UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Multimodal Video Summarization and Highlight Detection

Diploma Thesis

Spyridon Barmpakos

Supervisor: Gerasimos Potamianos

January 2025

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Multimodal Video Summarization and Highlight Detection

Diploma Thesis

Spyridon Barmpakos

Supervisor: Gerasimos Potamianos

January 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Πολυτροπική Συνόψιση Βίντεο και Εντοπισμός Στιγμιοτύπων Ενδιαφέροντος

Διπλωματική Εργασία

Σπυρίδων Μπαρμπάκος

Επιβλέπων: Γεράσιμος Ποταμιάνος

Ιανουάριος 2025

Approved by the Examination Committee:

Supervisor **Gerasimos Potamianos**
Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Elias Houstis**
Emeritus Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Ioannis Pavlopoulos**
Assistant Professor, Department of Informatics, Athens University of Economics and Business

Acknowledgements

First and foremost, I would like to express my sincere gratitude to Charalampos Antoniadis from King Abdullah University of Science and Technology for his invaluable contribution to this project.

Additionally, I would like to thank Professor Gerasimos Potamianos for his supervision during my thesis, as well as Professor Elias Houstis, whose lessons greatly influenced my academic interests and provided me with valuable knowledge.

Lastly, I would like to thank my family and friends for their love and support, which helped me through this journey.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Spyridon Barmpakos

Diploma Thesis

Multimodal Video Summarization and Highlight Detection

Spyridon Barmpakos

Abstract

In an era dominated by digital media, video consumption has become an integral part of daily life. However, despite the valuable information videos offer throughout their length, watching them in full can often prove tedious. This is why researchers in recent years have engaged themselves in the field of video summarization and highlight detection, that promises the localization of the most important segments of videos.

While most of the existing methods are based on supervised learning with manually annotated datasets, such datasets are limited due to high costs and time consumed by human annotation. Recently, some researchers have explored weak-supervision and Reinforcement Learning to overcome this limitation. Additionally, instead of using the video frames as unimodal input, they built models that could handle both the video frames and the video transcript as multimodal input, in order to improve the performance of the models.

Despite the fact that there is a lot of research on both supervised and unsupervised unimodal models, there is a research gap regarding the unsupervised multimodal approaches. To this end, we present a new multimodal approach that brings together Transformers, Large Language Models and modality fusion to generate video summaries and locate key highlights. Our pipeline leverages visual frames and transcripts of un-annotated raw videos and is trained under a Reinforcement Learning framework. In our experiments, we demonstrated that incorporating the text modality significantly improves the model's performance compared to when it is not used. Additionally, we showcased the advantages of using a Transformer architecture compared to a Recurrent Neural Network architecture within the scope of our study.

Keywords:

Deep Learning, Computer Vision, Natural Language Processing, Video Summarization, Highlight Detection, Reinforcement Learning, Modality Fusion

Διπλωματική Εργασία

Πολυτροπική Συνόψιση Βίντεο και Εντοπισμός Στιγμιοτύπων Ενδιαφέροντος

Σπυρίδων Μπαρμπάκος

Περίληψη

Σε μια εποχή που κυριαρχείται από τα ψηφιακά μέσα, η κατανάλωση βίντεο έχει γίνει αναπόσπαστο μέρος της καθημερινής ζωής. Παρόλο που τα βίντεο συχνά περιέχουν πολύτιμες πληροφορίες, η παρακολούθηση ολόκληρου του περιεχομένου τους μπορεί να είναι χρονοβόρα και κουραστική. Αυτός είναι ο λόγος για τον οποίο οι ερευνητές τα τελευταία χρόνια ασχολούνται με τον τομέα της συνόψισης βίντεο και του εντοπισμού σημείων ενδιαφέροντος, ο οποίος υπόσχεται τον εντοπισμό των πιο σημαντικών τμημάτων των βίντεο.

Ενώ οι περισσότερες από τις υπάρχουσες μεθόδους βασίζονται στην επιβλεπόμενη μάθηση με χειροκίνητα επισημειωμένα σύνολα δεδομένων, τα δεδομένα αυτά είναι περιορισμένα λόγω του υψηλού κόστους και του χρόνου που καταναλώνεται από την διαδικασία της ανθρώπινης επισημείωσης. Πρόσφατα, ορισμένοι ερευνητές διερεύνησαν μεθόδους ενισχυτικής μάθησης και μάθησης με ασθενή επίβλεψη. Επιπλέον, εκτός από τη χρήση των καρέ του βίντεο ως μονοτροπική είσοδο, χρησιμοποίησαν το απομαγνητοφωνημένο κείμενο σε συνδυασμό με τα καρέ του βίντεο, δημιουργώντας μια πολυτροπική είσοδο για τα μοντέλα. Παρά το γεγονός ότι έχει γίνει αρκετή έρευνα τόσο για τα εποπτευόμενα όσο και για τα μη εποπτευόμενα μονοτροπικά μοντέλα, υπάρχει ένα ερευνητικό κενό όσον αφορά τις μη εποπτευόμενες πολυτροπικές προσεγγίσεις. Για τον σκοπό αυτό, παρουσιάζουμε μια νέα πολυτροπική προσέγγιση που συνδυάζει Transformers, Μεγάλα Γλωσσικά Μοντέλα και τεχνικές πολυτροπικής σύντηξης για τη δημιουργία συνόψισης των βίντεο και τον εντοπισμό των σημείων ενδιαφέροντος.

Το μοντέλο που δημιουργήσαμε αξιοποιεί τα καρέ και το απομαγνητοφωνημένο κείμενο από μη επισημειωμένα, ακατέργαστα βίντεο. Ακόμη, για την εκπαίδευση του μοντέλου χρησιμοποιήσαμε ενισχυτική μάθηση. Στα πειράματά μας δείξαμε ότι η εκμετάλλευση του απομαγνητοφωνημένου κειμένου βελτιώνει σημαντικά την απόδοση του μοντέλου, σε αντίθεση με την χρήση μόνο των καρέ του βίντεο ως είσοδο. Επιπλέον, αναδείξαμε τα πλεονεκτήματα της χρήσης της αρχιτεκτονικής Transformer σε σύγκριση με τα επαναληπτικά νευρωνικά

δίκτυα.

Λέξεις-κλειδιά:

Βαθιά Μάθηση, Όραση Υπολογιστών, Επεξεργασία Φυσικής Γλώσσας, Συνόψιση Βίντεο, Εντοπισμός Σημείων Ενδιαφέροντος, Ενισχυτική Μάθηση

Table of contents

List of figures

List of tables

Abbreviations

A2SUMM	Align and Attend Multimodal Summarization
AREDSUM	Adaptive Redundancy-Aware Iterative Sentence Ranking for Extractive Document Summarization
BERT	Bidirectional Encoder Representations from Transformers
biLSTM	bidirectional Long Short-Term Memory
CNN	Convolutional Neural Network
DSN	Deep Summarization Network
kTau	Kendall τ
LLM	Large Language Model
LSTM	Long Short-Term Memory
RL	Reinforcement Learning
RoBERTa	Robustly Optimized BERT Pre-training Approach
sRho	Spearman's ρ
S3D	Separable 3D Convolution
RNN	Recurrent Neural Network

Chapter 1

Introduction

1.1 Motivation

As digital media continues to dominate daily life, videos have become a central source of information and entertainment for people. Yet, watching the full duration of them can be very time-consuming. To tackle this, researchers in recent years have engaged themselves in the fields of video summarization and highlight detection. These areas aim to automatically detect the most informative and engaging parts of videos, and in the case of video summarization, generate a shorter, more condensed version of the video.

In recent studies [21], [22], [23],[24], [25], deep learning models are trained on manually annotated datasets in a supervised manner to generate video summaries. The main constrain in this case is that annotated video summarization datasets are quite limited, as the process of employing human annotators is both costly and time-consuming [26], [27].

To address this limitation, researchers have developed models that utilize weak-supervision or Reinforcement Learning to locate the important parts of videos and generate video summaries [28], [29].

Another key-component of the research on video summarization is the way the models process the input videos. A video consists of three essential components: the visual frames, the speech (transcript), and the audio. Some works only use the frames of the video as input (unimodal models) [22], [28], while others use the frames along with the transcript (multimodal models), or they augment the dataset by using models to generate descriptive captions for each frame [21], [23], [24], [25].

Based on the above, we conclude that there is a significant amount of research on uni-

modal and multimodal models trained using supervised learning, as well as unimodal models trained in an unsupervised manner. However, there is limited research on multimodal models trained in an unsupervised way. Following this under-researched direction could lead to the development of models that can exploit multimodal information of a vast amount of raw videos, hence leading to robust models.

This remark motivated us to develop a multimodal model that leverages Transformers, Large Language Models and modality fusion techniques to generate video summaries and detect video highlights. Our model exploits the frames and the transcripts of raw unannotated videos, and through a Reinforcement Learning Framework learns to create concise summaries and identify key highlights in the videos. Despite the fact that we use un-annotated videos for our training dataset [16], we use annotated videos to evaluate our model on the two tasks.

1.2 Related Work

The field of video summarization has seen significant advancements in recent years, leading to a lot of different approaches. Here, we will focus more on the works related to multimodal video summarization. We will start by exploring the multimodal supervised-learning approaches, without diving deep into each one. However, the unsupervised approaches are going to be explained in greater detail.

"SUSiNet" is a multi-task spatio-temporal network that can jointly tackle the spatio-temporal problems of saliency estimation, action recognition, and video summarization [22].

The "UMT" presents the first unified framework for joint video moment retrieval and highlight detection using bottleneck transformers [23].

The "CLIP-It" model introduces a language-guided multimodal transformer for generic and query-focused video summarization [24].

The "Clover" model introduces a Correlated Video-Language pre-training method that improves cross-modal feature alignment and fusion via a novel tri-modal alignment pre-training task [21].

In [25], the model generates both video and text summaries from an input video and its subtitles, with each summary enhancing the other through the fusion of the two modalities. The researchers propose a fusion mechanism called "align and attend", which uses a masking approach: visual features can only attend to textual features when they are temporally

aligned—that is, when they appear at the same time in the video.

The above was a brief overview of supervised multimodal video summarization approaches. However, as mentioned above, a major challenge in video summarization is the limited availability of annotated datasets, which can be explained by the expensive annotation process that requires manual labor. This led some researchers to believe that supervised learning techniques are not enough to unlock the full potential of the deep learning models in video summarization. Consequently, they have proposed approaches that train models without annotated datasets, although they still use such datasets for evaluation.

In [29], researchers focus solely on instructional videos and argue that the key parts of videos on a specific topic will be similar across different videos on that topic. By comparing multiple videos on the same topic, they create synthetic annotations based on these shared key elements, which then serve as the ground-truth summary. They then use these pseudo-summaries as weak supervision. As mentioned above, they use a manually annotated dataset for evaluation. In the fusion process, the video is segmented into parts, and a representation of each segment is created by combining and averaging its visual and textual features (transcript). These representations are then fed into a transformer encoder, which evaluates the saliency of each segment and the summary is generated accordingly.

The work of [28] only considers the visual features of videos, focusing on a unimodal summarization task. Building on previous research, the researchers argue that human-generated summaries are both diverse—meaning the selected segments are not repetitive—and representative, capturing most of the content shown in the original video. In their work, they develop formulas to quantify the presence of these characteristics in a video summary as a score, which is then used as a reward to train an LSTM-based summarization network, using Reinforcement Learning.

Our method builds on [25] and [28] by utilizing a multimodal Transformer architecture for the summarization network and by extending the reward function.

1.3 Thesis Contribution

The contributions of this thesis can be summarized as follows:

1. We developed a multimodal model that can generate video summaries and locate highlights. The model is trained using both the frames and transcripts of raw videos as input,

utilizing a Reinforcement Learning framework.

2. We use a sliding window approach to address the text sequence length limitation of the BERT language model [30].

3. We compare unimodal and multimodal models using various metrics to identify their respective strengths and weaknesses.

1.4 Thesis Structure

The structure of the thesis is as follows:

Chapter 1: Introduction - Describes the motivation of the research, outlines the contributions and presents the related work.

Chapter 2: Background - Presents the background concepts that are needed for understanding the proposed method.

Chapter 3: Proposed Method - Provides a detailed description of the proposed pipelines and their implementation.

Chapter 4: Data and Evaluation Setup - Presents the datasets and describes the benchmark that will be used to evaluate the models.

Chapter 5: Experiments and Results - Provides details regarding the implementation, and presents the experiments and the results.

Chapter 6: Conclusions - Provides Conclusions based on the results of the previous chapter and suggests future steps of this research work.

Chapter 2

Background

In this chapter, we will outline the fundamental models and mathematical concepts that form the basis of our work.

2.1 Embeddings

The first topic we will explore is Vector Embeddings, which are a fundamental element in Machine Learning. At an abstract level, Embeddings are a method to represent concepts in a high-dimensional area as vectors. Those concepts can be in the form of words, texts, images, videos, or virtually any type of data.

The strength of embeddings lies in their ability to position similar ideas as close vectors, while dissimilar ones are farther apart (as can be seen in Figure 2.1). This practice enables the machine-learning models to recognize the meaning of each concept by its relative position in this high-dimensional space. In the later sections, we will talk about how these Embeddings are created.

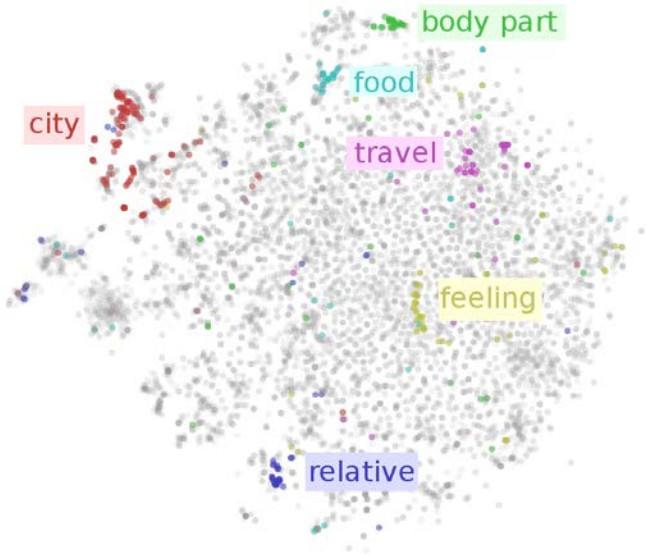


Figure 2.1: 3D Word Embeddings that form a cloud. The colored segments indicate the position of words that concern a certain topic.

2.2 Neurons and Feedforward Neural Networks

2.2.1 Neurons

The research of biological neurons and the attempt to model them with mathematical tools eventually led to the creation of the first artificial neuron by McCulloch and Pitts [31], which was soon followed by the Perceptron model. In Figure 2.2 we can see the similarities between the Biological Neuron and the Artificial Neuron

The transfer function of the Perceptron model is the following:

$$y = f(\sum_{i=1}^{n} w_i x_i + b)$$

where y is the output of the Neuron, x_i are the inputs, w_i and b are the weights and bias respectively, and f is the activation function. In essence, it is a weighted sum transformed by a function f.

If the activation function is linear, the neuron performs a linear transformation of the inputs, which lets it make decisions based on whether the sum is above or below a threshold. This can be used for binary classification problems.

If the activation function is non-linear, the neuron gains the ability to introduce non-linearity into its output, which lets it perform more complex decision-making, by approximating non-linear functions.

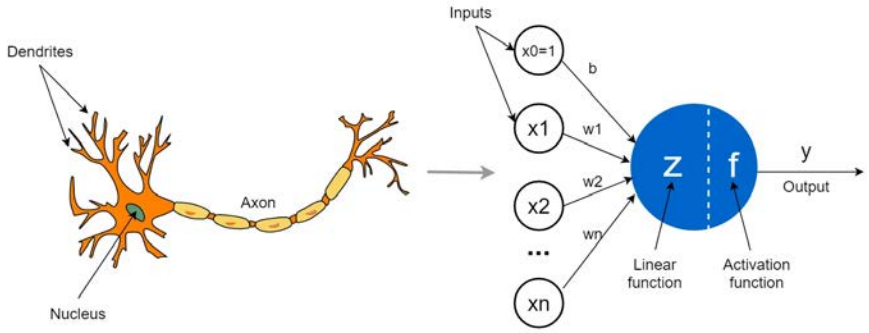


Figure 2.2: Similarities between the Biological Neuron (left) and the Artificial Neuron (right) (Figure from [1]).

.

2.2.2 Feedforward Neural Networks

The capabilities of a single neuron are quite limited. To overcome this, neurons are interconnected to create feedforward neural networks, as shown in Figure 2.3. By working together, these networks can model and solve much more complex problems, by approximating very complex functions.

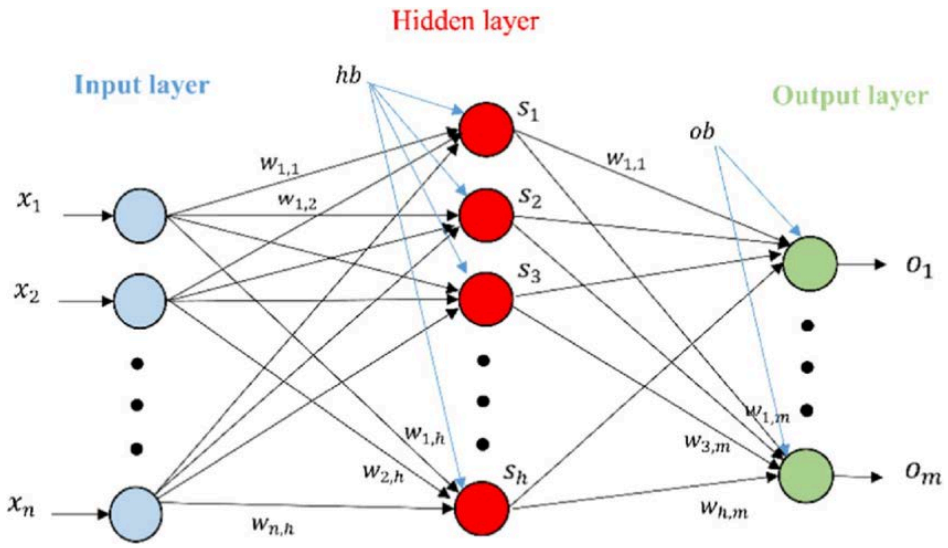


Figure 2.3: A Feedforward Neural Network with one hidden layer (Figure from [2]).

We have discussed how the model generates outputs by creating linear or non-linear combinations of the weighted sums of its inputs. A key challenge in machine learning is determining how to adjust these weights effectively to approximate the desired function.

This process is known as Backpropagation or the backward pass (as opposed to the

forward pass). This process starts by performing a forward pass with randomly initialized weights, and then calculating the error between the predicted output and the desired output. The derivative of the error is then calculated with respect to each weight and bias, showing the direction that each weight and bias has to be adjusted to minimize the error. In a process called gradient descent, the weights and biases are adjusted in this direction. The process is repeated many times until the error has reached a minimum threshold.

2.3 The Need for Processing Sequential Data

In many domains, data are sequential, meaning each element's order matters. Examples of sequential data include time-series data, text, and audio signals. To process this type of data, the models need to capture dependencies between elements and understand how different parts of the sequence influence each other.

Despite their effectiveness in processing independent data points, feedforward neural networks cannot process sequential data. This limitation led researchers to design new architectures such as Recurrent Neural Networks and Transformers that can handle sequential and contextual dependencies in the data.

2.4 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) work similarly to Feedforward Neural Networks, with the difference that the output depends not only on the current input, but also on past inputs. This is achieved through a hidden memory that is updated in each forward pass by storing information that is used by the model combined with the current input to calculate the output. In Figure 2.4, we can see an RNN unfolded through different time steps.

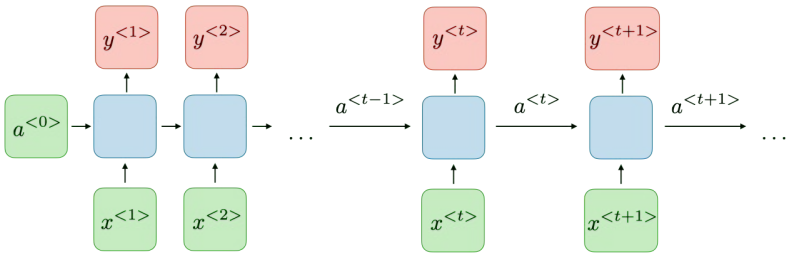


Figure 2.4: A simple RNN unfolded through time (Figure from [3]).

Recurrent Neural Networks became very popular because they could process sequences

of arbitrary length while having a fixed size. Subsequently, they were used a lot in Natural Language and Audio Processing. However, this RNN architecture, due to its simplicity, tends to "forget" information from the distant past, also known as the vanishing gradient problem. It can struggle with instability when the flow of information grows uncontrollably, leading to what is known as the exploding gradient problem. To overcome these issues, researchers invented more complex RNN architectures such as the Long Short-term Memory Networks (LSTMs), which help model the complex long-term relationships between different elements in a sequence.

2.4.1 LSTM

To address the limitations of simple RNNs, LSTMs [32] introduce a cell state (C in Figure 2.5), separate from the hidden state (h in Figure 2.5), that stores information. Additionally, the flow of information in the cell is controlled by three gates:

- **Forget Gate**: Determines what information should be discarded.
- **Input Gate**: Determines what new information from the current input should be introduced.
- **Output Gate**: Determines how the cell state will influence the current output and hidden state.

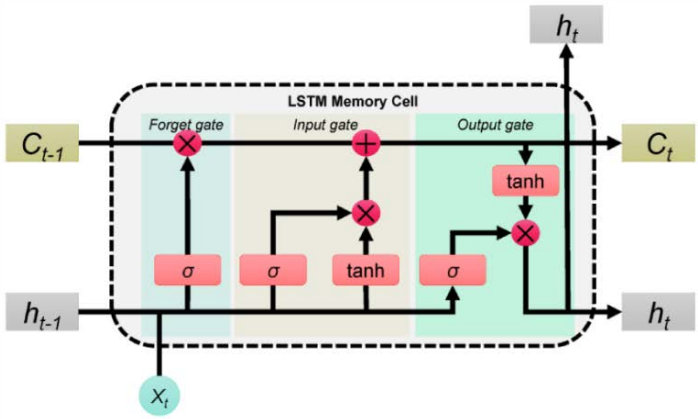


Figure 2.5: The LSTM cell (Figure from [4]).

This process lets the LSTMs model long-term relationships between elements in a sequence reliably, while minimizing the effects of vanishing and exploding gradients, as the flow of information is regulated by the gates and the Cell.

This sophisticated architecture led to a breakthrough in Natural Language Processing and many other domains that required the processing of sequences of data.

However, LSTMs process the sequences one element at a time, which limits the ability to utilize parallel computation, as each step is dependent on the previous step. Additionally, the problem of vanishing and exploding gradients may be addressed but is not entirely solved. Moreover, after processing a sequence, LSTMs encode the information in a single vector. This can lead to an information bottleneck, as important information might be lost in long sequences.

Those limitations were largely addressed by the Transformer models.

2.5 Transformers

Transformers were first introduced in 2017 [5] and since then, have dominated the area of Natural Language Processing, leading to breakthroughs such as Large Language Models (LLMs). Their popularity stems from the fact that they effectively address many of the issues associated with LSTMs, making them highly suited for sequence processing tasks.

In the original paper, the Transformer architecture included a Transformer Encoder and a Transformer Decoder, which are used together for sequence generation tasks. The Encoder processes and encodes the input sequence, while the Decoder takes this encoded representation and generates the output sequence.

In our work, we will not use a Transformer Decoder, as we will not perform any sequence generation, so we will skip its explanation.

2.5.1 Transformer Encoders

In Figure 2.6, the structure of a Transformer encoder is illustrated. It consists of many modules which are covered next.

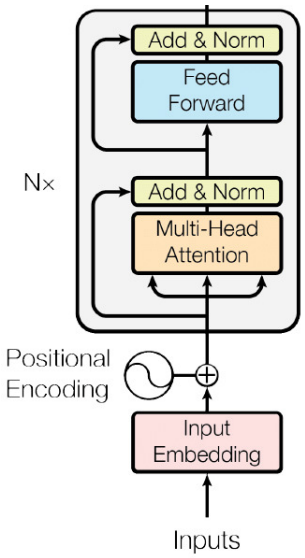


Figure 2.6: Transformer Encoder (Figure from [5]).

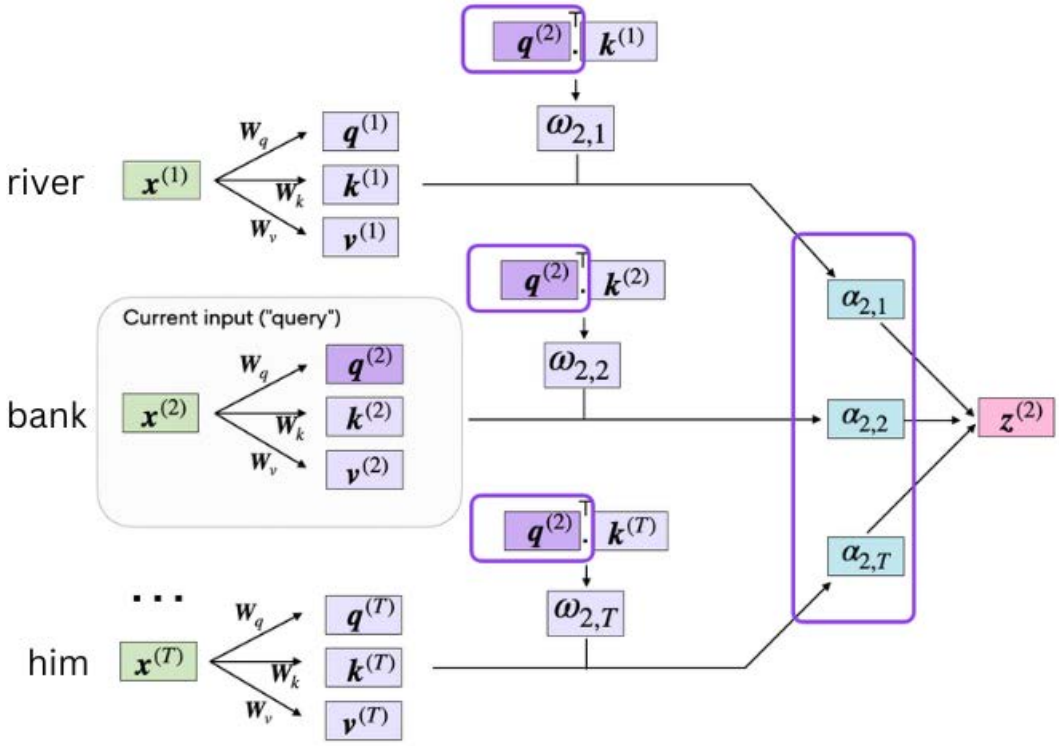


Figure 2.7: Forward pass of the self-attention mechanism for the word “bank” (Figure adapted from [6]).

Self-attention

Self-attention is the key innovation that led to the popularity of Transformer’s architecture. We will examine the case where the inputs are words, even though every other form of data works in the same way.

The core idea of self-attention is to let words interact with each other, in such a way that their meanings get contextualized. This is achieved by shifting their vector towards the direction of the vectors of related words in the text.

On the left side of Figure 2.7, we see the input words ***river bank ... him***, that are represented by the vectors $x^{(1)}, x^{(2)}, \ldots, x^{(T)}$ respectively.

Self-attention uses three weight matrices, more specifically $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v$, which are learned during training. These matrices are multiplied with each input vector to produce different representations, that serve different purposes, namely the Query, Key, and Value.

- Query: $\mathbf{q}^{(i)} = \mathbf{x}^{(i)}\mathbf{W}_q$ for $i \in [1, T]$
- Key: $\mathbf{k}^{(i)} = \mathbf{x}^{(i)}\mathbf{W}_k$ for $i \in [1, T]$
- Value: $\mathbf{v}^{(i)} = \mathbf{x}^{(i)}\mathbf{W}_v$ for $i \in [1, T]$

The index i refers to the index of the words in the text sequence, which has length T.

Let *d* be the dimensionality of the input vectors ($\mathbf{x}^{(i)}$). Let d_q, d_k, d_v be the dimensionality of the vectors $\mathbf{q}^{(i)}, \mathbf{k}^{(i)}, \mathbf{v}^{(i)}$ respectively. Because of that, the weight matrices $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v$ need to have dimensionality $d \times d_q, d \times d_k, d \times d_v$ respectively. In the simplest case (single headed attention), $d_q = d_k = d_v = d_{model}$, where d_{model} is the size of the output vector.

Figure 2.7 illustrates the process of generating the output vector for the second word (***bank***, $\mathbf{x}^{(2)}$). The output $\mathbf{z}^{(2)}$ for this specific token should be a contextualized version of the input $\mathbf{x}^{(2)}$.

The word ***bank*** can either mean a river bank or a financial establishment, however, the vector that represents it should be unique. In this specific example, the word ***river*** is right before it, so the attention mechanism should allow the word ***river*** to interact with the word bank, so that the generated vector $\mathbf{z}^{(2)}$ better represents the concept of the bank as a river ***bank***. On the contrary, the last word of the sentence (him) most likely doesn't interfere with the meaning of ***bank***, so it should not influence the output vector.

To achieve that, the model has to determine the influence of each word on the second word and quantify it by assigning weights. The model first needs to compute the dot product of the query vector of the second word with the key vector of every word in the sequence.

$$w_{ij} = \mathbf{q}^{(i)\intercal}\mathbf{K}^{(j)}, j \in [1, T], \text{in this case } i = 2$$

To achieve the desired result, the key vector $\mathbf{k}^{(\mathbf{i})}$ of the words that should influence the meaning of the current word must have a high dot product with the query vector $\mathbf{q}^{(2)}$. This

is made possible by the weight matrices $\mathbf{W}_q$ and $\mathbf{W}_k$, which are trained to ensure that these two vectors produce a large dot product in such cases.

The next step is to normalize the unnormalized attention weights W to obtain the normalized attention weights α. On top of that, the factor $\frac{1}{\sqrt{d_k}}$ is used to normalize W before passing it through the softmax function, where d_k is the dimension of the input vectors.

$$a_{ij} = \text{softmax}(\frac{W_{i,j}}{\sqrt{d_k}}), j \in [1, T], \text{in this case } i = 2$$

The last step involves calculating $\mathbf{z}^{(2)}$, as the weighted sum of the value vectors $\mathbf{v}^{(\mathrm{j})}$ of the inputs.

$$\mathbf{z}^{(i)} = \sum_{j=1}^{T} \alpha_{i,j}\mathbf{v}^{(j)}, \text{in this case } i = 2$$

Following the same process for all the inputs, the model generates all the outputs of the self-attention $\mathbf{z}^{(i)}$.

To present the method in a more compact way, we introduce three matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V}$, which are constructed by stacking the vectors $\mathbf{q}^{(i)}, \mathbf{k}^{(i)}, \mathbf{v}^{(i)}$ respectively, for all values of i. The self-attention operation can then be defined in a single formula:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\intercal}{\sqrt{d_k}})\mathbf{V}$$

Because of the operations that are performed, this type of self-attention mechanism is also called **scaled-dot-product attention**.

Multi-Head Attention

Scaled-dot-product attention is a great way to contextualize the meaning of words, and the learned matrices $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v$ help the inputs interact with each other in the desired way. However, the relationships between words in a sentence can be very complicated, as shown in Figure 2.8. This highlights the need for a more complex process, that provides ways to capture many different relationships between words and contextualize them in various ways.

Multi-head attention does that by introducing different attention heads, each one responsible for capturing different kinds of relationships among the tokens in the sequence.

Let h be the number of heads and d_{model} be the size of the output of the attention mechanism. In multi-head attention, the queries, keys, and values are each linearly projected h times using different learned linear transformations $\mathbf{W}_q^i, \mathbf{W}_k^i, \mathbf{W}_v^i$ respectively, where $i \in [1, h]$.

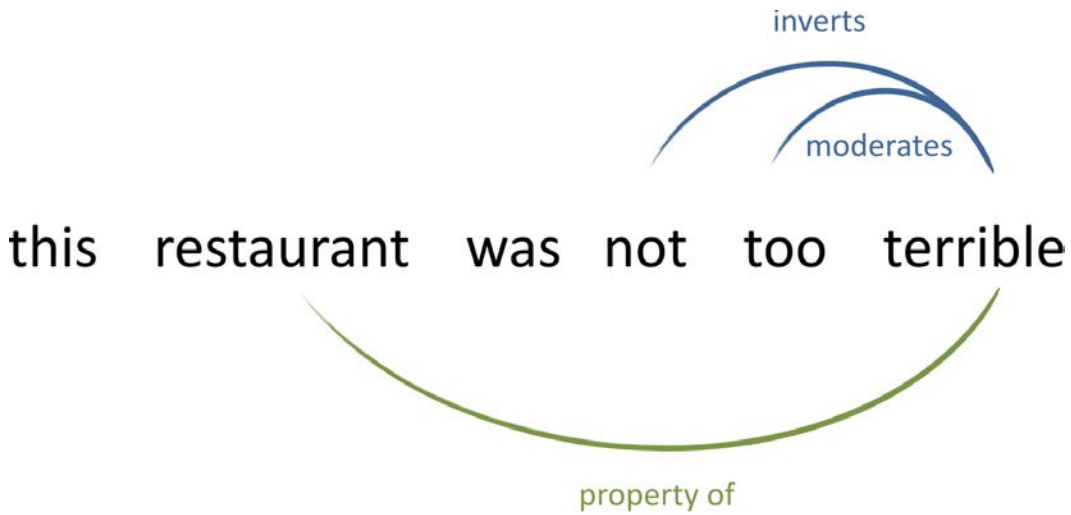


Figure 2.8: The relationship between words in a sentence, highlighting the need for Multi-Head Attention (Figure from [7]).

These projections map the inputs to d_k, d_k, and d_v dimensions respectively, where $d_k = d_v = d_{model}/h$.

The dimensions of the key and value vectors must always match, because the attention mechanism computes a dot product between them. After these projections, the attention function is applied individually to each of the h sets of projections, producing output values of dimension d_k.

Finally, the h outputs from the attention function are concatenated and passed through another linear projection $\mathbf{W}_O$.

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(head_1, ..., head_h)\mathbf{W_O}$$

$$\text{where } head_i = Attention(\mathbf{QW_q^i}, \mathbf{KW_k^i}, \mathbf{VW_v^i})$$

The overview of this mechanism is illustrated in Figure 2.9

Positional Encoding

Self-attention updates each sequence's vectors by incorporating information about their context. However, understanding the context of the tokens alone is not enough, because the position of the tokens within the sequence is equally important. To address this, researchers developed a mechanism to encode positional information directly into the embedding vector of each token.

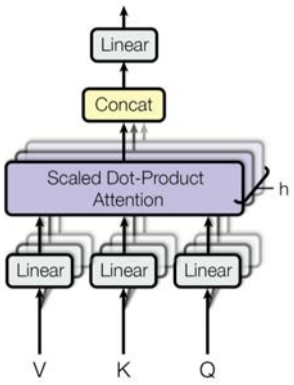


Figure 2.9: Multi-Head Attention mechanism overview (Figure from [5]).

To accomplish this, they introduced a new type of vector embedding, where each vector corresponds to a specific position in the sequence. This is achieved by utilizing sinusoidal waves of different frequencies, one for each dimension of the Positional Embedding vector. Another alternative is to make positional embeddings learnable parameters, so their values are updated during training.

The Positional Embedding is added to the "meaning" Embedding for each token and their sum is used as the input to the model.

The rest of the architecture

As illustrated in Figure 2.6, apart from the Multi-Head Attention sublayer, there is also a feedforward sublayer. Also, there is a residual connection [33] around each of the two sublayers, followed by layer normalization [34].

2.6 BERT Model

One of the most popular usages of Transformer Encoders is the BERT model (Bidirectional Encoder representation from Transformers) [30]. It is a pre-trained model that consists of a stack of Transformer Encoders, as shown in Figure 2.10. It was trained using a masked language modeling task, where sentences are presented with certain words masked, and the model's goal is to predict the masked words correctly. To accomplish this, a feedforward neural network layer and a softmax function were added on top of the Transformer Encoders.

The power of the BERT architecture lies in the fact that the trained version of it can be used in a variety of tasks that have nothing to do with masked language modeling. This is

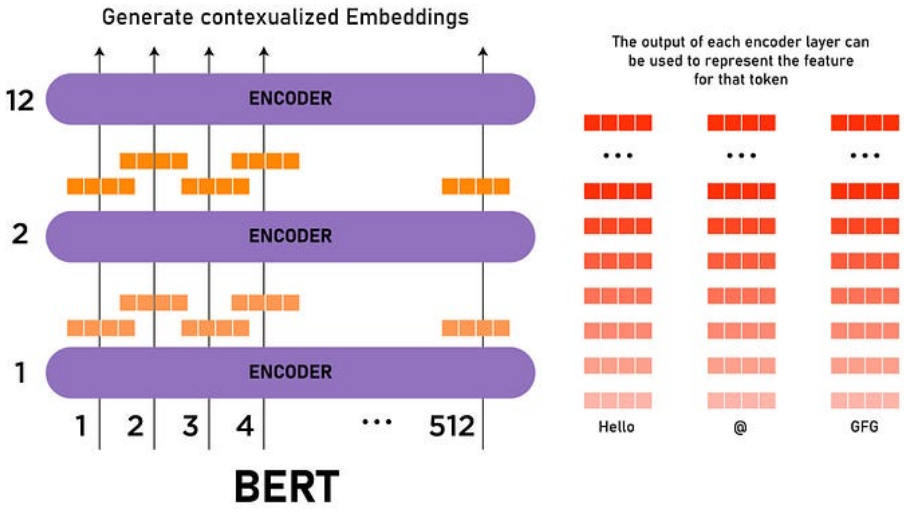


Figure 2.10: Illustration of the BERT architecture (Figure from [8]).

true, because learning to predict masked words correctly enabled it to generate rich feature vectors for each input token, injected with information about their context.

These enhanced features can be used in a variety of tasks, like sentiment analysis, spam detection, Named Entity Recognition, Part-of-Speech Tagging, Dependency Parsing and Extractive Text Summarization. All these tasks can be accomplished by adding other neural networks on top of the pre-trained BERT architecture and training the model for a few examples. Because BERT already captures a lot of information about the language, the model converges quickly and reaches the desired result.

2.7 AREDSUM Model

In this section, we will present the Adaptive Redundancy-Aware Iterative Sentence Ranking for Extractive Document Summarization (AREDSUM) model [9], which is illustrated in Figure 2.11.

This model is designed for extractive summarization, which means it creates a summary by selecting and combining the most important segments of the original text directly, without generating new content.

The work of [9] suggests that given an original text, the model needs to select sentences that have the right balance of *salience*, which represents how much information each sentence carries, and *redundancy*, which reflects how much of that information is already covered by other selected sentences.

To achieve that, ARDESUM utilizes a Document Encoder to generate contextualized features for all the sentences and then uses these features to rank the sentences.

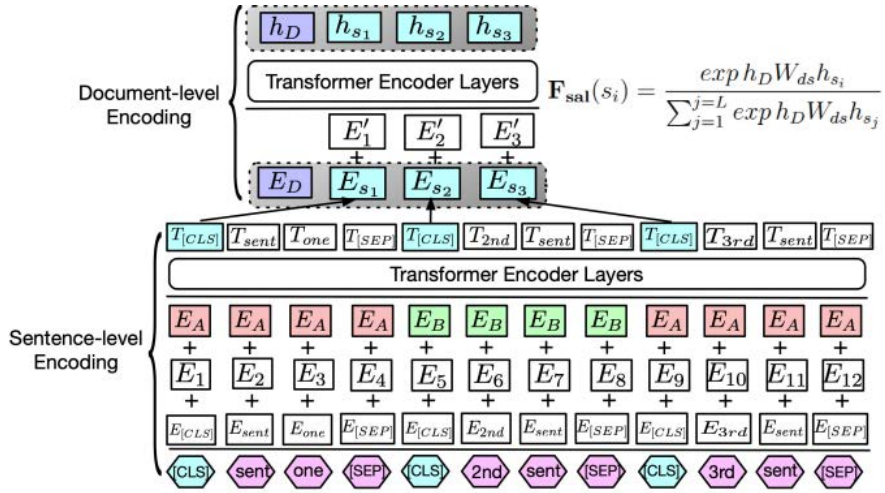


Figure 2.11: Illustration of the AREDSUM model. It shows that the Sentence-level Encoding features pass through the Transformer Encoder layers and the Document-level Encoding features are generated. The equation on the right shows how the saliency is calculated from this Document-level Encoding features (Figure from [9]).

2.7.1 Document Encoder

To encode the sentences, a [SEP] token is added at the end of each sentence to separate them. Additionally, a [CLS] token is placed at the beginning of each sentence to serve as a placeholder. The contextualized representation of the [CLS] token is then used to represent the meaning of each sentence.

Using the tokenizer of the pre-trained BERT model, embeddings are generated for each token. Similar to standard approaches, AREDSUM incorporates positional encodings E_i (see Section 2.5.1). However, it also introduces interval segment embeddings E_A and E_B to distinguish between sentences in odd and even positions within the document.

The final encoding for each token is computed as the sum of these three components: the token embeddings, positional encodings, and interval segment embeddings.

These combined encodings are then fed into the BERT model (see Section 2.6), which contextualizes them and generates global representations for the tokens, denoted as T_* (see Figure 2.11). As mentioned above, the representation tokens $T_{[CLS]}$ should be learned in a way that incorporates information about the sentences that follow them.

Let s_i represent the sentences of the original text. Now, let E_{s_i} be equivalent to the $T_{[CLS]}$ token that precedes the sentence s_i and E'_i be its positional encoding. An additional E_D embedding is added before the sequence of the E_{s_i} tokens, to act as the embedding of the whole document.

The sequence that consists of the E_D token followed by the E_{s_i} + E'_i tokens is then fed into another stack of transformer encoder layers. The transformer encoder outputs the final representations of the entire document, h_D, and each sentence h_{s_i}.

2.7.2 Sentence Ranker

Bilinear Matching

Bilinear matching is a process that relates two vectors through a learnable matrix. Their matching score is computed as

$$s(\mathbf{u}, \mathbf{v}) = \mathbf{u}^\intercal \mathbf{W} \mathbf{v}$$

where:

- $\mathbf{u} \in \mathbb{R}^n$: The first input vector
- $\mathbf{v} \in \mathbb{R}^m$: The second input vector
- $\mathbf{W} \in \mathbb{R}^{n \times m}$: A learnable matrix
- $s(\mathbf{u}, \mathbf{v}) \in \mathbb{R}$: A scalar output indicating the matching between $\mathbf{u}$ and $\mathbf{v}$

The matrix $\mathbf{W}$ allows for a more sophisticated interaction between the two inputs by capturing relationships between their dimensions, compared to simpler methods like dot products.

Saliency Ranking

Saliency denotes the importance of a sentence in a text and in the context of extractive summarization, it also indicates its probability of being selected for the summary. This probability is represented by the function F_{sal}, which is based on the bilinear matching between h_D and h_{s_i}, which is the output of the transformer after the document-level encoding, as described in Section 2.7.1.

$$F_{sal}(s_i) = \frac{\exp\ h_D \mathbf{W_{ds}} h_{s_i}}{\sum_{j=1}^{L} \exp\ h_D \mathbf{W_{ds}} h_{s_j}} \tag{2.1}$$

where $\mathbf{W_{ds}}$ is the learnable matrix that enables the bilinear matrix and L is the number of sentences in the text.

The softmax function normalizes the bilinear matching scores between the document representation h_D and the sentence representations h_{s_i}. This ensures that the probabilities $\mathbf{F_{sal}}$ for all the sentences sum to 1, which lets it be interpreted as the relative importance score of a sentence s_i within the document.

The learning objective is to maximize the log-likelihood of the summary sentences in the training data. So, given that $\hat{S}*$ is the list of the sentences that should be included in the summary, the loss of the model can be defined as:

$$\mathcal{L} = \sum_{s_i \in \hat{S}*} \log\ F_{sal}(s_i)$$

The training algorithm uses the gradient-descent algorithm to minimize this value.

To generate the summary, the AREDSUM model incorporates not only the saliency function, but also a module designed to identify the redundancy of each sentence. This module evaluates the extent to which the information in a given sentence is already covered by the summary prior to adding it. However, since this step is not relevant to our work, it will be omitted.

2.8 Convolutional Neural Networks

Another important category of data that cannot be handled effectively by the architectures we have covered are those with a grid-like structure, where local spatial relationships play a crucial role. The most common example of this type of data is images. Convolutional Neural Networks (CNNs) are specifically designed to process and analyze this type of data. Figure 2.12 presents an overview of the CNN pipeline.

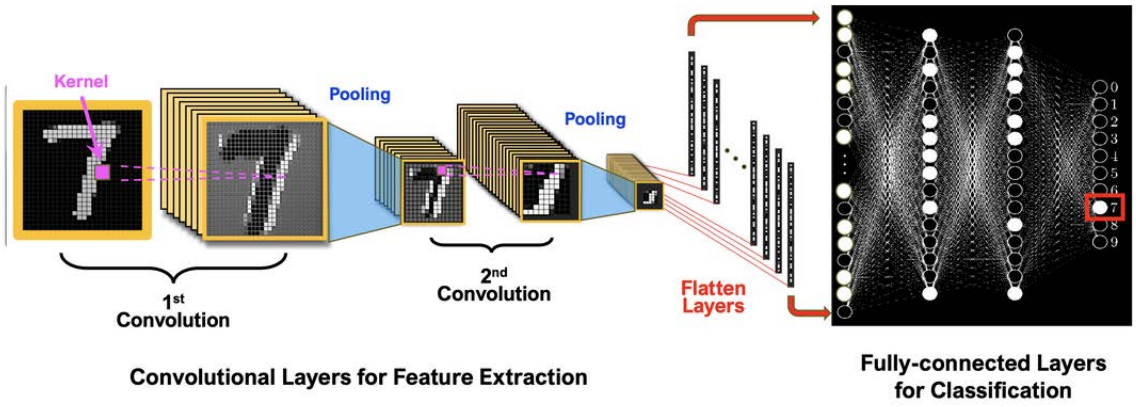


Figure 2.12: Overview of the CNN pipeline (Figure from [10]).

The main building blocks of the CNN pipeline are the convolutional and the pooling layers.

2.8.1 Convolutional Layers

The objective of the Convolutional Layers is to extract features from the input data by detecting patterns such as edges, textures and shapes. This is achieved by using trainable matrices called kernels. These kernels, also known as filters, slide across the input image and perform element-wise multiplication and then summation. The results of all these operations are stored in a separate matrix, also called feature map. An illustration of this operation is shown in Figure 2.13.

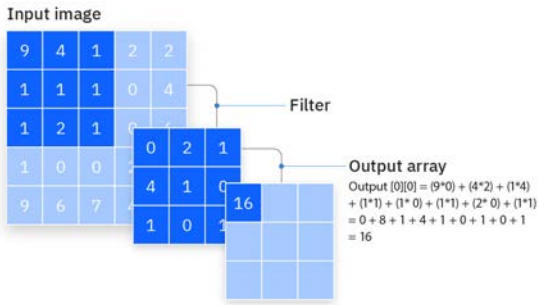


Figure 2.13: Convolution operation (Figure from [11]).

2.8.2 Max Pooling Layers

Max pooling is the most common form of pooling. Essentially, it is a downsampling operation, used to reduce the dimensions of feature maps.

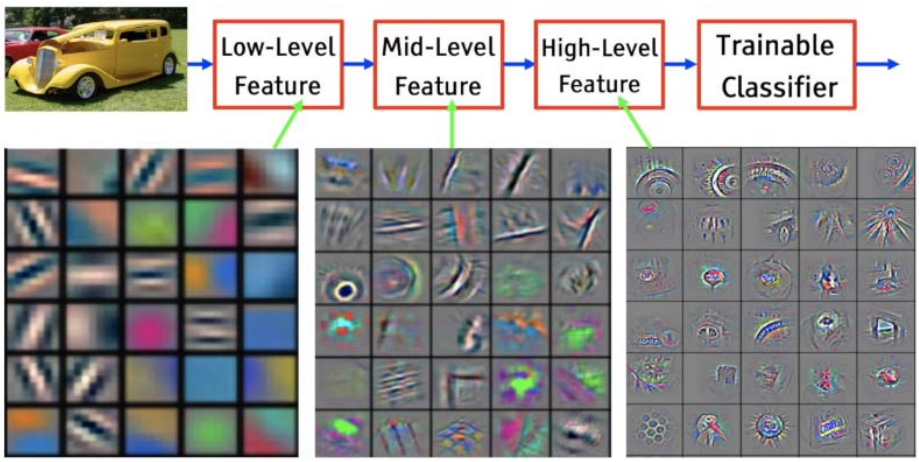


Figure 2.14: Visualization of feature extraction using convolutional layers (Figure from [12]).

This dimensionality reduction helps in keeping the most important information of the feature map, while reducing its size. In Figure 2.15, we see an example of a Max Pooling operation with a 2×2 filter. The filter slides over the feature map, and for each region it covers, it selects the maximum value.

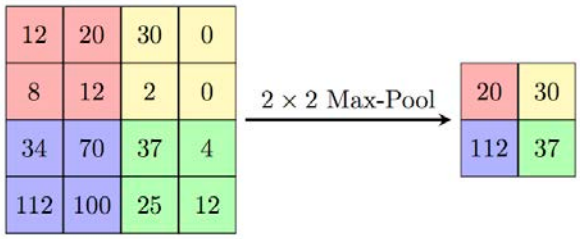


Figure 2.15: 2×2 Max Pooling operation (Figure from [13]).

2.8.3 Feature Extraction

Figure 2.12 illustrates how the modules explained in the two previous subsections work together to form the CNN pipeline. As shown in the figure, after many consecutive applications of the convolutional and the max pooling operations, the resulting representation can be fed into a feedforward neural network, in order to perform classification.

However, after the network is trained in some downstream tasks, such as image classification in this case, the vector that is fed to the feedforward neural network can be used as the embedding of the input image, as explained in Section 2.1.

A trained CNN can be utilized to convert images into vector embeddings that can then be applied to various downstream tasks, even if those tasks are unrelated to the original task

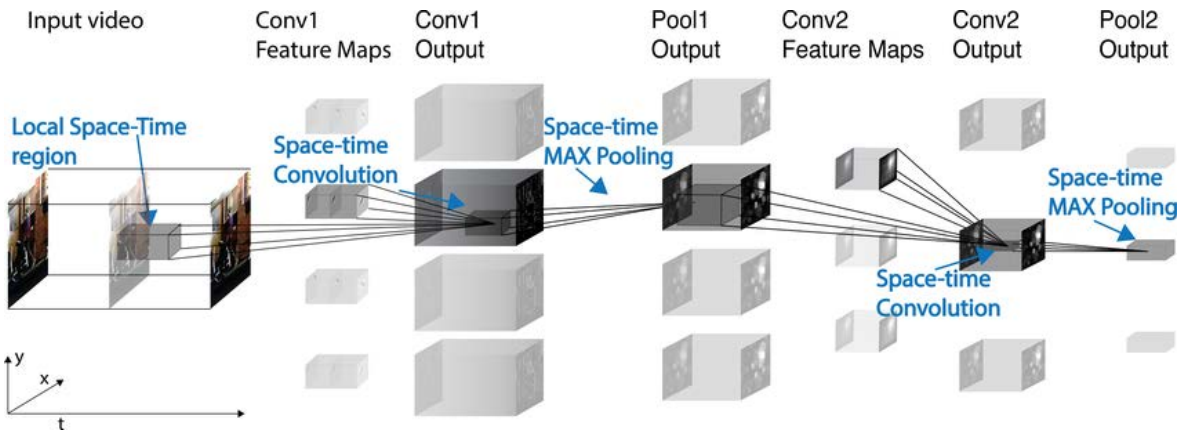


Figure 2.16: Extraction of 3D video features (Figure from [14]).

the CNN was trained for.

2.8.4 3D Features

In the previous sections, we briefly explained how CNNs use 2D convolutions to extract vector embeddings from images. However, when dealing with videos, which have both spatial and temporal dimensions, plain 2D CNNs are not sufficient, as they cannot capture the information across the frames of the videos and can only handle the frames independently, as images. However, by treating the frames as images, the temporal dependencies between them cannot be exploited.

To address this issue, 3D models, such as the Separable 3D Convolutional Network (S3D) are employed [14].

This network processes video frames as 2D matrices, treating the input video as a 3D matrix, which is essentially a stack of these 2D matrices. Instead of plain 2D convolutions, it employs **spatiotemporal separable convolutions**, where 3D convolutions are factorized into 2D spatial convolutions followed by 1D temporal convolutions. An illustration of the process is illustrated in Figure 2.16.

2.9 Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning, where the model (called agent) learns to make decisions, engaging with the environment. In supervised learning, the model learns through labeled data, while in reinforcement learning, the environment gives

the agent feedback through rewards or penalties based on the agent's actions.

The agent's aim is to maximize the total reward over time. To achieve that, it chooses the actions that bring the best outcome. The agent uses the trial-and-error approach to improve its performance, meaning it continuously updates its strategy, which is called a policy.

One of the main concepts of RL is the Markov Decision Process, which models the decision-making process through states, actions, and rewards. The agent starts in a certain state where it performs an action and then ends up in a different state, while also receiving a reward. Therefore, the agent over time aims to learn the optimal policy that maps states to actions, in such a way that the expected sum of future rewards is maximized. This calls for a careful balance of exploration, where the agent tries out new actions to get more information about the environment, and exploitation, where the agent focuses on actions that have previously given high rewards. A balance of these two factors is required for effective learning in the context of RL. The specific RL algorithms that will be used in our implementation, will be explained in the following chapters.

Chapter 3

Proposed Method

The proposed model is designed to perform video summarization and highlight detection and uses the video frames and the video's transcript as input. For video summarization, it will select the most important frames to create a concise summary, while in highlight detection, scores will be assigned to each frame, showing their importance in the video.

First, we'll introduce the model's architecture at a high level, followed by a detailed explanation of each component. Finally, we'll discuss the training framework.

3.1 Proposed Method, a Top-Down Approach

Following [28], we approach Video Summarization as a decision-making task. More specifically, we propose a pipeline that incorporates a Deep Summarization Network (DSN), which takes a video as input and generates probability distributions for the frames. Based on these probabilities, the Pipeline decides which frames to include in the summary, and which not to.

The model generates summaries, which are then evaluated by the reward functions. These rewards are incorporated into a reinforcement learning-based framework that trains the DSN.

There are two types of Pipelines that we are going to explore. The **unimodal** and the **multimodal** one. In the unimodal configuration, only the video frames are processed, while in the multimodal configuration, both the video frames and the transcript are processed. The Multimodal Pipeline is illustrated in Figure 3.1, while the Unimodal Pipeline is illustrated in Figure3.3.

For the Multimodal Pipeline, we employ a Transformer-based architecture as the DSN.

In the Unimodal Pipeline, there are two versions: one version uses a Transformer-based architecture as the DSN, while the other version uses an RNN-based architecture as the DSN. The following sections will explain the process in detail.

3.2 Feature extraction

The input to the models consists of the video frames and the video transcript, which will be referred to, as the visual and the textual modality, respectively. As described in Section 2.1, the raw input must first be transformed into embeddings for the model to be able to process it effectively. The following subsections provide a detailed explanation of how each modality is converted into embeddings.

3.2.1 Visual Features

In Section 2.8.4, we talked about the S3D model [14] that generates 3D features for videos. To generate the features for the input videos, we used an open-source implementation of the S3D feature extractor [35]. This implementation generates one feature vector for every second of the video. Therefore, we assume that the videos are subsampled at a rate of one frame per second, with each feature representing a single frame. Moving forward, when we refer to "frames," we are specifically referring to these subsampled frames. This means that a video of length n seconds will consist of n frames, with each frame corresponding to a feature.

The assumption that the video is subsampled to one frame per second is a convention to facilitate the interchangeable usage of the terms "frame features" and "features that correspond to each second of the video". When calculating the features of each second of the video, the S3D model considers all the frames present.

3.2.2 Textual Features

To extract the features of the transcript, we utilize a pre-trained LLM. More specifically, each subtitle of the video is fed into a RoBERTa model, which stands for "A Robustly Optimized BERT Pretraining Approach" [36], which is very similar to the BERT model explained in Section 2.6, but with slight differences in training, like larger batches and learning rates.

Similarly to the AREDSUM model, a [CLS] token is aggregated in the front of each input sentence and gets contextualized based on the meaning of the subsequent text. After inference, the representation of this [CLS] token serves as the embedding for the input sequence, which corresponds to the sentence. In our implementation, we segmented the transcript into subtitles and we calculated a feature for each subtitle.

In practice, we used the SparkNLP library to generate the embeddings. This library offers a Pipeline that enables fast and efficient generation of RoBERTa Embeddings for a large volume of texts.

3.3 Deep Summarization Network

The DSN takes the input embeddings and generates contextualized representations, which are then passed to a feedforward network that makes the final decision on whether each frame will be selected. Subsections 3.3.1, 3.3.2 and 3.3.3 will discuss the three types of DSNs, while Subsection 3.4 will describe how the outputs of these models are processed to create the final summary and highlights.

3.3.1 Multimodal Transformer (A2SUMM)

The **A**lign and **A**ttend Multimodal **Summ**arization (A2SUMM) model is introduced in [25] and was adapted to our use-case. Essentially, it is a transformer-based model that enables the fusion of the visual and text modalities via a mechanism called Alignment-Guided Self-Attention. The complete Pipeline that incorporates the A2SUMM model is illustrated in Figure 3.1.

Input Embeddings

Section 3.2 explained the process of extracting visual features from raw video data and textual features from subtitles. Since different models were used for these extractions, the resulting feature vectors for the two modalities are mapped to separate embedding spaces.

To prepare the features for modality fusion, a linear fully-connected layer is used to project the features into a common C-dimensional embedding space. Let M be the number of subtitle sentences and N be the number of the video frames. The generated video and text features are denoted as $\mathbf{V} \in \mathbb{R}^{N \times C}$ and $\mathbf{S} \in \mathbb{R}^{M \times C}$, respectively. Additionally, positional

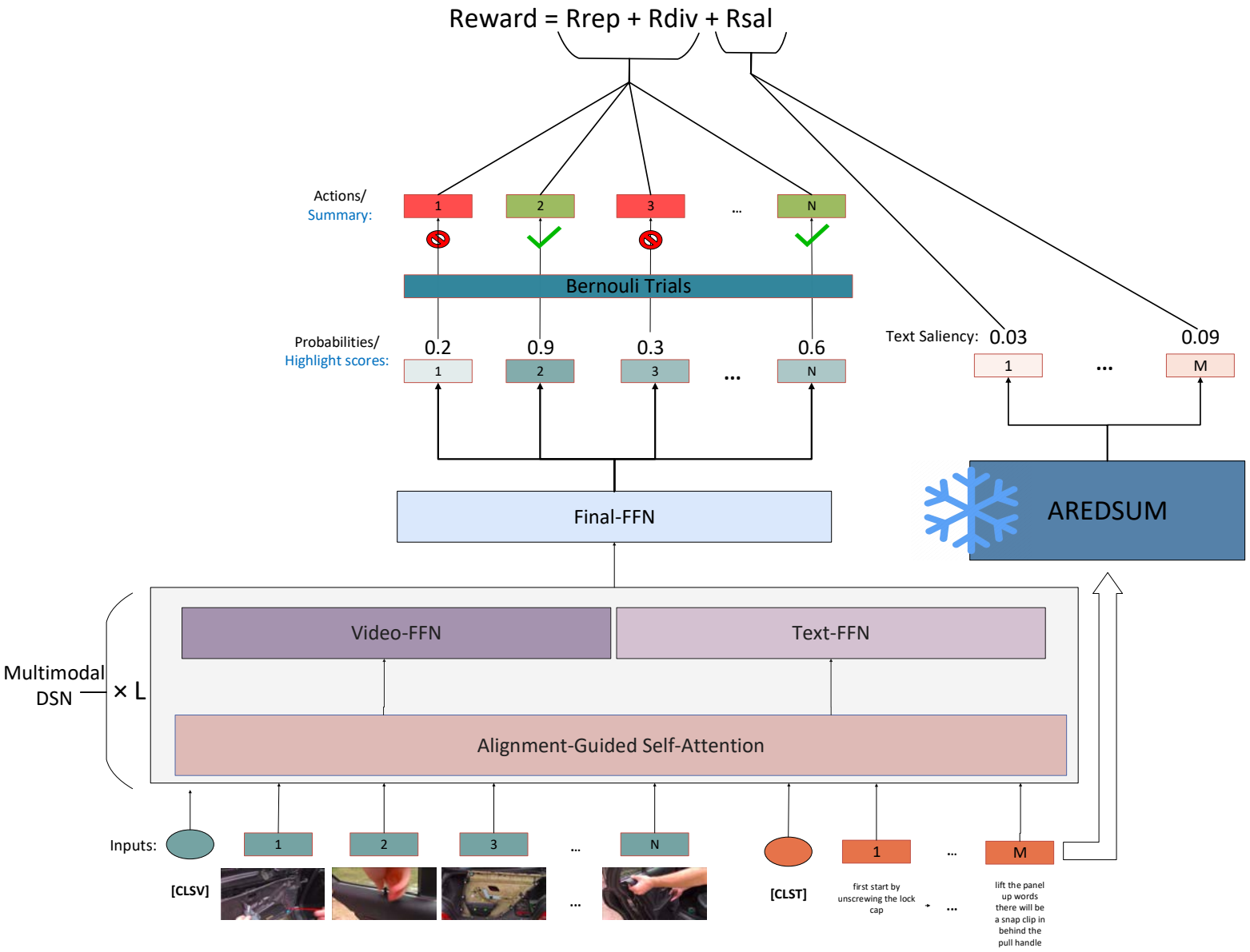


Figure 3.1: Illustration of the Pipeline for the Multimodal Transformer employing the Alignment-Guided Self-Attention mechanism. The input blocks from both modalities represent the combination of the Position and Segment Embeddings with the output of the fully-connected layer that projects both modalities into a common embedding space. The snowflake indicates that the AREDSUM model [9] is already trained and is used for inference. The L represents the number of layers.

encodings are used to include information about the position of each token in the sequence. As discussed in Section 2.5.1, these encodings can be implemented as learnable parameters, which is the approach adopted in this work.

This model aims to exploit the time correspondence between the video frames and the

subtitles that make up the transcript. The first step to achieve this is to introduce learnable segment-based positional embeddings at the input stage. More precisely, each subtitle appears between a starting moment t_s and an ending moment t_e. Thus, $[t_s, t_e]$ indicates the interval during which the subtitle occurs. One embedding is assigned to each subtitle, which is shared by the frames that occur during this interval $\{V_i\}_{i\in[t_s,t_e]}$.

With the addition of these positional encodings, the input sequence from both modalities is concatenated along the time axis, denoted as $\mathbf{X} \in \mathbb{R}^{(M+N)\times C}$

Multimodal Alignment and Fusion

A core component of the model is the **Alignment-Guided Self-Attention** module [25], which facilitates the fusion of modalities by utilizing their temporal correspondence. More precisely, it lets the features of one modality to attend to features of the other modality that occur at the same time. The integration of the Alignment-Guided Self-Attention in the Pipeline is illustrated in Figure 3.1.

If the model used plain self-attention, as described in Section 2.5.1, across all features, the differing nature of the modalities could introduce noise. By contrast, using **Alignment-Guided Self-Attention** ensures that features attend only to the relevant features of the other modality. We implement this mechanism by using a masked self-attention operation, as shown in Figure 3.2. More precisely, we define an attention mask $\mathbf{A} \in \mathbb{R}^{(N+M)\times(N+M)}$, initialized with all zeros. This mask represents the alignment between features, where N and M denote the length of the video and text feature sequences, respectively. The objective is to set the positions that correspond to aligned features to 1.

The features of the same modality should attend to all the other features of the same modality (inter-modality attention), so the cells at positions $\mathbf{A}[1 : N, 1 : N]$ and $\mathbf{A}[N : N + M, N : N + M]$ are filled with the value 1 (green and purple cells in Figure 3.2).

For the attention between video and text features, we set the mask entries to 1 only for pairs of features belonging to the same segment, indicating that they occur at the same time. For example, consider the k^{th} sentence S_k that occurs during the time window $[t_s, t_e]$. We consider the frames that also occur in the same time window $[t_s, t_e]$, denoted as $\{V_i\}_{i\in[t_s,t_e]}$ to be in the same segment as the sentence S_k. We fill with the value 1 the entries $\mathbf{A}[N+k, t_s : t_e]$. and $\mathbf{A}[t_s : t_e, N + k]$

The model then proceeds with the standard self-attention method, as described in Section

2.5.1. More precisely:

$$\mathbf{Q} = \mathbf{X}\mathbf{W_Q}, \mathbf{K} = \mathbf{X}\mathbf{W_K}, \mathbf{V} = \mathbf{X}\mathbf{W_V},$$
$$D_{i,j} = \frac{A_{i,j}\exp(Q_i K_j^\intercal/\sqrt{D})}{\sum_k A_{i,k}\exp(Q_i K_k^\intercal/\sqrt{D})}$$
$$\mathbf{Z} = \mathbf{X} + \mathbf{D}\mathbf{V}\mathbf{W_O}$$

where $i \in [1, M+N]$ are the indices of the matrix $\mathbf{A}$, $\mathbf{X}$ are the inputs features, and $\mathbf{W_Q}, \mathbf{W_K}, \mathbf{W_V}, \mathbf{W_O} \in \mathbb{R}^{C\times C}$ are the matrices explained in Section 2.5.1.

By adopting this mechanism, we exploit the alignment of the features, while minimizing the negative effects of noisy background frames and irrelevant sentences.

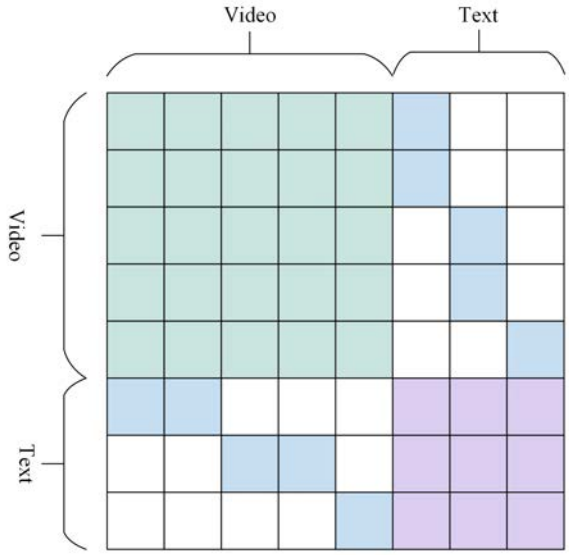


Figure 3.2: Alignment-Guided Self-Attention. Colored squares represent ones and white squares represent zeros.

The Rest of the Architecture

Following [25], instead of adding a single feedforward network, as depicted in Figure 2.6, we add one feedforward Network for each modality. This approach was described in [37], and suggests that applying the same feedforward network to all modalities, may introduce noise, due to the inherent difference in the features. To address this issue, the paper recommends employing distinct feedforward Networks for each modality, allowing each network to capture features unique to its respective modality. The distinct feedforward neural networks are shown in Figure 3.1, as Video-FFN and Text-FFN accordingly.

Apart from the sophisticated attention mechanism and the separate FFNs for each modality, the architecture follows the principles of the original Transformer Encoder (see Section 2.5.1). That means that it incorporates residual connections [33] and layer normalization [34].

The output of the multimodal transformer encoder described above is denoted as $\{y_t\}_{t=1}^{T}$, where T is the length of the video.

3.3.2 Unimodal Transformer

In addition to using the multimodal Transformer as the DSN, we will also utilize a unimodal Transformer for comparison. The complete pipeline that incorporates the unimodal Transformer is illustrated in Figure 3.3.

The architecture of the unimodal Transformer is a stack of Transformer Encoders, as described in Section 2.5.1, and only uses the video frames as input.

Similarly to Section 3.3.1, the visual features are passed through a fully-connected layer, in order for the feature vectors to be projected into a C-dimensional embedding space. The generated features are denoted as $\mathbf{V} \in \mathbb{R}^{N \times C}$. Similarly, positional encodings are added to add spatial information. The resulting input features are denoted as $\mathbf{X} \in \mathbb{R}^{N \times C}$

As mentioned above, the unimodal Transformer DSN is a stack of Transformer Encoders. It takes $\mathbf{X}$ as input and generates $\mathbf{Y}$ as the output, which can also be expressed as $\{y_t\}_{t=1}^{T}$, where T is the length of the video.

3.3.3 Baseline (biLSTM)

This model was used in [28] and will serve as the baseline. The network is a bidirectional LSTM network, as shown in Figure 3.4. The difference between regular LSTM networks (explained in Section 2.4.1) and bidirectional LSTM networks, is that the latter process the input sequence in both forward and backward directions, enabling them to capture context from past and future time steps.

Let $\{x_t\}_{t=1}^{T}$ be the visual features from the input video frames with the length T. The biLSTM takes those features as input and produces corresponding output states $\{y_t\}_{t=1}^{T}$. Each state is the concatenation of the forward and the backward output state. This process lets the model contextualize the inputs with information forward and backwards in the sequence, enabling a richer understanding of the features.

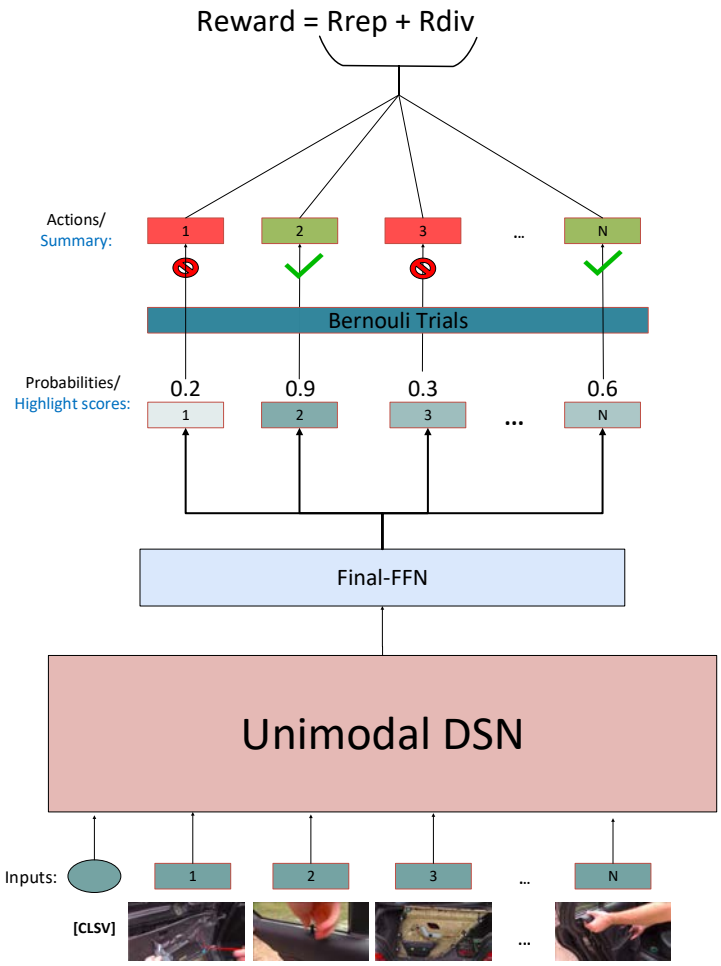


Figure 3.3: Illustration of the pipeline for unimodal models. This pipeline is similar to the one shown in Figure 3.1, but in this case, the DSN is unimodal and therefore does not include modules designed to process the transcript. The DSN can either be a biLSTM, or a stack of Transformer Encoders. The input blocks represent the combination of the Feature, Position and Segment Embeddings.

3.4 Frame Selection and Highlight Scores

So far, we've described the three models we're using in our implementation, up to the point where they generate a contextualized representation of each frame $\{y_t\}_{t=1}^{T}$. In this subsection, we will examine how this representation is utilized to generate the final output. This stage remains the same across all three models.

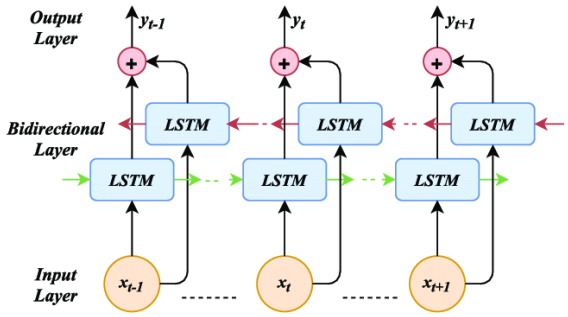


Figure 3.4: Bidirectional LSTM (unimodal DSN) (Figure from [15]).

The output vector is passed through a feedforward neural network (denoted as Final-FFN in Figures 3.1 and 3.3) with a single output node. This node applies the sigmoid function to map the output to a probability value between 0 and 1.

$$p_t = \sigma(\mathbf{W} y_t) \tag{3.1}$$

The value of p_t indicates the probability of each frame to be included in the final summary. To make the final decision the pipeline performs Bernoulli trials for each probability

$$\alpha_t \sim \text{Bernouli}(p_t) \tag{3.2}$$

$\alpha_t \in \{0, 1\}$ indicates whether the frame in position t is selected in the final summary. In later sections, we will discuss how the final summary is generated and evaluated.

3.5 Rewards

As the model is trained using Reinforcement Learning, we need to define reward functions to reward it for generating high-quality summaries. To achieve this, following [28], we must establish an automated method to evaluate the quality of the generated summaries.

We define two reward functions to evaluate the visual content of the video and one reward function to assess its textual content.

3.5.1 Diversity Reward

The first visual reward, called Diversity Reward, ensures that the visual content of the summary is diverse. This stems from the assumption that a high-quality summary will not contain repetitive content and will instead showcase a wide range of distinct visual elements from the original video.

Let the indices of the selected frames be $\mathcal{Y} = \{y_i | a_{y_i} = 1, i = 1, ..., |\mathcal{Y}|\}$ and x_t be the frame features as described in Section 3.2.1. We define R_{div} as the average pairwise dissimilarity among the selected frames:

$$R_{div} = \frac{1}{|\mathcal{Y}|(|\mathcal{Y}| - 1)} \sum_{t \in \mathcal{Y}} \sum_{\substack{t' \in \mathcal{Y} \\ t' \neq t}} d(x_t, x_{t'}), \tag{3.3}$$

where $d(\cdot, \cdot)$ is the dissimilarity function calculated by

$$d(x_t, x_{t'}) = 1 - \frac{x_t^T x_{t'}}{||x_t||_2 ||x_{t'}||_2}. \tag{3.4}$$

The diversity reward function achieves higher values when the selected frames are more diverse, meaning they are more dissimilar from one another.

Nevertheless, because equation (3.3) considers the video frames as single vectors, it ignores the temporal structure of the video. For example, if one scene appears during the beginning of the video and also appears in the end of the video, R_{div} will enforce the model to select frames from one occurrence and ignore the other, as they have similar frames. However, this practice is wrong, because both scenes might be essential for the video. To address this problem, [28] suggests that $d(x_t, x_{t'}) = 1$ if $|t - t'| > \lambda$, where λ controls the temporal distance that frames should have in order for their similarity to be ignored. This hypothesis is supported by their experiments, and the value of λ is adjusted accordingly.

3.5.2 Representativeness Reward

The second visual reward, called Representativeness Reward, ensures that the visual content of the summary is representative of the original video.

The scenes of a video contain visually similar frames that should be relatively close to one another in the feature space. This fact leads to the observation that the feature vectors form clusters in the feature space. The Representativeness Reward R_{rep} achieves high values when the selected frames are close to the cluster centers in the feature space, and is defined as:

$$R_{rep} = \exp(-\frac{1}{T} \sum_{t=1}^{T} \min_{t' \in \mathcal{Y}} ||x_t - x_{t'}||_2). \tag{3.5}$$

The algorithm explained in equation 3.5 above iterates over all the feature vectors of the original video and. for each one, finds the closest feature vector of the generated summary

and computes the distance. It then takes the mean and performs exponential normalization. If the distance between the selected frames and the frames of the whole video is minimal, the reward will be maximized.

3.5.3 Textual Reward

The Textual Reward ensures that the DSN selects frames that correspond to subtitles containing important information. This process guarantees that the final summary includes segments from the original video where the subtitles hold crucial information. To achieve that, we use the trained AREDSUM model [9], discussed in Section 2.7.

Let each subtitle be denoted as s_i, and let the transcript of the video, formed by concatenating all the subtitles, be denoted as D. The outputs of the transformer after document-level encoding for subtitle s_i and transcript D are denoted by h_{s_i} and h_D, respectively, as described in Section 2.7.

Let the following hold:

- s_i is a subtitle occuring over the time interval $[t_s^i, t_e^i]$.
- N_i is the number of frames within $[t_s^i, t_e^i]$, where frames are indexed as $f_{i,1}, ..., f_{i,N_i}$.
- $F_{sal}(s_i)$ is the saliency score of subtitle s_i as described in Equation (2.1)
- $1_{\{f_{i,j} \in \mathcal{Y}\}}$ is an indicator function that is equal to 1 if frame $f_{i,j}$ is in summary $\mathcal{Y}$, and 0 otherwise.

The saliency of each subtitle s_i, $F_{sal}(s_i)$, is evenly distributed across the N_i frames in the interval $[t_s^i, t_e^i]$. The saliency for each frame $f_{i,j}$ (where $j = 1, ...N_i$) in the interval $[t_s^i, t_e^i]$ is:

$$F_{sal\_frame}(f_{i,j}) = \frac{F_{sal}(s_i)}{N_i}.$$

We define the saliency reward as:

$$R_{sal} = \sum_{i}^{M} \sum_{j=1}^{N_i} \frac{F_{sal}(s_i)}{N_i} \cdot 1_{\{f_{i,j} \in \mathcal{Y}\}},$$

where the outer sum iterates over all subtitles s_i and the inner sum iterates over all frames $f_{i,j}$ in the time window of subtitle s_i. As described above, the term $1_{\{f_{i,j} \in \mathcal{Y}\}}$) is an indicator function that ensures that the Reward is only influenced by the frames that are selected in the summary.

This distribution of the saliency across frames simulates the significance of each frame based on the importance of the subtitle it co-occurs with. If a subtitle is short but highly important, the corresponding frames will have a relatively high saliency value, because the saliency is distributed over fewer frames. Conversely, if a subtitle is long, its saliency is spread across many frames, resulting in a smaller saliency value per frame.

This reward function encourages the agent to prioritize segments of the video that contain brief subtitles with significant information, while discouraging the inclusion of segments that contain lengthy subtitles with non-significant information.

Handling Longer Text Sequences

So far, we have explained how the saliency scores generated by AREDSUM are utilized to assign frame-level importance scores. However, as described in Section 2.7, the AREDSUM model is based on the BERT model, which has a maximum input limit of 512 tokens.

To address this issue, in order to be able to handle longer input sequences, we applied a sliding window technique.

Let's define the following:

- Transcript Tokens: $Tk = [b_1, b_2, ...b_n]$, where n is the number of tokens
- Window size: $Win = 512$ tokens. This is the maximum input size of the BERT model
- Sliding step: $Step = 256$ tokens. This represents the step size (stride) between the starts of consecutive windows.

Now, we will explain the sliding window process.

- if $n \leq 512$: Use the Tk directly as the input to the BERT model
- if $n > 512$:
  - Let k be the number of windows we need to handle the input, calculated by:

$$k = \lceil \frac{n - Win}{Step} \rceil + 1$$

This will ensure that the maximum distance between the beginnings of two consecutive windows is no more than 256 tokens.

– The starting index of each token is symbolized as start_idx_i and is calculated by:

$$\text{start\_idx}_i = 1 + (i-1) \cdot Step, \quad \text{for } i = 1, 2, \ldots, k$$

– Given the above, each window is represented by

$$Win_i = [b_{\text{start\_idx}_\text{i}}, b_{\text{start\_idx}_\text{i}+1}, \cdots, b_{min(\text{start\_idx}_\text{i}+W-1,n)}]$$

The term $min(\text{start\_idx}_\text{i} + Win - 1, n)$ ensures that the final window does not exceed the position of the end token.

– Handling the overlapping windows: The BERT model processes input sequences in windows, with each window having a maximum size of 512. Inference is performed separately for the input sequence in each window. However, since the windows overlap, many tokens will appear in multiple windows and will therefore receive different saliency values for each occurrence. To address this issue, the final value of each token will be the average of all values from different runs.

$$F_{sal}(b_j) = \frac{1}{|R_j|} \sum_{r \in R_j} F^r_{sal}(b_j)$$

where $F_{sal}(b_j)$ is the final saliency score of the token b_j, R_j is the set of windows that b_j appears and lastly, $F^r_{sal}(b_j)$ is the saliency value of b_j from the run on window r

3.6 Training

3.6.1 Overview

In this section, we discuss how the DSN, which will be referred as the agent, is trained to perform summarization. For this purpose, we use a method called policy gradient, specifically the REINFORCE algorithm, following [28].

The goal of the agent is to find the optimal actions that will maximize the reward function, which in this case is

$$R(S) = R_{rep}(S) + R_{div}(S) + R_{sal}(S)$$

where S is the predicted summary.

3.6.2 Problem Setup and Objective

The agent acts according to a policy π_θ, where θ represents the parameters of the policy function. In Reinforcement Learning, a policy is the agent's strategy, which given the current state, generates a probability distribution over possible actions.

In the current context, the policy defines how the DSN determines what actions to take, in respect to the input video. The parameters θ represent the weights of the model.

To quantify the process by which the agent gets trained, we define the following objective function:

$$J(\theta) = \mathbb{E}_{p\theta(\alpha_{1:T})}[R(s)] \tag{3.6}$$

- $J(\theta)$: Represents the expected reward if the agent follows a policy π_θ. This is the function to be maximized.

- $p\theta(\alpha_{1:T})$: The probability distribution of all possible actions, which in this case is the selection of frames for the summary. The actions are denoted as $\alpha_{1:T}$, with α_t being the action at time t. The probability distribution is dictated by the policy π_θ.

In other words, the agent changes its policy, by changing its weights, in order to achieve higher rewards.

3.6.3 Policy Gradient and REINFORCE Algorithm

In order to maximize $J(\theta)$, we need to calculate its gradient with respect to θ and change θ in the opposite direction of the gradient.

Computing the gradient directly would be very hard, since it needs the computation of the expectation over all possible action sequences. This would lead to an exponential growth of computation time.

Following [28], we use the REINFORCE algorithm, proposed by [38]. This algorithm is a way to approximate the gradient using a method called Monte Carlo Sampling. The Gradient is expressed as

$$\nabla_\theta J(\theta) = \mathbb{E}_{p_\theta(a_{1:T})}[R(S) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|h_t)], \tag{3.7}$$

Let's break down the individual parts:

- $\nabla_\theta J(\theta)$: The gradient of the expected reward, with respect to the parameters θ
- $R(S)$: The reward that given a summary S that gets generated by following the current action sequence.
- $\log \pi_\theta(a_t|h_t)$: The log-probability of action α_t given the state h_t. The gradient of this term dictates how changing θ will influence the policy.
- h_t: The state of the DSN model on time t.

Using this process, the agent learns to "favor" action sequences that result in higher rewards.

3.6.4 Monte Carlo Approximation

Computing the expectation over all possible combination of actions would be very computationally expensive. As an alternative, we utilize Monte Carlo approximation. More specifically, we let the agent interact with its environment for N episodes, compute the reward for each episode and then take the average to compute the gradient.

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} R_n \nabla_\theta \log \pi_\theta(a_t|h_t), \tag{3.8}$$

- N: The number of the episodes
- R_n: The reward for the n_{th} episode
- T: The number of the frames

This method lets us approximate the gradient by sampling the behavior of the model over many episodes, instead of precisely calculating the effects of every possible action sequence.

3.6.5 Reducing Variance with the baselines

Even though equation (3.8) is a good estimate to the gradient, it will lead to high variance, which may make it difficult for the model to converge. Following [28], we subtract a constant baseline b from the reward in each step, so the gradient's approximation becomes:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} (R_n - b) \nabla_\theta \log \pi_\theta(a_t|h_t), \tag{3.9}$$

where b is the moving average of the rewards so far. This addition will bring stability to the training process according to [28].

3.7 Regularization

By examining the reward functions, we observe that selecting more frames will lead to higher reward values [28]. This means that the model can exploit this fact and learn to select more frames in order to maximize the reward. To prevent this from happening, we utilize a regularization function that constraints the percentage of frames selected by the model.

$$L_{\text{percentage}} = ||\frac{1}{T}\sum_{t=1}^{T} p_t - \epsilon||^2, \tag{3.10}$$

where:

- p_t : The probability of the frame at time t.
- T : The number of the frames.
- ϵ : A number that determines the percentage of frames to be selected.

This regularization method prevents the model from generating very high probabilities for the frames.

Additionally, to avoid overfitting, we include another regularization term following [28]:

$$L_{\text{weight}} = \sum_{i,j} \theta_{i,j}^2, \tag{3.11}$$

where $\theta_{i,j}$ are all the parameters of the model. This term will prevent the parameters from growing uncontrollably.

3.8 Optimization

To form the final optimization function, we combine Equation (3.9) , (3.10) and (3.11), and update θ as

$$\theta = \theta - \alpha\nabla_\theta(-J + \beta_1 L_{\text{percentage}} + \beta_2 L_{\text{weight}}), \tag{3.12}$$

where α is the learning rate and β_1 and β_2 are hyperparameters that balance the regularization.

To optimize the parameters θ of the DSN, we use the Adam [39] optimization algorithm. Through learning, the network increases the log-probability of actions that have led to high rewards, while decreasing the log-probability of actions that have resulted in low rewards.

Chapter 4

Data and Evaluation Setup

In this chapter we will explore the datasets used to train and evaluate the model, as well as the evaluation metrics.

4.1 Datasets

A common practice in Deep Learning is to use one dataset and split it to create a training and an evaluation subset. However, since our approach uses Reinforcement Learning, ground-truth annotations are only required for evaluation. This allows us to train the model using a dataset with raw videos and evaluate it using a separate annotated dataset.

4.1.1 Training Dataset

Since our proposed model is multimodal, it is crucial that the dataset includes videos with a significant amount of speech to fully exploit the modality fusion process.

General Information

Instructional videos are a great choice, because they have large amounts of speech and cover a large variety of topics. Based on this reasoning, we chose to use the HowTo100M dataset [16], which comprises 1.22 million instructional YouTube videos with a mean duration of 6.5 minutes, along with their transcript.

Researchers in [16] originally use this dataset to create joint text-video embeddings, for text-to-video retrieval. However, we will just use the raw videos and subtitles for our task. Figure 4.1 depicts screenshots of the videos, along with the subtitles that accompany them.

Figure 4.1: Examples of videos in the HowTo100M dataset, along with the narration (Figure from [16]).

Despite the fact that the dataset contains 1.22 million videos, due to time and hardware constraints, we were unable to use the entire dataset. Instead, we selected a subset of 34,000 videos to use for training. This number was chosen empirically, as it allowed the training to be completed in approximately 1.5 days with the available hardware (Nvidia Tesla V100 with 26GB VRAM).

4.1.2 Evaluation Dataset

While choosing a video dataset for training is relatively simple, because only raw videos are needed, picking an evaluation dataset is more complex as it requires ground-truth data to evaluate the quality of the generated output.

As mentioned in the previous sections, obtaining ground-truth data through human annotators is an expensive process, which significantly limits the availability of such datasets. Subsequently, the ones that exist are small and not diverse [26], [27].

Researchers in [20] argue that individuals do not necessarily need to be hired in order to annotate the dataset manually. Instead, valuable data can be collected through their interactions with content without their explicit awareness that they are contributing to a dataset.

This can be accomplished using YouTube's engagement graph, shown in Figure 4.2. The purpose of this graph is to indicate how engaged the users are with each video part. More specifically, high values on the graph suggest that viewers found that section particularly engaging. Conversely, low values indicate less interest from the viewers. This data is collected by examining how the user interacts with the video. A detailed explanation can be found in Figure 4.2.

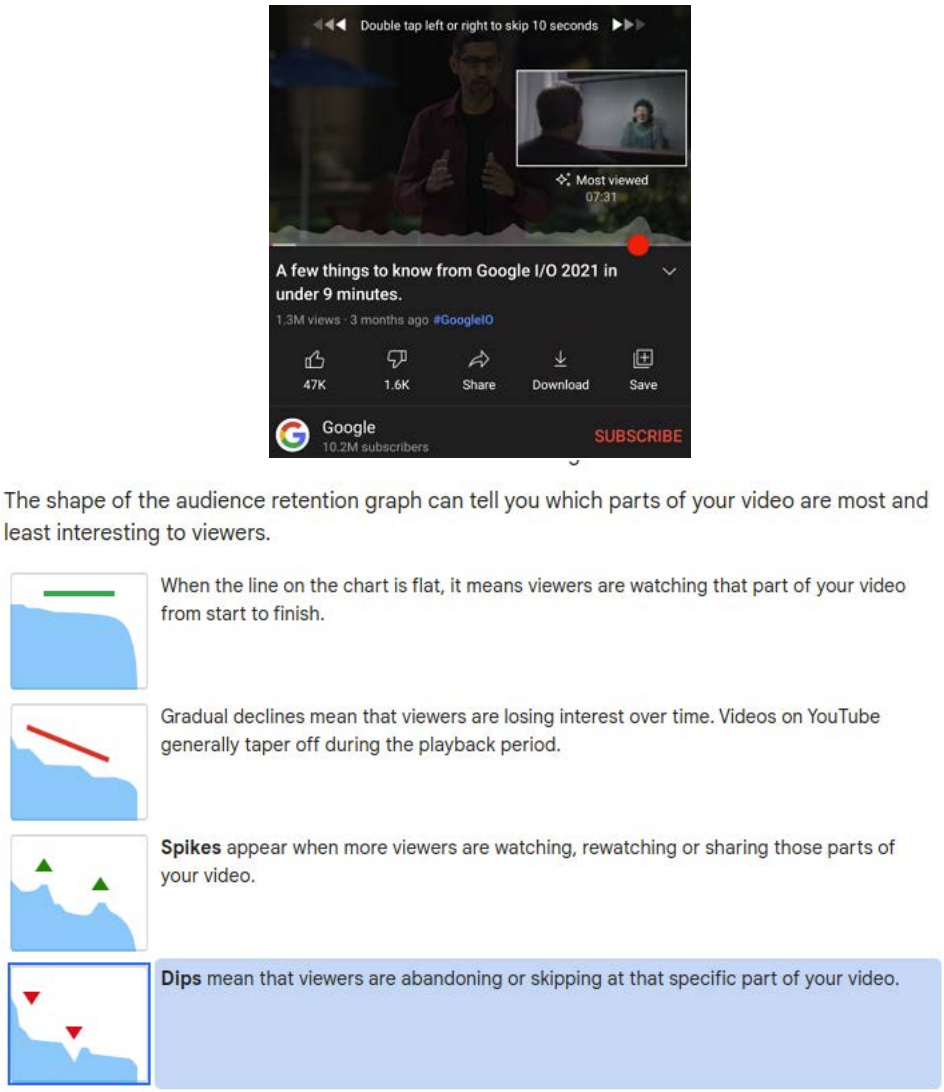


Figure 4.2: User engagement Graph (Figure from [17]).

To this end, they propose a Large-scale Dataset for Video Highlight Detection and Summarization, named **M**ost-**replayed Hi**light Detection and **Sum**marization (Mr. HiSum), where they scrape YouTube videos from multiple categories, along with their engagement graphs. They mention that such graphs are present in every video with over 50,000 views.

This dataset's advantage is its annotation process, which involves crowdsourcing from tens of thousands of unwitting contributors. In contrast, related existing datasets are annotated by a relatively small number of individuals [26], [27].

To ensure that the user-engagement graphs, also known as Most Replayed Stats [20], reflect the highlight score of each video segment, the researchers perform a user study employing human annotators. Subsequently, they instructed the humans to annotate the most salient parts of the video, which were then compared with the engagement graphs. The find-

ings demonstrated a strong agreement between annotators' highlights and the segments where viewers most replayed the video, thus providing robust evidence for the researchers' hypothesis.

As stated in the paper, *"The user engagement stats are provided as a sequence of 100 normalized scores (between 0 and 1), where each corresponds to the relative view frequency of 100 uniformly segmented clips. For instance, if a video is 600-second-long, each score indicates the relative play frequency of each 6-second-long segment"*

Instead of sourcing and downloading new videos, the researchers utilized the pre-existing YouTube-8M dataset [40], which contains approximately 8 million videos from YouTube. They selected a subset of 31,892 videos from this collection, each with a moderate length and a minimum of 50,000 views. The original dataset did not include video captions. To address this, we downloaded the captions separately from an external source [41]. However, this source provided captions for only 12,106 videos. As a result, we used this subset of videos as our evaluation dataset.

4.1.3 AREDSUM Dataset

In our implementation, the AREDSUM model is not trained along with the DSN, but is rather trained independently, and its trained version is used for inference in our main Pipeline. We follow [9] and train it on the CNN/DailyMail Dataset [42]. This Dataset contains texts from news sources, along with selected sentences that act as the extractive summarization for each text. Those sentences act as the $\hat{S}*$ in Section 2.7.2.

The training dataset contains a total of 287,226 documents, with each document truncated to 512 tokens to align with the BERT architecture requirements.

4.2 Evaluation

In this section, we will outline the metrics used to evaluate the models. As discussed in earlier chapters, our approach aims to achieve two closely related tasks: video summarization and highlight detection. In Section 3.4, we explained how probabilities are generated for each frame. These probabilities can be interpreted as highlight scores for the frames or used to generate a video summary. The following subsections demonstrate how these probabilities are utilized to create the final video summary and how the quality of the video summary and

Figure 4.3: KTS segmentation example. Shows the positions of the video that the KTS algorithm detected a scene transition (Figure from [18]).

highlights is evaluated.

4.2.1 Video Summarization

The DSN generates a probability for each frame (Section 3.4), representing the likelihood that the frame will be included in the summary. These probabilities are utilized in various ways to evaluate the model's performance on the video summarization task.

F1 Score

According to the method described in [28], creating a video summary requires first dividing the video into distinct segments, referred to as shots. To achieve this segmentation, we employ the Kernel Temporal Segmentation (KTS) technique [18], designed to identify significant shifts in the video's content. These shifts typically indicate transitions between shots, such as when the camera angle or scene changes. An example of this segmentation is shown in Figure 4.3.

This approach averages the frame-level scores within the same shot to get the shot-level scores. To generate the summary, the algorithm selects shots to maximize the total score, while ensuring that the summary length does not exceed 15% of the video length. This can be achieved using the 0/1 Knapsack algorithm [43].

The annotations in the Mr. HiSum Dataset include engagement scores for each segment of the video. These scores can be treated as equivalent to the probabilities produced by the

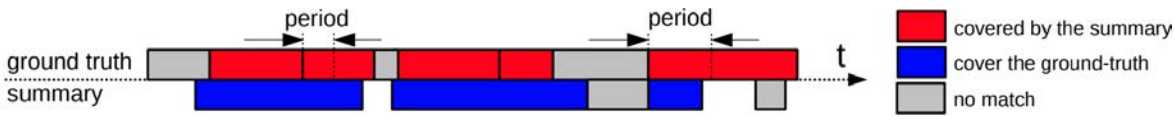


Figure 4.4: Example comparison of the predicted and the ground-truth summary (Figure from [18]).

DSN. By applying the Knapsack method mentioned earlier, in combination with the KTS segmentation algorithm, a ground-truth summary is generated for each video in the Mr. HiSum Dataset.

The ground-truth summary is generated using the ground-truth scores, and the predicted summary is generated using the predicted score. Figure 4.4 shows an illustration of a ground-truth summary and a predicted summary. We define the following:

- True Positives (TP): Shots that appear in both the ground-truth and the predicted summary.
- False Positives (FP): Shots that appear in the predicted summary but not in the ground-truth summary.
- False Negatives (FN): Shots that appear in the ground-truth but not in the predicted summary.

The F1 score is defined as

$$\text{F1} = \frac{2TP}{2TP + FP + FN}$$

The F1 score measures how well the predicted summary matches the ground-truth summary. A higher F1 score means better alignment between the two, indicating that the predicted summary captures the important parts effectively. We use it because it provides a single, balanced metric and penalizes the model for both missing important shots and including unnecessary ones.

Rank Correlation Coefficients

Despite its popularity in a lot of work in this domain, the authors in [44] argue that the F1 score is an unreliable metric for Video Summarization. They found that the F1 score is influenced heavily by the segmentation of the video, rather than by how well the summary captures important content. They proved that random summaries can yield comparable or even superior scores to the state-of-the-art models. In some cases, the summaries generated by human annotators have worse performance than the random summaries.

To overcome this issue, the authors use two rank correlation coefficients to assess the performance of the models. More specifically, they use Kendall's τ [45] and Spearman's ρ [46] coefficients, which measure the similarities between the rankings provided by the

generated and ground truth summaries. The rankings are generated by sorting the frames of the video based on the predicted or annotated importance scores accordingly. The Rank correlation coefficient will measure how these rankings are correlated.

Spearman's ρ:

$$\rho = 1 - \frac{6 \sum_{i=0}^{n} d_i^2}{n(n^2 - 1)} \tag{4.1}$$

- d_i: The difference between the position of the i_{th} frame in the ground truth ranking and its position in the predicted ranking.
- n: The number of frames

Kendall's τ:

$$\tau = \frac{(P - Q)}{\frac{1}{2}n(n-1)}$$

- P: The number of concordant pairs. A pair of frames is concordant if the relative order of the two is the same in both rankings. For example, if i is ranked higher than j in both the ground-truth and the predicted ranking, the pair (i, j) is said to be concordant.
- Q: The number of discordant pairs. A pair of frames in discordant if the relative order of the items is different between the two rankings. For example, if i is ranked higher than j in the ground truth but lower in the predicted ranking, the pair (i, j) is said to be discordant.
- n: The number of frames.

The values of Spearman's ρ and Kendall's τ range between -1 and 1, with 1 indicating complete agreement and -1 indicating complete disagreement. Scores are calculated for each video individually, and the final score is obtained by averaging these individual video scores.

Following recent research, we will use these alternative metrics in our evaluation. Even though [44] demonstrated the unreliability of the F1 score for video summarization, we will still include it in our study for comparison purposes. However, we will not place significant emphasis on assessing its performance.

4.2.2 Highlight Detection

The DSN generates a probability for each frame (Section 3.1), which can be interpreted as its highlight score.

Mean Average Precision

Mean average precision at $\rho\%$ (MAPρ) is a ranking quality metric that measures the model's ability to rank the top $\rho\%$ most important parts of the video in high positions. This metric was introduced in [20].

Their approach involves dividing the video into 5-second-long shots and aggregating the predicted frame scores within each shot by taking the average. The number of shots that fall within the top $\rho\%$ of all shots is denoted as K.

We apply this method for both the ground-truth and the predicted summaries. Similarly to Section 4.2.1, we end up with ground-truth and predicted rankings, but this time the elements are shots and not individual frames.

The top-K shots in the ground-truth ranking are identified as the relevant shots. MAP measures how closely the top-K shots in the predicted ranking align with these ground-truth relevant shots.

Figure 4.5: Example calculation of the $Precision@K$ (Figure from [19]).

First, we define $Precision@K$, a metric that calculates the percentage of the top-K shots in the predicted ranking that are relevant.

$$Precision@K = \frac{\#\text{of relevant shots at } K}{K}$$

An example calculation of the $Precision@K$ is shown in Figure 4.5.

The next step is to define Average Precision@K ($AP@K$).

$$AP@K = \frac{1}{N}\sum_{k=1}^{K} Precision@k \times rel(k)$$

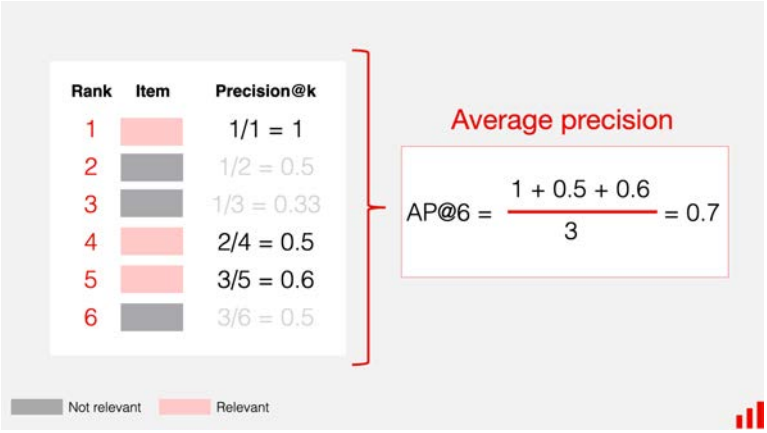


Figure 4.6: Example calculation of the Average $Precision@K$ (Figure from [19])

- N : The total number of relevant shots in the top-K.
- $Precision@k$: The precision calculated at each position k.
- $rel(k)$: Equals 1 if the item at position k is relevant, and 0 otherwise.

Below is an explanation of the algorithm:

- **Step 1**: First, we count the number of relevant shots in the top-K results. This will be N.
- **Step 2**: We iterate through all positions in the ranking up to the K-th position, that contain relevant shots and calculate the $Precision@k$ at each position k.
- **Step 3:** We take the average of all the calculated precision values.

An example calculation of $AP@K$ is shown in Figure 4.6.

The *Mean Average Precision@K* (MAP@K) is the mean of the Average Precision values of all the videos. However, because the videos do not have the same number of shots, using the top-K shots would not be consistent, so we utilize the MAPρ score, which uses a different K for each video, by selecting its top $\rho\%$ shots

As mentioned above, the MAPρ score will be used to evaluate the models on the highlight detection task.

Chapter 5

Experiments and Results

In this section, we will discuss the technical details of our implementation, as well as the experimental setup. Following that, we will demonstrate the experiments conducted, and lastly we will present and analyze the results.

5.1 Implementation Details

In this section, we will discuss the details of our implementation.

- The context window for the Unimodal Transformer is **1024 frames**. For the Multimodal Transformer, the context window is **1024 frames** and **768 sentences**. As described in Section 3.2.1, the videos are subsampled at a rate of 1 frame per second. This means the model can process videos up to a maximum length of 1024 seconds. For videos longer than this threshold, frames beyond the 1024-second mark are discarded. Conversely, for videos shorter than 1024 seconds, padding is applied to fill up the remaining space. The same applies for the sentences of the transcript.
- The dropout probability of the attention layer and both the Video-FFN and Text-FFN is **0.1**. The dropout probability of the final-FFN is **0.5**.
- The hidden size is **128**.
- The number of episodes is **5**. This is the N in Equation (3.8).
- The weights of the Transformer models are initialized using the Truncated Normal distribution [25], and the biLSTM model is initialized using the default PyTorch initialization.

- All models were trained for **60** epochs.

The values for the rest of the parameters are part of the experiments, so they are going to be discussed in the following sections.

5.2 Experimental Setup

In the experiments below, we will show how the models compare with each other in the different metrics, and we will highlight the strengths and weaknesses of each model.

For all the experiments below, the models were trained on the training dataset as described in Section 4.1.1 and evaluated on the evaluation dataset described in Section 4.1.2. As explained in Section 4, the training dataset consists of raw, unannotated videos, so tracking the performance of the model during training is not possible. However, after each epoch, the model is evaluated on the Evaluation Dataset. This approach allows us to monitor the model's performance during training by relying on the evaluation dataset.

Before presenting the experiments, it is important to note that the models will be evaluated on two similar but inherently different tasks: video summarization and highlight detection.

During training, the models try to maximize the reward functions (see Section 3.5), by assigning probabilities to the video frames. However, during evaluation, these probabilities are interpreted differently for the task of video summarization and for the task of highlight detection (as described in Section 4.2). As a result, a model may perform well on one task while underperforming on another, as reported in [20].

The metrics that we will utilize were explained in Section 4. For the Mean Average Precision (MAP) metric, we will mainly use $\rho = 50\%$ and $\rho = 15\%$, denoted as MAP50 and MAP15 respectively.

It is widely accepted that RL pipelines often produce fluctuating results, compared to Supervised Learning ones. To address this, we apply Exponential Moving Average (EMA) smoothing with a smoothing factor of 0.5 to the presented graphs.

5.3 Hyperparameter Tuning

To optimize the learning rate and weight decay parameters, we conducted a grid search on a subset of the datasets. More specifically, we used a training dataset of 400 samples and

an evaluation dataset of 80 samples.

We chose the candidate hyperparameters:

- Learning Rate: 1e-3, 1e-4, 1e-5, 5e-5, 1e-6
- Weight Decay: 1e-7, 1e-6, 1e-5, 1e-4

After training all the models for all the combinations of the above values for 60 epochs, we concluded that the most optimal values for all the models are:

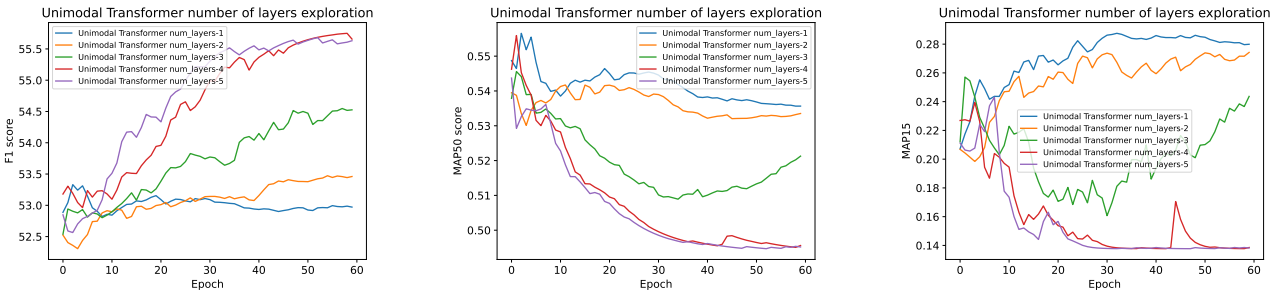
- LR: 1e-5
- Weight Decay: 1e-5

We chose not to include the number of layers as a hyperparameter in the grid search. Instead, we decided to examine this parameter separately on the full dataset, as varying the number of layers could provide some interesting insights for the models.

For the grid-search above, the number of layers was chosen to be 2, following [25].

5.4 Unimodal Transformer Number of Layers Exploration

The unimodal Transformer model was explained in Section 3.3.2. It is a stack of transformer encoders that takes the visual features as inputs. In this section, we will showcase the performance of this model with different numbers of layers on the two tasks.



(α′) F1 vs Epochs (β′) MAP50 vs Epochs (γ′) MAP15 vs Epochs

Figure 5.1: The F1, MAP50 and MAP15 scores of the Unimodal Transformer across Epochs for a varying number of layers.

5.4.1 Video Summarization Task

In Figure 5.1α′, we observe that the model achieves its best performance when using 4 or 5 layers, outperforming configurations with fewer layers. We observe that the model variation utilizing a single layer remains stagnant, with its performance not improving over additional epochs. By adding more layers, the model becomes more complex, exploits more information, and thus delivers better results.

5.4.2 Highlight Detection Task

In Figures 5.1β′ and 5.1γ′, we observe that the model with a single layer outperforms the others on the MAP score, which is the opposite of what we observed on the Video summarization task.

In order for a model to excel in the highlight detection metrics, it needs to place the top-$\rho\%$ of the frames in the top of the ranking, regardless of their order. Conversely, to achieve a high score on the video summarization metrics, the predicted ranking needs to align with the ground-truth ranking (for the Rank Correlation Coefficients) or the predicted summary has to align with the ground truth summary (for the F1 score).

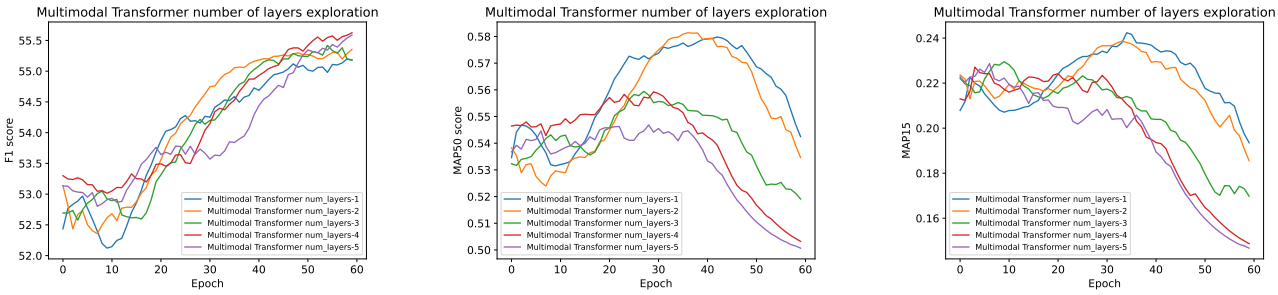
From this, we can infer that highlight detection is a relatively simpler task compared to video summarization. As a result, more complex models are likely to be more prone to overfitting when applied to highlight detection, given the task's lower complexity. This could explain why configurations with fewer layers outperform those with more layers for this task.

5.5 Multimodal Transformer Number of Layers Exploration

The Multimodal Transformer model was explained in detail in Section 3.3.1. In this section we will evaluate the performance of the model with a varying number of layers on the two tasks.

5.5.1 Video Summarization Task

In Figure 5.2α′, we observe that, similar to the experiments with the unimodal Transformer, configurations with 4 or 5 layers achieve the best results compared to those with fewer layers.



(α′) F1 vs Epochs (β′) MAP50 vs Epochs (γ′) MAP15 vs Epochs

Figure 5.2: The F1, MAP50 and MAP15 scores of the Multimodal Transformer across Epochs for a varying number of layers.

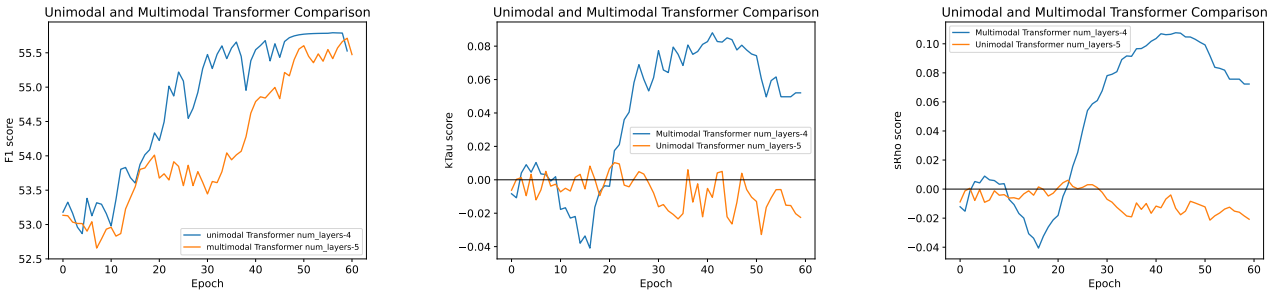
This behavior can be explained by the fact that models with more layers are more complex and therefore have the capacity to process and utilize more information compared to models with fewer layers.

5.5.2 Highlight Detection Task

In Figures 5.2γ′ and 5.2β′, we observe that the performance on the MAP score for all model configurations increases at first, then reaches a maximum point and finally drops. This drop in performance may suggest that the models start overfitting to the training dataset.

Following [20], to overcome the problem of overfitting, we save the weights of each model at the epoch with the best performance. However, since performance varies across the two tasks, we save one snapshot of the model that performs best on the first task and another snapshot that performs best on the second task. This ensures that we retain the model configurations that are optimized for each specific task, helping to achieve better overall performance without overfitting to either task.

Regarding the exploration of the performance across configurations with a varying number of layers, we observe that similar to Section 5.4, the model with a single layer outperforms the rest. As outlined in Section 5.4, this behavior could be explained by the fact that less complex models may be a better fit for highlight detection, since it is a simpler task compared to video summarization.



(α′) F1 vs Epochs (β′) kTau vs Epochs (γ′) sRho vs Epochs

Figure 5.3: Comparison of the F1, kTau and sRho scores of the Unimodal and the Multimodal Transformer.

5.6 Multimodal and Unimodal Transformers Comparison

So far, we have experimented with the individual models and their parameters. In this section, we will compare the Multimodal and the Unimodal Transformer across the different metrics. We will choose the best performing configuration of the models for each task.

5.6.1 Video Summarization Task

In Figure 5.3α′, we observe that the two models achieve similar performance in terms of the F1 score, with each reaching a value of **55.79**.

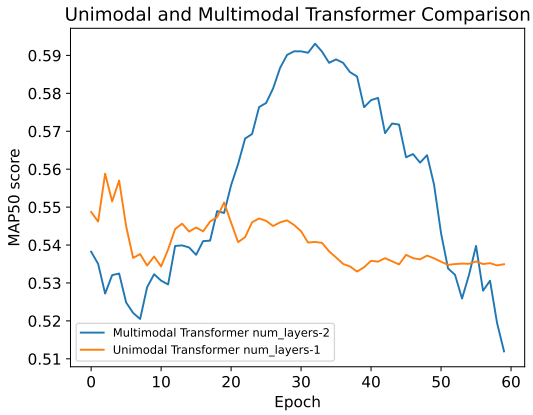
This observation may lead to the assumption that the Multimodal model does not have any advantages compared to the Unimodal model. However, as described in Section 4.2.1, according to Section [44], the F1 score may not be a reliable metric for assessing video summarization, so we utilized the Rank Correlation Coefficients described in 4.2.1.

Figures 5.3β′ and 5.3γ′ show the Kendall τ (kTau) and Spearman's ρ (sRho), respectively. It is evident that the multimodal Transformer outperforms the unimodal Transformer in both sRho and kTau scores. Table 5.1 presents the exact scores for both models at the epoch of their peak performance.

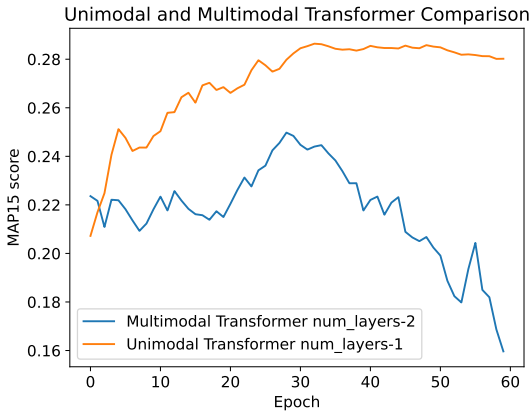
The values of the scores may appear small, but we need to consider that the Rank Correlation Coefficients are very sensitive metrics and small values still indicate an existing correlation. In [44], researchers conducted experiments with human annotators on a small dataset unrelated to ours, and found that the annotated ranking, when compared to the ground truth ranking, achieved an sRho score of **0.204** and a kTau score of **0.177**. This fact is an indication that the values in Table 5.1 are reasonable and align with the expected performance for tasks

Model	Spearman's Rho	Kendall Tau
Multimodal Transformer	0.1225	0.0916
Unimodal Transformer	0.0176	0.0127

Table 5.1: Comparison of Spearman's Rho and Kendall Tau scores for Multimodal and Unimodal Transformers. The values are taken from the graphs on Figures 5.3γ′ and 5.3β′.



(α′) MAP50 vs Epochs



(β′) MAP15 vs Epochs

Figure 5.4: Comparison of the MAP50 and MAP15 scores of the unimodal and the multimodal Transformer across epochs

of this nature.

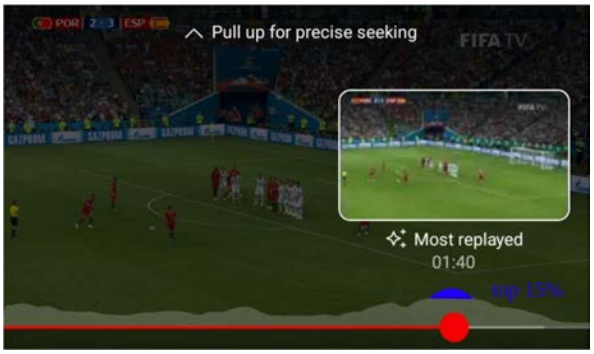
Given all the above, we can assume that the Multimodal Transformer has superior performance compared to the Unimodal Transformer in the video summarization task, according to the sRho and kTau metrics.

5.6.2 Highlight Detection Task

Figure 5.4α′ shows the performance of the Transformers across epochs based on the MAP score with $\rho = 50$ (MAP50), while Figure 5.4β′ presents the same analysis with $\rho = 15$ (MAP15).

From the two graphs in the figures mentioned above, we can conclude that the multimodal model outperforms the unimodal model in the MAP50 score, whereas the unimodal model performs better in the MAP15 score. This behaviour can be explained by the nature of the MAP metric, which considers the top-$\rho\%$ frames as ground truth and evaluates the model's ability to rank these frames in high positions.

(a)



(b)

Figure 5.5: The grey distribution represents the viewers' engagement throughout the video and the blue sections capture the top $\rho\%$ highlights ($\rho = 50$ for (a) and $\rho = 15$ for (b)). The red dot indicates the current position of the cursor in the duration of the video (Figures adapted from [20]).

For MAP15, where $\rho = 15\%$, the top-15% of frames constitute a small portion of the video and may even correspond to a single localized event. In such cases, the visual modality, being stronger than the text modality, is more capable of showcasing this event to the model. The text modality may even act as noise in this scenario, hence the Multimodal Transformer's lower performance. In Figure 5.5(b), we see a screenshot of a video of a football match. The highlighted blue section shows the top-15% frames of the video, which probably corresponds to a goal in this case. We can argue that the visual features alone are sufficient for the model to localize the event, and in this case, the video transcript may instead introduce noise.

In the case of MAP50, where $\rho = 50\%$, it is evident from the blue section in Figure 5.5(a) that the top-50% of frames are numerous and distributed across the video, rather than being concentrated on a single event. As a result, the text modality (the speaker's commentary during the game in this case) plays a larger role in influencing viewer engagement. Thus, the multimodal model achieves a higher score.

Since the multimodal Transformer outperforms the unimodal Transformer in MAP50 and the unimodal Transformer performs better in MAP15, we thought it would be insightful to explore the Mean Average Precision Metric with more values or ρ. More specifically, we explored the values $\rho = \{50, 45, 40, 35, 30, 25, 20, 15, 10, 5\}$

We observed that the multimodal Transformer outperforms the unimodal Transformer when $\rho = \{50, 45, 40, 30, 25\}$. Beyond this point, the unimodal Transformer surpasses the Multimodal transformer for $\rho = \{20, 15, 10, 5\}$.

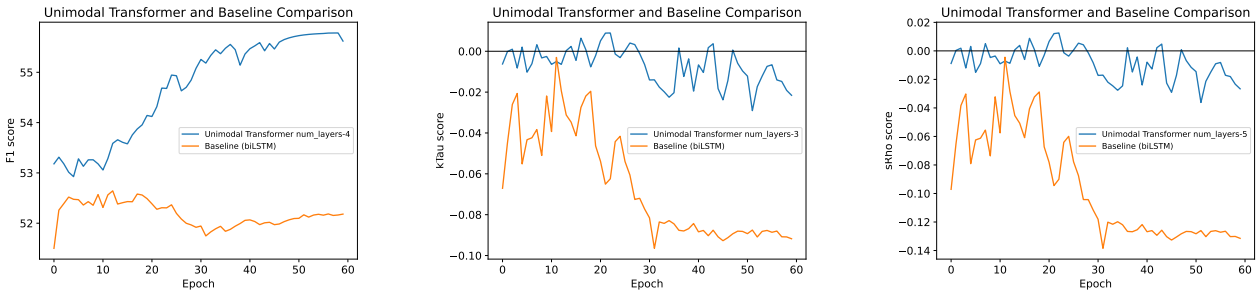
5.7 Unimodal Transformer and Baseline Comparison

As stated in Section 3.3.3, the study in [28] utilized a bidirectional LSTM to act as the DSN, which serves as a baseline for our approach.

In this section, we will compare the Unimodal biLSTM with the Unimodal Transformer.

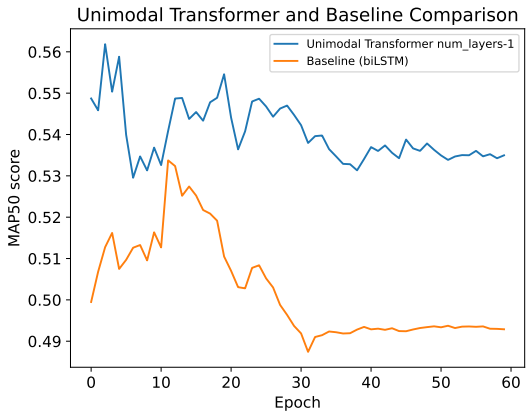
The Unimodal Transformer outperforms the baseline in every metric across both tasks. This is evident in Figures 5.6α′, 5.6β′, 5.6γ′, 5.7α′ and 5.7β′ that illustrate the performance of the models for the F1, kTau, sRho, MAP50, MAP15, metrics, respectively.

The Baseline utilizes a biLSTM model, which is less complex compared to the Transformer. Additionally, the training dataset (34K videos) is orders of magnitude larger than the one used in the original work [28] (50 videos). Due to these two factors, the biLSTM quickly reaches its peak performance within the first few epochs and then begins to overfit. This results in progresively worse performance on the evaluation dataset as the training continues.

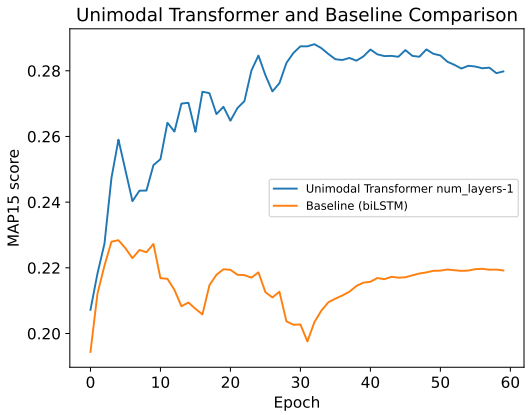


(α′) F1 vs Epochs (β′) kTau vs Epochs (γ′) sRho vs Epochs

Figure 5.6: Comparison of the F1, kTau and sRho scores of the unimodal Transformer and the Baseline (biLSTM).



(α′) MAP50 vs Epochs



(β′) MAP15 vs Epochs

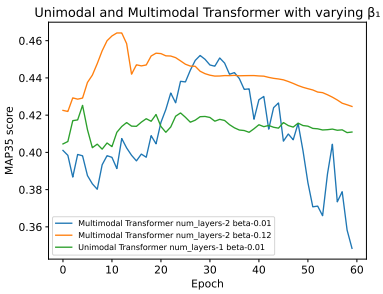
Figure 5.7: Comparison of the MAP50 and MAP15 scores of the unimodal Transformer and the Baseline (biLSTM).

5.8 Experimentation with the Regularization Parameter β_1

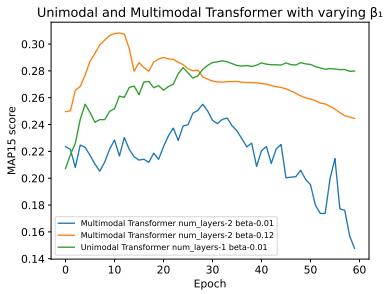
In the experiments described above, we observed that the Multimodal Transformer consistently assigns higher probabilities to the frames in the final stage of the pipeline described in Section 3.4, compared to the probabilities assigned by the other models.

In Section 3.7 and specifically in Equation (3.10) we introduced the regularization term $L_{percentage}$ that constrains the magnitude of the probabilities that the model generates. This term is incorporated into the final optimization function (3.12) and is weighted by a coefficient, denoted as β_1.

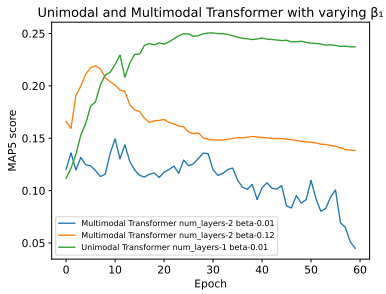
The multimodal Transformer generates higher probabilities compared to the other models because its reward function includes three terms instead of two (see Section 3.6), allowing it



(α′) MAP35 vs Epochs



(β′) MAP15 vs Epochs



(γ′) MAP5 vs Epochs

Figure 5.8: Comparison of the MAP35, MAP15 and map5 scores of the Unimodal Transformer, and the Multimodal Transformer for different values of β_1.

to reach higher values. As a result, the model prioritizes maximizing the reward while paying less attention to the regularization parameter that constrains these probabilities.

To address this issue, we increased the weight of the regularization parameter, in order to emphasize its importance and to ensure that the model considers it.

The default value of β_1, taken from [28], is 0.01. We experimented with the values $\beta_1 = \{0.03, 0.06, 0.1, 0.12, 0.15, 0.2, 0.25\}$.

We observed that as the value of β_1 increases, the performance metrics for the Video Summarization task decline. Similarly, for the Highlight Detection task, the model's performance (measured by MAP) also declines when we increase β_1 for $\rho = \{50, 45, 40\}$.

However, for MAP with $\rho = 35$, the performance of the model increases by increasing the β_1 parameter and the peak performance is reached with $\beta_1 = 0.12$. Figure 5.8α′ illustrates the performance of the Multimodal Transformer when using $\beta_1 = 0.12$. For comparison, the figure also includes the performance of the Multimodal Transformer with the default β_1 value, as well as the performance of the unimodal Transformer.

In Section 5.7, we showed that the unimodal Transformer performs better than the Multimodal Transformer in MAP15 (when $\rho = 15$). However, as shown in Figure 5.8β′, The multimodal Transformer with $\beta = 0.12$ outperforms the unimodal Transformer in MAP15. While we do not include a separate figure for MAP10 to avoid redundancy, it is worth noting that the Multimodal Transformer with $\beta = 0.12$ also outperforms the unimodal Transformer in MAP10.

Nevertheless, as shown in Figure 5.8γ′, the unimodal Transformer outperforms the multimodal models in MAP with $\rho = 5\%$ (MAP5).

This demonstrates that our conclusion in Section 5.7 holds: the unimodal Transformer is more effective than the multimodal Transformer for selecting the top-$\rho\%$ moments of the video, when ρ is relatively small. In Section 5.7, we identified the threshold at which the unimodal model outperforms the multimodal model as being at $\rho = 15\%$. However, as discussed in this section, adjusting the β_1 parameter shifts this threshold to $\rho = 5\%$.

The experiments in this section demonstrate that increasing the weight of the regularization parameter, which constrains the model's average probabilities, helps the multimodal model achieve higher scores in the highlight detection task for relatively small values of ρ.

Chapter 6

Conclusions

6.1 Summary

Our model was developed to address the underexplored potential of unsupervised training for multimodal models in video summarization and highlight detection. Built on an extended Transformer architecture, it exploits information from the video frames and the video transcripts, enabling it to perform both tasks. The training process utilizes Reinforcement Leaning to optimize the performance across the two tasks.

For comparison, we used a plain Transformer Encoder and a biLSTM, which only utilize the visual modality.

In the experiments, we showed that for the video summarization task, increasing the number of layers improved the performance of both the unimodal and the multimodal Transformer. In contrast, for the highlight detection task, models with fewer layers outperformed those with more layers. This finding indicates that the video summarization task is inherently more sophisticated, requiring the use of more complex models to effectively address it.

Moreover, we showed that the unimodal Transformer is better than the biLSTM in all tasks and that the multimodal Transformer is better than the unimodal Transformer in Video Summarization. However, Regarding the Highlight Detection task, the unimodal Transformer performs better at detecting the highest-ranking highlights, while the multimodal model is better at identifying the lower-ranking highlights. We assumed that this occurs because the text modality, being weaker than the visual modality, has little impact on user engagement with the top-ranking highlight moments of a video.

6.2 Future Work

The scope of this work can be further expanded by exploring several potential directions, as outlined below.

As mentioned in the previous sections, the HowTo100M dataset [16] which was used for training contains 1.22 million videos. However, due to time and hardware limitations, we used a subset of 34,000 videos for training, which is 35 times smaller than the original dataset. The utilization of a larger subset, or the whole dataset, will almost certainly yield better results.

Moreover, the training dataset and the Evaluation dataset contain videos from a large variety of domains, but in our work, we treated all videos equally. A potential improvement would be to create domain-specific datasets and train separate models for each domain. This would result in multiple models, each optimized for performance within its respective domain, based on the characteristics of the training dataset.

Lastly, since video summarization and highlight detection are highly subjective tasks, evaluating models on more datasets could provide more robust and reliable results. The WikiHow and COGNIMUSE datasets [47], [48] contain annotated videos and can be used to evaluate models on the video summarization task.

Bibliography

[1] The concept of artificial neurons (perceptrons) in neural networks. https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc.

[2] Elham Pashaei and Elnaz Pashaei. Training feedforward neural network using enhanced black hole algorithm: A case study on COVID-19 related ACE2 gene expression classification. *Arabian Journal for Science and Engineering*, 46, 01 2021.

[3] Recurrent Neural Networks cheatsheet. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.

[4] Understanding Long Short-Term Memory (LSTM) Networks. https://mlarchive.com/deep-learning/understanding-long-short-term-memory-networks/.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[6] Understanding and Coding the Self-Attention Mechanism of Large Language Models From Scratch. https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html.

[7] Self-attention Deep Learning at VU University Amsterdam. https://dlvu.github.io/sa/.

[8] BERT: State-of-the-Art Model for Natural Language Processing. https://www.comet.com/site/blog/bert-state-of-the-art-model-for-natural-language-processing/.

[9] Keping Bi, Rahul Jha, W. Bruce Croft, and Asli Celikyilmaz. Aredsum: Adaptive redundancy-aware iterative sentence ranking for extractive document summarization. In *Conference of the European Chapter of the Association for Computational Linguistics*, 2020.

[10] Types of Deep Neural Networks. https://mriquestions.com/deep-network-types.html#/.

[11] What are Convolutional Neural Networks? https://www.ibm.com/think/topics/convolutional-neural-networks.

[12] What makes Deep Learning different from traditional Machine Learning methods? https://towardsdatascience.com/what-makes-deep-learning-different-from-traditional-machine-learning-methods-fc154db33939.

[13] Max Pooling. https://paperswithcode.com/method/max-pooling.

[14] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In *Computer Vision – ECCV 2018: 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XV*, page 318–335, Berlin, Heidelberg, 2018. Springer-Verlag.

[15] Understanding Bidirectional LSTM for Sequential Data Processing. https://medium.com/@anishnama20/understanding-bidirectional-lstm-for-sequential-data-processing-b83d6283befc.

[16] Antoine Miech, Dimitri Zhukov, Jean-Baptiste Alayrac, Makarand Tapaswi, Ivan Laptev, and Josef Sivic. HowTo100M: Learning a text-video embedding by watching hundred million narrated video clips. In *ICCV*, 2019.

[17] Measure Key Moments for Audience Retention. https://support.google.com/youtube/answer/9314415?hl=en#zippy=%2Ctop-moments%2Cintros.

[18] Danila Potapov, Matthijs Douze, Zaid Harchaoui, and Cordelia Schmid. Category-specific video summarization. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI 13*, pages 540–555. Springer, 2014.

[19] Mean Average Precision (MAP) in ranking and recommendations. https://www.evidentlyai.com/ranking-metrics/mean-average-precision-map.

[20] Jinhwan Sul, Jihoon Han, and Joonseok Lee. Mr. HiSum: A large-scale dataset for video highlight detection and summarization. *Advances in Neural Information Processing Systems*, 36, 2024.

[21] Jingjia Huang, Yinan Li, Jiashi Feng, Xinglong Wu, Xiaoshuai Sun, and Rongrong Ji. Clover: Towards a unified video-language alignment and fusion model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14856–14866, 2023.

[22] Petros Koutras and Petros Maragos. SUSiNet: See, understand and summarize it. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 809–819, 2019.

[23] Ye Liu, Siyuan Li, Yang Wu, Chang Wen Chen, Ying Shan, and Xiaohu Qie. UMT: Unified multi-modal transformers for joint video moment retrieval and highlight detection. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3032–3041, 2022.

[24] Medhini Narasimhan, Anna Rohrbach, and Trevor Darrell. CLIP-it! language-guided video summarization. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 13988–14000. Curran Associates, Inc., 2021.

[25] Bo He, Jun Wang, Jielin Qiu, Trung Bui, Abhinav Shrivastava, and Zhaowen Wang. Align and attend: Multimodal summarization with dual contrastive losses. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[26] Yale Song, Jordi Vallmitjana, Amanda Stent, and Alejandro Jaimes. Tvsum: Summarizing web videos using titles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[27] Michael Gygli, Helmut Grabner, Hayko Riemenschneider, and Luc Van Gool. Creating summaries from user videos. In *European Conference on Computer Vision*, 2014.

[28] Kaiyang Zhou and Yu Qiao. Deep reinforcement learning for unsupervised video summarization with diversity-representativeness reward. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32, 12 2017.

[29] Medhini Narasimhan, Arsha Nagrani, Chen Sun, Michael Rubinstein, Trevor Darrell, Anna Rohrbach, and Cordelia Schmid. *TL;DW? Summarizing Instructional Videos with Task Relevance and Cross-Modal Saliency*, pages 540–557. 10 2022.

[30] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, June 2019.

[31] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943.

[32] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 11 1997.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[34] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint*, arXiv:1607.06450, 2016.

[35] ArrowLuo. Video feature extractor for S3D-HowTo100M. `https://github.com/ArrowLuo/VideoFeatureExtractor`, 2021.

[36] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[37] Hangbo Bao, Wenhui Wang, Li Dong, Qiang Liu, Owais Khan Mohammed, Kriti Aggarwal, Subhojit Som, and Furu Wei. VLMo: Unified vision-language pre-training with mixture-of-modality-experts. *arXiv preprint*, arXiv:2111.02358, 2022.

[38] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, May 1992.

[39] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[40] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. YouTube-8M: A large-scale video classification benchmark. *arXiv preprint arXiv:1609.08675*, 2016.

[41] Jiani Qu, Anny Marleen Hißbach, Tim Gollub, and Martin Potthast. Towards Crowdsourcing Clickbait Labels for YouTube Videos. In Yiling Chen and Gabrielle Kazai, editors, *6th AAAI Conference on Human Computation and Crowdsourcing (HCOMP 2018)*, July 2018.

[42] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Gulçehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In Stefan Riezler and Yoav Goldberg, editors, *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, August 2016.

[43] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, 1990.

[44] Mayu Otani, Yuta Nakashima, Esa Rahtu, and Janne Heikkila. Rethinking the evaluation of video summaries. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7596–7604, 2019.

[45] M. G. Kendall. The treatment of ties in ranking problems. *Biometrika*, 33:239–251, November 1945.

[46] Daniel Zwillinger and Stephen Kokoska. *CRC standard probability and statistics tables and formulae*. CRC Press, 1999.

[47] Mahnaz Koupaee and William Yang Wang. Wikihow: A large scale text summarization dataset. *arXiv preprint*, arXiv:1810.09305, 2018.

[48] Athanasia Zlatintsi, Petros Koutras, Georgios Evangelopoulos, Nikolaos Malandrakis, Niki Efthymiou, Katerina Pastra, Alexandros Potamianos, and Petros Maragos. COGNIMUSE: a multimodal video database annotated with saliency, events, semantics and emotion with application to summarization. *EURASIP Journal on Image and Video Processing*, 2017(1):54, Aug 2017.