Dissertation Type: Research

University of BRISTOL

DEPARTMENT OF COMPUTER SCIENCE

Generating Summarised Videos Using Transformers

Tim Roderick

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

Tuesday 27th October, 2020

Abstract

Video summarisation is the process of creating a shortened version of a video to communicate the original video's content in less time. This has been explored thoroughly as a sequence modelling problem in recent years, coinciding with the surge in popularity of the long-short term memory unit (LSTM). Soft self-attention has also been used throughout recent years along with LSTM based approaches for video summarisation. However, only recently has self-attention been considered as a standalone technique for sequence modelling, thanks to the explosive introduction of the Transformer model. This has lead several to utilise these attention-based techniques for video summarisation and similar problems. What I propose is a direct conversion of the Transformer model for the purposes of generating summarised videos. This is formulated as a sequence-to-sequence, left-to-right generation problem that utilises the multi-headed, multi-layered, soft self-attention based Transformer model. This is done through utilising convolutional neural network (CNN) spatio-temporal features to generate the importance of each frame through a "translation". From this, we choose the frames we want to keep for our summarised video. Our model produces results that are compared to state of the art video summarisation approaches on the SumMe, TVSum, OVP and YouTube datasets.

Contents

Chapter 1

Introduction

Video summarisation is the task of taking some video input and condensing it into a shortened set of clips that still capture the original material. There are many interesting applications of video summarisation – from using it to automatically determine the highlights from a sporting event, to sifting through security or car-ready camera footage to find incidents. The recent advent of small, mountable cameras has lead to a surge in long form recordings in a variety of different scenarios, such as drone footage. This naturally leads to a common application of video summarisation for many.

This to most individuals would be relatively simple, as proven by the multitude of videos online summarising films and television programmes. However, for general content the notion of what should be kept in some summarisation of a video is far from trivial. This boils down to judging the importance of the content at any given moment, and how that should then be acted upon to retrieve an appropriately shortened clip.

To tackle this problem, many different techniques have been applied over many years. Commonly these can be categorised into several types of approaches. The first type of approach is most easily described as the approaches that use visual characteristics and features to determine what is a video clip, and from this decide their importance. This has varied from low-level visual consistency [7, 6], to high-level semantic concept changes [54, 28], to attention based metrics [37, 58, 10]. Also, pre-trained convolutional neural networks (CNNs) have been applied similarly to extract visual features from the video frames to then perform some form of clustering [42]. These approaches can mostly be described as unsupervised learning approaches.

The second type could be broadly described as supervised learning approaches, and have received the most success. However, these approaches vary considerably. This problem naturally lends itself to techniques that employ recurrent neural networks (RNNs), or more specifically long-short term memory units (LSTM) that specialise in operating on and "remembering" sequential data – where videos can often be thought of as a sequence of dependent video frames. LSTM units have been used often for their specific benefit in determining whether the next frame in a sequence is of note based on the long-term dependencies from previous frames [56, 38, 15]. These all also view the video summarisation problem as a sequence-to-sequence problem.

These techniques often use an encoder-decoder framework, where a feature vector is output by the encoder and is then interpreted by the decoder. More specifically, an attention based encoder-decoder approach is used in the AVS framework [24]. These encoder frameworks have been expanded by some to be a generative adversarial network (GAN) [24, 15, 21].

GANs as described above have been used in supervised and unsupervised learning approaches. In the supervised case, such as [15], they propose a discriminator for judging whether a summarised video is generated by their summariser, or belongs to some dataset of summarised videos. In the unsupervised case, [38] instead use a variational autoencoder (VAE) as their summariser to generate video summarisations and a discriminator to distinguish non-summarised, original videos and summarised videos.

More recently, thanks to the popularity of the Transformer model [50], many have started to utilise self-attention schemes for sequence modelling and video summarisation specifically [11, 35].

Our approach will focus on manipulating the popular Transformer model, commonly applied in sequence-to-sequence machine translation problems. We will aim to utilise the core concepts and methods used within the Transformer to generate a sequence one element at a time (left-to-right) representing our summarisation.

Chapter 2

Background

2.1 Background

A variety of machine learning approaches and paradigms have been used to aid in the task of video summarisation. These approaches are varied, but can also be categorised to help us capture their contributions and methods. The following is structured to help highlight this categorisation.

It will also be of use to define some terminology explicitly. We can describe many of the following approaches based on the specific type of summarisation they are trying to perform. Most commonly these can be described as *keyframe* selection or *keyshot* selection.

Keyframe selection - In this type of approach, the goal is to generate a set of representative keyframes that provide an adequate summarisation of the original video. This effectively generates a slideshow that describes the original video.

Keyshot selection - In this type of approach, the goal is to generate video shots – groups of sequential frames – that provide an adequately representative summarisation of the original video. This effectively generates a shorter version of the original video.

2.1.1 Unsupervised approaches

Unsupervised approaches can be described as those that do not provide a labelled ground truth from which the learning approach can determine its accuracy. Instead, these approaches perform inference by finding patterns, clustering to give structure and meaning to the input data.

2.1.1.1 Attention based approaches

A type of approach that has been used by [37, 58, 10] is the idea of modelling attention for video summarisation. Through observing visual, audio, and linguistic features from a video, a model can be defined to incorporate the idea of how much attention any given frame draws. Generally this is done by defining a model over these three different channels, which are then fused in a variety of ways to assign an importance score to each frame. A common choice of visual model is to use optical flow features to determine motion over frames [37], with the idea that humans are more attentive to motion. [10] use temporal gradients similarly.

This mapping can then be smoothed over all the frames in the video to give a differentiable function. The intuition behind this mapping can be described as how much attention a human would reasonably give to each frame, where peaks/spikes in attention correspond to moments that are most important. These peaks are found by differentiating the mapping over all the frames.

From this, one then needs to form a full video clip around these peaks to create the highlight clip.

In [37], they assign a percentage of the full video clip length to be the length of the highlight clip, with sentence detection being used to avoid cutting off the clip mid-sentence.

2.1.1.2 Visual Relevance and Consistency

Using low level visual features to determine visual consistency dominated early video summarisation approaches [7, 6, 3, 31]. These methods often involve segmenting the video into several distinct shots. This is done by [7] using the spatio-temporal slice model found in [41]. [6] alternatively use an information theory-based approach using the Mutual Information and Joint Entropy between frames to find shot boundaries, with more impressive results for vague boundaries such as fade transitions.

After shot detection, most approaches then perform clustering to determine similar shots and subsequently choose a select few to represent the original video. In the case of [7], a graph model is used to represent the shots in the video, where they then cluster based off of an attention metric similar to those found in the previous section.

In [3], the video is not initially partitioned into shots. Instead they perform k-means clustering on the HSV and RGB histograms for each frame to group similar frames together. They then choose one frame from each cluster to represent that shot. This doesn't produce a *video* summarisation, instead it produces a representative *slideshow* of the key-frames: a summary of the video.

2.1.1.3 Semantic Consistency

Semantic consistency between different shots from a video has been used to cluster similar sequences from a video. The semantic nature of this is that it attempts to cluster things based on *what* is taking place in the video, instead of capturing the presence of low-level features such as motion.

The approach taken by [54] uses Latent Dirichlet Allocation to learn the local-activities found in *each individual scene*. Clustering is then performed in a similar fashion to that described in the prior two sections. They perform this process on several separate scenes, clustering now across these scenes to generate a set of profiles that represent specific behaviours. This is then used to acquire representative key-frames for summarisation.

This type of video summarisation is more suited to tasks such as trying to classify types of events from a CCTV footage or other long form videos which contain many similar events. From this [54] construct profiles of these event types so that they can be queried by providing a video example.

[28] propose an approach that also tries to capture semantic consistency. In their approach, they propose that the canonical camera viewpoints chosen in user-generated videos often capture their subject in a similar, maximally informative way. From this idea, they construct a prior over user-generated web-images to try to capture these semantics. Clustering is performed in this viewpoint space to classify an input video's scenes. Those scenes that are closest to the centroid of their particular viewpoint are then chosen as representative scenes from which to construct a full video summarisation.

The drawback of this approach is that it is less effective in the context of viewpoints that are deliberately chosen in a way such that they aren't maximally informative. In most cases these are not user-generated videos and are instead viewpoints found most commonly in film or television programmes.

2.1.1.4 Domain-specific knowledge

Another form of unsupervised learning in this context is to use domain-specific knowledge, such as the word "Goal" in a football match's subtitles. This has been employed by [12], but an issue with this approach is that it involves curating domain-specific features for which the learning can be performed in. This starts to diminish the benefits of this as an unsupervised approach. These ideas in practice have not been as effective as some supervised approaches.

2.1.1.5 Dictionary learning

Sparse dictionary learning has also been used for video summarisation tasks [8, 39]. Dictionary learning is a type of unsupervised learning, where the approach is to infer from the input data a "dictionary" that represents a basis of the space of the input. This means that by linearly combining elements in the "dictionary", you can form any element in the input space – a sparse representation of the input. The intuition behind the naming of this type of approach is that any input element can be thought of as a sentence, and every atom in a dictionary can be thought of as a word.

In [8], they convert each frame in a scene into a CENTRIST [52] PCA [25] feature vector, belonging to some higher dimensional feature space. From this, they then use a dictionary selection approach to choose a subset of these feature vectors, representing the key-frames to be chosen for a video summarisation. This is done over all scenes to form the full video summarisation.

[39] form a similar approach in which they instead form a minimum-sparse reconstruction of the video. They also switch from a scene-to-scene clustering approach to a global clustering approach.

2.1.2 Supervised approaches

Supervised approaches can be described as those that provide a labelled ground truth set of data which can be tested against to improve accuracy by minimising error – "learning" from the labelled data. In the scenario we are currently exploring, this would be video and labelled summaries. As a human is most commonly needed to label these pairs, this helps encode our intuition of how a human would interpret the input data.

2.1.2.1 Determinantal Point Processes

[16] use a supervised approach that is described as a subset selection problem, in this case the subset being frames from a video and how we select them. They do this by proposing a "Sequential Determinantal Point Process," (seqDPP). This model is proposed as a probabilistic model for selecting diverse subsets from a *sequence*, and is based on the determinantal point process (DPP [32]) which views the data as a *set*. Their approach upon extension then becomes the task of using labelled data to improve their seqDPP model in a supervised manner, so that it improves at picking representative subsets of a video sequence.

By introducing a latent variable that controls the time-span of each video segment, DySeqDPP (dynamic seqDPP) [34] enabled seqDPP to learn the local diversity of these dynamic segments as a conditional DPP. In other words, "How Local Is the Local Diversity."[34]

2.1.2.2 Support Vector Machines

[44] use an SVM classifier to assign an importance score to each video segment found through their kernel-based temporal segmentation (KTS). Video summarisations are then composed by collating sequential video segments with high importance scores.

2.1.2.3 Recurrent Neural Networks

Long-Short term memory units (LSTMs [23]) are a specific type of gated recurrent neural network designed to model long-range dependencies over a sequence. In video summarisation, it has been successfully applied to model the dependencies between frames in a video sequence to find those frames that are most important.

In [56], a bidirectional LSTM – an LSTM designed to model both forward and backward dependencies – is used to model the variable-range dependencies of the image sequence. Their model takes a sequence of input frames and then outputs a set of frame-level importance scores corresponding to the likelihood that any frame should belong to the video summarisation. Summary diversity was then further improved by their proposal of a Determinantal Point Process (dppLSTM). This used the importance score output of frames to construct the kernel matrix used in a DPP. This inclusion helps to take into consideration the similarity of chosen video summarisation frames.

[14] also present a similar approach to the previous, instead focusing on the text from live chat associated to a video at each frame. They create a new model that combines a text based LSTM model with the model found in [56] to form an LSTM based approach that incorporates text and visual features.

This is based on their League of Legends dataset [14] that contains highlights of longer matches from the game of the same name, coupled with a record of the messages sent discussing the game live.

An issue found in the type of implementation used by [56, 14] is that although their models can determine how likely it is that an individual frame belongs to the video summarisation, it doesn't capture it's representative power or importance relative to other elements in the sequence. This then means that all highlight frames are viewed as equally important and does not capture more complex temporal structure.

These issues are tackled by [24, 57], where they instead view this problem as a sequence-to-sequence problem. [24] take a sequence of input frames and output a set of frame-level importance scores. From this they then perform keyshot selection that aims to generate a shot based summary that maximises the importance score. They also incorporate an attention based mechanism that assigns weights to each output frame, allowing for some greater notion of what should be incorporated into a summarisation based on the temporal structure of the sequence also. Practically, this is less feasible as for training, labelled attention weights for each frame of the ground truth are required.

2.1.2.4 Self-Attention

Not to be confused with the attention based approaches mentioned prior, self-attention is an approach applied often in machine translation. It has been used to add greater depth to LSTM based approaches [24, 15]. The "attention" aspect comes in the form of how "attentive" a given element in a sequence is to the others based on weighted attention vectors. The greater the weighting between two elements in the sequence, the greater the dependence. This allows for greater depth in creating dependencies in many tasks. However, some recently have utilised attention alone for sequence-to-sequence problems [50].

Self-attention has been applied to video summarisation by [11] where they use a sequence-to-sequence model that utilises attention alone to attend to the dependencies between all the frames. This has a benefit over LSTM based approaches as it can incorporate all elements of the input sequence when interpreting dependencies, independent of sequence length. Unlike other similar approaches that utilise LSTMs, they do not use an encoder-decoder scheme, claiming that the information in the intermediate encoding of the sequence is lossy with variable length sequences. This approach also has the benefit of being more parallelisable than similar approaches.

2.1.3 Generative Adversarial Networks

2.1.3.1 Unsupervised Approaches

Generative adversarial networks are an approach to creating a generative model. The general concept is to have a generator that attempts to create some data. The discriminator is then designed to take that data and determine whether it was generated by said generator, or is actually some real data. The generator then continually improves based off of the discriminator. A wonderful example of this is the website "`https://www.thispersondoesnotexist.com/`" which applies this concept to generating faces artificially as described in [26].

This type of approach has shown state-of-the-art results across several different unsupervised implementations. [38] were the first to apply GANs to video summarisation. They proposed selecting key-frames to summarise a video using their GAN approach – SUM-GAN. They use a variational autoencoder (VAE) as their summariser to generate video summarisations and a discriminator that is LSTM-based to distinguish non-summarised, original videos and summarised videos. They did also extend this approach to be applied in the supervised learning context in the form of SUM-GAN$_{sup}$.

More recently, [21] have used GANs, where the generator weights frame-features to predict the frame's importance. From this, the discriminator attempts to distinguish raw frame-features and their weighted frame-feature counterparts.

The issue with these GAN-based approaches is that described for encoder-decoder scheme – the variable length sequence encoding being lossy. In the case of SUM-GAN, their variational auto-encoder also required training before hand.

2.1.3.2 Supervised Approaches

Supervised GAN based approaches most often use pairs of summarised and original videos in their discriminator to determine what is and isn't a summarisation. [15] incorporate this into their discriminator which judges whether a sequence of frames, described as a "video fragment", is from the labelled summarisation set or is generated by their generator. However, they do also incorporate an unsupervised learning aspect into their training by allowing the comparison, instead of being between a labelled pair, to be done between unrelated video fragments where one is a summarisation-like video. This then helps guide the discriminator without original videos being present.

2.1.4 Reinforcement learning approaches

Reinforcement learning is the third pillar of basic machine learning paradigms alongside supervised and unsupervised learning. The concept of this type of learning is to learn the actions that can be taken in an environment to maximise the cumulative reward.

The first to utilise reinforcement learning for video summarisation where [48]. Their model extracts keyframes, incorporating the category of the video to improve accuracy. This means that their model, for learning, required labelled keyframes and a category description for each video. The reward function they defined is based on successful category classification and minimising cross-entropy loss through their model.

[59] also applied reinforcement learning to video summarisation. They remove the need for labelled keyframes and other video descriptions by instead using a reward function that accounts for chosen frame diversity and representativeness. Their approach can be described as fully unsupervised.

Chapter 3

Proposed Approach

3.1 Methodology

It is important to describe a few specific concepts before moving further. The following sections will provide a description of several concepts utilised for the approach described in chapter 3.

3.1.1 Self-attention

When we consider machine translation, or video summarisation in our case, we want to consider how we can relate elements in the sequence. We do this because all elements in our output sequence are entirely dependent on how elements in the input sequence relate to each other. This is where one can utilise attention.

Attention as a concept in machine translation was introduced by [5]. At the heart of this approach, attention is used to determine how much importance to distribute among elements in the input sequence based on the output of the system. For each element in the input sequence, an attention vector composed of attention weights describes how the element depends on the other elements. Self-attention is the same concept, where the output sequence is swapped for the input sequence. This then describes how important an element is in the input sequence relative to the other elements in the same sequence. More intuitively, how *attentive* any element in the input sequence should be to the other elements in the input sequence. This relative importance based on the input sequence captures exactly the patterns present in the input sequence that we need to perform summarisation.

This technique has been considered in several different forms, but most commonly as *hard* self-attention and *soft* self-attention. Hard attention or self-attention formulates the attention weights as binary decisions; whether an element should be considered or not. This then generates a mask over the elements of the input/output sequence to determine what elements should be considered.

I will be focusing more specifically on soft self-attention, where instead we generate attention weights as probabilities. This then describes *how much* each element in the sequence influences another. This technique has been used successfully in a variety of tasks including translation [50] and image captioning [53]. It has been used in the context of video summarisation by [15, 24] to add extra depth to LSTM based models. It has also been used in video summarisation as a standalone technique by [11] and with the addition of initial hierarchical keyframe detection by [35].

In the following sub-sections, I define the versions of self-attention necessary for the readers comprehension in later sections.

3.1.1.1 Scaled Dot-Product Attention

Now I must define the form of attention we will be focusing on for this problem – Scaled dot-product attention. This is a form of self-attention that is defined and used by [50], but shares many similarities to those approaches used in other works [11, 36].

To start describing what we mean by attention here and the motivation for this definition, it is important to elaborate on what we want attention to do for us. An attention implementation would ideally allow us to produce a discrete distribution over the input sequence for any given element in the input sequence. This weighting determines how much focus to place on other parts of the input sequence as we encode an element at a certain position. In other words, let us define the input sequence $X = (x_0, ..., x_N)$, $x \in \mathbb{R}^{d_{\text{model}}}$, of length $N \in \mathbb{N}$ where each x is a d_{model} dimensional vector. For each position in the sequence $i \in [1, N]$, we want to generate a discrete distribution S_i over the input sequence.

To generate this distribution, we must consider how each element attends to another. In scaled dot-product attention, this is captured by the use of three vectors called query, key and value that are defined for each element in the input sequence. These vectors are utilised to generate the distribution. We can define the query, key and value vectors for an index i in the input sequence as $q_i, k_i \in \mathbb{R}^{d_k}$ and $v_i \in \mathbb{R}^{d_v}$ where traditionally $d_k = d_v \leq d_{\text{model}}$. To generate the distribution S_i we follow the following procedure. First we perform the dot-product of q_i and k_j to generate a score e_{ij}, where $j \in [1, N]$. by doing this for all j with a fixed i we generate a full sequence of scores for each element in the input sequence relative to i. By utilising a softmax function, we can generate a distribution over these scores. In full:

For a given index $i \in [1, N]$, we are given $q_i, k_i \in \mathbb{R}^{d_k}$. We then define the score sequence $\boldsymbol{e}_i$ where each $e_{ij} \in \boldsymbol{e}_i$ is

$$e_{ij} = \; q_i \cdot k_j \text{ for } j \in [1, N]$$

The distribution S_i is the result of a softmax function on $\boldsymbol{e}_i$,

$$S_i = softmax(\boldsymbol{e}_i)$$

The way we can think about this process more intuitively is that we have a query oracle for a given element in the input sequence. To query this oracle, we use the key defined for each element in the sequence to generate our score representing the relative importance of that element. This process encompasses this methodology through multiplying vectors.

With our distribution over the input sequence, we want to move on to generate our final output vector that encodes the relative dependencies we've described. This is where the value vector v comes into play. We now multiply the vector v_j by the importance score corresponding to this index, S_{ij}. The intuition behind this is that we weight all the value vectors, which represent the values of the input sequence, by the scores we have generated.

By summing over these weighted value vectors, we generate a single vector that encodes the other elements in the sequence relative to the element at index i, scaled by their importance. This is described by the following:

For a given index $i \in [1, N]$, value vectors $v \in \mathbb{R}^{d_v}$, the sequence of attention-encoded intermediate representation vectors $z_i \in \mathbb{R}^{d_v}$ is

$$z_i = \sum_{j=1}^{N} v_j \cdot S_{ij}$$

Now that we have formalised this process for one element of the sequence, we want extend this to perform this process for all i. By design, this process is able to parallellised efficiently through matrix multiplication. Meaning we now define the matrices $Q, K \in \mathbb{R}^{N \times d_k}$ and $V \in \mathbb{R}^{N \times d_v}$, the query, key and value *matrices* where each row is a vector as described above. Performing dot-product attention, calculating all of the output vectors Z, can now be described by the following:

$$Z = softmax(Q \cdot K^T) \cdot V$$

An important addition made by [50] is the scaling term $\frac{1}{\sqrt{d_{\text{model}}}}$ where d_{model} is the dimension of the input sequence vectors, which differentiates *scaled* dot-product attention from regular dot-product attention. The reason they do this is to counteract the ill effects this has on the *softmax* function due to the large magnitude of the dot-product as d_k grows large. Therefore we define scaled dot-product attention

(SDPA) as

$$\text{SDPA}(Q, K, V) = softmax(\frac{Q \cdot K^T}{\sqrt{d_{\text{model}}}}) \cdot V$$

Importantly, I have not described how we acquire the matrices Q, K, V. So far we have presumed that they are defined for each element in the input sequence, but this is where we will ideally employ learning. These matrices can all be calculated as a product of the input sequence vectors and some learned weight matrices $W_Q, W_K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $W_V \in \mathbb{R}^{d_{\text{model}} \times d_v}$,

Let X be the input sequence, now interpreted as an $N \times d_{\text{model}}$ matrix, then

$$\begin{aligned} Q = & \ X \cdot W_Q \\ K = & \ X \cdot W_K \\ V = & \ X \cdot W_V \end{aligned}$$

This allows us to re-parameterise the SDPA function in terms of the weight matrices and the input sequence,

$$\text{SDPA}(X, W_Q, W_K, W_V) = softmax\left(\frac{(X \cdot W_Q) \cdot (X \cdot W_K)^T}{\sqrt{d_{\text{model}}}}\right) \cdot (X \cdot W_V)$$

Now with the SDPA function defined, it is important to assess how effective it is for tackling the problem at hand. It does allow us to globally attend to how each element depends on another in the input sequence, but to what complexity? This function is limited in its ability to represent patterns of dependencies between sequence elements. This is because how each element attends to every other in the sequence is defined by a single value. This may not capture the nuance of said dependency. This can be thought of as having one representation of how each element attends every other; a column vector in $\text{SDPA}(X, W_Q, W_K, W_V)$.

In summary, SDPA allows us to efficiently encode the dependencies of the entire input sequence into intermediate vectors that we can utilise for learning purposes. We define how we utilise three learnt weight matrices W_Q, W_K, W_V for the purpose of mapping our input sequence to a representation we use for calculating the SDPA. The input to this process is the input sequence $X \in \mathbb{R}^{N \times d_{\text{model}}}$, producing the dependency-encoded intermediate sequence $Z \in \mathbb{R}^{N \times d_v}$. This process is limited by the complexity of which one element is viewed as depending on another.

3.1.1.2 Multi-headed Self-Attention

The above definition of scaled dot-product attention can be thought of as defining a map to an attention representation of the input sequence. The limitations of this being that how an element of the sequence depends on another is captured as a single value, as described above. *Multi-head* self-attention then extends this to create multiple attention maps, providing multiple attention representations. This allows more complex dependencies to be stored as the dependencies between elements in the sequence are represented by a group of values. This attention mechanism was popularised by [50] as an extension of SDPA, and has been applied to video summarisation by [35].

Instead of performing the SDPA function once, we now want to perform it h times where $h \in \mathbb{N}$ is the number of attention heads. This will define how many representations of the input sequence we want. We now set $d_k = d_{\text{model}}/h$ so that the same complexity is preserved. Formally,

Let W_Q^i, W_K^i, W_V^i for $i \in [1, h]$ be the weight matrices for all attention heads, where $W_Q^i \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_K^i \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_V^i \in \mathbb{R}^{d_{\text{model}} \times d_v}$.

We then can calculate for each attention head the dependency encoded sequence,

$$H_i = \text{SPDA}(X, W_Q^i, W_K^i, W_V^i)$$

As with the original attention function, we want to generate $Z \in \mathbb{R}^{N \times d_v}$ an intermediate sequence that encodes the dependencies in the sequence. Instead, we have through this process h matrices of size $N \times d_v$. To incorporate this into one output matrix Z, we concatenate the h matrices together horizontally and multiply them by a third weight matrix $W_O \in \mathbb{R}^{(d_v \cdot h) \times d_{\text{model}}}$ to get the correct dimensionality. Formally,

$$Z = \text{Concatenate}(H_1, ..., H_h) \cdot W_O$$

We can then define Multi-Head SDPA as the following function for our purposes, parameterised by the weight matrices, the input sequence, and the number of attention heads.

For $i \in [1, h]$, $H_i = \text{SPDA}(X, W_Q^i, W_K^i, W_V^i)$.

$$\text{Attention}(X, W_Q^1, ..., W_Q^h, W_K^1, ..., W_K^h, W_V^1, ..., W_V^h, W_O) = \text{Concatenate}(H_1, ..., H_h) \cdot W_O$$

With this, it is important for us to consider how effective this approach is at tackling the problem at hand. This function as defined now allows us to capture the dependencies we care to describe to the level of granularity we desire. This by itself can be more than enough to handle the dependencies as we have formulated, as shown by [11]. However, it is important to consider other ways that we may utilise this function as defined to capture more attention representations of the input sequence. An example of this is performing the the Attention function multiple times, as utilised by the Transformer model [50]. This is also explored in the following.

In summary, we have defined an extension to SPDA called multi-head self-attention. This expanded attention function allows the output sequence Z to *"jointly attend to information from different representation subspaces at different positions"* [50]. This is done by providing h different sets of weight matrices to perform the SPDA function h times, projecting the input sequence into h different representations. We then provide one more weight matrix W_O to allow us to preserve the dimensionality of the output sequence as we've defined in the prior sub-section.

3.1.2 The Transformer model

The Transformer model is a model proposed by [50], designed to tackle sequence-to-sequence problems. The Transformer is based around the use of self-attention to learn the sequences dependencies and incorporate them into the sequence translation. Most commonly, this model has been used for machine translation tasks. This model attempts to increase the complexity of dependencies considered by repeatedly performing the Attention function, as defined above, to capture a greater number of attention representations.

Similar to other research in the area of sequence-to-sequence modelling [56, 38, 15], the Transformer is an encoder-decoder style model. The encoder in this type of scheme takes an input sequence $X = (x_0, ..., x_N)$ of length $N \in \mathbb{N}$ and maps it to an equally long intermediate representation $Z = (z_0, ..., z_N)$, where $x, z \in \mathbb{R}^{d_{\text{model}}}$. The purpose of this is so that the sequence translation becomes length agnostic, only dependent on intermediate representation. The decoder in this scheme then takes the intermediate representation Z and produces an output sequence $Y = (y_0, ..., y_M)$ where $M \in \mathbb{N}$, $y \in \mathbb{R}^{d_{\text{model}}}$. These approaches can also be described as auto-regressive, meaning they consume the previous elements of the sequence produced by the decoder as it generates the next to help further improve prediction as generated.

The Transformer follows this structure, using self-attention as described in section 3.1.1.2 and fully-connected feed forward networks as elements in its encoder and decoder. Below we will describe the structure of the different components of the Transformer model, and how they coalesce to perform sequence-to-sequence modelling.

3.1.2.1 Embeddings

When we reference the dimensionality of the input and output sequence d_{model}, we are referring to the dimensionality of the elements of the sequence embedded using a learnt embedding algorithm often employed for sequence-to-sequence problems. These embeddings are essentially look-up tables that have a fixed dictionary and size, where here size $= d_{\text{model}}$. Each element in the dictionary is assigned a unique embedding vector $e \in \mathbb{R}^{d_{\text{model}}}$ which is used to convert the input sequence to a matrix $X \in \mathbb{R}^{N \times d_{\text{model}}}$ and then passed to the encoder of our encoder-decoder scheme.

3.1.2.2 Positional Encoding

Unlike many other sequence-to-sequence models that are based on recurrence or convolution, the notion of position is not encoded into the models function. To give attention this sensitivity to position, we also add an additional positional encoding to the input sequence. This positional encoding will be applied to the input sequence as a vector with the same dimension as each element in the input sequence, d_{model}, so that the positional encoding can be applied by summing with each element. As [50] have done, we do not use a learnt positional encoding and instead adopt a sine and cosine frequency encoding. One of the main benefits to this encoding scheme is that its able to scale to sequence lengths not observed in training, which will be common in the problem we are exploring. Below is the definition of the position function.

We define the positional encoding as a matrix $P \in \mathbb{R}^{N \times d_{\text{model}}}$ that we add to the input sequence. Let $i \in [0, N-1]$ refer to the positions of the embedded input sequence, and $d \in [0, d_{\text{model}} - 1]$ refer to the dimensions of each input sequence element.

$$P_{i,2\cdot d} = sin\left(\frac{i}{10000^{\frac{2\cdot d}{d_{\text{model}}}}}\right)$$
$$P_{i,2\cdot d+1} = cos\left(\frac{i}{10000^{\frac{2\cdot d}{d_{\text{model}}}}}\right)$$

The way we practically implement this however will be in log space,

$$P_{i,2\cdot d} = sin\left(\frac{i}{e^{\frac{2\cdot d}{d_{\text{model}}} log(10000)}}\right)$$
$$P_{i,2\cdot d+1} = cos\left(\frac{i}{e^{\frac{2\cdot d}{d_{\text{model}}} log(10000)}}\right)$$

This forms a geometric progression that allows the model to incorporate an input sequence elements relative position when learning.

3.1.2.3 Encoder

The encoder of the Transformer model takes the embedded and positionally encoded input sequence $X = (x_0, ..., x_N)$ of length $N \in \mathbb{N}$ and maps it to an equally long intermediate representation $Z = (z_0, ..., z_N)$, where $x, z \in \mathbb{R}^{d_{\text{model}}}$. This process is performed over several layers composed of sub-layers. In the following, we will describe these separate layers and how they are connected.

Each encoder is composed of $L \in \mathbb{N}$ encoder layers. Each layer shares the input and output of the entire encoder – X, Z as described above. Encoder layers are comprised of several sub-layers.

Self-attention layer - This layer performs the Multi headed self attention function defined in section 3.1.1.2. This allows us to distribute importance across the input sequence in order to capture dependencies in the sequence.

Position-wise Feed-forward Network layer - After the self-attention layer, we now want to consider how this representation may be interpreted by the following self-attention layer. By providing the output of the previous self-attention layer directly into the next, it will cause the attention to potentially focus on the pre-existing patterns found in the attention representation. By utilising another sub-layer, we control how this attention representation is provided to the next attention layer.

This is where we utilise a position-wise feed-forward network as a separate sub-layer. This is described specifically as position-wise as the network is applied to each position separately. However, the mapping performed by the network is identical. The reason we perform this identical mapping for each position is that we are effectively controlling how we consider each element in the sequence in a new space, identical for each element as to not create some bias. This can be described by the following,

Given an input sequence $X = (x_0, ..., x_N)$ of length $N \in \mathbb{N}$, the feed forward network applies the following mapping for each $x \in X$ to produce $Z = (z_0, ..., z_N)$. This can be described as two linear mappings with an activation function between them, with the ReLU activation function [22] being our choice here,

$$Z = \text{ReLU}(x \cdot W_1 + bias_1) \cdot W_2 + bias_2$$

For $W_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ffn}}}, W_2 \in \mathbb{R}^{d_{\text{ffn}} \times d_{\text{model}}}$, where d_{ffn} is the width of the feed forward network. These weight matrices are shared for all elements in the sequence, but are not shared for every encoder-layer.

This is because we may want to change how we represent each element of the sequence in some space in different layers. As this operation is only dependant on one x at a time, these networks can be executed in parallel for greater time-efficiency.

Residual Sub-layer Connection - With the position-wise feed-forward network and self-attention layers defined, we have almost completely defined our encoder layer. Importantly however, between each sub-layer is a residual connection [19] and a layer normalisation function as defined by [4]. The purpose of the residual connection is to tackle common issues found in deep networks such as vanishing gradients or the curse of dimensionality. It does this by adding the input into a network to its output to make a "residual connection" between the output and the input, bypassing activation functions and other processes that may cause long-term information degradation. The layer normalisation is used to normalise the activity between elements in the sequence; to perform regularisation.

This entire process can be defined as its own sub-layer, taking the output of the previous layer Z and the input sequence to the previous layer, X. This process also produces a sequence $Y \in \mathbb{R}^{N \times d_{\text{model}}}$. We define this process as one function below,

$$Y = \text{LayerNorm}\,(Z + X)$$

The full encoder layer structure is described in the following diagram 3.1, illustrating the connections between these sub-layers. In summary, the encoder takes the input sequence that has been embedded and

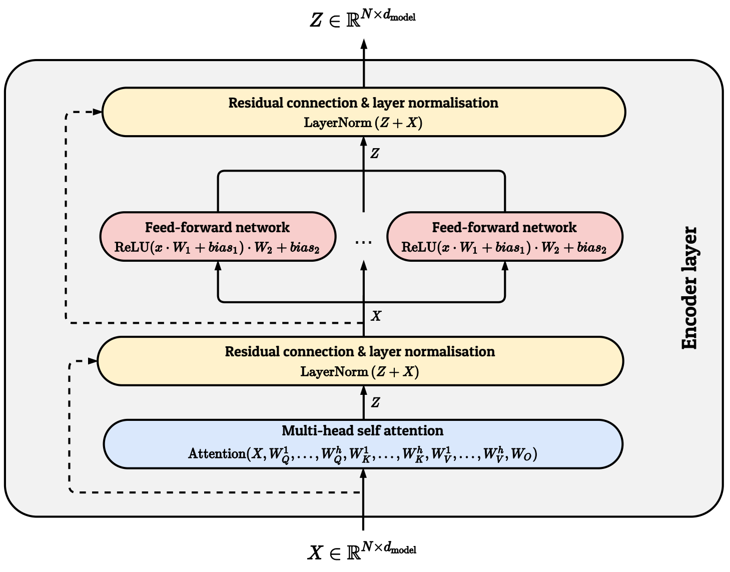
$Z \in \mathbb{R}^{N \times d_{\text{model}}}$
Residual connection & layer normalisation
$\text{LayerNorm}\,(Z + X)$
Z
Feed-forward network
$\text{ReLU}(x \cdot W_1 + bias_1) \cdot W_2 + bias_2$
...
Feed-forward network
$\text{ReLU}(x \cdot W_1 + bias_1) \cdot W_2 + bias_2$
X
Residual connection & layer normalisation
$\text{LayerNorm}\,(Z + X)$
Z
Multi-head self attention
$\text{Attention}(X, W_Q^1, \ldots, W_Q^h, W_K^1, \ldots, W_K^h, W_V^1, \ldots, W_V^h, W_O)$
Encoder layer
$X \in \mathbb{R}^{N \times d_{\text{model}}}$

Figure 3.1: This diagram illustrates the structure of one encoder layer described in section 3.1.2.3. The arrows indicate the flow of information between layers, with the appropriate dimensionality specified. Any time X, Z are referenced, they refer to how they are interpreted by the functions as specified in our definition of these concepts.

positionally encoded and produces an output sequence of the same dimensionality. It does this through several stacked layers, each composed of several sub-layers: Self-attention, feed-forward networks, and residual sub-layer connections. The stacked layers allow the model to capture the complex dependencies between elements of the sequence through stacked attention representations.

3.1.2.4 Decoder

The decoder of the encoder-decoder based Transformer functions much the same as other models. The goal of the decoder is to generate our output sequence using the intermediate representation of the input sequence generated by the encoder. Our decoder is also auto-regressive, meaning that it also considers the previous elements of the sequence when generating the next. The way the decoder models the dependencies is much the same as the encoder, and is designed specifically to capture the more complex dependencies. How the decoder takes this intermediate sequence into consideration when generating is non-trivial and will be defined below.

Similar to the encoder, the decoder takes a sequence as input and outputs a sequence Z. Unlike the input sequence defined for the encoder, the sequence is initially empty and is then filled with the values generated by the decoder. This means that unlike the encoder, which processes the entire input sequence at once, the decoder must be used $M \in \mathbb{N}$ times to generate the sequence.

The decoder is comprised of L layers, much the same as the encoder. In fact the structure of the decoder is almost identical to the encoder with one extra sub-layer. This sub-layer is very similar to the self-attention layer defined in section 3.1.1.2 for the encoder, with the important difference coming in how it uses the output of the encoder. This sub-layer is defined below. The decoder self-attention sub-layer is found between the self-attention layer and the feed-forward network layer. Just as with the sub-layers found in the encoder, there is a residual connection between all sub-layers 3.1.2.3. This layout is visualised in diagram 3.2.

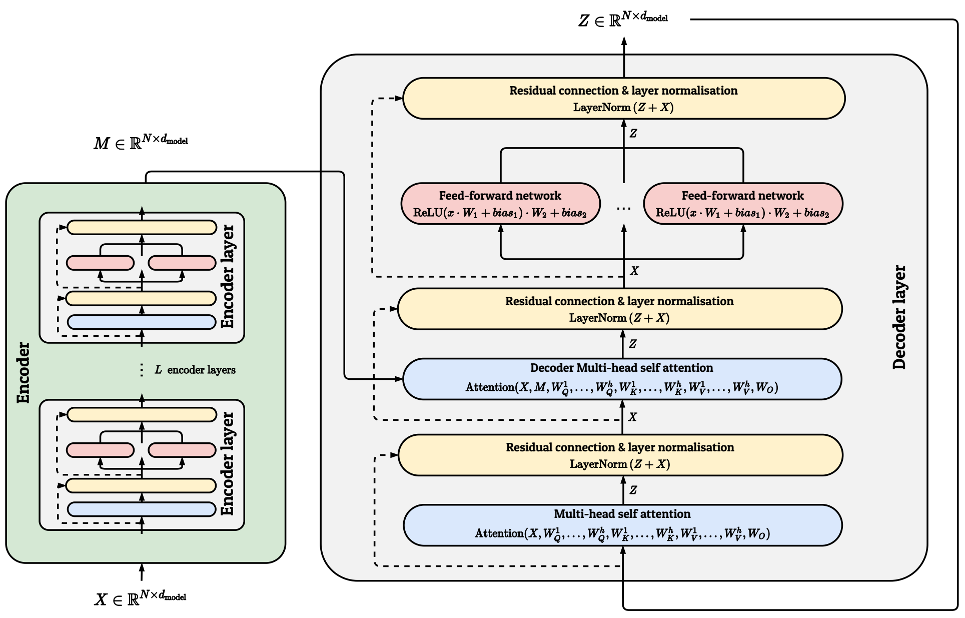
$Z \in \mathbb{R}^{N \times d_{\text{model}}}$
$M \in \mathbb{R}^{N \times d_{\text{model}}}$
$X \in \mathbb{R}^{N \times d_{\text{model}}}$
Encoder
Encoder layer
Encoder layer
$\vdots$ L encoder layers
Decoder layer
Residual connection & layer normalisation
LayerNorm $(Z + X)$
Feed-forward network
$\mathrm{ReLU}(x \cdot W_1 + bias_1) \cdot W_2 + bias_2$
Decoder Multi-head self attention
$\mathrm{Attention}(X, M, W_Q^1, \dots, W_Q^h, W_K^1, \dots, W_K^h, W_V^1, \dots, W_V^h, W_O)$
Multi-head self attention
$\mathrm{Attention}(X, W_Q^1, \dots, W_Q^h, W_K^1, \dots, W_K^h, W_V^1, \dots, W_V^h, W_O)$
Z
X

Figure 3.2: This diagram illustrates the structure of one decoder layer described in section 3.1.2.4. The arrows indicate the flow of information between layers, with the appropriate dimensionality specified. Any time X, Z are referenced, they refer to how they are interpreted by the functions as specified in our definition of these concepts.

Decoder self-attention layer - Much the same as the self-attention layer for the encoder 3.1.2.3, this layer performs Multi-head self-attention through the function defined in section 3.1.1.2. It does this in-between the first self-attention layer and the feed-forward network layer. An important difference however is that this is where we incorporate the intermediate sequence generated by the encoder. In this layer instead of providing the input sequence X to the attention function, we provide the output sequence of the decoder so far, Z, and the output of the encoder referred to here as "memory" M. This is done so that, along with the attention representation generated so far by the decoder to describe the generated sequence, it also takes into consideration the complex dependency structure formed from the encoder. This can be defined as the following amended SPDA function from section 3.1.1.2, with the multi-head extension functioning the same.

$$\mathrm{SDPA}(Z, M, W_Q, W_K, W_V) = softmax\left(\frac{(Z \cdot W_Q) \cdot (M \cdot W_K)^T}{\sqrt{d_{\text{model}}}}\right) \cdot (M \cdot W_V)$$

This can be viewed as using the memory M to generate the key and value matrices K, V. This allows us to utilise the keys and values interpreted from the encoder on a new query matrix in this new output space. This process is described in diagram 3.2.

As described prior, the decoder is auto-regressive. This process is formalised by the following,

The output sequence $Z = (z_0, ..., z_M)$ is initially empty. The decoder, without input, performs a decoder iteration to produce the first element of the input sequence z_0. This is then fed into the decoders next iteration after passing through the generator 3.1.2.5 to produce z_1. This process is repeated until the decoder has finished generating the output sequence of length M. This can also be thought of as initially $Z = \{\}$, the sequence takes $(z_0, ..., z_{i-1}) \in Z$ as input at each iteration i to produce z_i for $i \in [0, M]$.

In summary, the decoder functions as the standard decoder found in an encoder-decoder scheme. The decoder interprets the encoded sequence generated by the encoder to produce an output sequence in a new dictionary one element at a time, where dictionary here refers to the embeddings found in section 3.1.2.1. It does this by performing the attention function defined in section 3.1.1.2, but instead using the sequence it is generating and the output of the encoder to generate the query, key, and value matrices.

3.1.2.5 Generator

The generator is the final step of an encoder-decoder scheme such as the Transformer. In the generator, we take the output of the decoder and pass it through a linear fully-connected neural network layer, to allow a learnt representation of the final decoder output, that projects the decoder sequence onto the output space. This projection makes the output of this layer the size of the number of possible elements the output sequence can contain. We then perform a $softmax$ function on this output to create a distribution, representing the probability of which element should be chosen next. This is described formally by the following,

Given the decoder output at a given decoder iteration i, $Z \in \mathbb{R}^{i \times d_{\text{model}}}$, the output distribution over the output vocabulary $S_Y = (y_0, ..., y_V)$ where each $y \in [0, 1]$ and V is the size of the output vocabulary is calculated by the following,

$$S_Y = softmax(Z \cdot W_T + b),$$

Where $W_T \in \mathbb{R}^{d_{\text{model}} \times V}$ is a weight matrix, and b is the bias.

This generator is used during every decoder iteration to generate the next element in the sequence, which is then fed back into the input of the decoder for the following iteration. With this, the full picture of the Transformer model as defined by [50] is formed. The full model is summarised in diagram 3.3.

3.2 Proposed Approach

3.2.1 Problem Statement

I formulate video summarisation here as a sequence-to-sequence problem, much the same as [11, 56]. Our goal will be to take an input sequence of frames and generate an output sequence of keyframes/keyshots. More specifically, the approach will be an adapted version of the widely successfully Transformer model [50]. This model is designed for sequence-to-sequence problems, specifically tailored to language translation. As this problem can also be loosely thought of as a translation problem – from input frames to summarisation elements – adapting the Transformer model seems appropriate for this task. With this brings the benefits exhibited by the Transformer model, such as superior training performance due to parallelisation. This in-fact tackles one of the main issues with LSTM based approaches in its lack of training parallelisation [50]. In section 3.1 I defined the Transformer model and self-attention to provide the necessary background for describing my implementation for video summarisation.

3.2.2 Video Summarisation Transformer

With our description of the Transformer model from section 3.1, we can now move forward to define the implementation of the Video Summarisation Transformer Framework (VST). The input to our model will be a sequence $X = (x_0, ..., x_N)$, $x \in \mathbb{R}^{d_{\text{model}}}$, of length $N \in \mathbb{N}$ where each element x is a CNN feature vector with dimension d_{model} for each sub-sampled input video frame, lying within a shot boundary. The output sequence will be a sequence $Y = (y_0, ..., y_N)$, $y \in [0, 1]$, also of length N. Y corresponds to the probability that a sub-sampled frame will belong to the summarisation; the *frame-level importance score*.

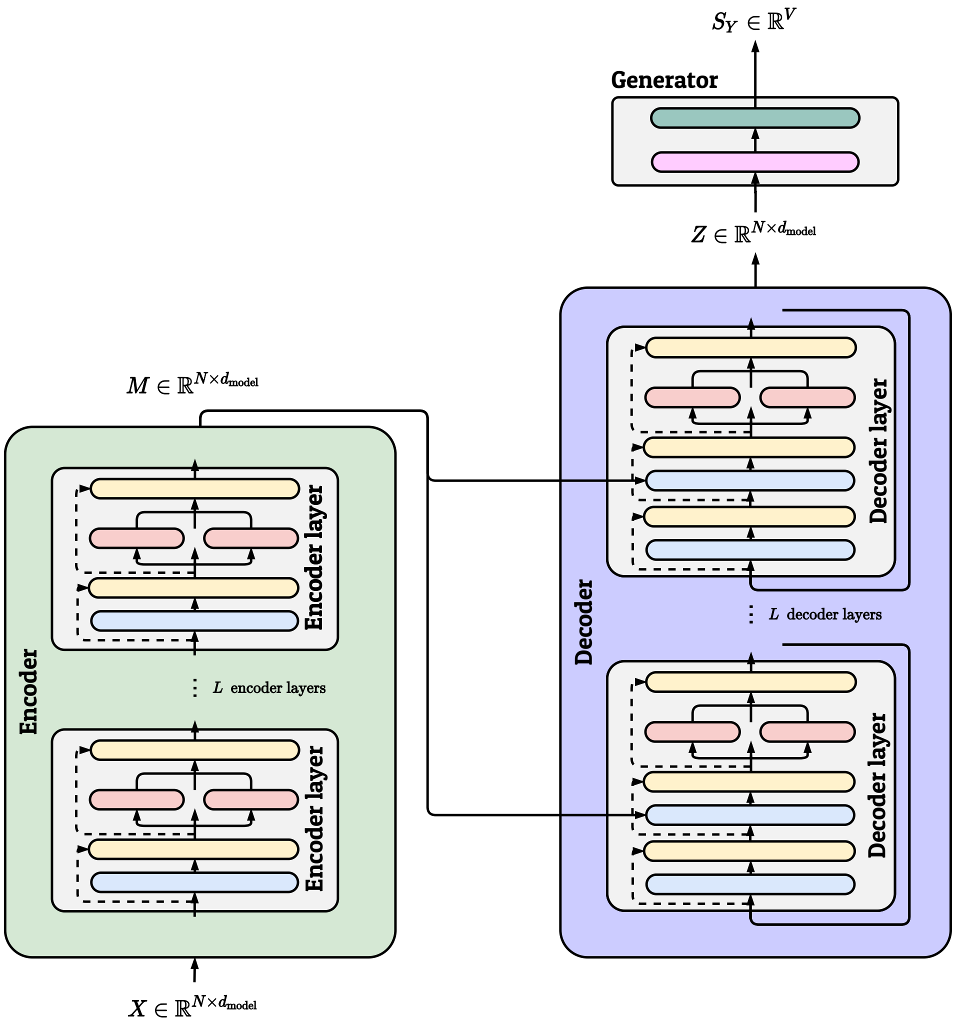
$S_Y \in \mathbb{R}^V$
Generator
$Z \in \mathbb{R}^{N \times d_{\text{model}}}$
$M \in \mathbb{R}^{N \times d_{\text{model}}}$
Encoder
Encoder layer
L encoder layers
Encoder layer
Decoder
Decoder layer
L decoder layers
Decoder layer
$X \in \mathbb{R}^{N \times d_{\text{model}}}$

Figure 3.3: Here we have the full Transformer model. The encoder layers 3.1.2.3, decoder layers 3.1.2.4, and generator 3.1.2.5 are utilised as defined in their relevant sections. The arrows indicate the flow of information between layers, with the appropriate dimensionality specified. Any time X, Z are referenced, they refer to how they are interpreted by the functions as specified in our definition of these concepts.

From this output, Y will then be converted into a sequence $S = (s_0, ..., s_F)$, $s \in \{0, 1\}$, where $F > N$ corresponds to the number of frames in the original video. This will then allow the direct generation of the summarisation video by only keeping those frames that, at their index $i \in F$, have $s_i = 1$.

This conversion from Y to S must be defined, and I choose to use a keyshot formulation that utilises a 0-1 knapsack problem solver along with [11, 59]. The knapsack problem [27] is a bin packing problem where your aim is to maximise the number of items in the "knapsack" without exceeding the capacity, where each item is weighted differently. This is also an accurate description of the problem at hand, where we are instead trying to maximise the frame-level importance scores over frames. The capacity chosen can vary, but as [11, 18] a capacity equivalent to 15% of the original videos length is used.

This is combined with the approach to format the output as a keyshot summary. This is used both for evaluation and for generating the output sequence S. The way this is formulated is to utilise the shot boundaries generated using KTS as described in [44] to set the frame-level scores for each $s \in S$ to be the value of the sub-sampled frame $y \in Y$ that falls within the shot boundary.

As described above, the standard Transformer model uses an input vocabulary and an output vocabulary to embed the input into vectors 3.1.2.1, and subsequently decode the output vectors. An adaption I have made for this problem is to forgo the input embedding of frames, as we already have the d_{model}

dimensional CNN feature vectors for each. This means we will only need an output embedding for our implementation; different choices for this are explored below 3.2.2.1, 3.2.2.2. An important point to make in our choice of output embedding is that it must be sequence-length agnostic. This means that our model could not, for example, output the indices of frames to keep in the summarisation.

Another important deviation from the Transformer model is that we do not allow the output of sequences of varying lengths. The output sequence must be length N, the same length as the input sequence. Practically this will be simple to implement as all we must do is set the maximum length for an output sequence to N before generation. From our definition of the Transformer model 3.1.2.4, this means we change the output sequence length from $M \in \mathbb{N}$ to $M = N$.

3.2.2.1 Direct keyframe prediction

My first attempt at utilising the Transformer for video summarisation was fueled by the initial purpose of the model – translation. This lead to viewing this problem as "translating" the input features from an embedding of size D into a sequence of whether a frame should belong to the summarisation. This can be viewed more formally as taking the input feature sequence X to produce the output sequence $Y = (y_0, ..., y_N)$, $y \in \{0, 1\}$, a binary sequence of length N. The way that we can adapt the Transformer model to perform this binary classification is to make the output embedding size $V = 2$. This removes the need for a conversion to a binary sequence as we are generating it directly. There is, however, the need still for a conversion from the sequence Y of length N to the full summarisation sequence S of length F.

This approach is referred to later as $\text{VST}_{\text{direct}}$. This is a direct approach to summarisation that hasn't been as common in recent research as approaches that utilise frame-importance scores. The main issue with this direct classification is that it is a further abstraction of the problem, which may lead to lost information in the translation. There are a variety of issues usually associated with this kind of binary classification, most notably the bias associated with non-positive class. These problems are explored in chapter 4.

3.2.2.2 Frame-level importance score prediction

The second attempt I took at utilising the Transformer for video summarisation was targeted more at utilising the advantages of the Transformer model. The Transformer is capable of generating sequences composed of elements from large vocabularies. With this I propose constructing a vocabulary that allows us to describe the frame-level importance scores.

Each importance score is a probability representing how likely a frame is to belong to the summarisation. This means that there is no discrete vocabulary that describes all possible probabilities. However, practically we can define a vocabulary that covers the probabilities we care to represent to a certain level of accuracy. We can do this through the following formulation.

For a given number of decimal places $d \in \mathbb{N}$, we can define the vocabulary that represents all possible probabilities as,

$$V = \{\ i/10^d \mid i \in \mathbb{Z}, i \in [1, 10^d]\ \}$$

The size of V is then $10^d + 1$ as we include 1 as a certain probability in our set of potential probabilities.

With this vocabulary, we can utilise the Transformer roughly as intended originally. I use no input embedding and instead use the CNN feature vectors with an output vocabulary V as defined above 3.2.2.1. Similar to the direct approach 3.2.2.1, we can view this formally as taking the input feature sequence X to produce the output sequence $Y = (y_0, ..., y_N)$, $y \in [0, 1]$, a sequence of probabilities of length N. This is then converted to a sequence $S = (s_0, ..., s_F)$, $s \in \{0, 1\}$ through the conversion described above in section 3.2.2. This approach is referred to later as VST_{FLIP}

Chapter 4

Experiments and Evaluation

4.1 Experiments and Evaluation

4.1.1 Datasets and ground truth preparation

In sections 3.2.2.1, 3.2.2.2, I defined the input to the VST framework as being a sequence of d_{model}-dimensional CNN feature vectors corresponding to the sub-sampled frames of the input sequence. For our output, we defined in sections 3.2.2.1, 3.2.2.2 that we will be using an output binary sequence keyshot summary and frame-level importance scores to determine our summarisation. This means that for the datasets we use to tackle this problem, we must curate these necessary features for learning purposes.

Common datasets used for video summarisation are TVSum [55], SumMe [18], OVP [2], YouTube [3]. These datasets have been used by [24, 38, 56, 59, 11, 21] for sequence-to-sequence video summarisation problems. Specifically, [59, 56] have curated these datasets for the purpose of video summarisation in the form that we desire. They extracted the CNN feature vectors for each frame from the penultimate layer of the CNN GoogLeNet [49] pretrained on ImageNet [9]. The specific features found in each dataset can be found in the following table. [59, 11] also converted the different formats of the summarisations found in each dataset to a consistent frame-level importance score format. The different dataset formats are discussed below.

For direct comparison, we use the exact same ground truth data used by [11], similar to that used by [56, 59]. This curated ground truth data contains the information necessary for training: (a) CNN sub-sampled frame embeddings we require as input to our model, (b) The segmented scene changepoints used to break up each video into distinct shots, (c) Sub-sampled frame-level scores and labelled keyshots used for training and evaluation. Specifically, the CNN embeddings size is 1024 for the sub-sampled frames taken by downsampling the original videos to 2 frames-per-second. This means that for our implementation, we will be using $d_{\text{model}} = 1024$. I also take the contribution to this dataset made by [11], who converted the formats found in this dataset to a consistent weighted keyshot based format. The way this was done was to convert the frame-level importance score summarisations as described formally by the following,

For each shot $k \in K$, where i and j are the indices of the first and last frames of the shot, s_k is the score of the k^{th} shot defined as,

$$s_k = \frac{1}{j-i+1} \sum_{n \in [i,j]} f_n$$

Where f_n is the frame-level importance score at index n.

4.1.1.1 TVSum

The TVSum dataset is a dataset created by [55] for the problem of tackling title-based video summarisation. The dataset takes 50 videos from YouTube representing a variety of genres. 10 categories of videos were selected from the TRECVid Multimedia Even Detection task [46], 5 from each category, to get a

Dataset	# of videos	Duration (sec) [11]	Annotations		Category
			#	Format	
TVSum	50	83-647 (avg 235)	20	Frame-level scores	YouTube videos
SumMe	25	32-324 (avg 146)	15-18	Keyshots	User-made videos
OVP	50	46-209 (avg 98)	5	Keyframes	Documentary videos
YouTube	39	9-572 (avg 196)	5	Keyframes	YouTube videos

Table 4.1: Here we list the datasets and their different features. # of videos refers to the number of videos contained in the dataset. Duration refers to the length of each video in seconds. Under Annotations, # refers to the number of people who provided a ground truth summarisation and Format refers to the format for the summarisation. The different formats can be summarised again as follows: (a) Keyshots are binary vectors provided along with shot boundaries that can be used to segment and classify which individual frames belong to shots that belong to the summarisation, (b) Frame-level scores here refer to the frame-level importance scores that describe how likely an individual frame is to belong to the summarisation, (c) Keyframes refers to a binary vector describing which exact frames belong to the summarisation. Importantly, the YouTube dataset contains 50 videos but after removing cartoon videos, which we don't wish to consider for this problem, it contains 39.

representative set of samples for this problem.

The data collected in this dataset that we care about most are the shot-level importance scores collected using Amazon Mechanical Turk. These shot-level importance scores are a rating from 1 (not important) to 5 (very important) provided for each uniform-length shot. These were averaged across all annotations to produce frame-level importance scores as we know.

4.1.1.2 SumMe

The SumMe benchmark dataset is a dataset created by [18] created to evaluate their model for video summarisation that uses an interestingness metric and the 0-1 knapsack problem [27]. The dataset is comprised of 25 user made videos that are raw and unedited.

This dataset contains user summaries formatted as *keyshot* summaries. These keyshots have been created by using their super-frames segmentation approach to delineate shot boundary frames. From these shot-boundaries, users were asked to annotate what shots they believed should belong to the summarisation, producing a binary vector per annotator corresponding to the shots to keep.

4.1.1.3 OVP and YouTube

The open video project (OVP)[2] is an archive of digitized video content available to be used for research. There are a variety of different topics explored in the 3931 videos as of May 2020, most of a "documentary" style. [3] curated 50 videos from this archive and provided summarisations for the purpose of evaluating a model designed for video summarisation. The YouTube dataset is a collection of videos from websites similar to and including YouTube that are diverse in colour, length, motion and subject. They were curated by [3] similarly.

50 users provided 5 video summaries for 5 separate videos each from OVP and YouTube. They did this by choosing the individual frames they believe collated to form an apt summarisation of the video. This is a keyframe approach to video summarisation. This can easily be interpreted as a frame-level importance score format by viewing the binary vector as a probability vector where certain frames are guaranteed to belong to the summarisation.

4.1.2 Implementation details

The models as outlined in sections 3.2.2.1, 3.2.2.2 have been implemented in PyTorch [43] and can be found along with all code used for this project at the following repository `https://github.com/tim-roderick/VST`. Importantly, I make specific design choices when implementing the VST framework for training purposes and practicality. Those choices can be found here with justification.

Parameter	Value
L	6
h	8
d_{model}	1024
d_{ffn}	-

Table 4.2: Here we list the parameter values chosen for our implementation of VST: (a) L refers to the number of layers in the encoder or decoder, (b) h refers to the number of attention heads per layer and must divide d_{model}, (c) d_{model} is the width of the input vectors that are then propagated throughout the rest of the model, (d) d_{ffn} refers to the width of the feed-forward network layers found within each layer of the encoder and decoder.

For the number of individual encoder and decoder layers, I chose $L = 6$ as [50]. This was chosen due to the convention of previous work using the Transformer, and that there was a limitation of the model size usable on the hardware available. The benefit to more layers is that it adds more representations of the sequence as it passes through the layers. A discussion of these representations and their effectiveness as they increase in number can be found in section 4.1.4.5.

Similarly, the same reasoning can be applied to the number of attention heads found in each self-attention layer. For this, I choose $h = 8$ also following [50]. The number of attention heads per layer is limited for the same reasons as the number of encoder or decoder layers. The trade-off between these two however is more interesting. As we decrease the number of attention heads per layer, we can increase the number of layers to fill the same amount of space. This trade-off is explored in section 4.1.4.5.

As described in section 4.1.1 we are choosing a model width of $d_{\text{model}} = 1024$ as that is the width of the CNN feature vectors provided in the curated datasets we will be using. For the feed-forward networks found in each layer, d_{ffn} is a variable we can vary. For our purposes initially, I explore $d_{\text{ffn}} = 1024$ as in [50]. For our choice in granularity of the dataset frame-level importance scores as described in 3.2.2.2, we utilise $d = 5$ meaning we have 5 decimal places of accuracy in our probabilities. This was chosen as it meant that our use of the dataset, as prepared by [11], has no loss associated and can be utilised as provided.

4.1.2.1 Decoding Schemes

We have discussed the generator in section 3.1.2.5 where we describe how we repeatedly generate distributions over the output vocabulary to choose the next element in the sequence. The reason we do this is to generate the sequence that maximises the likelihood of being correct according to our model. However, doing this is not trivial as to find the true maximal likelihood element, you would have to consider all potential sequences to determine their relative probabilities. Formally this problem is finding for input sequence X and output sequence S

$$\text{argmax}_S \; \mathbb{P}(S|X)$$

This is where we must utilise specific decoding schemes used in sequence-to-sequence problems [45]. A variety of approaches are described below, but due to the length of the sequences we are considering we mainly focus on *greedy decoding* and *beam search decoding.*

Exhaustive Search - This is the most costly approach to decoding we will consider. In this approach, we would consider all sequences possible to generate and compare them to find the most probable. This has a computational complexity of $O(V^N \cdot D)$ where V is the vocabulary size, N is the length of the sequence and D is the time to decode each element. This for any reasonable sized vocabulary or sequence is intractable, but does guarantee the most likely sequence is found.

Greedy Decoding - This is the simplest approach to decoding we will consider. In this approach, we choose the most likely element of the vocabulary for each decoder iteration. This has the disadvantage of losing out on potential more likely sequences due to decisions made earlier in the sequence. The complexity of this approach is $O(N \cdot D)$, for sequence length N and time to decode at each iteration D, and is therefore very reasonable to compute for longer sequences.

Beam Search - Beam search [13] as an approach tries to find a middle ground in decoding, where we consider as much as we reasonably want out of the potential sequences but sacrifice reaching the analytical solution. At each iteration of beam search, we consider the K most likely options from the output of our decoder so far and continue considering these sequences only. This advances the decoding as a “beam”. This allows us to consider the exact number of potential sequences we want each iteration. The complexity of this approach is $O(K \cdot N \cdot D)$ where N is the length of the sequence and D is the time to decode at each iteration. This is more feasible for small K and allows us to consider far more options than greedy decoding for linear additional computational cost. For $K = 1$, this problem is exactly the greedy decoding approach.

Random Sampling - This approach is similar to the greedy decoding approach. Here instead of choosing the most likely element of the vocabulary at each decoder iteration, we instead sample from the output distribution of the decoder. This by itself has the same time complexity as the greedy decoding approach. We can repeat this process however, as each time we do so is effectively sampling another sequence from the total output sequence's distribution. This means that if we sample X times, we are effectively performing the greedy decoding approach X times. This means that the computational complexity is $O(N \cdot X \cdot D)$ for sequence length N and time to decode at each iteration D. The disadvantage of this approach in comparison to beam search, which has similar computational complexity, is that it doesn't have any guarantees for greater performance per extra X iteration – it is stochastic.

4.1.2.2 Beginning-of-Sequence Token

Commonly in approaches that use the Transformer for machine translation, the sequences begin with a token described as the beginning-of-sequence (BOS) token, and end with the end-of-sequence token (EOS). This is used to provide an element that receives no probability of being produced that can be fed to the decoder to generate the first element of the sequence. The EOS token is commonly used for generating sequences less than the maximum sequence length specified.

For our purposes, we set this maximum length to the size of the input sequence and remove the inclusion of the EOS token entirely. We will still however include the BOS token and add that to the beginning of each sequence in our ground truth. This will involve increasing our vocabulary size by 1 for each implementation.

4.1.3 Training

To train the VST model, we follow many of the techniques implemented by others using the Transformer model for machine translation. The model was trained on an Nvidia P100 GPU with a variety of parameters leading to large changes in training time, discussed in 4.1.4. Many of our choices were influenced by the OpenNMT [30] implementation of the Transformer and the blog post exploring said implementation [1].

4.1.3.1 Loss computation

The way we consider loss during training is similar to that of many distribution based objective functions – Kullback–Leibler divergence (KL Divergence) [33]. This is applied in the OpenNMT implementation of the Transformer [30], but is also common in general in encoder-decoder schemes as they are also estimating an output distribution. Also a common choice in this scenario is Cross-entropy loss [40].

We also consider a regularisation addition to our loss computation called *label smoothing*. Label smoothing acts to even out the distribution across all labels to improve regularisation and prevent overfitting to specific labels. This is done by taking our distribution over the vocabulary of our ground truth at any point in the sequence, a 0 vector with one element that is 1, and reducing our certainty in our classification by distributing some of the probability equally among all other elements in the vocabulary. Formally,

For some iteration of the decoder i, we produce a distribution S_i over the vocabulary describing the likelihood that an element should be next in the sequence. We perform a loss computation with this distribution and our ground truth distribution. Our ground truth distribution is e_j where the appropriate next element in the sequence is the j^{th} element in our vocabulary. e_j can be thought of as a basis vector.

Our label-smoothed output ls_i for a given smoothing coefficient $s \in [0, 1)$ is,

$$ls_i = \begin{bmatrix} s \\ s \\ \vdots \\ s \end{bmatrix} + e_j \times (1 - s)$$

This is visualised in diagram 4.1.

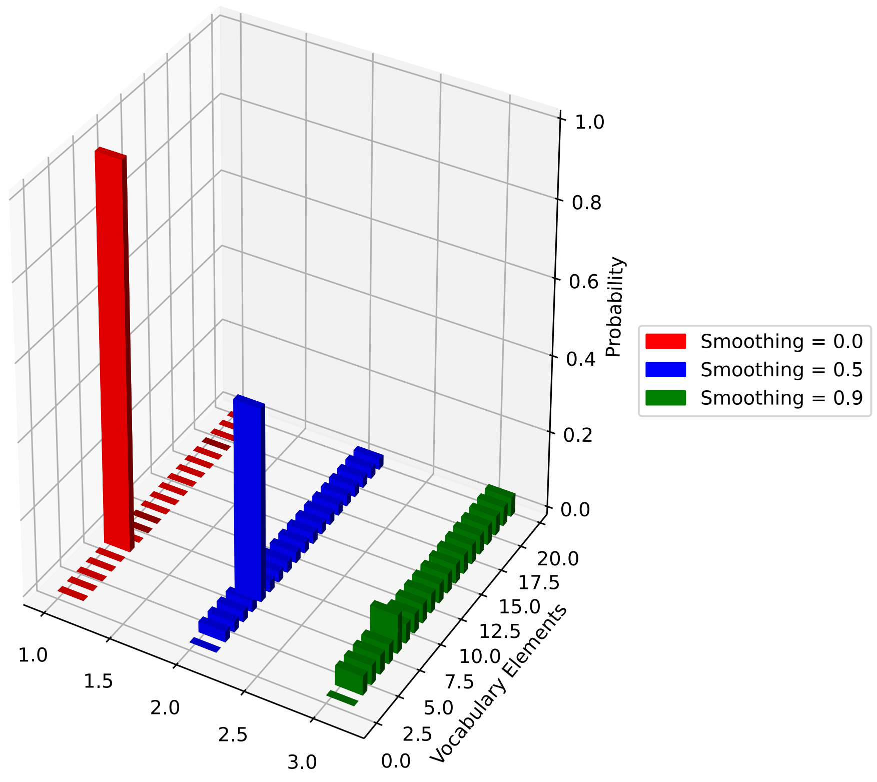

Figure 4.1: In this plot, we visualise three bar charts. In all these charts, we are visualising the label smoothed distribution of a vector $e_5 \in \mathbb{R}^{20}$. In red we have the distribution with no smoothing. In blue we have the distribution with $s = 0.5$. In green, $s = 0.9$. These plots highlight how the weight is "smoothed" over the probability space to add regularisation to our training.

4.1.3.2 Optimiser

The Adam optimiser [29] was used for training, where the decay rates β_1, β_2 were set to (0.9, 0.98) and denominator stability term $\epsilon = 1 \times 10^{-9}$.

Following the choice made in the original Transformer model training procedure [50], I also implement the slow warm-up to eventual drop-off in the learning rate l_{rate}. This is described formally below. Importantly, we do not implement batching in the same way as the standard Transformer model as explained in section 4.1.3.5. Because of this the learning rate term is scaled to be smaller so that it is not swayed significantly by an individual training sequence.

For n the current training step, w the number of warm-up steps, and s the scale factor,

$$l_{\text{rate}} = s \cdot d_{\text{model}}^{-0.5} \cdot \min(n^{-0.5},\ n \cdot w^{-1.5})$$

This has the effect of making the initial increase in learning rate gradual up to the number of warm-up steps, decreasing after at a rate of the inverse square-root of the step number. The exact parameters in our training where $s = 1,\ w = 400$.

4.1.3.3 Teacher Forcing

As a machine translation technique, teacher forcing [17] is a common approach to help early training of the model and to help the training converge quicker. In teacher forcing, instead of feeding the output of the model back into the decoder during training, we provide the decoder at each iteration with the correct value from our ground truth.

The main benefits of this are that initially, during training, this provides the model with the help necessary to converge. It also makes convergence quicker. This has the unfortunate downside of causing a potential difference in performance between inference and training – exposure bias [20]. For the VST implementation, we utilise teacher forcing for our implementation.

With the inclusion of teacher forcing however comes a new issue. For if we provide the correct values of the sequence at each run through the decoder, it may learn to attend to elements in the future of the sequence which haven't been generated. This is where we implement masking. Masking allows us to, as the sequences pass through each layer, remove all weight placed to elements of the sequence that are after the element being generated. This means that any element in the decoder can only have self-attention that attends to the sequence generated so far. This is done practically during the last stage of each self attention function, where a mask is applied to each attention head H_i.

As we also want to provide no input at all to the very first iteration of the decoder during training, we must provide the output sequence we know is correct for teacher forcing but right-shifted.

4.1.3.4 Train-Test Split

As [11, 56, 59], I use a 5-fold cross validation. Specifically, I use the exact same train/test splits as in [11] for direct comparison. The methodology behind how these were generated is as follows.

OVP and YouTube do not by themselves provide the necessary information for keyshot evaluation as we have defined prior, full details of the dataset can be found in section4.1.1. Because of this, the two datasets we consider for evaluation are *TVSum* and *SumMe*.

There are two settings used for both these dataset, canonical and augmented. Canonical refers to generating the splits based on 80% of the samples from a dataset are used for training and 20% are used for testing. Augmented splits for both these datasets are formulated by creating a similar split ratio as found in the canonical split regime. The important difference, however, is that the training sets are *augmented* by also including all of the samples for training from the other datasets not used including OVP and YouTube.

This means that we will have four splits to consider for evaluation:

1. TVSum canonical split
2. TVSum augmented split using OVP, YouTube and SumMe in training
3. SumMe canonical split
4. SumMe augmented split using OVP, YouTube and TVSum in training

4.1.3.5 Batching

We have described above how we consider each sequence for training and how we draw these sequences from our datasets. Now we must consider how this is then applied to batched sequences. Commonly [50, 30], batching is applied in machine translation for training. This however comes with the issue that each sequence in the batch must be of the same dimension. This is counteracted by most by grouping sequences based on their length, padding where necessary. But following [11], we choose to not batch our sequences as the datasets are already limited in size. This removes the need for complex padding procedures that can tamper with training and can easily be accounted for with changed parameters as described in our choice of learning rate.

4.1.4 Comparison and Analysis

Here I will discuss how the VST framework has performed on the datasets provided, and will explore how the model has adapted to the video summarisation problem. I will also compare my results with other state of the art models that have been formulated to tackle this problem [24, 38, 56, 59, 11, 21].

4.1.4.1 Evaluation

To discuss our results, especially in comparison to other state of the art models for vide summarisation, we must consider *what* a successful summarisation for a video is. This, as you may imagine, is very subjective. What one person may describe as an apt summarisation of a video may be lacking to another. We must consider whether a summarisation should be informative, or visually descriptive, or representative, or concise, or visually stimulating. This list of preferences is impossible to complete, which is why we try to capture this notion latently. Methods described in chapter 2 that focused on visual relevancy are a suitable example for describing a specific principle that could lead to successful video summarisations, but do not capture the inherent nature of what a person desires in a summarisation.

This is why we evaluate based on labelled ground truth summarisation from users, to attempt to capture this notion. I have defined in the model definition of VST 3 that the output we are generating is a binary sequence of length F corresponding to the frames we are going to keep in the summarisation. This scheme can also be described as a $keyframe$ approach to summarisation, as we used to describe OVP and YouTube in 4.1. For evaluation however, we are choosing to use keyshots as [11, 56, 59]. The struggle with this in video summarisation is that, unlike many other deep learning tasks, there is not a sufficiently large dataset of user annotations. The greater this breadth of ground truth, the more successful future models may become.

To evaluate our machine summarisation against human user annotations, we compare the decisions made for which frames to keep in the summarisation. Our model can be deemed more successful if it imitates the user annotations accurately. However, it is important to note that this does not determine whether a model has successfully created a summarisation. One may watch the summarisation produced by the model and decide that it is more informative, or visually descriptive, or representative, or concise, or visually stimulating. Fortunately, as shown by [18], human annotations have been shown on the datasets curated to be highly correlated and consistent. This brings confidence to the idea that matching our model to our ground truth should lead to capturing some greater notion of how to summarise a video.

As we assess our results based on the correct selection of frames from a sequence, we can use the F_1 score as [56, 59, 11, 35]. The F_1 score [47] is a measure of binary classification accuracy. This score is formulated as the weighted harmonic mean of the $precision$ and $recall$,

$$2 \cdot w \cdot \frac{precision \cdot recall}{precision + recall}$$

This can be expressed as a percentage by setting $w = 100$. This represents a weighted comparison of two other useful analysis in $precision$ and $recall$. These are measures that take into account the false positive rate FPR, true positive rate TPR, and false negative rate FNR. We view a True positive as the overlap in our machine classification and our ground truth, a false negative as a lack of overlap, and a false positive as when a frame is incorrectly selected to belong to the summarisation. $precision$ can then be defined as the ratio $\frac{TPR}{TPR+FPR}$. $recall$ can be described as the ratio $\frac{TPR}{\text{Total number of positive classifications}}$. This score weights $precision$ and $recall$ equally which may be a detriment in certain scenarios. For our use, this is appropriate and will provide us a measure of our classification accuracy.

4.1.4.2 Discussion of initial results

As discussed in the formulation of the VST framework, we generate an output sequence S_f which corresponds to the frames to be kept in the summarisation 3.1.2.4. Here, we will discuss the preliminary results for these outputs and how they compare to our ground-truth. For the frame-level importance score formulation 3.2.2.2, we generate an output of scores that we then convert into a summarisation sequence through a solver for the 0-1 knapsack problem. Implementing the training procedure as described in section 4.1.3, we generate summaries for the samples in our evaluation set. We use the parameters as described above in table 4.2. The following figure 4.2 illustrates our summarisation in comparison to the ground truth video summarisation for video 23 in the TVSum dataset (a member of our evaluation samples). Another important trend is revealed upon initial analysis of generated summaries. What can

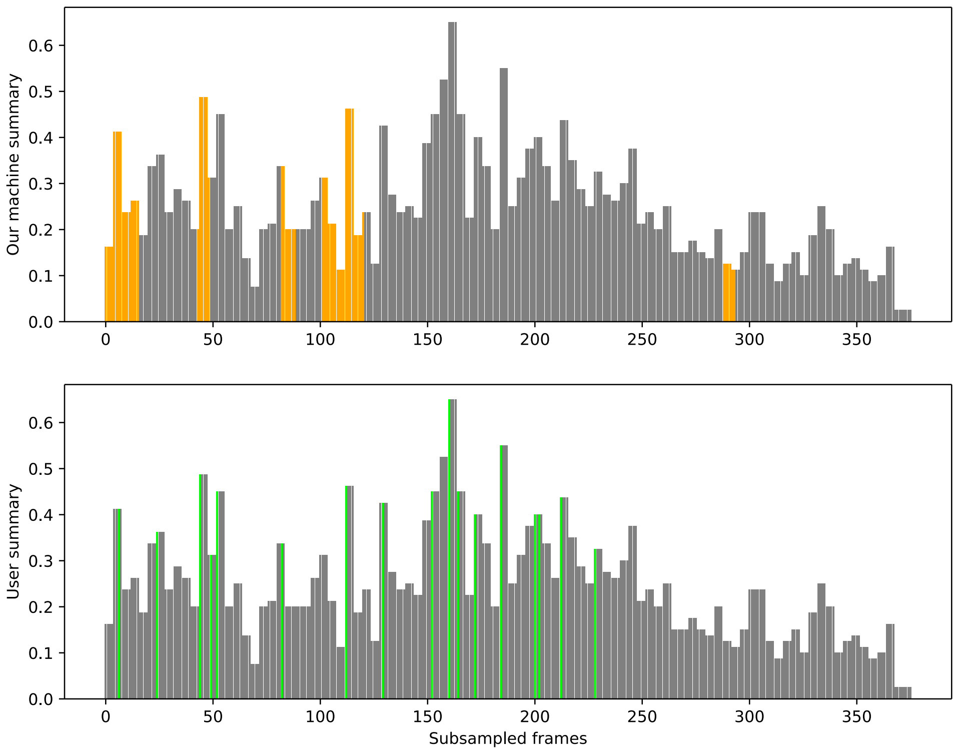

Figure 4.2: In this plot, both graphs show the frame-level importance scores for all subsampled frames found in our ground truth scores for TVSum video 23. On the top plot is the summary generated by our model, the bottom is a ground truth summarisation curated by an individual. Immediately some similarity can be seen between the two, with our machine summarisation creating larger groups of frames to belong to the summarisation than the user summary. The model for this was trained as described in the implementation details with greedy decoding and $d_{\text{ffn}} = 4096$.

be found when looking at the frame-scores, before using the knapsack problem solver, is that the scores are mostly uniform. This means that the model very rarely places probability with scores that deviate from some consistent value.

This highlights an important downside found in the implementation of VST. We are generating at each decoder step an element in a sequence. This element is a *probability* as we have defined prior. However, we instead describe this probability as one of several thousand other elements in a vocabulary. This means that the weight we assign to any individual element in the vocabulary is close to 0. But for common elements in the vocabulary, for example 0.2, we find far more weight is distributed. This means that our model loses the sensitivity found in standard linear neural network models due to its lack of summative nature. Subtle changes go unnoticed due to being uncommon, not irrelevant. Because of this, and the choice of greedy decoding, we rarely choose a value that differs from the most common.

We also consider the output generated from the direct approach described prior 3.2.2.1. Here, our output vocabulary is one of two elements. These elements describe whether an element in the the sequence should or should not belong to the summarisation respectively. This has the problem, similar to the previous approach, in that it places too much probability with the most common element in the sequence – 0. Because of this it rarely chooses an element to belong to the summarisation as it has a strong bias. Initial exploration of the results leads to the conclusion that the model almost always produces a full sequence of 1's or 0's potentially for this reason.

The issues described above can be mitigated through tools and techniques such as label smoothing, beam search, and other parameter changes. This will be explored in the following sections.

4.1.4.3 Visualising attention

When we consider self-attention as a tool for capturing sequence dependencies, we have the added benefit of a direct way to describe how each element of the sequence depends on other elements. Based off of the preliminary exploration described above, we would like to visualise the attention found in our implementation of VST to assess *how* it is forming sequence dependencies.

In figure 4.3, we visualise the relative attention weights for the encoder as trained on the TVSum dataset. When compared with figure 4.2, you can match the frame-scores to their position and observe its weighting. Patterns are immediately visible, with several vertical streaks through the attention-weight space. This implies that there are a few indices in the input sequence that receive a greater weighting for all elements in the sequence. There is also a pattern of the values becoming more subtle as the layers progress. It is also of note that as the sequence is large, these weight values are very small. This leads to issues as even large differences get lost in translation.

We also visualise the decoder attention, shortened to the length of the sequence found in video 23 of the TVSum dataset. This is found in figure 4.4. As we mask future elements of the sequence in the decoder, all values past an individual element are 0. Patterns are immediately visible, with several vertical streaks through the attention-weight space as with the encoder. This implies that there are a few indices in the input sequence that receive a greater weighting for all elements in the sequence. Noticeably more than the encoder, the different attention heads have formed significantly different representations of the sequence. Although the same vertical patterns can be observed in each head. There is also a pattern of the values becoming more subtle as the layers progress. Specifically, the values eventually start to be heavily dependent on the distance from the beginning of the sequence. This implies a greater weighting is given to early elements of the sequence. If this is indicative of a pattern in the summarisations, this could be related to the temporal bias as explored by [18]. It is also of note that as the sequence is large, these weight values are very small. This leads to the same issues as described with the encoder.

4.1.4.4 Effect of changing d_{ffn}

We discussed in our implementation details of the VST the feed-forward network 3.1. The width of this feed-forward network is a variable parameter that can have a large effect on the results we accrue. As observed by [50], the increase in the width of d_{ffn} lead to performance increase with an eventual decrease in performance. The reasons for this are unclear, but could easily be attributed to an increase in complexity causing overfitting. For our implementation, we found choosing $d_{\text{ffn}} > 2048$ lead to performance decrease on our evaluation set, so we left $d_{\text{ffn}} = 2048$.

4.1.4.5 Discussion of L and h

The purpose of the increased number of layers L and attention heads h is not obvious. The Transformer model was analysed by [51], who isolated what affect greater attention heads and layers provided to the task of translation. As we still use the Tranformer model fundamentally, these traits are transferable.

[51] found that attention heads were found to target lexical patterns. This means that, to an extent, an increase in the number of attention heads would tend to lead to capturing more complex patterns found in the frame-sequence. From the visualisation of the decoder self-attention in figure 4.4, we can observe that specific attention heads seem to capture patterns in the form of isolating specific frames where weighting is placed for all elements in the sequence – the visible vertical streaks. From this, the implication may be that greater number of attention heads could lead to greater focus on pivotal frames.

[51] also found that distant dependencies were captured by the deepest layers. This would imply that the deeper layers of our implementation capture dependencies between frames that are farther apart in our video. As our sequences are very long in comparison to those normally used in machine translation, this would seem to be a significant benefit. From the visualisation of the decoder self-attention in figure 4.4, we can observe that as the layers move deeper into the decoder, the vertical streaks as described prior become smaller and more numerous. This would have the implication that a greater number of longer-distance dependencies are being weighted with significance as the layers move deeper, as each streak can be considered a consistent dependence on a frame for all frames. The issue with this however is that these dependencies become significantly smaller as the layers progress, which means that

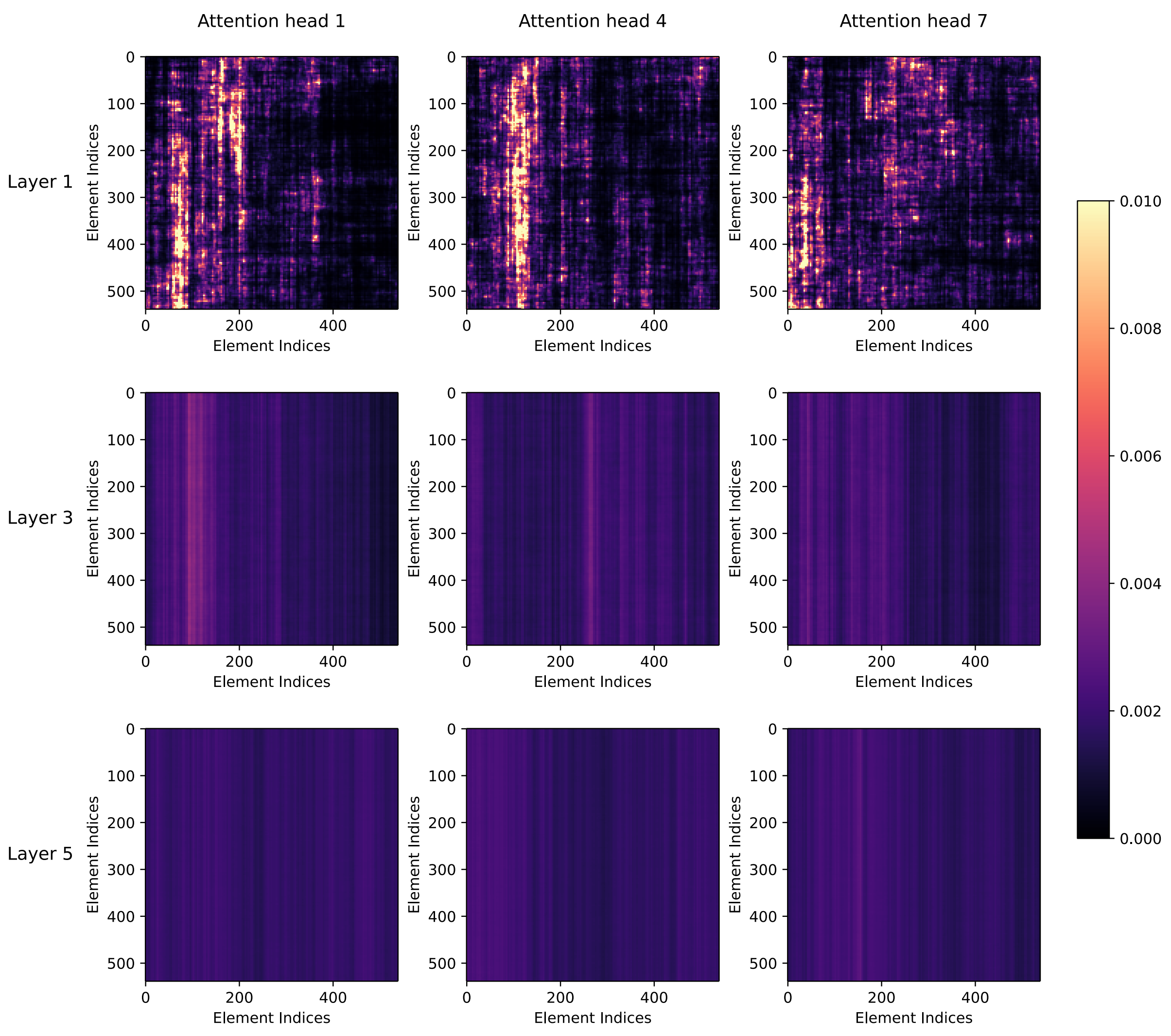

Figure 4.3: Here we have visualised the self-attention weightings for the input sequence encoder. This was taken from the model trained on the TVSum dataset. The layer numbers refer to the depth of the layer in the encoder, and the head number refers to attention head we are visualising. The element indices refer to the sequence elements at those indices, describing how the element at a given index relates to another. We have chosen a subset of the layers and attention heads that capture a representative view of our self-attention. Patterns are immediately visible, with several vertical streaks through the attention-weight space. This implies that there are a few indices in the input sequence that receive a greater weighting for all elements in the sequence. There is also a pattern of the values becoming more subtle as the layers progress. It is also of note that as the sequence is large, these weight values are very small. This leads to issues as even large differences get lost in translation.

they do not contribute significantly enough to the choice of importance-score to affect the summarisation.

Importantly, [51] also found that the patterns formed were heavily dependent on the training objective. This could have significant implications for how the model performs. For example, as the model is a sequence-to-sequence model that generates the sequence one element at a time, a dependence on the immediate surrounding frames can be found to grow more-so than those models that generate the sequence at once. This could also provide motivation for the strong dependence, found in layers of the decoder, for earlier elements in the sequence.

For our practical implementation, with limitations, we found that a balance of $L = 6$ and $h = 8$ lead to the most performance from our model. This is also similar to the results found in [50], where a similar ratio was found to be more effective than a strong preference for extra layers or attention heads.

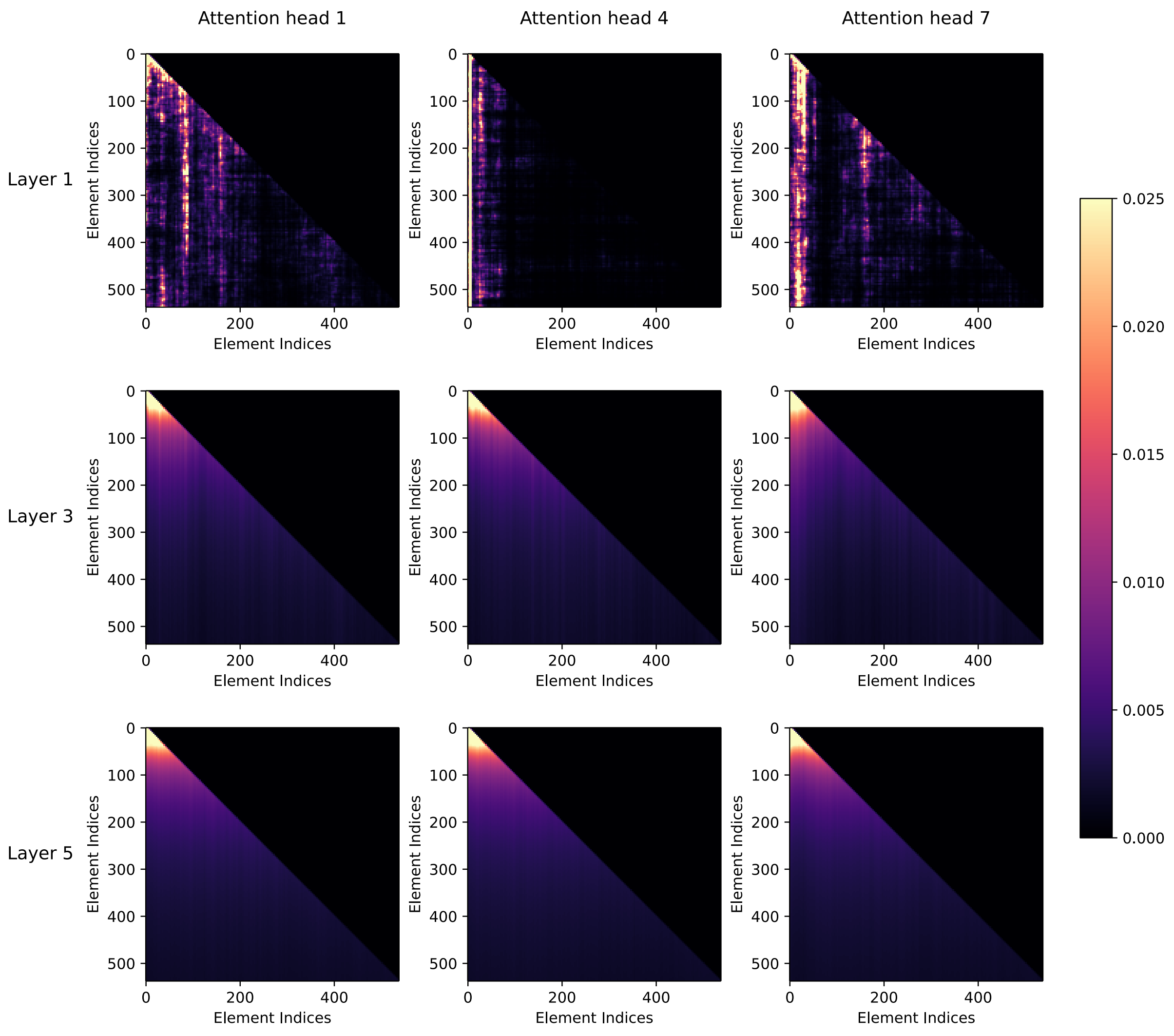

Figure 4.4: Here we have visualised the self-attention weightings for the input sequence decoder. This was taken from the model trained on the TVSum dataset. The layer numbers refer to the depth of the layer in the decoder, and the head number refers to attention head we are visualising. The element indices refer to the sequence elements at those indices, describing how the element at a given index relates to another. We have chosen a subset of the layers and attention heads that capture a representative view of our self-attention. As we mask future elements of the sequence in the decoder, all values past an individual element are 0. Patterns are immediately visible, with several vertical streaks through the attention-weight space. This implies that there are a few indices in the input sequence that receive a greater weighting for all elements in the sequence. Noticeably more than the encoder, the different attention heads have formed significantly different representations of the sequence. Although the same vertical patterns can be observed in each head. There is also a pattern of the values becoming more subtle as the layers progress. Specifically, the values eventually start to be heavily dependent on the distance from the beginning of the sequence. This implies a greater weighting is given to early elements of the sequence. If this is indicative of a pattern in the summarisations, this could be related to the temporal bias as explored by [18]. It is also of note that as the sequence is large, these weight values are very small. This leads to the same issues as described with the encoder.

4.1.4.6 Label smoothing

Label smoothing is a concept discussed and defined in section 4.1.3. The effect label smoothing has had as a regularisation term for our frame-level score output is very mixed. Depending on the decoding scheme used, further explored in section 4.1.4.7, label smoothing has had both positive and negative effects for evaluation over all dataset configurations.

For the direct summarisation approach, results where mixed. Label smoothing counteracted the strong bias found for either the 0 or 1 label. This difference however was small and only improved canonical results on the TVSum dataset, having a worse effect on the SumMe dataset. For augmented settings, this made little difference.

For the frame level-importance score approach, label smoothing did help with regularisation, improving augmented settings and some test evaluations. This however did imbalance the accuracy found in training causing a decrease in performance elsewhere.

In summary, label smoothing provided no significant benefit to our approaches, but can be applicable in specific settings to improve regularisation and subsequently performance.

4.1.4.7 Beam search and greedy decoding

Beam search, greedy decoding, and other decoding schemes were explored in section 4.1.2. Here we discuss how these decoding schemes have an effect on evaluation.

Greedy decoding in our implementation has been our baseline as it is relatively simple and inexpensive to perform. In most scenarios, implementing beam search made little difference to our results due to the issues associated with a large vocabulary discussed in section 4.1.4.3 . This is because there is still so little weighting placed among most of the vocabulary that the resulting beam search will most likely find a similar solution as the greedy decoding. This does however lead into the benefit of combining this with label smoothing.

When implemented alongside label smoothing, beam search improved the evaluation results as the distribution over the vocabulary is more spread. This lead to greater consideration of elements in the vocabulary that could lead to more performance. As described above, the slight negative impact label smoothing had on performance seems to mitigate the performance gain made through utilising beam search.

In summary, in very few cases did employing beam search over greedy decoding improve our evaluation results. In certain scenarios, beam search provided an improvement to our evaluation results but in these cases, the improvement was relatively insignificant and not universal across our evaluation datasets.

4.1.4.8 Augmentations effect on the model

As described in our training/test split arrangement 4.1.3, we augment the SumMe [18] and TVSum [55] datasets with data from OVP [2, 3] and the YouTube [3] dataset. In practice, this means that the ground truth scores found in the OVP and YouTube datasets are used to train the VST model, but are not used during evaluation. These training samples take the form of keyframe summaries, binary vectors. This is still interpreted as frame-level importance scores, where the keyframes are viewed as belonging to the summarisation with certainty.

When assessing the performance of the augmented setting, we compare with the canonical splits to assess whether the datasets provide a greater improvement on the evaluation set.

Performance on the augmented splits was mixed. When used with greedy decoding, the augmented model performed better on both the SumMe and TVSum datasets. However, when used alongside label smoothing or beam search decoding, performance decreased. Why this is the case is initially unclear, but with some inspection the following is a reasonable explanation.

Unlike other approaches that tackle video summarisation using self-attention, [11, 35], we form a distribution over all the potential frame-level importance scores as a vocabulary. Explored briefly in the attention visualisation 4.1.4.3, we see that this may lead to certain importance scores being predominantly chosen due to the breadth of the vocabulary. What augmentation has done to this distribution is that it has increased the weighting associated specifically to the vocabulary elements 0 and 1. This in some cases has had improvement, as frames that would ideally have less focus are more likely to be distributed 0 as a frame-level importance score. However, in label smoothing and beam search this has not been the case. This is likely because the likelihood that a frame would be classified as having a frame-level score of 0 and

Model	TVSum		SumMe	
	Canonical	Augmented	Canonical	Augmented
[24]	61.0	61.8	44.4	46.1
[38]	56.3	61.2	41.7	43.6
[56]	54.7	59.6	38.6	42.9
[59]	58.1	59.8	42.1	43.9
[11]	61.4	62.3	49.7	51.0
[21]	59.4	-	47.2	-
VST$_{\mathbf{direct}}$	11.1	0.0	11.8	0.0
VST$_{\mathbf{FLIP}}$	58.0	57.4	27.1	43.0

Table 4.3: I provide the evaluation of both formulations of the VST framework on all of the evaluation splits specified in 4.1.4.1. For the VST$_{\text{direct}}$ approach, a $d_{\text{ffn}} = 4096$ was used. No label smoothing was applied and greedy decoding was utilised for sequence generation. All other parameters were set to the defaults as defined in section 4.1.2. For the VST$_{\text{FLIP}}$ approach, a $d_{\text{ffn}} = 2048$ was used. No label smoothing was applied and greedy decoding was utilised for sequence generation. All other parameters were set to the defaults as defined in section 4.1.2.

1 becomes more likely, leading to erratic behaviour in score classification and reducing evaluation accuracy.

In the model proposed by [11], or others that utilise augmentation for video summarisation [56, 59], we can see that this augmented inclusion is used to increase the weighted contribution of that element to its score. This is directly related to the summative, distributed nature of large feed-forward neural network layers. In contrast, our approach instead places greater individual likelihood at a specific choice – the distribution is imbalanced not skewed.

4.1.4.9 Comparison with state of the art

In this section, I will compare the evaluated results of our VST approach against other state of the art approaches. I will be comparing my results against those other approaches that in their literature evaluate their results on the SumMe and TVSum datasets using the F_1 score as described in 4.1.4.1.

In table 4.3, I provide the evaluation of both formulations of the VST framework on all of the evaluation splits specified in section 4.1.4.1. For the VST$_{\text{direct}}$ approach, a $d_{\text{ffn}} = 4096$ was used. No label smoothing was applied and greedy decoding was utilised for sequence generation. All other parameters were set to the defaults as defined in section 4.1.2.

The results from VST$_{\text{direct}}$ highlight some serious issues with this approach, many discussed in the previous subsections of section 4.1.4.1. Noticeably, any F$_1$ score found greater than 0 for this approach was the result of predicting every frame as belonging to the summarisation for certain evaluation samples. This highlights the problem of the bias formed for one of the two elements found in the output vocabulary. Label smoothing when considered improved evaluation results for certain samples, but not universally. This problem is even further illustrated by the results in the augmented setting. For these samples, the model had been trained on the samples provided from OVP and YouTube datasets that provide further bias for the 0 element of the vocabulary due to the sparsity of summarisation frames.

For the VST$_{\text{FLIP}}$ approach, a $d_{\text{ffn}} = 2048$ was used. No label smoothing was applied and greedy decoding was utilised for sequence generation. All other parameters were set to the defaults as defined in section 4.1.2. The merits of this approach are more noticeable, with comparable results found in the canonical TVSum setting and the augmented SumMe setting. The large difference between augmented and canonical settings in our results for this approach can be attributed to the issue of 0/1 bias found in the frame-level importance score augmentation as described in section 4.1.4.8. In some cases this provides benefit, some others not so. This would reduce the merit found in the augmented setting results for SumMe. Because of this, utilising beam search decoding in this setting exacerbates the issue, further widening the difference in performance of the two settings.

Chapter 5

Conclusion

5.1 Conclusion

In this project, I have proposed an interpretation of the popular Transformer model [50] for video summarisation. Specifically I propose two approaches to utilising the structure of the Transformer for video summarisation; VST_{FLIP} and $\text{VST}_{\text{direct}}$. VST is formulated as utilising the multi-layer, multi-head, self-attention structure of the Transformer model to perform sequence-to-sequence modelling. The proposed framework is also an encoder-decoder scheme as the Transformer model. Through this, I thoroughly explore the Transformer model as introduced by [50]. The sequences we provide to this model are the CNN spatio-temporal feature vectors generated for the frames of a video, with the output a binary sequence representing the frames we wish to keep for a summarisation. This supervised video summarisation approach is evaluated as a keyshot summarisation against state of the art models for this problem, achieving comparable results in the form of VST_{FLIP}.

We utilise the SumMe, TVSum, OVP, and YouTube datasets for training and evaluation, using the same ground truth data as [11]. VST_{FLIP} uses the Transformer model structure to generate the frame-level importance scores for each frame, whereas $\text{VST}_{\text{direct}}$ attempts to directly generate the binary output sequence. The full implementation details can be found in the following GitHub repository `https://github.com/tim-roderick/VST`.

The benefits of this approach come in self-attention's computational simplicity in comparison to LSTM based schemes, with comparable returns. The drawbacks of this approach are found in the form of the output of our generator. Further extensions to this project can be found in the following.

5.2 Future Work

5.2.1 Deeper architecture

Upon observation of the initial results of the VST framework, the way the model has learnt to attend to different positions in the sequence is promising. A further extension to the project as proposed would be the consideration of a deeper model. This would involve increasing the number of layers L and attention heads h. The benefits to this are the greater number of representation spaces of the sequence as described by [51].

For machine translation [50] found that after increasing the number of attention heads, the performance improved with an eventual drop-off. This drop-off is not trivial to diagnose, but a reasonable explanation could be found in overfitting. By providing too many attention representation spaces the patterns found in the sequence could be over-fit too leading to decreased performance. An important note to make however is that the sequence lengths tackled when performing machine translation (individual sentences) are far smaller than the sequences generated when tackling this problem (subsampled video frames). This could highlight the need for a greater number of layers and attention heads to represent the many patterns possible in the dependence of video frames.

5.2.2 Implementing VST with a smaller vocabulary

When implementing $\text{VST}_{\text{direct}}$ and VST_{FLIP}, the consideration for the granularity taken in our assessment of each video frame was vital. For $\text{VST}_{\text{direct}}$, an abstract vocabulary is considered. This utilised the ground truth summarisation in a keyframe based format for training. For VST_{FLIP} however, we still utilised the ground truth scores provided for each subsampled frame by forming a vocabulary of the approximate probabilities. Importantly, we chose $d = 5$, where d is the number of decimal places. This meant that no further processing needed to be performed on our ground truth scores as the probabilities provided were all accurate to 5 decimal places. This lead to the core issue of this approach found in the sparsity of the vocabulary generated. Many values were not ever considered as they were never present in the training data. This lead to the distribution over our values being too sparse and dominated by specific more common vocabulary elements.

An extension I propose to this is the consideration of smaller values for d. By approximating the ground truth scores to this many decimal places, some information may be lost. But as we perform the keyshot based evaluation on these scores, it could reasonably generate the correct distribution given that the model can capture the patterns found in the sequence. This will further reduce computational complexity also.

The TVSum dataset specifically is not originally in the form as provided, with the details of this specified in 4.1.1. Because of this, a vocabulary of size 5 could be drawn over the rating system they used for frames before they were converted to frame-level importance scores. This however doesn't coincide with the other benchmark datasets for evaluation.

5.2.3 Bypassing the need for a vocabulary

Another key factor of our implementation that may not be the most appropriate direction for this type of problem came in how we generate the frame-level importance scores at each position. We follow the left-to-right dynamic found in many encoder-decoder schemes to generate the sequence of vocabulary elements as defined in [50]. This loses a key feature found in other state of the art models [11] in the summative, single-step generation process of the output sequence.

Each element in our vocabulary for the VST_{FLIP} refers to a value representing a probability. Intrinsically, two values close together have some inherent connection as they are close in value. This is lost in our implementation as two values are viewed as two completely separate elements with no connection, until learnt. This summative nature that is lacking in our approach is what is commonly found in other similar approaches [11]. To tackle this, an extension to this project could come in the form of proposing an interpretation of the output of the decoder that allows the model to generate values, not probabilities over vocabulary elements. This would be a much further departure from the original Transformer structure we have emulated.

5.2.4 Creating a larger dataset and how it may change evaluation

Mentioned prior in this project, the amount of data available for tackling this problem is limited. This is especially true for a learning problem that is considered "deep". Because of this, consideration for how a larger dataset could be curated is valuable.

An important aspect of curating these video summaries is the human element. We want to capture the implicit human notion of summarising a video. This depends on several factors such as video length and video genre. This would ideally still all be captured in some latent aspect if possible. Because of this, specific aspects of how this project and others have been implemented may need to be reconsidered. For example, utilising the 0-1 knapsack problem over importance scores, as done in our VST_{FLIP} model is effective, but does not provide adequate general summarisation as we must define a preferred output sequence size in relation to the original. In reality, a summarisations length would wholly depend on the content of the video and other preferences made by the observer. Consideration for this when creating summaries would force us to allow an observer, if annotating a video, to choose what they felt was the appropriate length for the final summarisation.

The above highlights an important issue faced when curating video summarisations. By asking people to annotate how important a video is at each moment, or asking them to create a summarisation, passes on some potential bias into the data. However, is one wrong for thinking that there is little other way to collect the annotated ground truth required? I propose a way to curate video summarisations that were created for a completely separate task.

YouTube, and similar other video hosting websites, contain a vast number of videos from all genres. This would immediately lend many to utilise it as a source of videos that can then be annotated for video summarisation, as [3] have done. What I propose however is that it also contains a sufficient number of pairs of videos, a full video and a summarised counterpart. One only needs to look as far as the YouTube front page before being recommended shortened clips from the most popular film of the last four months. Because of this, I believe that you could curate a sufficiently large dataset of video/summarisation pairs to utilise for the problem of video summarisation. This avoids the bias in asking an individual to create a summarisation for you.

A more complex notion may be captured in this approach also. It is very well possible that multiple summarisations may be available for a video on YouTube. Because of this, a comparison – weighted by popularity for example – can be made between these videos to further capture the notion of what provides the best summarisation. It is important to note that a different bias is formed through this collection in the form of the popularity of the content. Videos that also have summarisation will be those videos that were popular enough to warrant some form of summarisation. This would hopefully not be a significant bias as long as the videos chosen are representative of a variety of genres. It is also important to ensure that appropriate non-stylised versions of summarisations are chosen. For example, a trailer for a movie is a summarisation, but not necessarily one we are trying to generate.

5.2.5 Generative genre-based video summarisation

When we have considered what makes a "good" video summarisation, we have often referred to human intuition. But this comes with a large caveat in the form of *what* summarisation that human is attempting to make. If you ask someone to create a trailer for a film they will most likely produce a summarisation that is wildly different to that of one where you ask them for a summarisation that maximises the plot-line or other information. Because of this, video summarisation may be applied more effectively as genre-based.

We can consider the extra information provided alongside videos from websites such as YouTube. Extra information referring to genre or video-tags could be used to further improve the preferences we can apply when creating a summarisation. Specifically in the generative model context, genre preference could be utilised to consider the style of summarisation you would want to create. This may lead to far greater success generally on datasets as the genre of the video can be tailored to. Alone, this could assist in the models usability as a generative tool.

Bibliography

[1] The annotated transformer. Last accessed 10 May 2020.

[2] Open video project. Last accessed 10 May 2020.

[3] Sandra Avila, Ana Lopes, Antonio da Luz, and Arnaldo Araújo. Vsumm: A mechanism designed to produce static video summaries and a novel evaluation method. *Pattern Recognition Letters*, 32:56–68, 01 2011.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *ArXiv*, 1409, 09 2014.

[6] Zuzana Cernekova, Ioannis Pitas, and Christophoros Nikou. Information theory-based shot cut/fade detection and video summarization. *IEEE Trans. Circuits Syst. Video Techn.*, 16:82–91, 01 2006.

[7] Chong-Wah Ngo, Yu-Fei Ma, and Hong-Jiang Zhang. Video summarization and scene detection by graph modeling. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(2):296–305, 2005.

[8] Yang Cong, Junsong Yuan, and Jiebo Luo. Towards scalable summarization of consumer videos via sparse dictionary selection. *IEEE Transactions on Multimedia*, 14:66–75, 02 2012.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei Fei Li. Imagenet: a large-scale hierarchical image database. pages 248–255, 06 2009.

[10] Naveed Ejaz, Irfan Mehmood, and Sung Baik. Efficient visual attention based framework for extracting key frames from videos. *Signal Processing: Image Communication*, 28:34–44, 01 2013.

[11] Jiri Fajtl, Hajar Sadeghi Sokeh, Vasileios Argyriou, Dorothy Monekosso, and Paolo Remagnino. Summarizing videos with attention, 2018.

[12] Michael Fleischman, Brandon Roy, and Deb Roy. Temporal feature induction for baseball highlight classification. pages 333–336, 01 2007.

[13] Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. *Proceedings of the First Workshop on Neural Machine Translation*, 2017.

[14] Cheng-Yang Fu, Joon Lee, Mohit Bansal, and Alexander Berg. Video highlight prediction using audience chat reactions. pages 972–978, 01 2017.

[15] Tsu-Jui Fu, Shao-Heng Tai, and Hwann-Tzong Chen. Attentive and adversarial learning for video summarization. pages 1579–1587, 01 2019.

[16] Boqing Gong, W.-L Chao, K. Grauman, and Fei Sha. Diverse sequential subset selection for supervised video summarization. *Advances in Neural Information Processing Systems*, 3:2069–2077, 01 2014.

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

[18] Michael Gygli, Helmut Grabner, Hayko Riemenschneider, and Luc Van Gool. Creating summaries from user videos. pages 505–520, 09 2014.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. pages 770–778, 06 2016.

[20] Tianxing He, Jingzhao Zhang, Zhiming Zhou, and James Glass. Quantifying exposure bias for neural language generation, 2019.

[21] Xufeng He, Yang Hua, Tao Song, Zongpu Zhang, Zhengui Xue, Ruhui Ma, Neil Robertson, and Haibing Guan. Unsupervised video summarization with attentive conditional generative adversarial networks. pages 2296–2304, 10 2019.

[22] Geoffrey Hinton. Rectified linear units improve restricted boltzmann machines vinod nair. volume 27, pages 807–814, 06 2010.

[23] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[24] Zhong ji, Kailin Xiong, Yanwei Pang, and Xuelong Li. Video summarization with attention-based encoder-decoder networks. *IEEE Transactions on Circuits and Systems for Video Technology*, PP, 08 2017.

[25] Ian Jolliffe. *Principal Component Analysis*, pages 1094–1096. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[26] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. pages 4396–4405, 06 2019.

[27] Hans Kellerer, Ulrich Pferschy, and David Pisinger. Multidimensional knapsack problems. In *Knapsack problems*, pages 235–283. Springer, 2004.

[28] Aditya Khosla, Raffay Hamid, Chih-Jen Lin, and Neel Sundaresan. Large-scale video summarization using web-image priors. pages 2698–2705, 06 2013.

[29] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[30] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. Opennmt: Open-source toolkit for neural machine translation. In *Proc. ACL*, 2017.

[31] Sanjay K. Kuanar, Rameswar Panda, and Ananda S. Chowdhury. Video key frame extraction through dynamic delaunay clustering with a structural constraint. *Journal of Visual Communication and Image Representation*, 24(7):1212 – 1227, 2013.

[32] Alex Kulesza. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2-3):123–286, 2012.

[33] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[34] Yandong Li, Liqiang Wang, Tianbao Yang, and Boqing Gong. *How Local Is the Local Diversity? Reinforcing Sequential Determinantal Point Processes with Dynamic Ground Sets for Supervised Video Summarization: 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VIII*, pages 156–174. 09 2018.

[35] Yen-Ting Liu, Yu-Jhe Li, Fu-En Yang, Shang-Fu Chen, and Yu-Chiang Frank Wang. Learning hierarchical self-attention for video summarization. pages 3377–3381, 09 2019.

[36] Minh-Thang Luong, Hieu Pham, and Christoper Manning. Effective approaches to attention-based neural machine translation. 08 2015.

[37] Yu-Fei Ma, Lie Lu, Hong-Jiang Zhang, and Mingjing Li. A user attention model for video summarization. In *Proceedings of the Tenth ACM International Conference on Multimedia*, MULTIMEDIA '02, page 533–542, New York, NY, USA, 2002. Association for Computing Machinery.

[38] B. Mahasseni, M. Lam, and S. Todorovic. Unsupervised video summarization with adversarial lstm networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2982–2991, 2017.

[39] Shaohui Mei, Genliang Guan, Zhiyong Wang, Shuai Wan, Mingyi He, and David Dagan Feng. Video summarization via minimum sparse reconstruction. *Pattern Recogn.*, 48(2):522–533, February 2015.

[40] Kevin P Murphy. *Machine learning: a probabilistic perspective.* 2012.

[41] Chong-Wah Ngo, Ting-Chuen Pong, and Roland T Chin. Video partitioning by temporal slice coherency. *Circuits and Systems for Video Technology, IEEE Transactions on*, 11:941 – 953, 09 2001.

[42] Mayu Otani, Yuta Nakashima, Esa Rahtu, Janne Heikkilä, and Naokazu Yokoya. Video summarization using deep semantic features. pages 361–377, 03 2017.

[43] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[44] Danila Potapov, Matthijs Douze, Zaid Harchaoui, and Cordelia Schmid. Category-specific video summarization. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *ECCV - European Conference on Computer Vision*, volume 8694 of *Lecture Notes in Computer Science*, pages 540–555, Zurich, Switzerland, September 2014. Springer.

[45] Abigail See. Cs224n neural machine translation lecture 8.

[46] Alan Smeaton, Paul Over, and Wessel Kraaij. Evaluation campaigns and trecvid. *Smeaton, Alan F. and Over, Paul and Kraaij, Wessel (2006) Evaluation campaigns and TRECVid. In: MIR 2006 - 8th ACM SIGMM International Workshop on Multimedia Information Retrieval, 26-27 October 2006, Santa Barbara, CA, USA.*, 01 2006.

[47] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information processing & management*, 45(4):427–437, 2009.

[48] Xinhui Song, Ke Chen, Jie Lei, Li Sun, Zhiyuan Wang, Lei Xie, and Mingli Song. Category driven deep recurrent neural network for video summarization. pages 1–6, 07 2016.

[49] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[50] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 06 2017.

[51] Jesse Vig and Yonatan Belinkov. Analyzing the structure of attention in a transformer language model, 2019.

[52] J. Wu and J. M. Rehg. Centrist: A visual descriptor for scene categorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1489–1501, 2011.

[53] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. 02 2015.

[54] Xun Xu, Timothy M. Hospedales, and Shaogang Gong. Discovery of shared semantic spaces for multiscene video query and summarization. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(6):1353–1367, Jun 2017.

[55] Yale Song, J. Vallmitjana, A. Stent, and A. Jaimes. Tvsum: Summarizing web videos using titles. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5179–5187, 2015.

[56] Ke Zhang, Wei-Lun Chao, Fei Sha, and Kristen Grauman. Video summarization with long short-term memory. volume 9911, pages 766–782, 10 2016.

[57] Ke Zhang, Kristen Grauman, and Fei Sha. Retrospective encoders for video summarization. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 391–408, Cham, 2018. Springer International Publishing.

[58] Yunzuo Zhang, Ran Tao, and Yue Wang. Motion-state-adaptive video summarization via spatio-temporal analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, 27:1–1, 01 2016.

[59] Kaiyang Zhou and Yu Qiao. Deep reinforcement learning for unsupervised video summarization with diversity-representativeness reward. 12 2017.