

POLITECNICO DI TORINO

The Department of CONTROL AND COMPUTER ENGINEERING (*DAUIN*)

**Master of Science
in Computer Engineering**

Master's Degree Thesis

**3D convolutional neural networks for
Online Detection of Action Start**



Supervisors

Prof. Fabrizio Lamberti

Dr. Lia Morra

Candidate

Lei Qing

April 2019

Abstract

Online action detection refers to detect an action as it happens and ideally even before the action is fully completed. In this thesis, I focus on online action detection in untrimmed videos. This is important because videos in many applications such as surveillance monitoring systems need to detect anomaly actions as soon as possible and then issue an alert to allow timely security response. In this project, I implemented a state-of-the-art article. I use C3D as the backbone network and use three novel methods to train the network: (1) use adaptive sampling to handle the scarcity training samples problem; (2) model the temporal consistency to make the feature of action start window close to the actual action; (3) fine-tuning the model via generative adversarial network. I conduct experiments using THUMOS'14 dataset and a sub-set of UCF-Crime. The results proved that the training methods can improve the performance of the networks, but still have a large room for improvement.

Acknowledgements

I would like to express my great thanks to my supervisor Prof. Fabrizio Lamberti and Dr. Lia Morra, for their guidance and supports.

Table of Contents

<i>Abstract</i>	<i>I</i>
<i>Acknowledgements</i>	<i>II</i>
<i>Table of Contents</i>	<i>III</i>
<i>List of Figures</i>	<i>VI</i>
<i>List of tables</i>	<i>VIII</i>
<i>1 Introduction</i>	<i>1</i>
<i>2 Background</i>	<i>3</i>
<i>2.1 Neural Network</i>	<i>3</i>
<i>2.2 Convolutional neural network</i>	<i>4</i>
<i>2.3 3D convolutional neural network</i>	<i>7</i>
<i>2.4 Generative adversarial network</i>	<i>8</i>
<i>2.5 Transfer learning</i>	<i>10</i>
<i>2.6 Evaluation metrics</i>	<i>12</i>
<i>3 Related works</i>	<i>15</i>
<i>3.1 Action classification</i>	<i>15</i>
<i>3.2 Temporal action localization</i>	<i>16</i>
<i>3.3 Early Recognition and Detection</i>	<i>17</i>
<i>3.4 Online Action Detection</i>	<i>17</i>
<i>3.5 Datasets</i>	<i>18</i>
<i>4 Design and implementation</i>	<i>21</i>
<i>4.1 Backbone network</i>	<i>22</i>
<i>4.1.1 C3D network</i>	<i>22</i>

4.1.2	Implementation of C3D network.....	22
4.2	Training methodology and additional models	25
4.2.1	Adaptive sampling	25
4.2.2	Temporal consistency	26
4.2.3	Generate hard negative data via GAN	28
4.3	Training steps.....	33
4.4	Testing procedure	33
4.5	Evaluation protocol.....	34
5	<i>Experiments on THUMOS'14</i>.....	37
5.1	THUMOS'14 dataset.....	37
5.2	Construct training batches.....	37
5.3	Training phase.....	41
5.3.1	Training C3D model	41
5.3.2	Training TC model.....	47
5.3.3	Training GAN model.....	49
5.4	Results	53
5.4.1	Results of C3D model.....	53
5.4.2	Results of TC model	54
5.4.3	Results of the model when a GAN is used for data augmentation	54
5.4.4	Comparison of results	56
5.5	Examples of prediction	58
6	<i>Experiments on a subset of UCF-Crime</i>.....	61
6.1	UCF-crime dataset.....	61
6.2	Training phase.....	62
6.2.1	Training C3D model	63
6.2.2	Training TC model.....	64

6.2.3	Training GAN model.....	65
6.3	Results	66
6.3.1	Results of C3D model.....	66
6.3.2	Results of TC model	67
6.3.3	Result of GAN	67
6.3.4	Comparison of results.....	68
6.4	Examples of prediction	68
7	Conclusions	71
Appendix A	73
Appendix B	75
Bibliography	79

List of Figures

FIG. 2.1 A SIMPLE EXAMPLE OF NEURAL NETWORK. $ai(j)$ IS THE I-TH NODE OF J-TH LAYER.	3
FIG. 2.2 AN ARTIFICIAL NEURAL.....	4
FIG. 2.3 AN EXAMPLE OF CONVOLUTION [9].....	5
FIG. 2.4 AN EXAMPLE OF MAX POOLING AND AVERAGE POOLING	6
FIG. 2.5 AN EXAMPLE OF THE FULLY-CONNECTED LAYER	6
FIG. 2.6 AN EXAMPLE OF NEURAL NETWORK WITH MANY CONVOLUTIONAL LAYERS [9].....	7
FIG. 2.7 THE DIFFERENCE BETWEEN 2D CONVOLUTION AND 3D CONVOLUTION [10].....	8
FIG. 2.8 AN EXAMPLE OF GAN [14]	9
FIG. 2.9 A DIAGRAM THAT HELPS US TO DECIDE HOW TO TRAIN OUR NETWORKS.....	11
FIG. 2.10 PRECISION AND RECALL [15]	12
FIG. 3.1 MULTI-STAGE CNN [20]	16
FIG. 3.2 SAMPLE FRAMES OF THE ACTIONS FROM THUMOS'14	19
FIG. 4.1 THE START WINDOW AND ITS FOLLOW-UP WINDOW	21
FIG. 4.2 C3D ARCHITECTURE [7].....	22
FIG. 4.3 AN ILLUSTRATION OF DATA DISTRIBUTION [4]	26
FIG. 4.4 THE TEMPORAL CONSISTENCY MODEL	27
FIG. 4.5 AN EXAMPLE OF HARD NEGATIVE [4].....	28
FIG. 4.6 THE GAN MODEL [4].....	29
FIG. 4.7 TRAINING G. THE WEIGHTS OF FC1 AND FC2 WILL BE UPDATED.....	31
FIG. 4.8 TRAINING D. THE WEIGHT OF FC6, FC7 AND FC8 WILL BE UPDATED.....	32
FIG. 5.1 THE ARCHITECTURE OF TWO DIFFERENT C3D NETWORKS.....	41
FIG. 5.2 THE TRAINING RESULT OF 5 LAYERS C3D. THE ACC IS THE ACCURACY ON.....	42
FIG. 5.3 THE RESULTS OF C3D MODEL ON THE SUB-TRAIN-SET (3 ACTION CLASSES).....	44
FIG. 5.4 THE TRAINING RESULTS ON THE ENTIRE TRAIN-SET AND VAL-SET	47

FIG. 5.5 TRAINING RESULT OF THE FIRST EXPERIMENT, TEMPORAL CONSISTENCY STRATEGY IS USED FOR ALL WINDOWS.....	48
FIG. 5.6 TRAINING RESULT OF C3D + TC WHEN λ IS 0.001.....	49
FIG. 5.7 AN EXAMPLE OF THE PREDICTION. THE PICTURE ON THE TOP IS THE OUTPUT OF THE MODEL THAT TRAINED WITH 1.5K ITERATIONS. THE BOTTOM ONE IS THE OUTPUT OF THE MODEL WITH 5K ITERATIONS. THE GROUND TRUTH IS CLEANANDJERK.	52
FIG. 5.8 RESULTS OF C3D MODEL WHEN ADAPTIVE SAMPLING STRATEGY IS USED.....	53
FIG. 5.9 RESULTS OF THE MODEL WHEN ADAPTIVE SAMPLING AND TEMPORAL CONSISTENCY ARE USED...	54
FIG. 5.10 RESULTS OF THE MODEL WHEN ALL METHODS ARE USED.....	55
FIG. 5.11 RESULTS OF MODEL TRAINED WITH THE SAMCE PARAMETERS	55
FIG. 5.12 THUMOS'14 RESULTS	56
FIG. 5.13 THUMOS'14 RESULTS. THE DOTTED LINES ARE THE RESULTS IN THE ORIGINAL PAPER.....	56
FIG. 5.14 COMPARE THE RESULTS OF TWO EVALUATION PROTOCOLS.	57
FIG. 5.15 AN EXAMPLE OF PREDICTION	58
FIG. 5.16 AN EXAMPLE OF PREDICTION	59
FIG. 5.17 AN EXAMPLE OF PREDICTION	60
FIG. 6.1 EXAMPLES OF DIFFERENT ANOMALIES FROM UCF-CRIME DATASET [30].....	61
FIG. 6.2 ACCURACIES AND LOSSES.....	63
FIG. 6.3 ACCURACIES AND LOSSES.....	64
FIG. 6.4 ACCURACIES AND LOSSES.....	65
FIG. 6.5 RESULTS OF C3D MODEL WHEN ADAPTIVE SAMPLING STRATEGY IS USED.....	66
FIG. 6.6 THE RESULT OF C3D MODEL WHEN ADAPTIVE SAMPLING AND TEMPORAL CONSISTENCY STRATEGIES ARE USED.	67
FIG. 6.7 RESULTS OF THE MODEL WHEN ALL METHODS ARE USED.....	67
FIG. 6.8 COMPARE THE RESULTS	68
FIG. 6.9 EXAMPLES OF PREDICTION	69
FIG. 6.10 EXAMPLES OF PREDICTION	69

List of tables

TABLE 4.1 KERNEL SIZE AND OUTPUT SIZE OF EACH LAYER. K IS NUMBER OF ACTION CLASSES	24
TABLE 5.1 THE LIST OF ACTIONS AND COUNT OF VIDEOS OF VALIDATION DATA IN THUMOS'14	39
TABLE 5.2 STATISTICS OF THE TRAIN-SET AND VALIDATION-SET	40
TABLE 5.3 STATISTICS OF TRAIN-SET AND VALIDATION-SET	40
TABLE 5.4 DIFFERENT SETTINGS OF C3D MODEL.....	45
TABLE 5.5 EXPERIMENT SETTINGS	51
TABLE 6.1 STATISTICS OF TRAIN-SET AND TEST-SET	62
TABLE 6.2 SETTINGS OF THE GAN MODEL.....	66

1 Introduction

Over the last decade, with the proliferation of surveillance cameras and camera-equipped mobile devices, there has been an exponential growth in the number of videos being captured and archived. Detecting actions are of great importance for analysing these videos due to various applications. Action detection is an important basis of intelligent video analysis, video tagging, human-computer interaction and many other fields. It has been gained extensive attention in academic and engineering communities.

Action detection could be divided into different tasks. In this thesis, I focus on online action detection in untrimmed videos. Unlike traditional action detection [1, 2], the goal of online action detection is to detect an action as it happens and ideally even before the action is fully completed [3]. In the setting of offline action detection, the whole video is given during the testing phase, which means we can use information of the whole video to predict the boundary of the action and the action class. While, in online action detection, we are processing a data flow, we can only use the current frame and previous ones to detect (or predict) the actions.

Offline action detection systems could be used for video search. For example, YouTube may use this technic to generate the preview of the videos. Online action detection is more suitable for the scenarios that need real-time detection or early alert generation. For instance, the surveillance monitoring system needs to detect anomaly actions as soon as possible and then issue an alert to allow timely security response; a self-driving car needs to recognise the start of accidents happening around it as quickly as possible so that it can take some actions to avoid the collision.

In this thesis, I studied action detection technologies and implemented a state-of-art article [4]. Zheng et al. [4] proposed to detect the start of actions, also, they identified three challenges: (1) they defined action start window-a mixture of background and action

frames, such windows are rare in the training dataset; (2) since the action start window is a mixture, it's difficult to correctly detect it; (3) some background windows and action windows may share similar scenes, such background windows should be identified during training, but it's not feasible to identify them manually. To deal with these challenges, they designed three methods: (1) during training, they use adaptive sampling strategy to increase the percentage of action start windows; (2) they model the temporal consistency to increase the similarity between action windows and action start windows; (3) they use a GAN model to generate hard negatives. Following the state-of-the-art articles [5, 6, 7], I use C3D [7] as the backbone network to help illustrate technical ideas and implemented three methods as mentioned above.

Chapter two introduce background knowledge. This chapter consists of five parts. The first three parts introduce three networks: convolutional network, 3D convolutional network, and generative adversarial network. The fourth part introduces the transfer learning. The last part introduces some commonly used evaluation metrics.

Chapter three is a literature survey. This part reviews the works that related to action recognition, action detection, early detection and online action detection.

Chapter four elaborates the implementation of the networks and three training methods: adaptive sampling, model the temporal consistency and fine-tune the parameters via GAN.

Chapter five and six describes the experiments on THUMOS'14 dataset and a sub-set of UCF-Crime.

Finally, Chapter seven will conclude this thesis.

2 Background

2.1 Neural Network

A neural network is a computer system that uses a network of functions to understand and translate a data input of one form into a desired output, usually in another form. It was inspired by human biology neural network and the way neurons of the human brain function together to understand inputs from human senses. A neural network is a framework that combines deep learning algorithms together and process data inputs [8].

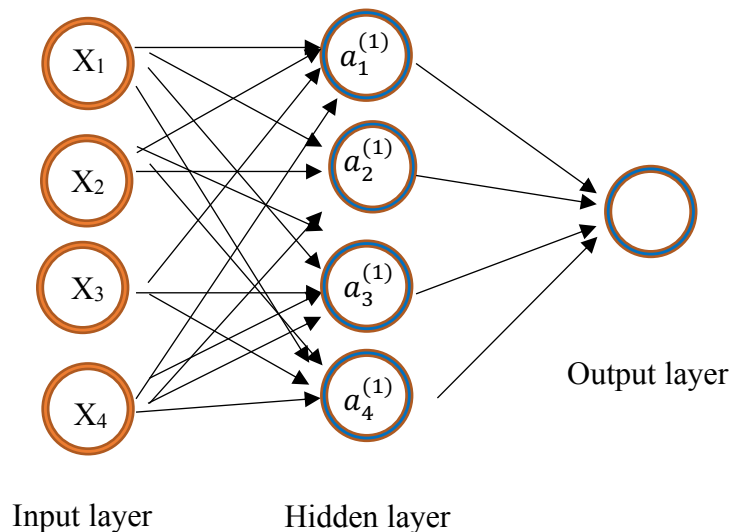


Fig. 2.1 A simple example of neural network. $a_i^{(j)}$ is the i -th node of j -th layer.

Fig. 2.1 shows a two layers neural network (when we count layers in neural networks, we don't count the input layer). The nodes in the network are also called neurons, which perform two operations: (1) $z = W x + b$, where x is the input data, W refers to the

weight and b is the bias; (2) $a = \sigma(z)$, where $\sigma(\bullet)$ refers sigmoid activation function. The activation function could be replaced by other functions, such as ReLU and tanh. A simple example is presented in Fig. 2.2.

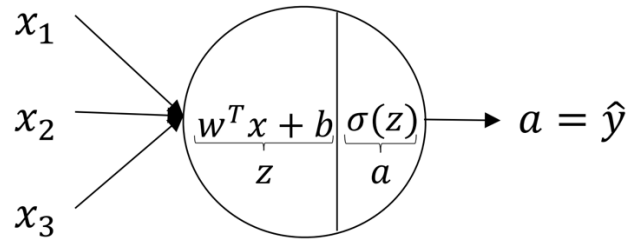


Fig. 2.2 An artificial neural

As exemplified in Fig. 2.2, when training, it will calculate $\hat{y} = \sigma(\sum_{i=1}^3 w_i y_i + b)$, where the sigmoid function is $\sigma(x) = \frac{1}{1+e^{-x}}$. This is step one, called forward propagation. After all layers finish its calculation, it will move to step two, backward propagation, which will update the weight W and bias b . Repeat these two steps until it converges.

2.2 Convolutional neural network

The neural network in Fig. 2.1 contains only one hidden layer. Thus, it's a shallow network. In deep learning, we are deal with complex problems; such a simple network may not powerful enough. So, we need a more complex network with more hidden layers. The network with multiple hidden layers called deep neural network. A convolutional neural network (CNN) is one of the deep neural networks that used to analyse visual imagery. Nowadays, CNNs have become the most commonly used algorithm for all computer vision tasks; more generally, they work on all perceptual tasks.

Commonly, a CNN contains the following layers:

- 1) **Convolutional Layer (Conv layer):** The Conv layer is the most important layer block of a CNN that perform convolution operations. It contains a set of filters that holds the learnable weights. During forward propagation, we slide the filter across all the areas of the input volume compute the element-wise multiplications and the sum of these multiplications. As the filter slide over the width and height of the input volume, we will get a 2D array, which is called a feature map or activation map. The convolution operation will reduce the dimension of the input, to address this issue, we could pad zeros to the input. An example of a convolution operation is shown in Fig. 2.3

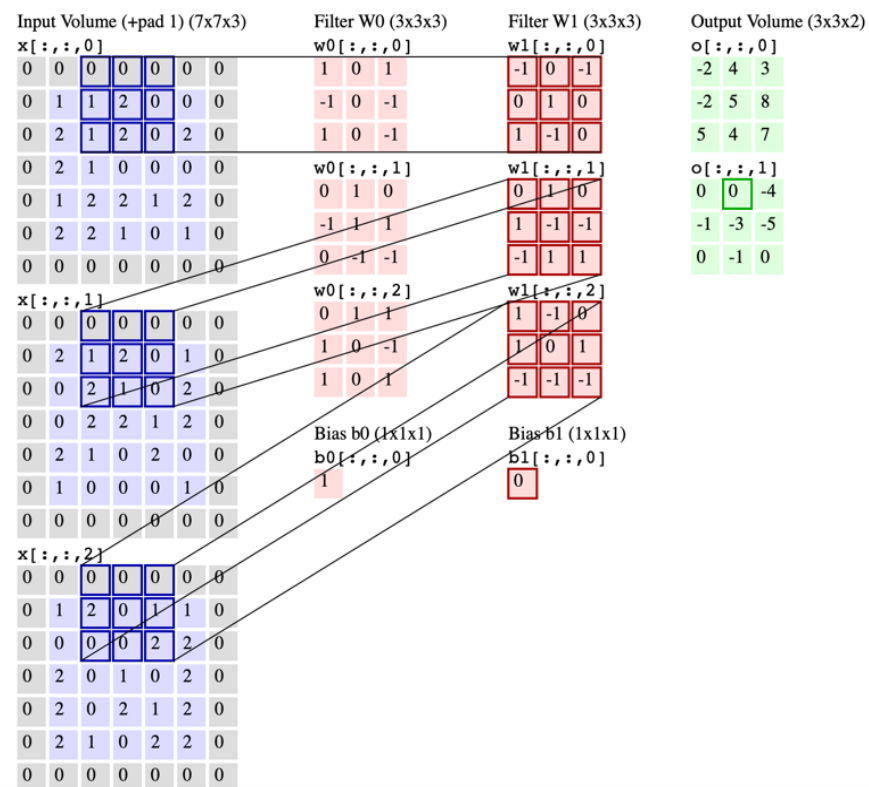


Fig. 2.3 an example of convolution [9]

- 2) **Pooling Layer:** Other than convolution layers, the function of pooling layer is to reduce the size of the representation, to speed the computation, and hence to also

control overfitting. [10] The most commonly used pooling layers are max-pooling layer and average-pooling layer.

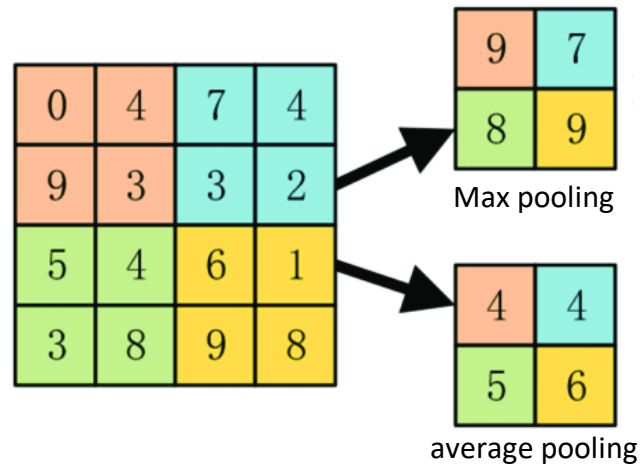


Fig. 2.4 An example of max pooling and average pooling

- 3) **Fully-Connected Layer:** Fully connected layers connect every node in one layer to every node in another layer. It is in principle the same as regular neural network. Generally, the fully-connected layers are placed at the end of the convolutional neural networks.

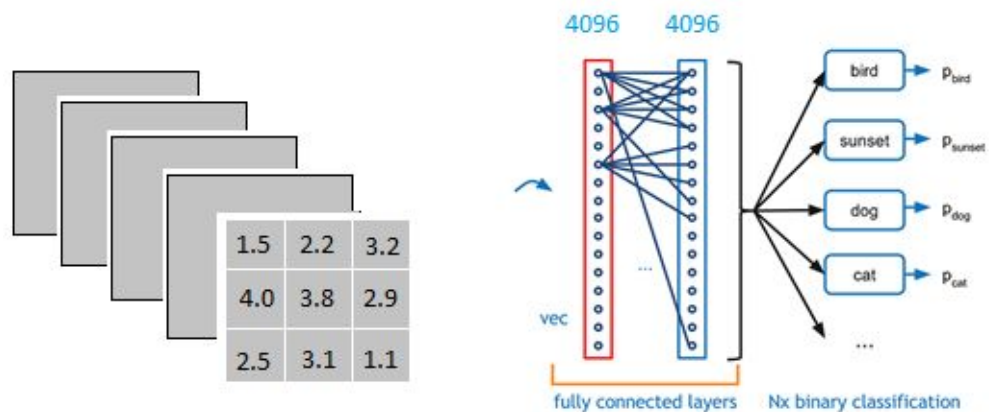


Fig. 2.5 An example of the fully-connected layer

As shown in Fig. 2.6, a CNN may also contain other layers, such normalization layers and activation function. We can compose a CNN by stacking these layers.

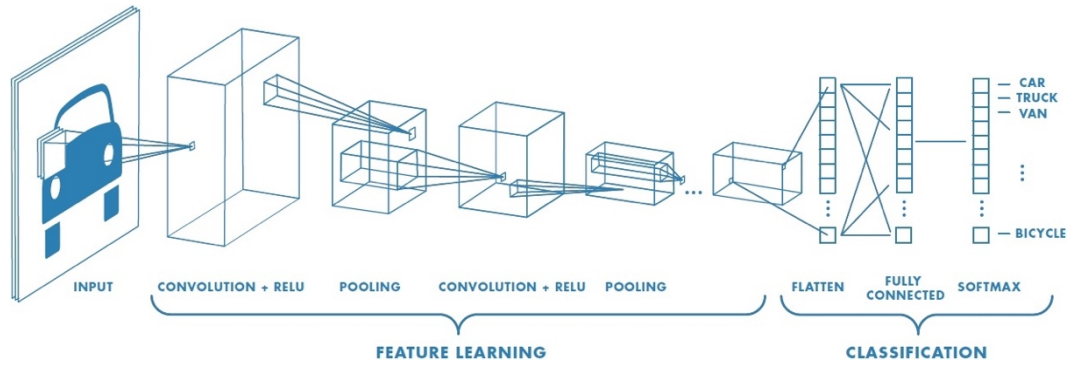


Fig. 2.6 An example of Neural network with many convolutional layers [9]

2.3 3D convolutional neural network

CNNs have achieved good results for action recognition, but most of the approaches only use frame-based features, the temporal information is neglected. 3D CNN is a good solution to this issue.

As shown in Fig. 2.7, in the 2D CNN, the convolution operation is performed at the convolutional layers and applied to the 2D feature maps to extract features from the spatial dimensions only. More specifically, the convolutional filter moves in 2-direction (x, y) to calculate the features from the input volume. However, in 3D CNNs, a 3-dimensional filter will be applied to the input volume and the filter moves 3-direction (x, y, z) to calculate the feature representations, where x and y are spatial coordinates, z represents time. 3D CNNs are widely used video analysis task since they can not only capture spatial information but also temporal information. To my knowledge, Ji et al. [12] propose a 3D CNN model to recognize human actions in real world environment, which is the first 3D CNN used for action detection.

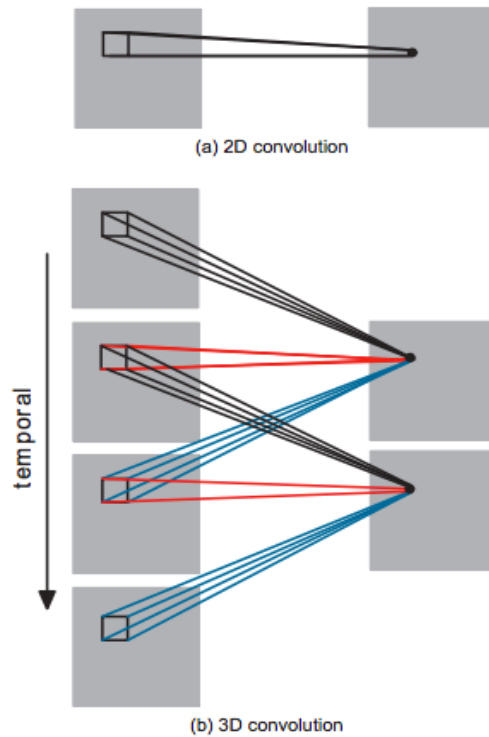


Fig. 2.7 The difference between 2D convolution and 3D convolution [10]

2.4 Generative adversarial network

Generative adversarial networks (GANs) was introduced by Goodfellow. et al. [13] in 2014. It is a framework that has two neural networks:

- Generator: decodes a random vector into a synthetic image
- Discriminator: classify generated images and real images.

These two networks against each other, i.e. the generator trying to fool the discriminator, the discriminator also improves its ability to distinguish the fake image generated by the generator.

As exemplified in Fig. 2.8, the generator takes a random vector as input and generate an image, then feed it to the discriminator. The discriminator takes as input a real or generated image and predicts whether it's from the real world or generated by the generator. The generator gets the feedback from discriminator, start a new loop. In the meanwhile, the discriminator is constantly adapting to the capabilities of the generator. Once training is done, the generated image should indistinguishable from real images.

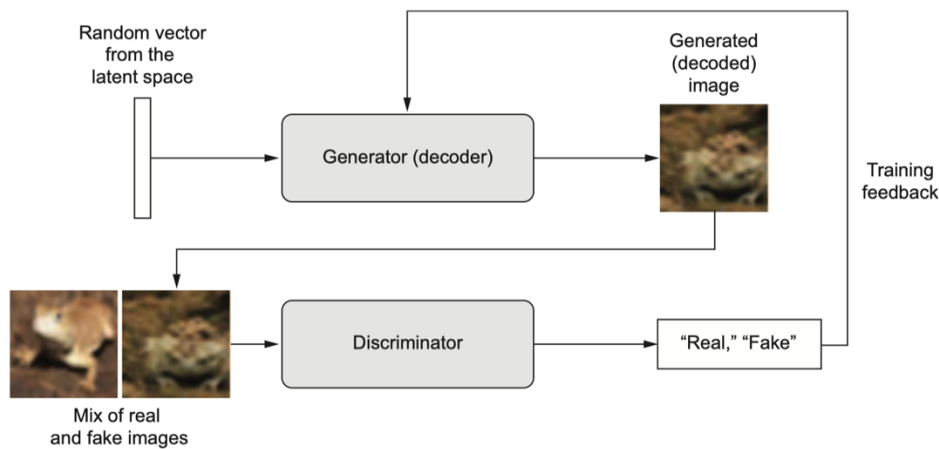


Fig. 2.8 An example of GAN [14]

Schematically, the generative adversarial network looks like this [14]:

- A generator network (G) takes random noise as input and maps it to an image of shape (w, h, c) , where w is the width, h is height and c is the channel.
- A discriminator network (D) takes images of shape (w, h, c) as input, determine whether the image is real or generated by the G .
- Setup a GAN network, which chains the generator and the discriminator: $GAN(x) = D(G(x))$.
- D is a classifier. When we train it, the input data is a mixture of real and generated images along with "real" or "fake" labels.
- To train G via GAN. The weights of G are moved in a direction that makes D to classify the generated images as "real". In other words, we train the G to fool the D .

Unlike other networks, GANs are not easy to train because training a GAN is a dynamic process. Technically is a minimax game, an increase in one network's (G or D) score results into the decrease in another network's score. There are some tricks to train the GAN [14]:

- For the activation in the generator, we use tanh instead of sigmoid.
- Sample random points from a normal distribution instead of a uniform distribution. Usually, we use the standard normal distribution.
- Use LeakyReLU layer instead of a ReLU activation since the derivative of the LeakyReLU in the negative part is not 0.

2.5 Transfer learning

“Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned.” [15].

Generally, we train a neural network on a dataset so that this network gains knowledge from this data. Instead of training the network from scratch, we can directly use this pre-trained network on another dataset.

There are few reasons we use the transfer learning:

- The dataset is not large enough so that it is rare to get enough data to train a neural network from scratch. Using pre-trained weights as initializations or a fixed feature extractor helps in solving most of the problems in hand.
- Training a deep network is time-consuming, a complex model may take weeks to train using machines equipped with expensive GPUs.
- Sometimes, determining the training method or hyperparameters for a network is like alchemy with not much theory to guide you.

In the deep neural network, the features are generic in early layers and specific to the details of the classes contained in the original data in later layers. Considering the size of the new dataset and data similarity, we can list four scenarios [16]:

1) **Small dataset, high similarity:**

In this situation, training the model from scratch is not a good idea due to overfitting concerns. Since the data similarity is high, we expect higher-level features in the network to be relevant to this new dataset as well. So, the best idea is to customize and modify the output layers according to our problem statement and freeze the early layers.

2) **Small dataset, low similarity:**

In this case, we may want to fine-tune the network from the earlier layer. We could freeze very few early layers, fine-tune the rest layers. For example, in the convolutional neural network, we can freeze convolution layers, train the fully-connected layers.

3) **Large dataset, high similarity:**

This is the ideal situation. Since we have enough data, the network will not be overfitted easily. The best way may be to use the pre-trained weight to initialize the new network, fine-tune the whole network.

4) **Large dataset, low similarity:**

Since the new data is very different as compared to the original data, can directly train the network from scratch.

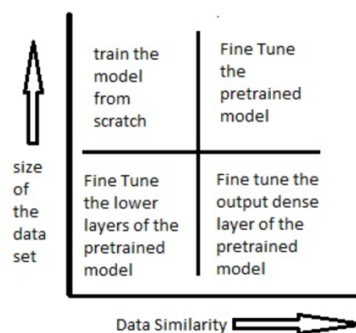


Fig. 2.9 A diagram that helps us to decide how to train our networks

2.6 Evaluation metrics

Evaluating a deep learning algorithm is an essential part of any project. Choosing the right metric is also important for you to judge your model.

Let's first take an example and make the following definitions.

We want to classify whether an image contains a dog or not. So, "dog" is a positive class, "no dog" is a negative class.

True positive (TP): The image contains a dog and the model says it's a "dog".

True negative (TN): The class of the image is "no dog", the model says it's a "no dog".

False positive (FP): The image contains a dog, but the model says it's "no dog".

False negative (FN): The class of the image is "no dog", the model says it's a "dog".

So, TP and TN are correct predictions, and FP and FN are wrong predictions. To measure the performance, one of the most commonly used metrics is accuracy, which has the following definition (for binary classification):

$$accuracy = \frac{TP + TN}{Total\ number\ of\ predictions}$$

However, it is a poor measurement since it is dominated by non-relevant elements (TN and FP).

Precision and recall are the two other fundamental measures of search effectiveness. Precision is the proportion of relevant elements among the retrieved elements, it is defined as

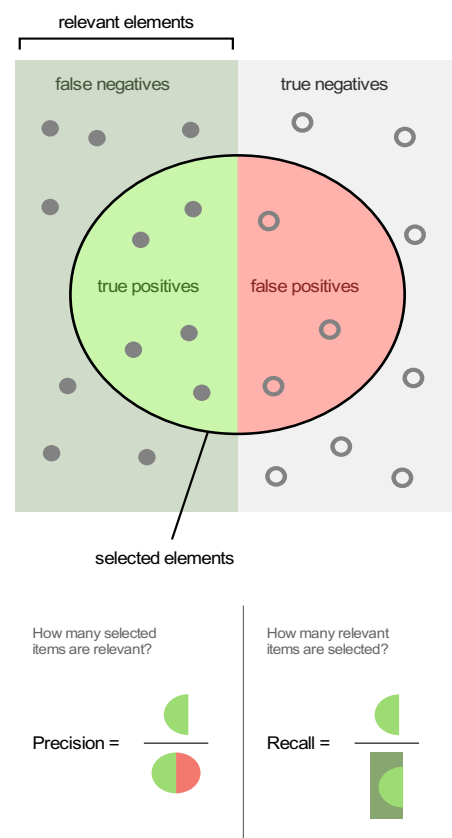


Fig. 2.10 Precision and recall [15]

$precision = \frac{TP}{TP+FP}$. While the recall is the proportion of relevant elements that have been retrieved over the total amount of relevant elements [17]. Recall is defined as: $recall = \frac{TP}{TP+FN}$. Improving precision typically reduces recall and vice versa. A good classifier will keep its precision stay high as recall increases. Usually, we present a precision-recall curve to show how this tradeoff looks for our classifier.

Another commonly used metric is average precision, which is a measure that combines precision and recall for ranked retrieval results. Let's denote the rank as k , the average precision (AP) is defined as follow:

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

Where $P(k)$ is the precision at rank k , $\Delta r(k)$ refers to the change in recall between cutoff $k - 1$ and cutoff k .

There is another metric also called interpolated average precision; often, people still call it average precision. The difference is, at rank k , the interpolated average precision uses the maximum precision over all k recalls. The interpolated AP can be defined as follow:

$$interpolated\ AP = \sum_{k=1}^n \max_{\check{k} \geq k} P(\check{k}) \Delta r(k),$$

In classification problem, we calculate the AP of each class, then compute the average over all classes. The average of all AP is call mean average precision (mAP).

3 Related works

Action detection is an active research topic in intelligent video analysis and is gaining extensive attention in academic and engineering communities. In this chapter we review the methods for action detection published in recent years.

3.1 Action classification

Action classification is also called action recognition. It refers to the act of classifying an action that is present in a given video. Both training and testing data are trimmed video clips that have a short duration and contain only one action. The topic of human action recognition has been widely exploited in the recent decade, and various networks have been proposed, such as single stream network [18] and two-stream convolutional network [19]. The single stream network [18] was built on a pre-trained 2D convolutional neural network. The authors were trying to fuse the temporal information, but they ignored the motion features. For this reason, their results were worse than the state-of-art articles based on the hand-crafted feature. Build on the failure of single-stream network, two-stream convolutional network pays more attention to the motion features. The two-stream network has two separated networks: A spatial-stream network for spatial information; and a temporal network for motion context. Since the two-stream network can capture both temporal and spatial information, it improved the performance of the single-stream network. But there were still a few disadvantages: (1) The long temporal information was missing. (2) All video clips are samples from the entire video and labelled as the ground truth of the video. If the duration of the action is short, some clips may contain backgrounds only. Thus, there is a “false label assignment” problem. (3) We have to train two networks respectively and it’s time-consuming.

3.2 Temporal action localization

Temporal action localization refers to locating actions in long untrimmed videos and classify the located action. This is more challenging than the action recognition since it requires the system to correctly detect the start time and end time.

Zheng et al. [20] proposed a Multi-stage CNN, which has 4 steps:

1. Generate Multi-scale varied lengths segments
2. Predict action for each segment. If the segment is an action, go next step
3. Classify the segments that labelled as action.
4. Localizes action instances in time and outputs confidence scores.

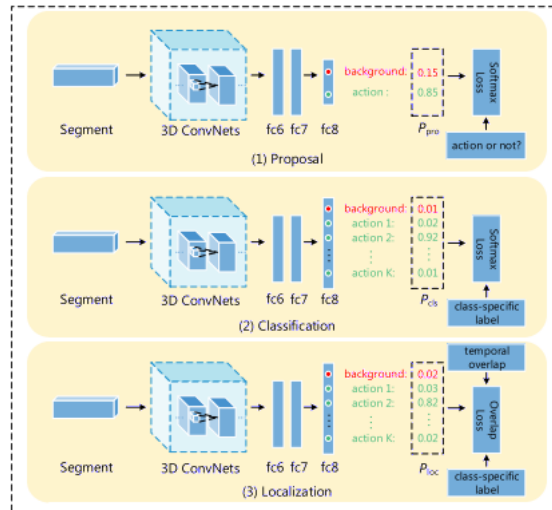


Fig. 3.1 Multi-stage CNN [20]

Shyamal Buch1 et al. [21] proposed end-to-end network (SS-TAD) for temporal action detection. They use the pre-trained C3D network to capture low-level spatiotemporal information. Then they pass the extracted feature to two recurrent memory modules P and C. P is used to accumulate evidence across time to help the model to distinguish action

from the background, C is used for classifying actions. Their model processes the input video in a “single-stream”, detect the end of the action and remember the start of the action.

3.3 Early Recognition and Detection

Similar to online action detection, early action detection also aims to detect the action as soon as it happens in a streaming video, but the videos used in early recognition literatures are usually relatively short. To my knowledge, Shugao Ma et al. [22], are the first author designed an LSTM network to deal with Action detection and early action detection. For activity detection, they detect the class of the action and its start and end. For early detection, they detect the class and the start points of actions after observing only a fraction of the action. Online action start detection is closer to this task. Ma et al. propose to cut the first certain portion of the testing video and then conducting temporal localisation on the cut video. The difference is they not only detect the start of actions but also focuses on detecting whether the action ends or not.

3.4 Online Action Detection

Online action detection aims to detect the occurrence of action as soon as possible, De et al. [3] first simulated this problem using untrimmed videos. Gao et al. [6] implemented an Encoder-Decoder network based on an LSTM network. In their work, the encoder encodes history information and the decoder anticipates future video representations. Also, there is a reward network to make correct anticipation as early as possible. To my knowledge, they achieved the state-of-the-art performances on the THUMOS’14 in online action detection task. However, both of them use the per-frame labelling strategy, which requires the network correctly classifying every frame rather than just detecting AS. A good per-frame labelling method might not be necessarily good at detecting AS.

The RED module consists of three parts:

1. Video representation extractor

A video is segmented into small chunks, each chunk contains 6 consecutive frames. In order to extract frames from video, they investigate two video feature extractors: two-stream model and VGG-16 model.

2. Encoder-decoder network (anticipation module)

This network uses an LSTM network as basic cell. The encoder encodes history information and the decoder anticipates future video representations. It's a Recurrent neural network.

3. Classification module

This network has two fully connected layers, to output a classification distribution on action categories. The loss function of classification is cross-entropy:

$$L_{cls} = \frac{1}{N} \sum_{K=1}^N \sum_{t=1}^{T_{dec}} \log (p(y_1^k | y_{1:t-1}^k)) ,$$

where P_y^j is the probability.

4. Reinforcement module

A natural expectation of action prediction is to make the correct prediction as early as possible. However, cross-entropy loss is calculated at each step to output higher confident scores on the ground truth category. To solve this issue, the author designed a reward function to encourage the agent to make the correct anticipation as early as possible.

3.5 Datasets

Thumos'14.

THUMOS'14 [23] is the second THUMOS challenges, took place the following year in conjunction with the 2014 European Conference on Computer Vision (ECCV). Two main

tasks were offered in the THUMOS'14 challenge: the action recognition task and the temporal action localization task. For the action recognition task, the goal is to predict the presence/absence of an action class in a given sequence. The objective of the temporal action localization task, however, is to identify the class of the 20 pre-defined actions. For both tasks, also, detect the start and end of the actions. There are four types of dataset: training set, validation set, background set, and test set. The training data were videos extracted from the UCF101 dataset, which were trimmed such that each sequence contains only one action instance and. The other three datasets were collections of untrimmed videos. Fig. 3.2 shows the sample frames of the actions from THUMOS'14 dataset.



Fig. 3.2 sample frames of the actions from THUMOS'14

For the action recognition task, the entire UCF101 dataset which consists of temporally trimmed videos is provided for training. The validation set consist of 1000 untrimmed videos in total to allow participants to fine-tune their algorithms and to use as further training data, if necessary. there is at least one primary action in each video, some videos contain one or more instances of other classes. The background videos, which consist of 2500 video clips, were videos relevant to each action but did not contain any actions. For example, background data for “baseball” is a clip of a baseball court without a baseball game taking place. The test data contains 1574 untrimmed videos, which contains one or more instances of one or multiple action classes. There are 11 teams took part in the challenge and 35 runs were submitted. Four participants used convolutional networks while 10 used DT features. Using iDT features with CNN and SVM as classifier, the winner of the action recognition challenging task achieved the mAP score of 0.71. The action recognition task of THUMOS'14 was deemed more challenging than the previous

challenging as the test videos were untrimmed, which meant that significant portion of some videos did not contain any of the 101 actions. Furthermore, variations of instances, where multiple or no instance of any actions were possibilities in test videos, was another factor that made the classification task more difficult than the before. These added features in the test videos were embedded to the competition to guide the next generation of action recognition algorithms to be more useful in practical settings.

For the temporal action detection task, participants are requested to localize 20 action classes. The training data consist of trimmed videos from the UCF101 dataset of the 20 action classes, 200 validation videos with temporal annotations of all instances of the 20 actions were provided in the validation set, the background videos are same as the previous task. There are 1574 temporally untrimmed videos in the test data. As in the action recognition task, each action class and each run were measured by interpolated AP and mAP metrics, respectively. A detection was considered correct if the intersection over union (IoU) score was greater than 0.5. However, only 3 teams took part in this challenge with 11 submissions. All of them used the iDT with CNN features and used SVM over temporal windows. The variation amongst the three approaches depended on using either the early or late fusion of the features, parameters and combining with classification scores. The best performance work attained a score of 0.14.

Nowadays, THUMOS'14 is still a popular dataset for action detection, a lot of literature are working on this dataset. Such as Reinforced encoder-decoder networks [6].

ActivityNet

In conjunction with CVPR 2016, the action recognition challenge of ActivityNet took place, and the challenge is still open. Different from the THUMOS'14 challenge, the ActivityNet Challenge 2018 comprised of three main tasks: Temporal Action Proposals, Temporal Action Localization and Dense-Captioning Events in Videos. The objective of the Temporal Action Proposals challenge was to produce a set of segments that are seeming to contain a human action. The Temporal Action Localization is similar to the THUMOS'14, temporally localize actions in an untrimmed video. The last one is more interesting; it requests the system to detect and describe the action.

4 Design and implementation

In this chapter, we introduce the framework and methods for action detection, which are used throughout the thesis.

Zheng et al. [4] proposed a 3D convolutional network to detect the start of actions. Firstly, they defined the action start (AS) window and its follow-up window. As shown in Fig. 4.1, the start window consists of background frames and action frames. Secondly, they identified three challenges:

- Training data scarcity. The percentage of AS window is much lower than the others.
- In the feature space, the AS window may close to the background window since it's a mixture of background and action frames.
- Some background windows may share similar scenes with AS windows, such hard negative should be identified.

To address the challenges, they implemented three novel methods:

- Adaptively sampling training batches.
- Modelling the temporal consistency between the AS window and its follow-up window.
- Generating hard negative samples via GAN.

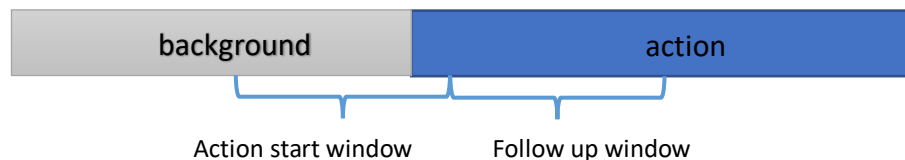


Fig. 4.1 The start window and its follow-up window

Since they achieved the state-of-the-art performances on THUMOS'14 (to my knowledge) I reimplemented the networks and methods. The networks are coded in Keras 2.2.0 framework that running on top of TensorFlow.

4.1 Backbone network

The difference between video and images is the videos contains temporal information. Thus, the network should have the ability to handle both temporal and spatial information. There are a lot of networks has been Implemented to deal with this problem, suck C3D [7] and TSN [24]. I use C3D as my backbone network since it's used by Zheng et al. [4] in their network.

4.1.1 C3D network

The C3D network was proposed by Tran et al [7]. It is a 3D CNN for action recognition. As we mentioned in section 2.1, 3D CNNs are more suitable for spatiotemporal feature learning, the C3D model has been exploited for temporal action localization and action detection in untrimmed videos [25] [26].

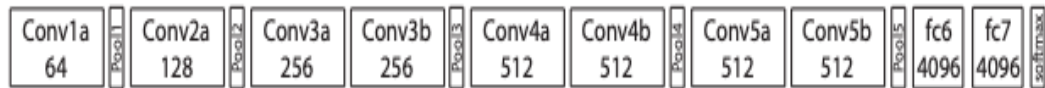


Fig. 4.2 C3D architecture [7]

The C3D network architecture is presented in Fig. 4.2. It consists of 8 convolution layers, 5 max-pooling layers, followed by two fully connected layers, and a softmax output layer. The C3D model has trained on the Sport-1M dataset.

4.1.2 Implementation of C3D network

I keep the same backbone architecture as original C3D while setting the number of outputs in the last layer to $K + 1$ (K action classes and 1 background).

The C3D network can be divided into four parts:

Input Layer. The model takes a $16 \times 112 \times 112 \times 3$ volume (16 112×112 consecutive frames with 3 channels) as input and passes it to the following layers.

Convolutional Network (ConvNet). The ConvNet consist of convolution layers and max-polling layers (Conv1 to Pool5). It is proved that $3 \times 3 \times 3$ is the best filter choice for 3D convolutional networks [7]. Thus, the filters in all convolution layers are $3 \times 3 \times 3$ with stride $1 \times 1 \times 1$. The convolution layers take previous layers' output as input, perform a convolution operation, output a 4-dimension vector (extracted features). Since the convolution operation is a linear operation, the outputs would lose the classification capabilities. Thus, a non-linear function, ReLU activation function, is applied to the output of each convolution layer. In order to reduce the computational cost, five max-pooling layers are applied. The kernel size of the max-pooling layers is $2 \times 2 \times 2$ with stride $2 \times 2 \times 2$, except pool1 which has kernel size of $1 \times 2 \times 2$, and stride $1 \times 2 \times 2$ with the intention of preserving the temporal information in the early phase.

In conclusion, the ConvNet takes a $16 \times 112 \times 112 \times 3$ vector as input, output a feature vector to server as input to the next layer. The output dimension of each layer is shown in Table 4.1.

Fully Connected Layers. There are two fully-connected layers, each layer contains 4096 nodes. As same as the convolution layers, a ReLU function is also applied to the output of each fully-connected layer. In order to avoid overfitting problem, each fully-connected layer (FC6 and FC7) is followed by a dropout layer with dropout-rate 0.5.

Output Layer. The output layer (FC8) is also a fully-connected layer, it output a $K + 1$ dimensions vector (o_1, \dots, o_{K+1}) , where K is the number of actions, and the $K + 1$ -th output stands for the background. Finally, a softmax function takes as input a vector of the $K + 1$ outputs and normalizes it into a probability distribution consisting of $K + 1$ probabilities.

For each class i : $P_{model}(i|x) = \frac{e^{o_i}}{\sum_{k=1}^{K+1} e^{o_k}}$.

layers	C3D model kernel dims (D, H, W)	output dims (D, H, W, C)
Conv1	(3, 3, 3)	(16, 112, 112, 64)
MaxPool1	(1, 2, 2)	(16, 56, 56, 64)
Conv2	(3, 3, 3)	(16, 56, 56, 128)
MaxPool2	(2, 2, 2)	(8, 28, 28, 128)
Conv3a	(3, 3, 3)	(8, 28, 28, 256)
Conv3b	(3, 3, 3)	(8, 28, 28, 256)
MaxPool3	(2, 2, 2)	(4, 14, 14, 256)
Conv4a	(3, 3, 3)	(4, 14, 14, 512)
Conv4b	(3, 3, 3)	(4, 14, 14, 512)
MaxPool4	(2, 2, 2)	(2, 7, 7, 512)
Conv5a	(3, 3, 3)	(2, 7, 7, 512)
Conv5b	(3, 3, 3)	(2, 7, 7, 512)
MaxPool5	(2, 2, 2)	(1, 3, 3, 512)
FC6	-	4096
FC7	-	4096
FC8	-	K

Table 4.1 kernel size and output size of each layer. K is number of action classes

4.2 Training methodology and additional models

Zheng et al. [4] designed three novel methods to improve the capability of the C3D network to detect the action more accurately and timely. In this section, I will talk about these three methods and two additional models.

4.2.1 Adaptive sampling

Since the goal of online action detection is to detect actions as soon as possible in untrimmed videos, the model has to accurately classify the start of actions. But in untrimmed videos, the number of action frames are much scarcer than the background frames and the action start frames are even more rare. If each training batch is constructed randomly, i.e. sample data from the entire samples, the training batch may not contain action start samples. To address this issue, the authors of the paper [4] developed an adaptive sampling strategy, manually control the number of actions start samples in each batch. More specifically, half of a training batch are action start samples and the other half batch are non-action-start samples (background samples or action samples).

Slightly different from their strategy, I divide the training samples into three groups (the reason is that when I train my model with their strategy, the results are not stable):

- Action Start (**AS**) samples: if and only if the first frame of the action is inside the windows. This kind of samples contain both background and action frames.
- Action (**A**) samples: only action frames are inside the samples.
- Background (**BG**) samples: the sample is labelled as background, i.e. the last frame of the window is a background frame.

For each training batch, I randomly sample half of the training batch from AS, a quarter from A, and a quarter from BG. Using this strategy, the model can learn a better classification boundary to distinguish start windows against negatives more accurately, as illustrated in Fig. 4.3 (a). The training batches are fed into the C3D network. I train the network via minimizing the multi-class classification softmax loss $\mathcal{L}_{classification}$. Since the

network is coded in Keras framework and it is a many-class classification problem, I use categorical cross-entropy as the loss function, which provided by Keras.

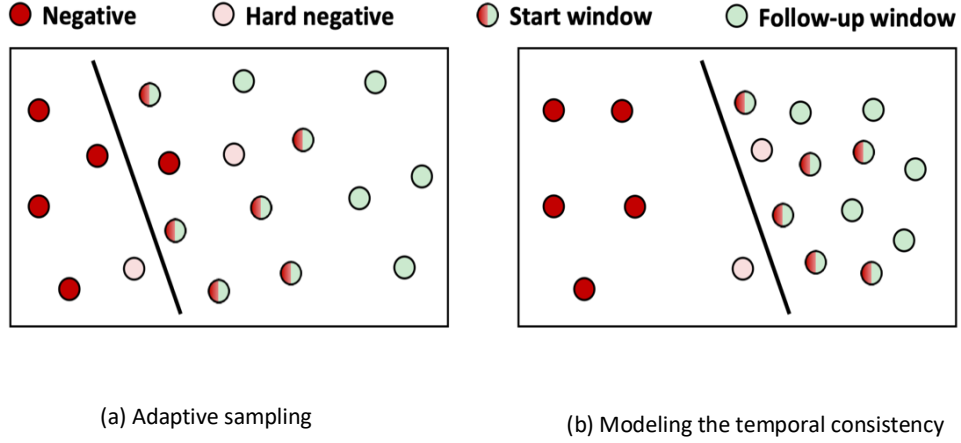


Fig. 4.3 An illustration of data distribution [4]

4.2.2 Temporal consistency

Since the AS window contains both action frames and background frames, the feature of the AS window may be closer to the preceding background window than the action window after the start. For instance, a start window consists of 15 background frames and 1 action frames. Thus, in the feature space, it could be close to negatives. To address this problem, Zhen et al. [4] propose to model the temporal consistency between the action start window and its follow up window during training.

As illustrated in Fig. 4.1 and Fig. 4.3 (b), the follow-up window is completely inside the action. Thus, in the feature map, they should be far away from negatives. The authors developed the temporal consistency method to encourage the feature of action start window move closer to the feature of action window. Therefore, the AS windows become more separable from background windows.

Since the temporal consistency has two data flow, I implement a Siamese mode of C3D. The Siamese model contains two identical subnetworks (from Conv1 to FC7) with shared weights, as shown in Fig. 4.4. The first subnetwork plays the same role as C3D network, a

classifier. It takes the start window as input, output the probabilities of the $K + 1$ actions. While, the second subnetwork takes the start window's follow-up window as input, extract the feature, then combine the output of FC7 of the two subnetworks, compute the Euclidean distance.

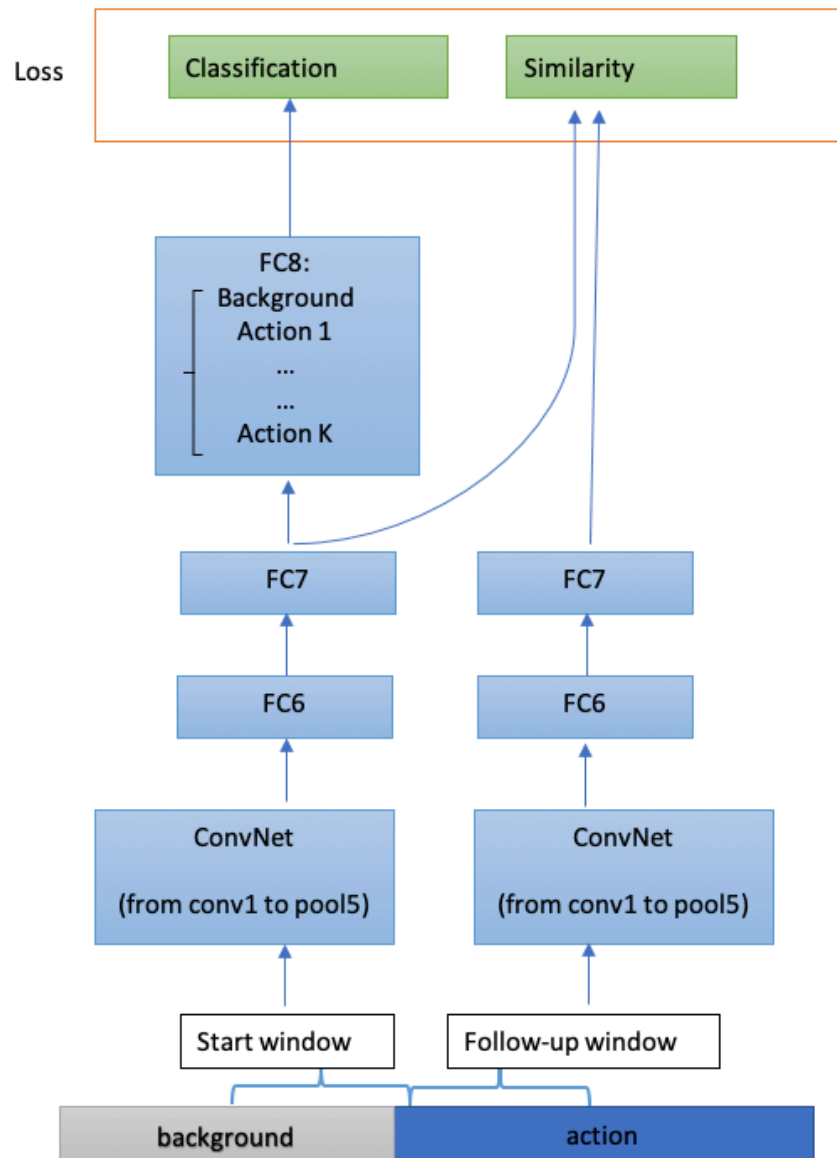


Fig. 4.4 The temporal consistency model

In more detail. The paired input is action start window x_s and its follow-up window x_f . As same as the original paper, I set the output of FC7 to be the extracted feature because it is the data for classification. The feature of x_s and x_f are $F(x_s)$ and $F(x_f)$, the similarity loss $\mathcal{L}_{similarity}$ is:

$$\mathcal{L}_{similarity} = \| F(x_s) - F(x_f) \|_2^2, \quad (1)$$

4.2.3 Generate hard negative data via GAN

As exemplified in Fig. 4.5, the hard negative has subtle difference from the start window, so it's close to the start window in the feature space. To improve the decision boundary, such hard-negative samples need to be identified in the training data during training. However, it's not feasible to find such samples exhaustively because such hard-negative samples are rare and may even not exist in the training data. Therefore, Zheng et al. [4] use a GAN to automatically synthesise samples that are hard to distinguish from true positives.

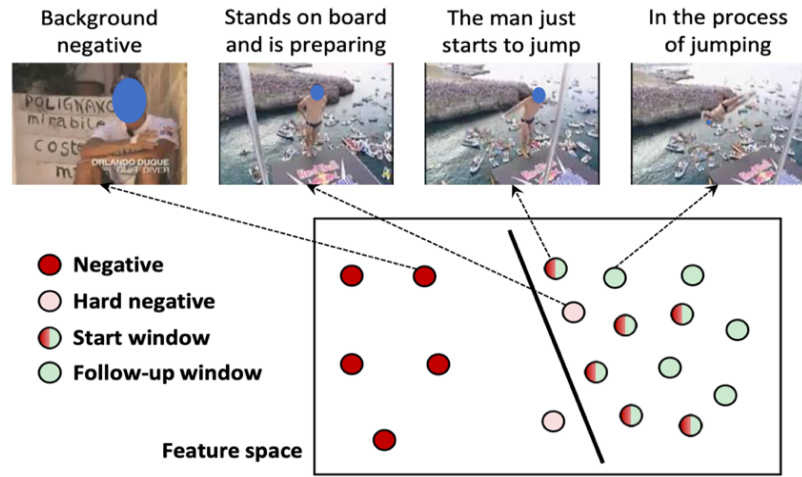


Fig. 4.5 An example of hard negative [4]

The architecture of their GAN model is presented in Fig. 4.6, it consists of two main parts:

- **Generator (G):** It consists of two fully-connected layers, two BatchNormalization layers and two ReLU layers.
- **Discriminator (D):** It consists of three fully connected layers: FC6, FC7 and FC8.

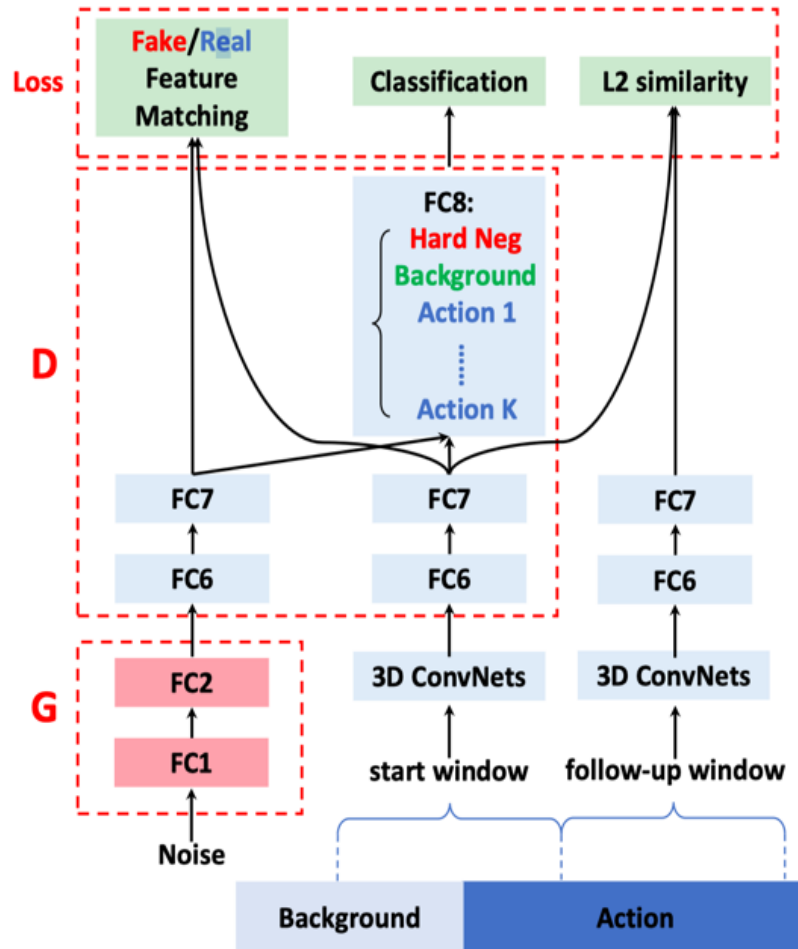


Fig. 4.6 The GAN model [4]

The purpose of the G is to generate hard negative features, because directly generating video is very challenging and not necessary. Their GAN model has a generator to generate fake features and also have a fixed 3D convolution network to extract real features from real data. For the generator, it takes a random noise z as input and learns to capture the

distribution of real action start windows. Following [27], the 100-dimensional vector z is drawn from the standard normal distribution.

According to the information provided by the author, I implemented the GAN model.

Firstly, I implemented the generator (G), which contains 2 fully-connected layers. I use 8192 nodes for both fully-connected layers in G since the output of Pool5 has 8192 dimensions. As same as the original paper, each fully-connected layer is followed by a BatchNormalization layer and a ReLU activation function.

Secondly, I implemented the discriminator (D). D consists of three fully connected layers: FC6, FC7 and FC8. Different from the original paper, FC6 and FC7 followed by a LeakyReLU layer respectively, instead of a ReLU. The reason is that LeakyReLU relaxes sparsity constraints by allowing small negative activation values, it may have a better performance [14]. FC6, FC7 and FC8 share the same weights with C3D network. More specifically, update the weights of D will also update the weight of last three layers in C3D network.

Finally, I setup stacked model, which chains the G and the D. The D in the stacked model is set to untrainable. The stacked model is that, when trained, will move the G in a direction that improves its ability to fool the D.

In summary, there are two differences between my implementation and the original implementation: I use 8192 nodes for fully-connected layers in G instead of 4096; I use LeakyReLU in D instead of ReLU.

When training the G, i.e. training the stacked model, the weights of the D are frozen. Usually, a convolution GAN model turns latent-space points into a binary classification decision—real or fake—and it's meant to be trained with labels that are always "real." However, this method usually encounters an instability issue. Following [28], instead of use a binary classifier, Zheng et al. [4] train G to generate fake features matching the statistics of the real one. Specifically, they train the G to match the expectation of the features on an intermediate layer of the D. The feature extraction process of using 3D convolution

network is donated as $f(\cdot)$ and the process from FC6 to FC7 is donated as $\psi(\cdot)$. The loss of feature matching is defined as follow:

$$L_{matching} = \| \mathbb{E}_{(x_s, y) \sim p_{start}} [\psi(f(x_s))] - \mathbb{E}_{z \sim noise} [\psi(G(z))] \|_2^2, \quad (2)$$

where $G(\cdot)$ denotes the generator, p_{start} is the training set that only contains the action start windows, $\mathbb{E}(\cdot)$ is the average of the tensor.

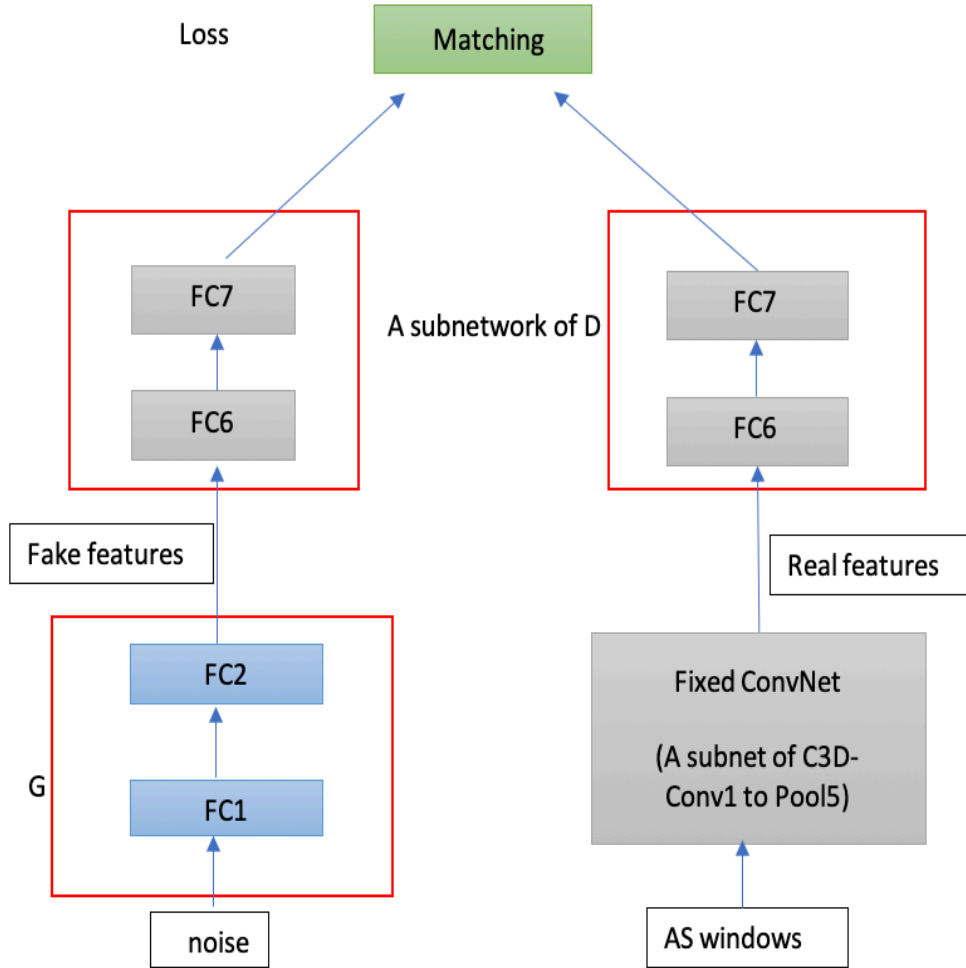


Fig. 4.7 Training G. The weights of FC1 and FC2 will be updated

For the D, it should have the ability to distinguish the fake data from real start windows, so that the generated samples can be regarded as hard negatives. The authors of the paper add an additional node in FC8 layer to represent the generated data instead of using a binary classifier. The motivation is to let the D distinguish hard negatives from not only actions but also backgrounds. Since D is a subnetwork of C3D (from FC6 to FC8), the weights of these three layers will be updated when train the D, i.e. the purpose of D is to fine-tune the C3D model. The loss function L_d is same as C3D, categorical cross-entropy.

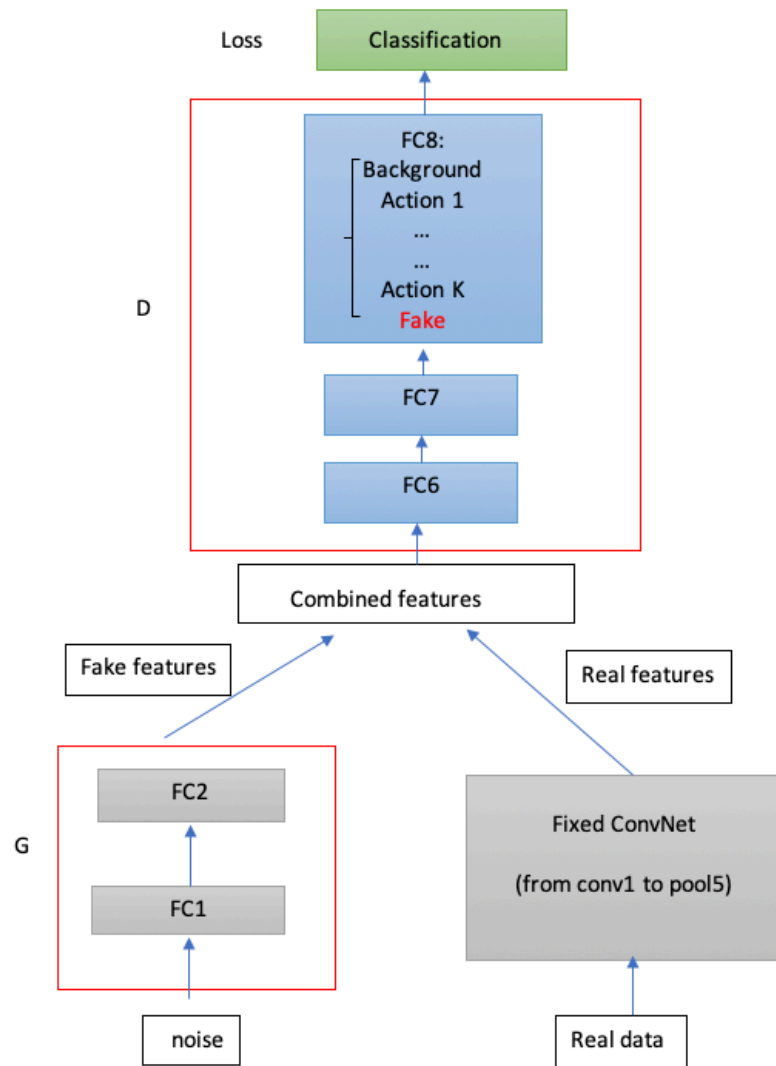


Fig. 4.8 Training D. The weight of FC6, FC7 and FC8 will be updated

4.3 Training steps

First, I use the adaptive sampling strategy to construct the training batches, train the C3D model. Since the C3D is pre-trained on sport-1M dataset, I use this pre-trained weight to initialize my C3D mode.

Secondly, impose temporal consistency constraint, train the Siamese model (TC model). I use the same way to construct the training batches and use the same pre-trained weights to initialize the TC model.

Finally, train the GAN model. I use the weights that pre-trained by TC model to initialize the GAN, fine-tuning the last three layers (FC6, FC7 and FC8). During each iteration, I train the generator and discriminator in an alternation manner:

```
for each iteration:
    * Generate random noise from standard normal distribution.
    * The generator decodes the random noise, generate fake data.
    * Mix the generated data with real ones.
    * Train D using these mixed data, with corresponding labels.
Update the weights of FC layers.
    * Draw new random points in the latent space.
    * Train G via GAN using these random vectors, minimize the
 $\mathcal{L}_{matching}$ . (the labels are not useful because we just need the intermediate
features.)
```

4.4 Testing procedure

During testing, I use test videos to simulate streaming videos. When a new frame arrives at time t , I construct a window (16 frames) ending at t and feed it into the C3D network. The prediction steps are:

1. A new frame arrives at time t .
2. The C3D network takes the window ending at t and make a prediction for the time stamp t .
3. The C3D network outputs:
 - a) time t .
 - b) the classes of the window C_t .
 - c) the confidence score S_t .
4. Compare prediction at $t - 1$ and t , generate a AS prediction if the following conditions are all satisfied:
 - a) C_t is an action;
 - b) $C_t \neq C_{t-1}$;
 - c) S_t exceeds the threshold obtained by grid search on the training data.

The implementation of the prediction code is based on a producer-consumer algorithm. The producer will pre-fetch a few videos into memory, then waiting for the prediction task to process it.

During testing, only the C3D model is used. The C3D model is initialized by three different weights, which were trained by three different methods (adaptive sampling, TC and GAN). Note that the GAN model has $K + 2$ outputs, where the $K + 2$ -th output refers to the generated data (fake sample). This additional node is removed during testing.

4.5 *Evaluation protocol*

Zheng et al. [4] analysed commonly used evaluation protocols such as classification accuracy and localization precision and frame level classification mean Average Precision (mAP). However, none of them are suitable for online action detection. Therefore, Zheng et al. [4] developed a novel metric that evaluates the point level mAP for each action start instance.

As we mentioned in section 4.4, the system outputs t , C_t , and S_t . The author proposed to evaluate the detection accuracy and timeliness compared to the ground truths. As for the timeliness, they measure the temporal offset (absolute distance) between the ground truth AS point and the predicted AS point.

Each AS point prediction is counted as true positive if its action class is correct and its offset is smaller than the evaluation threshold. They evaluate the point-level AP for each action class and average overall action classes to get the mAP. Duplicate detections for the same ground truth AS are not allowed. [4]

Since it's a novel metric and no source code available, I implemented it by myself. I calculate the interpolated average precision for each action class and compute the mAP overall action classes.

5 Experiments on THUMOS'14

Zheng et al. [4] trained their networks on THUMOS'14 dataset. To reproduce their work, I trained my networks on the same dataset.

5.1 THUMOS'14 dataset

There are four datasets in THUMOS'14 [29] action detection task:

- **Training data:** a subset of UCF101 dataset, 20 action classes is used for training. The training data are trimmed videos.
- **Validation data:** 200 untrimmed videos were provided in this dataset. Each video includes one primary action instance; some videos may include more than one instances.
- **Background data:** 2500 videos in total, which were verified to make sure they include background only.
- **Test data:** 213 untrimmed videos were provided in this dataset. Each video includes one or more action instances.

The training videos of THUMOS'14 contains only trimmed videos, which are not suitable for online action detection task, following [4], I train my models on the validation data set and use the test data to simulate streaming videos for testing.

5.2 Construct training batches

Before training the model, the first thing is to construct the training set. Zheng et al. [4] just say they use the validation videos for training, so I assume all validation videos are used

when they train their networks. In order to understand if the training is going well or not before I apply the model on the test set, I decide to split the training data (validation videos) into two sub-set: train-set and validation-set. The validation-set is used to tune the hyperparameter. There are 200 untrimmed validation videos, each video contains one or more action instances, how do we perform a split on these videos?

Data analysis. Before training the model, we need to understand the dataset. First, I group the videos by action class, count the number of videos of each class, the statistics is listed in Table 5.1. The sum of the video number is 227, which is great than the actual number (200) of videos. So, some videos include more than one action and the actions could be different. To make sure both train-set and validation-set contains all of the actions, I group the videos by action class, then preform an 80-20 split on each group randomly. Here is the pseudo-split-code:

```
#video split, pseudo code
Train_set #train set,
Val_set #validation set,
for action in actions:
    video_list = group by (videos)
    N_videos = len(video_list) #the number of videos
    random.shuffle(video_list)
    'Split video_list into 80/20'
    'append to train_set and val_set'
'remove duplicate videos from val_set if video in train_set' #a video may
have multiple action instances
```

Index	Action class	Number of videos
1	BaseballPitch	11
2	BasketballDunk	10

3	Billiards	10
4	CleanAndJerk	10
5	CliffDiving	10
6	CricketBowling	17
7	CricketShot	16
8	Diving	20
9	FrisbeeCatch	10
10	GolfSwing	11
11	HammerThrow	10
12	HighJump	10
13	JavelinThrow	10
14	LongJump	11
15	PoleVault	10
16	Shotput	10
17	SoccerPenalty	10
18	TENNISWING	10
19	ThrowDiscus	11
20	VolleyballSpiking	10

Table 5.1 The list of actions and count of videos of validation data in THUMOS'14

Using this strategy, 200 videos are split into two sets. The details are listed in Table 5.2. Since different training dataset may lead to a different result and I will train the network multiple times, the split operation is performed only once.

	train-set	validation-set
Number of videos	167	33
Number of action instances	2525	482

Table 5.2 Statistics of the train-set and validation-set

Generate images and construct training samples. When training, I use images instead of videos. Since all videos are 30 FPS, I generate 30 images for each second of video by using OpenCV library.

The original C3D [7] network takes 16-frame samples as input. Therefore, I split each video into 16 frame long samples with a 15-frame overlap between two samples. All frames are resized into 128×171 (half resolution of the videos). During training, the frames are also randomly cropped to 112×112 (centre crop during testing). The statistics of train-set and validation-set are listed in Table 5.3. An AS sample contains both background frames and action frame, an A sample contains only action frames, a BG sample contains background frames.

	train-set	validation-set
Number of AS samples	40142	7667
Number of A samples	244045	65343
Number of BG samples	686682	144269

Table 5.3 Statistics of train-set and validation-set

Construct training batches. Once we get the train-set and validation-set, we can construct the training batches. As we can see from Table 5.3, the number of AS samples is much lower than others. If I randomly sample each batch from the entire samples, the training batches may not contain AS samples. Since I want to detect the start of actions, the AS samples are important while training. In order to increase the percentage of the AS samples, I sample half batch from AS samples, sample the other half batch from A samples and BG samples. More specifically, each training batch consists of AS samples, A samples and BG samples in the proportion 2:1:1. Each sample is labelled as class of the last frame of the window.

5.3 Training phase

I implemented three models: the C3D model, the Siamese model and the GAN model. These models are trained on Amazon Web Services, p2.xlarge instance, which is equipped with K80 GPU.

5.3.1 Training C3D model

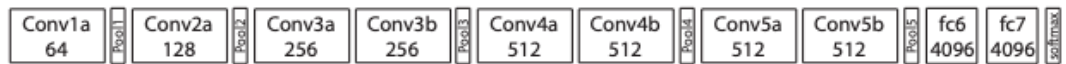
Since the C3D model was pre-trained on the sports-1M dataset, Zheng et al. [4] use the pre-trained weights to initialize the model. They use Adam optimizer and trained 5000 iterations; the learning rate is $1e-4$ for the FC8 layer and $1e-5$ for the rest layers. In all my experiments, I use the same pre-trained weights to initialize the C3D model and tried different hyperparameters and optimizer.

Training a 5-convolution-layer C3D model.

To reduce the computation time, in the first experiment, I use a new version of C3D network that released by the author of C3D [30]. It contains only five convolution layers (see Fig. 5.1 a). I use Adam optimizer with batch size 16 and 16 training epochs (1 epoch approximately equal to 5K iterations). Learning rate is $1e-4$ for the FC8 layer and $1e-5$ for the rest layers. Each batch consists of 8 AS samples and 8 non-AS samples (A and BG samples).



(a) 5 convolution layers C3D



(b) 8 convolution layers C3D

Fig. 5.1 The architecture of two different C3D networks

However, as we can see from Fig. 5.2, the accuracy on the val-set is not good. The model seems overfitted. Besides, the acc is barely satisfactory; a more complex model is needed.

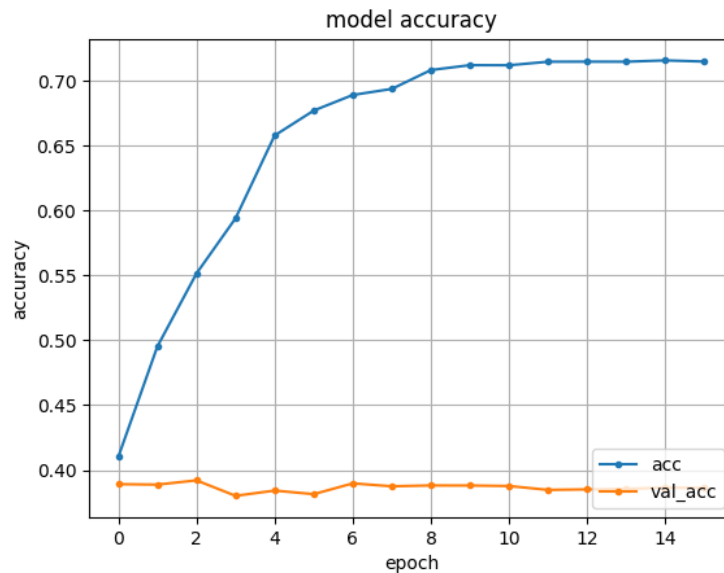


Fig. 5.2 The training result of 5 layers C3D. the acc is the accuracy on the train-set, while, the val_acc is the accuracy on the validation-set

Training on the sub-set of TUMOS'14.

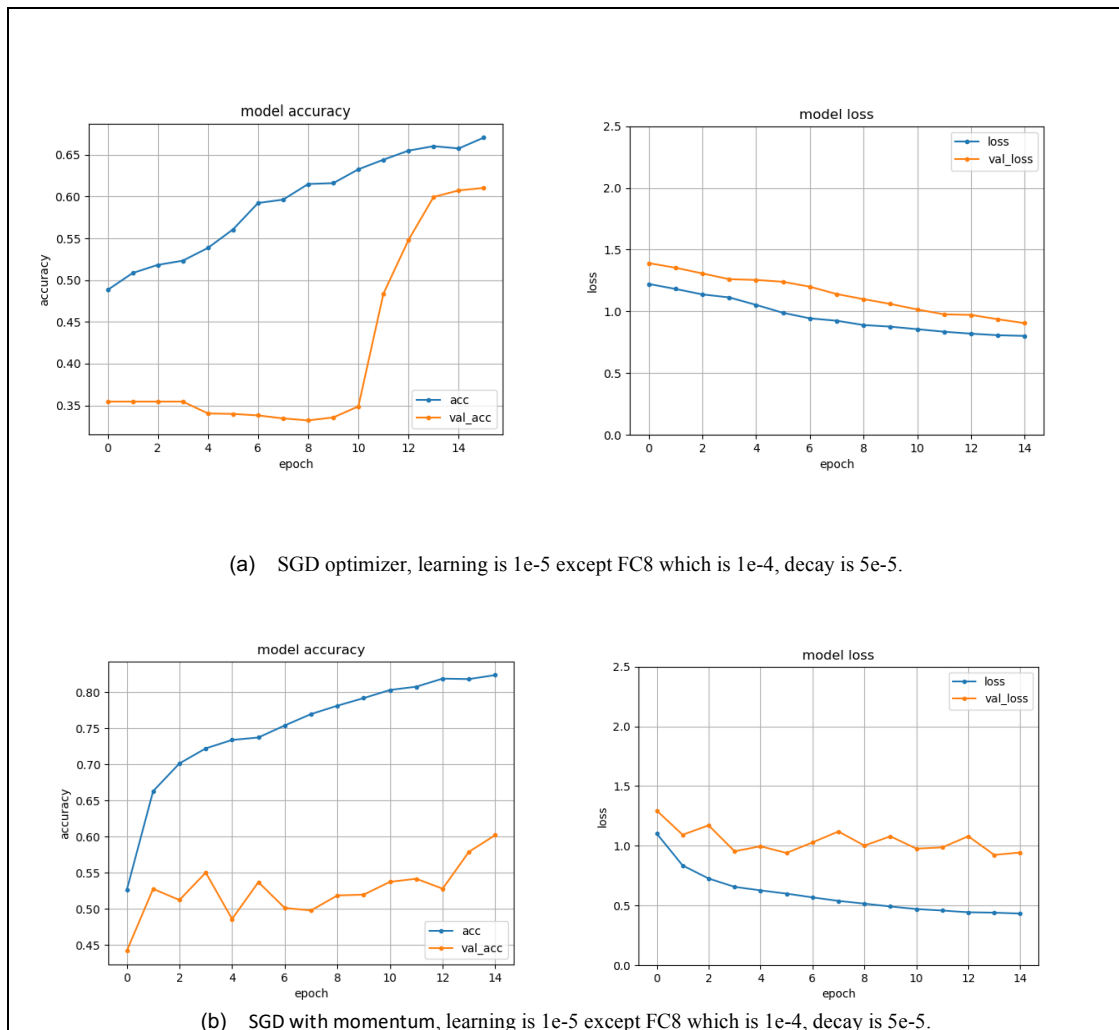
To increase the complexity of the model, three more layers (Conv3b, Conv4b and Conv5b; see Fig. 5.1 b) were added to the C3D model.

Since training the network with the entire train-set consumes too much time, I simply extract 3 action classes (BaseballPitch, BasketballDunk and Billiards) to construct a sub-dataset. By using this sub-dataset to train and fine-tune the hyperparameters, the training time from 8 hours down to approximately 1 hour per epoch.

In this experiment, I first trained the model with the following setting: learning rate is $1e-5$ except $1e-4$ for the last layer, optimizer is Adam, activation function is ReLU. To obtain a better setting, I re-trained multiple times with the following changes:

- Add a l2 kernel_regularizer to the convolution layers. (The l2 kernel_regularizer is a Keras regularizer that allows to apply penalties on layer parameters-sum of the squared weights-during optimization. It provides an approach to reduce the overfitting.)
- Use leakyRelu instead of ReLu
- Use smaller learning rate
- Use SGD optimizer instead of Adam
- Use SGD optimizer with momentum

As shown in Fig. 5.3(a), there is no overfitting problem when using the SGD optimizer, but both the val_loss and the val_acc are worse than the validation accuracy in Fig. 5.3 (c).



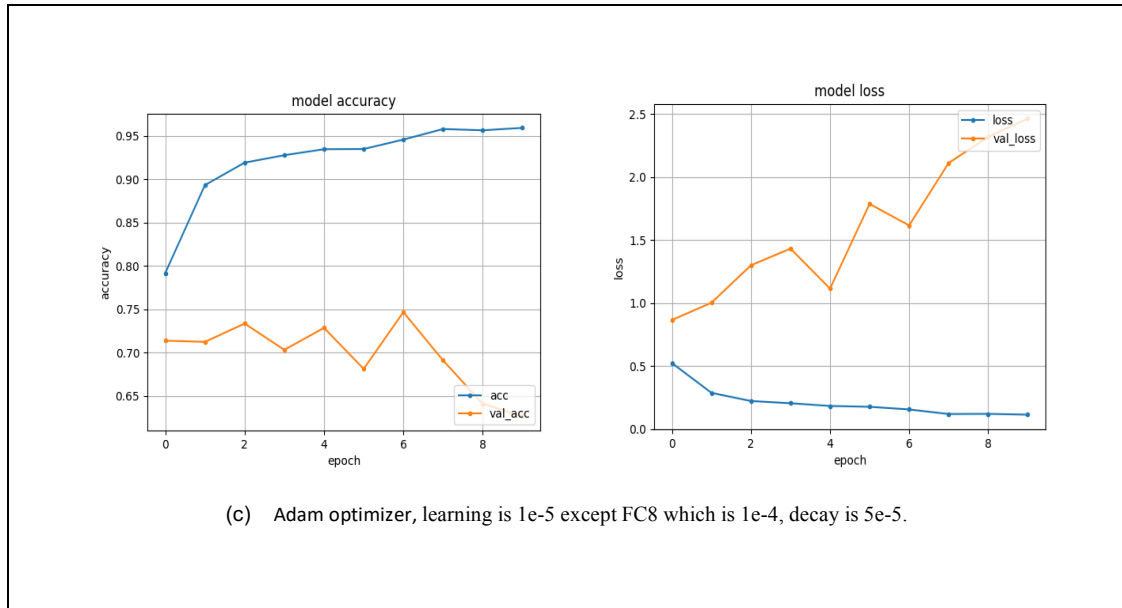


Fig. 5.3 The results of C3D model on the sub-train-set (3 action classes)

Compare the results and training time, Adam optimizer has a better performance than SGD optimizer: using SGD, the loss decreased smoothly, but it still worse than the model that trained with Adam optimizer with only one epoch. Also, I tried to use SGD with momentum and learning-rate scheduling, the performance is still bad.

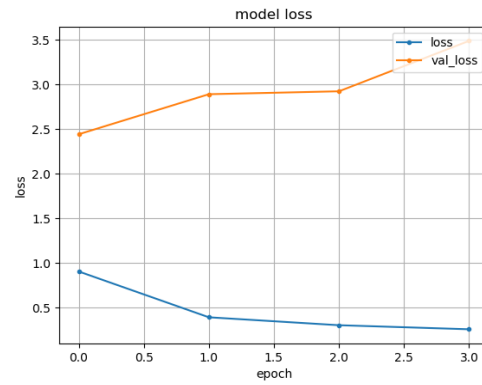
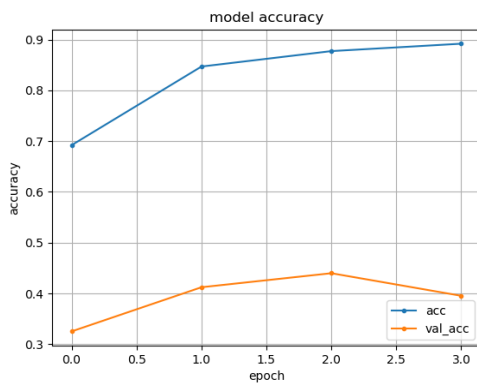
Training on the whole THUMOS'14 dataset.

On the whole train-set, I trained the model with different settings. The parameters are listed in Table 5.4; the accuracy and losses are presented in Fig. 5.4.

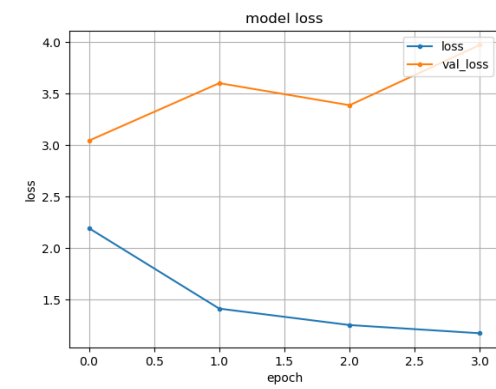
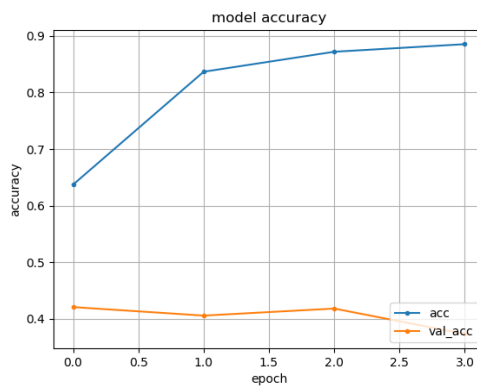
As we can see from Fig. 5.4, setting 1 is the best, and it is the same setting as the original paper: use Adam optimizer; learning rate is $1e-5$ except $1e-4$ for FC8 since it is randomly initialized; activation function is ReLu and l2 kernel_regularizer is not applied. But it suffers from an overfitting problem.

	optimizer	l2 kernel_regularizer	Learning rate
1	Adam	No	Conv1a–Fc7: $1e-5$ FC8: $1e-4$
2	Adam	Yes	Conv1a–Fc7: $1e-5$ FC8: $1e-4$
3	SGD	No	Conv1a–Fc7: $5e-5$ FC8: $5e-4$
4	Adam	No	Conv1a–Conv5b are fixed (not trainable). Fc6 and Fc7: $1e-5$ FC8: $1e-4$
5	Adam	No	Conv1a–Conv4a are fixed (not trainable). Conv4b–Fc7: $1e-5$ FC8: $1e-4$

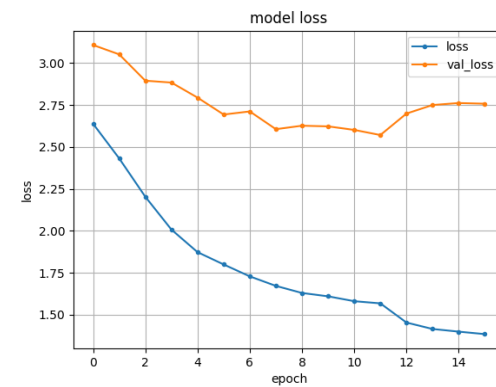
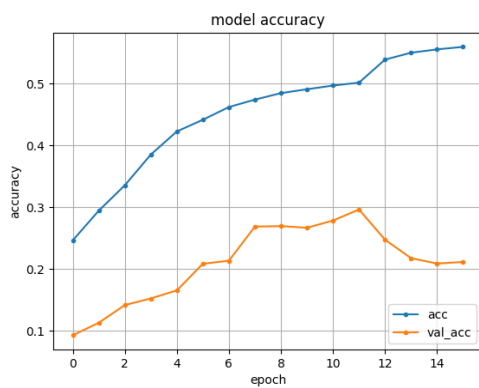
Table 5.4 Different settings of C3D model



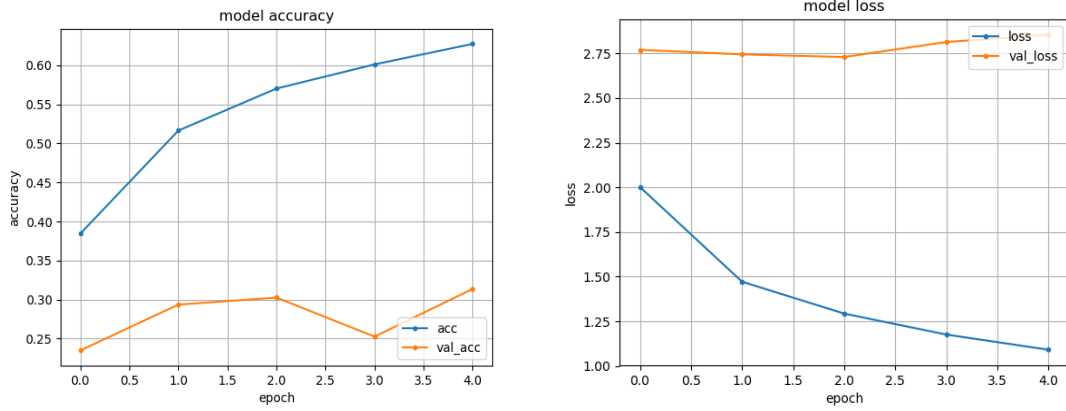
(a) The result of the model that trained with setting 1



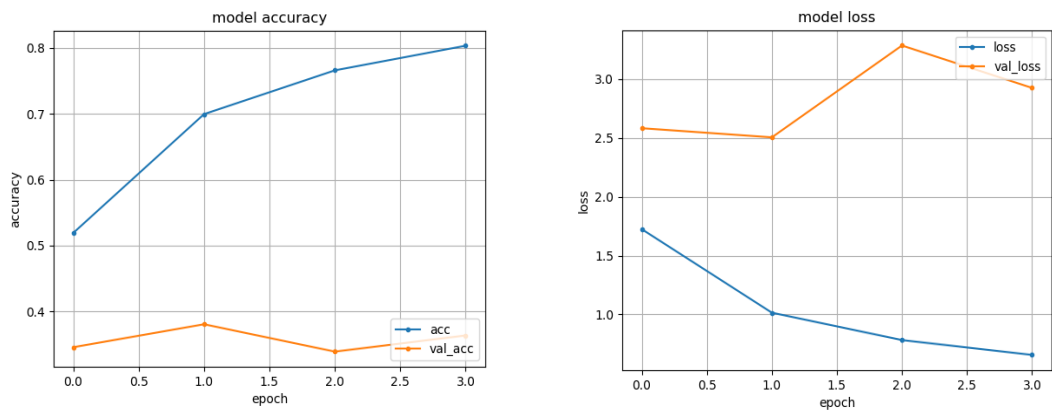
(b) The result of the model that trained with setting 2



(c) The result of the model that trained with setting 3



(d) The result of the model that trained with setting 4



(e) The result of the model that trained with setting 5

Fig. 5.4 The training results on the entire train-set and val-set

5.3.2 Training TC model

I train the TC model via minimize $\mathcal{L}_{classification} + \lambda \cdot \mathcal{L}_{similarity}$. The authors didn't provide any hyperparameters in their final journal version, but in the first version, they said the λ is 0.1.

In the first experiment, I used the same parameters as the original paper: I used Adam optimizer; λ is 0.1; learning rate is $1e-5$ for Conv1a to FC7 layers, $1e-4$ for FC8 layer. Each training sample consists of a start window and its follow-up window. The accuracy and loss are presented in Fig. 5.5

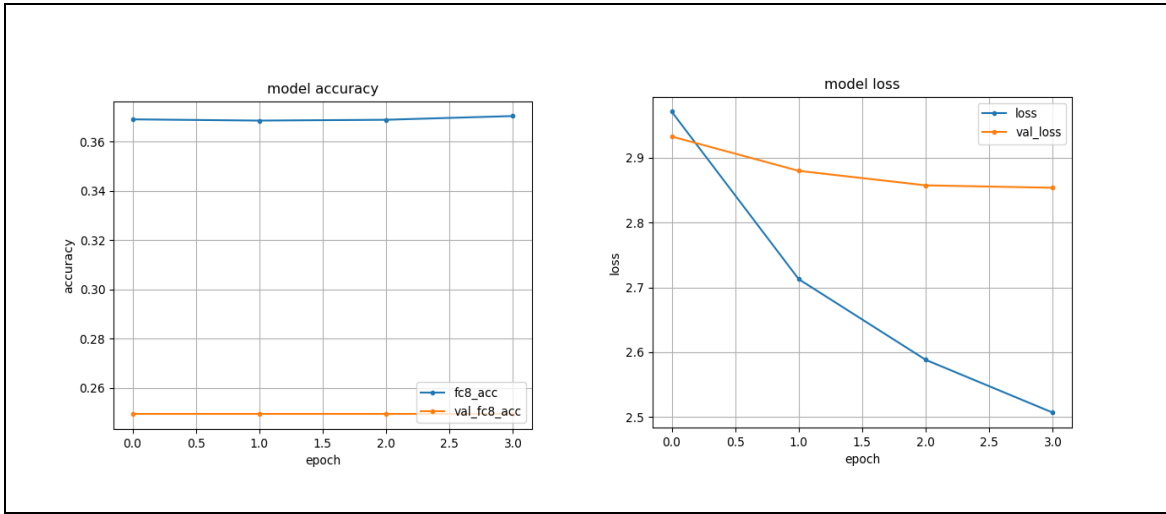


Fig. 5.5 Training result of the first experiment, temporal consistency strategy is used for all windows

From Fig. 5.5, we can see that the loss and val_loss decrease, but accuracies are not changed. On the test set, the model cannot detect anything, all windows are predicted as background.

The TC model takes two inputs: a start window and its follow-up window. The purpose of this model is to make the feature of AS windows close to its follow up window. However, in the first experiment, all training samples are followed by its follow-up window, i.e. non-action-start windows also followed by its follow up window. Thus, in the feature space, all windows will look similar. What we want is to make the AS windows close to the action window. So, I modified the paired training sample: only the AS windows are followed by its follow-up window, non-AS windows are paired with itself (the follow-up window cannot be empty because it's a Siamese model), i.e. I use temporal consistency only for AS windows.

By using the new paired samples, I trained the TC model again with $\lambda=0.1$. The result is better but still not satisfactory.

To obtain the best value of λ , I retrained the model with different value of λ : 0.01, 0.001 and 0.0001. The model has the best performance when λ is 0.001. The loss and accuracy of the train-set and val-set are shown in Fig. 5.6. As we can see, from the second epoch, the validation loss starts increasing and validation accuracy starts decreasing, the model suffers from an overfitting problem.

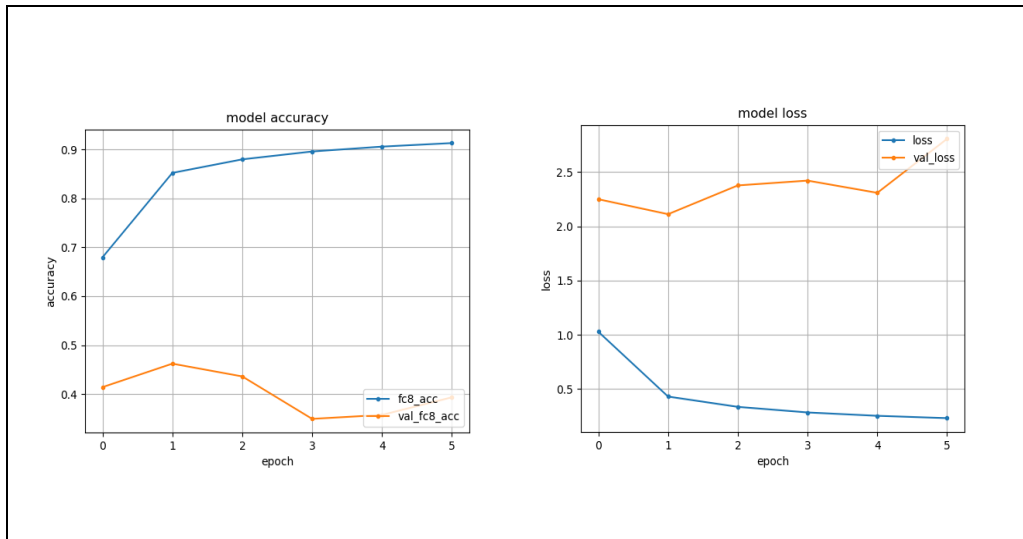
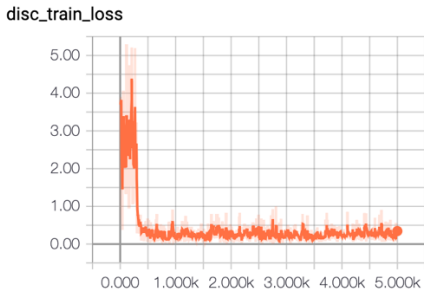
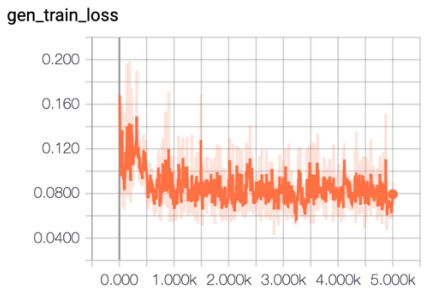
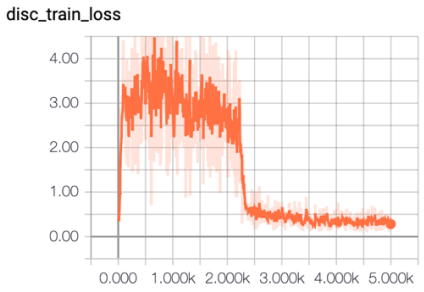
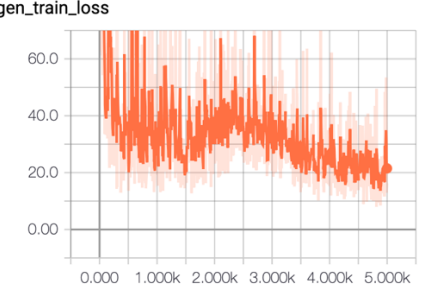


Fig. 5.6 training result of C3D + TC when λ is 0.001

5.3.3 Training GAN model

The GAN model is used to fine-tune the weights of the last three fully-connected layers. In the original paper, the learning rate for D is $1e - 5$ and for G is 0.003 (experiment 1 in Table 5.5), all other information is unknown. Table 5.5 listed the settings of 4 experiments. For all experiments, the GAN model is initialized by the same weights that pretrained by TC model.

experiment		D	G
1	Training batch	16 samples (8 fake windows, 4 AS windows, 2 A windows, 2 BG windows)	32 samples (16 fake windows, 16 AS windows)
	Learning rate	1e-5	0.003
	Loss		
2	Training batch	16 samples (8 fake windows, 4 AS windows, 2 A windows, 2 BG windows)	32 samples (16 fake windows, 16 AS windows)
	Learning rate	1e-6	0.0006
	Loss		

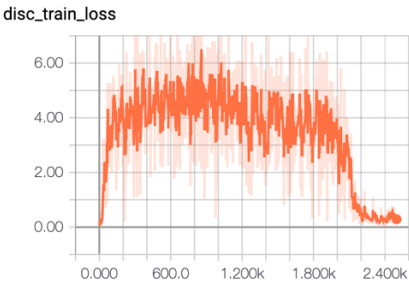
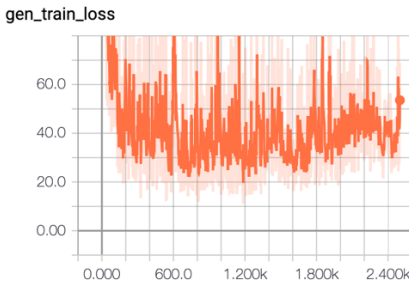
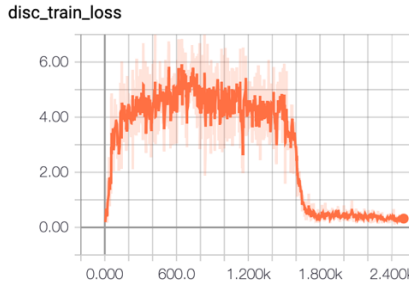
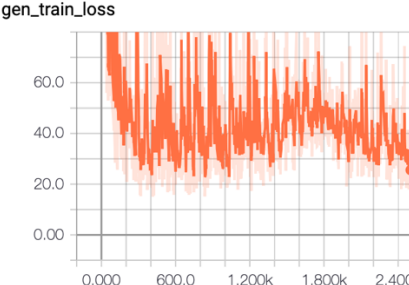
3	Training batch	16 samples (8 fake windows, 8 AS windows)	32 samples (16 fake windows, 16 AS windows)
	Learning rate	1e-6	0.0006
	Loss		
4	Training batch	32 samples (16 fake windows, 8 AS windows, 4 A windows, 4 BG windows)	32 samples (16 fake windows, 16 AS windows)
	Learning rate	1e-6	0.0006
	Loss		

Table 5.5 experiment settings

I use the weights that pre-trained by The TC model to initialize the GAN model and hence the baseline for the GAN model is the result of TC model.

In Table 5.5 , experiment 1, the model was trained 5K iterations; the result is worse than the baseline. Losses of D and G decreased too fast and hence I decreased the learning rate, trained the second time. In the second experiment, the model is trained 5K iteration and saved three weights (saved at the end of 1.5K iterations, 2.5K iterations and 5K iterations). Compare these three weights, the weights saved at the end of 1.5K iterations is the best one. As illustrated in Fig. 5.7, after 5K iterations, the model predicts an AS sample as a fake sample, which means the generator already has the ability to generate believable AS features (we want to use it to generate hard negative features, which is similar to AS features but still distinguishable) and the discriminator already trained by this kind of features which is labels as “fake”. To find a better setting, I trained the model with different training batches (Experiment 3 and 4). The result of experiment 4 (1.5K training iterations) is the best.



Fig. 5.7 An example of the prediction. The picture on the top is the output of the model that trained with 1.5K iterations. The bottom one is the output of the model with 5K iterations. The ground truth is CleanAndJerk.

Also, I trained three more times with the same parameters as experiment 2 to check the stability; the results prove that its stable.

5.4 Results

Following the original paper [4], the mAP is evaluated at different temporal offset thresholds. Since the duration of most actions in THUMOS'14 is 1 to 20 seconds, the offset is set to 1 to 10 seconds.

5.4.1 Results of C3D model

Fig. 5.8 are the results of C3D model tested on the testing set. For result (1), the optimizer is Adam, learning rate for Conv1a to FC7 is $1e-5$, learning rate for the last layer is $1e-4$, 1 training epoch (5017 iterations); for result (2), l2 kernel_regularizer used; is for result (3), first 5 convolution layers (Conv1a to Conv4a) are fixed, learning rate for Conv4b to FC7 is $1e-5$, learning rate for FC8 is $1e-4$, 2 training epochs.

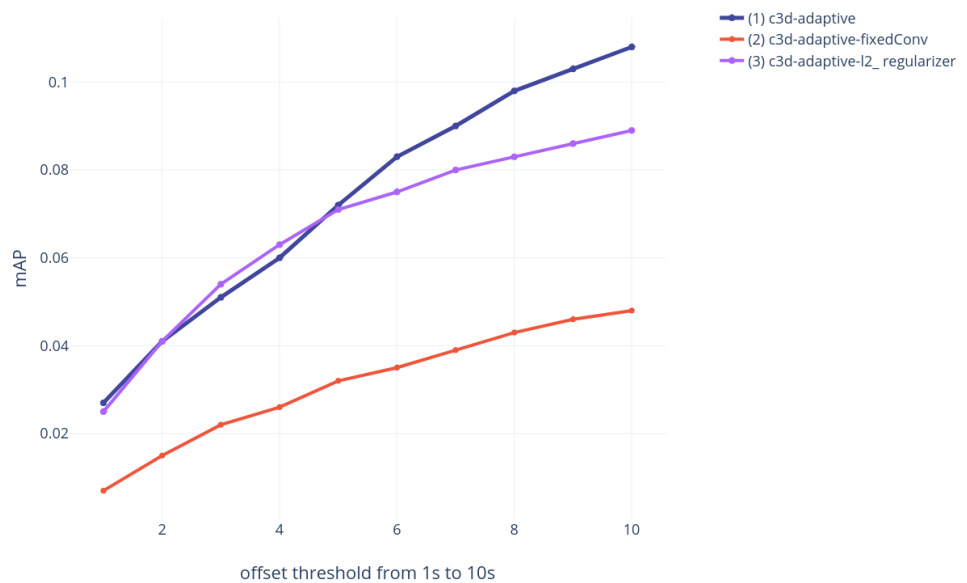


Fig. 5.8 Results of C3D model when adaptive sampling strategy is used

5.4.2 Results of TC model

The TC model was initialized by the weight pre-trained on sports-1M dataset. The results are presented in Fig. 5.9.

For the result (1), (2), (3) and (4) in Fig. 5.9, the model was trained with the same hyperparameters except for λ : optimizer is Adam, and the learning rate is $1e-5$. For result (5), the first five convolution layer (Conv1a to Conv4a) were fixed during training; λ is set to 0.001 since the model has the best performance when $\lambda = 0.001$.

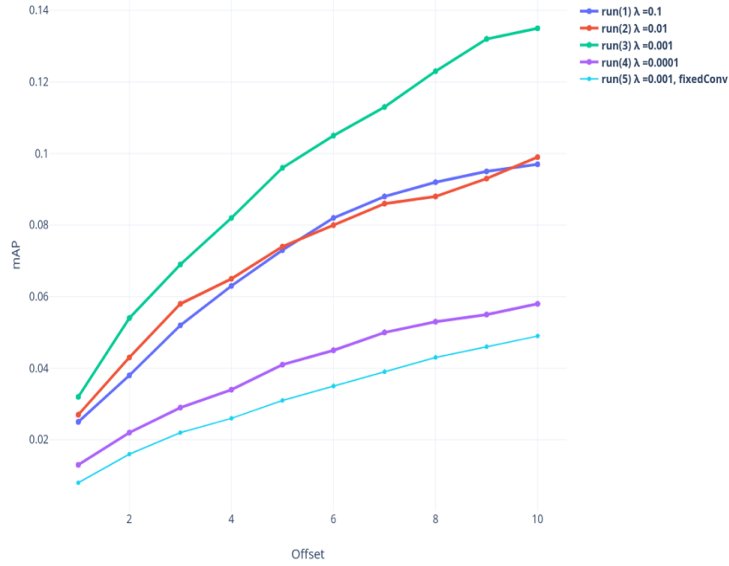


Fig. 5.9 Results of the model when adaptive sampling and temporal consistency are used

5.4.3 Results of the model when a GAN is used for data augmentation

Fig. 5.10 shows the mAP of the model trained with different settings. Corresponding settings of results (1), (2), (3) and (4) are listed in Table 5.5 experiment 1, 2, 3 and 4.

To check the stability, I retrained multiple times with the same parameter as run (4) since it has the best performance. The results are presented in Fig. 5.11. The baseline in Fig. 5.11 is the result of the TC model (Fig. 5.9 result (4)).

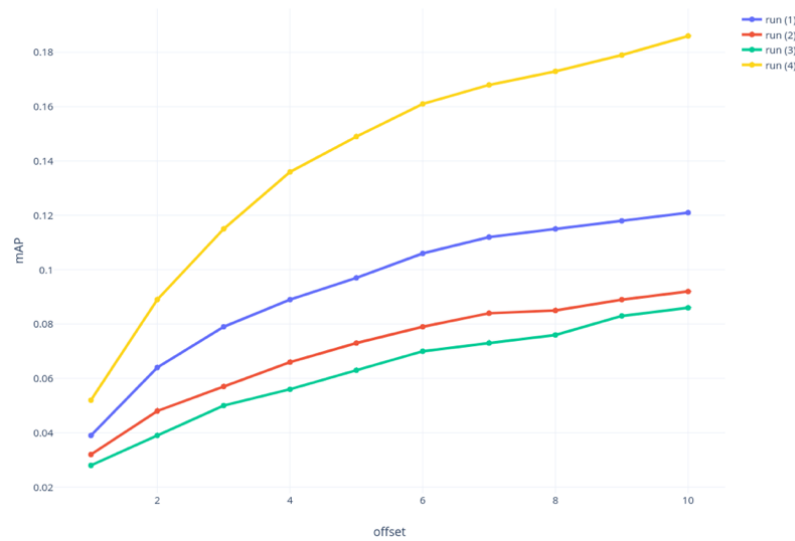


Fig. 5.10 Results of the model when all methods are used

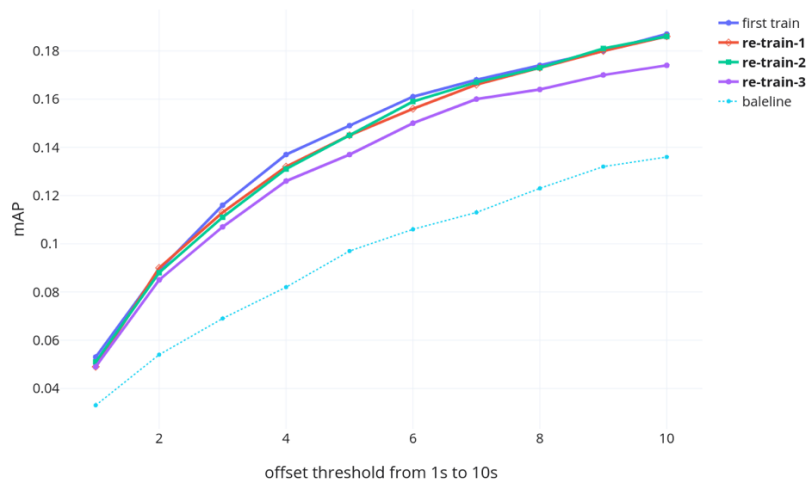


Fig. 5.11 Results of model trained with the samce parameters

5.4.4 Comparison of results

Fig. 5.12 shows the results of the three models, Fig. 5.13 shows the comparison of my results and the results obtained from the original paper. **Adaptive** means the model was trained with the adaptive sampling strategy, **Adaptive-TC** means the model was trained with the adaptive sampling and temporal consistency strategy, as while **Adaptive-TC-GAN** means the model was trained with all three methods.

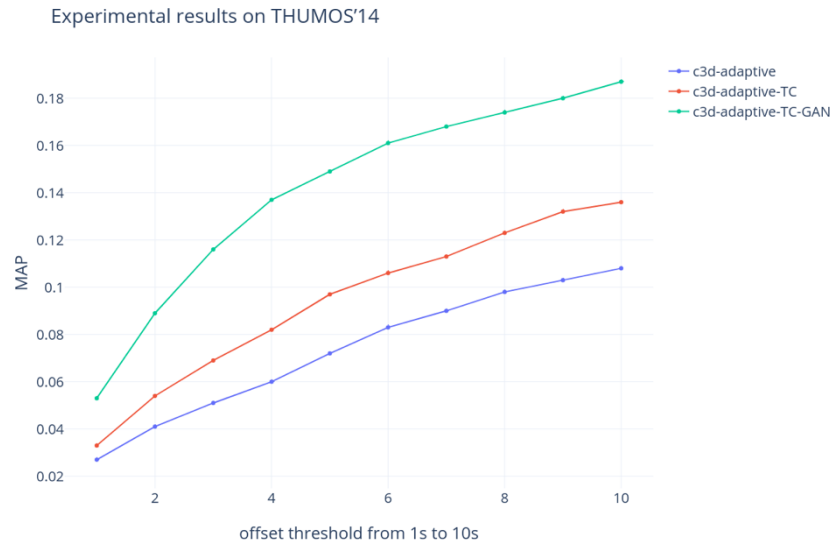


Fig. 5.12 THUMOS'14 results

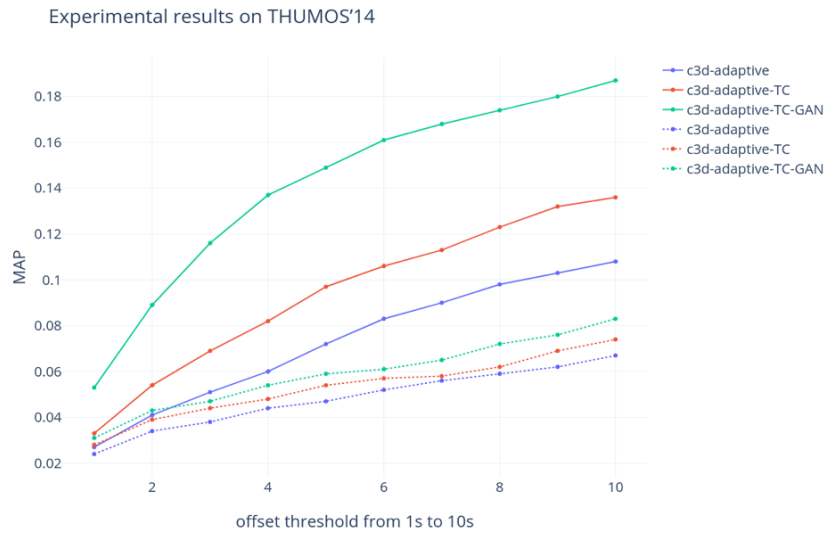


Fig. 5.13 THUMOS'14 results. The dotted lines are the results in the original paper

Note that the results of the original paper are approximate values that obtained from a plot. As we can see from Fig. 5.13, the results are quite different from the original paper's results; the potential causes include: (1) during training, each training batch consists of AS samples, action samples and background samples in the proportion 2:1:1 instead of half AS samples, half non-AS samples; (2) the authors didn't release the evaluation code, perhaps some details of my implementation were different from the original evaluation system.

Based on the point-level metric, I implemented two different protocols:

1. The prediction satisfies the follow conditions (the results are evaluated by this protocol, unless stated otherwise):
 - a. an action is detected, and the predicted class is correct.
 - b. The distance between the detected AS and ground truth is smaller than corresponding offset threshold.
2. The prediction satisfies previous conditions and:
 - c. The detected AS should not later than the end of action, otherwise, it will be counted as false positive.

The results with the second protocol are presented in Fig. 5.1 (dotted line).

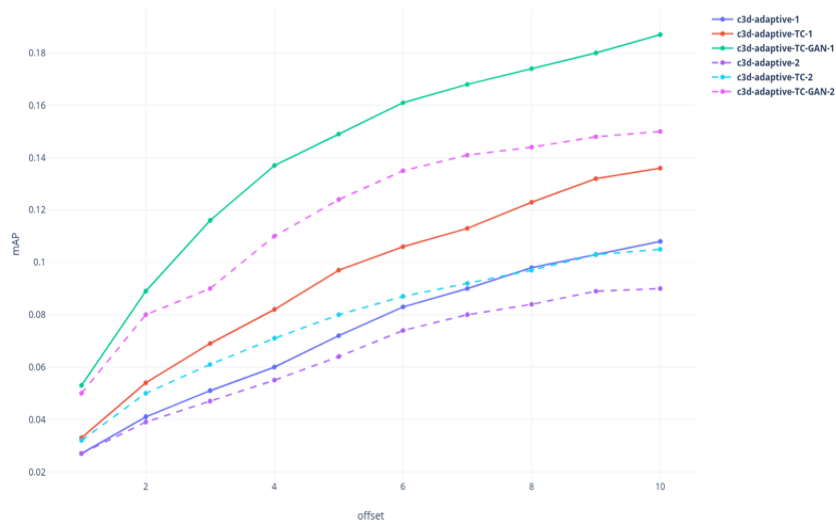


Fig. 5.14 Compare the results of two evaluation protocols.

5.5 Examples of prediction

Example 1.

Fig. 5.15 is a clip of a video from the fifth second to ninth second, the ground truth of AS is marked with a green border. As we can see, the output of C3D is a background. This is because the AS window is a mixture of action and background; the model cannot distinguish it from the background correctly. The TC model was trained to move the features of AS close to the feature of A, the backgrounds are distinguishable and hence gets better performance. The GAN is trained to distinguish hard negatives such as the output of TC, and thus the GAN correctly detected the action but have a 0.7s delay.

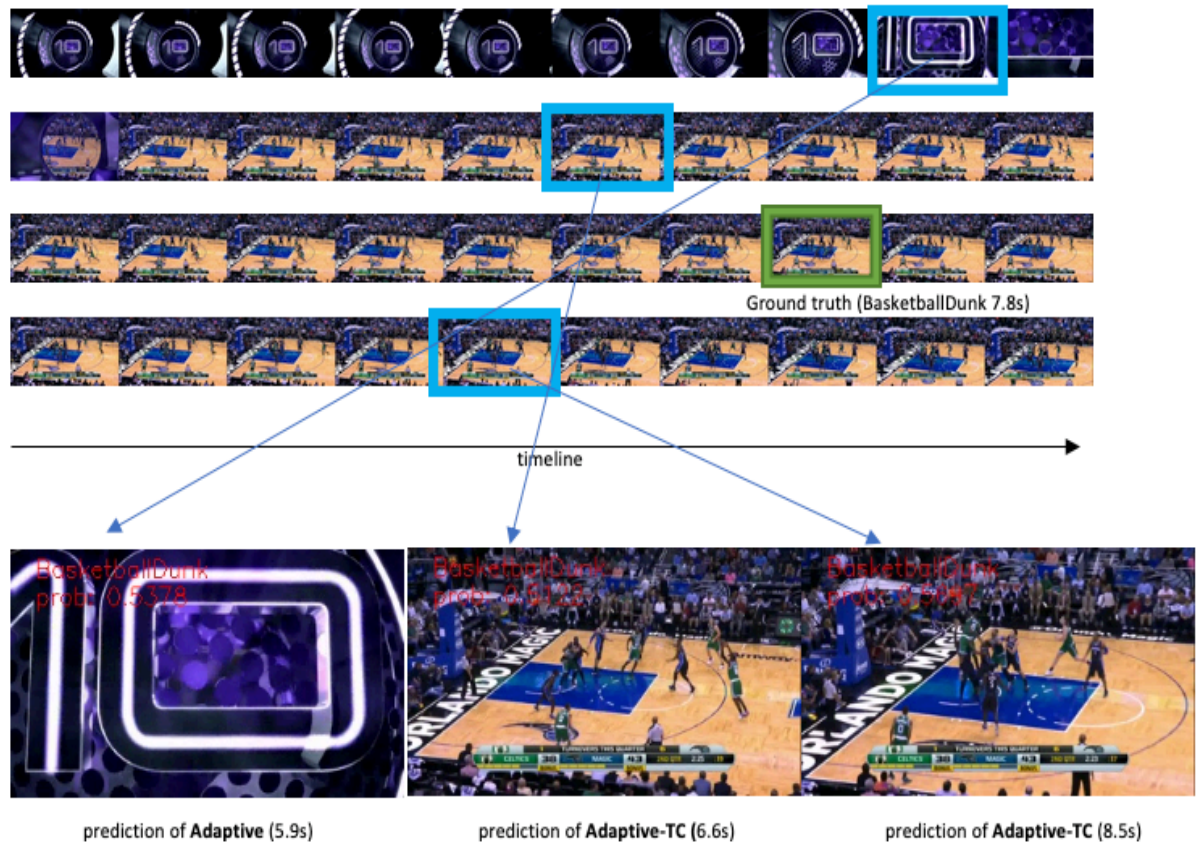


Fig. 5.15 An example of prediction

Example 2.

In Fig. 5.16, all models correctly detected an action start window (marked with blue borders). **Adaptive** outputs a wrong prediction since the class of window (b) and window (a) are different, and its confidence score is greater than the threshold (0.5).

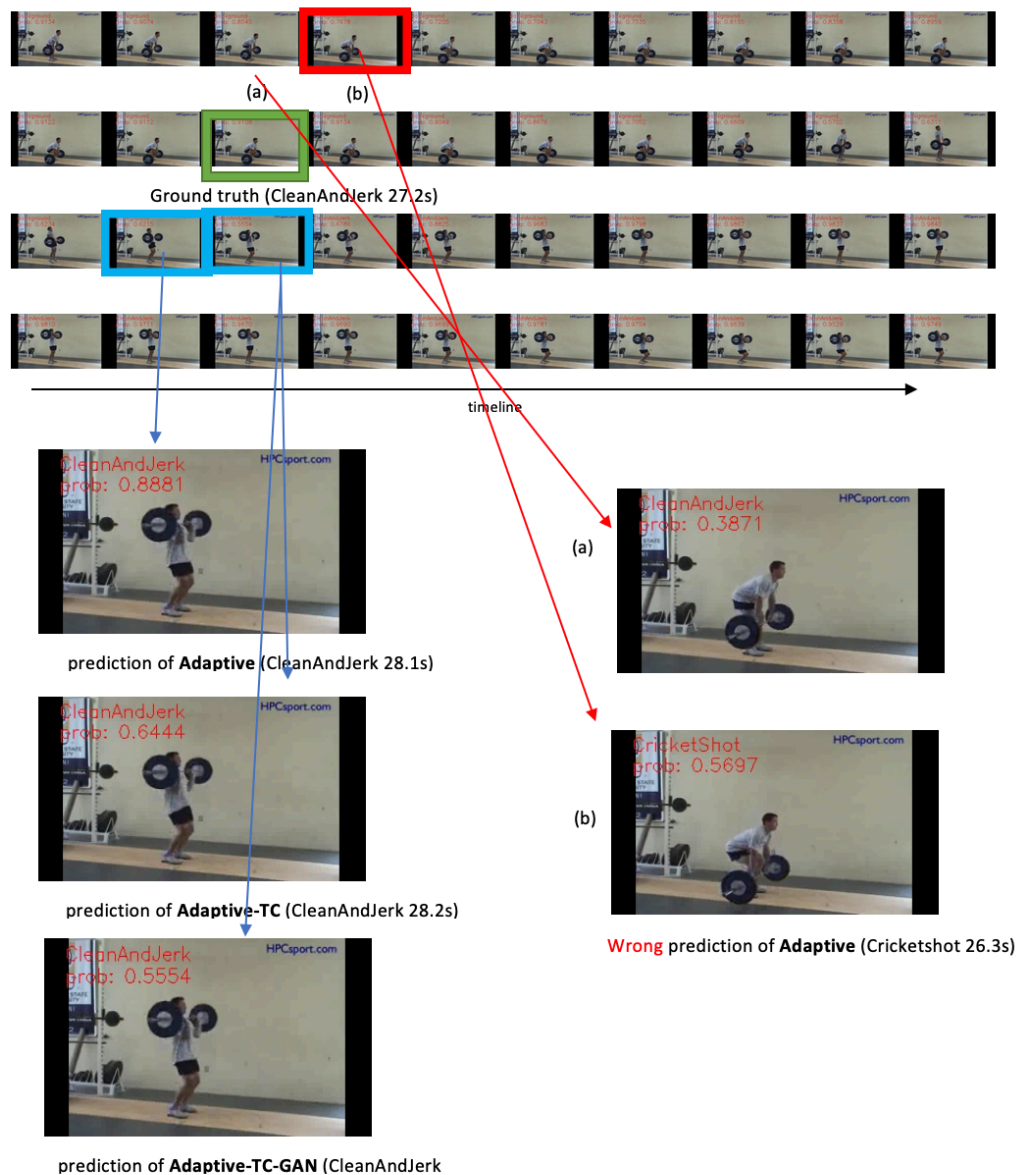


Fig. 5.16 An example of prediction

Example 3.

Another example is presented in Fig. 5.17. There are no correct predictions. The window with red border is predicted as BaseballPitch by **Adaptive**, but no AS point detected since its score is less than the threshold (0.5).



Fig. 5.17 An example of prediction

6 Experiments on a subset of UCF-Crime

6.1 UCF-crime dataset

UCF-crime dataset [30] is a video anomaly detection dataset. It consists of 1900 surveillance videos, with 13 anomaly actions. Examples of anomalies are shown in Fig. 6.1.

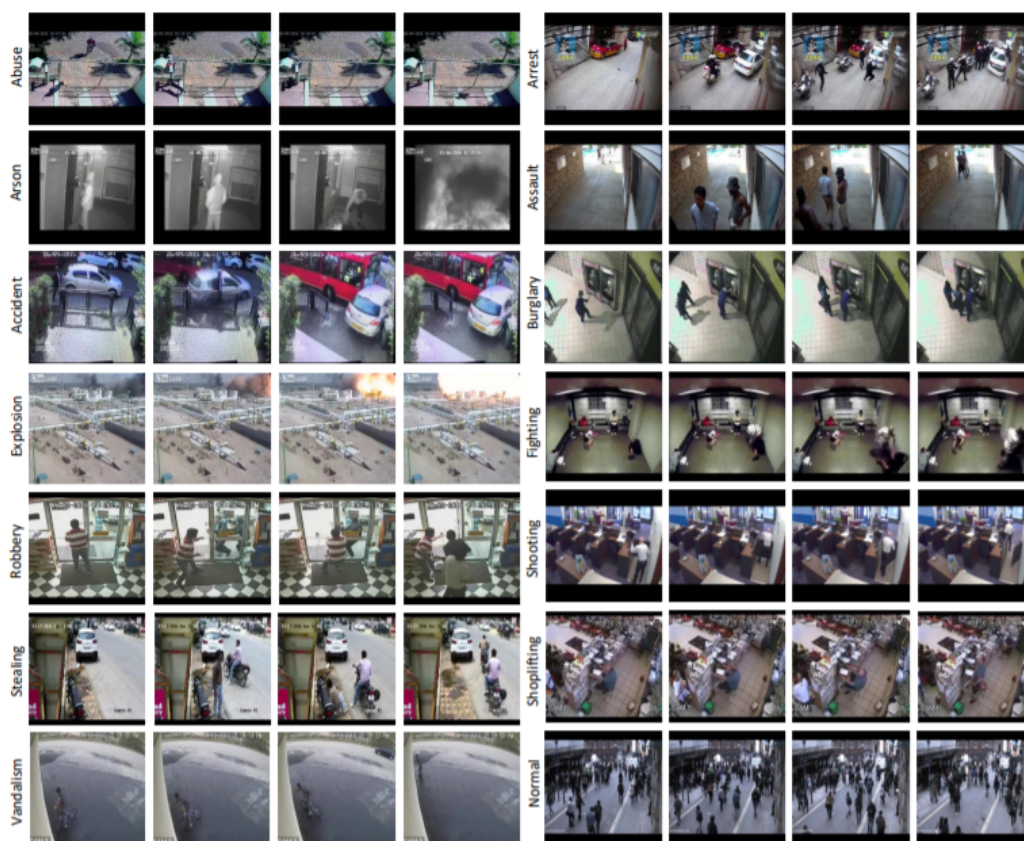


Fig. 6.1 Examples of different anomalies from UCF-crime dataset [30]

For simplify, I extracted 150 road accident videos and constructed a sub-dataset, RoadAccident dataset.

The RoadAccident dataset is split into training and testing set (the train-test split is the same as the original dataset):

Train-set: 127 untrimmed surveillance extracted from UCF-crime training data, each video contains one or two traffic collisions. Since there are only video-level annotations, I labelled the temporal extent of each accident.

Test-set: the test-set contains 23 untrimmed surveillance videos; each video contains one or two traffic collisions. Temporal annotations are provided by authors, but there is an obvious mistake: video *RoadAccidents022_x264.mp4* contains two accidents, but they only labelled one.

During training and testing, all frames are resized to 120×160 since it's half resolution of the videos. The statistics of train-set and test-set are listed in Table 6.1.

	Train-set	Test-set
Number of videos	127	23
Number of accident instances	136	24
Number of AS samples	2159	384
Number of A samples	12570	1893
Number of BG samples	213452	23218

Table 6.1 Statistics of train-set and test-set

6.2 Training phase

Since the size of the training sample is 120×160 , the architectures of the networks are slightly different: the input of C3D network is $16 \times 120 \times 160 \times 3$ instead of $16 \times 112 \times 112 \times 3$; the number of nodes for fully-connected layers in G is 10240 because pool5 output has 10240 dimensions. Since the dataset is small, during training, I directly use test-set as the validation set.

6.2.1 Training C3D model

Experiment 1.

Same as the previous experiments on THUMOS'14 dataset, each training batch consists of 8 AS samples, 4 A samples and 4 BG samples. Learning rate is $1e - 5$ except $1e - 4$ for the last layer. The model is initialized by the weights pre-trained on sport-1M.

The accuracy and loss are presented in Fig. 6.2.

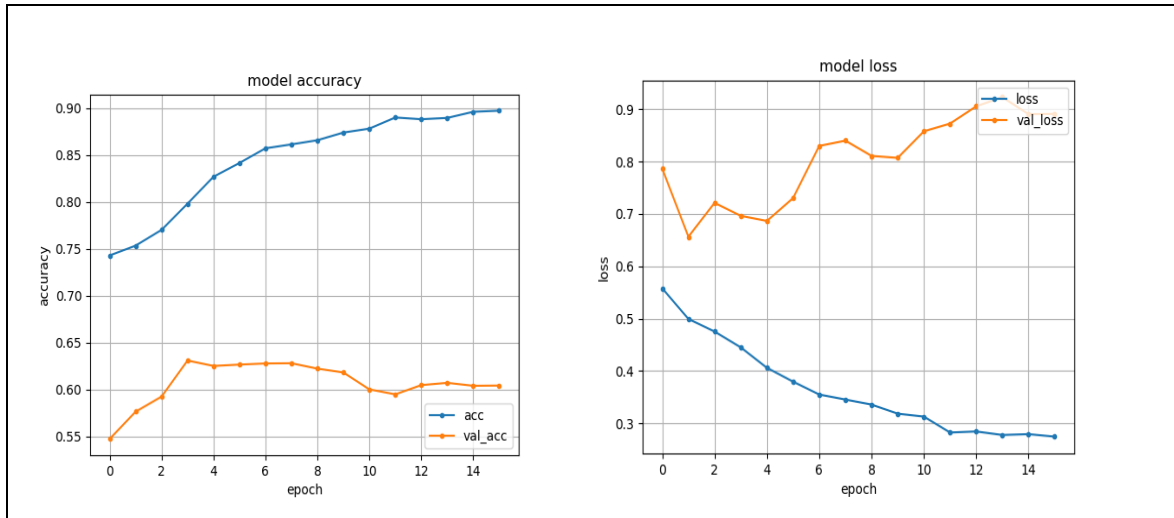


Fig. 6.2 Accuracies and losses

Experiment 2.

There are only 2159 AS samples in the train-set, it's difficult to train the model with such a small amount of data. In the second experiment, I pre-trained the model with A and

BG samples. More specifically, each training sample consists of 8 A samples and 8 BG samples, and the model was pre-trained 4 epochs. With such initialization, I fine-tune the weights of the fully-connected layers, i.e. convolution layers are not learnable; each training batch consists of AS samples, A samples and BG samples in the proportion 2:1:1. The losses and accuracies are presented in Fig. 6.3

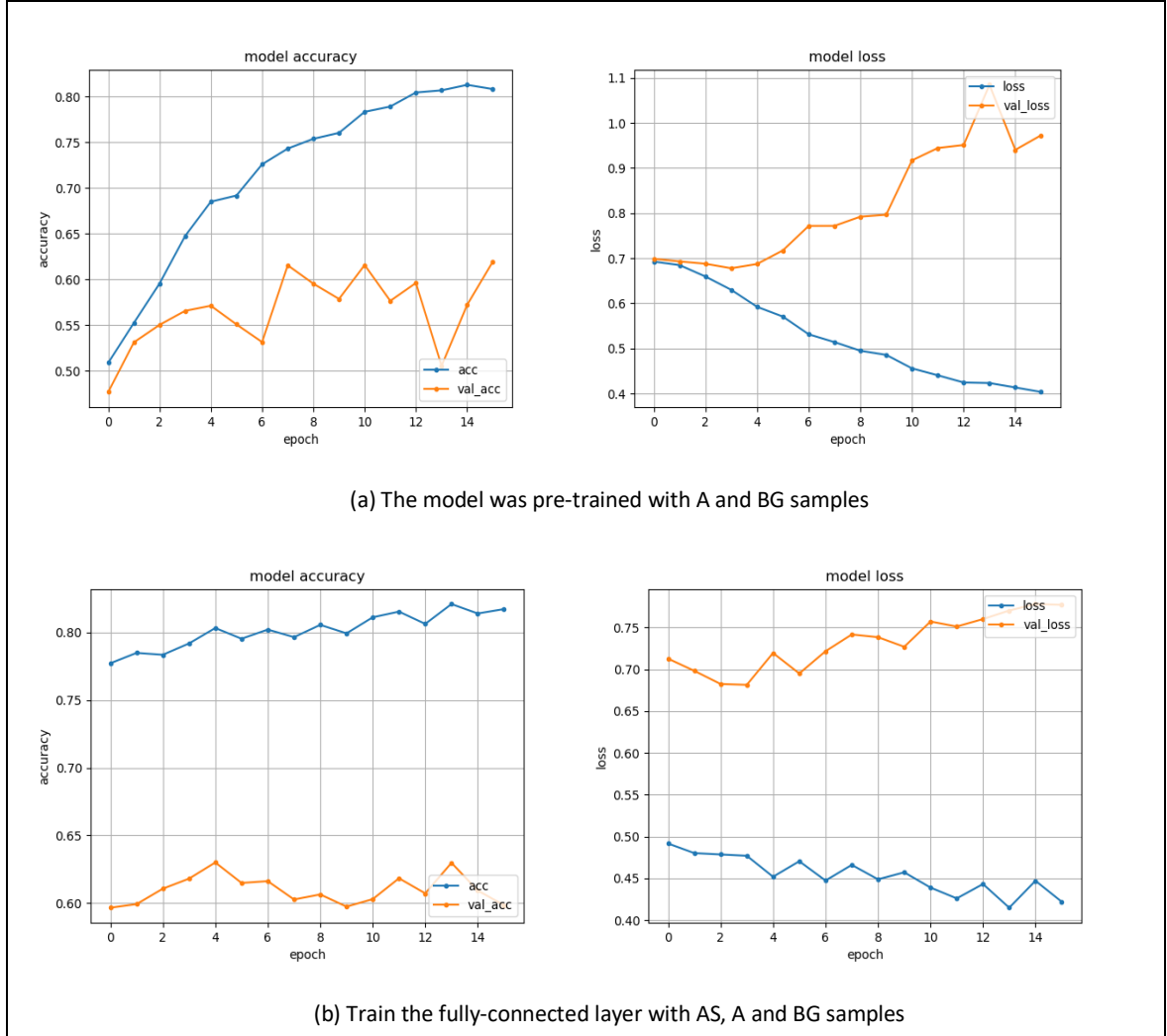


Fig. 6.3 Accuracies and losses

6.2.2 Training TC model

TC model was initialized by the weights pre-trained on the A and BG data and trained with the following settings: optimizer is Adam; learning rate is rate is $1e - 5$ for FC6 and

FC7, $1e - 4$ for the last layer (convolution layers are not learnable); λ is 0.001. The losses and accuracies are presented in Fig. 6.4

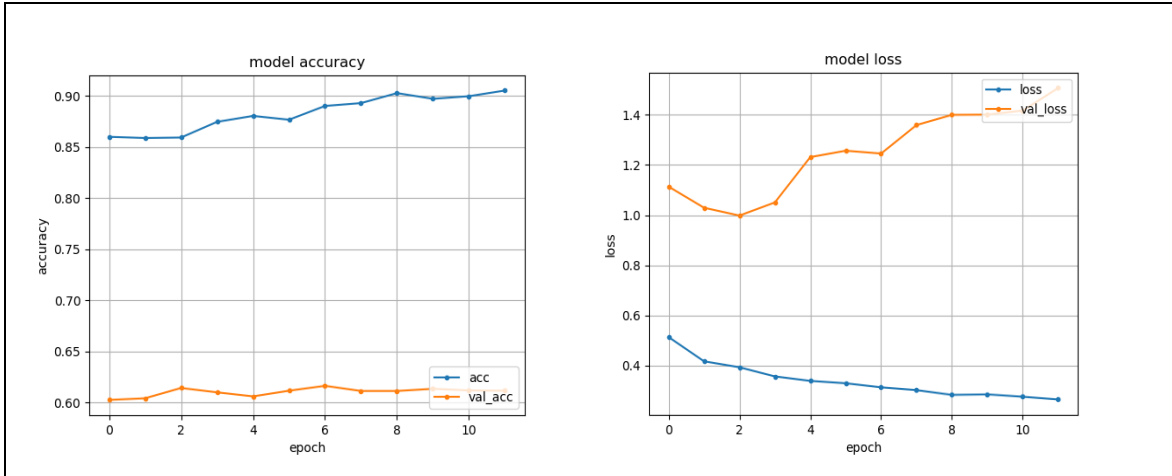


Fig. 6.4 Accuracies and losses

6.2.3 Training GAN model

The discriminator was initialized by the weights pre-trained by TC model, the generator was randomly initialized. The loss, accuracy and settings are listed in Table 6.2. Testing the C3D model on test-set, the model has the best performance when training iteration is 180 (compared with 150, 200, 250, 500 and 1000 training iterations).

	D	G
Training batch	32 samples (16 fake windows, 8 AS windows, 4 A windows, 4 BG windows)	32 samples (16 fake windows, 16 AS windows)
Learning rate	1e-6	0.0006

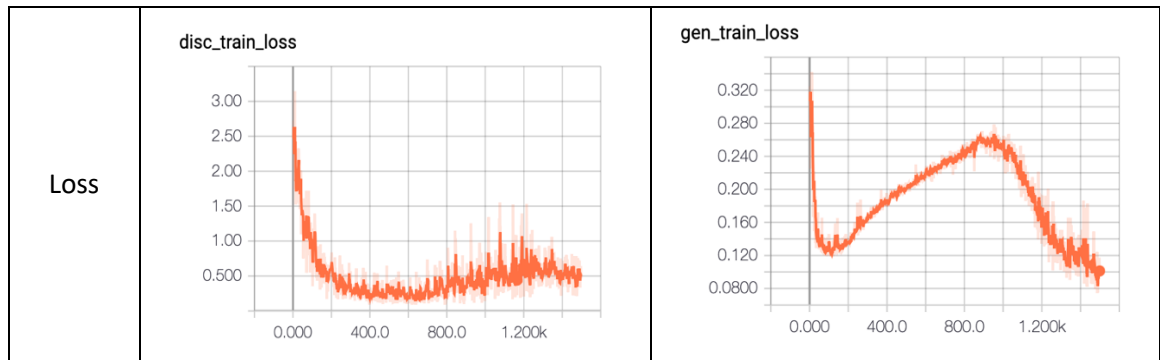


Table 6.2 Settings of the GAN model

6.3 Results

Since the duration of an accident varies from 1s to 5 seconds, the offset is set to 0.2 to 2 seconds.

6.3.1 Results of C3D model

The results are presented in Fig. 6.5, solid line represents the result of the model that pre-trained on A and BG data and fine-tuned on the whole train-set. Dotted line represents the result of the model directly trained on the train-set.

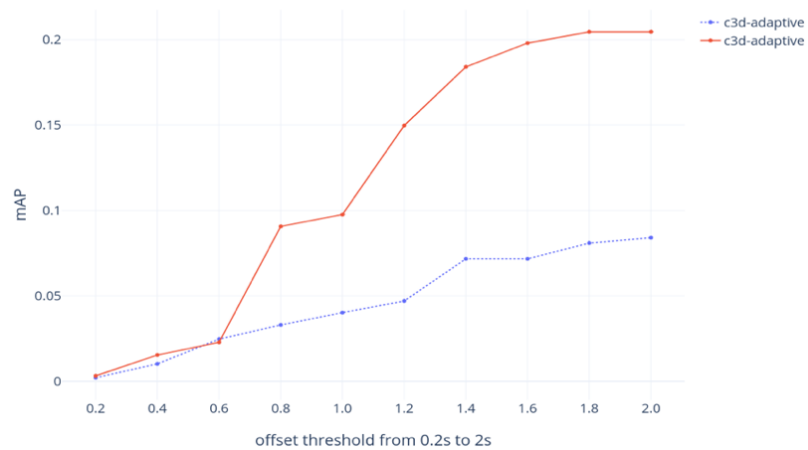


Fig. 6.5 Results of C3D model when adaptive sampling strategy is used.

6.3.2 Results of TC model

Fig. 6.6 shows the result of C3D model when adaptive sampling and temporal consistency strategies are used during training.

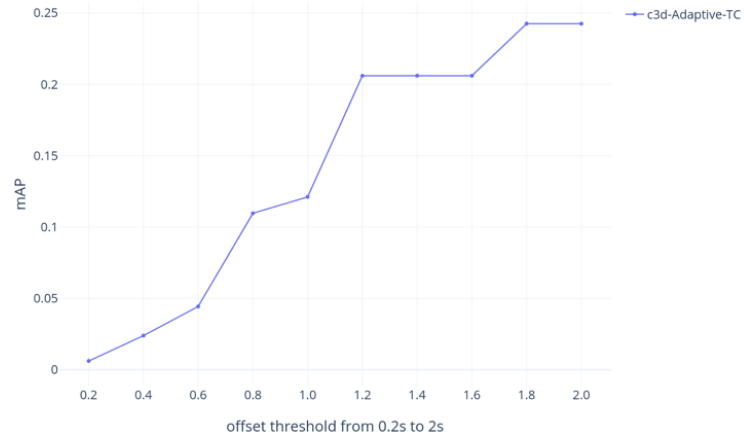


Fig. 6.6 The result of C3D model when adaptive sampling and temporal consistency strategies are used.

6.3.3 Result of GAN

Fig. 6.7 shows the result of C3D model when all training methods are used during training.

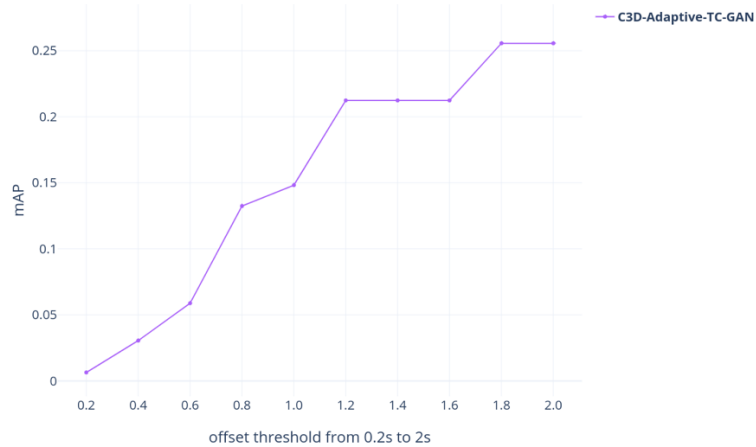


Fig. 6.7 Results of the model when all methods are used

6.3.4 Comparison of results

All results are presented in Fig. 6.8. As we can see, c3d-Adaptive-TC outperforms c3d-Adaptive; when a GAN is used for data augmentation, the model achieves the best performances.

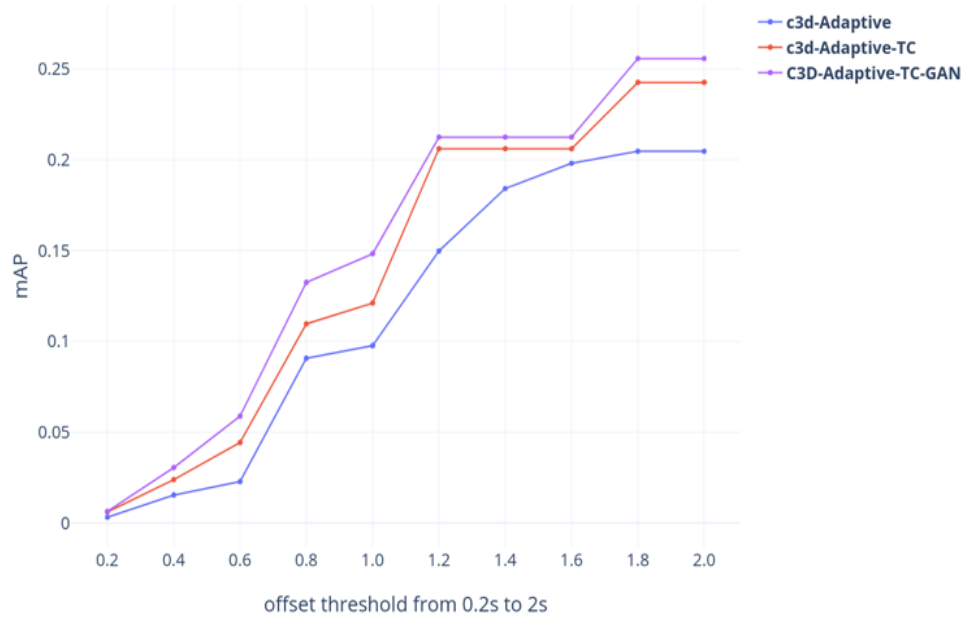


Fig. 6.8 Compare the results

6.4 Examples of prediction

Example 1.

In Fig. 6.9, all models worked well, the accident correctly detected.

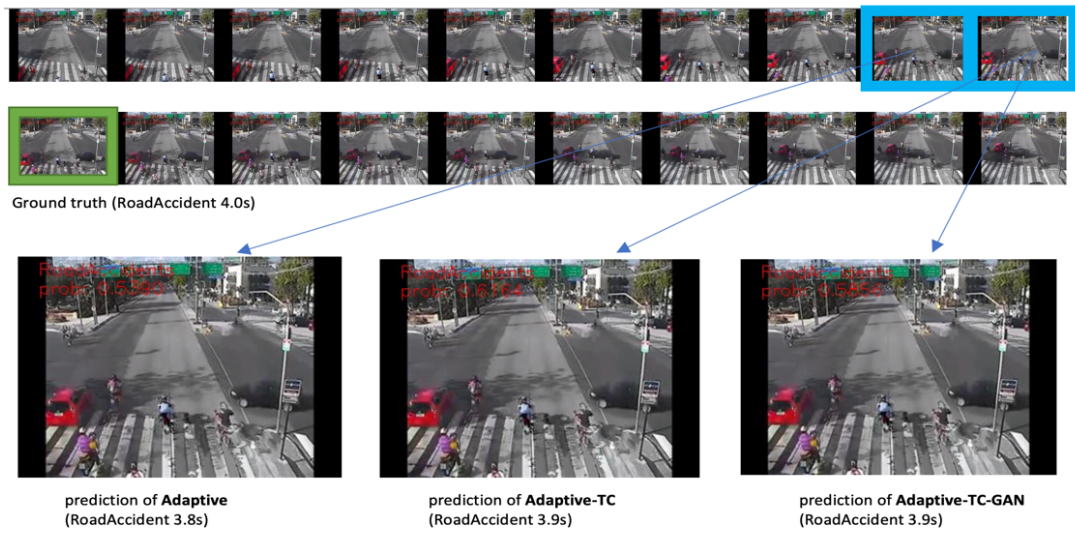


Fig. 6.9 Examples of prediction

Example 2.

In Fig. 6.10 , a motorcycle Crashed into a Car, but all model failed, the collision was not detected.



Fig. 6.10 Examples of prediction

7 Conclusions

This thesis reproduces the work of online detection [4]. The aim of this thesis is to review and analyse different human action detection framework based on deep learning, implement a neural network that able to detect actions in videos as soon as possible. More specifically, this thesis is summarized in the following points.

Firstly, I analysed the online action detection setting and reviewed its related works and literature. one part of this thesis is targeted at the human action start detection problem. To obtain efficient and effective action start detection, I implemented three methods that proposed by Zheng et al. [4]: (1) use adaptive sampling to handle the scarcity action start samples problem; (2) model the temporal consistency to make the features of action start windows close to the actual action; (3) fine-tuning the model via Generative adversarial network.

Secondly, I trained three models and performed extensive experiments to obtain the appropriate hyperparameters. Unlike normal GANs, the GAN network in this thesis is used to generate hard negative samples that distinguishable from backgrounds and actual actions, fine-tuning the weights of C3D network. During testing, only the backbone network (C3D) is used.

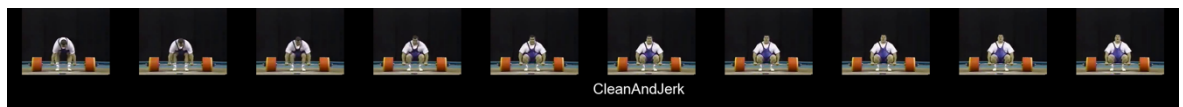
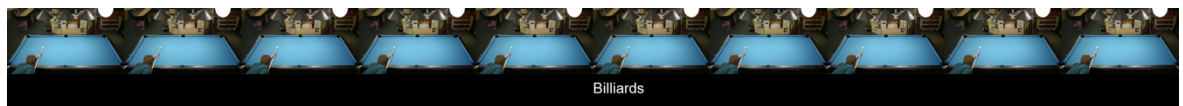
To summarize, this thesis had identified and addressed the problems in online action detection and implemented the state-of-the-art article. It is proved that the three novel methods can improve the performance of the network. When temporal consistency strategy is used, action start windows are more distinguishable from background windows. Therefore, the model can detect the action start windows more accurately and timely. When a GAN is used for data augmentation, the model could see more negative data and hence false positives are reduced. On the RoadAccident dataset, I tried a different strategy: pre-train the model with action and background windows, i.e. learn easier things first; then fine-tune the weights of last three fully-connected layers with Zheng et al.[4] proposed methods. It outperforms the model trained directly with these three methods.

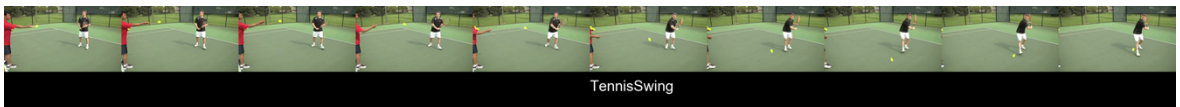
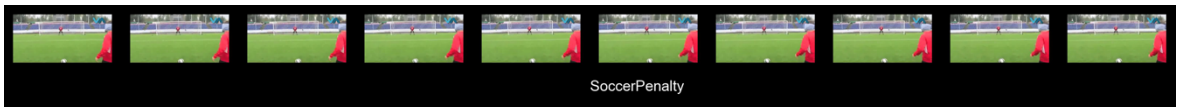
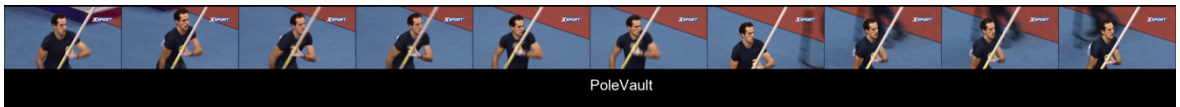
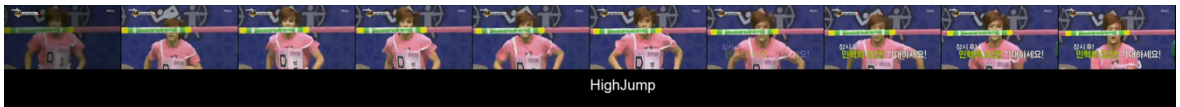
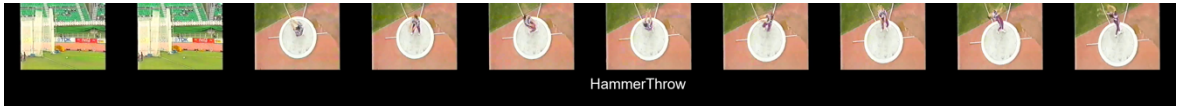
Although these methods can improve performance of the networks, they still have disadvantages: during training, temporal annotations are required; it can only detect a point level action; the performance has a large room for improvement.

Different from action detection, action prediction needs to predict the action when action is still on-going or even not start yet. This is an interesting and important task. For example, the surveillance monitoring system needs to detect anomaly actions as soon as possible and then issue an alert to allow timely security response. If we can predict the anomaly actions, we may have more time to handle it. In future work, online action anticipation on the untrimmed surveillance videos would be of great interest to the research industry and community.

Appendix A

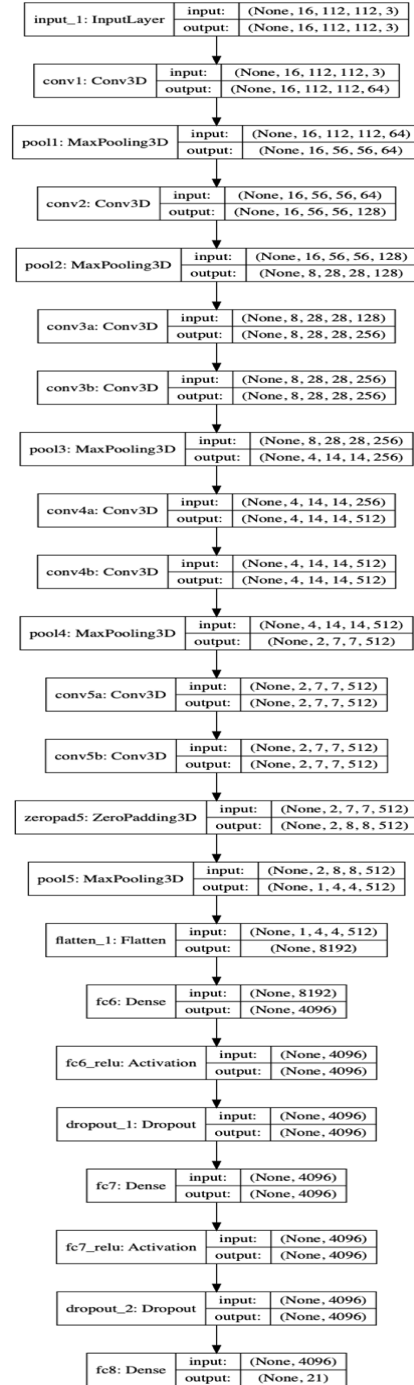
Examples of actions from THUMOS'14 dataset:



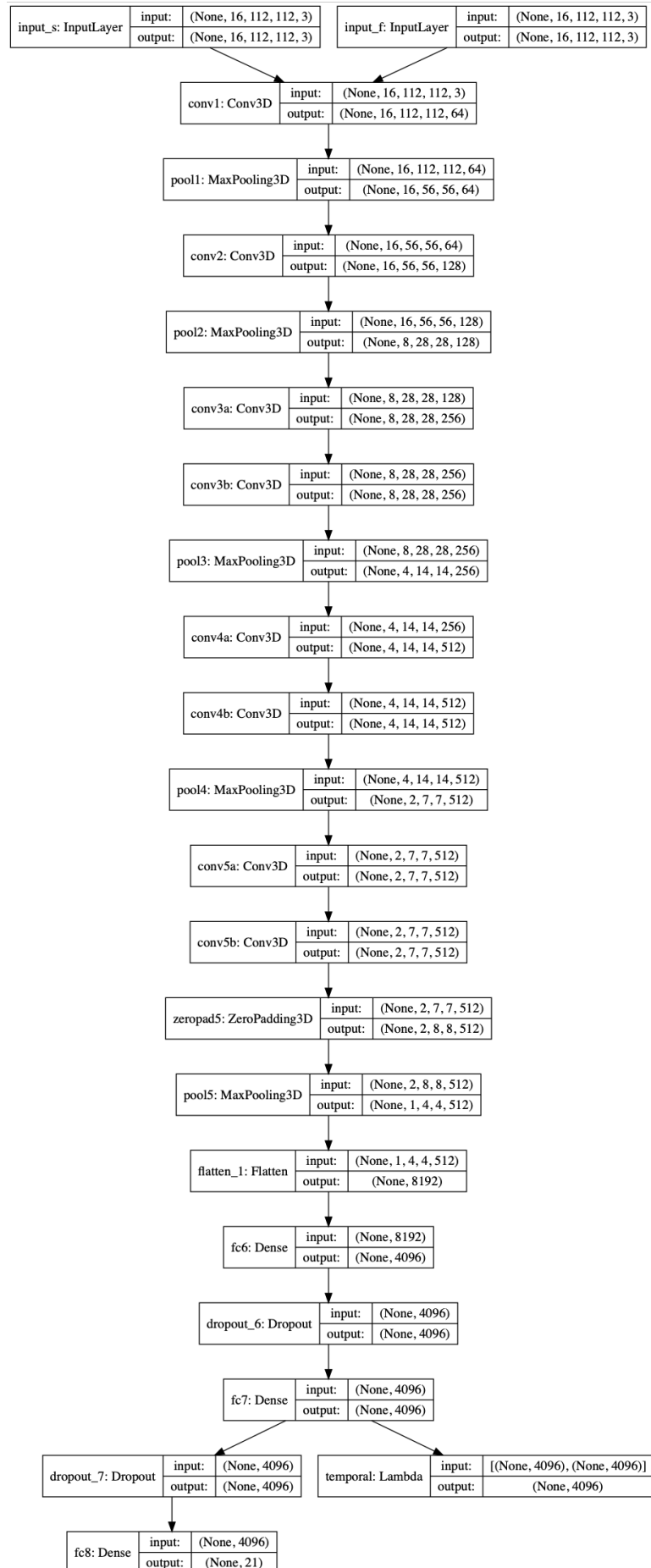


Appendix B

Architecture of C3D model:

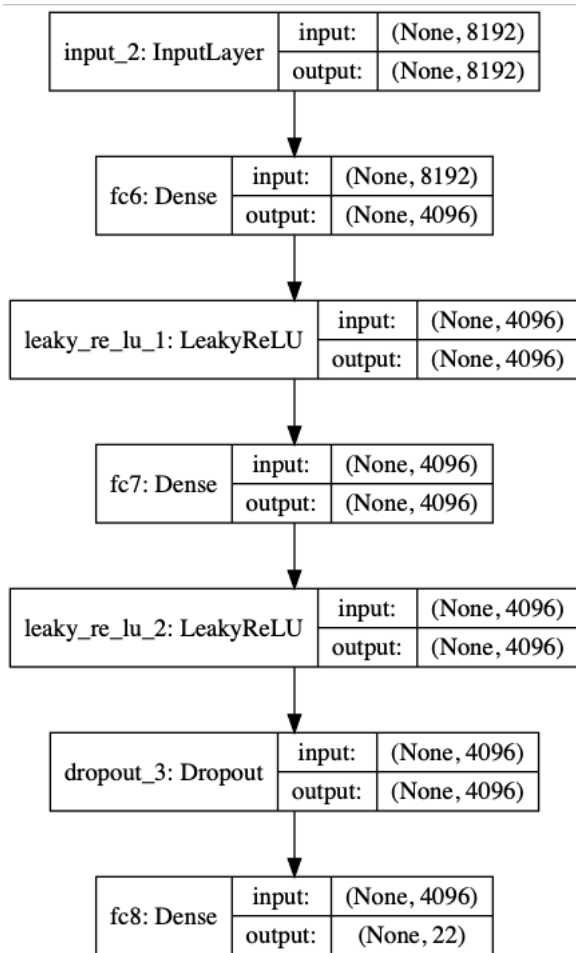


Architecture of TC model:

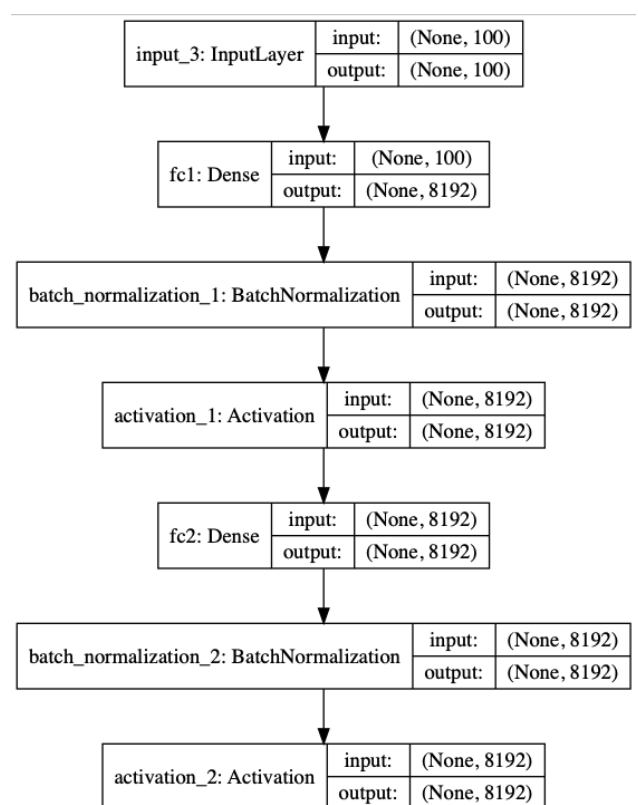


Architecture of GAN:

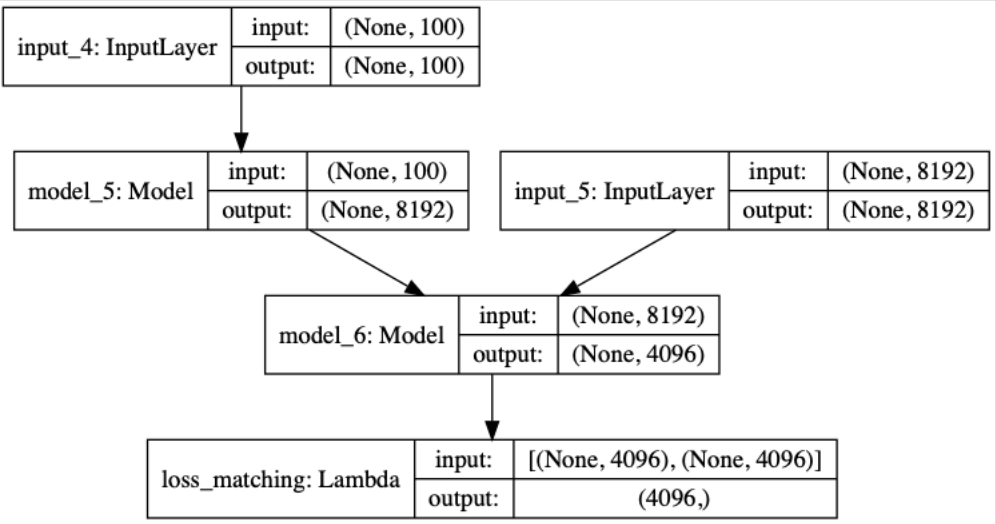
Discriminator:



generator:



GAN:



Bibliography

- [1] H. Wang and S. Cordelia , “Action recognition with improved trajectories.,” Proceedings of the IEEE international conference on computer vision, 2013.
- [2] A. Gaidon, Z. Harchaoui and C. Schmid, “Actom sequence models for efficient action detection,” CVPR . IEEE, 2011.
- [3] R. De Geest, “Online action detection,” European Conference on Computer Vision, pp. 269-284., 2016.
- [4] S. Zheng, P. Junting, C. Jonathan, M. Kazuyuki, M. Hassan, V. Anthony, G.-i.-N. Xavier and C. Shih-Fu, “Online Detection of Action Start in Untrimmed, Streaming Videos,” 2018.
- [5] J. Gao, Z. Yang, C. Sun, K. Chen and R. Nevatia, “Temporal unit regression network for temporal action proposals.,” ICCV, 2017.
- [6] J. Gao, Z. Yang and R. Nevatia, “Red: Reinforced encoder-decoder networks for action anticipation,” BMVC , 2017.
- [7] D. Tran, L. Bourdev, R. Fergus, L. Torresani and M. Paluri, “Learning Spatiotemporal Features with 3D Convolutional Networks,” ICCV, 2015.
- [8] “wikipedia,” [Online]. Available: https://it.wikipedia.org/wiki/Rete_neurale_artificiale. [Accessed 2019].
- [9] J. Carreira and A. Zisserman, “Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset,” CVPR, 2017.
- [10] A. Karpathy and J. Johnson, "CS231n Convolutional Neural Networks for Visual Recognition," [Online]. Available: <http://cs231n.github.io/convolutional-networks/#pool>. [Accessed 3 2 2019].
- [11] MathWorks, “Convolutional Neural Network,” [Online]. [Accessed 20 1 2019].

- [12] S. Ji, W. Xu, M. Yang and K. Yu, "3d convolutional neural networks for human action recognition," IEEE transactions on pattern analysis and machine intelligence, 2013.
- [13] I. J. Goodfellow, J. Pouget-Abadie, B. X. D. W.-F. Mehdi Mirza, h. Ozair, A. Courville and Y. Bengio, "Generative Adversarial Nets," Advances in neural information processing systems, 2014.
- [14] F. Chollet, Deep Learning with Python, Manning Publications Company, 2017.
- [15] L. Torrey and J. Shavlik, Transfer Learning, University of Wisconsin, USA, 2010.
- [16] "CS231n Convolutional Neural Networks for Visual Recognition," [Online]. Available: <http://cs231n.github.io/transfer-learning/>. [Accessed 2 2019].
- [17] "Precision and recall," [Online]. Available: https://en.wikipedia.org/wiki/Precision_and_recall#Precision.
- [18] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar and L. Fei-Fei, "Large-scale Video Classification with Convolutional Neural Networks," Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, 2014.
- [19] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," NIPS, 2014.
- [20] Z. Shou, D. Wang and a. S.-F. Chang, "Temporal action localization in untrimmed videos via multi-stage cnns," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016.
- [21] S. Buch, V. Escorcia, B. Ghanem, L. Fei-Fei and J. Niebles, "End-to-End, Single-Stream Temporal Action Detection in Untrimmed Videos," BMVC, 2017.
- [22] S. Ma, L. Sigal and S. Sclaroff, "Ma, Shugao, Leonid Sigal, and Stan Sclaroff. "Learning activity progression in lstms for activity detection and early detection," CVPR, 2016.
- [23] Idrees, Haroon, A. R. Zamir, Y.-G. Jiang, A. Gorban, I. Laptev, R. Sukthankar and M. Shah, "The THUMOS challenge on action recognition for videos "in the wild"," Computer Vision and Image Understanding , 2017.
- [24] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang and L. V. Gool, "Temporal Segment Networks: Towards Good Practices for Deep Action Recognition," ECCV, 2016.

- [25] Z. Shou, J. Chan, A. Zareian, K. Miyazawa and S.-F. Chang, "Cdc: Convolutional-de-convolutional networks for precise temporal action localization in untrimmed videos," CVPR, 2017.
- [26] H. Xu, A. Das and K. Saenko, "R-c3d: Region convolutional 3d network".
- [27] A. Radford, L. Metz and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," arXiv preprint arXiv:1511.06434, 2015.
- [28] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford and X. Chen, "Improved techniques for training gans," NIPS, 2016.
- [29] Y.-G. Jiang, J. Liu, A. Roshan Zamir, G. Toderici, I. Laptev, M. Shah and R. Sukthankar, "THUMOS challenge: Action recognition with a large number of classes.," <http://crcv.ucf.edu/THUMOS14/>, 2014.
- [30] "facebook C3D 1.1," [Online]. Available: <https://github.com/facebook/C3D>.
- [31] W. Sultani, C. Chen and M. Shah, "Real-world Anomaly Detection in Surveillance Videos".
- [32] Z. Shou, D. Wang and S.-F. Chang, "Temporal action localization in untrimmed videos via multi-stage cnns," CVPR, 2016.
- [33] D. P. Kingma and J. Ba, "A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.
- [34] M. Hoai and D. I. Torre, "Max-margin early event detectors," IJCV, 2014.
- [35] W. Xu, M. Yang and K. Yu, "3D convolutional neural networks for human action recognition".
- [36] J. Gao, Z. Yang, C. Sun, K. Chen and R. Nevatia, "Temporal unit regression network for temporal action proposals," Proceedings of the IEEE International Conference on Computer Vision, 2017.