UNIVERSITY OF TECHNOLOGY SYDNEY
Faculty of Engineering and Information Technology

From Video Classification to Video Prediction: Deep Learning Approaches to Video Modelling

by

Hehe Fan

Doctor of Philosophy

Sydney, Australia

2020

Certificate of Original Authorship

I, Hehe Fan declare that this thesis, is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise reference or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Signature: Production Note:
Signature removed prior to publication.

Date: 7 July 2020

Acknowledgements

I would like to thank my supervisor, Prof. Yi Yang, for the patient guidance, encouragement and advice he has provided throughout my time as his student. I have been extremely lucky to have a supervisor who cared so much about my work, and who responded to my questions and queries so promptly.

I would also like to thank all the members in our group and center who helped me in research and daily life. In particular I would like to thank Dr. Liang Zheng, Dr. Linchao Zhu and Dr. Xiaojun Chang for their suggestions they made for my research.

Hehe Fan

Sydney, Australia, 2020.

List of Publications

Journal Papers

J-1. **H. Fan**, L. Zheng, C. Yan and Y. Yang, "Unsupervised Person Re-identification: Clustering and Fine-tuning," *ACM Transactions on Multimedia Computing, Communications and Applications (ACM TOMM)*, vol. 14, no. 4, pp. 83:1-83:18, 2018.

J-2. **H. Fan**, L. Zhu, Y. Yang and F. Wu, "Recurrent Attention Network with Reinforced Generator for Visual Dialog," *ACM Transactions on Multimedia Computing, Communications and Applications (ACM TOMM)*, 2020.

J-3. Y. Ding, **H. Fan**, M. Xu and Y. Yang, "Adaptive Exploration for Unsupervised Person Re-Identification," *ACM Transactions on Multimedia Computing, Communications and Applications (ACM TOMM)*, vol. 16, no. 1, pp. 3:1-3:19, 2020.

J-4. Q. Feng, Y. Wu, **H. Fan** and Y. Yang, "Cascaded Revision Network for Novel Object Captioning," *IEEE Transactions on Circuits and Systems for Video Technology (IEEE TCSVT)*, 2020.

Conference Papers

C-1. **H. Fan**, X. Chang, D. Cheng, Y. Yang, D. Xu and A. G. Hauptmann, "Complex Event Detection by Identifying Reliable Shots from Untrimmed Videos," *IEEE International Conference on Computer Vision (ICCV)*, pp. 736-744, Oct. 22-29, 2017.

C-2. **H. Fan**, Z. Xu, L. Zhu, C. Yan, J. Ge and Y. Yang, "Watching a Small Portion could be as Good as Watching All: Towards Efficient Video Classification," *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 705-711, Jul. 13-19, 2018.

C-3. **H. Fan**, L. Zhu and Y. Yang, "Cubic LSTMs for Video Prediction," *Association for the Advancement of Artificial Intelligence (AAAI)*, pp. 8263-8270, Jan. 27-Feb. 1, 2019.

C-4. **H. Fan** and Y. Yang, "Person Tube Retrieval via Language Description," *Association for the Advancement of Artificial Intelligence (AAAI)*, pp. 10754-10761, Feb. 7-12, 2020.

C-5. Q. Feng, G. Kang, **H. Fan** and Y. Yang, "Attract or Distract: Exploit the Margin of Open Set," *IEEE International Conference on Computer Vision (ICCV)*, pp. 7990-7999, Oct. 27-Nov. 2, 2019.

Dedication

I dedicate my dissertation work to my parents, Mr. Yuqi Fan and Mrs. Huijuan Wu, who gave me strength when I thought of giving up and continually provided their moral, spiritual, emotional and financial support.

Contents

List of Figures

Abbreviation

SVM - Support Vector Machine.

CNN - Convolutional Neural Network

RNN - Recurrent Neural Network

LSTM - Long Short-Term Memory

GRU - Gated Recurrent Unit

1D - One-dimensional

2D - Two-dimensional

3D - Three-dimensional

RL - Reinforcement Learning

MIL - Multiple Instance Learning

Nomenclature and Notation

Bold capital letters denote matrices.

Bold lower-case alphabets denote column vectors.

$(.)^T$ denotes the transpose operation.

I_n is the identity matrix of dimension $n \times n$.

0_n is the zero matrix of dimension $n \times n$.

$\mathbb{R}$ denotes the field of real numbers.

ABSTRACT

From Video Classification to Video Prediction: Deep Learning Approaches to Video Modelling

by

Hehe Fan

Intelligent systems need the ability to understand not only the static scenes around them but also the dynamic changes in the environment. In order to understand the dynamic changes, intelligent systems are expected to model spatio-temporal sequences, *i.e.*, videos. Learning video representations and predicting future movements constitute two fundamental missions of video modelling.

This dissertation presents two works on video classification. In the first work, based on video shot representations, which are extracted by convolutional neural networks, a selective multi-instance learning method is proposed to automatically detect whether an event of interest happens in temporally untrimmed videos which usually consist of multiple video shots. This work can be seen as a binary video classification problem. The second work focuses on efficiency, which aims to reduce the computational cost for classification of temporally untrimmed videos while retaining similar accuracy. With deep reinforcement learning, a fast forward network is devised to skip most of uninformative frames, and an adaptive stop network is incorporated to generate timely trigger to stop the agent watching videos. The proposed method enables agents to classify videos by watching a very small portion of frames.

This dissertation also presents two works on video prediction. The core of video prediction involves moving object capture and future motion prediction. While object capture specifies which objects are moving in videos, motion prediction describes their future dynamics. Motivated by this analysis, a Cubic Long Short-Term

Memory (CubicLSTM) unit is proposed by splitting internal states into a spatial state and a temporal state, which can better capture the spatio-temporal structure for video prediction. In addition to conventional 2D RGB videos, prediction is also conducted on moving 3D point clouds. Compared to RGB videos, point cloud videos provide accurate displacement measurements, which are generally unaffected by lighting conditions. Prediction based on 3D point cloud videos provides more flexibilities for future action planning in poor visibility environments, and covers more precise geometry dynamics than the conventional RGB video prediction. To extend recurrent neural network for point cloud video prediction, a series of point spatio-temporal correlation recurrent neural networks, including PointRNN, PointGRU and PointLSTM, are proposed to jointly aggregate point coordinates and features in input, states and output, which enable point to interact in a unified way. Because point cloud videos exhibit unorderedness in the spatial dimension and emerge inconsistently across different frames, a point spatio-temporal correlation operation is used to dynamically model point relations during input-to-state and state-to-state transitions.

Dissertation directed by Professor Yi Yang

Chapter 1

Introduction

This dissertation presents two video classification works, *i.e.*, complex event detection in untrimmed videos, which can be seen as a binary video classification problem, and efficient video classification, and two video prediction works, *i.e.*, conventional 2D RGB video prediction and 3D point cloud video prediction. This chapter introduces the background of video classification and video prediction, respectively, research objectives and thesis organization.

1.1 Background

Since 2012, with the great success of AlexNet [4] on image classification, deep neural networks, especially Convolutional Neural Networks (CNNs), *e.g.*, VGG [5], GoogLeNet [6], Inception v3 [7], ResNet [8], SqueezeNet [9], Wide ResNet [10], DenseNet [11], and ResNeXt [12], and Recurrent Neural Networks (RNNs), *e.g.*, LSTM [13] and GRU [14], have been widely used in computer vision [15, 16, 17, 18, 19, 20, 21], natural language processing [22, 23] and speech recognition [24, 25]. Generally speaking, CNN has become the dominant method for image processing and RNN is well suitable for time series data processing. Because a video is essentially a sequence of images, deep learning approaches are also widely used for video processing.

1.1.1 Video Classification

Large progress has been made in generating compact and discriminative representations for video classification. However, unlike image processing, there is not

a dominant method for video encoding. Basically, the deep learning methods for video classification can be divided in to the following categories.

Two-stream 2D CNN. Two-stream convolutional neural networks [26] exploits a spatial stream and a temporal stream to capture the complementary information on appearance from still frames and motion between frames, respectively. To be specific, given a video, the spatial stream uses a 2D CNN to extract the appearance feature from a RGB frame in the video, while the temporal stream takes stacked optical flow (the distribution of apparent velocities of movement of brightness pattern in an image) frames as inputs and uses another 2D CNN to extract the motion feature. The appearance feature and the motion feature are then used for classification. This method achieves comparable performance to tradition methods and has motivated many researchers to use two-stream convolution neural networks for action recognition [27, 28].

3D CNN. C3D networks [29] directly use 3D convolutional network to learn spatio-temporal representations for short video clips. Then, 3D convolution based neural networks become another family for action recognition [30, 31, 32, 33]. In the work [31], 3D convolution is decomposed in to a 2D convolution to model spatial information and a 1D convolution to model temporal information, respectively. In this dissertation, this (2+1)D convolution is considered as a variant of 3D convolution.

2D CNN + RNN. Because video is a kind of temporal sequence, based 2D CNN frame features, recurrent neural networks are also applied into video analysis [34, 35]. For example, Ng et al. [36] used an LSTM to aggregate frame-level CNN features for modelling video temporal sequences. Yeung et al. [37] proposed a recurrent neural network-based agent for action detection. Zhu et al. [38] proposed an unsupervised temporal modeling method that learns from untrimmed videos by encoding frames of a clip with different intervals.

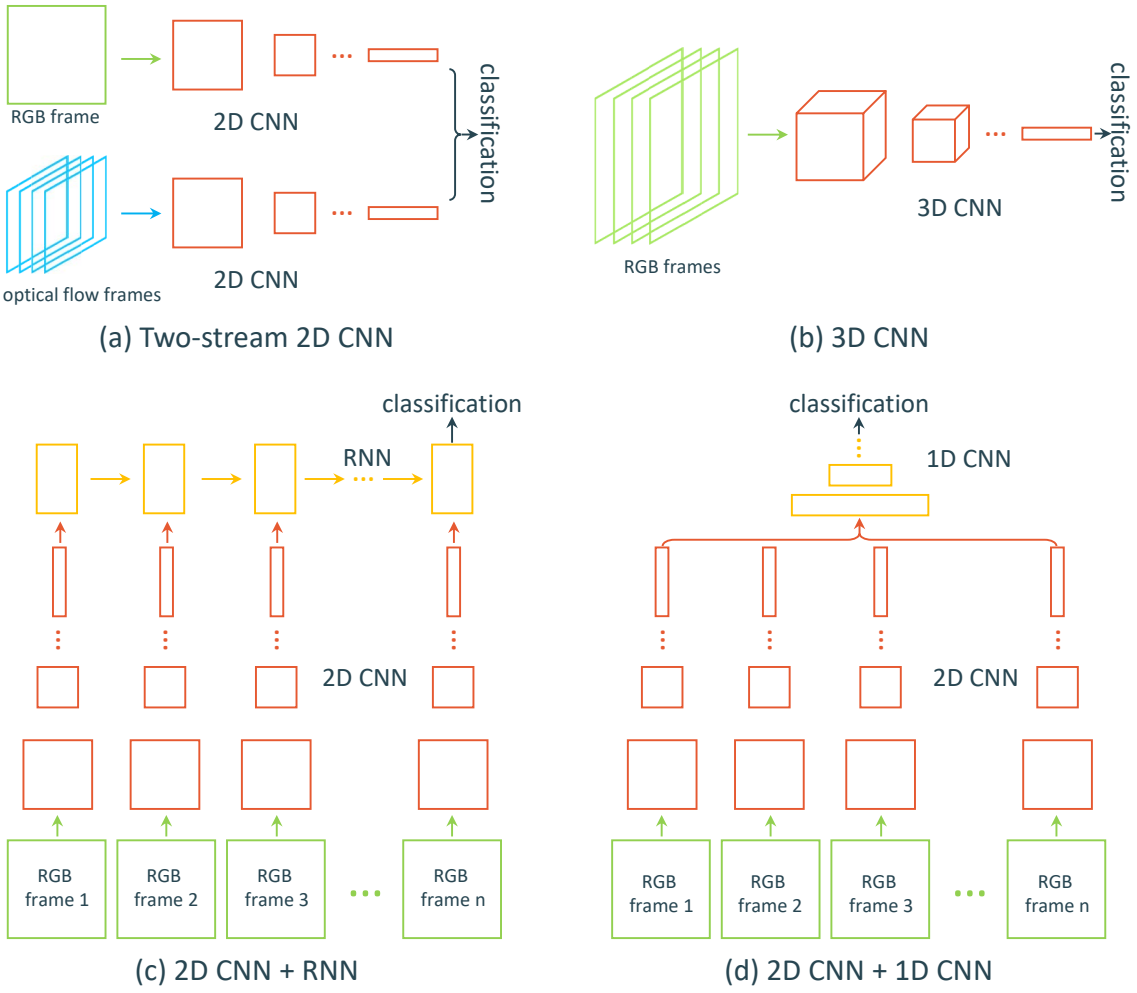


Figure 1.1 : Illustrations of the for deep learning techniques for video classification. The goal of these methods is to encode a video to a feature vector, which is then used for classification.

2D CNN + 1D CNN. Similar to 2D CNN + RNN, 1D convolution neural networks [39, 40] can also be used to encode frame features for generating a global video representation. which is referred to as 2D CNN + 1D CNN in this dissertation.

In summary, two-stream 2D CNN, 3D CNN, 2D CNN + RNN and 2D CNN + 1D constitute the basic techniques to embed videos for classification. Illustrations of these methods are shown in Figure 1.1. Sometimes, two-stream 2D CNN and 3D CNN are combined to further enhance video representation learning [30]. Besides,

the simple **2D CNN + Pooling** method, which first uses CNN to extract frame-level features and then employs max pooling or average pooling to aggregate frame-level features to a video-level feature, is also used to extract video features when motion information is not quite important for some scenarios.

This dissertation does not introduce new basic techniques for video representation learning, but aims to address another two problems in video classification. The first problem focuses on the noise from context or irrelevant frames in untrimmed videos, which should be removed to improve classification accuracy. The second problem focuses on video classification efficiency. Usually, all video frames are exploited during classification, which consumes a large amount of time and computation. To reduce frames to be processed, uninformative frames should be skipped and it is not necessary to watch the entire video to make a classification decision. In the first work, 2D CNN + Pooling is used to extract shot representation for complex event detection in untrimmed videos. In the second work, 2D CNN + RNN is adopted as the backbone for efficient video classification.

1.1.2 Video Prediction

Predicting future frames in videos has become a promising direction of research in both computer vision and robot learning communities. Different from video classification, which focuses on sequence-level video modelling, video prediction aims to encode and decode videos in the pixel or point level. Generally, suppose we observe a sequence of m frames $\hat{\boldsymbol{X}}_{t-m+1}, \hat{\boldsymbol{X}}_{t-m+1}, \ldots, \hat{\boldsymbol{X}}_{t}$. The video prediction problem is to predict the most likely length-n sequence in the future given the previous m observations, which can be formulated as follows,

$$\tilde{\boldsymbol{X}}_{t+1}, \ldots, \tilde{\boldsymbol{X}}_{t+n} = \underset{\boldsymbol{X}_{t+1}, \ldots, \boldsymbol{X}_{t+n}}{\arg\max}\ p(\boldsymbol{X}_{t+1}, \ldots, \boldsymbol{X}_{t+n} | \hat{\boldsymbol{X}}_{t-m+1}, \ldots, \hat{\boldsymbol{X}}_{t}). \tag{1.1}$$

An example of video prediction is illustrated in Figure 1.2. Compared to video classification, because the future frames are self-contained in a video, video prediction

has an innate advantage that it usually does not require external human-annotated supervised information. A number of existing works have addressed video prediction with different settings. They can essentially be classified as follows.

Generation vs. Transformation. Video prediction can be classified as patch level [41], feature level [41, 42, 43] and image level. For image-level prediction, the generation group of methods generates each pixel in frames [1, 44]. Some of these methods have difficulty in handling real-world video prediction due to the curse of dimension. The second group first predicts a transformation and then applies the transformation to the previous frame to generate a new frame [45, 46, 3, 47]. These types of methods can reduce the difficulty of predicting real-world future frames. In addition, some methods [48, 49, 50] first decompose a video into a stationary content part and a temporally varying motion component by multiple loss functions, and then combine the predicted motion and stationary content to construct future frames.

Short-term prediction vs. Long-term prediction. Video prediction can be classified as short-term ($<$ 10 frames) prediction [51, 52] and long-term ($\geq$ 10 frames) prediction [53, 1, 54, 3, 55] according to the number of predicted frames. Most methods can usually undertake long-term predictions on virtual-word datasets (*e.g.*, game-play video datasets [53]) or synthetic datasets (*e.g.*, the Moving-MNIST dataset [42]). Since real-world sequences are less deterministic, some of these methods are limited to making long-term predictions. For example, dynamic filter network [45] is able to predict ten frames on the Moving-MNIST dataset but three frames on the highway driving dataset. A special case of short-term prediction is the so-called next-frame prediction [51, 56], which only predicts one frame in the future.

Single dependency vs. Multiple dependencies. Most methods need to

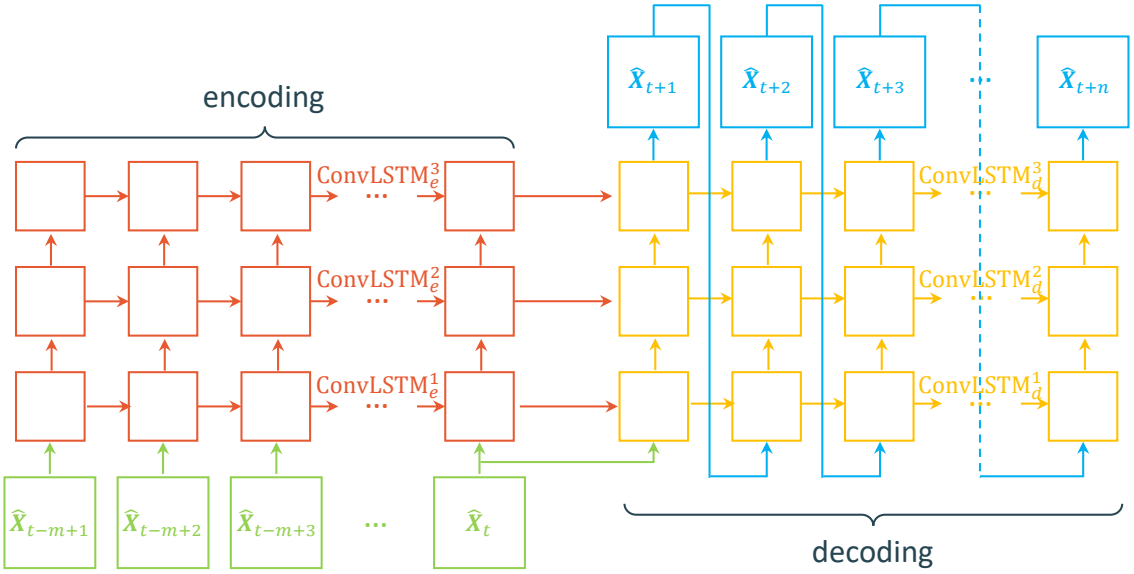


Figure 1.2 : Illustration of a stacked ConvLSTM [1] sequence-to-sequence [2] model for video prediction. The model consists of a encoding part and a decoding part. Each part includes three ConvLSTMs. Given m observations, the encoding part first embeds the sequence to three states, which are then used to initialize the three states of decoding ConvLSTMs, respectively. Last, the decoding part produces n predictions.

observe multiple context frames before making video predictions. Other methods, by contrast, aim to predict the future based only on the understanding of a single image [57, 58, 59, 51, 60, 61].

Unconditional prediction vs. Conditional prediction. The computer vision community usually focuses on unconditional prediction, in which the future only depends on the video itself. The action-conditional prediction has been widely explored in the robotic learning community, *e.g.*, video game videos [53, 62] and robotic manipulations [3, 52, 44].

Deterministic model vs. Probabilistic model. One common assumption for video prediction is that the future is deterministic. Most video prediction meth-

ods therefore belong to the deterministic model. However, the real-world can be full of stochastic dynamics. The probabilistic model predicts multiple possible frames at one time [63, 64]. The probabilistic method is also adopted in single dependency prediction because the precise motion corresponding to a single image is often stochastic and ambiguous [51].

This dissertation focuses on short-term, multiple-dependency, unconditional video prediction with deterministic models. For conventional RGB video prediction, the proposed method can be seen as a generation fashion because the future frames are directly generated. For point cloud video prediction, the transformation-based method is used, in which the 3D scene flow is first predicted and then is added to the previous point cloud frame to generate the new point cloud frame.

1.2 Research Objectives

The aims of the project are as follows:

i. In order to improve accuracy for complex event detection, conduct studies of removing noise from unrelated shots by identifying reliable shots from untrimmed videos with CNN representation.

ii. Motivated by humans who classify videos by watching a very small portion of frames, conduct studies of efficient video classification via deep reinforcement learning based fast forward and adaptive stop mechanisms.

iii. In order to reduce the burden of spatio-temporal modeling for conventional 2D RGB video prediction, conduct studies of splitting internal states into a temporal state and a spatial state.

iv. In order to extend video prediction to moving 3D point clouds, develop a series of point spatio-temporal correlation recurrent neural networks to model point cloud videos.

1.3 Thesis Organization

This thesis is organised as follows:

- *Chapter 2*: This chapter presents a survey of complex event detection, efficient video classification, conventional 2D RGB video prediction and 3D point cloud prediction.

- *Chapter 3*: Based on CNN representation, a multi-instance learning method for complex event detection in untrimmed videos is derived in this chapter. This work has been published at the International Conference on Computer Vision 2017 [65].

- *Chapter 4*: This chapter presents a deep reinforcement learning method for efficient video classification. This work has been been published at the International Joint Conference on Artificial Intelligence 2018 [66].

- *Chapter 5*: A cubic LSTM for conventional 2D RGB video prediction is described in this chapter. This work has been been published at the Association for the Advancement of Artificial Intelligence 2019 [67].

- *Chapter 6*: This chapter introduces a PointRNN, PointLSTM and PointGRU for 3D point cloud video prediction.

- *Chapter 7*: A brief summary of the thesis contents and its contributions are given in the final chapter. Recommendation for future works is given as well.

Chapter 2

Literature Survey

This chapter provides a literature survey about complex event detection in untrimmed videos, efficient video classification, conventional 2D RGB video prediction and 3D moving point cloud prediction.

2.1 Complex Event Detection in Untrimmed Videos

Complex event detection aims to automatically detect an event of interest in temporally untrimmed long videos, which can be seen as a kind of video classification task. Unlike human action recognition [26, 36, 29] where actions are usually well-defined and videos are constrained, complex event detection is more challenging where the target event is complex and may only occur within a short period of time in a video.

Existing works [65, 68] propose to use different strategies to exploit temporal information for complex event detection. By decomposing the complex event like "grooming a dog" as a set of actions like "prepare water", "walk to dog", "groom dog", "leave dog" and "clean up", Tang et al. [65] adopted a Hidden Markov Model based approach to model the complex event. Cheng et al. [68] used a sequence memorizer approach to exploit temporal information by treating each video shot as an individual visual word. However, the performance is still not quite satisfactory because it is a non-trivial task to effectively exploit temporal information in long videos like those from MEDTest-13 and MEDTest-14. Meanwhile, researchers also explored information from Wikipedia to provide more detailed explanation and

definition for each complex event. Chang et al. [69] proposed to first prioritize the shots according to the salience scores from some pretrained concept detectors, and then used a SVM based method to exploit ordering information of semantic concepts. Yan et al. [70] proposed a dictionary learning approach for complex event detection, in which the dictionary consists of a set of selected meaningful concepts for all events. However, these approaches heavily depend on human knowledge in order to design the elementary concepts for each complex event. It remains unclear what and how many concept detectors should be pretrained for arbitrary events. Li et al. [71] proposed a dynamic pooling method based on sliding windows. However, this sliding window based approach is computationally expensive. Moreover, the work in [72] partitioned each untrimmed long video as a set of video shots and used the multi-instance learning method $\propto$-SVM [73] for complex event detection. The major limitation of this work is the assumption that almost all video shots in positive bags are positive and all video shots in negative bags are negative, which is overstrict in the real-world complex event detection applications.

2.2 Efficient Video Classification

The accuracy of video classification task has been significantly improved, befitting from the recent advances of Convolutional Neural Networks (CNNs) [4, 8] and Recurrent Neural Networks (RNNs) [13]. While a lot of attention has been paid to the accuracy aspect of video classification, very few existing works have tried to enhance another crucial aspect: efficiency.

A common procedure for video classification is first extracting frame features by CNNs and then aggregating the features into a single representation by pooling methods or RNNs. While pooling methods ignore the order of frames and temporal information in videos, RNNs such as Long Short-Term Memory (LSTM) [13] model video temporal structure by encoding the frame features sequentially. Both

approaches have shown promising results on video classification [36, 26]. However, extracting frame features by CNNs for large-scale video classification is computationally expensive because the amount of computation scales linearly with the video length, which makes it difficult to deploy current video classification algorithms into web-scale applications.

The solutions of dealing with the efficiency of video processing can form two different axises which are orthogonal to each other, *i.e.*, the time to process one frame and the number of processed frames. For the first axis, there has been a vivid community to lower the computation of CNNs [74] or to build specific hardwares to accelerate the CNNs computation time [75]. The efficiency of feature extraction can directly benefit from this line of research. For the second axis, the solution is to lower the number of frames the algorithm needs to process, which is more inherent to the video classification task. This dissertation will show that the second solution has a great potential to improve video classification efficiency.

2.3 Conventional 2D RGB Video Prediction

Videos contain a large amount of visual information in scenes as well as profound dynamic changes in motions. Understanding scenes, learning video representations and imagining motions and are fundamental missions in computer vision. It is indisputable that a model, which is able to predict future frames after watching several context frames, has the ability to achieve these missions [42]. As video is a kind of spatio-temporal sequences, recurrent neural networks (RNNs), especially Long Short-Term Memory (LSTM) [13] and Gated Recurrent Unit (GRU) [76], have been widely applied to video prediction.

One of the earliest RNN models for video prediction [41] directly borrows a structure from the language modeling literature [77]. It quantizes the space of frame patches as visual words and therefore can be seen as patch-level prediction. Later

work [42] builds encoder-decoder predictors using the fully connected LSTM. In addition to patch-level prediction, it also learns to predict future frames at the feature level. Since the traditional LSTM (or GRU) units learn from one-dimensional vectors where the feature representation is highly compact and the spatial information is lost, both of these methods attempt to avoid directly predicting future video frames at the image level.

To predict spatio-temporal sequences, convolutional LSTM (ConvLSTM) [1] modifies LSTM by taking three-dimensional tensors as the input and replacing fully connected layers by convolutional operations. ConvLSTM has become an important component in several video prediction works [3, 55, 63, 56]. However, directly adapting LSTM makes it difficult for ConvLSTM to simultaneously exploit the temporal information and the spatial information in videos. The convolutional operation in ConvLSTM has to process motions on one hand and capture moving objects on the other hand. Similarly, the state of ConvLSTM must be able to carry the motion information and object information at the same time. It could be insufficient to use only one convolution and one state for spatio-temporal prediction.

2.4 3D Point Cloud Video Prediction

Most modern robot and self-driving car platforms rely on 3D point clouds for geometry perception. Compared to 2D images, 3D point clouds provide accurate displacement measurements, which are generally unaffected by lighting conditions.

In the general setting, a point cloud is a set of points in 3D space. Usually, the point cloud is represented by the three coordinates $\boldsymbol{P} \in \mathbb{R}^{n\times 3}$ of points and their features $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ (if features are provided), where n and d denote the number of points and feature channels, respectively. Essentially, point clouds are unordered sets and invariant to permutations of their points. For example, the set $\{(1,1,1),(2,2,2),(3,3,3)\}$ and $\{(3,3,3),(1,1,1),(2,2,2)\}$ represent the same point

cloud. This irregular format data structure considerably increases the challenge to reason point clouds, making many existing achievements of deep neural networks on image and video fail to directly process point clouds.

However, most of the existing works focus on static point cloud analysis, *e.g.*, classification, segmentation and detection [78, 79, 80, 81]. Few works study point cloud video prediction. Following the conventional 2D RGB video prediction, we can try to employ RNNs and their variants to model point cloud videos. However, they have a severe limitation on processing point cloud sequences. To be specific, RNNs and their variants aggregate the previous state and the current input based on a concatenation operation. However, because point cloud sequences exhibit unorderedness in the spatial dimension and emerge inconsistently across different frames, concatenation can not be directly applied to point cloud frames.

Chapter 3

Selective Multi-Instance Learning for Complex Event Detection in Untrimmed Videos with CNN Representation

3.1 Introduction

This chapter presents a multi-instance learning method for complex event detection in untrimmed videos. Complex event detection aims to automatically detect an event of interest in temporally untrimmed long videos, which can be seen as a kind of binary video classification task. Because only video-level labels are given and untrimmed videos usually contain a number of unrelated context shots or frames, the proposed method treats complex event detection as a weakly supervised learning problem. To be specific, each untrimmed video can be partitioned as a set of video shots. For each event class, it is observed that some video shots in positive (resp. negative) videos are irrelevant (resp. relevant) to this event. This problem becomes even more severe for long videos. To address the unreliability issue, in this chapter, the complex event detection task is formulated as a multi-instance learning (MIL) [82, 83, 84, 85, 73, 86, 87, 88] problem by treating each video as a bag and the video shots in the video as instances. In contrast to the existing MIL methods that directly infer instance labels, a new multi-instance learning method is developed for complex event detection by identifying reliable instances from positive and negative bags, rather than predicting instance labels. The new objective function simultaneously learns a linear SVM classifier and infers a binary indicator for each instance, in which the indicator indicates whether this instance is selected as a reliable instance.

3.2 Method

The framework for complex event detection is shown in Figure 3.1. After partitioning each untrimmed long video as a set of video shots, the CNN features are extracted from a set of uniformly sampled key-frames in each video shot. The frame features are then merged as a single feature vector to represent the video shot by using pooling methods. Next, the shot labels are initialized as the corresponding video label and the linear SVM classifier is Single Instance Learning SVM (SIL-SVM). Then the framework selects reliable shots to learn and update the classifier, until no more shots are added into the reliable shot set.

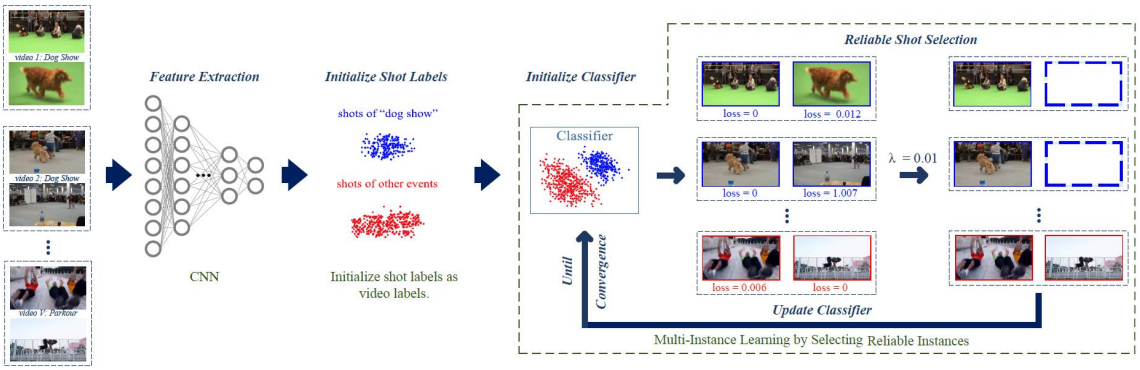


Figure 3.1 : Illustration of the training process of the detector for the event "dog show" by using the proposed MIL-SRI method.

Three types of instances are involved in untrimmed videos:

- Reliable instances. Positive (resp. negative) instances in positive (resp. negative) bags.
- Ambiguous instances. Instances that are difficult to be defined or classified as positive or negative instances.
- Noisy instances. Positive (resp. negative) instances in negative (resp. positive) bags.

The three types of shots in videos are illustrated in Figure 3.2 (a).

3.2.1 Problem Formulation

Suppose we have V videos and $\boldsymbol{x}_v^i$ is the feature vector of the i-th shot in the v-th video. In the proposed method, each shot is considered as an "instance" and each video is considered as a "bag". A positive or negative bag $\mathcal{B}_v$ is associated with a bag label $Y_v \in \{\pm 1\}$ and the instance-level labels are unknown. It is assumed that the reliability ratio is known and not less than P_v, which is defined as the proportion of positive instances in the positive bag or the proportion of negative instances in the negative bag. The aim of complex event detection is to learn a linear SVM classifier for each event class, only based on video-level labels.

3.2.2 The MIL-SRI Model

According to the above settings, complex event detection can be transformed into binary classification and solved by multi-instance learning. In traditional multi-instance learning methods, one step is to infer instance labels, which means that instance-level labels are updated in training iterations. However, it is non-trivial to infer the instances' labels. The proposed method does not infer instance-level labels but train the SVM classifier by inferring a set of reliable instances.

This work assumes the decision function is in the form of $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$. Suppose q_v^i denotes the selection indicator of instance $\boldsymbol{x}_v^i$. If q_v^i equals to 1, $\boldsymbol{x}_v^i$ is selected as a reliable instance; otherwise, the shot is discarded when training the SVM classier. Suppose $\boldsymbol{q}_v = [q_v^1, ..., q_v^{|\mathcal{B}_v|}]$ denotes the selection indication vector for the v-th video and $|\mathcal{B}_v|$ stands for the cardinality of bag $\mathcal{B}_v$. The proposed method

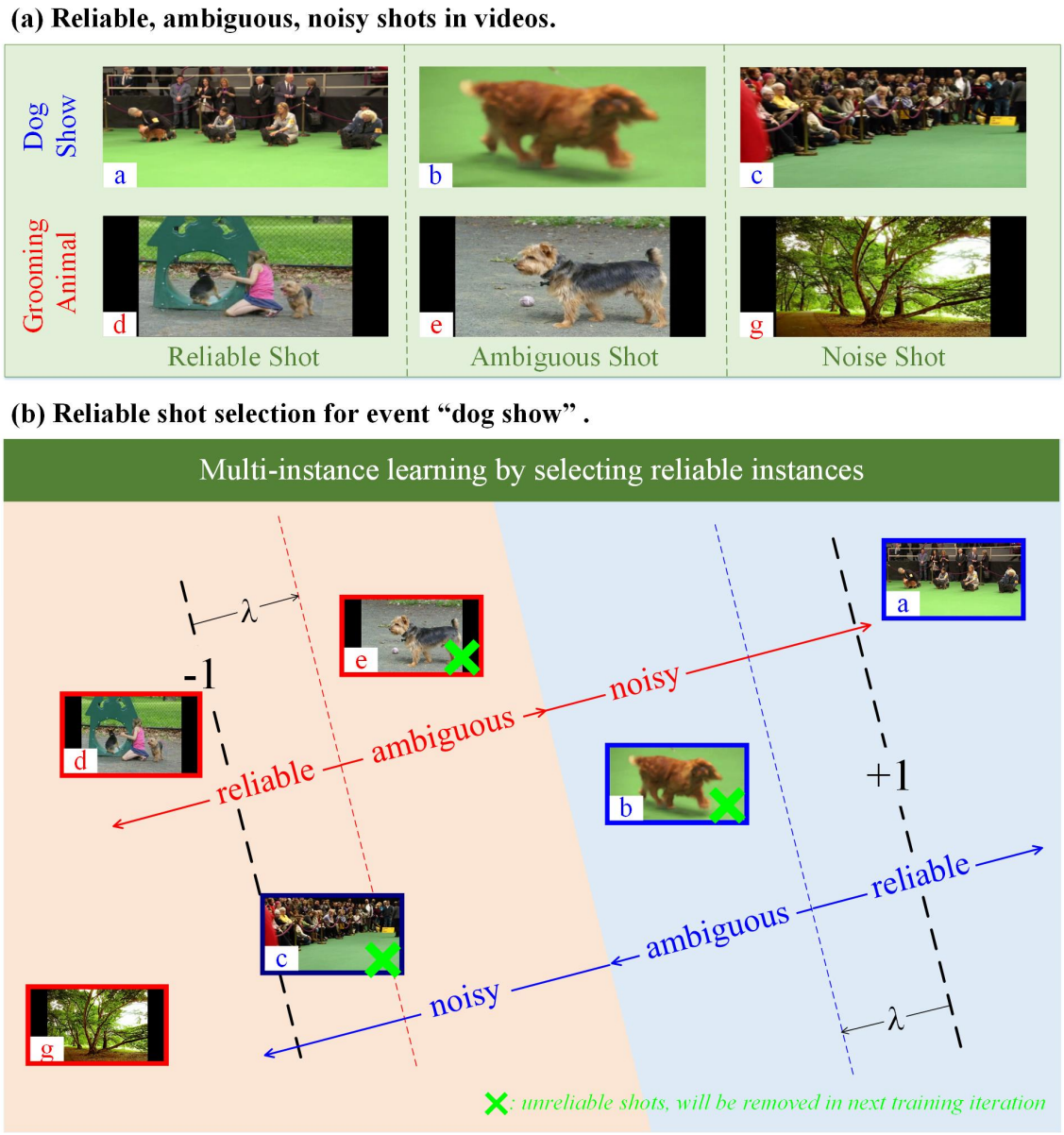


Figure 3.2 : Illustration of three types of shots in videos and the basic principle of the proposed MIL-SRI method. Besides dogs and their hosts, audience usually appears in the event "dog show" when compared with the event "grooming animal". A shot only with a dog is ambiguous in both events. From the perspective of SVM, shots with $loss \leq \lambda$ are reliable shots, while those with $\lambda < loss \leq 1$ are ambiguous shots, and shots with $1 < loss$ will be treated as noisy shots. These ambiguous and noisy shots should not be used when training the SVM classifier.

can be formulated as the following optimization problem,

$$\min_{\boldsymbol{w},b,\boldsymbol{q}_v,\boldsymbol{\varepsilon}_v} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{v=1}^{V}\left(\boldsymbol{q_v}^T\boldsymbol{\varepsilon_v} - \lambda\|\boldsymbol{q_v}\|_1 - \gamma\|\boldsymbol{q_v}\|_2\right)$$
$$s.t., \begin{cases} Y_v(\boldsymbol{w}^T\boldsymbol{x}_v^i + b) \geq 1 - \varepsilon_v^i; \\ \varepsilon_v^i \geq 0; \\ \|\boldsymbol{q_v}\|_1 \geq P_v|\mathcal{B}_v|; \\ q_v^i \in \{0,1\}. \end{cases} , v = 1, 2, \cdots, V, \tag{3.1}$$

where $C > 0$ is the parameter that controls the relative importance of detection correctness, $\boldsymbol{\varepsilon}_v = [\varepsilon_v^1, ..., \varepsilon_v^{|\mathcal{B}_v|}]$ is the slack variable vector of the v-th video and λ, γ are two non-negative parameters. The first term $\frac{1}{2}\|\boldsymbol{w}\|^2$ controls the complexity of the proposed model to avoid overfitting. The second term aims to minimize the training errors from a set of selected confident training instances. To minimize the second term, we need to set all $\boldsymbol{q}_v$ as zeros, which tends not to select any training instances as confident ones. By adding the negative sign in front of the l_1-l_2 mixed-norm regularizer ($\lambda\|\boldsymbol{q_v}\|_1 + \gamma\|\boldsymbol{q_v}\|_2$), we need to set all $\boldsymbol{q}_v$ as ones in order to minimize this -l_1-l_2 mixed-norm regularizer, which encourages all training instances to be selected as training instances. By balancing the training errors and the l_1-l_2 mixed-norm regularizer in Eq. (3.1), we can select the most reliable training instances (shots) by using the proposed approach. Next, I describe how the mixed-norm regularizer and constraints influence the behavior of multi-instance learning.

•$\boldsymbol{\lambda \neq 0, \gamma = 0}$: When $\lambda > 0$, the instance whose loss is less than λ is discarded in the iterative process. The parameter λ is a trade-off to remove unreliable instances. Meanwhile, when the training errors are becoming smaller during the iterative process, more selection indicators $\{q_v^i\}$ are expected to become 1 in order to minimize the negative l_1-l_2 mixed-norm regularization term. As a result, more and more instances are added into the reliable instance set, until no more reliable

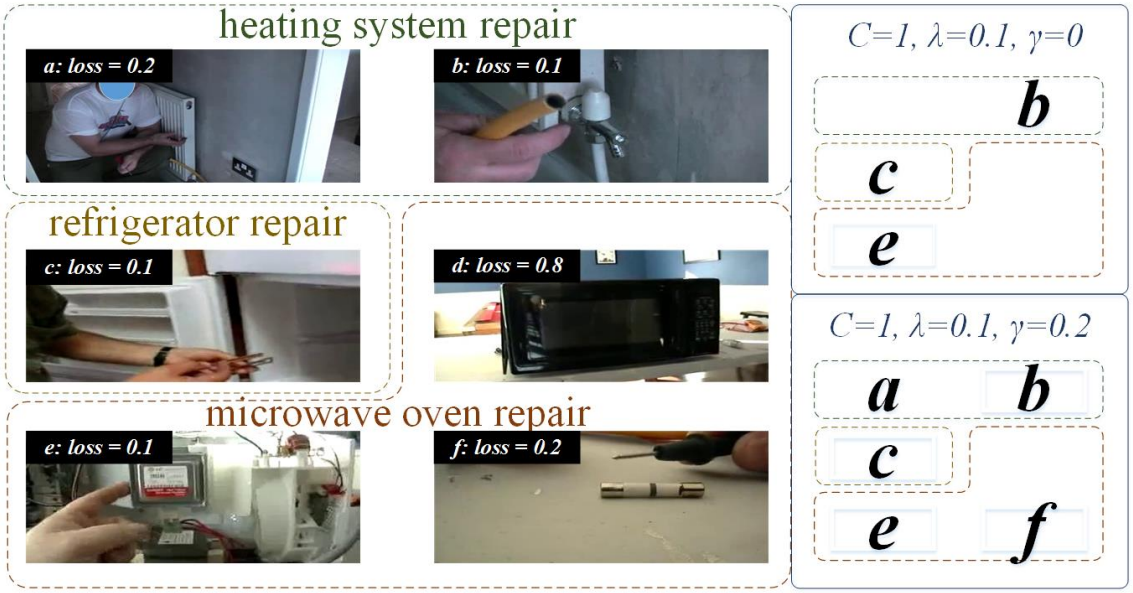


Figure 3.3 : Effect of the l_2 norm for the event "repairing appliance". In this example, the regularizer $\frac{1}{2}\|\boldsymbol{w}\|^2$ is ignored and $C = 1$, $P_v = 1/|\mathcal{B}_v|$. When setting $\lambda = 0.1, \gamma = 0$, the algorithm tends to select shots $\{b,\ c,\ e\}$ in order to achieve the minimal loss 0 while the constraint $\|\boldsymbol{q_v}\|_1 \geq P_v|\mathcal{B}_v|$ is satisfied. When setting $\lambda = 0.1, \gamma = 0.2$, the algorithm selects shots $\{a,\ b,\ c,\ e,\ f\}$ as it can lead to the minimal loss -0.4, which is smaller than -0.346 when selecting $\{b,\ c,\ e\}$. Therefore, the proposed algorithm can select the shots from a diverse set of videos when we additionally consider the l_2-norm based regularizer.

instances can be added. Figure 3.2 illustrates how to select shots for event "dog show" according to λ.

•$\boldsymbol{\lambda \neq 0, \gamma \neq 0}$: The P_v has been defined as the lower bound for the percentage of reliable shots in each video, which can guarantee that at least $P_v|\mathcal{B}_v|$ shots will be selected from each video when learning the SVM classifier. This constraint can conduct selection diversity to some extent, however, we aim to flexibly select more reliable instances from different videos by utilizing an additional l_2-norm regularization term. Figure 3.3 illustrates that more reliable instances can be selected from a

diverse set of bags after using this l_2-norm regularizer.

•$\|\boldsymbol{q_v}\|_1 \geq P_v|\mathcal{B}_v|$: With this constraint, we will choose at least a certain percentage of reliable instances from each bag. This can avoid a special case that all instances are inferred as unreliable instances, which may lead to an incorrect hyperplane. In the case of $P_v = 1/|\mathcal{B}_v|$ if $Y_v = +1$ and $P_v = 1$ if $Y_v = -1$, the new constraint will become the constraint in many existing multi-instance learning methods [82], namely, there is at least one positive instance in each positive bag and all instances in a negative bag are negative.

3.2.3 Optimization Strategy

To solve MIL-SRI, the involved parameters $(\boldsymbol{w}, b)$ and $\{\boldsymbol{q}_v\}_{v=1}^V$ are alternately optimized. Let $L(\cdot) \geq 0$ be a classic loss function for supervised learning, *e.g.*, hinge loss.

• Optimize $(\boldsymbol{w}, b)$ when $\{\boldsymbol{q}_v\}_{v=1}^V$ is fixed: In this step, the optimal problem degenerates to the classic weighted SVM problem [89]:

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{v=1}^{V}\sum_{i=1}^{|\mathcal{B}_v|} q_v^i L(Y_v, \boldsymbol{w}^T\boldsymbol{x}_v^i + b), \tag{3.2}$$

which can be easily solved. Suppose $|\mathcal{B}|$ is the average number of shots per video, and d is the number of the feature dimension. Since I aim to learn a linear SVM classifier, the computational complexity of this step is $\mathcal{O}(\max(V|\mathcal{B}|, d)\min(V|\mathcal{B}|, d)^2)$.

• Optimize $\{\boldsymbol{q}_v\}_{v=1}^V$ when $(\boldsymbol{w}, b)$ is fixed: The goal of this step is to select the instances according to the reliability and diversity. The optimization problem becomes the following:

$$\begin{aligned} \min_{\boldsymbol{q}_v} \sum_{v=1}^{V}\Big(\sum_{i=1}^{|\mathcal{B}_v|} q_v^i L(Y_v, \boldsymbol{w}^T\boldsymbol{x}_v^i + b) - \lambda\|\boldsymbol{q_v}\|_1 - \gamma\|\boldsymbol{q_v}\|_2\Big) \\ s.t.\ \|\boldsymbol{q_v}\|_1 \geq P_v|\mathcal{B}_v|;\ q_v^i \in \{0,1\}; v = 1, 2, ..., V \end{aligned} \tag{3.3}$$

This optimal problem can be solved by [90] and the details are provided in Alg

Algorithm 1: Optimization Procedure

Input : Instances $\{\{\boldsymbol{x}_v^i\}\}$; Video-level labels $\{Y_v\}$; Reliability ratio $\{P_v\}$; Video length $\{|\mathcal{B}_v|\}$; Parameters C, λ, γ.

Output: Classifier $(\boldsymbol{w}, b)$.

Initialization: $q_v^i \leftarrow 1$.

1 **while** *not convergence* **do**
2 **optimize** $(\boldsymbol{w}, b)$ when $\boldsymbol{q}_v$ is fixed;
3 **for** $v = 1$ *to* V **do**
4 calculate loss $\delta_i = L(Y_v, \boldsymbol{w}^T \boldsymbol{x}_v^i + b)$;
5 sort $\{\delta_i\}$ in ascending order $\rightarrow \{\delta'_j\}$;
6 denote $\tau(j)$ as $\delta'_j = \delta_{\tau(j)}$;
7 **for** $j = 1$ *to* $|\mathcal{B}_v|$ **do**
8 **if** $\delta'_j < \lambda + \frac{\gamma}{\sqrt{j}+\sqrt{j-1}}$ ***or*** $j \leq P_v|\mathcal{B}_v|$ **then**
9 $q_v^{\tau(j)} \leftarrow 1$;
10 **else**
11 $q_v^{\tau(j)} \leftarrow 0$;
12 **end**
13 **end**
14 **end**
15 **end**

1 (see step 3 to step 14). Specifically, the video shots with $j \leq P_v|\mathcal{B}_v|$ or $\delta'_j < \lambda + \gamma/(\sqrt{j}+\sqrt{j-1})$ will be selected into the training process, where j is the instance rank w.r.t. its loss value within its video. Since the threshold $\lambda + \gamma/(\sqrt{j}+\sqrt{j-1})$ decreases as the rank j grows for each video, this optimization strategy penalizes the shots that are monotonously selected from the same video and thus naturally conducts diversity. The computational complexity of loss calculation is $\mathcal{O}(V|\mathcal{B}|d)$

and the sort operation costs $\mathcal{O}(V|\mathcal{B}|log|\mathcal{B}|)$. Therefore, the computation complexity of this step is $\mathcal{O}(V|\mathcal{B}|d)$ because we usually have $d >> log|\mathcal{B}|$.

3.3 Experiments

In this section, extensive experiments are conducted to validate the proposed method on two real-world datasets. The parameter C is set to 1 in all experiments. Training is stopped when the difference of $\sum_{v=1}^{V}\|\boldsymbol{q_v}\|_1$ between two iterations is less than 100.

Datasets: Following recent works on Multimedia Event Detection (MED) [69, 70], the MIL-SRI method is tested on two real-world event detection datasets.

- MEDTest-13: Similar to MEDTest 2014 dataset. Note that 10 of 20 events overlap with those in MEDTest-14. The detailed event description can be found at *.
- MEDTest-14: The TRECVID MEDTest 2014 dataset contains approximately 100 positive training examples per event, and all events share the same (∼5000) negative training exemplars. The testing set has approximately 23,000 videos. There are 20 events in total, whose detailed descriptions can be found at †.

For the two MED datasets, I detect each event separately according to the NIST standard. Both 100Ex and 10Ex settings provided by NIST are considered, which have 100 and 10 positive training exemplars respectively.

Feature extraction: Each video is split into multiple video shots using the color histogram difference as the indication of the shot boundary. For simplicity,

*http://nist.gov/itl/iad/mig/med13.cfm

†http://nist.gov/itl/iad/mig/med14.cfm

the center frame is chosen from each shot, resized it to 224×224 and extracted the frame feature from the fc6 layer of VGG16 [5].

Evaluation metric: According to the NIST standard, mean Average Precision (mAP) is used as evaluation metric. Average precision (AP) is a single-valued metric approximating the area under the precision-recall curve, which is widely used in information retrieval tasks. Mean Average Precision is the mean of AP over all event classes.

Comparison with the baseline algorithms

In this section, the baselines and a Recurrent Neural Network (RNN) based method, *i.e.*, CNN-LSTM, are evaluated.

In CNN-LSTM, one frame is sampled per second from the video and then extract the same CNN feature for each frame. Afterwards, an LSTM takes CNN features as inputs and generates an output at each step. I average-pool the output activations and conduct a 21-way classification (20 categories and 1 background category). The LSTM cell size is set to 1,024.

In the SMIL-TopK method, the parameter k is decided by using cross-validation from the range $\{0.1, 0.2, ..., 0.9\}$. In the MIL-SRI method, both λ and γ are decided by using cross-validation from the range of $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^{2}\}$. The percentage P_v is set to 0.1 and 0.5 for positive and negative videos, respectively. Following the TRECVID protocol, a classifier is trained for each event. The experimental results are reported in Table 3.1.

From the experimental results, we can see that sparse MIL and SIL-SVM methods achieve lower performances. The CNN-LSTM method can be treated as one type of pooling operation and it achieves better performance. Both SMIL-TopK and MIL-SRI attain higher performances, which demonstrates the importance of

Table 3.1 : mAP(%) comparison of baselines on the TRECVID MEDTest-14 and MEDTest-13 datasets.

	MEDTest-14		MEDTest-13	
	100Ex	10Ex	100Ex	10Ex
Sparse MIL	19.8	14.3	23.3	16.4
SIL-SVM	22.2	18.5	24.0	19.8
CNN-LSTM	31.1	20.4	33.8	21.7
SMIL-TopK	36.9	26.2	40.5	26.9
MIL-SRI (Ours)	**38.6**	**28.4**	**43.1**	**28.7**

using reliable shots.

Comparison with models using a single feature

In this section, the proposed method is compared with a few recent state-of-the-art methods that use a **single** type of feature. The experimental results are reported in Table 3.2. Following [91], a Spatial Temporal Network (STN) is pre-trained by using the Sports-1M dataset [91]. Then, the top three layers are fine-tuned. Additionally, a pre-trained C3D network [29] is fine-tuned for feature extraction.

From the experimental results, we can observe that the proposed method compares favorably against the other methods. Note that previous methods can only achieve the best performance on some datasets, *e.g.*, Xu et al.[96] achieved the state-of-the-art results on MEDTest-13 100Ex while Chang et al.[69] obtained the best performance on MEDTest-13 10Ex. However, my method consistently achieves the state-of-the-art performance on both MEDTest-13 and MEDTest-14 datasets. Specifically, the proposed method outperforms the state-of-the-art methods by 2.9%

Table 3.2 : mAP(%) comparison between our MIL-SRI method and the state-of-the-art baselines using a **single** type of feature on the TRECVID MEDTest-13 and MEDTest-14

(a) MEDTest-13

Method	100Ex	10Ex
Video Story [92]	32.0	19.6
$\propto$-SVM [72]	33.4	21.6
ESR [93]	36.2	20.1
C3D [29]	36.9	22.2
MIFS [94]	36.9	19.3
STN [91]	37.1	20.4
StMM + TP [95]	38.6	21.8
CNN + VLAD [96]	40.3	25.6
NI-SVM [69]	39.2	26.8
MIL-SRI (Ours)	**43.1**	**28.7**

(b) MEDTest-14

Method	100Ex	10Ex
DP [71]	28.8	17.6
$\propto$-SVM [72]	28.6	17.4
ESR [93]	29.6	18.4
CNN-Exp [97]	29.7	–
STN [91]	30.4	19.8
C3D [29]	31.4	20.5
MIFS [94]	29.0	14.9
CNN + VLAD [96]	35.7	23.2
NI-SVM [69]	34.4	26.1
MIL-SRI (Ours)	**38.6**	**28.4**

(2.3%) on MEDTest-14 100Ex (10Ex) and 2.8% (1.9%) on MEDTest-13 100Ex (10Ex), which is a large improvement for both datasets.

To demonstrate how the proposed method works in complex event detection, a few of examples of shot selection after the second training iteration in shown Fig 3.4. As can be seen, the MIL-SRI method is able to distinguish the three types of shots, *i.e.*, reliable shots, ambiguous shots and noisy shots.

Comparison with the state-of-the-art systems

In this section, the proposed method is compared with some state-of-the-art systems that combine **multiple** types of features. Improved Dense Trajectories

Figure 3.4 : Visualization of shot selection results. There are three types of shots, *i.e.*, reliable shots, ambiguous shots and noisy shots. In ambiguous shots, we can observe **semantic ambiguity** and **visual ambiguity**. Specifically, semantically ambiguous shots have similar semantic meaning to the target event or contain a few of elementary concepts about it. In visually ambiguous shots, the event actually happens but can not be detected evidently due to dim weather, camera angles or distances. These ambiguous shots lead to non-discriminative event features and MIL-SRL removes them from the training set. The figures are best viewed in color.

(IDT) [98] have dominated complex event detection in the past few years due to their excellent performance over other features. The prediction results are fused from VGG16 and IDT by averaging classification scores.

The experimental results are reported in Table 3.3, from which we observe that the proposed method performs competitively against all the existing methods. After fusing the IDT feature, mAP can be improved from 28.4% to 29.6% on MEDTest-14 10Ex and even more improvement is obtained on MEDTest-13 100Ex (from 43.1% to 49.7%). It shows the benefit of fusing motion features in complex event detection.

Xu et al. [96] fused multiple features from three different layers of VGG16, *i.e.*, pool5, fc6 and fc7. The proposed method consistently outperforms [96] in both MEDTest-13 and MEDTest-14 datasets with about 5% absolute improvement. The proposed also outperforms the previous state-of-the-art system [69] by more than 3%

Table 3.3 : mAP (%) comparison against state-of-the-art systems that fuse **multiple** types of features on the TRECVID MEDTest-14 and MEDTest-13 datasets.

	MEDTest-14		MEDTest-13	
	100Ex	10Ex	100Ex	10Ex
C3D [29] + IDT	33.6	22.1	39.5	26.7
CNN-Exp [97]	38.7	–	–	–
CNN + VLAD [96]	36.8	24.5	44.6	29.8
NI-SVM + IDT [69]	38.1	27.2	46.3	31.5
MIL-SRI + IDT	**41.5**	**29.6**	**49.7**	**34.6**

absolute improvement on MEDTest-13 dataset. The MIL-SRI method thus achieves the new state-of-the-art result on both MEDTest-13 and MEDTest-14 datasets.

3.4 Summary

To avoid that irrelevant shots may mislead event detection in untrimmed videos, an MIL-SRI is proposed to distinguish reliable and unreliable shots based solely on video-level labels. Compared to existing MIL methods, MIL-SRI relies on a more reasonable assumption that both reliable and unreliable shots can occur in all videos. To reduce noise form irrelevant shots, MIL-SRI learns SVM classifiers only by reliable shots and formulates complex event detection as a multi-instance learning problem. To be specific, MIL-SRI can adaptively select reliable instances and do not need to infer instance label. By this method, the state-of-the-art performance in complex event detection is attained on two real-world datasets.

Chapter 4

Deep Reinforcement Learning for Efficient Video Classification

4.1 Introduction

This chapter presents a reinforcement learning method for efficient video classification, which enables an agent to classify videos by watching a very small portion of frames like what humans do.

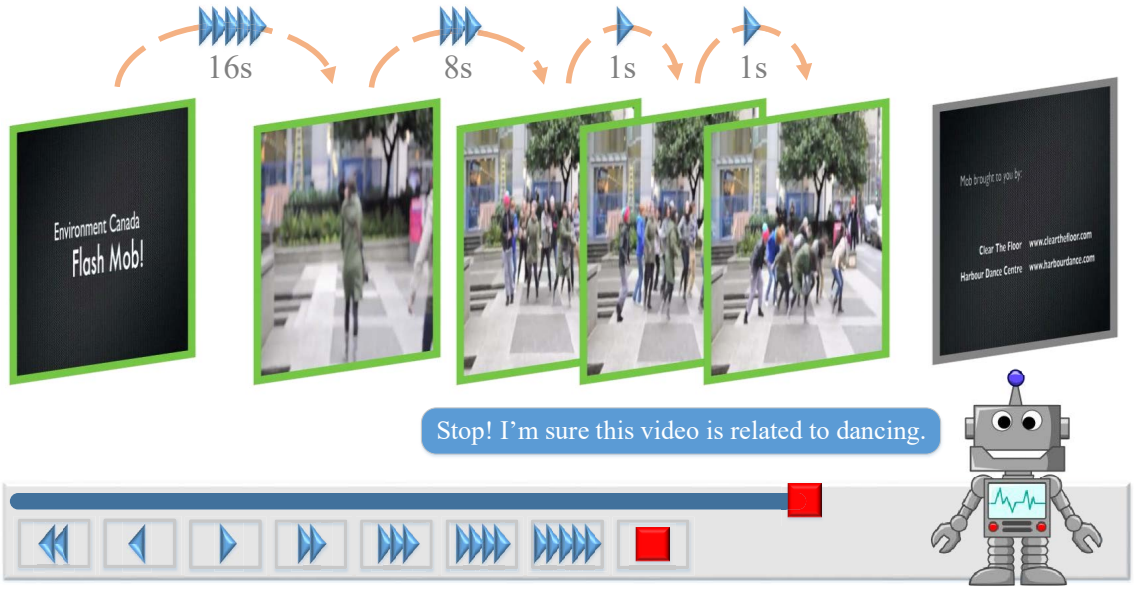


Figure 4.1 : A demonstration for the "fast forward" and "adaptive stop" mechanisms. At each time step, the agent chooses a skip interval from seven "fast forward" actions. After collecting enough information to make decision, the agent stops watching the video and emits the classification decision.

To be specific, first, whenever the agent finds the current frame does not add any helpful information to classify the video, it can play the video in a faster speed.

When finding some informative frames, the agent can slow down the watching speed and look into details. This mechanism is referred to as "fast forward". Specifically, a series of discrete time intervals are predefined. The agent decides where to watch at the next time step by selecting a time interval to skip forward. As shown in Figure 4.1, when classifying category like "dancing", the agent would prefer to select a longer time interval, *e.g.*, $16s$, to skip after watching some video introduction frames. When the agent catch a potential dancing frame, it would slow down and select a shorter forward time, *e.g.*, $1s$, to get more details. Second, the agent is capable of measuring confidence score to make decision. When confident enough, the agent can stop watching the video, which is referred to as "adaptive stop" mechanism. The "fast forward" and "adaptive stop" mechanisms lead to a Partially observable Markov decision process (POMDP) problem in Reinforcement Learning literature, which can be trained with REINFORCE algorithms [99] though the decision processes are not differentiable.

4.2 Method

The proposed method is formulated into the reinforcement learning paradigm, which is about an agent interacting with an environment, and learning an optimal policy, by trial and error, for sequence decision making. The "fast forward" and "adaptive stop" mechanisms are considered as two sequential decision processes of a goal-driven agent interacting with a video player. At each time step, the agent is limited to observe only one frame from a video by a pre-trained CNN. However, the agent is capable of controlling which frame to be watched at the next time step or stop watching the video to emit classification decision at the current time step. The agent acts based on the current frame and history to watch the video efficiently. At the final step, the agent receives a scalar reward which depends on whether the classification decision is correct and the number of watched frames. The goal of the

agent is to maximize such rewards.

4.2.1 Architecture

The agent is built around a core network as shown in Figure 4.2. At each time step t, the agent takes a frame feature i_t as input, integrates history information h_{t-1}, and determines where to watch a_t at the next time step (fast forward network) or whether to stop watching c_t (adaptive stop network) to make classification at the current step. When the agent decides to stop, a classification network is applied on the agent's internal state h_t and emits the prediction p. A baseline network is also added to predict the expected return b_t according to the current internal state, which is used to reduce variance during training [99].

Core network: The core network maintains an internal state which summarizes watching history information about the past frames; it encodes the agent's knowledge of the video and provides the information to guide the agent how to act at the next time step. The internal state h_t at time t is formed by a recurrent neural network f_h, which is parameterized by θ_h and updated over time with taking the external frame feature vector i_t as input $h_t = f_h(h_{t-1}, i_t; \theta_h)$. The internal state h_t plays a core role in the proposed approach. The fast forward action, adaptive stop action, baseline and classification decision are all based on this internal state.

Fast forward network: After observing a frame, the agent determine where to watch at the next time step by choosing an action a_t from a fast forward action space $\mathcal{A}$. In this chapter, the fast forward action space consists of seven discrete actions:

$$\mathcal{A} = \{-2s, -1s, +1s, +2s, +4s, +8s, +16s\} \tag{4.1}$$

where '+' means forward and '−' means rewind. The motivation of this definition is that videos have local patterns, which can be learned and used to predict what will happen or appear within a few seconds. It is premised that, exceeding 16 seconds,

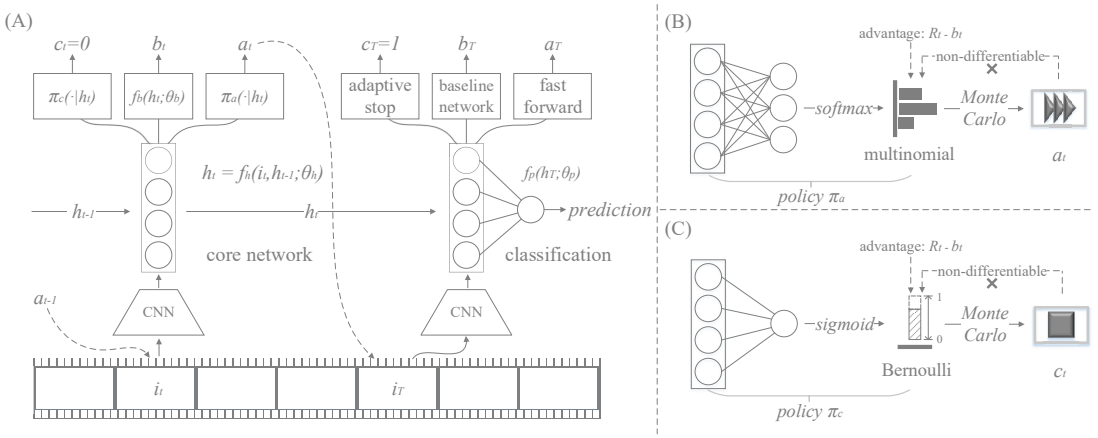


Figure 4.2 : (A) Architecture: The agent is built around a recurrent neural network. At each time step, it takes the frame feature as input, integrates history information (internal state), and determine how to act (forward or stop) at the next time step. The classification network generates the prediction based on the internal state. Additionally, a baseline network is added to reduce variance during training. (B) Fast forward network: Provide the policy π_a that maps the internal state to the fast forward action space $\mathcal{A}$, *e.g.*, $\{-2s, -1s, +1s, +2s, +4s, +8s, +16s\}$. (C) Adaptive stop network: Provide the policy π_c that maps the internal state to the adaptive stop action space $\mathcal{C}$, *i.e.*, $\{0, 1\}$, where 0 represents 'continue' and 1 represents 'stop'.

the pattern is too weak to be exploited. Therefore, the agent is limited to forward $16s$ at most at each time step. Sometimes, the agent may forward too much and miss information. To rescue this case, two rewind actions are added. However, the agent is not allowed to rewind too much because this would damage the efficiency. The agent should learn how to avoid over forward.

The fast forward network is implemented by a fully connected layer which is applied on h_t, parameterized by θ_a, and followed by a softmax. This network outputs a multinomial probability distribution, *i.e.*, the policy $\pi_a(\cdot|h_t; \theta_a)$, which represents the probability of each action in $\mathcal{A}$ should be adopted. As shown in Figure 4.2(B), a_t is sampled from π_a during training $a_t \sim \pi_a(\cdot|h_t; \theta_a)$. During evaluation, maximum a

posteriori estimation is used to choose forward action, *i.e.*, $a_t = \text{argmax}_a\ \pi_a(\cdot|h_t;\theta_a)$.

Adaptive stop network: Adaptive stop is the other action which can be controlled by the agent. This action decides when to stop watching videos to begin classification. Unlike a_t, the adaptive stop action c_t is a binary variable whose range is defined as:

$$\mathcal{C} = \{continue, stop\} \tag{4.2}$$

In this chapter, 0 to represents "continue" and 1 represents "stop". The adaptive stop network is a fully connected layer parameterized by θ_c and followed by a sigmoid, which takes h_t as input and outputs a Bernoulli probability distribution, *i.e.*, the policy $\pi_c(\cdot|h_t;\theta_c)$, representing the probability of stop. As shown in Figure 4.2(B), c_t is sampled from the π_c during training $c_t \sim \pi_c(\cdot|h_t;\theta_c)$

Baseline network: To reduce variance during training, a baseline network is employed to establish a baseline b_t to encourage (if $R_t > b_t$) or discourage (if $R_t < b_t$) both the fast forward action a_t and adaptive stop action c_t, where R_t is the return (cumulative discounted reward) at the time step t with a discount factor $\gamma \in (0, 1]$: $R_t = \sum_{k=0}^{T-t} \gamma^k r_{k+t}$. The baseline network is a linear regressor parameterized by θ_b and takes the core network's state h_t as input $b_t = f_b(h_t;\theta_b)$. The quantity $R_t - b_t$, which is also called advantage, will be used to scale the policy gradients.

Classification network: When the adaptive stop is triggered, the classification network f_p is applied on the final internal state h_T. I consider a binary classification setting and train the agent for every class. Suppose the positive video is labeled as 1 and the negative video is labeled as 0, then f_p outputs the likelihood p of the video to be positive. A fully connected layer is used as the classification network, which parameterized by θ_p and followed by a sigmoid to predict the video label $p = f_p(h_T;\theta_p)$, where T is the final time step which satisfies $c_T = 1$.

Reward function: After classification, the agent receives a reward signal based on

whether the prediction is correct and the number of watched frames. The goal of the agent is to maximize this reward. The reward function is defined as follows,

$$r_t = \begin{cases} 1 - 2|p - g| - \mu T, & \text{for } t = T; \\ 0, & \text{for } t < T, \end{cases} \tag{4.3}$$

where g denotes the ground truth of the video label. When $t = T$, the reward function consists of two parts. The first one $1 - 2|p - g|$ is called accuracy reward, which depends on the correctness of prediction. Since $p \in [0, 1]$ and $g \in \{0, 1\}$, this part maps the accuracy reward to $[-1, +1]$. The second part $-\mu T$ is called frame penalty which is used to encourage the agent to watch less frames.

4.2.2 Training

The parameters of the agent consists of the parameters from the core network, fast forward network, adaptive stop network, baseline network and classification network, *i.e.*, $\theta = \{\theta_h, \theta_a, \theta_c, \theta_b, \theta_p\}$. The standard backpropagation is used to train f_v, f_p and the REINFORCE is used to train π_a, π_c. The parameter of the core network θ_h is not directly optimized but can be updated by the gradients from the other four networks.

Classification loss: In this work, every agent focuses only on one specific video class. The output of the agent is a scalar variable with range $[0, 1]$, which indicates the confidence score of a video example being positive. Therefore, the classification network parameter θ_p is optimized by minimizing the binary cross entropy loss,

$$B(\theta_p, \theta_h) = \log\, p + (1 - g)\log(1 - p) \tag{4.4}$$

Baseline loss: The baseline network f_b is trained by minimizing the mean squared error loss:

$$M(\theta_b, \theta_h) = 1/T \sum\nolimits_{t=1}^{T} ||R_t - b_t||^2 \tag{4.5}$$

At each training iteration, the fast forward network and adaptive stop network are first updated by the reward and the baseline. The baseline network is then optimized by Eq. (4.5). Note that this loss does not backpropagate to the fast forward network or the adaptive stop network.

Policy gradients: This section derives the expressions of the parameter gradients for the proposed method. Recall that the method takes two actions, *i.e.*, fast forward action a_t and adaptive stop action c_t at each time step. The objective of the agent is to maximize the reward r_T achieved at the final step T:

$$J(\theta_a, \theta_c, \theta_h) = \mathop{\mathbb{E}}_{\pi_a, \pi_c} r_T \tag{4.6}$$

Note that $r_t = 0$ for $t < T$ according to the reward function Eq. (4.3). The gradient of θ_a can be derived as follows:

$$\nabla_{\theta_a} J = \mathop{\mathbb{E}}_{\pi_a, \pi_c} \nabla_{\theta_a} \log\, \pi_a(a_t|h_t) R_t \tag{4.7}$$

As we can see from Eq. (4.7), the gradient $\nabla_{\theta_a} J$ is directly proportional to R_t. Updating the parameter directly according to R_t will make training very unstable. To reduce the variance of the gradient estimate, the baseline b_t is subtracted from R_t:

$$\nabla_{\theta_a} J = \mathop{\mathbb{E}}_{\pi_a, \pi_c} \nabla_{\theta_a} \log\, \pi_a(a_t|h_t)(R_t - b_t) \tag{4.8}$$

This is an unbiased estimation to the gradient because $\mathop{\mathbb{E}}_{\pi_a, \pi_c} \nabla_{\theta_a} \log\, \pi_a(a_t|h_t) b_t = b_t \nabla_{\theta_a} 1 = 0$. Similarly, gradient w.r.t. θ_c, *i.e.*, $\nabla_{\theta_c} J$ can be derived as follows:

$$\nabla_{\theta_c} J = \mathop{\mathbb{E}}_{\pi_a, \pi_c} \nabla_{\theta_c} \log\, \pi_c(c_t|h_t)(R_t - b_t) \tag{4.9}$$

Since the dimension of the possible action sequence can be very high, it is impossible to optimize Eq. (4.8) and Eq. (4.9) directly. Following REINFORCE [99], Monte Carlo sampling is used to approximate the policy gradients.

Encourage exploration: To avoid highly-peaked π_a towards a few fast forward actions or a few fixed action sequences, which over-optimizes on a small portion of the environment, *i.e.*, video in this work, an entropy term is used to encourage diversity [100]:

$$H(\theta_a, \theta_h) = -\frac{1}{T}\sum_{t=1}^{T} \mathop{\mathbb{E}}_{a\sim\mathcal{A}} \pi_a(a|h_t)\log\pi_a(a|h_t) \tag{4.10}$$

When all actions almost have the same probability, the entropy will be high. Conversely, when one action has near 1 probability, the entropy will be low. Therefore, adding a negative entropy term to the loss function will penalize the actions that dominate the policy too quickly and encourage exploration.

Finally, the proposed objective is to minimize the following hybrid loss function:

$$L(\theta) = \alpha B + \beta M - \lambda J - \rho H \tag{4.11}$$

where α, β, λ and ρ are positive factors to control the weights of different sub-losses.

4.3 Experiments

4.3.1 Datasets and Settings

Since reinforcement learning usually needs large-scale datasets, the proposed method is evaluated on the **YouTube-8M** [101] dataset, which is a large-scale video classification benchmark. YouTube-8M consists of ∼8 million videos - 500K hours of video, annotated with 4800 classes. The average length of the videos in the dataset is 230 seconds. 20 classes are selected from the 4800 visual entities to evaluate the proposed model. Specifically, first, the videos whose lengths are less than 16 seconds are removed. Then, for every class, 10,000 positive videos and 40,000 negative videos are randomly collected for training, and 1,000 positive videos and 4,000 negative videos for evaluation. If the class contains less than 10,000 positive videos, all positive videos are used. The videos in YouTube-8M are usually with multiple labels. When selecting negative videos, the target class does

Table 4.1 : Evaluation of the fast forward network with different numbers of frames allowed to be watched.

class	5	10	15	30
vehicle	92.6	95.0	95.2	95.4
video game	88.6	90.0	90.2	90.4
concert	95.3	96.3	96.9	96.9
car	94.9	96.3	96.5	96.6
dance	89.2	90.1	90.2	90.3

not appear in them, meaning that the positive videos and negative videos may have some intersection labels. The mean average precision (mAP) is reported for all experiments.

4.3.2 Implementation Details

Videos are decoded at one frame per-second and extract features from the last hidden representation of Inception-v3 network [102] before the classification layer. The features are then normalized by ℓ_2 normalization. The hidden state size of RNN is set to 128. Each agent is trained for 50 epochs. The batch size, dropout rate and gradient clipping are set to 32, 0.5 and 5.0, respectively. The Adam optimizer is used to optimize the model, with a fixed the learning rate of 1×10^{-3}. The values of α, β, λ and ρ in Eq. (4.11) are searched on the *vehicle* class. Then, $\alpha = 1.0$, $\beta = 0.5$, $\lambda = 1.0$ and $\rho = 0.01$ are fixed in all experiments. The discount factor γ is set to 0.9.

4.3.3 Ablation Study

Fast forward network. To explore what policy the fast forward network learns, the frame penalty μ is set to but the agent is forced to watch 5, 10, 15, and 30

Table 4.2 : Evaluation of the adaptive stop network with different frame penalty μ.

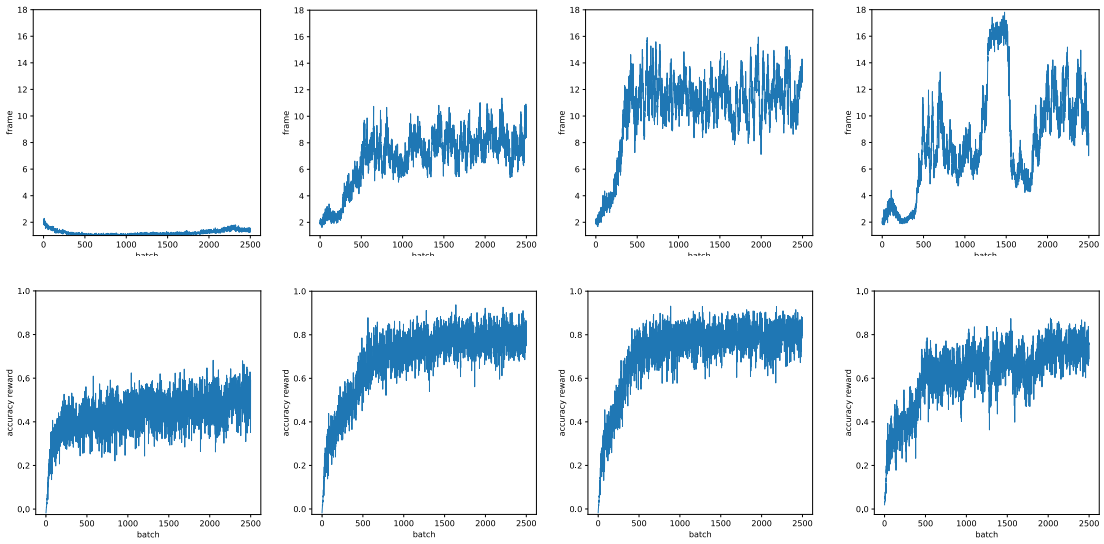
class	$\mu = 10^{-1}$		$\mu = 10^{-2}$		$\mu = 10^{-3}$		$\mu = 10^{-4}$	
	mAP	# frames	mAP	# frames	mAP	# frames	mAP	# frames
vehicle	68.4	2.12	95.1	7.58	95.2	10.24	95.1	10.64
video game	62.0	1.91	90.2	7.84	90.4	9.04	90.4	9.25
concert	71.0	2.42	96.5	6.73	96.8	7.82	96.9	8.46
car	67.1	2.33	96.3	7.46	96.5	9.30	96.7	10.66
dance	60.7	1.88	90.1	6.69	90.5	7.99	90.6	9.01

frames in this experiment.

The fast forward mechanism is evaluated on the *vehicle*, *video game*, *concert*, *car* and *dance* classes. The experiment results are listed in Table 4.1. Basically, when the number of frames allowed to be watched increases, the mAP improves. However, when the number of frames allowed to be watched exceeds 10, the mAP does not improve much.

Adaptive stop network. The step penalty factor μ controls the number of frames the agent will see. Generally, a large μ can improve the efficiency for video classification but may damage the accuracy. This section explores how the step penalty factor influences the behavior of the adaptive stop network. The μ is selected from $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$. The average of used frames and the accuracy reward are recorded during training. The results of first 2,500 training batches are shown in Figure 4.3.

When μ changes from 10^{-1} to 10^{-3}, the agent uses more and more video frames for classification. However, the accuracy reward do not increase when $\mu \leq 10^{-2}$, which indicates that increasing the number of watched frames may not always significantly improve the performance.



(a) $\mu = 10^{-1}$ (b) $\mu = 10^{-2}$ (c) $\mu = 10^{-3}$ (d) $\mu = 10^{-3}$, no baseline

Figure 4.3 : Change of watched frames (top row) and accuracy reward (bottom row) during training agent to recognize *vehicle* with different frame penalty μ. The video class used in this experiment is *vehicle* and the RNN is GRU.

I evaluate the influence of different μ on the classes *vehicle*, *video game*, *concert*, *car* and *dance*. The experiment results are shown in Table 4.2. Basically, when μ decreases, the number of frames to be watched increases. However, when $\mu \leq 10^{-2}$, the mAP is not significantly improved.

Dose the baseline network reduce variance? This section explores whether the baseline network helps reduce variance during training. To remove the influence of baseline network, the β in Eq. (4.9) is set to 0. The frame penalty μ is set to 10^{-3}. The video class is *vehicle* and GRU is used. The average of frames to be used and the accuracy reward during the first 2,500 training batches are shown in Figure 4.3(d). Compared with Figure 4.3(c), the training process without baseline network is considerably unstable, especially for the change of the amount of watched frames as shown in Figure 4.3(d). Therefore, replacing the return R_t by the advantage $R_t - b_t$

can significantly reduce variance during training.

4.3.4 Comparison with Other Methods

This section compares the proposed approach with other four methods, *i.e.*, max pooling, average pooling, LSTM and GRU. For each method, three cases are considered: randomly sampling 10 frames (R10), uniformly sampling 10 frames (U10) and using all frames (All). For each model using random sampling, I evaluate three times and report the average results. For the proposed approach, the μ is set to 10^{-2}. Both LSTM and GRU are evaluated. All 20 classes are used in this experiment. The results are listed in Table 4.3. In order to facilitate the distinction between the proposed model and the RNN models, the proposed method with LSTM is denoted as RL-LSTM and the proposed method with GRU is denoted as RL-GRU.

Compared with max pooling, average pooling, LSTM and GRU using 10 frames, The proposed method achieves the best performance. Especially for RL-GRU, it only uses 7.74 frames on average but acquire 88.0% mAP, which outperforms R10-GRU by 1.0% and U10-GRU by 1.2%. RL-GRU also achieves very similar mAP to the All-GRU (88.5%). What is more, for some classes, RL-GRU even achieves better performance the All-GRU. Take the class *album* as an example, All-GRU achieves 79.3% while RL-GRU achieves 79.5% but only uses 9.36 frames on average. For the class *village*, All-GRU achieves 82.3% while RL-GRU achieves 82.4% using only 7.24 frames on average. This proves that the proposed method is capable of achieving comparable and even better performance than using all frames.

Recall that the average length of videos in YouTube-8M is 230s and videos are decoded at one frame per-second, which indicates that using all frames means using 230 frames on average. The proposed RL-GRU model achieves similar mAP to the All-GRU using only 7.74 frames on average, which significantly improves the classification efficiency.

4.4 Summary

This chapter presents a deep reinforcement learning method which can make classification decision by watching temporally untrimmed videos efficiently. The method contains two mechanisms, *i.e.*, "fast forward" and "adaptive stop" mechanism, which are trained by REINFORCE. By the "fast forward" and "adaptive stop" mechanisms, agents significantly reduce the computational cost for video classification while maintaining the accuracy.

Table 4.3 : Experiment results (mAP%) of max pooling, average pooling, LSTM, GRU and the proposed methods (RL-LSTM, RL-GRU). For the first four methods, randomly sampling 10 frames (R10), uniformly sampling 10 frames (U10) and using all frames (All) are evaluated. For the proposed methods, the frame penalty μ is set to 10^{-2}. All the 20 video classes are used in this experiment.

class	Pooling Method						Recurrent Neural Network						Proposed Method			
	Max Pooling			Average Pool			LSTM			GRU			RL-LSTM		RL-GRU	
	R10	U10	All	R10	U10	All	R10	U10	All	R10	U10	All	mAP	#frms	mAP	#frms
vehicle	88.3	86.3	83.1	93.4	93.2	94.7	93.7	93.7	94.8	94.1	93.9	95.5	94.2	5.68	95.1	7.58
video game	77.6	75.9	65.7	81.7	81.6	83.0	88.9	90.6	88.5	88.9	88.6	90.6	88.8	5.57	90.2	7.85
concert	93.3	91.1	88.3	94.8	94.6	95.1	96.0	97.0	96.6	96.2	96.0	97.0	96.3	5.17	96.5	6.73
car	92.4	90.5	87.6	94.8	94.7	95.3	95.8	95.5	96.4	95.6	95.1	96.6	95.7	7.57	96.3	7.45
dance	85.7	84.0	79.3	88.4	88.4	89.3	89.3	89.2	89.6	89.6	89.7	90.4	89.9	7.48	90.1	6.69
animation	82.9	80.2	76.1	86.8	86.9	87.8	88.6	88.3	88.9	89.1	88.8	90.1	88.5	5.54	90.2	8.33
musician	88.6	87.7	84.3	92.0	92.1	92.6	92.5	92.6	93.1	92.8	92.7	94.2	92.4	5.56	93.9	7.77
music video	73.0	71.5	64.6	78.0	77.7	80.5	82.2	82.5	84.9	83.5	83.1	87.7	83.1	5.83	87.3	7.08
animal	82.0	80.1	63.1	88.1	87.9	89.8	89.7	89.5	90.6	89.0	89.2	91.4	89.3	6.18	90.5	8.02
album	73.0	70.4	67.1	76.2	76.2	77.3	76.8	76.7	77.9	79.0	78.7	79.3	78.7	5.27	79.5	9.36
medicine	74.1	71.9	62.8	79.7	79.8	81.0	83.0	83.2	84.3	84.8	84.6	85.5	84.0	7.15	85.0	8.69
kitchen	94.1	93.0	88.3	95.4	95.2	95.3	95.2	95.0	96.1	95.5	95.2	96.6	94.8	5.38	95.9	9.01
computer	90.4	87.9	84.7	93.2	93.0	93.9	94.1	94.0	95.2	94.3	93.9	95.4	94.1	6.51	94.7	6.51
Christmas	65.6	64.3	49.3	71.7	71.8	73.2	74.0	74.3	76.2	76.1	76.5	78.4	74.9	5.85	77.2	7.71
eating	84.2	83.9	67.9	88.7	88.6	89.3	89.5	89.2	90.1	89.7	89.6	90.8	90.2	7.45	89.9	7.64
shooter	87.6	85.9	79.3	89.8	89.5	90.8	91.1	90.3	92.2	91.3	91.1	92.8	91.9	5.47	92.5	8.75
rain	71.1	70.3	62.4	77.1	77.2	78.7	78.2	78.3	79.7	78.4	78.6	80.5	78.3	5.81	79.8	8.06
champion	63.7	62.9	58.1	68.9	69.0	70.2	72.4	72.4	73.1	73.0	73.1	75.6	72.6	7.02	74.1	7.84
egg	65.6	63.8	51.3	74.4	74.8	76.3	77.0	76.7	78.3	77.5	77.3	78.7	77.6	6.76	77.9	6.57
village	71.2	70.0	60.6	76.0	75.7	77.9	80.8	80.7	81.5	81.1	80.9	82.3	80.8	5.79	82.4	7.24
mean	80.2	78.6	71.2	84.5	84.4	85.6	86.4	86.5	87.9	87.0	86.8	88.5	86.8	6.15	88.0	7.74

Chapter 5

Cubic LSTMs for Conventional Video Prediction with Independent Spatial and Temporal States

5.1 Introduction

This chapter presents a new recurrent unit, *i.e.*, the Cubic Long Short-Term Memory (CubicLSTM) unit, for conventional 2D RGB video prediction. Given a series of history visual observations, the goal of this task is to predict the most likely frames in the future. To better capture the spato-temporal structure of videos, the CubicLSTM unit is equipped with two states, a temporal state and a spatial state, which are respectively generated by two independent convolutions. The motivation is that different kinds of information should be processed and carried by different operations and states. CubicLSTM consists of three branches, which are built along the three axes in the Cartesian coordinate system.

- The temporal branch flows along the x-axis (temporal axis), on which the convolution aims to obtain and process motions. The temporal state is generated by this branch, which contains the motion information.

- The spatial branch flows along the z-axis (spatial axis), on which the convolution is responsible for capturing and analyzing moving objects. The spatial state is generated by this branch, which carries the spatial layout information about moving objects.

- The output branch generates intermediate or final prediction frames along the y-axis (output axis), according to the predicted motions provided by the

temporal branch and the moving object information provided by the spatial branch.

Stacking multiple CubicLSTM units along the spatial branch and output branch can form a two-dimensional network. This two-dimensional network can further construct a three-dimensional network by evolving along the temporal axis, which can be used for conventional video prediction.

5.2 CubicLSTM

This section first reviews fully-connected Long Short-Term Memory (LSTM) [13] and Convolutional LSTM (ConvLSTM) [1], and then presents the proposed CubicLSTM unit in detail.

5.2.1 LSTM

LSTM is a special recurrent neural network (RNN) unit for modeling long-term dependencies. The key to LSTM is the cell state $\boldsymbol{c}_t$ which acts as an accumulator of the sequence or the temporal information. The information from every new input $\boldsymbol{x}_t$ will be integrated to $\boldsymbol{c}_t$ if the input $\boldsymbol{i}_t$ is activated. At the same time, the past cell state $\boldsymbol{c}_{t-1}$ may be forgotten if the forget gate $\boldsymbol{f}_t$ turns on. Whether $\boldsymbol{c}_t$ will be propagated to the hidden state $\boldsymbol{h}_t$ is controlled by the output gate $\boldsymbol{o}_t$. Usually, the cell state $\boldsymbol{c}_t$ and the hidden state $\boldsymbol{h}_t$ are jointly referred to as the internal LSTM state, denoted as $(\boldsymbol{c}_t, \boldsymbol{h}_t)$. The updates of fully-connected LSTM for the t-th time step can be formulated as follows:

$$\text{LSTM}\begin{cases} \boldsymbol{i}_t = \sigma(\boldsymbol{W}_i \cdot [\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_i), \\ \boldsymbol{f}_t = \sigma(\boldsymbol{W}_f \cdot [\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_f), \\ \boldsymbol{o}_t = \sigma(\boldsymbol{W}_o \cdot [\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_o), \\ \hat{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}_c \cdot [\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_c), \\ \boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-t} + \boldsymbol{i}_t \odot \hat{\boldsymbol{c}}_t, \\ \boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t), \end{cases} \tag{5.1}$$

where $\cdot$, $\odot$ and $[\cdot]$ denote the matrix product, the element-wise product and the concatenation operation, respectively.

To generate $\boldsymbol{i}_t$, $\boldsymbol{f}_t$, $\boldsymbol{o}_t$ and $\boldsymbol{c}_t$ in the LSTM, a fully-connected layer is first applied on the concatenation of the input $\boldsymbol{x}_t$ and the last hidden state $\boldsymbol{h}_{t-1}$ with the form $\boldsymbol{W} \cdot [\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}$, where $\boldsymbol{W} = [\boldsymbol{W}_i, \boldsymbol{W}_f, \boldsymbol{W}_o, \boldsymbol{W}_c]$ and $\boldsymbol{b} = [\boldsymbol{b}_i, \boldsymbol{b}_f, \boldsymbol{b}_o, \boldsymbol{b}_c]$. The intermediate result is then split into four parts and passed to the activation functions, *i.e.*, σ and tanh. Last, the new state $(\boldsymbol{c}_t, \boldsymbol{h}_t)$ is produced according to the gates and $\boldsymbol{c}_t$ by element-wise product. In summary, LSTM takes the current input $\boldsymbol{x}_t$ as input, the previous state $(\boldsymbol{c}_{t-1}, \boldsymbol{h}_{t-1})$ and generates the new state $(\boldsymbol{c}_t, \boldsymbol{h}_t)$, parametrized by $\boldsymbol{W}$ and $\boldsymbol{b}$. For simplicity, in this chapter, LSTM is reformulated as follows:

$$\text{LSTM}: \quad (\boldsymbol{c}_t, \boldsymbol{h}_t) = \text{LSTM}\big(\boldsymbol{x}_t, (\boldsymbol{c}_{t-1}, \boldsymbol{h}_{t-1}); \boldsymbol{W}, \boldsymbol{b}; \cdot\big). \tag{5.2}$$

The input $\boldsymbol{x}_t$ and the state $(\boldsymbol{c}_t, \boldsymbol{h}_t)$ in LSTM are all one-dimensional vectors, which cannot directly encode spatial information. Although we can use one-dimensional vectors to represent images by flattening the two-dimensional data (greyscale images) or the three-dimensional data (color images), such operation loses spatial correlations.

5.2.2 ConvLSTM

To exploit spatial correlations for video prediction, ConvLSTM takes three-dimensional tensors as input and replaces the fully-connected layer (matrix product)

in LSTM with the convolutional layer. The updates for ConvLSTM can be written as follow:

$$\text{ConvLSTM}: \quad (\boldsymbol{C}_t, \boldsymbol{H}_t) = \text{LSTM}\big(\boldsymbol{X}_t, (\boldsymbol{C}_{t-1}, \boldsymbol{H}_{t-1}); \boldsymbol{W}, \boldsymbol{B}; *\big), \tag{5.3}$$

where $*$ denotes the convolution operator and $\boldsymbol{X}_t$, $(\boldsymbol{C}_{t-1}, \boldsymbol{H}_{t-1})$, $(\boldsymbol{C}_t, \boldsymbol{H}_t)$ are all three-dimensional tensors with shape (*height*, *width*, *channel*). As LSTMs are designed to learn only one type of information, directly adapting LSTM makes it difficult for ConvLSTM to simultaneously process the temporal information and the spatial information in videos. To predict future spatio-temporal sequences, the convolution in ConvLSTM has to catch motions on one hand and capture moving objects on the other hand. Similarly, the state of ConvLSTM must be capable of storing motion information and visual information at the same time.

5.2.3 The Proposed CubicLSTM

To reduce the burden of prediction, CubicLSTM unit processes the temporal information and the spatial information separately. Specifically, CubicLSTM consists of three branches: a temporal branch, a spatial branch and a output branch. The temporal branch aim to obtain and process motions. The spatial branch is responsible for capturing and analyzing objects. The output branch generates predictions according to the predicted motion information and the moving object information. As shown in Figure 5.1(a), the unit is built along the three axes in a space Cartesian coordinate system.

- Along the x-axis (temporal axis), the convolution operation obtains the current motion information according to the input $\boldsymbol{X}_{t,l}$, the previously motion information $\boldsymbol{H}_{t-1,l}$ and the previously object information $\boldsymbol{H}'_{t,l-1}$. The current motion information is then used to update the previous temporal cell state $\boldsymbol{C}_{t-1,l}$ and produce the new motion information $\boldsymbol{H}_{t,l}$.

- Along the z-axis (spatial axis), the convolution captures the current spatial layout of objects. This information is then used to rectify the previous spatial cell state $\boldsymbol{C}'_{t,l-1}$ and generate the new object visual information $\boldsymbol{H}'_{t,l}$.
- Along the y-axis (output axis), the output branch combines the current motion information and the current object information to generate an intermediate prediction for the input of the next CubicLSTM unit or construct the final prediction frame.

The topological diagram of ConvLSTM is shown in Figure 5.1(b). The updates of CubicLSTM are formulated as Eq. (5.4), where l, $'$ and $''$ denote the spatial network layer, the spatial branch and the output branch respectively.

$$\text{CubicLSTM}\begin{cases} \text{temporal branch}: (\boldsymbol{C}_{t,l}, \boldsymbol{H}_{t,l}) = \text{LSTM}(\boldsymbol{X}_{t,l}, \boldsymbol{H}'_{t,l-1}, (\boldsymbol{C}_{t-1,l}, \boldsymbol{H}_{t-1,l}); \boldsymbol{W}, \boldsymbol{B}; *), \\ \text{spatial branch}: (\boldsymbol{C}'_{t,l}, \boldsymbol{H}'_{t,l}) = \text{LSTM}(\boldsymbol{X}_{t,l}, \boldsymbol{H}_{t-1,l}, (\boldsymbol{C}'_{t,l-1}, \boldsymbol{H}'_{t,l-1}); \boldsymbol{W}', \boldsymbol{B}'; *), \\ \text{output branch}: \boldsymbol{Y}_{t,l} = \boldsymbol{W}'' * [\boldsymbol{H}_{t,l}, \boldsymbol{H}'_{t,l}] + \boldsymbol{B}''. \end{cases} \tag{5.4}$$

Essentially, in a single CubicLSTM, the temporal branch and the spatial branch are symmetrical. However, in experiment, experiments show that adopting 1×1 convolution for temporal branch achieves higher accuracy than adopting 5×5 convolution. This may indicate that "motion" should focus on temporal neighbors while "object" should focus on spatial neighbors. Their functions also depend on their positions and connections in stacked CubicLSTM networks. To deal with a sequence, the same unit is repeated along the temporal axis. Therefore, the parameters are shared along the temporal dimension. For the spatial axis, multiple units can be stacked to form a multi-layer structure to better exploit spatial correlations and capture objects. The parameters along the spatial axis are different.

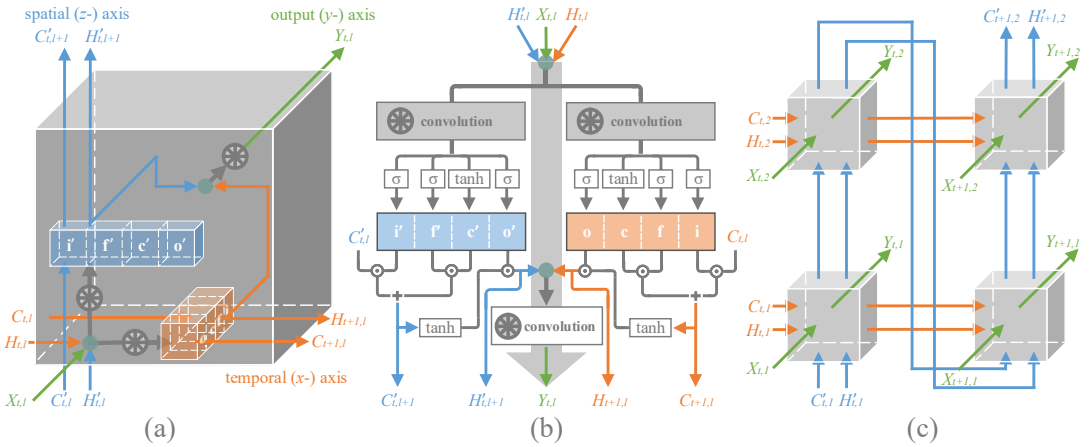


Figure 5.1 : (a) 3D structure of the CubicLSTM unit. (b) Topological diagram of the CubicLSTM unit. (c) A two-spatial-layer CubicLSTM network. The unit consists of three branches, a spatial (z-) branch for extracting and recognizing moving objects, a temporal (y-) branch for capturing and predicting motions, and an output (x-) branch for combining the first two branches to generate the predicted frames.

At the end of spatial direction, rather than being discarded, the spatial state is used to initialize the starting spatial state at the next time step. Formally, the spatial states between two time steps are defined as follows,

$$\boldsymbol{C}'_{t,1} = \boldsymbol{C}'_{t-1,L}, \quad \boldsymbol{H}'_{t,1} = \boldsymbol{H}'_{t-1,L}, \tag{5.5}$$

where $L > 1$ is the number of layers along the spatial axis. A 2-spatial-layer CubicLSTM network is demonstrated in Figure 5.1(c), which is the smallest network formed by CubicLSTMs.

5.3 Stacked CubicLSTM Network for Video Prediction

The aim of video prediction is to predict the possible frames in the future after watching several consecutive frames. This section presents how to apply the proposed CubicLSTM unit for video prediction. The prediction network is created by

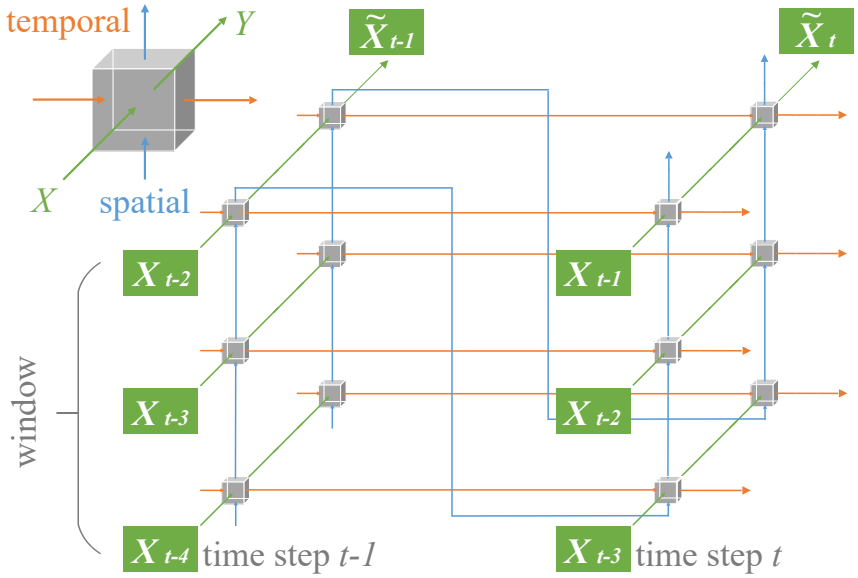


Figure 5.2 : A stacked CubicLSTM network with three spatial layers and two output layers, which can watch three frames at one time.

first stacking multiple CubicLSTM units along the spatial axis and along the output axis, which forms a two-dimensional network, and then evolving along the temporal branch, which forms a three-dimensional structure.

In contrast to traditional LSTM structures, CubicLSTM is capable of watching multiple adjacent frames at one time along the spatial axis, which forms a sliding window. The size of the sliding window is equal to the number of spatial layers. Suppose we have L spatial layers: CubicLSTM network will observe the previous $\boldsymbol{X}_{t-L+1}, \cdots, \boldsymbol{X}_{t-1}$ frames to predict the t-th frame. The sliding window enables the CubicLSTM network to better capture the information about both motions and objects. An example of stacked CubicLSTM network is illustrated in Figure 5.2.

5.4 Experiments

The proposed CubicLSTM is evaluated on three video prediction datasets, Moving-MNIST dataset [42], Robotic Pushing dataset [3] and KTH Action dataset [103],

including synthetic and real-world video sequences. All models are trained using the ADAM optimizer [104] and implemented in TensorFlow with eight GPUs in parallel. Batch size is set 4 for each GPU.

5.4.1 Moving MNIST dataset

The Moving MNIST dataset consists of 20 consecutive frames, 10 for the input and 10 for the prediction. Each frame contains two potentially overlapping handwritten digits moving and bouncing inside a 64×64 image. The digits are chosen randomly from the MNIST dataset and placed initially at random locations. Each digit is assigned a velocity whose direction is chosen uniformly on a unit circle and whose magnitude is also chosen uniformly at random over a fixed range. The size of the training set can be considerable large. I use the code provided by [1] to generate training samples on-the-fly. For the test, following [55], models are evaluated with two settings, *i.e.*, two moving digits (MNIST-2) and three moving digits (MNIST-3) with the mean square error (MSE) and binary cross-entropy (BCE) were used. Pixel values are converted into the range $[0, 1]$.

Each state in the implementation of CubicLSTM has 32 channels. The size of the spatial-convolutional kernel was set to 5×5. Both the temporal-convolutional kernel and output-convolutional kernel were set to 1×1. Models are trained using MSE and BCE, corresponding to the different evaluation metrics. The encoder-decoder framework [42, 1] is adopted for prediction, in which the initial states of the decoder network are copied from the last states of the encoder network. The inputs and outputs are fed and generated as shown in Figure 5.2. All models are trained for $300K$ iterations with a learning rate of 10^{-3} for the first $150K$ iterations and a learning rate of 10^{-4} for the latter $150K$ iterations.

Improvement by the spatial branch. Compared to ConvLSTM [1], CubicLSTM has an additional spatial branch. To prove the improvement achieved by

Table 5.1 : Results of stacked CubicLSTM network and state-of-the-art models on the Moving-MNIST dataset. The "CubicLSTM ($c \times y \times z$)" denotes that the CubicLSTM model has z spatial layer(s) and y output layer(s), and the channel size of its state is c. The per-frame mean square error (MSE) and per-frame binary cross-entropy (BCE) of generated frames are reported. Lower MSE or CE means better prediction accuracy.

Models	MSE		BCE
	MNIST-2	MNIST-3	MNIST-2
LSTM [42]	118.3	162.4	483.2
CridLSTM [105]	111.6	157.8	419.5
ConvLSTM (128×4) [1]	103.3	142.1	367.0
PyramidLSTM [106]	100.5	142.8	355.3
CDNA [3]	97.4	138.2	346.6
DFN [45]	89.0	130.5	285.2
VPN baseline [52]	70.0	125.2	110.1
PredRNN with spatialtemporal memory [55]	74.0	118.2	118.5
PredRNN + ST-LSTM (128×4) [55]	56.8	93.4	97.0
CubicLSTM ($32 \times 3 \times 1$)	111.5	158.4	386.3
CubicLSTM ($32 \times 1 \times 3$)	73.4	127.1	210.3
CubicLSTM ($32 \times 3 \times 2$)	59.7	110.2	121.9
CubicLSTM ($32 \times 3 \times 3$)	**47.3**	**88.2**	**91.7**

this branch, I compare two models, both of which have 3 CubicLSTM units. The first model stacks units along the output axis forms, forming a structure of 1 spatial layer and 3 output layers. Since each state has 32 channels, I denote this structure as $(32 \times 3 \times 1)$. The second model stacks the three units along the spatial axis, forming a structure of 3 spatial layers and 1 output layer structure, denoted as $(32 \times 1 \times 3)$. The results are listed in Table 5.1.

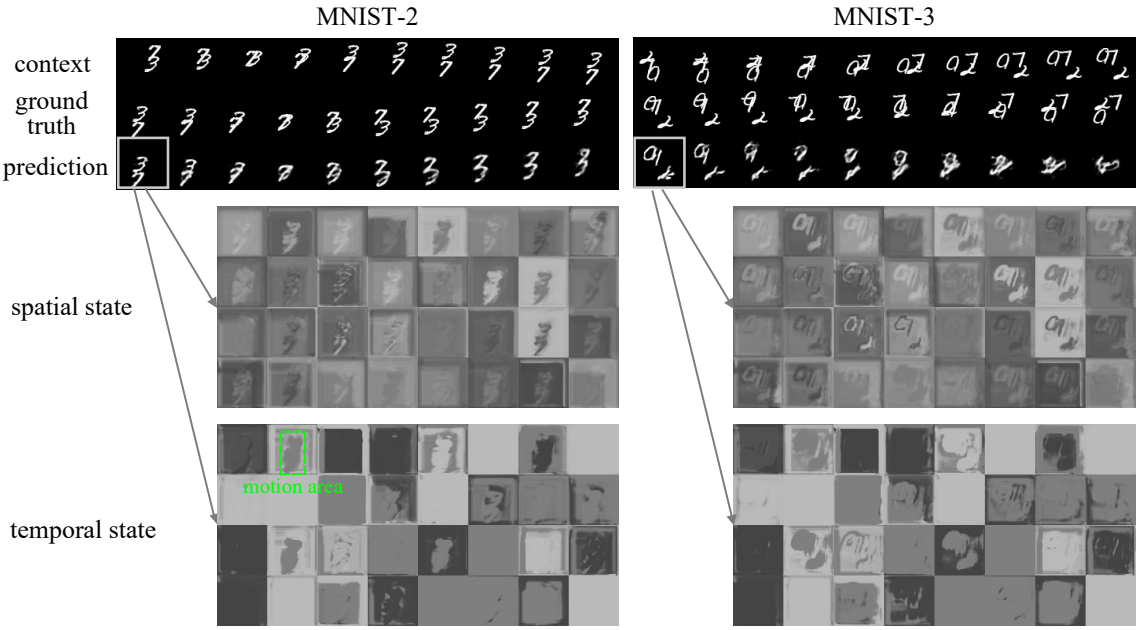


Figure 5.3 : Prediction examples on the Moving MNIST dataset (top) and visualizations of the spatial hidden state (middle) and temporal hidden state (bottom). The spatial state provides the object information such as contours and appearances, while the temporal state provides the motion information of potential motion areas. The two states will be exploited to generate the future frame by the output branch.

On one hand, although both have 3 units, CubicLSTM ($32 \times 1 \times 3$) significantly outperforms CubicLSTM ($32 \times 3 \times 1$). As the spatial branch and the temporal branch in CubicLSTM ($32 \times 3 \times 1$) are identical, the model does not exploit spatial correlations well and only achieves similar accuracy to ConvLSTM (128×4). On the other hand, even though CubicLSTM ($32 \times 1 \times 3$) uses fewer parameters than ConvLSTM (128×4), it obtains better predictions. This experiment validates that the temporal information and the spatial information should be processed separately.

Comparison with other models. All the results from existing models on the Moving MNIST dataset are listed in Table 5.1. Two CubicLSTM settings are reported. The first has three output layers and two spatial layers ($32 \times 3 \times 2$), and the second one has three output layers and three spatial layers ($32 \times 3 \times 3$). The

CubicLSTM ($32 \times 3 \times 3$) model produces the best prediction. The prediction of CubicLSTM is improved by increasing the number of spatial layers.

Two prediction examples produced by CubicLSTM ($32 \times 3 \times 3$) are illustrated in the first row of Figure 5.3. The model is capable of generating accurate predictions for the two-digit case (MNIST-2). In the second and third rows of Figure 5.3, the spatial hidden states and the temporal states of the last CubicLSTM unit in the ($32 \times 3 \times 3$) structure are visualized when it predicts the first frames of the two prediction examples. These states have 32 channels and I visualize each channel by the function $\sigma(\cdot) \times 255$. As can be seen from the visualization, the spatial hidden state reflects the visual and the spatial information of the digits in the frame. The temporal hidden state suggests that it is likely to contain some "motion areas". Compared with the relatively precise information of objects provided by the spatial hidden state, the "motion areas" provided by the temporal hidden state are somewhat rough. The output branch will apply these "motion areas" on the digits to generate the prediction.

5.4.2 Robotic Pushing dataset

Robotic Pushing [3] is an action-conditional video prediction dataset which recodes 10 robotic arms pushing hundreds of objects in a basket. The dataset consists of 50,000 iteration sequences with 1.5 million video frames and two test sets. Objects in the first test set use two subsets of objects in the training set, which are so-called "seen objects". The second test set involves two subsets of "novel objects", which are not used during training. Each test set has 1,500 recorded sequences. In addition to RGB images, the dataset also provides the corresponding gripper poses by recoding its states and commanded actions, both of which are 5-dimensional vectors. Following [3], images are center-cropped and downsampled to 64×64, and use the Peak Signal to Noise Ratio (PSNR) [58] to evaluate the prediction accuracy.

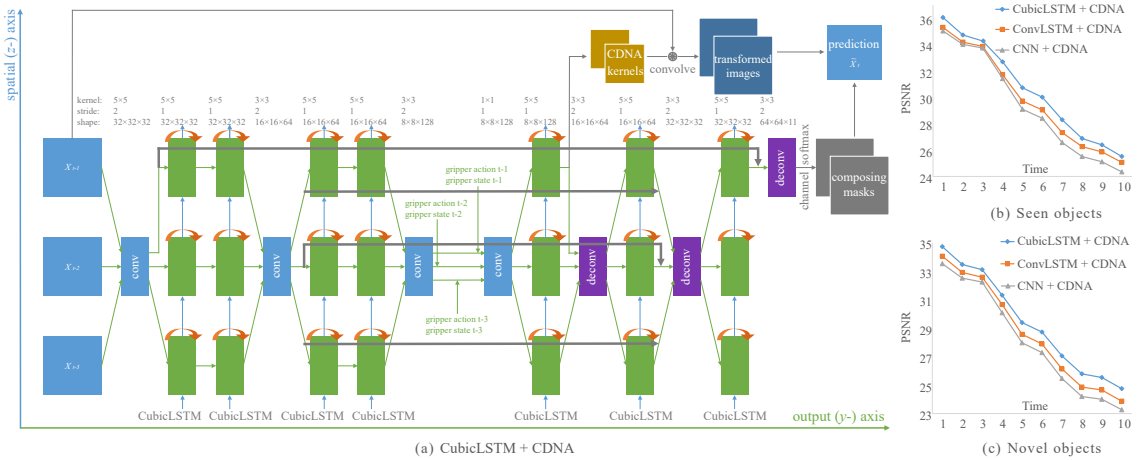


Figure 5.4 : (a): Architecture of the model for the Robotic Pushing dataset. The model is largely borrowed from the convolutional dynamic neural advection (CDNA) model proposed in [3] and replaces ConvLSTMs in the CNDA model with CubicLSTMs. The model has three spatial layers, among which the convolutions and the deconvolutions are shared. (b)-(c): Frame-wise PSNR comparisons of "CubicLSTM + CDNA", "ConvLSTM + CDNA" [3] and "CNN + CDNA" on the Robotic Pushing dataset. Higher PSNR means better prediction accuracy.

The model is illustrated in Figure 5.4(a). The model is largely borrowed from the convolutional dynamic neural advection (CDNA) model [3] which is an encoder-decoder architecture that expands along the output direction and consists of several convolutional encoders, ConvLSTMs, deconvolutional decoders and CDNA kernels. The CDNA model in [3] is denoted as "ConvLSTM + CDNA". The model replaces ConvLSTMs in the CNDA model with CubicLSTMs and expands the model along the spatial axis to form a three-spatial-layer architecture. The convolutional encoders and the deconvolutional decoders are shared among these three spatial layers. The proposed model is referred to as "CubicLSTM + CDNA". A baseline model is also designed by replacing ConvLSTMs in the CNDA model with CNNs, denoted as "CNN + CDNA". Since CNNs do not have internal state that flows

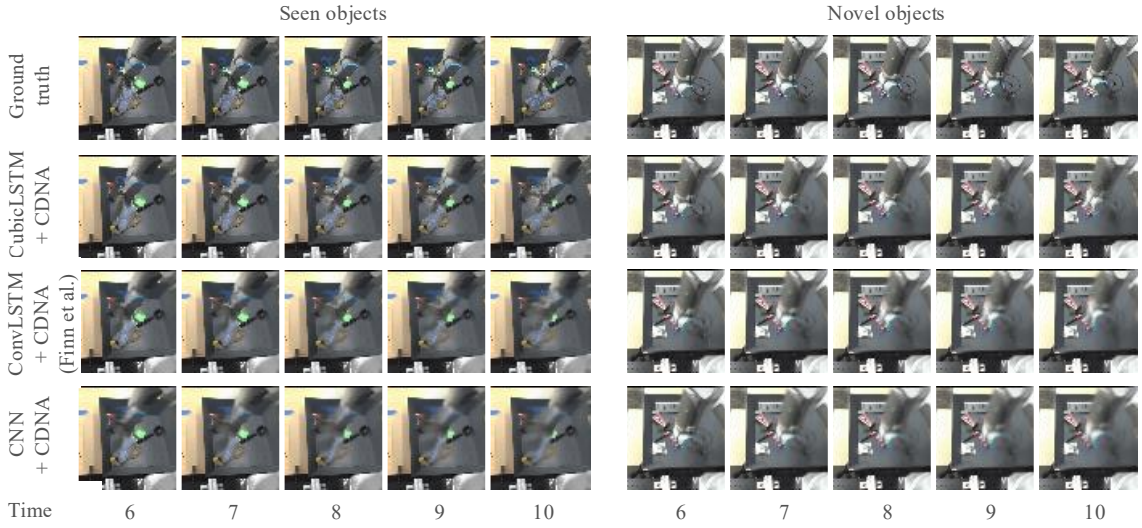


Figure 5.5 : Qualitative comparisons of "CubicLSTM + CDNA", "ConvLSTM + CDNA" [3] and "CNN + CDNA" on the Robotic Pushing dataset. The proposed "CubicLSTM + CDNA" method can generate clearer frames than others, especially for the videos with novel objects.

along the temporal dimension, it cannot exploit the temporal information. In this experiment, all models are given three context frames before predicting 10 future frames and were trained for $100K$ iterations with the mean square error loss and the learning rate of 10^{-3}. The results are shown in Figure 5.4(b) and Figure 5.4(c).

The "ConvLSTM + CDNA" model only obtains similar accuracy to the "CNN + CDNA" model, which indicates that convolutions of ConvLSTMs in the CDNA model may mainly focus on the spatial information while neglecting the temporal information. These ConvLSTMs almost degenerate to CNNs. The "CubicLSTM + CDNA" model achieves the best prediction accuracy among these three models. Therefore, CubicLSTMs is capable of exploiting the temporal information better. A qualitative comparison of predicted video sequences is given in Figure 5.5. The "CubicLSTM + CDNA" model generates sharper frames than other models.

5.4.3 KTH Action dataset

The KTH Action dataset [103] is a real-world videos of people performing one of six actions (walking, jogging, running, boxing, handwaving, hand-clapping) against fairly uniform backgrounds. The proposed method is compared with the Motion-Content Network (MCnet) [48] and Disentangled Representation Net (DrNet) [49] on the KTH Action dataset.

Figure 3: Comparision with DrNet and MCNet on KTH Actions dataset.

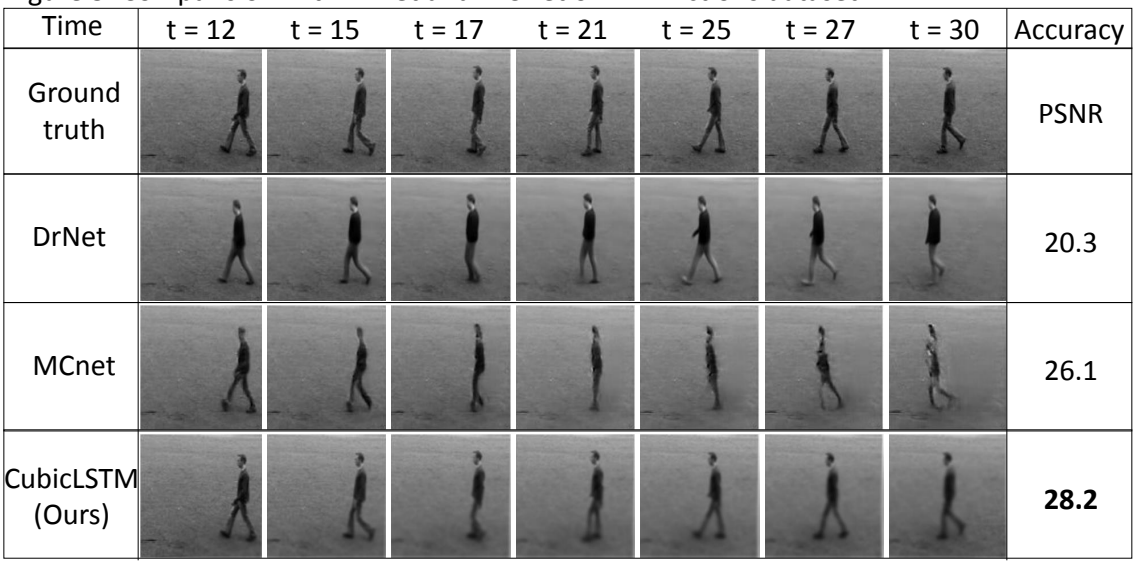


Figure 4: Obje

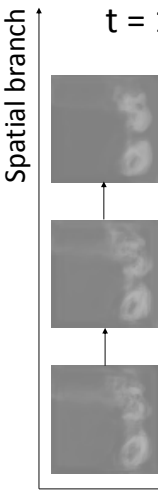


Figure 5.6 : Qualitative and quantitative comparison of generated sequences between DrNet, MCnet and CubicLSTM.

The qualitative and quantitative comparisons between DrNet, MCnet and CubicLSTM are shown in Figure 5.6. Although DrNet can generate relatively clear frames, the appearance and position of the person is slightly changed. Therefore, the PSNR accuracy is not quite high. For MCnet, the generated frames are a little distorted. Compared to DrNet and MCnet, the proposed CubicLSTM model can predict more accurate frames and therefore achieves the highest accuracy.

5.5 Summary

This chapter presents the CubicLSTM unit for conventional RGB video prediction. The unit processes spatio-temporal information separately by a spatial branch and a temporal branch. The spatial branch recognizes and extracts moving objects while the temporal branch captures and predicts motions. The spatial branch aim to capture moving objects and the temporal branch is responsible for processing motions. This separation can efficiently reduce the video prediction burden for deep neural networks. The stacked CubicLSTM network generates better predictions than prior models, which validates our belief that the spatial information and temporal information should be processed separately.

Chapter 6

Point Spatio-temporal Correlation RNNs for Point Cloud Video Prediction

6.1 Introduction

Modern robotic and automatic driving systems usually employ real-time depth sensors such as LiDAR to perceive the geometric information of scenes accurately while being robust to different lighting conditions. Generally, a point cloud is expressed as a set of coordinates $\{(x, y, z)\}$, *i.e.*, $\boldsymbol{P} \in \mathbb{R}^{n\times 3}$ indicating positions of each measured point in 3D space, associated with point features $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ if features are available, where n is the number of points and d is the feature dimension. Because motion is the change in position with respect to time, point cloud videos are more suitable for motion analysis than conventional videos. Therefore, it is promising to conduct video prediction on moving 3D point clouds.

However, unlike conventional grid based RGB videos, point cloud videos are irregular and unordered in the spatial dimension, and emerge inconsistently in the temporal dimension. The conventional RNNs have two severe limitations on processing point cloud sequences. On one hand, RNNs learn from one-dimensional vectors, in which the input, state and output are highly compact. It is difficult for a vector to represent an entire point cloud. Although the $\boldsymbol{P}$ and $\boldsymbol{X}$ can be flatten to one-dimensional vectors, such operation heavily damages the data structure and increases the challenge for neural networks to understand point clouds. Moreover, this operation is not independent of point permutations. If we use the global feature to represent a point cloud, the local structure will be lost. One the other hand,

RNNs aggregate the previous state and the current input based on a concatenation operation. Because point clouds are unordered, concatenation can not be directly applied to point cloud frames. Therefore, new RNN units should be extended to overcome this problem.

This chapter presents a series of point spatio-temporal correlation recurrent neural networks [107], including PointRNN, PointGRU and PointLSTM, for point-level point cloud video modelling, which are then applied to point cloud video prediction by building sequence-to-sequence (seq2seq) [2] models. Experimental results on a synthetic moving MNIST point cloud dataset and two large-scale autonomous driving datasets, *i.e.*, Argoverse [108] and nuScenes [109], show that PointRNN, PointGRU and PointLSTM produce promising predictions, confirming their ability to process moving point clouds.

6.2 PointRNN

This section first reviews the standard (vanilla) RNN and then describes the proposed PointRNN in detail.

The RNN is a class of deep neural networks which uses its state $\boldsymbol{s} \in \mathbb{R}^{d' \times 1}$ to process sequences of inputs, which allows it to exhibit temporal dynamic behavior. Usually, RNNs rely on a concatenation operation to aggregate the past $\boldsymbol{s}_{t-1}$ and the current $\boldsymbol{x}_t \in \mathbb{R}^{d \times 1}$, which is referred to as the rnn function in this chapter,

$$\boldsymbol{s}_t = \mathrm{rnn}(\boldsymbol{x}_t, \boldsymbol{s}_{t-1}; \boldsymbol{W}, \boldsymbol{b}) = \boldsymbol{W} \cdot [\boldsymbol{x}_t, \boldsymbol{s}_{t-1}] + \boldsymbol{b}, \tag{6.1}$$

where $\cdot$ denotes matrix multiplication and $[\cdot, \cdot]$ denotes concatenation. The $\boldsymbol{W} \in \mathbb{R}^{d' \times (d+d')}$ and $\boldsymbol{b} \in \mathbb{R}^{d'}$ are the parameters of RNN to be learned. This operation can be implemented by fully-connected (FC) layer in deep neural networks (shown in Figure 6.1(a)). Usually, RNN uses its state $\boldsymbol{s}_t$ as output, *i.e.*, $\boldsymbol{y}_t = \boldsymbol{s}_t$.

The conventional RNN learns from one-dimensional vectors, limiting its applica-

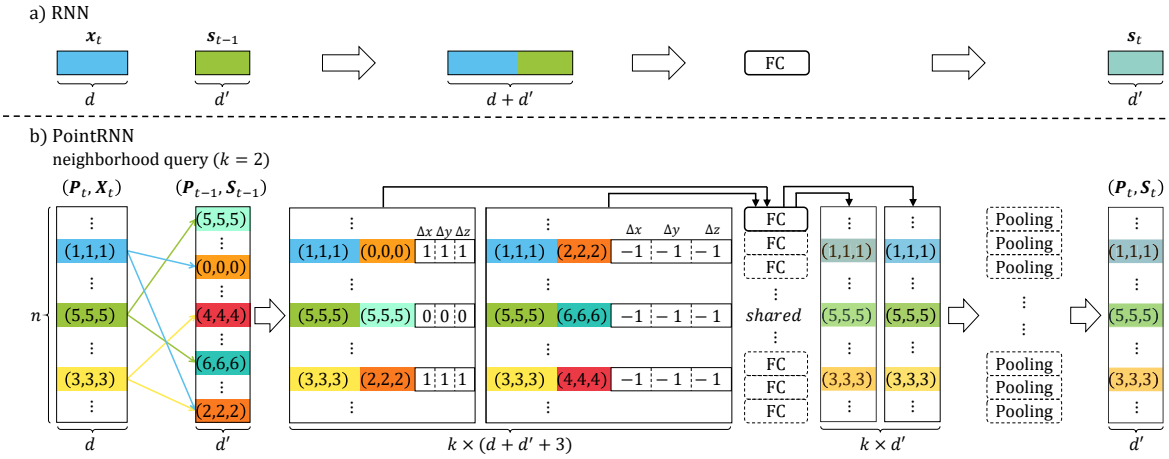


Figure 6.1 : a) The RNN aggregates the input $\boldsymbol{x}_t \in \mathbb{R}^d$ and state $\boldsymbol{s}_{t-1} \in \mathbb{R}^{d'}$ by a concatenation operation, followed by an fully-connected (FC) layer. b) The PointRNN aggregates $\boldsymbol{X}_t \in \mathbb{R}^{n \times d}$ and $\boldsymbol{S}_{t-1} \in \mathbb{R}^{n \times d'}$ according to point coordinates by a point spatio-temporal correlation. First, for each point in $\boldsymbol{P}_t \in \mathbb{R}^{n \times 3}$, PointRNN searches k neighbors in $\boldsymbol{P}_{t-1}$. Second, for each neighbor, the feature of the query point in $\boldsymbol{X}_t$, the state of the neighbor in $\boldsymbol{S}_{t-1}$, and the displacement from the neighbor to the query point are concatenated. Third, the k concatenated representations are processed by an FC layer. At last, pooling is used to merge the k representations.

tion to process point cloud sequences. To keep the spatial structure, the proposed PointRNN directly takes the coordinates $\boldsymbol{P}_t$ and features $\boldsymbol{X}_t$ as input. Similarly, the state $\boldsymbol{s}_t$ and the output $\boldsymbol{y}_t$ in RNN are extended to $\boldsymbol{S}_t \in \mathbb{R}^{n \times d'}$ and $\boldsymbol{Y}_t \in \mathbb{R}^{n \times d'}$ in PointRNN. The update for the point states of PointRNN at the t-th time step is formulated as follows,

$$\begin{aligned} \boldsymbol{S}_t &= \text{point-rnn}\big((\boldsymbol{P}_t, \boldsymbol{X}_t), (\boldsymbol{P}_{t-1}, \boldsymbol{S}_{t-1}); \boldsymbol{W}, \boldsymbol{b}\big) \\ &= \left\{ \underset{j|\boldsymbol{P}_{t-1}^j \in \mathcal{N}(\boldsymbol{P}_t^i)}{\text{pooling}} \left\{ \boldsymbol{W} \cdot \left[\boldsymbol{X}_t^i, \boldsymbol{S}_{t-1}^j, \boldsymbol{P}_t^i - \boldsymbol{P}_{t-1}^j \right] + \boldsymbol{b} \right\} \right\}_{i \in \{1, \cdots, n\},} \end{aligned} \tag{6.2}$$

where $\boldsymbol{W} \in \mathbb{R}^{d' \times (d+d'+3)}$, $\boldsymbol{b} \in \mathbb{R}^{d'}$ and $\mathcal{N}(\cdot)$ is neighborhood query. This point-based spatial-temporal correlation is referred as point-rnn. Similar to RNN, by default, PointRNN uses $\boldsymbol{S}_t$ as output, *i.e.*, $\boldsymbol{Y}_t = \boldsymbol{S}_t$.

The goal of the point-rnn function is to aggregate the past and the current according to point coordinates (shown in Figure 6.1 (b)). Given $(\boldsymbol{P}_t, \boldsymbol{X}_t)$ and $(\boldsymbol{P}_{t-1}, \boldsymbol{S}_{t-1})$, point-rnn merges $\boldsymbol{X}_t$ and $\boldsymbol{S}_{t-1}$ according to $\boldsymbol{P}_t$ and $\boldsymbol{P}_{t-1}$. Specifically, for each point in $\boldsymbol{P}_t$, taking the i-th point $\boldsymbol{P}_t^i$ for instance, point-rnn first finds its neighbors in $\boldsymbol{P}_{t-1}$. The neighbors potentially share the same geometry and motion information about the query point $\boldsymbol{P}_t^i$. Second, for each neighbor, taking the neighbor $\boldsymbol{P}_{t-1}^j$ for instance, the feature $\boldsymbol{X}_t^i$ of the query point, the state $\boldsymbol{S}_{t-1}^j$ of the neighbor, and the displacement $\boldsymbol{P}_t^i - \boldsymbol{P}_{t-1}^j$ from the neighbor to the query point are concatenated, which are subsequently processed by a shared FC layer. Third, the processed concatenations are pooled to a single representation. The output of point-rnn, *i.e.*, $\boldsymbol{S}_t$, contains the past and the current information of each point in $\boldsymbol{P}_t$. Note that, compared with the rnn function in Eq. (6.1), if we ignore the parameters caused by the displacements, the point-rnn function does not introduce additional parameters.

Two methods for neighborhood query are studied in this chapter. The first one directly finds k nearest neighbors for the query point (kNN). The second one works by first finding all points that are within a radius to the query point and then sampling k neighbors from the points (ball query [79]). Since a point cloud is a set of orderless points, point states or features of two time steps at the same row may correspond to different points. Without point coordinates, an independent $\boldsymbol{S}_t$ or $\boldsymbol{Y}_t$ is meaningless. Therefore, the point coordinates $\boldsymbol{P}_t$ is added into the state and output of PointRNN.

The PointRNN provides a prototype that extends RNN to process point cloud sequence. Each component in PointRNN is necessary. However, we can design more effective point spatio-temporal correlation methods to replace point-rnn, which can be further studied in the future.

6.3 PointGRU and PointLSTM

Because PointRNN inherits RNN, it may encounter the same exploding and vanishing gradient problems as RNN. Therefore, two variants, which combine PointRNN with GRU and LSTM, respectively, are proposed to overcome these problems. The concatenation operations in GRU are replaced with the point spatio-temporal correlation operations, *i.e.*, point-rnn, forming the PointGRU unit. The updates for PointGRU at the t-th time step are formulated as follows,

$$\begin{aligned}
\boldsymbol{Z}_t &= \sigma\Big(\text{point-rnn}\big((\boldsymbol{P}_t, \boldsymbol{X}_t), (\boldsymbol{P}_{t-1}, \boldsymbol{S}_{t-1}); \boldsymbol{W}_z, \boldsymbol{b}_z\big)\Big), \\
\boldsymbol{R}_t &= \sigma\Big(\text{point-rnn}\big((\boldsymbol{P}_t, \boldsymbol{X}_t), (\boldsymbol{P}_{t-1}, \boldsymbol{S}_{t-1}); \boldsymbol{W}_r, \boldsymbol{b}_r\big)\Big), \\
\hat{\boldsymbol{S}}_{t-1} &= \text{point-rnn}\big((\boldsymbol{P}_t, \text{None}), (\boldsymbol{P}_{t-1}, \boldsymbol{S}_{t-1}); \boldsymbol{W}_{\hat{s}}, \boldsymbol{b}_{\hat{s}}\big), \\
\tilde{\boldsymbol{S}}_t &= \tanh\big(\boldsymbol{W}_{\tilde{s}} \cdot [\boldsymbol{X}_t, \boldsymbol{R}_t \odot \hat{\boldsymbol{S}}_{t-1}] + \boldsymbol{b}_{\tilde{s}}\big), \\
\boldsymbol{S}_t &= \boldsymbol{Z}_t \odot \hat{\boldsymbol{S}}_{t-1} + (1 - \boldsymbol{Z}_t) \odot \tilde{\boldsymbol{S}}_t,
\end{aligned} \tag{6.3}$$

where $\boldsymbol{Z}_t$ is the update gate and $\boldsymbol{R}_t$ is the reset gate. The $\sigma(\cdot)$ denotes the sigmoid function and $\odot$ denotes the Hadamard product. Similar to PointRNN, the input of PointGRU is $(\boldsymbol{P}_t, \boldsymbol{X}_t)$, the state is $(\boldsymbol{P}_t, \boldsymbol{S}_t)$ and by default, the output is $(\boldsymbol{P}_t, \boldsymbol{S}_t)$. Note that, besides the point-rnn function, another difference between GRU and PointGRU is that, PointGRU has an additional step $\hat{\boldsymbol{S}}_{t-1}$. The goal of this step is to weight and permute $\boldsymbol{S}_{t-1}$ according to the current input points $\boldsymbol{P}_t$. Only after this step can we perform Hadamard product between the previous state $\hat{\boldsymbol{S}}_{t-1}$ and the current reset gate $\boldsymbol{R}_t$.

Similarly, the concatenation operations in LSTM are replaced with the point-rnn functions, forming the PointLSTM unit. The updates for PointLSTM at the t-th time step are formulated as Eq. (6.4), where $\boldsymbol{I}_t$ is the input gate, $\boldsymbol{F}_t$ is the forget gate, $\boldsymbol{O}_t$ is the output gate, $\boldsymbol{C}_t$ is the cell state and $\boldsymbol{H}_t$ is the hidden state. The input of PointLSTM is $(\boldsymbol{P}_t, \boldsymbol{X}_t)$, the state is $(\boldsymbol{P}_t, \boldsymbol{H}_t, \boldsymbol{C}_t)$ and by default, the output is $(\boldsymbol{P}_t, \boldsymbol{H}_t)$. Like from GRU to PointGRU, compared with LSTM, PointLSTM has

an additional step $\hat{\boldsymbol{C}}_{t-1}$, which weights and permutes $\boldsymbol{C}_{t-1}$ according to the current input points $\boldsymbol{P}_t$.

$$
\begin{aligned}
\boldsymbol{I}_t &= \sigma\Big(\text{point-rnn}\big((\boldsymbol{P}_t, \boldsymbol{X}_t), (\boldsymbol{P}_{t-1}, \boldsymbol{H}_{t-1}); \boldsymbol{W}_i, \boldsymbol{b}_i\big)\Big), \\
\boldsymbol{F}_t &= \sigma\Big(\text{point-rnn}\big((\boldsymbol{P}_t, \boldsymbol{X}_t), (\boldsymbol{P}_{t-1}, \boldsymbol{H}_{t-1}); \boldsymbol{W}_f, \boldsymbol{b}_f\big)\Big), \\
\boldsymbol{O}_t &= \sigma\Big(\text{point-rnn}\big((\boldsymbol{P}_t, \boldsymbol{X}_t), (\boldsymbol{P}_{t-1}, \boldsymbol{H}_{t-1}); \boldsymbol{W}_o, \boldsymbol{b}_o\big)\Big), \\
\hat{\boldsymbol{C}}_{t-1} &= \text{point-rnn}\big((\boldsymbol{P}_t, \text{None}), (\boldsymbol{P}_{t-1}, \boldsymbol{C}_{t-1}); \boldsymbol{W}_{\hat{c}}, \boldsymbol{b}_{\hat{c}}\big), \\
\tilde{\boldsymbol{C}}_t &= \tanh\Big(\text{point-rnn}\big((\boldsymbol{P}_t, \boldsymbol{X}_t), (\boldsymbol{P}_{t-1}, \boldsymbol{H}_{t-1}); \boldsymbol{W}_{\tilde{c}}, \boldsymbol{b}_{\tilde{c}}\big)\Big), \\
\boldsymbol{C}_t &= \boldsymbol{F}_t \odot \hat{\boldsymbol{C}}_{t-1} + \boldsymbol{I}_t \odot \tilde{\boldsymbol{C}}_t, \\
\boldsymbol{H}_t &= \boldsymbol{O}_t \odot \tanh(\boldsymbol{C}_t),
\end{aligned}
\tag{6.4}
$$

6.4 Point Cloud Video Prediction

6.4.1 Architecture

This section presents a basic model and an advanced model based on the seq2seq framework. The basic model (shown in Figure 6.2 (a)) is composed two parts, *i.e.*, one part for encoding the given point cloud sequence and one part for predicting. Specifically, an encoding PointRNN, PointGRU or PointLSTM unit watches the input point clouds one by one. After watching the last input $\boldsymbol{P}_t$, its state is used to initialize the state of a predicting PointRNN, PointGRU or PointLSTM unit. The predicting unit then takes $\boldsymbol{P}_t$ as input and begins to make predictions. Rather than directly generating future point coordinates, the proposed models predict displacements $\Delta\boldsymbol{P}_t$ that will happen between the current step and the next step, which can be seen as 3D scene flow.

Multiple PointRNN, PointGRU or PointLSTM units can be stacked to build up a multi-layer structure for hierarchical prediction. However, because all points are processed in each layer, a major problem of this structure is that it is computationally intensive, especially for high-resolution point sets. To alleviate this problem,

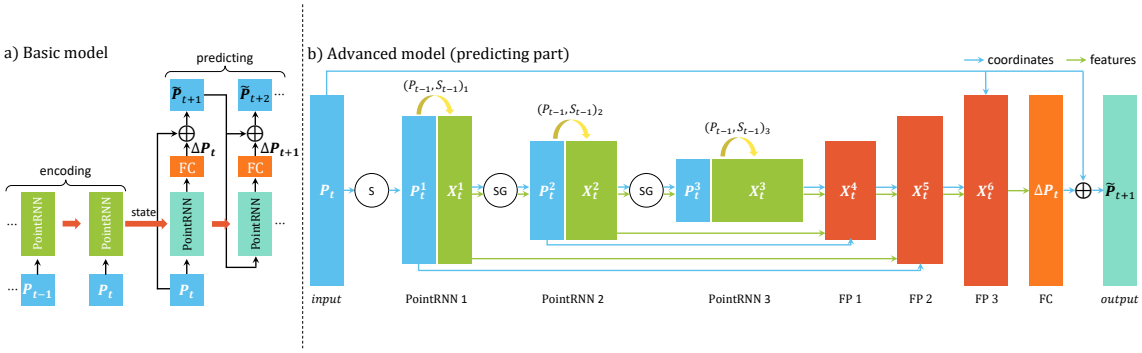


Figure 6.2 : Architectures for point cloud video prediction. a) Basic model (with one PointRNN layer). A PointRNN encodes the given point cloud sequence to a state, *i.e.*, $(\boldsymbol{P}_t, \boldsymbol{S}_t)$, which is then used to initialize the state of a predicting PointRNN. Multiple PointRNN layers can be stacked along the output direction. The prediction $\tilde{\boldsymbol{P}}_{t+1}$ is achieved by $\boldsymbol{P}_t + \Delta\boldsymbol{P}_t$. b) Predicting part of the advanced model (with three PointRNN layers). Sampling (S) and grouping (G) are used to down-sample points and group features, respectively. PointRNNs aggregate the past and the current, and outputs features for prediction. Feature propagation (FP) layers are used to propagate features from subsampled points to the original points. The fully-connected (FC) layer regresses features to predicted displacements $\Delta\boldsymbol{P}_t$.

an advanced model (shown in Figure 6.2 (b)) is proposed. This model borrows two components from PointNet++ [79], *i.e.*, 1) a sampling operation and a grouping operation for down-sampling points and their features and 2) a feature propagation layer for up-sampling the representations associated with the intermediate points to the original points. By this down-up-sampling structure, the advanced model can take advantage of hierarchical learning, while reduce the points to be processed in each layer.

6.4.2 Training

There are two strategies to train recurrent neural networks. The first one uses ground truths as inputs during predicting, which is known as **teacher-forcing** training. The second one works using predictions generated by the network as inputs

(Figure 6.2 (a)), which is referred to as **free-running** training. When using the teacher-forcing training, it shows that models quickly get stuck in a bad local optima, in which $\Delta \boldsymbol{P}_t$ tends to be $\mathbf{0}$ for all inputs. Therefore, the free-running strategy is used for training.

Since point clouds are unordered, point-to-point loss functions can not directly apply to compute the difference between the prediction and ground truth. Loss functions should be invariant to the order of input points. In this chapter, the Chamfer Distance (CD) and Earth Mover's Distance (EMD) are adopted. The CD between the point set $\boldsymbol{P}$ and $\boldsymbol{P}'$ is defined as follows,

$$\mathcal{L}_{CD}(\boldsymbol{P}, \boldsymbol{P}') = \sum_{\boldsymbol{p} \in \boldsymbol{P}} \min_{\boldsymbol{p}' \in \boldsymbol{P}'} \|\boldsymbol{p} - \boldsymbol{p}'\|^2 + \sum_{\boldsymbol{p}' \in \boldsymbol{P}'} \min_{\boldsymbol{p} \in \boldsymbol{P}} \|\boldsymbol{p}' - \boldsymbol{p}\|^2. \tag{6.5}$$

Basically, this loss function is a nearest neighbour distance metric that bidirectionally measures the error in two sets. Every point in $\boldsymbol{P}$ is mapped to the nearest point in $\boldsymbol{P}'$, and vice versa. The EMD from $\boldsymbol{P}$ to $\boldsymbol{P}'$ is defined as follows,

$$\mathcal{L}_{EMD}(\boldsymbol{P}, \boldsymbol{P}') = \min_{\phi: \boldsymbol{P} \to \boldsymbol{P}'} \sum_{\boldsymbol{p} \in \boldsymbol{P}} \|\boldsymbol{p} - \phi(\boldsymbol{p})\|^2, \tag{6.6}$$

where $\phi : \boldsymbol{P} \to \boldsymbol{P}'$ is a bijection. The EMD calculates a point-to-point mapping between two point clouds. The overall loss is as follows,

$$\mathcal{L}(\boldsymbol{P}, \boldsymbol{P}') = \alpha \mathcal{L}_{CD}(\boldsymbol{P}, \boldsymbol{P}') + \beta \mathcal{L}_{EMD}(\boldsymbol{P}, \boldsymbol{P}'), \tag{6.7}$$

where the hyperparameter $\alpha, \beta \geq 0$.

6.5 Experiments

I conduct experiments on one synthetic moving MNIST point cloud dataset and two large-scale real-world datasets, *i.e.*, Argoverse [108] and nuScenes [109]. Models are trained for 200k iterations using the Adam [104] optimizer with a learning rate of 10^{-5}. Gradients are clipped in the range $[-5, 5]$. The α and β in Eq. (6.7) are

set to 1.0. Following point cloud reconstruction [110, 111, 112], CD and EMD are used for quantity evaluation. Model details are listed in Table 6.1. Max pooling is used for the proposed models. For the two real-world datasets, because they contain too many points to be processed by the basic model, only the advanced model is evaluated. I randomly synthesize or sample 5,000 sequences from test subsets of these datasets as the test data.

Three baselines are devised for comparison. **1) Copy last input.** This method does not make predictions and just copies the last input in the given sequence as outputs. Models that outperform this method can prove their effectiveness to model point cloud sequence. **2) PointNet++ · LSTM.** This method first uses set abstraction (SA) layers from PointNet++ [79] to encode a point cloud to a global feature. Then, LSTMs take the global features as inputs and output predicted features. At last, like the advanced model, FP layers are used to propagate features to predicted displacements. **3) PointCNN · ConvLSTM**. The PointCNN [80] provides an $\mathcal{X}$-Conv operator that can weight and permute input points and features before they are processed by a typical convolution. Therefore, PointCNN and ConvLSTM can be combined for point cloud sequence processing by using $\mathcal{X}$-Conv to permute ConvLSTM states. This combination is evaluated with the advanced model.

6.5.1 Moving MNIST Point Cloud

Experiments on the synthetic Moving MNIST point cloud dataset can provide some basic understanding of the behavior of the proposed recurrent units. To synthesize moving MNIST digit sequences, a generation process similar to that described in [42] is used. Each synthetic sequence consists of 20 consecutive point clouds, with 10 for inputs and 10 for predictions. Each point cloud contains one or two potentially overlapping handwritten digits moving and bouncing inside a 64×64

area. The digits are chosen randomly from the MNIST dataset and placed initially at random locations. Each digit is assigned a velocity whose direction is chosen uniformly on a unit circle and whose magnitude is also chosen uniformly at random over a fixed range. Pixels whose brightness values (ranged from 0 to 255) are less than 16 are removed. Locations of the remaining pixels are transformed to (x, y) coordinates. The z-coordinate is set to 0 for all points. I randomly sample 128 points for one digit and 256 points for two digits as input, respectively. Batch size is set to 32.

Besides the three baselines, the proposed methods is also compared with two conventional video prediction models, *i.e.*, ConvLSTM [1] and CubicLSTM [67]. Essentially, the Moving MNIST point cloud dataset is 2D. Digit point clouds can be first voxelized to images and then apply conventional video based methods to process them. Specifically, a pixel in the voxelization image is set to 1 if there exists a point at the position. Otherwise, the pixel is set to 0. In this way, a 2D point cloud sequence is converted to a video. For training, the binary cross entropy loss is used to optimize each output pixel. For evaluation, because the number of output points is usually not consistent with that of input points, points whose brightness are in top 128 (for one digit) or top 256 (for two digits) are collected as the output point cloud.

Experimental results are listed in Table 6.2 (the proposed methods are with ball query). The kNN results are listed in Table 6.3. The performance of advanced models (ball query) with different numbers of layers is reported in Table 6.4. Two prediction examples are visualized in Figure 6.3 (the proposed models are with the advanced architecture and ball query). The following conclusions on the moving MNIST point cloud dataset can be drawn.

PointRNN, PointGRU and PointLSTM are superior to other methods. For ex-

Table 6.1 : Architecture specs. Each component (C) is described by four attributes, *i.e.*, number of output points (#pts), search radius (r), number of neighbors (k) and number of output channels (c). S: sampling. G: grouping. PU: PointRNN, PointGRU or PointLSTM unit. FP: feature propagation. FC: fully-connected layer.

C	Moving MNIST Point Cloud								Argoverse & nuScenes			
	basic model				advanced model				advanced model			
	#pts	r	k	c	#pts	r	k	c	#pts	r	k	c
S		-			$n/2$	1.0	4	-	$n/2$	0.5	8	-
PU	n	4.0	8	64	$n/2$	4.0	12	64	$n/2$	1.0	24	128
SG		-			$n/4$	2.0	4	-	$n/4$	1.0	8	-
PU	n	8.0	8	128	$n/4$	8.0	8	128	$n/4$	2.0	16	256
SG		-			$n/8$	4.0	4	-	$n/8$	2.0	8	-
PU	n	12.0	8	256	$n/8$	12.0	4	256	$n/8$	4.0	8	512
FP		-			$n/4$	-	-	128	$n/4$	-	-	256
FP		-			$n/2$	-	-	128	$n/2$	-	-	256
FP		-			n	-	-	128	n	-	-	256
FC	n	-	-	64	n	-	-	64	n	-	-	128
FC	n	-	-	3	n	-	-	3	n	-	-	3

ample, in Table 6.2, the CD of advanced PointLSTM on one-digit prediction is 1.16, less than ConvLSTM by 56.93, CubicLSTM by 8.35, PointNet++ · LSTM by 174.1 and PointCNN · ConvLSTM by 14.21.

Compared with the basic architecture, the advanced architecture significantly reduces computation. For example, in Table 6.2, for the one-digit prediction, the FLOPs of the basic PointGRU is 14.84 billion and the FLOPs of the advanced PointGRU is only 2.00 billion.

The advanced architecture also achieves lower prediction error than the basic architecture. For example, in Table 6.2, for the two-digit prediction, the CD of the

Table 6.2 : Prediction error (CD and EMD), #params (million) and FLOPs (billion) on the moving MNIST point cloud dataset.

Method		#params	One digit			Two digits		
			FLOPs	CD	EMD	FLOPs	CD	EMD
Copy last input		-	-	262.46	15.94	-	140.14	15.18
ConvLSTM [1] (voxel-based)		2.16	345.26	58.09	8.85	345.26	13.02	5.99
CubicLSTM [67] (voxel-based)		6.08	448.88	9.51	4.20	448.88	6.19	4.42
PointNet++ · LSTM		2.50	1.01	175.26	15.86	1.94	100.08	15.10
PointCNN · ConvLSTM		2.97	1.43	15.37	5.45	2.86	49.92	10.31
Basic	PointRNN	**0.27**	5.32	2.65	2.79	10.65	16.08	6.62
Basic	PointGRU	0.96	14.84	1.43	2.03	29.73	7.29	4.87
Basic	PointLSTM	1.22	24.74	1.40	2.00	49.55	7.28	4.82
Advance	PointRNN	0.36	**0.77**	1.78	2.32	**1.54**	11.44	5.87
Advance	PointGRU	1.04	2.00	1.18	1.85	4.00	5.79	4.49
Advance	PointLSTM	1.30	3.18	**1.16**	**1.78**	6.37	**5.18**	**4.21**

Table 6.3 : Prediction error of PointRNN, PointGRU and PointLSTM with kNN on the moving MNIST point cloud dataset.

Method		One digit		Two digits	
		CD	EMD	CD	EMD
Basic	PointRNN	5.86	3.76	22.12	7.79
Basic	PointGRU	2.61	2.53	19.74	7.64
Basic	PointLSTM	2.72	2.56	17.34	7.20
Advance	PointRNN	2.25	2.53	14.54	6.42
Advance	PointGRU	1.55	2.07	8.88	5.33
Advance	PointLSTM	**1.43**	**1.98**	**8.29**	**5.13**

basic PointRNN is 16.08 and the CD of the advanced PointRNN is 11.44.

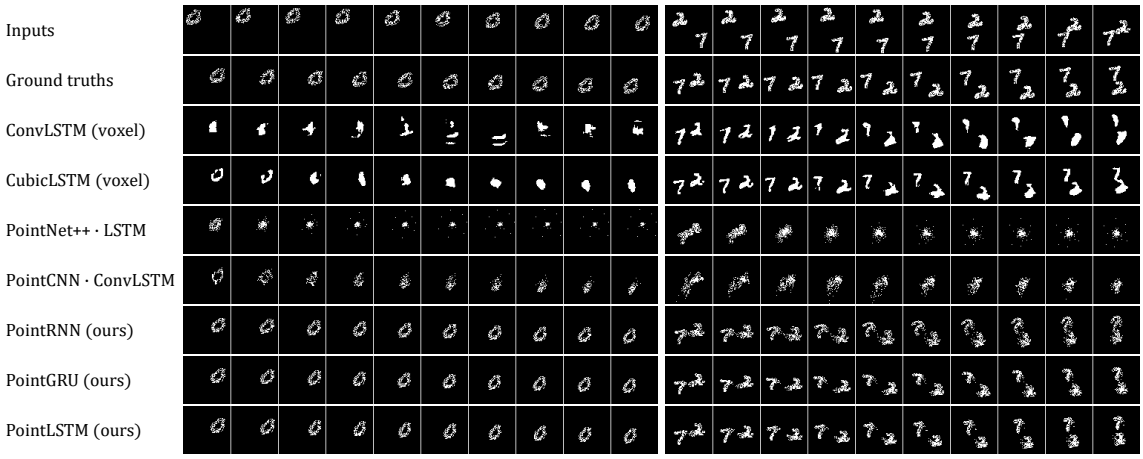


Figure 6.3 : Visualization of moving MNIST point cloud prediction. Left: one moving digit. Right: two moving digits.

PointGRU and PointLSTM are superior to PointRNN. However, it is difficult to declare a winner between PointGRU and PointLSTM. For example, in Table 6.2, for the two-digit prediction, the EMDs of the advanced PointGRU and PointLSTM are 4.49 and 4.21, respectively, which are similar and lower than that (5.87) of PointRNN.

The ball query is superior to the kNN. For example, when applying the advanced PointLSTM model to the two-digit prediction, the EMD of the ball query is 4.21 (in Table 6.2) and the EMD of the kNN is 5.13 (in Table 6.3).

Hierarchical structure by stacking multiple layers can effectively improve accuracy. For example, on the one-digit prediction, the CDs of the advanced PointGRU with one, two and three layers are 3.72, 1.48 and 1.18, respectively.

Learning via global features fails to make predictions. In Table 6.2, although the PointNet++ · LSTM combination obtains lower CDs than the copy-last-input method, their EMDs are similar. In Figure 6.3, PointNet++ · LSTM fails to predict neither appearance nor motion.

Point-based models consume less FLOPs than voxel-based models. Taking the

Table 6.4 : Performance of advanced models (ball query) with different numbers of layers on the moving MNIST point cloud dataset.

Method		#params	One digit			Two digits		
			FLOPs	CD	EMD	FLOPs	CD	EMD
1 layer	PointRNN	**0.025**	**0.176**	8.23	4.35	**0.354**	29.92	8.30
	PointGRU	0.051	0.454	**3.72**	**2.99**	0.912	19.41	**6.99**
	PointLSTM	0.060	0.710	4.34	3.17	1.427	**18.37**	7.14
2 layers	PointRNN	**0.109**	**0.485**	2.77	2.80	**0.972**	16.51	6.62
	PointGRU	0.268	1.224	**1.48**	**2.05**	2.453	8.22	5.13
	PointLSTM	0.328	1.962	1.50	2.06	3.931	**7.82**	**4.96**

two-digit prediction for instance, the basic PointLSTM model consumes only 49.55 billion FLOPs, while the voxel-based ConvLSTM consumes 345.26 billion FLOPs. The reason is that, the FLOPs of point-based models only depends on the number of input points, while voxel-based models have to process the entire space no matter how many points are actually in the space.

Voxel-based models are not sensitive to sparse point clouds. According to Table 6.2 and Figure 6.3, the predictions of ConvLSTM and CubicLSTM on one moving digit are worse than those on two moving digits, which is counterintuitive. One possible reason is that, for one-digit prediction, there is only $128/(64 \times 64) = 3.125\%$ point area, which is too sparse for voxel-based methods.

6.5.2 Argoverse and nuScenes

Argoverse [108] and nuScenes [109] are two large-scale autonomous driving datasets. The Argoverse data is collected by a fleet of autonomous vehicles in Pittsburgh (86km) and Miami (204km). The nuScenes data is recorded in Boston and Singapore, with 15h of driving data (242km travelled at an average of 16km/h). These

Table 6.5 : Details of the Argoverse and nuScenes datasets. #pcs: number of point clouds, #pts/pc: number of points per point cloud on average.

Dataset	trainval		test		frequency	#pts/pc	range
	#logs	#pcs	#logs	#pcs			
Argoverse [108]	89	18,211	24	4,189	10Hz	90,549	200m
nuScenes [109]	68	297,737	15	52,423	20Hz	34,722	70m

datasets are collected by multiple sensors, including LiDAR, RADAR, camera, *etc.* In this chapter, only the data from LiDAR sensors is used, without any human-annotated supervision, for moving point cloud prediction. Details about Argoverse and nuScenes are listed in Table 6.5.

Predicting moving point clouds on real-world driving datasets is considerably challenging. Content of a long driving point cloud sequence may change dramatically. Because we can not predict what are not provided in the given inputs, models are asked to make short-term prediction on the driving datasets. Each driving log is considered as a continuous point cloud sequence. 10 successive point clouds are randomly choiced from a driving log for training, with 5 for inputs and 5 for predictions. Since Argoverse and nuScenes are high-resolution, using all points requires considerable computation and running memory. Therefore, for each cloud, only the points whose coordinates are in the range $[-5m, 5m]$ are used. Then, 1,024 points are randomly sampled from the range as inputs and ground truths. Batch size is set to 4. Experimental results are listed in Table 6.6 (the proposed methods are with ball query). The result details over time are illustrated in Figure 6.4. The kNN results are listed in Table 6.7.

PointRNN, PointGRU and PointLSTM are superior to PointNet++ $\cdot$ LSTM and PointCNN $\cdot$ ConvLSTM. For example, in Table 6.6, the CD of advanced PointRNN

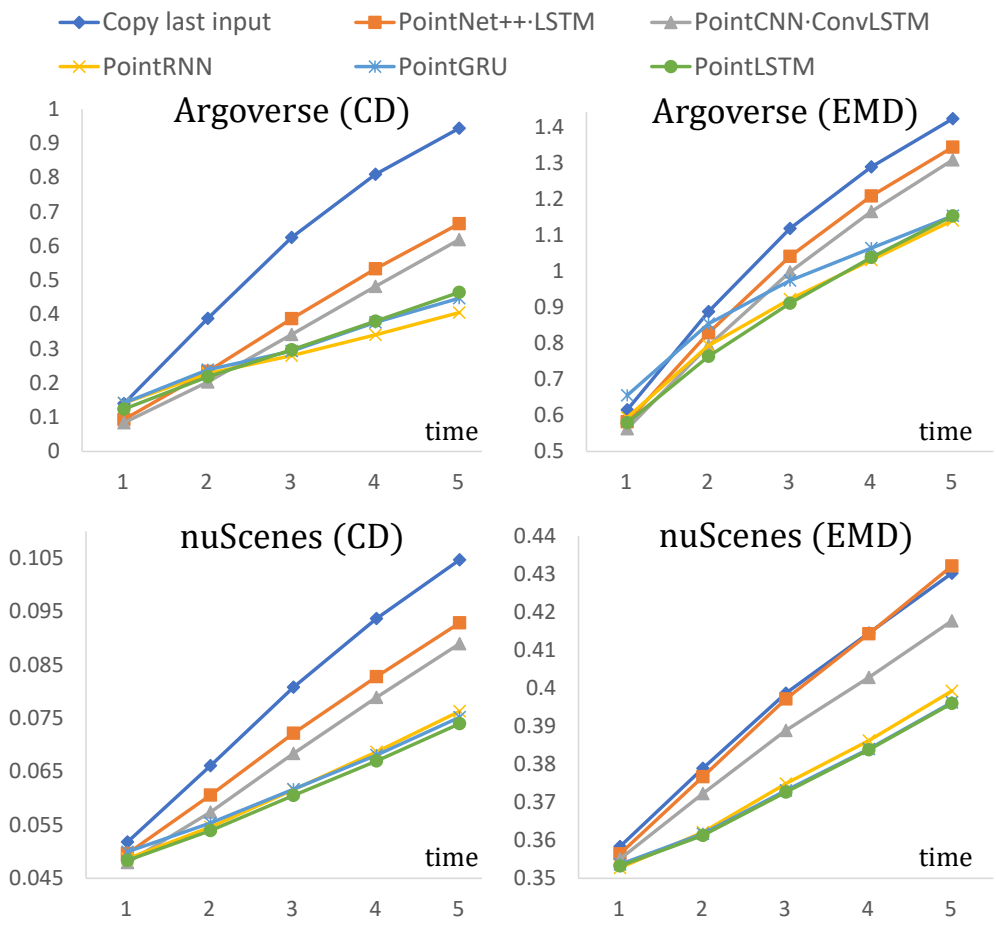


Figure 6.4 : Comparison of different methods over time.

on Argoverse is 0.2789, less than PointNet++ · LSTM by 0.1037 and PointCNN · ConvLSTM by 0.0668. In Figure 6.4, the proposed models achieve lower prediction error than PointNet++ · LSTM and PointCNN · ConvLSTM at most time steps.

The kNN and ball query achieve similar accuracy on Argoverse and nuScenes. For example, when applying the advanced PointLSTM model to Argoverse, the EMD of the ball query is 0.8892 (in Table 6.6) and the EMD of the kNN is 0.8856 (in Table 6.7).

Two prediction examples are visualized in Figure 6.5 (the proposed models are with the ball query). For the first example, PointNet++ · LSTM and PointCNN · ConvLSTM fail to predict car A or car B to move backward, while PointRNN,

Table 6.6 : Prediction error (CD and EMD), #params (million) and FLOPs (billion) on Argoverse and nuScenes.

Method	#params	FLOPs	Argoverse		nuScenes	
			CD	EMD	CD	EMD
Copy last input	-	-	0.5812	1.0667	0.0794	0.3961
PointNet++ · LSTM	9.98	26.54	0.3826	1.0011	0.0716	0.3953
PointCNN · ConvLSTM	12.04	24.17	0.3457	0.9659	0.0683	0.3874
PointRNN	**1.42**	**22.40**	**0.2789**	0.8964	**0.0619**	0.3750
PointGRU	4.15	59.49	0.2994	0.9084	0.0620	**0.3738**
PointLSTM	5.18	98.49	0.2966	**0.8892**	0.0624	0.3745

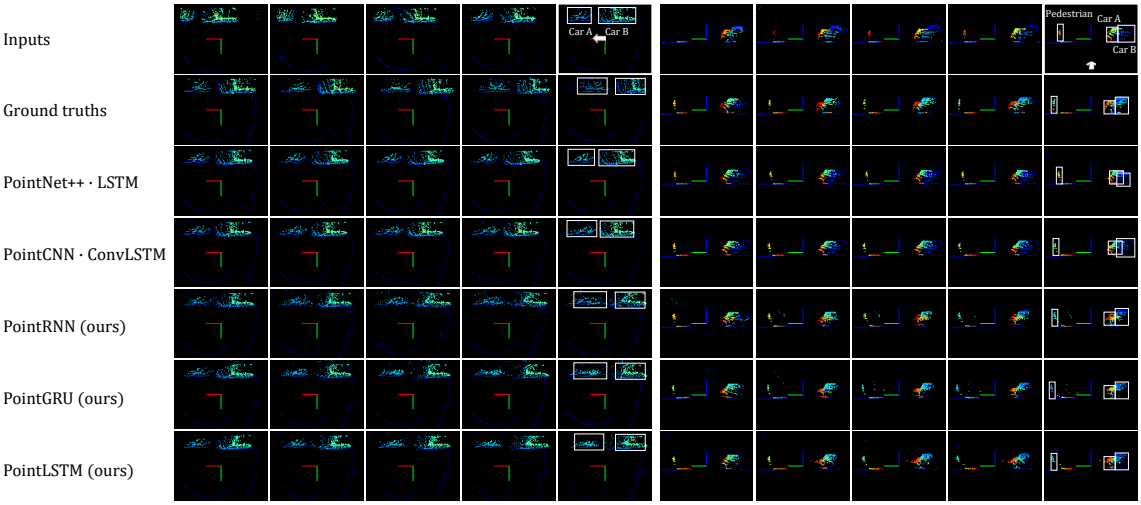


Figure 6.5 : Visualization of moving point cloud prediction on Argoverse. Left: an example of bird's-eye view (color encodes height). Right: an example of worm's-eye view (color encodes depth). Moving objects are marked with bounding boxes at the last time step.

PointGRU and PointLSTM can predict the movement of car B correctly and the movement of car A with a little error at the rear. For the second example, PointNet++ · LSTM fails to predict the movements of the pedestrian or the two cars.

Table 6.7 : Prediction error of PointRNN, PointGRU and PointLSTM with kNN on Argoverse and nuScenes.

Method	Argoverse		nuScenes	
	CD	EMD	CD	EMD
PointRNN	**0.2541**	**0.8743**	0.0627	0.3739
PointGRU	0.2922	0.9054	**0.0610**	**0.3711**
PointLSTM	0.2890	0.8856	0.0619	0.3730

The PointCNN · ConvLSTM model can predict the pedestrian to a certain degree, but fails to predict the cars. PointRNN, PointGRU and PointLSTM can predict the movements of both the pedestrian and the cars.

The predicted scene flow ($\Delta \boldsymbol{P}$) by the advanced PointLSTM model (ball query) is visualized in Figure 6.6. The point whose flow magnitude $||\Delta \boldsymbol{p}||$ is less than $0.01m$ is removed. The model outputs reasonable scene flow. This suggests a promising method to learn 3D scene flow from point clouds in an unsupervised manner.

6.6 Summary

This chapter extends RNN, GRU and LSTM to PointRNN, PointGRU and PointLSTM, respectively, for point cloud sequence processing. Based on the proposed recurrent units, a basic seq2seq model and an advanced seq2seq model are devised for point cloud video prediction. Experimental results on a synthetic dataset and two real-world datasets demonstrate their ability to model point cloud sequences. This chapter also shows the proposed method provides a promising unsupervised learning solution for 3D scene flow estimation. Additionally, the proposed PointRNN, PointGRU and PointLSTM have the potential for other temporal-related applications, such as 3D action recognition based on point cloud and sequential 3D

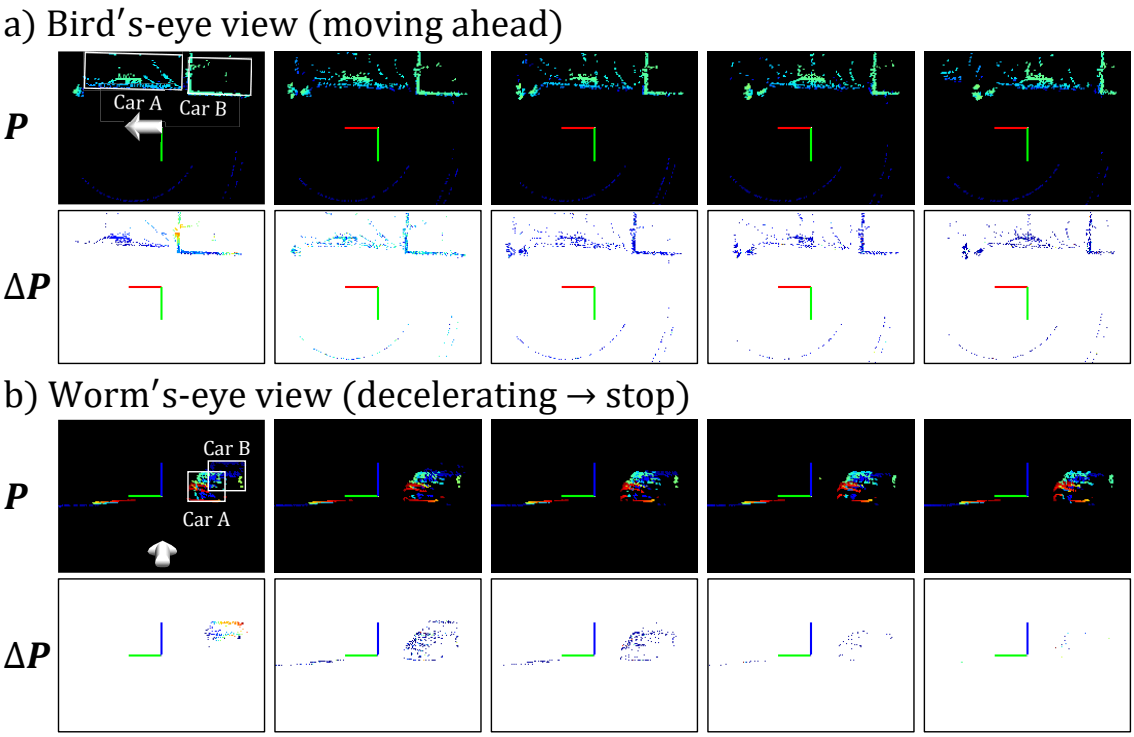


Figure 6.6 : Visualization of predicted scene flow ($\Delta \boldsymbol{P}$). The color of scene flow encodes flow magnitude. a) When the LiDAR is uniformly moving, the model generates similar scene flow. b) When the LiDAR stops moving, the scene flow vanishes.

scene semantic segmentation. For example, in the 3D scene semantic segmentation task, the proposed units can help intelligent agents to parse whether a car is moving or parking according to a few consecutive observations. More effective spatio-temporally correlation methods can be studied to improve PointRNN, PointGRU and PointLSTM.

Chapter 7

Conclusion

This dissertation presents four works about video modelling, with two works on video classification and two works on video prediction. The first video classification work focused on complex event detection in untrimmed videos, which can be seen as binary video classification problem. Because that only video-level labels are given, the proposed MIL-SRI method treat complex event detection as a weakly-supervised learning task and formulate it as an multi-instance learning problem. By treating each video as a bag and the video shots in the video as instances, MIL-SRI identifies reliable instances from positive and negative bags, rather than predicting instance labels, which simultaneously learns a linear SVM classifier and infers a binary indicator to indicates whether the instance is selected as a reliable instance or not. Experimental results on real-world datasets demonstrate the proposed MIL-SRI method effectively improve complex event detection accuracy in untrimmed videos. By extending MIL-SRI to multiple classification, the proposed method have a potential to be applied in to temporal action proposals (producing a set of candidate temporal segments that are likely to contain a human action) and temporal action localization (temporally localizing activities in untrimmed video sequences, where videos can contain more than one activity instance, and multiple activity categories can appear in the video).

The second video classification work focuses on the efficiency problem of video classification. Motivated by humans who classify videos by watching a very small portion of frames, a reinforcement learning based method is proposed for efficient

video classification. The proposed method consists of a "fast forward" mechanism and and an "adaptive stop" mechanism. when finding the current frame does not add any helpful information to classify the video, "fast forward" plays the video in a faster speed. When finding informative frames, "fast forward" slow downs the watching speed and look into details. When confident enough, "adaptive stop" stop watching the video and the model emits the classification decision. By the "fast forward" and and "adaptive stop" mechanisms, the computational cost for video classification is significantly reduced while maintaining the similar accuracy. To improve the accuracy, a promising method is to avoid the noise from watched irrelevant frames being merged into the model state, which can be studied in future.

For video prediction, the first work focuses on conventional 2D RGB videos. To reduce the burden of deep neural networks for spato-temporal modelling, a CubicLSTM unit is developed which processes spatio-temporal information separately by a spatial branch and a temporal branch. The spatial branch recognizes and extracts moving objects while the temporal branch captures and predicts motions. The sequence-to-sequence architecture, which stacks multiple CubicLSTM units, generates better predictions than existing models, validating the motivation that the spatial information and temporal information should be processed separately. In order to further improve prediction accuracy, hierarchical architectures can be devised with CubicLSTMs to better capture the spatio-temporal structure, rather than simple stack multiple units.

The second video prediction work extends prediction to moving 3D point cloud sequences, which has been largely ignored by the community. To model the spatio-temporal correlations in 3D point cloud videos, a series of point spatio-temporal recurrent neural networks, including PointRNN, PointGRU and PointLSTM, are proposed. To apply PointRNN, PointGRU and PointLSTM to 3D point cloud video prediction, based on the ssequence-to-sequence architecture, a basic model and an

advanced model are developed. Experimental results on a synthetic dataset and two real-world self-driving datasets demonstrate the proposed method is able to effectively model point cloud videos. The proposed units have the potential for other temporal-related applications, such as 3D action recognition based on point cloud and sequential scene semantic segmentation. More effective spatiotemporally-local correlation methods can be studied to improve PointRNN, PointGRU and PointLSTM, which can be further studied in future.

Bibliography

[1] X. Shi, Z. Chen, H. Wang, D. Yeung, W. Wong, and W. Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, 2015, pp. 802–810.

[2] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, 2014, pp. 3104–3112.

[3] C. Finn, I. J. Goodfellow, and S. Levine, "Unsupervised learning for physical interaction through video prediction," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 64–72.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, 2012, pp. 1106–1114.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, 2015, pp. 1–9.

[7] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *arXiv*, vol. 1512.00567, 2015.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 770–778.

[9] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *arXiv*, vol. 1602.07360, 2016.

[10] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*, R. C. Wilson, E. R. Hancock, and W. A. P. Smith, Eds., 2016.

[11] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, 2017, pp. 2261–2269.

[12] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, 2017, pp. 5987–5995.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv*, vol. 1412.3555, 2014.

[15] R. B. Girshick, "Fast R-CNN," in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, 2015, pp. 1440–1448.

[16] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, 2015, pp. 3431–3440.

[17] H. Fan, L. Zheng, C. Yan, and Y. Yang, "Unsupervised person re-identification: Clustering and fine-tuning," *ACM Transactions on Multimedia Computing, Communications, and Applications TOMM*, vol. 14, no. 4, pp. 83:1–83:18, 2018.

[18] Q. Feng, G. Kang, H. Fan, and Y. Yang, "Attract or distract: Exploit the margin of open set," in *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, 2019, pp. 7989–7998.

[19] Q. Feng, Y. Wu, H. Fan, C. Yan, M. Xu, and Y. Yang, "Cascaded revision network for novel object captioning," *IEEE Transactions on Circuits and Systems for Video Technology*, 2020.

[20] Y. Ding, H. Fan, M. Xu, and Y. Yang, "Adaptive exploration for unsupervised person re-identification," *ACM Trans. Multim. Comput. Commun. Appl.*, vol. 16, no. 1, pp. 3:1–3:19, 2020.

[21] H. Fan, L. Zhu, , Y. Yang, and F. Wu, "Recurrent attention network with reinforced generator for visual dialog," *ACM Transactions on Multimedia Computing, Communications, and Applications TOMM*, 2020.

[22] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[23] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds., 2019, pp. 4171–4186.

[24] H. Sak, A. W. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*, H. Li, H. M. Meng, B. Ma, E. Chng, and L. Xie, Eds., 2014, pp. 338–342.

[25] C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina, N. Jaitly, B. Li, J. Chorowski, and M. Bacchiani, "State-of-the-art speech recognition with sequence-to-sequence models," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018, Calgary, AB, Canada, April 15-20, 2018*, 2018, pp. 4774–4778.

[26] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, 2014, pp. 568–576.

[27] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. V. Gool, "Temporal segment networks: Towards good practices for deep action recognition," in *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII*, 2016, pp. 20–36.

[28] C. Feichtenhofer, A. Pinz, and A. Zisserman, "Convolutional two-stream network fusion for video action recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*,

2016, pp. 1933–1941.

[29] D. Tran, L. D. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, 2015, pp. 4489–4497.

[30] J. Carreira and A. Zisserman, "Quo vadis, action recognition? A new model and the kinetics dataset," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, 2017, pp. 4724–4733.

[31] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, "A closer look at spatiotemporal convolutions for action recognition," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, 2018, pp. 6450–6459.

[32] K. Hara, H. Kataoka, and Y. Satoh, "Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet?" in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, 2018, pp. 6546–6555.

[33] H. Fan and Y. Yang, "Person tube retrieval via language description," in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, 2020, pp. 10 754–10 761.

[34] X. Li and M. C. Chuah, "SBGAR: semantics based group activity recognition," in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, 2017, pp. 2895–2904.

[35] ——, "Rehar: Robust and efficient human activity recognition," in *2018 IEEE Winter Conference on Applications of Computer Vision, WACV 2018, Lake Tahoe, NV,*

USA, March 12-15, 2018, 2018, pp. 362–371.

[36] J. Y. Ng, M. J. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: Deep networks for video classification," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, 2015, pp. 4694–4702.

[37] S. Yeung, O. Russakovsky, G. Mori, and L. Fei-Fei, "End-to-end learning of action detection from frame glimpses in videos," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 2678–2687.

[38] L. Zhu, Z. Xu, and Y. Yang, "Bidirectional multirate reconstruction for temporal modeling in videos," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, 2017, pp. 1339–1348.

[39] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks for action segmentation and detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, 2017, pp. 1003–1012.

[40] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv*, vol. 1803.01271, 2018.

[41] M. Ranzato, A. Szlam, J. Bruna, M. Mathieu, R. Collobert, and S. Chopra, "Video (language) modeling: a baseline for generative models of natural videos," *CoRR*, vol. abs/1412.6604, 2014.

[42] N. Srivastava, E. Mansimov, and R. Salakhutdinov, "Unsupervised learning of video representations using lstms," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, 2015, pp. 843–852.

[43] C. Vondrick, H. Pirsiavash, and A. Torralba, "Anticipating visual representations

from unlabeled video," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 98–106.

[44] S. E. Reed, A. van den Oord, N. Kalchbrenner, S. G. Colmenarejo, Z. Wang, Y. Chen, D. Belov, and N. de Freitas, "Parallel multiscale autoregressive density estimation," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, 2017, pp. 2912–2921.

[45] X. Jia, B. D. Brabandere, T. Tuytelaars, and L. V. Gool, "Dynamic filter networks," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 667–675.

[46] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, 2015, pp. 2017–2025.

[47] C. Vondrick and A. Torralba, "Generating the future with adversarial transformers," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, 2017, pp. 2992–3000.

[48] R. Villegas, J. Yang, S. Hong, X. Lin, and H. Lee, "Decomposing motion and content for natural video sequence prediction," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[49] E. L. Denton and V. Birodkar, "Unsupervised learning of disentangled representations from video," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 4414–4423.

[50] S. Tulyakov, M. Liu, X. Yang, and J. Kautz, "Mocogan: Decomposing motion and content for video generation," in *2018 IEEE Conference on Computer Vision and*

Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018, 2018, pp. 1526–1535.

[51] T. Xue, J. Wu, K. L. Bouman, and B. Freeman, "Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 91–99.

[52] N. Kalchbrenner, A. van den Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, and K. Kavukcuoglu, "Video pixel networks," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, 2017, pp. 1771–1779.

[53] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. P. Singh, "Action-conditional video prediction using deep networks in atari games," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, 2015, pp. 2863–2871.

[54] C. Vondrick, H. Pirsiavash, and A. Torralba, "Generating videos with scene dynamics," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 613–621.

[55] Y. Wang, M. Long, J. Wang, Z. Gao, and P. S. Yu, "Predrnn: Recurrent neural networks for predictive learning using spatiotemporal lstms," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 879–888.

[56] W. Lotter, G. Kreiman, and D. D. Cox, "Deep predictive coding networks for video prediction and unsupervised learning," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[57] R. Mottaghi, H. Bagherinezhad, M. Rastegari, and A. Farhadi, "Newtonian image understanding: Unfolding the dynamics of objects in static images," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 3521–3529.

[58] M. Mathieu, C. Couprie, and Y. LeCun, "Deep multi-scale video prediction beyond mean square error," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[59] J. Walker, C. Doersch, A. Gupta, and M. Hebert, "An uncertain future: Forecasting from static images using variational autoencoders," in *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VII*, 2016, pp. 835–851.

[60] Y. Zhou and T. L. Berg, "Learning temporal transformations from time-lapse videos," in *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII*, 2016, pp. 262–277.

[61] W. Xiong, W. Luo, L. Ma, W. Liu, and J. Luo, "Learning to generate time-lapse videos using multi-stage dynamic generative adversarial networks," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, 2018, pp. 2364–2373.

[62] S. Chiappa, S. Racanière, D. Wierstra, and S. Mohamed, "Recurrent environment simulators," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[63] M. Babaeizadeh, C. Finn, D. Erhan, R. H. Campbell, and S. Levine, "Stochastic variational video prediction," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[64] K. Fragkiadaki, J. Huang, A. Alemi, S. Vijayanarasimhan, S. Ricco, and R. Sukthankar, "Motion prediction under multimodality with conditional stochastic networks," *CoRR*, vol. abs/1705.02082, 2017.

[65] K. D. Tang, F. Li, and D. Koller, "Learning latent temporal structure for complex event detection," in *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, 2012, pp. 1250–1257.

[66] H. Fan, Z. Xu, L. Zhu, C. Yan, J. Ge, and Y. Yang, "Watching a small portion could be as good as watching all: Towards efficient video classification," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, 2018, pp. 705–711.

[67] H. Fan, L. Zhu, and Y. Yang, "Cubic lstms for video prediction," in *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, 2019, pp. 8263–8270.

[68] Y. Cheng, Q. Fan, S. Pankanti, and A. N. Choudhary, "Temporal sequence modeling for video event detection," in *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, 2014, pp. 2235–2242.

[69] X. Chang, Y. Yu, Y. Yang, and E. P. Xing, "Semantic pooling for complex event analysis in untrimmed videos," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 8, pp. 1617–1632, 2017.

[70] Y. Yan, Y. Yang, D. Meng, G. Liu, W. Tong, A. G. Hauptmann, and N. Sebe, "Event oriented dictionary learning for complex event detection," *IEEE Trans. Image Processing*, vol. 24, no. 6, pp. 1867–1878, 2015.

[71] W. Li, Q. Yu, A. Divakaran, and N. Vasconcelos, "Dynamic pooling for complex event recognition," in *IEEE International Conference on Computer Vision, ICCV*

2013, Sydney, Australia, December 1-8, 2013, 2013, pp. 2728–2735.

[72] K. Lai, F. X. Yu, M. Chen, and S. Chang, "Video event detection by inferring temporal instance labels," in *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, 2014, pp. 2251–2258.

[73] F. X. Yu, D. Liu, S. Kumar, T. Jebara, and S. Chang, "\(\propto\)svm for learning with label proportions," in *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, 2013, pp. 504–512.

[74] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

[75] N. P. Jouppi, C. Young, N. Patil, D. A. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*, 2017, pp. 1–12.

[76] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk,

and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, 2014, pp. 1724–1734.

[77] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *J. Mach. Learn. Res.*, vol. 3, pp. 1137–1155, 2003.

[78] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, 2017, pp. 77–85.

[79] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 5099–5108.

[80] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "Pointcnn: Convolution on x-transformed points," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, 2018, pp. 828–838.

[81] C. R. Qi, O. Litany, K. He, and L. J. Guibas, "Deep hough voting for 3d object detection in point clouds," *CoRR*, vol. abs/1904.09664, 2019.

[82] S. Andrews, I. Tsochantaridis, and T. Hofmann, "Support vector machines for multiple-instance learning," in *Advances in Neural Information Processing Systems 15 [Neural Information Processing Systems, NIPS 2002, December 9-14, 2002, Vancouver, British Columbia, Canada]*, 2002, pp. 561–568.

[83] W. Li, L. Duan, D. Xu, and I. W. Tsang, "Text-based image retrieval using progressive multi-instance learning," in *IEEE International Conference on Computer Vi-*

sion, ICCV 2011, Barcelona, Spain, November 6-13, 2011, D. N. Metaxas, L. Quan, A. Sanfeliu, and L. V. Gool, Eds., 2011, pp. 2049–2055.

[84] Y. Li, J. T. Kwok, I. W. Tsang, and Z. Zhou, "A convex method for locating regions of interest with multi-instance learning," in *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2009, Bled, Slovenia, September 7-11, 2009, Proceedings, Part II*, W. L. Buntine, M. Grobelnik, D. Mladenic, and J. Shawe-Taylor, Eds., vol. 5782, 2009, pp. 15–30.

[85] R. C. Bunescu and R. J. Mooney, "Multiple instance learning for sparse positive bags," in *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, Z. Ghahramani, Ed., vol. 227, 2007, pp. 105–112.

[86] G. Liu, J. Wu, and Z. Zhou, "Key instance detection in multi-instance learning," in *Proceedings of the 4th Asian Conference on Machine Learning, ACML 2012, Singapore, Singapore, November 4-6, 2012*, S. C. H. Hoi and W. L. Buntine, Eds., vol. 25, 2012, pp. 253–268.

[87] W. Li and N. Vasconcelos, "Multiple instance learning for soft bags via top instances," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, 2015, pp. 4277–4285.

[88] D. Zhang, D. Meng, C. Li, L. Jiang, Q. Zhao, and J. Han, "A self-paced multiple-instance learning framework for co-saliency detection," in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, 2015, pp. 594–602.

[89] X. Yang, Q. Song, and Y. Wang, "A weighted support vector machine for data classification," *IJPRAI*, vol. 21, no. 5, pp. 961–976, 2007.

[90] L. Jiang, D. Meng, S. Yu, Z. Lan, S. Shan, and A. G. Hauptmann, "Self-paced learning with diversity," in *Advances in Neural Information Processing Systems 27:*

Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada, 2014, pp. 2078–2086.

[91] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and F. Li, "Large-scale video classification with convolutional neural networks," in *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, 2014, pp. 1725–1732.

[92] A. Habibian, T. Mensink, and C. G. M. Snoek, "Videostory: A new multimedia embedding for few-example recognition and translation of events," in *Proceedings of the ACM International Conference on Multimedia, MM '14, Orlando, FL, USA, November 03 - 07, 2014*, 2014, pp. 17–26.

[93] K. Lai, D. Liu, M. Chen, and S. Chang, "Recognizing complex events in videos by learning key static-dynamic evidences," in *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part III*, 2014, pp. 675–688.

[94] Z. Lan, M. Lin, X. Li, A. G. Hauptmann, and B. Raj, "Beyond gaussian pyramid: Multi-skip feature stacking for action recognition," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, 2015, pp. 204–212.

[95] M. Nagel, T. Mensink, and C. G. M. Snoek, "Event fisher vectors: Robust encoding visual diversity of visual streams," in *Proceedings of the British Machine Vision Conference 2015, BMVC 2015, Swansea, UK, September 7-10, 2015*, 2015, pp. 178.1–178.12.

[96] Z. Xu, Y. Yang, and A. G. Hauptmann, "A discriminative CNN video representation for event detection," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, 2015, pp. 1798–1807.

[97] S. Zha, F. Luisier, W. Andrews, N. Srivastava, and R. Salakhutdinov, "Exploiting image-trained CNN architectures for unconstrained video classification," in *Proceed-*

ings of the British Machine Vision Conference 2015, BMVC 2015, Swansea, UK, September 7-10, 2015, 2015, pp. 60.1–60.13.

[98] H. Wang and C. Schmid, "Action recognition with improved trajectories," in *IEEE International Conference on Computer Vision, ICCV 2013, Sydney, Australia, December 1-8, 2013*, 2013, pp. 3551–3558.

[99] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

[100] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, 2016, pp. 1928–1937.

[101] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan, "Youtube-8m: A large-scale video classification benchmark," *CoRR*, vol. abs/1609.08675, 2016.

[102] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 2818–2826.

[103] C. Schüldt, I. Laptev, and B. Caputo, "Recognizing human actions: A local SVM approach," in *17th International Conference on Pattern Recognition, ICPR 2004, Cambridge, UK, August 23-26, 2004*, 2004, pp. 32–36.

[104] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[105] N. Kalchbrenner, I. Danihelka, and A. Graves, "Grid long short-term memory," in *4th International Conference on Learning Representations, ICLR 2016, San Juan,*

Puerto Rico, May 2-4, 2016, Conference Track Proceedings, 2016.

[106] M. F. Stollenga, W. Byeon, M. Liwicki, and J. Schmidhuber, "Parallel multi-dimensional lstm, with application to fast biomedical volumetric image segmentation," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, 2015, pp. 2998–3006.

[107] H. Fan and Y. Yang, "Pointrnn: Point recurrent neural network for moving point cloud processing," *CoRR*, vol. abs/1910.08287, 2019.

[108] M. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays, "Argoverse: 3d tracking and forecasting with rich maps," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, 2019, pp. 8748–8757.

[109] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," *CoRR*, vol. abs/1903.11027, 2019.

[110] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. J. Guibas, "Learning representations and generative models for 3d point clouds," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, 2018, pp. 40–49.

[111] L. Yu, X. Li, C. Fu, D. Cohen-Or, and P. Heng, "Pu-net: Point cloud upsampling network," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, 2018, pp. 2790–2799.

[112] P. Mandikal and V. B. Radhakrishnan, "Dense 3d point cloud reconstruction using a deep pyramid network," in *IEEE Winter Conference on Applications of Computer Vision, WACV 2019, Waikoloa Village, HI, USA, January 7-11, 2019*, 2019, pp. 1052–1060.