Spatio-Temporal Action Detection

Submitted April 2023, in partial fulfilment of
the conditions for the award of the degree
BSc(Hons) Computer Science with Artificial Intelligence.

Student ID: 20215757

School of Computer Science
University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature: Y.C

Date: 23 / 04 / 2023

Abstract

With the development of action recognition, it has been widely used in video surveillance, intelligent driving, medical care and other fields. To deal with complex real-world scenarios, There is a new direction of action recognition: spatio-temporal action detection, which can not only classify the action subjects in the video but also label their bounding boxes to localize. Building a spatio-temporal action detection model and then training and evaluating it on different action video datasets is the subject of this project. In this project, TubeR, a tubelet-level transformer-based action detection model is chosen to implement spatio-temporal action detection. The TubeR model takes a video sequence as input, uses a backbone algorithm to extract feature information in the video and uses a transformer to generate tubelets to detect actions. Finally, the model outputs the categories of actions and the bounding boxes of action subjects in the video.

This project reconstructs the TubeR model using PyTorch and trains the TubeR model on JHMDB-21 and AVA2.1 datasets. The training process is implemented by distributed learning on a GPU cluster with 7 NVIDIA GeForce RTX 2080 Ti. Then, these two trained models are evaluated by calculating the frame-mAP on different benchmarks. When setting IoU=0.5, TubeR has a mAP of 0.72 on the JHMDB-21 dataset and 0.29 on the AVA2.1 dataset. When using the COCO benchmark(IoU=0.5:0.95), TubeR has a mAP of 0.5 on the JHMDB-21 dataset and 0.18 on the AVA2.1 dataset. These results are close to or even exceed the results of the previously state-of-the-art spatio-temporal action detection model. Finally, this project visualizes the prediction data for further analysis and builds a test program that can perform spatio-temporal action detection on input videos.

Contents

1 Introduction and Motivation **1**

2 Background Information **1**
2.1 Object Detection . . . 1
2.2 Convolution Neural Network(CNN) . . . 2
2.3 Channel-Separated Convolution Networks(CSN) . . . 3
2.4 Transformer . . . 3

3 Related Work **4**
3.1 Action recognition . . . 4
3.1.1 Traditional Method . . . 5
3.1.2 Deep Learning Method . . . 5
3.2 spatio-temporal action detection . . . 7
3.2.1 Frame-level action detection . . . 7
3.2.2 Tubelet-level action detection . . . 8
3.2.3 Transformer for Spatio-Temporal Action Detection . . . 8
3.3 Datasets . . . 10

4 Aim and Objective **11**

5 Design and Methodology **11**
5.1 Chosen Model . . . 11
5.1.1 Backbone . . . 12
5.1.2 Encoder . . . 12
5.1.3 Decoder . . . 12
5.1.4 Task Head . . . 13
5.1.5 Loss Function . . . 14
5.2 Justification . . . 15

6 Implementation **15**
6.1 Dataset Selection . . . 16
6.1.1 JHMDB Dataset . . . 16
6.1.2 AVA Dataset . . . 16
6.1.3 Justification for Dataset Selection . . . 16
6.2 Code Implementation of Model . . . 17
6.2.1 Language and package . . . 17
6.2.2 Code Structure . . . 18
6.3 Data Pre-process . . . 19
6.3.1 Data Collection . . . 19
6.3.2 Reading Data . . . 20
6.3.3 Video Transform . . . 20
6.4 Model Training . . . 21
6.4.1 Pipeline of Training . . . 21
6.4.2 Training Technique . . . 21
6.4.3 Training Environment . . . 22
6.4.4 Hyper parameter Setting . . . 22

7 Evaluation and Result **23**
7.1 Evaluation Method . . . 23
7.1.1 Intersection over Union (IoU) . . . 23
7.1.2 Mean Average Precision (mAP) . . . 23
7.1.3 Evaluation Implementation . . . 24
7.1.4 Experimental Setup . . . 24
7.2 Result and Analysis . . . 24
7.3 Visualization . . . 26

8 Discussion and Conclusion **27**
8.1 Discussion . . . 27
8.2 Limitation . . . 29
8.3 Contribution and Further Direction . . . 29

9 Progress and Reflection **30**
9.1 Project Management . . . 30
9.2 Resource Management . . . 33
9.3 Project Reflection . . . 33
9.4 Self-Reflection . . . 33

1 Introduction and Motivation

The population are growing older in countries all over the world. In most developed countries, the ageing society has been going on for decades. In addition, this phenomenon also takes place in developing countries, alongside the decrease in fertility and mortality levels. [1] Recently, with the advancement of technology in recent years, the modern robot becomes part of daily life which can be an effective approach to ease the stress of caring old man. Under such a trend, assistive robots appear as a research hotpot which can improve the quality of elderly people. [2]

However, robots still could not replace humans' role in the environment. Developing an assistive robot which has socially competent service in daily life is very challenging. [3] Aiming to enhance the performance of the assistive robot, the assistive robot needs various advanced technologies to evolve, and action recognition is such advanced technology. Action recognition is an indispensable skill for a robot which is operating in an assistive capacity for persons who may have care. Equipped with this technology, the robot can perform action recognition and action localization and then react accordingly to improve the service quality and prevent accidents. [4]

Action recognition is a significant part of human-AI interaction as actions provide information on the identity of a person, for instance, their personality and psychological [5]. The general to implement action recognition is by detecting the action subject and then classifying the detected action. This action detection area has three basic topics: action detection, temporal action detection and spatio-temporal action detection. Action detection classifies each input video and identifies the action of the characters in the video while temporal action detection receives an untrimmed video and outputs not only the label of the actions but also the temporal interval of the action. The spatio-temporal action detection is based on both temporal and spatial features and outputs the action label, the time interval and the bounding box of the human. [6] For the assistive robot, it needs to know the location of the human as well as the type of action in order to make the following response more effective. Therefore, the main task of the project is to build a model to achieve spatio-temporal action detection and train and evaluate the built model.

This project will reconstruct the TubeR model [7] and select the appropriate dataset for spatio-temporal action detection task to train the built TubeR model. The TubeR is a tubelet-level transformer-based model for spatio-temporal action detection which takes a video sequence as input, uses a backbone algorithm to extract features in the video and uses a transformer to generate tubelets to detect actions. This model outperforms the previous state-of-the-art(SOTA) spatio-temporal action detection model on commonly used spatio-temporal action detection datasets, such as AVA [8] and JHMDB51-21 [9].

2 Background Information

Before the following concepts of action recognition can be understood, there are certain basic concepts and definitions related to computer vision and deep learning that must be laid out.

2.1 Object Detection

Object detection is one of the fundamental tasks in computer vision that involves identifying and locating objects within an image or video stream. Object detection has two main goals which are localization and classification. Localization refers to the process of identifying the location or bounding box coordinates of objects within an image or video, while classification involves assigning labels or categories to the detected objects. This area holds promising prospects for applications in

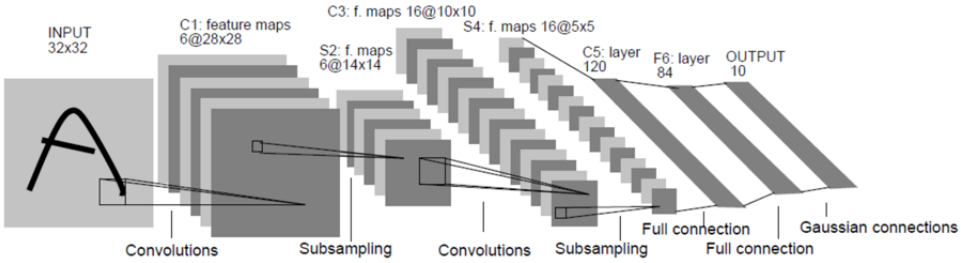

Figure 1: Architecture of a typical CNN: LeNet [13].

today's society, such as in autonomous vehicles for identifying pedestrians, traffic signs, and vehicles and in surveillance systems for detecting intruders.

In recent years, due to the development of deep learning, a large number of well-performance object detection algorithms have been proposed, such as Faster R-CNN [10], YOLO(You Only Look Once) [11] and Detr(detection transformer) [12]. Object detection can often be used as a pre-step for action recognition, because when performing action recognition, the human body or object in the image or video needs to be detected first, and then the action classification can be performed on it.

2.2 Convolution Neural Network(CNN)

Convolution neural network(CNN) is the most basic and common deep learning architecture in the computer vision area which is widely used in image classification, object detection, facial recognition and image generation. CNN is inspired by the human visual system, where the visual information is processed in a hierarchical manner. This architecture can extract low-level features such as edges, corners, and textures from visual information and gradually construct high-level features rich in semantic information to achieve different computer vision tasks.

Here are the key components of a typical CNN architecture:

1. Convolution Layers: These layers apply convolution operations to the input image, using a set of kernels(filters) to extract the features and resulting in a feature map.

2. Pooling Layers: These layers downsample the spatial dimensions of the feature maps obtained from the previous convolution layers, reducing the computing complexity and using the higher-level features to represent images.

3. Non-linear Activation Function: In typical CNN architecture, non-linear activation functions are always used to model the complex non-linear relationship in the image information. Common activation functions used in CNNs are ReLU (Rectified Linear Unit), sigmoid, and tanh.

4. Fully Connected Layers: These layers connect all the neurons from the previous layers to the current layer. Fully connected layers are always used as the final part of the CNN for classification and regression tasks.

To summarize, a typical CNN consists of several convolution layers and pooling layers alternately, and finally, the resulting feature map is passed into a fully connected layer to obtain the final desired result. The following figure is the architecture of a typical CNN–LeNet [13](figure 1)

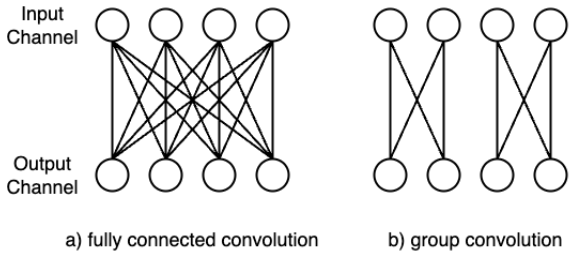

Figure 2: Group Convolution. Filters of CNN can be divided into groups and only receive input within the group. a) a fully connected convolution with only one group, b) 2 groups of filters.

2.3 Channel-Separated Convolution Networks(CSN)

CSN [14] is a type of 3D convolution neural network using group convolution. Group convolution saves a lot of calculations for various network frameworks of traditional 2D image classification tasks. When this technology is transferred to 3D convolution networks for processing video data, it can still effectively improve the accuracy of the model and reduce the amount of calculation. CSN can be 2-3 times faster than other SOTA algorithms while maintaining the same accuracy.

For the traditional convolution algorithm, the filters of the input channel and output channel are fully connected, while in the group convolution algorithm, the convolution filter is divided into different groups. And filter only accepts channels from the same group as input(Figure 2).

2.4 Transformer

The transformer [15] is a deep learning model originally used for NLP(natural language process). It is a neural network architecture based on a self-attention mechanism for processing sequential data such as text. The Transformer establishes a series of self-attention layers between input and output sequences, allowing it to model sequences without relying on their order.

Transformers have been widely used not only in NLP but also in CV. Transformer is mainly composed of two parts: encoders and decoders(Figure 3). The encoder is responsible for segmenting the input image into a series of image patches and embedding these image patches into a high-dimensional feature vector. The decoder processes these feature vectors through the self-attention mechanism to realize the modelling and analysis of the image or video.

Both the encoder and decoder process the input through multiple self-attention layers and feed-forward neural network layers. This self-attention mechanism is the core of the transformer which allows the model to pay weighted attention to the information at different locations when processing sequence data, such as video and text, so as to better capture long-distance dependencies and improve the representation ability and performance of the model.

Here is the formula of the self-attention:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

In this formula, Q is the query, which is the information to be queried, K is the key, which is the vector to be queried, V is the value, which is the content to be queried and d_k is the dimension of K,

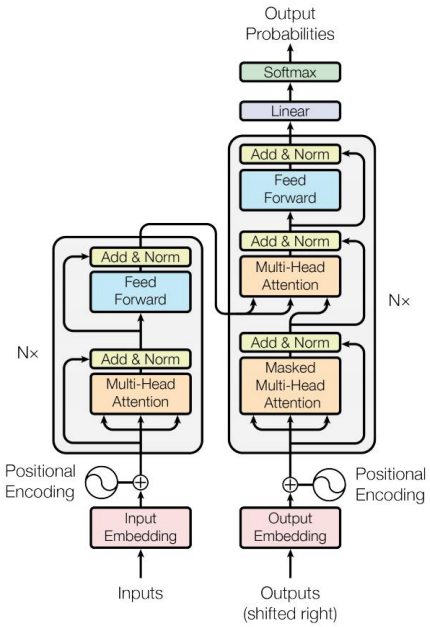

Figure 3: Transformer Structure [15]. The transformer consists of two parts: encoder and decoder, and processes the context in sequence tasks through the self-attention mechanism.

which is used to keeping the gradient stable during training. Q, K and V are all obtained from the input X through linear transformation, and the three matrices multiplied by them respectively W is the parameter matrix obtained through learning. Using such linear transformation can improve the fitting ability of the model.

3 Related Work

As an important research direction in the field of computer vision, action recognition involves the task of automatically recognizing and understanding human actions from videos, image sequences or sensor data. With the deepening of research, in order to deal with more complex environments and a wider range of application scenarios, a new research direction of action recognition has emerged: spatio-temporal action detection which not only recognize the human actions but also locates the action subjects and labels the bounding boxes. This section will focus on the research related to action recognition and spatio-temporal action detection, and introduce the current popular action video datasets.

3.1 Action recognition

Action recognition classifies each input video to identify the actions done by the characters in the video, which is input into the video sequence to obtain the corresponding category of the video. There are two main approaches to implementing action recognition: the traditional method and the deep learning method, which both have been widely used and achieved breakthrough results.

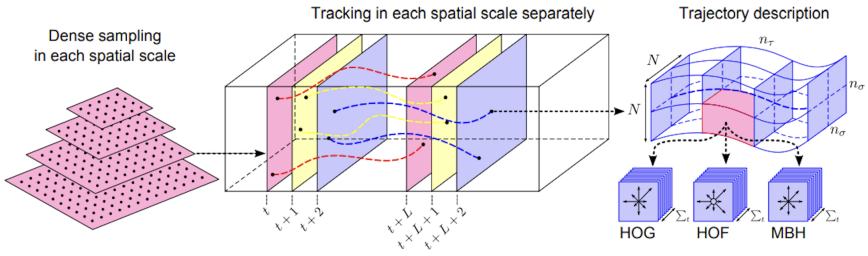

Figure 4: The process DT and iDT algorithm extracts features [16]. This process is divided into three main steps. Firstly, feature points are densely sampled at each spatial scale. In the second step, the trajectory is obtained using a median filter on the corresponding spatial scale image over an L-frames long optical flow field. Finally, HOG, HOF, MBH along trajectory are calculated to obtain features.

3.1.1 Traditional Method

For the traditional method, features are extracted manually and then using a classifier for classification. DT(Dense Trajectories algorithm) [16] and iDT(Improve Dense Trajectories algorithm) [17] are typical and optimal algorithms in the tradition action recognition area. The framework of DT and iDT to extract features has three steps(Figure 4):

1. Dense sampling feature points for each spatial scale.
2. Track the feature points through the optical flow to get the trajectory by filters.
3. extract the features(e.g. histogram of gradient, histogram of flow, motion boundary histograms) along the trajectory manually.

After obtaining the features, encode the features with a bag of feature(DT) or Fisher Vector [18](iDT) and get classification result with SVM(support vector machine).

In general, traditional methods usually rely on manually extracted features such as optical flow, trajectories, key points, etc., and perform well on small datasets. However, these methods may be limited to complex scenes and large datasets due to their difficulties in dealing with complex temporal variations, viewpoint changes, and action diversity.

3.1.2 Deep Learning Method

Besides, deep learning becomes hot research in the activity recognition area in recent years. There are already a large number of deep learning models that perform well in activity recognition. These deep learning models can be mainly divided into three categories according to the architecture and extracted feature type: Two-stream based, Convolution 3D(C3D) [19] based and recurrent neural network(RNN) [20] based.

Two-Stream Based Here is the working principle of the Two-Stream based method(Figure 5). This method was first reported by Two stream CNN [21] which consists of two same CNNs, one is used to processing the spatial information represented by RGB images, and the other is used to process the temporal information represented by optical flow from multi frames. These two CNNs normalize the output with SoftMax and then pass it to a classification full connected layer to obtain the classification score. TSN(Temporal Segment Network) [22] improves the two-stream method based on Two-Stream CNN. TSN divides the video into k parts, randomly samples and extracts

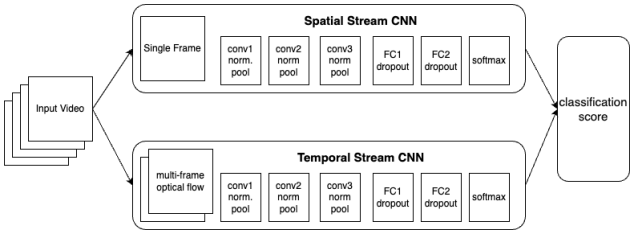

Figure 5: Working principle of Two-Stream based method. This method uses two CNNs to process temporal and spatial information separately.

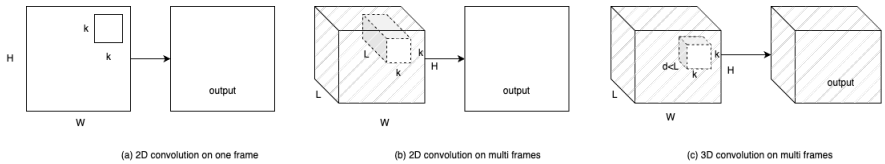

Figure 6: Comparison between 2D convolution on one frame, 2D convolution on multi frames and 3D convolution on multi frames. 2D convolution can only result in one feature map while 3D convolution results in a sequence of feature maps by applying on a video which retains the temporal information from the original video.

features using the two-stream method in each part then fuses all features. In this way, TSN can model long-term video instead of only extracting context from adjacent frames like Two-Stream CNN.

C3D Based C3D is a variant of a convolution neural network which used 3D convolution kernels to extract video information. Here is the comparison between 2D convolution on one frame, 2D convolution on multi frames and 3D convolution on multi frames(Figure 6). Compared with using 2D convolution kernels for a video sequence to get a feature map, using 3D convolution kernels can obtain a sequence of feature maps which retains the temporal information in the original video. Despite generally having lower performance compared to the two-stream based method, the C3D based method is still a popular research direction because it offers advantages such as end-to-end training, a simpler network structure, and faster processing speed. Similar to the two-stream method, C3D based method also focuses on the extraction of spatial and short-term context, but lacks the ability to extract long-term information. To solve this problem, Varol [23] proposed a long-term temporal convolution network, which increases the receptive field of 3D convolution in the temporal dimension at the cost of reducing the spatial resolution.

RNN based To address the issue of poor performance in modelling long-term videos with two-stream based and C3D based methods, researchers have proposed incorporating recurrent neural network (RNN) models into action recognition research. RNNs are well-suited for sequence tasks and can potentially improve the accuracy of recognizing actions in longer video sequences. Here is the working principle of a CNN-RNN framework(Figure 7). The network takes a video as input, uses a CNN to extract the feature in the image, and then passes these temporal correlation frame features into the RNN to obtain the time series result. This architecture was first proposed by Donahue [24] and uses long short-term memory (LSTM) to process temporal information. Subsequent researchers introduce the attention mechanism to combine temporal information and spatial information to optimize the model [25]. And some researchers have combined the two-stream method and RNN, using the two-stream network to extract spatial and short-term context, and LSTM to

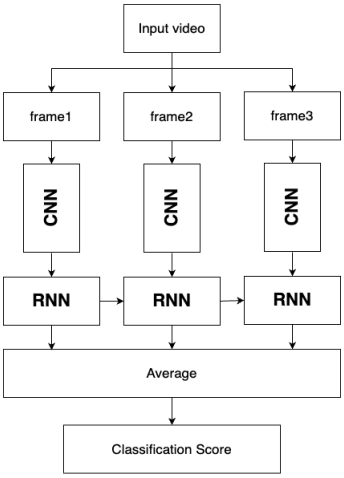

Figure 7: CNN-RNN framework. After extracting frames from the input video, the CNN is used to extract the features of the input image, and then the features of these images with the temporal association are sent to the subsequent RNN for processing, then output is obtained.

extract long-term temporal feature [26].

Compared with traditional methods, deep learning algorithms have better performance in accuracy and flexibility. With the development of the deep learning field and the emergence of more excellent deep learning architectures, deep learning has great potential in action recognition research.

3.2 spatio-temporal action detection

Compare to the action recognition task, spatio-temporal action detection not only needs to identify the interval and the corresponding category of the action but also uses a bounding box to mark the spatial position of the person in the spatial scope. According to different processing methods, the spatio-temporal action detection model can be divided into the frame-level detector and tubelet-level detector. In this project, the chosen model, TubeR, is a tubelet-level transformer-based spatio-temporal action detection model so this section will also introduce how transformer applys to the spatio-temporal action detection.

3.2.1 Frame-level action detection

Frame-level action detection analyses each frame to detect and classify the action separately with image object detector, then links all detected action into a coherent action tube to get temporal context then improve prediction result. Here is the framework of the most frame detector(Figure 8). In addition to generating tubelets by tracing bounding boxes, different frame-level detectors adopt different methods to enhance the prediction results(augmentation module in Figure 8), such as optical flow [27] and 3D convolution network [28]. In general, the frame-level detector first extracts the spatial information in the video for prediction and then uses the temporal information to improve the prediction value. It means this method handles spatial and temporal information separately which may lead to low precision of prediction and easily influenced by environmental context. Different from frame-level action detection, TubeR is a tubelet-level detection method with a unified configuration to simultaneously implement detection and classification [7].

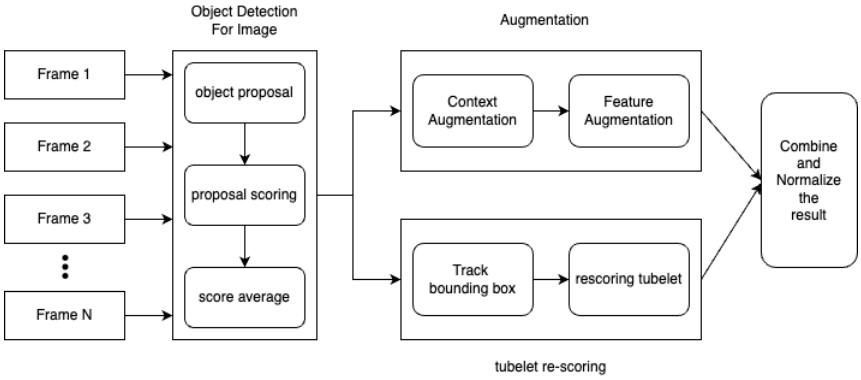

Figure 8: The framework of the frame-level detector. The model reads the video frame by frame and uses object detection to make a prediction for each frame. Then the prediction results are passed to different modules to improve the performance. The tubelet re-scoring module generates tubelets through tracking bounding boxes, which is combined with the temporal context while the augmentation module uses context and other features (such as optical flow) for optimization.

3.2.2 Tubelet-level action detection

Tubelet-level action detection generates spatio-temporal volumes from the whole video clip which can capture more coherence and dynamic actions than frame-level action detection. Each detection of the tubelet-level detector is performed by reading in multiple consecutive video frames and initializing a detection box of the same size at the same position on these frames. These detection boxes are a cube in the time dimension which is called anchor cuboid (left in Figure 9). During the detection process, the model corrects these detection boxes, so a tubelet containing temporal and spatial information is generated (right in Figure 9). Ideally, the tubelet should track the person in the whole video clip. However, the huge configuration space results that previous tubelet-level action detection using only a short cuboid. Although there are several methods to optimise the tubelet representation and extend anchor time such as re-design the cuboid [29] [30] and predict the tubelet by relying on centre position hypotheses [31], tubelet-level action detection is still hard to tackle long video clips. The TubeR model regards the activity recognition task as a sequence-to-sequence problem and embeds temporal information into tubelets by training a small set of tubelet queries. Therefore, TubeR can handle long clips since it can get the temporal correlations within the tubelets.

3.2.3 Transformer for Spatio-Temporal Action Detection

After being proposed for machine translation, transformer [15] became the most popular backbone algorithm for sequence-to-sequence tasks, and it also has significant advances in object detection, image classification and activity recognition. Here is the main structure of transformer-based spatio-temporal action detection(Figure 10). Transformer-based detector uses the backbone algorithm(usually the action recognition algorithm) to extract the feature from the video. Then, the self-attention and cross-attention mechanisms of the transformer are used to process and combine temporal information and spatial information. Finally, different task heads are used to complete the prediction of results. Girdhar [32] propose the video action transformer network firstly for detecting actions and use a region-proposal network for localization. Utilizing the transformer, features from the spatio-temporal context around actors are aggregated to further enhance action recognition. However, this transformer has the same disadvantage that it can not handle spatial information

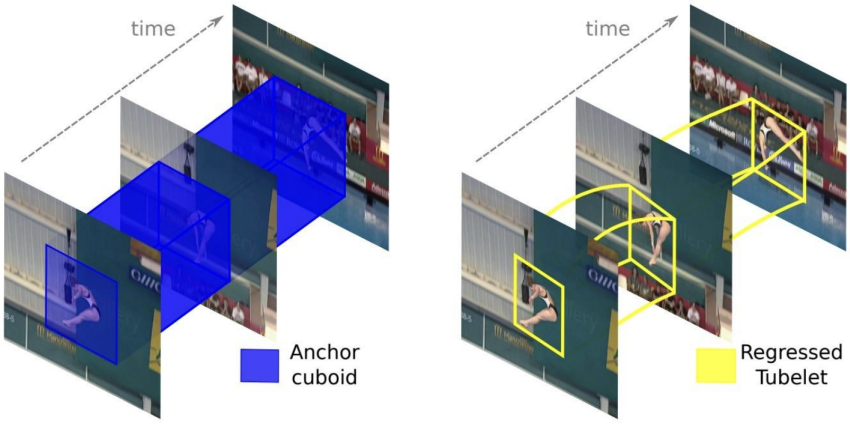

Figure 9: Tubelet generated by the tubelet-level detector. The left one is the cube tubelet in the initial state which is called anchor cuboid, and the right one is the tubelet corrected by regression.

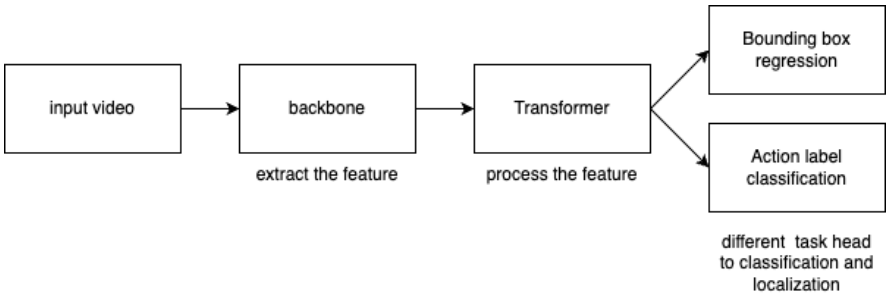

Figure 10: The main structure of transformer-based action detection. Using the backbone algorithm to extract features, Then the self-attention and cross-attention mechanisms of the transformer are used to combine temporal information and spatial information, and finally, the output of the transformer is passed into different task heads to obtain the prediction results.

Dataset	Year	Video	Action	Environment
Weizmann [33]	2004	90	10	Controlled
INRIA XMAS [34]	2006	390	13	Controlled
Hollywood2 [35]	2009	3,669	12	Uncontrolled
KTH [36]	2005	600	6	Controlled
HMDB51 [37]	2011	6,849	51	Uncontrolled
UCF-101 [38]	2012	13,320	101	Uncontrolled
Sports-1M [39]	2014	1,133,158	487	Uncontrolled
THUMOS'14 [40]	2014	413	20	Uncontrolled
ActivityNet [41]	2015	28000	203	Uncontrolled
Charades [42]	2016	9848	157	Controlled
AVA [8]	2017	385446 frames	80	Uncontrolled
Kinetics400 [43]	2017	306,245	400	Uncontrolled
Kinetics600 [44]	2018	495,547	600	Uncontrolled
Kinetics700 [45]	2019	650,317	700	Uncontrolled
HACS [46]	2019	50,000+	200	Uncontrolled
MultiSports [47]	2020	3200	4	Uncontrolled
AVA-Kinetics [48]	2020	238,906	80	Uncontrolled
Kinetics700-2020 [49]	2020	647,907	700	Uncontrolled

Table 1: Details of Action video Datasets

and temporal information at the same time. Meanwhile, TubeR will simultaneously localize and recognize actions because it is a tubelet-level action detection.

3.3 Datasets

With in-depth research in the field of action recognition, there are more and more different types of video datasets that can be used for research in this field. These datasets differ in the number of human subjects, background noise, appearance and pose variations and camera motion, and have been widely used for the comparison of various algorithms. Here is a table showing the detail of popular action video datasets used in action recognition research(Table 1).

We can analyze each dataset from different dimensions, and different types of datasets play different roles in research. From the perspective of dataset size, small datasets, such as Weizmann and KTH, are more suitable for the action recognition task of traditional methods, while AVA, HMDB51 and UCF-101, which have larger data sizes, are more suitable for action recognition tasks based on a deep learning framework. From the perspective of the dataset recording environment, although the dataset of a controlled environment lays a solid theoretical foundation and valuable practical experience for the field of action recognition, it may not be possible to train a method that can be applied to the real environment due to the single recording environment. However, the datasets of the uncontrolled environment are often collected and labelled from the Internet. Such datasets are from real life and have a large amount of data, which can train robust and applicable models. With the development of the field of action recognition, the scale of human action video datasets is expanding rapidly in order to train models with better performance. In the latest action recognition research, the AVA dataset introduced by Google and Kinetics dataset introduced by Deepmind is the most popular benchmark dataset. Both datasets reach million-level annotations. Moreover, these two companies continue to expand these two datasets and launch large datasets such as Kinetics600, Kinetics700, and AVA-Kinetics, which provide rich training samples for research in the field of action recognition.

Since human actions are complex and numerous, no dataset can cover all of them. Moreover, the research in the field of video analysis is not very mature, and the quality, quantity and generalization ability of video samples have a huge impact on the performance and accuracy of the model. Therefore, in the research of action recognition, the selection of the dataset has a great impact on the final performance of the model. When selecting a dataset, it should be carefully investigated from various dimensions, such as the number of actions, the number of action subjects, the resolution of the video, and the frame rate.

4 Aim and Objective

The main aim of the project is to build a spatio-temporal action detection model for an end-to-end action detection system which will be deployed on the assistive robot. The system receives the video captured in real time by the camera as input and outputs the label of the action and the bounding box of the human to handle different situations.

Here are the key objectives of this project:

1. Research existing methodologies of spatio-temporal action detection and select methodology based on the performance in different datasets.
2. Research existing action detection video datasets and select the appropriate datasets for the chosen spatio-temporal action detection model.
3. Construct the chosen spatio-temporal action detection model at the code level.
4. Train and evaluate the constructed spatio-temporal action detection model with the chosen dataset.
5. Visualize the output data of the trained spatio-temporal action detection model and build a visualization program that makes predictions on the input video to test the model.

5 Design and Methodology

5.1 Chosen Model

The core part of this project is to build a spatio-temporal action detection model to classify actions in the input video and detect bounding boxes of action subjects. This project chooses the TubeR model which is a tubelet-level transformer-based action detection model with SOTA performance on commonly used action detection datasets. This model gets inspired by sequence-to-sequence modelling in NLP (natural language processing) and applies it to tubelet detection which makes TubeR can change the size and position of the tubelet.

The TubeR is consist of four modules including the backbone algorithm, encoder, decoder and two task heads. TubeR receives a video clip $V \in T \times H \times W \times C$ where T is the number of frames, H, W are the size of each frame and C is the colour channels of the video and applies the 3D backbone algorithm to extract the video feature $F_b \in T' \times H' \times W' \times C'$ with temporal dimension T', spatial dimension H', W' and feature dimension C'. Then F_b will be transformed to a set of tubelet-specific features $F_{tub} \in N \times \ T' \times C'$ where N is the number of tubelets after processing by the encoder and decoder. finally, the two specific task heads will receive the F_{tub} and out the activity label and bounding box separately. Here is the architecture of the TubeR(Figure 11).

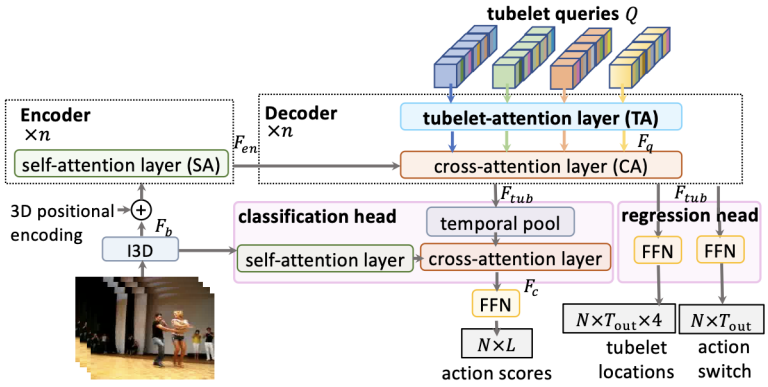

Figure 11: Structure of the TubeR model [7]. This model takes a video as input and uses the 3D backbone algorithm to extract spatio-temporal features in the video. The model then passes the features through position embedding to the transformer. The transformer consists of n encoders and n decoders. The encoder will process the extracted features through the self-attention layer. The decoder will obtain the tubelet-level spatio-temporal feature by applying the cross-attention layer on the output of the encoder and the generated tubelet query. Meanwhile, the tubelet query is obtained by applying self-attention to spatial information and temporal information separately. Finally, two specific task heads are used to generate bounding boxes and classify actions.

5.1.1 Backbone

The backbone algorithm is used to extract the feature from the original video clips which is always a 3D convolution neural network or other similar models. By using the backbone algorithm, the TubeR can receive the original video clips as input to avoid the complex process of feature extraction and data reconstruction in traditional activity recognition algorithms. Additionally, the backbone algorithm can apply down sampling on the temporal dimension to compress the input size, which can promote the efficiency of the algorithm.

5.1.2 Encoder

The TubeR set n encoders to process the video feature F_b and get a set of vectors $F_{en} \in T'H'W' \times C'$. Similar to other transformers, each encoder is composed of a self-attention layer(SA), two normalization layers and a feed-forward network(FFN) [15], (Figure 12). Here is the equation of the self-attention layer which is the core part of the encoder.

$$\mathrm{SA}\left(F_{\mathrm{b}}\right)=\operatorname{softmax}\left(\frac{\sigma_q\left(F_{\mathrm{b}}\right) \times \sigma_k\left(F_{\mathrm{b}}\right)^T}{\sqrt{C'}}\right) \times \sigma_v\left(F_{\mathrm{b}}\right), \tag{2}$$

$$\sigma(*)=\mathrm{Linear}(*)+\mathrm{Emb}_{\mathrm{pos}} \tag{3}$$

The $\sigma(*)$ is the linear transformation the adds the 3D position embedding which includes the spatial and temporal information to represent the sequence of the frames.

5.1.3 Decoder

The decoder utilizes a small set of trained tubelet queries to reconstruct F_{en} to tubelet-specific F_{tub}. Different from the vanilla transformer, the decoder uses a tubelet-attention layer to train the

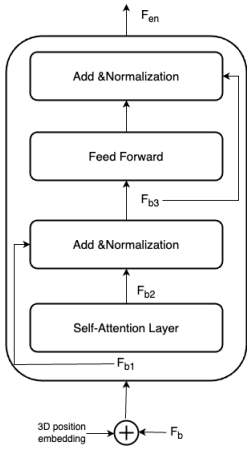

Figure 12: Encoder Structure. The encoder models the input features by a self-attention layer, normalizes them and passes them through a feed forward neural network to get the output

tubelet queries Q and pass it into the cross-attention layer to get the tubelet feature. Here is the detailed structure of the decoder(Figure 13).

For the tubelet queries $Q \in N \times T_{out} \times C'$, this model initialize N tubelets identically with $Q_i = \{q_{i,1}, q_{i,2}, ..., q_{i,T_{out}}\}$. $q_{i,t} \in C'$ is the embedding feature of the frame which means Q is the set of the feature map in all frames. After training, the tubelet queries can represent the tubelets' information dynamically instead of using fixed 3D anchors.

In the tubelet-attention layer, there are 2 self-attention layers, one for the spatial feature and another one for the temporal feature. The spatial-self-attention layer rebuilds the queries based on the spatial relations between the tubelets in the same frame then the temporal-self-attention layer rebuilds the queries based on the temporal relations between different frames of the same tubelet. This layer will output the tubelet query features $F_q \in N \times T_{out} \times C'$ which have the same dimension as the original queries Q. For the cross-attention layer, it will receive the F_q and Fen (generated by the encoder) as parameters to execute the cross-attention. Here is the equation of cross-attention which is similar to self-attention.

$$\mathrm{CA}\left(F_{\mathrm{q}}, F_{\mathrm{en}}\right) = \mathrm{softmax}\left(\frac{F_{\mathrm{q}} \times \sigma_k \left(F_{\mathrm{en}}\right)^T}{\sqrt{C'}}\right) \times \sigma_v \left(F_{\mathrm{en}}\right), \tag{4}$$

The output of the cross-attention layer and decoders are the tubelet feature $F_{tub} \in N \times T_{out} \times C'$. In this project, we can assume that $T' = T_{out}$.

5.1.4 Task Head

In the project, we need the TubeR model for spatio-temporal action detection. Therefore the specific task of the model is to get the label of the actions and the bounding box of the action subjects. Apparently, the first task is a classification task and the second one is a regression task.

For the classification task, context(e.g. change of the environment) is also an important concept in a sequence-to-sequence model. To emphasize the importance of the context and the accuracy of the

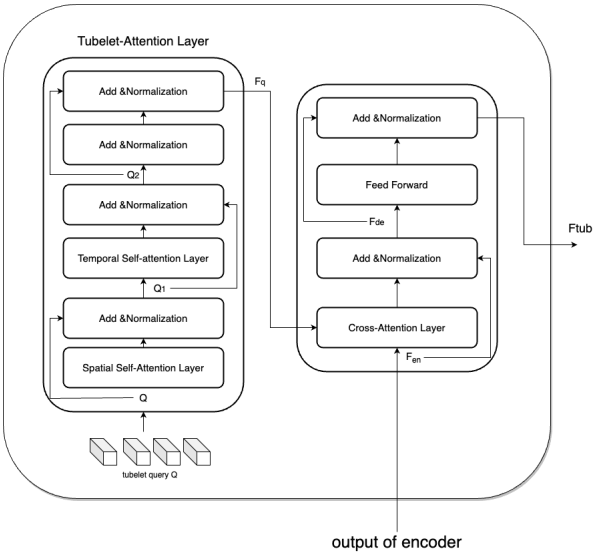

Figure 13: Decoder Structure. The decoder consists of two parts, the tubelet-attention layer and the cross-attention layer. For the Tubelet-attention layer, the model is initialized with a set of identical tubelet queries, and tubelet query with spatial-temporal context is obtained by the spatial-self layer and temporal-self layer respectively. Then the cross-attention layer obtains the tubelet feature by modelling the output of the encoder and the tubelet query.

classification, the TubeR model uses the original extracted feature F_b as the context to strengthen F_{tub} and pool the F_{tub} to reduce the calculation. Here is the equation:

$$F_{\mathrm{c}}=\mathrm{CA}\left(\mathrm{Pool}_t\left(F_{\mathrm{tub}}\right),\mathrm{SA}\left(F_{\mathrm{b}}\right)\right)+\mathrm{Pool}_t\left(F_{\mathrm{tub}}\right) \tag{5}$$

This classification head has three main modules which are the temporal pooling layer, self-attention layer and cross-attention layer. Cross-attention layer will receive the feature after F_{tub} is pooled and F_b rebuilding by the self-attention layer as parameters and output the feature $F_c \in N \times C'$. F_c can be seen as the 2D feature with context information as the temporal dimension is reduced after pooling. Finally, pass F_c as input into a full connection layer to get the classification score $Y_{class} \in N \times L$ where L is the number of the possible labels.

For the regression task, the task can be achieved by passing F_{tub} into a full connection layer. It will output $Y_{coor} \in N \times T_{out} \times 4$ which means the position of the bounding box in every frame for each tubelet(4 means the horizontal coordinate, vertical coordinate, width and height of the bounding box). Additionally, in the video clip, there are a few frames where the character may not be doing any valid activity. Consequently, another full connection layer can be used to predict whether there is a valid activity and output the action switch score in every frame for each tubelet $Y_{switch} \in N \times T_{out}$. If the score is less than the set threshold, the bounding box will not be printed(Figure 14).

5.1.5 Loss Function

the loss function of TubeR can be regarded as the summary of the loss function of each module. Here is the equation:

$$\begin{aligned}\mathcal{L}=&\lambda_1\mathcal{L}_{\text{switch}}\ \left(y_{\text{switch}}\ ,Y_{\text{switch}}\ \right)+\lambda_2\mathcal{L}_{\text{class}}\ \left(y_{\text{class}}\ ,Y_{\text{class}}\ \right)\\&+\lambda_3\mathcal{L}_{\text{box}}\ \left(y_{\text{coor}}\ ,Y_{\text{coor}}\ \right)+\lambda_4\mathcal{L}_{\text{iou}}\ \left(y_{\text{coor}}\ ,Y_{\text{coor}}\right)\end{aligned} \tag{6}$$

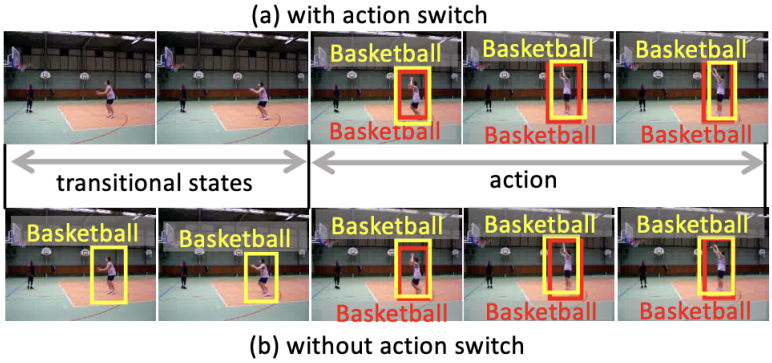

Figure 14: Action Switch [7]. The yellow box and yellow label are the prediction result while the red means the ground truth. Action switch can avoid the model from giving the wrong prediction when the action subject transforms action.

The loss function consists of 4 parts, which are the binary cross entropy loss of the predicted action switch, the cross entropy loss of the classification, the error of bounding box matching and the error of generating the IoU.

5.2 Justification

The TubeR is a well-structured and powerful spatio-temporal action detection model which is a good choice to develop the project. The specific reason is as follows:

1. The TubeR is an end-end model, which means it receives a video stream and outputs the action label and bounding box without other operations and configurations.
2. The TubeR is the newest model in the action detection area and performs outstanding results on commonly used action detection datasets [7].
3. The model constructed in this project will be deployed on the robot and working in the real environment which needs a robust system. Compared with other models of action detection, TubeR can detect human action and generate the bounding box in a more flexible, robust and persistent way.
4. The constructed model will be used in the real world which is more complex and has more distraction compared to the datasets. TubeR uses context to strengthen the feature when executing the classification task. This makes the system more robust and accurate in the face of a complex real environment.
5. TubeR uses a backbone algorithm to extract and pool the feature before training the transformer model. Therefore, under the same computational conditions, more training data can be used to improve the accuracy of the TubeR model.

6 Implementation

The implementation of the end-to-end action recognition system can be divided into the following steps: datasets selection, data pre-processing, code implementation, model training and model evaluation and visualization.

6.1 Dataset Selection

In this project, AVA2.1 and JHMDB datasets were selected to implement this spatio-temporal action detection system, this subsection will introduce the detail of these datasets and explain the reasons for choosing these two datasets.

6.1.1 JHMDB Dataset

JHMDB is a secondary annotation of the HMDB51 dataset, so it is called joint-annotated HMDB. The HMDB51 dataset has 51 classes and more than 5100 videos, while JHMDB only annotates 21 classes of actions that only include one person's body movement action in the HMDB(such as sit, run, kick ball and golf), and also removes some samples that are not obvious among the 21 classes of actions. JHMDB is a densely annotated dataset, which labels each frame of a video. There are 36-55 samples for each of the 21 categories of actions, each sample includes the start and end time of the action, each sample includes 14-40 frames, and a total of 31838 images are annotated. Each video has at most one target action, and only the bounding box of the subject acting is annotated.

6.1.2 AVA Dataset

AVA(Atomic Visual Actions) dataset is a large video dataset released by Google for spatio-temporal action detection. Different from the JHMDB dataset, there are multiple action subjects in each frame of the AVA dataset, and each action subject may have multiple actions at the same time. Therefore, the dataset not only needs to annotate the bounding box and action category of each action subject but also needs to annotate the number of action subjects to track it. Meanwhile, the AVA dataset is a sparsely annotated dataset due to the huge amount of data. Instead of annotating each frame in the video, it takes one frame every second as a keyframe for annotation. The dataset has a total of 80 different action labels which can be divided into the following three categories:

- Person Movement: crawl, dance, sit, walk, fall down, etc.
- Object Manipulation: answer phone, brush teeth, chop, dig, drink, etc.
- Person Interaction: hand shake, play with kid, push (another person), watch (a person), etc.

The AVA dataset comprises data from 430 different movies, with clips ranging from 15 to 30 minutes (902s-1798s) that were labelled in 1-second intervals, resulting in a total of 1.62 million action labels. These 430 videos in the dataset are divided into 235 for training, 64 for validation, and 131 for testing purposes.

6.1.3 Justification for Dataset Selection

Unlike traditional action recognition tasks, this project is a spatio-temporal action detection task. AVA and JHMDB datasets are the most popular datasets in this field. Using these two datasets is convenient for horizontal comparison with other spatio-temporal action detection models and evaluating the performance of the model intuitively and effectively. In addition, there are several reasons to choose AVA and JHMDB.

1. Dataset Size: JHMDB annotates 31838 frames, while AVA annotates 430 15-minute movie clips with one-second intervals. Both datasets have a sufficient amount of data to meet the training requirements of the deep learning framework.
2. Annotation Method: The AVA and JHMDB datasets adopt different annotation methods, with JHMDB using dense annotations and AVA using sparse annotations. Choosing these datasets with different annotation densities can allow for investigating the performance of models when given varying levels of temporal information.

3. Public Availability: The TubeR model used in this project is based on a deep learning framework and requires a large amount of video data for training. Collecting and annotating video data, either by recording videos from testers or searching for videos online, can be a daunting and complex task, and may involve various ethical considerations. Therefore, choosing open-source datasets such as AVA and JHMDB not only meets the research needs but also facilitates reproducibility and comparison of experiments.

4. Data Integrity: Research in the field of spatio-temporal action detection is based on a deep learning framework. Therefore, datasets in this field are generally downloaded from video websites such as YouTube by URL, and part of the data and annotations will be lost and unavailable due to the deletion of the original video. For example, the annotation file of the UCF101-24 dataset can no longer be downloaded from the official website. The AVA dataset is obtained from movie clips and the JHMDB dataset provides compressed files of all videos on the official website, thus ensuring the integrity of the dataset.

5. Environment: AVA and JHMDB datasets are both derived from movie clips or movies recorded in real life, which are recorded in an uncontrolled environment. The environmental context of such datasets is more complex and closer to real life scenes. The trained spatio-temporal action detection model will have better performance and be more robust in real life scenarios.

6.2 Code Implementation of Model

6.2.1 Language and package

Python The language selected to build the TubeR model in Python. The TubeR is a deep learning model, which needs to process a mass of data and use existing frameworks to build the model. python provides a large number of excellent deep learning frameworks and libraries for processing the data and building the model(e.g. Pandas, Numpy, Scikit-learn, Keras, TensorFlow and PyTorch). In addition, representing data in a human-readable format is crucial in AI, deep learning, and machine learning. Python comes with a lot of great visualization libraries such as Matplotlib enable users to create histograms, graphs, and plots to improve understanding, display, and visualization of data. Certainly, another reason for choosing Python for action recognition research is that a large portion of the research in this field is implemented in Python and this ensures the code is reproducible. Meanwhile, Python has a vast range of image processing libraries, such as OpenCV and Pillow, that enable researchers to read, manipulate, and display images easily.

Pytorch, CUDA and TensorboardX The main package to build, train and evaluate the deep neural network is PyTorch. PyTorch has several features that make it particularly relevant for deep learning. First, it uses GPUs to accelerate computations, which are typically 50 times faster than performing the same computation on a CPU. Second, Tensors in PyTorch have the ability to track the operations performed on them and analyze and compute the derivative of the output corresponding to any input. This functionality is used for numerical optimization and is provided by the tensor itself, which is scheduled through the PyTorch low-level automatic derivative engine. Third, PyTorch is based on NVIDIA's GPU parallel computing framework - CUDA, so it also supports distributed training. As a result, we can use PyTorch to train models on multiple devices and GPUs, which greatly improves the training efficiency. In addition, PyTorch can also use TensorboardX, which can log the curve of loss function during training to visualize the training process(Figure 15).

Gluon CV [50] and VMZ Gluon CV is a GitHub repository which provides implementations of the state-of-the-art deep learning models in computer vision. In this project, I refer to the implementation code of CSN-152 as the backbone algorithm of TUbeR. VMZ is a Pytorch codebase for video modelling developed by the Computer Vision team at Facebook AI and I download the

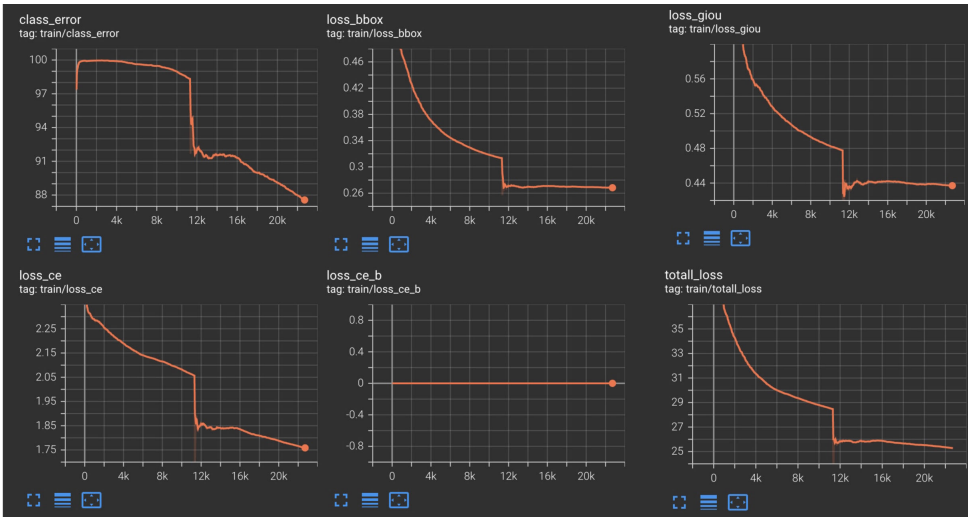

Figure 15: Visualize the training process with TensorBoardX. In this figure, the tensor board records the 4 parts of the loss function, which are classification error, loss of matching bounding box, loss of IoU and cross entropy of action switch(binary cross entropy is not used to calculate the loss of action switch in this training, so the curve for loss_ce_b does not change). Since the model updates the gradient by back propagation at about 11000 steps, the loss decreases significantly for all parts.

pre-train parameter of CSN-152 from VMZ for transferring learning.

TensorFlow Similar to PyTorch, TensorFlow is a framework for deep learning in Python, providing a large number of model implementations, visualization tools, and training support. In this project, I modified the TensorFlow object detection evaluation API to evaluate the performance of the TubeR model.

6.2.2 Code Structure

This is the structure of the codebase for this project(Figure 16):

This project consist of following package:

- datasets: the package to download and process the datasets.
- model: implement the TubeR, backbone algorithm, transformer and criterion using PyTorch.
- evaluates: the package to evaluate the model result refers to TensorFlow API.
- utils: the package contains modules to support other packages.
- pipeline: the package to implement the pipeline of the distributed learning process.

The Python script files in the TubeR directory are used for model training, model evaluation, rendering predictions, visualizing predictions on video, and a simple spatio-temporal action detection system. Please read the README.MD for details to run the project.

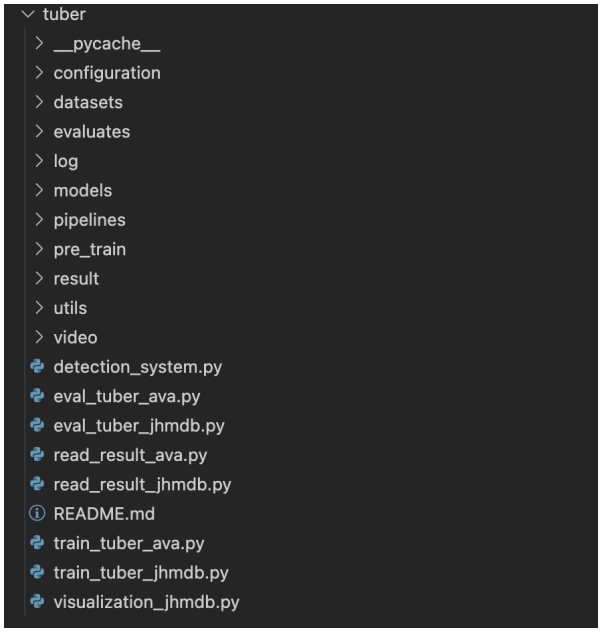

Figure 16: Code Structure.

6.3 Data Pre-process

6.3.1 Data Collection

For the JHMDB dataset, since there are only annotation files of joint position and puppet mask on the official website of this dataset, I downloaded the frame file of the JHMDB dataset and the pickle file of bounding box annotation from the GitHub repository of mmaction2 [51]. All annotations are stored in the JHMDB-GT.pkl file, which can be unpickled into a Python dictionary, which contains 6 items as follows:

1. labels(list): the name of the 21 labels.
2. gttubes(dict): a dictionary which contains the ground truth of the dataset. The index is the name of each video which include action labels, while the value is a $1 * 5$ Numpy array which represents the frame ID and the coordinate of the bounding box.
3. nframes(dict): a dictionary that stores the frame number for each video.
4. train_video(list): the list of training videos' names.
5. test_video(list): the list of testing videos' names.
6. resolution(dict): a dictionary contains the resolution for each video.

To get the AVA dataset, firstly, download annotation files and ava_file_names_trainval_v2.1.txt from the official website which contains the video names of all the training and validation sets in the dataset. Then run the following three .sh files to get the extracted frames: download_ava.bash(download videos on the list), chunk_video.sh(using ffmpeg [52] to clip each video from 15 to 30 minute) and extracr_frame.sh(using ffmpeg to extract frame). Here is the diagram of the folder structure(Figure 17).

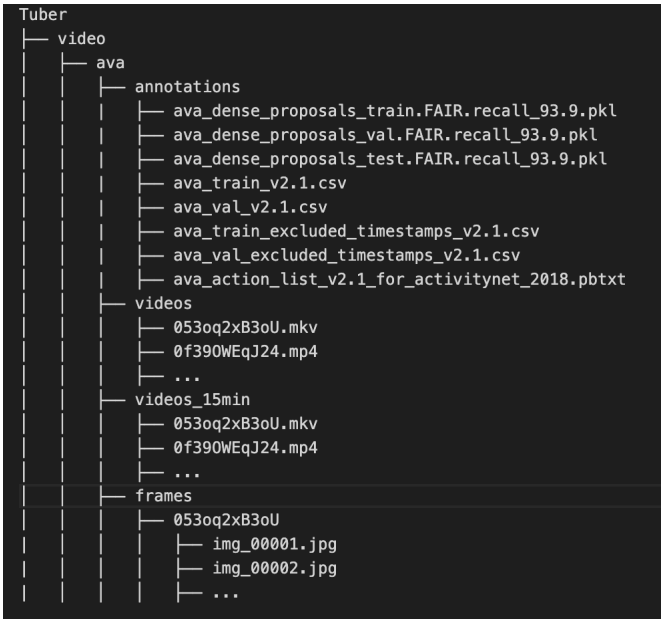

Figure 17: Folder Structure of AVA dataset.

6.3.2 Reading Data

Since the video dataset annotates the frames in the video, in order to correspond to the annotation, this project have clipped the video into frames and saved it as images when acquiring the data. However, human action is a continuous process, and it is obviously impossible to accurately predict the action by only one frame. Therefore, to take into account the temporal context of the current frame, we set a parameter called TEMP_LEN. When reading the dataset by frame, the data-loader reads $\frac{TEMP\_LEN}{2}$ frames before and after the current frame and fills in the out-of-bounds frames with the first and last frames. Finally, it results a list of Numpy arrays, where each element is a TEMP_LEN long video sequence corresponding to each frame.

6.3.3 Video Transform

Data Augmentation In order to enhance the diversity and robustness of the training data and improve the generalization ability of the trained model. We perform data augmentation by transforming the read video data. For the validation set, I only resize the image, while the following transforms are applied on the training set:

- Random Horizon Flip: Flip the image randomly.
- Random Size Crop: Randomly cropping a portion of the image, the size and aspect ratio of the image can be changed.
- Color Jitter: Adjust the brightness, contrast, hue and saturation of the image and improve the quality of the image.

Normalization To improve training speed and model accuracy, it's necessary to normalize this pre-processed data. In this project, Z-score scaling was chosen as the normalization method. Firstly, we need to convert these transformed images into tensor images. Then, calculate the mean and

standard deviation of each channel in all tensor images. Finally, use the following formula to calculate the normalized value of each pixel:

$$z = \frac{(x - \mu)}{\sigma} \tag{7}$$

Where x is the original value of the pixel, μ is the mean, σ is the standard deviation, and z is the normalized value. With this formula, each pixel in the tensor image can be normalized to have a mean of 0 and a standard deviation of 1.

6.4 Model Training

6.4.1 Pipeline of Training

This project's model training pipeline is no different from a normal PyTorch pipeline. Firstly, use DataLoader to load the dataset and build the model, criterion (loss function), optimizer and learning rate scheduler. Secondly, set the model to train mode, and pass the data in DataLoader into the model to get output. Then pass the target and output to the criterion to obtain loss. Finally, call the following three functions to update the parameters and start the training for the next epoch:

- optimizer.zero_grad(): set the gradient to zero.
- loss.backward(): use back propagation to calculate the gradient of each parameter.
- optimizer.step(): perform gradient descent to update the parameter of the model.

When the termination criterion is met or the iteration reaches the specified epoch, we save the checkpoint using torch.save() and complete the training.

6.4.2 Training Technique

Because video data has the characteristics of high dimension and a large amount of data, the neural network for extracting video information features often needs a large number of neurons to ensure the performance and expression ability of the model. For example, in this project, the TubeR model built for the JHMDB dataset has 49.81M parameters. To ensure the efficiency of the training process and enhance the accuracy of the model, we use transfer learning and distributed data parallel:

Transfer Learning Transfer learning refers to the process of transferring the trained model parameters to a new model to help it train. These parameters can be used as initialization parameters or frozen as constants. The TubeR model can be divided into 3 parts which are the backbone algorithm, transformer and fully connected layer. For this project, I chose the ir-CSN-152 model taken from the VMZ repository on GitHub as the backbone algorithm to reduce training time and improve computational efficiency. This pre-trained CSN model used the IG-65M dataset [53] to train, then fine-tuned on the Kinetics-400 dataset. The parameters of this pre-trained model are only used as initialization parameters for the TubeR model, and these parameters will be updated rather than frozen during the training process.

Distributed Data Parallel Distributed Data Parallel(DDP) is a technique for model training in a distributed computing environment. When running the training program using DDP mode on N GPUs, the system starts N processes. Each process loads the model on its own GPU with the same parameters. During training, the processes exchange their gradients to get all the gradient information. Subsequently, the individual processes update the parameters using the average of the gradients. Since the initial and updated values are exactly the same, the updated parameters

remain the same for each process. This project utilizes DDP during the training process, enabling the model to be trained on multiple GPUs while ensuring the load is balanced. This results in a significant reduction in training time and an improvement in model performance.

6.4.3 Training Environment

The training for this project was done on the Linux GPU cluster server Gerty at the University of Nottingham. Here are the configuration details for this Linux server.

Hardware Environment:

- CPU: The server is equipped with 2 Intel(R) Xeon(R) Gold 5218 @ 2.30GHz CPU. Each physical CPU has 16 cores and each core has two processes, resulting in a total of 64 logic processors.
- GPU: The server is equipped with 7 NVIDIA GeForce RTX 2080 Ti, Each GPU has 11 GB of video memory.
- Memory: The server is equipped with 4 64GB RAM memory GIGABYTE G291-281-00, a totally of 256 GB RAM.
- Disk: This server is equipped with a disk with about 1 TB of memory (actually 914 GB) and a file server connected to it for storing large data such as video datasets.
- Network Interface Controller: This server is equipped with 4 Ethernet controllers which are 2 Intel Corporation Ethernet Controller 10G X550T and 2 Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection to ensure the stability of the network connection.

Software Environment:

- Python 3.7.12
- Torch 1.12.1(initial) and Torch 2.0.0(updated)
- CUDA 12.1
- TensorboardX

6.4.4 Hyper parameter Setting

Training Parameter

- Batch Size: 2
- Number of Epoch: 20
- Learning Rate: For parameters of the backbone algorithm, the learning rate is set to 1×10^{-5}, while for parameters of the transformer and task head, the learning rate is initialized to 1×10^{-4} and updated with lr_scheduler in PyTorch.
- Optimizer: AdamW

Neural Network Structure These are the main network structure related hyper parameters:

- Output Channel of backbone: 2048
- Number of Query: 10(JHMDB), 15(AVA)
- Dimension of Features($C^{'}$ in methodology section): 256
- Number of Encoder: 6
- Number of Decoder: 6
- Head of Multi-Head-Attention: 8
- Dropout: 0.1
- Length of Clip: 32
- Down sampling Rate: 8

Where the number of query is the number of tubelets generated in the decoder for query action and the down sampling rate is used in the pooling layer of the backbone algorithm.

7 Evaluation and Result

7.1 Evaluation Method

7.1.1 Intersection over Union (IoU)

IoU is a common computer vision metric to calculate the accuracy of object detection. IoU calculates the overlap ratio between the candidate bounding box and the ground truth bounding box, which is the ratio of their intersection and union.

$$IoU = \frac{area(C) \cap area(G)}{area(C) \cup area(G)} \tag{8}$$

Where $area(C)$ is the area of the candidate bounding box and $area(G)$ is the area of the ground truth bounding box. The closer the IoU is to 1, the more accurate the prediction is. Under normal circumstances, if the IoU reaches a certain threshold, the algorithm is considered to successfully detect the target. In experiments, researchers usually evaluate the algorithm by adjusting the IoU threshold.

7.1.2 Mean Average Precision (mAP)

mAP is the average AP value of each action category of multiple validation sets, which is an important evaluation index to measure the detection accuracy in action detection. To calculate the mAP, it needs to start with a confusion matrix which is often used as an evaluation method in almost machine learning tasks(figure 18). Then we need to calculate the precision and recall:

$$precssion = \frac{TP}{TP + FP}, \tag{9}$$

$$recall = \frac{TP}{TP + FN}, \tag{10}$$

For this action detection task, to get TP and FP, we need to calculate the IoU to determine whether a test result(Positive) is True or False. For example, we set a threshold of 0.5, if IoU > 0.5 it is considered a True Positive, otherwise it is considered a False Positive. To calculate FN, We only need to count actions that are not detected by the model. By changing the confidence threshold, we can change whether a prediction box is Positive or Negative. Next, the precision and recall at different confidence thresholds are recorded and the P-R image is plotted(Figure 19), while the area of the image is the AP value for that category. Finally, the mAP is obtained by calculating the mean of APs for all classes. To evaluate the performance of the action detection model at different levels, there are two different mAP calculation methods, frame-mAP and video-mAP. Frame-mAP

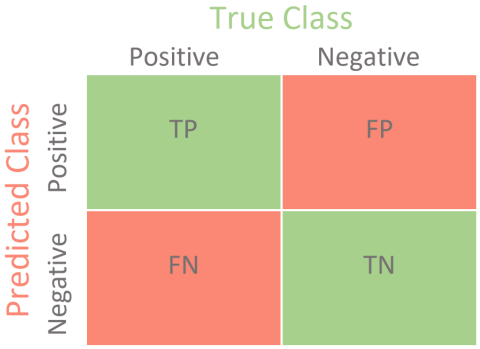

Figure 18: Confusion Matrix.

calculate the area under P-R curve of each frame while video-mAP focuses on the action tubelet. When computing the video-mAP, a detected tubelet is correct if the average of the IoU of each frame in it is greater than the threshold.

7.1.3 Evaluation Implementation

After feeding the input data into the model, we get a predicted bounding box coordinate and a set of classification scores for all action classes. Then, a post-processor is used to convert the bounding box coordinates to a form accepted by other benchmarks (such as COCO and VOC Pascal) and normalize the classification score using the SoftMax function. These predictions are temporarily stored in log/AVA_Tuber/tmp or log/JHMDB_Tuber/tmp for further evaluation. Since the TubeR model produces more than 10 tubelets during prediction, some useless detection boxes will be generated, and a threshold value should be set to filter the bounding box (set 0.01 in this project). Finally, different benchmarks are used to calculate the mAP. There are three popular mAP benchmarks:

- COCO: Calculate the mAP with IoU from 0.5-0.95 and stride of 0.05, then calculate the mean of these mAPs as the result.
- VOC PASCAL: set IoU = 0.5.
- Strict Metric: set IoU =0.75, with the higher requirement of bounding box.

7.1.4 Experimental Setup

In this project, I report the experiment on two popular spatio-temporal action detection datasets: AVA2.1 and JHMDB-21. This project uses three different benchmarks mentioned in section 7.1.3 to compare the performance of TubeR on both datasets and calculates the AP of each action class in the dataset to evaluate which actions the model is more sensitive to.

7.2 Result and Analysis

Table 2 is the mAP with 3 different IoU settings on AVA2.1 and JHMDB-21. With an IOU threshold of 0.5, mAP of JHMDB-21 is 0.72, while mAP of AVA2.1 is 0.29. When the accuracy requirement of the bounding box is increased and the strict metric with IoU=0.75 is used for evaluation, the mAP of the TubeR model on the JHMDB-21 dataset can still reach 0.58, while the mAP on the AVA

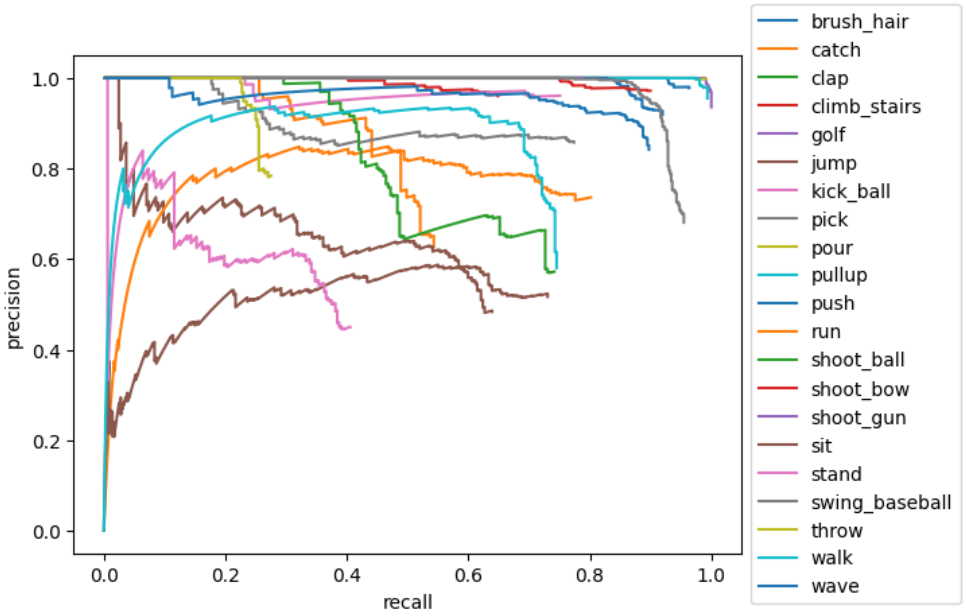

Figure 19: P-R plot of JHMDB-21, IoU = 0.5

IoU(Benchmark)	f-mAP: JHMDB-21	f-mAP: AVA2.1
0.5(VOC PASCAL)	0.72	0.29
0.75(strict metric)	0.58	0.20
0.5:0.95(COCO)	0.50	0.18

Table 2: Result with different mAP benchmark on AVA2.1 and JHMDB-21

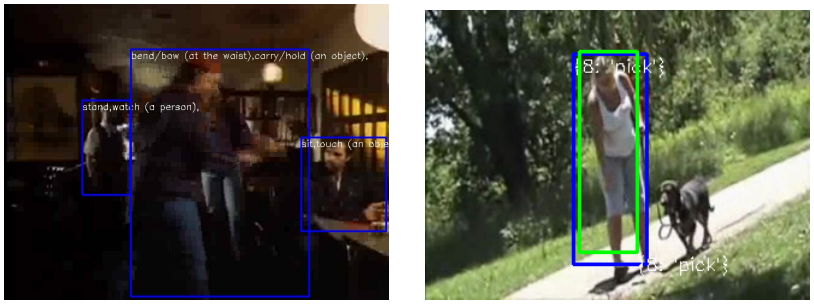

Figure 20: **Compare the frame in AVA2.1(left) and JHMDB-21(right)** The AVA dataset not only has more actions and action subjects but also more complex, hard-to-recognize environments.

dataset is only 0.2. The COCO metric reflects the average performance of the model in different IoU settings. TubeR's mAP on the JHMDB dataset still can reach 0.5, but it is only 0.18 on the AVA2.1 dataset. Among these three metrics, TubeR's mAP on the JHMDB dataset is greater than 0.5 and reaches 0.72 when IoU = 0.5. This is already an acceptable or even satisfactory result for an end-to-end spatio-temporal action detection system. Obviously, the TubeR model on the JHMDB-21 dataset outperforms the AVA2.1 dataset in all three metrics. One possible reason is that there are multiple action subjects in each frame in the AVA2.1 dataset, and each action subject may perform multiple actions. In contrast, although there may be multiple people in the JHMDB-21's video, there is only one action subject and only one action at the same time(Figure 20). While the model trained on the AVA dataset may have better robustness and generalization ability in the real world, a model with a mAP of 0.3 is not useful for practical deployment, so we will focus on the JHMDB-21 dataset in the following discussion.

Figure 21 is a histogram of the AP of all action classes in JHMDB-21 dataset. From this histogram, we can see that most actions' AP is above 0.7, but catch, throw, sit, jump, and stand only have an AP below 0.5. By comparing the ground truth and predict result after visualization, we find that there are two reasons for the low AP of these actions. Firstly, some actions may contain other actions, for example, in the video of stand, the person is in the state of sit before standing, so some frames will be classified as sit, resulting the AP decreasing. Calculating the AP at video-level instead of frame-level can circumvent this problem. Another reason is that some actions are too similar in a certain process, for example, sit and jump, their movements in bending down are almost the same, even people are difficult to distinguish. Increasing the input length of the video sequence accepted by the model might solve this problem, but it would also greatly increase the computing resource requirements. Therefore, in this project, instead of providing the action with the highest classification score as the predict result, we provide the predicted value of multiple actions based on their confidence.

7.3 Visualization

Initially, I used the API in the OpenCV library to annotate the predicted values and ground truth of the bounding box and action label on the clipped frames and then converted these annotated frames into videos to complete the visualization of the results. When analyzing the experimental results with the visualized data, I found that human action has multiple meanings. For example, in some videos, "stand" refers to the transition from "sit" to "stand", while in some videos, "stand" refers to the "stand" state only. Therefore, the reason some action classes' APs are low is not the wrong prediction value of the model, but because the dataset is not rigorously labelled. To solve this problem, I used the top 3 actions with the highest classification scores (classification

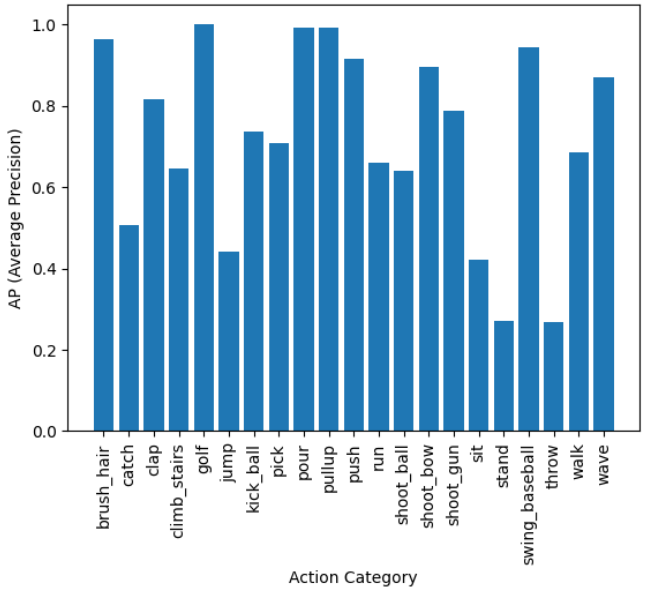

Figure 21: Histogram of the AP of Actions in JHMDB-21, IoU = 0.5. By observing the histogram, it can be found that the AP of catch, throw, sit, jump and stand is much lower than the other categories. This is because the annotation of these actions in the JHMDB-21 dataset is very ambiguous. For example, during the process of the stand, its first few frames should be labelled as sit but the JHMDB-21 dataset labels the entire video as stand.

score>0.01) to replace the predictions given by the model and annotate them on the video. Because the classification score is normalized by the SoftMax function, it can also be used as a probability for a more intuitive visualization. When there is a state transition, the prediction result will be presented in the form shown in Figure 22, so as to avoid the model giving the wrong prediction of ambiguous action.

8 Discussion and Conclusion

8.1 Discussion

Due to a number of action subjects and actions and the complex environment in the AVA dataset, the TubeR model performs poorly on AVA data, the same as most spatio-temporal action detection models. Therefore, I choose the TubeR model trained with the JHMDB dataset to deploy to the end-to-end action detection system. To demonstrate that the trained TubeR model is good enough to deploy, I compare TubeR with other SOTA models using frame-mAP@IoU=0.5 on the JHMDB-21 dataset(Table 3). These models can be divided into frame-level and tubelet-level based on how they process the input, while these models can be divided into two-stream and RGB-stream based on the input stream type. TubeR model is a tubelet-level transformer-based model that outperforms all the above frame-level and Tubelet-level models on the JHMDB dataset. Although the TubeR model accepts video sequences as input and uses the generated tubelet to detect actions, its mAP at the frame-level is still higher than other SOTA models.

In addition to achieving good performance on the benchmark of the model, I also improved the way the model makes predictions in order to improve the robustness and confidence of the model

Figure 22: Example of Final Visualization on JHMDB-21. The green bounding box is the predicted result while the yellow text is the probability and label of the action. The blue bounding box and purple text indicate ground truth. To avoid the influence of ambiguous annotation of the JHMDB-21 dataset, I sorted the classification scores and selected the three actions with the highest scores for labelling (score >0.01). Therefore, only one action is annotated when the action confidence is high enough, and multiple predictions are output for user reference when the prediction result is ambiguous.

Model	Input Stream	Type	Backbone	f-mAP: JHMDB
TacNet	TWO-stream	Frame-level	VGG	0.66
T-CNN	RGB-stream	Frame-level	C3D	0.61
ACT	TWO-stream	Tubelet-level	VGG	0.66
MOC	RGB-stream	Tubelet-level	DLA34	0.71
TubeR	RGB-stream	Tubelet-transformer	CSN-152	0.72

Table 3: **Comparison on JHMDB-21** Compare the frame-mAP of TubeR model with other state-of-the-art model on JHMDB-21, IoU=0.5

in practice. In the JHMDB dataset, because all frames in a video share the same action class annotation, the annotation of some frames in the video is incorrect or ambiguous. For example, in the first few frames of the video labelled as stand, the action subject is actually sitting but is annotated as 'stand'. To avoid the wrong prediction given by the model due to the imprecise annotation of the dataset, I printed 3 actions with the highest probability(probability > 0.01) on the video and gave the classification score for user reference.

8.2 Limitation

Although this project has trained and deployed a TubeR model with good results on the JHMDB-21 dataset, it still has some possible limitations:

Limitation of Datasets AVA and JHMDB are already the most popular datasets, but they still have many problems that may affect the accuracy of the model prediction. The AVA dataset is not densely annotated, so there is no clear boundary for its actions and it is hard to track the action subject effectively. Although the JHMDB dataset is densely annotated, its huge amount of annotations leads to the low quality of some bounding boxes and only one action is labelled in one video, which leads to the inaccuracy of action category annotation.

Limitation of Long-term Context To ensure that the generated tubelet covers the video sequence as long as possible, the TubeR model needs to generate enough tubelet queries to cover the different actions of each person in the video to obtain the complete action features of the video. As the length of the input video sequence and the generated tubelet grows, the demand for computing resources increases considerably. Therefore, the TubeR model is difficult to realize the analysis of long-term context under the existing computing resources of the GPU cluster. One possible solution is to generate queries for action subjects instead of actions, which may effectively reduce the computational cost.

Lacking of computing resources The GPU cluster of the school has been equipped with 7 NVIDIA GeForce RTX 2080 Ti GPUs, but it is still insufficient for training the action detection model. In addition, the GPU cluster of the school will be disconnected after 18:00, and only the GPU can be used for training during the daytime. The process of processing data, training, and evaluating can take over a week at a time. Due to the deadline for the dissertation, this project did not have enough time to train a model with better performance by tuning the hyper parameters.

8.3 Contribution and Further Direction

The ultimate aim of this project is to deploy an end-to-end action detection system on assistive robots. The system needs to receive the video from the robot's camera, predict the action category and bounding box of the action subject in the video in real time, and track the action subject. This is a very complex task since it needs to consider not only the changing context of the environment caused by the camera rotation, but also the complex environment in the real world, the tracking of the human, the speed predicted by the model, and so on. Therefore, this project builds a spatio-temporal action detection model as a prototype of this end-to-end action detection system.

After a year of research and development, I finally completed the core part of my project and other all objectives. This project is a very successful attempt and exploration of spatio-temporal action detection. In this project, I have extensively studied the literature on action recognition and spatio-temporal action detection and ultimately chose to implement the TubeR model with SOTA performance. Then I used PyTorch to reconstruct the TubeR model and trained it on JHMDB-21 and AVA2.1 datasets using the school's GPU cluster. After the training process, I evaluate

the model by calculating the mAP of these two trained models with three different benchmarks. Among them, the performance of the model trained with the JHMDB-21 dataset not only reaches the SOTA level but also has more practical significance than the model trained with the AVA2.1 dataset. Finally, I visualized the predicted data of the JHMDB-21 dataset and designed a test program which could perform spatio-temporal action detection on the single input video.

Except for the reconstruction of the TubeR model, I also found that the output results of the TubeR model on the JHMDB-21 dataset were defective by analyzing the visualized data. On some specific actions, TubeR may give an ambiguous or wrong prediction. Therefore, I improved the way of giving prediction results to make the SOTA model–TubeR gives more accurate results and has stronger robustness, interpretability and generalization ability when deployed in practice.

Although I have successfully trained the TubeR model and made improvements to it, it is very necessary to continue to modify and optimize the TubeR model. Over the process of this year's development, I recommend further research into:

1. Find a better backbone algorithm to reduce the computational resources required to extract feature information in videos. Choosing the transformer as backbone might have led to better results for the model.

2. Using a method similar to the multi-meme algorithm to find a better transform method during the training process is a possible further direction.

3. In the TubeR model, all queries are initialized in the same way. It might be possible to tune the query initialization strategy to make the TubeR model perform better.

4. Modifying the multi attention layer of the model decoder to let the model track the action subject instead of the action, which can improve the utilization of tubelet and reduce the required computing resources.

9 Progress and Reflection

9.1 Project Management

Agile methodology is an approach to project management that this project decided to stick on. Each task is divided into several sprints which will finish in 2 weeks and frequent communication with the supervisor is required to allow for change direction and continuous improvement. One formal meeting was scheduled per week to report the progress and get feedback from the supervisor. Additionally, I also fill out the form weekly to record the progress. After the implementation of the agile methodology for this year, I follow it very well. I met with the supervisor almost every week to report the progress and discuss the future direction to track the progress.

Originally, the project consisted of two main sections, one was to design and train a traditional action recognition model and the other was to deploy the model on the robot. However, traditional action recognition algorithms need a lot of data pre-processing to extract video feature information. Therefore, in the original plan, I started the process of selecting datasets and data pre-processing after only one week of relevant research, which caused potential risks. Meanwhile, I will complete the training of the model before the exam week and deploy it on the robot within a month of the beginning of the second semester. Here is the Gantt chart of the original plan(Figure 23).

Reviewing the plan from the current perspective, there is no doubt that it is flawed and has many

potential risks. Since this is the first time for me to conduct an individual project, I am not experienced enough in many details. Here are two main potential risks in the initial plan.

Firstly, the initial plan is too ambitious. In the initial plan, I need to implement action recognition and action prediction then deploy it on the robot. This not only took time in coding but also a large number of model training tasks, which were very difficult to complete in two semesters. After several discussions with the supervisor, we finally determined that the main objective of the new project was to build an end-to-end spatio-temporal action detection system. Another risk is bad time allocation. I didn't attach much importance to the research of a project, so I only arranged one week of research time and spent the rest of the time implementing the project. This led me to start without a systematic understanding of the performance and implementation of my chosen method (IDT). With the progress of the project, I found that the performance of the traditional algorithm IDT could not meet the requirements of this project, so I had to replace the TubeR model and wasted 2 weeks due to the lack of preliminary research.

After reflection and adjustment, I get a new plan at the end of the first semester(Figure 24). In this plan, I mainly focused on the research of action recognition and related datasets in the first semester and implementing the project in the remaining time. Due to the change in the model of deep learning architecture, this project no longer needs a lot of time for data pre-processing, so I will focus more time on the model construction. To track the progress, I list all the components of TubeR as sprint instead of putting them into a chunk of time.

In the further research process, I found this plan also had the limitation. Since I had no prior experience building computer vision deep learning networks with PyTorch, I allocated 9 weeks to implement the model code and no time for training the model, which was very unreasonable. In the actual development process, it took me only 6 weeks to complete the implementation of the model. Besides, I had intended to implement the model training process on Google Colab. However, it is difficult to store and process the AVA dataset on Google Drive, and the GPU is out of memory when training the model. After reporting the situation to the supervisor, she applied for access permission for the GPU cluster for me. But the application process took almost a month, during this period I was only able to test my training code with a small batch dataset on Google Colab. It was mid-March when I had access to the GPU cluster, and I needed to complete the processing of the dataset, the training of the model, the evaluation of the model and the writing of the dissertation within a month, which was a very challenging task. Therefore, I need to evaluate the trained model when training a new model to make development as efficient as possible. Fortunately, there were no issues with the training code tested on Colab and I ended up completing all the milestones of the project. Here is the Gantt Chart of the actual project process(Figure 25).

Reviewing the process of the project, I overcame the difficulties and challenges encountered and finally completed all the objectives of this project. In the first semester, I conducted a literature review and evaluated a large number of spatio-temporal models. And the TubeR model with state-of-the-art performance is selected. After that, I collected a large number of video datasets including UCF-101, Charades, JHMDB-21 and AVA, and then analyzed and evaluated their action classes and annotations. Finally, JHMDB-21 and AVA with higher quality and more suitable for human-robot interaction scenarios are selected to train the model. During the Christmas break, I finished coding the model using PyTorch and started working on the small batch dataset to prove the concept of the model on Colab. In the second semester, I completed the training of the model on the GPU cluster and evaluated the trained model. In the end, I chose the TubeR model trained with the JHMDB-21 dataset and backbone algorithm CSN152, then implement the visualization of model prediction results and build the test program for the model.

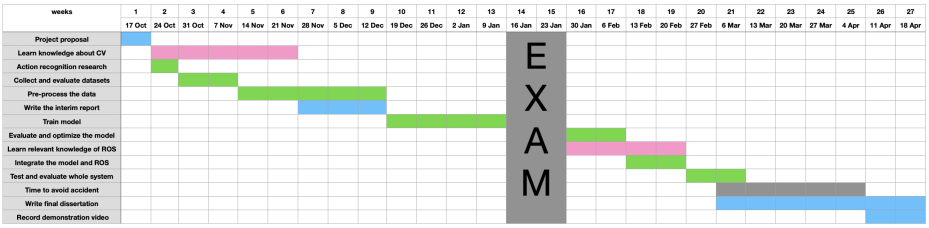

Figure 23: Plan in Proposal

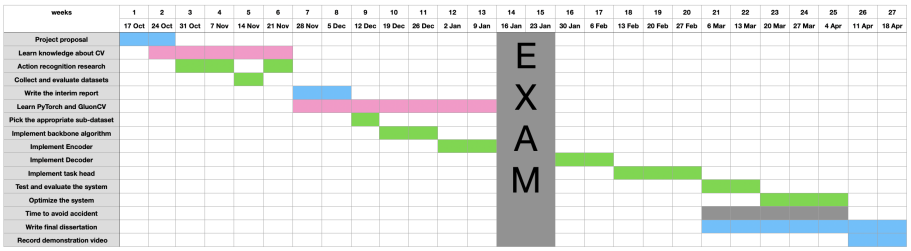

Figure 24: Plan in Interim Report

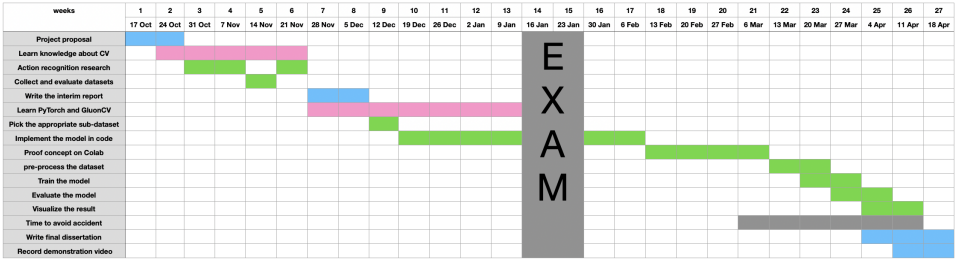

Figure 25: Final Plan

9.2 Resource Management

All source code and data are stored in the school's Linux GPU cluster server–Gerty in Cobot Maker Space. In addition, the log generated by each training and evaluation is also saved in the folder named with the current timestamp and stored on the GPU cluster. These files include checkpoint files to store model parameters, tensorboard log files to store loss curves during training, configuration files for evaluation, and predictions from the previous run. For safety purposes, I also save the source code to my pc and Google Drive, besides developing the project on the GPU cluster. Aside from that, I will wrap and document the entire code and upload it to the GitHub repository after the project is finished.

9.3 Project Reflection

This project is only a prototype of an end-to-end action detection system at the present stage and will not be commercially developed. After the project is finished, I will upload my code to the GitHub repository as open-source code. In the process of training the model, I used the open-source action video datasets AVA and JHMDB-21, which take people as the data subjects. In order to avoid ethics issues in the trained model, I submitted the ethics clearance form to the supervisor at the beginning of the project. The action detection system designed by this project will eventually be deployed to robots to assist in monitoring patient or elderly health. However, when this project is deployed in real life, there will inevitably be abuse. Therefore, I advocate the responsible use of this system and compliance with the corresponding laws and regulations.

9.4 Self-Reflection

Although the deadline for each milestone was met, several obstacles were encountered during the project. Since computer vision and deep learning were fields that I had never touched before, I made a lot of mistakes at the beginning of the project and didn't know where to start implementing the project. For example, I chose the backward traditional algorithm to implement action recognition at the beginning, which led to implementing nothing in the first semester. But as the project goes on, I read a lot of papers and accumulated practical experience in deep learning and action detection. I have the ability to deal with unexpected situations during the project. I was able to complete the data preprocessing, model training and model evaluation within one month because of a lot of previous research and the accumulation of relevant experience.

This project was stressful and challenging for me, but also rewarding. This project gives me a good opportunity to get in touch with the latest computer science research results and implement them. After this project, I have not only accumulated practical experience in deep learning but also gained a lot of theoretical knowledge in the field of action recognition. I also have a clear understanding of how to complete a personal project and the importance of a literature review to a project. In general, this project was an unforgettable experience in my life and had a significant impact on my future research direction.

References

[1] W. H. Organization, *World report on ageing and health*. World Health Organization, 2015.

[2] S. Coşar, M. Fernandez-Carmona, R. Agrigoroaie, F. Ferland, F. Zhao, S. Yue, N. Bellotto, A. Tapus, *et al.*, "Enrichme: Perception and interaction of an assistive robot for the elderly at home," *International Journal of Social Robotics*, vol. 12, no. 3, pp. 779–805, 2020.

[3] A. S. Prabuwono, K. H. S. Allehaibi, and K. Kurnianingsih, "Assistive robotic technology: a review," *Computer Engineering and Applications Journal*, vol. 6, no. 2, pp. 71–78, 2017.

[4] M. S. Ryoo, "Human activity prediction: Early recognition of ongoing activities from streaming videos," in *2011 International Conference on Computer Vision*, pp. 1036–1043, IEEE, 2011.

[5] M. Vrigkas, C. Nikou, and I. A. Kakadiaris, "A review of human activity recognition methods," *Frontiers in Robotics and AI*, vol. 2, p. 28, 2015.

[6] Y. Kong and Y. Fu, "Human action recognition and prediction: A survey," *International Journal of Computer Vision*, vol. 130, no. 5, pp. 1366–1401, 2022.

[7] J. Zhao, Y. Zhang, X. Li, H. Chen, B. Shuai, M. Xu, C. Liu, K. Kundu, Y. Xiong, D. Modolo, *et al.*, "Tuber: Tubelet transformer for video action detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13598–13607, 2022.

[8] C. Gu, C. Sun, D. A. Ross, C. Vondrick, C. Pantofaru, Y. Li, S. Vijayanarasimhan, G. Toderici, S. Ricco, R. Sukthankar, *et al.*, "Ava: A video dataset of spatio-temporally localized atomic visual actions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6047–6056, 2018.

[9] H. Jhuang, J. Gall, S. Zuffi, C. Schmid, and M. J. Black, "Towards understanding action recognition," in *International Conf. on Computer Vision (ICCV)*, pp. 3192–3199, Dec. 2013.

[10] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[11] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[12] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, pp. 213–229, Springer, 2020.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[14] D. Tran, H. Wang, L. Torresani, and M. Feiszli, "Video classification with channel-separated convolutional networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5552–5561, 2019.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[16] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu, "Dense trajectories and motion boundary descriptors for action recognition," *International journal of computer vision*, vol. 103, no. 1, pp. 60–79, 2013.

[17] H. Wang and C. Schmid, "Action recognition with improved trajectories," in *Proceedings of the IEEE international conference on computer vision*, pp. 3551–3558, 2013.

[18] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek, "Image classification with the fisher vector: Theory and practice," *International journal of computer vision*, vol. 105, pp. 222–245, 2013.

[19] 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*, vol. 35, no. 1, pp. 221–231, 2012.

[20] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[21] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," *Advances in neural information processing systems*, vol. 27, 2014.

[22] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Van Gool, "Temporal segment networks for action recognition in videos," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 11, pp. 2740–2755, 2018.

[23] G. Varol, I. Laptev, and C. Schmid, "Long-term temporal convolutions for action recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 6, pp. 1510–1517, 2017.

[24] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2625–2634, 2015.

[25] S. Sharma, R. Kiros, and R. Salakhutdinov, "Action recognition using visual attention," *arXiv preprint arXiv:1511.04119*, 2015.

[26] Z. Wu, X. Wang, Y.-G. Jiang, H. Ye, and X. Xue, "Modeling spatial-temporal clues in a hybrid deep learning framework for video classification," in *Proceedings of the 23rd ACM international conference on Multimedia*, pp. 461–470, 2015.

[27] X. Peng and C. Schmid, "Multi-region two-stream r-cnn for action detection," in *European conference on computer vision*, pp. 744–759, Springer, 2016.

[28] C. Sun, A. Shrivastava, C. Vondrick, K. Murphy, R. Sukthankar, and C. Schmid, "Actor-centric relation network," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 318–334, 2018.

[29] R. Hou, C. Chen, and M. Shah, "Tube convolutional neural network (t-cnn) for action detection in videos," in *Proceedings of the IEEE international conference on computer vision*, pp. 5822–5831, 2017.

[30] X. Yang, X. Yang, M.-Y. Liu, F. Xiao, L. S. Davis, and J. Kautz, "Step: Spatio-temporal progressive learning for video action detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 264–272, 2019.

[31] Y. Li, Z. Wang, L. Wang, and G. Wu, "Actions as moving points," in *European Conference on Computer Vision*, pp. 68–84, Springer, 2020.

[32] R. Girdhar, J. Carreira, C. Doersch, and A. Zisserman, "Video action transformer network," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 244–253, 2019.

[33] L. Gorelick, M. Blank, E. Shechtman, M. Irani, and R. Basri, "Actions as space-time shapes," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 12, pp. 2247–2253, 2007.

[34] D. Weinland, R. Ronfard, and E. Boyer, "Free viewpoint action recognition using motion history volumes," *Computer vision and image understanding*, vol. 104, no. 2-3, pp. 249–257, 2006.

[35] M. Marszalek, I. Laptev, and C. Schmid, "Actions in context," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2929–2936, IEEE, 2009.

[36] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: a local svm approach," in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, vol. 3, pp. 32–36, IEEE, 2004.

[37] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, "Hmdb: a large video database for human motion recognition," in *2011 International conference on computer vision*, pp. 2556–2563, IEEE, 2011.

[38] K. Soomro, A. R. Zamir, and M. Shah, "Ucf101: A dataset of 101 human actions classes from videos in the wild," *arXiv preprint arXiv:1212.0402*, 2012.

[39] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014.

[40] 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.

[41] F. C. Heilbron, V. Escorcia, B. Ghanem, and J. C. Niebles, "Activitynet: A large-scale video benchmark for human activity understanding," in *2015 IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 961–970, IEEE, 2015.

[42] G. A. Sigurdsson, A. Gupta, C. Schmid, A. Farhadi, and K. Alahari, "Charades-ego: A large-scale dataset of paired third and first person videos," *arXiv preprint arXiv:1804.09626*, 2018.

[43] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, *et al.*, "The kinetics human action video dataset," *arXiv preprint arXiv:1705.06950*, 2017.

[44] J. Carreira, E. Noland, A. Banki-Horvath, C. Hillier, and A. Zisserman, "A short note about kinetics-600," *arXiv preprint arXiv:1808.01340*, 2018.

[45] J. Carreira, E. Noland, C. Hillier, and A. Zisserman, "A short note on the kinetics-700 human action dataset," *arXiv preprint arXiv:1907.06987*, 2019.

[46] H. Zhao, A. Torralba, L. Torresani, and Z. Yan, "Hacs: Human action clips and segments dataset for recognition and temporal localization," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 8668–8678, 2019.

[47] Y. Li, L. Chen, R. He, Z. Wang, G. Wu, and L. Wang, "Multisports: A multi-person video dataset of spatio-temporally localized sports actions," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 13536–13545, 2021.

[48] A. Li, M. Thotakuri, D. A. Ross, J. Carreira, A. Vostrikov, and A. Zisserman, "The ava-kinetics localized human actions video dataset," *arXiv preprint arXiv:2005.00214*, 2020.

[49] L. Smaira, J. Carreira, E. Noland, E. Clancy, A. Wu, and A. Zisserman, "A short note on the kinetics-700-2020 human action dataset," *arXiv preprint arXiv:2010.10864*, 2020.

[50] J. Guo, H. He, T. He, L. Lausen, M. Li, H. Lin, X. Shi, C. Wang, J. Xie, S. Zha, A. Zhang, H. Zhang, Z. Zhang, Z. Zhang, S. Zheng, and Y. Zhu, "Gluoncv and gluonnlp: Deep learning in computer vision and natural language processing," *Journal of Machine Learning Research*, vol. 21, no. 23, pp. 1–7, 2020.

[51] M. Contributors, "Openmmlab's next generation video understanding toolbox and benchmark." https://github.com/open-mmlab/mmaction2, 2020.

[52] S. Tomar, "Converting video formats with ffmpeg," *Linux Journal*, vol. 2006, no. 146, p. 10, 2006.

[53] D. Ghadiyaram, D. Tran, and D. Mahajan, "Large-scale weakly-supervised pre-training for video action recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12046–12055, 2019.