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 network for recognizing and localising human actions in videos. Our network aims to spatio-temporally localize a recognized action within a video producing a sequence of 2D boxes, one per frame, which includes the actor performing the recognized action.

Detecting and Recognizing actions in videos is one of the biggest challenges in the field of Computer Vision. Most recent approaches includes an object detection network which proposes bounding boxes per frame, a linking method for creating candidate action tubes and a classifier for classifying these. On top of that, most of these approaches extract temporal information from a network which estimates optical flow in frame level. The introduction of 3D Convolutional Networks has helped us estimating spatio-temporal information from videos and simultaneously extract spatio-temporal features. Our approach tries to combine the benefits from using object detection networks and 3D Convolution.

We design a network whose structure is based on standard action localization networks and we name it ActionNet. Its first element is a 3D ResNet34 which is used for spatio-temporal feature extraction. Also, we design a network for proposing action tubes based on spatio-temporal features, called Tube Proposal Network. This network is an expansion of Region Proposal Network and it gets as input the extracted features and outputs k-proposed action tubes. We explore 2 approaches for defining 3D anchors, which TPN uses. On top of that, we design a linking algorithm for connecting proposed action tubes. Finally, we explore several classification techniques including a SVM classifier and a MLP for datasets JHMDB and UCF101.

Key words

Action Localization, Action Recognition, Action Tubes

Acknowledgements

Pending...

Efstatios Galanakis,

Athens, December -1, 2019

This thesis is also available as Technical Report CSD-SW-TR-42-14, National Technical University of Athens, School of Electrical and Computer Engineering, Department of Computer Science, Software Engineering Laboratory, December 2019.

URL: `http://www.softlab.ntua.gr/techrep/`
FTP: `ftp://ftp.softlab.ntua.gr/pub/techrep/`

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.

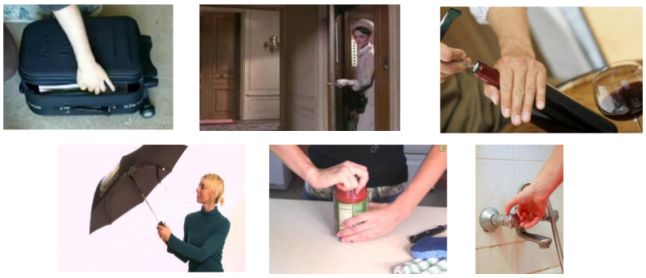


Figure 1.1: Examples of "Open" action

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) (Kuehne et al. 2011) and UCF-101 (Soomro, Zamir, and Shah 2012).

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 loss functions and evaluation metrics that we used. Also, Chapter 2 presents an brief overview of literature on human action recognition and localization. Chapter 3 introduces the first basic element of our network, Tube Proposal Network (TPN), a network which proposes Tubes of Interest (ToIs), which are sequences of bounding boxes, with are likely to contain a performed action. Furthermore, it contains all the proposed architectures for achieving this. Chapter 4 proposes algorithms for linking the proposed TOIs from every video segment and proposal performance is presented. In Chapter 5, we present all the classification approaches we used for designing our architecture and some classification results. Chapter 6 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 $Y = f(x)$.

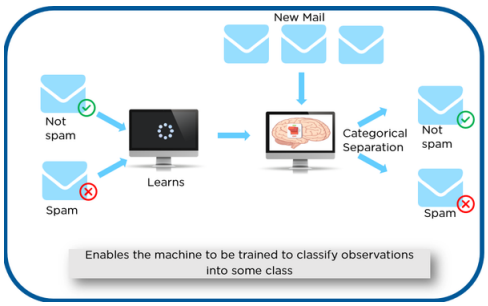


Figure 2.1: Example of supervised Learning

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

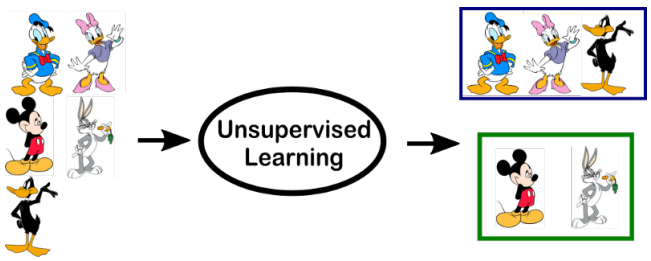


Figure 2.2: Example of unsupervised Learning

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.

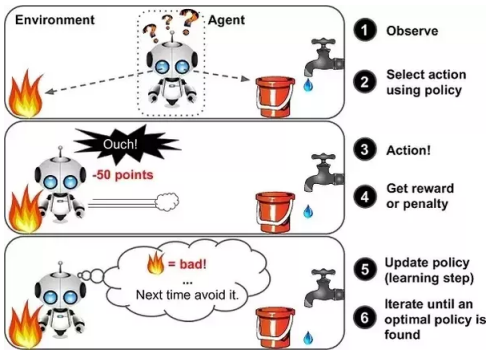


Figure 2.3: Example of Reinforcement Learning

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.

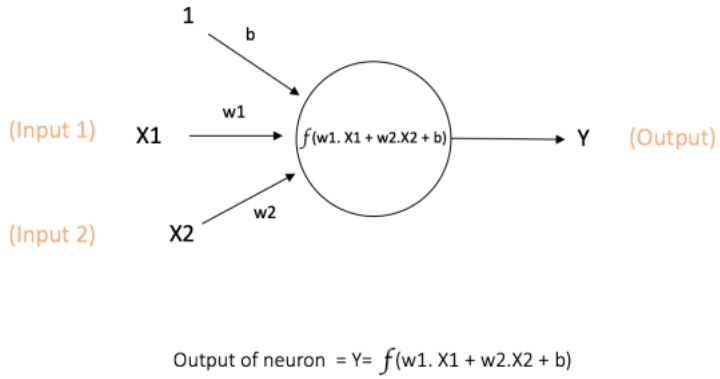


Figure 2.4: An example of a single Neuron

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.

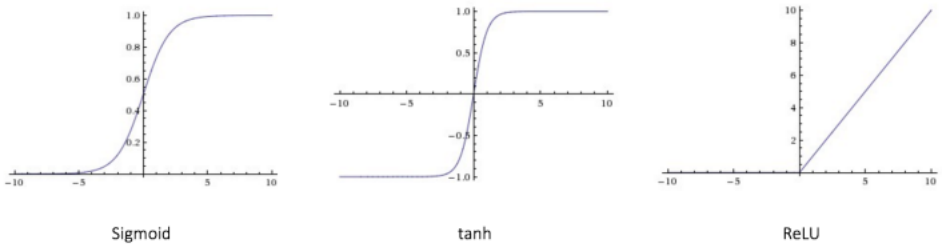


Figure 2.5: Plots of Activation functions

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

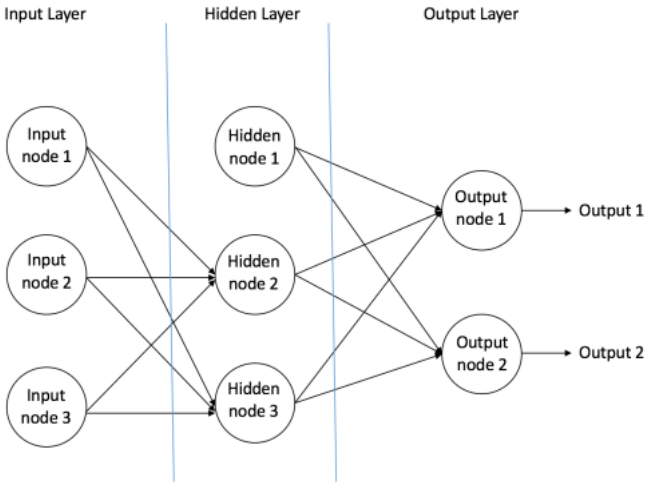


Figure 2.6: An example of a Feedforward Neural Network

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.

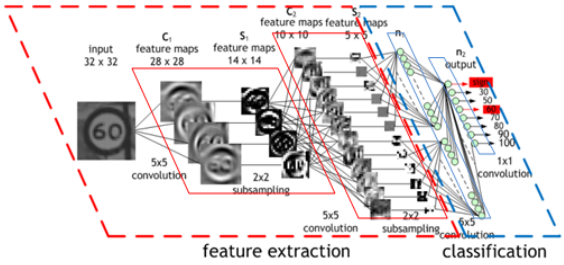


Figure 2.7: Typical structure of a ConvNet

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.

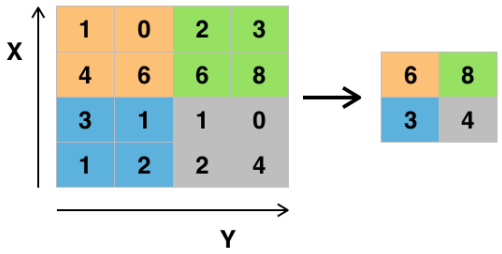


Figure 2.9: Example of Max pooling operation with a 2x2 filter and stride of 2

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.

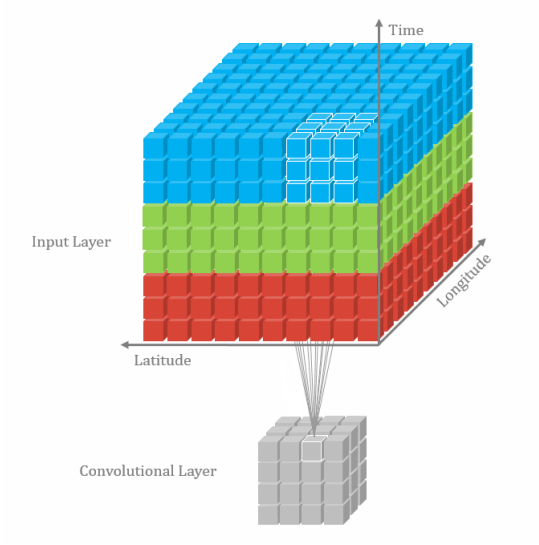


Figure 2.10: 3D Convolution operation

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.

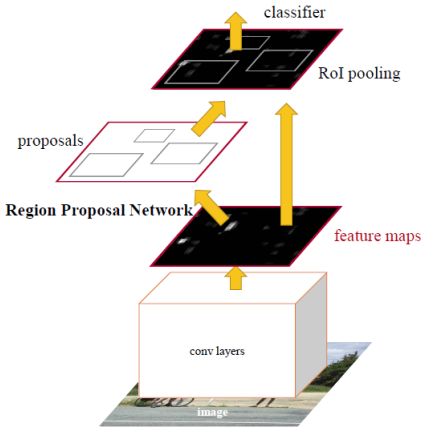


Figure 2.11: Region Proposal Network's structure

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.

2.2.2 Roi Align

The biggest problem facing Object Detection Networks is the need for fixed input size. Classification networks require a fixed input size, which is easy for image classification because it is handle by resizing the input image. However, in object recognition, each proposal has a different size and

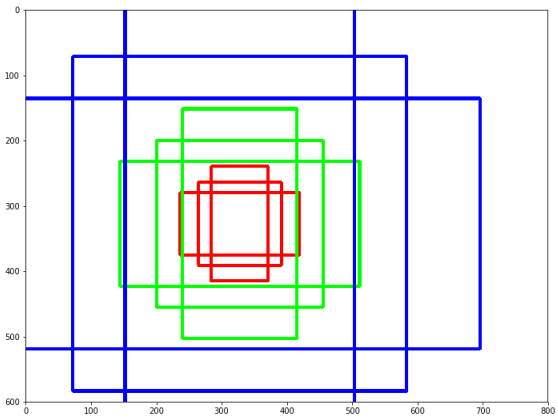


Figure 2.12: Anchors for pixel (320,320) of an image (600,800)

shape. This creates the need for converting all proposals to a fixed shape. At Fast-RCNN(Girshick 2015) and Faster-RCNN(Ren et al. 2015), this operation happens by applying Roi Pooling. However, this wrapping is digitalized because the cell boundaries of the target feature map are forced to realign with the boundary of the input feature maps as shown in Figure 2.13a at the top left diagram. As a result, each target cells may not be in the same size (Figure 2.13c).

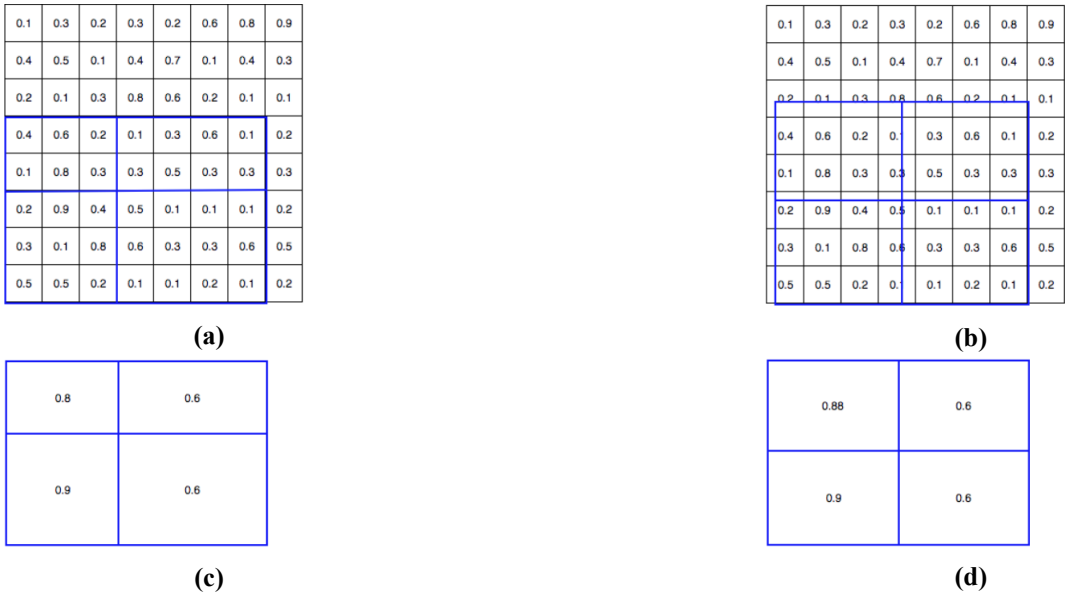


Figure 2.13

On the other hand, Mask-RCNN (He et al. 2017) introduced Roi Align operation. Roi Align avoids digitalizing the boundary of the cells as shown in Figure 2.13b, and achieves to make every target cell to have the same size according to Figure 2.13d. In order to calculate feature maps values, Roi Align uses bilinear interpolation as shown in Figure 2.14. This means that we calculate the value of the desired bins according to their neighbours'.

2.2.3 Non-maximum suppression (NMS) algorithm

That's because the neighbour windows have similar scores to some extent. This Another proble that object detection networks face is that neighbourd bounding boxes have similar scores to some extent. Most object detection systems employ a sliding window or a Region Proposal Network for proposing areas in the image that is likely to contain an object. These techniques,which return several

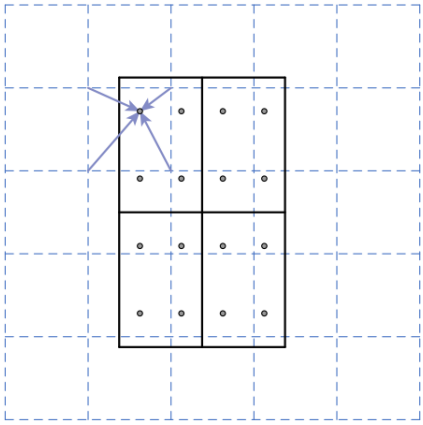


Figure 2.14

areas in the images, achieve high recall performance. However, in these approaches, more that 1 proposal may be related with only one groundtruth object coordinates. This situation creates the need for choosing the best proposals, because, alternatively, hundreds of unnecesssary proposals will be classified. For that reason, Non-Maximum Suppression (NMS) algorithm was proposed for filtering these proposals base on some criteria. NMS gets as input a list of proposal bounding boxes B, their corresponding confidence score S and an overlap threshold N and return as output a list of the final filtered proposals D. NMS algorith's steps are:

1. Initialize an empty list D. Select the proposal with the highest confidence score, remove it from B and add it to D.
2. Calculate the overlap score between this proposal and all the other proposals. For all the proposals that their overlap score is bigger than N, remove from B.
3. From the remaining proposals, picked again the one with the highest score and remove it from B.
4. Repeat steps 2 and 3 until no more proposals are left in list B.

The aforementioned algorithm shows that the whole process depends mostly on a sigle threshold value. So that makes the selection of threshold a crucial factor for the perfomance of the model. In some situations, bad choice of the threshold may make the network to remove bounding boxes with good confidence score, if there are side by side. Figure 2.15 shows a situation like this, where red and blue boxes will be removed because of the presence of the black box.

Soft NMS

A simple and efficinet way to deal the aforementioned situation is to use Soft-NMS algorithm, which is presented in Bodla et al. 2017. Soft-NMS algorithm is based on the idea of reducing condifence score of the proposals propotional to their overlap score, instead of completely removing them. The score calculation follows the formula

$$s_i = \begin{cases} s_i, & overlapscore(M, b_i) < N_t \\ s_i(1 - overlapscore(M, b_i)), & overlapscore(M, b_i) \geq N_t \end{cases}$$

where s_i is the score of proposal i, b_i is the box coordinates of proposal i, M is the coordinates of the box with the maximum confidence and N_t is the overlap threshold. So steps of soft-NMS algorithm are:

1. Select the proposal with the highest confidence score, remove it from B and add it to D.

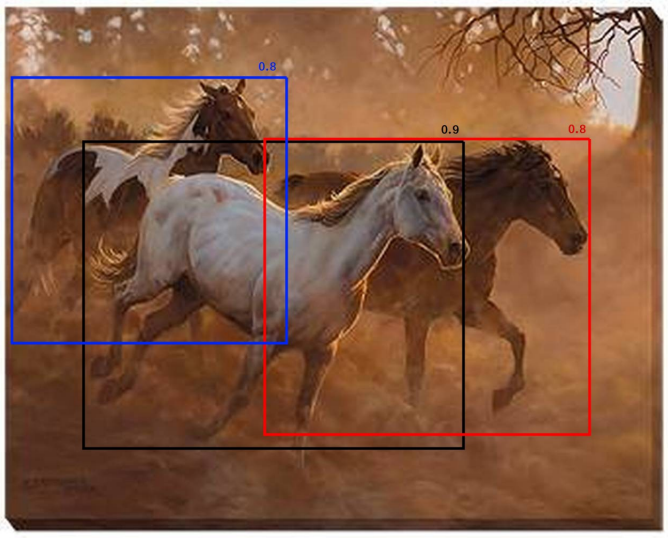


Figure 2.15: Example of a situation where NMS algorithm will remove good proposals

2. Calculate the overlap score between this proposal and all the other proposals. For all the proposals that their overlap score is bigger than N, recalculate their confidence score according to previous formula.

3. From the remaining proposals, picked again the one with the highest score and remove it from B.

4. Repeat steps 2 and 3 until no more proposals are left in list B.

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.

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 2.16.

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:

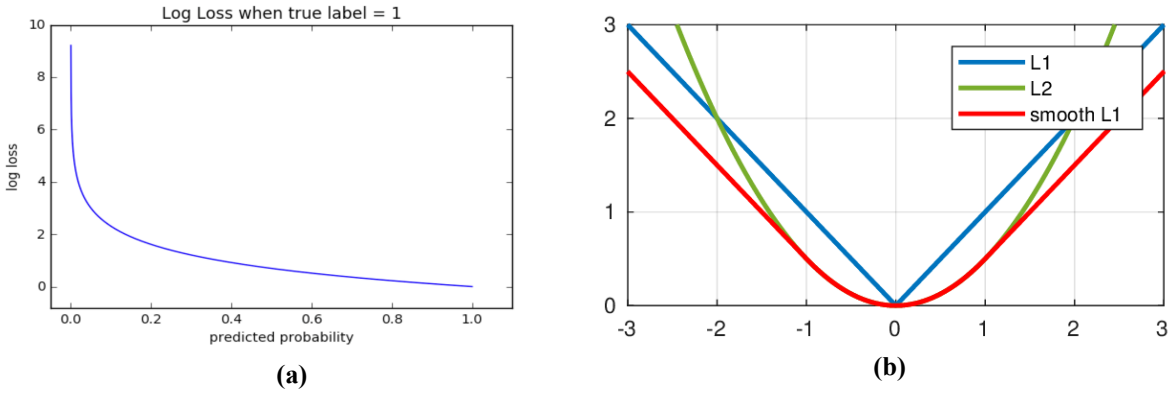
$$smooth_{L1}(x) = \begin{cases} 0.5x^2 & \text{if } x < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

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 2.16 b shows a comparison between L1-norm, L2-norm and smooth-L1 .



(a) (b)

Figure 2.16: (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.

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 2.17. We predefine an IoU threshold (say 0.5) in classifying whether the prediction is a true positive or a false positive.

Figure 2.17: Example of IoU scoring policy

Intersection over Union is defined as:

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

In Figure 2.17, 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.

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:

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

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

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:

$$\begin{aligned}\text{Volume of overlap } &= \|(min(x_2, x_4) - max(x_1, x_3),\\ min(y_2, y_4) - max(y_1, y_3), &min(z_2, z_4) - max(z_1, z_3))\|\\ \text{Volume of union } &= \|(max(x_2, x_4) - min(x_1, x_3),\\ max(y_2, y_4) - min(y_1, y_3), &max(z_2, z_4) - min(z_1, z_3)\end{aligned}$$

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.

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.

mAP

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 mean Average Precision (mAP) metric.

AP (Average precision) Before defining mAP metric, we will define Average Precision metric (AP). AP 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. As mention before, during classification stage, our prediction result in True positive(TP), False positive(FP), True Negative(TN) and False Negative(FN). For object recognition and action localization networks, we don't care about TN. We consider a prediction as True positive when our prediction (an bounding box for object detection network or a sequence of bounding boxes in action localization networks) overlaps with the groundtruth bounding box/action tube over a predefined threshold, and our predicted class is the same as groundtruth's. In addition, we consider a False Negative when either no detection overlaps with the groundtruth bounding box/action tube or our prediction was incorrect. We consider a prediction as False positive, when more that one predictions overlap with the groundtruth. In this situation, we consider the prediction with the biggest confidence score as TP and the rest as FP.

For a class, we need to calculate, first, its precision and recall scores in order to calculate its AP score. We sort our predictions according to their confidence score and for each new prediction we calculate precision and recall values. An example, for a class containing 4 TP and 8 predictions is shown in Table 2.1. Precision and recall are calculated according to the number of elements that are above in the order. So, for rank #3, Precision is calculated as the proposion of TP = 2/3 = 0.67 and Recall as the proportion of TP out of all the possible TP = 2/4 = 0.5.

Rank	Prediction	**Precision**	**Recall**
1	Correct	1.0	0.25
2	Correct	1.0	0.5
3	False	0.67	0.5
4	False	0.5	0.5
5	False	0.4	0.5
6	Correct	0.5	0.75
7	False	0.42	0.75
8	Correct	0.5	1

Table 2.1: Ordered by confidence predictions and their precision and recall values

We plot Precision against Recall and their curve is shown in Figure 2.18. The general definition for Average Precision(AP) is finding the area under the precision-recall curve and its formula is:

$$AP = \int_0^1 p(r)dr$$

Precision and Recall values $\in [0, 1]$, so AP $\in [0, 1]$, too. This integral can be replace with a finite, as we have a finite number of predictions. So its formula is:

$$AP = \sum_{k=1}^{n} P(k)\Delta r(k)$$

where $P(k)$ is the precision until prediction k and Δr is the change in recall from $k-1$ to k.

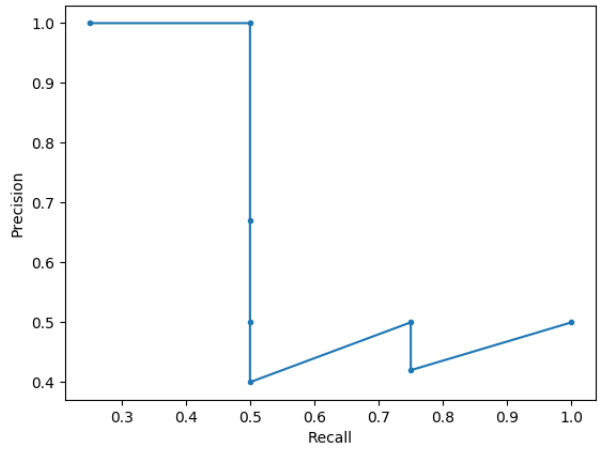


Figure 2.18: Precision/Recall curve

Interpolated Precision At we can see in 2.18, P-R curve has a zigzag pattern as it goes down with false predictions, and goes up with correct. So, before calculation AP, we need to smooth out this zigzag pattern using Intenpolated precision, as introduced in Everingham et al. 2010. Interpolated precision is calculated at each recall level r by taking the maximum precision measured for that r and it is defined as:

$$p_{interp}(r) = \max_{\tilde{r}:\tilde{r}\geq r} p(\tilde{r})$$

where $p(\tilde{r})$ is the measured precision at recall $\tilde{r}$. Graphically, at each recall level, we replace each precision value with the maximum precision value to the right of that recall level. At Figure 2.19 are shown both P-R curvers. The previous P-R curve has blue colour and the interpolated has red color.

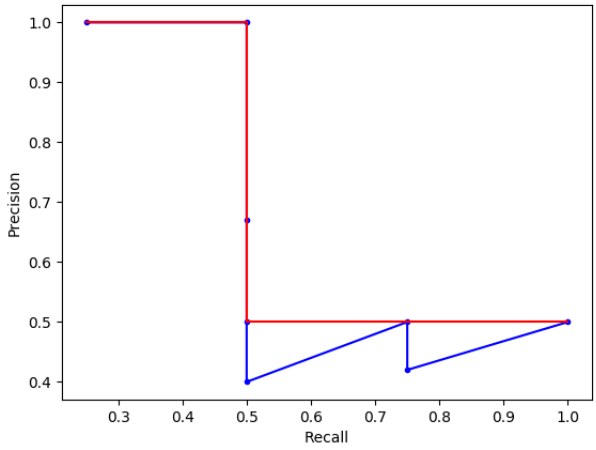


Figure 2.19: Both P-R curves.

In order to calculate AP, we sample the curve at all unique recalls values, whenever the maximum precision drops. On top of that, we define mean Average Precision (mAP) the mean of the AP for each class. So, AP and mAP are defined as

$$AP = \sum (rn + 1 - r_n) p_{interp}(r_{n+1})$$

$$p_{interp}(r_{n+1}) = \max_{\tilde{r}\geq r_{n+1} p(\tilde{r})} p(\tilde{r})$$

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i$$

Mean Average Best Overlap - 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 close the proposals are to the groundtruth tubes. In order to quantify this performance, Mean Average Best Overlap (MABO) was introduced by Winschel, Lienhart, and Eggert 2016. The importance of MABO can be clarified we consider figure 2.20.

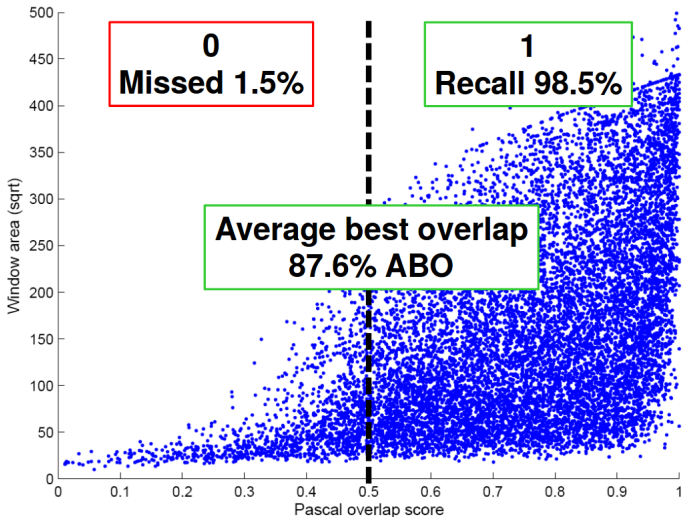


Figure 2.20: Recall versus MABO exaple

As we can see, recall performance is almost perfect, but, MABO perfonmance, which tells us where most proposal overlap scores are gather, is just fine. In order to define MABO, we need first to define Average Best Overlap. Let $c \in C$ denote a class c from the set of all classes C and G^c the set of ground truth annotations of this class in all images; let L be the set of all generated object proposals for all images. Average Best Overlap is defined as the average value of the maximum overlap score, (in our situation, we use intersection over union) of L with each groundtruth annotation $g \in G^c$. The Mean Average Best Overlap (MABO) is defined as the average value of all class ABO values :

$$MABO = \frac{1}{|C|} \sum_{c \in C} [\frac{1}{|G^C|} \sum_{g \in G^c} \max_{l \in L} IoU(g, l)]$$

In order our situation, we consider only one class, for defining proposals, so, MABO identifies with ABO.

2.4 Related work

In this section, we present some of the most relevant methods to our work and others studied for designing this approach. These methods are divided into two sections: *action recognition* and *action localization*. The first part refers to classic action classification methods introduced until recently and the second part, respectively, to recent action localization methods.

2.4.1 Action Recognition

First approaches for action classification consisted of two steps a) compute complex handcrafted features from raw video frames such as SIFT, HOG, ORB features 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

appearance and motion patterns. Another problem was that most of those approaches make assumptions about the circumstances under which the video was taken due to problems such as cluttered background, camera viewpoint variations etc. A review of the techniques, used until 2011, is presented in Aggarwal and Ryoo 2011.

Recent results in deep architectures and especially in image classification motivated researchers to train CNN networks for the task of action recognition. The first significant attempt was made by Karpathy et al. 2014. They design their architecture based on the best-scoring CNN in the ImageNet competition. they explore several methods for fusion of spatio-temporal features using 2D operations mostly and 3D convolution only in slow fusion. Simonyan and Zisserman 2014 used a 2 CNNs, one for spatial information and one for optical flow and combined them using late fusion. They show that extacting spatial context from videos and motion context from optical flow can improve significantly action recognition accuracy. Feichtenhofer, Pinz, and Zisserman 2016 extend this approach by using early fusion at the end of convolutional layers, instead of late fusion which takes places at the last layer of the network. On top that, they used a second network for temporal context which they fuse with the other network using late fusion. Futhermore, Wang et al. 2016 based their method on Simonyan and Zisserman 2014, too. They deal with the problem of capturing long-range temporal context and training their network given limited training samples. Their approach, which they named Temporal Segment Network (TSN), seperates the input video in K segments and a short snippet from each segment is chosen for analysis. Then they fuse the extracted spatio-temporal context, making, eventually, their prediction.

Some other methods included a RNN or LSTM network for classification like Donahue et al. 2014, Ng et al. 2015 and Ma et al. 2017. Donahue et al. 2014 address the challenge of variable lenghts of input and output sequences, explointing convolutional layers and long-range temporal recursions. They propose a Long-term Recurrent Convolutional Network (LRCN) which is capable of dealing with the tasks of action Recognition, image caption and video description. In order to classify a given sequence of frames, LRCN firstly gets as input a frame, and in particular its RGB channels and optical flow and predicts a class label. After that, it extracts video class by averaging label probabilities, choosing the most probable class. Ng et al. 2015 firstly explore several approaches for temporal feature pooling. These techniques include handling video frames individually by 2 CNN architectures: either AlexNet or GoogleNet, and consisted of early fusion, late fusion of a combination between them. Futhermore, they propose a recurrnet neural Network architecture in order to consider video clips as a sequences of CNN activations. Proposed LSTM takes an input the output of the final CNN layer at each consequentive video frame and after five stacked LSTM layers using a Softmax classifier, it proposes a class label. For video classification, they return a label after last time step, max-pool the predictions over time, sum predictions over time and return the max or linearly weight the predictions over time by a factor g, sum them and return the max. They showed that all approaches are 1% different with a bias for using weighting predictions for supporting the idea that LSTM becomes progressively more informed. Last but not least, Ma et al. 2017 use a two-stream ConvNet for feature extraction and either a LSTM or convolutional layers over temporally-constructed feature matrices, for fusing spatial and temporal information. They use a ResNet-101 for extracting feature maps for both spatial and temporal streams. They divide video frames in several segments like Wang et al. 2016 did, and use a temporal pooling layer to extract distinguished features. Taken these features, LSTM extracts embedded features from all segments.

Additionally, Tran et al. 2014 explored 3D Convolutional Networks (Ji, Yang, and Yu 2013) and introduced C3D network which has 3D convolutional layers with kernels $3 \times 3 \times 3$. This network is able to model appearence and motion context simultaneously using 3D convolutions and it can be used as a feature extractor, too. Combining Two-stream architecture and 3D Convolutions, Carreira and Zisserman 2017 proposed I3D network. On top of that, the authors emphasize in the advantages of transfer learning for the task of action recognition by repeating 2D pre-trained weights in the 3rd dimension. Hara, Kataoka, and Satoh 2017 proposed a 3D ResNet Network for action recognition based on Residual Networks (ResNet) (He et al. 2015) and explore the effectiveness of ResNet with 3D

Convolutional kernels. On the other hand,Diba et al. 2017 based their approach on DenseNets(Huang, Liu, and Weinberger 2016) and extend DenseNet architecture by using 3D filters and pooling kernels instead of 2D, naming this approach as DenseNet3D. Futhermore, they introduce Temporal Transition Layer (TTL), which concatenates temporal feature-maps extracted at different temporal depth ranges and replaces DenseNet's transition layer. Last but not least, Tran et al. 2017 experiment with several residual network architectures using combinations of 2D and 3D convolutional layer. Their purpose is to show that a 2D spatial convolution followed by a 1D temporal convolution achieves state of the art classification performance, naming this type of convolution layer as R(2+1)D. A more detailed presentation for Action Recognition techniques used until 2018 is included in Kong and Fu 2018.

2.4.2 Action Localization

As mentioned before, Action Localization can be seen as an extention of the object detection problem. Instead of outputing 2D bounding boxes in a single image, the goal of action localization systems is to output action tubes which are sequences of bounding boxes that contain an performed action. 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 improvement in the performance of Object Detection Networks. This architecture, firstly, proposes regions in the image which are likely to contain an object and then it classifies them using a SVM classifier. Inspired by this architecture, Gkioxari and Malik 2014 design 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 the viterbi connection algorithm. Weinzaepfel, Harchaoui, and Schmid 2015 extend this approach, by performing frame-level proposals and using a tracker for connecting those proposals using both spatial and optical flow features. Also their method performs temporal localization using a sliding window over the tracked tubes.

Peng and Schmid 2016 and Saha et al. 2016 use Faster R-CNN (Ren et al. 2015) instead of RCNN for frame-level proposals, using RPN for both RGB and optical flow images. After getting spatial and motion proposals,Peng and Schmid 2016 fuse them exploring and from each proposed ROI, generate 4 ROIs in order to focus in specific body parts of the actor. After that, they connect the proposal using Viterbi algorithm for each class and perform temporal localization by using a sliding window, with multiple temporal scales and stride using a maximum subarray method. From the other hand, Saha et al. 2016 perform, too, frame-level classification. After that, their method performs fusion based on a combination between the actioness scores of the appearence and motion based proposals and their overlap score. Finally, temporal localization takes place using dynamic programming.

On top of that, Singh et al. 2017 and Kalogeiton et al. 2017 design their networks based on the Single Shot Multibox Detector Liu et al. 2015). Singh et al. 2017 created an online real-time spatio-temporal network. In order their network to execute real-time, Singh et al. 2017 propose a novel and efficient algorithm by adding boxes in tubes in every frame if they overlap more than a threshold, or alternatively, terminate the action tube if for k-frames no box was added. Kalogeiton et al. 2017 designed a two-stream network, which they called ACT-detector, and introduced anchor cuboids. For K frames, for both networks, Kalogeiton et al. 2017 extract spatial features in frame-level, then they stack these features. Finally, using cuboid anchors, the network extracts tubelets, that is a sequence of boxes, with their corresponding classification scores and regression targets. For linking the tubelets, Kalogeiton et al. 2017 follow about the same steps as Singh et al. 2017 did. For temporal localization, they use a temporal smoothing approach.

Most recently, YOLO Network (Redmon et al. 2015) became the inspiration for Hu et al. 2019 and El-Nouby and Taylor 2018. In Hu et al. 2019, concepts of progression and progress rate were introduced. Except from proposing bounding boxes in frame level, they use YOLO together with a RNN classifier for extracting temporal information for the proposals. Based on this information, they create action tubes, seperated into classes. Some other approaches include pose estimation like Luvizon, Picard, and Tabia 2018 do.They proposed a method for calcualatin 2D and 3D poses and then

they performed action classification. They use the diffferentiable Soft-argamax function for estimating 2D and 3D joints, because argmax function is not differentiable. Then, for T adjacent poses, they create an image representation with time and N_j joins as $x - y$ dimensions and having 2 channes for 2D poses and 3 channels for 3D poses. They use Convolutional Layers in order to produce action heats and then using max plus min pooling and a Softmax activation they perform action classification. Most of aforementioned networks use per-frame spatial proposals and extract their temporal infomation by calculating optical flow. On the other hand, Hou, Chen, and Shah 2017 design an architecture which icludes proposal in video segment level, which they called Tube CNN (T-CNN). Video segment level means that the whole video is seperated into equal length video clips, and using a C3D for extracting features, it returns spatio-temporal proposals. After getting proposals, Hou, Chen, and Shah 2017 link the tube proposals by an algorithm based on tubes' actioness score and overlap. Finally, classification operation is performed for the linked video proposals.

Chapter 3

Tube Proposal Network

3.1 Our implementation's architecture

In this chapter, we get involved with Tube Proposal Network(TPN), one of the basic elements of ActionNet. Before describing it, we present the whole structure of our model. 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 2018 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.

The basic procedure ActionNet follows is:

1. Given a video, we seperate it into video segments. These video segments in some cases overlap temporally and in some others don't.
2. For each video segment, after performing spatiotemporal resizing, we feed its frames into ResNet34 in order to perform feature extraction. These activation maps are, next, fed into TPN for proposing sequences of bounding boxes. We name them Tubes of Interest (ToIs), like Hou, Chen, and Shah 2017, and are likely to contain a person performing an action. N
3. After getting proposed ToIs for each video segment, using a linking algorithm, ActionNet finds final cancidate action tubes. These action tubes are given as input to a classifier in order to get their action class.

A diagram of ActionNet is shown at Figure 3.1.

3.2 Introduction to TPN

The main purpose of Tube Proposal Network (TPN) 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.

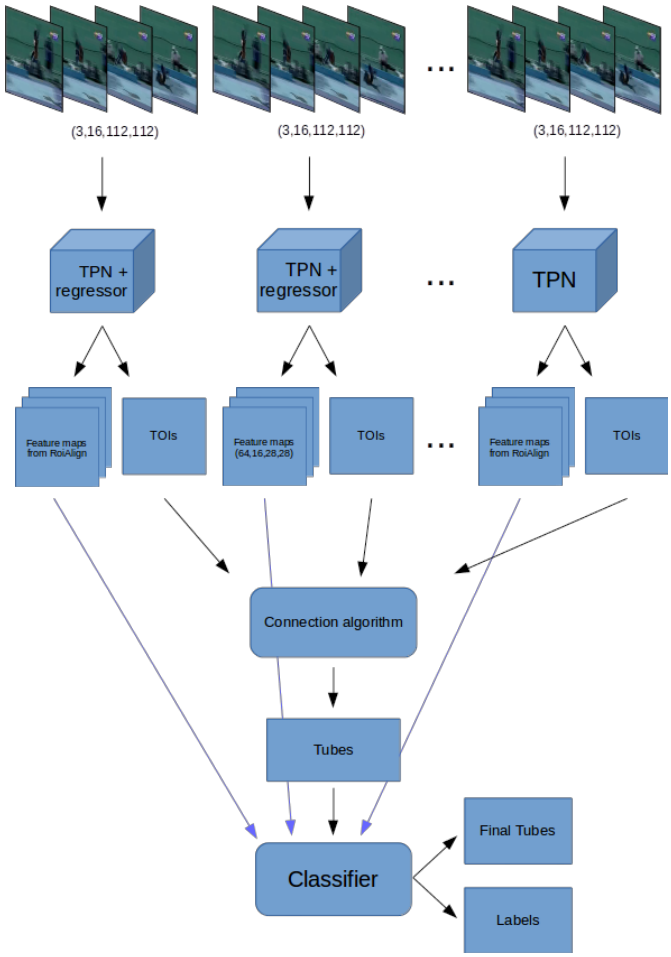


Figure 3.1: Structure of the whole network

3.3 Preparation for TPN

3.3.1 Preparing data

Before getting a video as input to extract its features and ToIs, this video has to be preprocessed. Preprocess procedure is the same for both approaches of TPN. 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 2018. 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 3.2 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 3.2b and 3.2d show that).

3.3.2 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

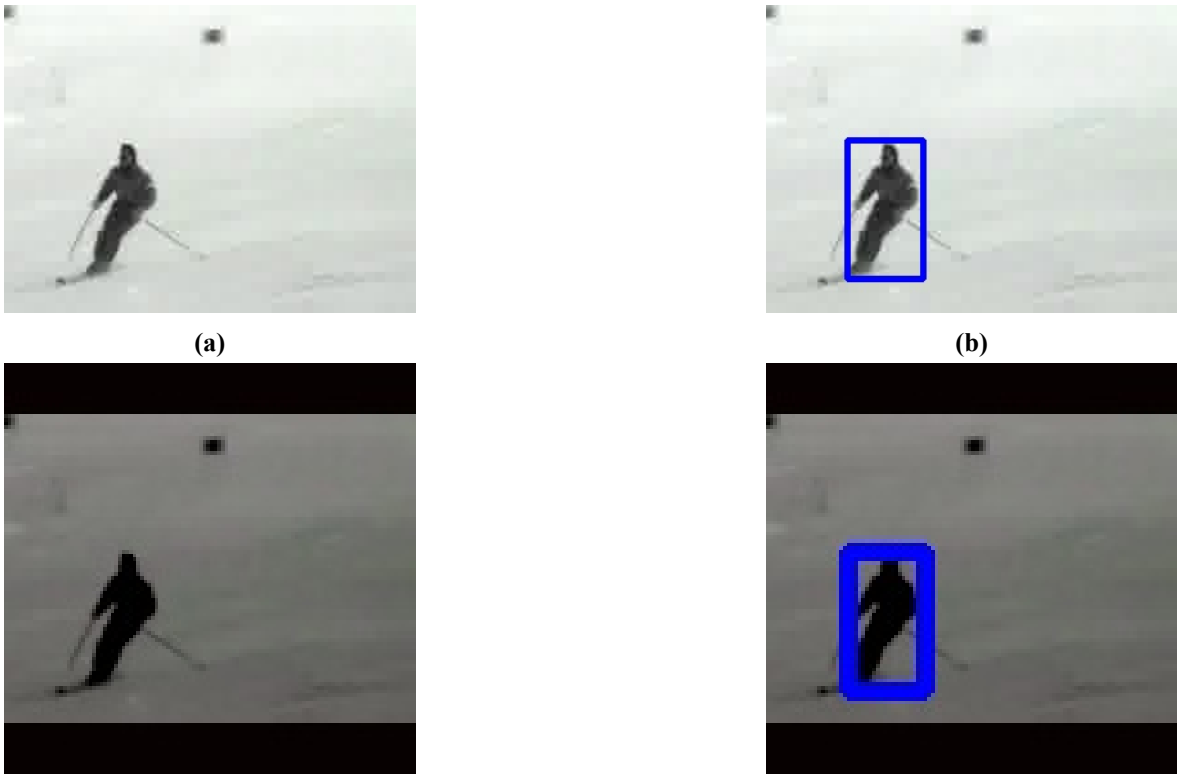


Figure 3.2: At (a), (b) frame is its original size and at (c), (d) same frame after preprocessing part

(256,16,7,7). We feed this feature map to TPN, which is described in following sections.

3.4 3D anchors as 6-dim vector

3.4.1 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 3.3.

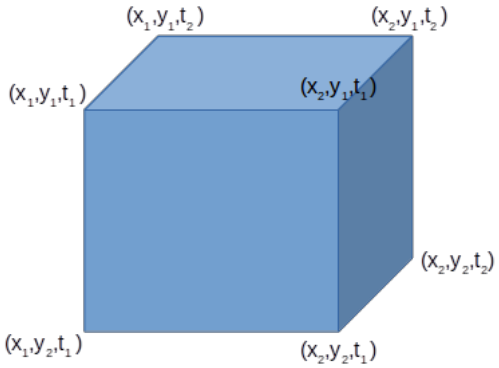


Figure 3.3: An example of the anchor $(x_1, y_1, t_1, x_2, y_2, t_2)$

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. In our approach, we use the same anchors for both datasets, because we want our network not to be dataset-specific but to be able to generalize for several datasets. As sample duration, we chose 16 frames per video segment because our pre-trained ResNet is trained for video clips with that duration. So the structure of TPN is:

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

which is shown in figure 3.4

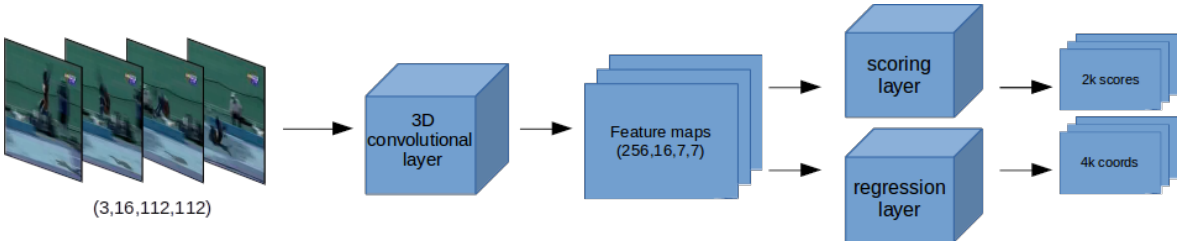


Figure 3.4: Structure of TPN

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

3.4.2 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 3.5, these tubes are 3D boxes which include all the groundtruth rois, which are different for each frame.

Figure 3.5: 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. We consider an anchor as foreground if its overlap score with a groundtruth action tube is bigger than 0.5. Otherwise, it is considered as background anchor. We use scoring layer in order to correctly classify those anchors and we use Cronss Entropy Loss as loss function. We have a lot of anchors for proposing an anction but few numbers of actions, so we choose 256 anchors in total for each batch. We set the maximum number of foreground anchors to be 25% of the 256 anchors and the rest are the background.

Classifying correctly an anchor isn't enough for proposing an action tube. It is necessary to overlap as much as possible with the groundtruth action tubes. That's the reason we use a regression layer. This layer "moves" the cuboid closer to the area that it is believed that is closer to the action. For regression loss we use smooth-L1 loss as proposed in Girshick et al. 2013. In order to calculate the regression targets, we use pytorch FasterRCNN implementation (Yang et al. 2017) for bounding box regression and we modified the code in order to extend it for 3 dimensions. So we have:

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

where *x, y, z, w, h, d* denote the 3D box's center coordinates and its width, 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*). Of course, we calculate the regression loss only for the foreground anchors and not for the background, so at the most we will calculate 64 targets for each batch.

To sum up training procedure, we train 2 layers for our TPN, scoring and reggression layers. The training loss includes the training losses obtained by these layers and its formula is:

$$L = \sum_i L_{cls}(p_i, p_i^*) + \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

where:

- L_{cls} is the Cross Entropy loss we use for classifying the anchors, with p_i is the predicted label, p_i^* is the groundtruth class and $p_i, p_i^* \in \{0, 1\}$
- L_{reg} is the smooth-L1 loss function, which multiply it with p_i^* in order to set active only when we have a positive anchor ($p_i^* = 1$) and to be deactivated for background anchors ($p_i^* = 0$).

3.4.3 Validation

Validation procedure is a bit similar to training procedure. We randomly select 16 frames from a validation video and we examine if there is at least 1 proposed TOI which overlaps ≥ 0.5 with each groundtruth action tube and we get recall score. In order to get good proposals, after getting classification scores and targets prediction from the corresponding layers, we use Non-Maximum Suppresion (NMS) algorithm. We set NMS threshold equal with 0.7, and we keep the first 150 cuboids with the biggest score.

3.4.4 Modified Intersection over Union(mIoU) TODO check again

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

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

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

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

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

In order to verify our idea, we train TPN using both IoU and mIoU criterion for tube-overlapping. At Table 3.1 we can see the performance in each case for both datasets, JHMDB and UCF. Recall threshold for this case is 0.5 and during validation, we use regular IoU for defining if 2 tubes overlap.

Dataset	Criterion	Recall(0.5)
JHMDB	IoU	0.70525
	mIoU	0.7052
UCF	IoU	0.4665
	mIoU	0.4829

Table 3.1: Recall results for both datasets using IoU and mIoU metrics

Table 3.1 shows that modified-IoU give us slightly better recall performance only in UCF dataset. Thats reasonable, because JHMDB dataset uses trimmed videos so time duration doesn't affect a lot. So, from now own, during training we use mIoU as overlapping scoring policy.

3.4.5 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 came with the idea to add 1 scoring layer and 1 reggression layer which will proposed ToIs with fixed duration equal with sample duration and they will take into account the spatial information produced by activation maps. The new structure of TPN is shown in figure 3.6. After getting proposals from both scores, we concat them with ration 1:1 between ToI extracted from those 2 subnetworks.

Our goal is to "compress" feauture maps in temporal dimension in order to propose action tubes according only to the spatial information. So, we came with 2 techniques for doing such thing:

1. Use 3D Convolutional Layers with kernel size = (sample duration, 1,1), stride =1 and no padding for scoring and regression. This kernel "looks" only in the temporal dimension of the activation maps and doesn't consider any spatial dependencies.
2. Get the average values from temporal dimension and then use a 2D Convolutional Layer for scoring and regression.

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

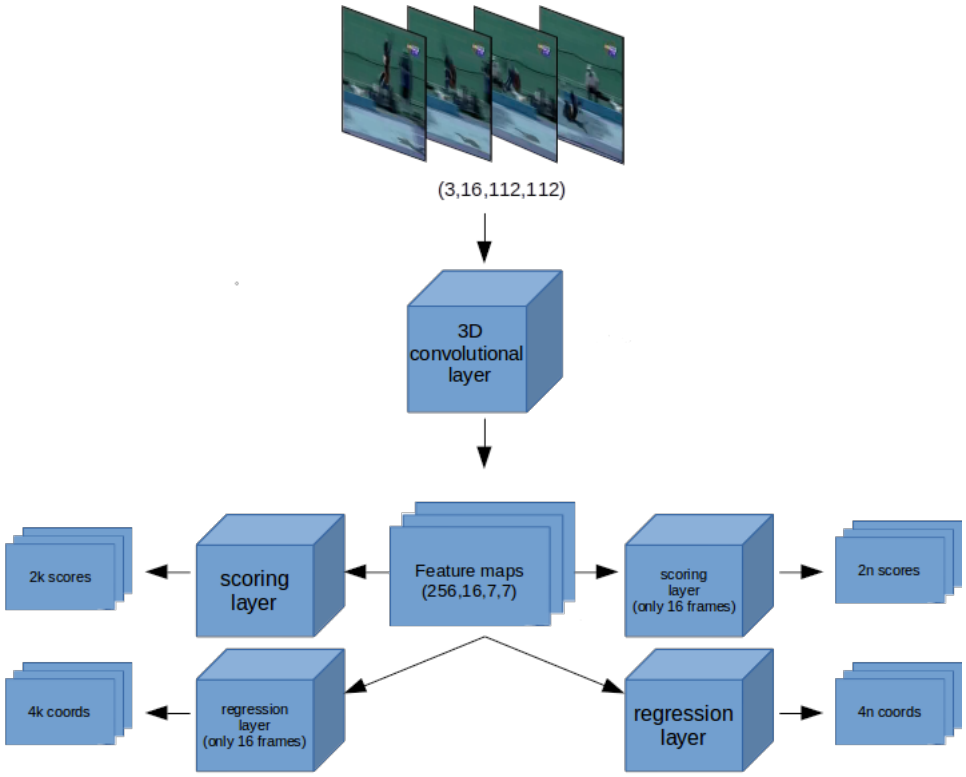


Figure 3.6: TPN structure after adding 2 new layers, where k = 5n.

same procedure. For training loss, we have 2 different cross-entropy losses and 2 different smooth-L1 losses, each for every layer correspondly. So training loss is, now, defined as :

$$L = \sum_i L_{cls}(p_i, p_i^*) + \sum_i L_{cls}(p_{fixed,i}, p_{fixed,i}^*) + \sum_i p_i^* L_{reg}(t_i, t_i^*) + \sum_i p_{fixed,i}^* L_{reg}(t_{fixed,i}, t_{fixed,i}^*) \tag{3.1}$$

where:

- L_{cls} is the Cross Entropy loss we use for classifying the anchors, with p_i is the predicted label, p_i^* is the groundtruth class and $p_i, p_i^* \in \{0, 1\}$
- L_{reg} is the smooth-L1 loss function, which multiply it with p_i^* in order to set active only when we have a positive anchor ($p_i^* = 1$) and to be deactivated for background anchors ($p_i^* = 0$).
- p_i are the anchors from scoring layers with mutable time duration and p_i^* are their corresponding groundtruth label.
- $p_{fixed,i}$ are the anchors from scoring layers fixed time duration = 16 and $p_{fixed,i}^*$ are their corresponding groundtruth label.

We train our TPN Network using both techniques and their recall performance is show in Table 3.2.

As we can see from the previous results, the new layers increased recall performance significantly. On top of that, Table 3.2 shows that getting the average values from time dimension gives us the best results.

Dataset	Fix-time anchors	Type	Recall(0.5)
JHMDB	No	-	0.7052
	Yes	Kernel	0.6978
		Mean	0.7463
UCF	No	-	0.4829
	Yes	Kernel	0.4716
		Mean	0.4885

Table 3.2: Recall results after adding fixed time duration anchors

3.4.6 Adding regressor

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

Regression procedure

At first, for each proposed ToI, we get its corresponding activation maps using 3D Roi Align. These features are given as input to a regressor. This regressor returns $16 \cdot 4$ predicted transforms $(\delta_x, \delta_y, \delta_w, \delta_h)$, 4 for each frame, where δ_x, δ_y specify the coordinates of proposal's center and δ_w, δ_h its width and height, as specified in Girshick et al. 2013. We keep only the predicted translations, for the frames that are $\geq t_1$ and $< t_2$ and for the other frames, we set a zero-ed 2D box. After that, we modify each anchor from a cuboid written like $(x_1, y_1, t_1, x_2, y_2, t_2)$ to a sequence of 2D boxes, like:
$(0, 0, 0, 0, ..., x_{T_1}, y_{T_1}, x'_{T_1}, y'_{T_1}, ..., x_i, y_i, x'_i, ..., x_{T_2}, y_{T_2}, x'_{T_2}, y'_{T_2}, 0, 0, 0, 0, ...)$,
where:

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

Training In order to train our Regressor, we follow about the same steps followed previously for previous TPN's training procedure. This means that we randomly pick 16 ToI from those proposed by TPN's scoring layer. From those 16 tubes, 4 are foreground tubes, which meands 25% of the total number of the tubes as happened previously. We extract their corresponding features using 3D Roi Algin and calculate their targets like we did for regression layer. We feed Regressor Network with these features and compare the predicted targets with the expected. Again, we use smooth-L1 loss for loss function, calculated only for foreground ToIs. So, we add another parameter in training loss

formula which is now defines as:

$$L = \sum_i L_{cls}(p_i, p_i^*) + \sum_i L_{cls}(p_{fixed,i}, p_{fixed,i}^*) + \sum_i p_i^* L_{reg}(t_i, t_i^*) + \sum_i p_{fixed,i}^* L_{reg}(t_{fixed,i}, t_{fixed,i}^*) + \sum_i q_i^* L_{reg}(c_i, c_i^*) + \tag{3.2}$$

where except the previously defined parameters, we set c_i as the regression targets for picked tubes q_i. These tubes are the ones randomly selected from the proposed ToIs and q_i^* are their corresponding groundtruth action tubes, which are the closest to each q_i tube. Again we use q_i^* as a factor because we consider a tube as background when it doesn't overlaps with any groundtruth action tube more that 0.5 .

First regression Network

The architecture of reggression network is show in Figure 3.7, and it is described below:

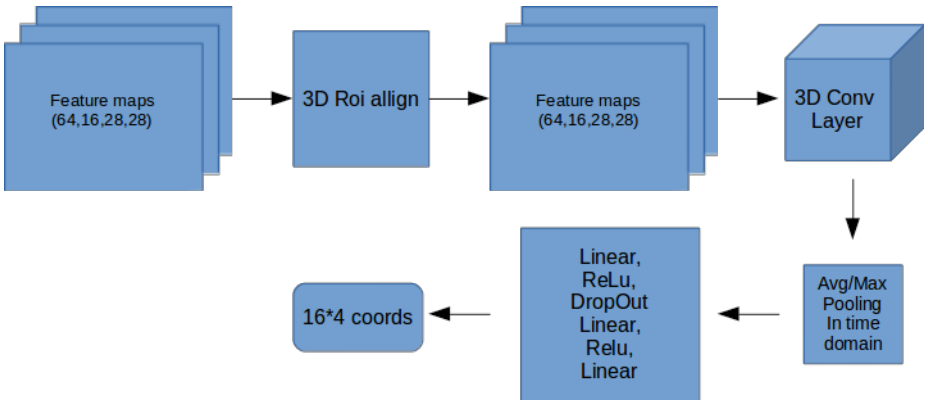


Figure 3.7: Structure of Regressor

1. Regressor is consisted, at first, with a 3D convolutional layer with kernel = 1, stride = 1 and no padding. This layer gets as input ToI's normalized activation map extracted by 3D Roi Align.
2. After that, we calculate the average value in time domain, so from a feature map with dimensions (64,16,7,7), we get as output a feature map (64,7,7).
3. These feature maps are given as input to a Linear layer followerd by a Relu Layer, a Dropout Layer, another Linear Layer and Relu Layer and a final Linear.

We use Recall metric In order to assess the performance of regressor. We calculate 3 recall perfomaces:

Cuboid Recall, which is the recall perfomance for proposed cuboids. We interested in this metric, because, we want to know how good are our proposals before modifying them into sequences of boxes.

Single frame Recall, which is the recall perfomance for the proposed ToI against the groundtruth tubes.

Follow-up Single Frame Recall, which is the recall performance for only the cuboids that were over the overlap threshold between proposed cuboids and groundtruth cuboids. We uses this metric in order to know how many of our proposed cuboids end up in being good proposals.

Dataset	Pooling	Cuboid	Singl. Fr.	Follow-up S.F.
JHMDB	avg	0.8545	0.7649	0.7183
	max	0.8396	0.7761	0.5783
UCF	avg	0.5319	0.4694	0.5754
	max	0.5190	0.5021	0.5972

Table 3.3: Recall results after convertying cuboids into sequences of frames

As the above results show, we get lower recall perfomance in frame-level. On top of that, when we translate a cuboid into a sequence of boxes, we miss 20-40% of our proposals. This means that we don't modify good enough our cuboids, algouth we get only 10% decrease. Probably, we get such score from cuboids, that even though didn't overlap well (according to overlap threshold), achieve to become a good proposal in frame-level and in temporal level.

3.4.7 Changing Regressor - from 3D to 2d

After getting first recall results, we experiment using another architecture for the regressor network, in ordet to solve the modification problem, introduced in previous section. Instead of having a 3D Convolutional Layer, we will use a 2D Convolutional Layer in order to treat the whole time dimension as one during convolution operation. So, as shown in Figure 3.8, the 2^{nd} Regression Network is about the same with first one, with 2 big differences:

1. We performing a pooling operation at the feature maps extracted by 3D Roi Align operation, after we are normalized.

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

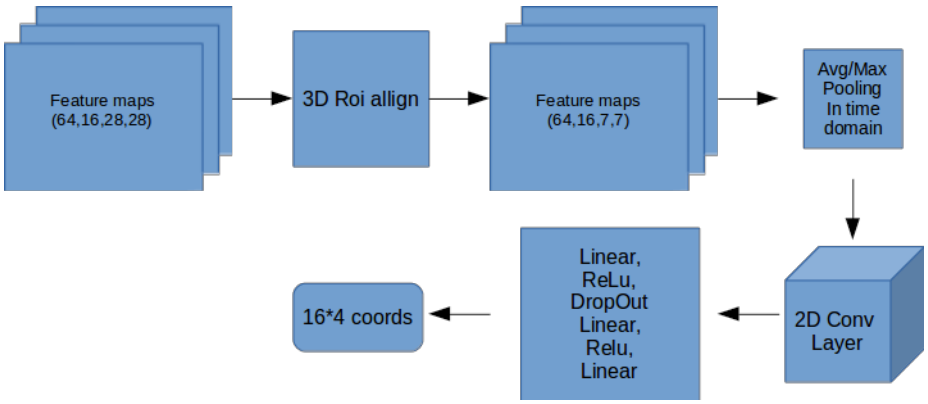


Figure 3.8: Structure of Regressor

On top of that, we tried to determine which feature map is the most suitable for getting best-scorig recall performance. This feature map will be given as input to Roi Algin operation. At Table 3.4, we can see the recall performance for different feature maps and different pooling methods.

As we noticed from the above results, again, our system has difficulty in translating cuboids into 2D sequence of ROIs. So, that makes us rethink the way we designed our TPN.

3.5 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

Dataset	Pooling	F. Map	Recall	Recall SR	Recall SRF)
JHMDB	mean	64	0.6828	0.5112	0.7610
		128	0.8694	0.7799	0.6756
		256	0.8396	0.7687	0.7029
	max	64	0.8582	0.7985	0.5914
		128	0.8358	0.7724	0.8118
		256	0.8657	0.8022	0.7996
UCF	mean	64	0.5055	0.4286	0.5889
		128	0.5335	0.4894	0.5893
		256	0.5304	0.4990	0.6012
	max	64	0.5186	0.4990	0.5708
		128	0.5260	0.4693	0.5513
		256	0.5176	0.4878	0.6399

Table 3.4

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 3.9 we can an example of this type of anchor.

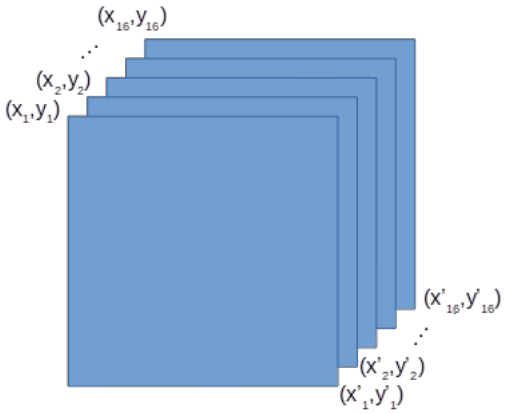


Figure 3.9: An example of the anchor $(x_1, y_1, x'_1, y'_1, x_2, y_2, ...)$

The main advantage of this approach is that we don't need to translate the 3D anchors into 2D boxes, which caused many problems at the previous approach. 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 3.10. As we can see, we added scoring and regression layers for each duration. So, TPN follows the next step in order to propose action tubes:

1. At first, we get the feature map, extracted by ResNet, as input to a 3D Convolutional Layer with kernel size = 1, stride = 1 and no padding.

2. From Convolutional Layer, we get as output an activation map with dimensions (256,16,7,7). For reducing time dimension, we use 4 pooling layer, one for each sample duration with kernel sizes *(16,1,1), (12,1,1,), (8,1,1) and (4,1,1)* and stride = 1, for sample durations 16, 12, 8 and 4 respectively. So, we get activation maps with dimensions *(256,1,7,7), (256,5,7,7), (256,9,7,7)*

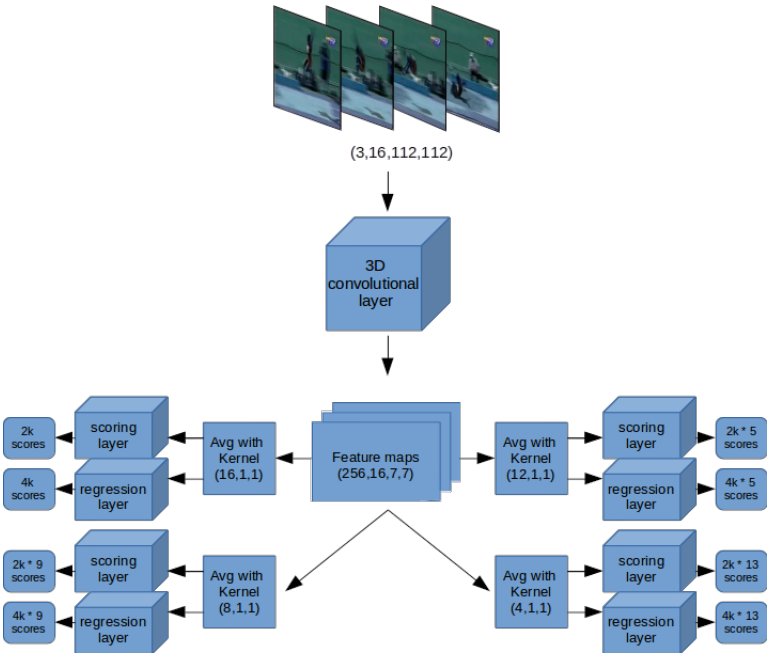


Figure 3.10: The structure of TPN according to new approach

and (256,13,7,7), in which second dimension is the number of possible time variations. For example, in $(256, 5, 7, 7)$ feature map, which is related with anchors with duration 12 frames, we can have 5 possible cases, from frame 0 to frame 11, frame 1 to frame 12 etc.

3. Again, like in previous approach, for each pixel of the activate map we correspond **n = k = 15** anchors (5 scale of 1, 2, 4, 8, 16, 3 aspect rations of 1:1, 1:2, 2:1). Of course, we have 4 different activate maps, with 1, 5, 9 and 13 different cases and a 7×7 shape in each filter. So, in total we have $28 \cdot 15 \cdot 49 = 20580$ different anchors. Respectively, we have 20580 different regression targets.

3.5.1 Training

Training procedure stays almost the same like previous approach's. So, again, we randomly choose a video segment and we consider anchors with overlap bigger than 0.8 with any groundtruth tube, alongside with background anchors whose overlap is bigger that 0.1 and smaller than 0.3.

Dataset	Pooling	Recall(0.5)	Recall(0.4)	Recall(0.3)
JHMDB	mean	0.6866	0.7687	0.8582
	max	0.8134	0.8694	0.9216
UCF	avg	0.5435	0.6326	0.7075
	max	0.6418	0.7255	0.7898

Table 3.5: Recall results

As Table 3.5, it is obvious that we get better recall performances compared to previous' approach. Additionally, we can see that 3Dmax pooling performs better than 3D avg pooling. The difference between max pooling and avg pooling is about 10%, which is big enough to make us choose max pooling operation before getting anchors' scores and regression targets.

3.5.2 Adding regressor

Even though, our TPN outputs frame-level boxes, we need to improve these predictions in order to overlap with the groundtruth boxes as well as possible. So, in full correspondance with the previous approach, we added an regressor for trying to get better recall results.

3D Roi align In this approach, we know already the 2D coordinates. So, we can use the method proposed in Girdhar et al. 2017. They extend RoiAlign operator by splitting the tube into T 2D boxes. Then, they use classic RoiAlign to extract a region from each of the temporal slices in the feature map. After that, they concatenate the region in time domain so they get a $T \times R \times R$ feature map, where R is the output resolution of RoiAlign, which is 7 in our situation.

As a first approach, we use a 3D convolutional layer followed by 2 linear layer. Our regressors follows the following steps:

1. At first, use 3D RoiAlign in order to extract the feature maps of the proposed action tubes. We normalize them, and give them as input to the 3D convolutional layer.

2. The output of the 3D Convolutional Layer is fed into 2 Linear layers with ReLu faction between them and finally we get $sampleduration \times 4$ regression targets. We keep only the proposed targets, that there is a corresponding 2D box.

We train our regressor using the same loss function as previous approach's formula which is:

$$L = \sum_i L_{cls}(p_i, p_i^*) + \sum_i L_{cls}(p_{fixed,i}, p_{fixed,i}^*) + \sum_i p_i^* L_{reg}(t_i, t_i^*) + \sum_i p_{fixed,i}^* L_{reg}(t_{fixed,i}, t_{fixed,i}^*) + \sum_i q_i^* L_{reg}(c_i, c_i^*) +$$

We want again to find the best matching feature maps, so we train our regressor for feature maps $(64, 8, 7, 7)$ and $(128, 8, 7, 7)$. We didn't experiment using $(256, 8, 7, 7)$ feature map because we got OutOfMemory error during training, despite several modifications we did in the implementation code.

Dataset	Feat. Map	Recall(0.5)	Recall(0.4)	Recall(0.3)
JHMDB	64	0.7985	0.903	0.9552
	128	0.7836	0.8881	0.944
UCF	64	0.5794	0.7206	0.8134
	128	0.5622	0.7204	0.799

Table 3.6

According to Table 3.6, we got the best results when we use $(64, 16, 7, 7)$ feature map. This is the expected result, because these feature maps are closer to the actual pixels of the actor, than $(128, 16, 7, 7)$ feature maps, in which because of $3 \times 3 \times 3$ kernels, which combine spation-temporal information from neighbour pixels. However, as we can see, we got worst recall performance than when we didn't use any regressor if we compare results from Tables 3.5 and 3.6.

3.5.3 From 3D to 2D

Following the steps we used before, we design an architecture that uses instead of a 3D Convolutional Layer, a 2D. Unlike we did before, in this case, we don't use any pooling operation before feeding the first 2D Convolutional Layer. On the contrary, we manipulate our feature maps like not being spatio-temporal but, only spatial. So, our steps are:

1. At first, we use, again ,3D RoiAlign in order to extract the feature maps of the proposed action tubes and normalize them. Let us consider a feature map extracted from ResNet which has dimensions $(64, sampleduration, 7, 7)$ and after applying RoiAlign and normalization, we get a $(k, 64, sampleduration, 7, 7)$ feature map, where *k* is the number of proposed action tubes for this video segment.

2. We slice the proposed action tubes into T 2D boxes, so the dimensions of the Tensor, which contains the coordinates of action tubes, from $(k, 4{\cdot}sampleduration)$ become $(k, sampleduration, 4)$. We reshape the Tensor into $(k \cdot sampleduration, 4)$, in which, first k coordinates refer to the first frame, the second k coordinates refer to the second frame and so on.

3. Respectively, we reshape extracted feature maps from $(k, 64, sampleduration, 7, 7)$ to $(k \cdot sampleduration, 64, 7, 7)$. So, now we deal with 2D feature maps, for which as we said before, we consider that contain only spatial information. So, we use 3 Linear Layers in order to get 4 regression targets. We keep only those we have a corresponding bounding box.

Again, we experiment using 64, 128 and 256 feature maps (in this case, there is no memory problem). The results of our experiments are shown at Table 3.7.

Dataset	Feat. Map	Recall(0.5)	Recall(0.4)	Recall(0.3)
JHMDB	64	0.8358	0.9216	0.9739
	128	0.8172	0.9142	0.9627
	256	0.7724	0.8731	0.9328
UCF	64	0.6368	0.7346	0.7737
	128	0.6363	0.7133	0.7822
	256	0.6363	0.7295	0.7822

Table 3.7

As we can see, we get improve recall performance up 3% for JHMDB dataset and about the same perfomance for UCF dataset. Again, we get best performance if we choose $(64, 16, 7, 7)$ feature maps.

3.5.4 Changing sample duration

After trying all the previous versions, we noticed that we get about the same recall performances with some small improvements. So, we thought that we could try to reduce the sample duration. This idea is based on the fact that reducing sample duration, means that anchor dimensions will reduce, so the number of candidate anchors. That's because, now we have smaller number of cases, so smaller number of parameters alongside with small number of dimensions for regression targets. We train our TPN for sample duration = 8 frames 4 frames. We use, of course, TPN's second architecture, because as shown before, we get better recall performance.

Without Regressor

At first, we train TPN, again without regressor. We do so, in order to compare recall performance for all sample durations, without using any regressor. The results are shown in Table 3.8. For all cases, we use max pooling before scoring and regression layers, and we didn't experiment at all with avg pooling. Of course, for sample duration = 16, we used the calculated one in Table 3.5.

According to Table 3.8, we notice that we get best performance for sample duration = 8 for both datasets. For dataset JHMDB sample duration equal with 8 is gets far better results from the others aprroaches, followed by approach with sample duration = 4. For UCF dataset, although sample duration equal with 8 gives us best performances sample duration equal with 4 gives us about the same. The difference between those 2 duration is less that 1%.

Dataset	Sample dur	Recall(0.5)	Recall(0.4)	Recall(0.3)
JHMDB	16	0.8134	0.8694	0.9216
	8	0.9515	0.9888	1.0000
	4	0.8843	0.9627	0.9888
UCF	16	0.6418	0.7255	0.7898
	8	0.7942	0.8877	0.9324
	4	0.7879	0.8924	0.9462

Table 3.8

With Regressor

Following the idea of reducing sample duration for getting better recall performance, we trained TPN with a regressor. We trained for both approaches, which means both 3D and 2D Convolutional Layer approaches were trained. Recall performances are presented at Table 3.9.

Dataset	Sample dur	Type	Recall(0.5)	Recall(0.4)	Recall(0.3)
UCF	8	2D	0.8078	0.8870	0.9419
		3D	0.8193	0.8930	0.9487
	4	2D	0.7785	0.8914	0.9457
		3D	0.7449	0.8605	0.9362
JHDMBD	8	2D	0.9366	0.9851	0.9925
		3D	0.8918	0.9776	0.9963
	4	2D	0.9552	0.9963	1.0000
		3D	0.9142	0.9701	0.9888

Table 3.9

According to 3.9, it is clear that using a 2D Convolutional Layer as presented above results in better recall performance that using a 3D. Futhermore, we notice that the addition of a regressor causes both improvements and deteriorations in recall performances. For dataset UCF, approach with sample duration = 8 improves by about 1-2% recall performance but for sample duration = 4 it reduce it by 1-3%. On the other hand, for dataset JHMDB, now, sample duration = 4 gets better results by adding a regressor and sample duration = 8 gets worse. So, after considering both results from Tables 3.8 and 3.9, we think that the best approach is using sample duration equal with 8, with the adddition of a regressor, which uses a 2D Convolutional Layer. We know that this appoarch gets worse performance at JHMDB but it gives us the best results in UCF. But, since JHMDB's results are high enough, we are most interested in improving UCF's results. That's the reason, we will use the aformentioned approach in the rest chapters.

Chapter 4

Connecting Tubes

4.1 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.

4.2 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 4.1.

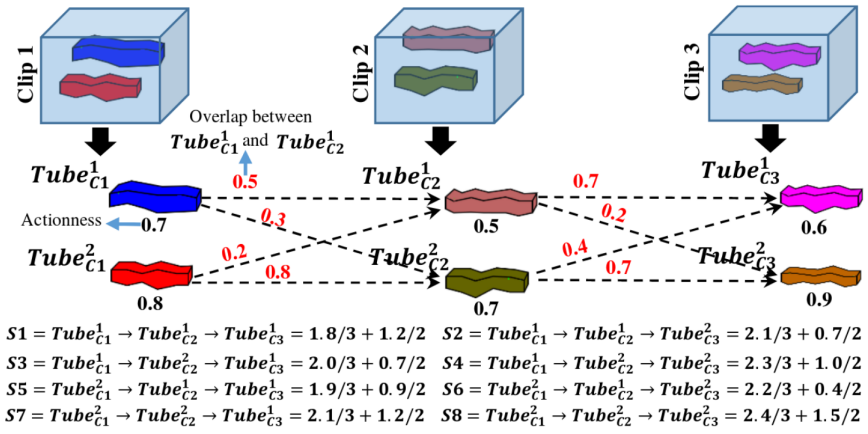


Figure 4.1: An example of calculating connection score for 3 random TOIs

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).

4.2.1 JHMDB Dataset

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 4.1 (in bold are the frames with which we calculate the overlap score).

combination	overlap thresh 0.5	0.4	0.3
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 4.1: 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 4.2.

combination	overlap thresh 0.5	0.4	0.3

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 4.2: 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 4.3

combination	overlap thresh		
	0.5	0.4	0.3
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 4.3: 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 4.4 shows recall performance for sample duration = 8 and video step = 4 and Table 4.5 for video steps = 6, 7 and 8.

combination	overlap thresh		
	0.5	0.4	0.3
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 4.4: Recall results for step = 4

combination	overlap thresh		
	0.5	0.4	0.3
0,1,2,3,4,5**{6,7}** **{6,7}**,8,9,10,11,12,13	0.3134	0.7015	0.8619
0,1,2,3,4,5,**{6,}**7 **{6,}**7,8,9,10,11,12,13	0.3209	0.6679	0.847
0,1,2,3,4,5,6,**{7}** 6,**{7}**8,9,10,11,12,13	0.3172	0.6567	0.8507
0,1,2,3,4,5,6**{7}** **{7,}**8,9,10,11,12,13,14	0.5597	0.7687	0.903
0,1,2,3,4,5,6**{7}** **{8}**9,10,11,12,13,14,15	0.653	0.8396	0.9179

Table 4.5: Recall results for steps = 6, 7 and 8

According to Tables 4.4 and 4.5, it is clearly shown that, we achieve best results, for $step = sampleduration$ and overlaping scores is calculated between the last box of the current tubes and the first box of next tubes.

4.2.2 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.

The first experiments we performed were related with the number of the final tubes, our network proposes alongside with TPN's proposed tubes' number. We experiment for cases, in which TPN proposes 30, 100 and 150, our final network proposes 500, 2000 and 4000 for sample durations equal with 8 and 16 frames. For sample duration equal with 8 we return 100 proposed action tubes because, when we tried to return 150 proposed ToIs, we got OutOfMemory error. Table 4.6 show the spatio-temporal recall and MABO performance of those approaches. Futhermore, Table 4.7 show their tempolar recall and MABO performance. We are interested in temporal performance, because UCF is consisted of untrimmed videos, unlike JHMDB which has only trimmed videos. So, we want to know how well our network is able to propose action tubes that overlap temporally with the groundtruth action tubes over a "big" threshold. For temporal localization, we don't use 0.5, 0.4 and 0.3 overlapping threshold, but instead, we use 0.9, 0.8 and 0.7, because it is very important our network to be able to propose

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 4.7: Temporal Recall results for UCF dataset

tubes that contain an action. In order to calculate the temporal overlap, we use IoU for 1D as described before.

combination	TPN tubes	Final tubes	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
		4000	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 4.6: Recall results for UCF dataset

According to Table 4.6, we achieve better recall and MABO performance when we set sample duration equal with 16. In all cases, recall performace of simulations with sample duration equal with 16 outweight the corresponding with 8, with the difference varying from 2% to 8%. In addition, we get best recall and MABO performance when our system proposes 4000 tubes. As we can see, the ratio of good proposals increases about 5%-7% when we change number of proposed tubes from 500 to 2000. This ratio increases more when we double returned action tubes, from 2000 to 4000. However, this increase is only about 1%-2%, which make us rethink if this increase is worth to be performed. That's because, this modification increases memory usage, because of 4000 proposed action tubes, instead of 2000. Finally, Table 4.6 shows that, for sample duration = 8, changing the number of ToIs produced by TPN, slightly helps our network to achieve better results. This contribution is measured about 1%-2%. On the contrary, when we set sample duration equal with 16, it slightly reduces network's performance. Taking all the aforementioned results into account, we think that the most suitable

choices for connection approaches are, for sample duration equal with 8, the one in which TPN returns 100 ToIs and our network proposes 4000 action tubes, and for sample duration equal with 16, the one in which, TPN returns 30 ToIs and the network 4000 action tubes.

Addionally, Table 4.7 shows some interesting facts, too. At first, it confirms that increasing the number of proposed action tubes, from 500 to 4000, increases recall and MABO performance. Also, we get better result when network has 16 frames as sample duration, too. However, unlike Table 4.6, Table 4.7 shows that when we increase TPN's number of proposed ToIs, it increases performances for both sample durations. For sample duration equal with 8, this increases results in Improving recall performances by 2% and MABO performance by 1% like spatio-temporal recall and MABO. For sample duration equal with 16, recall performance is increasing by about 8% and MABO by 1%-2%.

Taking both tables into cosideration, we think that the best approach is TPN returning 30 proposed ToIs, network returning 4000 proposed action tubes and sample duration equals with 16. We didn't choose TPN returning 150 proposed ToIs because, based on MABO performances, they different only by 1%, difference which is insignificant.

Adding NMS algorithm

Previous section describe the performances of network's proposals for variations in the returnig number of TPN's ToIs, number of returned proposed aciton tubes and sample duration. For each situation, we choose the k-best scoring action tubes, without taking into account any relation between these aciton tubes, like their Spatio-temporal overlap. So, like TPN's approach, we thought that we should apply nms algorithm before choosing k-best scoring tubes, in order to further improve spatio-temporal and temporal, recall and MABO performance. We experiment using again two sample durations, 16 and 8 frames per video segment, number or TPN's returning tubes equal with 30 and number of final picked action tubes equal with 4000. NMS algorithm uses a threshold in order to choose if 2 action tubes overlap enough. We experiment setting this threshold equal with 0.7 and 0.8 and results are shown at Table 4.8 for Spatio-temporal perfomance and at Table 4.9 for temporal performace.

combination	NMS thresh	PreNMS tubes	overlap thresh 0.9	0.8	0.7	MABO
0,1,...,6,**{7,}** **{8,}**9,...,15	0.7	20000	0.346	0.5202	0.657	0.3824685269
	0.8		0.3643	0.5392	0.6578	0.3904727407
	0.9		0.397	0.5574	0.6677	0.4031543642
0,1,...,14,**{15,}** **{16,}**17,...,23	0.7	20000	0.3939	0.5559	0.6882	0.404689056
	0.8		0.4259	0.5764	0.6981	0.419487652
	0.9		0.4494	0.5856	0.7019	0.4302611039

Table 4.8: Spatio-temporal Recall results for UCF dataset

combination	NMS thresh	PreNMS tubes	overlap thresh 0.9	0.8	0.7	MABO
0,1,...,6,**{7,}** **{8,}**9,...,15	0.7	20000	0.6281	0.8251	0.9027	0.8885141223
	0.8		0.7369	0.8616	0.9148	0.9106069806
	0.9		0.7787	0.8753	0.9209	0.9212593589
0,1,...,14,**{15,}** **{16,}**17,...,23	0.7	20000	0.7452	0.8920	0.9361	0.920331595
	0.8		0.8160	0.9278	0.9506	0.93612757
	0.9		0.854	0.9346	0.9529	0.9434986107

Table 4.9: Temporal Recall results for UCF dataset

Comparing Table 4.8 with Table 4.6, we notice that NMS algorithm impoves recall and MABO performance when NMS threshold is equal with 0.9. When we set it equal with 0.7 or 0.8, we get worse results. This happens probably because, **Pending...**

Stop updating threshold

In previous approaches, scoring threshold was updated each time our algorithm gathered a significant number of "active" tubes in order not to add action tubes with score below this score. However, after serious cosideration we came with the conclusion that some times, the updated threshold leads to not detecting action tubes that start after some frames. That's because, until then, linking threshold may be too big that won't let new action tubes to be created. So, we came with the modification of not updating linking threshold but just filtering proposed tubes, by keeping k-best scoring each time their number is bigger that the a specific number. The rest algorithm remains the same. Tables 4.10 and 4.11 show spatio-temporal and temporal, recall and MABO performace respectively. We experiment for cases in which either we don't use NMS algorithm at all, either we set overlap threshold equal with 0.7 and 0.9 as shown below.

combination	NMS thresh	PreNMS tubes	overlap thresh 0.9	0.8	0.7	MABO
0,1,...,6,**{7,}** **{8,}**9,...,15	-		0.3779	0.5316	0.6471	0.393082961
	0.7	20000	0.3483	0.5194	0.6471	0.3783524086
	0.9		0.416	0.5605	0.6722	0.4074053106
0,1,...,14,**{15,}** **{16,}**17,...,23	-		0.438	0.5635	0.6829	0.4231788
	0.7	20000	0.4525	0.5848	0.7034	0.429747438
	0.9		0.3802	0.5133	0.6068	0.3862278851848662

Table 4.10: Spatio-temporal Recall results for UCF dataset

combination	NMS thresh	PreNMS tubes	overlap thresh 0.9	0.8	0.7	MABO
0,1,...,6,**{7,}** **{8,}**9,...,15	-	-	0.7087	0.8281	0.8913	0.899210587
	0.7	20000	0.6586	0.854	0.9278	0.903373468
	0.9		0.8137	0.8973	0.9361	0.9333068498
0,1,...,14,**{15,}** **{16,}**17,...,23	-		0.8327	0.9156	0.9399	0.940143272
	0.7	20000	0.8646	0.9369	0.9567	0.946701832
	0.9		0.6183	0.7696	0.8388	0.8628507037919737

Table 4.11: Temporal Recall results for UCF dataset

Pending... commentary

Soft-nms instead of nms

Pending... Introduction After widely experiment using NMS algorithm, we thought that we should try to use Soft-NMS algorithm, introduced by Bodla et al. 2017 and described in chapter 2. We implement our own soft-nms algorithm modifying it in order to calculate spatiotemporal overlapping

scores, and not just spatial, like the one implemented by Bodla et al. 2017. As mentioned before, instead of removing action tubes, Soft-NMS algorithm just reduces their score for those which overlap over a predefined threshold, We experiment for sample duration equal with 8 and thresholds equal with 0.7 and 0.9, because, our implementation ran out of memory for sample duration equal with 16. Recall and MABO perfomance are presented at Tables 4.12 and 4.13

combination	NMS thresh	PreNMS tubes	overlap thresh 0.9	0.8	0.7	MABO
0,1,...,6,**{7,}** **{8,}**9,...,15	0.7	20000	0.3916	0.5384	0.6464	0.3964639
	0.9		0.4023	0.5430	0.6502	0.398845313

Table 4.12: Spatio-temporal Recall results for UCF dataset using Soft-NMS

combination	NMS thresh	PreNMS tubes	overlap thresh 0.9	0.8	0.7	MABO
0,1,...,6,**{7,}** **{8,}**9,...,15	0.7	20000	0.7521	0.8586	0.9110	0.915746097
	0.9		0.7741	0.8768	0.9255	0.922677864

Table 4.13: Temporal Recall results for UCF dataset using SoftNMS

Pending... commentary

4.3 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	0.5	0.4	0.3
8	6	0.3284	0.5	0.6082
8	7	0.209	0.459	0.6119
8	8	0.3060	0.5672	0.6866
16	8	0.194	0.4366	0.7164
16	12	0.3358	0.5336	0.7537
16	16	0.2649	0.4664	0.709

Table 4.14: Recall results for second approach with step = 8, 16 and their corresponding steps

According to 4.14, we get best performance when we set sample duration equal with 16 and overlap step equal with 12. Comparing this performance with first approach, for both sample durations equal with 8 and 16, we notice that second approach falls short comparing to the first one.

4.4 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)** . We run case for sample duration equal

with 8 and 16 frames and we modify the video step each time. For sample duration = 8, we return only 15 ToIs and for sample duratio = 16, we return 30 because, if we return more, we get "out of memory error". In the follown table, we can see the recall results.

combination		**overlap thresh**		
sample dur	step	0.5	0.4	0.3
8	6	0.7873	0.8657	0.9366
8	7	0.7836	0.8731	0.9366
8	8	0.7910	0.8806	0.9515
16	8	0.7873	0.8843	0.9291
16	12	0.7948	0.8881	0.9403
16	16	0.7985	0.8918	0.9515

Table 4.15: Recall results for second approach with

From the above table, firstly, we corfirmed that overlap = sample duration gives us the best recall results. we notice that sample duration = 16 is slightly better that the 8. However, using sample 16 increases the memory usage even though it reduces the number of video segments. So for classification stage we will experiment using mostly sample duration = 8.

Chapter 5

Classification stage

5.1 Description

After getting all proposed tubes, it's time to do classification. As classifiers we use several approaches including a Linear Classifier, a Recursive Neural Network (RNN) Classifier, a Support Vector Machine (SVM) Classifier and a Multilayer perceptron (MLP).

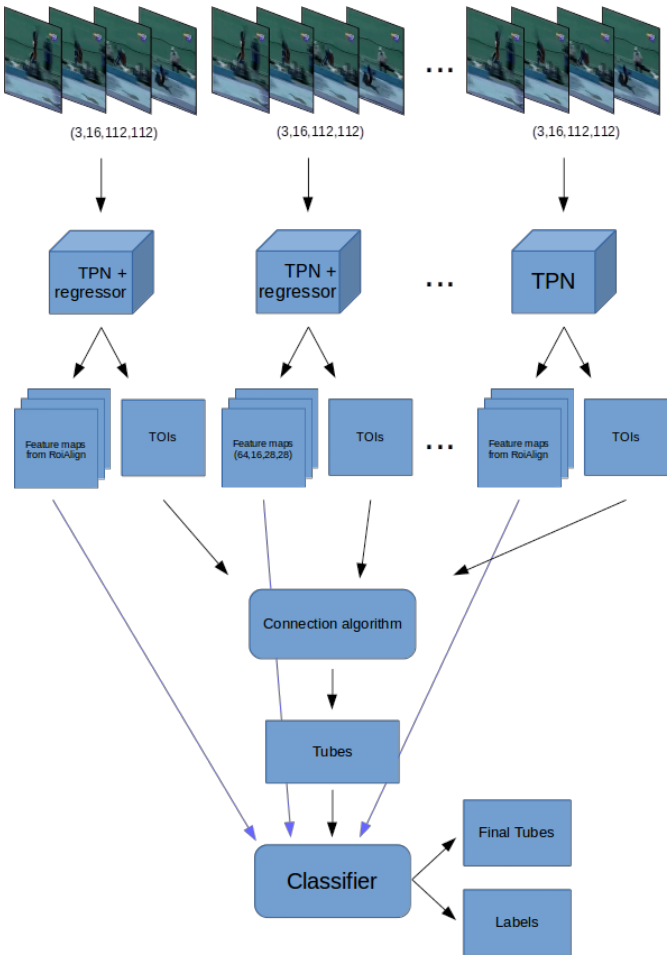


Figure 5.1: Structure of the whole network

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

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

2. Connect the proposed ToIs in order to get video tubes which may contain an action.

3. For each candidate video tube, which is a sequence of ToIs, feed it into the classifier for verification.

The general structure of the whole network is depicted in figure 5.1, in which we can see the aforementioned steps if we follow the arrows.

In first steps of classification stage we refer only to JHMDB dataset because it has smaller number of video than UCF dataset which helped us save a lot of time and resources. That's because we performed most experiments only JHMDB and after we found the optimal situation, we implemented to UCF-dataset, too.

5.2 Preparing data and first classification results

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

Linear Classifier Linear classifier is a type of classifier which is able to discriminate objects and predict their class based on the value of a *linear combination* of object's feature values, which usually are presented in a feature vector. If the input feature vector to the classifier is a real vector $\vec{x}$, then the output score is :

$$y = f(\vec{w} \cdot \vec{x}) = f\left(\sum_j w_j x_i\right)$$

RNN Recurrent neural networks, or RNNs for short, are a type of neural network that was designed to learn from sequence data, such as sequences of observations over time, or a sequence of words in a sentence. RNN takes many input vectors to process them and output other vectors. It can be roughly pictured like in the Figure 5.2 below, imagining each rectangle has a vectorial depth and other special hidden quirks in the image below. For our case, we choose **many to one** approach, because we want only one prediction, at the end of the action tube.

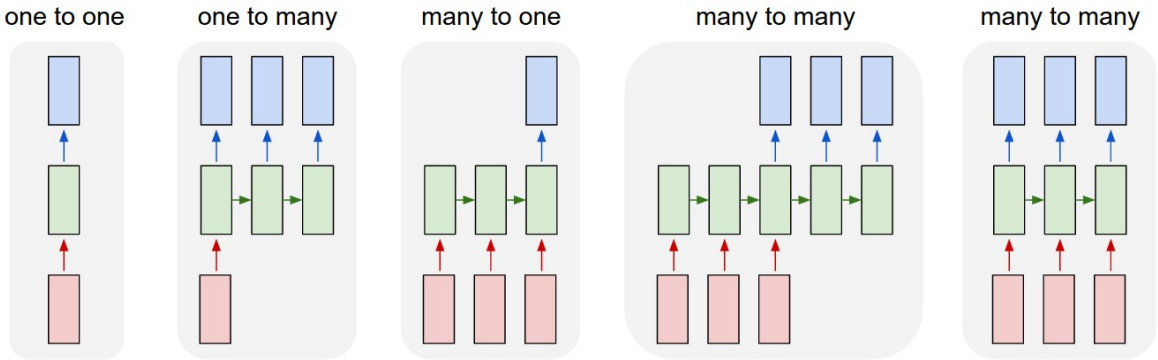


Figure 5.2: Types of RNN

Training 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 and then to feed those features to our classifier for training it in order to disciminate classes. This solution includes the following steps:

1. 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. Of course, we use a pre-trained TPN in order to get those action tubes.

2. After extracting those features, we trained both Linear and RNN classifiers. The Linear classifier needs a fixed input size, so we used a pooling function in the dimension of the 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 avg pooling as shown at Table 5.1. For the RNN classifier, we do not use any pooling function before feeding it.

In order to train our classifiers, we use Cross-Entropy Loss as training loss function.

Validation Validation stage includes using both pre-trained TPN and classifier. So, for each video, we get classification scores for proposed action tubes. Most approaches usually consider a confidence score for considering an action tube as foreground. However, we don't use any confidence score. On the contrary, because we know that JHMDB has trimmed videos with only 1 performed action, we just consider the best-scoring tube as our prediction.

Classifier	**Pooling**	**mAP**		
		0.5	0.4	0.3
Linear	mean	14.18	19.81	20.02
	max	13.67	16.46	17.02
RNN	-	11.3	14.14	14.84

Table 5.1

Table 5.1 shows first classification results, which are not very good. The only useful deduction that we can come with, using above results is that, avg pooling method outclass max pooling. So, for all the rest classifications using Linear classifier, we use avg pooling before classification stage.

5.3 Support Vector Machine (SVM)

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.

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 5.2 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			0.5	0.4	0.3
(k,64,8,7,7)	(1,64,8,7,7)	mean	1	3.16	4.2	4.4
			2	2.29	2.68	2.86
			3	1.63	3.16	4
			4	2.42	4.83	5.46
			5	0.89	1.12	1.21
(k,64,8,7,7)	(1,64,8,7,7)	max	1	1.11	2.35	2.71
			2	2.31	2.62	2.64
			3	1.11	2.35	2.71
			4	1.41	2.76	3.84
			5	0.33	0.51	0.58
(k,256,8,7,7)	(1,256,8,7,7)	mean	1	11.41	11.73	11.73
			2	10.35	10.92	11.89
			3	8.93	9.64	9.94
			4	12.1	13.04	13.04
			5	5.92	6.92	7.79
(k,256,8,7,7)	(1,256,8,7,7)	max	1	22.07	24.4	25.77
			2	14.07	16.56	17.74
			3	14.22	18.94	21.6
			4	21.05	24.63	25.93
			5	11.6	13.92	15.81

Table 5.2: Our architecture's performance using 5 different policies and 2 different feature maps while pooling in tubes' dimension. With bold is the best scoring case

From the above results we notice that features map with dimension (256,8,7,7) outperform in all

cases, both for mean and max pooling and for all the policies. Also, we can see that max pooling outperforms mean pooling in all cases, too. Last but not least, we notice that policies 2, 3 and 5 give us the worst results which means that svm needs both data from other classes positives and from background tubes.

5.3.1 Modifying 3D Roi Align

As we mentioned before, we extract from each tube its activation maps using 3D Roi Align procedure and we set equal to zero the pixels outside of bounding boxes for each frame. We came with the idea that the 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 5.3

Dimensions		**Pooling**	**Type**	**mAP precision**		
before	after			0.3	0.4	0.5
(k,64,8,7,7)	(1,64,8,7,7)	mean	1	9.75	11.92	13.34
			4	5.74	6.62	7.59
(k,64,8,7,7)	(1,64,8,7,7)	max	1	6.46	10.26	10.83
			4	4.19	6.27	7.52

Table 5.3: Our architecture's performance using 2 different policies and 2 different pooling methods using modified Roi Align.

According to Table 5.3, modified Roi Align doesn't improve mAP performace. On the contrary, it reduces it. However, the gap between those 2 approaches is small, so we don't abandon this idea, becuase, for different approaches, modified Roi Align may outclass regular Roi Align.

5.3.2 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.

We experiment using a SVM classifier for training policies 1 and 4 and using both regular and modifier Roi Align. The perfomance results are presented at Table 5.4.

Dimensions		**Pooling**	**Type**	**mAP precision**		
before	after			0.5	0.4	0.3

k,256,8,7,7	2,256,8,7,7	RoiAlign	1	25.07	26.91	29.11
			4	23.27	25.96	28.25
		mod RoiAlign	1	7.01	9.69	10.52
			4	5.5	7.25	8.99

Table 5.4: mAP results using temporal pooling for both RoiAlign approaches

Comparing Tables 5.3 and 5.4, we clearly notice that we get better results when using temporal pooling. Also, the difference between regular Roi Align and modified Roi Align become much bigger than previously, so this makes us abandon the idea of modified Roi Align. So, the rest section, we only experiment using regular Roi Align.

5.4 Increasing sample duration to 16 frames

Next, we though that a good idea would be to increase the sample duration from 8 frames to 16 frames. We experiment both using and not using temporal pooling, again for policies 1 and 4. Results are included at table 5.5.

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

Table 5.5: mAP results for policies 1,4 for sample duration = 16

As shown at Table 5.5, we get better performance when we don't use temporal pooling, fact that is unexpected. However, the difference between those performaces is about 2%. Probably, this is caused by the fact that, in the temporal pooling approach, SVM classifier has to train too many parameters when it uses temporal pooling, on the contrary with the approach not using temporal pooling, in which SVM has to train half the number of parameters. Futhermore, comparing above results with results shown at Table 5.3, we can see that we get about the same results for both approaches. So, we choose to keep using approach with sample duration equal with 8. That's because, we don't have to use too much memory during training and validation.

5.5 Adding more groundtruth tubes

Pending more comments...

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 from 1 to 2, 8. Table 5.8

F. map	**FG tubes**	**Total tubes**	**mAP**		
			0.5	0.4	0.3

(k,256,8,7,7)	1	3	11.3	14.14	14.84
	2	3	1.96	5.07	7.27
		4	3	5.03	5.77
		6	1.34	3.89	4.49
		8	0.77	1.51	2.72
	4	6	13.23	21.74	25.4
		8	20.73	28.25	29.50
		12	16.55	24.35	25.22
		16	20.11	25.50	27.62
	8	12	13.82	19.93	22.80
		16	15.47	23.08	24.19
		24	15.88	23.44	24.48
		32	12.66	23.50	25.61

Table 5.6: RNN results

F. map	**FG tubes**	**Total tubes**	**mAP**		
			0.5	0.4	0.3
(k,256,8,7,7)	1	3	14.18	19.81	20.02
	2	3	12.68	13.38	15.14
		4	11.5	14.95	16.22
		6	10.74	13.36	15.18
		8	8.00	9.83	11.17
	4	6	15	17.55	19.39
		8	17.04	20.12	22.07
		12	17.57	19.9	21.88
		16	14.24	17.24	17.95
	8	12	17.91	22.51	24.62
		16	16.76	20.34	22.72
		24	17.61	19.12	24.48
		32	14.45	18.07	19.14

Table 5.7: Linear results

F. map	**FG tubes**	**Total tubes**	**Policy**	**mAP**		
				0.5	0.4	0.3
(2,256,8,7,7)	1	3	1	25.07	26.91	29.11
			4	23.27	25.96	28.25
	3	8	1	24.38	25.97	26.4
			4	Pending...		
	4	8	1	Pending...		
			4	Pending...		
		12	1	Pending...		
			4	Pending...		
		16	1	Pending...		
			4	Pending...		

	8	16	1	Pending...
			4	Pending...
		24	1	Pending...
			4	Pending...
		32	1	Pending...
			4	Pending...

Table 5.8: SVM results

5.5.1 Increasing again sample duration (only for RNN and Linear)

Table 5.5 showed that SVM classifier gets about the same performance for both sample durations 8 and 16 frames. Triggered by this fact, we trained RNN and Linear classifiers for sample duration equal with 16 frames. Table 5.9 shows RNN's mAP performance and Table 5.10 Linear's mAP performance.

FG tubes	Total tubes	mAP		
		0.5	0.4	0.3
4	8	Pending...		
	12	Pending...		
	16	Pending...		
8	16	Pending...		
	24	12.85	18.35	20.00
	32	9.38	14.33	16

Table 5.9: RNN results for sample duration equal with 16

FG tubes	Total tubes	mAP		
		0.5	0.4	0.3
4	8	15.11	19.78	21.14
	12	11.39	15.74	18.15
	16	13.62	16.11	18.15
8	16	12.98	17.52	19.10
	24	12.92	17.64	19.95
	32	11.51	13.98	14.82

Table 5.10: Linear results for sample duration equal with 16

Pending... commentary

5.6 MultiLayer Perceptron (MLP)

In previous sections we used classic classifiers like Linear, RNN and SVM. Last but not least approach, another widely category of classifiers is Multilayer Perceptorn (MLP) classifiers. MLP is a class of feedforward Neural Network, so its function is described in chapter 2. So, we design a MLP which is shown in Figure 5.3 for sample duration equal with 8, and is described below:

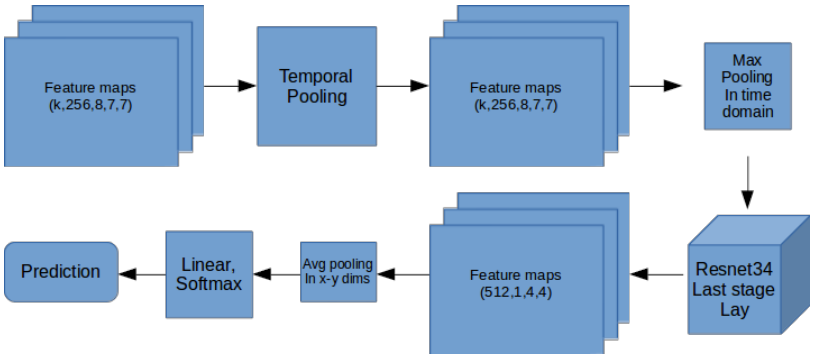


Figure 5.3: Structure of the MLP classifier

- At first, after 3D Roi align and for sample duration = 8, we get an activation map of $(k, 256, 8, 7, 7)$ where k is the number of linked ToIs. Inspired by previous sections, we perform temporal pooling followed by a max pooling operation in sample duration's dimension. So, we now have an activation maps with dimensions equal with $(2, 256, 7, 7)$, which we reshape it into $(256, 2, 7, 7)$. we extracted layers from the last stage of ResNet34. This stages includes 3 Residual Layers with stride equal with 2 in all 3 dimensions and output number of filters equal with 512.
- After Residual Layers, we perfom temporal pooling for x-y dimensions. So we get as output activation maps with dimension size equal with (512,). Finally, we feed these feature maps to a linear layer in order to get class condifence score, after applying soft-max function.

5.6.1 Regular training

According to figure 5.1, the trainable parts of our network is TPN and the classifier. As mentioned before, training code requires running only one video per GPU, because, videos have different duration. For previous approaches we came with the idea of pre-calculating video features and then training only the classifier. However, for this step, we normally trained our in order to get classification results. Of course, we used a pre-trained TPN, whose layers were freezed in order not to be trained. We tried to explore different ratios between the number of foreground tubes and the total number of tubes per video. First 3 simulations included fixed number of total tubes and variable ratio between the number of foreground and background tubes. We started using only foreground tubes, which means 32 out 32 tubes are foreground, then half of the proposed tubes aka 16 out of 32 and finally less than half, namely 14 out of 32. After that, we experiment using a fixed number of foreground tubes and variable number of total tubes, which are 16, 24 and 32. The performance results are presented at Table 5.11.

FG tubes	Total tubes	mAP		
		0.5	0.4	0.3
32	32	1.28	1.73	1.87
16		3.98	4.38	4.38
14		0.40	0.40	0.40
8	16	9.41	12.59	14.61
	24	12.32	15.53	18.57
	32	7.16	10.92	13.00

Table 5.11: MLP'smAP performance for regular training procedure

The results show that when first 3 approaches give us very bad results. Comparing them with the rest 3, we came with the conclusion that we need at the most 8 foreground tubes, even thought the ration between the number of foreground and background is in favor the second one. Probably, too many foreground action tubes make our architecture overfitted so unable to generalize.

FG tubes	Total tubes	mAP		
		0.5	0.4	0.3
4	8	5,89	9,54	13,61
	12	9,51	12,8	14,6
	16	Pending...		
8	16	Pending...		
	24	Pending...		
	32	Pending...		

Table 5.12: mAP results for MLP trained using extracted features

Pending.. commentary

5.7 Adding nms algorithm

Pending... Introduction

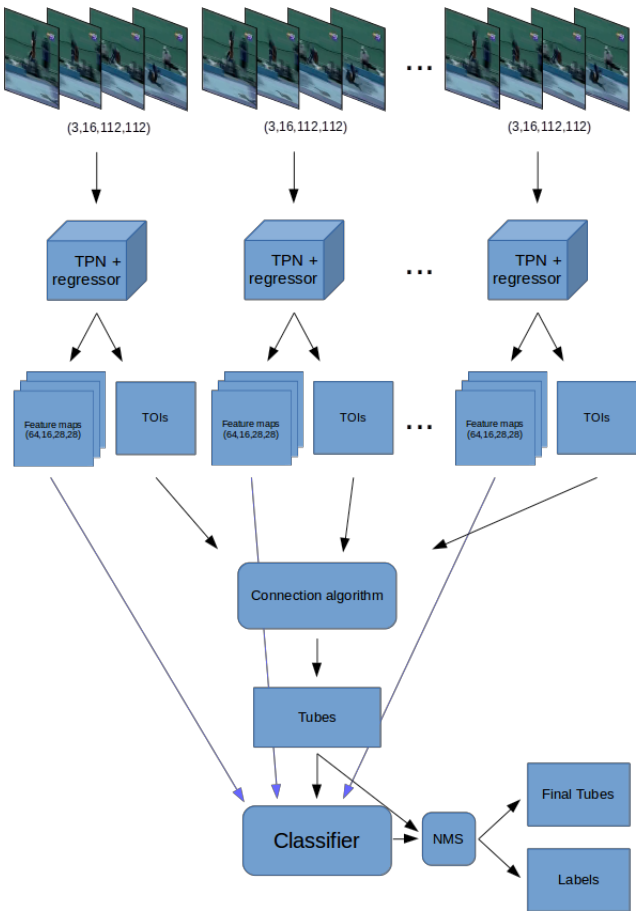


Figure 5.4: Structure of the network with NMS

FG tubes	Total tubes	mAP		
		0.5	0.4	0.3

4	8	5,89 9,54 13,61
	12	Pending...
	16	Pending...
8	16	Pending...
	24	Pending...
	32	Pending...

Table 5.13: mAP results for SVM classifier after adding NMS algorithm

FG tubes	**Total tubes**	**mAP** 0.5 0.4 0.3
4	8	Pending...
	12	Pending...
	16	Pending...
8	16	Pending...
	24	Pending...
	32	Pending...

Table 5.14: mAP results for SVM classifier after adding NMS algorithm

FG tubes	**Total tubes**	**mAP** 0.5 0.4 0.3
4	8	Pending...
	12	Pending...
	16	Pending...
8	16	Pending...
	24	Pending...
	32	Pending...

Table 5.15: mAP results for SVM classifier after adding NMS algorithm

FG tubes	**Total tubes**	**mAP** 0.5 0.4 0.3
4	8	Pending...
	12	Pending...
	16	Pending...
8	16	Pending...
	24	Pending...
	32	Pending...

Table 5.16: mAP results for SVM classifier after adding NMS algorithm

Pending... Commentary

5.8 Classifying dataset UCF

5.8.1 Using RNN, Linear and MLP classifiers

At first, we use the same approach we did for classifying JHMDB dataset. However, we don't use a SVM classifier because its training requires too much resources, which we don't have available. That's because, during training, it loads every feature map and keeps it for future training of the SVM classifier, according to hard mining negative training procedure. So, we again extract foreground and background action tubes and save them in order to use them during classifiers' training phase. The only difference with JHMDB's approach is that before saving them, we perform max pooling in sample duration's dimension because saving those feature maps requires too much memory, and by doing this, we reduce significantly.

FG tubes	Total tubes	mAP		
		0.5	0.4	0.3
4	8	Pending...		
	12	Pending...		
	16	Pending...		
8	16	Pending...		
	24	Pending...		
	32	Pending...		

Table 5.17: RNN results

FG tubes	Total tubes	mAP		
		0.5	0.4	0.3
4	8	Pending...		
	12	Pending...		
	16	Pending...		
8	16	Pending...		
	24	Pending...		
	32	Pending...		

Table 5.18: Linear results

Pending... commentary

5.8.2 Only temporal classification

As presented in chapter 5, our connection algorithm is able to get good temporal recall and MABO performance. In most cases, MABO performance got score about 92-94%. So, we came with idea of just performing temporal localization, instead of spatio-temporal localization.

In order to temporally localize action in videos, we use only the temporal information containing in the proposed action tubes, which means the first and the last frame of the action tube. After that, we will classify the proposed action tubes without performing spatiotemporal localization, but only

temporal. Although we don't use the extracted bounding boxes for classification, we take advantage of the spatial information in order to perform better temporal localization. Intuitively,that's because, in order to extract the action tubes, we consider the spatial overlap between the connected ToIs . This aformentioned approach includes the following steps:

1. Fist, we use TPN in order to propose spatio-temporal ToIs, just like we did in previous approaches. Then, we link those ToIs based on the proposed algorithm in the chapter 5, using spatiotemporal NMS algorithm with threshold equal with 0.9, for removing overlapping action tubes.

2. Previous steps is exactly the same as previous classification approaches. However, in this approach, we don't use any kind of Roi Align in order to extract action tubes' feature maps. On the contrary, for all the proposed action tubes, we find their duration, aka their first and their last frame. After that, we perform temporal nms in order to remove overlapping action tubes. The only difference between spatio-temporal and temporal nms is the overlapping criterion, which is used. For spatio-temporal nms, we use spatiotemporal IoU and respectively, for temporal we use temporal IoU as presented in chapter 2.

3. Of course, the proposed action tubes last more that 16 frames, which we set as sample duration. So, we seperate action tubes into video clips lasting 16 frames (like our sample duration). These video segments are fed, again at a 3D resNet34 (Hara, Kataoka, and Satoh 2018), but this time, we don't use it only for feature extraction but, also for classification for each video segment.

4. So, for each video clip, for each class we get a confidence score after performing softmax operation. Finally, we get average confidence score for each class, and we consider the best-scoring class as the class label of each actio tube. Of course, some action tubes may not contain any action, so we set a confidence score for Seperate foreground action tubes with background.

Training The only trainable part of this architecture is the ResNet34. We use a pre-trained TPN as presented in chapter 4. ResNet34 training procedure is based on the code given by Hara, Kataoka, and Satoh 2018. We modified it in order to be able to be trained for dataset UCF-101, only for the 24 classes, for which there are spatiotemporal notations and our TPN is trained.

Validation Based on the aforementioned steps, it is clear that the parameters that can be modified are temporal NMS' threshold and confidence threshold for deciding if an action is contained or not. All the different combinations used during validation are presented at Table 5.19.

NMS thresh	**Conf thresh**	**mAP**		
		0.5	0.4	0.3
-	-	Pending...		
0.9	0.75	Pending...		
	0.8	Pending...		
	0.9	Pending...		
0.8	0.75	Pending...		
	0.8	Pending...		
	0.9	Pending...		
0.7	0.75	Pending...		
	0.8	Pending...		
	0.9	Pending...		
0.5	0.75	Pending...		
	0.8	Pending...		
	0.9	Pending...		

Table 5.19: UCF's temporal localization mAP perfomance

Pending... commentary

Chapter 6

Conclusion - Future work

6.1 Conclusion

In this thesis we explore the problem of action recognition and localization. We design a network base on Hou, Chen, and Shah 2017 combined with some elements from Girdhar et al. 2017, Ren et al. 2015, Girshick 2015, Hu et al. 2019 and Hara, Kataoka, and Satoh 2018.

We write a pytorch implementation expanding code only from Yang et al. 2017. Futhermore, we wrote our own code using some CUDA functions designed by us (like calculating connection scores, modifying tubes etc).

We tried to design a design a Tube Proposal Network for proposing action tubes in given video segments, inspired by Faster R-CNN's RPN. We designed it using general anchors and not dataset specific anchors in order to try to generalize our approach for several datasets, on the contrary with the approach proposed by Girdhar et al. 2017, in which it uses the most frequently appearing anchors as the general anchors.

On top of that, we designed a naive connection algorithm for connecting our proposed action tubes based on the one proposed by Girdhar et al. 2017. In our approach, we use the same scoring policy, which is a combination between actioness and overlaping scores. The main difference is that we avoid to calculate all the possible combinations using an updating threshold. We, also, tried another connection algorithm inspired by Hu et al. 2019. However, our implementation wasn't very good so, we didn't explore all of its pontentials.

Finally, we explored several classifiers for the classification stage of our network, which are a RNN, a SVM and a MLP. We used an implementation taken from Fast RCNN for the SVM classifier, which included hard negatives mining trainig procedure. Futhermore, we explore some training techniques for best classification performance and 2 training approaches, the classic one and using pre-extracted features.

6.2 Future work

There is a lot of room for improvement for our network, in order to achieve state-of-the-art results. The most important are described in next paragraphs.

Improving TPN proposals We implemented 2 networks for proposing action tubes in a video segment. We managed to achieve about 63% recall score for sample duration = 16 and about 80% recall for sample duration = 8. Theses scores show that there is plenty room for improvement especially for sample duration = 16. Even though a lot of networks' architectures have been explored for regression, a good idea would be to try other networks, not necessarily inspired by object detection networks like we did. On top of that, adding a λ factor in training loss would be a good idea and exploring which is the best approach. So training loss could be defined as:

$$L = \sum_i L_{cls}(p_i, p_i^*) + \lambda_1 \sum_i p_i^* L_{reg}(t_i, t_i^*) + \lambda_2 \sum_i q_i^* L_{reg}(c_i, c_i^*) \tag{6.1}$$

Furthermore, it would be a good idea to use SSD's (Liu et al. 2015) proposal network instead of RPN, in order to compare result. Finally, we could experiment using Feature Pyramid Networks, which could be extracted in 3 dimensions as another feature extractor or some other type of 3D ResNet.

Changing Connection algorithm In this thesis, another challenge we came was connecting proposed ToIs for proposing action tubes. We implemented a very naive algorithm, which wasn't able to give us very good proposals despite the changes we tried to do. We implemented another connection algorithm which was base in a estimation on temporal progress of an action and their overlap. Although it also didn't give us very good proposals, we believe that we should explore this algorithm's pontelials. That's because it takes advantage of the progress of the action, which the previous algorithm didn't.

Explore other classification techniques For classification stage, we experiment mainly on a SVM classifier for JHMDB dataset and we didn't get involved a lot with UCF dataset. We found the best feature maps from JHDMB and we used the same for UCF. We think that we should explore UCF's feature maps even though we believe that there will be the same. It is essential to confirm our assumption. In addition, we could try other classification techniques like random forest or experiment more with RNN classifier for the UCF dataset. Finally, another classification procedure would be a good idea, like extracting first all the possible action tubes and then using other network's features for classification stage.

Bibliography

[1] Mark Everingham et al. “The Pascal Visual Object Classes (VOC) Challenge”. In: *Int. J. Comput. Vision* 88.2 (June 2010), pp. 303–338. issn: 0920-5691. doi: 10.1007/s11263-009-0275-4. url: http://dx.doi.org/10.1007/s11263-009-0275-4.

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

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

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

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

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

[7] Shuiwang Ji, Ming Yang, and Kai Yu. “3D convolutional neural networks for human action recognition.” In: *IEEE transactions on pattern analysis and machine intelligence* 35.1 (2013), pp. 221–31.

[8] Jeff Donahue et al. “Long-term Recurrent Convolutional Networks for Visual Recognition and Description”. In: *CoRR* abs/1411.4389 (2014). arXiv: 1411.4389. url: http://arxiv.org/abs/1411.4389.

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

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

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

[12] Du Tran et al. “Learning Spatiotemporal Features with 3D Convolutional Networks”. In: *2015 IEEE International Conference on Computer Vision (ICCV)* (2014), pp. 4489–4497.

[13] Ross Girshick. “Fast R-CNN”. In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. ICCV ’15. Washington, DC, USA: IEEE Computer Society, 2015, 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.

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

[15] Wei Liu et al. “SSD: Single Shot MultiBox Detector”. In: *CoRR* abs/1512.02325 (2015). arXiv: 1512.02325. url: http://arxiv.org/abs/1512.02325.

[16] Joe Yue-Hei Ng et al. “Beyond Short Snippets: Deep Networks for Video Classification”. In: *CoRR* abs/1503.08909 (2015). arXiv: 1503.08909. url: http://arxiv.org/abs/1503.08909.

[17] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. url: http://arxiv.org/abs/1506.02640.

[18] Shaoqing Ren et al. “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, 2015, pp. 91–99. url: http://dl.acm.org/citation.cfm?id=2969239.2969250.

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

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

[21] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. “Densely Connected Convolutional Networks”. In: *CoRR* abs/1608.06993 (2016). arXiv: 1608.06993. url: http://arxiv.org/abs/1608.06993.

[22] Xiaojiang Peng and Cordelia Schmid. “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, Oct. 2016, pp. 744–759. doi: 10.1007/978-3-319-46493-0_45. url: https://hal.inria.fr/hal-01349107.

[23] Suman Saha et al. “Deep Learning for Detecting Multiple Space-Time Action Tubes in Videos”. In: *CoRR* abs/1608.01529 (2016). arXiv: 1608.01529. url: http://arxiv.org/abs/1608.01529.

[24] Limin Wang et al. “Temporal Segment Networks: Towards Good Practices for Deep Action Recognition”. In: *CoRR* abs/1608.00859 (2016). arXiv: 1608.00859. url: http://arxiv.org/abs/1608.00859.

[25] Anton Winschel, Rainer Lienhart, and Christian Eggert. “Diversity in Object Proposals”. In: *CoRR* abs/1603.04308 (2016). arXiv: 1603.04308. url: http://arxiv.org/abs/1603.04308.

[26] Navaneeth Bodla et al. “Improving Object Detection With One Line of Code”. In: *CoRR* abs/1704.04503 (2017). arXiv: 1704.04503. url: http://arxiv.org/abs/1704.04503.

[27] João Carreira and Andrew Zisserman. “Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset”. In: *CoRR* abs/1705.07750 (2017). arXiv: 1705.07750. url: http://arxiv.org/abs/1705.07750.

[28] Ali Diba et al. “Temporal 3D ConvNets: New Architecture and Transfer Learning for Video Classification”. In: *CoRR* abs/1711.08200 (2017). arXiv: 1711.08200. url: http://arxiv.org/abs/1711.08200.

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

[30] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. “Learning Spatio-Temporal Features with 3D Residual Networks for Action Recognition”. In: *CoRR* abs/1708.07632 (2017). arXiv: 1708.07632. url: http://arxiv.org/abs/1708.07632.

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

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

[33] Vicky Kalogeiton et al. “Action Tubelet Detector for Spatio-Temporal Action Localization”. In: *ICCV 2017 - IEEE International Conference on Computer Vision*. Venice, Italy, Oct. 2017.

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

[35] Chih-Yao Ma et al. “TS-LSTM and Temporal-Inception: Exploiting Spatiotemporal Dynamics for Activity Recognition”. In: *CoRR* abs/1703.10667 (2017). arXiv: 1703.10667. url: http://arxiv.org/abs/1703.10667.

[36] Gurkirt Singh et al. “Online Real time Multiple Spatiotemporal Action Localisation and Prediction”. In: 2017.

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

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

[39] Alaaeldin El-Nouby and Graham W. Taylor. “Real-Time End-to-End Action Detection with Two-Stream Networks”. In: *CoRR* abs/1802.08362 (2018). arXiv: 1802.08362. url: http://arxiv.org/abs/1802.08362.

[40] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. “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)*. 2018, pp. 6546–6555.

[41] Yu Kong and Yun Fu. “Human Action Recognition and Prediction: A Survey”. In: *CoRR* abs/1806.11230 (2018). arXiv: 1806.11230. url: http://arxiv.org/abs/1806.11230.

[42] Diogo C. Luvizon, David Picard, and Hedi Tabia. “2D/3D Pose Estimation and Action Recognition using Multitask Deep Learning”. In: *CoRR* abs/1802.09232 (2018). arXiv: 1802.09232. url: http://arxiv.org/abs/1802.09232.

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