

https://dspace.org/

SummaryNet: two-stream convolutional networks for automatic video summarisation

Jappie, Ziyad

2020

https://hdl.handle.net/10539/30207

Jappie, Ziyad (2020) SummaryNet: two-stream convolutional networks for automatic video summarisation, University of the Witwatersrand, Johannesburg, https://hdl.handle.net/10539/30207

Downloaded from a DSpace repository. https://dspace.org/

SUMMARYNET: TWO STREAM CONVOLUTIONAL NETWORKS FOR AUTOMATIC VIDEO SUMMARISATION

SCHOOL OF COMPUTER SCIENCE AND APPLIED MATHEMATICS
UNIVERSITY OF THE WITWATERSRAND

ZIYAD JAPPIE
557803

SUPERVISED BY
PROF. TURGAY CELIK

05 FEBRUARY 2020



A dissertation submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science.

Abstract

Video summarisation is the task of automatically summarising a video sequence, to extract "important" parts of the video so as to give an overview of what has occurred. The benefit of solving this problem is that it can be applied to a myriad of fields such as the entertainment industry, sports, e-learning and many more. There is a distinct inherent difficulty with video summarisation due to its subjectivity - there is no one defined correct answer. As such, it is particularly difficult to define and measure tangible performance. This is in addition to the other difficulties associated with general video processing. We present a novel two-stream network framework for automatic video summarisation, which we call SummaryNet. The SummaryNet employs a deep two-stream network to model pertinent spatio-temporal features by leveraging RGB as well as optical flow information. We use the Two-Stream Inflated 3D ConvNet (I3D) network to extract high-level, semantic feature representations as inputs to our SummaryNet model. Experimental results on common benchmark datasets show that the considered method achieves comparable or better results than the state-of-the-art video summarisation methods.

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the Degree of Master of Science at the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination at any other University.

Signature:

Ziyad Jappie

February 5, 2020

Acknowledgements

The author would like to acknowledge Turgay Celik, professor at the University of the Witwatersrand, Johannesburg, for his assistance in the research idea and in aiding this proposed project. The author would also like to thank David Torpey for his unwavering assistance throughout the development of this project.

Contents

Abstract i

Declaration i

Acknowledgements ii

List of Figures vi

List of Tables ix

1 Introduction 1

2 Background and Related work 5

2.1 Background: Neural Networks 5

2.1.1 Artificial Neural Networks 5

2.1.2 Convolutional Neural Networks 11

2.1.3 Recurrent Neural Networks 12

2.1.4 Auto-Encoders 15

2.2 Background: Algorithms Adopted In Video Data 17

2.2.1 Transfer Learning 17

2.2.2 Optical Flow 17

2.2.3 Kernel Temporal Segmentation (KTS) 19

2.2.4 Knapsack Algorithm 20

2.2.5 Determinant point processing 22

2.2.6 K-means Clustering 23

2.3 Background: Additional Concepts 24

2.3.1 Video Sequence 24

2.3.2 Hand crafted global descriptors 25

2.3.3 Conclusion 25

3 Literature Review **26**

3.1 Previous Approaches . . . 27

3.1.1 Non-Deep Learning Approaches . . . 27

3.1.2 Deep Learning Approaches . . . 28

3.2 Transfer Learning Models/Architectures . . . 35

3.2.1 Transfer Learning Used Previously . . . 39

3.3 Conclusion . . . 40

4 Research Method **41**

4.1 Introduction . . . 41

4.2 Research Motivation . . . 41

4.3 Research Hypothesis . . . 41

4.4 Research Method . . . 42

4.4.1 Hardware and Software . . . 42

4.4.2 Proposed Model . . . 42

4.4.3 Datasets . . . 43

4.4.4 Pre-processing . . . 47

4.4.5 Pre-trained Models/Features . . . 48

4.4.6 SummaryNet Model . . . 50

4.4.7 Training . . . 50

4.4.8 Testing and Evaluation Metrics . . . 52

4.4.9 Hyper-parameter Optimisation . . . 54

4.4.10 Baseline Model . . . 54

4.5 Conclusion . . . 54

5 Experimental Results **56**

5.1 Introduction . . . 56

5.2 Previous Results . . . 57

5.3 Hyper-parameters . . . 57

5.4 Experimental Results . . . 58

5.4.1 TvSum Dataset . . . 58

5.4.2 SumMe Dataset . . . 63

5.5 Comparing Results . . . 68

5.6 Computational Runtime . . . 69

5.7 Conclusion . . . 71

6 Conclusion and Future Work **72**

Bibliography **74**

Appendix **79**

List of Figures

2.1 A visual example of a perceptron [39]. We see that the inputs x is multiplied by its corresponding weight value. The results are summed up and is pushed through an activation function to construct an output. . . . 8

2.2 Typical CNN architecture as shown by [30] which demonstrated the power of CNNs. Known as the Le-Net network, it consists of 7 layers and takes an input image of 32×32 pixels. This networks is made up of 3 convolutional layers, 2 fully connected layers and 2 sampling layers. The final layer consists of 10 neurons representing the 10 numerical digits. . . . 12

2.3 A basic RNN architecture [16]. By unfolding a RNN we are able to see that outputs from previous time steps are taken as inputs to present time steps. This is an example of passing information through time. The black square represents a delay in a single time step. . . . 14

2.4 Computation of a bidirectional RNN architecture [16]. . . . 15

3.1 Model overview of attention mechanisms [24]. . . . 30

3.2 Residual blocks architecture proposed by [19]. . . . 37

3.3 A high-level overview of the I3D network which was proposed by [8]. . . . 39

4.1 The process flow of the research. There are three sections to guide this research. Section 1 involves reading the video data and processing. Section 2 is using the newly processed videos and feeding it into the models and evaluating (SummaryNet model). Section 3 deals with combining two models together to create a final result. . . . 44

4.2 Various examples of the SumMe dataset. These frames were captured manually to show the different content in each video. . . . 45

4.3 These are examples from the SumMe dataset. The summaries for each user can be seen in the top plots while the average scores for each user summary can be seen in the bottom plots. . . . 46

4.4 Example of TVSum dataset under the category animal grooming. . . . 46

4.5 The target associated with the middle frame of the current video cube represents the target for the entire cube. . . . 48

4.6 Our SummaryNet model is two-fold. Firstly, an encoder-decoder model is trained and the latent variable are extracted. Secondly, the latent variables are used as features to the second LSTM-MLP model. The second model is responsible for predicting the frame-level scores. . . . 51

5.1 Samples of bad summary results from TvSum50 dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a bad summary i.e. a low F1-score 61

5.2 Samples of good summary results from TvSum50 dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a good summary i.e. a high F1-score 62

5.3 Samples of bad summary results from SumMe dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a low F1-score where the model generated a bad summary. 66

5.4 Samples of good summary results from SumMe dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a high F1-score where the model generated a good summary. 67

6.1 Sample results from SumMe dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a good summary i.e. a high F1-score. 80

6.2 Sample results from SumMe dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a bad summary i.e. a low F1-score. 81

6.3 Sample results from SumMe dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a good summary i.e. a high F1-score. 82

6.4 Sample results from SumMe dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a average summary. . 83

6.5 Sample results from SumMe dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a good summary i.e. a high F1-score 84

6.6 Sample results from TvSum50 dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a good summary i.e. a high F1-score. 86

6.7 Sample results from TvSum50 dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a bad summary i.e. a low F1-score. 87

6.8 Sample results from TvSum50 dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a average summary. 88

6.9 Sample results from TvSum50 dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a good summary i.e. a high F1-score. 89

6.10 Sample results from TvSum50 dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a bad summary i.e. a low F1-score. 90

List of Tables

3.1 Summary of a AlexNet results in the ILSVRC-2012 36

3.2 A comparison of the top ILSVRC models [1]. 36

4.1 Descriptive Information on the Datasets. 47

5.1 Results of previous works done for the video summarisation task. State-of-the-art results are shown in bold. 56

5.2 Succinct table of hyper-parameter selection for the window size of our SummaryNet model. The values in this table correspond to the SumMe dataset. The best result is shown in bold. 58

5.3 Succinct table of hyper-parameter selection for the window size of our SummaryNet model. The values in this table correspond to the TvSum50 dataset. The best result is shown in bold. 58

5.4 List of testing videos from the TvSum50 dataset used to evaluate the systems performance. The FPS column has been rounded off to the nearest integer for brevity. 59

5.5 Results obtained for the TvSum50 dataset using our proposed method. These results clearly show the advantage of using spatio-temporal features. The best result is shown in bold. 59

5.6 Descriptive information about videos the system is tested on. The information in this table is taken from [18] 64

5.7 Results obtained for the SumMe dataset using our proposed method. These results clearly show the advantage of using spatio-temporal features. The best result is shown in bold. 64

5.8 Comparing previous results to our method. The best results are shown in dark blue and second best is shown in light blue. 68

5.9 Run-time of our models for the SumMe dataset. Here we see the number of parameters for each setting. We also see the time take for each setting to complete for training and testing 70

5.10 Run-time of our models for the TvSum50 dataset. Here we see the number of parameters for each setting. We also see the time take for each setting to complete for training and testing 70

Chapter 1

Introduction

Video summarisation has attracted more attention in recent years due to numerous applications in the entertainment industry, sports, surveillance and so forth [49; 50]. It can be argued that videos are the most important source of visual data [37]. One of the more prevalent applications is video summaries that people post on social media to share their experiences with others [49]. Most people currently summarise their videos manually as in the case of people using tools such as GoPros to record their activities, later on going on their personal devices to edit and crop out some parts of the video. One of the more prevalent applications performed by YouTube users do is to summarize sports (eg. football matches) videos. This involves only including clips when a team scores a goal or a promising opportunity is presented. Manual video summarisation can be a very laborious task which can be automated with the help of intelligent algorithms. Solving this particular problem will likely have a large impact on many fields that have some form of video records or video database. This can be applied to both audio and visual inputs for summarisation [46]. There is currently more than 300 hours of video uploaded per minute to YouTube alone [24]. If we consider CISCOs 2018 visual networking index, 80% of internet traffic will be video by 2022 [3].

Many people that record videos do so with the intention of capturing all the information at the present time and editing later. This mind-set leads to much redundant and useless information being recorded and thus more editing later on [17]. With the number of videos available online increasing daily, an effective system to manage these dumps of data has become more in demand with more attention being applied to video summarisation mechanisms [50]. Users would ideally like to browse through videos quickly to get an idea of the semantic content. This task will enable faster browsing of large video sets

as well as better grouping and access to these videos.

The task of video summarisation is challenging due to its subjective nature. This suggests the need for an effective evaluation metric in order to compare results gathered with previous works, and measure performance with. One approach to evaluate the video summarisation performance is to adopt a similar testing criteria as done by [50; 49], who presented a few baseline methods for their thumbnail/summary selections. This included Random Selection [50], Video Representative Attributes [50] and Random Sampling to name a few. Thereafter, mean Average Precision (mAP) [49] was used for evaluation. However, since we aim to compare our results against multiple previously published results and benchmarks we will be adopting the same evaluation used in [18; 52; 44] - using the F1-score by measuring the temporal overlap between the ground truth and the predicted values. By using the same evaluation metric we are able to directly compare our video summarisation system with previous methods.

This problem has attracted much attention across many different industries. However, there is still much work to be done for this problem domain, which yearns for a concrete solution that can be applied on non-trivial/unseen data.

This work is slightly inspired by the success achieved by deep models in outperforming traditional, hand-crafted methods. Traditional methods for video summarisation include low level features such as colour (Red, Green, Blue - RGB or Hue, Saturation, Value - HSV), texture, motion, etc. where the idea is to have descriptors for each frame and compare these with each other [26].

A key aspect with using hand-crafted features is that you would generally have domain knowledge on the tasks and features chosen to be used. Hand-crafted features, although very useful and effective, could restrict us to a specific problem. Conventional video summarisation methods leverage hand-crafted criteria based features in order to summarise videos in both supervised and unsupervised methodologies, [4; 11; 53; 32; 18]. Hand crafted features are commonly not well suited to transfer learning, as the features are specialised for a particular purpose.

Recent advances in deep learning show many advantages over traditional methods in various fields such as object detection, action recognition, language processing, amongst others. Thus, we opt for a deep learning approach to solve the video summarisation problem. An advantage deep learning models have is that previously-trained models can be used as a feature extractor on similar problem sets. Alternatively, you can also fine-tune these pre-trained models to suit your dataset and use these fine-tuned models instead of creating an entire model and waiting for that to train. This approach is extremely desirable if

you have limited resources available. In the realm of deep learning it is important to consider both unsupervised and supervised approaches in order to solve different aspects of this problem. One prevalent method, originally introduced as a dimensionality reduction algorithm, is the auto-encoder. In order to model sequential data effectively, one would generally use a recurrent neural network (RNN) architecture which also aims to capture long-term temporal dependencies. However, if there are spatial dependencies, which are inherent in natural images, the typical choice would be a convolutional neural network (CNN). These methods have proven to model temporal and spatial information relatively well [16]. Deep architectures include many variants as mentioned previously, where each architecture has proven to be particularly successful in certain domain problems.

Due to the inherent spatial relationships and temporal nature of video data, opting for the aforementioned deep learning architectures will enable the effective modelling of these two streams of signal. Generally, deep models require large amounts of data in order to detect patterns in the data, and generalise accordingly. With the combination of spatio-temporal features we aim detect these patterns in the data.

Although deep architectures such as RNNs and CNNs seek to learn their own set of features given raw input, another way is to feed the model hand crafted features. Feeding in hand crafted features vary on the problem description; if the problem is well defined, then certain computer vision techniques can be used to extract certain information needed to enhance the performance of the above-mentioned algorithms. Along with these features, meta-data can also be included to evaluate the performance of the algorithm as in [50], who refers to it as side information, since this additional information can be used as features as well. Examples of meta-data will include comments, video titles, video thumbnails, descriptions of the video etc.

This work will seek to solve the dynamic video summarisation problem. Concretely, this problem is defined as short video snippets being combined together to create the video summary (as opposed to static video summary, in which key images are taken from the video in order to create the summary).

There are numerous datasets that exist such as Thumb1k [31], TVSum50 [44] and SumMe [18] which can be used to evaluate video summarisation algorithms. These datasets have some additional information we might find useful, such as the meta-data discussed earlier. The datasets are also relatively new and there has been work done [50; 52; 24; 35; 54] which our results can be evaluated against.

The dissertation is structured as follows. The background and related work will be discussed in Chapters 2 and 3. Chapter 4 will then cover the research method and our proposed SummaryNet model followed

by experimental results covered in Chapter 5. Finally, in Chapter 6 we conclude this dissertation and add some final remarks.

Chapter 2

Background and Related work

This chapter provides necessary background and relevant work needed in order to become acquainted with certain concepts and methods used in this dissertation. This chapter aims to give a holistic view of the approach we pursued.

2.1 Background: Neural Networks

2.1.1 Artificial Neural Networks

The Building Blocks

Neural networks (NNs) have for many years followed the concept of mimicking the human brain. Key design principles were drawn from neuroscience, where behaviours of animals were observed by how certain neurons responded to images [16].

The layers in a NN can be grouped into 3 sections namely, the input layer(s) (first layer), hidden layer(s) and output layer(s) (final layer). Inputs to a neural network are often referred to as features. This occurs at the input layer. In the case of images, the inputs are the individual normalised pixels of each image, arranged in a raster format. Features that fed into a network can range between pixel values, hand crafted features, features learnt from another network and so on. The features learnt by a network is usually at the second last layer before the output layer(s). It is the scalar values of each neuron in that layer concatenated together to form a vector. Features are the phenomenon being observed by the pattern

recognition algorithm. A good selection of features is one of the most crucial tasks in computer vision and machine learning. We usually normalise the values of our inputs during the pre-processing step. This technique is done so that we can get all the input data to be on the same scale/ in the same range of values. This is done without distorting range values or losing information [2]. One of the simplest normalisation techniques is given in Equation (2.1) known as min-max normalisation.

$$\boldsymbol{X}_b \leftarrow \frac{\boldsymbol{X}_a - \min(\boldsymbol{X}_a)}{\max(\boldsymbol{X}_a) - \min(\boldsymbol{X}_a)} \tag{2.1}$$

where $\boldsymbol{X}_a \in \mathbb{R}^{m \times n}$ is the input matrix before normalisation and $\boldsymbol{X}_b \in \mathbb{R}^{m \times n}$ is the new input matrix after the normalisation process. In the case of RGB images we usually divide each pixel by 255 since each pixel is an integer value between $[0, 255]$.

In the hidden layer(s) it is common to include a node known as a bias. It is the node that is usually "on", i.e. it is usually non-zero. It is a scalar real number usually denoted by b. Bias has the same function as an intercept in a regression model. This terms helps a single node within a neural network have some real value output. The function of the bias is to avoid having a zero output which is vital for the network to learn. Each layer the NN has a weight matrix that is multiplied by the inputs to assign certain importance to particular neurons, this will determine how much a certain neuron will impact the summation function. A neuron simply takes one or more inputs, performs some arbitrary operations and produces an output. As the model trains, the weights and bias are updated according to some rule, usually back-propagation with the help of some loss function.

Once we have the weighted values of the inputs and bias we can feed it into a activation function. The activation function defines how the output of each node/neuron is post-processed using a function. The activation function takes the weighted sum of all the inputs, puts it through some function and subsequently produces an output. In the case of a multi-layered perceptron (discussed later), the output is then used as inputs to the next layer. The types of activation functions can vary from linear to non-linear functions. Some examples include the Rectified Linear Unit - ReLU, tanh, sigmoid, and so forth. The reason we include non-linear functions is so that we can approximate non-trivial signals/patterns within our data.

The *universal approximation theorem* [22] states that any continuous function of D-variables defined in a subset of $\mathbb{R}^D$ can be approximated by a 3-layer neural network (one hidden layer) with some activation function. In order to adhere to the *universal approximation theorem*, certain conditions needs to be

imposed on these activation functions. The activation functions must be continuous on a closed and bounded subset of $\mathbb{R}^D$ [16]. As mentioned above, there are numerous activation functions one could use. However, two functions stand out that we will be using most frequently. They are the ReLU function and sigmoid function. The ReLU function was introduced to enable better training of deep networks and is arguably the most used activation function currently in deep learning. The ReLU is defined mathematically as,

$$\text{ReLU}(x) = f(x) = \max(0, x), \tag{2.2}$$

where x is the input to a neuron. As seen in the ReLU function, only positive outputs are fed forward through the network. The sigmoid function is used for the classification layer of the network since we implicitly define only two classes. The sigmoid function is defined as,

$$\text{sigmoid}(x) = g(x) = \frac{1}{1 + e^{-x}}, \tag{2.3}$$

where x is the input to the neuron. The output $g(x)$ takes on a value between 0 and 1. In a model setting, this is to indicate which class label it predicted. Hence, the larger the linear combination, x which is fed into the function, the closer the output value will be to 1. The lower the x value, the closer the output value will be to 0.

A combination of the above-mentioned terms can be defined by a perceptron. A perceptron is an architecture that involves inputs, weights, a neuron, an activation function and some output. We can define this mathematically as:

$$\mathbf{z}_i = \boldsymbol{W}_i \mathbf{x} + \mathbf{b}_i, \tag{2.4}$$

$$\mathbf{y} = \sigma(\mathbf{z}_i), \tag{2.5}$$

where $\mathbf{b}$ is the bias term - "this is introduced to avoid always having zero outputs if inputs are zero" [16], $\mathbf{x} \in \mathbb{R}^n$ is the input features, $\mathbf{W}$ is the weighted matrix, and σ is some activation function.

A visual representation of Equation (2.4) and Equation (2.5) can be seen in Figure 2.1.

NNs can be thought of as multi-layered perceptrons. This is a combination of many perceptrons organ-

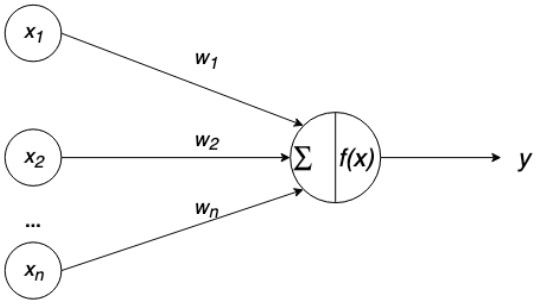


Figure 2.1: A visual example of a perceptron [39]. We see that the inputs x is multiplied by its corresponding weight value. The results are summed up and is pushed through an activation function to construct an output.

ised into multiple layers. The general flow of a NN is arranged as follows: 1) Inputs/features are fed into the network. 2) Inputs are multiplied by a weight matrix (usually randomly initialised) and a bias is added. 3) The result is fed into a activation function (usually non-linear) to get an output. 4) The output of the previous layers activation function is now the inputs to the new layer. Steps 1-4 is repeated until we reach the final layer of the network.

At the final layer we define a cost/loss function, sometimes referred to as objective function, which informs us of how much error our model has incurred. This is the point where we compare a true known output with the output generated by our network.

Some examples of loss functions are *binary cross-entropy*, *mean squared error (MSE)*, *categorical cross-entropy*, *negative log likelihood*, *hinge loss*, etc. Therefore, it is important to note the different loss functions will generate different errors for the same network output. The choice of loss function depends on the task, i.e. whether it is a regression or classification task. In the instance of multi-class classification where the number of classes is greater than 2, you calculate separate loss for each class label and sum the total. This is known cross-entropy. It is defined as follows:

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}), \tag{2.6}$$

where M is the number of class, $y_{o,c}$ is a binary indicator of whether class c is in observation o and $p_{o,c}$ is the predicted label from the model that observation o is in class c.

Another important loss function that we will be considering is the binary cross-entropy. Since we have

only two classes, i.e. summary frame and non-summary frame. It can be defined as follows,

$$\textit{binary cross-entropy} = -(y_{o,c}\log(p_{o,c}) + (1 - y_{o,c})\log(1 - p_{o,c})), \tag{2.7}$$

where $y_{o,c}$ is the true class label of whether class c is in observation o and $p_{o,c}$ is the predicted label of whether class c is in observation o from the model. Binary cross-entropy is the same as multi-class cross-entropy with the number of classes equalling to 2.

The cost function tells us how much error the model has incurred. However, we would ideally like to minimise this so that our model can perform better than previous attempts. To do this we employ optimisation algorithms to the network. An optimisation algorithm is used to find a suitable value for the weights and bias so as to minimise the error between the actual predictions and the predictions produced by the network. The simplest first order gradient-based optimisation algorithm used in neural networks is the gradient descent algorithm (shown in equation Equation (2.8)). This algorithm takes the derivative of the loss function with respect to the parameters of the network $\boldsymbol{W}$. There are many variations of the gradient descent algorithm such as stochastic gradient descent (SGD), adaGrad [12], Adam [27] etc. which can be used instead of the classic gradient descent since it yields faster convergence.

$$\boldsymbol{W} := \boldsymbol{W} - \alpha\frac{\partial J}{\partial \boldsymbol{W}}, \tag{2.8}$$

where $\frac{\partial J}{\partial \boldsymbol{W}}$ is the partial derivative of the cost function J with respect to $\boldsymbol{W}$ and α is the learning rate.

One of the variants of the standard gradient descent is the Adaptive Moment Estimation (Adam) algorithm [27]. This is also a first-order gradient-based optimisation algorithm for stochastic objective functions. This algorithm computes individual adaptive learning rates for different parameter estimates of first and second moments of the gradients. The aim is to combine the advantages of AdaGrad [12] which excel when dealing with sparse gradients and RMSProp [47] which work well with non-stationary settings [27]. [27] arrive at the following for the moments of the gradients,

$$m(t) = \beta_1 m_{t-1} + (1 - \beta_1)g_t, \tag{2.9}$$

$$\hat{m}(t) = m_t + (1 - \beta_1^t), \tag{2.10}$$

$$v(t) = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \tag{2.11}$$

$$\hat{v}(t) = v_t + (1 - \beta_2^t), \tag{2.12}$$

where g_t is the gradient of the objective function with respect to the parameters $\boldsymbol{W}$ at time step t, $m(t)$ is an estimate of the first moment of the moving average (the mean) of the gradient, $v(t)$ is an estimate of the second moment of the moving average (the variance) of the gradient, $\beta_1, \beta_2 \in [0, 1)$ are hyper-parameters which control the exponential decay rates of the moving averages. $\hat{m}(t), \hat{v}(t)$ are bias-corrected terms, since the moving averages are initialised as 0 this leads to moment estimates that are bias toward 0, especially when the decay rates are small (βs are close to 1). After this the parameters are updated as follows [27],

$$\boldsymbol{W}_t = \boldsymbol{W}_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{2.13}$$

where $\boldsymbol{W}_t$ is the parameters of the objective function at time step t, α is the learning rate (just as in the gradient descent) and ϵ is a small number to prevent division by zero. Empirical analysis shown in [27] indicates that suitable values for the hyper-parameters should be as follows, α = 0.001, β_1 = 0.9, β_2 = 0.999 and $\epsilon = 10^{-8}$

Adam works well in practise, it is computationally efficient (converges fast), has few memory requisites and is suitable for tasks that are large in terms of data and number of parameters [27].

The cost function is used to update our weight matrices in the network with the help of back-propagation, this is repeated until convergence. Back-propagation introduced by [40] is a way of calculating the gradient of the cost function and propagates the error derivatives back through the network. Back-propagation demands a known output for each input value. At this point, the estimation produced by the output is compared with the ground truth supplied when constructing the model. After each iteration the error produced is fed to the previous layer where the weight matrix is then updated according to Equation (2.8), in the case where we use gradient descent as the optimiser, until the update changes by less than some threshold or the derivative is zero. After convergence we end up with an optimal parameter set.

Hyper-parameters

A hyper-parameter is simply those values that require to be set before the training process is started.

One of the hyper-parameters to consider when feeding data into the NN is the batch-size. Normally when speaking about batch-size we refer to mini-batch. This is the number of training examples pushed through the model in order to get a single update of the model parameters. In mini-batch the number of training samples is greater than one but less than the total number of training samples in the dataset. We introduce this technique for computational efficiency. If we have 10 training samples and a batch-size of 2, we would end up with a total of 5 model-parameter updates for the network to pass through all our data.

Another hyper-parameter to contemplate is the number of epochs. An epoch is a complete run-through by the model through the data set out to be learnt. Usually we train our models for more than 1 epoch to learn a good representation of the data.

Depending on the type of problem and how it is solved, you could have many more hyper-parameters.

2.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [29] are a special kind of neural network that does more than just multiplying weights by inputs such as a plain vanilla neural network. It can be used to solve time-series problems but most of its success has come from visual tasks. Convolution is a specialised kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers [16].

When we refer to convolution on an image, we usually refer to the image as the input. This input gets convolved with a kernel to produce a feature map. In neural networks this operation is substituted for dense matrix multiplication i.e. multiplying inputs by the weight matrix. The input and kernel are both multi-dimensional arrays, often referred to as tensors. We can define this discrete convolutional operation mathematically as follows:

$$S(i,j) = (I * K)(i,j) = \sum_{m} \sum_{n} I(m,n)K(i-m, j-n), \tag{2.14}$$

where I is the input image, K is the kernel, S is the output(feature map), $m < i$ and $j < n$. It should be

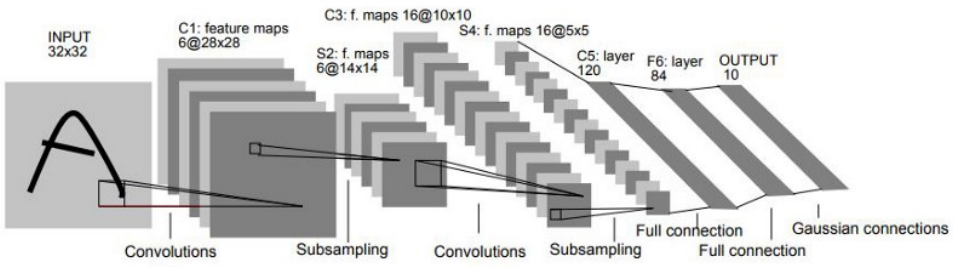


Figure 2.2: Typical CNN architecture as shown by [30] which demonstrated the power of CNNs. Known as the Le-Net network, it consists of 7 layers and takes an input image of 32×32 pixels. This networks is made up of 3 convolutional layers, 2 fully connected layers and 2 sampling layers. The final layer consists of 10 neurons representing the 10 numerical digits.

noted that most machine learning libraries actually do cross-correlation rather than convolution, however both are equivalent when performing optimization as cross-correlation essentially just uses a flipped version of the kernel used in Equation (2.14). Usually the size of the kernel is kept small in order to capture small meaningful features.

In a CNN the kernels are learnt during the training step and parameters are shared across receptive fields. This means that only one set of parameters are learnt for every location [16] and the same kernel is used for the receptive field of a neuron. Hence, convolution is much more memory efficient than dense matrix multiplication, as employed in vanilla neural networks.

Usually CNNs go through 3 stages: convolution, detection (usually non-linear activation function) and pooling. A pooling function replaces the output of the network at a certain location with a summary statistic of the nearby outputs [16]. The most common form of pooling used is Max pooling [55] which returns the maximum of a rectangular neighbourhood. The reason for adding this layer is so that it helps the representation become somewhat invariant to small alterations in the input. The architecture typically used when constructing CNNs can be seen in Figure 2.2

2.1.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) introduced by [40] are a type of network that excels at sequential modelling. It specialises in processing a sequence of input values such as language modelling, time series data, videos with dependencies, etc. As mentioned previously in CNNs, RNNs also have parameter sharing. This being weights shared across multiple time steps rather than across input weights.

When we refer to recurrent or recursive networks we refer to a function that state is a function of its

previous state i.e.

$$s^{(t)} = f(s^{(t-1)}; \boldsymbol{W}) \tag{2.15}$$

where s is that state of the system, t refers to a time step, $\boldsymbol{W}$ is the parameters of the system at that time step. Usually RNNs are of a deep nature, which means we have a few hidden layers in the model as well. In these hidden layers we have the "recurrent" stage of the network, i.e. the model will have a feedback loop back to the hidden state from the previous hidden state, although other variations are possible such as output nodes back to hidden nodes and so forth. The reason for this recurrent architecture is that during training, the model will be able to keep some aspects of previous inputs (or temporal content).

RNNs usually have inputs, hidden and output nodes which can be formulated as follows;

$$a^{(t)} = b + \mathbf{H}h^{(t-1)} + \mathbf{U}x^{(t)}, \tag{2.16}$$

$$h^{(t)} = \sigma(a^{(t)}), \tag{2.17}$$

$$o^{(t)} = c + \mathbf{V}h^{(t)}, \tag{2.18}$$

$$\hat{y}^{(t)} = \texttt{softmax}(o^{(t)}), \tag{2.19}$$

where b and c are bias vectors along with the weight matrices **U**, **V** and **H** which refer to input-hidden, hidden-output and hidden-hidden respectively [16]. The above equations refer to an output sequence of the same size as the inputs. Other variable outputs can also be employed depending on the task at hand where models such as encode-decoder sequence-to-sequence architectures can be used. A very basic RNN architecture can be seen in Figure 2.3

Gated RNNs

The challenge with many RNN architectures is long-term dependencies. They face either the vanishing or exploding gradient problem [16]. Long-term dependencies are one of the main challenges in deep learning when dealing with sequential data. One of the most effective ways to address this problem is to

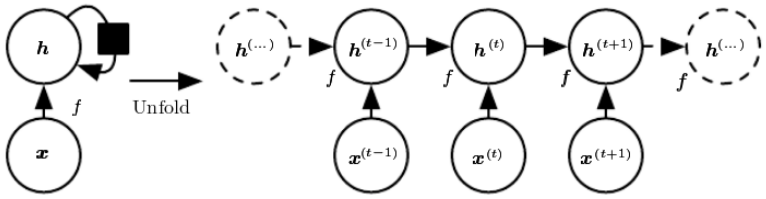


Figure 2.3: A basic RNN architecture [16]. By unfolding a RNN we are able to see that outputs from previous time steps are taken as inputs to present time steps. This is an example of passing information through time. The black square represents a delay in a single time step.

use gated RNNs. The long short-term memory (LSTM) model [21] helps model long-term dependencies in data, which allow gradients to flow for long durations. These LSTM cells are nodes within the RNN that have internal recurrence (self-loop) which is separate from the normal feedback loop present in typically RNN architectures. The internal state of these LSTM cells are now governed by a set of "gates". LSTM cells consists of 3 gates; input, forget and output gates. These allow certain neurons to remember, forget or to send information to the outputs. The self loop in the LSTM cell is controlled by a forget gate, this gate assigns either 0 or 1 to the output via a sigmoid unit.

Other gated cells also exist such as peephole connections [15], GRU cells [9], etc. These are essentially just variations of the LSTM cells.

Bidirectional RNNs

Previously we have discussed sequences that only capture information of the past and present. In some tasks we would like to consider the entire sequence when making predictions. Examples include, natural language processing (NLP), speech recognition, handwriting recognition, amongst others. We ideally want to have a holistic semantic understanding on the full input sequence before making predictions. Fortunately, we have bidirectional RNNs to assist in this regard. Bidirectional RNNs were introduced to address exactly this need [42]. These architectures have been extremely successful in hand-writing recognition, speech recognition and so on [16].

As the name suggests, we start with a RNN that starts at the beginning of the sequence and we have another RNN that starts at the end of the sequence moving backward through time [16]. In Figure 2.4 we see the bidirectional RNN architecture. A bidirectional RNN computes a mapping $f : \mathbf{x}^{(t)} \rightarrow \mathbf{y}^{(t)}$ with a loss of L at time t. $\mathbf{x}^{(t)}$, $\mathbf{y}^{(t)}$ are the inputs and targets at time t respectively. As can be seen in Figure 2.4, $\mathbf{h}$ represents the forward pass of information through time in the network while

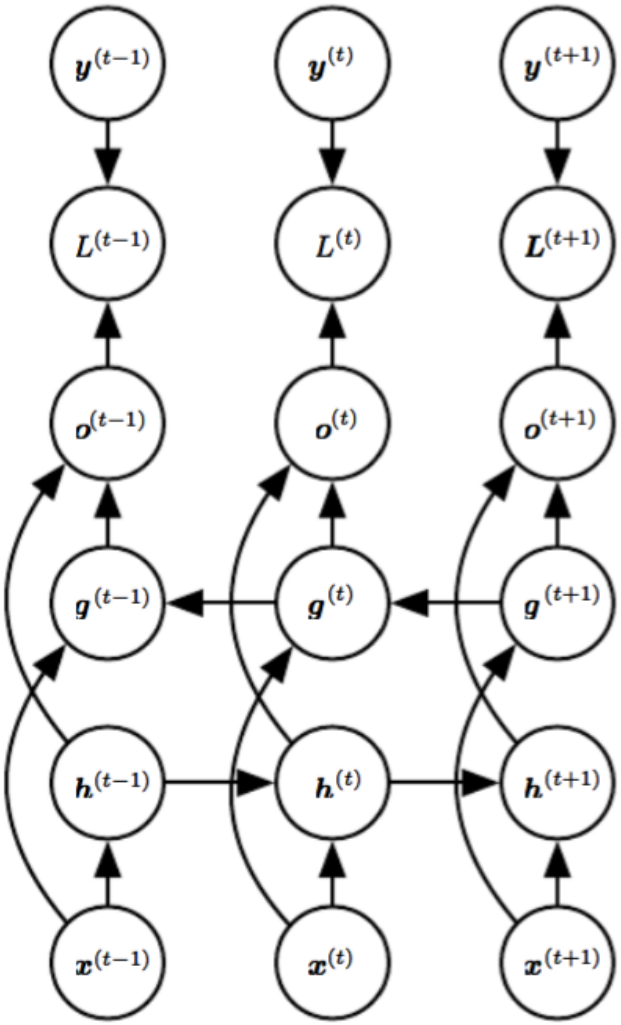


Figure 2.4: Computation of a bidirectional RNN architecture [16].

$\mathbf{g}$ constitutes the backward propagation of information through time, represented by the right and left arrows respectively. This means that the output units $o^{(t)}$ benefits from the information at time t of the past $\mathbf{h}^{(t)}$ and the future $\mathbf{g}^{(t)}$.

2.1.4 Auto-Encoders

An Auto-Encoder (AE) is a type of neural network that maps its inputs to its outputs. This means that the size of the outputs must correspond to the size of its inputs i.e, $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$. AEs were originally developed for dimensionality reduction. When [20] first introduced the concept AEs were compared with Principal Component Analysis (PCA), another dimensionality reduction algorithm, and showed that AEs outperform PCA when non-linear mappings are needed to represent the data. AEs' architecture consists

of an encoder part and a decoder part, where the last layer (output) of the encoder is the first layer (input) of the decoder. This layer is called the **code** layer or also known more recently as the latent variables when discussing generative models. Both encoder and decoder models are feed-forward neural networks. Generally we use under-complete AEs in order to reduce the dimensions of the input, this means that the size of the **code** layer is smaller than the size of the input layer. By doing this, we force the network to capture the most important features of the input data. It should be noted that when the decoder part of the network is linear and the cost function is the mean squared error, then the linear AEs essentially learns to span the same subspace as PCA [16].

We can formulate of the above mentioned network mathematically as follows: $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ and $g : \mathbb{R}^m \rightarrow \mathbb{R}^n$, where f represents the encoder network and g represents the decoder network and $n > m$. If we consider one hidden layer for simplicity then formulate the AE as follows,

$$\mathbf{h} = \sigma(\boldsymbol{W}\mathbf{x} + \mathbf{b}), \tag{2.20}$$

$$\mathbf{x}' = \sigma^{'}(\boldsymbol{W'}\mathbf{h} + \mathbf{b}'), \tag{2.21}$$

where $\mathbf{h} \in \mathbb{R}^m$ is the hidden layer also referred to as the code layer. $\boldsymbol{W}$, $\boldsymbol{W'}$ is the weight matrix from the input layer to the code layer and from code layer to output layer respectively.$\mathbf{x}$, $\mathbf{x}' \in \mathbb{R}^n$ is the input and outputs respectively. $\mathbf{b}$, $\mathbf{b}'$ is the bias for the hidden and output layers respectively. $\sigma, \sigma^{'}$ are some activation functions.

Like many machine learning algorithms, AEs exploit the idea of the data being concentrated around a low dimensional manifold - this is some space that resembles some space near each point, i.e seeks to describe the low-dimensional, smooth structure of our high dimensional data otherwise referred to "high-dimensional surfaces". Thus, if the data is concentrated around some low-dimensional manifold, it yields representations that capture a local coordinate system for this manifold [16]. This means that the encoder learns a mapping from the input space to the representation space.

There are many variations of AEs such as, over-complete AEs, sparse AEs, de-noising AEs, etc. Each one carrying an importance in certain tasks. Applications of AEs range from dimensionality reduction to information retrieval.

2.2 Background: Algorithms Adopted In Video Data

2.2.1 Transfer Learning

Pre-trained models are models that have been developed and trained previously, usually by someone else that has tried to solve a similar problem. More specifically, pre-trained means the weights are initialised by pre-training on a different dataset offline so that the weights are at a good location on the multi-dimensional error surface (probably close to a minimum). The benefit of this is that we can apply some sort of transfer learning in order to get those features that were previously used. Transfer learning is a process used in deep learning where a model trained on one particular task is re-purposed to be used in a different, but still related task. The general idea is to use what was previously learnt and adapt it for what your task is. In certain cases, some people have access to much more resources than others, this enables them to train huge networks with a large amount of data.

Even though these pre-trained models might not be completely applicable to your task, it saves a lot of costs having to train your own network. These networks are made available to the public to use and/or tune to their needs. In most cases its much more time-efficient to use a pre-trained model rather than building up one from scratch. Fortunately, implementations of pre-existing models are available in a myriad of libraries.

Usually after using these pre-trained models we would fine tune our network to suit our problem task and to increase its accuracy. To do this, you would usually adjust the last/final layer of the pre-trained network to suit your model, an example would be the last layer would have 10 outputs instead of 1000 class predications as in the GoogleNet architecture. We can also eliminate the last layer of our pre-trained network and use this as a feature extractor for our task.

2.2.2 Optical Flow

Optical flow refers to the change/flow of pixels from one image to another. Thus, it is usually applied to videos. When dealing with optical flow in machine learning code, the dense optical flow returned signifies how much a pixel moved in the x direction and in the y direction from one image to its successor image. There are two common ways used to track pixel movements, namely, sparse optical flow and dense optical flow. Sparse optical uses some feature points (interest points/features) and computes the

optical flow between successive frames.

Dense optical flow, which we will be using, usually involves using the Gunner Farnebacks algorithm [14], also referred to as two-frame motion estimation algorithm. This algorithm, unlike in the sparse optical flow, computes optical flow for each pixel in a rectangular grid, spaced at every i-th pixel. If i=1 then all pixels are computed.

Farneback's algorithm approximates each neighbourhood of each pixel with a quadratic polynomial. Thus we have,

$$f_1(\mathbf{x}) \sim \mathbf{x}^T \boldsymbol{A}_1 \mathbf{x} + \mathbf{b}^T \mathbf{x} + c_1, \tag{2.22}$$

where $\boldsymbol{A}_1$ is the symmetric matrix, $\mathbf{b}$ a vector and c a scalar. The coefficients are estimated from a weighted least squares fit to the signal values of the neighbourhood [14]. If we consider a new signal $f_2(x)$ by some global displacement $\mathbf{d}$ then,

$$\begin{aligned} f_2(\mathbf{x}) &= f(\mathbf{x} - \mathbf{d}) \\ &= \mathbf{x}^T \boldsymbol{A}_1 \mathbf{x} + (\mathbf{b}_1 - 2\boldsymbol{A}_1 \mathbf{d})^T \mathbf{x} + \mathbf{d}^T \boldsymbol{A}_1 \mathbf{d} - \mathbf{b}_1^T \mathbf{d} + c_1 \\ &= \mathbf{x}^T \boldsymbol{A}_2 \mathbf{x} + \mathbf{b}_2^T \mathbf{x} + c_2. \end{aligned} \tag{2.23}$$

Equating the coefficients in the quadratic polynomial above yields,

$$\boldsymbol{A}_1 = \boldsymbol{A}_2, \tag{2.24}$$

$$\mathbf{b}_2 = \mathbf{b}_1 - 2\boldsymbol{A}_1 \mathbf{d}, \tag{2.25}$$

$$c_2 = \mathbf{d}^T \boldsymbol{A}_1 \mathbf{d} - \mathbf{b}_1^T \mathbf{d} + c_1, \tag{2.26}$$

Then, if $\boldsymbol{A}_1$ is non-singular we can solve for the translation $\mathbf{d}$ in Equation (2.25). We get, $\mathbf{d} = -\frac{1}{2}\boldsymbol{A}_1^{-1}(\mathbf{b}_2 - \mathbf{b}_1)$

2.2.3 Kernel Temporal Segmentation (KTS)

The KTS algorithm [38], originally introduced as part of a video summarisation algorithm, aims to segment a video sequence into semantically-consistent segments i.e. better understood as spatio-temporal volumes. That is, given a video input sequence the KTS algorithm will return a set of points that groups similar frames within a segment together. Hence the original video of size n will be broken up into non-uniform snippets of size l_i, $i = 0, ..., m$ where m is the number of snippets and l is the size of the snippet with $l < n$.

To get shot boundaries, one would generally compare successive frames relying on image descriptors. However, the KTS algorithm compares all pairs of frames which allows it not only to produce shot boundaries but also general change points within a video. This gives us valuable information of where the semantic content differs to other segments. These change points are detected by identifying the jumps in signals, which considers the entire sequence at once.

The KTS algorithm can be defined as in [38] as follows, define a sequence of feature descriptors $\mathbf{x}_i \in \mathbf{X}, i = 0, ..., n-1$. Let $K : \mathbf{X} \times \mathbf{X} \to \mathbb{R}$ be a kernel function between descriptors. Let Υ be the feature space of the kernel $K(.,.)$. Then denote $\phi : \mathbf{X} \to \Upsilon$ the associated feature map and $\|.\|_\Upsilon$ the norm in the feature space Υ. Thus, the aim is to minimize the following objective function:

$$\min_m J_{m,n} := L_{m.n} + Cg(m, n), \tag{2.27}$$

where m is the number of change points, n is the number of feature descriptors per video, $g(m, n)$ is the penalty term, defined as $g(m, n) = m(\log(n/m) + 1)$, which penalises segmentations with too many segments, $L_{m,n}$ measures the overall within-segment variance. This objective function yields a trade-off between under and over-segmentation. $L_{m,n}$ is then defined as :

$$\begin{aligned} L_{m,n} &= \sum_{i=0}^{m} v_{t_{i-1},t_i} \\ v_{t_i,t_{i+1}} &= \sum_{t=t_i}^{t_{i+1}-1} \|\phi(x_t) - \mu_i\|_\Upsilon^2 \\ \mu_i &= \frac{\sum_{t=t_i}^{t_{i+1}-1} \phi(x_i)}{t_{i+1} - t_i} \end{aligned} \tag{2.28}$$

The full KTS pseudocode can be seen in Algorithm 1 which summaries the above process.

Algorithm 1: Kernel Temporal Segmentation Algorithm [38].

Result: Set of change point positions $t_0, ..., t_{m^*-1}$

Input: temporal feature descriptors, $\mathbf{x}_0, ..., \mathbf{x}_{n-1}$

1. Compute the Gram matrix $\mathbf{A}$

$$a_{i,j} = K\langle \mathbf{x}_i, \mathbf{x}_j \rangle \text{ (inner product)}$$

2. Compute cumulative sums of $\mathbf{A}$

$$\hat{a}_k = \sum_{i=0}^{n} a_i,\ k = 0, ..., m$$

3. Compute unnormalised variances

$$v_{t,t+d} = \sum_{i=t}^{t+d-1} \hat{a}_{i,i} - \frac{1}{d} \sum_{i,j=t}^{t+d-1} \hat{a}_{i,j}$$

$$t = 0, ..., n-1,\ d = 0, ..., n-t \text{ (t = starting point, d = segment duration)}$$

4. Do the forward pass for the dynamic programming algorithm

$$\min_{t=1,...,j-1} (L_{i-1,t} + v_{t,j})$$

$$i = 0, ..., m,\ j = 0, ..., n$$

5. Select optimal number of change points as shown in Equation (2.27)

6. Find change points by back-tracking

$$t_m^* = n, t_{i-1} = argmin_t (L_{i-1,t} + v_{t,t_i})$$

$$i = m^*, ..., 1$$

2.2.4 Knapsack Algorithm

Given a set of items, each having an associated weight and value, we have to maximize the total value of the combined items so that the combined total weight of the items is less that a predefined limit. Here we address the 0/1 knapsack dynamic programming algorithm. This means that an individual item which

has an associated weight and value assigned is either chosen or not. There is no partial choice as is the case with a greedy approach to the knapsack problem.

This problem can be stated mathematically as follows:

$$\begin{aligned} &\max \sum_{i=1}^{n} v_i x_i \\ &\text{subject to } \sum_{i=1}^{n} w_i x_i \leq W \text{ and } x \in [0,1] \end{aligned} \tag{2.29}$$

where v_i, w_i represents the values and weights respectively per item, n represents the number of items and x_i restricts the number of copies for each item, either to be included or not. This gives us the maximum value we can obtain given certain restrictions.

Once this is solved, we need to back track to see which items were selected that maximised our total value. To do this we refer to Algorithm 2.

Algorithm 2: Back tracking the knapsack algorithm to get the selected items [6].

Result: Unique items chosen

Input: value_array, weight_budget, n_unique_items

set i = n_unique_items and w = weight_budget

while *i and w > 0* **do**

    instructions;

    if *value_array[i,w] != value_matrix[i-1,w]* **then**

        mark item;

        set w = w - weights[i];

        set i = i - 1;

    else

        set i = i - 1;

    end

end

value_array$_i$ is the value item i holds in the value_array. The weight_budget$_i$ refers to the weight item i has. Then, for each item i there is an associated value and weight. Thus, after the algorithm is finished executing we have a single array of length n, consisting of ones and zeros that represent weather the item

was selected or not.

2.2.5 Determinant point processing

Determinantal point process (DPP) was first introduced by [33] to address the Pauli exclusion principle, which states that two particles cannot simultaneously occupy the same quantum state. Thus, it has since made this a very useful tool for modelling diversity in applications such as document summarisation, image ranking, etc. [5]. In the case of video summarisation, this means that similar scenes should not be selected for the final summary. When we relate DPP to the task of summaries, the goal is to generate a summary by having diverse as well as representative information in the summary. The idea is that each item has an associated feature vector. When the dot product is taken between two feature vectors, the angle resulting from these two feature vectors will determine how similar two items are.

Then we can define the DPP as follows. Let Ψ = {1, 2, ... , N} be a ground set of N items (in our case images). DPP defines a discrete probability distribution over 2^N possible subsets. Let Y denote the random variable of selecting a subset. Then we have the probability measure P over subsets $Y \subseteq \Psi$. Y is then distributed according to,

$$P(Y = y) = \frac{det(\boldsymbol{L}_y)}{det(\boldsymbol{L} + \boldsymbol{I})} \tag{2.30}$$

where $\boldsymbol{L} \in S^{N \times N}$ is known as the kernel which is a Gram matrix, $\boldsymbol{I}$ is a N $\times$ N identity matrix. $\boldsymbol{L}_y$ is the principle minor with rows and columns selected by indices contained in y and the the denominator in Equation 2.30 refers to the normalising term [33]. The kernel matrix $\boldsymbol{L}$ records the pairwise frame-level similarity. We see that the probability of the subset y is proportional to the determinant of the corresponding principal minor sub-matrix $\boldsymbol{L}_y$. By using the determinant we see an interesting property of pairwise repulsion. That is, if we consider selected two item i and j then we have - $P(Y = \{i, j\}) \propto \boldsymbol{L}_{ii}\boldsymbol{L}_{jj} - \boldsymbol{L}_{ij}^2$. If items i and j are the same then $P(Y = \{i, j\}) = 0$ which means identical items should not be in the same set. Thus the most diverse subset selection of Ψ would then be the one that attains the highest probability (pairwise dissimilarity):

$$y^* = \text{argmax}_y P(Y = y) \tag{2.31}$$

In order to exploit the diversity in DPP the the kernel matrix $\boldsymbol{L}$ is broken up into a quality score and

diversity features [5] as follows,

$$\boldsymbol{L}_{i,j} = q_i \phi_i^T \phi_j q_j \tag{2.32}$$

$$q_i = exp(\frac{1}{2}\boldsymbol{W}^T \mathbf{x}_i) \tag{2.33}$$

where $q \in \mathbb{R}_+$ is the quality score and $\phi \in \mathbb{R}^D$ is the diversity features which inherently computes the cosine angle between two items. $\mathbf{x}_i$ is the quality feature vector which encodes contextual information about i and its representativeness of other items. In a machine learning setting, the quality scores can be learnt. $\boldsymbol{L}$ is parametrised by $\boldsymbol{W}$ which is then optimised by using maximum likelihood estimation (MLE),

$$\boldsymbol{W}^* = \text{argmax}_{\boldsymbol{W}} \sum_n log P(Y = y_n^*; \boldsymbol{L}_n(\boldsymbol{W})) \tag{2.34}$$

where $\boldsymbol{L}_n$ is the $\boldsymbol{L}$ matrix formulated using items in the n-th ground-set and y_n^* is the corresponding ground truth summary.

One major drawback of using DPPs presented in [5] is that this algorithm is agnostic to the sequential data. Another drawback is that the DPP algorithm does not take full advantage of complex dependencies amongst items. In [5] they develop a new form of DPP that combats these challenges.

2.2.6 K-means Clustering

Originally introduced by [34], K-means is one of the simplest and arguably one of the most well known unsupervised algorithms that can be adopted to solve a myriad of problems.

The idea of K-means is to select the number of clusters a-priori before starting the algorithm computation. The number of clusters serves as the number of centroids the data will be clustered into. These centroids are usually selected randomly. Each point from the dataset is taken and the distance between the point and each centroid is calculated, the point is then assigned to the nearest centroid. Once all points has been assigned, the cluster centroid moves to the mean of the data points associated with the different clusters. These steps are repeated until there are no more changes of data points to the cluster centroids i.e. the cluster centroids do not move or they move less than some threshold. Thus the objective

is to minimize the squared error function:

$$J(v) = \sum_{i=1}^{N} \sum_{j=1}^{c} ||\mathbf{x}_i - \mu_j||^2, \tag{2.35}$$

where $\mathbf{x}_i$ is the observed data points, μ_j is the cluster centroids, $||\mathbf{x}_i - \mu_j||$ is the Euclidean distance between $\mathbf{x}_i$ and μ_j. The cluster centres are updated by:

$$\mu_i = \frac{1}{c_i} \sum_{j=1}^{c_i} \mathbf{x}_i, \tag{2.36}$$

where c_i represents the number of data points in the i^{th} cluster. Equation (2.35) and Equation (2.36) is repeated until convergence.

We only compute distances between points and cluster centres. Thus, we end up having linear complexity $O(n)$. There are some disadvantages with K-means such as selecting the number of clusters a-priori which is some cases will not be beneficial. Another disadvantage is that when starting the algorithm, cluster centres are initialised at random which means multiple runs of the algorithm can yield dissimilar results due to the fact that we are optimising a function that is non-convex and thus can lead to local minima instead of global minima.

Other algorithms exist to address choosing cluster centres, distance metrics, random initialisation, etc. as shown in [10; 13]

2.3 Background: Additional Concepts

2.3.1 Video Sequence

A video sequence is a concatenation of images stacked after each other. When considering a particular frame (image) in a video, the frame preceding it shows small variations of pixel changes from one image to the other. When many images are stacked next to each other in this fashion and looped through at a reasonable rate, we get to witness the flow of pixels through these images. This constitutes our video sequence.

Video sequences have visual queues and more often than not it will have audio queues as well. It is a

series of scenes concatenated together to produce some context.

2.3.2 Hand crafted global descriptors

One of the challenges faced when feeding features to a model is that, generally, we need them to be of the same dimensionality. Additionally, we would like them to be computed fast and easily. When computing frame level features by hand crafting, one of the main tasks is to get a global descriptor to represent an image. We came across a few techniques that can help us accomplish this.

Bag of Features (BoF) helps us get a k-dimensional output vector [23]. It groups local descriptors into a histogram. First it computes some feature descriptors on a particular image, then k-means clustering is performed to group each descriptor to a particular so-called visual word. The histogram is populated by the number of descriptors that are assigned to a certain cluster/visual word. Thus BoF produces a k-dimensional feature that will be normalized. Secondly we get Fisher kernels which is a powerful algorithm used to calculate a fixed size vector given a set of variable length input samples [23]. This too takes a set of local descriptors as inputs, after which a Gaussian Mixture Model (GMM) is trained on these inputs. The derivative of the log-likelihood is taken with respect to the GMM parameters and concatenated to produce the Fisher Vector of fixed-size. The GMM parameters consist of the mixture weights, means and standard deviations.

2.3.3 Conclusion

This chapter presented a brief survey of definitions, general terms and selected algorithms needed to understand the content presented in this dissertation. In this chapter we discussed some necessary algorithms that will directly be used in the remainder of this project.

The next section will cover previous works done in the realm of video summarisation as well as feature representations available for us to leverage.

Chapter 3

Literature Review

Over the past few years since the rush of deep learning, we have had access to more powerful machines that allow us to feed our models much larger amounts of data, such as videos. However, when we compare the amount of work done in video summarisation to other fields such as action recognition, object detection, etc. we see only a handful of notable work that leverage deep learning. This indicates that there is still a lot of work to be done for the task of video summarisation.

With the recent advances in deep learning in various fields such as object detection, action recognition, language processing, amongst others, we see that these deep learning architectures outperform traditional methods. Traditional methods require some insight and domain knowledge in order to get reliable results. One disadvantage when using hand-crafted features/traditional methods is that it could restrict us to a certain domain whereas with deep architectures we are able to employ some form of transfer knowledge. Thus, we pursue the deep learning approach to solve the video summarisation problem. A major advantage deep learning models have is that previously-trained models can be used as a feature extractor on similar problem sets. Alternatively, you can also fine-tune these pre-trained models to suit your dataset and use these fine-tuned models instead of creating an entire model.

The most popular approach to tackle this problem is to start off by extracting frame feature representations and subsequently feed it into a model to learn how to perform video summaries. The bulk of the work we review in this section deals mainly with the choice of architecture rather than a combination of good, suitable features along with the architecture.

3.1 Previous Approaches

3.1.1 Non-Deep Learning Approaches

[38] introduced the KTS algorithm that is widely used in segmenting videos into non-uniform snippets. However, in their paper they also attempt to solve the video summarisation problem. They initially introduce the KTS algorithm to segment videos by getting change points. They then introduce the concept of importance scoring on segments by taking the segments that had the highest classification scores. They extract local descriptors for every 5^{th} frame and encode it by using Fisher Vectors producing a 16512 dimensional vector per frame. For segmentation they normalise each descriptor by applying L_2 normalisation. They formulate this as supervised problem and use binary labels to feed this into a SVM for classification. They report an average F1-score using leave-one-out evaluation. Their results are reported on the MED-Summaries dataset which they also introduced in this paper. They achieve a average F1-score of 41% for this dataset.

In [18] they introduce the SumMe dataset which is widely used as a benchmark dataset for video summarisation. The method they used to address this problem starts off by segmenting the video by using superframe segmentation. They then evaluate visual interestingness per superframe by selecting a set of features. Based on this interestingness scoring they select an optimal subset of superframes to create a summary. Superframes are a motion-based video segmentation algorithm. Superframes find points in the video that creates a change point when there is no motion. Secondly, they compute per-frame interestingness by taking a weighted sum of features which consists of attention, quality, landmarks, people and follow objects. Thus, the interestingness score is then given by:

$$I(S_i) = \sum_{k=m}^{n} i_k, \tag{3.1}$$

where m_i and n_i are start and end points of a superframe S_i and i_k refers to the interestingness of the frame k. To get an optimal summary they find a subset of superframes that is under a pre-defined threshold. This is done by using the 0/1 knapsack algorithm. They sub-sample every 5^{th} frame for interestingness features and all frames for motion features. They formulate this as a unsupervised problem and achieve a average F1-score of 39.7% on the SumMe dataset.

In [44] they introduce the TvSum50 dataset which along with the SumMe dataset is used as benchmarks

when addressing the video summarisation problem. Experimental results show that there is a high degree of labelling consistency when compared to the SumMe dataset. They formulate the video summarisation problem as a unsupervised problem that uses title-based image search to get visually important shots. They argue that video titles are carefully chosen to be descriptive of its main topic and hence images related to the title can aid in extracting visually important concepts of the video. The idea of this framework is to use titles to extract visually important shots. This motivated them to collect title-based image search results and use that to select key shots that are relevant to canonical visual concepts shared by the video and the images. For each search they collect 200 images using Yahoo image search engine. Canonical visual concepts refer to the patterns shared between a video and the title-based image search. They call these patterns co-archetypes. They compute shot-level importance scores by taking the average of frame importance scores within each shot. They then use the 0/1 knapsack algorithm to extract summary shots to be under some constrained budget so that an actual summary can be produced. They extract a set of image descriptors namely, RGB, Histogram of Oriented Gradients (HOG) (shape information), GIST (global descriptor that gives a representation of scene information. In other words, the gist of a scene) and Scale Invariant Feature Transform (SIFT) (appearance information). They then encode this as a bag-of-words by using k-means on a 5% subset of the descriptors. They concatenate these features to obtain a 1640 dimensional vector. They achieve an average F1-score of 51.3% on the TvSum dataset

3.1.2 Deep Learning Approaches

Supervised Learning

The approach which served as a baseline to many subsequent papers was proposed by [52]. They introduce a standard of converting key-frames (single independent frames from a video sequence) to key-shots (multiple sequential frames taken at different points in a video) to frame-level importance scores and vice versa which is essential when evaluating against previous papers. This method for converting is used in many papers which will be discussed later. They introduce three different forms of testing namely, **canonical**- using a 80-20 split from the same dataset, **augmented**- 80% of the dataset with the remaining datasets for training and the 20% from that dataset for testing and **transfer**- use all the remaining datasets for training and an entire single dataset for testing. They initially extract visual features for each frame using the pool 5 layer of the GoogLeNet model. Thus for each frame they end up with a feature representation of 1×1024 dimension. They sub-sample 2 fps since the datasets contain slow-varying

contents. The model which they developed is called dppLSTM using a supervised approach which aims to widen the diversity in the selected frames by removing redundant frames. This model consists of a 2 bidirectional LSTM layers with 256 units each. Thereafter, splits into 2 MLPs, both consisting of one hidden layer with 256 sigmoid units. The first MLP is for frame-level-importance scores and the second is for outputting the similarity. The output for the first MLP is a sigmoid activation while the second MLP is a linear activation. The model essentially combines LSTM layers' hidden state and visual features with a MLP. Their model can use both binary and frame-level scores for training. For the first MLP they use the squared loss, for the second MLP which deals with the dpp-similarity they use the likelihood. They perform KTS on all testing videos to segment the videos into non-uniform snippets. They use the remaining unseen data for testing, in which they get the model predictions for all the testing videos. They then rank each snippet in descending order based on the average score for each snippet. Thereafter, they use the 0/1 knapsack dynamic algorithm mentioned above (Algorithm 2) to select the most valuable snippets so that the total duration of the summary is under some threshold. They evaluate their results on two key datasets, namely, **SumMe** [18] (shot-based) and **TVSum** [44] (frame-level-importance) where they achieve state-of-the-art at the time of publication using harmonic mean F-score for evaluation. They achieve $38.6 \pm 0.8\%$ and $54.7 \pm 0.7\%$ on the SumMe and TvSum datasets, respectively.

Usually, supervised approaches achieve better accuracies than unsupervised approaches [24]. An *Attentive encoder-decoder framework* was introduced by [24] which they call AVS. This is a supervised approach that aims to mimic the way humans summarise videos. They formulate the problem as a sequence-to-sequence problem where the inputs are the video features and the output is a sequence of keyshots. The framework itself consists of two components, firstly, the encoder-decoder model-which measures the importance of each frame and secondly the keyshot selection model-that aims to convert frame-level importance scores to shot level scores and produce a summary which similar to [44]. For the AVS model, they construct the encoder to capture the temporal information by using a bi-directional RNN - this takes information of frames previous and successive frames for a specific frame i_t, in essence captures contextual information around a specific frame. The features of the video is the input to the encoder. The decoder uses an attention-based LSTM network that takes the representation vectors from the encoders as inputs and produces the corresponding output sequence. They aim to exploit the temporal ordering in videos by introducing attention mechanisms in the decoder. This allows the decoder to focus on only a subset of inputs of the representation vectors by adjusting the weights of each vector. The attention weight is computed at each time step, to get these weights of attention the relevance score

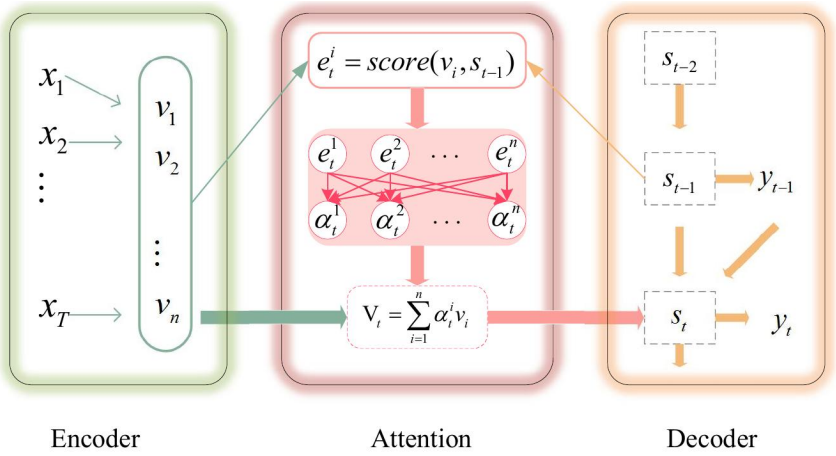


Figure 3.1: Model overview of attention mechanisms [24].

e_i^t for each feature is computed by adding the previous hidden state i in the LSTM decoder to the output of the encoder at time step t and later normalised. Once this step is completed, the weighted sum is fed into the decoder inputs. The model overview proposed by [24] can be seen in Figure 3.1. They extract GoogLeNet features for each frame which are fed as inputs to the encoder. Both the encoder and decoder models consists of three LSTM layers of 256 units. They use gradient decent with a learning rate of 0.15 and batch size of 16. The model is run 5 times and the average is reported which is consistent with [52]. Once the predicted importance scores are computed for all frames, they run the KTS algorithm to get change points in the video which is then converted into snippets. After which shot-level scores are obtained by taking the average of frame scores within each shot. Similar to [52] they then push these shots into the 0/1 knapsack dynamic algorithm to get the most valuable segments that fall below a specific pre-defined budget. By adopting this method, they achieve state-of-the-art results, getting an F-score of 44.4% and 61.0% for the SumMe and TvSum datasets respectively.

[37] proposed a fully-convolutional sequence network (FCSN) to solve this problem. They formulate this as a sequence labelling problem by assigning each individual frame a binary label of 0 or 1. The aim is to choose a subset of frames to form the video summary. Their key contribution is to draw similarities between video summarisation and semantic segmentation, which is essentially assigning a class label to each pixel in an image. FCSN consists of a stack of convolutions whose size grows as the network gets deeper. This is aimed to capture long-range dependencies among frames. The architecture used in FCSN is an encoder-decoder formulation. The encoder network is tasked with processing input frames features to capture both high level semantic features and to capture long-range structural relationships between the input frames. The decoder is tasked with producing a sequence of 0/1 class labels. The design of this

network was inspired by existing models available in semantic segmentation. This network performs temporal convolutions, pooling and deconvolution. The model consists of 8 convolutional layers and each layer performs temporal convolution. The first 5 layers is followed by batch normalisation, ReLU activation and maxpooling. Layers 6 and 7 consists of ReLU and dropout for regularisation. At the 8^{th} convolutional layer they perform batch normalisation and deconvolution across time. They then perform a skip connection from the 4^{th} convolutional layer to the deconvolutional layer and merge it to get the final class labels. Their intuition behind using a skip connection is similar to [19] which aims to recover temporal information. They formulate the problem as a supervised problem (although their method was also adapted to an unsupervised setting in their paper) and define the following loss function:

$$loss = -\frac{1}{N}\sum_{n=1}^{N} w_{c_n} log(\frac{\exp(\phi_n, c_n)}{\sum_{c=1}^{C}\exp(\phi_n, c)}), \tag{3.2}$$

where c_n is the ground truth label for the n-th frame, ϕ_n, c indicate the score of predicting the n-th frame as the c-th class, w_c is the weight of class c and N represents the total number of frames of the video. They subsample 2 fps and extract GoogLeNet features as frame-level feature descriptors for each frame in the video. For training they use SGD with a learning rate of 0.001, momentum 0.9 and batch size to 5. During testing they temporally segment videos using KTS and get the average interval score. They then rank these intervals based on these scores and perform the 0/1 knapsack algorithm to select a summary of a predefined length. Thus, they follow [52] to create key-shots from key-frames. They achieve 47.5% and 56.8% on two benchmark datasets the SumMe and TvSum respectively.

[36] took a different approach by analysing human cues for video summaries: the nature and intensity of actions, they term this as *actioness*. They run an experiment that investigates the correlation of human generated summaries and actioness ranks. They develop a method that incorporates actioness data to regulate a learning algorithm. Temporal actioness is defined as "what an agent can deliberate do with a bodily movement that leads to side effects" [36]. This involves the likelihood of a given video segment to contain an action. To estimate actioness they segment the video using KTS and ask five users to label each segment on whether a segment contains some action or not. They extract spatial features for each frame using a pre-trained CNN. These features are fed into a bi-directional GRU model to extract the corresponding temporal features. They then aggregate these features to get a spatio-temporal feature vector. These features represent visual information as well as well temporal information of other frames. The features are into two independent MLP networks. The first one is to minimise importance estimation

loss and the second one is for actioness ranking by minimising the actioness classification loss. They then combine the two loss functions into a single joint loss function which the network is then trained to learn this new set of trainable parameters $\mathbf{W}$. Thus, the loss is defined as:

$$\text{argmin}_{\mathbf{W}} S(\mathbf{W}) + \lambda R(\mathbf{W}), \tag{3.3}$$

where $S(\mathbf{W})$ is the summarisation loss, $R(\mathbf{W})$ is the actioness summarisation loss and λ is the regularisation weight to prevent bias towards one loss. For the first MLP, importance estimation, they employ DPP to ensure diversity selection amongst selected frames. Then

$$S(\mathbf{W}) = -\log(\frac{det(L_y)}{det(L+I)}), \tag{3.4}$$

where L is the marginal kernel, I is the identity matrix and L_y is the principle minor. For the second MLP, actioness ranking, they map spatio-temporal features to an actioness rank by using categorical cross-entropy loss as follows:

$$R(\mathbf{W}) = -\sum_{i=1}^{n}\sum_{j=0}^{k} t_{i,j}\log(p_{i,j}) \tag{3.5}$$

where $p_{i,j}, t_{i,j}$ is the predicted and target values of rank j for the i-th frame. n is the number of frames and k is the total number of types of actioness. They extract GoogLeNet features as frame level descriptors for each frame. They use single hidden layer of 256 units for both the GRU network and the MLP networks. They use a Adam optimiser with a learning rate of 0.001. They run their models 5 times and report the average. Their results are 40.1% and 56.3% for the SumMe and TvSum datasets respectively.

Unsupervised Learning

In order to take into consideration the temporal dependencies of video highlighting like some of the previously-mentioned papers, [49] took advantage of using LSTM cells in their Robust Recurrent Auto-Encoder (RRAE) model. They also used a bidirectional RNN which, as the name suggests, flows both forward and backward through the network. This is so that all information is captured when the model is trained, the model is trained on *inputs from the past and the future* [16]. Their model is an unsupervised approach which they argue is more suitable for highlight clip generation. The key idea mentioned in this paper is that the auto-encoder is expected to correctly reconstruct the positive instances while the negative instances are not. Their auto-encoder consists of 1 hidden layer with linear activation functions.

For training they use gradient descent to optimise their parameters. They use a learning rate of 0.01, weight decay of 0.0005, and momentum of 0.9. They decide to extract deep features as opposed to the traditional bag-of-words or Fisher vectors. They extract C3D features, which are spatio-temporal features, by taking 16 input frames from the "FC6" layer. Initially they first performed web scraping in order to obtain user-edited data from YouTube and performed some preprocessing in order to get the videos into the desired format by segmenting their video into snippets using KTS. In order to test their system they asked 6 people to annotate highlights for each video, and they consider a snippet to be highlighted only if 4 people considered it as a highlight. For evaluation they sort the snippets in ascending order based on their reconstruction errors. They then compare the hit rate of the snippets with their ground truth. Finally, mean average precision (mAP) was used to evaluate their results and achieved score of 0.434 on the dataset which they scraped from online which is approximately 0.06% better than a standard AE.

Another unsupervised approach is done by [35] who uses a generative adversarial network for their summary predictions. They aim to select key frames such that the distance between the feature representation and the video is minimised. The reason they decide to use a generative approach is because identifying a suitable distance between deep features is very challenging. The model consists of a summariser network and a discriminator network. The summariser is an auto-encoder LSTM network, aimed to reconstruct the selected key frames, where learning is achieved by minimizing the negative log-likelihood of the data distribution. The discriminator network is another LSTM model aimed at distinguishing the original video from the reconstructed video. This means that instead of using the normal Euclidean distance to find reconstruction errors, they use the discriminator to find the error instead. The summariser and the discriminator is jointly trained so as to maximally confuse the discriminator. This means they wish to have a high error rate in recognising the original video from the reconstructed. They extract GoogLeNet features for each frame which are fed into the summariser model. The first stage these features go through is the selector LSTM which selects a subset of frames by a binary assignment. Thereafter, these subset of selected features go through an encoder LSTM which encodes the selected frames to deep features, e. The final component of the summariser network is the decoder-LSTM which takes the input and reconstructs the sequence of features corresponding to the input video. Then, the discriminator model comes into play, whereby it's tasked with discriminating between the original feature and the reconstructed feature and estimates a distance between these two. In a similar fashion to [52], they also use diversity regularization using DPP as one of the variants to their approach which enforces visual diversity amongst

selected frames. However, to compare their results with the supervised setting, they use *cross-entropy* for regularization. For evaluation they use the same key-shot based metric as mentioned above by [52]. They achieve very compelling results with F1-scores of 41.7% and 56.3% for SumMe and TvSum datasets, respectively.

Similar to [35; 24], [50] also used a encoder-decoder architecture. In this work, they aim to select meaningful summaries of videos by leveraging additional meta data-called side information, attached to the video such as the title, comments, click-rate, thumbnail, etc. The model introduced, Deep Side Semantic Embedding (DSSE), serves to correlate meta-data to visual data. They use two uni-modal autoencoders for accomplish the task of video summarisation. The first model dealing with the visual data from video frames which aims to reconstruct features. The second model deals with the "side information" which aims to learn semantic relevance. They argue that, by correlating the two uni-modal autoencoders via the latent subspace, the semantic relevance between video frames and side information can be more effectively managed [50]. The distance metric used that is minimised in the subspace is the L_2 distance. In order to generate a summary, they first run a segmentation algorithm to get video snippets. They then get snippet/shot-level semantic relevance scores by taking averages of each frame-level semantic relevance scores as opposed to frame level predictions as previously mentioned above since this model is a unsupervised setting. Thereafter, the results are pushed through the 0/1 knapsack dynamic algorithm to get the overall summary to a constrained budget. For the side information they use skip-thought vectors as sentence representations. The skip-thought model is trained on 11038 books from the BookCorpus dataset [56]. For visual features they extract AlexNet features as feature representations for each video frame. When training they use gradient decent with a learning rate of 0.0001 and a batch size of 128. The latent subspace dimension is 256 units for both models. They used the TVSum50 and Thumb1K datasets to train their model followed by mAP and F1-score to evaluate their results. They achieved a F1-score of 57.0% on the TvSum dataset.

One of the most unique methods for solving this problem was done by [54]. Here they proposed a reinforcement learning-based network where they designed their own reward function which judges how diverse and representative the generated summaries are. The model-called deep summarisation networks(DSN) proposed is fully unsupervised since no labels are required at all. They argue that supervised approaches do not fully capture the power of deep networks since there exists more than one ground truth due to the subjective nature of the problem. The DSN is a encoder-decoder model where

the encoder is a CNN structure used as a feature extractor, namely GoogLeNet. For the decoder, they employ a bi-directional LSTM network to capture temporal information topped with a fully-connected layer. The decoder network takes in the full visual features extracted. The fully-connected layer at the end of the decoder used a sigmoid function that predicts a probability for each frame being selected. The objective of DSN is to maximise the reward functions over time. The diversity reward, R_{div}, measures how dissimilar two frames are in the feature space and represents the mean pairwise dissimilarity. The representative reward R_{rep} measures how well the generated summary represents the original video. Thus, the total reward of the system is governed by $R = R_{div} + R_{rep}$. The aim is to learn a policy gradient that maximises the expected reward. Thus the objective would then be:

$$\nabla_w J(W) \approx 1/N \sum_{n=1}^{N} \sum_{t=1}^{T} R_n \nabla_w log \pi_w(a_t|h_t), \tag{3.6}$$

where π_w is the policy function with parameters w, a_t is the action at time t, h_t is the hidden state at time t and R_n is the reward function of the system. The gradient is approximated by running the agent N times and T represents total frames of the video. In order to generate summaries they segment their videos using KTS and select shots by maximising total scores and feeding it into the 0/1 knapsack algorithm to constrain the budget to be under a certain threshold. For training they use the stochastic gradient decent method. They down-sample to 2 fps similar to [52] and use 256 units in the hidden state of the RNN cell. The model is run for 60 epochs. When extending their method to supervised learning for better comparison, they use the Maximum likelihood Estimation (MLE) as the objective function. They achieve F-scores of 42.1% and 58.1% for SumMe and TvSum datasets respectively.

3.2 Transfer Learning Models/Architectures

Since 2012 CNNs have attracted a large amount of attention due to work done by [28]. They achieved top-1 and top-5 test set error rates of 37.5% and 17.0% respectively on the ILSVRC-2010. In the ImageNet Large Scale Visual Recognition Challenge(ILSVRC)-2012 they achieved a error rate of 15.3%. This was more than 10% lower than the runner up. A summary of their results can be seen in Table 3.1. This revolutionary algorithm is referred to as AlexNet.

Thanks to the ILSVRC [41] competition many models have been showcased and have excelled in this task. Along with this, most of these models have been made readily available for public use and in some

Model	Top-1(val)	Top-5(val)	Top-5(test)
SIFT + FVs	-	-	*26.2%*
1 CNN	40.7%	18.2%	-
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	-
7 CNNs*	36.7%	15.4%	**15.3%**

Table 3.1: Summary of a AlexNet results in the ILSVRC-2012 compared to the runner up who employed Scale Invariant Feature Transform(SIFT) with Fisher vectors(FVs). Models with an asterix are pre-trained models that were trained on the 2011 Fall ImageNet dataset.

cases in libraries for ease of access. These models include; AlexNet, ZFNet, GoogleNet, VGGNet and ResNet. A summary of how these models compare can be viewed in Table 3.2.

Year	Name	Architecture	Developed by	Ranking	Top-5 error
2012	AlexNet	CNN	[28]	1st	*15.3%*
2013	ZFNet	CNN	[51]	1st	*14.8%*
2014	GoogleNet/Inception	Inception-CNN	[45]	1st	*6.67%*
2014	VGGNet	CNN	[43]	2nd	*7.3%*
2015	ResNet	CNN	[19]	1st	*3.6%*

Table 3.2: A comparison of the top ILSVRC models [1].

As previously discussed we know that there are many pre-trained models that can be used for a plethora of domains. Having this tool available to us allows us to get creative with the way we feed features to our model. One of the most unexpected usages of the ImageNet challenge was that the deep architectures developed can be used for other domains [8]. This delves into the transfer learning domain discussed in section 2.2.1.

One important model we consider in this research for feature representation is the ResNet model introduced by [19]. In this research they construct a network having 152 layers with ReLU activations. This is approximately 8 times deeper than a VGG architecture but still has a lower complexity. They argue that it is easier to optimise the residual mapping as compared to the original mapping without the residual. They run extensive results on their model which indicate two very important aspects: 1) residual networks are easy to optimise; 2) the residual network can easily increase in accuracy with more depth. As mentioned previously, deep neural networks suffer from the vanishing and exploding gradient problem. This in turn limits how deep we can make our networks. With the inception of the ResNet model, it aims to address this problem. A residual block is formally defined as: $y = F(x, W_i) + x$, where x and y are inputs and outputs of the layer considered. $F(x, W_i)$ represents the residual mapping to be learned. The

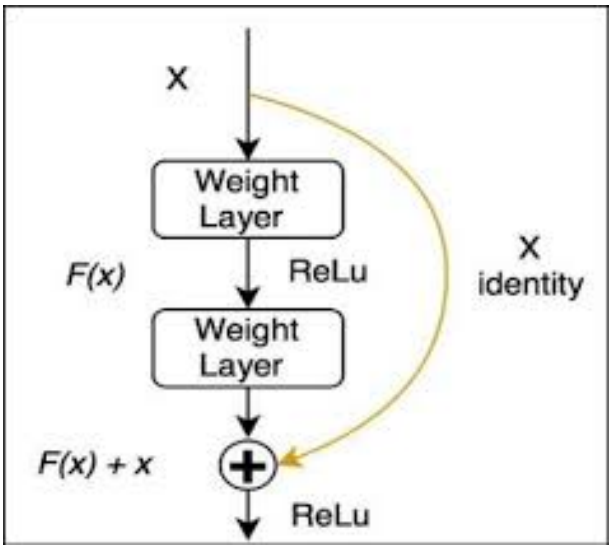


Figure 3.2: Residual blocks architecture proposed by [19].

idea is to have residual blocks stacked together to create the network, referred to as skip connections or short-cut connections. Skip connections take the input from one layer, lets call this layer n, and adds it to the linear combination of layer $n+2$ before applying the ReLU activation function as shown in the above equation. The construction of a residual block can be seen in Figure 3.2. Stacking many of these residual blocks together gives us a residual network. A 224×224 normalised image crop is fed into this network. They perform batch normalisation after each convolution operation. They use SGD with a mini-batch of 256. The learning rate is initialised to 0.1. They use a weight decay of 0.0001 and momentum on 0.9. This architecture was presented on ImageNet and at the time was the deepest network ever submitted. They test this model on numerous datasets which they achieved the best results for. This indicates strong evidence that the residual network is generic and has a good underlying feature representation of image data [19]. A summary of their results compared to previous results can be seen in Table 3.2.

However, these image-based deep features are not generally suitable for videos due to its lack of motion modelling [48]. This brings rise to the need for a generic video descriptor that help solve large scale video tasks. In [48] they use a 3D ConvNet to learn spatio-temporal features (known as C3D features/descriptors) trained on Sports-1M dataset [25]. This C3D network was tested on various benchmark datasets such as action recognition, action similarity labelling, scene classification and object recognition. They achieve state-of-the-art on all the dataset except action recognition. Their model consists of 8 convolutional layers, 5 pooling layers followed by 2 fully-connected layers and finally softmax output layer. Because they use 3D max pooling and convolution, they are able to capture spatio-temporal information while the counterpart of 2D operations are only done spatially. If we take 2D convolutions

on an image or a video the result gives an image which disregards the temporal integrity. However, if we consider 3D convolutions on a video volume the result is a video volume which preserves the spatio-temporal integrity. Once the model is trained, C3D can be used as a feature extractor for video tasks. These videos snippets, consisting of 16 frames, are passed into the C3D network to extract features from the fc6 layer, which is essentially the last layer before the output layer. These video descriptors consist of 4096 dimensions which need to be normalised before passing through a new model. The advantage 3D ConvNets has over 2D ConvNets is that it learns both motion and appearance features.

One architecture that also leverages spatio-temporal features and that has achieved very promising results with video processing (in particular action recognition) is the I3D network, originally introduced by [8]. They introduce a two-stream inflated 3D ConvNet (I3D). This model is based on 2D ConvNet inflation. That is, the filters and pooling kernels are expanded into 3D which allow the network to capture spatio-temporal features. In this work they show 3D ConvNets can directly learn from temporal patterns from the RGB stream however, the overall performance of their model can still be improved by adding optical flow which they state are in some sense recurrent. Hence, the two-stream network. The advantage of this model in particular is that using optical flow as well as raw RGB pixels takes advantage of spatio-temporal features which is essential when working with video data. They inflate 2D ConvNets by inflating the pooling and filter kernels from square to cubic. This adds the temporal dimension. Additionally, they also bootstrap 3D filters from 2D filters by creating a "boring" video by copying an image repeatedly into a video volume. This leads to the 3D network being able to be pre-trained on ImageNet. They run the two networks (RGB and optical flow) separately and average predictions at test time. These two models use ImageNet pre-trained Inception as the base networks. This means that they do not come up with a new deep architecture for their streams, instead they build upon existing state-of-the-art architectures. All convolutional layers are followed by a batch normalisation and the ReLU activation function. They train their networks using a standard SGD with momentum of 0.9. They train the models on the Kinetics dataset for 110k steps. During training they perform data augmentation which is critical for performance of deep architectures. This is done by cropping both spatially (by taking random 224×224 patch of the image) and temporally (picking the starting frame). During testing they take a 224×224 center crops and the predictions are averaged out. They argue that videos that has more camera motion makes it harder to capture the motion stream. Thus, optical flow applied to videos with such properties will have lower accuracies compared to videos that do not have much camera motion. A high-level representation of this model can be seen in Figure 3.3

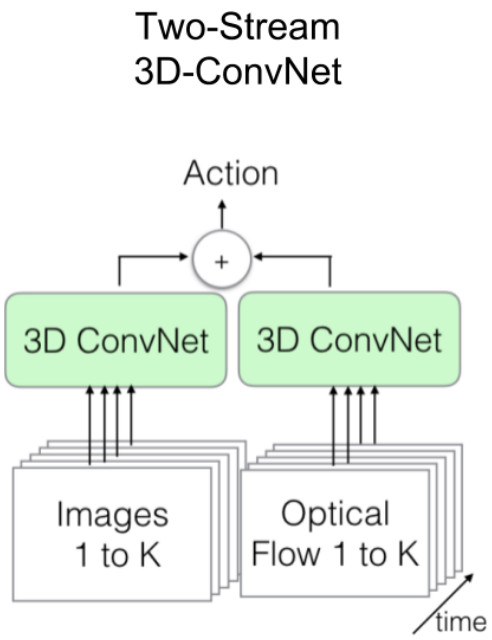


Figure 3.3: A high-level overview of the I3D network which was proposed by [8].

These above-mentioned pre-trained models can be used in a number of ways. One way is to use the trained model weights, detach the output layer and replace it will an output layer that suits your task i.e. if you have 5 classes to predict then you will add 5 neurons as the output layer with a softmax activation function. After doing this you will "fine-tune" the network to adjust the weights matrices to suit your data. Another way is, after fine tuning or even without fine-tuning, you would feed a image to the network and you would take the output of the second last layer as a feature representation for that image. These feature representations of all the images can now be used as features to train a new, usually smaller, network.

3.2.1 Transfer Learning Used Previously

In the realm of video summarisation, [49] extracted C3D features at the fc6 layer for their feature representation. After some processing they feed the processed C3D feature vectors to their auto-encoder. [50] used AlexNet to generate their image representation which was extracted at the fc7 layer of the network. These features are then fed into one of the uni-modal auto-encoders used. Many other papers [52; 49; 24; 35; 54] also extract GoogLeNet features as image representations and feed this to their models. This shows the advantage of pre-trained models on large datasets which can be transferred across different problem domains.

3.3 Conclusion

As shown in this brief survey there has been a great deal of work done in the realm of visual learning for machines. There has been many different approaches taken in order to solve the problem of video summarisation. Other advances in video processing might prove to be useful in the future of tackling the video summarisation task. The approaches adopted in this survey use spatial deep features in order to solve the video summarisation problem. Previous approaches dealt with temporal information in the architecture by introducing some form of recurrence. In this work, we address the need to consider the temporal aspect at a lower level inherent in video structures, along with the spatial information on a frame level. In this way, we model both appearance (spatial) and motion (temporal) information.

One key aspect when dealing with video sequences is the need to retain temporal information. The need to retain long-term dependencies is one of the main challenges in deep-learning. A few methods exist that address this problem such as LSTM, GRU and so on.

The advantage of using pre-trained networks is made clear in the above literature survey. It shows the power of re-using models for a similar problem domain. Apart from its ability to generalise image representations well, it saves us time and cost of training these large networks from scratch.

There is still however a need for a robust system which can automatically summarise a video. A problem faced when employing hand-crafted features is that they might not generalise well between datasets but can perform well on a well-defined problem set. One could use transfer learning to get good generalisation of feature representations by using pre-trained models such as the above-mentioned architectures. The problem of video summarisation can thus be addressed by using a combination of deep-features and/or hand-crafted features.

Based on the literature we will be following the work done by [52] very closely since this serves as a benchmark for many of the previous research done in video summarisation. However, we will use the I3D network to extract spatio-temporal information instead of the GoogLeNet network since this allows us to capture the temporal information as well. Our network follows [37; 52; 49; 24] in that it is a encoder-decoder network with convolutional and recurrent layers. The main aspect however was inspired by [50] to use the code layer for further processing. This gives us a good representation of the most salient features in our data. All post processing of the final predictions will follow [52] so that we are able to compare our results to previous methods.

Chapter 4

Research Method

4.1 Introduction

In Chapter 3, we discussed some techniques which we will be using to solve the video summarisation problem. The best approaches which require the least amount of preprocessing are deep learning, since deep feature representations are learned automatically. This chapter aims to address a baseline method which we will compare our model against, as well as the techniques that will be employed for the this research in order to solve this problem.

4.2 Research Motivation

Manual video summarisation can be a very laborious task which can be automated with the help of some machine learning models. Solving this particular problem will be very desirable to many fields that have some sort of video records, usually companies that keep a large amount of video data. This task will enable faster browsing of large video sets, as well as better grouping and access to these videos.

4.3 Research Hypothesis

Augmenting deep learnt features together with high-level spatial and temporal deep feature representations from pre-trained models will improve the video summarisation performance of the deep learning-based video summarisation algorithms employed, which aims to be near human-level accuracies.

4.4 Research Method

This section covers the bulk of the time spent to finish the project. It includes the collection of the datasets we will be evaluating on, the approach taken to solve the problem, implementing it, and testing it on the various benchmark datasets.

4.4.1 Hardware and Software

Software

All model development and processing is done in Python. Models were built with the help of Keras using the Tensorflow backend, a library available in Python. Libraries such as NumPy, pandas, SciPy, and scikit-learn were all used as well while developing the models. To evaluate our models on the TvSum50 dataset we use MATLAB for convenience.

Hardware

The computer machine consists of:

- 1 2080TI RTX Nvidia GPU
- AMD RYZEN THREADRIPPER 2950X 3.5GHZ 16-CORE 32 threads
- 128GB - DDR5 RAM
- Asus ROG ZENITH EXTREME; TR4 THREADRIPPER; X399

4.4.2 Proposed Model

The process flow of how this problem task was solved is shown in Figure 4.1. It should be noted that we have two streams which consist of the RGB stream, and the flow stream. Section 1 in Figure 4.1 deals with reading in the video files, processing it to suit our needs, and finally saving the processed video files. Further details of this section can be viewed in Chapter 4.4.4. Section 2 deals with reading in the processed video files, splitting it to training and testing, and preparing our training data to feed into our pre-trained models. Features are extracted for both RGB and flow for all videos and saved. We

then propose our SummaryNet model which is two-fold: we feed our first model all the training feature representation. Latent variables are used as features for our second model. Once trained, we get frame-level scores on our testing data. These steps are discussed in more detail in Chapters 4.4.5, 4.4.6, 4.4.7. Section 3 combines the RGB and flow streams together which is commonly known as late fusion. Some post processing is then applied to follow the same convention as [52]. Thereafter, we can evaluate the system against the ground truths. Further details of how this process can be seen in Chapter 4.4.8.

4.4.3 Datasets

The video summarisation datasets that is used are publicly available. For this research, two datasets will be used, namely the **SumMe**[18] dataset and the **TVSum**[44] dataset, mentioned in Chapter 2. The reason for using these two datasets is because we wish to accurately measure the performance of our system against previous methods that used only these two datasets for the canonical settings (80% training and 20% testing split of the dataset) such as [24; 54; 52; 36; 37].

SumMe Dataset

This dataset was originally introduced by [18]. It consists of 25 videos in both .mp4 and .webm formats. All videos differ in resolution and duration, ranging between 1-6 minutes. Scenes in these videos generally change slowly. Each video is accompanied by a .mat file which has some meta-data about the video. The additional information includes frames per second, number of frames in the video sequence, and video duration. Additionally, each video has been manually annotated by 15-18 different users. That is, each user indicated snippets that they believed were worthy of being part of the summary. From this, the ground truth and user score for each frame was computed, and included as part of the dataset. The types of camera positions of the videos are either moving, egocentric, or static, with the bulk of the videos taken with a moving camera.

In Figure 4.2 we see some examples of the type of videos that are in this dataset. Each horizontal row represents one video where frames from different points in the video were manually captured to show a key-shot summary of the videos. The first row i.e. Figure 4.2a-Figure 4.2e is titled “air-force plane landing”, the second row, Figure 4.2f-Figure 4.2j is titled “bus in rock tunnel” and the third row, Figure 4.2k-Figure 4.2o is titled “jump”. These titles, along with the summary frames shown, give us an idea of what the videos are about, as well as what kind of videos are in this dataset.

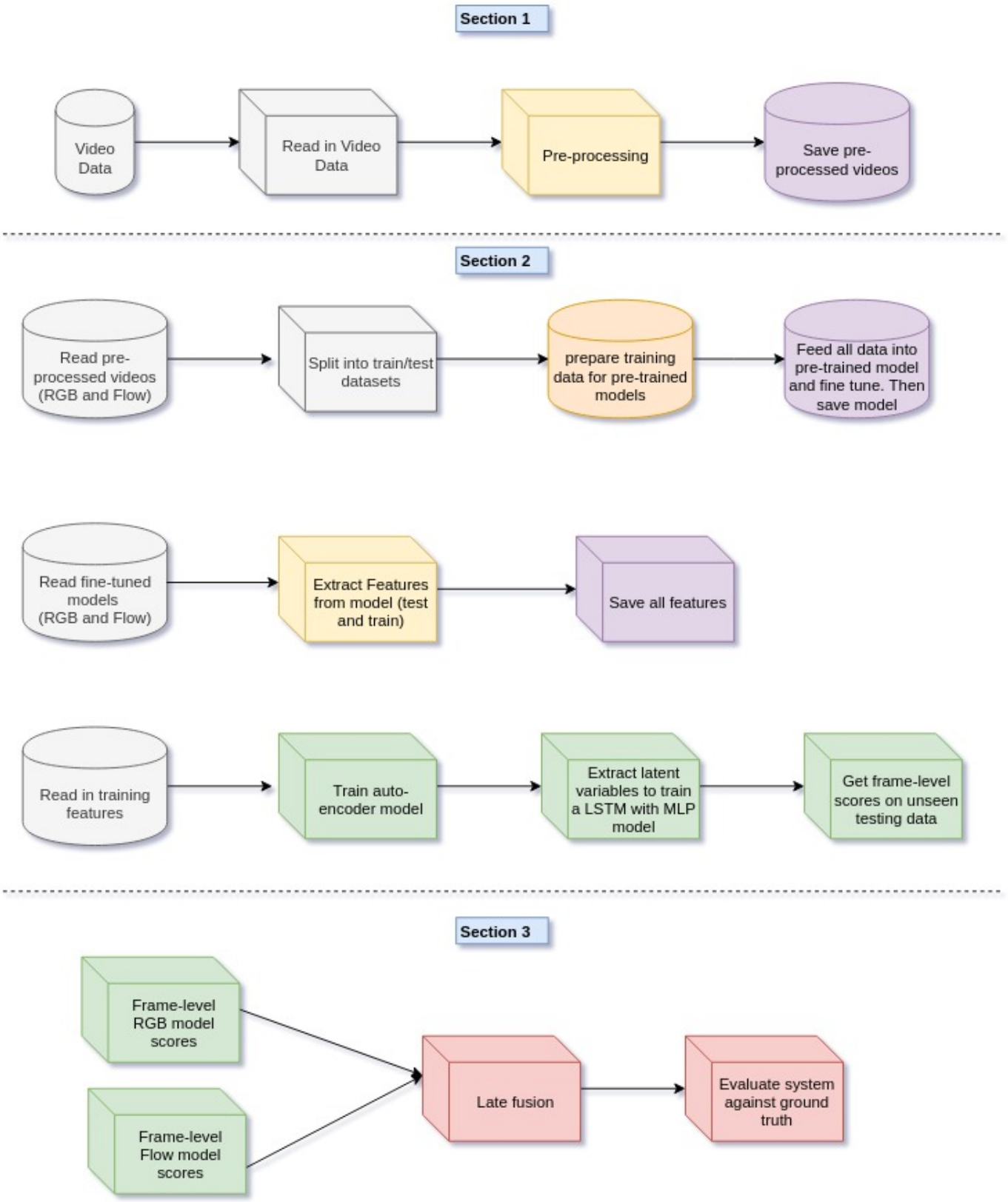


Figure 4.1: The process flow of the research. There are three sections to guide this research. Section 1 involves reading the video data and processing. Section 2 is using the newly processed videos and feeding it into the models and evaluating (SummaryNet model). Section 3 deals with combining two models together to create a final result.

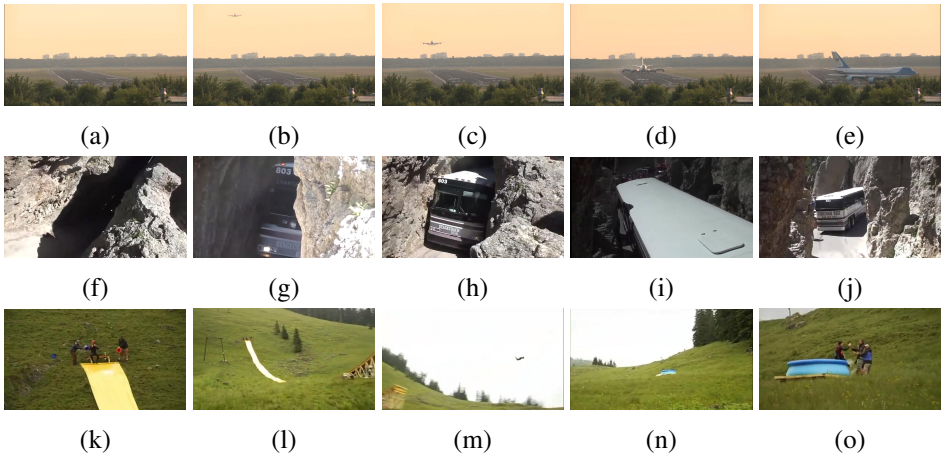


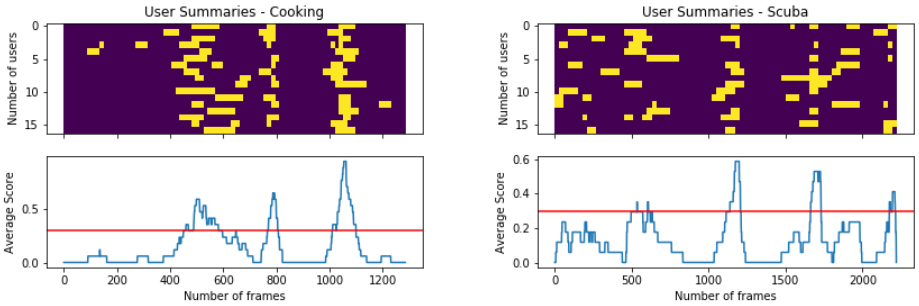
Figure 4.2: Various examples of the SumMe dataset. These frames were captured manually to show the different content in each video.

In Figure 4.3, the titles represent which videos we are plotting from our dataset. We plot the summaries of each user to the number of frames per video which can be seen by the yellow horizontal markings on the first plot. In the plot below we sketch the average user summaries for each frame against the number of frames. The red horizontal line denotes the 30% threshold found empirically. This line can be adjusted to suit our needs i.e. anything above the line we construct can be labelled as a target of 1, indicating it is part of a summary, and every frame below the line will be labelled 0, indicating that it is not part of the video summary.

TvSum50 Dataset

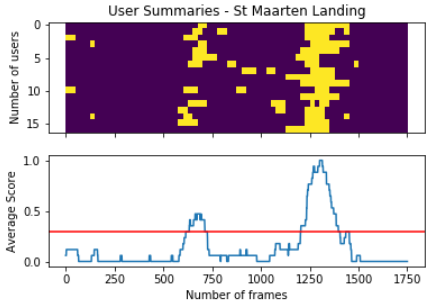
This dataset was originally introduced by [44]. It consists of 50 videos all in .mp4 format. The videos differ in resolution and duration, ranging between 1-11 minutes. This dataset has 10 different categories, i.e. 5 videos for each category. The dataset is accompanied by two .tsv files which has information about each video. In the first file, the video category, video name, video title, source url, and video duration are included. In the second file, we are given 20 different people that annotated each video file. We are given the video name, video category, and the importance score. The importance score ranges between 1-5, with 5 being very important, and 1 being relatively important. Users gave a single importance score for every 2 seconds of the video, which means every frame between the two importance scores will have the same value.

In Figure 4.4, we see an example of the TVSum dataset, these videos supplied come with an accompanying video thumbnail to represent the video title, seen in Figure 4.4f.



(a)

(b)



(c)

Figure 4.3: These are examples from the SumMe dataset. The summaries for each user can be seen in the top plots while the average scores for each user summary can be seen in the bottom plots.



(a) (b) (c) (d) (e)



(f) Thumbnail image supplied referring video title “animal grooming” 4.4(a)-(e)

Figure 4.4: Example of TVSum dataset under the category animal grooming.

In Table 4.1, we can get a comprehensive summary of the two datasets. This information allows us to gauge the differences and similarities between the two datasets considered.

Dataset	#Video	Resolution	#User Annotations	Duration(Mins)	Annotations	Categories
SumMe	25	Varies per video	15-17	1-6	Frame-level importance scores	All different
TvSum	50	Varies per video	20	1-11	Frame-level importance scores	10 (5 for each category)

Table 4.1: Descriptive Information on the Datasets.

4.4.4 Pre-processing

The pre-processing mentioned in Section 1 of Figure 4.1 consists of a few stages. Firstly, we read the .mp4 files into the system memory as RGB images for each frame. We then resize each frame to $n \times m \times 3$. This corresponds to the height, width, and colour channels for each frame. This makes all of our videos of the same frame dimensions. While resizing we preserve the aspect ratio of the video. That is, the smallest side between the width and the height is resized to 128 pixels, and the remaining side is usually greater than 128 pixels.

In Section 2 of Figure 4.1, the computation is two-fold. On one branch, we save these resized videos as NumPy files, which represents the RGB pixels. On the other branch we compute the dense optical flow for each video in the dataset, and we save them to disk as NumPy files as well. The optical flow is computed in order to model temporal dependencies inherent in the videos.

Before we train our model, in the case of the RGB saved videos, we read the RGB NumPy files in and we normalise them by dividing each frame by 255 since all pixels are integers between 0 and 255. We then crop 5 random parts of each frame of size $112 \times 112 \times 3$ to augment our data, and we create cubes of length 16 to feed into the network. Since we are using the I3D network as a feature extractor we use 16 frames per cube because this is what is required as input to the network. The reason for selecting 5 frames per cube is discussed in Chapter 5.3. Thus, we end up with cubes, each of size $16 \times 112 \times 112 \times 3$. A step size of 1 frame is used, and the target for each cube is the target associated with the middle frame of the cube. Meaning that if we have a cube of length 16, the target that is assigned to the frame at position 8 is the target for the entire cube. This procedure described can be seen diagrammatically in Figure 4.5. At this point we randomly sample a percentage of our total data from each video for training since all the data cannot fit into memory at once. The aforementioned process described is used for the optical flow videos as well, the only difference is that we normalise using min-max normalisation, since there is no predefined finite range for optical flow values. This depicts how we prepare our data to train our models.

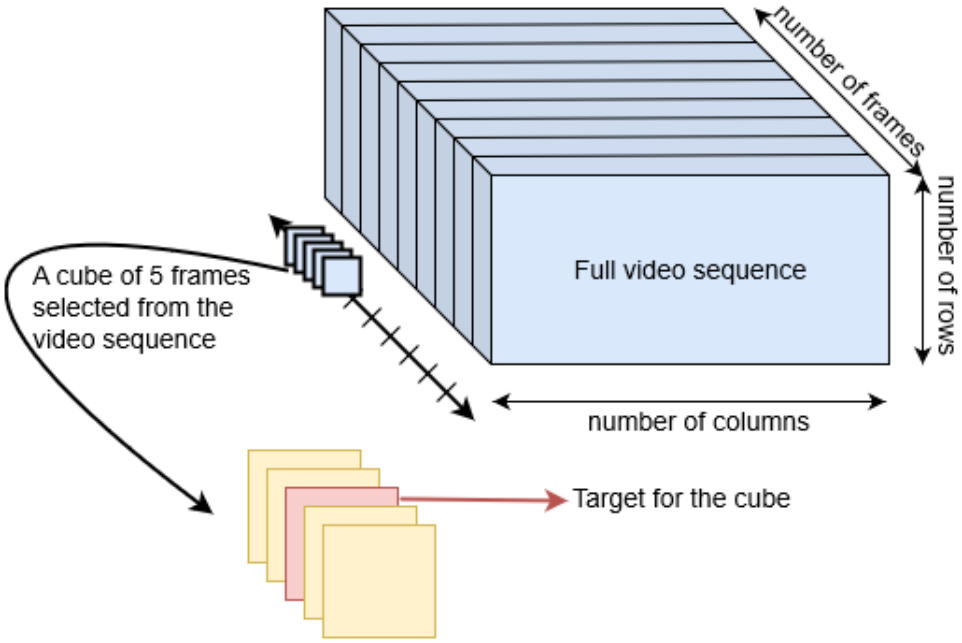


Figure 4.5: The target associated with the middle frame of the current video cube represents the target for the entire cube.

4.4.5 Pre-trained Models/Features

This stage was the most vital step for the success of this project. It set the standard on what the model would learn, and how well it would perform in the testing environment. This stage directly effected the accuracies gathered during the testing phase. The features are a combination of spatio-temporal features by means of deep features.

In this dissertation, we compare using ResNet features with I3D features. Using ResNet features aims to serve as a baseline for our model. By using I3D features, we aim to demonstrate the importance of using temporal features along with spatial features to improve the overall performance. Since we aimed to follow a similar approach to [52] we create a baseline method and compare it to our approach to demonstrate its advantages. However, for our baseline we decided to use ResNet features instead of GoogleNet features as done in [52]. The reason for this is shown in Table 3.2 where ResNet achieved a top-5 error of 3.6% compared to GoogleNet that achieved a top-5 error of 6.67%.

ResNet Features - The model weights are initialised with the ImageNet features available in the Keras library. We fine-tune the model by adding our own classification layer and removing the existing one. Once the model is fine-tuned on a sample of our data, we take the outputs of the second last layer in this network as feature representation of each individual frame. These vector features now represent

each associated frame. Thus, we can define a mapping for this process as $f : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{j}$, where f is a function mapping the image frame $\mathbb{R}^{m \times n}$ of dimension $m \times n$ to a vector $\mathbb{R}^{j}$, where j refers to the number of neurons in the second last layer of the network.

I3D features - Similarly to Resnet, this model is also fine-tuned and used as a feature extractor. This architecture consists of two streams as mentioned in Chapter 2, namely the RGB stream and the dense optical flow stream. This architecture is fine-tuned on sub-samples of our data and then used as a feature extractor to get a high-level feature representation of our data.

Since the I3D features is what we aim to exploit we will delve deeper into this. Once the cubes are created as mentioned in Chapter 4.4.4, we feed our data into the I3D network which consists of two separate networks, one for the RGB data and one for the optical flow data. Thus we fine-tune these two networks by removing the previous classification layer and adding our own sigmoid layer for classification. Each snippet is given the value of 1 or 0 depending on weather it is a summary or not. The weights used as a starting point on the I3D models is the ImageNet and Kinetics available online [7].

The aforementioned process takes the most amount of computing time because of the size of the training data. Once it is completed, we feed all our videos through these networks so as to get feature representations for each frame. This process involves creating cubes as mentioned above with a stride of 1 frame. We pass all these through the network and take the features in the second last layer of the network which is of size 1024. This vector serves as a feature representation for the middle frame of the cubes we created. To view this mathematically for an entire video we get; $\mathbb{R}^{N \times 112 \times 112 \times 3} \rightarrow \mathbb{R}^{M \times 1024}$ where N represents the number of frames in a video and M is the resulting number of features $M < N$ since we apply no wrapper for the edges of the video. Furthermore, we can see the mapping for each individual cube as, $\mathbb{R}^{n \times 112 \times 112 \times 3} \rightarrow \mathbb{R}^{1 \times 1024}$, where n is the size of each cube. Note that this process is done for both the RGB stream and the flow stream.

When using the feature representations we extracted, we use the *MinMaxScaler* available in the scikit-learn library to normalise the features. It is normalised in a similar fashion shown in Equation (2.1).

These two models above (RGB and flow) are trained for 8 epochs each, and the best model parameters are used. This is selected by which epoch returns the lowest loss value for validation and these are taken as the optimal parameters of the network.

4.4.6 SummaryNet Model

This included designing and implementing an appropriate model for the system to learn how to summarise videos automatically, and be as general as possible. The features extracted, as described above, will be fed into the model.

The SummaryNet approach we have decided on pursuing is two-fold. The idea was to train our own **encoder-decoder** model offline (phase 1), which is used as a features to our **LSTM-MLP** model (phase 2). This involves dividing the full bank of features into smaller feature snippets and feeding it through the model, similar to the process described in Figure 4.5. The encoder part consists of 3-hidden-layers, 2 being densely connected and the last one before the code layer is a convolution layer. The decoder leveraged 2 hidden-layers densely connected. After the encoder-decoder model is trained, we use the encoder part as a feature extractors for each video snippet which is of size $n \times 512$ where n is the length of the cube. We then feed these feature representations, or code layers, into a newly trained model i.e. **LSTM-MLP** model.

This model consists of 5 hidden layers. The first of these hidden layers is a LSTM layer which aims to map long-term dependencies. The second and fifth layers consist of convolutional layers. Hidden layers 3 and 4 consist of MLPs, each with 256 units and sigmoid activations, as done in [52] (apart from the final layer which is 1 sigmoid unit which is to classify weather a snippet is part of the summary or not). The full flow of the model architecture can be seen in Figure 4.6.

4.4.7 Training

Since the datasets have multiple user annotations, we process this data to give us a single ground truth vector of size $1 \times n$ where n is the number of frames per video. Along with the multiple user annotations, we are supplied with single frame-level scores for each video which is the average score of all users. We formulate this task as a binary classification problem, i.e. if a certain number of x users regard frame m as a summary then we give the value of 1 else the value is 0. We call this process binarising the video targets.

At this point, the dataset, features, models, and targets have been prepared so that we may train our model. For each dataset, there is a split between training and testing which consists of a 80% training and 20% testing. Back-propagation with the Adam optimizer is used with the default Keras settings to

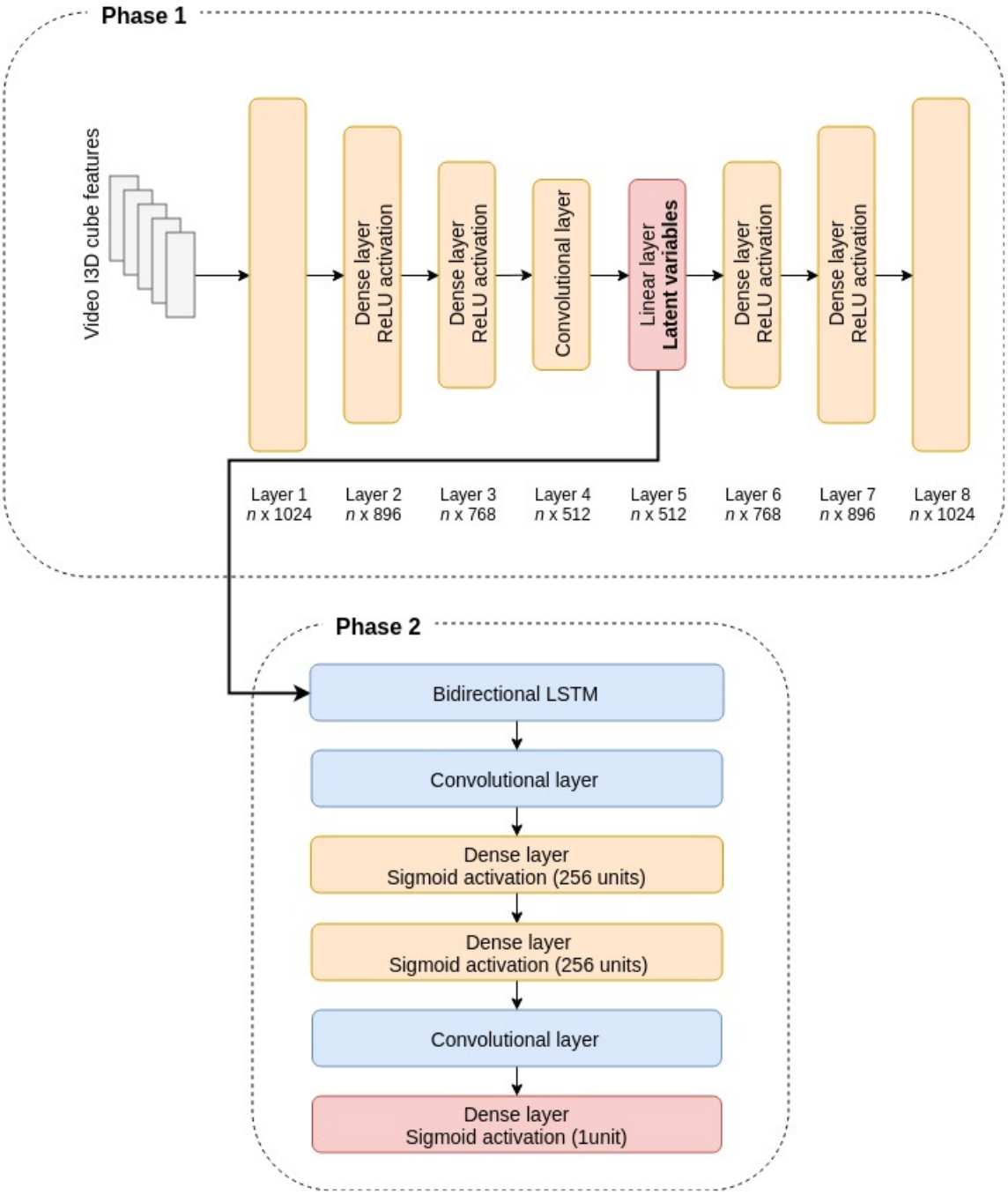


Figure 4.6: Our SummaryNet model is two-fold. Firstly, an encoder-decoder model is trained and the latent variable are extracted. Secondly, the latent variables are used as features to the second LSTM-MLP model. The second model is responsible for predicting the frame-level scores.

minimize the loss function with a learning rate of θ and a batch size of 8. Since we have two classes we will be using binary cross-entropy as our loss function. This step takes around 2 hours to train which is reasonable since the models are of a deep nature.

When training, we sub-sample 2fps as done in [52], since the scenes are slowly changing in the datasets

considered. The training process is ran for 8 epochs and the best model based on validation loss is selected.

4.4.8 Testing and Evaluation Metrics

The system is tested to evaluate the summary results produced by the model. As mentioned above there is a hold out set which is data the model has not seen before to gauge the system's performance. Since previous approaches performed a 80-20 split [24; 54; 52; 36; 37; 35] we also opt for this same split for fair comparison. The testing of the system has been constructed to follow [52] as close as possible.

To test the system, we feed in both the RGB and flow feature representations. We create the snippets (n number of frames stacked together) of 5 frames in length, i.e. we end up with snippets of size $\mathbb{R}^{5\times1024}$. The model predicts the value for the middle frame of the snippet, we also take a stride of 1 frame when creating these snippets so that we may get predictions for each individual frame.

Once the model predicts the frame-level scores, we do some post-processing. Thereafter, it is evaluated against previous results. The model is run 5 times and the average is reported which is consistent with [52].

Post-processing

Since our model predicts on a per-frame basis, we need to extract snippets from the video which gives the highest value so that we can include it in the final summary. Hence, we are selecting key-shots/snippets as opposed to selecting key-frames.

The first stage is to segment the temporal structure of the video, to do this we use the KTS algorithm discussed in Chapter 2. The chosen algorithm groups frames of similar content together [52].

Once we have these snippets of no more than 5 seconds, we employ the same method used by [52] to rank each snippet based on their scores. Thus, we need to move from our frame-level scores to key-shots. To do this, we compute interval-level scores by averaging the individual scores of each frame within each interval. Then we rank all these intervals in descending order based on these interval scores.

Lastly, we aim to select key intervals as the final summary so that it is less than a certain threshold. We choose this threshold to be 15% of the original video as done by [35; 52; 18; 44; 24; 54]. To do this we

adopt the 0/1 knapsack algorithm discussed in Chapter 2.

Evaluation

The model's performance can be evaluated using one of two ways. The two common metrics used in literature is the mean average precision (mAP) and the F1-score, which are defined as:

$$mAP = \frac{1}{|Q|}\sum_{j=1}^{|Q|}\frac{1}{m_j}\sum_{i=1}^{m_j} Prec(i) \times relevance(i), \tag{4.1}$$

where $|Q|$ is a query set, m_j is the total number of relevant frames(positives), $Prec(i)$ is the precision at the the top i frames, $relevance(i)$ is 1 if relevant and 0 otherwise.

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}, \tag{4.2}$$

$precision$ is the percentage of positive predictions that were correct, $recall$ is how well we find our all positive predictions:

$$Precision = \frac{\textit{Overlap duration of S and GT}}{\textit{duration of S}}, \tag{4.3}$$

$$Recall = \frac{\textit{Overlap duration of S and GT}}{\textit{duration of GT}}, \tag{4.4}$$

where S is the summary produced by our model and GT is the ground truth summary. In Equation (4.3) and Equation (4.4), we calculate the temporal overlap between the models predictions and the ground truth. Since each frame has an associated target with it, there exists some temporal information which we leverage to evaluate our models.

For the evaluation, because both dataset have multiple user annotations, we pre-process the targets to be one single ground truth per video, as done in previous works [52; 35; 24; 54]. It is taken by threshold by some value σ to binarise the targets to be 0 or 1, depending on whether it is a summary frame or not. The probabilities produced by the model is then processed as well, whereby the 85^{th} percentile of values are taken from the model output. These values are then binarised. This means the only 15% of the total output is considered to ensure that we end up with a video summary.

4.4.9 Hyper-parameter Optimisation

The parameters of the I3D network is left as default as recommended by the authors [8]. Since this network is used as a feature extractor in our model and not our final model we did not pursue further investigation into this.

For our SummaryNet model the parameters are set to that setting which minimises the validation set loss for both the RGB and the flow streams. Since we are using a encoder-decoder structure as the first part of our model, we selected the parameters that achieve the lowest mean squared error on the validation set. For the second part of our model which uses the latent variables from the encoder-decoder structure, we select the parameters that achieved the lowest binary cross-entropy loss for the validation set.

The size of the cubes (number for frames for each cube) is selected using taking 4 different window sizes and the optimal number of frames per cube is selected by taking that setting that achieved the highest test accuracy score.

4.4.10 Baseline Model

In this research we first develop a baseline which we evaluate against. This baseline uses feature representations which we extract for each video frame. The model consists of 2 hidden layers with 256 units, both with sigmoid activation functions. The output layer is a single unit with a sigmoid activation for classification. The model uses a batch size of 8, binary cross-entropy as the loss function and the Adam algorithm as the optimiser. We also sub-sample 2 fps, as mentioned previously.

We opted for such a baseline to show the power of our proposed SummaryNet approach. This baseline model was developed to the specifications mentioned in [52] so as to give us a good benchmark. These baseline methods has also been shown as a first pass approach. From this we were able to set these simple models as benchmarks while developing newer/more complex models to evaluate against. We show the various advantages SummaryNet has compared to the simple 2 hidden layer network.

4.5 Conclusion

In this chapter the methodology adopted was discussed. The first and biggest phase was the actual implementation of the project which involved data collection, feature extraction, and model building. All

this was done in Python using the Tensorflow library and associated application programming interfaces (APIs). Thereafter, the models were trained and evaluated to gauge the performance of the system. This was iterative process to get a good combination of the hyper-parameters to work well with the model. We also discussed the testing and evaluation procedure that we followed so that we may compare results to previous work. Lastly, we discussed our baseline model that we will be using to also evaluate our results against.

Chapter 5

Experimental Results

5.1 Introduction

In the previous section we discussed our hypothesis, research motivation, and the proposed method. Along with these topics, we also discussed datasets we will be using as well as how we will be training and testing our system. In order to prove or negate our hypothesis, we run some experimental results on our data using our proposed method. In this section, we will discuss the network architecture we have decided to employ, previous results, insights, and finally, some experimental results achieved.

Method	**SumMe**	**TvSum50**
[35]	41.7%	56.3%
[50]	-	57%
[52]	38.6%	54.7%
[24]	44.4%	**61%**
[54]	42.1%	58.1%
[37]	**47.5%**	56.8%
[36]	40.1%	56.3%
[18]	39.7%	-
[44]	-	51.3

Table 5.1: Results of previous works done for the video summarisation task. State-of-the-art results are shown in bold.

5.2 Previous Results

From Table 5.1, we see results of some previous work that was done by various authors for the video summarisation task. We see that [24] and [37] achieved state-of-the-art results previously in the TvSum50 and SumMe datasets, respectively. [54] comes a close runner-up by introducing their own reward functions and formulating the problem as a reinforcement learning problem. All the previous results shown in this table use a deep architecture, as well as deep spatial features (mostly GoogLeNet features).

5.3 Hyper-parameters

In Table 5.2 and Table 5.3 we present succinct tables of our SummaryNet model results on different window sizes. This is also the only hyper-parameter we need to optimise in our model. We notice that having the window size of 5 achieves the best results in terms of the test accuracy. This applies to both the SumMe and TvSum50 datasets. All settings are ran identically apart from the window size that was changed.

Since the model achieves the best results for a window length on 5 we will be adopting this setting in all our comparison test to our baseline models. This means that all the cubes created will have a total of 5 frames which constitutes 1 sample. The weights of our SummaryNet model are selected by taking the model that achieves the minimum validation loss.

As mentioned previously the default settings are used for the I3D network, which is used as our feature extractor. This setting is recommended by the authors [8].

The training accuracy in these tables are included for completeness. During training, each cube of frames has a binary target of 0 or 1, since we formulate this as a sequence labelling task, which indicates if the center frame is a summary of not. This means that for a cube size of 5, there are 2 frames on either side of the center frame that the target is associated with. The weights are then updated accordingly based on the binary cross-entropy loss. During testing, the model predicts values for each frame in the video sequence. After this, the predictions go through a post-processing stage which follows the literature of [24; 54; 52; 36; 37] and is described in Chapter 4.4.8. This means that the testing accuracy differs from the training accuracy.

When increasing the cube lengths it results in less samples since the cube lengths are larger. In Table

Method	Cube Size	Training					Testing		
		#Inputs Cubes	Val Loss	Train Loss	Train Acc	Time (s)	#Input Cubes	Acc	Time (s)
SummaryNet	**5**	6264	0.6116	0.6024	0.7096	870.47	24820	**44.6%**	262.88
	11	6144	0.6654	0.5964	0.7083	1141.78	24790	41.77%	273.71
	31	5744	0.6369	0.6034	0.7069	2397.59	24690	40.02%	372.66
	61	5144	0.7416	0.5729	0.7224	4138.44	24540	41.35%	509.56

Table 5.2: Succinct table of hyper-parameter selection for the window size of our SummaryNet model. The values in this table correspond to the SumMe dataset. The best result is shown in bold.

Method	Cube Size	Training					Testing		
		#Inputs Cubes	Val Loss	Train Loss	Train Acc	Time (s)	#Input Cubes	Acc	Time (s)
SummaryNet	**5**	21186	0.6215	0.5754	0.7007	2599.2	51378	**73.2%**	460.58
	11	20946	0.7080	0.5503	0.7223	4043.68	51318	70.7%	516.79
	31	20146	0.7114	0.596	0.6841	8342.28	51118	71.1%	697.73
	61	18946	0.8072	0.6741	0.5823	15035.3	50818	71.2%	976.85

Table 5.3: Succinct table of hyper-parameter selection for the window size of our SummaryNet model. The values in this table correspond to the TvSum50 dataset. The best result is shown in bold.

5.2 and Table 5.3 we also see that by increasing the cube lengths, the time taken for training and testing increases accordingly.

5.4 Experimental Results

In Figure 5.1 - Figure 5.4, we see graph plots that show the average user scores for all the frames in the video. Along with the user scores, there are red rectangles to signify where the model predicted summary frames. Below these graphs we see the first and last frames in the video. We also see the first and last frames for each snippet from the original video which were classified as summaries by the model, shown chronologically. In these figures, snippet $x - m$ represents a key shot consisting of frames starting from x and goes until frame m taken from the original video.

5.4.1 TvSum Dataset

In [24] they randomly leave 20% for testing and the remaining 80% for testing after which they report their results. A similar procedure is followed in [54; 52; 36; 37; 35]. We then opt for the same dataset split, 80% used for training and 20% used for testing, for fair comparison. Thus, since the dataset consists of 50 videos in total, we use 10 videos from each category selected randomly as our testing videos. A

Video Name	Category	FPS	Length
98MoyGZKHXc	VT	25	187.524s
HT5vyqe0Xaw	VU	30	322.711s
0tmA_C6XwfM	GA	25	141.293s
37rzWOQsNIw	MS	30	191.611s
GsAD1KT1xo8	PK	30	145.357s
4wU_LUjG5Ic	PR	24	167.044s
JKpqYvAdIsw	FM	25	152.091s
EE-bNr36nyA	BK	30	98.151s
EYqVtI9YWJA	BT	30	198.182s
-esJrBWj2d8	DS	30	230.412s

Table 5.4: List of testing videos from the TvSum50 dataset used to evaluate the systems performance. The FPS column has been rounded off to the nearest integer for brevity.

list of these videos with their descriptions can be seen in Table 5.4. For more information on this dataset please refer back to Chapter 4.4.3. The procedure for evaluating our system is as follows. We first create snippets of 16 frames throughout the videos for each video. That is, for both the RGB and flow features. These snippets are used to fine-tune the I3D network. We then extract features from the second last layer of the I3D network (the layer before the classification layer) as a feature representation for each frame. These features are fed into our SummaryNet architecture described in Chapter 4. Since the TvSum50 dataset gives us 5 videos for each category we can expect relatively high results compared to that of the SumMe dataset.

Method	Features	Results - RGB	Results - FLOW	Final Results
Baseline	ResNet	-	-	50.48%
Baseline	I3D	67.3%	67.28%	68.64%
SummaryNet	I3D	70.06%	**73.2%**	73.18%

Table 5.5: Results obtained for the TvSum50 dataset using our proposed method. These results clearly show the advantage of using spatio-temporal features. The best result is shown in bold.

In Table 5.5, we see the results of baseline-ResNet is more than 3% lower than the results obtained by [52], who used GoogLeNet features with a similar model architecture. In this table, we also clearly see the advantage of using spatio-temporal features. Since we are using a two-stream network, we get two separate predictions for a video. The results are combined and the average is taken as the final prediction. These scores are represented by the last column in the table. Our proposed method, SummaryNet is shown in the last row of Table 5.5. This method clearly shows significant improvement in the overall results when compared to the baseline methods. This could possibly be due to the fact that we have 4 videos for each category used in training which makes it easier for the model to learn how to summarise

a particular category. Another possibility is that most of the videos are slowly changing, this means that after performing the KTS algorithm to segment the video, the algorithm is able to distinguish between similar segments. Our approach performs better than baseline also because during the autoencoder stage, the most salient features are captured, i.e. the features that contribute the most to the make up of the representation feature. These features which are subsequently fed into the MLP stage clearly shows the power of using these salient features. Thus, by combining a unsupervised method with a supervised method we are able to capture a low-dimension feature space by using a under-complete structure as well as allowing the algorithm to learn from human summaries to better mimic this task.

In Table 5.5, we see that the baseline approach using I3D features performs the best when combining both the RGB and flow features together, which increases approximately by 1%. What is interesting is that when we combine the results of the RGB and flow for our SummaryNet model the results decrease by 0.02%. Even though this is not a significant decrease, one would expect the combination of both features to increase the system's overall accuracy. This could potentially be because of the way we combine the results of the predictions from the two networks. An optimal way to combine these predictions is worth future investigation. When we compare the results of baseline-I3D to SummaryNet we see that the system beats the baseline by just over 2.5% which is expected due to the power of the network we have constructed. When we compare the flow results of the two models we see that the SummaryNet model beats the baseline-I3D model by almost 5%. This is a significant jump which indicates that the model we have constructed is able to take better advantage of the long-term dependencies made apparent by the flow features. If we look at the final baseline-I3D results compared to the final baseline-ResNet results, we see that there is over a 18% increase in the overall accuracy. This huge increase in accuracy was unexpected. However, the increase in accuracy makes sense since we are now considering spatio-temporal features instead of only spatial features. Similar to the flow results, there is almost a 5% increase in the system accuracy for the final results between the baseline-I3D model and the SummaryNet model. These results makes intuitive sense since we are considering temporal information as well, which is vital when considering sequential data.

It is important to note that the KTS points are obtained by using the features that are supplied to the model, which directly influences the system's results. Thus, the change points supplied by using the I3D features clearly indicate its superiority over using only spatial features such as ResNet.

In Figure 5.1, we see the frames selected by our model for the video name "EYqVtI9YWJA" where

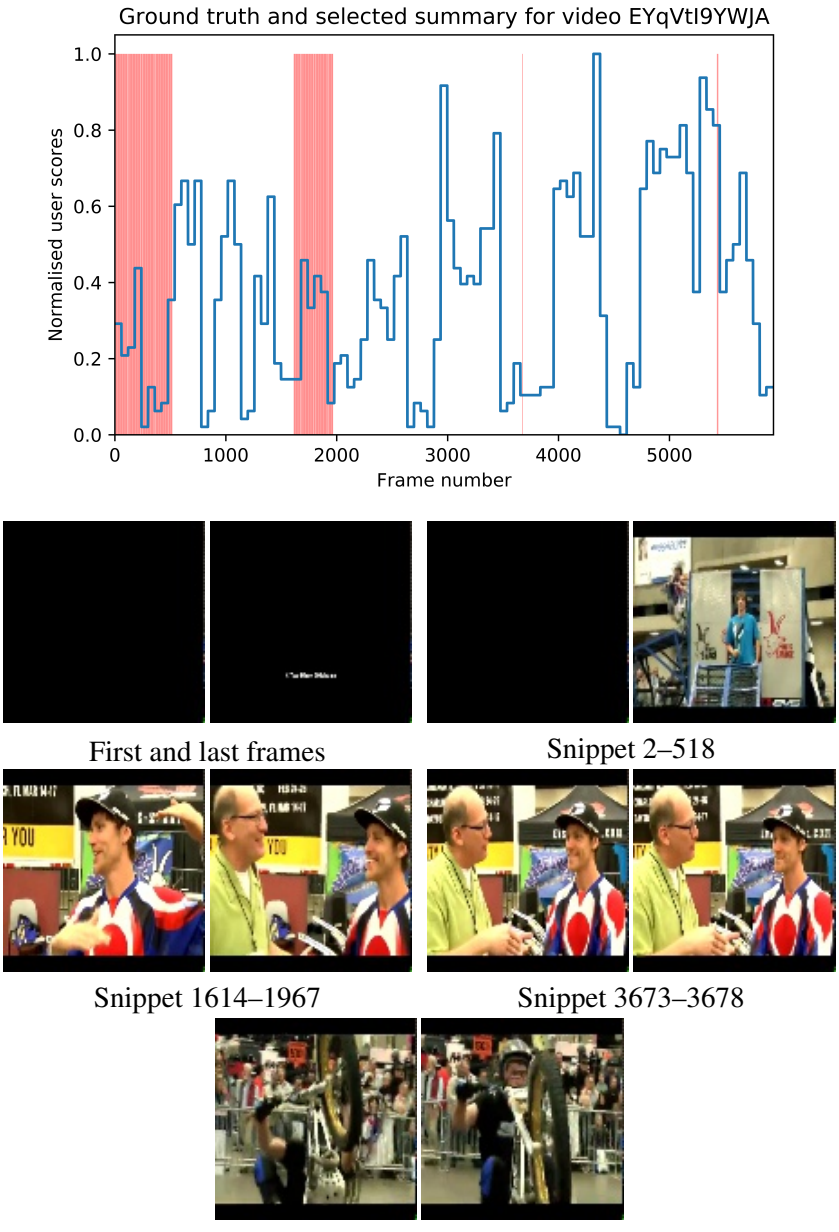


Figure 5.1: Samples of bad summary results from TvSum50 dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a bad summary i.e. a low F1-score

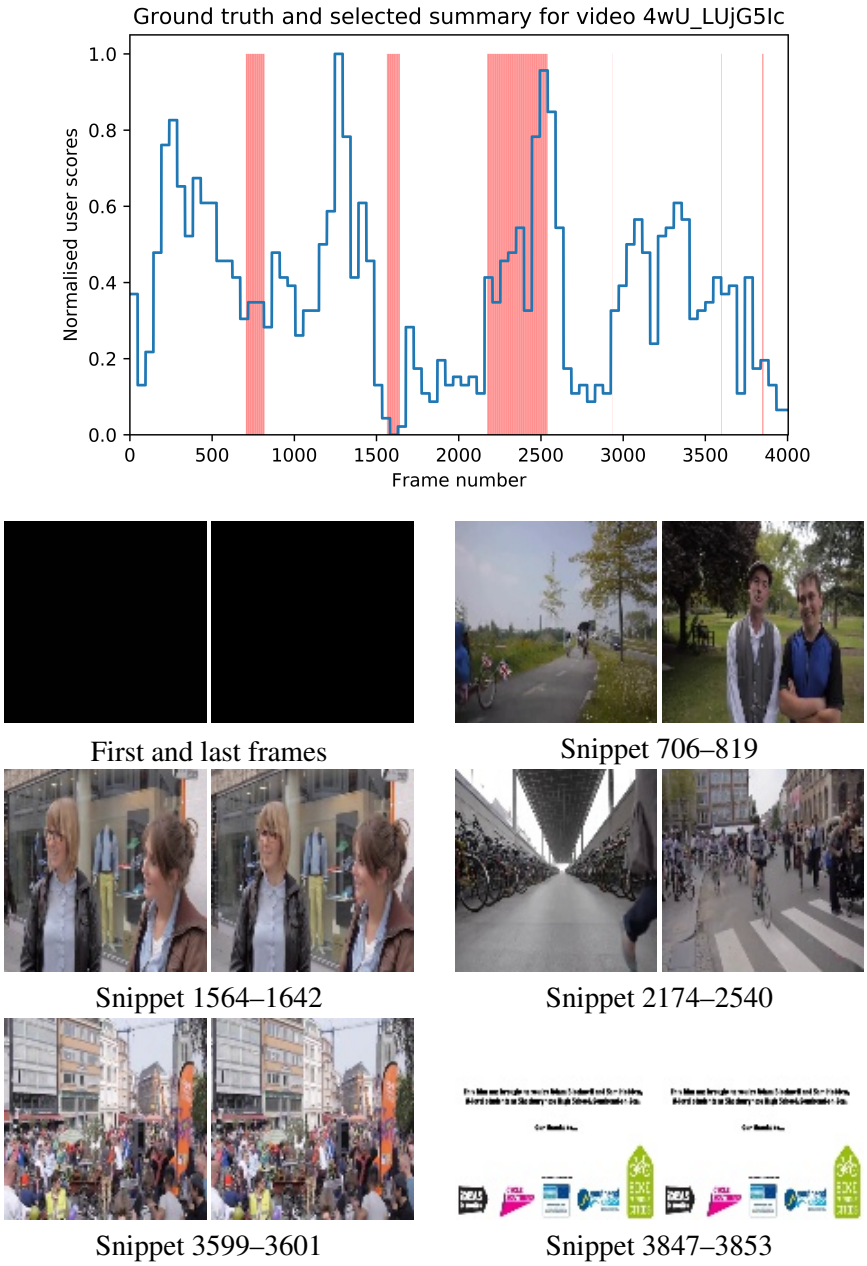


Figure 5.2: Samples of good summary results from TvSum50 dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a good summary i.e. a high F1-score

the model achieved a relatively low F1-score of 0.31688 and a total summary of 0.14985. This means that 14.985% of the total video in length was selected. This is under the video category *BT* which translates to: "attempting bike tricks". This video is taken with the camera moving and continuously alternates between the exact same interview state and a similar state where people are performing tricks on a motor-bike. In the summary frames seen in Figure 5.1, it is clear that similar frames were selected even though these frames appear multiple times over the video. We also see that the model selected the first few frames and then jumped to frames between 1614-1967. Thereafter, the model does not choose any other frames for the remainder of the ± 4000 frames we would generally want to get summaries throughout the entire video which will enable us to get a good synopsis of a videos content. This could be because of the high moving/alternating camera views (going back and forth to similar views) that causes the model to perform poorly. Another reason could be the seemingly-high disagreement between user scores for these types of videos makes it difficult for the model to learn an optimal summary. This again relates to the highly-subjective nature of the problem, lending to its inherent difficulty.

In Figure 5.2, we see the frames selected by our model for the video name "4wU_LUjG5Ic", where the model achieved a relatively high F1-score of 0.97910 and a total summary of 0.14257. This is under the video category *PR* which translates to: "PaRade". This video is taken with the camera moving and continuously alternates between an the different interview states and a dissimilar states where people are attending a parade. In the summary frames seen in Figure 5.2, it is clear that frames of varying content were selected it shows people on bicycles then an interview then the a different aspect of the parade and then an interview shot again. In the graph plot of Figure 5.2, we see that shots are selected throughout the video which gives us a good synopsis of the video content. The model seemingly performs well when there are varying scenes in the video that are dissimilar from previous scenes.

5.4.2 SumMe Dataset

Similar to the TvSum50 dataset, the SumMe dataset is also split up using the 80-20 split. Since this dataset consists of 25 videos, we use 5 videos for testing chosen randomly. Descriptive details about these videos can be seen in Table 5.6. The bulk of the videos in this dataset is taken while the camera is moving. The rest consists of egocentric and static videos which have 4 videos each. Features for each video frame is extracted in a similar way to what we have described above. It is also noteworthy that the fine-tuned I3D network is tuned specifically for the current dataset and not both. Since the videos in

Video name	Camera	FPS	Length
Uncut_Fvening_Flight	moving	30	5m23s
Valparaiso_Downhill	egocentric	30	2m53s
car_over_camera	static	30	2m26s
paluma_jump	moving	30	1m26s
playing_ball	moving	30	1m44s

Table 5.6: Descriptive information about videos the system is tested on. The information in this table is taken from [18]

Method	Features	Results - RGB	Results - FLOW	Final Results
Baseline	ResNet	-	-	38.058%
Baseline	I3D	33.864%	41.448%	40.638%
SummaryNet	I3D	35.828%	43.07%	**44.6%**

Table 5.7: Results obtained for the SumMe dataset using our proposed method. These results clearly show the advantage of using spatio-temporal features. The best result is shown in bold.

the SumMe dataset is not arranged by category and has a somewhat randomness of data associated with it, we can expect lower results as compared to the TvSum50 dataset. Another reason why the SumMe dataset is challenging is because even though the videos are slowly changing, almost 70% of the videos supplied is videos where the camera is moving which makes it difficult to gather change points from the KTS algorithm. Future work could possibly involve finding alternative algorithms to address these types of videos.

In Table 5.7, the baseline results using ResNet features is similar to the results achieved by [52] for their baseline model, which makes sense since the baseline model used is similar to theirs. In the subsequent row, we see the results of the model using the I3D features. As mentioned previously, since we are dealing with two-stream networks, the predictions of the RGB and flow networks are combined to give a final prediction which is shown in the last column of the table. Our SummaryNet method for this dataset indicate an overall increase in the systems accuracy when comparing it to the baseline methods.

When we compare the RGB results of the SummaryNet with the baseline-I3D, we can see almost a 4% increase in favour of our proposed model. This shows that the network architecture that we have constructed performs better on average when compared to the baseline method. When considering the flow aspect, we see that the baseline-I3D falls short by around 1.5% as compared to our model. What is particularly interesting is that when combining the RGB and flow results for the baseline-I3D model, the overall accuracy does not increase when compared with the flow aspect. Instead, the final predictions rather decrease by almost 1%. This is because there is a high disagreement with the final predictions

between the RGB and flow networks. Thus, when averaging the score out, one network could bring the final (important) shots value to be less meaningful when going through the knapsack algorithm. When we compare the RGB results we see that our model performs almost 2% better than the baseline model.

When considering the SummaryNet approach, we see that by combining the RGB and flow aspects together we get an overall increase in the system's accuracy. Which shows us that considering spatial (RGB) and temporal (flow) information is necessary when dealing with videos.

If we look at the final results of the baseline-ResNet and baseline-I3D we see there is over a 2.5% increase in the system results. This increase is expected since we are now considering temporal information in conjunction with spatial information. If we compare our SummaryNet model to the baseline-ResNet model we see there is a 6.5% increase in the overall system accuracy. When we consider the model we proposed - the two-stream SummaryNet network - we see an obvious improvement, with a rise of almost 4% compared to the baseline model with I3D features. The results shown in this table makes intuitive sense since we are considering spatio-temporal features and we construct a more powerful model when compared to the baseline model. The proposed architecture demonstrates the power of using our semi-supervised approach with spatio-temporal features.

As mentioned previously the results depend largely on the change points supplied by the KTS algorithm. Therefore, getting good feature representations for each video frame is a vital task for the overall system accuracy.

In Figure 5.3, we see the frames selected by our model for the video name "Valparaiso_Downhill" where the model achieved a relatively low F1-score of 0.23787 and a total summary of 0.14426. This video is taken with the camera moving (egocentric) and continuously alternates between different scenes i.e. fast pace scene changes. In the summary frames seen in Figure 5.3, the selected frames seems very similar even though they are taken at different points in the video. We also see that the model selected 4 different shots throughout the video which is ideally what we would want. However, the last shot (snippet) chosen by our model, shown in Figure 5.3, selected frames between 4461-4621 which has almost a 0 average user score. Similar to what was discussed for the TvSum50 dataset where the model performed poorly, this could be because of the high moving/alternating camera views (going back and forth to similar views) that causes the model to perform poorly.

In Figure 5.4, we see the frames selected by our model for the video name "car_over_camera" where the model achieved a relatively high F1-score of 0.50390 and a total summary of 0.11615. This video is

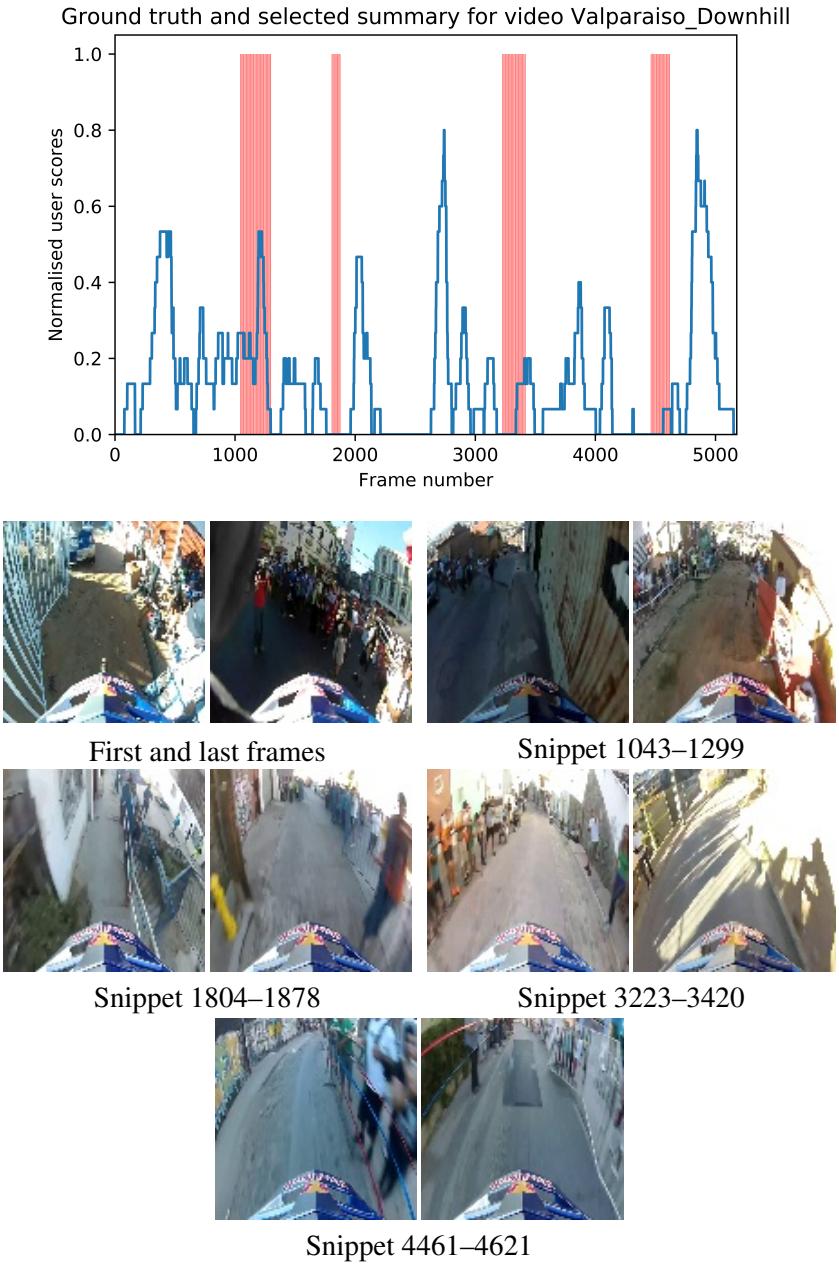


Figure 5.3: Samples of bad summary results from SumMe dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a low F1-score where the model generated a bad summary.

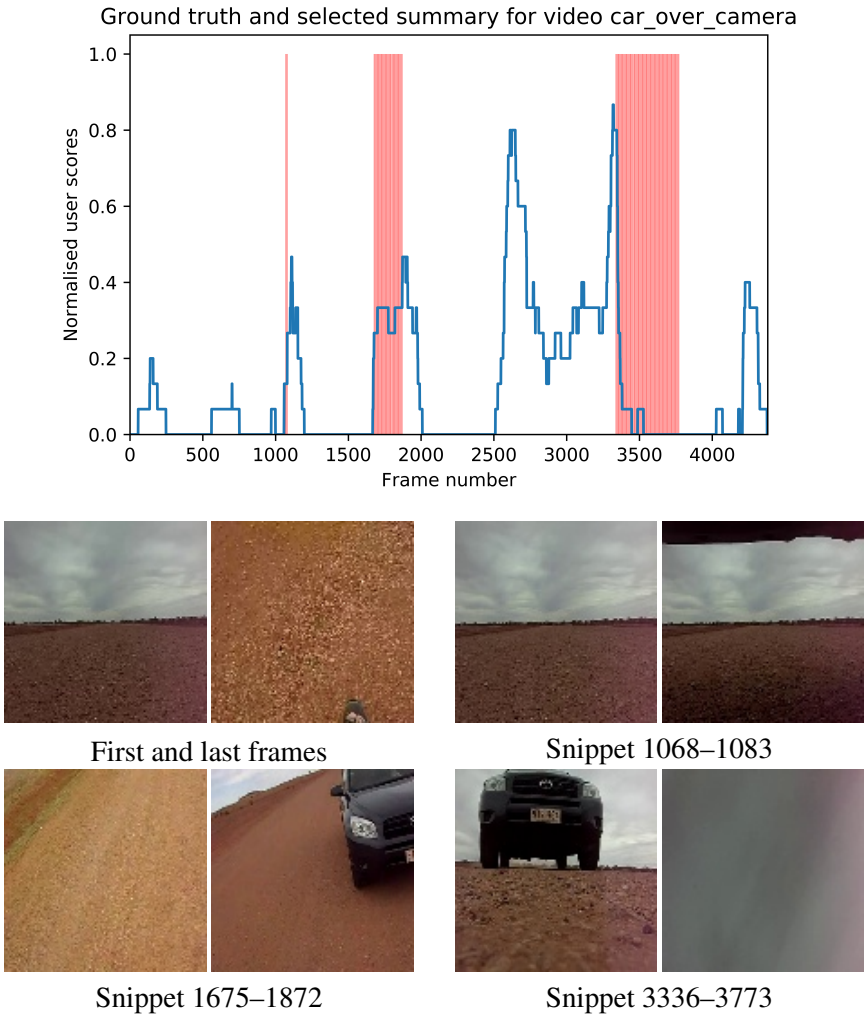


Figure 5.4: Samples of good summary results from SumMe dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a high F1-score where the model generated a good summary.

Method	SumMe	TvSum50
[35]	41.7%	56.3%
[50]	-	57%
[52]	38.6%	54.7%
[24]	44.4%	61%
[54]	42.1%	58.1%
[37]	47.5%	56.8%
[36]	40.1%	56.3%
[18]	39.7%	-
[44]	-	51.3
SummaryNet (ours)	44.6%	73.2%

Table 5.8: Comparing previous results to our method. The best results are shown in dark blue and second best is shown in light blue.

taken with the camera mostly stationary and focus on a object in the view of the camera. In this video the scene changes slowly and differences in the scenes are clearly visible. In the summary frames seen in Figure 5.4, it is clear that frames of varying content were selected. The camera is places on the ground (sand) and the car attempts to drive over it. In the plots of Figure 5.4, we see that shots are selected where there was high agreement between user scores. Although it missed the highest peaks between frames 2500-3500, it still gives a good synopsis and achieves a relatively high F1-score. Since there are varying scenes that are slower changing the model seemingly performs well when there are scenes that are dissimilar from previous scenes.

5.5 Comparing Results

In Table 5.8, we see previous results compared to our method. For the TvSum50 our method outperforms state-of-the-art by approximately 12% (absolute increase). This significant increase in accuracy is attractive since it demonstrates the power of our proposed method. However, for the SumMe dataset we achieve competitive results to state-of-the-art, where state-of-the-art is around 3% better than our method. This could be due to the fact that the SumMe dataset is more challenging dataset compared to the TvSum50 dataset since the SumMe dataset has no categorical structure associated with it. [37] achieves state-of-the-art by using full convolutional networks. The power of adopting this method is made evident. Since this model is based on semantic segmentation, they are able to exploit underlying similarities, such as visually diverse scene frames, when compared with the video summarisation problem. Overall, our system does well coming second to state-of-the-art for the SumMe dataset.

Since the majority of previous methods use strictly spatial features, the results we obtained makes sense. Surprisingly, most previous methods do not consider temporal features, they rather attempt to capture the long term dependencies by the use of recurrent cells, such as [52; 24; 35; 54; 49]. Thus, by using both spatial and temporal information with our SummaryNet model we are able to take advantage of sequential data more effectively. If we analyse the results presented in Table 5.5 and Table 5.7 we see the advantage that temporal information adds to the overall system. These results shows clear limitations of models that use only spatial information as inputs. By considering temporal information along with the spatial information we demonstrate that our method is superior to previous methods.

The closest results to ours is done by [24] where they attached attention mechanisms to give more weight to certain features. The paper that set the tone on how to compare results effectively was done by [52] which is widely used as a reference throughout papers attempting to address the video summarisation task. In our work we have also referenced many aspects from this paper. Their results however, are around 18% and 6% less than ours for the TvSum50 and SumMe datasets respectively.

Our approach is able to capture the most salient features for each frame as well as exploit temporal dependencies which makes our method perform well on the TvSum50 and SumMe datasets. The temporal information seemingly adds invaluable information to the overall system accuracy.

5.6 Computational Runtime

In this section we provide the runtime values along with other important metrics of our various networks. The summarised tables for the SumMe and TvSum50 dataset can be seen in Table 5.9 and Table 5.10.

When looking at Table 5.9 and Table 5.10 we see the total number of parameters for our baseline method and our SummaryNet method. The total number of parameters of the baseline-ResNet network is 329473 which is exactly half of the total number of parameters of the baseline-I3D network. This makes sense since there are two streams (RGB and flow) for the baseline-I3D network. The time taken to train and test these networks double in the case of the baseline-I3D compared to the baseline-ResNet method. What is particularly interesting is that the SummaryNet method has a total of $10.9M$ parameters which is around 16 times more than that of the baseline-I3D model but does not take 16 times longer to train. For testing, the times also increase as the number of parameters increase which is shown in the last column of Table 5.9 and Table 5.10. Our SummaryNet method would certainly take longer to train since there are two

Method	Features	#Params	Training					Testing	
			#Input Cubes	Val Loss	Train Loss	Train Acc	Time	#Input Cubes	Time
Baseline	**ResNet**	329473	6264	0.699	0.6051	0.6879	89.14s	24820	112.75s
Baseline	**I3D**	658946	6264	0.6250	0.6090	0.7101	199.50s	24820	231.4s
SummaryNet	**I3D**	10.9M	6264	0.6116	0.6024	0.7096	870.47s	24820	262.88s

Table 5.9: Run-time of our models for the SumMe dataset. Here we see the number of parameters for each setting. We also see the time take for each setting to complete for training and testing

Method	Features	#Params	Training					Testing	
			#Input Cubes	Val Loss	Train Loss	Train Acc	Time	#Input Cubes	Time
Baseline	**ResNet**	329473	21186	0.4790	0.4934	0.7613	160.93s	51378	198.45s
Baseline	**I3D**	658946	21186	0.6059	0.5831	0.6771	276.52s	51378	374.37s
SummaryNet	**I3D**	10.9M	21186	0.6215	0.5754	0.7007	2599.2s	51378	460.58s

Table 5.10: Run-time of our models for the TvSum50 dataset. Here we see the number of parameters for each setting. We also see the time take for each setting to complete for training and testing

networks being trained.

One important aspect to grasp is that the training accuracy differs from the testing accuracy which has been mentioned before. The training accuracy is included for completeness. During training, each cube of frames has a binary target of 0 or 1 which indicates if the center frame is a summary of not, the weights are then updated accordingly based on the binary cross-entropy loss. It is worth mentioning that a custom metric could be included, however, this would provide little value since we are sub-sampling and it is not possible to process the training data (as mentioned in Chapter 4.4.8) after each pass through the our networks. During testing, the model predicts values for each frame in the video sequence. After this, the predictions go through a post-processing stage which follows the literature of [24; 54; 52; 36; 37] and is described in Chapter 4.4.8.

All methods are ran for 8 epochs and the best performing one is selected based on the lowest validation loss achieved by the model.

At a first glance, the $\#InputCubes$ for training might seem low. However, since we are following the method of [52] we sub-sample 2 fps for each video in the training dataset. For testing all frames are processed and no sub-sampling takes place hence the $\#InputCubes$ for testing is more than that of the training.

5.7 Conclusion

The need to consider spatio-temporal features is made evident in Tables 5.5 - 5.7. We see that [37] achieves the best results for the SumMe dataset by using fully-convolutional networks. However, our model still performs relatively well in comparison coming second to state-of-the-art. For the TvSum dataset our model achieves state-of-the-art results by surpassing the second highest by approximately 12%. When compared to [37] for the TvSum50 dataset we outperform it by almost 18% which shows are model is more robust since it performs well on both datasets. In Figure 5.1 - Figure 5.4 we see that our model works well when videos do not have very fast scene changes. It also works well when the scenes are visually different when compared to other scenes throughout the video.

In summary we see that previous methods use strictly spatial features only. Previous approaches dealt with temporal information in the architecture by introducing some form of recurrence. In this work, we address the need to consider the temporal aspect at a lower level inherent in video structures, along with the spatial information on a frame level. In this way, we model both appearance (spatial) and motion (temporal) information. We posit that by our method of combining spatial and temporal information we achieve a more robust and superior system compared to previous results that only consider spatial information.

One important aspect of our method that should be addressed in future work is the need for a better way to combined the RGB and flow predictions. This is made clear by the tables results mentioned above.

The overall system accuracy still benefits from using both the spatial and temporal aspects present in video data.

We also seen the runtimes of the different proposed methods in Table 5.9 and Table 5.10. The SummaryNet model takes longer to train since there are two networks being trained. With approximately 16 times the number of parameters in the SummaryNet model compared to the baseline-I3D model the training and testing of the SummaryNet model runtime is still very appealing.

Chapter 6

Conclusion and Future Work

In this dissertation, previous methods that addressed some frame-based (video) problem were discussed along with other architectures/features that could prove to be useful while developing our own solution for the video summarisation problem. The advantages of developing a robust system to combat this issue is vast. Examples include creating automatic trailers from a movie, generating sport highlights, removing similar/redundant content from personal videos, faster browsing on YouTube/social media and generally any space that has some form of video data available.

We presented some transfer learning models (C3D, VGGNet, etc.) that could be used as a feature extractor. The advantage of this is clear - it saves us costs (time and training) and potentially saves us from reinventing the wheel. We adopt the I3D network which we use as a feature extractor and continue their two-stream approach to address the video summarisation problem. Thus we are able to capture spatio-temporal features which is vital for modelling sequential data such as videos.

The motivation behind using the autoencoder and a multi-layer Neural Network were also discussed. The need to include recurrent layers and some gated units such as LSTM cells are made clear, to address the Long-Term dependency issue and to include temporal information. The need for temporal features and not just spatial features are vital to tackle the video summarisation problem. The use of an auto-encoder is because AEs attempt to capture the most salient features in the latent-subspace (code) layer. With this important feature in mind, it will allow as to compare semantically meaningful parts of a video to each other. Similar models have previously shown to be highly effective and achieve state-of-the-art results.

The performance of the entire system is evaluated against other papers in the following way: The same

datasets is used, there is a holdout dataset which we test against, in the same testing manner employed in the comparing paper as mentioned in [18; 44; 49; 24; 50; 35; 54; 52]. Performance metrics such the F1-score were computed which is the metric used in previous papers. We create a baseline which we evaluate our method against and we show the need to consider temporal features in unison with spatial features.

Our SummaryNet model employs both spatial and temporal features extracted with deep learning architectures, and proves to be robust on both the SumMe and TvSum50 datasets. It achieves competitive results on the SumMe dataset and achieves state-of-the-art on the TvSum50 dataset. The SumMe dataset proves to be more challenging as discussed above. We see by the results presented in Table 5.8 by combining deep high-level spatial and temporal feature representations improves the video summarisation performance of the algorithms employed. This is made clear by Tables 5.7 and 5.5.

Future investigation warrants a better way of combining the RGB and flow aspects so as to maximise both sets of features so that a optimal solution can be found. The need for this is made clear in Table 5.7. Adding more resources could also increase the total accuracy of the proposed system. This will enable larger windows to be considered during training and testing which would benefit the models deep nature.

The model we designed does well on videos that have varying scenes throughout the video. This brings the need to develop a method that can deal with videos that have similar scenes throughout the video and still produce a relatively good synopsis of the video. Along with this, future models can include attention mechanisms to pay particular attention to certain generic scenes in a video. Additionally, in future one can also consider adding acoustic cues and other video meta-data to aid the task of video summarisation to push accuracies to be near human-level.

Bibliography

[1] Cnns architectures: Lenet, alexnet, vgg, googlenet, resnet. https://medium.com/@siddharthdas_32104/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-r Accessed: 2018-04-05.

[2] Normalize data. https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/normalize-data. Accessed: 2018-10-29.

[3] Visual networking index forecast - cisco. https://www.cisco.com/c/en/us/solutions/service-provider/vni-network-traffic-forecast/vni-forecast-info.html. Accessed: 2019-08-06.

[4] A. Aner and J. R. Kender. Video summaries through mosaic-based shot and scene clustering. *European Conference on Computer Vision*, 2002.

[5] Gong B., Chao W.L., Grauman K., and Sha F. Diverse sequential subset selection for supervised video summarization. *NIPS*, 2014.

[6] Richard E. Bellman. *Dynamic Programming*. 1956.

[7] João Carreira and Andrew Zisserman. https://github.com/dlpbc/keras-kinetics-i3d/releases/download/v0.2/.

[8] João Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. *CoRR*, abs/1705.07750, 2017.

[9] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. October 2014.

[10] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 24(5):603–619, 2002.

[11] Sandra Eliza Fontes de Avila, Ana Paula Brando Lopes, Antonio da Luz Jr., and Arnaldo de Albuquerque Arajo. Vsumm: A mechanism designed to produce static video summaries and a novel evaluation method. *Pattern Recognition Letters*, 2011.

[12] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. Technical Report UCB/EECS-2010-24, EECS Department, University of California, Berkeley, Mar 2010.

[13] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press, 1996.

[14] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In Josef Bigun and Tomas Gustavsson, editors, *Image Analysis*, pages 363–370, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[15] Felix A. Gers and Juergen Schmidhuber. Recurrent nets that time and count. 2000.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[17] M. Gygli, H. Grabner, and L. Van Gool. Video summarization by learning submodular mixtures of objectives. *CPVR*, 2015.

[18] Michael Gygli, Helmut Grabner, Hayko Riemenschneider, and Luc Van Gool. Creating summaries from user videos. 2014.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[20] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 2006.

[21] Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[22] Kur Hornik. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:389–366, 1989.

[23] Herve Jegou, Matthijs Douze, Cordelia Schmid, and Patrick Perez. Aggregating local descriptors into a compact image representation. *IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[24] Zhong Ji, Kailin Xiong, Yanwei Pang, and Xuelong Li. Video summarization with attention-based encoder-decoder networks. *CoRR*, abs/1708.09545, 2017.

[25] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

[26] P. Kaur and R. Kumar. Analysis of video summarization techniques. *International Journal for Research in Applied Science and Engineering Technology*, 2018.

[27] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Proceeding NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems*, 2012.

[29] Yann LeCun. Generalization and network design strategies. *Technical Report CRG-TR-89-4*, 1989.

[30] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradientbased learning applied to document recognition. *PROC OF THE IEEE*, 1998.

[31] W. Liu, T. Mei, Y. Zhang, C. Che, and J. Luo. Multi-task deep visual-semantic embedding for video thumbnail selection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[32] Zheng Lu and Kristen Grauman. Story-driven summarization for egocentric video. *Computer Vision and Pattern Recognition*, 2013.

[33] Odile Macchi. The coincidence approach to stochastic point processes. *Advances in Applied Probability*, 1975.

[34] J. Macqueen. Some methods for classification and analysis of multivariate observations. In *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[35] Behrooz Mahasseni, Michael Lam, and Sinisa Todorovic. Unsupervised video summarization with adversarial lstm networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2982–2991, 2017.

[36] Elfeki Mohamed and Borji Ali. Video summarization via actioness ranking. *arXiv:1903.00110v1*, 2019.

[37] Rochan Mrigank, Ye Linwei, and Wang Yang. Video summarization using fully convolutional sequence networks. *arXiv:1805.10538v2*, 2018.

[38] Danila Potapov, Matthijs Douze, Zaid Harchaoui, and Cordelia Schmid. Category-specific video summarization. *ECCV - European Conference on Computer Vision*, 2014.

[39] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 1958.

[40] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 1986.

[41] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[42] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *Trans. Sig. Proc.*, 45(11):2673–2681, November 1997.

[43] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[44] Y. Song, J. Vallmitjana, A. Stent, and A. Jaimes. Tvsum: Summarizing web videos using titles. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[45] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[46] Naoya Takahashi, Michael Gygli, and Luc Van Gool. Aenet: Learning deep audio features for video analysis. *IEEE Transactions on Multimedia*, 2018.

[47] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[48] Du Tran, Lubomir D. Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. C3D: generic features for video analysis. *CoRR*, abs/1412.0767, 2014.

[49] Huan Yang, Baoyuan Wang, Stephen Lin, Baining Guo, David Wipf, and Minyi Guo. Unsupervised extraction of video highlights via robust recurrent auto-encoders. *arXiv:1510.01442v1*, 2015.

[50] Yitian Yuan, Tao Mei, Peng Cui, and Wenwu Zhu. Video summarization by learning deep side semantic embedding. *IEEE transactions on circuits and systems for video technology*, 2017.

[51] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *European conference on computer vision*, 2014.

[52] Ke Zhang, Wei-Lun Chao, Fei Sha, and Kristen Grauman. Video summarization with long short-term memory. *arXiv:1605.08110v2*, 2016.

[53] B. Zhao and E. P. Xing. Quasi real-time summarization for consumer videos. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[54] Kaiyang Zhou, Yu Qiao, and Tao Xiang. Deep reinforcement learning for unsupervised video summarization with diversity-representativeness reward. *arXiv:1801.00054*, 2017.

[55] Y.T. Zhou and Rama Chellappa. Computation of optical flow using a neural network. pages 71 – 78 vol.2, 08 1988.

[56] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *Proceedings of the IEEE international conference on computer vision*, 2015.

Appendix

In this section we provide additional testing results achieved by our SummaryNet model. For the SumMe dataset, all testing results are shown since there are only 5 testing videos our of the 25 total videos. For the TvSum50 dataset there are a total of 10 testing videos. However, we display 5 testing videos with varying F1-scores to show how the model performs in these different settings.

SumMe Dataset

Figures 6.1 - 6.5 shows additional test results achieved by our SummaryNet model for the SumMe dataset.

In Figure 6.1, we see the frames selected by our model for the video name "car_over_camera" where the model achieved a relatively high F1-score of 0.50390 and a total summary of 0.11615.

In Figure 6.2, we see the frames selected by our model for the video name "Valparaiso_Downhill" where the model achieved a relatively low F1-score of 0.23787 and a total summary of 0.14426.

In Figure 6.3, we see the frames selected by our model for the video name "Paluma_jump" where the model achieved a relatively high F1-score of 0.517 and a total summary of 0.14064.

In Figure 6.4, we see the frames selected by our model for the video name "Playing_ball" where the model achieved a relatively average F1-score of 0.3715 and a total summary of 0.14743.

In Figure 6.5, we see the frames selected by our model for the video name "Uncut_Evening_Flight" where the model achieved a relatively high F1-score of 0.4952 and a total summary of 0.14940.

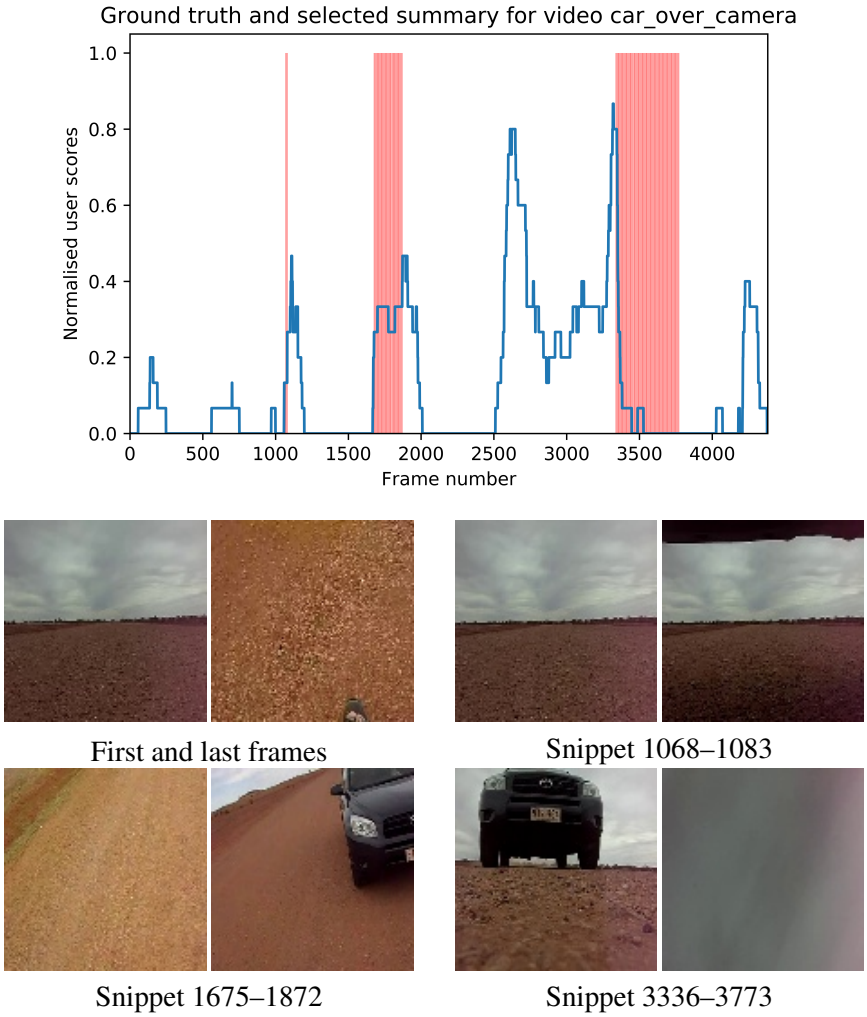


Figure 6.1: Sample results from SumMe dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a good summary i.e. a high F1-score.

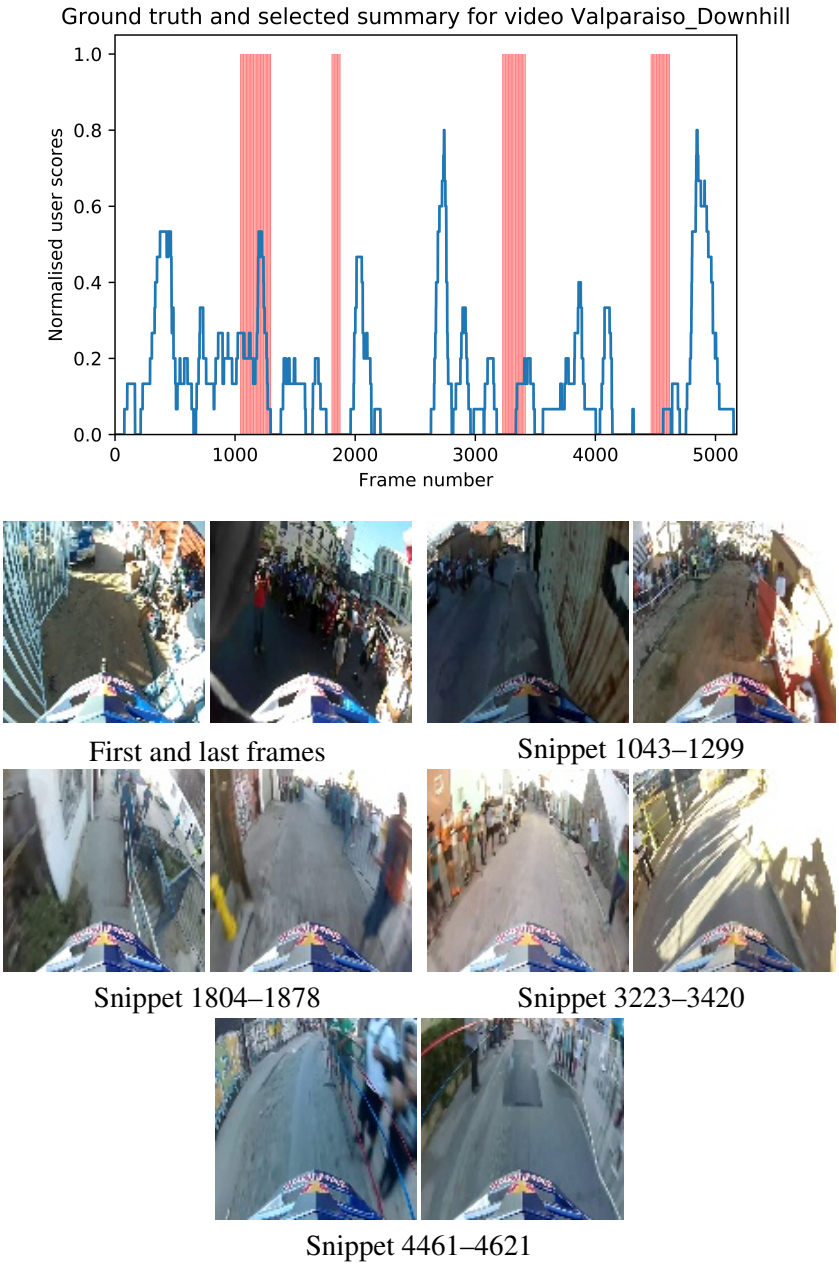


Figure 6.2: Sample results from SumMe dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a bad summary i.e. a low F1-score.

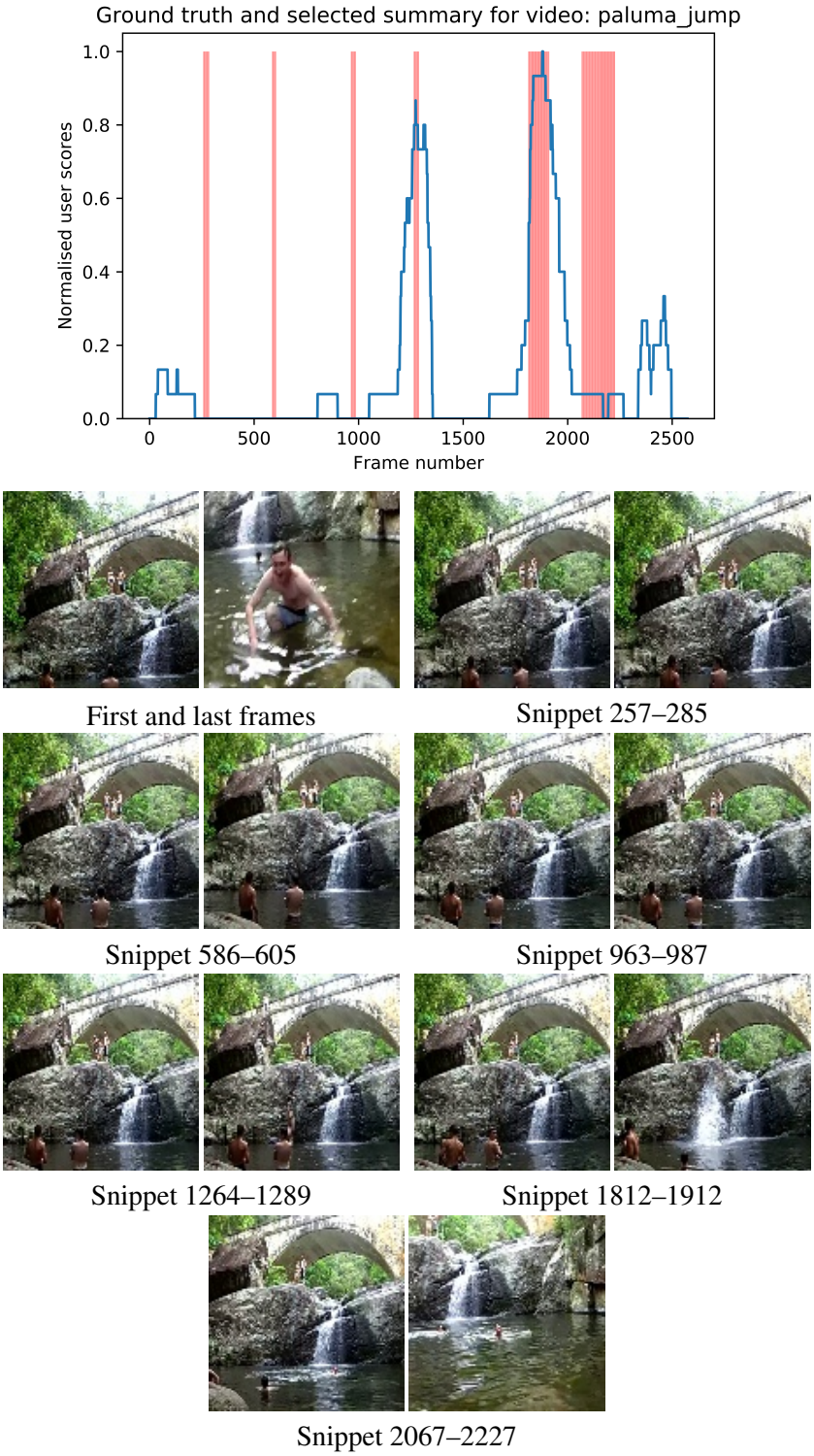


Figure 6.3: Sample results from SumMe dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a good summary i.e. a high F1-score.

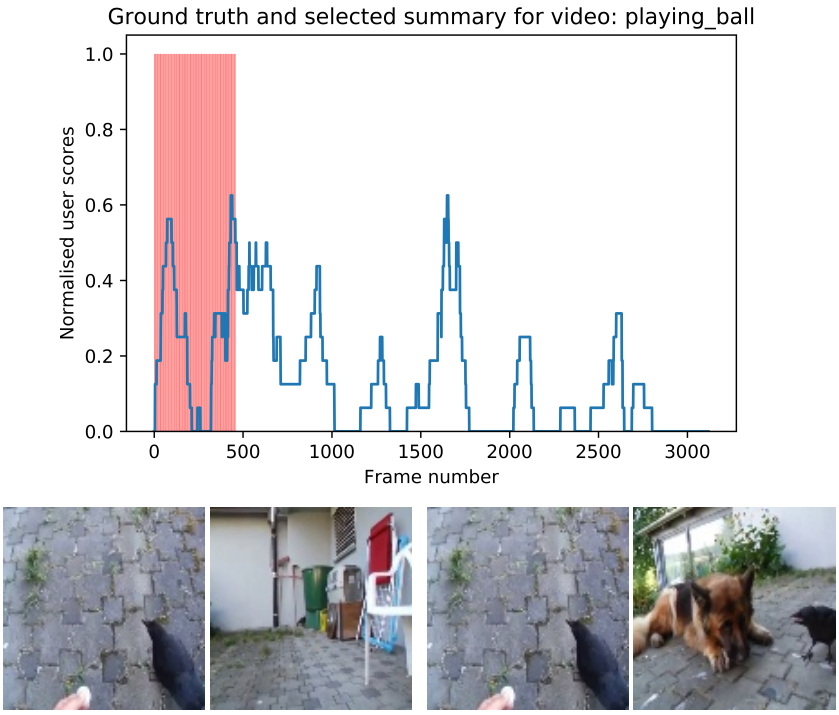


Figure 6.4: Sample results from SumMe dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a average summary.

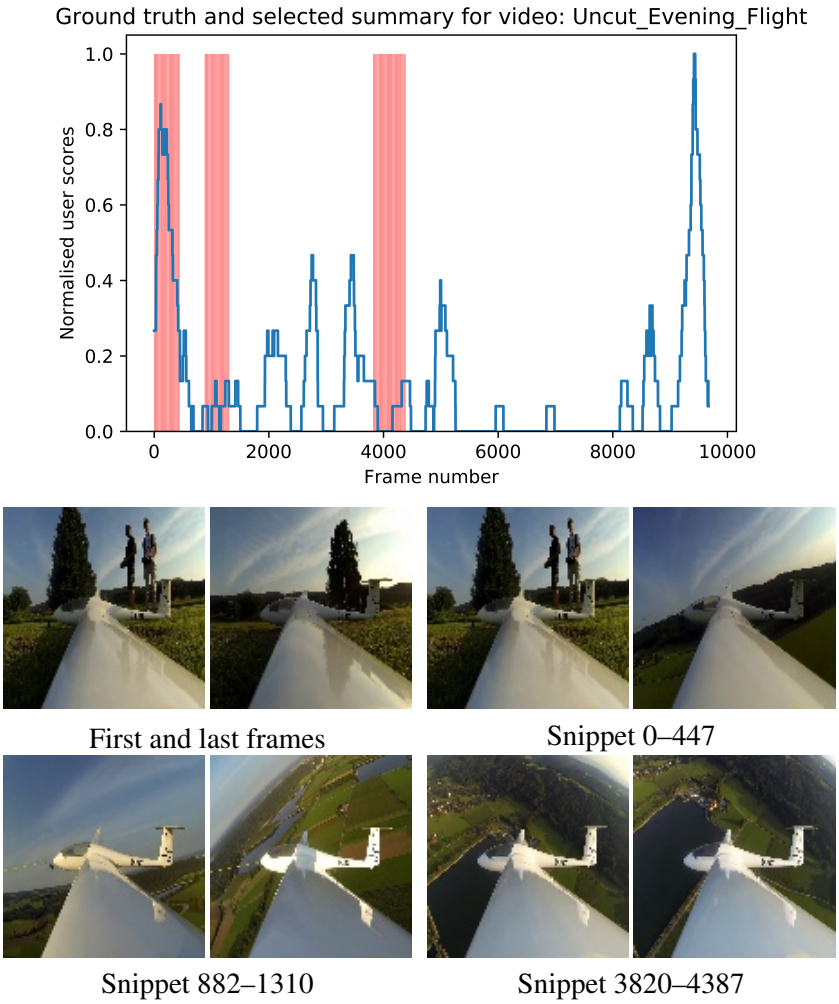


Figure 6.5: Sample results from SumMe dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a good summary i.e. a high F1-score

TvSum50

Figures 6.6 - 6.10 shows additional test results achieved by our SummaryNet model for the TvSum50 dataset.

In Figure 6.6, we see the frames selected by our model for the video name "0tmA_C6XwfM" where the model achieved a relatively high F1-score of 0.76228 and a total summary of 0.13533.

In Figure 6.7, we see the frames selected by our model for the video name "37rzWOQsNIw" where the model achieved a relatively low F1-score of 0.68128 and a total summary of 0.13462.

In Figure 6.8, we see the frames selected by our model for the video name "GsAD1KT1xo8" where the model achieved a relatively average F1-score of 0.58849 and a total summary of 0.08677.

In Figure 6.9, we see the frames selected by our model for the video name "HT5vyqe0Xaw" where the model achieved a relatively high F1-score of 0.79837 and a total summary of 0.13638.

In Figure 6.10, we see the frames selected by our model for the video name "JKpqYvAdIsw" where the model achieved a relatively low F1-score of 0.39228 and a total summary of 0.10836.

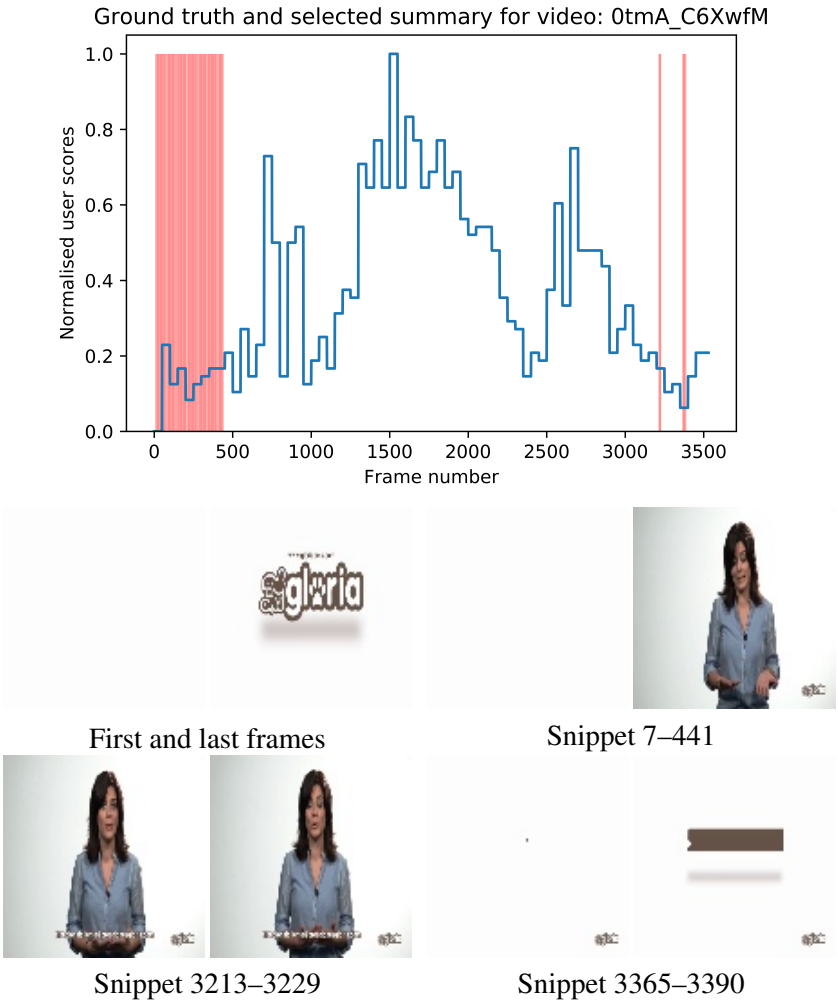


Figure 6.6: Sample results from TvSum50 dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a good summary i.e. a high F1-score.

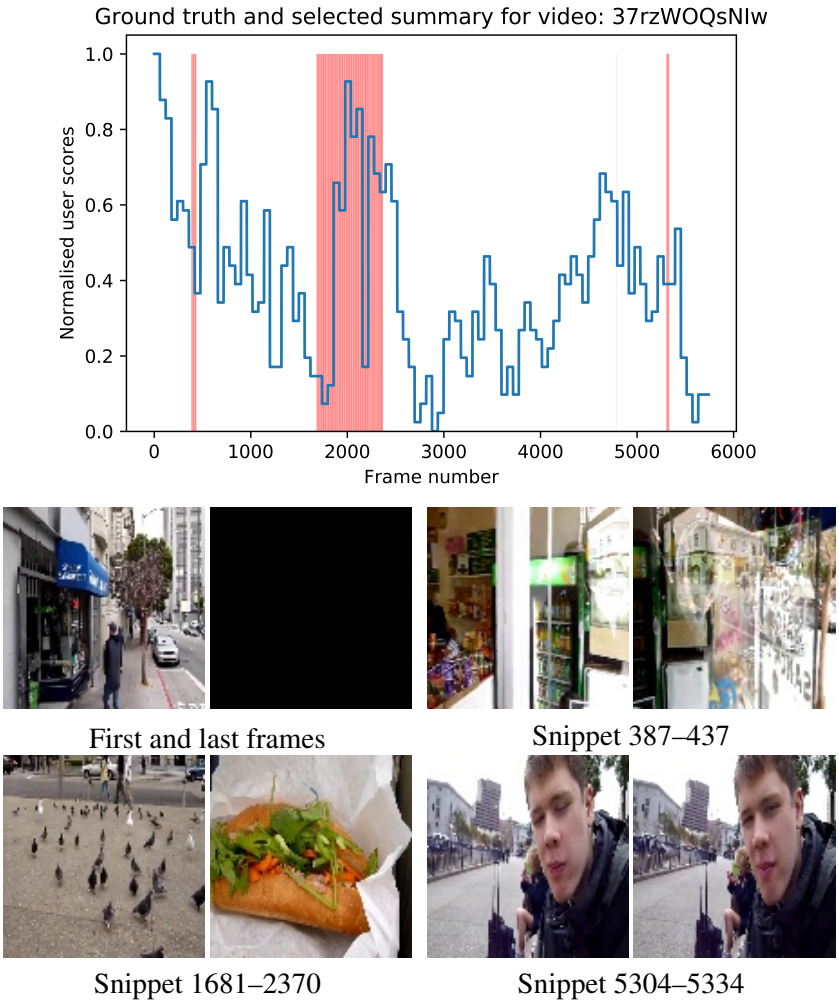


Figure 6.7: Sample results from TvSum50 dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a bad summary i.e. a low F1-score.

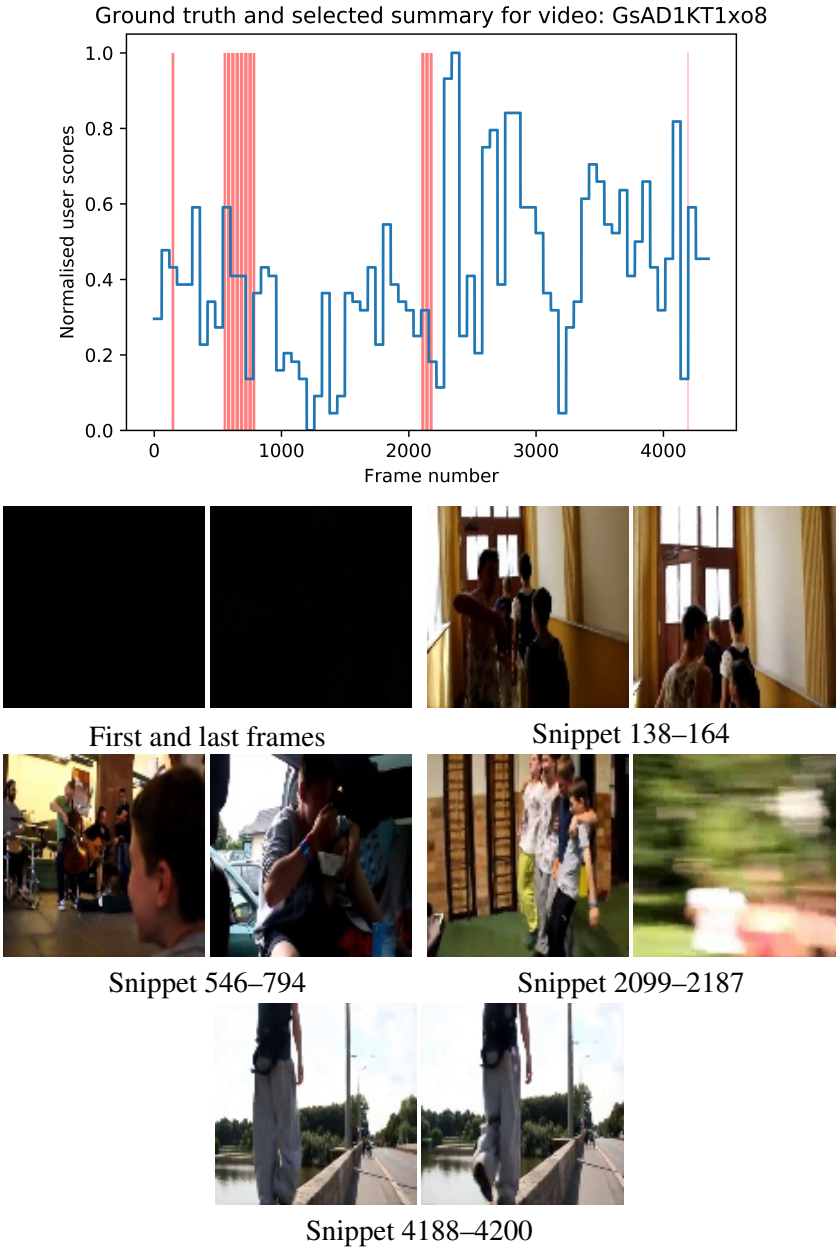


Figure 6.8: Sample results from TvSum50 dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a average summary.

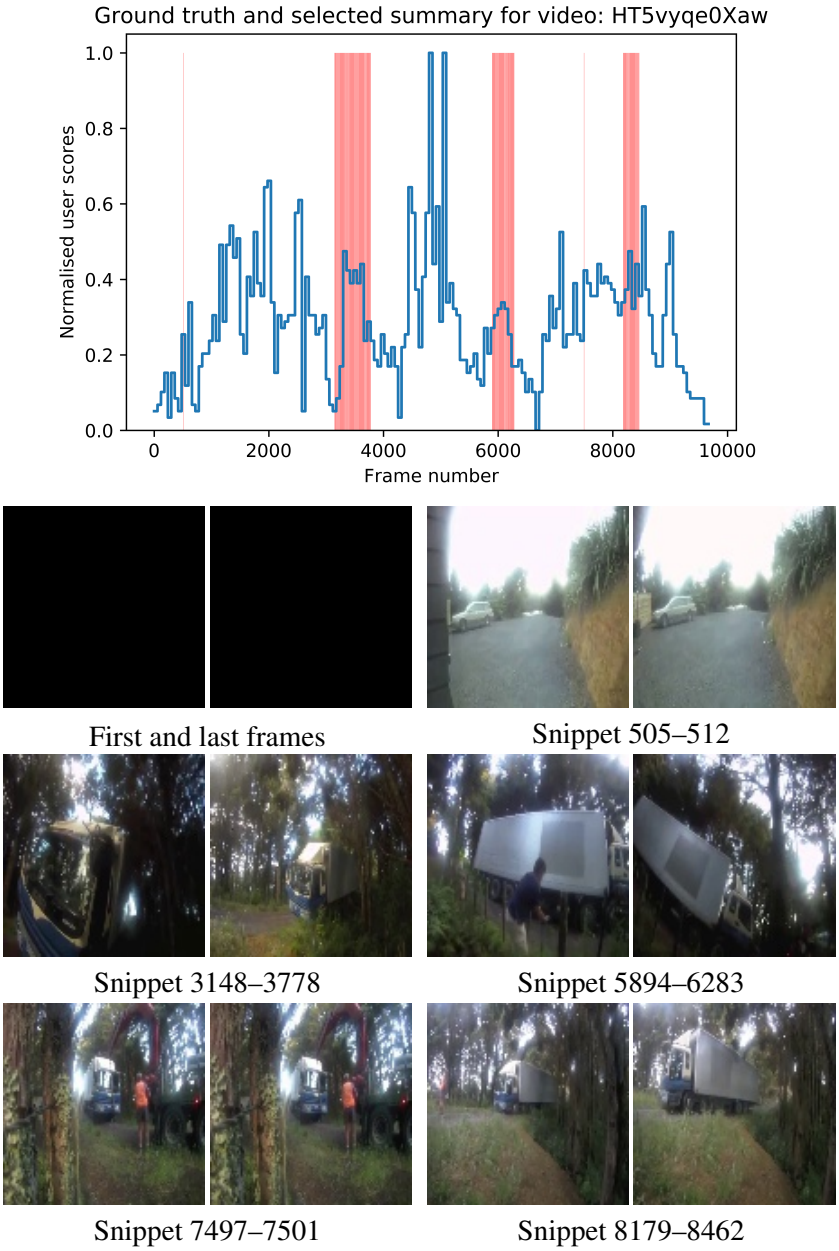


Figure 6.9: Sample results from TvSum50 dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a good summary i.e. a high F1-score.

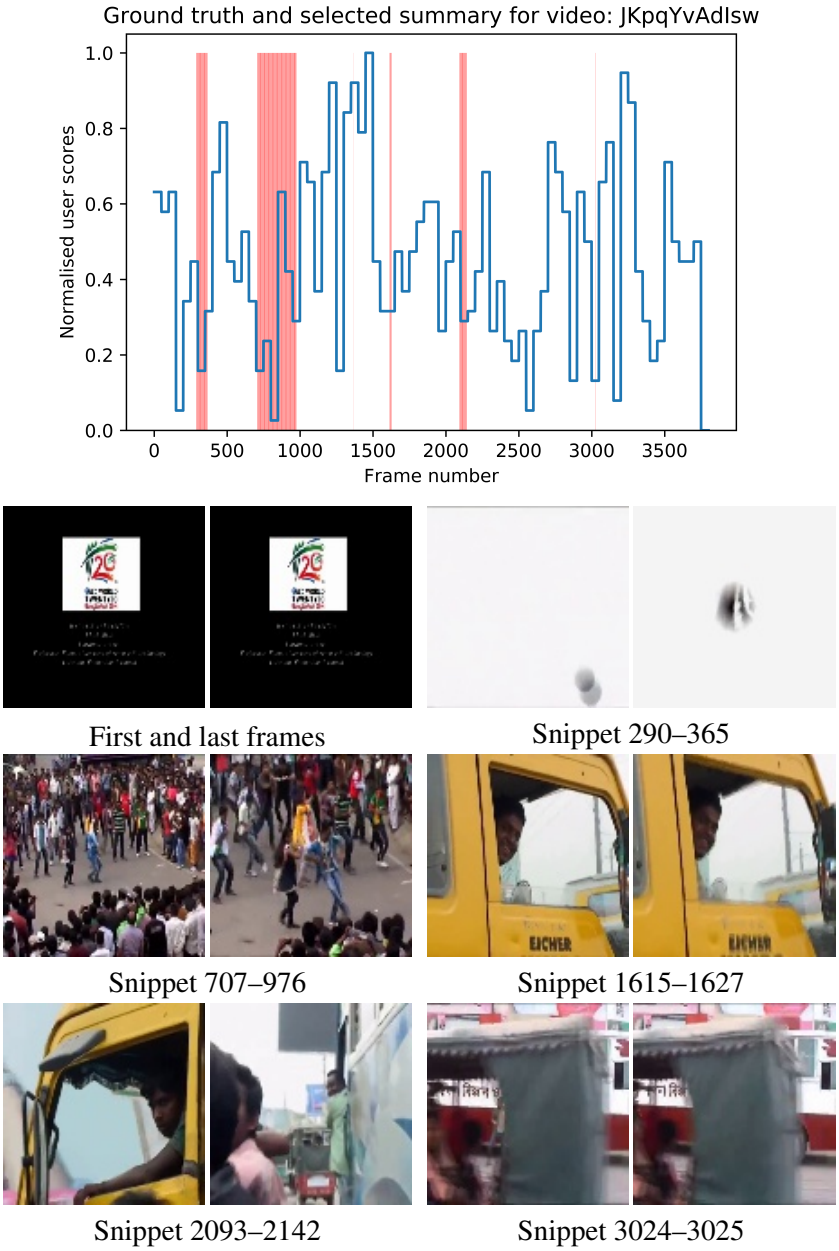


Figure 6.10: Sample results from TvSum50 dataset. The plots show the average user score per frame and the red rectangles signify which frames the model chose as a summary. The images in each column are the first and last frames (in pair) of input video and snippets of summary selected by the model. The numbers for each snippet indicate the frame numbers of first and last frame of the corresponding snippet. This is an example of a bad summary i.e. a low F1-score.