



## Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Τομέας Σημάτων, Ελέγχου και Ρομποτικής

Εργαστήριο Όρασης Υπολογιστών, Επικοινωνίας Λόγου και Επεξεργασίας Σημάτων

Αναγνώριση και εντοπισμός ανθρώπινης  
δραστηριότητας σε βίντεο

Διπλωματική εργασία

του

Ευστάθιου Ε. Γαλανάκη

Επιβλέπων: Πέτρος Μαραγκός  
Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2019





Εθνικό Μετσόβιο Πολυτεχνείο  
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών  
Τομέας Σημάτων, Ελέγχου και Ρομποτικής  
Εργαστήριο Όρασης Υπολογιστών, Επικοινωνίας Λόγου και Επεξεργασίας  
Σημάτων

## Αναγνώριση και εντοπισμός ανθρώπινης δραστηριότητας σε βίντεο

Διπλωματική εργασία

του

Ευστάθιου Ε. Γαλανάκη

Επιβλέπων: Πέτρος Μαραγκός  
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την - 2019.

(Υπογραφή)

(Υπογραφή)

(Υπογραφή)

.....  
Πέτρος Μαραγκός  
Καθηγητής  
Ε.Μ.Π

Αθήνα, Νοέμβριος 2019



(Τπογραφή)

.....  
**Ευστάθιος Ε. Γαλανάκης**

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Copyright © Ευστάθιος Ε. Γαλανάκης, 2019.  
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου



# Ευχαριστίες



# Περίληψη

Σκοπός αυτής της διπλωματικής εργασίας είναι ο σχεδιασμός ενός δικτύου αναγνώρισης και εντοπισμού των ανθρώπινων ενεργειών σε βίντεο. Το δίκτυο μας στοχεύει να προσδιορίσει χωροχρονικά μια αναγνωρισμένη ενέργεια μέσα σε ένα βίντεο παράγοντας ακολουθίες δισδιάστατων πλαισίων, ένα για κάθε χαρέ βίντεο, που περικλείει το άτομο που εκτελεί την αναγνωρισμένη ενέργεια.

Η ανίχνευση και η αναγνώριση των ενεργειών σε βίντεο είναι μια από τις μεγαλύτερες προκλήσεις στο πεδίο της Όρασης Υπολογιστών. Οι πιο πρόσφατες προσεγγίσεις περιλαμβάνουν ένα δίκτυο ανίχνευσης αντικειμένων το οποίο προτείνει δισδιάστατα κουτάκια ανά χαρέ, έναν αλγόριθμο σύνδεσης για τη δημιουργία υποψήφιων action tubes και έναν ταξινόμησή για την ταξινόμησή τους. Πάνω σ' αυτό, οι περισσότερες από αυτές τις προσεγγίσεις εξαγάγουν τις χρονικές πληροφορίες από ένα δίκτυο το οποίο εκτιμά οπτική ροή σε επίπεδο πλαισίου. Η εισαγωγή των τρισδιάστατων συνελικτικών δικτύων μας έχει βοηθήσει να μπορούμε να υπολογίσουμε τις χωροχρονικές πληροφορίες από τα βίντεο και ταυτόχρονα να εξάγουμε χωροχρονικά χαρακτηριστικά. Η προσέγγισή μας προσπαθεί να συνδυάσει τα οφέλη του να χρησιμοποιείς δίκτυα ανίχνευσης αντικειμένων και τρισδιάστατες συνελίξεις. Σχεδιάζουμε ένα δίκτυο του οποίου η δομή βασίζεται στα κλασσικά δίκτυα εντοπισμού δράσης και το ονομάζουμε ActionNet. Το πρώτο στοιχείο είναι ένα τρισδιάστατο ResNet34 το οποίο χρησιμοποιείται για τη χωροχρονική εξαγωγή χαρακτηριστικών. Επίσης, σχεδιάζουμε ένα δίκτυο για να προτείνει υποψήφιες ακολουθίες από δισδιάστατα πλαίσια με βάση τα χωροχρονικά χαρακτηριστικά, που το ονομάζουμε Tube Proposal Network. Αυτό το δίκτυο είναι μια επέκταση του Region Proposal Network παίρνοντας ως είσοδο τα εξαγόμενα χαρακτηριστικά και εξάγει k προτεινόμενες ακολουθίες από δισδιάστατα κουτιά. Εξετάζουμε 2 προσεγγίσεις για τον καθορισμό των τρισδιάστατων προκαθορισμένων κουτιών του, τα οποία χρησιμοποιεί το TPN. Επιπλέον, σχεδιάζουμε έναν αλγόριθμο σύνδεσης για τη σύνδεση και δημιουργία των προτεινόμενων action tubes. Τέλος, διερευνούμε αρκετές τεχνικές ταξινόμησης, συμπεριλαμβανομένου ενός ταξινόμησή SVM, ενός Linear, ενός RNN και ενός MLP για τα σύνολα δεδομένων JHMDB και UCF101.

## Λέξεις κλειδία

Αναγνώριση δράσης, Εντοπισμός δράσης, Action Tubes, TPN, ActionNet



# 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 spatiotemporally localize a recognized action within a video producing sequences 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 estimat spatiotemporal information from videos and simultaneously extract their spatiotemporal feature maps. Our approach tries to combine the benefits from using object detection networks and 3D Convolution.

We designed 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 spatiotemporal feature extraction. Also, we introduced a network for proposing action tubes based on spatiotemporal 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.

## Keywords

Action Localization, Action Recognition, Action Tubes, TPN, ActionNet,



# Contents

<b>Ευχαριστίες</b>	<b>7</b>
<b>Περίληψη</b>	<b>9</b>
<b>Abstract</b>	<b>11</b>
<b>Contents</b>	<b>13</b>
<b>List of Tables</b>	<b>16</b>
<b>List of Figures</b>	<b>18</b>
<b>1 Εισαγωγή</b>	<b>21</b>
1.1 Περιγραφή Προβλήματος . . . . .	21
1.1.1 Αναγνώριση ανθρώπινης δραστηριότητας . . . . .	21
1.1.2 Εντοπισμός ανθρώπινης δραστηριότητας . . . . .	21
1.2 Εφαρμογές . . . . .	22
1.3 Προκλήσεις και Datasets . . . . .	22
1.3.1 JHMDB Dataset . . . . .	23
1.3.2 UCF-101 Dataset . . . . .	24
1.4 Motivation and Contributions . . . . .	24
1.5 Thesis structure . . . . .	24
<b>2 Σχετική βιβλιογραφία</b>	<b>25</b>
2.0.1 Αναγνώριση Δραστηριότητας . . . . .	25
2.0.2 Εντοπισμός Δραστηριότητας . . . . .	28
<b>3 Tube Proposal Network</b>	<b>31</b>
3.1 Η αρχιτεκτονική του μοντέλου μας . . . . .	31
3.2 Εισαγωγή στο TPN . . . . .	32
3.3 Προετοιμασία πριν το TPN . . . . .	33
3.3.1 Η προετοιμασία των δεδομένων . . . . .	33
3.3.2 3D ResNet . . . . .	34
3.4 Τα τρισδιάστατα anchors ως 6-dim διανύσματα . . . . .	34
3.4.1 Πρώτη περιγραφή . . . . .	34
3.4.2 Training . . . . .	35
3.4.3 Validation . . . . .	36
3.4.4 Modified Intersection over Union(mIoU) . . . . .	36

3.4.5	Βελτιώνοντας το TPN score . . . . .	37
3.4.6	Προσθήκη Regressor . . . . .	39
3.5	Τα τρισδιάστατα anchors ως 4k διαινύσματα . . . . .	42
3.5.1	Training . . . . .	44
3.5.2	Προσθήκη Regressor . . . . .	45
3.5.3	Μείωση της διάρκειας του δείγματος . . . . .	46
<b>4</b>	<b>Αλγόριθμος σύνδεσης των action tubes</b>	<b>49</b>
4.1	Πρώτη προσέγγιση: συνδυασμός επικάλυψης και πιθανότητας δράσης . . . . .	49
4.1.1	JHMDB Dataset . . . . .	51
4.1.2	UCF Dataset . . . . .	53
4.2	Δεύτερη προσέγγιση . . . . .	55
4.3	Τρίτη προσέγγιση (μόνο για το JHMDB) . . . . .	57
<b>5</b>	<b>Classification stage</b>	<b>59</b>
5.1	JHMDB dataset . . . . .	60
5.1.1	Ταξινομητές Linear, SVM και RNN . . . . .	60
5.1.2	Temporal pooling . . . . .	61
5.2	Προσθήκη περισσότερων groundtruth tubes . . . . .	62
5.3	Ταξινομητής MultiLayer Perceptron (MLP) . . . . .	64
5.3.1	Κλασσικό training . . . . .	65
5.3.2	Εξαγωγή χαρακτηριστικών . . . . .	65
<b>6</b>	<b>Επίλογος - Μελλοντικές επεκτάσεις</b>	<b>67</b>
6.1	Επίλογος . . . . .	67
6.2	Μελλοντικές επεκτάσεις . . . . .	68
<b>7</b>	<b>Introduction</b>	<b>69</b>
7.1	Problem statement . . . . .	69
7.1.1	Human Action Recognition . . . . .	69
7.1.2	Human Action Localization . . . . .	69
7.2	Applications . . . . .	69
7.3	Challenges and Datasets . . . . .	70
7.3.1	JHMDB Dataset . . . . .	71
7.3.2	UCF-101 Dataset . . . . .	71
7.4	Motivation and Contributions . . . . .	71
7.5	Thesis structure . . . . .	72
<b>8</b>	<b>Background</b>	<b>75</b>
8.1	Machine Learning . . . . .	75
8.1.1	Introduction . . . . .	75
8.1.2	Neural Networks . . . . .	77
8.1.3	A single Neuron . . . . .	77
8.1.4	2D Convolutional Neural Network . . . . .	80
8.1.5	3D Convolutional Neural Network . . . . .	82
8.2	Object Detection . . . . .	83
8.2.1	Region Proposal Network . . . . .	83
8.2.2	Roi Align . . . . .	85
8.2.3	Non-maximum suppression (NMS) algorithm . . . . .	86
8.3	Losses and Metrics . . . . .	87

8.3.1	Losses . . . . .	87
8.3.2	Metrics . . . . .	88
8.4	Related work . . . . .	94
8.4.1	Action Recognition . . . . .	95
8.4.2	Action Localization . . . . .	97
<b>9</b>	<b>Tube Proposal Network</b>	<b>101</b>
9.1	Our implementation's architecture . . . . .	101
9.2	Introduction to TPN . . . . .	101
9.3	Preparation before TPN . . . . .	102
9.3.1	Preparing data . . . . .	102
9.3.2	3D ResNet . . . . .	103
9.4	3D anchors as 6-dim vector . . . . .	104
9.4.1	First Description . . . . .	104
9.4.2	Training . . . . .	104
9.4.3	Validation . . . . .	106
9.4.4	Modified Intersection over Union(mIoU) . . . . .	106
9.4.5	Improving TPN score . . . . .	107
9.4.6	Adding regressor . . . . .	108
9.4.7	Changing Regressor - from 3D to 2d . . . . .	111
9.5	3D anchors as 4k-dim vector . . . . .	111
9.5.1	Training . . . . .	113
9.5.2	Adding regressor . . . . .	114
9.5.3	From 3D to 2D . . . . .	115
9.5.4	Changing sample duration . . . . .	116
9.6	General comments . . . . .	117
<b>10</b>	<b>Connecting Tubes</b>	<b>119</b>
10.1	Introduction . . . . .	119
10.2	First approach: combine overlap and actionness . . . . .	119
10.2.1	JHMDB Dataset . . . . .	121
10.2.2	UCF dataset . . . . .	124
10.3	Second approach: use progression and progress rate . . . . .	129
10.4	Third approach : use naive algorithm - only for JHMDB . . . . .	130
10.5	General comments . . . . .	131
<b>11</b>	<b>Classification stage</b>	<b>133</b>
11.1	Introduction . . . . .	133
11.2	Preparing data and first classification results . . . . .	134
11.3	Support Vector Machine (SVM) . . . . .	136
11.3.1	Modifying 3D Roi Align . . . . .	138
11.3.2	Temporal pooling . . . . .	138
11.4	Increasing sample duration to 16 frames . . . . .	139
11.5	Adding more groundtruth tubes . . . . .	139
11.5.1	Increasing again sample duration (only for RNN and Linear) . . . . .	142
11.6	MultiLayer Perceptron (MLP) . . . . .	143
11.6.1	Regular training . . . . .	144
11.6.2	Extract features . . . . .	144
11.7	Adding nms algorithm . . . . .	145

---

11.7.1 General comments . . . . .	148
<b>12 Conclusion - Future work</b>	<b>149</b>
12.1 Conclusion . . . . .	149
12.2 Future work . . . . .	149
<b>Bibliography</b>	<b>151</b>

# List of Tables

3.1	Recall results for both datasets using IoU and mIoU metrics . . . . .	37
3.2	Recall results after adding fixed time duration anchors . . . . .	39
3.3	Recall results after converting cuboids into sequences of frames . . . . .	42
3.4	Recall performance using 3 different feature maps as Regressor's input and 2 pooling methods . . . . .	43
3.5	Recall results using 2nd approach for anchors . . . . .	45
3.6	Recall performance when using a 3D Convolutional Layer in Regressor's architecture	46
3.7	Recall performance when using a 2D Convolutional Layer instead of 3D in Regressor's model . . . . .	46
3.8	Recall results when reducing sample duration to 4 and 8 frames per video segment	47
3.9	Recall results when a regressor and sample duration equal with 4 or 8 frames per video segment . . . . .	47
4.1	Recall results for steps = 14, 15 and 16 . . . . .	52
4.2	Recall results for steps = 6, 7 and 8 . . . . .	52
4.3	Ρεςαλλ ρεσυλτς φορ ΤΦ δατασετ . . . . .	54
4.4	Temporal Recall results for UCF dataset . . . . .	54
4.5	Spatio-temporal Recall results for UCF dataset . . . . .	55
4.6	Temporal Recall results for UCF dataset . . . . .	55
4.7	Recall results for second approach with step = 8, 16 and their corresponding steps	56
4.8	Recall results for second approach with . . . . .	57
5.1	First classification results using Linear and RNN classifiers . . . . .	61
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 . . . . .	61
5.3	μΑΠ ρεσυλτς υσινγ τεμποραλ ποολινγ φορ βοτη ΡοιΑλιγν αππροασηες . . . . .	62
5.4	RNN results . . . . .	62
5.5	Linear results . . . . .	63
5.6	ΣΜ ρεσυλτς . . . . .	64
5.7	MLP's mAP performance for regular training procedure . . . . .	65
5.8	mAP results for MLP trained using extracted features . . . . .	66
8.1	Ordered by confidence predictions and their precision and recall values . . . . .	92
9.1	Recall results for both datasets using IoU and mIoU metrics . . . . .	107
9.2	Recall results after adding fixed time duration anchors . . . . .	109
9.3	Recall results after converting cuboids into sequences of frames . . . . .	111

9.4	Recall performance using 3 different feature maps as Regressor's input and 2 pooling methods . . . . .	112
9.5	Recall results using 2nd approach for anchors . . . . .	113
9.6	Recall performance when using a 3D Convolutional Layer in Regressor's architecture	115
9.7	Recall performance when using a 2D Convolutional Layer instead of 3D in Regressor's model . . . . .	116
9.8	Recall results when reducing sample duration to 4 and 8 frames per video segment	116
9.9	Recall results when a regressor and sample duration equal with 4 or 8 frames per video segment . . . . .	117
10.1	Recall results for step = 8 . . . . .	122
10.2	Recall results for step = 12 . . . . .	122
10.3	Recall results for steps = 14, 15 and 16 . . . . .	123
10.4	Recall results for step = 4 . . . . .	123
10.5	Recall results for steps = 6, 7 and 8 . . . . .	124
10.6	Recall results for UCF dataset . . . . .	125
10.7	Temporal Recall results for UCF dataset . . . . .	125
10.8	Spatiotemporal Recall results for UCF dataset . . . . .	126
10.9	Temporal Recall results for UCF dataset . . . . .	127
10.10	Spatiotemporal Recall results for UCF dataset . . . . .	127
10.11	Temporal Recall results for UCF dataset . . . . .	128
10.12	Spatiotemporal Recall results for UCF dataset using Soft-NMS . . . . .	128
10.13	Temporal Recall results for UCF dataset using SoftNMS . . . . .	128
10.14	Recall results for second approach with step = 8, 16 and their corresponding steps	130
10.15	Recall results for second approach with . . . . .	131
11.1	First classification results using Linear and RNN classifiers . . . . .	136
11.2	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 . . . . .	137
11.3	Our architecture's performance using 2 different policies and 2 different pooling methods using modified Roi Align. . . . .	138
11.4	mAP results using temporal pooling for both RoiAlign approaches . . . . .	139
11.5	mAP results for policies 1,4 for sample duration = 16 . . . . .	139
11.6	RNN results . . . . .	140
11.7	Linear results . . . . .	141
11.8	SVM results . . . . .	141
11.9	RNN results for sample duration equal with 16 . . . . .	142
11.10	Linear results for sample duration equal with 16 . . . . .	143
11.11	MLP's mAP performance for regular training procedure . . . . .	144
11.12	mAP results for MLP trained using extracted features . . . . .	145
11.13	mAP results for SVM classifier after adding NMS algorithm . . . . .	145
11.14	mAP results for RNN classifier after adding NMS algorithm . . . . .	147
11.15	mAP results for Linear classifier after adding NMS algorithm . . . . .	147
11.16	mAP results for MLP classifier after adding NMS algorithm . . . . .	147

# List of Figures

1.1 Παραδείγματα της δράσης «Ανοίγω» . . . . .	23
3.1 Network's structure . . . . .	32
3.2 At (a), (b) frame is its original size and at (c), (d) same frame after preprocessing part . . . . .	33
3.3 An example of the anchor $(x_1, y_1, t_1, x_2, y_2, t_2)$ . . . . .	34
3.4 Structure of TPN . . . . .	35
3.5 Groundtruth tube is colored with blue and groundtruth rois with color green . . . . .	35
3.6 TPN structure after adding 2 new layers, where $k = 5n$ . . . . .	38
3.7 Structure of Regressor . . . . .	41
3.8 Structure of Regressor . . . . .	42
3.9 An example of the anchor $(x_1, y_1, x'_1, y'_1, x_2, y_2, \dots)$ . . . . .	43
3.10 The structure of TPN according to new approach . . . . .	44
4.1 An example of calculating connection score for 3 random TOIs taken from Hou, Chen, and Shah 2017 . . . . .	50
5.1 Structure of the MLP classifier . . . . .	65
7.1 Examples of “Open” action . . . . .	71
8.1 Example of supervised Learning . . . . .	76
8.2 Example of unsupervised Learning . . . . .	76
8.3 Example of Reinforcement Learning . . . . .	77
8.4 An example of a single Neuron . . . . .	78
8.5 Plots of Activation functions . . . . .	79
8.6 An example of a Feedforward Neural Network . . . . .	80
8.7 Typical structure of a ConvNet . . . . .	81
8.8 Convolution with kernel of 3, stride of 2 and padding of 1 . . . . .	82
8.9 Example of Max pooling operation with a 2x2 filter and stride of 2 . . . . .	82
8.10 3D Convolution operation . . . . .	83
8.11 Region Proposal Network's structure . . . . .	84
8.12 Anchors for pixel (320,320) of an image (600,800) . . . . .	84
8.13 Roi Pooling and Roi Align examples . . . . .	85
8.14 Example of bi-linear interpolation for calculation Roi Align's final feature map . . . . .	86
8.15 Example of a situation where NMS algorithm will remove good proposals . . . . .	87
8.16 (a) and (b) show the behavior of cross-entropy loss and smooth-L1 respectively. . . . .	89
8.17 Example of IoU scoring policy . . . . .	89

8.18 Precision/Recall curve . . . . .	92
8.19 Both P-R curves. Interpolated P-R curve has red colour. . . . .	93
8.20 Recall versus MABO example . . . . .	94
9.1 Structure of the whole network . . . . .	102
9.2 At (a), (b) frame is its original size and at (c), (d) same frame after preprocessing part . . . . .	103
9.3 An example of the anchor $(x_1, y_1, t_1, x_2, y_2, t_2)$ . . . . .	104
9.4 Structure of TPN . . . . .	105
9.5 Groundtruth tube is colored with blue and groundtruth rois with color green . . . . .	105
9.6 TPN structure after adding 2 new layers, where $k = 5n$ . . . . .	108
9.7 Structure of Regressor . . . . .	110
9.8 Structure of Regressor . . . . .	112
9.9 An example of the anchor $(x_1, y_1, x'_1, y'_1, x_2, y_2, \dots)$ . . . . .	113
9.10 The structure of TPN according to new approach . . . . .	114
9.11 Visualization of Tois including groundtruth action tubes, too . . . . .	118
10.1 An example of calculating connection score for 3 random TOIs taken from Hou, Chen, and Shah 2017 . . . . .	120
10.2 Example of connected tubes . . . . .	131
11.1 Structure of the whole network . . . . .	134
11.2 Types of RNN . . . . .	135
11.3 Structure of the MLP classifier . . . . .	143
11.4 Structure of the network with NMS . . . . .	146

# Κεφάλαιο 1

## Εισαγωγή

Στις μέρες μας, η τεράστια αύξηση της υπολογιστικής ισχύος των Η/Υ μας βοηθά να αντιμετωπίσουμε πολλές δύσκολες καταστάσεις που εμφανίζονται στην καθημερινότητά μας. Πολλοί τομείς της επιστήμης κατάφεραν να αντιμετωπίσουν σημαντικά προβλήματα πριν από 20 χρόνια και σήμερα θεωρούνται ασήμαντα. Ένας επιστημονικός τομέας που επηρεάστηκε αρκετά είναι ο τομέας της Όρασης των Υπολογιστών (Computer Vision) και πιο συγκεκριμένα, αυτός που ασχολείται με το πρόβλημα της αναγνώρισης και εντοπισμού ανθρώπινης δράσης σε βίντεο.

### 1.1 Περιγραφή Προβλήματος

Η πρόκληση της αναγνώρισης και εντοπισμού ανθρώπινης δράσης σε βίντεο έχει δύο κύριους στόχους:

1. Την αυτόματη αναγνώριση και ταξινόμησή οποιασδήποτε ανθρώπινης δραστηριότητας στο βίντεο.
2. Τον αυτόματο εντοπισμό αυτής της δράσης στο βίντεο

#### 1.1.1 Αναγνώριση ανθρώπινης δραστηριότητας

Λαμβάνοντας υπόψη την αναγνώριση της ανθρώπινης δράσης, ένα βίντεο μπορεί να αποτελείται μόνο από ένα άτομο που κάνει κάτι. Ωστόσο, αυτό είναι ένα ιδανικό σενάριο. Στις περισσότερες περιπτώσεις, τα βίντεο περιέχουν πολλά άτομα, που εκτελούν πολλαπλές ενέργειες ή ενδέχεται να μην δρουν καθόλου σε ορισμένα τμήματα του βίντεο. Έτσι, ο στόχος μας δεν είναι μόνο να ταξινομήσουμε μια δράση, αλλά να αποκομίσουμε τα χρονικά όρια κάθε δράσης

#### 1.1.2 Εντοπισμός ανθρώπινης δραστηριότητας

Παράλληλα με την αναγνώριση της ανθρώπινης δράσης, ένα άλλο πρόβλημα είναι να προσδιορίσουμε τα χωρικά όρια κάθε δράσης. Συνήθως, αυτό σημαίνει να καθορίσουμε ένα δισδιάστατο πλαίσιο οριοθέτησης για κάθε καρέ βίντεο, το οποίο περιέχει τον δρώντα. Φυσικά, αυτό το κουτί οριοθέτησης κινείται μαζί με τον ηθοποιό.

## 1.2 Εφαρμογές

Το πεδίο της Αναγνώρισης και Εντοπισμού Ανθρώπινης Δράσης έχει πολλές εφαρμογές που περιλαμβάνουν ανάλυση περιεχομένου με βάση το βίντεο, αυτοματοποιημένη κατάτμηση βίντεο, συστήματα ασφάλειας και επιτήρησης, αλληλεπίδρασης ανθρώπου υπολογιστή.

Η τεράστια διαθεσιμότητα δεδομένων (ειδικά των βίντεο) δημιουργεί την ανάγκη να βρεθούν τρόποι για να επωφεληθούμε απ' αυτά. Περίπου 2,5 δισεκατομμύρια εικόνες μεταφορτώνονται στη βάση δεδομένων του Facebook κάθε μήνα, περισσότερες από 34K ώρες βίντεο στο YouTube και περίπου 5K εικόνες κάθε λεπτό. Επιπλέον, υπάρχουν περίπου 30 εκατομμύρια κάμερες παρακολούθησης στις ΗΠΑ, πράγμα που σημαίνει περίπου 700 ώρες βίντεο ανά ημέρα. Όλα αυτά τα δεδομένα πρέπει να χωριστούν σε κατηγορίες ανάλογα με το περιεχόμενό τους προκειμένου να γίνουν πιο εύκολα προς αναζήτηση. Η διαδικασία αυτή γίνεται, συνήθως, χειρωνακτικά από έναν χρήστη που συνδέει το κάθε βίντεο με λέξεις-κλειδιά ή ετικέτες. Ωστόσο, οι περισσότεροι χρήστες αποφεύγουν να το κάνουν, και έτσι πολλά βίντεο καταλήγουν χωρίς πληροφορίες σχετικά με τις ετικέτες. Αυτή η κατάσταση δημιουργεί την ανάγκη δημιουργίας αλγορίθμων για αυτοματοποιημένη εύρεση του κατάλληλου βίντεο με βάση το περιεχόμενο του.

Ένα άλλο πεδίο εφαρμογών είναι η περίληψη βίντεο. Αυτές οι εφαρμογές χρησιμοποιούνται συνήθως σε ταινίες ή ανθητικές εκδηλώσεις. Στις ταινίες, οι αλγόριθμοι ανάλυσης βίντεο μπορούν να δημιουργήσουν ένα μικρό βίντεο που περιέχει όλες τις σημαντικές στιγμές της ταινίας. Αυτό μπορεί να επιτευχθεί επιλέγοντας τμήματα βίντεο στα οποία λαμβάνει χώρα μια σημαντική ενέργεια, όπως η δολοφονία του κακοποιού της ταινίας. Στις ανθητικές εκδηλώσεις, οι εφαρμογές περίληψης βίντεο περιλαμβάνουν τη δημιουργία αυτόματων βίντεο προβολής, όπως π.χ. ένα βίντεο που περιέχει όλα τα επιτευχθέντα γκολ σ' έναν ποδοσφαιρικό αγώνα.

Επιπλέον, η αναγνώριση της ανθρώπινης δράσης μπορεί να αντικαταστήσει τους ανθρώπινους χειριστές στα συστήματα επιτήρησης. Μέχρι τώρα, τα συστήματα ασφαλείας περιλαμβάνουν ένα σύστημα πολλαπλών κάμερων που τα χειρίζεται ένας άνθρωπος-χειριστής, ο οποίος κρίνει εάν ένα άτομο ενεργεί συνηθισμένα ή όχι. Τα συστήματα αυτόματης ταξινόμησης ενέργειας μπορούν να ενεργούν όπως ο άνθρωπος, και αμέσως να κρίνουν εάν υπάρχει κάποιου είδους περίεργη συμπειριφορά ή ανωμαλία στον άνθρωπο.

Τελευταίο άλλα όχι ασήμαντο, ένα άλλο πεδίο εφαρμογής σχετίζεται με την αλληλεπίδραση ανθρώπου-υπολογιστή. Ρομποτικές εφαρμογές βιοηθούν τους ηλικιωμένους να αντιμετωπίζουν τις καθημερινές τους ανάγκες. Επίσης, οι εφαρμογές παιχνιδιών που χρησιμοποιούν το Kinect δημιουργούν νέα επίπεδα εμπειρίας παιχνιδιού χωρίς την ανάγκη ενός ελεγκτή φυσικού παιχνιδιού.

## 1.3 Προκλήσεις και Datasets

Τπάρχουν διάφοροι τύποι ανθρώπινων δραστηριοτήτων. Ανάλογα με την πολυπλοκότητά τους, θεωρούμε ότι οι ανθρώπινες δραστηριότητες ταξινομούνται σε τέσσερις διαφορετικές κατηγορίες επίπεδα: χειρονομίες, ενέργειες, αλληλεπιδράσεις και δραστηριότητες ομάδας. Οι χειρονομίες είναι στοιχειώδεις κινήσεις του σώματος ενός ατόμου και είναι τα ατομικά στοιχεία που περιγράφουν την ουσιαστική κίνηση ενός ατόμου. «Το στρίψιμο του βραχίονα» και «της ανύψωσης του ποδιού» είναι καλά παραδείγματα χειρονομιών. Οι ενέργειες είναι δραστηριότητες ενός ατόμου που μπορούν να αποτελούνται από πολλαπλές χειρονομίες που οργανώνονται προσωρινά, όπως «περπάτημα», «χαιρετισμός» «γροθιά». Οι αλληλεπιδράσεις είναι ανθρώπινες δραστηριότητες που περιλαμβάνουν δύο ή περισσότερα άτομα και/ή αντικείμενα. Για παράδειγμα, «δύο άτομα που αγωνίζονται» είναι μια αλληλεπίδραση μεταξύ δύο ανθρώπων και «ενός ατόμου που κλέβει μια βαλίτσα από κάποιον άλλο» είναι μια αλληλεπίδραση ανθρώπου-αντικειμένου που περιλαμβάνει δύο ανθρώπους και ένα αντικείμενο. Τέλος, οι δραστηριότητες ομάδας είναι οι δραστηριότητες που εκτελούνται από εννοιολογικές ομάδες που αποτελούνται από πολλαπλά πρόσωπα και/ή αντικείμενα. «Μια ομάδα

ανθρώπων που κάνουν πορεία», «μια ομάδα που έχει συνάντηση» και «δύο ομάδες παλεύουν» είναι τυπικά παραδείγματα αυτών.

Η μεγάλη ποικιλία ανθρώπινων δραστηριοτήτων και εφαρμογών δημιουργεί πολλές προκλήσεις που περιλαμβάνουν συστήματα αναγνώρισης της δράσης. Οι σημαντικότερες προκλήσεις περιλαμβάνουν μεγάλες διακυμάνσεις της εμφάνισης των ανθρώπων που δρουν, αλλαγές στην οπτική γωνία της κάμερας, αποκλείσεις, μη-άκαμπτες κινήσεις κάμερας κλπ. Επιπλέον, ένα μεγάλο πρόβλημα είναι ότι υπάρχουν πάρα πολλές κατηγορίες δράσης που σημαίνει ότι η χειροκίνητη συλλογή των δειγμάτων εκπαίδευσης είναι απαγορευτική. Επίσης, ορισμένες φορές, το λεξιλόγιο περιγραφής των δράσεων δεν είναι καλά καθορισμένο. Όπως δείχνει το σχήμα 7.1, η ενέργεια «Ανοίγω» μπορεί να περιλαμβάνει πολλά είδη ενέργειών, γι' αυτό πρέπει προσεκτικά να αποφασίσουμε ποια έννοια αυτής της πράξης θα λάβουμε υπόψη.



Σχήμα 1.1: Παραδείγματα της δράσης «Ανοίγω»

Προκειμένου να αντιμετωπιστούν αυτές οι προκλήσεις, έχουν δημιουργηθεί διάφορα σύνολα δεδομένων για ανθρώπινες δράσεις, με σκοπό να αναπτυχθούν ισχυρά συστήματα αναγνώρισης της ανθρώπινης δράσης και αλγόριθμοι ανίχνευσης. Τα πρώτα σύνολα δεδομένων περιέλαμβαν έναν δρώντα κάνοντας χρήση μιας στατικής κάμερας πάνω σε ομοιογενή φόντα. Παρόλο που αυτά τα σύνολα δεδομένων συνείσφεραν στο να σχεδιάσουμε τους πρώτους αλγορίθμους αναγνώρισης δράσης, δεν ήταν σε θέση να αντιμετωπίσουν αποτελεσματικά τις παραπάνω προκλήσεις. Έτσι, οδηγηθήκαμε στον να δημιουργήσουμε σύνολα δεδομένων που περιέχουν πιο αμφιλεγόμενα βίντεο, όπως τα Joint-annotated Human Motion Database (JHMDB) (Jhuang et al. 2013) και UCF-101 (Soomro, Zamir, and Shah 2012). Αυτά τα dataset περιλαμβάνουν μόνο ανθρώπινες ενέργειες, την δεύτερη δηλαδή κατηγορία που αναφέρθηκε πιο πριν.

### 1.3.1 JHMDB Dataset

Το σύνολο δεδομένων JHMDB (Jhuang et al. 2013) είναι ένα πλήρες σχολιασμένο σύνολο δεδομένων για ανθρώπινες ενέργειες και ανθρώπινες πόζες. Αποτελείται από 21 κατηγορίες δράσεων και 928 κλπ που εξάγονται από την βάση δεδομένων κίνησης του ανθρώπου (HMD51) (Kuehne et al. 2011). Αυτό το σύνολο δεδομένων περιέχει κομμένα βίντεο με διάρκεια μεταξύ 15 έως 40 καρέ. Κάθε κλιπ σχολιάζεται για κάθε καρέ χρησιμοποιώντας μια δισδιάστατη στάση και περιέχει μόνο 1 ενέργεια. Προκειμένου να εκπαιδεύσουμε το μοντέλο μας για τον εντοπισμό των ενέργειών, τροποποιούμε της δισδιάστατες πόζες σε δισδιάστατα πλαίσια που περιέχουν ολόκληρη τη στάση του δρώντα σε κάθε καρέ. Υπάρχουν διαθέσιμα 3 διαφορετικά χωρίσματα για να εκπαιδευτεί ένα μοντέλο, τα οποία προτείνουν οι συγγραφείς. Επιλέξαμε το πρώτο που περιέχει 660 βίντεο στο εκπαιδευτικό σετ και 268 για επικύρωση.

### 1.3.2 UCF-101 Dataset

Το σύνολο δεδομένων UCF-101 (Soomro, Zamir, and Shah 2012) περιέχει 13320 βίντεο από 101 κατηγορίες δράσεων. Από αυτά, παρέχονται χωροχρονικοί σχολιασμοί για 24 κλάσεις και 3194 βίντεο. Αυτό σημαίνει ότι σε κάθε βίντεο, υπάρχει ένα 2D οριοθετημένο πλαίσιο που περιβάλλει το άτομο που δρα για κάθε καρέ στο οποίο λαμβάνει χώρα μια δράση. Διαχωρίζουμε το σύνολο των δεδομένων σε 2284 βίντεο για εκπαίδευτικό σετ και 910 για σετ επικύρωσης σύμφωνα με το πρώτο προτεινόμενο διαχωρισμό. Για τα δεδομένα εκπαίδευσης, υπάρχουν βίντεο μέχρι 641 καρέ, ενώ για τα βίντεο επικύρωσης ο μέγιστος αριθμός καρέ είναι 900. Κάθε βίντεο, τόσο για την εκπαίδευση όσο και για την επικύρωση, είναι μη-κομμένο(untrimmed), ενώ σε πολλές περιπτώσεις περισσότερες από 1 ενέργειες πραγματοποιούνται ταυτόχρονα. Λάβαμε σχολιασμούς από Singh et al. 2017 επειδή εκείνους που μας παρείχαν οι συγγραφείς περιείχαν ορισμένα λάθη.

## 1.4 Motivation and Contributions

Τα τρέχοντα επιτεύγματα στα δίκτυα αναγνώρισης αντικειμένων και στα 3D Convolutional Neural Network για αναγνώριση ενεργειών μας προκάλεσαν να δοκιμάσουμε για να τα συνδυάσουμε, προκειμένου να επιτύχουμε τα καλύτερα αποτελέσματα στο πρόβλημα του εντοπισμού ανθρώπινης δράσης. Εισάγουμε μια νέα δομή δικτύου εμπνευσμένη από τους Hou, Chen, and Shah 2017, τους Girdhar et al. 2018 και τους Ren et al. 2017 ενώ την υλοποίηση από τους Yang et al. 2017.

Οι συνεισφορές μας είναι οι εξής: 1) Δημιουργούμε ένα νέο framework για τον εντοπισμό των ενεργειών που επεκτείνει τον κώδικα που έχει υλοποιηθεί το FasterR-CNN, 2) Δημιουργήσαμε ένα δίκτυο για πρόταση ακολουθιών δισδιάστατων κουτιών σε βίντεο κλπ τα οποία μπορεί να περιέχουν μια δράση, εκμεταλλεύμενοι τα χωροχρονικά χαρακτηριστικά που μας παρέχουν οι 3D Convolutions, 3) δημιουργούμε έναν αλγόριθμο σύνδεσης για τη σύνδεση των προτεινόμενων ακολουθιών προκειμένου να εξάγουμε υποφήφια action tubes και 4) προσπαθήσαμε να βρούμε τους καταλληλότερους χάρτες χαρακτηριστικών καθώς και τον κατάλληλο ταξινομητή για να πραγματοποιήσουμε αποδοτικό classification.

## 1.5 Thesis structure

Η υπόλοιπη διατριβή οργανώνεται ως εξής. Το κεφάλαιο 2 παρουσιάζει μια σύντομη επισκόπηση της βιβλιογραφίας σχετικά με την αναγνώριση και τον εντοπισμό της ανθρώπινης δράσης. Το Κεφάλαιο 3 εισάγει το πρώτο βασικό στοιχείο του δικτύου μας, το Tube Proposal Network (TPN), ένα δίκτυο που προτείνει Tubes of Interest (ToIs), οι οποίες είναι ακολουθίες δισδιάστατων πλαισίων, που είναι πιθανά να περιέχουν μια εκτελεσθείσα ενέργεια. Επιπλέον, παρουσιάζονται όλες τις προτεινόμενες από μας αρχιτεκτονικές για την επίτευξη αυτού του στόχου. Το κεφάλαιο 4 προτείνει αλγόριθμους για τη σύνδεση των προτεινόμενων ToIs από κάθε τμήμα βίντεο και παρουσιάζονται οι επιδόσεις των προτάσεων. Στο Κεφάλαιο 5 παρουσιάζουμε όλες τις προσεγγίσεις ταξινόμησης που χρησιμοποιήσαμε για τον σχεδιασμό της αρχιτεκτονικής μας και ορισμένα αποτελέσματα ταξινόμησης. Το Κεφάλαιο 6 χρησιμοποιείται για συμπεράσματα, περίληψη της συμβολής μας μαζί με πιθανές μελλοντικές εργασίες.

## Κεφάλαιο 2

# Σχετική βιβλιογραφία

Σε αυτή την ενότητα, παρουσιάζουμε ορισμένες από τις πιο συναφείς μεθόδους για την εργασία μας και όλες που μελετήθηκαν για τον σχεδιασμό αυτής της προσέγγισης. Οι μέθοδοι αυτές χωρίζονται σε δύο ενότητες *Αναγνώριση Δραστηριότητας* και *Εντοπισμός Δραστηριότητας*. Το πρώτο μέρος αναφέρεται σε κλασικές μεθόδους ταξινόμησης δράσης που εισήχθησαν μέχρι πρόσφατα και το δεύτερο μέρος, αντίστοιχα, σε πρόσφατες μεθόδους εντοπισμού της δράσης.

### 2.0.1 Αναγνώριση Δραστηριότητας

Οι πρώτες προσεγγίσεις για την κατάταξη της δράσης αποτελούνταν από δύο βήματα α) αρχικά υπολογισμός σύνθετων «χειροποίητων» χαρακτηριστικών από ακατέργαστα καρέ βίντεο και β) εκπαίδευση ενός ταξινομητή με βάση αυτά τα χαρακτηριστικά. Αυτά τα χαρακτηριστικά μπορούν να διαχωριστούν σε 3 κατηγορίες: 1) προσεγγίσεις χωροχρονικού όγκου (space-time volume), 2) τροχιές (trajectories) και 3) χωροχρονικά χαρακτηριστικά. Για τις μεθόδους χωροχρονικού όγκου, η προσέγγιση είναι η εξής: Με βάση τα training βίντεο, το σύστημα συνάπτει ένα μοντέλο τρισδιάστατου χωροχρόνου, συνενώνοντας δισδιάστατες εικόνες (διάσταση  $x-y$ ) κατά τη διάρκεια του χρόνου (διάσταση  $t$  ή  $z$ ), για την αναπαράσταση κάθε δράσης. Όταν το σύστημα δέχεται ένα βίντεο που δεν έχει ετικέτα, κατασκευάζει μια τρισδιάστατη χωροχρονική αναπαράσταση που αντιστοιχεί σε αυτό το βίντεο. Αυτό η νέα τρισδιάστατη αναπαράσταση, στη συνέχεια, συγχρίνεται με κάθε μοντέλο 3D χωροχρόνου, συγχρίνοντας την ομοιότητα στο σχήμα και την εμφάνιση μεταξύ αυτών των δύο χωροχρονικών όγκων. Το σύστημα εξάγει την κατηγορία του άγνωστου βίντεο, αντιστοιχώντας την με αυτήν της δράσης με την υψηλότερη ομοιότητα. Επιπλέον, υπάρχουν διάφορες παραλλαγές των χωροχρονικών αναπαραστάσεων. Αντί της αναπαράστασης space-time volume, το σύστημα μπορεί να αναπαριστά τη κάθε δράση ως τροχιές σε χωροχρονικές διαστάσεις ή ακόμη περισσότερο, η ενέργεια μπορεί να αναπαρασταθεί ως ένα σύνολο χαρακτηριστικών που εξάγονται από τον χωροχρονικό όγκο ή τις τροχιές. Οι «καθαρές» χωροχρονικές αναπαραστάσεις περιλαμβάνουν μεθόδους σύγκρισης των περιοχών προσκηνίου ενός απόμου (δηλ. σιλουέτες) όπως των Bobick and Davis 2001, συγχρίνοντας όγκους σε σχέση με επιφάνεια τους όπως οι Shechtman and Irani 2005. Η μέθοδος των Ke, Sukthankar, and Hebert 2007 χρησιμοποιεί oversegmented όγκους, αυτομάτως υπολογίζοντας ένα σύνολο τμημάτων τρισδιάστατου όγκου XYT που αντιστοιχεί σε έναν κινούμενο άνθρωπο. Οι Rodriguez, Ahmed, and Shah 2008 πρότειναν φίλτρα για να αποτυπώνουν τα χαρακτηριστικά του χωροχρονικού όγκου, προκειμένου να τα ταυτίζουν πιο αξιόπιστα και αποδοτικά. Από την άλλη πλευρά, οι προσεγγίσεις με βάση την τροχιά περιλαμβάνουν την αναπαράσταση μιας ενέργειας ως σύνολο 13 κοινών διαδρομών (Sheikh, Sheikh, and Shah 2005) ή τη χρήση ενός συνόλου XYZT-διαστάσεων κοινών τροχιών που λαμβάνονται

από κινούμενες κάμερες (Yilmaz and Shah 2005). Τέλος, διάφορες μέθοδοι χρησιμοποιούν τοπικά χαρακτηριστικά που εξάγονται από χωροχρονικούς όγκους τριών διαστάσεων, όπως η εξαγωγή τοπικών χαρακτηριστικών σε κάθε καρέ του βίντεο και η ένωση τους χρονικά (Chomat and Crowley 1999; Zelnik-Manor and Irani 2001; Blank et al. 2005, η εξαγωγή αραιών χωροχρονικών τοπικών σημείων ενδιαφέροντος από τρισδιάστατους όγκους (Laptev and Lindeberg 2003; Dollar et al. 2005; Niebles, Wang, and Li 2006; Alper Yilmaz and Mubarak Shah 2005; Ryoo and Aggarwal 2006) Οι προσεγγίσεις αυτές κατέστησαν την επιλογή των χαρακτηριστικών σημαντικό παράγοντα για την απόδοση του δικτύου. Αυτό συμβάλλει επειδή οι διαφορετικές κατηγορίες δράσεων μπορεί να διαφέρουν δραματικά από την άποψη της εμφάνισής τους και των μοτίβων κίνησης. Ένα άλλο πρόβλημα ήταν ότι οι περισσότερες από αυτές τις προσεγγίσεις κάνουν υποθέσεις, υπό τις οποίες το βίντεο λήφθηκε λόγω προβλημάτων όπως το γεμάτο φόντο, γωνίες κάμερας κλπ. Μια ανασκόπηση των τεχνικών, που χρησιμοποιούνταν μέχρι το 2011, παρουσιάζεται απ' τους Aggarwal and Ryoo 2011.

Τα πρόσφατα αποτελέσματα σε βαθιές αρχιτεκτονικές και ειδικά στον τομέα της ταξινόμησης εικόνας έδωσε κίνητρο στους ερευνητές να εκπαιδεύσουν δίκτυα CNN για το πρόβλημα της αναγνώρισης δράσης. Η πρώτη σημαντική απόπειρα έγινε από τους Karpathy et al. 2014. Σχεδίασαν την αρχιτεκτονική τους με βάση το καλύτερο CNN στον διαγωνισμό ImageNet. Εξερευνούν διάφορες μεθόδους για τη σύντηξη των χωροχρονικών λειτουργιών χρησιμοποιώντας δισδιάστατες διαδικασίες κυρίως και τρισδιάστατη συνέλιξη μόνο μέσω αργής σύντηξης. Οι Simonyan and Zisserman 2014 χρησιμοποίησαν 2 CNNs, ένα για χωρικές πληροφορίες και ένα για οπτική ροή και τα συνδύασαν με τη χρήση της καθυστερημένης σύντηξης. Δείχνουν ότι η εξόρυξη χωρικού περιεχομένου από τα βίντεο και περιεχόμενο κίνησης από την οπτική ροή μπορεί να βελτιώσει σημαντικά την ακρίβεια της αναγνώρισης της δράσης. Οι Feichtenhofer, Pinz, and Zisserman 2016 επέκτειναν αυτή την προσέγγιση με τη χρήση πρώιμης σύντηξης στο τέλος των convolutional layers αντί της καθυστερημένης σύντηξης, η οποία λαμβάνει χώρα στο τελευταίο επίπεδο του δικτύου. Πάνω σ' αυτό, χρησιμοποίησαν ένα δεύτερο δίκτυο για το χρονικό περιεχόμενο το οποίο συνδέουν με το το άλλο δίκτυο με χρήση της καθυστερημένης σύντηξης. Επιπλέον, οι Wang et al. 2016 στήριξαν την μέθοδος τους σε αυτήν που πρότειναν οι Simonyan and Zisserman 2014. Ασχολούνται με το πρόβλημα της εύρεσης χρονικού περιεχομένου και εκπαιδεύουν το δίκτυο τους, παρέχοντας του λίγα δείγματα. Η προσέγγισή τους, την οποία ονόμασαν Temporal Segment Network (TSN), διαχωρίζει το βίντεο εισόδου σε K τμήματα και ένα σύντομο απόσπασμα από κάθε τμήμα επιλέγεται για ανάλυση. Στη συνέχεια, συνδέουν το εξαγόμενο χωροχρονικό περιεχόμενο, πραγματοποιώντας τελικά την πρόβλεψή τους. Πιο πρόσφατα, οι Zhang et al. 2016 και οι Zhu et al. 2017 χρησιμοποίησαν την two-stream προσέγγιση, επίσης. Οι Zhang et al. 2016 αντικατέστησαν την οπτική ροή με ένα διάνυσμα κίνησης που μπορεί να ληφθεί απευθείας από τα συμπιεσμένα βίντεο χωρίς επιπλέον υπολογισμό και το τροφοδοτούν στο δίκτυο. Οι Zhu et al. 2017 εκπαίδευσαν ένα CNN για τον υπολογισμό της οπτικής ροής, καλώντας το, MotionNet, και χρησιμοποίησαν ένα CNN ως χρονικό stream για προβάλλουν τις πληροφορίες κίνησης έργου σε κατηγορίες δράσεων. Τέλος χρησιμοποιούν την καθυστερημένη σύντηξη μέσω της μέσης τιμής με βάρη τα σκορ πρόβλεψης των χρονικών και χωρικών stream. Από την άλλη πλευρά, μια νέα προσέγγιση εισήχθη από τους Girdhar and Ramanan 2017 ενσωματώνοντας χάρτες προσοχής με σκοπόν να βελτιώσουν σημαντικά την απόδοση της αναγνώρισης δράσης.

Ορισμένες άλλες μέθοδοι περιλαμβαναν ένα δίκτυο RNN ή LSTM για την ταξινόμηση όπως κάνουν οι Donahue et al. 2017, οι Joe Yue-Hei Ng et al. 2015 και οι Ma et al. 2017. Οι Donahue et al. 2017 αντιμετωπίζουν τη πρόκληση των μεταβλητών μεγεθών των ακολουθιών εισόδου και εξόδου, εκμεταλλευόμενοι τα convolutional layers και τις μεγάλου εύρους χρονικές αναδρομές (recursions). Προτείνουν ένα Long-term Recurrent Convolutional Network (LRCN), το οποίο είναι ικανό να αντιμετωπίσει τις εργασίες αναγνώρισης, λεζάντας εικόνας και περιγραφής βίντεο. Για να ταξινομήσει μια δεδομένη ακολουθία καρέ, το LRCN λαμβάνει αρχικά ως είσοδο ένα καρέ, και πιο

συγκεκριμένα τα κανάλια RGB και την οπτική ροή του, και προβλέπει μια ετικέτα. Μετά από αυτό, εξάγει την κλάση του βίντεο μέσω του μέσου όρου των πιθανοτήτων των ετικετών, επιλέγοντας την πιο πιθανή κλάση. Οι Joe Yue-Hei Ng et al. 2015 πρώτα διερευνούν διάφορες προσεγγίσεις για χρονική ομαδοποίηση (temporal pooling) των χαρακτηριστικών. Αυτές οι τεχνικές περιλαμβάνουν τον χειρισμό καρέ βίντεο ξεχωριστά από 2 αρχιτεκτονικές CNN: είτε απ' το AlexNet είτε απ' το GoogleNet, και αποτελούνται από πρώιμη σύντηξη, καθυστερημένη σύντηξη και ενός συνδυασμού αυτών. Επιπλέον, προτείνουν ένα RNN προκειμένου να εξετάσουν τα βίντεο κλιπ ως ακολουθίες ενεργοποιήσεων CNN. Το προτεινόμενο LSTM λαμβάνει ως είσοδο την έξοδο του τελικού CNN layer για κάθε συνεχόμενο καρέ και μετά από 5 LSTM layers και χρησιμοποιώντας έναν softmax ταξινομητή, προτείνει μία ετικέτα. Για την ταξινόμηση του βίντεο, επιστρέφουν μια ετικέτα μετά το τελευταίο βήμα, εφαρμόζουν max-pooling στις προβλέψεις στην διάσταση του χρόνου, ανθροίζουν τις προβλέψεις στην διάσταση του χρόνου και επιστρέφουν το μέγιστο ή έναν γραμμικό συνδυασμό με βάρη των προβλέψεων υπό έναν παράγοντα g, τα ανθροίζουν και επιστρέφουν το μέγιστο. Έδειξαν ότι όλες οι προσεγγίσεις είναι 1% διαφορετικές με προκατάληψη για τη χρήση των προβλέψεων με βάρη για την υποστήριξη της ιδέας ότι το LSTM γίνεται προοδευτικά πιο ενημερωμένο. Τελευταίοι αλλά όχι λιγότερο σημαντικοί, οι Ma et al. 2017 χρησιμοποίησαν ένα two-stream ConvNet για εξαγωγή χαρακτηριστικών και είτε ένα LSTM ή convolutional layer πάνω από τους χρονικώς κατασκευασμένους πίνακες χαρακτηριστικών για τη σύντηξη χωρικών και χρονικών πληροφοριών. Χρησιμοποιούν ένα ResNet-101 για την εξάρουξη χαρτών ενεργοποίησης τόσο για χωρικές όσο και για χρονικές ροές. Χωρίζουν το βίντεο σε διάφορα τμήματα, όπως έκαναν οι Wang et al. 2016, και χρησιμοποίησαν ένα επίπεδο temporal pooling για την εξαγωγή διακεκριμένων χαρακτηριστικών. Αφού λάβουν αυτά τα χαρακτηριστικά, το LSTM εξάγει ενσωματωμένες δυνατότητες από όλα τα τμήματα.

Επιπλέον, οι Tran et al. 2015 διερεύνησαν τα 3D Convolutional δίκτυα (Ji et al. 2013) και εισήγαγαν το C3D δίκτυο που έχει 3D convolutional layers με πυρήνες  $3 \times 3 \times 3$ . Αυτό το δίκτυο είναι σε θέση να μοντελοποιήσει την εμφάνιση και την κίνηση ταυτόχρονα χρησιμοποιώντας τρισδιάστατες συνελίξεις και μπορεί να χρησιμοποιηθεί ως εξαγωγέας χαρακτηριστικών. Συνδυάζοντας την αρχιτεκτονική δύο ροών και τις τρισδιάστατες συνελίξεις οι Carreira and Zisserman 2017 πρότειναν το δίκτυο I3D. Πάνω σ' αυτό, οι δημιουργοί τονίζουν τα πλεονεκτήματα της μεταφοράς μάθησης για την εργασία της αναγνώρισης επαναλαμβάνοντας τα δισδιάστατα προ-εκπαίδευμένα βάρη στην 3η διάσταση. Οι Hara, Kataoka, and Satoh 2017 πρότειναν ένα δίκτυο 3D ResNet για την αναγνώριση δράσης με βάση τα Residual δίκτυα (ResNet)(He et al. 2016) και διερευνούν την απόδοση των δικτύων ResNet με 3D Convolutional πυρήνες. Από την άλλη, οι Diba et al. 2017 βάσισαν την προσέγγισή τους στα DenseNets (Huang et al. 2017) και επέκτειναν την αρχιτεκτονική του DenseNet χρησιμοποιώντας τρισδιάστατα φίλτρα και pooling πυρήνες αντί για δισδιάστατους, ονομάζοντας αυτή την προσέγγιση ως DenseNet3D. Επιπλέον, εισάγουν το Layer χρονικής μετάβασης (TTL), το οποίο συνενώνει χρονικά χάρτες χαρακτηριστικών που εξάγονται σε διαφορετικά χρονικά βάθη και αντικαθιστά το επίπεδο μετάβασης του DenseNet. Παράλληλα, οι Diba et al. 2018 εισήγαγαν ένα νέο χρονικό layer το οποίο μοντελοποιεί μεταβλητούς χρονικούς πυρήνες συνέλιξης. Τελευταίοι αλλά εξίσου σημαντικοί, οι Tran et al. 2018 πειραματίστηκαν με διάφορες υπόλοιπες αρχιτεκτονικές Residual δικτύου χρησιμοποιώντας συνδυασμούς 2D και 3D convolutional Layer. Σκοπός τους είναι να δείξουν ότι η 2D χωρική συνέλιξη ακολουθούμενη από 1D χρονική συνέλιξη επιτυγχάνει state-of-the-art αποτελέσματα, ονομάζοντας αυτού του τύπου το layer ως R(2 + 1)D. Πρόσφατα οι Guo et al. 2018 πρότειναν ένα framework που μπορεί να μάθει να αναγνωρίζει μια προηγουμένως αθέατη 3D κλάση δράσης με λίγα μόνο παραδείγματα εκμεταλλευόμενο την εγγενή δομή των 3D δεδομένων μέσω μιας γραφικής αναπαράστασης. Ακόμα πιο λεπτομερή παρουσίαση των τεχνικών αναγνώρισης δράσης που χρησιμοποιήθηκαν μέχρι το 2018 πραγματοποιήθηκε από τους Kong and Fu 2018.

## 2.0.2 Εντοπισμός Δραστηριότητας

Όπως προαναφέρθηκε, ο εντοπισμός δράσης μπορεί να θεωρηθεί ως προέκταση του προβλήματος εντοπισμού αντικειμένων. Αντί να εξάγουμε δισδιάστατα πλαίσια οριοθέτησης σε μία μόνο εικόνα, ο στόχος των συστημάτων εντοπισμού δράσης είναι να εξάγουν action tubes, τα οποία είναι ακολουθίες πλαισίων οριοθέτησης που περιέχουν μια ενέργεια που εκτελέστηκε. Έτσι, υπάρχουν διάφορες προσεγγίσεις, συμπεριλαμβανομένου συνήθως ενός δικτύου ανιχνευτή αντικειμένων και ενός ταξινομητή.

Οι πρώτες προσεγγίσεις ανίχνευσης αντικειμένων περιλαμβαναν την επέκταση ενός αλγορίθμου πρότασης αντικειμένων σε 3 διαστάσεις. Οι Tian, Sukthankar, and Shah 2013 επέκτειναν τα παραμορφώσιμα (deformable) μοντέλα (Felzenszwalb et al. 2010) με το να αντιμετωπίζουν τις δράσεις ως χωροχρονικά μοτίβα και δημιούργησαν ένα παραμορφώσιμο μοντέλο για κάθε δράση. Οι Jain et al. 2014 εισήγαγαν την έννοια των tubelets, γνωστά και ως ακολουθίες πλαισίων οριοθέτησης και βάσισαν τη μέθοδό τους σε επιλεκτικό αλγόριθμο αναζήτησης (Uijlings et al. 2013), επεκτείνοντας τα superpixels σε supervoxels για την παραγωγή χωροχρονικών σχημάτων. Απ’ την άλλη, οι Oneata et al. 2014 επέκτειναν μια τυχαιοποιημένη διαδικασία συγχώνευσης superpixels που χρησιμοποιούνταν για προτάσεις αντικειμένων, όπως παρουσιάστηκαν απ’ τους Manen, Guillaumin, and Gool 2013. Οι Yu and Yuan 2015 πρώτα προτείνουν πλαίσια οριοθέτησης για κάθε καρέ με χρήση ενός ανιχνευτή ανθρώπου και κίνησης, ενώ, στη συνέχεια, με τη επιλογή των καλύτερων σε σκορ κουτιών, πρότειναν έναν άπληστο συνδετικό αλγόριθμο με τη διατύπωση την εργασίας σύνδεσης ως πρόβλημα μέγιστης κάλυψης. Οι Gemert et al. 2015 παράγουν χωροχρονικές προτάσεις κατευθείαν από τις πυκνές τροχιές, οι οποίες επίσης χρησιμοποιήθηκαν για ταξινόμηση. Οι Chen and Corso 2015 δημιουργούν ένα γράφημα χωροχρονικής τροχιάς και επιλέγουν προτάσεις δράσεων που βασίζονται μόνο στην εσκεμμένη κίνηση που εξάγεται από το γράφημα. Οι Soomro, Idrees, and Shah 2015 διαχωρίζουν τα τμήματα βίντεο σε supervoxels και χρησιμοποιούν το περιεχόμενο τους ως χωρική σχέση μεταξύ των supervoxels σε σχέση με την δράση του προσκηνίου. Δημιουργούν ένα γράφημα για κάθε βίντεο, όπου τα supervoxels σχηματίζουν τους κόμβους και οι κατευθυνόμενες ώρες απεικονίζουν τις χωρικές σχέσεις μεταξύ τους. Κατά τη διάρκεια των δοκιμών, κάνουν μια βόλτα στο περιβάλλον, όπου κάθε βήμα καθοδηγείται από τις σχέσεις περιβάλλοντος κατά τη διάρκεια της εκπαίδευσης, με αποτέλεσμα μια κατανομή πιθανότητας μιας δράσης για όλα τα supervoxels. Οι Mettes, Gemert, and Snoek 2016 αντί για τοποθέτηση πλαισίων σε όλα τα καρέ των βίντεο, σχολίασαν σημεία σε ένα αραιό υποσύνολο καρέ του βίντεο και χρησιμοποίησαν προτάσεις που λαμβάνονται μέσω ενός μέτρου επικάλυψης μεταξύ των προτάσεων δράσης και των σημείων. Οι Behl et al. 2017 ασχολούνται με την ανίχνευση και τον εντοπισμό ενεργειών σε πραγματικό χρόνο μέσω της λήψης προτάσεων δράσης ανά καρέ και την πρόταση ενός αλγορίθμου σύνδεσης που είναι σε θέση να κατασκευάσει και να ενημερώνει τα action tubes ανά καρέ. Πιο πρόσφατα, οι Soomro and Shah 2017 προσπάθησαν να ασχοληθούν με το πρόβλημα της ανίχνευσης και τον εντοπισμό δράσης χωρίς επίβλεψη. Η προσέγγισή τους περιελάμβανε αρχικά την εξόρυξη κατακερματισμένων supervoxel και στη συνέχεια την ανάθεση ενός βάρους σε κάθε supervoxel. Με την εξαγωγή supervoxels, δημιουργούν ένα γράφημα και στη συνέχεια χρησιμοποιούν μια διακριτική clustering προσέγγιση έτσι ώστε να εκπαιδεύται ένας ταξινομητής.

Η εισαγωγή του R-CNN (Girshick et al. 2014) κατάφερε σημαντικές βελτιώσεις στην απόδοση των δικτύων εντοπισμού αντικειμένων. Αυτή η αρχιτεκτονική, πρώτον, προτείνει περιοχές στην εικόνα που είναι πιθανό να περιέχουν κάποιο αντικείμενο και στη συνέχεια, τα ταξινομεί χρησιμοποιώντας ένα SVM. Εμπνευσμένοι από αυτή την αρχιτεκτονική, οι Gkioxari and Malik 2015 σχεδίασαν ένα δίκτυο RCNN 2-stream για να προτείνει προτάσεις δράσεων για κάθε καρέ, ένα stream για το επίπεδο καρέ και ένα για την οπτική ροή. Στη συνέχεια, τα συνδέουν χρησιμοποιώντας τον αλγόριθμο σύνδεσης Viterbi. Οι Weinzaepfel, Harchaoui, and Schmid 2015 επεκτείνουν αυτή την προσέγγιση, εκτελώντας προτάσεις στο επίπεδο καρέ και χρησιμοποιώντας ένα tracker για τη

σύνδεση των προτάσεων αυτών μέσω των χαρακτηριστικών της χωρικής και οπτικής ροής. Επίσης, η μέθοδός τους εκτελεί χρονικό εντοπισμό μέσω της χρήσης ενός συρόμενου παράθυρου πάνω από τα εντοπισμένα tubes.

Η εισαγωγή του Faster RCNN (Ren et al. 2017) συνείσφερε πολύ τη βελτίωση της απόδοσης των δικτύων εντοπισμού δράσης. Οι Peng and Schmid 2016 και Saha et al. 2016 χρησιμοποιούν το Faster R-CNN αντί για το RCNN για προτάσεις σε επίπεδο καρέ, χρησιμοποιώντας το RPN για εικόνες RGB και οπτικής ροής. Αφού λάβουν χωρικές προτάσεις και προτάσεις κίνησης, οι Peng and Schmid 2016 τις συγχωνεύουν και από κάθε προτεινόμενη ROI, παράγουν 4 ROIs για να επικεντρωθούν σε συγκεκριμένο μέρος του σώματος του δρώντα. Μετά από αυτό, συνδέουν την πρόταση χρησιμοποιώντας τον αλγόριθμο Viterbi για κάθε κλάση και εκτελούν χρονικό εντοπισμό χρησιμοποιώντας ένα συρόμενο παράθυρο, με πολλαπλές χρονικές κλίμακες και διασκελισμό κάνοντας χρήση μεθόδου μέγιστης υποσυστοιχίας (maximum subarray method). Απ' την άλλη, οι Saha et al. 2016 εκτελούν, επίσης, ταξινόμηση σε επίπεδο καρέ. Μετά απ' αυτό, η μέθοδός τους εκτελεί σύντηξη με βάση έναν συνδυασμό της εμφάνισης και της κίνησης με βάση τις προτάσεις και την βαθμολογία αλληλεπικάλυψης. Τέλος, η χρονική προσαρμογή λαμβάνει χώρα χρησιμοποιώντας δυναμικό προγραμματισμό. Παράλληλα, οι Weinzaepfel, Martin, and Schmid 2016 χρησιμοποιούν το Faster RCNN για την εξαγωγή ανθρώπινων tubes από βίντεο που εστιάζουν στο πρόβλημα του ασθενώς εποπτεύομενου εντοπισμού δράσης. Στη συνέχεια, χρησιμοποιώντας πυκνές τροχιές και μια multi-fold Multiple Instance Learning προσέγγιση (Cinbis, Verbeek, and Schmid 2016) εκπαιδεύουν ένα ταξινομητή. Οι Mettes and Snoek 2017 εισήγαγαν μια μέθοδο για zero-shot εντοπισμού δράσης. Η προσέγγισή τους περιλαμβάνει την βαθμολόγηση των προτεινόμενων action tubes σύμφωνα με τις αλληλεπιδράσεις μεταξύ των ατόμων που δρουν και αντικειμένων. Χρησιμοποίησαν το Faster-RCNN, στο πρώτο βήμα, για την ανίχνευση τόσο των ανθρώπων που δρουν όσο και των αντικειμένων και μετά, χρησιμοποιώντας χωρικές σχέσεις μεταξύ τους, συνδέουν τα προτεινόμενα πλαίσια στον άξονα του χρόνου βασιζόμενοι στην zero-shot πιθανότητα της παρουσίας των ατόμων, συναφών αντικειμένων γύρω απ' αυτούς και τις αναμενόμενες χωρικές σχέσεις μεταξύ αντικειμένων και ανθρώπων που δρουν. Επιπλέον οι He et al. 2018 πρότειναν το Tube Proposal Network (TPN) για τη δημιουργία ανεξαρτήτου κλάσης προτάσεων tubelet, οι οποίες χρησιμοποιούν το Faster R-CNN για να λάβουν δισδιάστατες προτάσεις περιοχών και έναν αλγόριθμο σύνδεσης για τη σύνδεση των tubelets με τις προτάσεις των περιοχών. Πιο πρόσφατα, οι Girdhar et al. 2018 πρότειναν μια μέθοδο για εντοπισμό δράσεων στο σύνολο δεδομένων AVA (Gu et al. 2018) συνδυάζοντας τις αρχιτεκτονικές των I3D (Carreira and Zisserman 2017) και Faster RCNN. Χρησιμοποιούν μπλοκ του I3D για την λήψη αναπαράστασης βίντεο και το RPN του Faster-RCNN για να προτείνει προτάσεις «ανθρώπου» για το κεντρικό πλαίσιο.

Παράλληλα μ' αυτά, οι Singh et al. 2017 και Kalogeiton et al. 2017 σχεδίασαν τα δίκτυα τους με βάση το Single Shot Multibox Detector Liu et al. 2015). Οι Singh et al. 2017 δημιούργησαν ένα χωροχρονικό δίκτυο πραγματικού χρόνου. Για να λειτουργεί το δίκτυο τους σε πραγματικό χρόνο, οι Singh et al. 2017 πρότειναν έναν νέο και αποδοτικό αλγόριθμο με την προσθήκη πλαισίων σε tubes σε κάθε καρέ, εάν επικαλύπτονται περισσότερο από ένα κατώφλι, ή εναλλακτικά, τερματίζουν το action tube εάν για καρέ αν δεν προστέθηκε κανένα πλαίσιο. Οι Kalogeiton et al. 2017 σχεδίασαν ένα δίκτυο δύο ροών, το οποίο κάλεσαν ACT-detector, και εισήγαγαν τα κυβικά (cuboids) anchors. Για K καρέ, και για τα δύο δίκτυα, οι Kalogeiton et al. 2017 εξάγουν χωρικά χαρακτηριστικά σε επίπεδο καρέ, στη συνέχεια, τα στοιβάζουν. Τέλος, με τη χρήση των κυβικών anchors, το δίκτυο εξάγει tubelets, με τις αντίστοιχες βαθμολογίες κατάταξης και στόχους παλινδρόμησης. Για τη σύνδεση των tubelets, οι Kalogeiton et al. 2017 ωριούσαν τα ίδια βήματα με τους Singh et al. 2017, ενώ για χρονικό εντοπισμό, χρησιμοποιούν μιά προσέγγιση χρονικής εξομάλυνσης.

Πιο πρόσφατα, το δίκτυο YOLO (Redmon et al. 2016) έγινε η έμπνευση για τους Hu et al. 2019 και τους Zhang et al. 2016. Στην προσέγγιση που προτάθηκε από τους Hu et al. 2019, οι έννοιες της εξέλιξης και τού ποσοστού προόδου εισήχθησαν. Εκτός από την πρόταση πλαισίων

οριοθέτησης σε επίπεδο καρέ, χρησιμοποιούν το YOLO μαζί με έναν ταξινομητή RNN για να εξάγουν χρονικές πληροφορίες για τις προτάσεις. Με βάση αυτές τις πληροφορίες, δημιουργούν action tubes, χωρίζοντας τα σε κλάσεις. Ορισμένες άλλες προσεγγίσεις περιλαμβάνουν εκτίμηση πόζας, όπως αυτή των Luvizon, Picard, and Tabia 2018. Πρότειναν μια μεθόδο υπολογισμού των δισδιάστατων και τρισδιάστατων ποζών και στη συνέχεια εκτέλεσαν ταξινόμηση δράσεων. Χρησιμοποιούν το διαφορίσμα Soft-argmax για την εκτίμηση των 2D και 3D αρθρώσεων, επειδή η συνάρτηση argmax δεν είναι διαφορίσμα. Στη συνέχεια, για  $T$  παρακείμενες πόζες δημιουργούν μια απεικόνιση εικόνας με το χρόνο και τις  $N_j$  αρθρώσεις ως  $x - y$  άξονες, έχοντας 2 κανάλια για την 2D πόζα και 3 για την 3D πόζα. Χρησιμοποιούν Convolutional Layers για να παράγουν χάρτες θερμότητας δράσης και στη συνέχεια χρησιμοποιώντας max plus min pooling και την συνάρτηση softmax εκτελούν ταξινόμηση δράσης. Οι Zolfaghari et al. 2017 πρότειναν μια αρχιτεκτονική τριών ροών που περιλαμβάνει 2D πόζα, οπτική ροή και πληροφορίες RGB. Αυτά τα streams ενώνονται μέσω του μοντέλου της αλυσίδας Markov. Επιπλέον, οι Zhu, Vial, and Lu 2017 πρότειναν μια αρχιτεκτονική με τη χρήση ενός χρονικού convolutional δικτύου παλινδρόμησης, για να πιάνουν την μακροπρόθεσμη εξάρτηση και πληροφορίες μεταξύ γειτονικών καρέ και ένα χωρικό δίκτυο παλινδρόμησης, για προτάσεις ανά καρέ. Χρησιμοποιούν μεθόδους παραχολούθησης και δυναμικού προγραμματισμού για τη δημιουργία προτάσεων δράσης.

Τα περισσότερα από τα προαναφερθέντα δίκτυα χρησιμοποιούν ανά καρέ χωρικές προτάσεις και εξάγουν τις χρονικές τους πληροφορίες υπολογίζοντας την οπτική ροή. Από την άλλη οι Saha, Singh, and Cuzzolin 2017 σχεδίασαν μια αρχιτεκτονική η οποία περιλαμβάνει προτάσεις σε επίπεδο τμήματος βίντεο, το οποίο σημαίνει περισσότερα από ένα καρέ ταυτόχρονα. Οι Saha, Singh, and Cuzzolin 2017 πρότειναν μια 3D-RPN αρχιτεκτονική που είναι σε θέση να δημιουργήσει και να ταξινομήσει τρισδιάστατες προτάσεις αποτελούμενες από 2 συνεχόμενα καρέ. Επίσης, πρότειναν έναν αλγόριθμο σύνδεσης, τροποποιώντας αυτόν που πρότειναν οι Saha et al. 2016. Πάνω σ' αυτό, οι Hou, Chen, and Shah 2017 σχεδίασαν μια αρχιτεκτονική για τη δημιουργία προτάσεων δράσης για περισσότερα από 2 καρέ, καλώντας το μοντέλο τους Tube CNN (T-CNN). Στην προσέγγισή τους, επεξεργασία στο επίπεδο του βίντεο σημαίνει ότι ολόκληρο το βίντεο χωρίζεται κλιπ βίντεο ίδιου αριθμού καρέ και με τη χρήση του C3D για την εξόρυξη χαρακτηριστικών, επιστρέφουν χωροχρονικές προτάσεις. Μετά την λήψη των προτάσεων, οι Hou, Chen, and Shah 2017 συνδέουν τις tube προτάσεις τους με έναν αλγόριθμο στηριζόμενος στην πιθανότητα ύπαρξης δράσης και την επικάλυψη μεταξύ των tubes. Τέλος, η λειτουργία ταξινόμησης λαμβάνει χώρα για τα συνδεδεμένα action tubes.

## Κεφάλαιο 3

# Tube Proposal Network

### 3.1 Η αρχιτεκτονική του μοντέλου μας

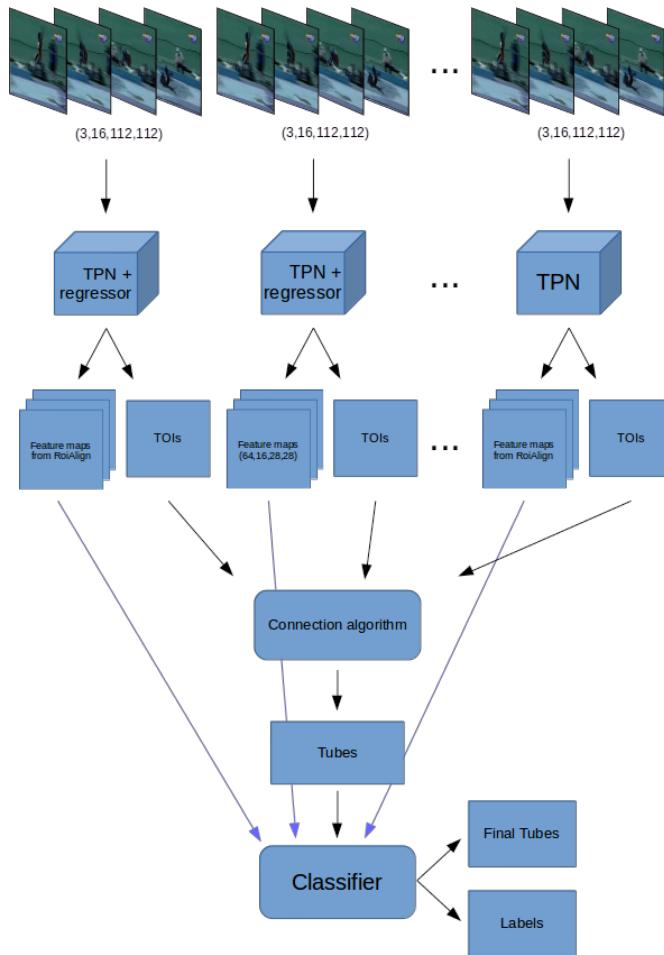
Σε αυτό το κεφάλαιο, ασχολούμαστε κυρίως με το Tube Proposal Network (TPN), ένα από τα βασικά στοιχεία του μοντέλου μας ActionNet. Πριν ξεκινήσουμε την περιγραφή του, παρουσιάζουμε τη δομή όλου του μοντέλου μας. Προτείνουμε ένα δίκτυο παρόμοιο με αυτό των Hou, Chen, and Shah 2017. Η αρχιτεκτονική μας αποτελείται από τα ακόλουθα βασικά στοιχεία:

- Ένα τρισδιάστατο συνελικτικό μοντέλο (3D Convolutional Network), το οποίο χρησιμοποιείται για την εξαγωγή χαρακτηριστικών. Στην υλοποίησή μας χρησιμοποιούμε ένα δίκτυο 3D ResNet34, του οποίου τον κώδικα λαμβάνουμε από τους Hara, Kataoka, and Satoh 2018 και βασίζεται στα ResNet CNNs για ταξινόμηση εικόνων (He et al. 2016).
- Ένα TPN για την εξαγωγή υποψήφιων ToIs (βασιζόμενοι στην ιδέα που παρουσιάζουν οι Hou, Chen, and Shah 2017 ).
- Έναν ταξινομητή για την εύρεση της κλάσης των προτεινόμενων action video tubes.

Η βασική διαδικασία που ακολουθεί το ActionNet είναι:

1. Δεδομένου ενός βίντεο, το διαχωρίσουμε σε τμήματα βίντεο. Αυτά τα τμήματα βίντεο σε ορισμένες προσεγγίσεις επικαλύπτονται χρονικά και σε άλλες όχι.
2. Για κάθε τμήμα βίντεο, μετά την εκτέλεση της αλλαγής μεγέθους χωροχρονικά, τροφοδοτούμε τα καρέ του στο ResNet34για να εξάγουμε τους χωροχρονικούς χάρτες του. Αυτοί οι χάρτες ενεργοποίησης, στη συνέχεια, τροφοδοτούνται στο TPN για την πρόταση ακόλουθιών πλαισίων που πιθανόν περιέχουν κάποια δράση τις οποίες θα ονομάσουμε Tubes of Interest (ToIs), όπως κάνουν και οι Hou, Chen, and Shah 2017.
3. Μετά την πρόταση ToIs για κάθε τμήμα βίντεο, χρησιμοποιώντας έναν αλγόριθμο σύνδεσης, το ActionNet βρίσκει τα τελικά υποψήφια action tubes. Αυτά τα action tubes δίνονται ως είσοδο σε έναν ταξινομητή για τον προσδιορισμό της κλάσης τους.

Ένα διάγραμμα του μοντέλου μας ActionNet εμφανίζεται στην εικόνα 3.1.



Σχήμα 3.1: Network's structure

## 3.2 Εισαγωγή στο TPN

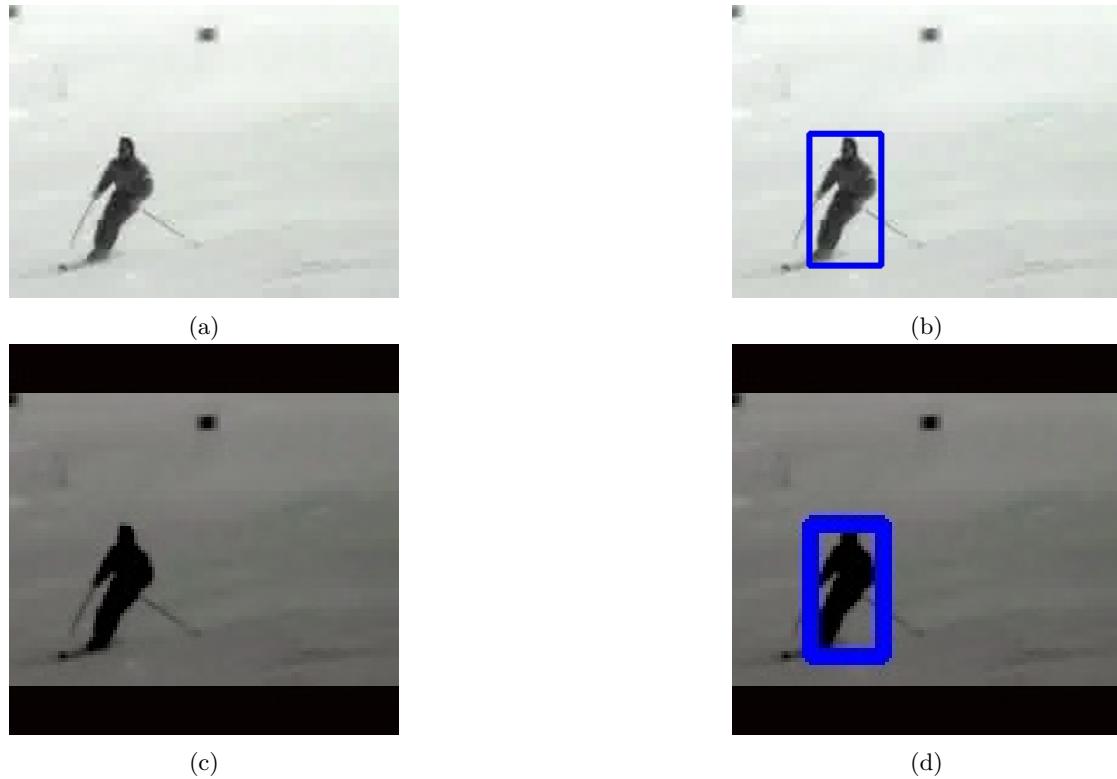
Ο κύριος σκοπός του TPN είναι να προτείνει **Tubes of Interest** (TOIs). Αυτά τα tubes είναι πιθανό να περιέχουν μια γνωστή δράση και αποτελούνται από μερικά δισδιάστατα πλαίσια (1 για κάθε καρέ βίντεο). Το TPN είναι εμπνευσμένο από το RPN που εισήχθη από το FasterRCNN (Ren et al. 2017), αλλά αντί για εικόνες, το TPN χρησιμοποιείται σε βίντεο όπως κάνουν και οι Hou, Chen, and Shah 2017. Σε πλήρη αντιστοιχία με το RPN, η δομή του TPN είναι παρόμοια με αυτή του RPN. Η μόνη διαφορά, είναι ότι το TPN χρησιμοποιεί 3D Convolutional Layers και 3D anchors αντί για 2D.

Σχεδιάσαμε 2 κύριες δομές για το TPN. Κάθε προσέγγιση έχει διαφορετικό ορισμό για τα χρησιμοποιούμενα τρισδιάστατα anchors. Η υπόλοιπη δομή του TPN είναι κυρίως η ίδια με ορισμένες μικρές διαφορές στο στάδιο του regression.

### 3.3 Προετοιμασία πριν το TPN

#### 3.3.1 Η προετοιμασία των δεδομένων

Πριν εισαχθεί ένα βίντεο στο ResNet και στο TPN για να εξαγάγουμε τα χαρακτηριστικά του και πιθανά ToIs, αυτό το βίντεο πρέπει να προεπεξεργαστεί. Η διαδικασία προεπεξεργασίας είναι η ίδια και για τις δύο προσεγγίσεις του TPN. Η αρχιτεκτονική μας λαμβάνει ως είσοδο μια ακολουθία από σταθερό αριθμό καρέ που έχουν σταθερό πλάτος και ύψος. Ωστόσο, κάθε βίντεο είναι πιθανόν να έχει διαφορετική ανάλυση. Αυτό δημιουργεί την ανάγκη να αλλάξουμε το μέγευθος κάθε καρέ και πλαισίου πριν εισαχθεί στο ActionNet. Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, το πρώτο στοιχείο του δικτύου μας είναι ένα 3D RenNet που υλοποιήθηκε από τους Hara, Kataoka, and Satoh 2018. Αυτό το δίκτυο έχει σχεδιαστεί να δέχεται βίντεο με διαστάσεις (112, 112). Ως αποτέλεσμα, μεταβάλλουμε το μέγευθος κάθε καρέ από τα βίντεο των dataset σε (112, 112). Για να διατηρήσουμε την αναλογία διαστάσεων, προσθέτουμε μηδενικές τιμές είτε αριστερά και δεξιά, είτε πάνω και κάτω, ανάλογα με το ποια διάσταση είναι μεγαλύτερη. Στο σχήμα 3.2μπορούμε να δούμε το αρχικό καρέ καθώς και το αναδιαμορφωμένο. Σε πλήρη αντιστοιχία, αλλάζουμε και το μέγευθος των πραγματικών πλαισίων οριοθέτησης για κάθε καρέ (Τα σχήματα 3.2b και 3.2d το απεικονίζουν).



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

### 3.3.2 3D ResNet

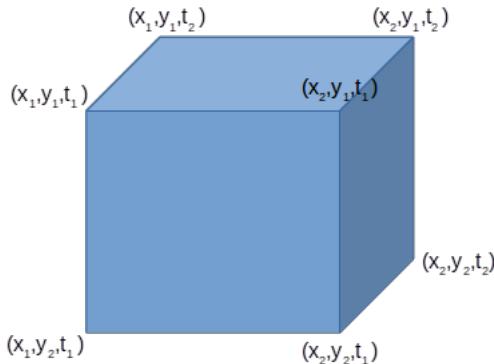
Πριν από τη χρήση του Tube Proposal Network, εξάγουμε χωροχρονικά χαρακτηριστικά από το βίντεο. Για να γίνει αυτό, εξάγουμε τα 3 πρώτα στρώματα ενός προεκπαιδευμένου 3D ResNet34. Αυτό το μοντέλο είναι προεκπαιδευμένο στο Kinetics dataset (Kay et al. 2017) για διάρκεια δειγμάτος ίση με 16 καρέ και μέγεθος δείγματος ίσο με (112, 122).

Αυτό το δίκτυο συνήθως χρησιμοποιείται για την ταξινόμηση ολόκληρου του βίντεο, οπότε μερικά από τα στρώματα του χρησιμοποιούν temporal stride ίσο με 2. Εμείς, όμως, υέτουμε το temporal stride ίσο με 1 γιατί δεν θέλουμε να χάσουμε χρονικές πληροφορίες κατά τη διάρκεια της διαδικασίας. Έτσι, η έξοδος του τρίτου στρώματος είναι ένα χάρτης χαρακτηριστικών με διαστάσεις (256, 16, 7, 7). Τροφοδοτούμε αυτό τον χάρτη χαρακτηριστικών στο TPN, το οποίο περιγράφεται στις ακόλουθες ενότητες.

## 3.4 Τα τρισδιάστατα anchors ως 6-dim διανύσματα

### 3.4.1 Πρώτη περιγραφή

Ξεκινήσαμε να σχεδιάζουμε το TPN εμπνευσμένοι από την δουλειά των Hou, Chen, and Shah 2017. Έτσι, υεωρούμε κάθε anchor ως ένα τρισδιάστατο πλαίσιο το οποίο γράφεται ως  $(x_1, y_1, t_1, x_2, y_2, t_2)$  όπου  $x_1, y_1, t_1$  είναι οι πάνω μπροστά αριστερές διαστάσεις του κύβου και  $x_2, y_2, t_2$  είναι οι κάτω πίσω δεξιά όπως φαίνεται και στην εικόνα 3.3.



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

Το κύριο πλεονέκτημα αυτής της προσέγγισης είναι ότι εκτός από τις διαστάσεις x-y, η διάσταση του χρόνου είναι μεταβαλλόμενη. Ως αποτέλεσμα, τα προτεινόμενα ToIs δεν έχουν καθορισμένη χρονική διάρκεια. Αυτό θα μας βοηθήσει να ασχοληθούμε με τα μη-κομμένα (untrimmed) βίντεο, επειδή τα προτεινόμενα TOIs θα μπορούν να εξαιρέσουν background καρέ. Για αυτήν την προσέγγιση, χρησιμοποιούμε  $n = 4K = 60$  anchors για κάθε pixel στους χάρτες ενεργοποίησης του TPN. Έχουμε  $k$  anchors για κάθε διαφορετική διάρκεια anchor (5 κλίμακες των 1, 2, 4, 8, 16, 3 aspect ratios 1:1, 1:2, 2:1 και 4 διάρκειες 16, 12, 8 και 4 καρέ). Σύμφωνα με τους Hou, Chen, and Shah 2017, τα anchors του δικτύου ορίζονται σύμφωνα με τα πιο συνηθισμένα anchors του συνόλου δεδομένων. Αυτό, ωστόσο, δημιουργεί την ανάγκη επανασχεδιασμού του δικτύου για κάθε σύνολο δεδομένων. Στην προσέγγισή μας, χρησιμοποιούμε τα ίδια anchors και για τα δύο σύνολα δεδομένων, επειδή θέλουμε το δίκτυο μας να μην να βασίζεται στο σύνολο δεδομένων που του παρέχεται, αλλά να είναι σε θέση να γενικεύσει για διάφορα σύνολα δεδομένων. Ως διάρκεια δειγματοληψίας, επιλέξαμε 16 καρέ ανά τμήμα βίντεο, επειδή η προ-εκπαιδευμένη έκδοση ResNet

που χρησιμοποιούμε έχει εκπαιδευτεί για βίντεο κλιπ με αυτή τη διάρκεια. Έτσι, η δομή του TPN είναι:

- 1 3D Convolutional Layer με kernel size = 3, stride = 3 και padding = 1
- 1 classification layer που εξάγει  $2n$  scores για το αν υπάρχει ή όχι δράση για  $n$  tubes.
- 1 regression layer που εξάγει  $6n$  διαστάσεις  $(x_1, y_1, t_1, x_2, y_2, t_2)$  για  $n$  tubes.

Η δομή του TPN παρουσιάζεται στην Εικόνα 3.4. Το αποτέλεσμα του TPN είναι τα k-καλύτερα κουτιά, τα οποία είναι τα πιο πιθανά να περιέχουν κάποια δράση.

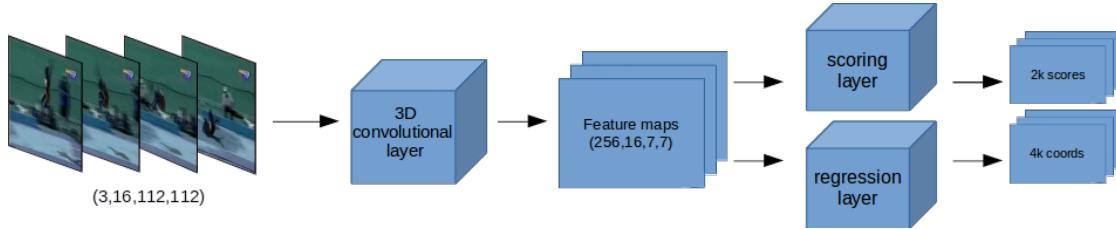
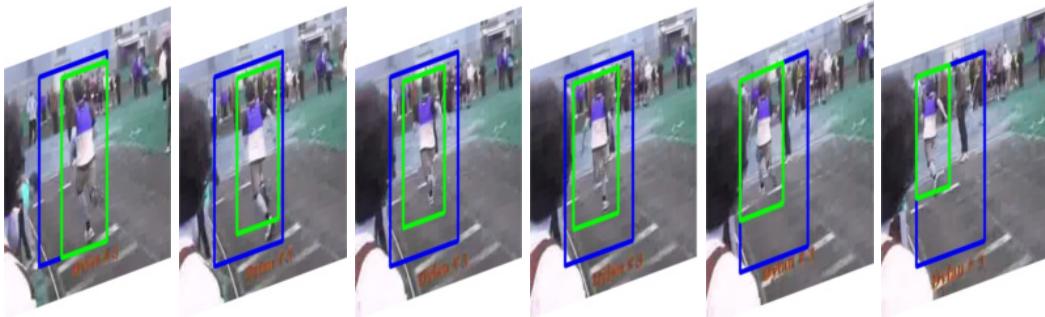


Figure 3.4: Structure of TPN

### 3.4.2 Training

Όπως προαναφέρθηκε, το TPN εξάγει ToIs ως 6-διάστατα διανύσματα. Για το λόγο αυτό, τροποποιήσαμε τα πραγματικά πλάσια ανά καρέ σε πραγματικά Tubes. Θεωρούμε δεδομένο ότι το άτομο που δρα, δεν μπορεί να κινηθεί πολύ σε 16 καρέ, γι' αυτό χρησιμοποιούμε τέτοιου είδους Tubes. Όπως φαίνεται στο σχήμα 3.5, αυτά τα tubes είναι τρισδιάστατα κουτιά που περιλαμβάνουν όλα τα πραγματικά πλαίσια, τα οποία είναι διαφορετικά ανά καρέ.



Σχήμα 3.5: Groundtruth tube is colored with blue and groundtruth rois with color green

Για τη διαδικασία training, για κάθε βίντεο, επιλέγουμε τυχαία ένα μέρος του, το οποίο έχει διάρκεια 16 καρέ. Θεωρούμε ένα anchor ως πρώτο πλάνο, αν η βαθμολογία επικάλυψης του με το πραγματικό tube είναι μεγαλύτερη από 0.5. Διαφορετικά, θεωρείται ως anchor φόντου. Χρησιμοποιούμε έναν scoring layer για να ταξινομήσουμε σωστά αυτά τα anchors και χρησιμοποιούμε την Cross Entropy Loss ως συνάρτηση κόστους (loss function). Έχουμε πολλά anchors για να προτείνουμε μια δράση, αλλά μικρό αριθμό δράσεων σε κάθε βίντεο, έτσι επιλέγουμε

256 anchors συνολικά για κάθε video. Ορίζουμε ότι ο μέγιστος αριθμός των anchors προσκηνίου να είναι 25% από τους 256 anchors και τα υπόλοιπα είναι anchors φόντου.

Η σωστή ταξινόμηση ενός anchor δεν είναι αρκετή για να προτείνουμε ToIs. Είναι, επίσης, απαραίτητο τα anchors να επικαλύπτονται όσο το δυνατόν περισσότερο με τα πραγματικά tubes. Αυτός είναι ο λόγος που χρησιμοποιούμε ένα επίπεδο παλινδρόμησης. Αυτό το layer «κινεί» τον κύβο στην περιοχή που πιστεύεται ότι είναι πιο κοντά στη δράση. Για συνάρτηση κόστους παλινδρόμησης χρησιμοποιούμε την συνάρτηση κόστους smooth-L1 όπως παρουσιάζεται από τους Girshick et al. 2014. Για να υπολογίσουμε τους στόχους παλινδρόμησης, χρησιμοποιούμε την pytorch εφαρμογή του FasterRCNN (Yang et al. 2017) για την παλινδρόμηση του πλαισίου και τροποποιούμε τον κώδικα επεκτείνοντας τον για 3 διαστάσεις. Έτσι έχουμε:

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

όπου τα  $x, y, z, w, h, d$  υποδεικνύουν τις συντεταγμένες του κέντρου του τρισδιάστατου κουτιού καθώς επίσης το πλάτος, το ύψος και τη διάρκειά του. Οι μεταβλητές  $x, x_a$ , και  $x^*$  αφορούν το προβλεπόμενο πλαίσιο, το πλαίσιο του anchor και το πραγματικό πλαίσιο αντίστοιχα (ομοίως για  $y, z, w, h, d$ ). Φυσικά, υπολογίζουμε την απώλεια παλινδρόμησης μόνο για τα anchors προσκηνίου και όχι αυτά του φόντου, συνεπώς στην χειρότερη θα υπολογίσουμε 64 στόχους για κάθε video.

Για να συνοψίσουμε, στη διαδικασία training, εκπαιδεύουμε 2 layers για το TPN, τα scoring και regression. Η συνάρτηση κόστους περιλαμβάνει τα training losses που προκύπτουν από αυτά τα layers και ο τύπος της είναι:

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

όπου:

- $L_{cls}$  είναι η Cross Entropy loss που χρησιμοποιούμε για να εκπαιδεύσουμε τα anchors, με  $p_i$  είναι η προβλεπόμενη κλάση,  $p_i^*$  είναι η πραγματική κλάση και  $p_i, p_i^* \in \{0, 1\}$ .
- $L_{reg}$  είναι η συνάρτηση κόστους smooth-L1, η οποία πολλαπλασιάζεται με  $p_i^*$  προκείμενου να ενεργοποιείται όταν υπάρχει θετικό anchor ( $p_i^* = 1$ ) και να απενεργοποιείται για τα background anchors ( $p_i^* = 0$ ).

### 3.4.3 Validation

Η διαδικασία validation είναι κάπως παρόμοια με τη διαδικασία του training. Επιλέγουμε τυχαία 16 καρέ από ένα βίντεο επικύρωσης, εξετάζουμε αν υπάρχει τουλάχιστον 1 προτεινόμενο ToI που επικαλύπτει  $\geq 0,5$  για κάθε πραγματικό tube και παίρνουμε το recall score. Για να λάβουμε καλές προτάσεις, μετά τη λήψη των classification scores και των regression targets από το αντίστοιχα layers, χρησιμοποιούμε τον αλγόριθμο Non-Maximum Suppression(NMS). Έχουμε ορίσει το κατώφλι του NMS ίσο με 0,7 και κρατάμε τους πρώτους 150 κύβους με τη μεγαλύτερη βαθμολογία.

### 3.4.4 Modified Intersection over Union(mIoU)

Κατά τη διάρκεια του training, έχουμε πολλά anchors. Πρέπει να τα ταξινομήσουμε ως anchors προσκηνίου ή anchors παρασκηνίου. Τα anchors προσκηνίου είναι εκείνα που περιέχουν κάποια

ενέργεια και, αντίστοιχα, του φόντου που δεν έχουν. Το IoU για τα cuboids υπολογίζει το ποσοστό μεταξύ του όγκου της επικάλυψης και του όγκου των Ένωσης. Διαισθητικά, αυτό το χριτήριο είναι καλό για την αξιολόγηση του βαθμού επικάλυψης 2 tube, αλλά έχει ένα μεγάλο μειονέκτημα: Θεωρεί ότι οι διαστάσεις x-y έχουν την ίδια σημασία με τη χρονική διάσταση, το οποίο δεν επιλύμονε. Κι αυτό διότι πρώτον, μας ενδιαφέρει να είμαστε ακριβείς στη χρονική διάσταση, και στη συνέχεια μπορούμε να διορθώσουμε τον τομέα x-y. Ως αποτέλεσμα, αλλάζουμε τον τρόπο με τον οποίο υπολογίζουμε το Intersection over Union. Υπολογίζουμε ξεχωριστά το IoU στις διαστάσεις x-y (IoU-xy) και στην τ διάσταση (IoU-t). Τέλος, πολλαπλασιάζουμε αυτά τα δύο score για να πάρουμε το τελικό IoU. Συνεπώς ο τύπος για 2 tubes  $(x_1, y_1, t_1, x_2, y_2, t_2)$  και  $(x'_1, y'_1, t'_1, x'_2, y'_2, t'_2)$  είναι:

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

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

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

Το παραπάνω χριτήριο μας βοηθά να εξισορροπήσουμε τις επιπτώσεις του χρόνου στο IoU score. Για παράδειγμα, ας εξετάσουμε 2 anchors: a = (22, 41, 1, 34, 70, 5) και b = (20, 45, 2, 32, 72, 5). Αυτά τα 2 anchor στις διαστάσεις x-y έχουν βαθμολογία IoU ίσο με 0,61. Αλλά δεν είναι ακριβώς επικαλυπτόμενα στην διάσταση του χρόνου. Χρησιμοποιώντας την πρώτη προσέγγιση έχουμε 0,5057 IoU βαθμολογία ενώ η δεύτερη προσέγγιση μας δίνει 0,4889. Έτσι, το δεύτερο χριτήριο θα απέρριπτε αυτό το anchor, διότι υπάρχει μια διαφορά στην χρονική διάρκεια.

Για να επιβεβαιώσουμε την ιδέα μας, εκπαιδεύουμε το TPN χρησιμοποιώντας τόσο το IoU χριτήριο όσο και το mIoU για την επικάλυψη των tubes. Στο πίνακα 3.1 μπορούμε να δούμε την απόδοση σε κάθε περίπτωση και για τα δύο σύνολα δεδομένων, JHMDB και UCF. Το recall όριο για αυτή την περίπτωση είναι 0,5 και κατά την διάρκεια του validation χρησιμοποιούμε το κανονικό IoU για να καθορίσουμε αν 2 tubes επικαλύπτονται.

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

Ο πίνακας 3.1 μας δείχνει ότι το τροποποιημένο-IoU μας δίνει ελαφρώς καλύτερη απόδοση recall μόνο στο σύνολο δεδομένων UCF. Αυτό είναι λογικό, επειδή το σύνολο δεδομένων JHMDB χρησιμοποιεί κομμένα βίντεο, συνεπώς η χρονική διάρκεια δεν επηρεάζει πολύ. Έτσι, από τώρα και στο εξής, κατά τη διάρκεια του training χρησιμοποιούμε το mIoU ως επικαλυπτόμενη πολιτική βαθμολογίας.

### 3.4.5 Βελτιώνοντας το TPN score

Μετά την πρώτη δοκιμή, μας ήρθε η ιδέα ότι σε ένα βίντεο που διαρκεί 16 καρέ, στην διάσταση του χρόνου, όλα τα είδη των ενεργειών μπορούν να χωρίζονται στις ακόλουθες κατηγορίες:

1. Η ενέργεια ζεκινά από το n-o πλάισιο και ολοκληρώνεται μετά το 16o καρέ του βίντεο που έχει υποβληθεί σε δειγματοληψία.

2. Η ενέργεια έχει ήδη ξεκινήσει πριν από το 1o χαρέ του βίντεο και τελειώνει στο n πλαίσιο.
3. Η ενέργεια έχει ήδη ξεκινήσει πριν από το 1o χαρέ του βίντεο και ολοκληρώνεται μετά το 16o χαρέ βίντεο.
4. Η ενέργεια ξεκινά και τελειώνει σε αυτά τα 16 χαρέ του βίντεο.

Επιπλέον, παρατηρήσαμε ότι οι περισσότερες ενέργειες, στα σύνολα δεδομένων μας, διαρκούν περισσότερο από 16 χαρέ. Έτσι, ήρθαμε με την ίδεα να προσθέσουμε 1 scoring layer και 1 regression Layer που θα προτείνει ToIs με σταθερή διάρκεια ίση με τη διάρκεια του δείγματος (16 χαρέ) και υπόληπτη τις χωρικές πληροφορίες που παράγονται από τους χάρτες ενεργοποίησης. Η νέα δομή του TPN εμφανίζεται στην εικόνα 3.6. Αφού λάβουμε τις προτάσεις και από και από τα δύο scoling layers, τις ενώνουμε με ποσοστό 1:1 μεταξύ των ToI που εξαχθήκαν από τα δύο υποδίκτυα.

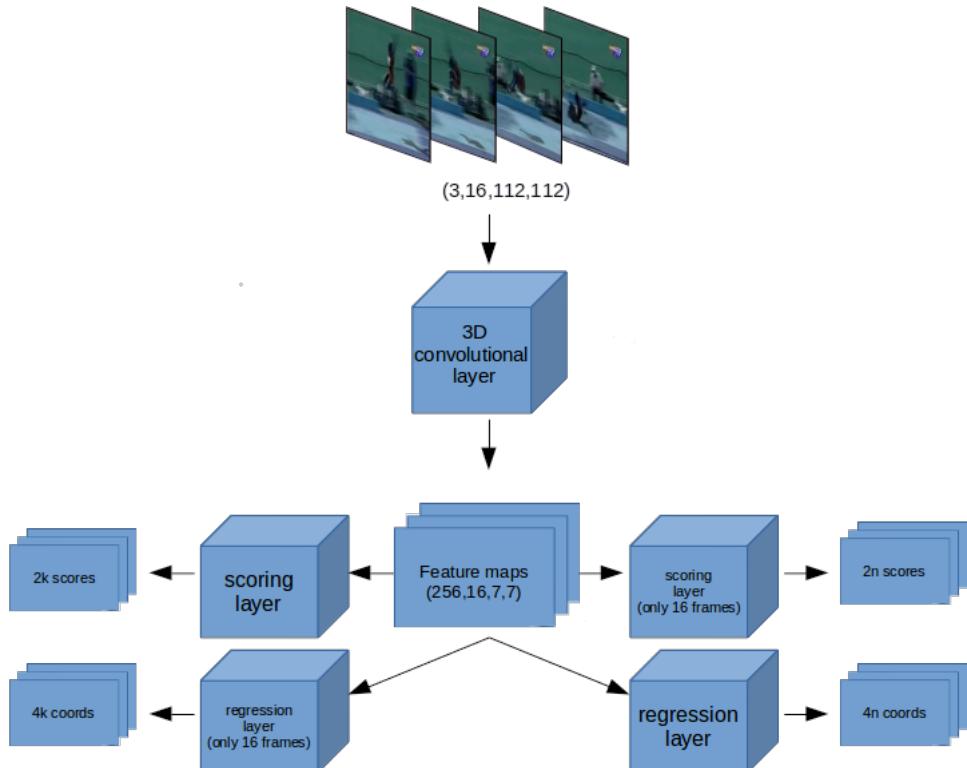


Figure 3.6: TPN structure after adding 2 new layers, where  $k = 5n$ .

Στόχος μας είναι να «συμπέσουμε» τους χάρτες της χρονικής διάστασης, προκειμένου να προτείνουμε ToIs σύμφωνα μόνο με τις χωρικές πληροφορίες. Έτσι, βρίκαμε με 2 τεχνικές για να πραγματοποιήσουμε κάτι τέτοιο:

1. Να χρησιμοποιήσουμε 3D Convolutional Layers με μέγευθος πυρήνα = διάρκεια δείγματος, 1, 1), stride = 1 και χωρίς padding για scoring και regression. Αυτός ο πυρήνας «κοιτάει» μόνο στη χρονική διάσταση των χαρτών ενεργοποίησης και δεν θεωρεί καμία χωρική εξάρτηση.
2. Να υπολογίσουμε τις μέσες τιμές από τη χρονική διάσταση και στη συνέχεια να χρησιμοποιήσουμε ένα 2D Convolutional Layer για scoring και regression.

Οι διαδικασίες training και validation παραμένουν ίδιες. Η μόνη μεγάλη διαφορά είναι ότι τώρα έχουμε κόστη από 2 συστήματα που προτείνουν ToIs. Παράλληλα, κατά την διάρκεια του validation, εμείς αρχικά, ενώνουμε τα προτεινόμενα ToIs και, στη συνέχεια, ακολουθούμε την ίδια διαδικασία, η οποία είναι να υπολογίσουμε το recall. Για το training loss, έχουμε 2 διαφορετικές Cross-entropy losses συναρτήσεις και 2 διαφορετικά smooth-L1 losses. Έτσι, η απώλεια εκπαιδευσης, τώρα, ορίζεται ως:

$$\begin{aligned} L = & \sum_i L_{cls}(p_i, p_i^*) + \sum_i L_{cls}(p_{fixed,i}, p_{fixed,i}^*) + \\ & \sum_i p_i^* L_{reg}(t_i, t_i^*) + \sum_i p_{fixed,i}^* L_{reg}(t_{fixed,i}, t_{fixed,i}^*) \end{aligned} \quad (3.1)$$

όπου:

- $L_{cls}$  είναι η Cross Entropy loss που χρησιμοποιούμε για να εκπαιδεύσουμε τα anchors, με  $p_i$  είναι η προβλεπόμενη κλάση,  $p_i^*$  είναι η πραγματική κλάση και  $p_i, p_i^* \in \{0, 1\}$
- $L_{reg}$  είναι η συνάρτηση κόστους smooth-L1, η οποία πολλαπλασιάζεται με  $p_i^*$  προκείμενου να ενεργοποιείται όταν υπάρχει θετικό anchor ( $p_i^* = 1$ ) και να απενεργοποιείται για τα background anchors ( $p_i^* = 0$ ).
- $p_i$  είναι τα anchors από τα scoring και regression layers με μεταβλητή χρονική διάρκεια και  $p_i^*$  είναι η αντίστοιχη πραγματική τους κλάση.
- $p_{fixed,i}$  είναι τα anchors από τα scoring και regression layers με σταθερή χρονική διάρκεια ίση με 16 καρέ και  $p_{fixed,i}^*$  είναι η αντίστοιχη πραγματική του κλάση.

Εκπαιδεύουμε το δίκτυο TPN χρησιμοποιώντας και τις δύο τεχνικές και η απόδοση του recall εμφανίζεται στον πίνακα 3.2.

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

Όπως μπορούμε να δούμε από τα προηγούμενα αποτελέσματα, τα νέα layers αύξησαν σημαντικά την απόδοση του recall. Πέρα από αυτό, ο πίνακας 3.2 δείχνει ότι η λήψη των μέσων τιμών από τη χρονική διάσταση μας δίνει τα καλύτερα αποτελέσματα.

### 3.4.6 Προσθήκη Regressor

Το αποτέλεσμα του TPN είναι τα α-υψηλότερα βαθμολογικά anchors που μετακινήθηκαν σύμφωνα με την regression πρόβλεψη τους. Μετά από αυτό, πρέπει να μετατρέψουμε τα προτεινόμενα anchors σε ToIs. Για να γίνει αυτό, προσθέτουμε ένα σύστημα παλινδρόμησης που παίρνει ως είσοδο τους χάρτες χαρακτηριστικών των τρισδιάστατων κουτιών και επιστρέφει μια ακολουθία από

δισδιάσταστα κουτιά, ένα για κάθε χαρέ. Το μόνο πρόβλημα είναι ότι η παλινδρόμηση χρειάζεται ως είσοδο χάρτες χαρακτηριστικών σταύρου μεγέθους. Αυτό το πρόβλημα είναι ήδη λυμένο από τα R-CNNs που χρησιμοποιούν ROI pooling και ROI align προκειμένου να λάβουν σταύρου μεγέθους χάρτες ενεργοποίησης από ROIs μεταβαλλόμενου μεγέθους. Στην περίπτωση μας, επεκτείνουμε την λειτουργία ROI Align, που παρουσιάζεται από το MaskR-CNN(He et al. 2017), και εμείς το ονομάζουμε **3D ROI align**.

**3D Roi Align** Το 3D ROI align είναι μια τροποποίηση του ROI Align που παρουσιάστηκε από το MaskR-CNN. Η κύρια διαφορά μεταξύ αυτών των δύο είναι ότι το MaskR-CNN, στο RoiAlign, χρησιμοποιεί διγραμμική παρεμβολή για την εξαγωγή των χαρακτηριστικών των ROIs και το δικός μας 3D ROI Align χρησιμοποιεί τριγραμμική παρεμβολή για τον ίδιο λόγο. Και πάλι, η 3η διάσταση είναι χρόνος. Συνεπώς, έχουμε ως είσοδο ένα χάρτη χαρακτηριστικών που εξάγεται από το ResNet34 με διαστάσεις (64, 16, 28, 28) και έναν τένσορα που περιέχει τα προτεινόμενα ToIs. Για κάθε ToI του οποίου ο χάρτης ενεργοποίησης έχει μέγεθος ίσο με (64, 16, 7, 7), έχουμε ως έξοδο ένα χάρτη χαρακτηριστικών με μέγεθος (64, 16, 7, 7).

### Regression procedure

Στην αρχή, για κάθε προτεινόμενο ToI, έχουμε τους αντίστοιχους χάρτες ενεργοποίησης χρησιμοποιώντας 3D ROI align. Αυτά τα χαρακτηριστικά δίνονται ως είσοδο σε έναν Regressor. Αυτός επιστρέφει  $16 \cdot 4$  προβλεπόμενες μεταβολές ( $\delta_x, \delta_y, \delta_w, \delta_h$ ), 4 για κάθε χαρέ, όπου  $\delta_x, \delta_y$  καθορίζουν τις συντεταγμένες του κέντρου των προτάσεων και  $\delta_w, \delta_h$  το πλάτος και το ύψος του, όπως ορίζεται από τους Girshick et al. 2014. Κρατάμε μόνο τις προβλεπόμενες μεταβολές, για τα χαρέ που  $\geq t_1$  και  $< t_2$  και για υπόλοιπα θέτουμε ένα μηδενικό 2D κουτί. Μετά από αυτό, τροποποιούμε κάθε anchor, γραμμένο ως έναν κύβο δηλαδή γραμμένο ως  $(x_1, y_1, t_1, x_2, y_2, t_2)$  σε μια ακολουθία πλαισίων 2D, όπως:

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

**Training** Για να εκπαιδεύσουμε τον Regressor μας, ακολουθούμε τα ίδια βήματα που ακολουθήσαμε προηγουμένως για την προηγούμενη διαδικασία εκπαίδευσης του TPN. Αυτό σημαίνει ότι επιλέγουμε τυχαία 16 ToIs από αυτές που προτείνονται από το scoring layer του TPN. Απ' αυτά, 4 είναι τα anchors προσκηνίου, το οποίο σημαίνει ότι αποτελούν το 25% του συνολικού αριθμού των anchors όπως συνέβη προηγουμένως. Εξάγουμε τα αντίστοιχα χαρακτηριστικά τους χρησιμοποιώντας 3D ROI Align και υπολογίζουμε τους στόχους τους, όπως κάναμε για το regression layer. Τροφοδοτούμε το δίκτυο μας με αυτά τα χαρακτηριστικά και συγχρίνουμε τους προβλεπόμενους στόχους με τούς αναμενόμενους. Ξανά πάλι, χρησιμοποιούμε smooth-L1 loss function για τη συνάρτηση κόστους, υπολογίζοντας την μόνο για ToIs που είναι στο προσκήνιο. Έτσι, προσθέτουμε μια άλλη παράμετρο στο φόρμουλα απώλειας εκπαίδευσης που πλέον ορίζεται ως:

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

όπου εκτός από τις παραμέτρους που καθορίστηκαν προηγουμένως, ορίζουμε  $c_i$  ως στόχους παλινδρόμησης για τα επιλεγμένα tubes  $q_I$ . Αυτά τα tubes είναι που επιλέγονται τυχαία από τα προτεινόμενα ToIs και  $q_i^*$  είναι τα αντίστοιχα πραγματικά tubes, τα οποία είναι τα πλησιέστερα σε κάθε  $q_i$  tube. Και πάλι χρησιμοποιούμε το  $q_i^*$  ως παράγοντα, επειδή θεωρούμε ένα tube ως φόντο όταν δεν επικαλύπτεται με οποιοδήποτε πραγματικό tube περισσότερο από 0,5.

**Validation** Χρησιμοποιούμε, ξανά, τη μετρική recall για να αξιολογήσουμε την απόδοση του παλινδρομητή. Υπολογίζουμε 3 επιδόσεις recall:

**Cuboid Recall**, που είναι η recall μετρική για τα προτεινόμενα τρισδιάστατα κουτιά.

Ενδιαφερόμαστε για αυτήν την επίδοση γιατί, θέλουμε να μάθουμε πόσο καλές είναι οι προτάσεις μας πριν τις τροποποιήσουμε σε σειρές κουτιών.

**Single frame Recall**, η οποία είναι η επίδοση recall για τις προτεινόμενες ακολουθίες κουτιών σε σχέση με τις πραγματικές.

**Follow-up Single Frame Recall**, που είναι η απόδοση της ανάκλησης μόνο για τα τρισδιάστατα

Κουτιά που ήταν πάνω από το όριο επικάλυψης μεταξύ των προτεινόμενων κύβων και των πραγματικών κύβων. Χρησιμοποιούμε αυτή τη μετρική για να γνωρίζουμε πόσα από τους προτεινόμενα τρισδιάστατα κουτιά κατέληξαν να είναι καλές προτάσεις.

### Αρχιτεκτονικές για τον Regressor

Σχεδιάσαμε 2 προσεγγίσεις για την υλοποίηση του Regressor. Αυτές απεικονίζονται στις Εικόνες 3.7 και 3.8, με την πρώτη προσέγγιση να αποτελείται από ένα 3D Convolutional Layer σε αντίθεση με την δεύτερη προσέγγιση που έχει ένα 2D Convolutional Layer.

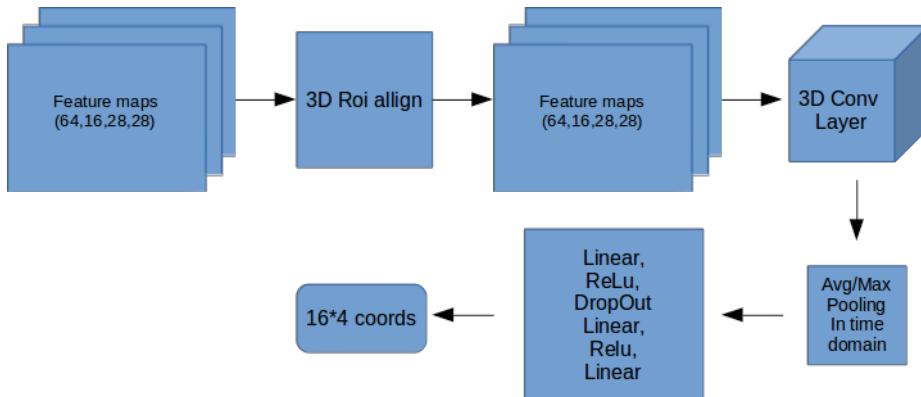


Figure 3.7: Structure of Regressor

Η διαδικασία που ακολουθούν και οι δύο Regressors περιγράφεται παρακάτω:

1. Αρχικά εξάγουμε τα αντίστοιχα feature maps για κάθε ToI χρησιμοποιώντας 3D Roi Align, και ακολούθως τα κανονικοποιούμε. Μετά, στην πρώτη προσέγγιση τροφοδοτούμε ένα 3D Convolutional Layer με kernel ίσο με 1, stride ίσο με 1 και χωρίς padding. Στην συνέχεια, εφαρμόζουμε μια pooling διαδικασία στην διάσταση του χρόνου (είτε avg είτε max pooling). Απ' την άλλη, στην δεύτερη προσέγγιση, πρώτα εφαρμόζουμε avg/max pooling στην διάσταση του χρόνου και στην συνέχεια τροφοδοτούμε ένα 2D Convolutional Layer με kernel = 1, stride = 1 και χωρίς padding.

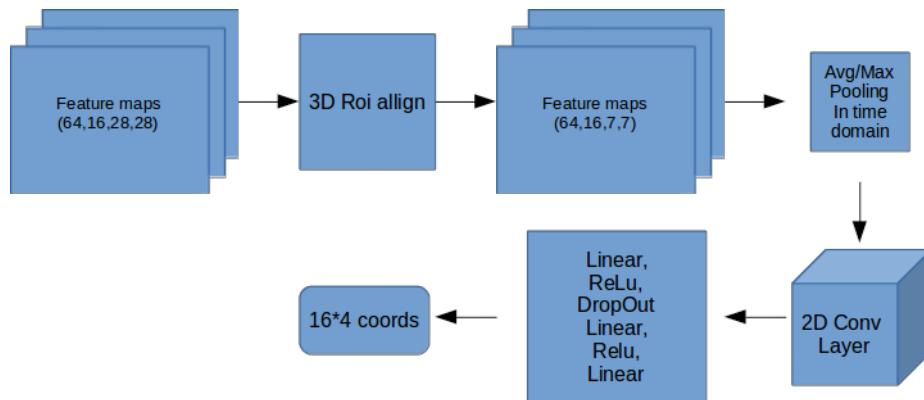


Figure 3.8: Structure of Regressor

2. Και στις δύο περιπτώσεις λαμβάνουμε ως έξοδο ένα feature map με διαστάσεις (64,7,7) το οποίο περνάμε διαδοχικά από 1 γραμμικό layer, 1 Relu Layer, 1 Dropout Layer, άλλο ένα γραμμικό Layer, άλλο ένα ReLu layer και ένα τελικό γραμμικό layer. Το τελευταίο layer μας δίνει ως έξοδο 64 στόχους,  $4 \cdot 16$  μετακινήσεις.

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 converting cuboids into sequences of frames

Οι πίνακες 3.3 και 3.4 περιέχουν τα αποτελέσματα για την πρώτη και δεύτερη προσέγγιση. Μάλιστα, για την δεύτερη προσέγγιση ελέγχαμε 3 διαφορετικούς χάρτες χαρακτηριστικών, ενώ και στις 2 περιπτώσεις πειραματιστήκαμε χρησιμοποιώντας και τις 2 προαναφερθέντες pooling μεθόδους για τα σύνολα δεδομένων JHMDB και UCF-101. Παρατηρούμε ότι, με βάση τα παραπάνω αποτελέσματα, λαμβάνουμε σχετικά χαμηλή recall απόδοση για το dataset UCF ενώ για το JHMDB τα αποτελέσματα είναι κάπως καλύτερα. Πιο συγκεκριμένα, με βάση την πρώτη προσέγγιση λαμβάνουμε τελικά recall απόδοση ίση με 76-77% για το JHMDB και 46-50% για το UCF. Με βάση την δεύτερη προσέγγιση, στην καλύτερη περίπτωση λαμβάνουμε 80% απόδοση recall για το JHMDB ενώ για το UCF παραμένουμε στα ίδια περίπου αποτελέσματα. Παράλληλα παρατηρούμε ότι χάνουμε περίπου 30-40% από τις καλές cuboid προτάσεις και στις 2 περιπτώσεις, το οποίο αποτελεί μεγάλο πρόβλημα και των δύο προσεγγίσεων. Όλ' αυτά μας κάνουν να ξανασκεφτούμε τον τρόπο που σχεδιάσαμε το TPN και μας οδήγησε στο να σχεδιάσουμε ένα νέο μοντέλο.

### 3.5 Τα τρισδιάστατα anchors ως 4k διανύσματα

Σε αυτή την προσέγγιση, ορίζουμε τους 3D anchors ως διανύσματα με 4k συντεταγμένες ( $k = 16$  καρέ = διάρκεια δείγματος). Έτσι ένα τυπικό anchor γράφεται ως  $(x_1, y_1, x'_1, y'_1, x_2, y_2, \dots)$  όπου  $x_1, y_1, x'_1, y'_1$  είναι οι συντεταγμένες για 1o καρέ,  $x_2, y_2, x'_2, y'_2$  για το 2o καρέ κλπ, όπως παρουσιάστηκε από τους Girdhar et al. 2018. Η εικόνα 3.9 απεικονίζει ένα τέτοιου τύπου anchor.

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: Recall performance using 3 different feature maps as Regressor's input and 2 pooling methods

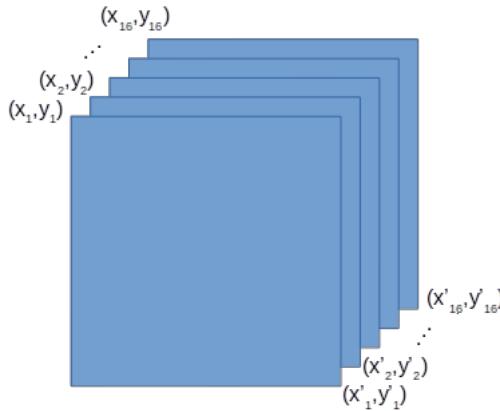


Figure 3.9: An example of the anchor  $(x_1, y_1, x'_1, y'_1, x_2, y_2, \dots)$

Το κύριο πλεονέκτημα αυτής της προσέγγισης είναι ότι δεν χρειάζεται να μεταφράσουμε τα 3D anchors σε δισδιάστατα κουτιά, γεγονός που προκαλεί πολλά προβλήματα στην προηγούμενη προσέγγιση. Ωστόσο, αυτή η προσέγγιση έχει ένα μεγάλο μειονέκτημα, το οποίο είναι το γεγονός ότι αυτός ο τύπος anchor έχει σταθερή χρονική διάρκεια. Για να αντιμετωπίσουμε αυτό το πρόβλημα, έχουμε ορίσει anchors με διαφορετικές χρονικές διάρκειες, οι οποίες είναι 16, 12, 8 και 4 καρέ. Anchors με διάρκεια < διάρκεια του δείγματος (16 καρέ) μπορούν να γραφτούν ως διάνυσμα 4k με μηδενικές συντεταγμένες στα καρέ μεγαλύτερα από τη χρονική διάρκεια. Για παράδειγμα, ένα anchor με 2 καρέ διάρκεια, ξεκινώντας από το 2o καρέ και τερματίζοντας στον 3o μπορεί να γραφεί ως  $(0, 0, 0, 0, x_1, y_1, x'_1, y'_1, x_2, y_2, x'_2, y'_2, 0, 0, 0, 0)$  εάνη διάρκεια δείγματος είναι 4 καρέ.

Αυτή η νέα προσέγγιση μας οδήγησε στο να αλλάξουμε την δομή του TPN. Η νέο δομή του απεικονίζεται στην εικόνα 3.10. Όπως μπορούμε να δούμε, προσθέσαμε scoring και regression layers για κάθε διάρκεια. Έτσι, το TPN ακολουθεί τα επόμενα βήματα για να παράγει ToIs.

- Στην αρχή, τροφοδοτούμε τον χάρτη χαρακτηριστικών, που εξάγεται από το 3D ResNet34, ως είσοδο σε ένα 3D Convolutional Layer με μέγενθος πυρήνα = 1, stride = 1 και χωρίς padding.

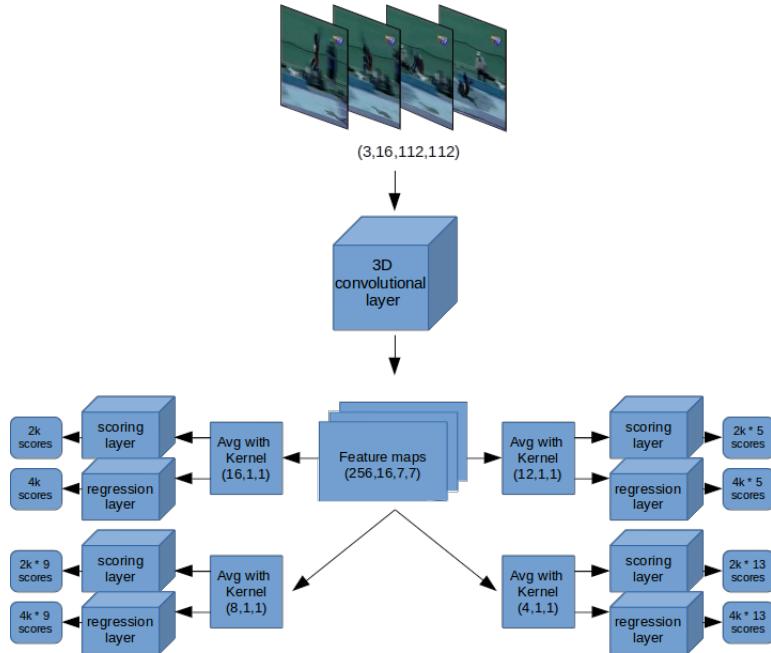


Figure 3.10: The structure of TPN according to new approach

2. Από το Convolutional Layer, έχουμε ως έξοδο ένα χάρτη ενεργοποίησης με διαστάσεις (256, 16, 7, 7). Για τη μείωση της χρονικής διάστασης, χρησιμοποιούμε 4 pooling layers, ένα για κάθε δείγμα διάρκειας με μεγέθυνη πυρήνα (16, 1, 1), (12, 1, 1), (8, 1, 1) και (4, 1, 1) και stride = 1, για τη διάρκεια του δείγματος 16, 12, 8 και 4 καρέ αντίστοιχα. Έτσι, έχουμε χάρτες ενεργοποίησης με διαστάσεις (256, 1, 7, 7), (256, 5, 7, 7), (256, 9, 7, 7) και (256, 13, 7, 7), στης οποίες η δεύτερη διάσταση είναι ο αριθμός των πιθανών χρονικών διαχυμάνσεων. Για παρόδειγμα, σε χάρτη χαρακτηριστικών με διαστάσεις (256, 5, 7, 7), το οποίο σχετίζεται με anchors με διάρκεια 12 καρέ, μπορούμε να έχουμε 5 πιθανές περιπτώσεις, από το καρέ 0 μέχρι το καρέ 11, από το καρέ 1 μέχρι το καρέ 12 κλπ.
3. Ξανά, όπως και στην προηγούμενη προσέγγιση, για κάθε pixel του χάρτη ενεργοποίησης αντιστοιχούμε  $n = k = 15$  anchors (5 κλίμακες από 1, 2, 4, 8, 16, και 3 aspect ratios 1:1, 1:2, 2:1). Φυσικά, έχουμε 4 διαφορετικούς χάρτες ενεργοποίησης, με 1, 5, 9 και 13 διαφορετικές περιπτώσεις και  $7 \times 7$  διαστάσεις σε κάθε φίλτρο. Έτσι, συνολικά έχουμε  $28 \cdot 15 \cdot 49 = 20580$  διαφορετικά anchors. Αντίστοιχα, έχουμε 20580 διαφορετικούς στόχους regression.

### 3.5.1 Training

Η διαδικασία training παραμένει σχεδόν η ίδια όπως και στην προηγούμενη προσέγγιση. Έτσι, και πάλι, εμείς τυχαία επιλέγουμε ένα τμήμα βίντεο και τα αντίστοιχα πραγματικά tubes. Όμως, θεωρούμε τα anchors ως προσκήνιο όταν έχουν επικάλυψη μεγαλύτερη από 0,8 με οποιαδήποτε πραγματικό tube, ενώ θεωρούμε anchors παρασκήνιου αυτά των οποίων η επικάλυψη είναι μεγαλύτερη που 0,1 και μικρότερη από 0,3. Δεν ασχολούμαστε με τα υπόλοιπα anchors.

Όπως δείχνει ο πίνακας 3.5, είναι προφανές ότι έχουμε καλύτερες επιδόσεις recall σε σύγκριση με την προηγούμενη προσέγγιση. Επιπλέον, μπορούμε να δούμε ότι το 3D max pooling αποδίδει

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 using 2nd approach for anchors

καλύτερα από το 3D avg pooling. Η διαφορά μεταξύ των δύο είναι περίπου 10%, η οποία είναι αρκετά μεγάλη για να μας κάνει να επιλέξουμε το max pooling ως λειτουργία ομαδοποίησης πριν από τη λήψη των scores και regression targets των anchors.

### 3.5.2 Προσθήκη Regressor

Ακόμα και αν, το TPN μας εξάγει κουτιά σε επίπεδο καρέ, πρέπει να βελτιώσουμε αυτές τις προβλέψεις προκειμένου να αλληλοεπικαλύπτονται με τα πραγματικά κουτιά όσο το δυνατόν καλύτερα. Έτσι, σε πλήρη αντιστοιχία με την προηγούμενη προσέγγιση, προσθέσαμε έναν Regressor για να προσταθήσουμε να πάρουμε καλύτερα αποτελέσματα recall.

**3D Roi align** Σε αυτή την προσέγγιση, γνωρίζουμε ήδη τις χωρικές συντεταγμένες. Έτσι, μπορούμε να χρησιμοποιήσουμε τη μέθοδο που προτείνεται από τους Girdhar et al. 2018. Αποτελεί επέκταση του RoiAlign χωρίζοντας το tube σε  $T$  δισδιάστατα κουτιά. Στη συνέχεια, χρησιμοποιείται το κλασικό RoiAlign για να εξαχθεί μια περιοχή από κάθε μίας από τα χρονικά slices στον χάρτη ενεργοποίησης. Μετά από αυτό, τα feature maps που προέκυψαν μέσω του RoiAlign συνδέονται στην διάσταση του χρόνου, ώστε να προκύψει χάρτης χαρακτηριστικών με διαστάσεις  $T \times R \times R$ , όπου  $R$  είναι η ανάλυση εξόδου του RoiAlign, το οποίο είναι 7 στην περίπτωσή μας.

Εκπαιδεύουμε τον Regressor μας χρησιμοποιώντας την ίδια loss function όπως ο τύπος της προηγούμενης προσέγγισης που είναι:

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

Και σ' αυτήν την προσέγγιση χρησιμοποιούμε 2 διαφορετικές αρχιτεκτονικές για την υλοποίηση του Regressor. Ως πρώτη προσέγγιση, χρησιμοποιούμε ένα 3D Convolutional Layer ακολουθούμενο από 2 γραμμικά Layer. Αντίστοιχα, στην δεύτερη προσέγγιση χρησιμοποιούμε ένα 2D Convolution Layer ακολουθούμενο, και αυτό, από 2 γραμμικά Layer. Η πρώτη προσέγγιση είναι ακριβώς η ίδια με πριν. Ωστόσο, η δεύτερη προσέγγιση αντιμετωπίζει τα feature maps σαν να μην υπάρχουν χρονικές εξαρτήσεις μεταξύ τους. Δηλαδή:

1. Στην αρχή, χρησιμοποιούμε το 3D Roi Align για να εξάγουμε τα feature maps. Και μετά τα κανονικοποιούμε. Έστω λοιπόν ότι προκύπτουν  $k$  χάρτες ενεργοποίησης με διαστάσεις  $(k, 256, 16, 7, 7)$
2. Χωρίζουμε τα υποψήφια ToIs σε  $T$  2D κουτιά, οπότε οι διαστάσεις του τένσορα που περιέχει τις συντεταγμένες των ToIs γίνονται από  $(k, 4 \cdot sampleduration, 4)$ .

Διαφοροποιούμε τον τένσορα ώστε να πάρει τις διαστάσεις  $(k \cdot sampleduration, 4)$ , όπου οι πρώτες  $k$  συντεταγμένες αναφέρονται στο πρώτο χαρέ, οι επόμενες  $k$  στο δεύτερο χλπ.

3. Αντίστοιχα διαφοροποιούμε και τους χάρτες ενεργοποίησης όπου από διαστάσεις  $(k, 64, sampleduration, 7, 7)$  καταλήγουμε σε χάρτες με διαστάσεις  $(k \cdot sampleduration, 64, 7, 7)$ . Πλέον λοιπόν επεξεργαζόμαστε τους χάρτες χαρακτηριστικών σαν να είναι δισδιάστατοι. Έτσι τροφοδοτούμε το 2D Convolutional Layer και ακολουθούμενο από τα άλλα δύο γραμμικά Layer.
4. Τα εξαγόμενα targets, φυσικά θα είναι μόνο 4 όσο δηλαδή για 1 χαρέ.

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: Recall performance when using a 3D Convolutional Layer in Regressor's architecture

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: Recall performance when using a 2D Convolutional Layer instead of 3D in Regressor's model

Με βάση τους παραπάνω πίνακες 3.6 και 3.7, η καλύτερη επίδοση προκύπτει για τους χάρτες χαρακτηριστικών μεγέθους  $(64, 16, 7, 7)$ . Αυτό το αποτέλεσμα ήταν αναμενόμενο γιατί αυτοί οι χάρτες ενεργοποίησης βρίσκονται «πιο κοντά» στα πραγματικά χαρακτηριστικά. Συγχρίνοντας τις δύο προτεινόμενες μεθόδους παρατηρούμε ότι η δεύτερη, αυτή δηλαδή που χρησιμοποιεί 2D Convolutional Layer έχει τα καλύτερα αποτελέσματα. Αν και βελτιώσαμε την απόδοση του TPN ακόμα δεν έχουμε καταφέρει να έχουμε σχεδόν σε όλες τις περιπτώσεις καλές προτάσεις tube.

### 3.5.3 Μείωση της διάρκειας του δείγματος

Προκειμένου να πετύχουμε καλύτερα αποτελέσματα αποφασίσαμε να μειώσουμε την διάρκεια του δείγματος από τα 16 χαρέ σε 8 και 4 αντίστοιχα. Κι αυτό γιατί με αυτόν τον τρόπο θα μειωθούν παράλληλα με τον αριθμό τους οι διαστάσεις των anchors, άρα και οι διαστάσεις των στόχων παλινδρόμησης και γενικά ο αριθμός των παραμέτρων που πρέπει να εκπαιδευτούν από το σύστημα.

Εκπαιδεύουμε το TPN με και χωρίς Regressor προκειμένου να βρούμε την κατάλληλη αρχιτεκτονική. Τα αποτελέσματα χωρίς Regressor παρουσιάζονται στον πίνακα 3.8 ενώ τα αποτελέσματα της αρχιτεκτονικής με Regressor στον πίνακα 3.9.

Το πρώτο συμπέρασμα που προκύπτει από τους πίνακες 3.8 και 3.9 είναι ότι όντως μειώνοντας την διάρκεια του δείγματος πετυχαίνουμε καλύτερα αποτελέσματα. Παράλληλα, με βάση τον πίνακα 3.9 και δεδομένων των προηγούμενων αποτελεσμάτων (Πίνακες 3.6 και 3.7) γίνεται ξεκάθαρο ότι η

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: Recall results when reducing sample duration to 4 and 8 frames per video segment

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: Recall results when a regressor and sample duration equal with 4 or 8 frames per video segment

προσέγγιση που περιλαμβάνει ένα 2D Convolutional Layer υπερέχει αυτής με το 3D Convolutional Layer. Σ' οτι αφορά τα σύνολα δεδομένων παρατηρούμε ότι η προσέγγιση με διάρκεια δείγματος 8 καρέ υπερέχει σχεδόν σε όλες τις περιπτώσεις. Συνεπώς, για τα επόμενα κεφάλαια θα προτιμάται να χρησιμοποιείται αυτή.



## Κεφάλαιο 4

# Αλγόριθμος σύνδεσης των action tubes

Στο προηγούμενο κεφάλαιο περιγράψαμε μεθόδους για την παραγωγή υποψήφιων ToIs, δεδομένου ενός μικρού τμήματος βίντεο που διαρκεί 8 ή 16 χαρέ. Ωστόσο, τα πραγματικά βίντεο και πραγματικές ανθρώπινες ενέργειες, σε εξωτερικές συνθήκες, διαφορούν πάνω από 16 χαρέ τις περισσότερες φορές. Τα τρέχοντα δίκτυα δεν είναι σε θέση να επεξεργαστούν ένα ολόκληρο βίντεο με την μία, προκειμένου να προτείνει υποψήφια ToIs, λόγω προβλημάτων μνήμης και υπολογιστικής ενέργειας. Πολλές προσεγγίσεις για εντοπισμό δράσης λύνουν αυτό το πρόβλημα, δεδομένου ενός βίντεο, είτε προτείνουν υποψήφιες περιοχές σε επίπεδο χαρέ και, στη συνέχεια, τις συνδέουν με σκοπό τη δημιουργία υποψήφιων action tubes, είτε το διαχωρίζουν σε τμήματα βίντεο, προτείνοντας ακολουθίες από δισδιάστα πλαίσια τα οποία στην συνέχεια συνδέουν για να δημιουργήσουν action proposals. Και οι δύο προαναφερθείσες προσεγγίσεις καθιστούν την κατάλληλη επιλογή της μεθόδου σύνδεσης σημαντικό παράγοντα για την απόδοση του δικτύου. Αυτό συμβαίνει επειδή, παρόλο που στο επίπεδο χαρέ ή στο επίπεδο τμήματος βίντεο οι προτάσεις μπορεί να είναι πολύ καλές, αν ο προτεινόμενος αλγόριθμος σύνδεσης δεν λειτουργεί καλά, οι τελικές προτάσεις action tubes δεν θα είναι αποτελεσματικές, οπότε το τελικό μοντέλο δεν θα είναι σε θέση να επιτύχει υψηλή απόδοση ταξινόμησης. Με άλλα λόγια, αν ο αλγόριθμος σύνδεσης δεν δημιουργεί προτάσεις δράσης με μεγάλο recall και καλή απόδοση MABO, ο ταξινομητής του μοντέλου δεν θα είναι σε θέση να εκτελέσει την κατάλληλη ταξινόμηση, επειδή πιθανώς θα του έχουν δοθεί action tubes χωρίς κανένα περιεχόμενο. Σε αυτό το κεφάλαιο, παρουσιάζουμε 3 διαφορετικές προσεγγίσεις που χρησιμοποιούνται για τη σύνδεση των προτεινόμενων ToIs που παράγονται από το TPN του προηγούμενου κεφαλαίου.

### 4.1 Πρώτη προσέγγιση: συνδυασμός επικάλυψης και πιθανότητας δράσης

Ο αλγόριθμος μας εμπνέεται από την προσέγγιση των Hou, Chen, and Shah 2017, η οποία υπολογίζει όλες τις πιθανές ακολουθίες των ToIs. Για να βρει τα καλύτερα υποψήφια action tubes, χρησιμοποιεί μια βαθμολογία που μας λέει πόσο πιθανό μια ακολουθία του TOIs είναι να περιέχει μια ενέργεια. Αυτή η βαθμολογία είναι ένας συνδυασμός 2 μετρικών:

**Πιθανότητα δράσης ή Δραστικότητα (Actionness),** που είναι η πιθανότητα ενός ToI να περιέχει μια δράση. Αυτό το σκορ παράγεται από τα scoring layers του TPN.

Σκορ επικάλυψης μεταξύ των ToIs, το οποίο είναι το IoU των τελευταίων πλαισίων του πρώτου ToI και των πρώτων πλαισίων του δεύτερου ToIs.

Η παραπάνω πολιτική βαθμολόγησης μπορεί να περιγραφεί από τον ακόλουθο τύπο:

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

Για κάθε πιθανό συνδυασμό ToIs, υπολογίζουμε το σκορ του όπως φαίνεται στην εικόνα 4.1.

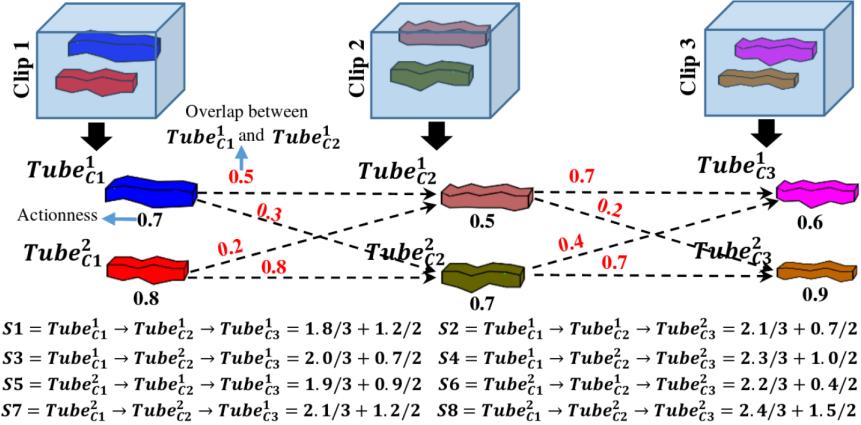


Figure 4.1: An example of calculating connection score for 3 random TOIs taken from Hou, Chen, and Shah 2017

Η παραπάνω προσέγγιση, όμως, χρειάζεται υπερβολικά πολύ μνήμη για την πραγματοποίηση όλων αυτών των υπολογισμών, έτσι ένα πρόβλημα μνήμης εμφανίζεται. Ο λόγος είναι πως για κάθε νέο βίντεο κλιπ, εμείς προτείνουμε  $k$  ToIs (16 κατά την διάρκεια της εκπαίδευσης και 150 κατά την διάρκεια του validation. Σαν αποτέλεσμα, για ένα μικρό βίντεο χωρίσμενο σε 10 μέρη, χρειάζεται να υπολογίσουμε τουλάχιστον  $150^{10}$  σκορ κατά την διάρκεια της επικύρωσης. Αυτό οδηγεί το σύστημα μας να χρειάζεται υπερβολικά πολύ χρόνο για να το πραγματοποιήσει.

Για να αντιμετωπίσουμε αυτό το πρόβλημα, δημιουργούμε έναν άπληστο αλγόριθμο για να βρούμε τα υποψήφια action tubes. Ο αλγόριθμος αυτός, για κάθε νέο τμήμα βίντεο, κρατά τα tubes με βαθμολογία υψηλότερη από ένα κατώφλι και διαγράφει τα υπόλοιπα. Έτσι, δεν χρειάζεται να υπολογίσουμε συνδυασμούς με πολύ χαμηλό σκορ. Γράφαμε κώδικα για τον υπολογισμό των βαθμολογιών των tubes στη γλώσσα CUDA, η οποία έχει ως δυνατότητα την παράλληλη επεξεργασία του ίδιου κώδικα χρησιμοποιώντας διαφορετικά δεδομένα. Ο αλγόριθμος μας περιγράφεται παρακάτω:

- Πρώτον, αρχικοποιούμε κενές λίστες για τα τελικά tubes, την διάρκεια τους, τις βαθμολογίες τους, τα ενεργά tubes, τη διάρκεια τους, το άθροισμα των σκορ επικάλυψης και δραστικότητας τους όταν:

- Η λίστα με τα τελικά tubes περιέχει όλα τα tubes που είναι πιθανότερο να περιέχουν μια ενέργεια και η λίστα βαθμολογίας τους περιέχει τις αντίστοιχες βαθμολογίες τους. Αναφερόμαστε σε κάθε tube από τον δείκτη του, ο οποίος σχετίζεται με ένα τένσορα, στον οποίο σώσαμε όλα τα ToIs που προτείνονται από το TPN για κάθε τμήμα βίντεο.

- Η λίστα ενεργών tubes περιέχει όλα τα tubes που θα συνδυαστούν με τα νέα ToIs. Η λίστα άνθροισης των επικαλυπτόμενων σκορ και η λίστα άνθροισης δραστικότητας περιέχουν τα αντίστοιχα ανθροίσματα τους, προκειμένου να αποφεύγεται ο υπολογισμός τους για κάθε βρόχο.

Επίσης, ορίζουμε αρχικά το όριο σύνδεσης ίσο με 0,5.

2. Για το πρώτο τμήμα βίντεο, προσθέτουμε όλα τα ToIs τόσο στα ενεργά tubes όσο και στα τελικά tubes. Οι βαθμολογίες τους είναι μόνο η δική τους δραστικότητα επειδή δεν υπάρχουν tubes για τον υπολογισμό της μεταξύ τους επικαλυπτόμενης βαθμολογίας. Έτσι, έτσι ορίζουμε το άνθροισμα επικάλυψης ίσο με 0.
3. Για κάθε επόμενο βίντεο, μετά την λήψη των προτεινόμενων ToIs, πρώτα υπολογίζουμε το σκορ επικάλυψης τους με κάθε ενεργό tube. Μετά, αδειάζουμε την λίστα με τα ενεργά tubes, με την διάρκεια τους, το άνθροισμα επικάλυψης και το άνθροισμα πιθανοτήτων δράσης. Για κάθε νέο tube που έχει βαθμολογία υψηλότερη από το κατώφλι σύνδεσης προσθέτουμε τόσο στα τελικά action tubes όσο και στα ενεργά, στις αντίστοιχες λίστες και αυξάνουμε τη διάρκειά τους.
4. Εάν ο αριθμός των ενεργών tubes είναι μεγαλύτερος από ένα κατώτατο όριο, ορίζουμε το όριο σύνδεσης ίσο με τη βαθμολογία του 100ου καλύτερου tube. Πέραν αυτού, ενημερώνουμε την τελική λίστα των tubes, αφαιρώντας όλα τα tubes που έχουν σκορ χαμηλότερο από το κατώφλι σύνδεσης.
5. Μετά από αυτό, προσθέτουμε στα ενεργά tubes, τα προτεινόμενα ToIs απ' το τρέχον τμήμα, μαζί με τα σκορ δραστικότητας στην λίστα με τα ανθροίσματα δραστικότητας και μηδενικές τιμές στις αντίστοιχες θέσεις στην λίστα με τα σκορ επικάλυψης.
6. Επαναλαμβάνουμε τα προηγούμενα 3 βήματα μέχρι να μην έχει μείνει κανένα τμήμα βίντεο.
7. Τέλος, όπως αναφέραμε προηγουμένως, έχουμε μια λίστα που περιέχει τα ευρετήρια των αποθηκευμένων tubes. Έτσι, τα τροποποιούμε για να έχουμε τα αντίστοιχα δισδιάστα πλαίσια. Ωστόσο, οι 2 διαδοχικά ToIs δεν έχουν, πάντα, τα ίδια δισδιάστα πλαίσια στα καρέ που επικαλύπτονται. Για παράδειγμα, τα ToIs από το πρώτο τμήμα βίντεο ξεκινούν από το 1ο καρέ έως το 16ο καρέ. Εάν έχουμε βήμα βίντεο ίσο με 8, αυτά τα ToIs επικαλύπτονται χρονικά με τα ToIs από το δεύτερο τμήμα βίντεο στα καρέ 8-16. Σε αυτά τα πλαίσια, στο τελικό action tube, επιλέγουμε την περιοχή που περιέχει και τα δύο πλαίσια οριοθέτησης που συμβολίζονται ως  $\min(x_1, x'_1), \min(y_1, y'_1), \max(x_2, x'_2), \max(y_2, y'_2)$  για τα δισδιάστα πλαίσια  $(x_1, y_1, x_2, y_2)$  και  $(x_1, y_1, x_2, y_2)$ .

#### 4.1.1 JHMDB Dataset

Ξεκινώντας, θα ασχοληθούμε αρχικά μόνο με το JHMDB dataset προκειμένου να καθορίσουμε την πολιτική που ακολουθούμε για να υπολογίσουμε το σκορ επικάλυψης. Κι αυτό γιατί τα βίντεο που περιέχει αυτό το σύνολο δεδομένων είναι πιο μικρά σε διάρκεια και λιγότερα στον αριθμό, οπότε θα μπορέσουμε να βγάλουμε συμπεράσματα πιο γρήγορα απ' το να εξετάζουμε και τα δύο σύνολα δεδομένων ταυτόχρονα.

**Διάρκεια δείγματος ίση με 16 καρέ** Ξεκινάμε ορίζοντας ως διάρκεια δείγματος ίση με 16 καρέ ανά βίντεο κλιπ. Αφού πραγματοποιήσαμε κάποια πρώτα πειράματα με βήμα βίντεο ίσο με 8 και 12 καρέ, στα οποία δεν είχαμε καλές επιδόσεις σε recall, αποφασίσαμε να εξετάσουμε την περίπτωση του βήματος βίντεο ίσο με 14, 15 και 16 τα οποία παρουσιάζονται στον πίνακα 4.1. Για κάθις διαφορετικό βήμα βίντεο έχουμε και διαφορετικές περιπτώσεις στις οποίες εξετάζουμε το σκορ επικάλυψης. Στις περιπτώσεις όπου έχουμε πάνω από 1 καρέ, λαμβάνουμε ως σκορ επικάλυψης την μέση τιμή των σκορ επικάλυψης των αντίστοιχων καρέ. Στον 4.1 αναφερόμαστε με πιο έντονο χρώμα στα καρέ, για τα οποία εξετάζουμε την επικάλυψη τους.

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 4.1: Recall results for steps = 14, 15 and 16

Παρατηρούμε ότι έχουμε την καλύτερη επίδοση recall για βήμα βίντεο ίσο με 16 καρέ όταν συγχρίνουμε χωρικά την επικάλυψη του τελευταίου πλαισίου με την επικάλυψη του πρώτου.

**Διάρκεια δείγματος ίση με 8** Θέλοντας να επιβεβαιώσουμε ότι έχουμε τα καλύτερα αποτελέσματα όταν έχουμε βήμα βίντεο ίσο με την διάρκεια του δείγματος, εξετάσαμε και την περίπτωση να έχουμε διάρκεια δείγματος ίση με 8. Τα αποτελέσματα παρουσιάζονται στον πίνακα 4.2 και περιλαμβάνει τις περιπτώσεις όπου έχουμε βήμα βίντεο ίσο με 6, 7 και 8 καρέ.

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 4.2: Recall results for steps = 6, 7 and 8

Με βάση και τα αποτελέσματα του πίνακα 4.2 είναι πλέον ξεκάθαρο ότι πετυχαίνουμε καλύτερα αποτελέσματα όταν θέτουμε το βήμα βίντεο ίσο με την διάρκεια του δείγματος και το σκορ επικάλυψης υπολογίζεται από το πλαίσιο του τελευταίου καρέ του πρώτου ToI με το πλαίσιο του πρώτο καρέ του δεύτερου ToI.

#### 4.1.2 UCF Dataset

Σε προηγούμενα βήματα, προσπαθήσαμε να βρούμε την καλύτερη πολιτική επικάλυψης για τον αλγόριθμο μας στο σύνολο δεδομένων JHMDB. Μετά από αυτό, είναι καιρός να εφαρμόσουμε τον αλγόριθμο μας στο σύνολο δεδομένων UCF χρησιμοποιώντας την καλύτερη βαθμολογική πολιτική επικάλυψης. Κάναμε κάποιες τροποποιήσεις στον κώδικα, για να χρησιμοποιούμε λιγότερη μνήμη, και μετακινήσαμε τα περισσότερα μέρη του κώδικα σε GPU. Αυτό συνέβη με τη χρήση τένσορων αντί για λίστες με βαθμολογίες ενώ οι περισσότερες πράξεις είναι, από τώρα και στο εξής, πράξεις πινάκων. Πάνω σ' αυτό, το τελευταίο βήμα του αλγόριθμου, η οποία είναι η τροποποίηση από δείκτες σε πραγματικές ακολουθίες από πλαίσια, είναι γραμμένο πλέον σε CUDA κώδικα έτσι λαμβάνει χώρα και αυτή στη GPU. Έτσι, τώρα μπορούμε να αυξήσουμε τον αριθμό των ToIs που επιστρέφονται από το TPN, τον μέγιστο αριθμό των ενεργών tubes πριν από την ενημέρωση του ορίου και τον μέγιστο αριθμό τελικών tubes.

Τα πρώτα πειράματα που διενεργήσαμε σχετίζονταν με τον αριθμό των τελικών σωλήνων, τα οποία το δίκτυο μας προτείνει, παράλληλα με τον αριθμό των προτεινόμενων ToIs από το TPN. Πειραματίζόμαστε για υποθέσεις, στις οποίες το TPN προτείνει 30, 100 και 150 ToIs, το τελικό δίκτυο μας προτείνει 500, 2000 και 4000 υποψήφια action tubes για διάρκεια δείγματος ίσο με 8 και 16 καρέ. Για διάρκεια δείγματος ίσο με 8 επιστρέφουμε 100 ToIs επειδή, όταν προσπαθήσαμε να επιστρέψουμε 150 ToIs, λαμβάνουμε OutOfMemory σφάλμα. Ο πίνακας 4.3 δείχνει τις αποδόσεις των χωροχρονικών recall και MABO, αυτών των προσεγγίσεων. Ο πίνακας 4.4 δείχνει την απόδοση των χρονικών recall και MABO. Ενδιαφερόμαστε για τη χρονική απόδοση, επειδή το UCF αποτελείται από ατριμάριστα βίντεο, σε αντίθεση με το JHMDB που έχει μόνο τριμαρισμένα βίντεο. Έτσι, θέλουμε να γνωρίζουμε πόσο καλά το δίκτυο μας είναι σε θέση να προτείνει action tubes που επικαλύπτονται με τα πραγματικά action tubes πάνω από ένα «μεγάλο» όριο. Για χρονικό εντοπισμό, δεν χρησιμοποιούμε τα 0,5, 0,4 και 0,3 ως επικαλυπτόμενο όριο, αλλά αντί αυτού, χρησιμοποιούμε 0,9, 0,8 και 0,7, επειδή είναι πολύ σημαντικό το δίκτυο μας να είναι σε θέση να προτείνει action tubes που περιέχουν μια ενέργεια, τουλάχιστον από χρονικής απόψεως. Για να υπολογίσουμε τη χρονική επικάλυψη, χρησιμοποιούμε το IoU για μια μόνο διάσταση.

combination	TPN tubes	Final tubes				MABO
			0.5	0.4	0.3	
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 4.3: Recall results for UCF dataset

combination	TPN tubes	Final tubes	overlap thresh			MABO
			0.9	0.8	0.7	
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 4.4: Temporal Recall results for UCF dataset

Όπως φαίνεται και από τους πίνακες 4.3 και ;;, για διάρκεια δείγματος ίση με 8 λαμβάνουμε την καλύτερη επίδοση όταν επιστρέφει το TPN 100 ToIs και συνολικά το ActionNet 4000 action tubes, ενώ για διάρκεια δείγματος ίση με 16 καρέ όταν επιστρέφει το TPN, 30 ToIs και το ActionNet 4000 action tubes.

#### Προτεινόμενη τροποποίηση του αλγορίθμου

Στην προηγούμενη προσέγγιση, το κατώφλι σύνδεσης ανανεώνεται και αυξάνεται κάθε φορά ο αριθμός από «ενεργά» tubes ξεπερνούν ένα συγκεκριμένο αριθμό. Ωστόσο, παρατηρήσαμε ότι με αυτόν τον τρόπο το σύστημα μας αδυνατεί να προτείνει action tubes τα οποία ξεκινούν μετά από ορισμένα καρέ. Κι αυτό γιατί μέχρι τότε το κατώφλι σύνδεσης έχει αυξηθεί τόσο που δεν επιτρέπει να δημιουργηθούν νέα tubes . Για τον λόγο αυτό τροποποιήσαμε τον αλγόριθμο μας έτσι ώστε να μην ανανεώνεται το κατώφλι σύνδεσης. Παράλληλα, προσθέσαμε τον αλγόριθμο NMS προκειμένου να απορρίπτει action tubes που επικαλύπτονται αρκετά με κάποια ήδη προτεινόμενα action tubes. Οι πίνακες 4.5 και 4.6 περιλαμβάνουν τα χωροχρονικά και χρονικά αποτελέσματα για το recall και το MABO, ενώ πειραματικόμαστε με κατώφλι σύνδεσης του NMS ίσο με 0.7, 0.9 και χωρίς καθόλου NMS.

combination	NMS thresh	PreNMS tubes	overlap thresh			MABO
			0.9	0.8	0.7	
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,}	-		0.438	0.5635	0.6829	0.4231788
{16,}17,...,23	0.7	20000	0.4525	0.5848	0.7034	0.429747438
	0.9		0.3802	0.5133	0.6068	0.3862278851848662

Table 4.5: Spatio-temporal Recall results for UCF dataset

combination	NMS thresh	PreNMS tubes	overlap thresh			MABO
			0.9	0.8	0.7	
0,1,...,6,{7,} {8,}9,...,15	-	20000	0.7087	0.8281	0.8913	0.899210587
	0.7		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	-	20000	0.8327	0.9156	0.9399	0.940143272
	0.7		0.8646	0.9369	0.9567	0.946701832
	0.9		0.6183	0.7696	0.8388	0.8628507037919737

Table 4.6: Temporal Recall results for UCF dataset

Συγκρίνοντας τις επιδόσεις των recall και MABO που παρουσιάζονται στον Πίνακας 4.5 μαζί με αυτές του Πίνακα 4.3, συμπεραίνουμε πως για διάρκεια δείγματος ίση με 8, η νέα τροποποίηση οδηγεί σε χειρότερα αποτελέσματα όταν το κατώφλι σύνδεσης είναι 0.7 αλλά καλύτερα για κατώφλι ίσο με 0.9. Απ' την άλλη, για διάρκεια δείγματος ίση με 16, παρατηρούμε πως έχουμε καλύτερα αποτελέσματα για κατώφλι σύνδεσης του NMS αλγορίθμου ίσο με 0.7.

## 4.2 Δεύτερη προσέγγιση

Όπως είδαμε και προηγουμένως, ο αλγόριθμος μας δεν έχει πάρα πολύ καλές recall επιδόσεις. Εποιητικά, δημιουργήσαμε έναν άλλο αλγόριθμο ο οποίος βασίζεται σε αυτόν που πρότειναν οι Hu et al. 2019. Αυτός ο αλγόριθμος εισάγει δύο νέες μετρικές σύμφωνα με τους Hu et al. 2019.

**Πρόοδος** που περιγράφει την πιθανότητα μιας συγκεκριμένης δράσης να εκτελείται στο ToI.

Προσθέτουμε αυτόν τον παράγοντα επειδή παρατηρήσαμε ότι η δραστικότητα είναι ανεκτική σε ψεudώς ψευτικά. Η πρόοδος είναι ένα μηχανισμός επαναβαθμολόγησης για κάθε κατηγορία (όπως αναφέρονται οι Hu κ.ά. 2019)

**Ρυθμός προόδου** που ορίζεται ως η αναλογία προόδου κατά την οποία κάθε κατηγορία δράσης έχει πραγματοποιηθεί.

Έτσι, κάθε action tube περιγράφεται ως ένα σύνολο TOIs

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

όπου το  $t_i^{(k)}$  περιέχει τις χωροχρονικές πληροφορίες των TOI, το  $s_i^{(k)}$  το σχορ σιγουριάς του και το  $r_i^{(k)}$  τον ρυθμό προόδου.

Σε αυτή την προσέγγιση, κάθε κλάση αντικειτωπίζεται ξεχωριστά, συνεπώς για την υπόλοιπη ενότητα συζητάμε για την παραγωγή action tubes μόνο για μία κλάση. Για τη σύνδεση 2 ToIs, για ένα βίντεο με  $N$  τμήματα βίντεο, εφαρμόζονται τα ακόλουθα βήματα:

1. Για το πρώτο τμήμα βίντεο ( $k = 1$ ), προετοιμάζουμε έναν πίνακα με τα  $M$  καλύτερα ToIs, τα οποία θα ψεωρούνται ως ενεργά tubes(AT). Αντιστοιχα, προετοιμάζουμε έναν πίνακα με  $M$  ρυθμούς προόδου και  $M$  βαθμολογίες εμπιστοσύνης.

2. Για  $k = 2:N$ , εκτελούμε τα βήματα (α') - (γ'):

(α') Υπολογίζουμε τις επικαλύψεις μεταξύ  $AT^{(k)}$  και  $TOIs^{(k)}$ .

(β') Συνδέουμε όλα τα tubes που ικανοποιούν τα ακόλουθα κριτήρια:

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

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

(γ') Για όλα τα νέα tubes ενημερώνουμε το σκορ εμπιστοσύνης και τον ρυθμό προόδου ως εξής:

Το νέο σκορ εμπιστοσύνης είναι η μέση βαθμολογία όλων των συνδεδεμένων ToIs:

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

Ο νέος βαθμός προόδου είναι ο υψηλότερος βαθμός προόδου:

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

(δ') Διατηρούμε τα  $M$ -καλύτερα action tubes ως ενεργά tubes που προορίζονται τελικώς για ταξινόμηση.

Αυτή η προσέγγιση έχει το πλεονέκτημα ότι δεν χρειάζεται να εκτελέσουμε ξανά την ταξινόμηση, επειδή γνωρίζουμε ήδη την κατηγορία του κάθε τελικού action tube. Για να επικυρώσουμε τα αποτελέσματά μας, τώρα, υπολογίζουμε την επίδοση του recall μόνο για τα tubes που έχουν την ίδια κλάση με το πραγματικό tube. Και πάλι ψεωρούμε ένα πραγματικό tube ότι είναι θετικό αν υπάρχει τουλάχιστον ένα tube που επικαλύπτεται με το πραγματικό περισσότερα από ένα προκαθορισμένο όριο.

combination		overlap thresh		
sample	dur	step	0.5	0.4
			0.3284	0.5
8	6		0.3284	0.5
8	7		0.209	0.459
8	8		0.3060	0.5672
16	8		0.194	0.4366
16	12		0.3358	0.5336
16	16		0.2649	0.4664
				0.6082
				0.6119
				0.6866
				0.7164
				0.7537
				0.709

Table 4.7: Recall results for second approach with step = 8, 16 and their corresponding steps

Σύμφωνα με τον Πίνακα 4.7, έχουμε βέλτιστη απόδοση όταν ορίζεται διάρκεια δείγματος ίση με 16 και βήμα βίντεο ίσο με 12. Συγχρίνοντας αυτή την απόδοση με την πρώτη προσέγγιση, τόσο για τη διάρκεια του δείγματος ίση με 8 και 16 καρέ, παρατηρούμε ότι η δεύτερη προσέγγιση υπολείπεται σε σύγκριση με την πρώτη.

### 4.3 Τρίτη προσέγγιση (μόνο για το JHMDB)

Όπως αναφέρεται στην πρώτη προσέγγιση, οι Hou, Chen, and Shah 2017 υπολογίζουν όλες τις πιθανές ακόλουθιες των ToIs προκεμένου να βρουν τις καλύτερες υποψήφιες. Σκεφτήκαμε ξανά αυτή την προσέγγιση και συμπεράναμε ότι όλα μπορούσαμε να την υλοποιήσουμε μόνο για το σύνολο δεδομένων JHMDB εάν μειώσουμε τον αριθμό των προτεινόμενων ToIs, που παράγονται από TPN, από 150 σε 30 για κάθε βίντεο κλιπ. Εκμεταλλευόμαστε το γεγονός ότι τα βίντεο του συνόλου δεδομένων JHMDB είναι κομμένα, οπότε δεν χρειάζεται να κοιτάζουμε για action tubes που ζεκινούν από το δεύτερο βίντεο κλιπ, γεγονός που μας σώζει πολύ μνήμη. Και πάνω απ' όλα, τροποποιήσαμε τον κώδικα μας με σκοπό να είναι πιο αποτελεσματικός στο θέμα της μνήμης γράφοντας κάποια μέρη στη γλώσσα προγραμματισμού CUDA, εξοικονομώντας πολύ επεξεργαστική ισχύ, επίσης.

Έτσι, μετά τον υπολογισμό όλων των πιθανών συνδυασμών ζεκινώντας από το πρώτο βίντεο κλιπ και καταλήγοντας στο τελευταίο, κρατάμε μόνο τα **k-καλύτερα action tubes (k = 500)**. Τρέχουμε πειράματα με διάρκεια του δείγματος ίση με 8 και 16 καρέ και τροποποιούμε το βήμα βίντεο κάθε φορά. Για τη διάρκεια του δείγματος = 8, επιστρέφουμε μόνο 15 ToIs και για το δείγμα = 16, επιστρέφουμε 30 επειδή, αν επιστρέψουμε περισσότερο, λαμβάνουμε σφάλμα «OutOfMemory». Στον ακόλουθο πίνακα 4.8 παρουσιάζονται τα αποτελέσματα του recall.

combination		overlap thresh		
sample	dur	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 4.8: Recall results for second approach with

Από τον παραπάνω πίνακα, πρώτον, ξαναεπιβεβαιώνουμε ότι όταν το βήμα βίντεο είναι ίσο με την διάρκεια δείγματος μας δίνει τα καλύτερα αποτελέσματα recall. Παρατηρούμε ότι όταν η διάρκεια του δείγματος ισούται με 16 καρέ η απόδοση του recall είναι ελαφρώς καλύτερη όταν ισούται με 8. Ωστόσο, η χρήση 16 καρέ ανά βίντεο κλιπ αυξάνει τη χρήση της μνήμης, ακόμα και αν μειώνει τον αριθμό των τμημάτων βίντεο, εξαιτίας της ανάγκης επεξεργασίας μεγαλύτερων βίντεο, χαρτών ενεργοποίησης κλπ. Έτσι, για το στάδιο της ταξινόμησης θα πειραματιστούμε χρησιμοποιώντας κυρίως δείγμα διάρκειας ίσο με 8 καρέ.



## Κεφάλαιο 5

# Classification stage

Στα προηγούμενα 2 κεφάλαια παρουσιάσαμε την διαδικασία που χρησιμοποιούμε για να δημιουργήσουμε υποψήφια action tubes, τα οποία πιθανώς να περιέχουν κάποια πραγματοποιούμενη δράση ή μπορεί όχι. Τις περισσότερες φορές τα προτεινόμενα action tubes ανήκουν στο φόντο, και γι' αυτό, όπως αναφέρθηκε και στον προηγούμενο κεφάλαιο, είναι σημαντικό να επιλέξουμε έναν καλό αλγόριθμο που προτείνει καλές ακολουθίες από πλαίσια. Ωστόσο, είναι αρκετά σημαντικό να επιλέξουμε και τον κατάλληλο ταξινομητή ο οποίος θα είναι σε θέση με μεγάλη ακρίβεια να προβλέψει αν ένα υποψήφιο action tube ανήκει σε μια γνωστή κατηγορία από δράσεις ή ανήκει στο φόντο. Κι αυτό γιατί μπορεί να παράγουμε καλές προτάσεις για υποψήφιες δράσεις, αλλά αν ο ταξινομητής μας δεν λειτουργεί στο έπακρο, το σύστημα μας πάλι θα αποτυγχάνει να αναγνωρίσει τις δράσεις.

Η σωστή επιλογή ενός ταξινομητή είναι μια μεγάλη απόφαση που καλούμαστε να πάρουμε. Ωστόσο, αυτός ο ταξινομητής θα δεχθεί ορισμένους χάρτες ενεργοποίησης τους οποίους θα κληθεί να ταξινομήσει. Συνεπώς, εκτός από την καλή επιλογή ταξινομητή, εξίσου σημαντική είναι η καλή επιλογή χαρακτηριστικών. Τέλος, μεγάλο ρόλο παίζει και η διαδικασία εκπαίδευσης του ταξινομητή προκειμένου να είναι σε θέση να γενικεύει και καταστάσεις overfitting να αποφεύγονται.

Σε αυτό το κεφάλαιο παρουσιάζουμε διάφορες μεθόδους που χρησιμοποιήσαμε οι οπίσες περιλαμβάνουν ένα Γραμμικό ταξινομητή, ένα Recursive Neural Network (RNN), ένα Support Vector Machine (SVM) και ένα Multilayer Perceptron(MLP). Επίσης, πειραματίζόμαστε χρησιμοποιώντας χάρτες χαρακτηριστικών που εξήγγιναν μέσω του 3D RoiAlign χρησιμοποιώντας παράλληλα avg ή max pooling. Τελευταίο αλλά εξίσου σημαντικό είναι το γεγονός ότι προσπαθήσαμε να βρούμε το καλύτερο ποσοστό μεταξύ action tubes προσκηνίου και φόντο αλλά και τον συνολικό αριθμό τους που είναι απαραίτητα κατά την διάρκεια της εκπαίδευσης προκειμένου ο ταξινομητής να λειτουργεί αποδοτικά.

Η όλη διαδικασία ταξινόμησης αποτελείται από τα ακόλουθα βήματα:

1. Διαχωρίζουμε το βίντεο σε μικρά βίντεο κλιπ και τροφοδοτούμε το δίκτυο TPN με αυτά τα βίντεο κλιπ και παίρνουμε ως αποτέλεσμα k-προτεινόμενα ToIs και τα αντίστοιχα χαρακτηριστικά τους για κάθε κλιπ βίντεο.
2. Συνδέουμε τα προτεινόμενα ToIs για να πάρουμε action tubes που μπορεί να περιέχουν μια ενέργεια.
3. Για κάθε υποψήφιο action tube, η οποία είναι μια ακολουθία του ToIs, τροφοδοτούμε τους χάρτης ενεργοποίησης του στον ταξινομητή για ταξινόμηση.

Για το βήμα ταξινόμησης πειραματίζόμαστε μόνο με το σύνολο δεδομένων JHMDB. Αυτό συμβαίνει επειδή καταφέραμε να επιτύχουμε καλή απόδοση recall μόνο για τα βίντεο του JHMDB

αντίθετα με το UCF-101. Για το σύνολο δεδομένων UCF-101, κατορθώσαμε να δημιουργήσουμε καλές προτάσεις action tubes σε λιγότερο από μισές περιπτώσεις. Έτσι, το σύστημά μας δεν θα είναι σε θέση να εκτελέσει καλά όχι λόγω της επιλεγέσα τάξη, αλλά λόγω της έλλειψης καλών προτάσεων.

## 5.1 JHDMB dataset

### 5.1.1 Ταξινομητές Linear, SVM και RNN

**Training** Για να εκπαιδεύσουμε τον ταξινομητή μας, πρέπει να εκτελέσουμε τα προηγούμενα βήματα, για κάθε βίντεο. Ωστόσο, κάθε βίντεο έχει διαφορετικό αριθμό καρέ και καταλαμβάνει μεγάλη ποσότητα μνήμης στη ΓΠΥ. Για να αντιμετωπίσουμε αυτή την κατάσταση και έχοντας 4 διαθέσιμες GPU, δίνουμε ως είσοδο ένα βίντεο ανά GPU. Έτσι μπορούμε να χειριστούμε 4 βίντεο ταυτόχρονα. Αυτό σημαίνει ότι ένα κλασσικό training πάιρνει πάρα πολύ χρόνο για μόλις 1 εποχή. Η λύση με την οποία ήρθαμε, είναι να προϋπολογίζουμε τους χάρτες χαρακτηριστικών τόσο για action tubes προσκηνίου όσο και φόντου και στη συνέχεια να τροφοδοτήσουμε αυτούς τους χάρτες στον ταξινομητή μας για να τον εκπαιδεύσουμε. Αυτή η λύση περιλαμβάνει τα ακόλουθα βήματα:

1. Αρχικά, εξάγουμε τους χάρτες χαρακτηριστικών από τα πραγματικά action tubes. Ακόμα εξάγουμε τα χαρακτηριστικά από action tubes φόντου τα οποία είναι διπλάσια στον αριθμό από αυτά του φόντου. Επιλέξαμε αυτή την αναλογία μεταξύ του αριθμού των θετικών και αρνητικών action tubes εμπνευσμένοι από τους Yang et al. 2017, των οποίων η μέσθιδος χρησιμοποιεί ποσοστό 25% μεταξύ των περιοχών ενδιαφέροντος προσκηνίου και των συνολικών περιοχών, και συνολικά επιλέγει 128 τέτοιες περιοχές. Αντίστοιχα, επιλέγουμε ένα λίγο μεγαλύτερο ποσοστό επειδή έχουμε μόνο ένα πραγματικό action tube σε κάθε βίντεο. Έτσι, για κάθε βίντεο λαμβάνουμε 3 action tubes συνολικά, 1 προσκηνίου και 2 φόντου. Θεωρούμε ως background action tubes εκείνα που το σκορ επικάλυψης τους με οποιοδήποτε action tube είναι μεγαλύτερο από 0.1 αλλά μικρότερο από 0.3. Φυσικά, προκειμενού να εξάγουμε αυτά τα action tubes, χρησιμοποιούμε ένα προεκπαίδευμένο TPN, για να μας προτείνει ToIs για κάθε τμήμα βίντεο και τον προτεινόμενο αλγόριθμο σύνδεσης για να συνδέσουμε αυτά τα ToIs. Τελικώς, για κάθε action tube λαμβάνουμε τους αντίστοιχους χάρτες ενεργοποίησης χρησιμοποιώντας 3D RoiAlign.
2. Αφού εξάγουμε αυτά τα χαρακτηριστικά, εκπαιδεύουμε τους ταξινομητές μας. Ο Γραμμικός ταξινομητής χρειάζεται ένα σταύρο μέγεθους εισόδου, συνεπώς χρησιμοποιούμε μια συνάρτηση pooling στην διάσταση του αριθμού των βίντεο. Έτσι, αρχικά έχουμε ένα χάρτη χαρακτηριστικών μεγέθους  $3,512,16$  και μετά λαμβάνουμε ως έξοδο έναν χάρτη χαρακτηριστικών μεγέθους  $512,16$ . Πειραματιζόμαστε χρησιμοποιώντας avg πολινό  $\sqrt{3}$  και avg pooling όπως φαίνεται στον Πίνακα χρησιμοποιώντας 5.1. Για τον ταξινομητή RNN δεν χρειάζομαστε καμία pooling διαδικασία ενώ για τον ταξινομητή SVM πειραματιζόμαστε ξανά χρησιμοποιώντας και τις δύο αυτές συναρτήσεις τα αποτελέσματα του οποίου φαίνονται στον Πίνακα 5.2.

**Validation** Το στάδιο επικύρωσης περιλαμβάνει τη χρήση τόσο προεκπαίδευμένου TPN όσο και του ταξινομητή. Έτσι, για κάθε βίντεο λαμβάνουμε σκορ ταξινόμησης για τα προτεινόμενα action tubes. Οι περισσότερες προσεγγίσεις συνήθως θεωρούν ένα κατώφλι σκορ εμπιστοσύνης πάνω από το οποίο θεωρούν ένα action tube ως προσκήνιο. Ωστόσο, εμείς δεν χρησιμοποιούμε κανένα σκορ εμπιστοσύνης. Αντιθέτως, επειδή γνωρίζουμε ότι JHMDDB έχει κομμένα βίντεο με μόνο 1

εκτελούμενη δράση ανά βίντεο, εμείς απλά θεωρούμε το καλύτερο ως προς το σκορ action tube ως πρόβλεψη.

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: First classification results using Linear and RNN classifiers

Dimensions before	Dimensions after	Pooling	mAP precision		
			0.5	0.4	0.3
(k,64,8,7,7)	(1,64,8,7,7)	mean	3.16	4.2	4.4
(k,64,8,7,7)	(1,64,8,7,7)	max	1.11	2.35	2.71
(k,256,8,7,7)	(1,256,8,7,7)	mean	11.41	11.73	11.73
(k,256,8,7,7)	(1,256,8,7,7)	max	<b>22.07</b>	<b>24.4</b>	<b>25.77</b>

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

### 5.1.2 Temporal pooling

Μετά τη λήψη των πρώτων αποτελεσμάτων, εφαρμόζουμε μια συνάρτηση χρονικής ομαδοποίησης (temporal pooling) εμπνευσμένη από το Hou, Chen, and Shah 2017. Χρειαζόμαστε ένα σταθερό μέγεθος εισόδου για το Σ'Μ. Ωστόσο, το χρονικό stride των action tube μας ποικίλλει από 2 έως 5, αφού ένα βίντεο με 15 καρέ αποτελείται από 2 συνεχόμενες ToIs ενώ ένα βίντεο με 40 καρέ αποτελείται από 5. Έτσι χρησιμοποιύμε ως σταθερή χρονική διάσταση ίσον με 2. Ως λειτουργία pooling χρησιμοποιούμε 3D max poolign, για κάθε φίλτρο του χάρτη χαρακτηριστικών ξεχωριστά. Για παράδειγμα, για ένα action tube με 4 συνεχόμενες ToIs, έχουμε (4, 256, 8, 7, 7) ως μέγεθος του χάρτη χαρακτηριστικών. Διαχωρίσουμε το feature map σε 2 ομάδες χρησιμοποιώντας την συνάρτηση *linspace* και αναδιαμορφώνουμε το χάρτη χαρακτηριστικών σε (256, k, 8, 7, 7) όπου k είναι το μέγεθος της κάθε ομάδας. Αφού κάνουμε χρήση 3D max pooling, θα πάρουμε ένα χαρακτηριστικό χάρτη διαστάσεων (256, 8, 7, 7), ακολούθως τους ενώνουμε και τελικά λαμβάνουμε χαρακτηριστικών μεγέθους (2, 256, 8, 7, 7). Σε αυτή την περίπτωση δεν πειραματίζόμαστε με χάρτες χαρακτηριστικών μεγέθους (64, 8, 7, 7) επειδή με βάση τα παραπάνω αποτελέσματα, δεν θα έχουμε καλύτερη επίδοση απ' τα χαρακτηριστικών μεγέθους (256, 8, 7, 7). Τα αποτελέσματα παρουσιάζονται στον πίνακα 5.3, όπου περιλαμβάνεται η καλύτερη προηγούμενη μέθοδος η οποία χρησιμοποιεί max pooling αντί για temporal pooling.

Dimensions before	Dimensions after	Temp Pooling	mAP precision		
			0.5	0.4	0.3

k,256,8,7,7	1,256,8,7,7	-	22.07	24.4	25.77
k,256,8,7,7	2,256,8,7,7	Yes	25.07	26.91	29.11

Table 5.3: mAP results using temporal pooling for both RoiAlign approaches

## 5.2 Προσθήκη περισσότερων groundtruth tubes

Τα προηγούμενα αποτελέσματα προήλθαν από την εκπαίδευση των ταξινομητών χρησιμοποιώντας μόνο 1 action tube προσκηνίου και 2 φόντο. Σκεφτήκαμε ότι θα έπρεπε να πειραματιστούμε με τον αριθμό των action tubes προσκηνίου καθώς επίσης και την αναλογία μεταξύ των action tubes προσκηνίου και φόντου, επειδή στις προηγούμενες προσεγγίσεις λειτουργήσαμε λιγάκι αυθαίρετα. Έτσι, επιλέγουμε να εκπαιδεύσουμε τους προηγούμενους ταξινομητές μας χρησιμοποιώντας 2, 4 και 8 action tubes προσκηνίου και αναλογία 2:3, 1:2, 1:3 και 1:4 μεταξύ του αριθμού των tubes προσκηνίου και του συνολικού αριθμού τους.

Πρώτον, εκπαιδεύουμε το RNN ταξινομητή χρησιμοποιώντας χάρτες χαρακτηριστικών με διαστάσεις (256, 8, 7, 7). Οι επιδόσεις τους με βάση την μετρική mAP παρουσιάζονται στον πίνακα 5.4 για το όριο επικάλυψης ίσο με 0,5, 0,4 και 0,3.

F. map	FG tubes	Total tubes	mAP		
			0.5	0.4	0.3
(k,256,8,7,7)	2	1	3	11.3	14.14
		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 5.4: RNN results

Σύμφωνα με τον πίνακα 5.4, πρώτον, μπορούμε να δούμε ότι η αύξηση του αριθμού των action tubes προσκηνίου από 1 έως 2 οδηγούν στην απότομη μείωση της απόδοσης του mAP. Αλλά, όταν θέτουμε τα action tubes προσκηνίου ίσα με 4 έχουμε καλύτερα αποτελέσματα. Πάνω σ' αυτό, έχουμε την καλύτερη απόδοση όταν η αναλογία είναι ίση με 1:2 και 1:4. Τέλος, όταν ορίζουμε τον αριθμό των tubes προσκηνίου ίσο με 8, η απόδοση βελτιώνεται ελαφρώς σε σύγχριση με τις αρχικές επιλογές (1 action tube προσκηνίου και 3 συνολικά), αλλά η κατάσταση αυτή δεν να μας φέρεις τα καλύτερα αποτελέσματα.

Στη συνέχεια, είναι καιρός να πειραματιστούμε χρησιμοποιώντας τη γραφική ταξινόμηση. Χρησιμοποιούμε ξανά το ίδιες υποθέσεις όπως κάναμε και για την ταξινόμηση με RNN. Όπως προαναφέρθηκε, χρειαζόμαστε μια μέθοδο ομαδοποίησης (pooling) πριν από το βήμα ταξινόμησης. Σύμφωνα με τον πίνακα 5.1, η μέθοδος του avg pooling έχει ως αποτέλεσμα καλύτερη απόδοση mAP από το max pooling, οπότε χρησιμοποιούμε avg pooling για όλες τις ακόλουθες περιπτώσεις. Τα αποτελέσματα περιλαμβάνονται στον πίνακα 5.5.

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
		3	12.68	13.38	15.14
		4	11.5	14.95	16.22
	2	6	10.74	13.36	15.18
		8	8.00	9.83	11.17
		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
	4	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 5.5: Linear results

Πρώτα απ' όλα, μετά την εξέταση των αποτελεσμάτων που παρουσιάστηκαν στους δύο πίνακες 5.4 και 5.5, είναι σαφές ότι όταν ορίζουμε τον αριθμό των action tubes προσκηνίου ίσο με 2, και για τις δύο περιπτώσεις, έχουμε χειρότερα αποτελέσματα απ' το αρχικό. Αυτό μάλλον οφείλεται στο γεγονός ότι αυξάνουμε επίσης τον αριθμό των action tubes φόντου για περιπτώσεις όταν η αναλογία είναι 1:2, 1:3 και 1:4 με αποτέλεσμα να θεωρούν οι ταξινομητές τα περισσότερα προτεινόμενα action tubes ότι είναι φόντου. Από την άλλη πλευρά, όταν έχουμε ορίσει αναλογία ίση με 2:3, αντί να θεωρήσουν τα περισσότερα προτεινόμενα action tubes, ως φόντο, τα ταξινομούν ως μια συγκεκριμένη δράση τάξη, που σημαίνει ότι καταλήγομε σε κατάσταση overfitting. Έτσι, αν και πιστεύουμε ότι δεν θα πρέπει να ερευνήσουμε για περιπτώσεις με 2 action tubes που ανήκουν στο προσκήνιο, θα εκπαιδεύσουμε τον SVM ταξινομητή μας χρησιμοποιώντας 2 action tubes προσκηνίου και όλα τα προαναφερθέντα ποσοστά επειδή θέλουμε να είμαστε βέβαιοι για την υπόθεσή μας. Από την άλλη πλευρά, παρατηρούμε ότι η χρήση 4 ή 8 action tubes μας οδηγεί σε καλύτερα αποτελέσματα από το τα αρχικά αποτελέσματα. Οι καλύτερες επιδόσεις έρχονται όταν η αναλογία μεταξύ των αριθμών των action tubes προσκηνίου και συνολικών είναι 1:3 και για τις δύο περιπτώσεις. Παράλληλα, έχουμε καλά αποτελέσματα για τις αναλογίες 2:3 και 1:2, και λαμβάνουμε την χειρότερη επίδοση όταν χρησιμοποιούμε αναλογία 1:4. Αυτό προκαλείται μάλλον από το μεγάλο αριθμός action tubes φόντου σε σχέση με τον αριθμό των action tubes προσκηνίου. Όπως προαναφέρθηκε, εκπαιδεύουμε τον ταξινομητή SVM χρησιμοποιώντας τις προαναφερθείσες περιπτώσεις. Οι επιδόσεις ταξινόμησης με χρήση της μέτρησης mAP εμφανίζονται στον πίνακα 5.6.

F. map	FG tubes	Total tubes	mAP		
			0.5	0.4	0.3
(2,256,8,7,7)	2	1	3	24.97	26.91
			3	13.87	18.74
		4	14.21	19.67	21.29
		6	12.88	18.62	21.59
		8	12.66	18.7	21.97
	4	6	25.04	26.91	27.82
		8	24.34	25.67	26.34
		12	23.47	25.31	25.9
		16	21.94	23.55	24.23
	8	12	24.83	27.13	27.46
		16	23.97	26.38	26.94
		24	24.17	26.24	26.76
		32	24.17	26.24	26.76

Table 5.6: SVM results

Τα αποτελέσματα μας δείχνουν κάποια ενδιαφέροντα γεγονότα. Πρώτον, επιβεβαιώνουν την υπόθεσή μας ότι το δίκτυο είναι αδύνατο να εκπαιδεύτει με μόνο 2 action tubes προσκηνίου. Επίσης, παρατηρούμε ότι έχουμε σχεδόν τα ίδια αποτελέσματα με τα αποτελέσματα που προέκυψαν για τη χρήση της πολιτικής 1, μόνο ένα action tube προσκηνίου, 3 συνολικά και χρονικό pooling, γεγονός το οποίο είναι λίγο παράξενο. Αυτό είναι μάλλον επειδή κατά τη διάρκεια του υπολογισμού της κλίμακας, στο στάδιο εκπαίδευσης, δεν έχουμε τόσο καλό δείγμα βίντεο όπως κάναμε κατά τη διάρκεια της προαναφερθείσας περίπτωσης. Άλλα θεωρούμε ότι είναι καλύτερο να συνεχίσουμε τις δοκιμές χρησιμοποιώντας 4 ή 8 action tubes προσκηνίου. Τελευταίο αλλά όχι λιγότερο σημαντικό, είναι σαφές ότι έχουμε το καλύτερο αποτέλεσμα όταν έχουμε μια ποσοστό 2:3 μεταξύ του αριθμού των action tubes προσκηνίου και των συνολικών. Επίσης, είναι προτυπότερο να έχουμε 4 action tube προσκηνίου αντί για 8. Αυτό σημαίνει ότι επειδή έχουν δοθεί πάρα πολλά το SVM μπερδεύεται, και έτσι αποτυγχάνει να λειτουργήσει αποτελεσματικά.

### 5.3 Ταξινομητής MultiLayer Perceptron (MLP)

Σε προηγούμενες ενότητες χρησιμοποιήσαμε κλασσικούς ταξινομητές όπως τον Γραμμικό, ένα RNN και SVM. Τελευταίοι αλλά εξίσου σημαντικοί, μια άλλη ευρέως χατηγορία ταξινομητών είναι οι Multilayer Perceptron (MLP). Σχεδιάζουμε ένα ΜΛΠ όπως φαίνεται στο σχήμα 5.1 για διάρκεια δείγματος ίση με 8, και περιγράφεται κατωτέρω:

- Στην αρχή, μετά το 3D Roi Align και για διάρκεια του δείγματος ίση με 8 καρέ, λαμβάνουμε ένα χάρτη ενεργοποίησης μεγέθους  $(k, 256, 8, 7, 7)$  όπου  $k$  είναι ο αριθμός των συνδεδεμένων ToIcs. Εμπνευσμένοι από προηγούμενες ενότητες, εκτελούμε temporal pooling ακολουθούμενο από max pooling στην διάσταση της διάρκειας του δείγματος. Έτσι, έχουμε τώρα έναν χάρτη χαρακτηριστικών με διαστάσεις ίσες με  $(2, 256, 7, 7)$ , τις οποίες αναδιαμορφώνουμε σε  $(256, 2, 7, 7)$  και τροφοδοτούμε layers που εξήγησαν από το τελευταίο στάδιο του ResNet34. Αυτά τα στάδια περιλαμβάνουν 3 Residual Layers με stride ίσο με 2 σε όλες τις 3 διαστάσεις και αριθμός εξόδου φίλτρων ίσου με 512.

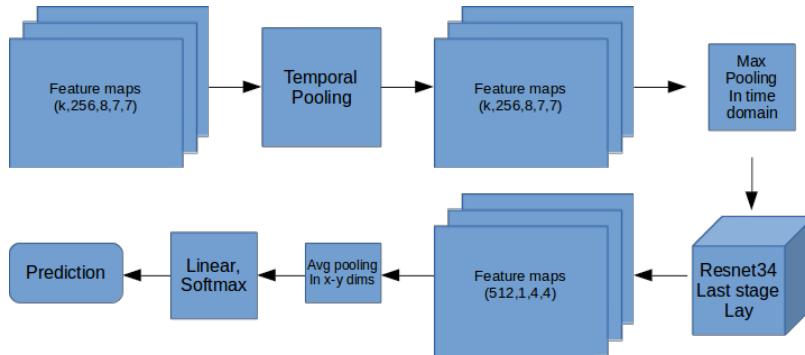


Figure 5.1: Structure of the MLP classifier

- Μετά τα Residual Layers, κάνουμε avg pooling για τις διαστάσεις  $x-y$ . Έτσι, έχουμε ως χάρτες ενεργοποίησης εξόδου με μέγευθος διαστάσεων ίσο με (512,). Τέλος, τροφοδοτούμε αυτούς τους χαρακτηριστικούς χάρτες σε ένα γραμμικό layer προκειμένου να εξάγουμε την κλάση του υποψήφιου action tube, μετά την εφαρμογή της λειτουργίας Soft-Max.

### 5.3.1 Κλασσικό training

Όπως προαναφέρθηκε προηγουμένως, ο κώδικας εκπαίδευσης απαιτεί την εκτέλεση ενός μόνο βίντεο ανά GPU, επειδή τα βίντεο έχουν διαφορετική διάρκεια. Για προηγούμενες προσεγγίσεις, μας ήρθε η ιδέα του προϋπολογισμού των χαρακτηριστικά των action tubes του βίντεο και στη συνέχεια εκπαιδεύουμε μόνο τον ταξινομητή. Ωστόσο, για αυτό το βήμα, εκπαιδεύσαμε τον ταξινομητή μας με τον κλασικό τρόπο για να λάβουμε αποτελέσματα ταξινόμησης. Φυσικά, χρησιμοποιήσαμε ένα προεκπαιδευμένο TPN, του οποίο παράσαμε τα layers για να μην εκπαιδευτούν. Προσπαθήσαμε να εξερευνήσουμε διαφορετικές αναλογίες μεταξύ του αριθμού των action tubes προσκηνίου και του συνολικού αριθμού των action tubes ανά βίντεο. Οι πρώτες 3 προσομοιώσεις περιλαμβάνουν σταθερό αριθμό συνολικών action tubes και μεταβλητή αναλογία μεταξύ του αριθμού των action tubes προσκηνίου και φόντου. Αρχίσαμε χρησιμοποιώντας μόνο action tubes προσκηνίου, το οποίο σημαίνει ότι 32 από 32 action tube είναι προσκηνίου, μετά τα μισά από τα προτεινόμενα action tubes, δηλαδή 16 από 32 και τέλος λιγότερο από το ήμισυ, δηλαδή 14 από τις 32. Μετά από αυτό, πειραματίζόμαστε χρησιμοποιώντας έναν σταθερό αριθμό action tubes προσκηνίου και μεταβλητού αριθμού συνολικών, ο οποίος είναι 16, 24 και 32. Τα αποτελέσματα των επιδόσεων παρουσιάζονται στον πίνακα 5.7.

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
	16	9.41	12.59	14.61
		12.32	15.53	18.57
		7.16	10.92	13.00

Table 5.7: MLP's mAP performance for regular training procedure

Τα αποτελέσματα δείχνουν ότι όταν οι πρώτες 3 προσεγγίσεις μας δίνουν πολύ άσχημα αποτελέσματα. Συγχρίνοντας τους με τους υπόλοιπους 3, ήρθαμε με το συμπέρασμα ότι χρειαζόμαστε το πολύ 8 action tubes προσκηνίου, ακόμη και όταν ο λόγος μεταξύ του αριθμού action tubes του προσκηνίου και του φόντου είναι υπέρ του δεύτερου. Πιθανότατα, πάρα πολλοί action tubes προσκηνίου κάνουν την αρχιτεκτονική μας να έρθει σε κατάσταση overfitting και συνεπώς να είναι ανίκανη να γενικεύει.

### 5.3.2 Εξαγωγή χαρακτηριστικών

Όπως εκτελέστηκε προηγουμένως, εκπαιδεύσαμε τον ταξινομητή MLP χρησιμοποιώντας πρωτότυπα μέσων ανάπτυξης γάριτες χαρακτηριστικών. Άμεσώς, οι γάριτες περιλαμβάνουν τέσσερα action tubes του

τμήματα, θα εκπαιδεύσουμε μόνο για αριθμό action tube προσκηνίου ίσο με 4 και 8. Επιπλέον Θα εκπαιδεύσουμε τον ταξινομητή μας για 3 διαφορετικές αναλογίες, οι οποίες είναι 1:1, 1:2 και 1:3. Ο πίνακας 5.8 δείχνει αυτές τις περιπτώσεις καθώς και τις αντίστοιχες επιδόσεις του mAP κατά τη διάρκεια του βήματος επικύρωσης.

<b>FG tubes</b>	<b>Total tubes</b>	<b>mAP</b>		
		0.5	0.4	0.3
4	6	4.37	8.54	10.12
	8	5.89	9.54	13.61
	12	9.51	12.8	14.6
	16	6.80	13.17	14.67
8	12	8.62	12.32	14.74
	16	8.49	13.94	15.09
	24	6.72	12.17	15.30
	32	13.27	17.64	18.97

Table 5.8: mAP results for MLP trained using extracted features

Συγκρίνοντας τα αποτελέσματα από τους πίνακες 5.8 και 5.7, είναι σαφές ότι χρειαζόμαστε 8 tubes προσκηνίου για να λειτουργεί καλά ο ταξινομητής MLP. Ωστόσο, δεν είναι πολύ σαφές ποια από τις δύο προτεινόμενες εκπαιδευτικές διαδικασίες είναι καλύτερη, αλλά αν πρέπει να αποφασίσουμε μία μέθοδο, θα επιλέξουμε τη χρήση προϋπολογισμένων χαρακτηριστικών. Η προσέγγιση αυτή κατορθώνει να επιτύχει τα καλύτερα αποτελέσματα, και ειδικά όταν έχουμε 8 tubes προσκηνίου και 32 συνολικά. Επίσης, συγκρίνοντας τις μενδόδους με 4 ή 8 θετικά action tubes, είναι σαφές ότι θα προτιμούσαμε να χρησιμοποιούμε 8 γενικά. Ωστόσο, δεν είναι σαφές ποια αναλογία είναι καλύτερη, επειδή, έχουμε καλύτερα αποτελέσματα όταν έχουμε 8 action tubes και αναλογία 1:4 ενώ έχουμε καλύτερα αποτελέσματα όταν η αναλογία είναι 1:3 με 4 action tubes.

## Κεφάλαιο 6

# Επίλογος - Μελλοντικές επεκτάσεις

### 6.1 Επίλογος

Σε αυτή τη διατριβή εξερευνούμε το πρόβλημα της αναγνώρισης και του εντοπισμού της δράσης. Σχεδιάζουμε ένα δίκτυο βασισμένο στην προσέγγιση των Hou, ήσην και Σηαη 2017 σε συνδυασμό με ορισμένα στοιχεία από τους Girdhar et al. 2018, Ren et al. 2017, Girshick 2015, Hu et al. 2019 και Hara, Kataoka, and Satoh 2018.

Γράψαμε μια pytorch υλοποίηση παίρνοντας κώδικα μόνο από το Yang et al. 2017. Επιπλέον, γράψαμε τον δικό μας κώδικα χρησιμοποιώντας μερικές λειτουργίες της γλώσσας CUDA που έχουν σχεδιαστεί από εμάς (όπως υπολογισμός των βαθμολογιών σύνδεσης, τροποποίηση σωλήνων κλπ).

Προσπαθήσαμε να σχεδιάσουμε το TPN, ένα δίκτυο για την προτάσεις ToIs, ακολουθίες πλαισίων δηλαδή σε δεδομένο βίντεο, εμπνευσμένο από το RPN του Faster R-CNN. Το σχεδιάσαμε χρησιμοποιώντας γενικευμένα anchors και όχι συγκεκριμένα για κάθε σύνολο δεδομένων. Προσπαθούμε δηλαδή να γενικεύσουμε την προσέγγισή μας για διάφορα σύνολα δεδομένων, αντίθετα με την προσέγγιση που προτείνεται από τους Girdhar et al. 2018, στην οποία χρησιμοποιούνται τα πιο συχνά εμφανιζόμενα anchors για κάθε σύνολο δεδομένων.

Επιπροσθέτως, σχεδιάσαμε έναν αφελή αλγόριθμο σύνδεσης για τη σύνδεση των προτεινόμενων ToIs με βάση αυτόν που προτάθηκε από τους Girdhar et al. 2018. Στην προσέγγισή μας, χρησιμοποιούμε την ίδια πολιτική βαθμολόγησης, η οποία είναι ένας συνδυασμός των βαθμολογιών της πιθανότητας δράσης και του σκορ επικάλυψης. Η κύρια διαφορά είναι ότι αποφεύγουμε να υπολογίζουμε πιθανούς συνδυασμούς χρησιμοποιώντας ένα όριο ενημέρωσης, που μπορεί να ανανεώνεται. Επίσης, δοκιμάσαμε κι άλλον έναν αλγόριθμο σύνδεσης εμπνευσμένος από τους Hu κ.ά. 2019. Ωστόσο, η εφαρμογή μας δεν ήταν πολύ καλή, συνεπώς δεν εξερευνήσαμε όλες τις δυνατότητες του.

Τέλος, διερευνήσαμε αρκετούς ταξινομητές για το στάδιο ταξινόμησης του δικτύου. Αυτοί είναι: ένα RNN, ένας Γραμμικός ταξινομητής, ένα SVM και ένα MLP. Χρησιμοποιήσαμε μια εφαρμογή από το Φαστ RNN για τον ταξινομητή SVM, η οποία περιελάμβανε την διαδικασία εκπαίδευσης μέσω σκληρών αρνητικών. Εξετάσαμε μερικές τεχνικές εκπαίδευσης για βέλτιστη απόδοση ταξινόμησης και 2 εκπαιδευτικές προσεγγίσεις για τον ταξινομητή MLP, την κλασσική και μία που χρησιμοποιούμε προεξαγόμενα χαρακτηριστικά.

## 6.2 Μελλοντικές επεκτάσεις

Την παραπάνω πολλά περιθώρια βελτίωσης για το δίκτυο μας, προκειμένου να επιτευχθεί τελευταίας τεχνολογίας αποτελέσματα. Οι σημαντικότερες περιγράφονται στις επόμενες παραγράφους.

**Βελτίωση των προτάσεων του TPN** Τηλοποιήσαμε 2 δίκτυα για την πρόταση ακολουθιών από πλαίσια σε ένα τμήμα βίντεο. Πετύχαμε περίπου 63% βαθμολογία recall για τη διάρκεια του δείγματος ίση με 16 καρέ και περίπου 80% recall για τη διάρκεια του δείγματος ίση με 8. Αυτά τα σκορ δείχνουν ότι υπάρχει αρκετός χώρος για βελτίωση ειδικά για την περίπτωση με δείγμα 16 καρέ. Παρόλο που έχουν διερευνηθεί πολλές αρχιτεκτονικές δικτύων για παλινδρόμηση, μια καλή ιδέα θα ήταν να δοκιμάσουμε άλλα δίκτυα, δεν είναι απαραίτητη εμπνευσμένη από δίκτυα εντοπισμού αντικειμένων όπως κάνουμε εμείς. Επιπλέον, προσθέτοντας έναν παράγοντα λ στον τύπο του training loss θα ήταν μια καλή ιδέα και θα διερευνούσε ποια είναι η καλύτερη προσέγγιση αυτού. Έτσι, η απώλεια εκπαίδευσης θα μπορούσε να οριστεί ως:

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

Επιπλέον, θα ήταν μια καλή ιδέα να χρησιμοποιήσουμε την μέθοδο του SSD (Liu et al. 2015) που προτείνει RoIs αντί για το RPN, για να συγκρίνετε το αποτέλεσμα. Τέλος, θα μπορούσαμε να πειραματιστούμε χρησιμοποιώντας τα δίκτυα Feature Pyramid (Lin et al. 2017), τα οποία θα μπορούσαν να επεκταθούν σε 3 διαστάσεις ως ένα άλλο δίκτυο εξαγωγής χαρακτηριστικών ή κάποιο άλλο είδος 3D ResNet.

**Αλλαγή του αλγορίθμου σύνδεσης** Σε αυτή τη διατριβή, μια άλλη πρόκληση που αντιμετωπίσαμε ήταν η σύνδεση των προτεινόμενων ToIs για την πρόταση action tubes. Τηλοποιήσαμε έναν πολύ αφελή αλγόριθμο, που δεν ήταν σε θέση να μπορεί να μας δώσει πολύ καλές προτάσεις παρά τις αλλαγές που προσπαθήσαμε να κάνουμε. Τηλοποιήσαμε έναν άλλο αλγόριθμο σύνδεσης που ήταν βασισμένος στην εκτίμηση της χρονικής πρόοδο ενός action tube και την αλληλεπίδραση του με άλλα. Αν και δεν μας έδωσε και πολύ καλές προτάσεις, πιστεύουμε ότι πρέπει να εξερευνήσουμε τις δυνατότητες αυτού του αλγορίθμου. Κι αυτό επειδή είναι εκμεταλλεύεται την πρόοδο της ενέργειας, την οποία δεν είχε ο προηγούμενος αλγόριθμος.

**Εξερεύνηση άλλων τεχνικών ταξινόμησης** Για το στάδιο ταξινόμησης, πειραματιστήκαμε κυρίως πάνω σε έναν ταξινομητή SVM για το σύνολο δεδομένων JHMDB και δεν ασχολήθηκαμε καθόλου με το σύνολο δεδομένων UCF. Ο πρώτος μας στόχος είναι να είμαστε σε θέση να εξάγουμε καλά αποτελέσματα ταξινόμησης για το σύνολο δεδομένων UCF. Πιστεύουμε ότι θα πρέπει να διερευνήσουμε τους χάρτες χαρακτηριστικών του UCF και τεχνικές που εφαρμόζονται στους χάρτες χαρακτηριστικών πριν από την ταξινόμηση. Επιπλέον, θα μπορούσαμε να δοκιμάσουμε άλλες τεχνικές ταξινόμησης όπως Random Forests ή να πειραματιστούμε περισσότερο με τον ταξινομητή RNN για το σύνολο δεδομένων UCF. Τέλος, μια άλλη διαδικασία ταξινόμησης θα ήταν μια καλή ιδέα, όπως η εξαγωγή πρώτα όλων των πιθανών action tubes και, στη συνέχεια, η χρήση άλλων δικτύων για εξαγωγή χαρακτηριστικών προκειμένου να ταξινομήσουμε τα action tubes.

# **Chapter 7**

## **Introduction**

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

### **7.1 Problem statement**

The area of human action recognition and localization 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.

#### **7.1.1 Human Action Recognition**

Considering human action recognition, a video may be consisted of only by 1 person doing something. However, this is an ideal situation. In most cases, the 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 determine the temporal boundaries of each action.

#### **7.1.2 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.

## **7.2 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) creates the necessity to find ways to take advantage of them. About 2.5 billion images are uploaded in the Facebook database every month, more than 34K hours of video in YouTube and about 5K images every minute. On top of that,

there are about 30 million surveillance cameras in the US, which means about 700K video hours per day. All those data need to be separated into categories according to their content in order to search them more easily. This process takes place by hand, by a user who attaches keywords or tags to each video. However, most users avoid doing that, so many videos end up without any tagging information. This situation creates the need to create algorithms for automated indexing based on the content of the video.

Another application is video summary. This area takes place usually in movies or sports events. Regarding movies, video analysis algorithms can create a small video containing all the important moments of the movie. This can be achieved by choosing video segments which an important action takes place such as killing the villain of the movie. In sports events, video summary applications include creating highlight videos automatically, like a video containing all achieved goals in a football match.

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 humans, and immediately judge if there is any human behavioral anomaly.

Last but not least, another field of application is related to 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.

### 7.3 Challenges and Datasets

There are various types of human activities. Depending on their complexity, we conceptually categorize human activities into four different levels: gestures, actions, interactions, and group activities. Gestures are elementary movements of a person's body part, and are the atomic components describing the meaningful motion of a person. "Stretching an arm" and "raising a leg" are good examples of gestures. Actions are single person activities that may be composed of multiple gestures organized temporally, such as "walking", "waving", and "punching". Interactions are human activities that involve two or more persons and/or objects. For example, "two persons fighting" is an interaction between two humans and "a person stealing a suitcase from another" is a human-object interaction involving two humans and one object. Finally, group activities are the activities performed by conceptual groups composed of multiple persons and/or objects. "A group of persons marching", "a group having a meeting", and "two groups fighting" are typical examples of them. The wide variety of human activities and applications creates a lot of challenges which involve action recognition systems. The most important challenges include large variations in the appearance of the actors, camera viewpoint 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, sometimes, action vocabulary is not well defined. As figure 7.1 shows, "Open" action can include a lot of kinds of actions, so we must carefully choose which granularity of the action we will consider.

In order to deal with those challenges, several standard action datasets have been created in order to develop robust human action recognition systems and detection algorithms. The first datasets included 1 actor performing using a static camera over homogeneous backgrounds. Even though, those datasets helped 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 ambiguous videos, such as Joint-annotated Human Motion Database(JHMDB) (Kuehne et al. 2011) and UCF-101 (Soomro, Zamir, and Shah 2012). These datasets contain only human actions, the



Figure 7.1: Examples of “Open” action

second category presented above.

### 7.3.1 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 poses 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 .

### 7.3.2 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 spatiotemporal annotations are included. This means that there is a 2D bounding box surrounding the actor for each frame in which an action is taking place. We separate dataset’s videos into 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 frames, and for validation data, max number of frames is 900. Each video, both training and validation, is untrimmed, including sometimes more than 1 actions taking place simultaneously. We took annotations from Singh et al. 2017 because those proposed by the authors contain some mistakes.

## 7.4 Motivation and Contributions

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. 2018, Ren et al. 2017 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 implementation. Based on the structure proposed by Hou, Chen, and Shah 2017, we modified it, using a 3D Resnet34 instead of C3D, which previous approach used.

2. Furthermore, we proposed our own TPN Network, a Network for proposing candidate action tubes give a small video segment. Following the approach Hou, Chen, and Shah 2017 proposed, we firstly implement an architecture which uses cuboids as anchors, which then using a regressor it becomes a sequence of bounding boxes, likely to contain an action. We experiment with two candidate regressor's architecture and proposed and implement a 3D RoiAlign which uses trilinear interpolation for extracting each proposed action tube's activation maps. Inspired by Girdhar et al. 2018, we proposed and implement a TPN which uses predefined sequences of bounding boxes as 3D anchors. We proposed anchors that last equal to and less than video segment's duration in order our architecture to be able to perform temporal localization. we explore two different regressors' architectures for better spatial precision using activation maps extracted from 2D RoiAlign, treating each frame separately.
3. Inspired by linking algorithm proposed by Hou, Chen, and Shah 2017, we introduce our own linking algorithm, which uses a combination of actionness and overlap scores in order to decide if 2 proposed action tubes would connect or not and some updatable lists. Our approach includes gathering all candidate action tubes whose score is bigger than a threshold, and use them as active action tubes for new possible connections. When the number of gathered active action tube is bigger than a threshold, we keep the k-best scoring action tubes and remove the rest. We implement this algorithm using, also, CUDA code in order to calculate connection score faster. We proposed 3 versions of this algorithm:
  - (a) An approach which uses an updatable scoring threshold, in order not to calculate unnecessary connection scores
  - (b) An approach which doesn't use an updatable scoring threshold, but it just updates "active" action tube more frequently.
  - (c) An approach which, also, uses NMS or softmax-NMS algorithms for getting wider action tube proposals.
- Also, we implement, from scratch, another connection algorithm proposed by Hu et al. 2019 and extending it in order to work for ToIs instead of frames, which they proposed. We modified our TPN structure in order to calculate progression and progress rate scores in order to calculate connection scores and generate candidate action tubes.
4. We experiment using several classifier in order to find the most suitable. We considered 2 feature maps extracted using 3D RoiAlign and proposed action tubes, without any other modification. Also, we explore the different ratios and number of groundtruth foreground tubes that should be used during training stage. Finally, we tried to perform only temporal localization using temporal information generated from proposed action tubes.

## 7.5 Thesis structure

The rest of Thesis is organized as follows. Chapter 2 provides a 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 a 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, which 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 8

## Background

### 8.1 Machine Learning

#### 8.1.1 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 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 realization that the only way to be able to achieve this task was to let machines learn from themselves. 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 realize it while using it.

Finding patterns in data on planet earth is possible only for human brains. Data being very massive and time taken to compute them made Machine Learning take action, in order to help people exploit them 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  $Y = f(x)$ .

As shown in Figure 8.1, 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 new mails and checking if the model is able to predict the right output.

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

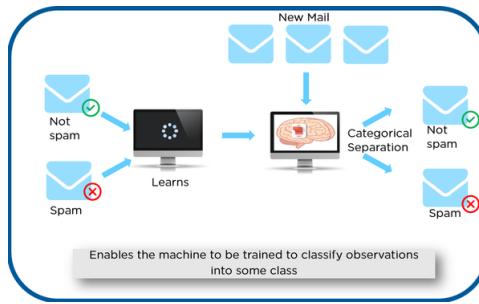


Figure 8.1: Example of supervised Learning

**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 8.2, we have given some characters to our model which are ‘Ducks’

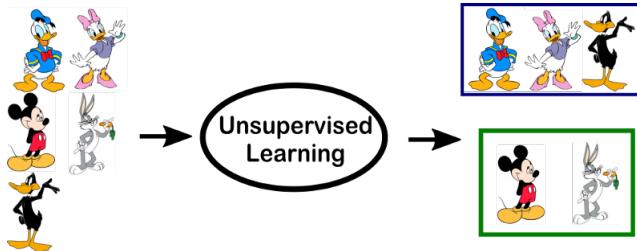


Figure 8.2: Example of unsupervised Learning

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 behavior.

### 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

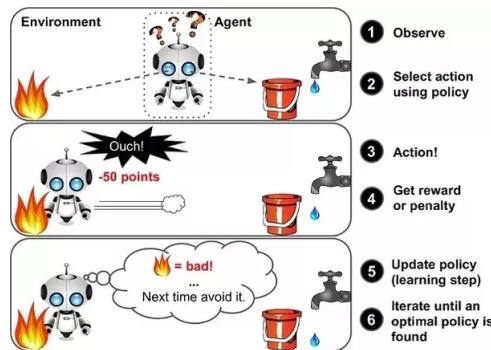


Figure 8.3: Example of Reinforcement Learning

Figure 8.3, 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

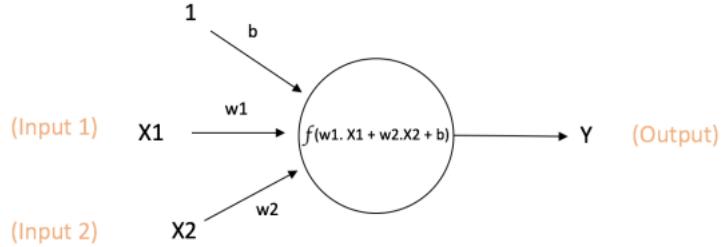
#### 8.1.2 Neural Networks

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.

#### 8.1.3 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 8.4. The network takes numerical inputs  $X_1$  and  $X_2$  and



$$\text{Output of neuron} = Y = f(w_1 \cdot X_1 + w_2 \cdot X_2 + b)$$

Figure 8.4: An example of a single Neuron

has weights  $w_1$  and  $w_2$  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 8.4. 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.

### 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.

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

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

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

Figure 8.5 show each of the above activation functions.

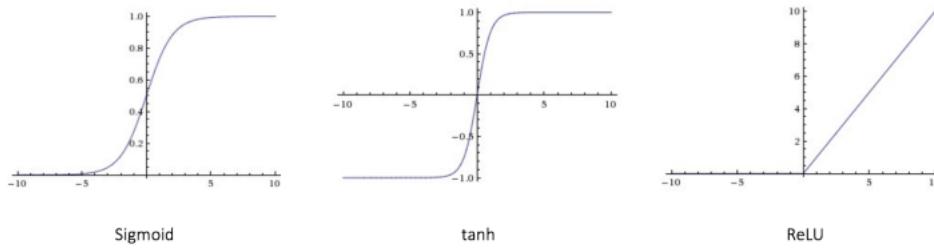


Figure 8.5: Plots of Activation functions

### Feed-forward 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 8.6. A feedforward neural 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.

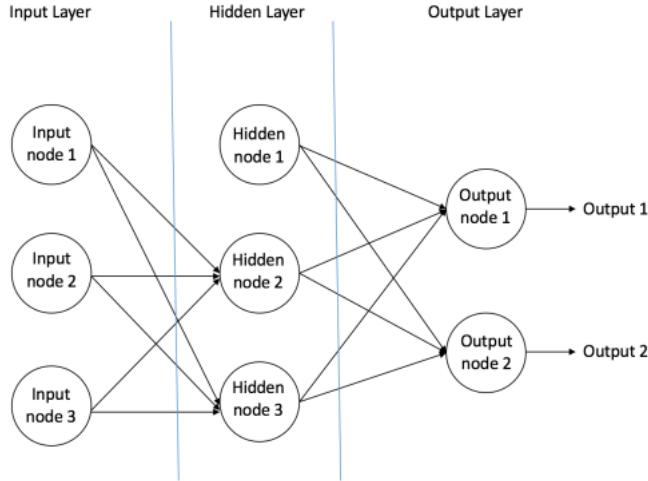


Figure 8.6: An example of a Feedforward Neural Network

**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.

#### 8.1.4 2D Convolutional Neural Network

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

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

- **Feature extractor :**

This part of the network takes as input the image and extract features that are meaningful

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

- **Classifier :**

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

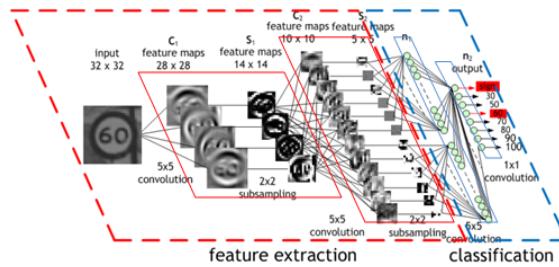


Figure 8.7: Typical structure of a ConvNet

**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 width and height), but extends through the full depth of input. During forward 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 8.8 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.

**Pooling Layers** Pooling Layers are also referred 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 reduced.
- Less parameters also means we deal with overfitting situations.



Figure 8.8: Convolution with kernel of 3, stride of 2 and padding of 1

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.

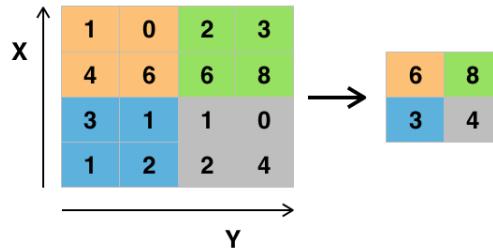


Figure 8.9: Example of Max pooling operation with a 2x2 filter and stride of 2

### 8.1.5 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, whose kernel shape for a 3D Convolution is specified along 3 dimensions. When thinking about the convolution operation in terms of a kernel sliding across a multidimensional input array, for a 3D Convolution, the kernel slides in 3 directions. Their output shape is a 3 dimensional volume space such as cube or cuboid.

Also, such 3D relationship is important for some applications, such as in 3D segmentations / reconstructions of biomedical imaging, e.g. CT and MRI where objects such as blood vessels meander around in the 3D space.

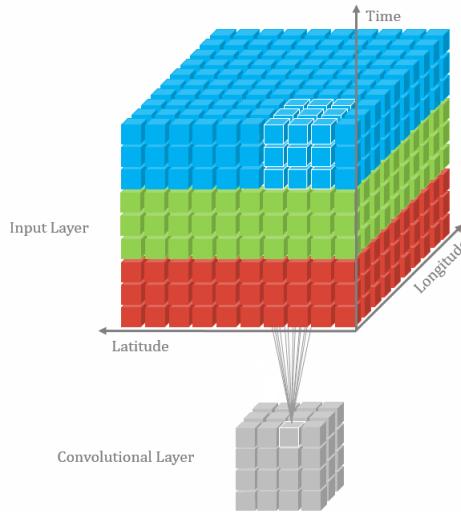


Figure 8.10: 3D Convolution operation

## 8.2 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 webcamera feed. The challenge of detecting all objects existing in image in counterpart of action localization in videos and a lot of object detection techniques are used in action localization architectures, so it is worth presenting it. Object Detection methods are used almost everywhere these days. The use cases are endless such as Tracking objects, Video surveillance, Pedestrian detection etc. An object detection model is trained to detect the presence and location of multiple classes of objects. For example, a model might be trained with images that contain various pieces of fruit, along with a label that specifies the class of fruit they represent (e.g. an apple, a banana, or a strawberry), and data specifying where each object appears in the image.

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

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

### 8.2.1 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.

The first Object-Detection CNNs use several algorithms for proposing ROIs. For example, R-CNN(Girshick et al. 2014), and Fast R-CNN(Girshick 2015) used Selective Search Algorithm for

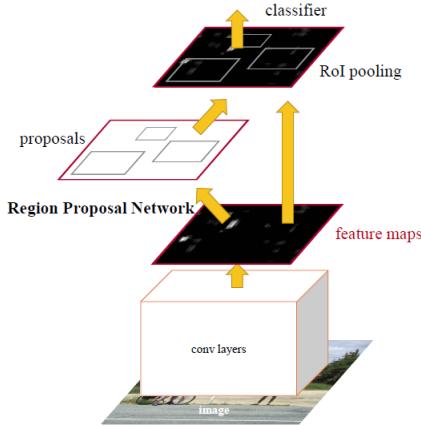


Figure 8.11: Region Proposal Network's structure

extracting ROIs. One of novelties introduced by the Faster R-CNN(Ren et al. 2017) is **Region Proposal Network** (RPN). Its function is to propose ROIs and its structure can be shown in 8.11. As we can see, RPN is consisted of:

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

Before describing RPN's function, we introduce another basic element of RPN which is its **anchors**. Anchors are predefined boxes used for extracting ROIs. In figure 8.12 is depicted an example of some anchors

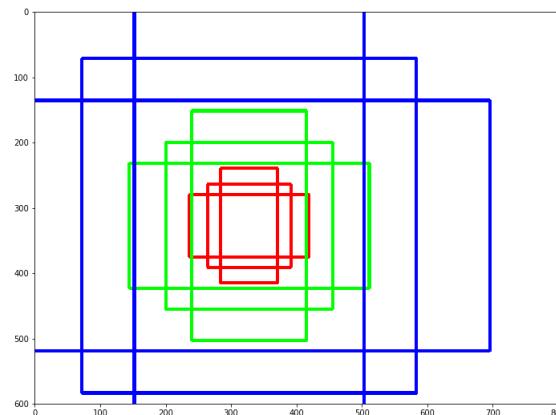


Figure 8.12: Anchors for pixel (320,320) of an image (600,800)

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

So, RPN's procedure is:

1. RPN gets as input feature maps extracted from the backbone CNN.
2. Then it performs 2D convolution over this input and passes the output to its scoring layer and regression layer.
3. Scoring layer produces confidence score of existing an object in each anchor's area. On the other hand, regression layer outputs 4k displacements, 4 for each anchor. Finally, we keep as output only the *n-best scoring* anchors.

### 8.2.2 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 handled by resizing the input image. However, in object recognition architectures, each proposal has a different size and shape. This creates the need for converting all proposals to a fixed shape. At Fast-RCNN(Girshick 2015) and Faster-RCNN(Ren et al. 2017) methods, 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 8.13a (the top left diagram). As a result, each target cells may not be in the same size (Figure 8.13c).

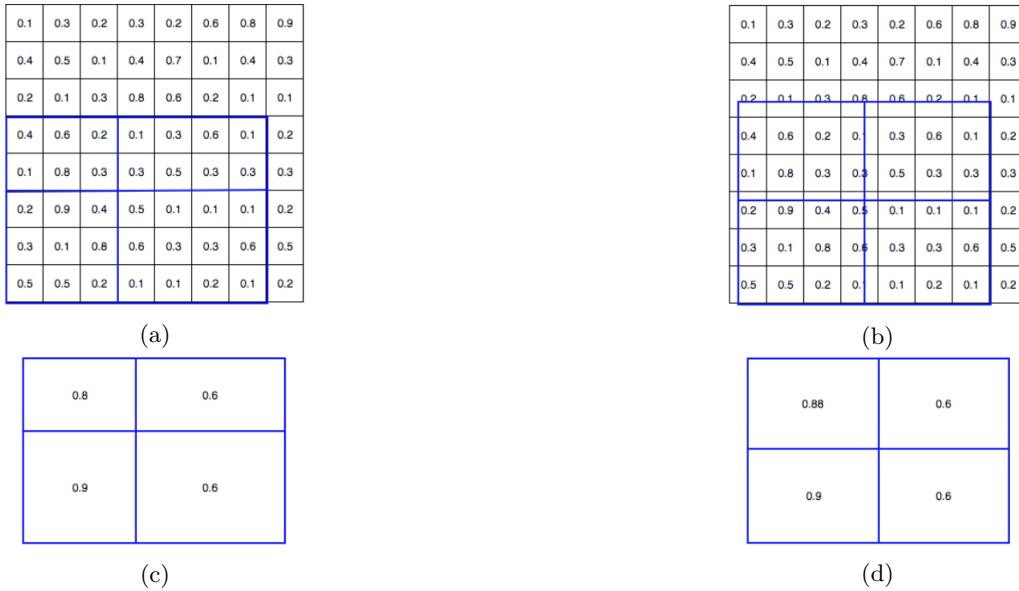


Figure 8.13: Roi Pooling and Roi Align examples

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 8.13b, and achieves to make every target cell to have the same size according to Figure 8.13d. In order to calculate feature maps values, Roi Align uses bi-linear interpolation as shown in Figure 8.14. This means that we calculate the value of the desired bins according to their neighbors'.

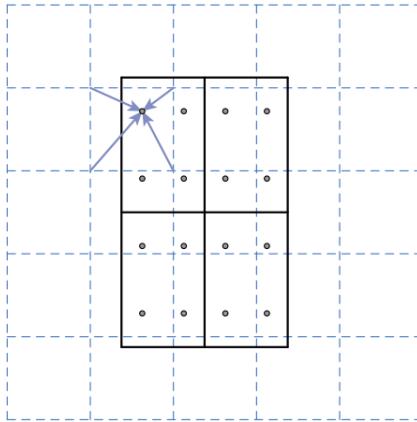


Figure 8.14: Example of bi-linear interpolation for calculation Roi Align’s final feature map

### 8.2.3 Non-maximum suppression (NMS) algorithm

Another problem that object detection networks face is that neighbor 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 areas in the images, achieve high recall performance. However, in these approaches, more than 1 proposal may be related with only one ground-truth object coordinates. This situation creates the need for choosing the best proposals, because, alternatively, hundreds of unnecessary 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 algorithm’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 single threshold value. So that makes the selection of threshold a crucial factor for the performance 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 8.15 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 efficient 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

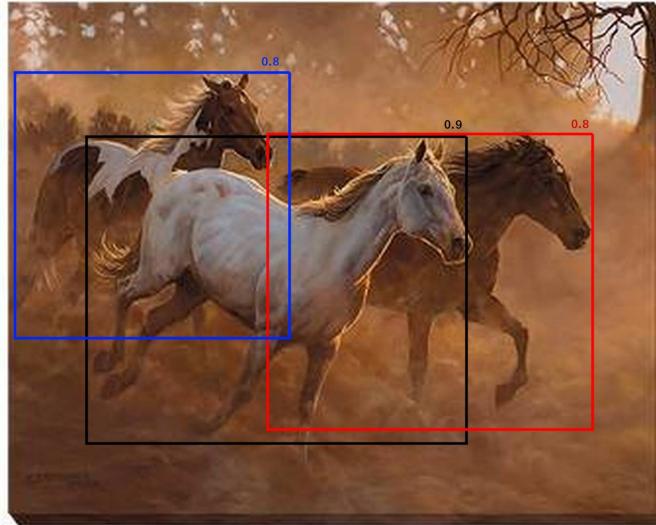


Figure 8.15: Example of a situation where NMS algorithm will remove good proposals

confidence score of the proposals proportional to their overlap score, instead of completely removing them. The score calculation follows the formula

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

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. Let's, again, consider as input a list of proposal bounding-boxes  $B$ , their corresponding confidence score  $S$  and as output a list of proposals  $D$ . Soft-NMS algorithm includes the following steps:

1. 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  $N$ , 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  $B$ .

## 8.3 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.

### 8.3.1 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 8.16.

### 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 associated 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 happen ( $p_E = 1$ ) we have 0 entropy (we are not surprised) and vice 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. 2017) and it is less sensitive to outliers according to Girshick 2015. As shown in Girshick 2015, its formula is:

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

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  $\delta$  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 8.16b shows a comparison between L1-norm, L2-norm and smooth-L1 .

### 8.3.2 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 classification algorithm. 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.

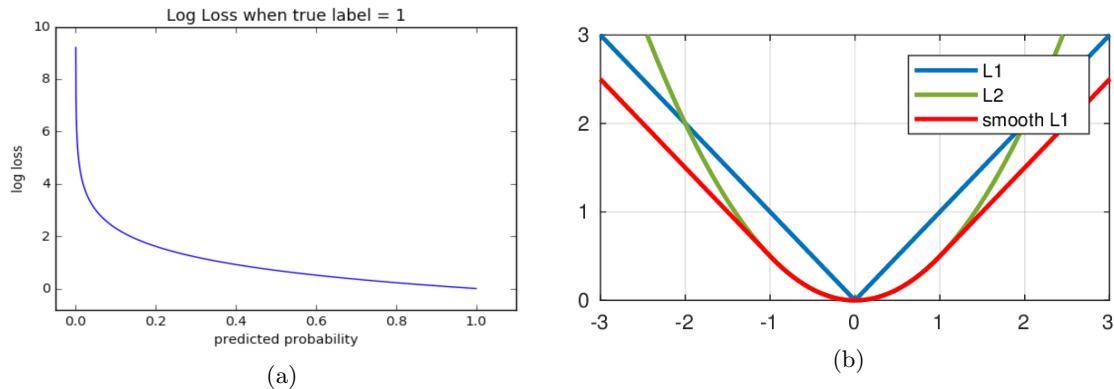


Figure 8.16: (a) and (b) show the behavior of cross-entropy loss and smooth-L1 respectively.

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

### 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 actual bounding box as shown in Figure 8.17. We predefine an IoU threshold (say 0.5) in classifying whether the prediction is a true positive or a false positive.



Figure 8.17: Example of IoU scoring policy

Intersection over Union is defined as:

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

In Figure 8.17, 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-dimensional 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:

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

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

**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 consider 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, given some data as input, is able to predict if a person is sick or not.

**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 well we found all the sick people. In our case, how many of the actual sick people we managed to find.

Their definitions are:

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

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

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 than a predefined 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, which for videos is named video-AP as introduced by Gkioxari and Malik 2015.

**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 mentioned before, during classification stage, our prediction results in True positive(TP), False positive(FP), True Negative(TN) or 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's class label was incorrect. We consider a prediction as False positive, when more than 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 at Table 8.1. 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 proportion 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 8.1: Ordered by confidence predictions and their precision and recall values

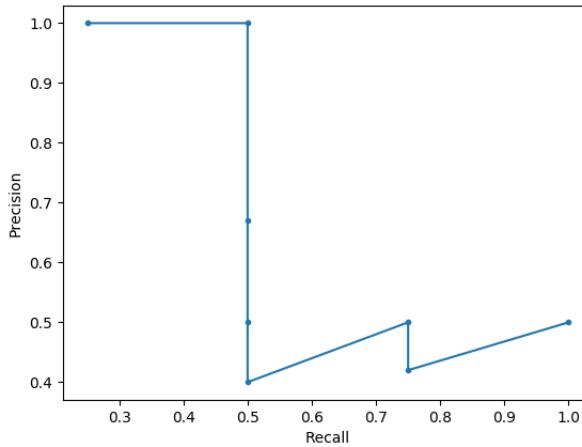


Figure 8.18: Precision/Recall curve

We plot Precision against Recall and their curve is shown in Figure 8.18. 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 replaced 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  $\Delta r$  is the change in recall from  $k - 1$  to  $k$ .

**Interpolated Precision** As we can see at Figure 8.18, 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 Interpolated 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 8.19 are shown both P-R curves. The previous P-R curve has blue color and the interpolated has red color.

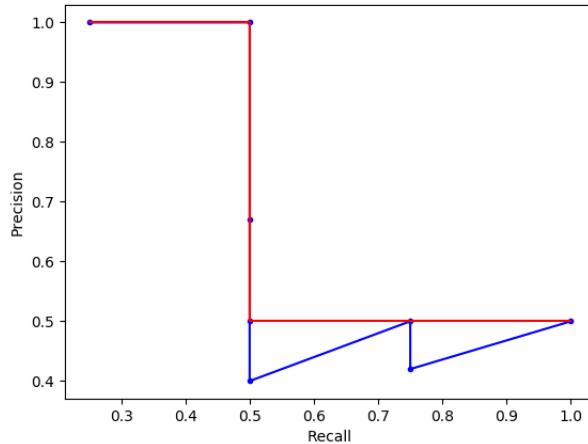


Figure 8.19: Both P-R curves. Interpolated P-R curve has red colour.

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 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 for how many actual objects/action tubes, there was at least 1 proposal that satisfied the detection criterion. However, it doesn't tell us how close these proposals are to the actual objects/action 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 8.20.

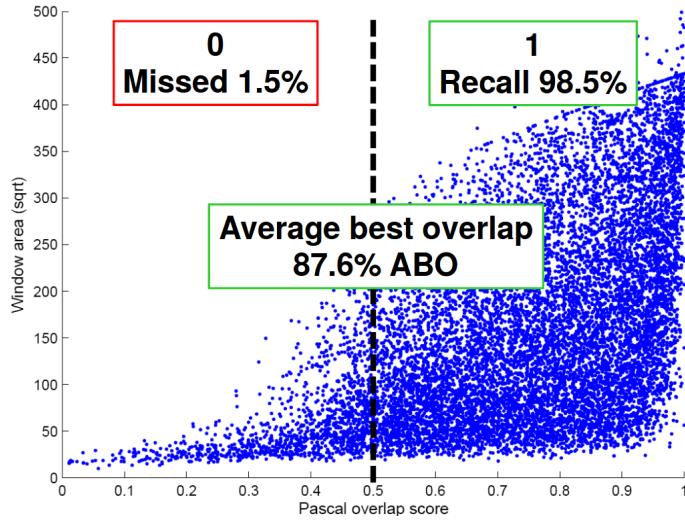


Figure 8.20: Recall versus MABO example

As we can see, recall performance is almost perfect, but, MABO performance, which tells us where most proposal overlap scores are gathered, is just fine and not perfect.

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 :

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

In our situation, which we care only for the quality of our proposals, we consider only 1 class, foreground class. As a result, MABO's performance identifies with ABO's.

## 8.4 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.

### 8.4.1 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 features can be separated into 3 categories: 1) space-time volume approaches, 2) trajectories and 3) space-time features. For space-time volume methods the approach is as follows: based on the training videos, the system constructs a 3D space-time model, by concatenating 2D images (x-y dimension) along time (t or z dimension), in order to represent each action. When the system is given an unlabeled video, it constructs a 3D space-time volume corresponding to this video. This new 3D volume, then, is compared with each activity model to measure the similarity in shape and appearance between these two volumes. The system extracts the class label of the unknown video by corresponding to the action with the highest similarity. Furthermore, there are several variations of space-time representations. Instead of volume representation, the system may represent the action as trajectories in space-time dimensions or even more, the action can be represented as a set of features extracted from the volume or the trajectories. Pure space-time volume representations include methods of comparing foreground regions of a person (i.e. silhouettes) like Bobick and Davis 2001 did, comparing volumes in terms of their patches like Shechtman and Irani 2005. Ke, Sukthankar, and Hebert 2007 method uses oversegmented volumes, automatically calculating a set of 3-D XYT volume segments that corresponds to a moving human. Rodriguez, Ahmed, and Shah 2008 proposed filters for capturing the volume's characteristics, in order to match them more reliably and efficiently. From the other hand, trajectory-based approaches include representing an action as a set of 13 joint trajectories (Sheikh, Sheikh, and Shah 2005) or using a set of XYZT-dimensions joint trajectories obtained from moving cameras (Yilmaz and Shah 2005). Finally, several methods use local features extracted from 3-dimensional space-time volumes, like extracting local features at every frame and concatenate them temporally (Chomat and Crowley 1999; Zelnik-Manor and Irani 2001; Blank et al. 2005, extracting sparse spatiotemporal local interest points from 3D volumes (Laptev and Lindeberg 2003; Dollar et al. 2005; Niebles, Wang, and Li 2006; Alper Yilmaz and Mubarak Shah 2005; Ryoo and Aggarwal 2006) These approaches made the choice of features a significant 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 designed their architecture based on the best-scoring CNN in the ImageNet competition. They explored several methods for the fusion of spatiotemporal features using 2D operations mostly and 3D convolution only in slow fusion. Simonyan and Zisserman 2014 used 2 CNNs, one for spatial information and one for optical flow and combined them using late fusion. They show that extracting 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. Furthermore, Wang et al. 2016 based their method on the one proposed by 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), separates the input video in K segments and a short snippet from each segment is chosen for analysis. Then they fuse the extracted spatiotemporal context, making, eventually, their prediction. Most recently, Zhang et al. 2016 and Zhu et al. 2017 used two-stream approach, too. Zhang et al. 2016 replace optical flow with motion vector which can be obtained directly from compressed videos without extra calculation and feeding it to the network . Zhu et al. 2017 trained a CNN for calculating optical flow, calling it MotionNet and used a CNN as temporal stream for project motion information to action labels. Finally, they use late fusion through the weighted averaging of the prediction scores of the temporal and spatial streams. On the other hand, a novel approach was introduced by Girdhar and Ramanan 2017 incorporating attention maps to give significant improvement in action recognition performance.

Some other methods included a RNN or LSTM network for classification like Donahue et al. 2017, Joe Yue-Hei Ng et al. 2015 and Ma et al. 2017. Donahue et al. 2017 address the challenge of variable lengths of input and output sequences, exploiting 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. Joe Yue-Hei 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 of them. Furthermore, they propose a recurrent neural Network architecture in order to consider video clips as a sequence of CNN activations. Proposed LSTM takes as input the output of the final CNN layer at each consecutive 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 an 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 into 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. 2015 explored 3D Convolutional Networks (Ji et al. 2013) and introduced C3D network, which has 3D convolutional layers with kernels  $3 \times 3 \times 3$ . This network is able to model appearance 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. 2016) and explore the effectiveness of ResNet with 3D Convolutional kernels. On the other hand, Diba et al. 2017 based their approach on DenseNets(Huang et al. 2017) 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. On top of

that, Diba et al. 2018 introduced a new temporal layer that models variable temporal Convolution kernel depths. Last but not least, Tran et al. 2018 experimented 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. Recently Guo et al. 2018 proposed a framework which can learn to recognize a previous unseen 3D action class with only a few examples by exploiting the inherent structure of 3D data through a graphical representation. A more detailed presentation for Action Recognition techniques used until 2018 is presented by Kong and Fu 2018.

#### 8.4.2 Action Localization

As mentioned before, Action Localization can be seen as an extension of the object detection problem. Instead of outputting 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 usually including an object-detector network and a classifier.

The first object detection approaches included extending a object proposal algorithm into 3-dimensions. Tian, Sukthankar, and Shah 2013 extended deformable part models (Felzenszwalb et al. 2010) by treating actions as spatiotemporal patterns and generated a deformable part for each action. Jain et al. 2014 introduced the concept of tubelets, aka sequences of bounding boxes and based their method on the selective search algorithm (Uijlings et al. 2013), extending superpixels to super-voxels for producing spatiotemporal shapes. On the other hand, Oneata et al. 2014 extend a randomized superpixel merging procedure which was used for object proposals as presented by Manen, Guillaumin, and Gool 2013. Yu and Yuan 2015 first propose bounding boxes for each frame using a human and motion detector and then by picking the best-scoring bounding boxes, they proposed a greedy linking algorithm by formulating linking task as a maximum set coverage problem. Gemert et al. 2015 generate spatiotemporal proposals directly from dense trajectories, which also used for classification. Chen and Corso 2015 create a spatiotemporal trajectory graph and select action proposals based only on intentional movement extracted from the graph. Soomro, Idrees, and Shah 2015 separate the video segments into supervoxels and use their context as a spatial relation between supervoxels relative to foreground action. They create a graph for each video, where supervoxels form the nodes and directed edges capture the spatial relations between them. During testing, they perform a context walk where each step is guided by the context relations learned during training, resulting in a probability distribution of an action over all the supervoxels. Mettes, Gemert, and Snoek 2016, instead of annotating boxes in all frames, annotate points on a sparse subset of video frames, and use proposals obtained by an overlap measure between action proposals and points. Behl et al. 2017 deal with on-line action detection and localization by getting per-frame action proposal and proposing a linking algorithm which is able to construct and update action tubes at each frame. Most recently, Soomro and Shah 2017 tried to deal with the problem of unsupervised action detection and localization. Their approach included extracting supervoxel segmentation and then assigning a weight to each supervoxel. Using extracted supervoxels, they create a graph and then using a discriminate clustering approach a classifier is trained.

The introduction of R-CNN (Girshick et al. 2014) achieved 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 an SVM classifier. Inspired by this architecture, Gkioxari and Malik 2015 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.

The introduction of Faster RCNN (Ren et al. 2017) contributed a lot to the improvement of the performance of Action Localization Networks. Peng and Schmid 2016 and Saha et al. 2016 use Faster R-CNN 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 and from each proposed ROI, generate 4 ROIs in order to focus on 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 appearance and motion based proposals and their overlap score. Finally, temporal localization takes place using dynamic programming. Additionally, Weinzaepfel, Martin, and Schmid 2016 use Faster RCNN for extracting human tubes from videos focusing on weakly-supervised action localization problem. Then, using dense trajectories and a multi-fold Multiple Instance Learning approach (Cimbis, Verbeek, and Schmid 2016) they train a classifier. Mettes and Snoek 2017 introduced a method for zero-shot action localization. Their approach includes scoring proposed action tubes according to the interactions between actors and local objects. They used Faster-RCNN, in the first step, for detecting both actors and objects and then using spatial relations between them, they link the proposed boxes over time based on zero-shot likelihood from the presence of actors, relevant objects around the actors and the expected spatial relations between objects and actors. Furthermore, He et al. 2018 proposed the Tube Proposal Network (TPN) for generating generic class-independent tubelet proposals, which uses Faster-RCNN for getting 2D region proposals and a linking algorithm for linking tubelets with these region proposals. Most recently, Girdhar et al. 2018 proposed a method for action Localization on the AVA dataset (Gu et al. 2018) combining I3D (Carreira and Zisserman 2017) and Faster-RCNN architectures. They use I3D blocks for getting video representation and Fast-RCNN’s RPN for generation “person” proposals for the center frame.

On top of that, Singh et al. 2017 and Kalogeiton et al. 2017 designed their networks based on the Single Shot Multibox Detector Liu et al. 2015). Singh et al. 2017 created an real-time spatiotemporal network. In order their network to execute real-time, Singh et al. 2017 proposed a novel and efficient algorithm by adding boxes in tubes in every frame if they overlap over 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, 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. 2016) became the inspiration for Hu et al. 2019 and Zhang et al. 2016. In approach proposed by 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, separated into classes. Some other approaches include pose estimation like Luvizon, Picard, and Tabia 2018 do. They proposed a method for calculating 2D and 3D poses and then they performed action classification. They use the differentiable 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  $N_j$  joints as  $x - y$  dimensions and having 2 channels 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. Zolfaghari et al. 2017 proposed a three-stream architecture which includes 2D pose, optical flow and RGB information. These streams are integrated sequentially via a Markov chain model. In addition, Zhu, Vial, and Lu 2017 proposed an architecture using a temporal convolutional regression network, for capturing the long-term dependency and contexts among adjacent frames, and a spatial regression network, getting per-frame proposals. They use tracking methods and dynamic programming for generating action proposals.

Most of aforementioned networks use per-frame spatial proposals and extract their temporal information by calculating optical flow. On the other hand, Saha, Singh, and Cuzzolin 2017 and Hou, Chen, and Shah 2017 designed an architecture which includes proposals in video segment level, which mean more than 1 frame simultaneously. Saha, Singh, and Cuzzolin 2017 proposed a 3D-RPN which is able to generate and classify 3D region proposals consisted of two successive frames. Also, they proposed a linking algorithm, modifying the one proposed by Saha et al. 2016. On top of that, Hou, Chen, and Shah 2017 designed an architecture for generating action proposals for more than 2 frames, which they called Tube CNN (T-CNN). In their approach, video segment level means that the whole video is separated into equal length video clips, and using a C3D for extracting features, it returns spatiotemporal proposals. After getting proposals, Hou, Chen, and Shah 2017 link the tube proposals by an algorithm based on tubes' actionness score and overlap. Finally, classification operation is performed for the linked video proposals.



# Chapter 9

## Tube Proposal Network

### 9.1 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 implementation we use a 3D Resnet network whose implementation is taken from Hara, Kataoka, and Satoh 2018 and it is based on ResNet CNNs for Image Classification (He et al. 2016).
- Tube Proposal Network for proposing ToIs (based on the idea presented in Hou, Chen, and Shah 2017).
- A classifier for classifying proposed action video tubes.

The basic procedure ActionNet follows is:

1. Given a video, we separate 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 these sequences as Tubes of Interest (ToIs), like Hou, Chen, and Shah 2017 did because they are likely to contain a person performing an action.
3. After getting proposed ToIs for each video segment, using a linking algorithm, ActionNet finds final candidate 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 9.1.

### 9.2 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

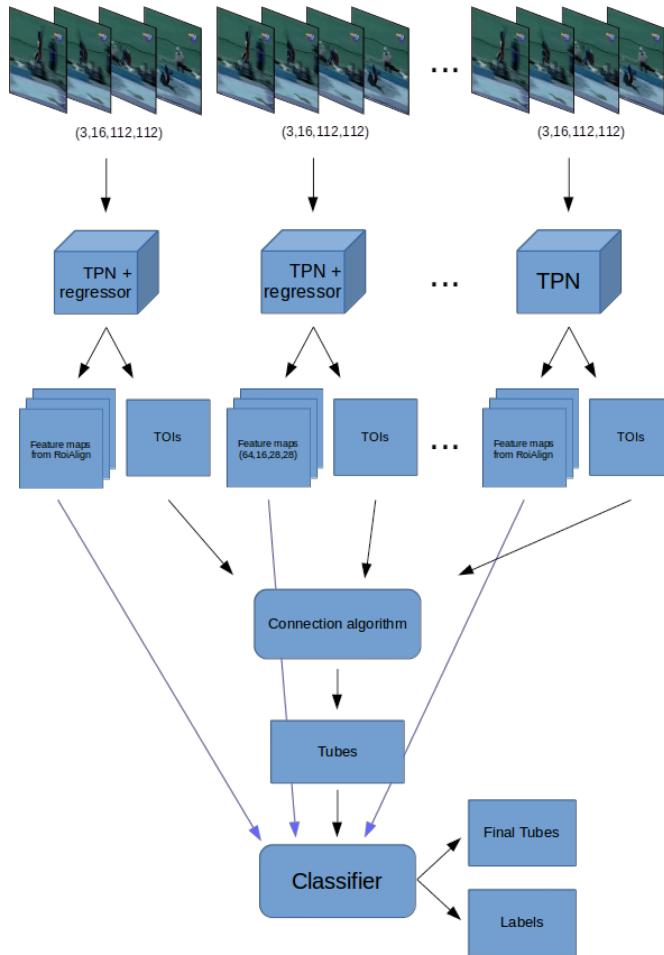


Figure 9.1: The structure of the whole network

each frame). TPN is inspired from RPN introduced by FasterRCNN (Ren et al. 2017), but instead of images, TPN is used in videos as performed by 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.

## 9.3 Preparation before TPN

### 9.3.1 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 sequence of frames which has a fixed width, height and duration. However, each video has a different resolution. That's creates the need to resize each frame before feeding it

to the architecture. As mentioned in the previous chapter, the first element of our network is a 3D ResNet 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 below depending which dimension is bigger. In figure 9.2 we can see the original frame and the resize and padded one. In full correspondence, we resize the groundtruth bounding boxes for each frame (figure 9.2b and 9.2d show that).

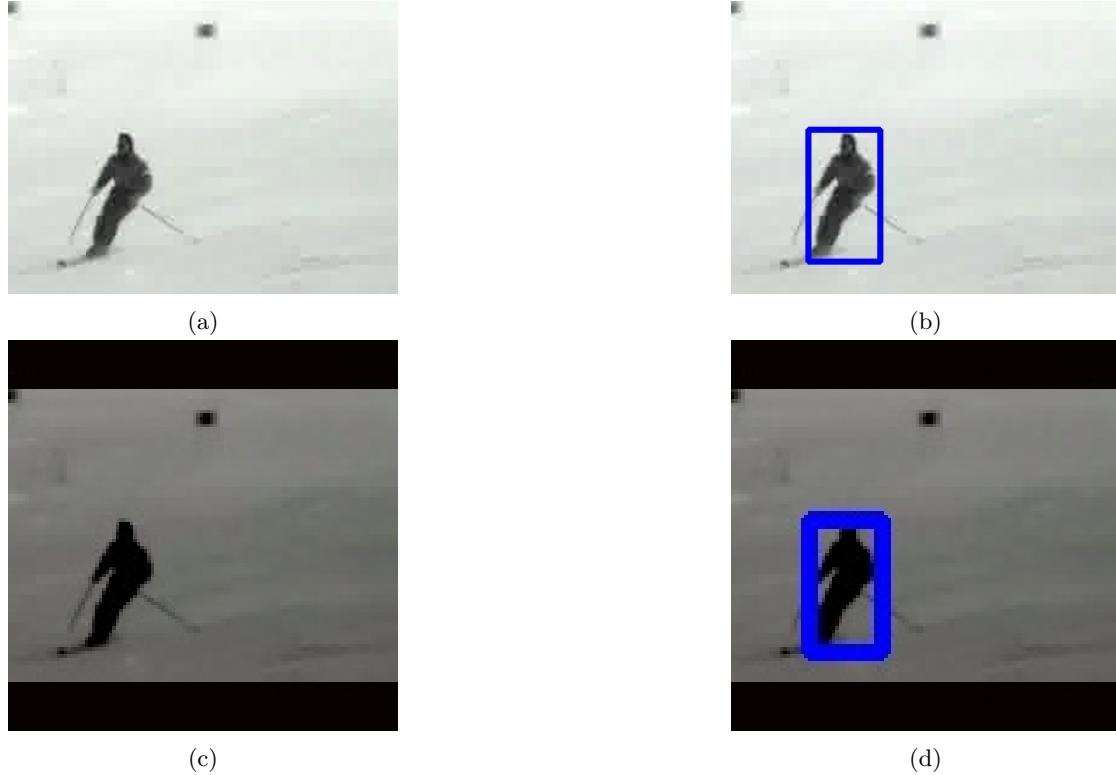


Figure 9.2: At (a), (b) frame is its original size and at (c), (d) same frame after preprocessing part

### 9.3.2 3D ResNet

Before using the Tube Proposal Network, we extract spatiotemporal features of the video. In order to do so, we extract the 3 first Layers of a pretrained 3D ResNet34. This Network is pretrained in Kinetics dataset Kay et al. 2017 for sample duration equal to 16 frames and sample size equal to (112, 122).

This network normally is used for classifying the whole video, so some of its layers use temporal stride equal to 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 dimensions (256,16,7,7). We feed this feature map to TPN, which is described in the following sections.

## 9.4 3D anchors as 6-dim vector

### 9.4.1 First Description

We started designing 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 cuboid and  $x_2, y_2, t_2$  are the lower back left as shown in figure 9.3.

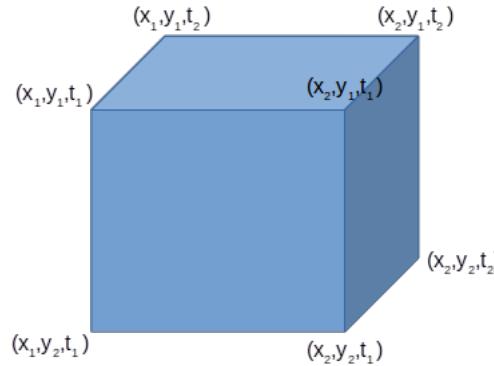


Figure 9.3: An example of the anchor  $(x_1, y_1, t_1, x_2, y_2, t_2)$

The main advantage of this approach is that, except from x-y dims, the 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 anchor 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.

The structure of TPN is shown in figure 9.4. The output of TPN is the k-best scoring cuboids, which are most likely to contain an action.

### 9.4.2 Training

As mentioned before, TPN extracts TOIs as 6-dim vectors. For that reason, we modify our groundtruth ROIs to groundtruth Tubes. We take for granted that the actor cannot move a lot during 16 frames, so that's why we use this kind of tubes. As shown in figure 9.5, these tubes are 3D boxes which include all the groundtruth rois, which are different for each frame.

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

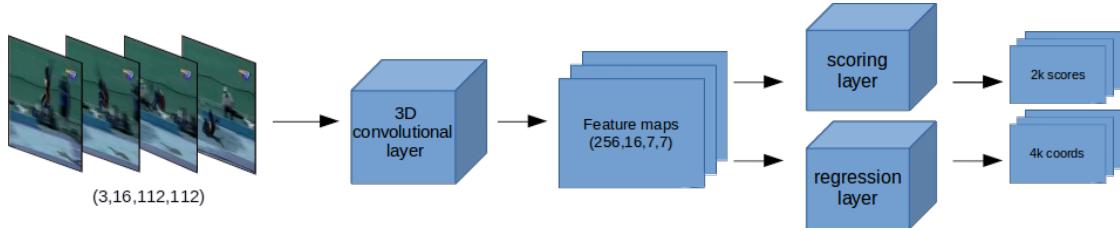


Figure 9.4: Structure of TPN

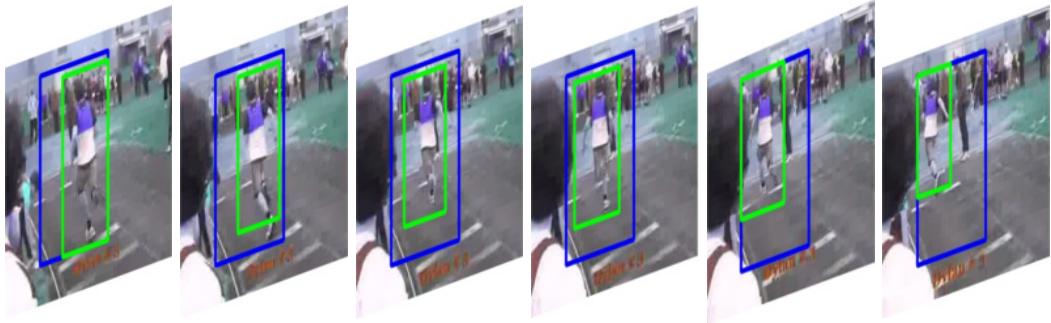


Figure 9.5: Groundtruth tube is colored with blue and groundtruth rois with color green

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 Cross Entropy Loss as loss function. We have a lot of anchors for proposing an action, but a small number of per-video 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 also necessary, the anchors 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 from Girshick et al. 2014. 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:

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

where  $x, y, z, w, h, d$  denote the 3D box's center coordinates and its width, height and duration. Variables  $x, x_a$ , and  $x^*$  are for the predicted box, anchor box, and groundtruth box respectively (likewise for  $y, z, w, h, d$ ). 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 regression 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 is multiplied with  $p_i^*$  in order to be set active only when there is a positive anchor ( $p_i^* = 1$ ) and to be deactivated for background anchors ( $p_i^* = 0$ ).

### 9.4.3 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  $\geq 0.5$  with each groundtruth action tube and we get recall score. In order to get good proposals, after getting classification scores and target prediction from the corresponding layers, we use Non-Maximum Suppression (NMS) algorithm. We set NMS threshold equal to 0.7, and we keep the first 150 cuboids with the biggest score.

### 9.4.4 Modified Intersection over Union(mIoU)

During training, we get numerous anchors. We have to classify them as foreground anchors or background anchors. Foreground anchors are those which contain some action, and, respectively, background don't. As presented before, IoU for cuboids calculates the ratio between the 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 dimensions to have the same importance with time dimension, which we do not desire. That's because firstly we care to be accurate in time dimension, and then we can fix x-y domain. As a result, we change the way we calculate the Intersection Over Union. We calculate separately 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:

$$\begin{aligned} IoU_{xy} &= \frac{\text{Area of Overlap in x-y}}{\text{Area of Union in x-y}} \\ IoU_t &= \frac{\max(t_1, t'_1) - \min(t_2, t'_2)}{\min(t_1, t'_1) - \max(t_2, t'_2)} \\ IoU &= IoU_{xy} \cdot IoU_t \end{aligned}$$

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 overlapped in time dimension. 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 9.1 we can see the performance in each case for both datasets, JHMDB and UCF. The recall threshold for this case is 0.5 and during validation, we use regular IoU for defining if 2 tubes overlap.

Table 9.1 shows that modified-IoU give us slightly better recall performance only in UCF dataset. That's 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.

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

Table 9.1: Recall results for both datasets using IoU and mIoU metrics

#### 9.4.5 Improving TPN score

After first tests, we came with the idea that in a video lasting 16 frames, in time domain, all kinds of actions can be separated into the following categories:

1. The action starts in the n-th frame and finishes after the 16th frame of the sampled video.
2. The action has already begun before the 1st frame of the video and ends in the n-th frame.
3. The action has already begun before the 1st frame of the video and finishes after the 16th video frame.
4. The 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 than 16 frames. So, we came with the idea to add 1 scoring layer and 1 regression layer which will propose ToIs with fixed duration equal to the sample duration (16 frames) and they will take into account the spatial information produced by activation maps. The new structure of TPN is shown in figure 9.6. After getting proposals from both scores, we concat them with ratio 1:1 between ToI extracted from those 2 subnetworks.

Our goal is to “compress” feature maps in the temporal dimension in order to propose ToIs based only on 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 losses obtained from 2 different systems which propose TOIs. On top of that, during validation, we, at first, concate proposed ToIs and, then, we follow the same procedure, which is calculating recall performance. For training loss, we have 2 different cross-entropy losses and 2 different smooth-L1 losses, each for every layer correspondingly. So training loss is, now, defined as :

$$L = \sum_i L_{cls}(p_i, p_i^*) + \sum_i L_{cls}(p_{fixed,i}, p_{fixed,i}^*) + \sum_i p_i^* L_{reg}(t_i, t_i^*) + \sum_i p_{fixed,i}^* L_{reg}(t_{fixed,i}, t_{fixed,i}^*) \quad (9.1)$$

where:

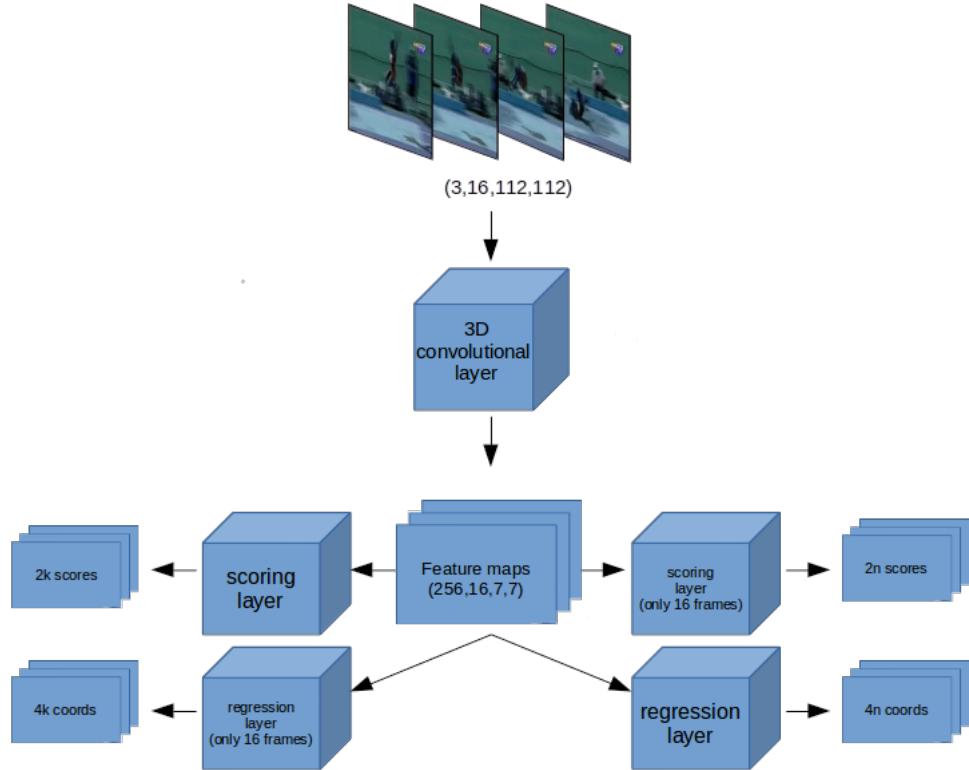


Figure 9.6: TPN structure after adding 2 new layers, where  $k = 5n$ .

- $L_{cls}$  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 and regression layers with mutable time duration and  $p_i^*$  are their corresponding groundtruth label.
- $p_{fixed,i}$  are the anchors from scoring and regression layers with 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 shown in Table 9.2.

As we can see from the previous results, the new layers increased recall performance significantly. On top of that, Table 9.2 shows that getting the average values from the time dimension gives us the best results.

#### 9.4.6 Adding regressor

The output of TPN is the  $\alpha$ -highest scoring anchors moved according to their regression prediction. After that, we have to turn the proposed anchors into ToIs. In order to do so, we

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 9.2: Recall results after adding fixed time duration anchors

add a regressor system which gets as input cuboids' feature maps and returns a sequence of 2D boxes, one per every frame. The only problem is that the regressor needs as input feature maps with fixed size . This problem is already solved 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(He et al. 2017), and we call it **3D Roi Align**.

**3D Roi Align** 3D Roi align is a modification of roi align presented by Mask R-CNN . The main difference between those two is that Mask R-CNN's roi align uses bi-linear 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 activation map has size equal to (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  $\delta_x, \delta_y$  specify the coordinates of proposal's center and  $\delta_w, \delta_h$  its width and height, as specified in Girshick et al. 2014. 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:

$(0, 0, 0, 0, \dots, x_{T_1}, y_{T_1}, x'_{T_1}, y'_{T_1}, \dots, x_i, y_i, x'_i, \dots, x_{T_2}, y_{T_2}, x'_{T_2}, y'_{T_2}, 0, 0, 0, 0, \dots)$ ,  
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 means 25% of the total number of the tubes as happened previously. We extract their corresponding features using 3D Roi Align 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 defined as:

$$\begin{aligned}
 L = & \sum_i L_{cls}(p_i, p_i^*) + \sum_i L_{cls}(p_{fixed,i}, p_{fixed,i}^*) + \\
 & \sum_i p_i^* L_{reg}(t_i, t_i^*) + \sum_i p_{fixed,i}^* L_{reg}(t_{fixed,i}, t_{fixed,i}^*) + \\
 & \sum_i q_i^* L_{reg}(c_i, c_i^*)
 \end{aligned} \tag{9.2}$$

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 overlap with any groundtruth action tube more than 0.5 .

### First regression Network

The architecture of regression network is shown in Figure 9.7, and it is described below:

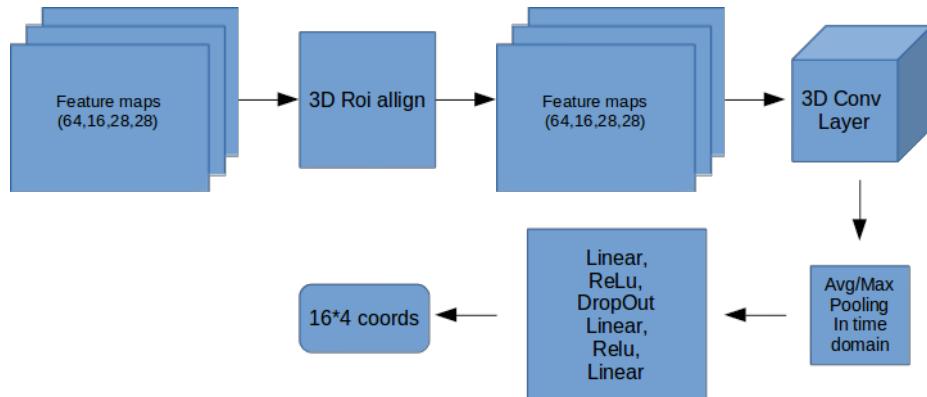


Figure 9.7: Structure of Regressor

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 from 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, followed 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 performances:

**Cuboid Recall**, which is the recall performance 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 performance 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 use 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 9.3: Recall results after converting cuboids into sequences of frames

As the above results show, we get lower recall performance 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, although 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.

#### 9.4.7 Changing Regressor - from 3D to 2d

After getting first recall results, we experiment using another architecture for the regressor network, in order to solve the modification problem, introduced in the 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 9.8, the 2<sup>nd</sup> 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 the 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.

On top of that, we tried to determine which feature map is the most suitable for getting best-scoring recall performance. This feature map will be given as input to the Roi Algin operation. At Table 9.4, 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.

## 9.5 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  $(x_1, y_1, x'_1, y'_1, x_2, y_2, \dots)$  where  $x_1, y_1, x'_1, y'_1$  are the coordinates for the 1st frame,  $x_2, y_2, x'_2, y'_2$  are the coordinates for the 2nd frame etc, as presented in Girdhar et al. 2018. In figure 9.9 we can an example of this type of anchor.

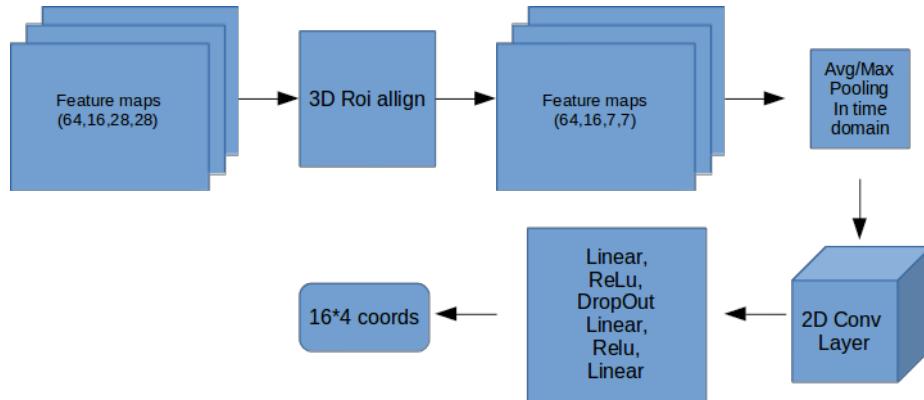


Figure 9.8: Structure of Regressor

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 9.4: Recall performance using 3 different feature maps as Regressor's input and 2 pooling methods

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 coordinates in the frames bigger than 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 the sample duration is 4 frames.

This new approach led us to change the structure of TPN. The new one is presented in figure 9.10. As we can see, we added scoring and regression layers for each duration. So, TPN follows the upcoming steps in order to propose ToIs:

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

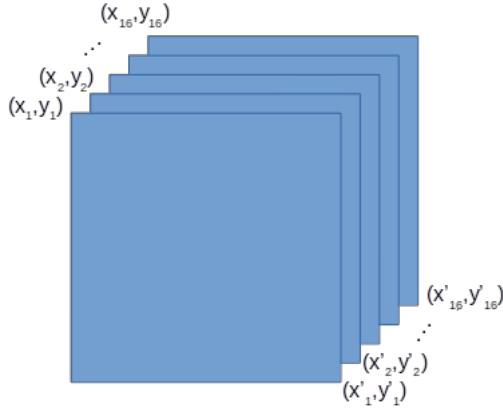


Figure 9.9: An example of the anchor  $(x_1, y_1, x'_1, y'_1, x_2, y_2, \dots)$

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  $(256,1,7,7)$ ,  $(256,5,7,7)$ ,  $(256,9,7,7)$  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  $\mathbf{n} = \mathbf{k} = 15$  anchors (5 scale of 1, 2, 4, 8, 16, 3 aspect ratios 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  $7 \times 7$  shape in each filter. So, in total we have  $28 \cdot 15 \cdot 49 = 20580$  different anchors. Respectively, we have 20580 different regression targets.

### 9.5.1 Training

Training procedure stays almost the same like previous approach's. So, again, we randomly choose a video segment and its corresponding groundtruth action tubes. But in this training procedure, we consider anchors as foreground when they overlap more than 0.7 with any groundtruth tube, alongside with background anchors whose overlap is bigger than 0.1 and smaller than 0.3. We are not concerned about the rest of the anchors.

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 9.5: Recall results using 2nd approach for anchors

As Table 9.5 shows, it is obvious that we get better recall performances compared to previous' approach. Additionally, we can see that 3D max pooling performs better than 3D avg pooling. The difference between max pooling and avg pooling is about 10%, which is big enough to make

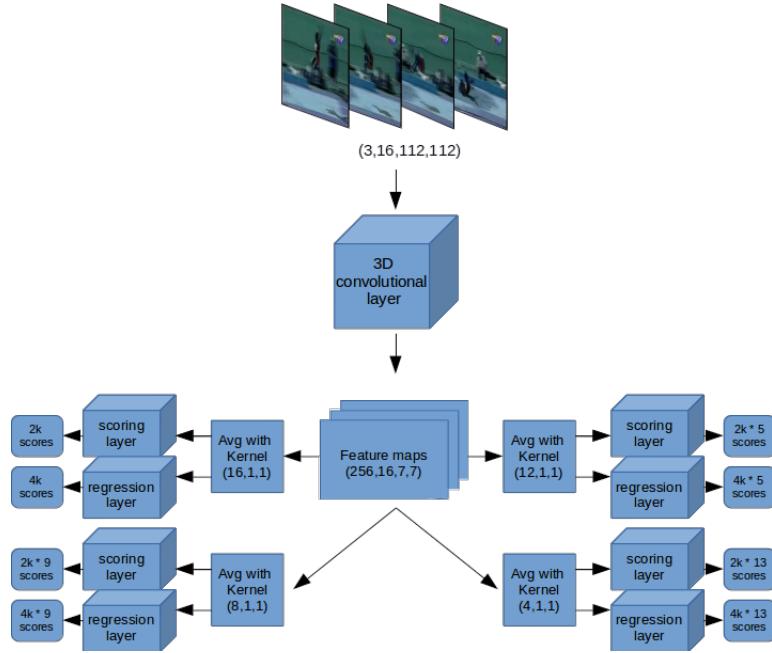


Figure 9.10: The structure of TPN according to new approach

us choose max pooling operation as pooling method before getting anchors' scores and regression targets.

### 9.5.2 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 correspondence 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 from Girdhar et al. 2018. They extend RoiAlign operator by splitting the tube into  $T$  2D boxes. Then, they use classic RoiAlign to extract a region from each one 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 layers. 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 activation between them and finally we get  $sampduration \times 4$  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:

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

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 9.6: Recall performance when using a 3D Convolutional Layer in Regressor's architecture

According to Table 9.6, 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 spatiotemporal 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 the results from Tables 9.5 and 9.6.

### 9.5.3 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 haven't used any pooling operation before feeding the first 2D Convolutional Layer. On the contrary, we manipulate our feature maps like not being spatiotemporal 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 in Table 9.7.

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 9.7: Recall performance when using a 2D Convolutional Layer instead of 3D in Regressor's model

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

#### 9.5.4 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 a smaller number of cases, so smaller number of parameters alongside with a 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 9.8. 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 9.5.

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 9.8: Recall results when reducing sample duration to 4 and 8 frames per video segment

According to Table 9.8, we notice that we get best performance for sample duration = 8 for both datasets. For dataset JHMDB sample duration equal to 8 gets far better results from the other approaches, followed by approach with sample duration = 4. For UCF dataset, although

sample duration equal to 8 gives us best performances sample duration equal to 4 gives us about the same. The difference between those 2 duration is less than 1%.

### With Regressor

Following the idea of reducing the 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 in Table 9.9.

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
JHMDB	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 9.9: Recall results when a regressor and sample duration equal to 4 or 8 frames per video segment

According to 9.9, it is clear that using a 2D Convolutional Layer as presented above results in better recall performance than using a 3D. Furthermore, 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 reduces 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 9.8 and 9.9, we think that the best approach is using sample duration equal to 8, with the addition of a regressor, which uses a 2D Convolutional Layer. We know that this approach 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 aforementioned approach in the rest chapters.

## 9.6 General comments

In the previous section, we presented 2 different approaches for proposing sequences of bounding boxes which are likely to include an actor performing an action. After considering both approaches, we deduce that the second approach results in better proposals according to their recall performance. In Figure 9.11, 3 different generated ToIs are presented.

Our goal is the bounding boxes to overlap with the actor precisely. According to Figure 9.11, all of three proposed actions tubes presented in Figures 9.11a, 9.11b and 9.11c overlap with the actor to some degree. However, by looking, it is clear that ToI, presented in Figure 9.11b, doesn't overlap very well like those appearing at Figures 9.11a and 9.11c. Intuitively, comparing proposals from 9.11a and 9.11c, we think that 9.11a overlaps better than 9.11c.

However, as shown in Figures 9.11d and 9.11e, the second ToI better overlaps with the groundtruth. It is clear that even though the first proposal overlaps very well with the upper body of the actor, it fails to capture the left leg, which may be a crucial factor for determining the class of the action. The ToI shown in Figure 9.11e manages to capture most of the actor's



Figure 9.11: Visualization of ToIs including ground truth action tubes, too

body, excluding only the head, which usually doesn't move a lot during an action. Although, judging intuitively, we would choose the first ToI, in reality, the second one is more preferable.

# Chapter 10

## Connecting Tubes

### 10.1 Introduction

In the previous chapter, we described methods for generating candidate action tubes given a small video segment lasting 8 or 16 frames. However, actual videos and actual human actions, in the wild, last more than 16 frames most of the times. Current networks are unable to process a whole video at once, in order to generate action tubes due to memory and computing power issues. As mentioned in chapter 2, a lot action localization approaches deal with this situation by giving a video either propose candidate areas in the frame-level and then they connect these in order to generate action tube proposal either, separate it into video segments, proposing sequences of bounding boxes for each video segment and then link them in order to generate action proposals. Both aforementioned techniques make the suitable choice of linking method an important factor for the performance of the network. That's because, even though frame-level or video segment-level proposals might be very good, if the proposed connection algorithm doesn't work well, final action tube proposals won't be efficient, so the final model will never be able to achieve high classification performance. In other words, if connecting algorithm doesn't generate action proposals with great recall and MABO performance, the model's classifier won't be able to perform suitable classification, because probably it would be given action tubes without any context. In this chapter, we present 3 different approaches used for linking proposed ToIs generated from TPN in the previous chapter.

### 10.2 First approach: combine overlap and actionness

Our algorithm is inspired by Hou, Chen, and Shah 2017, which calculates all possible sequences of ToIs. In order find the best candidate action tubes, 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 bounding boxes 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 10.1.

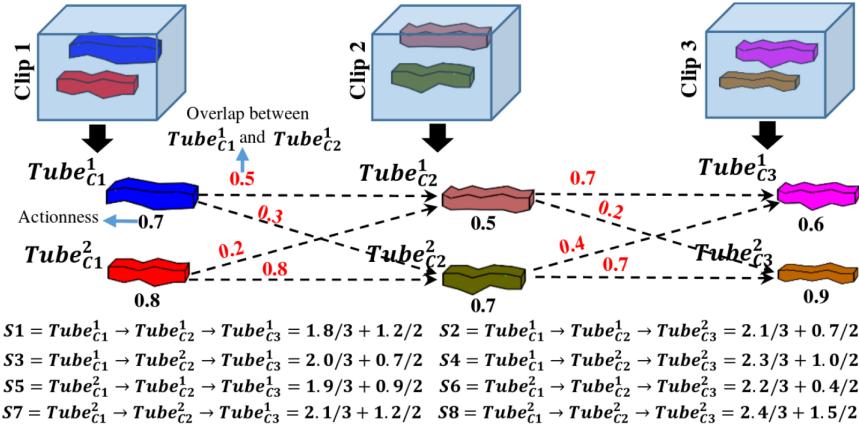


Figure 10.1: An example of calculating connection score for 3 random TOIs taken from Hou, Chen, and Shah 2017

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 segment, we propose  $k$  TOIs (16 during training and 150 during validation). As a result, for a small video separated into **10 segments**, we need to calculate **150<sup>10</sup> 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. Intuitively, this algorithm after getting a new video segment, keeps tubes with a score higher than a threshold, and deletes the rest. So, we don't need to calculate combinations with a 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' duration, their scores, active tubes, their corresponding duration, active tubes' overlapping sum and actionness 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 matched with the new TOIs. Their overlapping sum list and actionness sum list contain their sums in order to avoid calculating them for each loop.

Also, we initialize connection 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 overlapping 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 a score higher than the connection threshold, we add both to final tubes and to active tubes and their corresponding lists, and we increase their duration.
4. If the number of active tubes is higher than a threshold, we set connection 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, alongside with their actioness scores in the actioness sum list and zero values in corresponding positions in the 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 corresponding bounding boxes. However, 2 succeeding ToIs do not, always, have exactly the same bounding boxes in the frames that overlap. For example, ToIs from the 1<sup>st</sup> video segment start from frame 1 to frame 16. If we have video step equal to 8, these ToIs overlap temporally 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  $(x'_1, y'_1, x'_2, y'_2)$ .

### 10.2.1 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 metric. 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 the 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 equal to 16 frames** 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 10.1 (in bold are the frames with which we calculate the overlap score).

combination	overlap thresh		
	0.5	0.4	0.3

$0,1,\dots,\{8,\dots,15\}$	0.3172	0.4142	0.6418
$\{8,9,\dots,15\},16,\dots,23$			
$0,1,\dots,\{8,\dots,11,\}14,15$	0.3172	0.4142	0.6381
$\{8,\dots,11\},12,\dots,22,23$			
$0,1,\dots,\{10,\dots,13\}14,15,$	0.3209	0.4179	0.6418
$8,9,\{10,\dots,13\},14,\dots,22,23$			
$0,1,\dots,\{12,\dots,15\}$	0.3284	0.4216	0.6381
$8,9,\dots,\{12,\dots,15\},16,\dots,23,$			
$0,1,\dots,\{8,\dots,11\},14,15,$	0.3172	0.4142	0.6381
$\{8,9,\dots,11\},12,\dots,22,23$			
$0,1,\dots,\{10,\dots,13\}14,15,$	0.3209	0.4179	0.6418
$\{10,\dots,13\},14,\dots,22,23$			
$0,1,\dots,\{12,\dots,15\}$	0.3284	0.4216	0.6381
$8,9,\dots,\{12,\dots,15\},16,\dots,$			
$0,1,\dots,\{8,9\},10,\dots,14,15,$	0.3134	0.4104	0.6381
$\{8,9\},10,11,\dots,22,23$			
$0,1,\dots,\{10,11\},12,\dots,14,15,$	0.3209	0.4216	0.6418
$8,9,\{10,11\},12,\dots,22,23$			
$0,1,\dots,\{12,13\},14,15,$	0.3246	0.4179	0.6418
$8,9,\dots,\{12,13\},14,\dots,22,23$			
$0,1,\dots,13,\{14,15\}$	0.3321	0.4216	0.6306
$8,9,\dots,\{14,15\},16,\dots,22,23$			

Table 10.1: 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 result, 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 10.2.

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

Table 10.2: Recall results for step = 12

As we can see, recall performance is increased 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 in Table 10.3

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 10.3: 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 equal to 8** We wanted to confirm that we get the best results, when we have no overlapping frames and step = sample duration. So Table 10.4 shows recall performance for sample duration = 8 and video step = 4 and Table 10.5 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 10.4: 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 10.5: Recall results for steps = 6, 7 and 8

According to Tables 10.4 and 10.5, it is clearly shown that, we achieve the best results, for *step = sampleduration* and overlapping scores is calculated between the last box of the current tubes and the first box of next tubes.

### 10.2.2 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 use less memory and we moved most parts of the code to the GPU. This happened by using tensors instead of lists for scores while most operations are, from now on, matrix operations. On top that, the last step of the algorithm, which is the modification from indices to actual action tubes is written, now, in CUDA code so it takes place in the GPU, too. So, we are now able to increase the number of ToIs 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 to 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 ToIs, our final network proposes 500, 2000 and 4000 candidate action tubes for sample durations equal to 8 and 16 frames. For the sample duration equal to 8 we return 100 ToIs because, when we tried to return 150 proposed ToIs, we got OutOfMemory error. Table 10.6 show the spatiotemporal recall and MABO performance of those approaches. Furthermore, Table 10.7 show their temporal recall and MABO performance. We are interested in temporal performance, because UCF consists 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 tubes that contain an action, at least from the temporal perspective. In order to calculate the temporal overlap, we use IoU for 1 dimension as described before.

combination	TPN tubes	Final tubes				MABO
			0.5	0.4	0.3	
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 10.6: Recall results for UCF dataset

combination	TPN tubes	Final tubes	overlap thresh			MABO
			0.9	0.8	0.7	
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 10.7: Temporal Recall results for UCF dataset

According to Table 10.6, we achieve better recall and MABO performance when we set sample duration equal to 16. In all cases, recall performance of simulations with sample duration equal to 16 outweigh 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 10.6 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 to 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 the sample duration equal to 8, the one in which TPN returns 100 ToIs and our network proposes 4000 action tubes, and for the sample duration equal to 16, the one in which, TPN returns 30 ToIs and the network 4000 action tubes.

Additionally, Table 10.7 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 results when network has 16 frames as sample duration, too. However, unlike Table 10.6, Table 10.7 shows that when we increase TPN's number of proposed ToIs, it increases performances for both sample durations. For sample duration equal to 8, this increase results in improving recall performances by 2% and MABO performance by 1% like spatiotemporal recall and MABO. For the sample duration equal to 16, recall performance is increasing by about 8% and MABO by 1%-2%.

Taking both tables into consideration, 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 describes the performances of network's proposals for variations in the number of TPN's returned ToIs, number of returned proposed action tubes and sample duration. For each situation, we choose the k-best scoring action tubes, without taking into account any relation between these action tubes, like their spatiotemporal 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 spatiotemporal 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 to 30 and the number of final picked action tubes equal to 4000. NMS algorithm uses a threshold in order to choose if 2 action tubes overlap enough. We experiment setting this threshold equal to 0.7 and 0.8 and the results are shown in Table 10.8 for Spatiotemporal performance and at Table 10.9 for temporal performance.

combination	NMS thresh	PreNMS tubes	overlap thresh			MABO
			0.9	0.8	0.7	
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 10.8: Spatiotemporal Recall results for UCF dataset

combination	NMS thresh	PreNMS tubes	overlap thresh			MABO
			0.9	0.8	0.7	

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 10.9: Temporal Recall results for UCF dataset

Comparing Table 10.8 with Table 10.6, we notice that NMS algorithm improves recall and MABO performance when NMS threshold is equal to 0.9. When we set it equal to 0.7 or 0.8, we get worse results. This happens, probably, because nms algorithm removes some good proposals. Comparing these results with results presented at Tables 10.6 and 10.7 it becomes clear that using NMS algorithm is very useful. That's because, even though we get the same number of proposed action tubes, these tubes are not very close spatiotemporally, so this makes proposed action tubes more likely to contain actual foreground action tubes.

#### 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 consideration, we came to the conclusion that sometimes, 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 than the a specific number. The rest algorithm remains the same. Tables 10.10 and 10.11 show spatiotemporal and temporal, recall and MABO performance respectively. We experiment for cases in which either we don't use the NMS algorithm at all, either we set overlap threshold equal to 0.7 and 0.9 as shown below.

combination	NMS thresh	PreNMS tubes	overlap thresh			MABO
			0.9	0.8	0.7	
0,1,...,6,{7,} {8,}9,...,15	-	20000	0.3779	0.5316	0.6471	0.393082961
	0.7		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	-	20000	0.438	0.5635	0.6829	0.4231788
	0.7		0.4525	0.5848	0.7034	0.429747438
	0.9		0.3802	0.5133	0.6068	0.3862278851848662

Table 10.10: Spatiotemporal Recall results for UCF dataset

combination	NMS thresh	PreNMS tubes	overlap thresh			MABO
			0.9	0.8	0.7	

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	-	20000	0.8327	0.9156	0.9399	0.940143272
	0.7		0.8646	0.9369	0.9567	0.946701832
	0.9		0.6183	0.7696	0.8388	0.8628507037919737

Table 10.11: Temporal Recall results for UCF dataset

Comparing recall and MABO performances shown at Table 10.10 with those included in Tables 10.8 and 10.6, we deduce that for the sample duration equal to 8, stopping updating linking threshold results in worse performance when we set nms threshold equal to 0.7, but it achieves the best performances when setting NMS threshold equal to 0.9 . Furthermore, for sample duration equal to 16, we get, now, best performance performance when we set nms threshold equal to 0.7 and worse performance for nms threshold equal to 0.9 .

### Soft-nms instead of nms

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 threshol. We experiment for the sample duration equal to 8 and thresholds equal to 0.7 and 0.9, because, our implementation ran out of memory for sample duration equal to 16. Recall and MABO performance are presented in Tables 10.12 and 10.13

combination	NMS thresh	PreNMS tubes	overlap thresh			MABO
			0.9	0.8	0.7	
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 10.12: Spatiotemporal Recall results for UCF dataset using Soft-NMS

combination	NMS thresh	PreNMS tubes	overlap thresh			MABO
			0.9	0.8	0.7	
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 10.13: Temporal Recall results for UCF dataset using SoftNMS

Taking results at Tables 10.12 and 10.13 into consideration, alongside with those at Tables

10.10 and 10.11 for sample duration equal to 8, we notice that using softNMS results in slightly better results. This happens when we set overlapping threshold equal to 0.9, otherwise, for overlapping threshold 0.7, we get worst performance. Despite the fact that softnms results in better recall and MABO performance, our implementation is very slow, which means that for a 201-frame video, softNMS part lasts about 32 seconds on the contrary with standard NMS algorithm without updating linking threshold, in which this part last only 2 seconds. So, we choose to use the standard NMS algorithm without updating linking threshold as an algorithm for removing overlapping action tubes.

### 10.3 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 based on the one proposed by Hu et al. 2019. This algorithm introduces two new 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 actionness 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 action class has been performed.

So, each action tube is described 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 separately, so for the rest section, 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 ). Correspondingly, 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 criteria:
  - i.  $overlapscore(at_i^{(k)}, t_j^{(k)}) < \theta, at \in AT^{(k)}, t \in TOIs^{(k)}$
  - 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:

The new confidence 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 a groundtruth tube as positive if there is at least one proposed tube that overlaps with it 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 10.14: Recall results for second approach with step = 8, 16 and their corresponding steps

According to Table 10.14, we get the best performance when we set sample duration equal to 16 and overlap step equal to 12. Comparing this performance with the first approach, for both sample durations equal to 8 and 16, we notice that second approach falls short comparing to the first one.

## 10.4 Third approach : use naive algorithm - only for JHMDB

As mention in the first approach, Hou, Chen, and Shah 2017 calculate all possible sequences of ToIs in order to find the best candidates. We rethought about this approach and we concluded that it could be implemented 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 more memory efficient writing some parts using 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 experiments which have sample duration equal to 8 and 16 frames and we modify the video step each time. For sample duration = 8, we return only 15 ToIs and for sample duration = 16, we return 30 because, if we return more, we get "out of memory error". In the following 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 10.15: Recall results for second approach with

From the above table, firstly, we confirm that when video step is equal to the sample duration gives us the best recall results. Also, we notice that when sample duration is equal to 16 frames recall gets slightly better than when sample duration is equal to 8. However, using 16 frames per video segment sample increases the memory usage even though it reduces the number of video segments, because of the need to process bigger videos, bigger feature maps etc. So for the classification stage we will experiment using mostly sample duration equal with 8 frames.

## 10.5 General comments



Figure 10.2: Example of connected tubes

Figure 10.2 shows the example presented in chapter 3, after linking first video segment's first ToI with a ToI proposed for the second video segment. For this case, we used the third proposed method, including calculating all possible combinations. As shown in the Figure, our algorithm manages to track the actor performing the action efficiently enough. This means that even though the actor moves during the video, our approach manages not to loose contact with him. On top of that, it clear that the silhouette of the actor changes, and so does the area of proposed action tubes. The only problem which appears is that proposed action tubes sometimes exceeds the area of the actual video. In order to deal with this problem, we set bounding boxes not to exceed these areas by keeping the limits of the original picture. So, from now on, no bounding box will overlap with padding area.

# Chapter 11

## Classification stage

### 11.1 Introduction

In previous 2 chapters, we introduced the procedure we used to create candidate action tubes, which probably contain some action or may not. Most of the times, the proposed action tubes belong to the background, and for that reason, as mentioned and in the previous chapter, it is important to choose a good linking algorithm that generates good sequences of bounding boxes. However, it is quite important to choose the appropriate classifier who will be able very accurately predict whether a candidate action tube belongs to a known category of actions or it belongs to the background. This is because we may produce good proposals for candidate actions, but if our classifier is unable to distinguish them, our system will again fail to recognize the actions.

The right choice of a classifier is a big dilemma we are facing and we need to answer. However, this classifier will get as input some activation maps in order to be classified. Therefore, apart from the good selection of the classifier, equally, good choice of features is important, too. Finally, the training process of the classifier plays a major role in order to be able to generalize and to avoid overfitting situations.

For our implementation, we implement approaches, including a Linear Classifier, a Recursive Neural Network (RNN) Classifier, a Support Vector Machine (SVM) Classifier and a Multilayer perceptron (MLP). Additionally, we experiment using feature maps obtained from 3D RoiAlign using, also, avg or max pooling. Last but not least, we try to find the right ratio between foreground and background action tubes, including their total number needed during training stage in order our classifier to perform efficiently.

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

1. Separate video into small video clips and 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 its feature maps into the classifier in order to perform classification.

The general structure of the whole network is depicted in figure 11.1, in which we can see the aforementioned steps if we follow the arrows.

For classification step we experiment only with dataset JHMDB. That's because we managed to achieve good recall performance only for JHMDB's videos, on the contrary with UCF-101. For dataset UCF-101, we managed to generate good action tube proposals in less than half

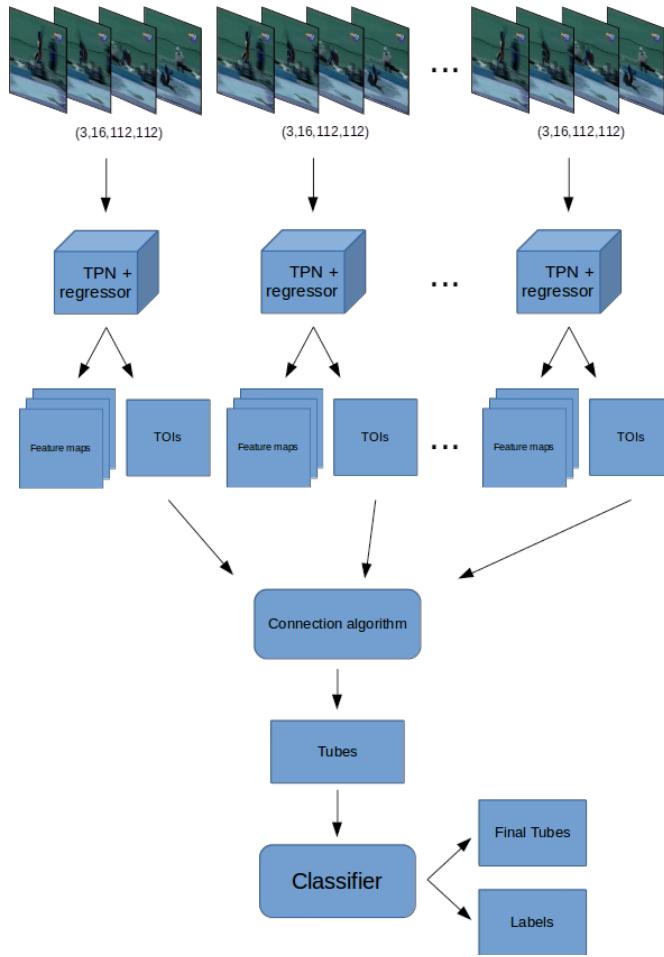


Figure 11.1: Structure of the whole network

situations. So, our system won't be able to perform well not because of the chosen classifier, but because of the lack of good proposals.

## 11.2 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 11.2 below, imagining each rectangle has a vectorial depth and other special hidden quirks in the image below. In our case, we choose **many to one** approach, because we want only one prediction, at the end of the action tube.

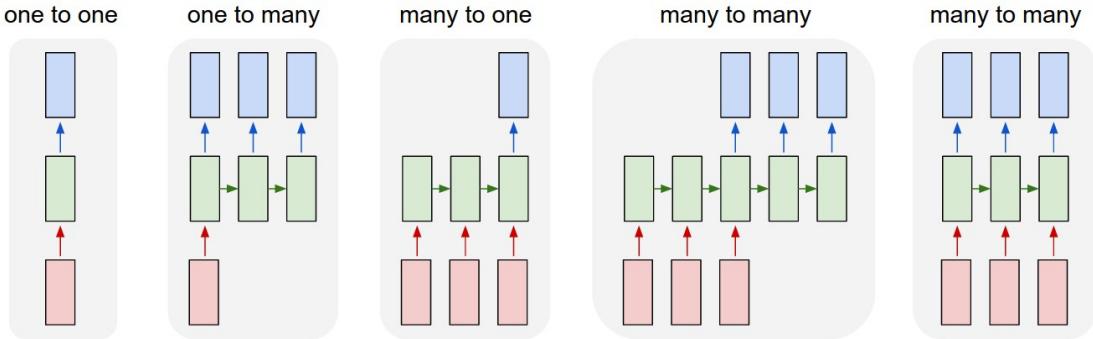


Figure 11.2: Types of RNN

**Training** In order to train our classifier, we have to execute the steps, presented in the first section, for each video. However, each video has a different number of frames and reserves too much memory on the GPU. In order to deal with this situation and considering there are 4 GPUs available, 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 foreground and background video tubes, and then to feed those features to our classifier for training it in order to discriminate classes. This solution includes the following steps:

1. At first, we extract only groundtruth action tubes' features. Also, we extract feature maps from background action tubes, which are double the number of groundtruth action tubes. We chose this ratio between the number of positive and negative tubes inspired by Yang et al. 2017, in which it has a 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 action tubes in total, 1 for positive and 2 for background. We considered background tubes those whose overlap scores with groundtruth tubes are  $\geq 0.1$  and  $\leq 0.3$ . Of course, in order to get those action tubes, we use a pre-trained TPN to generate ToIs for each video segment and then our proposed connection algorithm for linking proposed ToIs. Finally, we get each action tube's corresponding feature map using 3D RoiAlign
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 number of 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 experimented using both max and avg pooling as shown at Table 11.1. 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 threshold 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 one 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 11.1: First classification results using Linear and RNN classifiers

Table 11.1 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 the classification stage.

### 11.3 Support Vector Machine (SVM)

SVMs are classifiers defined by a separating hyper-plane between trained data in a N-dimensional space. The main advantage of using an 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 a small number of examples near decision boundary. Our goal is to find a set of negatives that are the closest to the separating hyper-plane. 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 taken from Microsoft's Azure github page in which there is an implementation of Fast RCNN using an SVM classifier. We didn't modify its parameters which means that it has a linear kernel, uses L2-norm as penalty and L1-norm as loss during training. Training procedure starts with randomly picking 100 videos in order to calculate the feature's scale. After that, each svm is provided with positive samples' feature maps and network looks for hard negatives for each class' svm. We consider as hard-negatives the tubes that got confidence score  $> -1.0$  during classification, and we add them to svm's samples. When we gather hard-negatives whose number is bigger than 500 or 1000 (depending on the approach) we retrain the class' SVM and remove samples with new score  $< -1.0$ .

This whole process makes the choice of the negatives a crucial factor. In order to find the best policy, we came with 5 different cases to consider as negatives:

1. Negatives are other classes' positives and all the background tubes
2. Negatives are only all the background videos
3. Negatives are only other classes' positives
4. Negatives are other classes' 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 the 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 11.2 contains the first classification results. In first column, we have the dimensions of feature maps before pooling function, where  $k = 1,2,\dots,5$ . In second column, we have feature maps' dimensions after pooling, and at the next 2 columns, the type of pooling function and the policy we followed. Finally, in the last 3 columns we have the mAP performance when we have threshold equal to 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	<b>22.07</b>	<b>24.4</b>	<b>25.77</b>
			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 11.2: 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.

### 11.3.1 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 environment surrounding the actor sometimes helps 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 experimented in policies 1 and 4 for both  $(256, 8, 7, 7)$  and  $(64, 8, 7, 7)$  feature maps as show in Table 11.3

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

According to Table 11.3, modified Roi Align doesn't improve mAP performance. On the contrary, it reduces it. However, the gap between those 2 approaches is small, so we don't abandon this idea, because, for different approaches, modified Roi Align may outclass regular Roi Align.

### 11.3.2 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, because a video lasting 15 frames is consisted of 2 ToIs and a video of 40 frames is consisted of 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 separate 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 with dimensions  $(256, 8, 7, 7)$ , so we concat them and finally get feature maps with size of  $(2, 256, 8, 7, 7)$ . In this case we didn't experiment with feature maps with size  $(64, 8, 7, 7)$  because they wouldn't perform better than feature maps with size  $(256, 8, 7, 7)$  as noticed from the previous section.

We experiment using an SVM classifier for training policies 1 and 4 and using both regular and modifier Roi Align. The performance results are presented at Table 11.4.

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	24.97	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 11.4: mAP results using temporal pooling for both RoiAlign approaches

Comparing Tables 11.3 and 11.4, 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.

## 11.4 Increasing sample duration to 16 frames

Next, we thought 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 in table 11.5.

Dimensions		Temporal Pooling	Type	mAP precision		
before	after			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 11.5: mAP results for policies 1,4 for sample duration = 16

As shown in Table 11.5, we get better performance when we don't use temporal pooling, a fact that is unexpected. However, the difference between those performances 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. Furthermore, comparing above results with results shown at Table 11.3, we can see that we get about the same results for both approaches. So, we choose to keep using the approach with sample duration equal with 8. That's because, we don't have to use too much memory during training and validation.

## 11.5 Adding more groundtruth tubes

The previous results came from when we train classifiers using only 1 groundtruth action tube and 2 background. We thought that we should experiment with the number of foreground action tubes and the ratio between foreground and background tubes because in previous approaches

these numbers were a little arbitrary. So, we choose to train our previous classifiers using 2, 4 and 8 foreground tubes and a ratio of 2:3, 1:2, 1:3, and 1:4 between the number of foreground tubes and the total number of both foreground and background tubes.

Firstly, we train the RNN classifier using feature maps with dimensions  $(256, 8, 7, 7)$  and mAP performance is presented in Table 11.6 for overlap threshold equal to 0.5, 0.4 and 0.3 .

<b>F. map</b>	<b>FG tubes</b>	<b>Total tubes</b>	<b>mAP</b>		
			0.5	0.4	0.3
(k,256,8,7,7)	2	3	11.3	14.14	14.84
		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 11.6: RNN results

According to Table 11.6, firstly we can see that increasing the number of foreground tubes from 1 to 2 leads to reduce rapidly mAP performance. But, when we set foreground tubes equal to 4 we get better results. On top of that, we get the best performance when the ratio is equal to 1:2 and 1:4. Finally, when we set the number of foreground tubes equal to 8, performance gets slightly better comparing with the initial conditions (1 foreground action tube and 3 in total), but this situation doesn't get us the best results.

Next, it's time to experiment using the Linear classifier. We use again the same cases like we did for RNN classification. As mentioned before, we need a pooling method before the classification step. According to Table 11.1, avg pooling method results in better mAP performance than max pooling, so we use avg pooling for all following cases. Results are included in Table 11.7.

<b>F. map</b>	<b>FG tubes</b>	<b>Total tubes</b>	<b>mAP</b>		
			0.5	0.4	0.3
(k,256,8,7,7)	2	3	14.18	19.81	20.02
		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
		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
	8				

Table 11.7: Linear results

First of all, after considering results presented at both Tables 11.6 and 11.7, it becomes clear that when we set the number of foreground tubes equal to 2, for both cases, we get worse results than the initial. This probably is due to the fact that we increase also the number of background tubes in cases when the ratio is 1:2, 1:3 and 1:4 resulting in considering more proposed tubes as background tubes. On the other hand, when we set ratio equal to 2:3, instead of considering most proposed action tubes as background, classifiers classify them as a specific action class, which means there is an overfitting situation. So, although we think that we shouldn't investigate any more for cases with 2 foreground action tubes, we will train our SVM classifier using 2 foreground tubes and all the aforementioned ratio because we want to be sure about our assumption. On the other hand, we notice that using 4 or 8 foreground tubes both get us better results than the initial results. The best results come when ratio between foreground and total tubes is 1:3 for both cases. Furthermore, we get good results for ratios 2:3 and 1:2, and we get the worst while using ratio 1:4. This is caused probably from the big number of background tubes comparing with the one of foreground tubes.

As mentioned before, we train SVM classifier using aforementioned cases only for policy 1 because it gives us the best results for all previous cases. Classification performance using mAP metric is shown in Table 11.8.

F. map	FG tubes	Total tubes	mAP		
			0.5	0.4	0.3
(2,256,8,7,7)	2	1	3	24.97	26.91
		3	13.87	18.74	21.29
		4	14.21	19.67	21.75
		6	12.88	18.62	21.59
		8	12.66	18.7	21.97
	4	6	25.04	26.91	27.82
		8	24.34	25.67	26.34
		12	23.47	25.31	25.9
		16	21.94	23.55	24.23
	8	12	24.83	27.13	27.46
		16	23.97	26.38	26.94
		24	24.17	26.24	26.76
		32	24.17	26.24	26.76

Table 11.8: SVM results

Results shows us some interesting facts. Firstly, it confirms our assumption that our network

is unable to train well with only 2 foreground tubes, so from now on, we will investigate training situations using 4 or 8 foreground action tube during training. Also, a strange fact occurs, which is that we get almost the same results with the results obtained when using policy 1, only one foreground tube, 3 in total and temporal pooling. This is probably because during calculation of feature scale, during training stage, we don't get such good sample set of video like we did during aforementioned situation. But we think that it is better to keep testing using 4 or 8 foreground action tubes. Last but not least, it is clear that we get the best result when we have ratio 2:3 between the number of foreground and total action tubes. Also, it is more preferable to have 4 foreground action tubes instead of 8. This means that given too many action tubes confuse SVM classifier, so it fails to perform well.

### 11.5.1 Increasing again sample duration (only for RNN and Linear)

Table 11.5 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 to 16 frames. Table 11.9 shows RNN's mAP performance and Table 11.10 Linear's mAP performance. We started experimenting with RNN classifier because it performed better than Linear classifier previously. As mentioned before, we experiment using 4 or 8 foreground tubes and ratios 2:3, 1:2, 1:3 and 1:4 between the number of foreground and total action tubes provided.

<b>FG tubes</b>	<b>Total tubes</b>	<b>mAP</b>		
		0.5	0.4	0.3
4	6	7.94	13.51	14.95
	8	10.88	14.02	14.74
	12	14.05	19.23	20.99
	16	11.69	15.77	16.87
8	12	10.47	15.06	19.93
	16	12.29	19.51	23.11
	24	12.85	18.35	20.00
	32	9.38	14.33	16

Table 11.9: RNN results for sample duration equal to 16

Comparing results from Table 11.9 and previous table 11.6 one by one, it is clear that RNN outperforms when sample duration equals with 8 frames. This result was expected, because, the increase of sample duration reduces the number of video segments needed for each video. So, this means that RNN has to classify sequences with 3 video segments at the most, which is more difficult than classifying bigger sequence like previously.

Next, we experiment using Linear classifier for the same cases like RNN's.

<b>FG tubes</b>	<b>Total tubes</b>	<b>mAP</b>		
		0.5	0.4	0.3
4	6	13.79	19.75	23.63
	8	15.11	19.78	21.14
	12	11.39	15.74	18.15
	16	13.62	16.11	18.15

8	12	10.63	19.37	21.65
	16	12.98	17.52	19.10
	24	12.92	17.64	19.95
	32	11.51	13.98	14.82

Table 11.10: Linear results for sample duration equal to 16

In this case, results from table 11.10 are worse than those presented in Table 11.7, with the difference being about 2%. So, after considering both cases, it becomes clear that it is preferable to experiment using sample duration equal to 8 and not to increase it to 16 frames.

## 11.6 MultiLayer Perceptron (MLP)

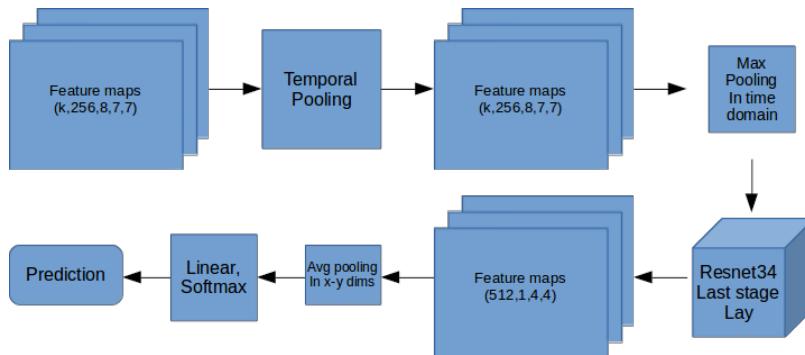


Figure 11.3: Structure of the MLP classifier

In previous sections we used classic classifiers like Linear, RNN and SVM. Last but not least, another widely category of classifiers is Multilayer Perceptron (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 11.3 for sample duration equal to 8, and is described below:

- 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 to  $(2, 256, 7, 7)$ , which we reshape it into  $(256, 2, 7, 7)$  which we feed to layers extracted from the last stage of ResNet34. This stage includes 3 Residual Layers with stride equal to 2 in all 3 dimensions and output number of filters equal to 512.
- After Residual Layers, we perform avg pooling for x-y dimensions. So we get as output, activation maps with dimension size equal to  $(512,)$ . Finally, we feed these feature maps to a linear layer in order to get class confidence score, after applying soft-max function.

### 11.6.1 Regular training

According to figure 11.1, the trainable parts of our network are the 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 frozen 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 in Table 11.11.

<b>FG tubes</b>	<b>Total tubes</b>	<b>mAP</b>		
		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 11.11: MLP's mAP 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 to the conclusion that we need at the most 8 foreground tubes, even though the ratio 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.

### 11.6.2 Extract features

As previously performed, we trained MLP classifier using pre-computed feature maps. These feature maps include both foreground and background action tubes. Based on the conclusions made in previous sections, we will train our classifier only for number of foreground tubes equal to 4 and 8. Furthermore, we will train it for 3 different ratios between the number of foreground and background action tubes, which are 1:1, 1:2 and 1:3. Table 11.12 shows these cases and their respective mAP performance during the validation step.

<b>FG tubes</b>	<b>Total tubes</b>	<b>mAP</b>		
		0.5	0.4	0.3
4	6	4,37	8,54	10,12
	8	5.89	9.54	13.61
	12	9.51	12.8	14.6
	16	6.80	13.17	14.67

8	12	8.62	12.32	14.74
	16	8.49	13.94	15.09
	24	6.72	12.17	15.30
	32	13.27	17.64	18.97

Table 11.12: mAP results for MLP trained using extracted features

Comparing results from Tables 11.12 and 11.11, it is clear we need 8 foreground tubes in order MLP classifier to perform well. However, it isn't very clear which of these two proposed training approaches is better, but if we have to decide one method, we would choose to use pre-extracted features training. This approach manages to achieve the best results, and especially when we have 8 foreground action tubes and 32 in total. Also, comparing methods using 4 or 8 positive action tubes, it is clear that we would prefer using 8 generally. However, it's not clear which ratio is better because, we get the best results when we have 8 foreground action tubes and ratio 1:4 while we get the best results when ratio is 1:3 having 4 positive action tubes.

## 11.7 Adding nms algorithm

After getting previous classification results, we came with the idea that a lot of proposed action tubes overlap spatiotemporally like presented in chapter 4, for the first linking algorithm. On top of that, even though, at last, we will keep only the best scoring action tube, maybe, our Network sometimes doesn't score the best overlapping action tube but a neighbor, which should have been removed. So, similar with chapter 4, we added NMS algorithm before classification stage in order to remove unnecessary overlapping action tubes. The structure of this approach is presented at Figure 11.4. So, we run again validation stage for our classifiers and results are presented in Tables 11.13, 11.14, 11.15 and 11.16 for SVM, RNN, Linear and MLP classifiers respectively.

We start experimenting using SVM classifier, whose mAP performance is presented in Table 11.13 when we use NMS algorithm.

FG tubes	Total tubes	mAP		
		0.5	0.4	0.3
4	6	21.42	24.3	25.11
	8	21.3	24.06	24.85
	12	21.73	24.19	25.04
	16	21.11	23.98	24.84
8	12	20.47	22.55	23.41
	16	21.21	23.89	24.97
	24	21.71	23.8	24.82
	32	21.43	23.5	24.52

Table 11.13: mAP results for SVM classifier after adding NMS algorithm

According to Table 11.13 and taking Table's 11.8 results into consideration, it is clear that

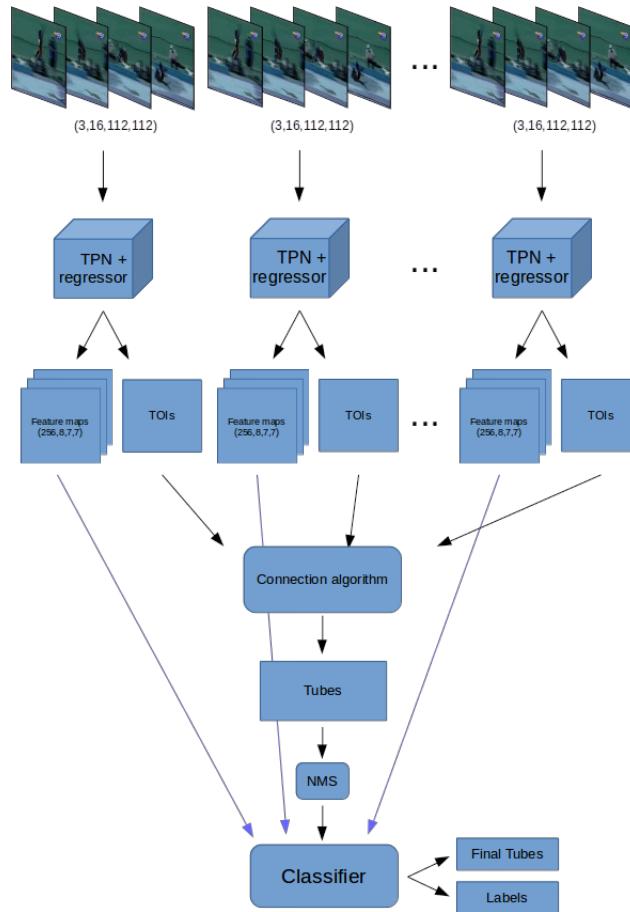


Figure 11.4: Structure of the network with NMS

mAP performance decreases while using NMS algorithm. We experiment only with the cases having foreground tubes equal to 4 or 8 and we didn't experiment using initial ratio, because we think that we are going to get the same attitude. This is probably because NMS algorithm removes some overlapping action tubes, so SVM classifier doesn't get the right action tubes for classifying them.

Next, we experiment using RNN classifier for sample duration equal to 8 frames per video segment. Results are included in Table 11.14

FG tubes	Total tubes	mAP		
		0.5	0.4	0.3
4	6	13.27	26.12	30.69
	8	16.06	25.81	27.63
	12	13.93	22.48	23.99
	16	17.24	26.44	28.36
8	12	2	5.11	7.75
	16	11.11	20.98	23.97
	24	13.84	22.77	24.31

	32	12.74	21.49	25.39
--	----	-------	-------	-------

Table 11.14: mAP results for RNN classifier after adding NMS algorithm

Alongside with SVM's mAP performances, RNN doesn't achieve any improvement in its mAP performance when adding NMS algorithm. On the contrary, it reduce about 5% in some cases, so it is more preferable to use ActionNet's architecture without NMS algorithm included while using RNN as a classifier.

Following RNN, we investigate the same situations with Linear classify instead of the RNN. Its performance is presented in Table 11.15.

<b>FG tubes</b>	<b>Total tubes</b>	<b>mAP</b>		
		0.5	0.4	0.3
4	6	12.59	14.91	17.79
	8	15.89	21.92	23.28
	12	13.23	19.72	23.17
	16	15.07	17.38	18.27
8	12	18.88	24.37	26.72
	16	15.42	22.31	24.70
	24	15.60	19.71	21.08
	32	16.1	19.99	21.47

Table 11.15: mAP results for Linear classifier after adding NMS algorithm

On the contrary with SVM and RNN's performances, Linear's mAP performance is getting better when adding NMS algorithm, comparing results presented at Tables 11.15 and 11.7. This is probably because Linear classifier gets, now, less proposed action tubes comparing with the previous approach. So, now it is less likely to misclassify an action tube, probably considering it as foreground when actually is a background action tube. Even though its performance increased, it doesn't achieve our best results, a fact which may corroborate our previous claim. Last but not least, we need to experiment using MLP classifier in order to determine if NMS algorithm is needed during classification. The results are presented in Table 11.16.

<b>FG tubes</b>	<b>Total tubes</b>	<b>mAP</b>		
		0.5	0.4	0.3
4	6	3.66	7.23	9.43
	8	3.43	8.17	12.77
	12	6.32	11.26	16.15
	16	4.82	11.38	15.85
8	12	5.92	12.42	15.81
	16	6	12.55	14.66
	24	4.73	11.33	15.25
	32	9.67	14.82	16.74

Table 11.16: mAP results for MLP classifier after adding NMS algorithm

Finally, comparing results from Tables 11.16 and 11.12, again adding NMS algorithm approach decreases mAP performance. Consequently, after considering all situations it is clear that NMS algorithm doesn't contribute at the improvement of Network's mAP performance, but, on the contrary, it reduces it. So, as a final comment, it is more preferable not to use it, except when we use Linear classifier for classification.

### 11.7.1 General comments

To sum up, in this section we tried to find the most suitable classifier for achieving good classification results based on mAP metric. It is clear that when using an SVM classifier, which uses temporal pooling in order to have a fix-size input, we get the best results. The most suitable method for training is using about 4 foreground action tubes and only 2 background which using both background action tubes and action tubes from other classes as hard-negatives.

# Chapter 12

## Conclusion - Future work

### 12.1 Conclusion

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

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

We tried to a design the Tube Proposal Network for proposing ToIs in given video segments, inspired by Faster R-CNN's RPN. We designed it using general anchors and not dataset specific anchors. This means that we try to generalize our approach for several datasets, on the contrary with the approach proposed by Girdhar et al. 2018, in which it uses the most frequently appearing anchors for each dataset.

On top of that, we designed a naive connection algorithm for connecting proposed ToIs based on the algorithm proposed by Girdhar et al. 2018. In our approach, we use the same scoring policy, which is a combination between actioness and overlapping scores. The main difference is that we avoid to calculate all the possible combinations using an updatable 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 potentials.

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

### 12.2 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 the 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. These 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  $\lambda$  factor in training loss would be a good idea and exploring which is the best approach. So training loss could be defined as:

$$L = \sum_i L_{cls}(p_i, p_i^*) + \lambda_1 \sum_i p_i^* L_{reg}(t_i, t_i^*) + \lambda_2 \sum_i q_i^* L_{reg}(c_i, c_i^*) \quad (12.1)$$

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(Lin et al. 2017), which could be extended 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 based on estimating temporal progress of an action tube and its overlap with others. Although it also didn't give us very good proposals, we believe that we should explore this algorithm's potential. 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 experimented mainly on an SVM classifier for JHMDB dataset and we didn't get involved with UCF dataset. Our first goal is to be able to get good classification results for UCF dataset, too. We think that we should explore UCF's feature maps and techniques applied at feature maps before classification. 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] O. Chomat and J. L. Crowley. “Probabilistic recognition of activity using local appearance”. In: *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*. Vol. 2. 1999, 104–109 Vol. 2. doi: [10.1109/CVPR.1999.784616](https://doi.org/10.1109/CVPR.1999.784616).
- [2] A. F. Bobick and J. W. Davis. “The recognition of human movement using temporal templates”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23.3 (2001), pp. 257–267. doi: [10.1109/34.910878](https://doi.org/10.1109/34.910878).
- [3] L. Zelnik-Manor and M. Irani. “Event-based analysis of video”. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Vol. 2. 2001, pp. II–II. doi: [10.1109/CVPR.2001.990935](https://doi.org/10.1109/CVPR.2001.990935).
- [4] Laptev and Lindeberg. “Space-time interest points”. In: *Proceedings Ninth IEEE International Conference on Computer Vision*. 2003, 432–439 vol.1. doi: [10.1109/ICCV.2003.1238378](https://doi.org/10.1109/ICCV.2003.1238378).
- [5] Alper Yilmaz and Mubarak Shah. “Actions sketch: a novel action representation”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. Vol. 1. 2005, 984–989 vol. 1. doi: [10.1109/CVPR.2005.58](https://doi.org/10.1109/CVPR.2005.58).
- [6] M. Blank et al. “Actions as space-time shapes”. In: *Tenth IEEE International Conference on Computer Vision (ICCV’05) Volume 1*. Vol. 2. 2005, 1395–1402 Vol. 2. doi: [10.1109/ICCV.2005.28](https://doi.org/10.1109/ICCV.2005.28).
- [7] P. Dollar et al. “Behavior recognition via sparse spatio-temporal features”. In: *2005 IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*. 2005, pp. 65–72. doi: [10.1109/VSPETS.2005.1570899](https://doi.org/10.1109/VSPETS.2005.1570899).
- [8] E. Shechtman and M. Irani. “Space-time behavior based correlation”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. Vol. 1. 2005, 405–412 vol. 1. doi: [10.1109/CVPR.2005.328](https://doi.org/10.1109/CVPR.2005.328).
- [9] Y. Sheikh, M. Sheikh, and M. Shah. “Exploring the space of a human action”. In: *Tenth IEEE International Conference on Computer Vision (ICCV’05) Volume 1*. Vol. 1. 2005, 144–149 Vol. 1. doi: [10.1109/ICCV.2005.90](https://doi.org/10.1109/ICCV.2005.90).
- [10] A. Yilmaz and M. Shah. “Recognizing human actions in videos acquired by uncalibrated moving cameras”. In: *Tenth IEEE International Conference on Computer Vision (ICCV’05) Volume 1*. Vol. 1. 2005, 150–157 Vol. 1. doi: [10.1109/ICCV.2005.201](https://doi.org/10.1109/ICCV.2005.201).
- [11] Juan Carlos Niebles, Hongcheng Wang, and Fei Fei Li. “Unsupervised Learning of Human Action Categories Using Spatial-Temporal Words.” In: vol. 79. Sept. 2006, pp. 1249–1258.

- [12] M. S. Ryoo and J. K. Aggarwal. “Semantic Understanding of Continued and Recursive Human Activities”. In: *18th International Conference on Pattern Recognition (ICPR'06)*. Vol. 1. 2006, pp. 379–378. DOI: 10.1109/ICPR.2006.1043.
- [13] Y. Ke, R. Sukthankar, and M. Hebert. “Spatio-temporal Shape and Flow Correlation for Action Recognition”. In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. 2007, pp. 1–8. DOI: 10.1109/CVPR.2007.383512.
- [14] M. D. Rodriguez, J. Ahmed, and M. Shah. “Action MACH a spatio-temporal Maximum Average Correlation Height filter for action recognition”. In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 2008, pp. 1–8. DOI: 10.1109/CVPR.2008.4587727.
- [15] Mark Everingham et al. “The Pascal Visual Object Classes (VOC) Challenge”. In: *Int. J. Comput. Vision* 88 (2010). URL: <http://dx.doi.org/10.1007/s11263-009-0275-4>.
- [16] P. F. Felzenszwalb et al. “Object Detection with Discriminatively Trained Part-Based Models”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.9 (2010), pp. 1627–1645. DOI: 10.1109/TPAMI.2009.167.
- [17] 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>.
- [18] H. Kuehne et al. “HMDB: A large video database for human motion recognition”. In: *2011 International Conference on Computer Vision*. 2011, pp. 2556–2563. DOI: 10.1109/ICCV.2011.6126543.
- [19] 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).
- [20] H. Jhuang et al. “Towards Understanding Action Recognition”. In: *2013 IEEE International Conference on Computer Vision*. 2013, pp. 3192–3199. DOI: 10.1109/ICCV.2013.396.
- [21] S. Ji et al. “3D Convolutional Neural Networks for Human Action Recognition”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.1 (2013), pp. 221–231. DOI: 10.1109/TPAMI.2012.59.
- [22] S. Manen, M. Guillaumin, and L. V. Gool. “Prime Object Proposals with Randomized Prim’s Algorithm”. In: *2013 IEEE International Conference on Computer Vision*. 2013, pp. 2536–2543. DOI: 10.1109/ICCV.2013.315.
- [23] Y. Tian, R. Sukthankar, and M. Shah. “Spatiotemporal Deformable Part Models for Action Detection”. In: *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 2642–2649. DOI: 10.1109/CVPR.2013.341.
- [24] J.R.R. Uijlings et al. “Selective Search for Object Recognition”. In: *International Journal of Computer Vision* (2013). DOI: 10.1007/s11263-013-0620-5. URL: <http://www.huppelen.nl/publications/selectiveSearchDraft.pdf>.
- [25] R. Girshick et al. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 580–587. DOI: 10.1109/CVPR.2014.81.
- [26] M. Jain et al. “Action Localization with Tubelets from Motion”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 740–747. DOI: 10.1109/CVPR.2014.100.

- [27] 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.
- [28] Dan Ooneata et al. “Spatio-Temporal Object Detection Proposals”. In: Sept. 2014. DOI: 10.1007/978-3-319-10578-9\_48.
- [29] 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.
- [30] W. Chen and J. J. Corso. “Action Detection by Implicit Intentional Motion Clustering”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 3298–3306. DOI: 10.1109/ICCV.2015.377.
- [31] Jan C. van Gemert et al. “APT: Action localization proposals from dense trajectories”. In: *Proceedings of the British Machine Vision Conference (BMVC)*. BMVA Press, 2015, pp. 177.1–177.12. ISBN: 1-901725-53-7. DOI: 10.5244/C.29.177. URL: <https://dx.doi.org/10.5244/C.29.177>.
- [32] R. Girshick. “Fast R-CNN”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169.
- [33] G. Gkioxari and J. Malik. “Finding action tubes”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 759–768. DOI: 10.1109/CVPR.2015.7298676.
- [34] Joe Yue-Hei Ng et al. “Beyond short snippets: Deep networks for video classification”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 4694–4702. DOI: 10.1109/CVPR.2015.7299101.
- [35] 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>.
- [36] K. Soomro, H. Idrees, and M. Shah. “Action Localization in Videos through Context Walk”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 3280–3288. DOI: 10.1109/ICCV.2015.375.
- [37] D. Tran et al. “Learning Spatiotemporal Features with 3D Convolutional Networks”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 4489–4497. DOI: 10.1109/ICCV.2015.510.
- [38] P. Weinzaepfel, Z. Harchaoui, and C. Schmid. “Learning to Track for Spatio-Temporal Action Localization”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 3164–3172. DOI: 10.1109/ICCV.2015.362.
- [39] G. Yu and J. Yuan. “Fast action proposals for human action detection and search”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1302–1311. DOI: 10.1109/CVPR.2015.7298735.
- [40] R. G. Cinbis, J. Verbeek, and C. Schmid. “Weakly Supervised Object Localization with Multi-Fold Multiple Instance Learning”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.1 (2016), pp. 189–203. DOI: 10.1109/TPAMI.2016.2535231.
- [41] C. Feichtenhofer, A. Pinz, and A. Zisserman. “Convolutional Two-Stream Network Fusion for Video Action Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 1933–1941. DOI: 10.1109/CVPR.2016.213.

- [42] K. He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.
- [43] Pascal Mettes, Jan C. van Gemert, and Cees G. M. Snoek. “Spot On: Action Localization from Pointly-Supervised Proposals”. In: *CoRR* abs/1604.07602 (2016). arXiv: 1604.07602. URL: <http://arxiv.org/abs/1604.07602>.
- [44] 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>.
- [45] J. Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.
- [46] 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>.
- [47] 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>.
- [48] Philippe Weinzaepfel, Xavier Martin, and Cordelia Schmid. “Towards Weakly-Supervised Action Localization”. In: *CoRR* abs/1605.05197 (2016). arXiv: 1605.05197. URL: <http://arxiv.org/abs/1605.05197>.
- [49] 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>.
- [50] B. Zhang et al. “Real-Time Action Recognition with Enhanced Motion Vector CNNs”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 2718–2726. DOI: 10.1109/CVPR.2016.297.
- [51] Bowen Zhang et al. “Real-time Action Recognition with Enhanced Motion Vector CNNs”. In: *CoRR* abs/1604.07669 (2016). arXiv: 1604.07669. URL: <http://arxiv.org/abs/1604.07669>.
- [52] Harkirat S. Behl et al. “Incremental Tube Construction for Human Action Detection”. In: *CoRR* abs/1704.01358 (2017). arXiv: 1704.01358. URL: <http://arxiv.org/abs/1704.01358>.
- [53] N. Bodla et al. “Soft-NMS — Improving Object Detection with One Line of Code”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 5562–5570. DOI: 10.1109/ICCV.2017.593.
- [54] J. Carreira and A. Zisserman. “Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 4724–4733. DOI: 10.1109/CVPR.2017.502.
- [55] 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>.

- [56] J. Donahue et al. “Long-Term Recurrent Convolutional Networks for Visual Recognition and Description”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.4 (2017), pp. 677–691. DOI: 10.1109/TPAMI.2016.2599174.
- [57] Rohit Girdhar and Deva Ramanan. “Attentional Pooling for Action Recognition”. In: *CoRR* abs/1711.01467 (2017). arXiv: 1711.01467. URL: <http://arxiv.org/abs/1711.01467>.
- [58] K. Hara, H. Kataoka, and Y. Satoh. “Learning Spatio-Temporal Features with 3D Residual Networks for Action Recognition”. In: *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*. 2017, pp. 3154–3160. DOI: 10.1109/ICCVW.2017.373.
- [59] K. He et al. “Mask R-CNN”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2980–2988. DOI: 10.1109/ICCV.2017.322.
- [60] R. Hou, C. Chen, and M. Shah. “Tube Convolutional Neural Network (T-CNN) for Action Detection in Videos”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 5823–5832. DOI: 10.1109/ICCV.2017.620.
- [61] G. Huang et al. “Densely Connected Convolutional Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2261–2269. DOI: 10.1109/CVPR.2017.243.
- [62] V. Kalogeiton et al. “Action Tubelet Detector for Spatio-Temporal Action Localization”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 4415–4423. DOI: 10.1109/ICCV.2017.472.
- [63] 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>.
- [64] T. Lin et al. “Feature Pyramid Networks for Object Detection”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 936–944. DOI: 10.1109/CVPR.2017.106.
- [65] 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>.
- [66] P. Mettes and C. G. M. Snoek. “Spatial-Aware Object Embeddings for Zero-Shot Localization and Classification of Actions”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 4453–4462. DOI: 10.1109/ICCV.2017.476.
- [67] S. Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (2017), pp. 1137–1149. DOI: 10.1109/TPAMI.2016.2577031.
- [68] S. Saha, G. Singh, and F. Cuzzolin. “AMTnet: Action-Micro-Tube Regression by End-to-end Trainable Deep Architecture”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 4424–4433. DOI: 10.1109/ICCV.2017.473.
- [69] G. Singh et al. “Online Real-Time Multiple Spatiotemporal Action Localisation and Prediction”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 3657–3666. DOI: 10.1109/ICCV.2017.393.
- [70] K. Soomro and M. Shah. “Unsupervised Action Discovery and Localization in Videos”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 696–705. DOI: 10.1109/ICCV.2017.82.
- [71] Jianwei Yang et al. “A Faster Pytorch Implementation of Faster R-CNN”. In: <https://github.com/jwyang/faster-rcnn.pytorch> (2017).

- [72] H. Zhu, R. Vial, and S. Lu. “TORNADO: A Spatio-Temporal Convolutional Regression Network for Video Action Proposal”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 5814–5822. doi: 10.1109/ICCV.2017.619.
- [73] Yi Zhu et al. “Hidden Two-Stream Convolutional Networks for Action Recognition”. In: *CoRR* abs/1704.00389 (2017). arXiv: 1704.00389. URL: <http://arxiv.org/abs/1704.00389>.
- [74] M. Zolfaghari et al. “Chained Multi-stream Networks Exploiting Pose, Motion, and Appearance for Action Classification and Detection”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2923–2932. doi: 10.1109/ICCV.2017.316.
- [75] Ali Diba et al. “Temporal 3D ConvNets Using Temporal Transition Layer”. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2018, Salt Lake City, UT, USA, June 18–22, 2018*. 2018, pp. 1117–1121. URL: [http://openaccess.thecvf.com/content\\_cvpr\\_2018\\_workshops/w19/html/Diba\\_Temporal\\_3D\\_ConvNets\\_CVPR\\_2018\\_paper.html](http://openaccess.thecvf.com/content_cvpr_2018_workshops/w19/html/Diba_Temporal_3D_ConvNets_CVPR_2018_paper.html).
- [76] R. Girdhar et al. “Detect-and-Track: Efficient Pose Estimation in Videos”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 350–359. doi: 10.1109/CVPR.2018.00044.
- [77] Rohit Girdhar et al. “A Better Baseline for AVA”. In: *CoRR* abs/1807.10066 (2018). arXiv: 1807.10066. URL: <http://arxiv.org/abs/1807.10066>.
- [78] C. Gu et al. “AVA: A Video Dataset of Spatio-Temporally Localized Atomic Visual Actions”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 6047–6056. doi: 10.1109/CVPR.2018.00633.
- [79] Michelle Guo et al. “Neural Graph Matching Networks for Fewshot 3D Action Recognition”. In: *The European Conference on Computer Vision (ECCV)*. 2018.
- [80] K. Hara, H. Kataoka, and Y. Satoh. “Can Spatiotemporal 3D CNNs Retrace the History of 2D CNNs and ImageNet?” In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 6546–6555. doi: 10.1109/CVPR.2018.00685.
- [81] J. He et al. “Generic Tubelet Proposals for Action Localization”. In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2018, pp. 343–351. doi: 10.1109/WACV.2018.00044.
- [82] 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>.
- [83] D. C. Luvizon, D. Picard, and H. Tabia. “2D/3D Pose Estimation and Action Recognition Using Multitask Deep Learning”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 5137–5146. doi: 10.1109/CVPR.2018.00539.
- [84] D. Tran et al. “A Closer Look at Spatiotemporal Convolutions for Action Recognition”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 6450–6459. doi: 10.1109/CVPR.2018.00675.
- [85] 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>.