9,}**,10,...,14,15, **{8,9,}**10,11,...,22,23	0.3134	0.4104	0.6381
0,1,...,**{10,11,}**,12,...,14,15, 8,9,**{10,11,}**12,...,22,23	0.3209	0.4216	0.6418
0,1,...,**{12,13,}**,14,15, 8,9,...,**{12,13,}**14,...,22,23	0.3246	0.4179	0.6418
0,1,...,13,**{14,15,}** 8,9,...,**{14,15,}**16,...,22,23	0.3321	0.4216	0.6306

Table 5.1: Recall results for step = 8

As we can from the above table, generally we get very bad performance and we got the best performance when we calculate the overlap between only 2 frames (either *14,15* or *12,13*). So, we thought that we should increase the video step because, probably, the connection algorithm is too strict into big movement variations during the video. As a results, we set video step = 12 which means that we have only 4 frames overlap. In this case, for #4 frames, we only have the combination *(12...15)*, for #2 frames we have *(12,13), (13,14) and (14,15)* as shown in Table 5.2.

combination	overlap thresh 0.3	0.4	0.5
0,1,...,11,**{12,...,15}** **{12,13,...,15}**,16,...,26,27	0.3769	0.4627	0.6828
0,1,...,**{12,13,}**,14,15, **{12,13,}**14,15,...,26,27	0.3694	0.4627	0.6903
0,1,...,12**{13,14,}**,15, 12,**{13,14,}**15,...,26,27	0.3843	0.4627	0.6828
0,1,...,12,13**{14,15,}** 12,13,**{14,15,}**16,...,26,27	0.3694	0.459	0.6828

Table 5.2: Recall results for step = 12

As we can see, recall performance is increase so that means that our assumption was correct. So again, we increase video step into 14, 15 and 16 frames and recall score is shown at Table **??**

combination	overlap thresh 0.3	0.4	0.5
0,1,...,13**{14,15}** **{14,15}**,16,...,28,29	0.3731	0.5336	0.6493
0,1,...,13,**{14,}**15, **{14,}**15,...,28,29	0.3694	0.5299	0.6455
0,1,...,14,**{15}** 14,**{15,}**16,...,28,29	0.3731	0.5187	0.6381
0,1,...,14,**{15}** **{15}**,16,...,30	0.3918	0.5187	0.6381
0,1,...,14,**{15}** **{16}**,17,...,31	0.4067	0.7313	0.8731

Table 5.3: Recall results for steps = 14, 15 and 16

The results show that we get the best recall performance when we have no overlapping steps and video step = 16 = sample duration. We try to improve more our results, using smaller duration because, as we saw from TPN recall performance, we get better results when we have sample duration = 8 or 4.

sample duration = 8 We wanted to confirm that we get the best results, when we have no overlapping frames and step = sample duration. So Table 5.4 shows recall performance for sample duration = 8 and video step = 4 and Table 5.5 for video steps = 6, 7 and 8.

combination	overlap thresh 0.3	0.4	0.5
0,1,2,3,13**{4,5,6,7}** **{4,5,6,7}**,8,9,10,11	0.2015	0.3582	0.5858

0,1,2,3,**{4,5,}**6,7 **{4,5,}**6,7,8,9,10,11	0.1978	0.3582	0.5933
0,1,2,3,4**{5,6,}**7 4,**{5,6,}**7,8,9,10,11	0.1978	0.3507	0.5821
0,1,2,3,4,5**{6,7}** 4,5,**{6,7,}**8,9,10,11	0.194	0.3433	0.585

Table 5.4: Recall results for step = 4

combination	overlap thresh 0.3	0.4	0.5
0,1,2,3,4,5**{6,7}** **{6,7}**,8,9,10,11,12,13	TODO	TODO	TODO
0,1,2,3,4,5,**{6,}**7 **{6,}**7,8,9,10,11,12,13	TODO	TODO	TODO
0,1,2,3,4,5,6,**{7}** 6,**{7}**8,9,10,11,12,13	TODO	TODO	TODO
0,1,2,3,4,5,6**{7}** **{7,}**8,9,10,11,12,13,14	TODO	TODO	TODO
0,1,2,3,4,5,6**{7}** **{8}**9,10,11,12,13,14,15	TODO	TODO	TODO

Table 5.5: Recall results for steps = 6, 7 and 8

5.2.1 UCF dataset

In previous steps, we tried to find the best overlap policy for our algorithm in JHMDB dataset. After that, it's time to apply our algorithm in UCF dataset using the best scoring overlap policy. We did some modifications in the code, in order to save memory and move most parts of the code to GPU. This happened by using tensors instead of lists for scores and most operations are, from now on, matrix operations. On top that, last step of the algorith, which is the modification from indices to actual action tubes was written in CUDA code so it takes place in GPU, too. So, we are now able to increase the number of tubes returned by TPN, the max number of active tubes before updating threshold and the max number of final tubes. In Table **??**

combination	TPN tubes	Final tubes	overlap thresh 0.5	0.4	0.3	MABO
0,1,...,6,**{7,}** **{8,}**9,...,14,15	30	500	0.2829	0.4395	0.5817	0.3501
		2000	0.3567	0.4996	0.6289	0.3815
		400	0.3749	0.5316	0.6487	0.3934
	100	500	0.2966	0.451	0.5947	0.356
		2000	0.3757	0.5163	0.6471	0.3902
		4000	0.3977	0.5506	0.6624	0.4029
0,1,...,14,**{15,}** **{16,}**17,18,...,23	30	500	0.362	0.5042	0.6243	0.3866
		2000	0.416	0.5468	0.6631	0.4108
		4000	0.4281	0.5589	0.6779	0.4182

	150	500	0.3589	0.4981	0.6198	0.3845
		2000	0.4129	0.5392	0.6563	0.4085
		4000	0.4266	0.5521	0.6722	0.4162

Table 5.6: Recall results for UCF dataset

combination	TPN tubes	Final tubes	overlap thresh 0.9	0.8	0.7	MABO
0,1,...,6,**{7,}** **{8,}**9,...,15	30	500	0.4464	0.581	0.6844	0.7787
		2000	0.635	0.7665	0.8403	0.8693
		4000	0.7034	0.8228	0.8875	0.8973
	100	500	0.454	0.5924	0.692	0.783
		2000	0.651	0.7696	0.8441	0.8734
		4000	0.7209	0.8312	0.8913	0.9026
0,1,...,14,**{15,}** **{16,}**17,18,...,23	30	500	0.6844	0.8327	0.9027	0.8992
		2000	0.7475	0.8684	0.9217	0.9175
		4000	0.7567	0.8745	0.9255	0.9211
	150	500	0.7498	0.8707	0.9171	0.9125
		2000	0.8243	0.911	0.9392	0.9342
		4000	0.8403	0.9179	0.9437	0.9389

Table 5.7: Temporal Recall results for UCF dataset

Adding NMS algorithm

combination	NMS thresh	PreNMS tubes	overlap thresh 0.9	0.8	0.7	MABO
0,1,...,6,**{7,}** **{8,}**9,...,15	0.7	10000	0.4464	0.581	0.6844	0.7787
		20000	0.635	0.7665	0.8403	0.8693
	0.5		TODO	TODO	TODO	TODO
0,1,...,14,**{15,}** **{16,}**17,18,...,23	0.7	10000	TODO	TODO	TODO	TODO
		20000	0.635	0.7665	0.8403	0.8693
	0.5		TODO	TODO	TODO	TODO

Table 5.8: Temporal Recall results for UCF dataset

5.2.2 Second approach: use progression and progress rate

As we saw before, our first connecting algorithm doesn't have very good recall results. So, we created another algorithm which is base in Hu et al. 2019. This algorithm introduces two 2 metrics according to Hu et al. 2019:

Progression, which describes the probability of a specific action being performed in the TOI. We add this factor because we have noticed that actioness is tolerant to false positives. Progression is mainly a rescoring mechanism for each class (as mentioned in Hu et al. 2019)

Progress rate, which is defined as the progress proportion that each class has been perfomed.

So, each action tube is describes as a set of TOIs

$$T = \{\mathbf{t}_i^{(k)} | \mathbf{t}_i^{(k)} = (t_i^{(k)}, s_i^{(k)}, r_i^{(k)})\}_{i=1:n^{(k)}, k=1:K}$$

where $t_i^{(k)}$ contains TOI's spatiotemporal information, $s_i^{(k)}$ its confidence score and $r_i^{(k)}$ its progress rate.

In this approach, each class is handled seperately, so we discuss action tube generation for one class only. In order to link 2 TOIs, for a video with N video segments, the following steps are applied:

1. For the first video segment (k = 1), initialize an array with the M best scoring TOIs, which will be considered as active action tubes (AT). Correspondly, initialize an array with M progress rates and M confidence scores.

2. For k = 2:N, execute (a) to (c) steps:

   (a) Calculate overlaps between $AT^{(k)}$ and $TOIs^{(k)}$.

   (b) Connect all tubes which satisfy the following criterions:

      i. $overlapscore(at_i^{(k)}, t_j^{(k)}) < \theta, at \varepsilon AT^{(k)}, t \varepsilon TOIs^{(k)}$

      ii. $r(at_i^{(k)}) < r(t_j^{(k)})$ or $r(t_i^{(k)}) - r(at_i(k)) < \lambda$

   (c) For all new tubes update confidence score and progress rate as follows:

   New cofidence score is the average score of all connected TOIs:

$$s_z^{(k+1)} = \frac{1}{n} \sum_{n=0}^{k} s_i^{(n)}$$

   New progress rate is the highest progress rate:

$$r(at_z^{(k+1)} = max(r(at_i^{(k)}), r(t_j^{(k)}))$$

   (d) Keep M best scoring action tubes as active tubes and keep K best scoring action tubes for classification.

This approach has the advantage that we don't need to perform classification again because we already know the class of each final tube. In order to validate our results, now, we calculate the recall only from the tubes which have the same class as the groundtruth tube. Again we considered as positive if there is a tube that overlaps with groudtruth over the predefined threshold.

combination		**overlap thresh**		
sample dur	step	0.3	0.4	0.5
8	6	TODO	TODO	TODO
8	7	TODO	TODO	TODO
8	8	0.3060	0.5672	0.6866
16	8	TODO	TODO	TODO
16	12	TODO	TODO	TODO
16	16	TODO	TODO	TODO

Table 5.9: Recall results for second approach with step = 8, 16 and their corresponding steps

(Pending Table...) As we can see from the table above, the results in recall are not very good either.

5.3 Third approach : use naive algorithm - only for JHMDB

As mention in first approach, Hou, Chen, and Shah 2017 calculates all possible sequences of ToIs in order to the find the best candidates. We rethought about this approach and we concluded that it could be implement for JHMDB dataset if we reduce the number of proposed ToIs, produced by TPN, to 30 for each video clip. We exploited the fact that JHMDB dataset's videos are trimmed, so we do not need to look for action tubes starting in the second video clip which saves us a lot of memory. On top of that, we modified our code in order to be memory efficient at the most writing some parts in CUDA programming language, saving a lot of processing power, too.

So, after computing all possible combinations starting of the first video clip and ending in the last video clip, we keep only the **k-best scoring tubes (k = 500)** . In the follown table, we can see the recall results for sample durations = 8 and 16.

combination		**overlap thresh**		
sample dur	step	0.3	0.4	0.5
8	6	0.7873	0.8657	0.9366
8	7	TODO	TODO	TODO
8	8	0.7910	0.8806	0.9515
16	8	TODO	TODO	TODO
16	12	TODO	TODO	TODO
16	16	0.7910	0.8806	0.9478

Table 5.10: Recall results for second approach with

From the above table, we notice that sample duration = 8 is slightly better that the 16.

Chapter 6

Classification stage

6.1 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). **Pending Change the sxima...**

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 150 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 6.1, in which blue arrows show that features go straight up to the classifier. **(Pending... Change that)**

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.

6.2 Preparing data for classification - Linear and RNN Classifiers

(Pending... Introduction about Linear and RNN classifiers) (Pending.. also an image of Linear classifier)

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

The solution we came with, is to precompute the features for both positive video tubes and negative video tubes. 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 Yang et al. 2017, 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 learnvery easily, with little data. So we chose to try a support vector machine classifier.

6.3 Support Vector Machine (SVM)

6.3.1 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 Girshick 2015 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 6.1 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	4.20	4.40
			2	2.29	2.68	2.86
			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 6.1: 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

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.

6.3.2 Temporal pooling

After getting first results, we implement a temporal pooling function inspired from Hou, Chen, and Shah 2017. 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 6.2: mAP results for second approach using temporal pooling

6.3.3 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	TODO	TODO	TODO
			2	TODO	TODO	TODO
			3	TODO	TODO	TODO
			4	TODO	TODO	TODO
			5	TODO	TODO	TODO
k,256,16,7,7	2,256,16,7,7	Yes	1	25.07	26.91	29.11
			2	TODO	TODO	TODO
			3	TODO	TODO	TODO
			4	TODO	TODO	TODO
			5	TODO	TODO	TODO

Table 6.3: mAP results for

6.3.4 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 6.4

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 6.4: Results after increasing the number of total tubes

6.3.5 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 6.5

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 6.5: mAP results for features extracted with modified 3D RoiAlign

6.4 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.

6.4.1 Extract features

We

6.4.2 Regular training

6.5 Classifying dataset UCF

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

6.6 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 6.2.

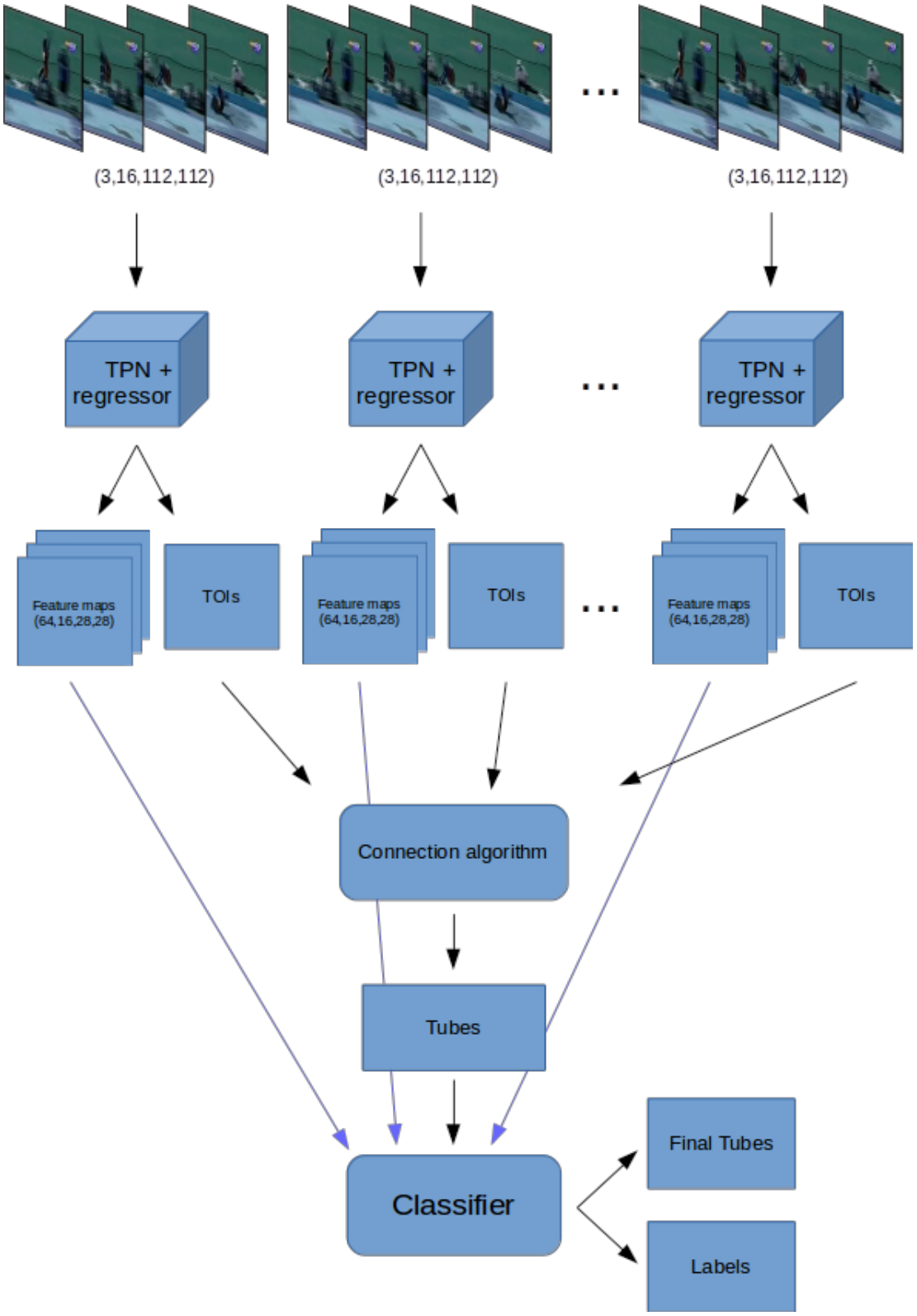


Figure 6.1: Structure of the whole network

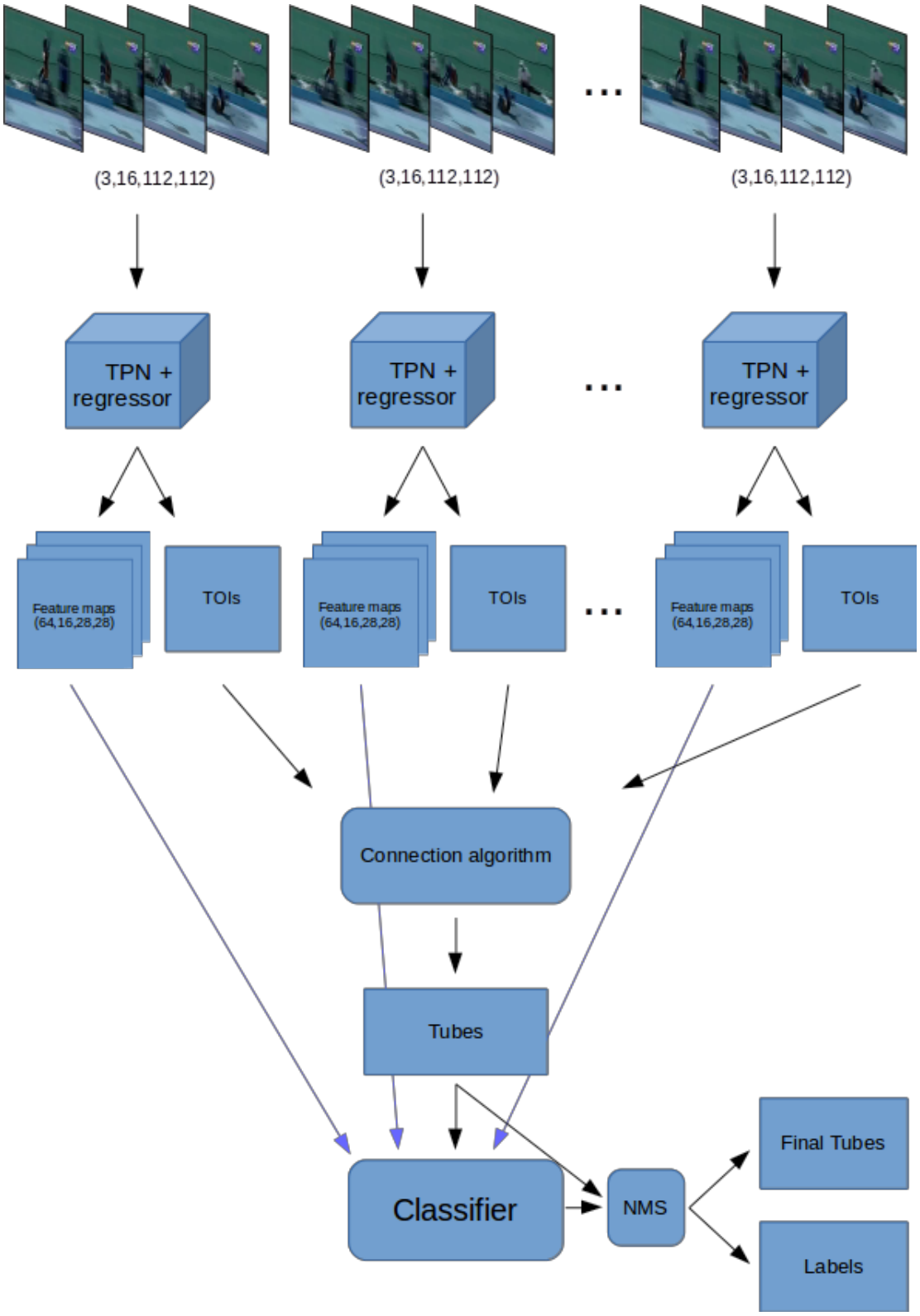


Figure 6.2: Structure of the network with NMS

Chapter 7

Conclusion

Bibliography

Aggarwal, J.K. and M.S. Ryoo (Apr. 2011). “Human Activity Analysis: A Review”. In: *ACM Comput. Surv.* 43.3, 16:1–16:43. issn: 0360-0300. doi: 10.1145/1922649.1922653. url: http://doi.acm.org/10.1145/1922649.1922653.

Kuehne, H. et al. (2011). “HMDB: a large video database for human motion recognition”. In: *Proceedings of the International Conference on Computer Vision (ICCV)*.

Soomro, Khurram, Amir Roshan Zamir, and Mubarak Shah (2012). “UCF101: A dataset of 101 human actions classes from videos in the wild”. In: *arXiv preprint arXiv:1212.0402*.

Girshick, Ross B. et al. (2013). “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *CoRR* abs/1311.2524. arXiv: 1311.2524. url: http://arxiv.org/abs/1311.2524.

Jhuang, H. et al. (Dec. 2013). “Towards understanding action recognition”. In: *International Conf. on Computer Vision (ICCV)*, pp. 3192–3199.

Ji, S. et al. (2013). “3D Convolutional Neural Networks for Human Action Recognition”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.1, pp. 221–231. doi: 10.1109/TPAMI.2012.59.

Gkioxari, Georgia and Jitendra Malik (2014). “Finding Action Tubes”. In: *CoRR* abs/1411.6031. arXiv: 1411.6031. url: http://arxiv.org/abs/1411.6031.

Jain, M. et al. (2014). “Action Localization with Tubelets from Motion”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 740–747. doi: 10.1109/CVPR.2014.100.

Karpathy, A. et al. (2014). “Large-Scale Video Classification with Convolutional Neural Networks”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1725–1732. doi: 10.1109/CVPR.2014.223.

Simonyan, Karen and Andrew Zisserman (2014). “Two-stream convolutional networks for action recognition in videos”. In: *Advances in Neural Information Processing Systems*, pp. 568–576.

Tran, Du et al. (2014). “C3D: Generic Features for Video Analysis”. In: *CoRR* abs/1412.0767. arXiv: 1412.0767. url: http://arxiv.org/abs/1412.0767.

Girshick, Ross (2015). “Fast R-CNN”. In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. ICCV ’15. Washington, DC, USA: IEEE Computer Society, pp. 1440–1448. isbn: 978-1-4673-8391-2. doi: 10.1109/ICCV.2015.169. url: http://dx.doi.org/10.1109/ICCV.2015.169.

He, Kaiming et al. (2015). “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385. arXiv: 1512.03385. url: http://arxiv.org/abs/1512.03385.

Ren, Shaoqing et al. (2015). “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. Montreal, Canada: MIT Press, pp. 91–99. url: http://dl.acm.org/citation.cfm?id=2969239.2969250.

Weinzaepfel, Philippe, Zaïd Harchaoui, and Cordelia Schmid (2015). “Learning to track for spatio-temporal action localization”. In: *CoRR* abs/1506.01929. arXiv: 1506.01929. url: http://arxiv.org/abs/1506.01929.

Feichtenhofer, Christoph, Axel Pinz, and Andrew Zisserman (2016). “Convolutional Two-Stream Network Fusion for Video Action Recognition”. In: *CoRR* abs/1604.06573. arXiv: 1604.06573. url: http://arxiv.org/abs/1604.06573.

Peng, Xiaojiang and Cordelia Schmid (Oct. 2016). “Multi-region two-stream R-CNN for action detection”. In: *ECCV - European Conference on Computer Vision*. Vol. 9908. Lecture Notes in Computer Science. Amsterdam, Netherlands: Springer, pp. 744–759. doi: 10.1007/978-3-319-46493-0_45. url: https://hal.inria.fr/hal-01349107.

Girdhar, Rohit et al. (2017). “Detect-and-Track: Efficient Pose Estimation in Videos”. In: *CoRR* abs/1712.09184. arXiv: 1712.09184. url: http://arxiv.org/abs/1712.09184.

He, Kaiming et al. (2017). “Mask R-CNN”. In: *CoRR* abs/1703.06870. arXiv: 1703.06870. url: http://arxiv.org/abs/1703.06870.

Hou, Rui, Chen Chen, and Mubarak Shah (2017). “Tube Convolutional Neural Network (T-CNN) for Action Detection in Videos”. In: *CoRR* abs/1703.10664. arXiv: 1703.10664. url: http://arxiv.org/abs/1703.10664.

Kalogeiton, Vicky et al. (2017). “Action Tubelet Detector for Spatio-Temporal Action Localization”. In: *CoRR* abs/1705.01861. arXiv: 1705.01861. url: http://arxiv.org/abs/1705.01861.

Kay, Will et al. (2017). “The Kinetics Human Action Video Dataset”. In: *CoRR* abs/1705.06950. arXiv: 1705.06950. url: http://arxiv.org/abs/1705.06950.

Singh, Gurkirt et al. (2017). “Online Real time Multiple Spatiotemporal Action Localisation and Prediction”. In:

Tran, Du et al. (2017). “A Closer Look at Spatiotemporal Convolutions for Action Recognition”. In: *CoRR* abs/1711.11248. arXiv: 1711.11248. url: http://arxiv.org/abs/1711.11248.

Yang, Jianwei et al. (2017). “A Faster Pytorch Implementation of Faster R-CNN”. In: *https://github.com/jwyang/faster-rcnn.pytorch*.

Hara, Kensho, Hirokatsu Kataoka, and Yutaka Satoh (2018a). “Can Spatiotemporal 3D CNNs Retrace the History of 2D CNNs and ImageNet?” In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

— (2018b). “Can Spatiotemporal 3D CNNs Retrace the History of 2D CNNs and ImageNet?” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6546–6555.

Hu, Bo et al. (2019). “Progress Regression RNN for Online Spatial-Temporal Action Localization in Unconstrained Videos”. In: *CoRR* abs/1903.00304. arXiv: 1903.00304. url: http://arxiv.org/abs/1903.00304.