Efficient Video Classification using Fewer Frames

A THESIS

submitted by

SHWETA BHARDWAJ

for the award of the degree

of

MASTER OF SCIENCE

(by Research)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

August 2019

THESIS CERTIFICATE

This is to certify that the thesis titled **Efficient Video Classification Using Fewer Frames**, submitted by **Shweta Bhardwaj**, to the Indian Institute of Technology Madras, for the award of the degree of **Master of Science (by Research)**, is a bona fide record of the research work done by her under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Mitesh M. Khapra
Research Guide
Professor
Dept. of CSE
IIT Madras, 600 036

Place: Chennai

Date:

ACKNOWLEDGEMENTS

First and foremost, I thank my parents and my brother Shubhendu for their constant love and encouragement. I am extremely lucky that they have always supported me in all my pursuits. I would like to express my sincere gratitude towards my advisor Dr. Mitesh M. Khapra. I deeply appreciate his contributions in providing research guidance, valuable time and funding to make my M.S work more productive. All of his students are always astonished by the amount of passion and effort he puts in for the work. I am deeply influenced by the way he structures his thoughts and conveys complex concepts. I have always enjoyed his courses, which helped in expanding my horizons of knowledge and understanding. I can't thank him enough for the constant genuine support and understanding beyond the context of research. Many a time he has stayed back during the late hours for the insightful discussions about my work. As an advisor and an inspiring person, he has always encouraged his students to walk an extra mile for their goals, even during the tough times.

I would like to thank my General Test Committee (GTC) members: Prof. C. Chandra Sekhar, Prof. N.S. Narayanaswamy and Prof. Lelitha Devi for critiquing my research work. I am extremely grateful to Prof. Balaraman Ravindran for being a wonderful collaborator and teacher, especially for his inspirational discussions regarding the experiments in my work.

The members of the RISE lab have immensely contributed to my personal and professional life at IIT Madras. I am immensely grateful to my lab senior Deepak Mittal for encouraging me in walking the baby steps of research during the initial days of M.S journey. I am very fortunate to find amazing friends in the lab: Ananya, Deepak, Gurneet, Nikita, Priyesh, Siddarth and Tarun. I am grateful to them for listening whenever I needed, for spending a wonderful time on our memorable trips to mountains and for not-so-short tea breaks. I am lucky to find a great friend and brother in Stanley who always believed in me and encouraged me to put in my best efforts in every situation. My lab juniors: Aakriti, Madhura, Nitesh and Pritha have always been very sweet, extremely helpful and great conversationalists.

My time was made even more enjoyable by my M.S batchmates: Himanshi, Indrani, Puneet, Saptakatha, Sayantan and Suman with their amazing birthday memories. I also appreciate my friends outside the lab: Kanika and Shikha, who have been an awesome company during these years.

I also thank Ashish Mishra for great discussions about the domain of Computer Vision and research ideas. I would like to extend a special thanks to my labmates Depen and Rahul for the insightful reviews about my research work. I am thankful for the useful discussions and CoTAing with fantastic people like Abhishek, Jaya, Mohan, Preksha, Sanchit, Shubham, and Soham. I am also thankful to our GPU management team, especially Karthik Thiagarajan, for their contributions and valuable time.

In regards to academic work, I am extremely thankful to CS office staff and assistants for their help and service, especially Ramya. Last but not least, I thank God for placing all of these people in my life.

ABSTRACT

KEYWORDS: Video Classification, Efficient Computation, Deep Learning, Reinforcement Learning, Knowledge Distillation, Frame Selection, Model Compression

Recently, there has been a lot of interest in building compact models for video classification which have a small memory footprint ($<$ 1 GB) (Joonseok *et al.*, 2018). While these models are compact, they typically operate by repeated application of a small weight matrix to all the frames in a video. For example, recurrent neural network based methods compute a hidden state for *every* frame of the video using a recurrent weight matrix. Similarly, *cluster-and-aggregate* based methods such as *NetVLAD* have a learnable clustering matrix which is used to assign soft-clusters to *every* frame in the video. Since these models look at every frame in the video, the number of floating point operations (FLOPs) is still large even though the memory footprint is small. In this work, we focus on building compute-efficient video classification models which process fewer frames and hence, have less number of FLOPs. To achieve this goal, we use the idea of *Knowledge Distillation* albeit in a different setting. Specifically, a compute-heavy teacher which looks at all the frames in the video is used to train a compute-efficient student which looks at only a small fraction of frames in the video. This is in contrast to a typical memory-efficient *Teacher-Student* setting, wherein both the teacher and the student look at all the frames in the video but the student has fewer parameters. Our work thus complements the research on memory-efficient video classification. We do an extensive evaluation with three types of models for video classification, *viz.*, (i) recurrent models (ii) cluster-and-aggregate models and (iii) memory-efficient cluster-and-aggregate models and show that in each of these cases, a see-it-all teacher can be used to train a compute-efficient see-very-little student. The student network, in this case, looks at a fixed number of uniformly spaced frames in the video. We evaluate our model on YouTube-8M dataset and empirically demonstrate that the proposed student network can reduce the inference time by 30% and the number of FLOPs by approximately 90% with a negligible drop in the performance.

Encouraged by these results, we ask the next logical question: “Instead of uniformly sampling frames from the video, can we dynamically select the most important frames in the video?” To do so, we train a reinforcement learning (RL) agent to select the best frames which improve the final reward in terms of the classification loss. However, we observe that the model learns to select frames that are located in the neighborhood of uniformly spaced frames. On further investigation, we realize that this may be due to the specific nature of the dataset that we used which contains multiple labels per video. These labels correspond to different portions of the video and hence it may be prudent to look at frames across the entire video instead of zooming into a particular fragment. In other words, given a budget for the number of frames to be selected, it seems it is hard for the RL agent to do something better than simply picking up approximately uniformly spaced frames.

ABBREVIATIONS

AI	Artificial Intelligence
A3C	Asynchronous Advantage Actor-Critic algorithm
BPTT	Backpropagation Through Time
CVPR	Computer Vision and Pattern Recognition
CG	Context Gating
CNN	Convolution Neural Network
DL	Deep Learning
ECCV	Europian Conference On Computer Vision
FLOP	Floating Point Operation
FV	Fisher Vector
GAP	Global Average Precision
GPU	Graphics Processing Unit
HOG	Histogram Of Oriented Gradients
IITM	Indian Institute of Technology, Madras
LSTM	Long Short Term Memory
MDP	Markov Decision Process
mAP	Mean Average Precision
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
RL	Reinforcement Learning
ResNet	Residual Network
SIFT	Scale Invariant Feature Transform
TSNE	T-distributed Stochastic Neighbor Embedding
VLAD	Vector of Locally Aggregated Descriptors

NOTATION

Σ	Summation over a set
σ	Logistic sigmoid function
$\mathbb{E}$	Expectation
$\mathcal{L}$	Loss function
K	Filter matrix in CNN
L_2	Euclidean distance or norm
$\mathscr{E}$	Entropy function
ϕ	Activation function
η	Learning rate or step-size
θ	Parameters or weights
∇_θ	Gradient w.r.t θ parameters
∂	Partial derivative
$\mathscr{Y}$	Set of classes
τ	Temperature parameter
$\mathcal{E}$	Video encoding
$\mathcal{I}$	Intermediate encoding of a video
$\mathcal{M}$	Markov Decision Process
$\mathcal{P}$	Probability distribution
$\mathcal{R}$	Reward function in RL
$\mathcal{J}$	Performance function in RL
$\mathcal{S}$	State space in RL
$\mathcal{A}$	Action space in RL
π	Policy (action probability distribution) in RL
γ	Degree of exploration in RL
Eq.	Equation
w.r.t	With respect to
i.e.	That is
etc.	Et cetera

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . **iii**

ABSTRACT . . . **v**

ABBREVIATIONS . . . **vii**

NOTATION . . . **ix**

LIST OF TABLES . . . **xv**

LIST OF FIGURES . . . **xviii**

1 Introduction . . . **1**

1.1 Video Classification Pipeline . . . 2

1.2 Efficient Video Classification . . . 3

1.3 Problem Statement . . . 4

1.4 Contributions . . . 5

1.5 Applications . . . 5

1.6 Organisation of Thesis . . . 6

2 Background . . . **9**

2.1 Feed Forward Neural Network . . . 9

2.1.1 Learning Algorithm . . . 10

2.2 Convolutional Neural Networks (CNN) . . . 11

2.2.1 Convolution Operation . . . 11

2.2.2 Pooling . . . 12

2.2.3 Fully Connected Layer . . . 12

2.3 Recurrent Neural Network (RNN) . . . 13

2.3.1 Long Short Temporal Memory Networks (LSTMs) . . . 14

2.3.2 Hierarchical Recurrent Neural Network . . . 15

2.4 VLAD and NetVLAD Feature Encoding . . . 16

2.5 Knowledge Distillation 18
2.6 Reinforcement Learning (RL) 19
2.6.1 Formulation 19
2.6.2 Policy Gradients 20
2.7 Summary 21

3 Literature Survey 23
3.1 Memory-Efficient Models 23
3.2 Video Classification Datasets 25
3.3 Feature Extraction From Images 26
3.4 Feature Aggregation For Video Classification 27
3.4.1 Single Stream Networks 27
3.4.2 3D Convolution Based Networks 28
3.4.3 Multi-Stream Networks 29
3.4.4 Encoder-Decoder Based Methods 30
3.4.5 Cluster-Aggregation Based Methods 31
3.5 Video Classification On YouTube-8M Dataset 32
3.6 Reinforcement Learning For Videos 33
3.7 Summary 34

4 A Teacher-Student Framework for Video Classification 35
4.1 Video Classification Models 36
4.1.1 Recurrent Network Based Models 36
4.1.2 Cluster and Aggregate Based Models 37
4.2 Proposed Approach 37
4.2.1 Teacher Network 38
4.2.2 Student Network: Final Matching 38
4.2.3 Student Network: Intermediate Matching 39
4.2.4 Knowledge Distillation on Class Probabilities 39
4.3 Baseline Models 40
4.4 Different Training Paradigms 40
4.5 Experimental Setup 41
4.5.1 Dataset 43

4.5.2 Hyperparameters: 43
4.5.3 Evaluation Metrics 44
4.6 Results And Discussion 44
4.6.1 Comparison of Different Baselines 45
4.6.2 Comparing Teacher-Student Network with Uniform-k Baseline 45
4.6.3 Serial Versus Parallel Training of Teacher-Student 46
4.6.4 Visualization of Teacher and Student Representations 47
4.6.5 Matching Intermediate and Final Representations 47
4.6.6 Computation time of different models 48
4.6.7 Performance using *NetVLAD* models 48
4.6.8 Combining with memory efficient models 49
4.7 Summary 49

5 Dynamic Frame Selection **51**
5.1 Overview of Proposed Approach 51
5.2 Modelling 52
5.2.1 Video-Encoding Network 52
5.2.2 Classification Network 53
5.2.3 Reinforcement Learning Agent 53
5.3 Training 55
5.3.1 Policy Gradients 57
5.3.2 Exploration 58
5.3.3 Designing Reward 58
5.4 Experimental Setup 59
5.4.1 Baselines 59
5.4.2 Dataset and Evaluation 59
5.4.3 Hyperparameters 60
5.5 Results and Discussion 60
5.5.1 Performance Comparison of *Baselines* and *SkipFrame* . . . 61
5.5.2 Frame Selection 62
5.5.3 Action Distributions 63
5.5.4 Label Distribution 63
5.5.5 Computation Cost 64

5.6 Summary 65

6 Conclusions and Future Work **67**

6.1 Conclusions 67

LIST OF TABLES

3.1 Standard datasets for benchmarking video classification methods . . 26

4.1 Performance comparison of proposed *Teacher-Student* models using different ***Student-Loss*** variants, with their corresponding baselines using k frames. ***Teacher-Skyline*** is the default model which processes all the frames in a video, and achieves a GAP and mAP score of **0.811** and **0.414** respectively. 45

4.2 Effect of amount of training data on performance of Serial and Uniform models using 30 frames 46

4.3 Comparison of *Final* and *Intermediate* representation matching by *Student* network using k=30 frames 48

4.4 Comparison of FLOPs and evaluation time of models using k frames with *Skyline* model on original validation set using Tesla k80s GPU. The average length of a video in the set is 230 seconds 48

4.5 Performance of *NetVLAD* model with k= 10, 30 frames in proposed Teacher-Student Framework 49

4.6 Performance and FLOPs comparison in *NeXtVLAD* model with k=30 frames in proposed *Teacher-Student* Framework 49

5.1 Performance comparison of different variants of the *SkipFrame* models and the baselines. For all the variants of *SkipFrame*, we fix a budget of 10 frames. 61

5.2 Comparison of FLOPs (computation cost) of different models. Here, B: Billion and K: Thousand are the order of #FLOPs 65

LIST OF FIGURES

1.1 Video Classification Pipeline 2

2.1 A feed-forward neural network 9

2.2 Convolution operation 11

2.3 Pooling operation 12

2.4 A convolutional neural network 12

2.5 Unfolding of Recurrent Neural Network 13

2.6 Internal structure of LSTM cell 15

2.7 Structure of Hierarchical RNN 16

2.8 Vector of Locally Aggregated Encoding with pre-trained CNN architecture 17

2.9 NetVLAD encoding with pre-trained CNN 17

2.10 An agent-environment interaction in reinforcement learning framework 19

3.1 Late Fusion: Two different CNNs with same parameters, which fuse predictions at end, Early Fusion: Fuses frames from a consecutive window in a video, and Hybrid Slow Fusion: Combine late and early fusion at multiple stages. 28

3.2 Modification of 2D convolution to 3D convolution 29

3.3 Different ways in which 'Temporal' and 'Spatial' streams are fused (Carreira and Zisserman, 2017). Here, ConvNet stands for *convolutional neural network* 30

3.4 An *Encoder-Decoder* architecture used by Donahue *et al.* (2015) for the classification of a video of length T 31

4.1 Architecture of TEACHER-STUDENT network for video classification 38

4.2 Performance comparison (GAP score) of different variants of *Serial* and *Parallel* methods in *Teacher-Student* training 46

4.3 TSNE-Embedding of teacher and student representations. Here, class c refers to the cluster representation obtained corresponding to c^{th} class, wherein t and s denote teacher and student embedding respectively. 47

5.1 Frame skipping in a video 51

5.2 Architecture of *SkipFrame* for video classification 52

5.3 Structure of policy network 54

5.4 Comparison of frame-indices picked by different models. Note: GAP score performance of each model is shown at the end of its series in the graph. The average number of frames in a video is 230. 62

5.5 Action distributions obtained in the training of *SkipFrame* at first and second step of the episode, with a degree of exploration $\gamma = 0.001$. Note: at each step of training, the action distribution shown is averaged over the current batch being processed. 63

5.6 Action distribution obtained in the training of *SkipFrame* at the first step of the episode, with a degree of exploration $\gamma = 0.01$. 64

5.7 Sketch of a sample video from YouTube-8M dataset (Abu-El-Haija *et al.*, 2016) with true labels: *Travel*, *Nature*, *Train Station*, *Vehicle* 64

CHAPTER 1

Introduction

In recent times, video content has become extremely prevalent on the internet influencing all aspects of our life such as education, entertainment, healthcare, *etc*. Indeed, according to a recent study by video marketing site ReelSEO[1], an average person can spend their entire lifespan watching all the video content uploaded to YouTube in just one day! This increase in supply is of course driven by the corresponding increase in demand for such video content which in turn is partly driven by the increase in internet bandwidth and ubiquity of portable devices. For instance, another recent study[2] claims that portable mobile devices account for 70% of the watch time on YouTube. This explosive growth in online video content has led to high demand for computers to perform automated video processing for various real time applications such as video surveillance, recommendation systems, video retrieval, *etc*.

This has of course caught the attention of the academic world as well, where there is an increasing interest in video related tasks such as identifying activities in videos (Simonyan and Zisserman, 2014*a*; Yue-Hei Ng *et al.*, 2015), generating textual descriptions from videos (Donahue *et al.*, 2015), generating visual summaries (Gong *et al.*, 2014; Pan *et al.*, 2016), answering questions from a video (Jang *et al.*, 2017) and so on. To fuel research on video processing, several large scale video datasets have been released in the last few years (for example, (Soomro *et al.*, 2012; Wang *et al.*, 2017*a*; Kuehne *et al.*, 2011; Abu-El-Haija *et al.*, 2016; Xiao *et al.*, 2016)). The availability of such datasets has made it possible to train increasingly deep and complex neural models which can perform well on these tasks. The more the data available for training, the more complex model can train to obtain better and better performance. Of course, with the increase in the complexity of the models there has also been a corresponding increase in the memory and computational requirements of such video processing applications. At the same time, there is also a demand for running these models on low power devices such as mobile phones and tablets with stringent constraints on latency,

[1]http://www.reelseo.com/youtube-300-hours/
[2]https://www.youtube.com/yt/about/press/

memory and computational cost. *It is thus crucial to design models which can learn from the large amounts of data to give better performance but can still be cost effective at inference time.* With this goal, in this thesis we focus on the task of video classification. In the following sub-sections, we first describe the typical pipeline used for video classification and then describe our problem statement.

1.1 Video Classification Pipeline

A typical video classification pipeline consists of three phases: (i) feature extraction, (ii) video encoding, and (iii) classification, see Figure 1.1 as briefly outlined below.

1. **Feature extraction**: Traditionally, hand-crafted features such as HOG (Dalal and Triggs, 2005), SIFT (Lowe, 2004) were used to extract the local or spatial features for each image in the video sequence. In more recent times, high capacity deep learning based models with a combination of convolutional neural networks (Krizhevsky *et al.*, 2012) and recurrent neural networks have become the de facto standard for extracting features from frames in a video.
2. **Video encoding**: Once the features for the individual frames have been extracted they need to be aggregated to compute a representation for the video. For this, techniques such as Bag-Of-Frames (Schindler and Van Gool, 2008), VLAD encoding (Xu *et al.*, 2015*b*), recurrent neural networks (Yue-Hei Ng *et al.*, 2015; Abu-El-Haija *et al.*, 2016), *etc.* are used which aggregate the local spatial features from all the frames and obtain a single video encoding. We explain some of these in detail in Chapter 2.
3. **Classification**: Once we have the encoding for the video, it is then fed to any standard classifier such as Mixture of Experts (Miller and Uyar, 1997), logistic regression (Dreiseitl and Ohno-Machado, 2002) *etc.*

For extracting features and computing an encoding of the video we typically use (i) *recurrent neural network* (RNN) based methods, (ii) *cluster-and-aggregate* based methods, and (iii) 3D convolution based methods. In particular, RNN based methods

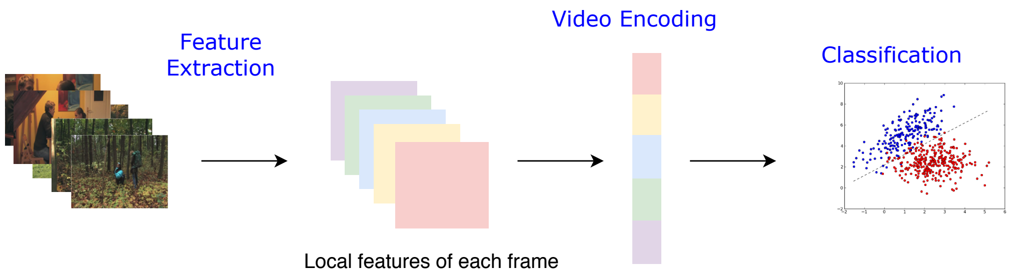


Figure 1.1: Video Classification Pipeline

(Pavel *et al.*, 2018; Choi and Zhang; Shivam, 2018) compute a hidden representation for *every* frame in the video and then compute a final representation for the video based on these frame representations in a sequential manner. Similarly, *cluster-and-aggregate* based methods (Miech *et al.*, 2017; Miha and David, 2018; Rongcheng *et al.*, 2018; Tang *et al.*, 2018; Kmiec *et al.*, 2018) have a learnable clustering matrix which is used for assigning soft clusters to *every* frame in the video. In both these methods, obtaining a video representation for longer videos can be computationally very expensive as it requires running the model for many time steps. Further, for every time step, the corresponding frame from the video needs to pass through a high capacity feature extraction network to get its representation. Thus, computing the final encoding of the video using all the frames in it becomes very time-consuming.

1.2 Efficient Video Classification

As hinted above, existing methods for aggregating frame features and computing a representation for the video have high memory and compute requirements. To address these issues, there is some recent work in the direction of (i) memory efficient models and (ii) compute efficient models as described below.

(i) Memory-Efficient Models:

This line of work focuses on reducing the memory requirement of deep networks by building a smaller and compact model, which in turn may also reduce the time taken to process a single video frame. This is known as *model compression* (Han *et al.*, 2015*a*; Wen *et al.*, 2016; Molchanov *et al.*, 2016; Zhang *et al.*, 2016*c*; Jian-Hao Luo and Lin, 2017; He *et al.*, 2017) and the primary goal here is to reduce memory footprint of network. The recently concluded European Conference on Computer Vision (ECCV) workshop on YouTube-8M Large-Scale Video Understanding (2018) (Joonseok *et al.*, 2018) focused on building such memory-efficient models for video classification which use less than 1GB of memory. The main motivation of the workshop was to discourage the use of ensemble-based methods and instead focus on memory-efficient single models. One of the main ideas explored by several participants (Miha and David, 2018; Pavel *et al.*, 2018; Rongcheng *et al.*, 2018) in this workshop was to use *knowledge distillation* to build more compact student models. More specifically, they first train a teacher network which has a large number of parameters and then use this network to guide a much smaller student network with limited memory requirements and can thus be employed at inference time. In addition to requiring less memory, such a model would also require fewer floating point operations (FLOPs) as the size of weight matrices, hidden representations, *etc.* would be smaller.

(ii) Compute-Efficient Models:

The memory-efficient models described above ensure that the memory footprint of the model is small and correspondingly the number of FLOPs required to process one frame may be small. However, in the case of videos, every frame in the video needs to be processed and hence the number of FLOPs grows linearly with the number of frames processed by the network. Hence, a memory efficient model may still be computationally inefficient. Such computations are still feasible on a Graphics Processing Unit (GPU) but become infeasible on low-end devices which have power, memory and computational constraints. In this work we address this issue by building compute efficient models.

1.3 Problem Statement

The primary focus of this thesis is on building compute efficient models which require fewer FLOPs for the task of video classification. We take motivation from the observation that when humans are asked to classify a video or recognize an activity in a video, they do not typically need to watch every frame or every second of the video. A human would typically fast forward through the video essentially seeing only a few frames and would still be able to recognize the activity (in most cases). Motivated by this, we propose a model which can compute a representation of the video by looking at only a few frames, and still be able to correctly classify the video. In other words, we focus on the problem of reducing the number of frames to be processed, and thus lowering the computation cost while maintaining the classification performance of the model.

1.4 Contributions

The primary contributions of this thesis are listed below:

- *Method to improve the efficiency of video classification, while maintaining the classification accuracy.* To achieve it, we propose a *distillation* approach, wherein a compute-heavy teacher is used to train a compute-efficient student. We do an extensive evaluation of our approach with two types of models for video classification, viz., (i) recurrent models (ii) cluster-and-aggregate models, and show that in each of these cases, a *see-all-frames* teacher can be used to train a compute-efficient *see-very-few-frames* student. We evaluate our approach on YouTube-8M dataset and show that the proposed student network can reduce the inference time by 30% and the number of FLOPs by approximately 90% with a negligible drop in the performance.

- *Complementary method to memory efficient models for video classification.* Current state-of-the-art works in this area have focused on reducing the memory footprint of high capacity models to make them more memory-efficient. Our work is complementary to these models and we show that implementing our ideas on top of a memory-efficient model gives significant reduction in computational cost.

- *Empirical analysis on dynamic selection of frames for video classification.* We introduce a reinforcement learning based model to dynamically select the most relevant frames from a video. We evaluate our model on Youtube-8M dataset and empirically establish that the dynamically selected frames lie within a small neighborhood of uniformly spaced frames. Based on this observation, we infer that under a strict budget on the number of frames, picking uniformly spaced frames seems to be a simple but effective strategy.

1.5 Applications

Below we list down some of the applications of video processing which require efficient processing and hence may benefit from the ideas presented in this thesis.

- Video Surveillance: Traditionally, surveillance techniques involved human intervention to track and identify suspicious objects or activities in a video. Now with the success of Machine Learning, in general, and Deep Learning, in particular, the process of identifying potential threats has been partly automated. To further enable real-time video surveillance it is important to build efficient video processing models as discussed in this work.

- Autonomous Driving: Computer vision is a crucial part in the development of autonomous cars. In particular, advanced deep learning techniques are used in the analysis of input video to predict the activity of various objects in the vicinity of the vehicle. In order to accelerate this analysis in real time, it is crucial to build models which are not only accurate, but computationally efficient as well.

- Smart Manufacturing: With the recent development of 5G technology, it seems possible to control robots working at remote locations with their instructions being processed on the cloud. However, this requires us to be able to process the sensory inputs being received by the robot instantly due to the added overhead of transmission. Given that video is one of the crucial sensory inputs here, it is likely that efficient video processing techniques could play a vital role in achieving this goal.

- Better Gaming Experience: With recent breakthroughs by OpenAI and Google in various Atari games and DOTA, computer vision coupled with Reinforcement Learning (RL) seems to be the way forward for designing better AI for games. The reaction time for these bots could be significantly brought down using efficient video processing techniques.

Apart from the mentioned applications, efficient video processing also plays a crucial role in medical diagnosis, space exploration, agriculture, augmented reality and so on.

1.6 Organisation of Thesis

The thesis is organized as follows:

- In this chapter, we briefly introduced the high demand for building compute efficient models for video related tasks, and their deployment on low power devices. We primarily focused on classification task and briefly introduced the video classification pipeline. We then defined our problem statement and enumerated the primary contributions of this thesis along with the potential applications that could benefit from this work.

- In Chapter 2, we explain some background concepts which are necessary to understand the work presented in this thesis. We introduce the basics of neural networks, including feed-forward neural networks (section 2.1). We then introduce convolutional neural networks (section 2.2) and recurrent neural networks (section 2.3) which form the backbone of a video classification model. We then briefly discuss the cluster-and-aggregate based methods for video classification (section 2.4). Subsequently, we describe the *Knowledge Distillation* approach (section 2.5) that we use in our thesis. We also discuss the basic terminology and some popular Reinforcement Learning methods (section 2.6), which are relevant to this work. We summarize the chapter in section 2.7.

- In Chapter 3, we present a survey of related works in the area of memory-efficient models (section 3.1). We also mention the standard benchmark datasets used for video classification (section 3.2). We discuss different video classification approaches and their evolution over time (section 3.3 and section 3.4). We also talk about some existing works in the domain of efficient video classification (section 3.5). We conclude this chapter with a discussion on the applications of reinforcement learning in various video-related tasks (section 3.7).

- In Chapter 4, we present our proposed *Teacher-Student* framework for the goal of efficient video classification (section 4.2). We devise different training paradigms for the learning of *student* network and discuss the empirical results in each case (section 4.4). In particular, we experiment with the YouTube-8M dataset and mention the implementation details and different configurations of the models that were compared (section 4.3 and section 4.5). In particular, we discuss our experiments with different variants of *compute-efficient student* and establish the pragmatic significance of our approach. Further, we experiment with *memory-efficient* models to show that our ideas can also be applied on top of such models (section 4.6).

- In Chapter 5, we introduce a reinforcement learning approach for *dynamic* selecting important frames from a video (section 5.1). We describe the proposed *SkipFrame* model (section 5.2) and discuss the empirical results in details (section 5.5). We also mention the implementation details, the hyperparameters used and the different configurations used for experimentation (section 5.3 and section 5.4).

- In Chapter 6, we conclude our work and discuss possible future research directions.

CHAPTER 2

Background

This chapter covers the concepts required to understand the work presented in this thesis.

2.1 Feed Forward Neural Network

With the intention to mimic the biological neurons in our nervous system, artificial neurons were first proposed by McCulloch and Pitts (1943). The initial model proposed by them was capable of handling only binary inputs and outputs. A more generic model called as Perceptron, was later developed by Rosenblatt (1958) which can take real-valued inputs and produce binary outputs. However, a well-known limitation with the standard perceptron model was that it could learn the decision boundaries only in the case of linearly separable data. This limitation was overcome by the introduction of *Multilayer Perceptrons* (Hornik *et al.*, 1989), which is capable of modeling any function with real inputs and real outputs.

A multilayer perceptron is implemented as a multi-layer structure containing at least three layers of neurons: (i) input, (ii) hidden and (iii) output layer, as shown in Figure 2.1, wherein each layer contains multiple neurons/nodes. In general, each node in a given layer L combines the outputs from all the nodes in the previous layer $L-1$. More

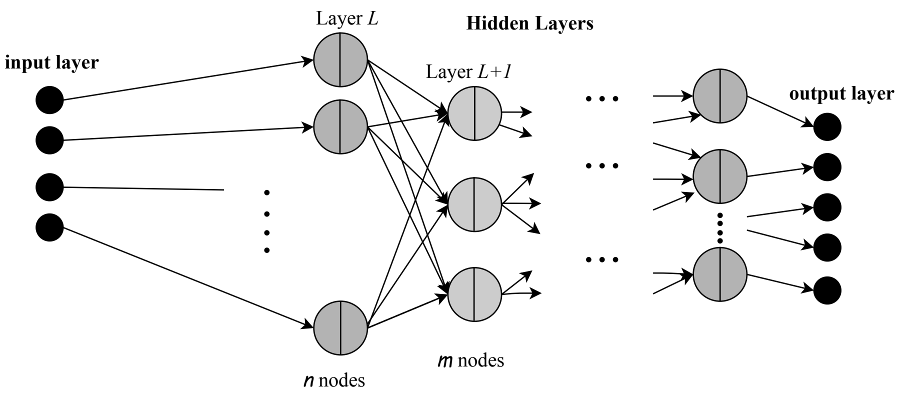


Figure 2.1: A feed-forward neural network

specifically, it computes a linearly weighted sum of the inputs from the previous layer to compute the pre-activations at this layer. A non-linear activation function ϕ is then applied to these pre-activations and the output is "fed forward" as input to the next layer $(L+1)$. Hence, this network is called a feed-forward neural network.

An activation function ϕ is a transformation function in the hidden layers to "squeeze" the pre-activation values. Some examples of popularly used activation functions are *logistic*, *tanh*, *linear*, *ReLU* (Nair and Hinton, 2010), *Leaky ReLU* (Maas *et al.*, 2013) and other variants of *ReLU*. The output layer activation depends on the task at hand. For example, the *softmax* activation function is used in the case of classification. An objective or loss function, $\mathcal{L}_{model}$ is defined to learn the desired functionality or mapping from the input (x) to the output (y). For example, for a classification task, the cross-entropy between the true distribution (y) and the predicted distribution $(\hat{y})$ is used as the loss function. The network is trained to find an optimal set of weights which minimizes the defined loss function.

2.1.1 Learning Algorithm

The learning algorithm used to train the parameters or weights of the network is the popular *backpropagation* algorithm. It was first introduced in the domain of AI by McClelland *et al.* (1986). In particular, it involves computing the gradients (derivatives) of the error function w.r.t. the weights of the network using the *chain rule* of derivatives. The feed-forward network is composed of continuous and differentiable activation functions. Hence, the derivative of the objective function w.r.t weights in each of the layer L, *i.e.*, W_L exists. The gradient which is a collection of partial derivatives $(\frac{\partial \mathcal{L}_{model}}{\partial W_1}$, ... $,\frac{\partial \mathcal{L}_{model}}{\partial W_L})$ is used for adjusting the trainable weights to minimize the objective function $\mathcal{L}_{model}$. According to the *gradient descent* approach, weights of a layer L at time t, *i.e.*, W_L^t are adjusted by moving in a direction opposite to the gradient with a constant step size η as given in the equation below:

$$W_L^{t+1} = W_L^t - \eta \frac{\partial \mathcal{L}_{model}}{\partial W_L^t} \tag{2.1}$$

These updates are made iteratively until some stopping criterion is met. The fixed step size in a vanilla gradient descent either slows down or hinders the convergence of the algorithm in high dimensional parameter space. To overcome this limitation, some recent optimization algorithms (Zeiler, 2012; Kingma and Ba, 2015) use adaptive step sizes for better exploration of the error surface.

2.2 Convolutional Neural Networks (CNN)

In a simplistic terms, a Convolutional Neural Network can be viewed as a combination of three different components as discussed below.

2.2.1 Convolution Operation

The primary component of CNN is the convolution operation. Typically, an image I is represented as a matrix of pixels with size $h_I \times w_I \times c_I$, where h_I, w_I and c_I are the height, width and number of channels in the image respectively. For example, for an RGB image, the number of channels is three. A convolution layer consists of a weight matrix **K** called a *filter*, which operates on the input image or input feature map to produce an output feature map of size $h_O \times w_O \times c_O$. Precisely speaking, it first computes an element-wise product of filter matrix **K** and the selected input window (indicated by red shade in Figure 2.2), and then, sums up all the values to produce the final value at the corresponding location in the output feature map (indicated by green shade in Figure 2.2). The filter then slides over to the next window in the input image or feature map by a fixed distance called *stride*. The primary purpose of the convolution operation is to extract features from different spatial locations of an image using *shared parameters*, which is the matrix **K** in this case. Since only a patch of the image affects the corresponding value in the output feature map, the network is more robust to the perturbations in other distant patches of the image. This is an important advantage of a convolution neural network over a standard multilayer perceptron.

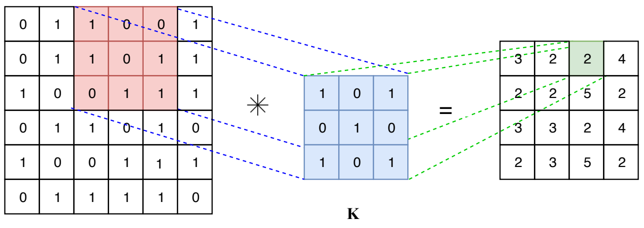


Figure 2.2: Convolution operation

2.2.2 Pooling

The max-pooling layers in the convolutional network downsample the large feature map representations and hence, save a lot of computations. For example, as shown in Figure 2.3, a *max-pooling* layer with a pool size of 2×2 divides the feature map into 2×2 grids and reduces the overall size from 4×4 to 2×2. There are other alternatives to *max-pooling* layers such as *average-pooling* or L_2-*norm pooling* but, *max-pooling* is known to work better in practice.

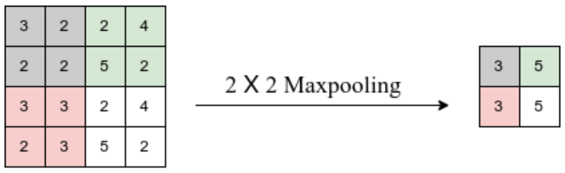


Figure 2.3: Pooling operation

2.2.3 Fully Connected Layer

In a standard CNN architecture, the output feature map from the final convolutional layer is flattened and passed onto a fully connected layer of a feed-forward neural network (see Figure 2.4). The convolutional layers are typically treated as 'feature extractors' to obtain the local spatial characteristics of an input image.

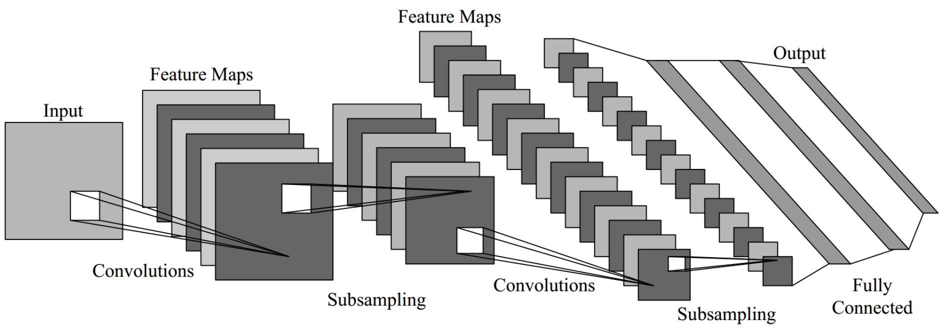


Figure 2.4: A convolutional neural network

ResNet: Recently, there have been many success stories of *deep* convolutional neural networks in several computer vision tasks, such as image classification (Simonyan and Zisserman, 2014*b*), object detection (Girshick, 2015; Ren *et al.*, 2015) and so on. The added depth in these networks boosted their representation ability but led to the standard problem of *vanishing gradients* (Glorot and Bengio, 2010*a*), which made the training process more difficult. Residual Networks (ResNet) by He *et al.* (2016) is an immensely popular modification to a vanilla CNN architecture, which introduced *identity* connections to pass the input to the successive layers in the network. These *identity* connections enable the gradients to backpropagate easily and thus prevent the problem of vanishing gradients.

2.3 Recurrent Neural Network (RNN)

To understand an ongoing activity in a video, one needs to remember the crucial information from the frames seen so far. However, this characteristic is not present in a vanilla feed-forward neural network. Recurrent neural network (RNN) (Jordan, 1997)

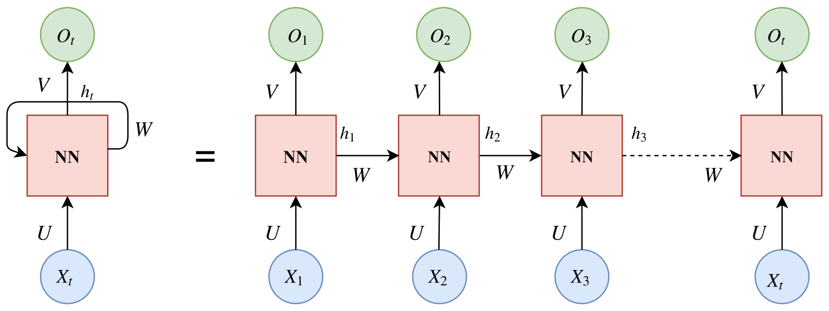


Figure 2.5: Unfolding of Recurrent Neural Network

was designed to capture the temporal dependencies between different steps in a sequential input. The idea is to memorize the information from each time step of the sequence in the form of a recurrent *hidden* state. An RNN can be viewed as the same neural network unrolled over the input sequence X in a step-by-step fashion (see Figure 2.5). Mathematically, we can summarize the RNN model as:

$$h_t = \sigma(UX_t + Wh_{t-1} + b_h) \tag{2.2}$$

$$O_t = \phi(Vh_t + b_O) \tag{2.3}$$

The trainable parameters in the recurrent neural network include weight matrices U, V and W, and the bias vectors b_h and b_O. Here, O_t denotes the output vector produced at a time step t, σ is the sigmoid activation and ϕ is the output activation function. For example, in the task of video classification, each input X_t denotes a frame in the input video sequence of length T. We encode the complete video sequence X in the form of a single hidden state h_T obtained at the end of the video. We use this final hidden state h_T to predict the category of the input video using $softmax$ activation: $O_T = softmax(Vh_T + b_O)$. An RNN is trained using the standard Backpropagation Through Time (BPTT) algorithm proposed by Williams (1989). Due to a recursive chain in the computation of hidden states h_t (refer to Eq.2.2), the gradients are computed through all the possible paths from the final state all the way back to the initial steps of the sequence. As the sequence length increases, the problem of vanishing/exploding gradients arises as discovered by Bengio *et al.* (1994). The recurrent neural networks thus fail to maintain long term dependencies in the case of longer input sequences.

2.3.1 Long Short Temporal Memory Networks (LSTMs)

To tackle the problem of maintaining long term dependencies in RNNs, Long Short Temporal Memory Network was proposed by Hochreiter and Schmidhuber (1997). LSTM is a variant of RNN which maintains an internal cell state c_t along with the LSTM cell or hidden state h_t. The internal state of the LSTM is computed using the following set of equations (ignoring the bias terms):

$$i_t = \sigma(X_t U^i + h_{t-1} W^i) \tag{2.4}$$

$$f_t = \sigma(X_t U^f + h_{t-1} W^f) \tag{2.5}$$

$$o_t = \sigma(X_t U^o + h_{t-1} W^o) \tag{2.6}$$

$$\hat{c}_t = \phi(X_t U^c + h_{t-1} W^c) \tag{2.7}$$

$$c_t = f_t * c_{t-1} + i_t * \hat{c}_t \tag{2.8}$$

$$h_t = \tanh(c_t) * o_t \tag{2.9}$$

The gating functions described in Eq.2.4, Eq.2.5 and Eq.2.6 are used to control the *information flow* from one time step to another, thus enabling the network to discard the irrelevant history. In particular, input gate i_t controls the information to be read from the *current intermediate* cell state $\hat{c}_t$ (see Eq.2.7). The forget gate f_t selectively learns to forget the information from previous cell state c_{t-1}. The information filtered through input and output gates is used to update the current cell state (see Eq.2.8). The output gate further filters the information from the current cell state c_t, which should be written

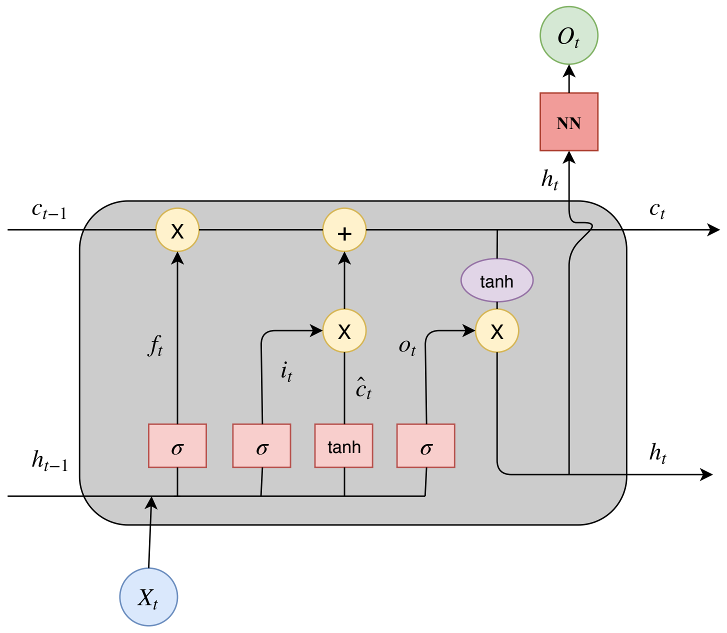


Figure 2.6: Internal structure of LSTM cell

to LSTM cell state h_t. Figure 2.6 shows a pictorial representation of the LSTM cell.

2.3.2 Hierarchical Recurrent Neural Network

Sequential inputs generally have a hierarchical structure across the dimension of time. For example, in a video, many factors like the spatial motion of objects, visual diversity of frames, scene change, *etc.*, give rise to well-defined boundaries in a *longer* sequence of video. Hence, the video can be conveniently divided into blocks wherein each block containing similar frames is encoded separately (see Figure 2.7). After obtaining RNN representations for each block, the second layer of RNN is used to obtain a single representation of the video. This hierarchical structure of the model enables it to memorize information over a much longer time as compared to the vanilla RNN model described earlier. This architecture is called the Hierarchical Recurrent Neural Network (H-RNN) and is popularly used in many video related tasks.

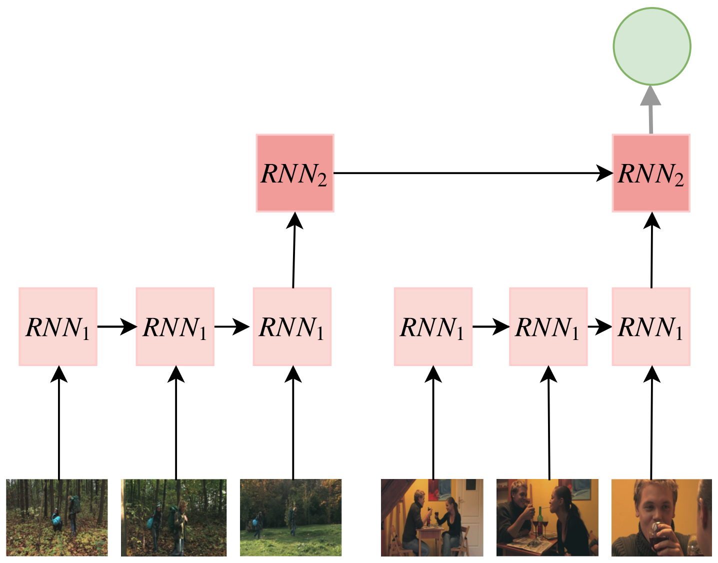


Figure 2.7: Structure of Hierarchical RNN

2.4 VLAD and NetVLAD Feature Encoding

To construct a single representation of a video containing a sequence of T frames $\{I_1, I_2, ..., I_T\}$, Vector of Locally Aggregated Descriptors (VLAD) is another popular method used to combine the frame-level descriptors (Xu *et al.*, 2015*b*). For example, for a given frame I_t, the frame level descriptors can be generated from the last convolutional layer of a CNN. In particular, the last convolutional layer of size $h \times w \times n$ is reshaped into a set d containing n-dimensional descriptor vectors, where the number of descriptors, *i.e.*, $|d|$ is $N = h \times w$. Each of these descriptor vectors $d_i \in \mathbb{R}^n$ is assigned to one of the K clusters having $c_1, c_2, \ldots, c_K$ as the cluster means (see Figure 2.8). The clusters are generated using standard k-means clustering on N descriptors. Intuitively, these descriptors capture the local spatial dependency among different patches of an image. For a cluster with c_k mean, VLAD encoding vector $u_k \in \mathbb{R}^n$ is computed by the sum of the difference vectors between a descriptor d_i and cluster mean c_k if the descriptor belongs to this cluster, *i.e.*, $d_i \in cluster(c_k)$, as shown below:

$$u_k = \sum_{i:\, d_i \in cluster(c_k)}^{N} (d_i - c_k) \tag{2.10}$$

To avoid non-differentiability due to the hard assignment of the clusters in the VLAD network , NetVLAD (Arandjelovic *et al.*, 2016) network uses soft assignment of clus-

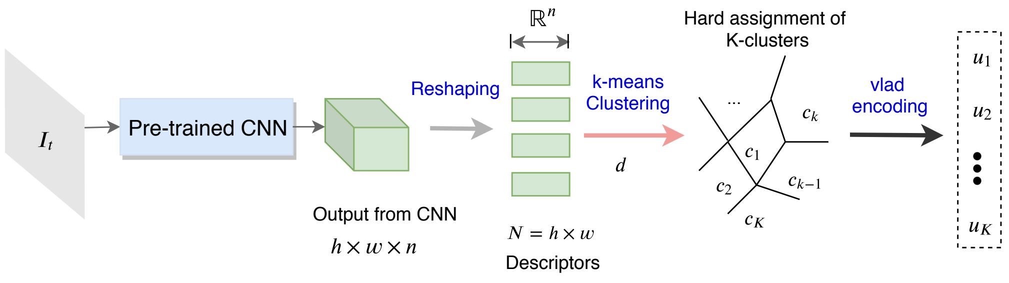


Figure 2.8: Vector of Locally Aggregated Encoding with pre-trained CNN architecture

ters with learnable parameters, see Figure 2.9. In general, soft assignment of a cluster c_k to a descriptor d_i is computed as follows:

$$\alpha_k(d_i) = \frac{e^{w_k^T d_i + b_k}}{\sum_{k'=1}^{K} e^{w_{k'}^T d_i + b_{k'}}} \tag{2.11}$$

where w_k, b_k and the means of all the clusters $(c_1, c_2, ..., c_K)$ are trainable parameters. Using this soft assignment α_k, the NetVLAD output u_k corresponding to a cluster mean c_k is computed as shown in Eq.2.12.

$$u_k = \sum_{i=1}^{N} \alpha_k(d_i)(d_i - c_k) \tag{2.12}$$

$$v_t = (u_1, u_2, ..., u_K) \tag{2.13}$$

These VLAD or NetVLAD encoding vectors u_k are concatenated to form a single representation vector $v_t \in \mathbb{R}^{nK}$ for the given image I_t, as in Eq.2.13. To obtain a video-level representation v, the VLAD representations across all the frames of video $v_1, v_2, .., v_T$ can be combined in different ways which are further explored in the section 3.4. It is to be noted that this kind of a differentiable network has an additional benefit of being end-to-end trainable.

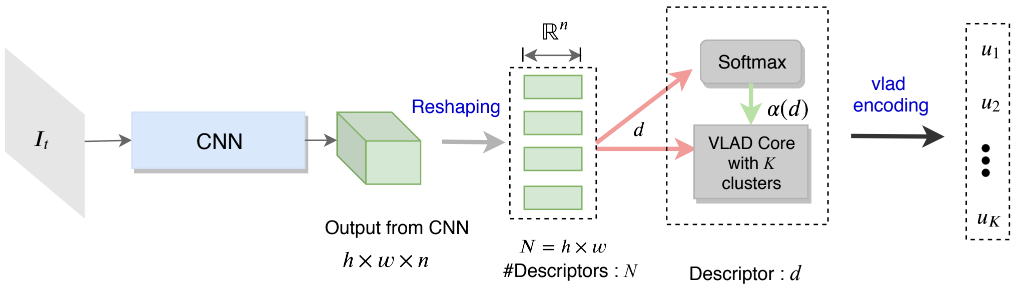


Figure 2.9: NetVLAD encoding with pre-trained CNN

2.5 Knowledge Distillation

Due to the remarkable success of high capacity deep neural networks, it has become extremely important to make these networks less complex and hence, more generalizable during inference. One way to achieve this is to somehow transfer this knowledge from a bigger model to a much lighter model in an explicit way. This is the core idea behind *Knowledge Distillation*. In general, a bigger complex model (or an ensemble of different models) is trained with a full model capacity to perform well on a task. This bigger network is called *teacher*. The teacher network is trained on *hard labels* of ground truth, which assigns equal weightage to all the incorrect class predictions. The network trained on hard labels only tries to maximize the probability of the *true* class alone and doesn't capture the similarity between classes. According to Hinton *et al.* (2015), knowledge about the generalizability of the network can be quantified in the output class distribution. Thus, instead of the hard class distribution provided in the ground truth, a softer class distribution or the *logits* themselves can be used for transferring the knowledge as they better capture the similarity between classes. Inspired by this insight, Hinton *et al.* (2015) introduced a smaller network termed as *student*. This student network is trained to learn from the *logits* or *soft targets* obtained from the trained teacher network as well as the ground truth.

Mathematically, in a classification task on $\mathscr{Y}$ classes with the given input X, assume that $z^T \in \mathbb{R}^{|\mathscr{Y}|}$ is the input to final *softmax* layer of the teacher network T termed as *logit* vector. The same input X is fed to the student network S with parameters W_S. The *logit* vector produced by the student network is denoted by $S(X; W_S)$. The final distillation loss $\mathcal{L}_{pred}$ to train the student network is formulated as given below:

$$\mathcal{L}_{pred} = d(S(X; W_S), z^T) \tag{2.14}$$

Here, d can be any suitable distance metric. In a similar way, student network can be trained to match the *soft target* class distribution $\mathcal{P}^T$(parameterized by a temperature parameter τ) from the teacher, as given below:

$$\mathcal{P}^T_j = \frac{\exp(z^T_j/\tau)}{\sum_{j'=1}^{|\mathscr{Y}|} \exp(z^T_{j'}/\tau)} \tag{2.15}$$

The final loss function to train the student network includes the weighted average of standard classification loss and the knowledge distillation loss $\mathcal{L}_{pred}$.

2.6 Reinforcement Learning (RL)

Reinforcement learning is a learning method which involves an agent continuously interacting with the environment and learning how to achieve a desired goal. The environment generates the 'evaluative feedback signals' (termed as *rewards*) to help the agent learn to select the favourable *actions* in order to achieve the goal.

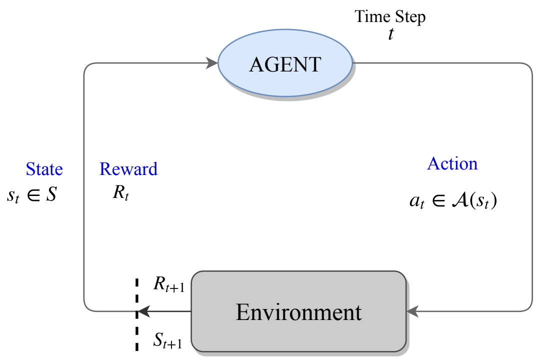


Figure 2.10: An agent-environment interaction in reinforcement learning framework

2.6.1 Formulation

The problem is modeled as a sequential decision making problem, more precisely as a Markov Decision Process (MDP), where the agent and the environment interact at each step t of the discrete time sequence. Here $t = 0, 1, 2, 3, 4$ ... T, and T is the terminal state of the sequence. *For the sake of simplicity, we consider only discrete and finite time sequence.* The agent looks at the condition of environment's state $s_t \in \mathcal{S}$ at every time step t, where $\mathcal{S}$ is the set of all possible states. Based on these observations, agent chooses an action $a_t \in \mathcal{A}(s_t)$, where $\mathcal{A}(s_t)$ is the set of actions possible given that the agent is currently looking at the state s_t (see Figure 2.10). The selected action changes the state of environment from s_t to s_{t+1} and generates reward a R_{t+1} for the agent accordingly. The environment dynamics are assumed to be Markov *i.e.*,

$$\mathcal{P}(s_{t+1} = s | s_0, a_0, s_1, a_1, \ldots, s_t, a_t) = \mathcal{P}(s_{t+1} = s | s_t, a_t) \tag{2.16}$$

2.6.2 Policy Gradients

We now discuss the algorithm used for training an RL agent. We begin by defining some concepts which will be used in the discussion.

1. **Return**: Return G_t is defined as the sum of total rewards (can be discounted) obtained by the agent starting from step *t* up to the terminal step T.
   $G_t = R_{t+1} + R_{t+2} + ... + R_T$

2. **Policy**: A policy π is a mapping from perceived states to the probability distribution $\mathcal{P}$ over the possible actions $a \in \mathcal{A}$ given a state *s* at time t.
   $\pi(a|s) = \mathcal{P}(a_t = a|s_t = s)$

3. **Value Function**: Value function for a policy π estimates how *good* a given action is for the agent in a given state. The *value function* in terms of state s at time step *t* can be written as:
   $V^{\pi}(s) = \mathbb{E}_{\pi}[G_t|s_t = s]$

The policy for which the value function has the maximum value at all the states is called an $optimal$ policy. There is always at least one policy that is optimal for all the states in the case of finite MDPs (Puterman, 2014).

Policy Approximation

Let's denote the parameters used to represent a policy by θ^{π}. The policy network is parameterized in such a way that $\pi(a|s, \theta^{\pi})$ is differentiable w.r.t θ^{π}. Intuitively, a certain *performance function* $\mathcal{J}$ is chosen such that it correlates with the goal of the agent. Mathematically, we can define the performance function $\mathcal{J}(\theta)$ as shown in Eq.2.17. Here, $d^{\pi}(s)$ is the stationary state probability distribution achieved (*Markov saturation property* by Chung (1967)) by starting from the initial state s_0 and following the parameterized policy π^{θ}, as defined in the Eq.2.18. The performance function is optimized w.r.t. θ using the standard gradient *ascent* update rule to maximize the performance (see Eq.2.19).

$$\mathcal{J}(\theta) = \sum_{s \in S} d^{\pi}(s) V^{\pi}(s) = \sum_{s \in S} d^{\pi}(s) \sum_{a \in \mathcal{A}} \pi(a|s, \theta) Q^{\pi}(s, a) \tag{2.17}$$

$$d^{\pi}(s) = \lim_{t \to \text{inf}} p(s_t = s|s_0, \pi_{\theta}) \tag{2.18}$$

$$\theta_{t+1} = \theta_t + \eta \nabla \mathcal{J}(\theta_t) \tag{2.19}$$

Policy Gradient Theorem

To compute gradients for the policy update Eq.2.19, the stationary distribution of a state $d(s)$ under a policy behavior π^{θ} *i.e.*, $d^{\pi}(s)$ should be known. However, it becomes difficult to estimate the effect of *policy updates* on the distribution $d^{\pi}(s)$ due to the unknown (uncontrollable) external environment. *Policy gradient theorem* (Sutton *et al.*, 2000) shows that the gradient of the performance function is independent of the effect of policy changes on the stationary distribution of states *i.e.*, it doesn't include any term related to $\nabla_{\theta} d^{\pi}(s)$ and can thus be approximated using sampling.

$$\nabla_{\theta}\mathcal{J}(\theta) = \sum_{s\in\mathcal{S}} d^{\pi}(s) \sum_{a\in\mathcal{A}} \nabla_{\theta}\pi(a|s,\theta) Q^{\pi}(s,a) \tag{2.20}$$

$$= \sum_{s\in\mathcal{S}} d^{\pi}(s) \sum_{a\in\mathcal{A}} \frac{\nabla_{\theta}\pi(a|s,\theta)}{\pi(a|s,\theta)} Q^{\pi}(s,a)\pi(a|s,\theta) \qquad \text{; rearranging the terms} \tag{2.21}$$

$$= \mathbb{E}_{\pi}[\nabla_{\theta} \ln \pi(a|s,\theta) Q^{\pi}(s,a)] \qquad \text{; because } \frac{d(\ln(x))}{dx} = 1/x \tag{2.22}$$

REINFORCE algorithm by Williams (1992), is a well-known policy gradient method which uses Monte Carlo samples to estimate the value function ($Q^{\pi}(s,a)$ in Eq.2.20). It measures the expected return G_t from a full trajectory (sample) generated by Monte Carlo method and uses it directly as $Q^{\pi}(s,a)$ (see Eq.2.23 and Eq.2.24).

$$\nabla_{\theta}\mathcal{J}(\theta) = \mathbb{E}_{\pi}[\nabla_{\theta} \ln \pi(a|s,\theta) Q^{\pi}(s,a)] \tag{2.23}$$

$$= \mathbb{E}_{\pi}[\nabla_{\theta} \ln \pi(a|s,\theta) G_t] \qquad \text{; by definition } Q^{\pi}(s,a) = \mathbb{E}_{\pi}[G_t|s,a] \tag{2.24}$$

2.7 Summary

In this chapter, we discussed some of the concepts which will be used for the remainder of the thesis. In particular, we discussed different types of neural networks, *viz.*, *feed forward networks*, *convolutional neural networks* and *recurrent neural networks*. We also discussed the learning algorithms used for training the networks, viz., Backpropagation and BPTT. We further discussed some methods for aggregating information from different frames in a video to compute the final representation of the video. We then introduced the idea of *Knowledge Distillation* followed by a brief introduction to RL with the intention of making the reader familiar with the basic terminology.

CHAPTER 3

Literature Survey

In this chapter, we discuss prior work in the literature which is most relevant to our work. We start with the literature survey of memory-efficient models, in particular, models relying on *knowledge distillation* for video related tasks. We introduce various video benchmark datasets and then discuss various video classification approaches. We then provide a detailed description of various video classification models developed on the Youtube-8M dataset. We conclude this chapter with a discussion of different reinforcement learning based methods used for video-related tasks.

3.1 Memory-Efficient Models

Several works have focused on reducing the memory footprint of a deep and compute-heavy model, which in turn accelerates the computations or reduces the processing time of the network. This direction of research is termed as model compression. Such compression techniques make it practically feasible to deploy these heavy models on devices with stringent memory and computational constraints. In recent times, there has been a lot of work on model compression by Cheng *et al.* (2017); Narang *et al.* (2017); Li *et al.* (2016); Jian-Hao Luo and Lin (2017); He *et al.* (2017), *etc.*, in the context of image classification. Some of the popular approaches explored in this field are:

(i) *Pruning Unimportant Weights*: It involves the removal of unimportant (mostly sparse) weight connections with a minimal drop in the performance of the model. This kind of pruning results in a network with fewer parameters and hence, more memory efficiency (LeCun *et al.*, 1990; Endisch *et al.*, 2007; Han *et al.*, 2015*b*).

Typically, in a convolutional neural network (Simonyan and Zisserman, 2014*b*; Szegedy *et al.*, 2015; He *et al.*, 2016), the number of floating point operations in a convolution layer is relatively higher as compared to a fully connected layer in the network. Thus, in addition to pruning the connections, computations can further be lowered by completely removing unimportant *filters* or weight matrices from the network. This approach has been recently explored by many researchers (Li *et al.*, 2016; Jian-Hao Luo and Lin, 2017; He *et al.*, 2017).

(ii) *Knowledge Distillation*: It focuses on transferring the *knowledge* from a deep and complex (computationally and memory expensive) network to a shallow and efficient network. This is achieved in several ways as explored by Hinton *et al.* (2015); Ba and Caruana (2014*a*); Chen *et al.* (2017*a*), *etc.*

(iii) *Low-Rank Factorization*: Another line of work focuses on building a compact representation of weight matrices in the network. The motivation behind this is to decompose the compute-intensive large matrix multiplications into smaller and relatively cheaper operations. This is achieved by the low-rank approximation of weight matrices (Jaderberg *et al.*, 2014; Denil *et al.*, 2013; Zhang *et al.*, 2015).

A similar approach is to design a compact network architecture which has fewer weight parameters or/and computations (Romero *et al.*, 2015; Iandola *et al.*, 2016; Howard *et al.*, 2017).

(iv) *Network Quantization*: An orthogonal approach to completely pruning the weights is to either quantize (Han *et al.*, 2015*a*; Zhou *et al.*, 2017) or binarize (Rastegari *et al.*, 2016; Courbariaux *et al.*, 2016) the weight connections in the network. This, in turn, reduces the memory footprint and computation cost in the network. Even in the context of video classification, some work (Miha and David, 2018; Liu Tianqi, 2018) has been done on using *Quantization* for model compression.

All of the above methods work with the same goal of building a memory-efficient model by adopting different strategies. We refer the reader to a survey paper by Cheng *et al.* (2017) for a detailed review of the field. In this thesis, we specifically focus on the idea of *Knowledge Distillation* for the video classification task. Hence, for brevity, we discuss the papers based on the idea of distillation. For example, Ba and Caruana (2014*b*); Hinton *et al.* (2015); Lopez-Paz *et al.* (2016); Chen *et al.* (2017*a*) use *Knowledge Distillation* to learn a more compact *student* network from a computationally expensive *teacher* network. The key idea is to train a shallow student network using soft targets (or class probabilities) generated by the teacher instead of the hard targets present in the training data. There are several other variants of this technique.

For example Romero *et al.* (2015) extend this idea to train a student model which not only learns from the outputs of the teacher but also uses the intermediate representations learned by the teacher as additional *hints*. This idea of *Knowledge Distillation* has also been tried in the context of pruning networks for multiple object detection (Chen *et al.*, 2017*a*), speech recognition (Wong and Gales, 2016) and reading comprehension (Hu *et al.*, 2018). Some work on video-based action recognition by Zhang *et al.* (2016*a*) tries to accelerate the processing speed in a two-stream CNN architecture by transferring the knowledge from 'motion' modality to 'optical' modality of the video frames.

In general, model compression techniques try to reduce the processing by removing the redundant parameters in the network. This is closely related to the concept of 'synaptic pruning' (Santos and Noggle, 2011) of redundant neurons in a human brain. In addition to the pruning of neurons, our brain also learns to prune the irrelevant input information from the external environment, such that only the relevant input is fed to the cortical center of the brain (Cromwell *et al.*, 2008). Following a similar idea, instead of reducing the processing required per input, we intend to reduce the number of inputs that we process. For example, if a neural network is processing a sentence, the network could potentially avoid certain words, if they do not provide any information relevant to the current task at hand. In this work, we utilize the idea of *Knowledge Distillation* to reduce the number of frames in a video (*i.e.*, number of inputs) processed by the network. While in the related works on *Knowledge Distillation*, the teacher and student differ in the number of layers, in our case, the teacher and student networks differ in the number of time steps or frames processed by the two networks.

3.2 Video Classification Datasets

In the domain of Computer Vision, large scale datasets such as ImageNet (Deng *et al.*, 2009) have been a crucial enabler of recent progress in the image related tasks (Kuehne *et al.*, 2011; Simonyan and Zisserman, 2014*b*; He *et al.*, 2016). In a similar way, there has been rapid progress in the development of video benchmarks. With the advent of these datasets (Soomro *et al.*, 2012; Karpathy *et al.*, 2014; Kay *et al.*, 2017; Abu-El-Haija *et al.*, 2016), the research community has shown great interest in the video clas-

Dataset	No. of Videos	No. of Categories	Theme
UCF-101	13,320	101	Human actions
Sports	1,000,000	487	Sports categories
Kinetics	650,000	700	Human-object activities
YouTube-8M (2017 version)	8,264,650	4716	Categories of objects, events, actions *etc.*

Table 3.1: Standard datasets for benchmarking video classification methods

sification task. In this section, we introduce some popular video classification datasets and their key contributions. We summarize the statistics of these datasets in Table 3.1.

Popular datasets like UCF-101 (Soomro *et al.*, 2012) have boosted the technological advancements on video classification task. However, due to extensive correlation between the video frames in this dataset, it suffers from the problem of 'less frame diversity'. This leads to poor generalization of models on various other video related tasks. Even in the case of datasets with diverse frames, due to the presence of single-themed videos (for example, *Sports* dataset by Karpathy *et al.* (2014)), it becomes difficult to generalize on other datasets. Kay *et al.* (2017) have tackled this problem by introducing *Kinetics* dataset, which covers a diverse range of classes (human-focused activities). With the introduction of *YouTube-8M* dataset (Abu-El-Haija *et al.*, 2016), the classes were no longer restricted to human actions. This dataset incorporates a much wider range of categories (like *sports*), objects (like *food and drinks*), events (like *concert*) and scenes (like *travel*), to reflect the central theme of a video. Due to its high diversity and unprecedented scale, Youtube-8M dataset is unarguably the most challenging dataset for video classification task. In this work, we use YouTube-8M dataset for all the experiments.

3.3 Feature Extraction From Images

A video clip is represented as a sequence of frames, wherein each frame is an image. Hence, to obtain a representation for the video, we first need to obtain a representation for the individual frames. Images can have several attributes like edges, corners, shapes, *etc.*, which can be used to describe it. The *feature extraction* techniques involve transforming the raw visual images (for example, RGB images) into informative attributes (called *features*) needed for the task at hand.

A popular technique called Scale Invariant Feature Transform (SIFT) (Lowe, 2004), was designed to detect local pixel-level attributes which are invariant to scaling, rotation and other image-related transformations. Histogram of Oriented Gradients (HOG) (Dalal and Triggs, 2005) is another technique which computes the histogram of image gradients, and based on it divides the image into local patches. This technique is mainly helpful for the tasks which need spatial (patch-level) information in the image, for example, *object detection*. In recent times, the remarkable success of convolutional neural network (CNN) based features for various image related tasks like image classification (Krizhevsky *et al.*, 2012; Simonyan and Zisserman, 2014*b*; He *et al.*, 2016), object detection (Ren *et al.*, 2015; Girshick, 2015), *etc.*, has made them more appealing for video tasks also. In the case of a video, each frame is passed through a CNN to produce a feature representation of the frame. The popular state-of-the-art CNN models like ResNet (He *et al.*, 2016), VGG-16 (Simonyan and Zisserman, 2014*b*), GoogleNet (Szegedy *et al.*, 2015), *etc.* are commonly used for extracting features from the images.

3.4 Feature Aggregation For Video Classification

To obtain a video level representation, the features extracted from each image/frame in the video are combined in a certain way which is called *feature aggregation*. Intuitively, the aggregation process should capture the crucial information (*spatial* as well as *temporal*) across the frames of a video. Different approaches for feature aggregation have been discussed ahead.

3.4.1 Single Stream Networks

One naive way of combining features from all the frames can be *mean pooling* or *concatenation* of all the features. However, this kind of pooling completely ignores the temporal order as well as the relative weightage of frames in the video. To infuse temporal information in the network, Karpathy *et al.* (2014) use 2D convolutional network for feature extraction and multiple fusion strategies to combine these features, as shown in Figure 3.1. The *late fusion* strategy involves using separate CNN models to encode the distant frames in a video and fuse their features later in the network. On the contrary, the *early fusion* strategy involves combining the features from a consecutive window of frames, wherein the input has one extra dimension added before convolution, and filters

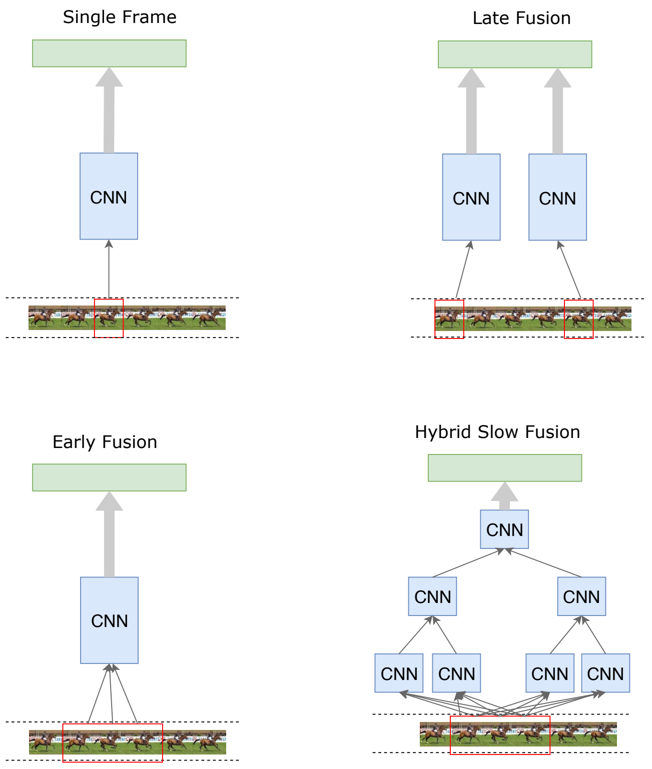


Figure 3.1: Late Fusion: Two different CNNs with same parameters, which fuse predictions at end, Early Fusion: Fuses frames from a consecutive window in a video, and Hybrid Slow Fusion: Combine late and early fusion at multiple stages.

in the initial convolutional layer are modified accordingly. To combine the best of both, *hybrid slow fusion* is designed which somewhat maintains the temporal information till a certain layer (see Figure 3.1). However, to generate the final predictions, all of these methods sample a window/set of frames multiple times and average-out the predictions at the end. Due to the loss in the order of frames during averaging, the network is still not able to capture the motion information completely.

3.4.2 3D Convolution Based Networks

Ji *et al.* (2013) introduced 3D convolutions to encode the discriminative features from the video in such a way that both spatial and temporal dimensions are captured. This is achieved by dividing a video of length T into stacks of consecutive frames, and using 3-dimensional filters to convolve across the depth L of each stack volume (see Figure 3.2(a)). The final representation of a video is obtained by combining the outputs from all the stacks. In contrast, 2D convolution treats multiple frames in a video as multiple channels of input and produces a 2D output (see Figure 3.2(b)), thus losing the temporal order. Along the similar lines, several works (Karpathy *et al.*, 2014; Tran *et al.*, 2015; Carreira and Zisserman, 2017) proposed new variants of 3D convolutional networks such as C3D, I3D *etc.*, for the task of video classification. Shou *et al.* (2016) and Xu *et al.* (2017) further modified 3D convolution network to leverage it for temporal action localization task in the untrimmed videos. 3D convolution based models have also advanced various video related domains such as scene understanding (Husain *et al.*, 2017), object recognition (Garcia-Garcia *et al.*, 2016), and so on. Despite the above use cases, 3D convolution based models still suffer from high computational requirements (Carreira and Zisserman, 2017).

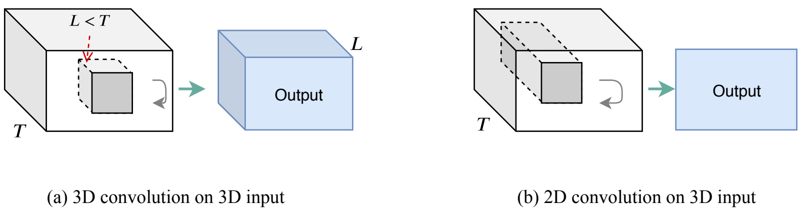

$L < T$
L
Output
T
(a) 3D convolution on 3D input
Output
T
(b) 2D convolution on 3D input

Figure 3.2: Modification of 2D convolution to 3D convolution

3.4.3 Multi-Stream Networks

To capture long-range dependencies in temporal information across video frames, a separate 'Temporal Stream' network is used to work with a stack of multi-frame *optical flow* (Fleet and Weiss, 2006) images. By definition, optical flow images consist of the motion pattern of objects, surfaces, edges *etc.*, which is generally pre-computed in practice. Simonyan and Zisserman (2014*a*) exploited this idea in the form of a *Two-Stream* architecture for video classification, where the streams of features are fused only at the end (*i.e.*, at the *classification* layer) in the network. Several recent works on video related tasks (Simonyan and Zisserman, 2014*a*; Carreira and Zisserman, 2017; Zhu *et al.*, 2017) have explored this idea about fusion of 'Spatial Stream' to capture local image level dependency (for example, *object in an image*) and 'Temporal Stream' to explicitly capture the temporal motion (for example, *motion of objects from one scene to another*). This sort of architecture makes common use of pre-trained cutting edge *CNN* models (Simonyan and Zisserman, 2014*b*; He *et al.*, 2016) for feature extraction.

On the top of previous works, Carreira and Zisserman (2017) take advantage of both multi-stream networks and C3D convolutions by using 3-D convolution networks in their 'Temporal Stream'. Unlike previous works, Zhu *et al.* (2017) generate optical flow images on-the-fly in an unsupervised manner. This effectively improves the training time and memory requirements of the model. The summary of different video classification (in other words, action recognition) architectures has been pictorially shown by Carreira and Zisserman (2017) and reproduced here for the reader's reference.

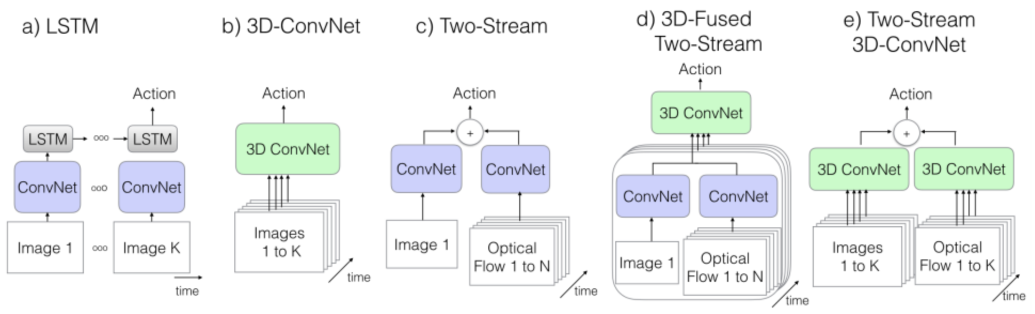


Figure 3.3: Different ways in which 'Temporal' and 'Spatial' streams are fused (Carreira and Zisserman, 2017). Here, ConvNet stands for *convolutional neural network*

.

3.4.4 Encoder-Decoder Based Methods

Due to their huge success in machine translation (Cho *et al.*, 2014; Bahdanau Dzmitry, 2015), encoder-decoder networks like Recurrent Neural Network (RNN) (Jordan, 1997), Long Short Term Memory Network (LSTM) (Hochreiter and Schmidhuber, 1997) are widely used for encoding the sequential data. These networks are described in detail in the background chapter. Several attempts have been made to explore and design various encoder-decoder based models for the task of video classification (Yue-Hei Ng *et al.*, 2015; Donahue *et al.*, 2015; Wu *et al.*, 2016; Li *et al.*, 2017; Pavel *et al.*, 2018; Choi and Zhang; Shivam, 2018). These networks possess inherent property of capturing long-term dependencies in the sequence. Hence, they are more suited for video input, which contains temporal dependencies among the frames.

Donahue *et al.* (2015) extract features for each of the input frames in the video, and pass the sequence of obtained visual features to an LSTM network (see Figure 3.4). Predictions $\hat{y}$ are made at each step of LSTM and are eventually pooled/averaged to obtain the *final class prediction* of the model. However, this method may lead to the problem of 'false class predictions' at some steps of LSTM, which adversely affects the final class prediction. Yue-Hei Ng *et al.* (2015) tries to improve the above architecture by computing a single video level representation using only the final output state of the LSTM. The encoder-decoder methods are widely used in other applications as well, for example, *video-captioning i.e.*, textual description of a video (Donahue *et al.*, 2015), generating visual summary (Gong *et al.*, 2014; Pan *et al.*, 2016) (generally, called *video abstraction*), answering questions (Jang *et al.*, 2017) from video and so on.

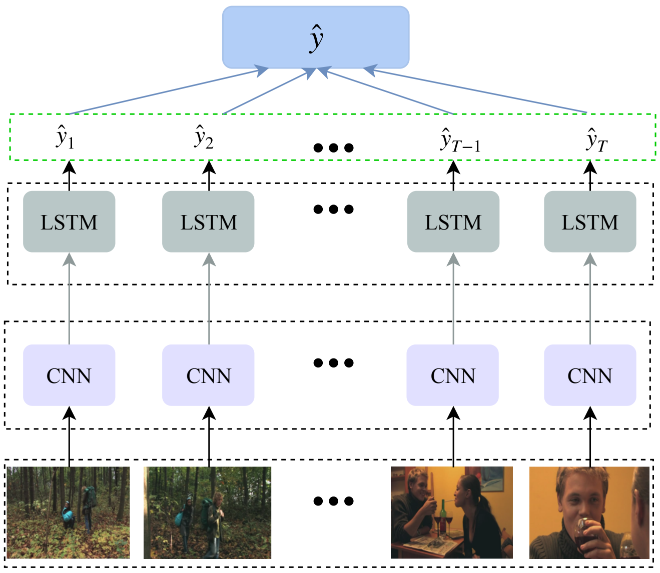


Figure 3.4: An *Encoder-Decoder* architecture used by Donahue *et al.* (2015) for the classification of a video of length T

3.4.5 Cluster-Aggregation Based Methods

Unlike the sequential aggregation models (*like Encoder-Decoder*) discussed above, there is an orthogonal direction of research which aggregates the features of a video in an unordered way. It is based on clustering methods such as Bag-of-words (Sivic and Zisserman, 2015), VLAD (Vector of Locally Aggregated Descriptors) (Xu *et al.*, 2015*b*), variants of Fisher Vectors (FV) (Lev *et al.*, 2016; Peng *et al.*, 2014) *etc.* All of these aggregation methods primarily focus on *learning* spatial dependencies among the frame-level features using clustering. They typically use pre-trained feature extraction models to generate the frame-level features and then, feed them to the *aggregation* module. By introducing differentiable modules, NetVLAD (Arandjelovic *et al.*, 2016) makes these networks end-to-end trainable. Further, Girdhar *et al.* (2017) explore the application of this differentiable NetVLAD model specifically for the task of video classification.

3.5 Video Classification On YouTube-8M Dataset

In this section, we specifically discuss previous work on video classification using the YouTube-8M dataset. As discussed earlier, it is one of the most challenging video classification datasets which contains videos having an average length of 200 seconds. We use this dataset in all our experiments. The authors of this dataset proposed a simple baseline model which treats the entire video as a sequence of one-second frames and uses a Long Short-Term Memory network (LSTM) to encode this sequence. Apart from this, they also propose some simple baseline models like Deep Bag of Frames (DBoF) and Logistic Regression (Abu-El-Haija *et al.*, 2016). Various other classification models (Miech *et al.*, 2017; Wang *et al.*, 2017*b*; Li *et al.*, 2017; Chen *et al.*, 2017*b*; Skalic *et al.*, 2017) have been proposed and evaluated on this dataset (2017 version) which explore different methods of: (i) feature aggregation in videos (temporal as well as spatial) (Chen *et al.*, 2017*b*; Miech *et al.*, 2017), (ii) capturing the interactions between labels (Wang *et al.*, 2017*b*) and (iii) learning new non-linear units to model the interdependencies among the activations of the network (Miech *et al.*, 2017). The state-of-the-art model on the 2017 version of the Youtube-8M dataset uses *NetVLAD* pooling (Miech *et al.*, 2017) to aggregate information from all the frames of a video.

In the recently concluded competition (2018 version) (Joonseok *et al.*, 2018), many methods (Miha and David, 2018; Pavel *et al.*, 2018; Rongcheng *et al.*, 2018; Tang *et al.*, 2018; Choi and Zhang; Liu Tianqi, 2018; Shivam, 2018; Joonseok *et al.*, 2018) were proposed to compress the models such that they fit in 1GB of memory. As mentioned by Joonseok *et al.* (2018), the major motivation behind this competition was to avoid the late-stage model ensemble techniques and focus mainly on single model architectures at inference time (Shivam, 2018; Rongcheng *et al.*, 2018; Choi and Zhang). One of the top performing systems in this competition was *NeXtVLAD* (Rongcheng *et al.*, 2018), which modifies *NetVLAD* (Miech *et al.*, 2017) to squeeze the dimensionality of the modules (embeddings). However, this model still processes all the frames of the video and hence has a large number of FLOPs. In this work, we take this compact *NeXtVLAD* model and make it compute efficient by using the idea of distillation. The workshop primarily emphasized on the development of single model architectures and not ensembles. Hence, in this thesis, we focus on single model-based solutions.

3.6 Reinforcement Learning For Videos

The key intuition behind using reinforcement learning (RL) for videos is that 'parsing a video' can be viewed as a sequential process of decision making *i.e.*, deciding whether to *look* at a frame or *skip* it. However, in most of the video-related tasks (Zhang *et al.*, 2016*b*; Yue-Hei Ng *et al.*, 2015; Song *et al.*, 2016), the model processes all (or fixed) frames in a video as there is no frame-level 'importance' information available. To automatically reduce the number of processed frames, reinforcement learning can be used to train an agent which learns to *skip* the irrelevant ones to generate a subset of *important* frames for the end task. From the literature of reinforcement learning, a popular algorithm called REINFORCE (Williams, 1992) has been extensively exploited to learn a policy network pertaining to the end-task for various computer vision applications. For example, Ba *et al.* (2014) and Mnih *et al.* (2014) utilize the *policy network* for learning *spatial* attention on the images for the classification task. Many recent works have applied RL algorithms on other image-related tasks as well, for example, image captioning Xu *et al.* (2015*a*) and object detection Mathe *et al.* (2016).

An important domain where RL algorithms have proven to be effective is *video summarization* (*i.e.*, selection of key-frames in a video). Previously, Gong *et al.* (2014) solved the task of video summarization by using the true labels information (*importance scores*) of frames and the overall category of a video. Further, Song *et al.* (2016) used RL algorithms to learn the key-frames but still relied on the true category information of the frames. However, intuitively, *video summarization* is an inherently unsupervised task, as there doesn't exist a unique ground truth summary for a video. Yeung *et al.* (2016) framed the problem of video summarization as a *temporal bound detection* problem in an end-to-end trainable framework. In this framework, an RL agent is trained to predict the 'temporal bounds' *i.e.*, the direct localization of activity in a video. Further, Zhou *et al.* (2018) effectively used RL to solve the problem of key-frames selection in an unsupervised way.

In a video classification task, the common approach is to either sample a fixed set of frames or use all the frames of a video. The large redundancy in the neighboring frames gives rise to the problem of generating a subset of frames, which are more relevant to the task. The primary motivation behind this problem is to prevent further processing of irrelevant frames. Hence, this problem is somewhat related to ‘unsupervised video summarization’. There is a newer line of research that focuses on the important *frame-selection* using Reinforcement Learning techniques. For example, in very recent work on video classification by Fan *et al.* (2018), the authors use a Reinforcement Learning agent to select the frames to process for a classification task. They only take a subset of the YouTube-8M dataset containing only the top-20 categories as opposed to the total $\sim$4k categories. However, unlike Fan *et al.* (2018), we take the diversity of the Youtube-8M dataset into account and build the RL framework accordingly. Another closely related work by Chen *et al.* (2018) in the domain of video captioning tried to tackle the *frame-selection* problem differently. Unlike the setting of our framework, which entirely *skips* the processing of unimportant frames, the framework proposed by Chen *et al.* (2018) observes all the frames at least once while parsing a video.

3.7 Summary

In this chapter, we discussed the previous work in the literature germane to the thesis. We started off with the discussion of memory-efficient models developed in the literature of video/image-related tasks, with a special emphasis on work based on the idea of knowledge distillation. We also described various video classification benchmarks datasets curated so far. Thereafter, we set forth various techniques used for the purpose of extracting features from images and aggregating them to get a single representation for the video. We further described different video classification models developed using the Youtube-8M dataset and concluded the chapter with a discussion of different RL methods explored for video-related tasks.

CHAPTER 4

A Teacher-Student Framework for Video Classification

In the previous chapter we discussed various methods for computing the representation of a video with the aim of using this representation for a downstream task such as classification. In particular, we discussed (i) *recurrent neural network* based methods (ii) *cluster-and-aggregate* based methods and (iii) 3D convolution based methods. We observe that in each of these methods, the computation to process an input video needs to be done for *every* frame in the video. Therefore, these methods require a large number of floating point operations (FLOPs) especially in the case of longer videos. We aim to reduce this computation cost (FLOPs) by building a framework that can compute a representation of the video by looking at only a fraction of frames in the video. To do so, we utilize the idea of *distillation* wherein the *teacher* network has access to *all* frames in the input video, while the *student* network is allowed to observe only a few frames. Our approach primarily focuses on accelerating the number of frame computations in a network at inference time, which can potentially be combined with any approach of building a compact memory-efficient network. In other words, our work complements the work done on building memory-efficient models for video classification.

In this chapter, we start with discussion on state-of-the-art models for video classification, *viz.*, (i) a hierarchical *RNN* based model (ii) *NetVLAD* and (iii) *NeXtVLAD* which is a memory-efficient version of *NetVLAD* and was the best single model in the ECCV'18 workshop, see section 4.1. We introduce our *Teacher-Student* framework in section 4.2, which is developed to achieve the goal of efficient video classification. We experiment with two different methods of training the *Teacher-Student* network. In the first method (which we call *Serial* Training), the teacher is trained independently and then the student is trained to match the teacher with or without an appropriate regularizer to account for the classification loss. In the second method (which we call *Parallel* Training), the teacher and student are trained jointly using the classification loss as well as the matching loss. This *parallel* training method is similar to on-the-fly knowledge distillation from a dynamic teacher as mentioned in Lan Xu (2018). We use only the *compute-efficient* student network for classification at the inference time.

Note that as per the findings of the recently concluded workshop on YouTube-8M Large-Scale Video Understanding (2018), 3D convolution based methods (Carreira and Zisserman, 2017) were not so popular as they are expensive in terms of their memory and compute requirements. For example, the popular I3D model Carreira and Zisserman (2017) is trained using 64 GPUs as mentioned in the original paper. Hence, in this thesis, we do not investigate such methods based on 3D convolutions.

4.1 Video Classification Models

Given a fixed set of m classes $y_1, y_2, y_3, ..., y_m \in \mathscr{Y}$ and a video $\mathbf{V}$ containing N frames $(F_0, F_1, \ldots, F_{N-1})$, the goal of video classification is to identify all the classes to which the video belongs. In other words, for each of the m classes we are interested in predicting the probability $P(y_i|\mathbf{V})$. This probability can be parameterized using a neural network f which looks at all the frames in the video to predict:

$$P(y_i|\mathbf{V}) = f(F_0, F_1, \ldots, F_{N-1})$$

Given the setup, we now briefly discuss the two state-of-the-art models that we have considered in our work.

4.1.1 Recurrent Network Based Models

We consider the Hierarchical Recurrent Neural Network (H-RNN) (Pan *et al.*, 2016) based model which assumes that each video contains a sequence of b equal sized blocks. Each of these blocks in turn is a sequence of l frames thereby making the entire video a sequence of sequences. The details of the model have been discussed in section 2.3.2. In the case of the YouTube-8M dataset, these frames are one-second shots of the video and each block b is a collection of l such one-second frames. The model contains a lower level RNN to encode each block (sequence of frames) and a higher level RNN to encode the video (sequence of blocks).

4.1.2 Cluster and Aggregate Based Models

We consider the *NetVLAD* model with Context Gating (CG) as proposed in Miech *et al.* (2017). This model does not treat the video as a sequence of frames but simply as a bag of frames. For every frame in this bag, it first assigns a soft cluster to the frame which results in a $M \times k$ dimensional representation for the frame where k is the number of clusters considered and M is the size of the initial representation of the frame (say, obtained from a CNN). Instead of using a standard clustering algorithm such as k-means, the authors introduce a learnable clustering matrix which is trained along with all the parameters of the network. The cluster assignments are thus a function of a parameter which is learned during training. The video representation is then computed by aggregating all the frame representations obtained after clustering. This video representation is then fed to multiple fully connected layers with Context Gating (CG) which help to model interdependencies among network activations. We also experiment with *NeXtVLAD* (Rongcheng *et al.*, 2018) which is a compact version of *NetVLAD* wherein the $M \times k$ dimensional representation is downsampled by grouping which effectively reduces the total number of parameters in the network. An important point to be considered here is that all the models described above look at all the frames in a video.

4.2 Proposed Approach

The main idea of this work is to leverage the distillation technique for *compute-efficient* video classification, wherein we train a computationally expensive *teacher* network which computes a representation for the video by processing *all* frames in the video. We then train a relatively inexpensive *student* network whose objective is to process only a few frames of the video and produce a representation which is very similar to the representation computed by the teacher. In particular, this is achieved by minimizing (i) the squared error loss between the representations of the student network and the teacher network and/or (ii) by minimizing the difference between the output distributions (class probabilities) predicted by the two networks. Figure 4.1 illustrates this idea where the teacher sees every frame of the video but the student sees fewer frames, *i.e.*, every j-th frame of the video. At inference time, we then use the *compute-efficient*

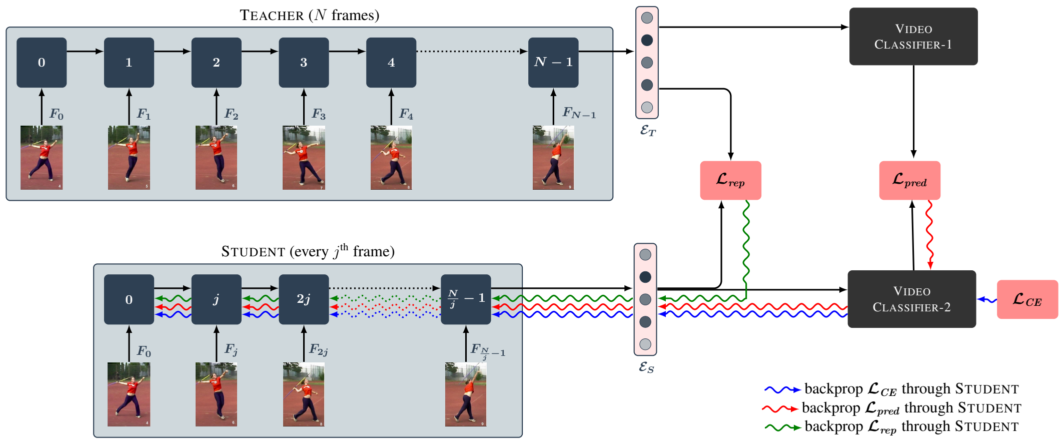


Figure 4.1: Architecture of TEACHER-STUDENT network for video classification

student network for classification thereby reducing the time required for processing the video.

4.2.1 Teacher Network

The teacher network can be any state-of-the-art model described above (*H-RNN*, *NetVLAD*, *NeXtVLAD*). This teacher network looks at all the N frames of the video (F_0, F_1,...,F_{N-1}) and computes an encoding $\mathcal{E}_T$ of the video, which is then fed to a simple feed-forward neural network with a multi-class output layer containing a sigmoid neuron for each of the $\mathscr{Y}$ classes. The parameters of the teacher network are learnt using a standard multi-label classification loss $\mathcal{L}_{CE}^{T}$, which is a sum of the cross-entropy loss for each of the $\mathscr{Y}$ classes. We refer to this loss as $\mathcal{L}_{CE}^{T}$ where the subscript CE stands for cross entropy between the true labels y and predictions $\hat{y}$.

$$\mathcal{L}_{CE}^{T} = -\sum_{i=1}^{|\mathscr{Y}|} y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \tag{4.1}$$

4.2.2 Student Network: Final Matching

In addition to this teacher network, we introduce a student network which only processes every j^{th} frame ($F_0, F_j, F_{2j}, \ldots, F_{\frac{N}{j}-1}$) of the video and computes a representation $\mathcal{E}_S$ of the video from these $\frac{N}{j} = k$ frames. We use only same family distillation wherein both the teacher and the student have the same architecture. For example, Figure 4.1 shows the setup when the teacher is *H-RNN* and the student is also *H-RNN*. Further, the parameters of the output layer are shared between the teacher and the student. The student is trained to minimize the squared error loss between the representation computed by the student network $\mathcal{E}_S$ and the representation computed by the teacher $\mathcal{E}_T$. We refer to this loss as $\mathcal{L}_{rep}$ where the subscript *rep* stands for representations:

$$\mathcal{L}_{rep} = ||\mathcal{E}_T - \mathcal{E}_S||^2 \tag{4.2}$$

4.2.3 Student Network: Intermediate Matching

We have tried a simple variant of the model, where in addition to ensuring that the final representations $\mathcal{E}_S$ and $\mathcal{E}_T$ are similar, we also ensure that the intermediate representations ($\mathcal{I}_S$ and $\mathcal{I}_T$) of the models are similar. In particular, we ensure that the representation of the frames j, $2j$ and so on computed by the teacher and student network are very similar by minimizing the squared error distance between the corresponding intermediate representations. We refer to this loss as $\mathcal{L}_{rep}^{\mathcal{I}}$ where the superscript $\mathcal{I}$ stands for intermediate:

$$\mathcal{L}_{rep}^{\mathcal{I}} = \sum_{i=j,2j,..}^{\frac{N}{j}-1} ||\mathcal{I}_T^i - \mathcal{I}_S^i||^2 \tag{4.3}$$

4.2.4 Knowledge Distillation on Class Probabilities

Alternately, the student can also be trained to minimize the difference between the class probabilities predicted by the teacher and the student. We refer to this loss as $\mathcal{L}_{pred}$ where the subscript $pred$ stands for *predicted probabilities*. More specifically if $\mathcal{P}_T = \{p_T^1, p_T^2, ..., p_T^m\}$ and $\mathcal{P}_S = \{p_S^1, p_S^2, ..., p_S^m\}$ are the probabilities predicted for the m classes by the teacher and the student respectively, then

$$\mathcal{L}_{pred} = d(\mathcal{P}_T, \mathcal{P}_S) \tag{4.4}$$

where d is any suitable distance metric such as *KL* divergence or squared error loss.

4.3 Baseline Models

The student network only processes k ($< N$) frames in the video. We have considered different baselines to explore the possible selections of frames in a video sequence. We report results with different values of k : 6, 10, 15, 20, 30 and compare the performance of our student networks with the following models:

a) Teacher-Skyline: The original hierarchical model which processes all the frames of the video. This, in some sense, acts as the upper bound on the performance.

b) Uniform-k : A hierarchical model trained from scratch which only processes k frames of the video. These frames are separated by a constant interval and are thus equally spaced. However, unlike the student model this model does not try to match the representations produced by the full teacher network.

c) Random-k: A hierarchical model which only processes k frames of the video. These frames are sampled randomly from the video and may not be equally spaced.

d) First-k: A hierarchical model which processes the first k frames of the video.

e) Middle-k: A hierarchical model which processes the middle k frames of the video.

f) Last-k: A hierarchical model which processes the last k frames of the video.

g) First-Middle-Last-k: A hierarchical model which processes k frames by selecting the first $\frac{k}{3}$, middle $\frac{k}{3}$ and last $\frac{k}{3}$ frames of the video.

4.4 Different Training Paradigms

Intuitively, it makes sense to train the teacher first and then use this trained teacher to guide the student. We refer to this as the *Serial* mode of training as the student is trained after the teacher as opposed to jointly. For the sake of analysis, we use different combinations of loss functions to train the student as described below:

(i) $\mathcal{L}_{rep}$: Here, we operate in two stages. In the first stage, we train the student network to minimize $\mathcal{L}_{rep}$ as defined above, *i.e.*, we train the parameters of the student network to produce representations which are very similar to the teacher network. The idea is to let the student learn by only mimicking the teacher and not worry about the final classification loss. In the second stage, we then plug in the classifier trained along with the teacher (see Eq.4.1) and fine-tune all the parameters of the student and the classifier using the cross entropy loss, $\mathcal{L}_{CE}$. In practice, we found that the fine-tuning done in the second stage helps to improve the performance of the student.

(ii) $\mathcal{L}_{pred}$: Here, in the first stage of training, we train the student to only minimize the difference between the class probabilities predicted by the teacher and the student. Similar to the case (i), in the second stage of training, we plug in the classifier trained with teacher and fine-tune the parameters using $\mathcal{L}_{CE}$.

(iii) $\mathcal{L}_{rep} + \mathcal{L}_{CE}$: Here, we train the student to jointly minimize the representation loss as well as the classification loss. The motivation behind this was to ensure that while mimicking the teacher, the student also keeps an eye on the final classification loss from the beginning (instead of being fine-tuned later as in the above cases).

(iv) $\mathcal{L}_{pred} + \mathcal{L}_{CE}$: Here, in addition to mimicking the probabilities predicted by the teacher, the student is also trained to minimize the cross entropy loss.

(v) $\mathcal{L}_{rep} + \mathcal{L}_{CE} + \mathcal{L}_{pred}$: Finally, we combine all the three loss functions. Figure 4.1 visually illustrates the process of training the student with different loss functions. For the implementation of this setting, we present the complete pseudo code in Algorithm 1. We experimented with different weightage values of the loss functions $\mathcal{L}_{rep}$, $\mathcal{L}_{CE}$ and $\mathcal{L}_{pred}$, and observe that the optimal combination assigns an equal weightage to all the loss functions (*i.e.*, $\theta_{rep} = \theta_{pred} = \theta_{CE}$).

For the sake of completeness, we also tried an alternate mode in which we train the teacher and student in parallel such that the objective of the teacher is to minimize $\mathcal{L}_{CE}$ and the objective of the student is to minimize one of the five combinations of loss functions described above. We refer to this as *Parallel* training.

4.5 Experimental Setup

In this section, we describe the dataset used for our experiments, the hyperparameters that we considered, the baseline models that we compared with and the effect of different loss functions and training methods.

Algorithm 1: Complete Pseudo-code for *Serial* mode of training *Student*

```
    Input: A labelled train set and an unlabelled test set of video clips
    Output: Predicted video category of each video in the test set

Stage 1: Train Teacher Network

  Initialize the weights of teacher network ($W_T$ and $W_T^{classify}$, see Figure 4.1)
  Train the parameters of teacher network ($W_T, W_T^{classify}$) using $\mathcal{L}_{CE}^T$, see Eq.4.1

Stage 2: Train Student Network

  Initialise the parameters of student network ($W_S$ and $W_S^{classify}$ in Figure 4.1)
  for $D$ epochs do
      Sample $k$ uniformly spaced frames from an input video
      Feed sampled frames through the student network to obtain $\mathcal{E}_S$
      Feed the whole video through teacher network to obtain $\mathcal{E}_T$
      Compute $\mathcal{L}_{rep}$ loss from video encoding obtained from the teacher ($\mathcal{E}_T$) and the
  student ($\mathcal{E}_S$)
      Compute $\mathcal{L}_{pred}$ loss using probability distributions $\mathcal{P}_T$ and $\mathcal{P}_S$ predicted by
  teacher and student classifiers respectively
      Compute the classification loss $\mathcal{L}_{CE}$ using $\mathcal{P}_S$
      Compute the final loss as weighted sum of all loss functions to train the student:

          $\mathcal{L}_{student} = \theta_{rep}\mathcal{L}_{rep} + \theta_{pred}\mathcal{L}_{pred} + \theta_{CE}\mathcal{L}_{CE}$

      Backpropagate $\mathcal{L}_{student}$ to train the parameters of student network
  end for

Stage 3: Inference Using Student Network

  for every video during inference do
      Sample $k$ uniformly spaced frames from the video
      Pass these frames through the student network ($W_S$ and $W_S^{classify}$ ) to obtain $\mathcal{P}_S$
      Predict the video category using the probability distribution $\mathcal{P}_S$
  end for
```

4.5.1 Dataset

The YouTube-8M dataset (2017 version) by Abu-El-Haija *et al.* (2016) contains 8 million videos with multiple classes associated with each video. The average length of a video is $200s$ and the maximum length of a video is $300s$. The authors of the dataset have provided pre-extracted audio and visual features for every video such that every second of the video is encoded as a single frame. The original dataset consists of 5,786,881 training (70%), 1,652,167 validation (20%) and 825,602 test examples (10%). Since the authors did not release the test set, we used the original validation set as test set and report results on it. In turn, we randomly sampled 48,163 examples from the training data and used these as validation data for tuning the hyperparameters of the model. We trained our models using the remaining 5,738,718 training examples.

4.5.2 Hyperparameters:

For all our experiments, we used Adam Optimizer (Kingma and Ba, 2015) with the initial learning rate set to 0.001 and then decreased it exponentially with 0.95 decay rate. We used a batch size of 256. For both the student and teacher networks we used a 2-layered MultiLSTM Cell with cell size of 1024 for both the layers of the hierarchical model. For regularization, we used dropout (0.5) and $\mathbf{L}_2$ regularization penalty of 2 for all the parameters. We trained all the models for 5 epochs and then picked the best model based on validation performance. We did not see any benefit of training beyond 5 epochs. For the teacher network we chose the value of l (number of frames per block) to be 20 and for the student network, we set the value of l to 5 or 3 depending on the reduced number of frames considered by the student.

For training the *NetVLAD* model, we used the standard hyperparameter settings as mentioned in Miech *et al.* (2017). We considered 256 clusters and 1024 dimensional hidden layers. Similarly, in the case of *NeXtVLAD*, we considered the hyperparameters of the single best model as reported by Rongcheng *et al.* (2018). In this network, we are working with a cluster size of 128 with hidden size as 2048. The input is reshaped and downsampled using 8 groups in the cluster as done in the original paper. For all these networks, we have worked with a batch size of 80 and an initial learning rate of 0.0002 exponentially decayed at the rate of 0.8. Additionally, we have applied dropout of 0.5

on the output of *NeXtVLAD* layer which helps in regularization.

4.5.3 Evaluation Metrics

We used the following metrics as proposed in Abu-El-Haija *et al.* (2016) for evaluating the performance of different models :

- GAP (Global Average Precision): is defined as:

  $$GAP = \sum_{i=1}^{P} p(i) \nabla r(i)$$

  where $p(i)$ is the precision at prediction i, $\nabla r(i)$ is the change in recall at prediction i and P is the number of top predictions that we consider. Following the original YouTube-8M Kaggle competition we use the value of P as 20.

- mAP (Mean Average Precision) : The mean average precision is computed as the unweighted mean of all the per-class average precisions.

4.6 Results And Discussion

Since we have 3 different base models (*H-RNN*, *NetVLAD*, *NeXtVLAD*), 5 different combinations of loss functions (see section 4.2), 2 different training paradigms (Serial and Parallel) and 5 different baselines for each base model, the total number of experiments that we needed to run to report all these results was very large. To reduce the number of experiments we first consider only the *H-RNN* model to identify the (a) best baseline (Uniform-k, Random-k, First-k, Middle-k, Last-k, First-Middle-Last-k) (b) best training paradigm (Serial v/s Parallel) and (c) best combination of loss function. We then run the experiments on *NetVLAD* and *NeXtVLAD* using only the best baseline, best training paradigm and best loss function thus identified. The results of our experiments using the *H-RNN* model are summarized in Table 4.1 to Table 4.4 and are discussed first followed by a discussion of the results using *NetVLAD* and *NeXtVLAD* as summarized in Tables 4.5 and 4.6:

Model		k=6		k=10		k=15		k=20		k=30	
		GAP	mAP	GAP	mAP	GAP	mAP	GAP	mAP	GAP	mAP
Model with k frames		Baseline Methods									
Uniform-k		0.715	0.266	0.759	0.324	0.777	0.35	0.785	0.363	0.795	0.378
Random-k		0.679	0.246	0.681	0.254	0.717	0.268	0.763	0.329	0.774	0.339
First-k		0.478	0.133	0.539	0.163	0.595	0.199	0.632	0.223	0.676	0.258
Middle-k		0.577	0.178	0.600	0.198	0.620	0.214	0.638	0.229	0.665	0.25
Last-k		0.255	0.062	0.267	0.067	0.282	0.077	0.294	0.083	0.317	0.094
First-Middle-Last-k		0.640	0.215	0.671	0.242	0.680	0.249	0.698	0.268	0.721	0.287
Training	*Student-Loss*	*Teacher-Student* Methods									
Parallel	$\mathcal{L}_{rep}$	0.724	0.280	0.762	0.331	0.785	0.365	0.794	0.380	0.803	0.392
Parallel	$\mathcal{L}_{rep}, \mathcal{L}_{CE}$	0.726	0.285	0.766	0.334	0.785	0.362	0.795	0.381	0.804	0.396
Parallel	$\mathcal{L}_{rep}, \mathcal{L}_{pred}, \mathcal{L}_{CE}$	0.729	0.292	0.770	0.337	**0.789**	0.371	0.796	0.388	**0.806**	0.404
Serial	$\mathcal{L}_{rep}$	0.727	0.288	0.768	0.339	0.786	0.365	0.795	0.381	0.802	0.394
Serial	$\mathcal{L}_{pred}$	0.722	0.287	0.766	0.341	0.784	0.367	0.793	0.383	0.798	0.390
Serial	$\mathcal{L}_{rep}, \mathcal{L}_{CE}$	0.728	0.291	0.769	0.341	0.786	0.368	0.794	0.383	0.803	0.399
Serial	$\mathcal{L}_{pred}, \mathcal{L}_{CE}$	0.724	0.289	0.763	0.341	0.785	0.369	0.795	0.386	0.799	0.391
Serial	$\mathcal{L}_{rep}, \mathcal{L}_{pred}, \mathcal{L}_{CE}$	**0.731**	**0.297**	**0.771**	**0.349**	**0.789**	**0.375**	**0.798**	**0.390**	**0.806**	**0.405**

Table 4.1: Performance comparison of proposed *Teacher-Student* models using different ***Student-Loss*** variants, with their corresponding baselines using k frames. ***Teacher-Skyline*** is the default model which processes all the frames in a video, and achieves a GAP and mAP score of **0.811** and **0.414** respectively.

4.6.1 Comparison of Different Baselines

First, we compare the performance of different baselines listed in the top half of Table 4.1. As is evident, the Uniform-k baseline which looks at equally spaced k frames performs better than all the other baselines. The performance gap between Uniform-k and the other baselines is even more significant when the value of k is small. The main purpose of this experiment was to decide the right way of selecting frames for the student network. Based on these results, we ensured that for all our experiments, we fed equally spaced k frames to the student.

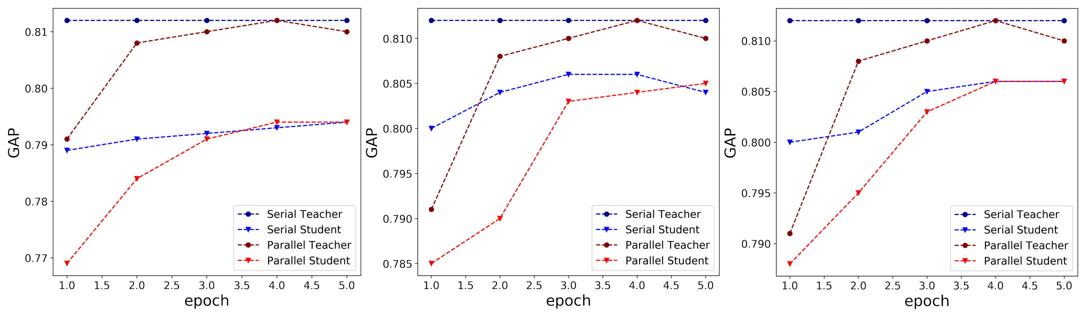
4.6.2 Comparing Teacher-Student Network with Uniform-k Baseline

As mentioned above, the Uniform-k baseline is a simple but effective way of reducing the number of frames to be processed. We observe that all the teacher-student models outperform this strong baseline. Further, in a separate experiment as reported in Table 4.2 we observe that when we reduce the number of training examples seen by the teacher and the student, then the performance of the Uniform-k baseline drops and is much lower than that of the corresponding *Teacher-Student* network. This suggests that the *Teacher-Student* network can be even more useful when the amount of training data is

Model	Metric	%age of training data		
		10	25	50
Serial	GAP	0.774	0.788	0.796
	mAP	0.345	0.369	0.373
Uniform	GAP	0.718	0.756	0.776
	mAP	0.220	0.301	0.349

Table 4.2: Effect of amount of training data on performance of Serial and Uniform models using 30 frames

limited.



(a) $\mathcal{L}_{rep}$ (b) $\mathcal{L}_{rep} + \mathcal{L}_{CE}$ (c) $\mathcal{L}_{rep} + \mathcal{L}_{CE} + \mathcal{L}_{pred}$

Figure 4.2: Performance comparison (GAP score) of different variants of *Serial* and *Parallel* methods in *Teacher-Student* training

.

4.6.3 Serial Versus Parallel Training of Teacher-Student

While the best results in Table 4.1 are obtained using *Serial* training, if we compare the corresponding rows of *Serial* and *Parallel* training we observe that there is not much difference between the two. We found this to be surprising and investigated this further. In particular, we compared the performance of the teacher after different epochs in the *Parallel* training setup with the performance of a static teacher trained independently (*Serial*). We plotted this performance in Figure 4.2 and observed that after 3-4 epochs of training, the *Parallel* teacher is able to perform at par with the *Serial* teacher (the constant blue line). As a result, the *Parallel* student now learns from this trained teacher for a few more epochs and is almost able to match the performance of the *Serial* student. This trend is same across the different combinations of loss functions that we used.

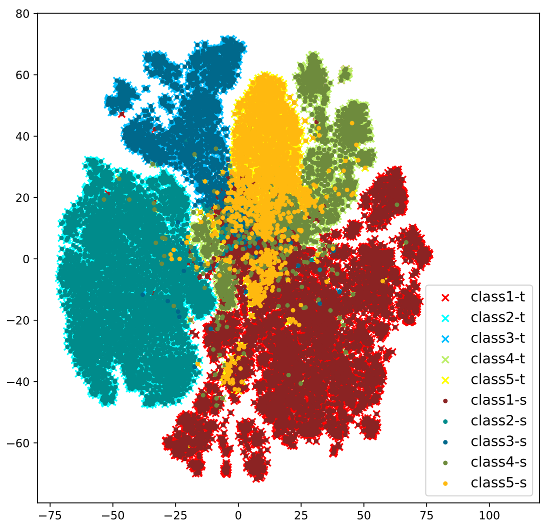


Figure 4.3: TSNE-Embedding of teacher and student representations. Here, class c refers to the cluster representation obtained corresponding to c^{th} class, wherein t and s denote teacher and student embedding respectively.

4.6.4 Visualization of Teacher and Student Representations

Apart from evaluating the final performance of the model in terms of mAP and GAP, we also wanted to check if the representations learned by the teacher and student are indeed similar. To do this, we chose top-5 classes (*class1: Vehicle, class2: Concert, class3: Association football, class4: Animal, class5: Food*) in the Youtube-8M dataset and visualized the TSNE-embeddings of the representations computed by the student and the teacher for the same video (see Figure 4.3). We use the darker shade of a color to represent teacher embeddings of a video and a lighter shade of the same color to represent the student embeddings of the same video. We observe that the dark shades and the light shades of the same color almost completely overlap showing that the student and teacher representations are indeed very close to each other. This shows that introducing the $\mathcal{L}_{rep}$ loss indeed brings the teacher and student representations close to each other.

4.6.5 Matching Intermediate and Final Representations

Intuitively, it seemed that the student should benefit more if we train it to match the intermediate representations of the teacher at different timesteps as opposed to only the final representation at the last time step. However, as reported in Table 4.3, we did not

Model		Intermediate		Final	
		GAP	mAP	GAP	mAP
Parallel	$\mathcal{L}_{rep}$	0.803	0.393	0.803	0.392
Parallel	$\mathcal{L}_{rep} + \mathcal{L}_{CE}$	0.803	0.396	0.804	0.396
Parallel	$\mathcal{L}_{rep} + \mathcal{L}_{pred}$	0.804	0.400	0.806	0.404
Serial	$\mathcal{L}_{rep}$	0.804	0.395	0.802	0.394
Serial	$\mathcal{L}_{rep} + \mathcal{L}_{CE}$	0.803	0.397	0.803	0.399
Serial	$\mathcal{L}_{rep} + \mathcal{L}_{pred}$	0.806	0.405	0.806	0.405

Table 4.3: Comparison of *Final* and *Intermediate* representation matching by *Student* network using k=30 frames

Model	Time (hrs.)	Avg. time taken by a video (in seconds)	FLOPs (Billion)
Teacher-Skyline	13.00	2.185e-09	5.058
k= 10	7.61	1.279e-09	0.167
k= 20	8.20	1.378e-09	0.268
k= 30	**9.11**	1.531e-09	0.520

Table 4.4: Comparison of FLOPs and evaluation time of models using k frames with *Skyline* model on original validation set using Tesla k80s GPU. The average length of a video in the set is 230 seconds

.

see any benefit of matching intermediate representations.

4.6.6 Computation time of different models

The main aim of this work was to ensure that the computational cost and time is minimized at inference time. The computational cost can be measured in terms of the number of FLOPs. As shown in Table 4.4 when k=30, the inference time drops by 30% and the number of FLOPs reduces by approximately 90%, but the performance of the model is not affected. In particular, as seen in Table 4.1, when $k = 30$, the GAP and mAP drop by 0.5-0.9% and 0.9-2% respectively as compared to the teacher skyline.

4.6.7 Performance using *NetVLAD* models

In Table 4.5 we summarize the results obtained using *NetVLAD* as the base model in the *Teacher-Student* network. Here the student network was trained using the best loss function ($\mathcal{L}_{rep}, \mathcal{L}_{pred}, \mathcal{L}_{CE}$) and the best training paradigm (*Serial*) as identified from the experiments done using the *H-RNN* model. Further, we consider only the Uniform-

Model: *NetVLAD*	k=10		k=30	
	mAP	GAP	mAP	GAP
Skyline			**0.462**	**0.823**
Uniform	0.364	0.773	0.421	0.803
Student	0.383	0.784	0.436	0.812

Table 4.5: Performance of *NetVLAD* model with k= 10, 30 frames in proposed Teacher-Student Framework

Model: *NeXtVLAD*	k=30		FLOPs
	mAP	GAP	(in Billion)
Skyline	0.464	0.831	1.337
Uniform	0.424	0.812	0.134
Student	0.439	0.818	0.134

Table 4.6: Performance and FLOPs comparison in *NeXtVLAD* model with k=30 frames in proposed *Teacher-Student* Framework

k baseline as that was the best baseline as observed in our previous experiments. Here again we observe that the student network does better than the Uniform-k baseline.

4.6.8 Combining with memory efficient models

Lastly, we experiment with the compact *NeXtVLAD* model and show that the student network performs slightly better than the Uniform-k baseline in terms of mAP but not so much in terms of GAP (note that mAP gives equal importance to all classes but GAP is influenced more by the most frequent classes in the dataset). Once again, there is a significant reduction in the number of FLOPs (approximately 89%).

4.7 Summary

We proposed a method to reduce the computation time for video classification using the idea of distillation. Specifically, we first train a teacher network which computes a representation of the video using all the frames in the video. We then train a student network which only processes k frames of the video. We use different combinations of loss functions which ensures that (i) the final representation produced by the student is similar to that produced by the teacher and (ii) the output probability distributions produced by the student are similar to those produced by the teacher. We compare the proposed models with a strong baseline and skyline and show that the proposed model outperforms the baseline and gives a significant reduction in terms of computational time and cost when compared to the skyline. In particular, we evaluate our model on the YouTube-8M dataset and show that the computationally less expensive student network can reduce the computation time by 30% while giving almost the performance as the teacher network.

CHAPTER 5

Dynamic Frame Selection

In the previous chapter, we showed that using *Knowledge Distillation* we can train a student network which only looks at a few uniformly sampled frames but still gives comparable performance to a teacher which looks at all the frames. In this chapter, we focus on the problem of dynamically selecting fewer frames from a video instead of selecting uniformly sampled frames. Here, the importance of a frame is determined by how relevant it is for the *central theme* of the video. To achieve this goal, we design a reinforcement learning agent *SkipFrame*, which learns to sample *fewer* frames from a video with an end goal to classify it correctly. This eventually results in the processing of fewer frames at inference time and hence, makes the network more *compute-efficient*.

5.1 Overview of Proposed Approach

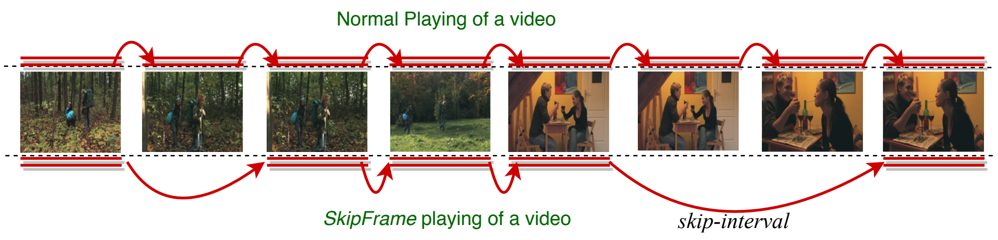


Figure 5.1: Frame skipping in a video

We train a reinforcement learning agent to look at the context of frames seen so far, and based on this information, learn to *skip* certain future frames by selecting an action from a predefined action set. We call the action set as *skip-interval*, which determines the number of frames to be skipped after the current frame is seen (selected). As a result, at each time step, the agent jumps directly to the frame after the skipped ones as depicted in Figure 5.1. After observing all the selected frames and reaching the end of the video, the agent feeds the selected sequence of frames to the classification network. The 'classification score' returned by the video classifier acts as a 'reward signal' (an evaluative feedback signal) for the agent. The end goal of an agent is always to maximize such 'reward signals' throughout the training process.

5.2 Modelling

To achieve our goal, we need to answer the following question: "*Does there exist a dynamic way in which we can select or skip the frames from a video, as a result of which only relevant frames are presented to the classification network?*" We address the above question using the model depicted in Figure 5.2. Essentially, we cast this problem in a reinforcement learning framework termed as *SkipFrame*, wherein the agent continually interacts with the outside environment, which is the overall context of the video seen so far. At each step $t = 1, 2, ..., T$ of an episode (training on a single video), it learns to skip the possibly unimportant future frames based on the history of video frames seen so far, wherein T is the length of an episode. We now move onto the discussion of different components in our proposed model.

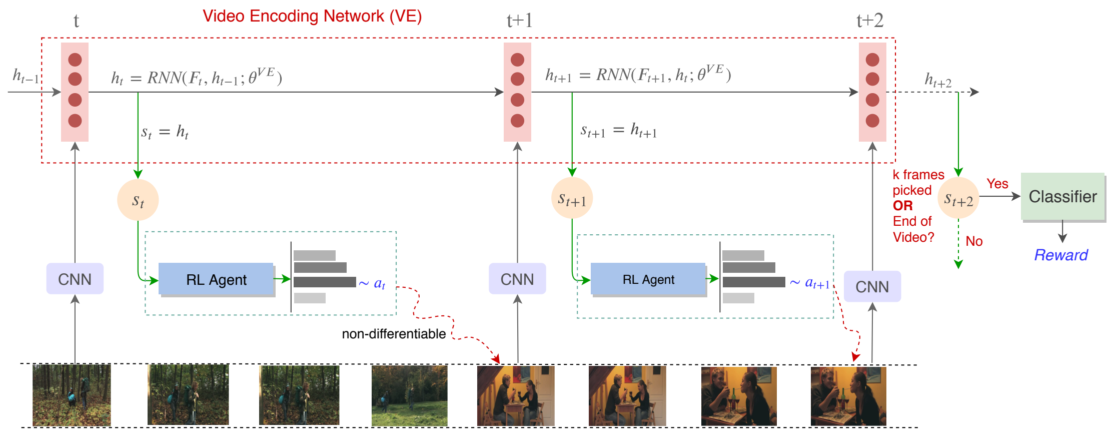


Figure 5.2: Architecture of *SkipFrame* for video classification

5.2.1 Video-Encoding Network

The first component in the *SkipFrame* model is a *video-encoding* network, which is used to encode an input video sequence of N frames $(F_1, F_2, \ldots, F_N)$. Abu-El-Haija *et al.* (2016) used the popular ResNet-50 model(He *et al.*, 2016) to pre-extract the spatial characteristics of each video frame F_t to obtain a frame feature at each step t. We further aggregate these frame features using a Hierarchical *RNN* (Pan *et al.*, 2016) to encode the temporal information spanned across the sequence (see section 2.3.2 for a detailed description). We denote the parameters of this video-encoding network by θ^{VE}. In general, the encoding network saves the history h_t of the video frames $F_1, F_2, ..., F_t$ seen until step t. The last hidden state h_N computed at the end of the video (*i.e.,* when $t = N$) intuitively represents an overall summary of the whole video sequence (see Eq.5.1).

$$h_t = \textit{H-RNN}(h_{t-1}, F_t; \theta^{VE}) \tag{5.1}$$

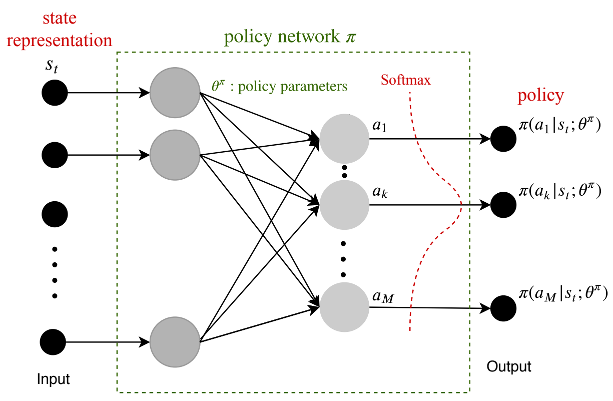
5.2.2 Classification Network

We feed the last hidden state of the encoding network h_N as an input to the classification network, which is parameterized by θ^C. We use Mixture-of-Experts classifier (MoE) (Miller and Uyar, 1997) for multi-label video classification(as proposed by Abu-El-Haija *et al.* (2016)). The output layer of this classifier generates a Bernoulli distribution (sigmoid output) for each of the $\mathscr{Y}$ classes in the data. The parameters of the video-encoding (θ^{VE}) and classification (θ^C) networks are trained using $\mathcal{L}_{CE}$, which is the sum of standard binary cross-entropy losses computed for each of the $\mathscr{Y}$ classes. Mathematically, we define this loss in Eq.5.2, wherein y and $\hat{y}$ are the true labels and predictions respectively.

$$\mathcal{L}_{CE} = -\sum_{i=1}^{|\mathscr{Y}|} y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \tag{5.2}$$

5.2.3 Reinforcement Learning Agent

The RL agent attempts to find the optimal policy for a finite and sequential Markov Decision Process (MDP) $\mathcal{M}$ ($\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}(.|s, a)$, $\mathcal{R}(s)$). Here, $\mathcal{S}$ is defined as a set of states ($s \in \mathcal{S}$), $\mathcal{A}$ is a set of plausible actions, $\mathcal{P}(.|s, a)$ is the probability distribution over all the next states, given that the agent selects an action a at a given state s and $\mathcal{R}(s)$ represents the expected reward obtained when transitioning to a state s . The state space $\mathcal{S}$ in the proposed setup represents the history of frames seen so far in the video. We restrict the agent to observe a maximum of k frames, *i.e.*, the length of an episode



state representation
s_t
policy network π
θ^{π} : policy parameters
Softmax
policy
a_1
a_k
a_M
$\pi(a_1|s_t;\theta^{\pi})$
$\pi(a_k|s_t;\theta^{\pi})$
$\pi(a_M|s_t;\theta^{\pi})$
Input
Output

Figure 5.3: Structure of policy network

can not exceed k, see Figure 5.2.

State Space: At each time step t, the agent looks at the current state s_t (encoding of video h_t), which is obtained from the integration of current frame F_t and the previous history h_{t-1} in the video-encoding network (see Figure 5.2). In general, we can use any feature-aggregation network for the purpose of encoding a video. The encoding h_t apparently summarizes the video seen so far and hence, is treated as the current state s_t of the environment seen by the agent.

Action Space: Given the current knowledge of state s_t, the agent selects an action a_t from the action set $\mathcal{A} = \{a^1, a^2, \ldots, a^M\}$, where M represents the maximum number of possible actions. Here, a_t determines the number of frames to be skipped by the agent from the current frame F_t called as *skip-interval*. In this work, we consider an action set containing 25 discrete actions: $\{a^1, a^2, \ldots, a^{25}\}$ mapped to the skip-intervals of {5, 6, ..., 29} and denoted by $\mathcal{A}^{25}$. The lower limit on the skip-interval makes sure that the agent learns to *skip* at least a small set of frames while parsing the video. Also, there is an upper bound on the number of frames to be skipped so as to not bypass any important frame in the sequence. The overall goal behind this design choice is to encourage the agent to explore the whole frame-sequence to find the important frames in a video.

Policy Network: We define *policy* as the probability distribution over the set of actions $\mathcal{A} = \{a^1, a^2, \ldots, a^M\}$ given the current state (see Figure 5.3). The goal of an RL agent is to find a policy π which maximizes the expected cumulative reward for the observed state s_t at each time step t. To achieve this goal, we implement the policy network by a simple feed-forward neural network f containing M neurons in the output layer. It is followed by a softmax activation on these M outputs, which is equivalent to the number of possible actions (see Figure 5.3). The parameters of the policy network f are denoted by θ^{π}. In *SkipFrame* architecture shown in Figure 5.2, the current hidden state h_t at each step obtained from the video-encoding network is passed to the policy network f as the state of the environment. The network outputs a probability distribution $\pi(a^i|h_t;\theta^{\pi})$ over all the possible actions $a^i \in \mathcal{A}$ (see Eq.5.3). The current action selected by the agent is sampled from this distribution *i.e.*, $a_t \sim \pi(.|h_t;\theta^{\pi})$ during the training process. Once the policy network has been trained well, the agent selects the action with the maximum probability estimate *i.e.*, $a_t = \arg\max_{a^i} \pi(a^i|h_t;\theta^{\pi})$ during inference.

$$\pi(a^i|h_t;\phi) = \frac{e^{f_i(s_t;\theta^{\pi})}}{\sum_{j=1}^{M} e^{f_j(s_t;\theta^{\pi})}} \tag{5.3}$$

Cumulative Reward: As explained above, based on the current action a_t selected by the agent, the next frame in the sequence F_{t+1} is picked, and the corresponding *CNN* feature is fed as the current input to *video-encoding* network (*H-RNN* in our case). As visually shown in Figure 5.3, video encoding obtained up to the step $t+1$ *i.e.,* h_{t+1} is treated as the next state of the environment s_{t+1}. The state s_{t+1} is then presented to the classifier to obtain an immediate reward r_{t+1} in the form of a classification score. The cumulative reward (expected future reward) at any time step t of an episode with length T can be defined as shown below:

$$R_t = \sum_{j=t}^{T-1} r_{j+1}(s_j, a_j) \tag{5.4}$$

5.3 Training

We train *SkipFrame* in a stagewise manner. In the first stage, we train the parameters of the video-encoding network *i.e.*, θ^{VE} and the classification network *i.e.*, θ^{C} with an objective function of a binary cross-entropy loss (classification loss) using standard backpropagation as defined in Eq.5.2. Subsequently, in the second stage, we utilize a Policy Gradient method namely REINFORCE (Williams, 1992) to train the parameters θ^{π} of the policy network, while keeping the parameters θ^{VE} and θ^{C} non-trainable. For further clarity on the implementation side, we provide the pseudocode of the training algorithm 2.

Algorithm 2: Pseudo-code for *SkipFrame* training with DELAY-REWARD design

Stage 1: *Train Video-Encoding (VE) Network*

```
Initialize the parameters of video-encoding network $\theta_{VE}$ (see Figure 5.2) and the classifier $\theta_C$
Train the parameters $\theta_{VE}$ and $\theta_C$ using clasisification loss $\mathcal{L}_{CE}$, see Eq.5.2
```

Stage 2: *Train SkipFrame*

```
Initialise the parameters of policy network $\theta^{\pi}$ (Figure 5.2)
for each video of $N$ frames in train set do
    Set timestep $t$=1 and budget of $k$ on the number of frames to be selected
    Select the first frame $F_1$ of video as the starting frame $F_t$ of the sequence
    repeat
        Feed the CNN feature of the current frame $F_t$ to the VE network (trained in Stage 1) to obtain the current state of environment $s_t$= $h_t$
        Feed the current state $s_t$ to policy network with parameters $\theta^{\pi}$
        Sample an action $a_t \sim \pi(.|s_t;\theta^{\pi})$, where $\pi$ is the action probability distribution generated by the policy network
        Select the next frame in the sequence $F_{t+1}$ according to the current action $a_t$
        Set the current frame $F_t$ = $F_{t+1}$
        Set time step $t = t + 1$

    until $t > k$ or end of the video
    Feed the state at final step $T$ (i.e., $s_T$) to the classifier to obtain a reward of $R_T$
    for each step $t$ of selected sequence do
        Compute the gradients of objective function $\mathcal{J}$ w.r.t policy network parameters $\theta^{\pi}$ (section 5.3.1)
        Backpropagate these gradients to train the parameters $\theta^{\pi}$ using Eq.5.8
    end for
end for
```

5.3.1 Policy Gradients

For training the policy network parameters θ^{π}, we need to design a suitable loss and reward function, and handle non-differentiable components using the REINFORCE trick. In this section, we explain the objective/performance function $\mathcal{J}$ and estimated gradient for training the parameters of the policy network. We can write the objective function of an RL agent as the expected reward of the whole sequence of states (frames) observed and the actions taken by the agent: $\{s_1,\ a_1,\ s_2,\ a_2,\ ...,\ s_{T-1}\ ,a_{T-1}\ ,s_T\}$ under the current policy $\pi(\cdot;\theta^{\pi})$, as given in Eq.5.5. The gradient of this objective w.r.t. the policy network parameters θ^{π} can be written as shown in Eq.5.6.

$$\mathcal{J}(\theta^{\pi}) = \mathbb{E}[\sum_{t=1}^{T-1} r_{t+1}] \tag{5.5}$$

$$\nabla_{\theta^{\pi}}\mathcal{J}(\theta^{\pi}) = \mathbb{E}[\sum_{t=1}^{T-1} \nabla \log \pi(a_t|s_t;\theta^{\pi}) \times R_t] \tag{5.6}$$

However, due to a high-dimensional search space comprising of the action sequences $\{a_1, a_2, ...,\ a_{T-1}\}$, estimating the value of the gradient using Eq.5.6 for a given policy becomes computationally expensive. Hence, we utilize REINFORCE by Williams (1992) to approximate the gradients using Monte-Carlo sampling as shown below:

$$\nabla_{\theta^{\pi}}\mathcal{J}(\theta^{\pi}) \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T-1} \nabla \log \pi(a_t^n|s_t^n;\theta^{\pi}) \times R_t^n \tag{5.7}$$

$$\theta^{\pi} \leftarrow \theta^{\pi} + \alpha\nabla_{\theta^{\pi}}\mathcal{J}(\theta^{\pi}) \tag{5.8}$$

It is to be noted that in Eq.5.7, N is the number of frame-sequences (Monte-Carlo *trajectories*) obtained according to the current policy $\pi(\cdot;\theta^{\pi})$. We estimate the cumulative reward R_t^n of the n^{th} trajectory at step 't' by following the current policy π itself to obtain the future rewards $r_{t+1}, r_{t+2}, ..., r_T$ up to the final step. Hence, the agent learns to maximize the log probability $\log \pi$ of actions which lead to better future rewards in the form of cumulative reward R. The policy parameters are then updated using standard gradient ascent as shown in Eq.5.8.

5.3.2 Exploration

An RL agent must intuitively try out any selected action multiple times to obtain an accurate estimate of the corresponding reward signal. To do so, the agent must *exploit* these selected actions in the future as well to receive more such rewards. However, it may result in a lack of sufficient *exploration* and evaluation of other actions. To encourage the exploration of different regions in the action space (in other words, different skip-intervals), ϵ-Greedy action selection (Sutton and Barto, 2018) is used.

In this work, we experiment with a better approach to encourage *exploration* as proposed by Mnih *et al.* (2016), which explicitly maximizes the entropy $\mathscr{E}(\theta^\pi)$ of the whole action probability distribution π, being generated at each step t by the *policy* network. Mathematically, it can be defined as written in Eq.5.9. By maximizing the entropy, we discourage the actions which achieve a high probability of selection in the distribution π.

$$\mathscr{E}(\theta^\pi) = -\frac{1}{T}\sum_{t=1}^{T}\{\frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}}\pi(a|h_t;\theta^\pi)\log\pi(a|h_t;\theta^\pi)\} \tag{5.9}$$

The final loss function $\mathcal{L}_{policy}$ used to train the policy network is a combination of two objective functions: $\mathcal{J}$ and $\mathscr{E}$, where the *entropy* function $\mathscr{E}$ has γ weightage in the final loss as shown below:

$$\min_{\theta^\pi}\{\mathcal{L}_{policy}(\theta^\pi)\} = \max_{\theta^\pi}\{\mathcal{J}(\theta^\pi) + \gamma\mathscr{E}(\theta^\pi)\} \tag{5.10}$$

5.3.3 Designing Reward

At each time step t, we receive an immediate reward signal r_{t+1} when the state changes from s_t to s_{t+1}. In the *SkipFrame* model, we experiment with two types of immediate reward signals namely, (i) negative classification loss (negative cross-entropy $\mathcal{L}_{CE}$, discussed in the Eq.5.2), (ii) GAP score (a performance metric discussed earlier in section 4.5.3). We further discuss two types of reward-designs for training *SkipFrame* using any one of these reward signals.

1. IMM-REWARD: In this reward-design, we consider the immediate reward received by the agent to be *non-zero* at each time step t. The video-encoding representation of the frame-sequence selected by the agent up to step t is passed onto

the classification network to obtain an immediate reward r_{t+1}. This immediate reward can be either negative $\mathcal{L}_{CE}$ or GAP performance.

2. DELAY-REWARD: In this reward-design, we consider *zero* immediate rewards for an incomplete parsing of the video when $t < T$, see Eq.5.11. Hence, the cumulative reward R_t at each step t for the frame-sequence chosen by the agent solely consists of the final-step immediate reward r_T, *i.e.*, $R_t = r_T \; \forall t$.

$$r_t = \begin{cases} (-\mathcal{L}_{CE}) \text{ or GAP} & \text{if t=T} \\ 0 & \text{if t<T} \end{cases} \tag{5.11}$$

5.4 Experimental Setup

5.4.1 Baselines

We consider the *skyline* network as the original hierarchical network (backbone network for video classification) trained with all the frames in a video. The *baseline* network also has the same structure, but is trained on a fixed budget of k frames, where k can be: (i) randomly spaced frames (Random-k) or (ii) uniformly spaced frames (Uniform-k) or (iii) the first k frames (First-k) or (iv) the middle k frames (Middle-k) or (v) the last k frames (Last-k) or (i) the first $\frac{k}{3}$, middle $\frac{k}{3}$ and last $\frac{k}{3}$ frames (First-Middle-Last-k), in the input video. For the experiments in this chapter, we use $k = 10$ to analyze the performance of an RL agent with a very small and strict budget.

5.4.2 Dataset and Evaluation

As mentioned earlier, we use the Youtube-8M dataset by Abu-El-Haija *et al.* (2016), which is a large-scale and the most diverse dataset for multi-label video classification. It is noteworthy that the authors provide only the pre-extracted features for images in the videos of this dataset for public use. We provided a detailed description of this dataset in section 4.5.1. The details of training, validation and test sets used in the experiments are mentioned in section 4.5.1. We limit our evaluation to only top-20 predictions for each test sample as mentioned in Abu-El-Haija *et al.* (2016). We use *mean Average Precision* (mAP) and *Global Average Precision* (GAP) metrics for evaluation in all the experiments as discussed earlier in section 4.5.3.

5.4.3 Hyperparameters

The hyperparameters used during the training of various networks have been described below.

Video-Encoding and Classification Network Training: The video-encoding network used is a Hierarchical RNN network (*H-RNN*) with 2-layered MultiLSTM Cell, wherein each cell has a hidden size of 1024. For the structure of *H-RNN*, we use the same configuration as explained earlier in section 4.5.2. The classification network used is a Mixture Of Experts (MoE) (Miller and Uyar, 1997) with three experts, each one being a standard logistic classifier. The video-encoding and classification network together are trained using the binary cross-entropy loss for approximately 5 epochs with a batch size of 64. For regularization, we used dropout (0.5) and $\mathbf{L}_2$ regularization penalty of 2 for all the parameters. For all our experiments, we used Adam Optimizer with the initial learning rate set to 0.001 and then decreased it exponentially with 0.95 decay rate.

Policy Network Training: The policy network used in this work consists of a single fully-connected layer with 25 hidden units. This is followed by a softmax activation on top, resulting in a probability distribution over the possible actions. The parameters of the policy network are initialized using Xavier (Glorot and Bengio, 2010*b*) initialization. For training the agent, we used Adam Optimizer with the initial learning rate set to 0.0001. The RL agent in our setting is trained for approximately 15 epochs with a batch size of 64 and the best model is chosen according to its performance on the validation set. We initialize the degree of exploration γ in the loss function $\mathcal{L}_{policy}$ (see Eq.5.10) to 0.001 and manually halve it after every 5 epochs of training.

5.5 Results and Discussion

We show the structure of the proposed framework in Figure 5.2. The performance of the *skyline* model, which uses all the frames in a video is shown in the first row of Table 5.1. We experiment with two different reward designs explained in section 5.3.3 namely: (i) IMM-REWARD, (ii) DELAY-REWARD using the action space $\mathcal{A}^{25}$ defined in section 5.2.3. Using the DELAY-REWARD, we also experiment with GAP as the 'reward' signal instead of the negative classification loss $(-\mathcal{L}_{CE})$. Further, we construct a new action space *Alt*-$\mathcal{A}^{25}$ by allowing alternating skip-intervals from the action set $\mathcal{A}^{25}$ *i.e.*, *Alt*-$\mathcal{A}^{25} = \{5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29\}$. It allows the agent to skip only the *alternating* number of frames after observing the current frame in a video. The results of these experiments are summarized in Table 5.1.

5.5.1 Performance Comparison of *Baselines* and *SkipFrame*

We initially compare the performance of baseline models, and as expected, Uniform-10 which looks at equally spaced 10 frames is a very strong baseline as compared to the rest of the models as shown in Table 5.1. It should be noted here that we use $-\mathcal{L}_{CE}$ as the reward signal unless otherwise mentioned. We then evaluate and compare the performance of the proposed *SkipFrame* model in different settings. Firstly, while comparing the reward-designs, we observe that DELAY-REWARD proves to be a better feedback signal for the sequence of frames sampled by the agent. It, therefore, correlates with the intuition that accurate evaluative feedback can only be obtained when the model has seen until the end of a video to classify it and hence, there is no true intermediate feedback.

Further, when we try to constrain the action space to *alternate* skip-interval {*Alt*-$\mathcal{A}^{25}$}, we observe that the agent performs poorly when compared to the standard action set $\mathcal{A}^{25}$. As reported in Table 5.1, we can see that none of the *SkipFrame* models explored so far is able to beat the Uniform-10 baseline. We achieve a better performance of the proposed model *SkipFrame* (GAP: 0.764 and mAP: 0.341) only when we use DELAY-REWARD design with GAP as the reward signal and utilize the full action space

Model	Reward-Design	Actions	GAP	mAP
Skyline	-	-	0.812	0.414
Uniform-10	-	-	**0.759**	**0.324**
Random-10	-	-	0.675	0.251
First-10	-	-	0.539	0.163
Middle-10	-	-	0.600	0.198
Last-10	-	-	0.267	0.067
SkipFrame	DELAY-REWARD	$\mathcal{A}^{25}$	0.755	0.322
	IMM-REWARD	$\mathcal{A}^{25}$	0.738	0.286
SkipFrame	DELAY-REWARD	*Alt*-$\mathcal{A}^{25}$	0.742	0.291
	IMM-REWARD	*Alt*-$\mathcal{A}^{25}$	0.739	0.288
SkipFrame	DELAY-REWARD: GAP	$\mathcal{A}^{25}$	**0.764**	**0.341**

Table 5.1: Performance comparison of different variants of the *SkipFrame* models and the baselines. For all the variants of *SkipFrame*, we fix a budget of 10 frames.

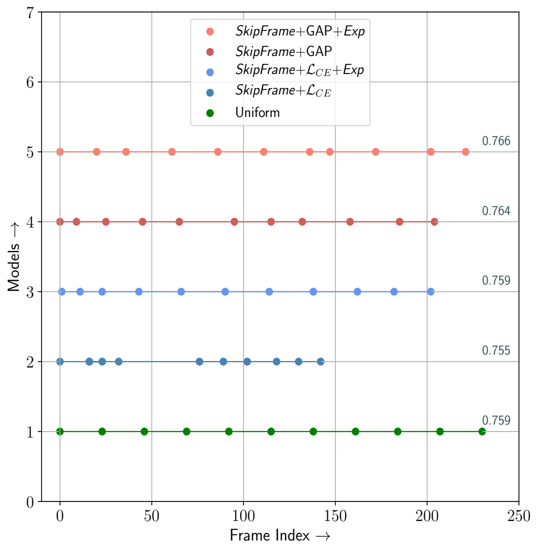


Figure 5.4: Comparison of frame-indices picked by different models.
Note: GAP score performance of each model is shown at the end of its series in the graph. The average number of frames in a video is 230.

of $\mathcal{A}^{25}$.

5.5.2 Frame Selection

We explicitly encourage the *SkipFrame* model for better exploration (denoted by *Exp*) of search space while skipping the irrelevant frames in the video. We compare the *location* (indices) of the frames eventually picked by the different models at inference time in Figure 5.4. The indices reported in Figure 5.4 are the frames selected for majority of the videos by the RL agent with the highest probability at inference time. We observe that *SkipFrame* model without any exploration (*Exp*) and $-\mathcal{L}_{CE}$ reward learns to focus on the initial half portion of the video alone (on average) and hence, performs even worse than the Uniform baseline. However with added exploration, it learns to better span the video sequence and performs as good as Uniform baseline. Similarly, *SkipFrame* model which uses GAP reward also benefits from the explicit exploration and thus, outperform the other models. A noteworthy observation is that these well-trained *SkipFrame* models however tend to select frames located in the *neighborhood* of uniformly spaced frames, as evident in Figure 5.4. In a separate experiment, we replace the initial half portion (precisely, initial 150 frames) of a video with random noise and observe the frame selection in this scenario. We notice that the sequence of skip-intervals selected at inference time (for majority of the videos) is: {20, 25, 22, 24, 29, 17, 19, 19, 20, 18}. We observe that the value of skip-interval selected by the agent

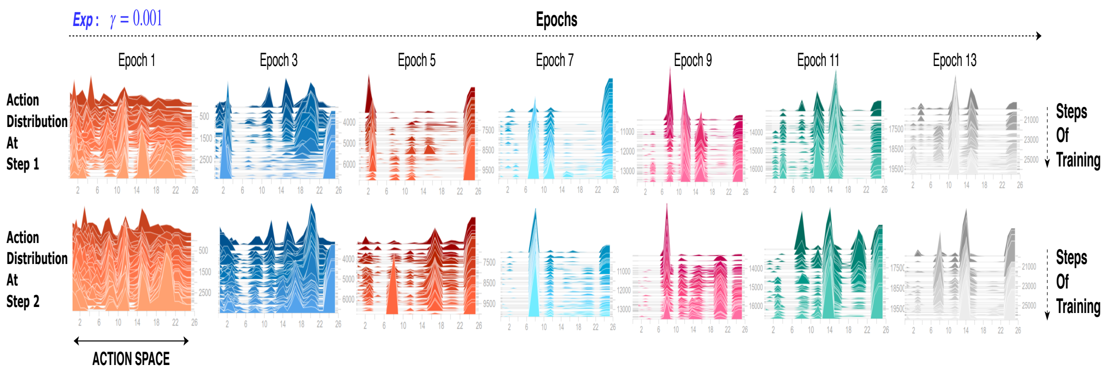


Figure 5.5: Action distributions obtained in the training of *SkipFrame* at first and second step of the episode, with a degree of exploration $\gamma = 0.001$.
Note: at each step of training, the action distribution shown is averaged over the current batch being processed.

in the initial half portion of video is relatively larger as compared to that of the other half. We can thus say that the agent learns to identify the noisy frames in the initial portion of the video and hence resorts to selecting large skip-intervals in the initial portion of the video.

5.5.3 Action Distributions

We try to pictorially visualize the training of *SkipFrame* in Figure 5.5. In particular, we show the transition in the histograms of action distributions (at first and second step of the sequence) with each epoch of training. As can be observed, initially, the agent picks the skip-interval almost randomly from the action space. However, with training, the agent's action distribution concentrates on one or two skip-intervals, which results in a better classification score. Moreover, increasing exploration (γ) beyond a point as shown in Figure 5.6 results in the agent being unable to focus on a few actions and instead, converges to a much more uniform distribution.

5.5.4 Label Distribution

We also tried to look at the temporal distribution of labels assigned to videos in the YouTube-8M dataset. We observe that the multiple labels assigned to a video are generally spread across its entire length, as shown in a sample video in Figure 5.7. Thus, it becomes necessary for the agent to observe until the end of a video to correctly classify multiple labels while processing only the relevant frames in the video. We hypothesise that given the nature of the dataset, selecting uniformly spaced frames is perhaps the

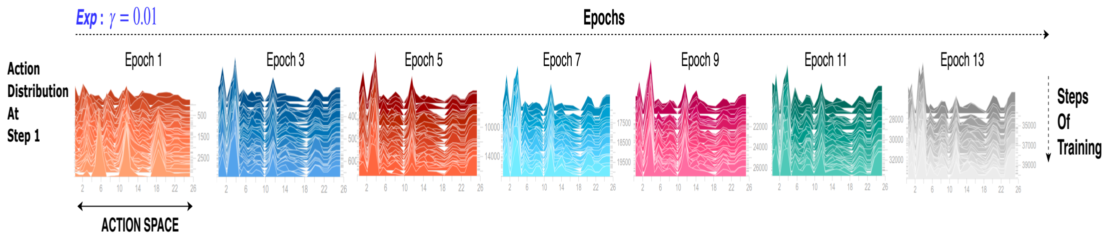


Figure 5.6: Action distribution obtained in the training of *SkipFrame* at the first step of the episode, with a degree of exploration $\gamma = 0.01$.

best strategy to cover all the categories in the video.

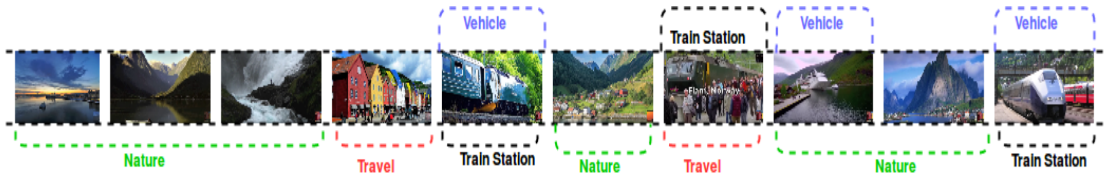


Figure 5.7: Sketch of a sample video from YouTube-8M dataset (Abu-El-Haija *et al.*, 2016) with true labels: *Travel*, *Nature*, *Train Station*, *Vehicle*

5.5.5 Computation Cost

We further compare the computational costs of *SkipFrame*, *skyline* and the Uniform baseline models in terms of the number of floating point operations (FLOPs). In comparison to the *skyline*, Uniform is able to lower the number of FLOPs by around 90%. One interesting observation is that although the *SkipFrame* model is able to outperform Uniform by a margin of 0.7% as seen in Figure 5.4, this improvement comes at the cost of a relatively more number of FLOPs (see Table 5.2). However, *SkipFrame* eventually converges to selecting frames in the neighborhood of uniformly spaced frames only as discussed earlier.

Some of the primary findings of the above analysis are: (i) the multiple labels assigned to a video, in general correspond to different portions spanning across the length of the video, (ii) *SkipFrame* model eventually selects the frames located around uniformly spaced frames, and (iii) it incurs additional FLOPs as compared to the Uniform baseline. This leads us to conclude that Uniform is an easier to implement, efficient and well-performing strategy for fewer frame selection on the video classification task.

Model	#Frames	#FLOPs
skyline	230	5.058 B
Uniform	10	0.167 B
SkipFrame	10	0.167 B + 81.92 K

Table 5.2: Comparison of FLOPs (computation cost) of different models. Here, B: Billion and K: Thousand are the order of #FLOPs

5.6 Summary

We introduce an RL framework *SkipFrame* which learns to skip the future irrelevant frames based on the contextual information about the video frames seen so far. We propose two different designs of rewards which can be used to train this RL agent: (i) DELAY-REWARD, (ii) IMM-REWARD. Further, we compare the different reward signals obtained by the agent from the *skyline* model in the form of: (i) negative classification loss, (ii) GAP score (an evaluation metric). We empirically establish the superiority of 'DELAY-REWARD' design with GAP reward signal over the other alternatives. We also study the effect of exploration on the selection of frame sequences picked by the *SkipFrame* model and establish that increasing it beyond a limit results in performance degradation due to the network not being able to focus on the optimal actions. We explore the temporal distribution of multiple video labels in the dataset and observe that these labels are generally spread across the length of the video. We conclude this chapter with a primary finding that Uniform is an easier and efficient strategy for finding relevant frames for the classification task, undermining the need for the *SkipFrame* model.

CHAPTER 6

Conclusions and Future Work

In this chapter, we summarize the work done in this thesis and elaborate on the final conclusions, listing the possible future directions and the primary outcomes of our research work.

6.1 Conclusions

In this thesis, we addressed the problem of efficient video classification, with a specific focus on building compute-efficient models utilizing fewer frames. We commenced the investigation with the exploration of state-of-the-art methods for video classification, which primarily included recurrent neural network and cluster-and-aggregate based models. One of the key drawbacks of these models was their high compute and memory requirements due to the processing of every frame in the video. Previous works in the literature have mostly addressed the problem of reducing the memory requirement of the model by making it more compact with fewer parameters, thereby making the model compute-efficient as well. However, inspired by the working of a human brain, we hypothesized that not all frames are required to semantically understand the video and there is scope to lower the number of frames processed by the network. Thus, complementing previous works, we focused on lowering the number of frames processed by the model.

We leveraged the idea of distillation to build a compute-efficient classification model that utilizes only few frames to classify the video. In particular, a compute-heavy teacher trained on the entire video is used for distilling knowledge to a student network, which is trained using only a fraction of the video frames. The knowledge is encoded either in the form of *video representation* or the *class probability distribution* learned by the teacher. As a part of our empirical study, we established that picking uniformly spaced frames is a strong baseline, and clearly, a potential candidate for the student network. We performed an extensive experimental analysis on YouTube-8M benchmark dataset of multi-label classification and demonstrated that the proposed student outperforms the baselines. In comparison to the teacher network, we empirically showed that the student is able to lower the number of FLOPs by 90%, with a negligible drop in GAP performance.

As a part of the above analysis, we focused on the two most popular paradigms of video classification: (i) recurrent neural network and (ii) cluster-and-aggregate based methods. Under cluster-and-aggregate based methods, we experimented with the state-of-the-art model , *NetVLAD*, and its memory-efficient version *i.e.*, *NeXtVLAD*. The results obtained were similar for all the above-stated models, which empirically shows that the *Teacher-Student* framework can be easily integrated with memory-efficient models as well and thus our work complements the existing research on memory-efficient models. The results presented in chapter 4 have been published in Bhardwaj and Khapra (2018) and Bhardwaj *et al.* (2019).

Further, in Chapter 5, we tackled the problem of dynamically selecting relevant frames from a video. We introduced an RL agent *SkipFrame*, which skips the unimportant frames by looking at the history of frames seen so far, wherein the agent is trained by Policy Gradient methods using the classification score (reward signal) obtained by feeding the selected frames through a classification network. We experimented with various performance metrics: standard classification loss and GAP score, as the reward signal, along with multiple reward-designs - one obtained at every step by feeding the frames selected so far to a classifier and the other obtained only after processing the entire video. Empirically, we showed that the latter reward-design combined with the GAP reward was able to outperform the Uniform baseline. Based on further experimental analysis, we observed that the well-performing models were generally biased towards selecting uniformly spaced frames spanning the entire video.One reason for this could be that the multiple labels associated with a video generally correspond to different portions of the video. Although *SkipFrame* was able to beat the Uniform baseline in terms of classification score, eventually the selected frames were more or less located in the neighborhood of uniformly spaced frames. Based on the above two observations and combined with the analysis that *SkipFrame* incurred slightly more FLOPs as compared to the Uniform baseline, we concluded that Uniform is an easier to implement strategy for dynamic frame selection, with a significant frame reduction and good performance.

REFERENCES

1. **Abu-El-Haija, S.**, **N. Kothari**, **J. Lee**, **A. P. Natsev**, **G. Toderici**, **B. Varadarajan**, and **S. Vijayanarasimhan**, Youtube-8m: A large-scale video classification benchmark. *In arXiv:1609.08675*. 2016.

2. **Arandjelovic, R.**, **P. Gronat**, **A. Torii**, **T. Pajdla**, and **J. Sivic**, Netvlad: Cnn architecture for weakly supervised place recognition. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 5297–5307. 2016.

3. **Ba, J.** and **R. Caruana**, Do deep nets really need to be deep? *In* **Z. Ghahramani**, **M. Welling**, **C. Cortes**, **N. D. Lawrence**, and **K. Q. Weinberger** (eds.), *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., 2014*a*, 2654–2662.

4. **Ba, J.** and **R. Caruana**, Do deep nets really need to be deep? *In Advances in Neural Information Processing Systems 27*, 2654–2662. 2014*b*.

5. **Ba, J.**, **V. Mnih**, and **K. Kavukcuoglu** (2014). Multiple object recognition with visual attention. *arXiv preprint arXiv:1412.7755*.

6. **Bahdanau Dzmitry, B. Y.**, **Cho Kyunghyun**, Neural machine translation by jointly learning to align and translate. *In International Conference on Learning Representations,ICLR*. 2015.

7. **Bengio, Y.**, **P. Simard**, **P. Frasconi**, *et al.* (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, **5**(2), 157–166.

8. **Bhardwaj, S.** and **M. M. Khapra**, I have seen enough: A teacher student network for video classification using fewer frames. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 1138–1142. 2018.

9. **Bhardwaj, S.**, **M. Srinivasan**, and **M. M. Khapra**, Efficient video classification using fewer frames. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 354–363. 2019.

10. **Carreira, J.** and **A. Zisserman**, Quo vadis, action recognition? a new model and the kinetics dataset. *In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

11. **Chen, G.**, **W. Choi**, **X. Yu**, **T. Han**, and **M. Chandraker**, Learning efficient object detection models with knowledge distillation. *In* **I. Guyon**, **U. V. Luxburg**, **S. Bengio**, **H. Wallach**, **R. Fergus**, **S. Vishwanathan**, and **R. Garnett** (eds.), *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017*a*, 742–751.

12. **Chen, S.**, **X. Wang**, **Y. Tang**, **X. Chen**, **Z. Wu**, and **Y. Jiang** (2017*b*). Aggregating frame-level features for large-scale video classification. *CoRR*, **abs/1707.00803**.

13. **Chen, Y.**, **S. Wang**, **W. Zhang**, and **Q. Huang**, Less is more: Picking informative frames for video captioning. *In The European Conference on Computer Vision (ECCV)*. 2018.

14. **Cheng, Y.**, **D. Wang**, **P. Zhou**, and **T. Zhang** (2017). A survey of model compression and acceleration for deep neural networks. *CoRR*, **abs/1710.09282**.

15. **Cho, K.**, **B. van Merrienboer**, **D. Bahdanau**, and **Y. Bengio** (2014). On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, **abs/1409.1259**.

16. **Choi, H.-D. L.** and **B.-T. Zhang** (). Temporal attention mechanism with conditional inference for large-scale multi-label video classification.

17. **Chung, K. L.**, *Markov chains*. Springer, 1967.

18. **Courbariaux, M.**, **I. Hubara**, **D. Soudry**, **R. El-Yaniv**, and **Y. Bengio** (2016). Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*.

19. **Cromwell, H. C.**, **R. P. Mears**, **L. Wan**, and **N. N. Boutros** (2008). Sensory gating: a translational effort from basic to clinical science. *Clinical EEG and Neuroscience*, **39**(2), 69–72.

20. **Dalal, N.** and **B. Triggs**, Histograms of oriented gradients for human detection. *In Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, 886–893. IEEE, 2005.

21. **Deng, J.**, **W. Dong**, **R. Socher**, **L.-J. Li**, **K. Li**, and **L. Fei-Fei**, Imagenet: A large-scale hierarchical image database. *In Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 248–255. IEEE, 2009.

22. **Denil, M.**, **B. Shakibi**, **L. Dinh**, **N. de Freitas**, *et al.*, Predicting parameters in deep learning. *In Advances in Neural Information Processing Systems*, 2148–2156. 2013.

23. **Donahue, J.**, **L. A. Hendricks**, **S. Guadarrama**, **M. Rohrbach**, **S. Venugopalan**, **K. Saenko**, and **T. Darrell**, Long-term recurrent convolutional networks for visual recognition and description. *In CVPR*. 2015.

24. **Dreiseitl, S.** and **L. Ohno-Machado** (2002). Logistic regression and artificial neural network classification models: a methodology review. *Journal of biomedical informatics*, **35**(5-6), 352–359.

25. **Endisch, C.**, **C. Hackl**, and **D. Schröder**, Optimal brain surgeon for general dynamic neural networks. *In Progress in Artificial Intelligence: 13th Portuguese Conference on Aritficial Intelligence, EPIA 2007*, 15–28. Springer Berlin Heidelberg, 2007.

26. **Fan, H.**, **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 International Joint Conference on Artificial Intelligence, IJCAI 2018, Stockholm, Sweden, July 13-19, 2018*, 705–711. 2018.

27. **Fleet, D.** and **Y. Weiss**, Optical flow estimation. *In Handbook of mathematical models in computer vision*. Springer, 2006, 237–257.

28. **Garcia-Garcia, A.**, **F. Gomez-Donoso**, **J. Garcia-Rodriguez**, **S. Orts-Escolano**, **M. Cazorla**, and **J. Azorin-Lopez**, Pointnet: A 3d convolutional neural network for real-time object class recognition. *In 2016 International Joint Conference on Neural Networks (IJCNN)*, 1578–1584. IEEE, 2016.

29. **Girdhar, R.**, **D. Ramanan**, **A. Gupta**, **J. Sivic**, and **B. Russell**, Actionvlad: Learning spatio-temporal aggregation for action classification. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 971–980. 2017.

30. **Girshick, R.**, Fast R-CNN. *In Proceedings of the International Conference on Computer Vision (ICCV)*. 2015.

31. **Glorot, X.** and **Y. Bengio**, Understanding the difficulty of training deep feedforward neural networks. *In Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 249–256. 2010*a*.

32. **Glorot, X.** and **Y. Bengio**, Understanding the difficulty of training deep feedforward neural networks. *In Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 249–256. 2010*b*.

33. **Gong, B.**, **W.-L. Chao**, **K. Grauman**, and **F. Sha**, Diverse sequential subset selection for supervised video summarization. *In* **Z. Ghahramani**, **M. Welling**, **C. Cortes**, **N. D. Lawrence**, and **K. Q. Weinberger** (eds.), *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., 2014, 2069–2077.

34. **Han, S.**, **H. Mao**, and **W. J. Dally** (2015*a*). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

35. **Han, S.**, **J. Pool**, **J. Tran**, and **W. Dally**, Learning both weights and connections for efficient neural network. *In* **C. Cortes**, **N. D. Lawrence**, **D. D. Lee**, **M. Sugiyama**, and **R. Garnett** (eds.), *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015*b*, 1135–1143.

36. **He, K.**, **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*, 770–778. 2016.

37. **He, Y.**, **X. Zhang**, and **J. Sun**, Channel pruning for accelerating very deep neural networks. *In The IEEE International Conference on Computer Vision (ICCV)*. 2017.

38. **Hinton, G.**, **O. Vinyals**, and **J. Dean**, Distilling the knowledge in a neural network. *In NIPS Deep Learning and Representation Learning Workshop*. 2015. URL `http://arxiv.org/abs/1503.02531`.

39. **Hochreiter, S.** and **J. Schmidhuber** (1997). Long short-term memory. *Neural computation*, **9**(8), 1735–1780.

40. **Hornik, K.**, **M. Stinchcombe**, and **H. White** (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, **2**(5), 359–366.

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

42. **Hu, M.**, **Y. Peng**, **F. Wei**, **Z. Huang**, **D. Li**, **N. Yang**, and **M. Zhou** (2018). Attention-guided answer distillation for machine reading comprehension. *CoRR*, **abs/1808.07644**.

43. **Husain, F.**, **B. Dellen**, and **C. Torras**, Scene understanding using deep learning. *In Handbook of Neural Computation*. Elsevier, 2017, 373–382.

44. **Iandola, F. N.**, **S. Han**, **M. W. Moskewicz**, **K. Ashraf**, **W. J. Dally**, and **K. Keutzer** (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5mb model size. *arXiv preprint arXiv:1602.07360*.

45. **Jaderberg, M.**, **A. Vedaldi**, and **A. Zisserman** (2014). Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*.

46. **Jang, Y.**, **Y. Song**, **Y. Yu**, **Y. Kim**, and **G. Kim**, Tgif-qa: Toward spatio-temporal reasoning in visual question answering. *In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

47. **Ji, S.**, **W. Xu**, **M. Yang**, and **K. Yu** (2013). 3d convolutional neural networks for human action recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, **35**(1), 221–231. ISSN 0162-8828.

48. **Jian-Hao Luo, J. W.** and **W. Lin**, ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. *In International Conference on Computer Vision (ICCV)*. 2017.

49. **Joonseok, L.**, **N. A. (Paul)**, **ReadeWalter**, **S. Rahul**, and **T. George** (2018). The 2nd youtube-8m large-scale video understanding challenge. *In: Proc. of the 2nd Workshop on YouTube-8M Large-Scale Video Understanding (ECCV 2018)*. URL `https://static.googleusercontent.com/media/research.google.com/en//youtube8m/workshop2018/c_01.pdf`.

50. **Jordan, M. I.**, Serial order: A parallel distributed processing approach. *In Advances in psychology*, volume 121. Elsevier, 1997, 471–495.

51. **Karpathy, A.**, **G. Toderici**, **S. Shetty**, **T. Leung**, **R. Sukthankar**, and **L. Fei-Fei**, Large-scale video classification with convolutional neural networks. *In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, 1725–1732. IEEE Computer Society, Washington, DC, USA, 2014. ISBN 978-1-4799-5118-5.

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

53. **Kingma, D. P.** and **J. Ba**, Adam: A method for stochastic optimization. *In International Conference on Learning Representations,ICLR*. 2015.

54. **Kmiec, S.**, **J. Bae**, and **R. An** (2018). Learnable pooling methods for video classification. *arXiv preprint arXiv:1810.00530*.

55. **Krizhevsky, A.**, **I. Sutskever**, and **G. E. Hinton**, Imagenet classification with deep convolutional neural networks. *In* **F. Pereira**, **C. J. C. Burges**, **L. Bottou**, and **K. Q. Weinberger** (eds.), *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, 1097–1105.

56. **Kuehne, H.**, **H. Jhuang**, **E. Garrote**, **T. Poggio**, and **T. Serre**, Hmdb: A large video database for human motion recognition. *In 2011 International Conference on Computer Vision*, 2556–2563. 2011. ISSN 2380-7504.

57. **Lan Xu, G. S., Zhu Xiatian** (2018). Knowledge distillation by on-the-fly native ensemble. *To Appear In Proceedings of Thirty-second Annual Conference on Neural Information Processing Systems NIPS*.

58. **LeCun, Y.**, **J. S. Denker**, and **S. A. Solla**, Optimal brain damage. *In* **D. S. Touretzky** (ed.), *Advances in Neural Information Processing Systems 2*. Morgan-Kaufmann, 1990, 598–605.

59. **Lev, G.**, **G. Sadeh**, **B. Klein**, and **L. Wolf**, Rnn fisher vectors for action recognition and image annotation. *In European Conference on Computer Vision*, 833–850. Springer, 2016.

60. **Li, F.**, **C. Gan**, **X. Liu**, **Y. Bian**, **X. Long**, **Y. Li**, **Z. Li**, **J. Zhou**, and **S. Wen** (2017). Temporal modeling approaches for large-scale youtube-8m video understanding. *CoRR*, **abs/1707.04555**.

61. **Li, H.**, **A. Kadav**, **I. Durdanovic**, **H. Samet**, and **H. P. Graf** (2016). Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*.

62. **Liu Tianqi, L. B.** (2018). Constrained-size tensorflow models for youtube-8m video understanding challenge. *In: Proc. of the 2nd Workshop on YouTube-8M Large-Scale Video Understanding (2018)*. URL https://research.google.com/youtube8m/workshop2018/c_06.pdf.

63. **Lopez-Paz, D.**, **B. Schölkopf**, **L. Bottou**, and **V. Vapnik**, Unifying distillation and privileged information. *In International Conference on Learning Representations*. 2016.

64. **Lowe, D. G.** (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, **60**(2), 91–110.

65. **Maas, A. L.**, **A. Y. Hannun**, and **A. Y. Ng**, Rectifier nonlinearities improve neural network acoustic models. *In Proc. icml*, volume 30, 3. 2013.

66. **Mathe, S.**, **A. Pirinen**, and **C. Sminchisescu**, Reinforcement learning for visual object detection. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2894–2902. 2016.

67. **McClelland, J. L.**, **D. E. Rumelhart**, **P. R. Group**, *et al.* (1986). Parallel distributed processing. *Explorations in the Microstructure of Cognition*, **2**, 216–271.

68. **McCulloch, W. S.** and **W. Pitts** (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, **5**(4), 115–133.

69. **Miech, A.**, **I. Laptev**, and **J. Sivic** (2017). Learnable pooling with context gating for video classification. *CoRR*, **abs/1706.06905**.

70. **Miha, S.** and **A. David** (2018). Building a size constrained predictive model for video classification. *In: Proc. of the 2nd Workshop on YouTube-8M Large-Scale Video Understanding (2018)*. URL https://static.googleusercontent.com/media/research.google.com/en//youtube8m/workshop2018/c_14.pdf.

71. **Miller, D. J.** and **H. S. Uyar**, A mixture of experts classifier with learning based on both labelled and unlabelled data. *In Advances in neural information processing systems*, 571–577. 1997.

72. **Mnih, V.**, **A. P. Badia**, **M. Mirza**, **A. Graves**, **T. Lillicrap**, **T. Harley**, **D. Silver**, and **K. Kavukcuoglu**, Asynchronous methods for deep reinforcement learning. *In International conference on machine learning*, 1928–1937. 2016.

73. **Mnih, V.**, **N. Heess**, **A. Graves**, *et al.*, Recurrent models of visual attention. *In Advances in neural information processing systems*, 2204–2212. 2014.

74. **Molchanov, P.**, **S. Tyree**, **T. Karras**, **T. Aila**, and **J. Kautz** (2016). Pruning convolutional neural networks for resource efficient transfer learning. *arXiv preprint arXiv:1611.06440*.

75. **Nair, V.** and **G. E. Hinton**, Rectified linear units improve restricted boltzmann machines. *In Proceedings of the 27th international conference on machine learning (ICML-10)*, 807–814. 2010.

76. **Narang, S.**, **G. F. Diamos**, **S. Sengupta**, and **E. Elsen** (2017). Exploring sparsity in recurrent neural networks.

77. **Pan, P.**, **Z. Xu**, **Y. Yang**, **F. Wu**, and **Y. Zhuang**, Hierarchical recurrent neural encoder for video representation with application to captioning. *In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1029–1038. 2016. ISSN 1063-6919.

78. **Pavel, O.**, **L. Elizaveta**, **S. Roman**, **A. Vladimir**, **S. Gleb**, **K. Oleg**, and **I. N. Sergey** (2018). Label denoising with large ensembles of heterogeneous neural networks. *In: Proc. of the 2nd Workshop on YouTube-8M Large-Scale Video Understanding (2018)*. URL https://research.google.com/youtube8m/workshop2018/c_07.pdf.

79. **Peng, X.**, **C. Zou**, **Y. Qiao**, and **Q. Peng**, Action recognition with stacked fisher vectors. *In European Conference on Computer Vision*, 581–595. Springer, 2014.

80. **Puterman, M. L.**, *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

81. **Rastegari, M.**, **V. Ordonez**, **J. Redmon**, and **A. Farhadi**, Xnor-net: Imagenet classification using binary convolutional neural networks. *In European Conference on Computer Vision*, 525–542. Springer, 2016.

82. **Ren, S.**, **K. He**, **R. Girshick**, and **J. Sun**, Faster R-CNN: Towards real-time object detection with region proposal networks. *In Advances in Neural Information Processing Systems (NIPS)*. 2015.

83. **Romero, A.**, **N. Ballas**, **S. E. Kahou**, **A. Chassang**, **C. Gatta**, and **Y. Bengio**, Fitnets: Hints for thin deep nets. *In In Proceedings of ICLR*. 2015.

84. **Rongcheng, L.**, **X. Jing**, and **J. Fan** (2018). Nextvlad: An efficient neural network to aggregate frame-level features for large-scale video classification. *In: Proc. of the 2nd Workshop on YouTube-8M Large-Scale Video Understanding (2018)*. URL https://research.google.com/youtube8m/workshop2018/c_03.pdf.

85. **Rosenblatt, F.** (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, **65**(6), 386.

86. **Santos, E.** and **C. A. Noggle**, *Synaptic Pruning*. Springer US, Boston, MA, 2011. ISBN 978-0-387-79061-9, 1464–1465. URL https://doi.org/10.1007/978-0-387-79061-9_2856.

87. **Schindler, K.** and **L. J. Van Gool**, Action snippets: How many frames does human action recognition require? *In CVPR*, volume 1, 3–2. 2008.

88. **Shivam, G.** (2018). Learning video features for multi-label classification. *In: Proc. of the 2nd Workshop on YouTube-8M Large-Scale Video Understanding (2018)*. URL https://research.google.com/youtube8m/workshop2018/c_17.pdf.

89. **Shou, Z.**, **D. Wang**, and **S.-F. Chang**, Temporal action localization in untrimmed videos via multi-stage cnns. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1049–1058. 2016.

90. **Simonyan, K.** and **A. Zisserman**, Two-stream convolutional networks for action recognition in videos. *In* **Z. Ghahramani**, **M. Welling**, **C. Cortes**, **N. D. Lawrence**, and **K. Q. Weinberger** (eds.), *Advances in Neural Information Processing Systems 27*, 568–576. Curran Associates, Inc., 2014*a*.

91. **Simonyan, K.** and **A. Zisserman** (2014*b*). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

92. **Sivic, J.** and **A. Zisserman**, Video google: A text retrieval approach to object matching in videos. *In Proceedings of the International Conference on Computer Vision (ICCV)*, 1470. 2015.

93. **Skalic, M.**, **M. Pekalski**, and **X. E. Pan** (2017). Deep learning methods for efficient large scale video labeling. *CoRR*, **abs/1706.04572**.

94. **Song, X.**, **K. Chen**, **J. Lei**, **L. Sun**, **Z. Wang**, **L. Xie**, and **M. Song**, Category driven deep recurrent neural network for video summarization. *In 2016 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, 1–6. IEEE, 2016.

95. **Soomro, K.**, **A. R. Zamir**, and **M. Shah** (2012). Ucf101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, **abs/1212.0402**.

96. **Sutton, R. S.** and **A. G. Barto**, *Reinforcement learning: An introduction*. MIT press, 2018.

97. **Sutton, R. S.**, **D. A. McAllester**, **S. P. Singh**, and **Y. Mansour**, Policy gradient methods for reinforcement learning with function approximation. *In Advances in neural information processing systems*, 1057–1063. 2000.

98. **Szegedy, C.**, **W. Liu**, **Y. Jia**, **P. Sermanet**, **S. Reed**, **D. Anguelov**, **D. Erhan**, **V. Vanhoucke**, and **A. Rabinovich**, Going deeper with convolutions. *In Computer Vision and Pattern Recognition (CVPR)*. 2015. URL http://arxiv.org/abs/1409.4842.

99. **Tang, Y.**, **X. Zhang**, **J. Wang**, **S. Chen**, **L. Ma**, and **Y.-G. Jiang** (2018). Non-local netvlad encoding for video classification. *arXiv preprint arXiv:1810.00207*.

100. **Tran, D.**, **L. Bourdev**, **R. Fergus**, **L. Torresani**, and **M. Paluri**, Learning spatiotemporal features with 3d convolutional networks. *In Proceedings of the IEEE international conference on computer vision*, 4489–4497. 2015.

101. **Wang, H.**, **I. Katsavounidis**, **J. Zhou**, **J. Park**, **S. Lei**, **X. Zhou**, **M.-O. Pun**, **X. Jin**, **R. Wang**, **X. Wang**, **Y. Zhang**, **J. Huang**, **S. Kwong**, and **C.-C. J. Kuo** (2017*a*). Videoset: A large-scale compressed video quality dataset based on jnd measurement. *Journal of Visual Communication and Image Representation*, **46**.

102. **Wang, H.**, **T. Zhang**, and **J. Wu** (2017*b*). The monkeytyping solution to the youtube-8m video understanding challenge. *CoRR*, **abs/1706.05150**.

103. **Wen, W.**, **C. Wu**, **Y. Wang**, **Y. Chen**, and **H. Li**, Learning structured sparsity in deep neural networks. *In Advances in Neural Information Processing Systems*, 2074–2082. 2016.

104. **Williams, R. J.** (1989). Gradient-based learning algorithms for recurrent networks. *Back-propagation: Theory, Architectures and Applications*.

105. **Williams, R. J.** (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, **8**(3-4), 229–256.

106. **Wong, J. H. M.** and **M. J. F. Gales**, Sequence student-teacher training of deep neural networks. *In Interspeech 2016, 17th Annual Conference of the International Speech Communication Association, San Francisco, CA, USA, September 8-12, 2016*, 2761–2765. 2016.

107. **Wu, Z.**, **T. Yao**, **Y. Fu**, and **Y. Jiang** (2016). Deep learning for video classification and captioning. *CoRR*, **abs/1609.06782**.

108. **Xiao, J.**, **K. A. Ehinger**, **J. Hays**, **A. Torralba**, and **A. Oliva** (2016). Sun database: Exploring a large collection of scene categories. *Int. J. Comput. Vision*, **119**(1), 3–22. ISSN 0920-5691.

109. **Xu, H.**, **A. Das**, and **K. Saenko**, R-c3d: Region convolutional 3d network for temporal activity detection. *In Proceedings of the IEEE international conference on computer vision*, 5783–5792. 2017.

110. **Xu, K.**, **J. Ba**, **R. Kiros**, **K. Cho**, **A. Courville**, **R. Salakhudinov**, **R. Zemel**, and **Y. Bengio**, Show, attend and tell: Neural image caption generation with visual attention. *In International conference on machine learning*, 2048–2057. 2015*a*.

111. **Xu, Z.**, **Y. Yang**, and **A. G. Hauptmann**, A discriminative cnn video representation for event detection. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1798–1807. 2015*b*.

112. **Yeung, S.**, **O. Russakovsky**, **G. Mori**, and **L. Fei-Fei**, End-to-end learning of action detection from frame glimpses in videos. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2678–2687. 2016.

113. **Yue-Hei Ng, J.**, **M. Hausknecht**, **S. Vijayanarasimhan**, **O. Vinyals**, **R. Monga**, and **G. Toderici**, Beyond short snippets: Deep networks for video classification. *In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

114. **Zeiler, M. D.** (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

115. **Zhang, B.**, **L. Wang**, **Z. Wang**, **Y. Qiao**, and **H. Wang**, Real-time action recognition with enhanced motion vector cnns. *In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016*a*.

116. **Zhang, K.**, **W.-L. Chao**, **F. Sha**, and **K. Grauman**, Video summarization with long short-term memory. *In ECCV*. Springer, 2016*b*.

117. **Zhang, X.**, **J. Zou**, **K. He**, and **J. Sun** (2016*c*). Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence*, **38**(10), 1943–1955.

118. **Zhang, X.**, **J. Zou**, **X. Ming**, **K. He**, and **J. Sun**, Efficient and accurate approximations of nonlinear convolutional networks. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1984–1992. 2015.

119. **Zhou, A.**, **A. Yao**, **Y. Guo**, **L. Xu**, and **Y. Chen** (2017). Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*.

120. **Zhou, K.**, **Y. Qiao**, and **T. Xiang**, Deep reinforcement learning for unsupervised video summarization with diversity-representativeness reward. *In Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

121. **Zhu, Y.**, **Z. Lan**, **S. Newsam**, and **A. G. Hauptmann** (2017). Hidden two-stream convolutional networks for action recognition. *arXiv preprint arXiv:1704.00389*.

LIST OF PAPERS BASED ON THESIS

1. **Bhardwaj S.**, Srinivasan M. and Khapra M.M., Efficient Video Classification Using Fewer Frames *In Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition in Long Beach California, USA in June 2019.*, CVPR'19.

2. **Bhardwaj S.** and Khapra M.M., *I have seen enough*: A Teacher Student Network for Video Classification Using Fewer Frames *In Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshop on Brave New Ideas for Video Understanding in Salt Lake City, Utah in June 2018.*, CVPR Workshop 2018.