Master of Science Thesis in Computer Science and Engineering
Department of Electrical Engineering, Linköping University, 2023

Efficient Utilization of Video Embeddings from Video-Language Models

Felix Lindgren

LINKÖPING
UNIVERSITY

Master of Science Thesis in Computer Science and Engineering

Efficient Utilization of Video Embeddings from Video-Language Models:

Felix Lindgren

LiTH-ISY-EX--23/5592--SE

Supervisor: **Emanuel Sanchez Aimar**
ISY, Linköpings University
Johan Karlsson
FOI, Swedish Defence Research Agency

Examiner: **Mårten Wadenbäck**
ISY, Linköpings University

Computer Vision Laboratory
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

In the digital age where video content is abundant, this thesis investigates the efficient adaptation of an existing video-language model (VLM) to new data. The research leverages CLIP, a robust language-vision model, for various video-related tasks including video retrieval. The study explores using pre-trained VLMs to extract video embeddings without the need for extensive retraining. The effectiveness of a smaller model using aggregation is compared with larger models and the application of logistic regression for few-shot learning on video embeddings is examined. The aggregation was done using both non-learning through mean-pooling and also by utilizing a transformer. The video-retrieval models were evaluated on the ActivityNet Captions dataset which contains long videos with dense descriptions while the linear probes were evaluated on ActivityNet200 a video classification dataset.

The study's findings suggest that most models improved when additional frames were employed through aggregation, leading to improved performance. A model trained with fewer frames was able to surpass those trained with two or four times more frames by instead using aggregation. The incorporation of patch dropout and the freezing of embeddings proved advantageous by enhancing performance and conserving training resources. Furthermore, using a linear probe showed that the extracted features were of high quality requiring only 2-4 samples per class to match the zero-shot performance.

Acknowledgments

I wish to extend my thanks to Johan Karlsson, my supervisor at FOI, for his assistance and valuable discussions during the course of my thesis. I also want to acknowledge David Gustafsson for his useful input. Additionally, I'm grateful to my colleagues at FOI for providing good company and fun discussions. I appreciate the guidance and feedback given by my university supervisor, Emanuel Sanchez Aimar, and examiner, Mårten Wadenbäck, which was greatly appreaciated. Lastly, I'd like to acknowledge the constant support from my friends and family throughout this process.

Linköping, Juni 2023
Felix Lindgren

Contents

1 Introduction

Video content is being produced at an overwhelming pace, with more than 500 hours uploaded to YouTube alone every minute[1]. The vastness of this data exceeds the monitoring capacity of any individual or organization. To manage this, machine-learning models such as a Video-Language Model (VLM) can be employed. These models, which find the best match between text and video, are more adaptable than models with fixed classes requiring retraining to detect new objects. This thesis investigates methods for efficiently adapting an existing VLM to new data such as longer videos or videos with more detailed captions.

CLIP [1] is a method for pre-training a vision model and a language model to work together, and pre-trains both at the same time on hundreds of millions of text–image pairs. Upon release, it achieved very competitive results for a range of different datasets without the need for fine-tuning to any specific dataset, also known as zero-shot learning [2]. While CLIP was originally introduced for still images, it has also been extended into a VLM for use in different video tasks such as video question answering [3], video summarization [4], captioning [5] and action classification [6].

Video-language retrieval is a common task where the objective is to identify the relevant video when given a text query, or vice versa. Given CLIP's exceptional ability to measure the similarity between a text and an image, new video models have reused CLIP. Some examples are CLIP4Clip [7], EclipSE [8], X-CLIP [6] and VideoCLIP [9]. These VLMs use different methods to create video representations from outputs generated by the visual transformer that was initiated with CLIP weights.

Similar to how CLIP is trained on a very large dataset of text-image pairs, these CLIP-video models are trained on a large number of text-video pairs. Existing models usually focus on shorter video datasets, like MSVD [10] and MSR-

[1] Statista.com

VTT [11], where the videos are around 10–30 seconds long. Some models also evaluate on Activity-Net Captions [12], which contain longer videos of around 180 seconds long and the captions can contain multiple sentences describing the events.

1.1 Motivation

This work is carried out at the Swedish Defence Research Agency (FOI). They want to investigate how CLIP can be used to create zero- or few-shot "video classifiers". Training VLMs from scratch require lots of data and computational resources, however using a pre-trained model can give a good base for further exploration without training from scratch.

1.2 Aim

The thesis aims to use a pre-trained VLM to extract video embeddings and investigate how they can be efficiently used without completely retraining the model for a specific dataset. First, it explores video retrieval, using a smaller model on multiple parts of a longer video and then fusing these video clip embeddings into a full video embedding, where only the part of the model that combines the smaller video embeddings will need to be trained from scratch. The goal is to see if this approach can improve the performance of a pre-trained model instead of simply using a larger model with more parameters or additional frames. Secondly, inspired by the original CLIP paper [1], where the authors explore few-shot learning using logistic regression to classify the image embeddings, this thesis will investigate if a logistic regression head is also effective on video embeddings. Since logistic regression requires defined classes, it will be evaluated on an action classification dataset instead.

1.3 Research questions

1. How does the performance of a smaller pre-trained model using aggregation compare to a larger model on the ActivityNet captions dataset?

2. Can the proposed aggregation method achieve comparable performance to other VLMs while using less memory and requiring fewer FLOPs?

3. To what extent does unfreezing different layers and patch dropout during VLM fine-tuning affect training time and video retrieval performance?

4. How do features extracted from a pre-trained VLM with a few-shot logistic regression classifier compare to zero-shot classification for action classification?

1.4 Delimitations

The investigation was limited to ActivityNet Captions for video retrieval and ActivityNet200 for action recognition. The only pre-trained VLM evaluated will be microsoft/X-CLIP[2] due to the lack of open-source weights for other VLMs built on CLIP. The fine-tuning was done using four Nvidia RTX 2080Ti'.

[2]`https://huggingface.co/microsoft/xclip-base-patch32`

2 Theory

This chapter provides an overview of relevant theory related to the research conducted in this thesis. Section 2.1 gives a brief overview of machine learning. Section 2.2 introduces neural networks and deep learning. Section 2.3 describes what embeddings are and why they are used. Section 2.4 shows how the transformer architecture works. Section 2.5 describes how the transformer was adapted for vision tasks. Section 2.6 describes the different methods of evaluating the models. Finally, Section 2.7 introduces some related works.

2.1 Machine learning

Machine learning (ML) is a subfield of artificial intelligence focused on developing algorithms capable of learning to make predictions from data. When data is sampled, there is an assumption underlying the distribution of the data. With labeled data samples, a model can be designed to attempt to find the underlying function that describes the distribution [13]. Since exhaustive data collection is unfeasible, the goal is for the model to generalize to new unseen data. To guide the optimization of a model, a so-called loss function is used. This is a mathematical function that describes how well the model is learning. An example of a loss function is contrastive loss, first introduced by Hadsell et al. [14]. It calculates the distance between pairs of inputs in a transformed space, with the goal of pushing similar pairs (positive pairs) closer together and pulling dissimilar pairs (negative pairs) further apart in this embedded space. The contrastive loss is defined as

$$\begin{aligned} \mathcal{L}(W) &= \sum_{i=1}^{P} L(W, (Y, \vec{X_1}, \vec{X_2})^i) \\ L(W, (Y, \vec{X_1}, \vec{X_2})^i) &= (1 - Y) L_S(D_W^i) + Y L_D(D_W^i) \end{aligned} \tag{2.1}$$

where $(Y, \vec{X_1}, \vec{X_2})^i$ represents the i-th of P total pairs of input vectors $\vec{X_1}, \vec{X_2}$, along with a binary label Y that is set to 1 if the inputs are similar and 0 if they are not. D_W is a distance function parameterized by the weights W which measures the distance between the input vectors in the transformed space. L_S and L_D are partial loss functions for similar and dissimilar points, designed such that minimizing the overall contrastive loss will result in decreased distances for similar pairs and increased distances for dissimilar pairs in the transformed space.

Logistic regression

Logistic regression is a common binary classification method that extends linear regression by applying a logistic function to map output to a probability between 0 and 1. To assess the model's performance, the binary cross-entropy loss is used. Model parameters are optimized using gradient-based methods, like stochastic gradient descent, and a threshold is applied to determine class labels.

2.2 Deep learning

An artificial neural network is a type of machine learning model, sometimes referred to as a fully connected neural network or a multi-layer perceptron [15]. A common type of neural network is a multi-layer perceptron made up of multiple layers connected to each other. Other types of neural networks are Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs).

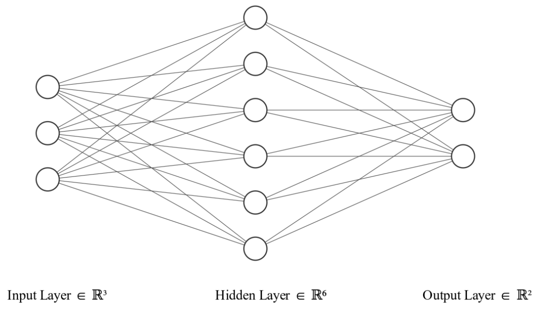

Figure 2.1: *An example multi-layer neural network.*

Between two layers in a fully connected neural network, there are connections from each node in the layer to each node in the next layer and each such connection has a weight associated with it, an example of a basic neural network is shown in Figure 2.1. The weights are arbitrary real values usually between -1 and 1 that are multiplied with the input to the layer. Inputs multiplied with the layer weights are just a linear combination, so in order to model more complex functions the network requires non-linearity. This is introduced by using an activation function after a linear layer. Some common activation functions are the sigmoid and ReLU. The process of finding the optimal weights is how the neural network "learns".

Optimizing through stochastic gradient descent

Neural networks have large parameter spaces, where models can have millions or billions of parameters that need to be tuned. It is not feasible to hand-tune these parameters to minimize the loss function, and instead, stochastic gradient descent (SGD) is used to find the negative gradient of the loss function. To efficiently calculate the gradients, the backpropagation algorithm is used, which was first introduced by Rumelhart et al. [16]. Backpropagation works by applying the chain rule to compute the derivative of the loss with respect to each parameter and does so by working backward from the output. These gradients are used during sgd to update the parameters

$$\theta_{t+1} = \theta_t - \eta \nabla f_i(\theta_t) \tag{2.2}$$

where θ_t are the parameters at timestep t, η is the learning rate, $\nabla f_i(\theta_t)$ is the gradient of the loss function with respect to the parameters on a single training example i at timestep t.

The learning rate is an important hyperparameter when tuning the learning, too low, the loss will take too long to converge. However too large and the optimization steps will jump over the local minima leading to bad learning.

Varying learning rate

There are different methods for deciding the learning rate, and instead of setting it at the beginning, there have been attempts to adjust it during training. For example, cyclically increasing and decreasing the learning rate[17] or starting from a small value, building to the original learning rate, and then decreasing until training is done, effectively "warming up" the learning rate[18].

Other algorithms that are variations of SGD adjust the learning rate during optimization. Examples of these algorithms include, Adam[19], Adagrad[20], and RMSProp[21]. These algorithms have been shown to lead to faster convergence than regular SGD. Adam is especially a common choice for the optimizer.

When using pretrained models, it is important not to use a learning rate that is too large, as it might overwrite the existing weights. One way of avoiding this is by *freezing* some parts of the network which means skipping them during the optimization. At least for vision models, the early layers tend to learn more general representations while later layers specialize more. Freezing different layers can therefore be beneficial when finetuning data is lacking. Since frozen layers are skipped when calculating gradients, it saves memory and computational resources, making it possible to fit larger models into memory. Slowly increasing the learning rate after a certain number of training iterations for some frozen layers, also known as unfreezing[22], has also been shown to be effective in increasing performance.

Overfitting & Dropout

When training deep networks an issue that can occur is overfitting. This is when the model fits too well to the training data and is unable to generalize to new unseen data. There are many ways to regularize models in order to reduce overfitting. One common technique for neural networks is to use dropout, which was first described by Srivastava et al. [23]. It works by setting the output of a number of nodes in a layer to zero, slightly altering the structure of the network. Conceptually this should make the network less reliant on specific nodes and force it to be more generalized. The proportion of nodes that should be dropped is a hyperparameter that should be tuned.

2.3 Embeddings

When working with text data, neural networks cannot accept raw text so it needs to be pre-processed into a format a neural network can use. The dimensionality of text data is very high which makes it challenging for neural networks to process. By reducing the dimensionality, it becomes more manageable for neural networks to process, this can be achieved by using embeddings, which map a token, such as a word, to a vector $V \in \mathbb{R}^n$. Said vector is known as an embedding and the vector space is called the embedding space. The embedding vectors are given random values to start with, but through training their positions can be adjusted to support downstream tasks. The embeddings will vary depending on which task they were trained on. General tasks such as predicting the next word in a sentence have been shown to produce good word embeddings[24].

Another type of high-dimensional data is images, which can also be embedded into a lower-dimensional embedding space. This allows embeddings from different modalities (text and images) to share a joint embedding space. Such a shared space can be trained to allow semantic relations to be learned between different modalities.

2.4 Transformer architecture

The transformer is a neural-network architecture first introduced by Vaswani et al. [25] in their paper *Attention Is All You Need*. The transformer was originally used for machine translation, utilizing an encoder-decoder design where word embeddings would first be encoded before being passed to the decoder to be processed and then output the translated predictions. The attention mechanism lets each token in a sequence look at the other tokens and adjust itself depending on what the other tokens are. In the original paper, this is done through scaled dot product attention. The equation for attention is described as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.3}$$

where Q, K, V are matricies named *Query, Key, Value*. These are vectors obtained by multiplying the input embeddings by a learned weight matrix. The queries represent the information in a token and the keys represent all other tokens that the queries can attend to. When multiplied they generate an attention map that shows how much a given token is attending to the other tokens in the sequence. The attention map is divided by $\sqrt{d_k}$, which is the dimensionality of the key vector, this is done in order to prevent very large values in the softmax from causing small gradients. The attention map will weigh the output values. This makes it possible for all tokens to influence each other.

When the attention map is created there is no information about which position a token has in a sequence. For translation, the order of the words can have a big impact on the meaning of a sentence, this makes it necessary for the model to have some way of keeping track of the position of the tokens. The authors solve this by using *positional encodings*, wherein each input embedding is modified by adding a positional vector representing where in the sequence each token is located. In the original Transformer paper, they use a sinusoidal function as such:

$$\begin{aligned}\mathrm{PE}_{i,2j} &= \sin(i/10000^{2j/d_{\mathrm{model}}}),\\ \mathrm{PE}_{i,2j+1} &= \cos(i/10000^{2j/d_{\mathrm{model}}}),\end{aligned} \tag{2.4}$$

where i is the position index and j is the dimension index. Others [26, 27] have used *position embeddings* that are learned during training. The positional encoding is applied at the same time as tokens are embedded so

$$y_i = E(x_i) + PE(i) \tag{2.5}$$

where x_i is the i-th token in the sequence, E is the token embedding and PE is the positional encoding, which could be a function as shown in 2.4 or an embedding.

The original Transformer paper describes two kinds of attention, self-attention, and cross-attention. Self-attention is when each token in a sequence attends to all other tokens in the same sequence. The queries, keys, and values all come from the same sequence. Cross-attention on the other hand is where each token in sequence A attends to all tokens in sequence B. This is done using the same method as described in equation 2.3, but the keys and values will come from the new sequence and the queries are from the original.

The authors also found that instead of using a single attention block if they linearly project the queries, keys, and values using multiple different learned projections the model performed better. This is known as multi-head attention where the attention function is used on each of the new queries, keys, and values in parallel. The results are concatenated and then projected back to the dimension of the original values. Having multiple heads is supposed to let each head attend varying amounts to different parts of the sequence. The entire process can be described as

$$\begin{aligned}\mathrm{MultiHead}(Q, K, V) &= \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_h)W^O\\ \text{where } \mathrm{head}_i &= \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned} \tag{2.6}$$

where W_i are projection matrices that are learned during training.

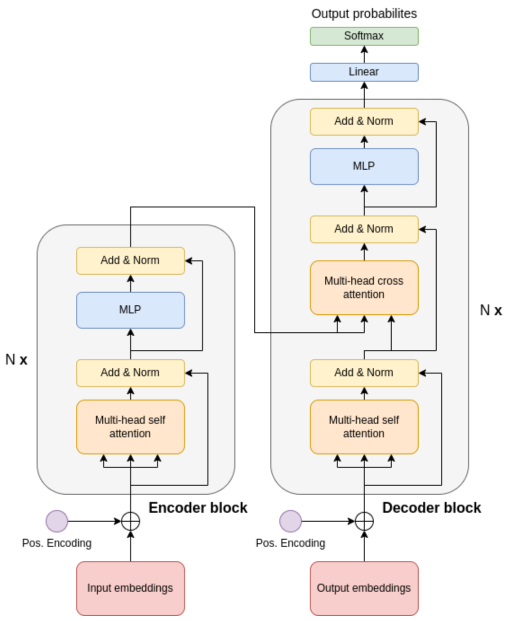

Figure 2.2: *Example of encoder-decoder blocks used in the original Transformer architecture.*

2.5 Vision transformer

Inspired by various successes achieved using Transformers for Natural Language Processing (NLP) tasks [26–28], especially the benefits gained from scaling up the models, Dosovitskiy et al. [29] attempted to recreate the same results using a Vision Transformer (ViT). Since transformers take all tokens into account when calculating the attention, they can process an entire image at once. This could be an advantage compared to CNNs which are more locally focused. While it is technically possible to consider each pixel in an image as a token and let all pixels attend to each other, this would be computationally unfeasible, especially since the computational complexity of the scaled dot-product attention grows quadratically with the number of tokens. To overcome this challenge, the authors divide the image into a grid of patches, which are then embedded. In order to identify the location of each patch embedding positional embeddings are also added. In addition, they also prepend a learnable embedding to the start of the sequence of patches, this token will be used when doing the final class prediction. These embeddings are fed into an encoder which is a slightly modified version of the encoder block described in Vaswani et al. [25].

There are two main ways of scaling up the vision transformer, by either increasing the number of parameters or by shrinking the patch size. Increasing the model size involves adding more encoding blocks and making the embedding dimensions and hidden layers bigger. When reducing the patch size, the number of patches will increase making the sequence longer which means a larger attention map requiring more memory and added computations at each stage of the transformer. The vision transformers are often named based on the patch size, often 14, 16, or 32 pixels, and the number of parameters used, usually named Small, Base, or Large. These are then shortened to B/16 or L/14 for example when referring to specific ViT configurations.

2.6 Evaluation

Video retrieval evaluation

Video retrieval, as defined in similar works [7–9], is the task of matching a video with a sentence or vice-versa. The standard video retrieval metric in the literature is Recall@K (R@K) and median rank (medR), which are used for both video-text and text-video retrieval. R@K measures if the correct match was predicted within the top K. The median rank measures the median position of where the correct match was ranked. The datasets commonly used for video retrieval are shown in table 2.1 below. If a video contains multiple sentences they are sometimes concatenated into a single paragraph, which is why video retrieval on those datasets is sometimes called paragraph-to-video retrieval.

Table 2.1: *Common video retrieval datasets.*

Dataset	Avg. video length(seconds)	Text length
MSR-VTT[11]	20	Sentence
MSVD [10]	20	Sentence
DiDeMo [30]	30	Paragraph
ActivityNet Captions [12]	180	Paragraph

Action classification

Action classification is a computer vision task that involves identifying and categorizing specific human actions or activities within video sequences. The objective is to classify the action or activity being performed in the video. Examples of such actions include walking, running, jumping, or more complex activities like cooking, playing sports, or dancing.

The evaluation of action classification models typically involves using metrics such as R@K (Recall@K), which measures if the true class is within the top-1, top-5, or top-k of the predicted results. A dataset for action classification is ActivityNet200 (AN200), which contains 200 classes of human activities. AN200 shares the same video content as the ActivityNet Captions (ANC) dataset but uses a separate set of annotations specifically for action classification.

2.7 Related work

2.7.1 CLIP

Contrastive Language-Image Pre-Training (CLIP) was introduced by Radford et al. [1] in 2021. Machine learning models usually require data to be labeled in a specific set of classes and are then only trained to make predictions on those classes. This can be limiting because when a new class is needed, there must exist labeled data and a model must be trained. CLIP uses a ViT and sentence Transformer to extract embeddings from images and text, respectively. These embeddings are then compared using cosine similarity to get the get similarity score between an image and text. CLIP differs from classifiers trained on specific class sets, which can only make accurate predictions for samples within the defined set of classes. By directly comparing text and image representations, CLIP avoids being limited to an initial set of classes. This makes it a powerful general-purpose classifier since it can be used for a wide variety of data without the need for finetuning. However, to learn these connections between natural text and images the model requires lots of pre-training and large amounts of data. For example, CLIP was trained on 400M text-image pairs.

To test the quality of the image features generated by the CLIP vision transformer, the authors use a "linear probe" for few-shot learning. They train a multi-class logistic regression model on several different datasets using the features generated by the ViT. They do this by using a small number of samples from each class to see how good the model is for few-shot learning. This simple linear classifier using extracted features is also called a linear probe.

2.7.2 Video language models

Video language models (VLM) are a relatively new fusion of existing video and text models to create a combined model that learns the relationship between videos and text. One of the first examples of a VLM is VideoBERT, which was introduced by Sun et al. [31]. VideoBERT converts frames into embeddings that are passed to the model together with the corresponding text. In this approach, a single transformer is used that takes in both the video frames as well as the text before using it in a downstream task, such as action classification or video captioning. Another approach is to use separate encoders for text and video similar to CLIP, an early example of a VLM using this approach is Xu et al. [32].

CLIP4Clip

In their paper Luo et al. [7] introduce a VLM that uses CLIP as a base for the encoding of the text and frames. They sample one frame per second from a video, and each frame is encoded using a ViT with weights initiated from the CLIP ViT. These frames are then merged using different methods to generate a video embedding that can be compared with the text embedding from the text encoder. The methods that were tried included mean-pooling, using an LSTM and transformer to generate a merged video embedding, and an approach where a Transformer takes in the text embedding and the frame embeddings to generate a score directly. When released, their method achieved state-of-the-art results for video retrieval and showed that the mean-pooling and using a Transformer were the best methods of combining the frames, achieving similar results.

EclipSE

Building on the architecture developed in CLIP4Clip[7], Lin et al. [8] proposed a model that accepts audiovisual input instead of just video. They use a pre-trained audio encoder to extract embeddings from audio clips sampled around each frame. These audio embeddings are then combined with the image embeddings in an audiovisual attention block using cross-attention. The class embedding from the encoder is combined using a mean pool to obtain the embedding which is compared to the text embedding. They find that when adding audio they can achieve similar or better results than CLIP4Clip while using fewer frames as input. Encoding the audio clips is computationally cheaper than computing the video frames, leading to a reduction in memory usage and FLOPs.

X-CLIP

In their paper, Ni et al. [6] proposed a video classification model that also utilizes pre-trained weights from CLIP. Similar to CLIP4Clip they employ contrastive learning on text embeddings and video embeddings that are generated from aggregating frames from the video. They add a module to the encoder block after the feedforward that allows for information to pass between the frames to integrate the temporal information that videos contain.

2.7.3 Patch dropout

Visual transformers can be very expensive to train, especially when scaling up the resolution or reducing the patch size. To address this issue Liu et al. [33] proposes *Patch Dropout* to reduce the resource requirements for training or fine-tuning a ViT. This technique works by randomly removing a percentage of the patches after the positional embedding has been added to each patch embedding. The authors primarily tested their approach on CSAW[34], a dataset containing high-resolution medical images, and showed a gain in accuracy and reduction in memory and FLOPs when keeping between 25-50% of the original patches. For other datasets such as CIFAR100[35] and ImageNet[36] they show that there are small accuracy gains when keeping around 70-80% of patches. Even when keeping only 50% of the patches, the reduction in accuracy was only around one percent while halving the FLOPs and memory used. Additionally, the authors found that patch dropout also has a regularizing effect on the model, reducing overfitting.

3 Method

This chapter presents the steps taken to prepare for the experiments and how they were conducted. Section 3.1 describes the setup needed to answer the research questions such as the dataset, preprocessing, baselines, and model fine-tuning. In Section 3.2, the various experiments conducted in this thesis are described. These include the exploration of different aggregation approaches, measurement of training resources, an ablation study of hyperparameters, and linear probing of different models.

3.1 Experiment setup

This section outlines the different steps taken to prepare for the experiments conducted in this thesis. Here the dataset and preprocessing is introduced, which model was used as a baseline, the proposed aggregation models, and how the models were fine-tuned.

3.1.1 Dataset

In order to measure video-paragraph retrieval, the ActivityNet Captions [12] (ANC) dataset was utilized. This dataset contains 10,000 training videos, with an average length of 180 seconds per video with the longest being 10 minutes. Each video in the dataset is accompanied by a caption, typically comprising three sentences averaging 50 words and describing the events of the video. Figure 3.1 illustrates the structure of a sample in the dataset. Following the methodology of previous works such as [7, 8], the validation set containing around 4900 videos, known as *val1*, was used for evaluation due to the lack of labels for the test set. To facilitate the selection of models and hyperparameters, 10% of the training data, approximately 1000 randomly selected videos, served as a validation set.

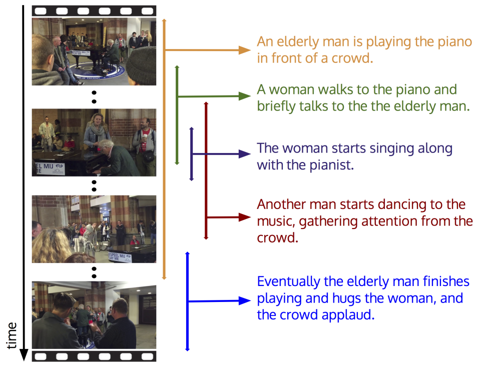
An elderly man is playing the piano in front of a crowd.
A woman walks to the piano and briefly talks to the the elderly man.
The woman starts singing along with the pianist.
Another man starts dancing to the music, gathering attention from the crowd.
Eventually the elderly man finishes playing and hugs the woman, and the crowd applaud.
time

Figure 3.1: *Exerpt from the ANC dataset, showing a sample video and the dense captions. Figure from original ANC paper. Copyright © 2017, IEEE.*

The ANC dataset is a relabeling of the ActivityNet200 [37] (AN200) dataset, which is instead used for action classification of an open set of tasks like *Braiding hair*, *Changing car wheel* or *Canoeing*. These labels are used to evaluate action classification using a linear probe.

3.1.2 Preprocessing

In order to avoid video decoding during training, 64 frames are uniformly sampled from each video and stored on the hard drive. Prior to saving, these images are resized to 224×224 pixels to avoid resizing during training. Smaller models that use 8 or 16 frames can then sample multiple clips from those 64 frames.

The sentences are converted to their corresponding token ids in the CLIP byte-encoding vocabulary [1]. With 50 words on average per caption, this gives an average of 58 tokens after being processed by the tokenizer. The first 72 tokens of each paragraph are included and passed on to the text encoder due to the 77-token limitation of the X-CLIP text encoder.

3.1.3 Baselines

The pre-trained baseline model is X-CLIP [6]. The pre-training dataset is Kinetics 400 [38] a large-scale video dataset for action recognition. All model variations used were downloaded from HuggingFace [1]. All model training and evaluation were implemented in PyTorch. All variations use the same text encoder while differing in the patch size and the number of frames included in the ViT. All models are evaluated on the previously mentioned *val1* split of ANC for video-to-paragraph retrieval, using R@1, R@5, R@10, and the median rank as performance metrics. Table 3.1 presents the different configurations of the baseline models.

[1]`https://huggingface.co/microsoft/xclip-base-patch32`

Table 3.1: Different model variations tested.

Model	Input frames	No. Parameters (M)
XCLIP-Base/32	8/16	196.59
XCLIP-Base/16	8/16/32	194.93

3.1.4 Fine-tuned models

All baseline models were fine-tuned on the ANC training data. Each baseline model was fine-tuned 3 times with randomly shuffled data so that the standard deviations could be reported. To maximize the batch size and optimize training efficiency, the model was split on multiple GPUs during training. The X-CLIP encoders consist of 12 blocks each, and 3 blocks are placed on each GPU. Since X-CLIP has been pre-trained on video-text pairs similar to [7] the fine-tuning is limited to 5 epochs. Consistent with previous works [7, 8] and the original CLIP paper [1], a cosine one-cycle learning rate schedule was used to adjust the learning during training. Table 3.2 displays the training batch size utilized for the various models, all of which were the maximum sizes that could fit into memory. Maximizing the batch size was done since previous work [39] has found that a larger batch size increases performance when using contrastive loss. The hyperparameters used during the fine-tuning process are detailed in Table 3.3. These hyperparameters were selected based on the ablation testing, as described in Section 3.2.3.

Table 3.2: Batch size used for each model with the different patch sizes and the number of frames.

Model	B/16-32f	B/16-16f	B/16-8f	B/32-16f	B/32-8f
Batch size	16	32	64	128	256

Table 3.3: Hyperparameters used when fine-tuning the pre-trained X-CLIP models.

Hyperparameters	Value
Num. Epochs	5
Learning rate	1e−5
Weight decay	0.01
Optimizer	AdamW
Patch dropout	0.5
Layers frozen	Embeddings

3.1.5 Proposed model

The pre-trained models and fine-tuned models were used to extract embeddings from the videos. These embeddings were then aggregated to form a final video embedding. This final video embedding can then be compared to the corresponding text embedding. This process is displayed in Figure 3.2. The three methods of aggregation that are investigated are mean pooling, mean pooling with attention, and attention with a class embedding. An overview of each aggregation method is shown in figure 3.3. This gives four types of models that are tested: the baselines, baseline with aggregation, fine-tuned models, and fine-tuning with aggregation. To train the aggregator, one of the baseline or fine-tuned models was utilized to extract the embeddings for the specified number of clips (1–4), effectively freezing the extractor. Subsequently, using these extracted embeddings, the aggregator was trained with the hyperparameters as listed in Table 3.4.

Table 3.4: *Hyperparameters used when training the aggregators on X-CLIP embeddings.*

Hyperparameters	Value
Num. Epochs	5
Learning rate	1e–3
Weight decay	0.01
Optimizer	AdamW
Batch size	256

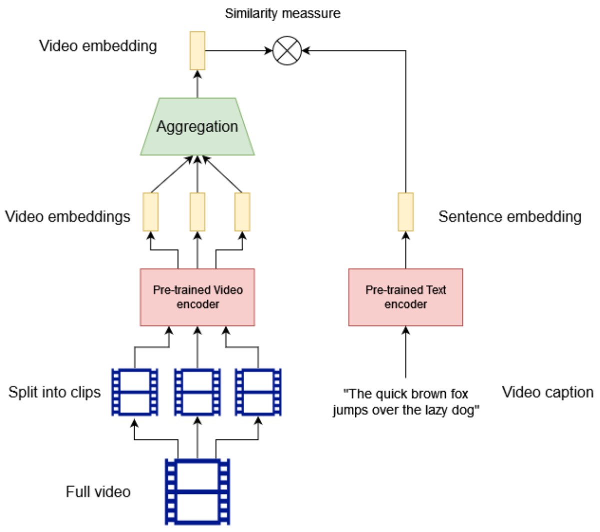

Figure 3.2: *Model for video retrieval using a pre-trained video encoder, red is pre-trained, green is trainable.*

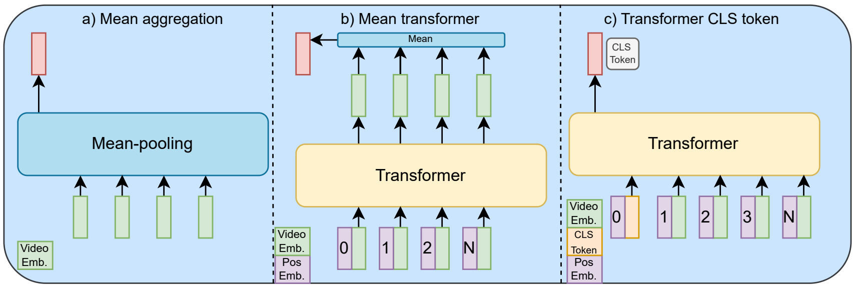

Figure 3.3: *Different aggregation methods. Green embeddings are extracted from an XCLIP model. Purple blocks are learnable positional embeddings. The orange block is a learnable class embedding. The mean pooling has no learnable parameters.*

3.2 Experiments

This section outlines the different experiments performed to answer the stated research questions. These include testing the different aggregation methods, measuring resource usage during training, an ablation study of hyperparameters, and investigating embedding quality using a linear probe.

3.2.1 Aggregation models

The pretrained X-CLIP models were fine-tuned on the training split of ANC. The learnable aggregators were trained 5 times using randomly initialized weights for each model and clip configuration.

Mean-pooling

The most straightforward aggregation approach involved computing the mean of all video embeddings and consolidating them into a single video embedding. This procedure was applied across all models for the baseline and fine-tuned variants.

Transformer mean

This approach used a transformer encoder block and positional embeddings in order to tune the video embeddings before combining them by taking the mean of all embeddings. Models from the baseline, as well as those that had been fine-tuned, were used to train this aggregator. The fine-tuned model chosen was the one with the highest R@1 on the validation data.

Transformer class embedding

This aggregation method introduced a trainable "class" embedding to the sequence of video embeddings. Positional embeddings were added to the sequence before being passed to the transformer. After having attended to the other video embeddings, the class embedding was then utilized as the aggregated video embedding.

3.2.2 Resource usage

To evaluate whether the proposed method successfully reduced the required training resources, measurements were taken for memory usage and training computation during both the training of the aggregators and the fine-tuning of the models.

Memory usage

The memory consumption of all models was evaluated using the PyTorch CUDA package, which measured GPU memory utilization. The memory was measured specifically for the model of the forward and backward passes of a single batch, set at a size of 6.

Computational requirements

As a proxy for computing power, the number of FLOPs was measured for the training and fine-tuning of the models. This was done using the fvcore[2] library which estimates the number of FLOPs in the model forwards pass. Following previous works [40], it is commonly estimated that the computational workload of the backward pass is approximately twice the size of the forward pass. The only library that was found that also computed the backward pass, DeepSpeed[3], required major changes to the code that was not possible due to time constraints.

3.2.3 Ablation testing

This section presents an overview of the various experiments conducted on different hyperparameters. To avoid biasing the evaluation of the final test data, the hyperparameters are tuned on the validation data.

Batch size

Since the contrastive loss is dependent on the batch size, a range of different batch sizes was used when fine-tuning the base models and training the aggregation model. Each model was tested with increasing batch sizes until reaching the largest batch size that could fit within the available GPU memory.

[2]https://github.com/facebookresearch/fvcore/tree/main
[3]https://github.com/microsoft/DeepSpeed

Learning rate and optimizer

A small sweep of learning rates was done for both AdamW and SGD in order to see which optimizer performed best. The values chosen for the sweep are shown in table 3.5

Table 3.5: *Learning rate values for the optimizers.*

Optimizer	Tested values
SGD	[1e-2, 3e-3, 1e-3,1e-4, 3e-5]
AdamW	[1e-4, 3e-5, 1e-5, 3e-6, 1e-6]

Layer freezing

To investigate the effects of freezing layers in the model, it was trained multiple times while varying how much was frozen. Specific configurations were investigated, where only the embedding layers were initially frozen. Then, additionally freezing 3, 6, 9, and, 12 encoding blocks within the models. The memory usage is measured together with the wall clock time of training since there were no profilers available to measure FLOPs of the backward pass.

Patch dropout

To assess the effects of patch dropout on model performance as well as memory usage and computation requirements, different amounts of patch dropout were tested.

Table 3.6: *Patch dropout keep rate values tested.*

Keep rate	[0.05, 0.25, 0.5, 0.75, 1.0]

3.2.4 Linear probe

In order to test the quality of different video embeddings, a linear probe was performed on the embeddings, following the original CLIP paper. A logistic regression classifier is trained using $1, 2, 4, 8, 16$ training samples from each class using the video embeddings as input features. For each model, the best value for the regularization parameter C was found by testing 30 evenly spaced values between $[0.001, 1000]$ on the validation data. These results are compared with the zero-shot response from the base model where the text input to the text encoder is the class label.

4 Results

This chapter presents the results of baseline models, the different aggregation approaches, the hyperparameter tuning, and the linear probe.

4.1 Baselines

The evaluation of the baseline XCLIP models is conducted on the *val1* of ANC as outlined in 3.1.3. The results are presented in Table 4.1, which displays the different models and their Recall at K (see section 2.6 for definition). The best baseline model was the X-CLIP B/16 model trained using 32 frames, performing the best in all metrics.

Table 4.1: *R@K results for the baseline models. R@K (↑) and median rank (↓) are reported. The best results are marked in blue.*

Model	Frames	medR	R@1 (%)	R@5 (%)	R@10 (%)
B/16-8-frames	8	12	13.20	33.68	47.73
B/16-16-frames	16	11	14.33	35.72	49.01
B/16-32-frames	32	***8***	***19.09***	***42.48***	***56.63***
B/32-8-frames	8	12	12.01	32.53	46.30
B/32-16-frames	16	11	13.12	34.31	49.32

4.2 Fine-tuning

This section presents the results obtained after fine-tuning the X-CLIP models on the ANC dataset. As outlined in 3.1.4, this was only done for the Base size models with a patch size of 32 and 16. Among the tested models, the B/16-32 frame model once again performed the best. These results are presented in Table 4.2

Table 4.2: *R@K and median rank results for the fine-tuned models. R@K (↑) and median rank (↓) are reported. The best results are marked in blue.*

Model	medR	R@1 (%)	R@5 (%)	R@10 (%)
B/16-8-frames	3.0 ± 0.0	34.86 ± 0.36	66.58 ± 0.11	80.76 ± 0.37
B/16-16-frames	***2.7 ± 0.6***	36.25 ± 0.53	***68.71 ± 0.88***	***81.99 ± 0.49***
B/16-32-frames	***2.7 ± 0.6***	***36.71 ± 0.71***	68.11 ± 0.48	81.14 ± 0.38
B/32-8-frames	3.0 ± 0.0	32.48 ± 0.10	64.50 ± 0.22	78.55 ± 0.22
B/32-16-frames	3.0 ± 0.0	35.27 ± 0.43	68.11 ± 0.34	81.69 ± 0.25

4.3 Aggregation

This section presents the results for the different aggregation methods outlined in section 3.1.5. It includes the aggregation of embeddings from the baseline models and the fine-tuned models using mean-pooling, transformer mean, and the transformer class token.

4.3.1 Mean aggregation

The results on mean aggregation, as outlined in Section 3.2.1, are presented here for both the baseline models and the fine-tuned ones.

Baseline

Table 4.3 shows the results obtained when using the mean aggregation on the baseline models. The B/16-32 frame model performed the best in all metrics. Figure 4.1 displays the R@1 when using a different number of frames for aggregation.

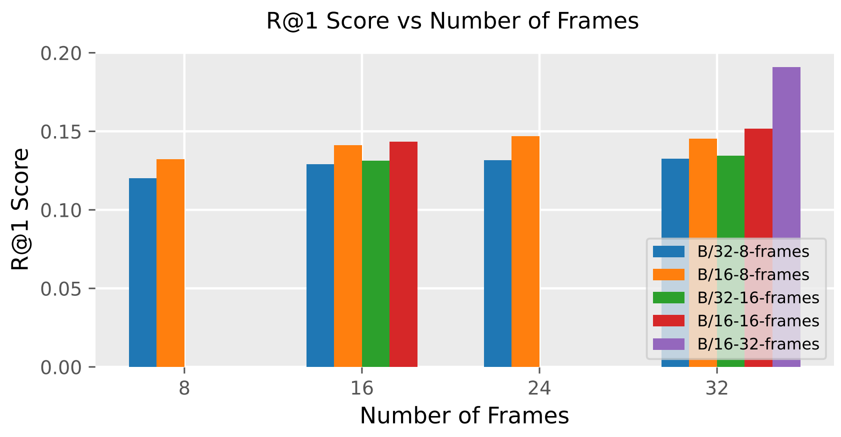

Figure 4.1: *Aggregation using baseline models.*

Table 4.3: *R@K results for the mean aggregation using the baseline models. R@K (↑) and median rank (↓) are reported. The best results are marked in blue.*

Model	Frames	medR	R@1 (%)	R@5 (%)	R@10 (%)
B/16-32-frames	32	***8***	***19.09***	***42.48***	***56.63***
B/16-16-frames	16	11	14.33	35.72	49.01
	32	10	15.16	35.87	50.36
B/16-8-frames	8	12	13.20	33.68	47.73
	16	10	14.12	35.83	50.49
	24	10	14.68	36.66	50.84
	32	10	14.53	37.11	51.64
B/32-16-frames	16	11	13.12	34.31	49.32
	32	11	13.42	35.53	49.95
B/32-8-frames	8	12	12.01	32.53	46.30
	16	11	12.90	34.79	48.99
	24	11	13.16	35.27	49.88
	32	11	13.25	35.90	49.77

Finetune

Table 4.4 shows the results of the fine-tuned models using mean aggregation for different numbers of frames. Figure 4.2 shows the R@1 for the number of frames used during aggregation. Except for the B/32-8 frame mode, all other models performed better with more frames. The B/16-8 frame model performed the best in all metrics.

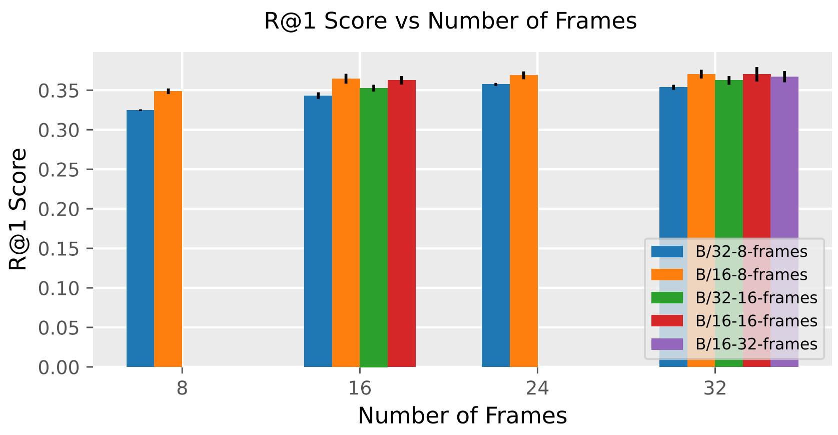

Figure 4.2: *Aggregation using mean with fine-tuned models.*

Table 4.4: *R@K results for the mean aggregation using the baseline models. R@K (↑) and median rank (↓) are reported. The best results are marked in blue.*

Model	Frames	medR	R@1 (%)	R@5 (%)	R@10 (%)
B/16-32-frames	32	2.7 ± 0.6	36.71 ± 0.71	68.11 ± 0.48	81.14 ± 0.38
B/16-16-frames	16	2.7 ± 0.6	36.25 ± 0.53	68.71 ± 0.88	81.99 ± 0.49
	32	***2.3 ± 0.6***	37.01 ± 0.91	69.03 ± 0.32	82.13 ± 0.38
B/16-8-frames	8	3.0 ± 0.0	34.86 ± 0.36	66.58 ± 0.11	80.76 ± 0.37
	16	***2.3 ± 0.6***	36.46 ± 0.63	69.16 ± 0.07	82.57 ± 0.10
	24	***2.3 ± 0.6***	36.88 ± 0.50	***69.28 ± 0.18***	***82.87 ± 0.28***
	32	***2.3 ± 0.6***	***37.03 ± 0.55***	69.07 ± 0.26	82.83 ± 0.16
B/32-16-frames	16	3.0 ± 0.0	35.27 ± 0.43	68.11 ± 0.34	81.69 ± 0.25
	32	3.0 ± 0.0	36.24 ± 0.55	68.85 ± 0.31	82.37 ± 0.21
B/32-8-frames	8	3.0 ± 0.0	32.48 ± 0.10	64.50 ± 0.22	78.55 ± 0.22
	16	3.0 ± 0.0	34.31 ± 0.42	66.97 ± 0.13	81.19 ± 0.25
	24	3.0 ± 0.0	35.73 ± 0.19	67.93 ± 0.05	81.66 ± 0.53
	32	3.0 ± 0.0	35.36 ± 0.32	67.90 ± 0.25	81.70 ± 0.26

4.3.2 Transformer mean

Here the results for the transformer mean aggregation using the baseline and fine-tuned models are shown. The experiments are carried out as described in Section 3.2.1.

Baseline

In table 4.5 the results of the token mean using the baseline models are displayed. Figure 4.3 also shows the R@1 given the number of frames used for the aggregation. Here the aggregation of 4 clips from the B/16-8 frame model performed the best.

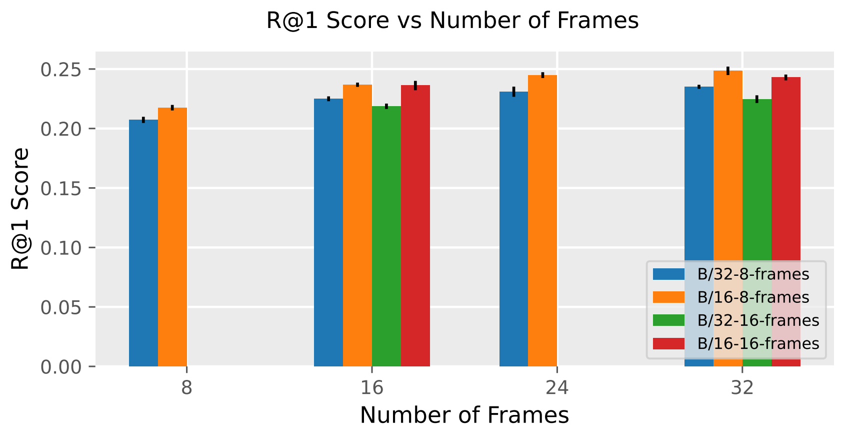

Figure 4.3: *Aggregation using transformer mean with baseline models.*

Table 4.5: *R@K results for the transformer token mean aggregation using the baseline models. R@K (↑) and median rank (↓) are reported. The best results are marked in blue. The difference between this method and the regular mean method is shown in parentheses.*

Model	Frames	medR	R@1 (%)	R@5 (%)	R@10 (%)
B/16-16-frames	16	5.0 ± 0.0	23.61 ± 0.39 (+9.28)	53.51 ± 0.46 (+17.79)	70.19 ± 0.11 (+21.18)
	32	5.0 ± 0.0	24.29 ± 0.25 (+9.13)	54.72 ± 0.25 (+18.84)	70.52 ± 0.29 (+20.17)
B/16-8-frames	8	5.0 ± 0.0	21.75 ± 0.23 (+8.55)	50.42 ± 0.37 (+16.74)	66.92 ± 0.20 (+19.19)
	16	5.0 ± 0.0	23.68 ± 0.18 (+9.56)	53.52 ± 0.15 (+17.69)	70.02 ± 0.42 (+19.54)
	24	4.8 ± 0.4	24.49 ± 0.25 (+9.81)	54.73 ± 0.42 (+18.08)	70.81 ± 0.31 (+19.97)
	32	***4.0 ± 0.0***	***24.85 ± 0.36 (+10.32)***	***55.20 ± 0.37 (+18.09)***	***71.47 ± 0.28 (+19.83)***
B/32-16-frames	16	5.0 ± 0.0	21.87 ± 0.23 (+8.75)	50.64 ± 0.27 (+16.33)	67.34 ± 0.26 (+18.03)
	32	5.0 ± 0.0	22.47 ± 0.32 (+9.05)	53.12 ± 0.30 (+17.59)	69.42 ± 0.39 (+19.48)
B/32-8-frames	8	6.0 ± 0.0	20.73 ± 0.26 (+8.72)	48.47 ± 0.37 (+15.94)	64.47 ± 0.39 (+18.17)
	16	5.0 ± 0.0	22.50 ± 0.20 (+9.60)	51.39 ± 0.29 (+16.60)	67.31 ± 0.12 (+18.32)
	24	5.0 ± 0.0	23.09 ± 0.43 (+9.93)	52.60 ± 0.13 (+17.33)	68.43 ± 0.28 (+18.55)
	32	5.0 ± 0.0	23.50 ± 0.17 (+10.26)	53.39 ± 0.19 (+17.49)	69.38 ± 0.19 (+19.61)

Finetune

Table 4.6 below shows the results for the aggregation using the embedding mean of the aggregation transformer. Figure 4.4 shows a plot of the R@1 with the number of frames aggregated. As indicated in the table the B/16-8 frame model performed the best, surpassing even the best-performing model that used regular mean aggregation.

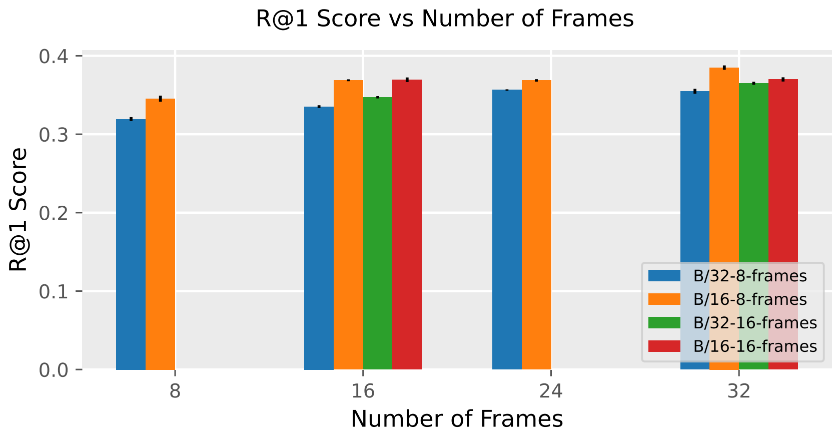

Figure 4.4: *Aggregation using transformer mean with fine-tuned models.*

Table 4.6: *R@K results for the transformer token mean aggregation using the fine-tuned models. R@K (↑) and median rank (↓) are reported. The best results are marked in blue. The difference between this method and the regular mean method is shown in parentheses.*

Model	Frames	medR	R@1 (%)	R@5 (%)	R@10 (%)
B/16-16-frames	16	***2.0 ± 0.0***	36.95 ± 0.29 (+0.70)	69.97 ± 0.21 (+1.26)	82.27 ± 0.25 (+0.28)
	32	***2.0 ± 0.0***	36.99 ± 0.27 (−0.02)	70.32 ± 0.18 (+1.29)	82.89 ± 0.29 (+0.76)
B/16-8-frames	8	3.0 ± 0.0	34.53 ± 0.39 (−0.33)	66.82 ± 0.31 (+0.24)	80.66 ± 0.28 (−0.10)
	16	2.4 ± 0.5	36.88 ± 0.12 (+0.42)	69.02 ± 0.37 (−0.14)	82.48 ± 0.35 (−0.09)
	24	***2.0 ± 0.0***	36.86 ± 0.16 (−0.02)	69.71 ± 0.39 (+0.43)	83.27 ± 0.19 (+0.40)
	32	***2.0 ± 0.0***	***38.50 ± 0.28 (+1.47)***	***70.49 ± 0.15 (+1.42)***	***83.50 ± 0.18 (+0.67)***
B/32-16-frames	16	3.0 ± 0.0	34.71 ± 0.15 (−0.55)	68.53 ± 0.45 (+0.41)	81.72 ± 0.35 (+0.04)
	32	3.0 ± 0.0	36.50 ± 0.19 (+0.26)	68.73 ± 0.14 (−0.11)	82.45 ± 0.25 (+0.08)
B/32-8-frames	8	3.0 ± 0.0	31.93 ± 0.25 (−0.54)	64.93 ± 0.25 (+0.43)	78.53 ± 0.12 (−0.02)
	16	3.0 ± 0.0	33.52 ± 0.18 (−0.79)	67.21 ± 0.36 (+0.24)	81.04 ± 0.29 (−0.16)
	24	3.0 ± 0.0	35.64 ± 0.09 (−0.09)	68.62 ± 0.23 (+0.68)	81.58 ± 0.08 (−0.08)
	32	3.0 ± 0.0	35.47 ± 0.32 (+0.11)	68.50 ± 0.33 (+0.60)	82.15 ± 0.15 (+0.45)

4.3.3 Class token

In this section, the results of the aggregation using a class embedding, as described in 3.2.1, are presented. However, due to its low-performance relative to the transformer mean, this method was only tested with the baseline models.

Baseline

In this section, the results for the transformer aggregation using a class token are presented. In Figure 4.5 a plot of the R@1 score is displayed that shows how performance R@1 changed with the number of frames used. Additionally Table 4.7 displays the full results for all models. This method had the most variance in the results, which was one of the contributing factors in deciding not to proceed further with it. The B/16-16 frame model performed the best using this aggregation method.

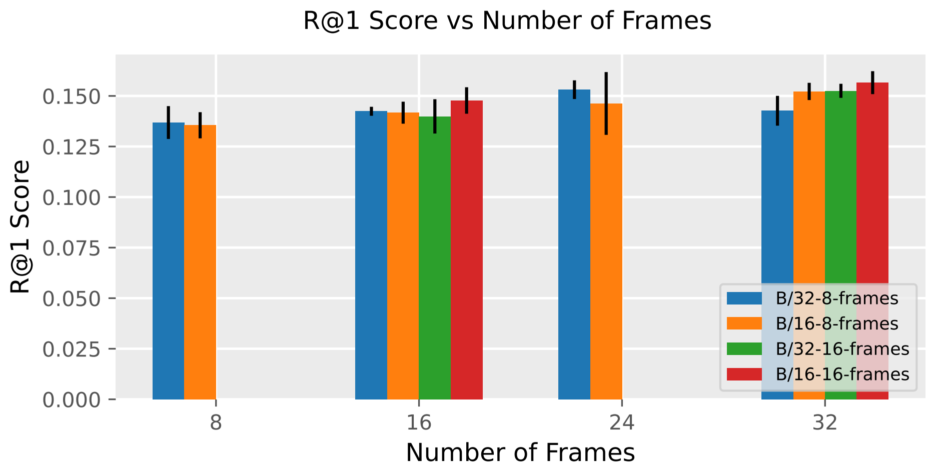

Figure 4.5: *R@1 test score when using class embedding aggregation with the baseline models.*

Table 4.7: *R@K results for the class embeddings aggregation using the baseline model features.*

Model	Frames	medR	R@1 (%)	R@5 (%)	R@10 (%)
B/16-16-frames	16	7.2 ± 0.4	14.78 ± 0.65	42.38 ± 1.12	59.59 ± 0.99
	32	***7.0 ± 0.0***	***15.66 ± 0.57***	***43.24 ± 0.87***	***60.22 ± 0.78***
B/16-8-frames	8	8.4 ± 0.5	13.55 ± 0.65	39.27 ± 0.98	56.47 ± 1.27
	16	7.4 ± 0.5	14.17 ± 0.54	41.16 ± 0.76	58.67 ± 1.34
	24	7.6 ± 0.9	14.63 ± 1.55	41.35 ± 3.28	58.60 ± 2.86
	32	7.2 ± 0.4	15.22 ± 0.43	42.83 ± 0.67	59.64 ± 0.82
B/32-16-frames	16	8.4 ± 0.5	13.99 ± 0.85	39.87 ± 1.13	56.26 ± 1.42
	32	7.4 ± 0.5	15.25 ± 0.35	41.99 ± 0.53	59.21 ± 1.14
B/32-8-frames	8	9.2 ± 0.4	13.69 ± 0.81	38.60 ± 1.33	54.28 ± 1.21
	16	8.0 ± 0.0	14.24 ± 0.22	40.51 ± 0.61	57.21 ± 0.42
	24	8.0 ± 0.0	15.31 ± 0.46	41.43 ± 0.53	58.03 ± 0.44
	32	8.2 ± 0.8	14.27 ± 0.74	40.08 ± 1.34	56.74 ± 1.39

4.4 Model resource usage

This section details the findings of the experiments on training resources, as outlined in Section 3.2.2. These experiments involved measuring the memory usage and computational requirements during the training process.

4.4.1 Memory

The memory usage of different models during training is reported in Table 4.8.

Table 4.8: *R@K results for the transformer mean aggregation on the test data.*

Aggregation	Model	Memory (GB)
Average	B/16-32-frames	11.08
	B/16-16-frames	6.13
	B/16-8-frames	3.64
	B/32-16-frames	2.62
	B/32-8-frames	2.59
Transformer mean	B/16-32-frames	11.11
	B/16-16-frames	6.15
	B/16-8-frames	3.66
	B/32-16-frames	2.66
	B/32-8-frames	2.64

4.4.2 Compute

Figure 4.6 illustrates the relationship between the amount of compute used to train one model and its corresponding performance. As observed in the figure, the B/16-8 frame model achieved the highest R@1 score while using fewer computing resources compared to the 16 and 32-frame B/16 models.

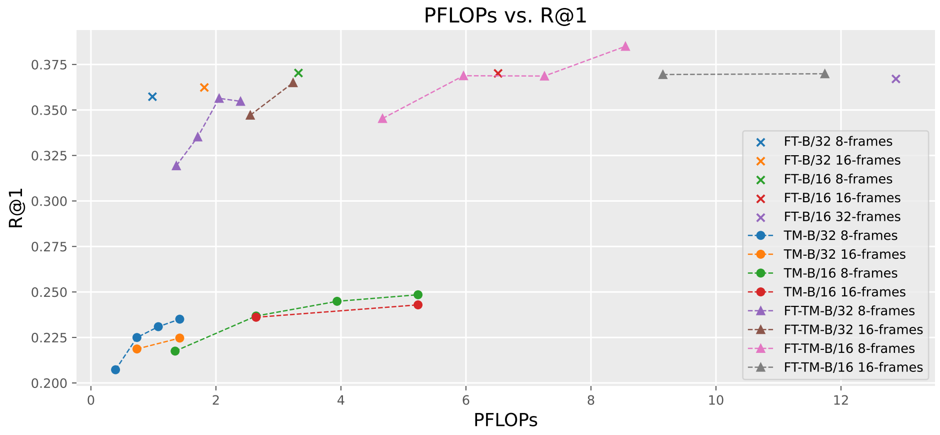

Figure 4.6: *Compute used for the training of different models. The model names indicate if it used the fine-tuned model (FT) and if the aggregation was the transformer mean (TM).*

4.5 Ablation testing

This section will present the impact of hyperparameter tuning on the models, namely the batch size, learning rate, layer freezing, and patch dropout. These results were obtained from the validation dataset. Therefore, the R@1 score may not align with those generated from the test data due to size differences between the two datasets.

4.5.1 Batch size

This section presents the different tests that were conducted by varying the batch size of the different models as described in Section 3.2.3. This was done both during fine-tuning of the baseline models as well as when training the aggregator.

Average finetuning

Figure 4.7 depicts the validation score obtained when training with different batch sizes. The performance seems to enhance with an increase in batch size. However, the rate of improvement appears to be model-specific.

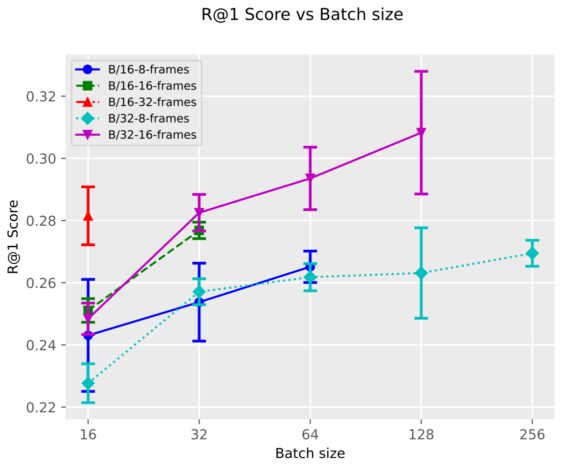

Figure 4.7: *R@1 validation score when training with different batch sizes.*

Transformer finetuning

Figure 4.8 shows the effect the batch size had when training the aggregator. Since the aggregator used pre-extracted features, it was not subject to the same memory limitations as the regular fine-tuning which made it possible to test all models at every batch size. This also shows that increasing the batch size leads to a better-performing model. The improvements seem to slow down between sizes 128 and 256.

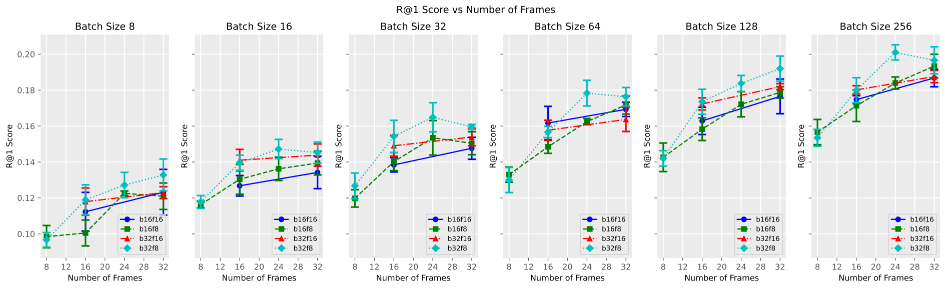

Figure 4.8: *R@1 validation score when training with different batch sizes.*

4.5.2 Learning rate

Different learning rates and optimizers were tested on the B/32-8 frame model as described in Section 3.2.3 and the results are displayed in Table 4.9. The AdamW optimizer achieved the best results, slightly outperforming the best results of the SGD.

Table 4.9: *Results for the learning rate sweeps using SGD and AdamW on the validation data.*

(a) *SGD*

Learning Rate	R@1 (%)	R@5 (%)	R@10 (%)
3e-5	13.96 ± 0.22	35.07 ± 0.33	47.59 ± 0.16
1e-4	18.00 ± 0.31	37.80 ± 0.45	52.38 ± 0.67
1e-3	21.01 ± 0.89	45.41 ± 0.74	59.96 ± 0.46
3e-3	22.57 ± 0.64	48.12 ± 0.33	61.37 ± 0.33
1e-2	13.38 ± 8.25	30.55 ± 16.53	41.54 ± 20.92

(b) *AdamW*

Learning Rate	R@1 (%)	R@5 (%)	R@10 (%)
1e-6	19.18 ± 0.30	41.11 ± 0.59	54.59 ± 0.22
3e-6	20.33 ± 0.39	45.56 ± 1.13	59.29 ± 1.13
1e-5	22.77 ± 0.82	48.29 ± 1.27	62.58 ± 0.95
3e-5	22.26 ± 0.63	47.59 ± 0.87	61.32 ± 1.61
1e-4	14.18 ± 0.89	37.25 ± 1.67	51.05 ± 0.80

4.5.3 Freezing layers

Figure 4.9 illustrates the performance when different layers are frozen, as discussed in Section 3.2.3. The results indicate that optimal performance was achieved when only the embedding layers were frozen.

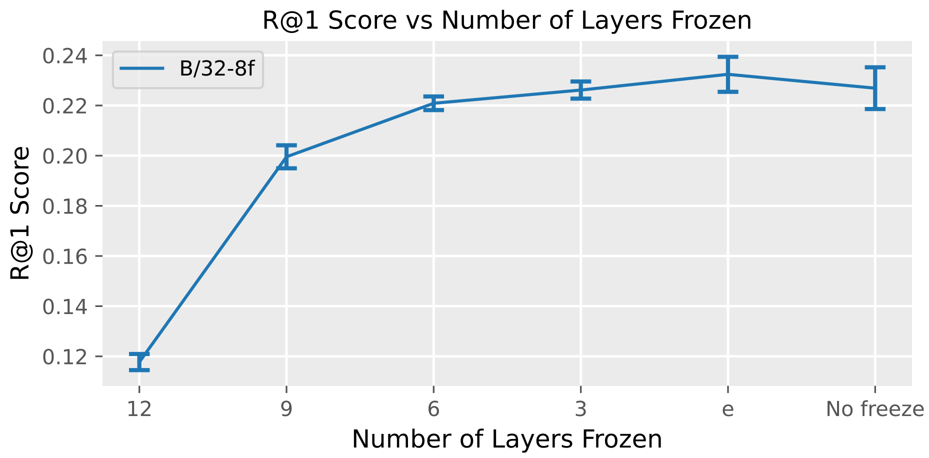

Figure 4.9: *Figure showing the R@1 validation score when training with different batch sizes.*

4.5.4 Patch dropout

The impact of using patch dropout, detailed in Section 3.2.3, is presented in Figure 4.10 and Table 4.10. Figure 4.10 depicts the R@1 score with various keep rates, while Table 4.10 displays the complete results.

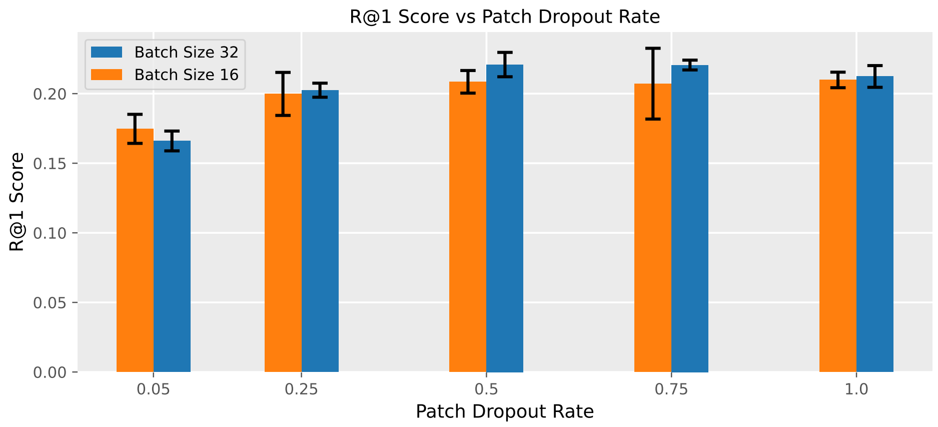

Figure 4.10: *R@1 validation score when training with different patch dropout rates.*

Table 4.10: *Results when varying patch dropout rate for two different batch sizes.*

(a) *Batch size 16*

Keep rate	Runtime	TFLOPs	R@1	R@5	R@10
1.0	459 0.0	348 1.0	0.21	0.473	0.609
0.75	455 -1.0%	266 ×0.76	0.207	0.461	0.608
0.5	451 -2.0%	191 ×0.55	0.209	0.472	0.617
0.25	449 -2.0%	116 ×0.33	0.2	0.448	0.593
0.05	449 -2.0%	54 ×0.16	0.175	0.394	0.526

(b) *Batch size 32*

Keep rate	Runtime	TFLOPs	R@1	R@5	R@10
1.0	371 0.0	348 1.0	0.212	0.476	0.617
0.75	351 -6.0%	266 ×0.76	0.221	0.478	0.628
0.5	342 -8.0%	191 ×0.55	0.221	0.485	0.624
0.25	336 -9.0%	116 ×0.33	0.202	0.461	0.604
0.05	337 -9.0%	54 ×0.16	0.166	0.387	0.512

4.6 Linear probe

The linear probe was performed as described in 3.2.4. Figure 4.11 shows the results, both the zero-shot performance of the regular X-CLIP models and the results of the linear probe when training with a varying number of samples from each class.

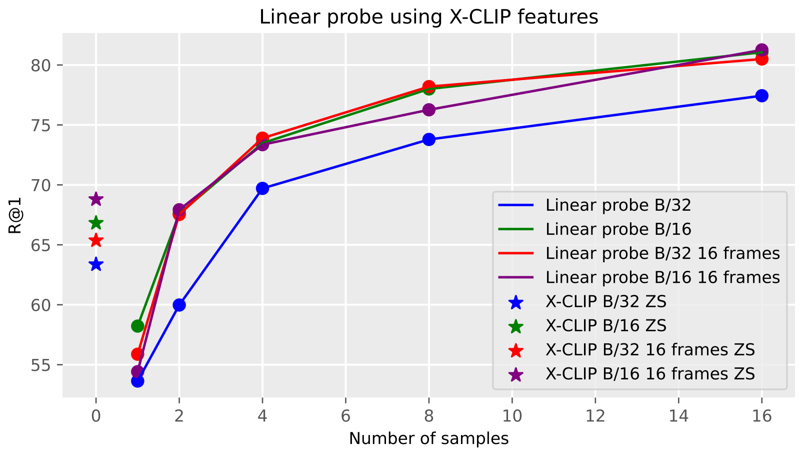

Figure 4.11: *Linear probe using features from different X-CLIP baseline models.*

5 Discussion

This chapter discusses the method, results, and some of the ethical considerations when training a video language model.

5.1 Method

The selection of a pretrained model to use as the base was an important choice. The two main contenders were the X-CLIP model and the CLIP4Clip model. X-CLIP was trained for action classification, whereas CLIP4Clip was pre-trained on 100 million video-text pairs. The CLIP4Clip model would have been a better fit since it was pre-trained on similar videos. However, using X-CLIP allowed for the testing of fine-tuning a model that had been pre-trained on dissimilar data. This emulates a more realistic scenario where there may not be any existing models trained on similar data.

The choice of aggregators was influenced by previous works such as CLIP4Clip, which employed a module for merging the frame embeddings. The methods chosen aimed to incorporate a combination of non-parametric and fully transformer-based approaches. The mean of the transformer embeddings is a middle ground, mixing both techniques. Other methods could have been used instead, for example, a simple linear layer on concatenated embeddings, an LSTM aggregator instead of a transformer, or some other kind of aggregation as proposed by Jiang et al. [41].

For the experiments in this thesis, all frames used were uniformly sampled from each video. However, an alternative approach could have a different sampling method such as randomly selecting frames from the entire video when training. This would have created more variation in the training data, serving as a simple kind of data augmentation.

There were many different profilers available to measure the FLOPs of the

PyTorch models. These profilers only estimated the computations that are done during the forward pass. For most of the measurements, this was fine since the backward pass could be estimated to be around two times larger than the forward pass and this would give the total compute for a training batch. Freezing a different amount of layers in the model will only affect the computations of the backward pass, making it difficult to estimate. The wall clock training time was instead used as an approximation to encompass the entire training process.

5.2 Results

In this section, the performance of aggregation methods, insights from the ablation study, and outcomes from the linear probe are reviewed.

Aggregation

The aggregation method yielded varied results, depending on whether mean-pooling was employed on the baseline models with or without fine-tuning. In the absence of fine-tuning, performance increases were observed for nearly all models. However, none could match the B/16 model using 32 frames at once.

Following fine-tuning, the B/16 8-frame model which aggregates 32 frames emerged with the highest R@1 score. The transformer mean aggregation also performed better when using the fine-tuned models, even outperforming the mean pooling. The incorporation of more frames seemed advantageous, though the B/32 8-frame model sometimes demonstrated diminished performance when extending from 24 to 32 frames. The B/32 model did not manage to outperform or reach parity using a smaller model with aggregation compared to the B/32 16-frame model. This was only achieved by the B/16 8-frame model, which even managed to outperform training with 32 frames at once. A possible cause for this is that when the patch size is too large there are too few tokens for the model to optimally learn. It could also be the case that when training for longer the B/16 16- and 32-frame models would also outperform the aggregated model. The class embedding aggregation proved to be less successful, underperforming by around 6-9 percentage points when compared to the transformer mean aggregation.

As indicated by the resource measurements, the smaller models unsurprisingly used less GPU memory and required fewer FLOPs to train. With the aggregation, the models with fewer frames performed better while training less, with some being superior. This could be related to the alignment between the typically three-sentence captions and the three clips extracted from the video. A model using 24 frames may be better tailored to this structure, enabling an optimized learning process through the alignment of video embeddings with individual sentences. In order to verify this, further testing would be required on the effects of combining different sentences.

Ablation study

The ablation study provided critical insights into the performance dynamics of different models and hyperparameters. Batch size emerged as a notable factor, with performance improvements seen with increased size. However, this gain showed diminishing returns with each doubling of size, particularly in the B/32-8-frame model. Due to hardware constraints, this phenomenon could not be thoroughly investigated for models with more frames or smaller patches. Regarding optimizers, AdamW slightly outperformed SGD, despite SGD lowering the memory requirements, potentially aiding in reducing memory usage. Examining layer freezing showed that freezing only the embeddings resulted in performance improvement while extending freezing beyond the first three layers led to a performance decline. The experiments also indicated that dropping up to half the patches did not negatively impact performance but slightly enhanced it, simultaneously reducing memory and computational load. The importance of balancing memory usage and batch size was reiterated by this finding, emphasizing the potential for enhanced performance through such optimization.

Linear probe

The results of the linear probe indicate that few-shot learning exceeds zero-shot predictions' performance after utilizing 2-4 samples from each class. This finding aligns with the results presented in the original CLIP paper. Moreover, the data distinctly demonstrate the improvement of the generated features when the patch size is reduced or more frames are added.

5.3 Ethical considerations

A common concern when training deep learning models is the amount of energy required to create highly performant models. One of the goals of this thesis was to create a model that reduced the amount of computing needed for fine-tuning. While big amounts of computing are needed when creating the base models like CLIP, efficient finetuning could allow for a reduction in training times.

It is also important to consider and investigate biases in the predictions, especially when using large pre-trained models such as CLIP that do not publically release what data was used to train. This is not only important for the creation of fair and equitable models, but also when considering reproducibility and reliability.

6
Conclusions

This chapter presents the conclusions of the thesis, which includes answering the research questions and looking at the potential for future works.

6.1 Research questions

1. **How does the performance of a smaller pre-trained model using aggregation compare with a larger model on the ActivityNet captions dataset?**
   From the discussion in Section 5.2, it was observed that the B/16-8 frame model benefited from aggregation in terms of performance. The performance increased even without fine-tuning, but it significantly improved after fine-tuning, allowing the smaller model to slightly outperform the larger B/16-32 frame model when aggregating the same number of frames.

2. **Can the proposed video retrieval method compare to other VLMs while using less memory and fewer FLOPs?**
   The proposed models were able to perform on par with larger models while using less computing and memory during training. As noted in Section 5.2 the proposed aggregation a smaller model was able to outperform the larger variants while using less GPU memory and fewer FLOPs.

3. **How much does unfreezing different layers and patch dropout during VLM finetuning affect training time and video retrieval performance?**
   Layer freezing and patch dropout both proved to be able to reduce the training time as well as increase the performance of the model.

4. **How well do the features from a pre-trained VLM with a few-shot logistic regression classifier compare to the zero-shot classification for action classification?**

The video embeddings generated by X-CLIP were able to be fitted using logistic regression to classify actions very efficiently, only requiring 2-4 samples from a given class to surpass the zero-shot performance.

6.2 Future work

While this thesis investigated the viability of using an aggregator, it is important to note that these methods were not extensively tested on multiple datasets. Therefore, while these findings suggest that the aggregation can allow a model with fewer frames to perform better, this might not be the case for other datasets with different video lengths or shorter captions.

To ensure a fair comparison among the models the number of frames used was capped to the size of the largest model, which was 32 frames. Since the aggregation is not limited to just 32 frames, further research can explore the potential benefits of incorporating additional frames into the aggregation process. This thesis investigated aggregation of frames, but aggregation in the spatial or textual domain is also possible. These could be additional avenues of research.

Finally, it would also be interesting to test if the aggregation approach can be applied to other downstream video tasks, such as video captioning or video summarization. There have been attempts to use CLIP for video captioning [5] and summarization [4], indicating the potential applicability of aggregation in those domains as well.

Bibliography

[1] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. URL `https://arxiv.org/abs/2103.00020`.

[2] Yongqin Xian, Christoph H. Lampert, Bernt Schiele, and Zeynep Akata. Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(9):2251–2265, 2019. doi: 10.1109/TPAMI.2018.2857768.

[3] Antoine Yang, Antoine Miech, Josef Sivic, Ivan Laptev, and Cordelia Schmid. Zero-shot video question answering via frozen bidirectional language models, 2022. URL `https://arxiv.org/abs/2206.08155`.

[4] Medhini Narasimhan, Anna Rohrbach, and Trevor Darrell. Clip-it! language-guided video summarization. 2021. doi: 10.48550/ARXIV.2107.00650. URL `https://arxiv.org/abs/2107.00650`.

[5] Mingkang Tang, Zhanyu Wang, Zhenhua LIU, Fengyun Rao, Dian Li, and Xiu Li. Clip4caption: Clip for video caption. In *Proceedings of the 29th ACM International Conference on Multimedia*, MM '21, page 4858–4862, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450386517. doi: 10.1145/3474085.3479207. URL `https://doi.org/10.1145/3474085.3479207`.

[6] Bolin Ni, Houwen Peng, Minghao Chen, Songyang Zhang, Gaofeng Meng, Jianlong Fu, Shiming Xiang, and Haibin Ling. Expanding language-image pretrained models for general video recognition. In *European Conference on Computer Vision (ECCV)*, 2022.

[7] Huaishao Luo, Lei Ji, Ming Zhong, Yang Chen, Wen Lei, Nan Duan, and Tianrui Li. Clip4clip: An empirical study of clip for end to end video clip retrieval, 2021. URL `https://arxiv.org/abs/2104.08860`.

[8] Yan-Bo Lin, Jie Lei, Mohit Bansal, and Gedas Bertasius. Eclipse: Efficient long-range video retrieval using sight and sound, 2022. URL https://arxiv.org/abs/2204.02874.

[9] Hu Xu, Gargi Ghosh, Po-Yao Huang, Dmytro Okhonko, Armen Aghajanyan, Florian Metze, Luke Zettlemoyer, and Christoph Feichtenhofer. Videoclip: Contrastive pre-training for zero-shot video-text understanding, 2021. URL https://arxiv.org/abs/2109.14084.

[10] Zuxuan Wu, Ting Yao, Yanwei Fu, and Yu-Gang Jiang. Deep learning for video classification and captioning. In *Frontiers of Multimedia Research*, pages 3–29. Association for Computing Machinery and Morgan & Claypool, dec 2017. doi: 10.1145/3122865.3122867. URL https://doi.org/10.1145%2F3122865.3122867.

[11] Jun Xu, Tao Mei, Ting Yao, and Yong Rui. Msr-vtt: A large video description dataset for bridging video and language. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5288–5296, 2016. doi: 10.1109/CVPR.2016.571.

[12] Ranjay Krishna, Kenji Hata, Frederic Ren, Li Fei-Fei, and Juan Carlos Niebles. Dense-captioning events in videos. In *International Conference on Computer Vision (ICCV)*, 2017.

[13] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition, 2007. ISBN 0387310738.

[14] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *Proceedings - 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2006*, Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 1735–1742, 2006. ISBN 0769525970. doi: 10.1109/CVPR.2006.100. 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2006 ; Conference date: 17-06-2006 Through 22-06-2006.

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

[16] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct 1986. ISSN 1476-4687. doi: 10.1038/323533a0. URL https://doi.org/10.1038/323533a0.

[17] Leslie N. Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472, 2017. doi: 10.1109/WACV.2017.58.

[18] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour, 2018.

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL `https://arxiv.org/abs/1412.6980`.

[20] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[21] Alex Graves. Generating sequences with recurrent neural networks, 2014. URL `https://arxiv.org/abs/1308.0850`.

[22] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1031. URL `https://aclanthology.org/P18-1031`.

[23] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, jan 2014. ISSN 1532-4435.

[24] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL `https://aclanthology.org/D14-1162`.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL `https://arxiv.org/abs/1706.03762`.

[26] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[27] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL `http://arxiv.org/abs/1810.04805`.

[28] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019. URL `https://arxiv.org/abs/1907.11692`.

[29] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In

International Conference on Learning Representations, 2021. URL `https://openreview.net/forum?id=YicbFdNTTy`.

[30] Lisa Anne Hendricks, Oliver Wang, Eli Shechtman, Josef Sivic, Trevor Darrell, and Bryan Russell. Localizing moments in video with natural language, 2017. URL `https://arxiv.org/abs/1708.01641`.

[31] Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid. Videobert: A joint model for video and language representation learning, 2019. URL `https://arxiv.org/abs/1904.01766`.

[32] Hu Xu, Gargi Ghosh, Po-Yao Huang, Prahal Arora, Masoumeh Aminzadeh, Christoph Feichtenhofer, Florian Metze, and Luke Zettlemoyer. Vlm: Task-agnostic video-language model pre-training for video understanding, 2021. URL `https://arxiv.org/abs/2105.09996`.

[33] Yue Liu, Christos Matsoukas, Fredrik Strand, Hossein Azizpour, and Kevin Smith. Patchdropout: Economizing vision transformers using patch dropout. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 3953–3962, January 2023.

[34] Karin Dembrower, Peter Lindholm, and Fredrik Strand. A multi-million mammography image dataset and Population-Based screening cohort for the training and evaluation of deep neural networks-the cohort of Screen-Aged women (CSAW). *J Digit Imaging*, 33(2):408–413, April 2020.

[35] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[36] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

[37] Bernard Ghanem Fabian Caba Heilbron, Victor Escorcia and Juan Carlos Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 961–970, 2015.

[38] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset, 2017. URL `https://arxiv.org/abs/1705.06950`.

[39] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020.

[40] Dario Amodei and Danny Hernandez. Ai and compute. `https://openai.com/research/ai-and-compute`, 2018. Accessed: 2023-05-23.

[41] Jie Jiang, Shaobo Min, Weijie Kong, Hongfa Wang, Zhifeng Li, and Wei Liu. Tencent text-video retrieval: Hierarchical cross-modal interactions with multi-level representations. *IEEE Access*, pages 1–1, 2022. doi: 10.1109/ACCESS.2022.3227973.