Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Ingrid Sofie Skjetne

Text-Video Retrieval Using Encoder and Cross Encoder Models

Master's thesis in Applied Physics and Mathematics
Supervisor: Thiago Martins
June 2022

Norwegian University of
Science and Technology

Ingrid Sofie Skjetne

Text-Video Retrieval Using Encoder and Cross Encoder Models

Master's thesis in Applied Physics and Mathematics
Supervisor: Thiago Martins
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Norwegian University of
Science and Technology

Abstract

In this thesis we investigate how encoder based deep learning models can be used for text-video retrieval. The motivation for using encoder based models is their demonstrated success on various text and vision tasks as well as their parallel treatment of input data, allowing for fast inference.

Image and text encoders from the CLIP model [1] by OpenAI are used as a base for our models. CLIP was published in 2021 and is a dual encoder model trained on a large number of image-caption pairs to create a common embedding space for images and text. Three main categories of models for video representation and retrieval are compared. The models take in CLIP features of frames sampled uniformly from a video, and the aim is to aggregate the information in the frame features over the temporal dimension to a video level feature. To create a baseline for zero-shot performance, we test a few aggregation methods without learnable parameters. Then, encoder models for temporal aggregation are tested. These have the advantage of allowing the video corpus to be pre-embedded, instead of embedding videos during search. Finally, cross encoder models for re-ranking are trained and tested.

The dataset used is the MSR-VTT dataset [2], and results are reported on the test-1k split, which is a common benchmark in recent video retrieval literature. The results show that the CLIP model provides a strong base for the task, with non-learnable methods having good performance.

The encoder models for aggregating temporal information prove successful in improving on the basic aggregations, and MRR@10 is improved by 20 % from 0.418 to 0.503. Encoder models with few layers grant the best retrieval results.

Various cross encoder training strategies are tested, but overall the performance of the cross encoder models in this report do not reach the levels of the ranking with pre-embedded features from encoders. The choice of loss function is found to influence results greatly, with listwise loss and contrastive loss outperforming pointwise loss.

Sammendrag

I denne oppgaven undersøker vi hvordan dyp læring med enkodermodeller kan brukes til videosøk. Motivasjonen for å bruke nettopp enkodermodeller er den store suksessen de har hatt på flere tekst-bildeoppgaver og deres parallelle behandling av data, som gir rask inferens.

Bilde- og tekstvektorer fra OpenAI sin CLIP-modell [1] brukes som et grunnlag for modellene våre. CLIP ble publisert i 2021 og er en dobbel enkoder som er trent på et stort datasett av bilde-tekst-par for å lage et felles vektorrom for de to datatypene. Tre modellkategorier for videorepresentasjon og videosøk sammenlignes. Stillbilder hentes uniformt fra en video og modellene tar inn CLIP-vektorer av disse stillbildene. Målet er å sammenfatte informasjonen fra disse vektorene over tidsdimensjonen i videoen og generere en ny vektor som er en representasjon på videonivå. For å få et mål på zero-shot-egenskapene til CLIP, tester vi aggregeringsmetoder uten trenbare parametre. Deretter trenes og testes enkodermodeller for sammenfatting over tidsdimensjonen. Med disse modellene kan videoer representeres som vektorer på forhånd, og trenger ikke å sendes gjennom modellen når et søk skal gjøres. Til slutt trenes og testes kryssenkodere for re-rangering av søkeresultater.

Datasettet som brukes er MSR-VTT [2], og søkeresultater rapporteres på test-1k-delen av datasettet. Resultatene viser at CLIP-modellen er et godt utgangspunkt for å bygge videosøkmodeller, ettersom den gir gode resultater for modeller uten trenbare parametre.

Søkeresultatene blir deretter forbedret ved bruk av enkodermodeller for aggregering av temporal informasjon. MRR@10 forbedres med 20 % fra 0,418 i den beste zero-shot metoden til 0,503 for de beste enkodermodellene. Modellene med få lag gir de beste søkeresultatene.

Forskjellige strategier for å trene kryssenkodere blir testet, men kryssenkoderene i denne rapporten gir ikke like gode rangeringsresultater som enkodermodellene. Valg av tapsfunksjon har stor påvirkning på resultatene og listetap og kontrasttap gir de beste resultatene, og utkonkurrerer punkttap.

Preface

This thesis marks the completion of my master's degree in applied physics and mathematics at the Norwegian University of Science and Technology (NTNU).

I would like to thank my supervisor Thiago Martins for introducing me to the topic of video retrieval and for his guidance during my specialization project and master thesis. I would also like to thank the NTNU HPC Group for giving me access to the IDUN cluster [3].

Finally, I would like to thank my family for all their support as well as my friends for the good times we have had during the last five years in Trondheim.

Ingrid Sofie Skjetne   Trondheim, June 13th, 2022

Contents

1 Introduction **1**

1.1 Research Questions 2

2 Theory **4**

2.1 Information Retrieval 4

2.1.1 Ranking and Re-Ranking 5

2.1.2 Retrieval Using Approximate Nearest Neighbor Search 5

2.1.3 Evaluation Metrics 6

2.2 Deep Learning 8

2.2.1 Multilayer Perceptron (MLP) 9

2.2.2 Activation Functions 10

2.3 Text Attention Models 10

2.3.1 Tokenization 11

2.3.2 The Transformer 12

2.3.3 Residual Connections 16

2.3.4 Layer Normalization 17

2.4 Vision Attention Models 18

2.4.1 The Vision Transformer (ViT) 18

2.5 Dual Encoders and the CLIP Model 20

2.5.1 Contrastive Loss 21
2.6 Cross Encoder Models 22

3 Retrieval Models 26
3.1 Non-Temporal Models 26
3.1.1 Pooling Aggregation 26
3.1.2 A Frame Model 26
3.2 Encoder Models for Temporal Aggregation 27
3.3 Cross Encoder Models 27

4 Related Work 29
4.1 CLIP4Clip: Sequence Models on Top of CLIP 29
4.2 CLIP2TV: Using Postprocessing for Search 29
4.3 Querybank Normalization 30
4.4 ViViT: A Pure Video Encoder 30
4.5 Mixture of Embedding Experts: Using Several Modalities 31

5 Data 32
5.1 The MSR-VTT Dataset 32
5.2 Data Preprocessing 33

6 Method 35
6.1 Optimization 35
6.1.1 Optimizer and Hyperparameters 35
6.2 Evaluation Method 37

7 Experiments and Results 39
7.1 Pooling Aggregation 39
7.2 The Max Frame Model 40
7.3 Temporal Encoder Models 40

7.3.1 Single-Head Attention Models 41
7.3.2 Multi-Head Attention Models 41
7.4 Re-ranking Using a Cross Encoder 43
7.4.1 Training Cross Encoders 43
7.4.2 Re-Ranking with Cross Encoders 44
7.4.3 Using a Contrastive Loss Function 47
7.4.4 Training Re-Ranking Models Using Mined Negatives 48
7.5 Discussion of Results and Research Questions 49

8 Conclusion 51

8.1 Summary 51
8.2 Further Research 52

Chapter 1

Introduction

In an information-driven society overflowing with data, the ability to search is valuable and enables us to utilize data better. Many use web search services daily, and our expectations and demands when it comes to speed and quality are increasing.

Deep learning methods have advanced greatly over the last decades, and are starting to find their way into everyday applications. Today, popular search engines are using machine learning models to provide useful hits. While most of us are familiar with the usefulness of text search, advancements in technology are enabling search in other types of data. Over the last years, search possibilities such as image search, reverse image search and music search have become available to the average consumer with a computer or smartphone. Whether hidden or obvious, machine learning models are transforming the search possibilities around us, increasing quality as well as enabling search in new types of media.

In this report, we will focus on text to video search. Video search on the web often uses additional text data such as a title, description or article and the name of the creator or publisher. What we will be concerning ourselves with in this report, however, is retrieval based on video content only.

The technical motivation for the interest in this topic lies in the improvements in representation learning for text, image and video which we have seen over the last few years. Image models have been synonymous with convolutional neural networks (CNNs) since the breakthrough that was AlexNet [4] in 2012. Since this, CNNs have evolved through a series of architectural innovations in models such as VGGNet [5], the Inception networks [6] and ResNet [7]. But since the introduction of the Transformer [8] in 2017, we have seen high-performing image models using the attention based architecture. In addition to inspiring new image models, the main impact of the Transformer was a breakthrough in text modelling. In this report, both the image and text models we are using will be Transformer based.

In 2021, OpenAI published a model called CLIP [1]. CLIP is a model which is able

to extract information from both text and images, and place these two modalities in a common embedding space. This enables computing similarities between text and images, giving remarkable results in tasks such as image retrieval and classification. The availability of pre-trained models which have learned a joint embedding space for text and image is crucial to this report. Pre-trained models for fine-tuning and transfer learning are promoting incremental development, more environmentally friendly model creation and training, and a democratization of deep learning. We want to explore and demonstrate the usefulness of pre-trained models, keeping additional training relatively cheap.

In this report, we will explore encoder models built on top of the CLIP dual encoder. Most of the heavy lifting is being done by the image-text understanding of the CLIP model. The task of the new parts of the model will be to utilize the temporal information in a video. Image features will be extracted from a series of video frames, and new encoder models will be tasked with aggregating these into one video level feature. Utilizing temporal information among frames is a much simpler task than creating a common embedding space for text and images covering a vast array of topics. Therefore, we hypothesize that simple encoder models might be powerful enough perform this task. In addition, cross encoder models for re-ranking of search results will be trained and tested.

1.1 Research Questions

To concretize the goals of this report, we formulate three research questions.

Research Question 1 *Can the knowledge in pre-trained image models be utilized for video retrieval?*

Strictly speaking, this is a question to which the answer is known. There are numerous papers describing models using pre-trained image models for video retrieval. The most successful ones are using the CLIP model. HunYuan_vtr [9] and DRL [10] are two examples. Still, we pose this question as we want to confirm and demonstrate the usefulness of CLIP for video retrieval.

Research Question 2 *Can an encoder model aggregating temporal information improve on trivial models for video retrieval?*

This question gets to the main aim of the report. We want to investigate whether using an encoder to extract temporal information adds value to the video retrieval model. Hypothesizing that a simple model could be suitable for this task, the main focus is on small encoder models. These also provide quick inference. A low inference time is especially desirable in the context of search, where retrieval time affects both usefulness and user experience.

Research Question 3 *Can re-ranking using a cross encoder improve retrieval results?*

Acknowledging that dual encoder models do not enable much interaction between document and query, the final aim of the report is to investigate the potential of cross encoder models for re-ranking in the video retrieval context. Such models allow for much more advanced interactions between the query and the documents, at the cost of slower inference. We wish to investigate the usefulness of text-video cross encoders, focusing on re-ranking.

Chapter 2

Theory

In this chapter we give a brief introduction to the field of information retrieval, and present some general background on deep learning. As the interest of this report is model architecture, we go into some detail on the theory behind the models used. The specific models used in later experiments and more on how these models can be used for search will be discussed in Chapter 3.

2.1 Information Retrieval

The field of information retrieval concerns itself with retrieving relevant information from a collection of data. The available data is called the *corpus*, and the data in the corpus is collected in *documents*. The task of a search engine is to receive a *query* representing the user's information need, and retrieve a set of documents ranked by relevance to the query.

Modern information retrieval is not simply a question of matching words like an index in the back of a book. Rather, it is a developing field taking advantage of new technologies and developing methods specializing in different types of retrieval. Some of the breadth in the text retrieval field can be seen by looking at the annual TREC (Text REtrieval Conference) conference [11], where participants use both traditional non-neural methods as well as neural nets and pre-trained models.

TRECVID, a separate video retrieval conference arranged by TREC, displays some of the variation within video retrieval needs. The 2021 TRECVID conference had tracks such as ad-hoc search, activity search, instance search and video summarization.

In this section we present the basics of the mechanics of a search, and a set of metrics which will be used for evaluating search rankings.

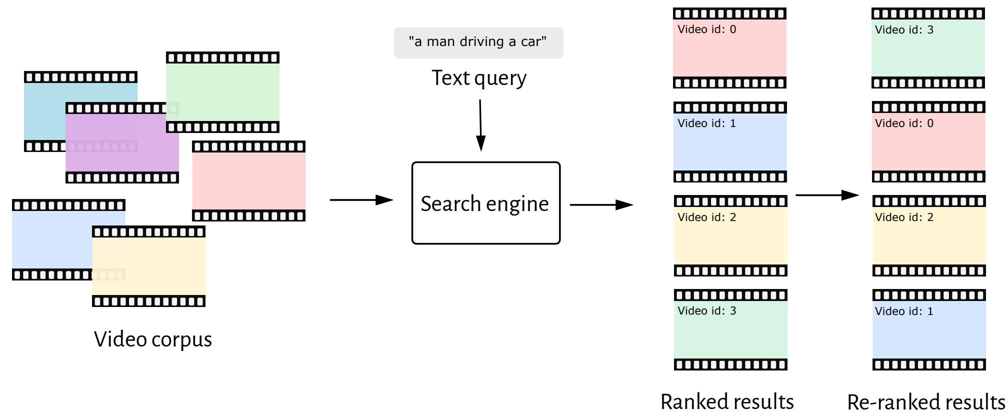


Figure 2.1: Illustration of the retrieval and ranking phases.

2.1.1 Ranking and Re-Ranking

Information retrieval can often be divided into three stages. First, documents satisfying some minimal requirement are retrieved in a *matching* stage. For a text search, this might be all documents containing a set of words. Then, matched documents are ranked according to their relevance to the query in the *ranking* stage. Finally, one can optionally perform *re-ranking*. Re-ranking would typically involve using a slower, more expensive ranking method to re-rank the top ranked results from the previous phase. Because it is being applied to fewer documents, re-ranking allows for use of slower, more accurate methods. A simple illustration of the phases of a search is shown in Figure 2.1, where a text query is sent to a search engine which retrieves, ranks and re-ranks results from a video corpus.

2.1.2 Retrieval Using Approximate Nearest Neighbor Search

It is not obvious how search can be carried out across data types. In order to make a search we need to calculate a relevance score between the query and the documents in the corpus. Often some type of similarity between the two is used to define which documents are relevant. In text retrieval, it is intuitive to imagine comparing the number of occurrences and the order of words, while in image-to-image search one can compare shapes, colors and pixels. But how do we know if the pixels, shapes and motions of a video are relevant to a query sentence? We solve this problem by representing the text query and the video documents in the same format. In this report, each sentence and each video is represented as a 512-dimensional vector. Once we have both query and document represented as vectors, calculating a similarity between the two is straight forward.

The search method which will allow us to perform text-video search is approximate nearest neighbor search (ANN). The idea is to embed a text query and the video documents

in the corpus as vectors in the same space. When a query is made, it is embedded in the joint embedding space and the closest videos by some similarity measure are retrieved. Closeness or similarity can be measured in several ways, and we must define a similarity measurement to use. Two examples of distance measures are the Euclidean distance and cosine similarity.

The Euclidean distance between two points defined by the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined as

$$d(\boldsymbol{x}, \boldsymbol{y}) = ||\boldsymbol{x} - \boldsymbol{y}||_2.$$

The cosine similarity of two vectors is the cosine of the angle between the vectors,

$$\text{sim}(\boldsymbol{x}, \boldsymbol{y}) = \cos(\theta(\boldsymbol{x}, \boldsymbol{y})) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}||_2 \cdot ||\boldsymbol{y}||_2}.$$

The cosine similarity is the closeness measure we will be using. Similar vectors will point in similar directions, while very different vectors will be approximately orthogonal to each other.

The reason why ANN search is used in place of exact nearest neighbor search, is that exact search is slow when the number of vectors in the search space is large. We therefore prefer to trade some accuracy for faster search. For vector search we are using the Vespa engine [12], which implements a modified version of the graph-based HNSW algorithm [13] for ANN search.

Of course, creating a common embedding space for video and text is not a simple task, and this is where deep representation learning comes in. We will discuss the creation of embeddings in later sections.

2.1.3 Evaluation Metrics

In order to be able to evaluate the models we will be discussing in this report, we need some evaluation metrics. The basics of what makes a search ranking good, is that it places more relevant results higher. But we can design different metrics depending on what is important to us. Perhaps we mostly care about the very top result. Or perhaps the order of the retrieved documents does not matter that much. In some cases we might care mostly about retrieving all relevant results, while in others it is more important not to retrieve too many irrelevant documents. Here we present some search evaluation metrics which we will be using to measure the ranking capabilities of our models. In this report our main interest is model architecture and without a specific application in mind, there is not one evaluation metric which sticks out as the most important. Therefore we will report several metrics in our later experiments.

Recall

A commonly cited information retrieval metric is recall, which measures how many of the relevant documents were retrieved. It is defined as

$$\text{Recall} = \frac{\text{No. of relevant documents retrieved}}{\text{No. of relevant documents in corpus}}.$$

Recall can also be calculated at a specific rank, meaning that recall is calculated using only the top k ranked documents. This is called recall-at-rank, and we denote it Recall@k or R@k, where k is the rank used. Common metrics for evaluation in video retrieval are R@1, R@5 and R@10.

Recall is perhaps especially useful when only one or very few documents are considered relevant to the query, or in applications where is it vital to retrieve most or all relevant documents. In recent video retrieval papers, recall-at-rank is one of the most commonly reported evaluation metrics.

Precision

Precision is another metric for evaluating the set of retrieved documents. Precision measures how many of the retrieved documents are relevant, and is defined as

$$\text{Precision} = \frac{\text{No. of relevant documents retrieved}}{\text{No. of documents retrieved}}.$$

Like recall, precision can be specified at a given rank k, this is denoted P@k. While a high precision is always desirable, precision might be an especially important metric when there are many almost equally relevant documents or when user experience is more important than retrieving all relevant documents in the corpus. High precision might give a better user experience because the user is not left to filter out irrelevant results manually.

Mean Rank, Median Rank and MRR

Mean and median rank are two other commonly cited retrieval metrics. The mean and median ranks are the mean and median ranks of the highest ranked relevant video. The mean reciprocal rank, MRR, is

$$\text{MRR} = \frac{1}{N_Q} \sum_{i=1}^{N_Q} \frac{1}{\text{rank}_i},$$

where rank_i is the rank of the highest ranked relevant video and N_Q is the number of queries tested. If no relevant video is retrieved, we set $1/\text{rank}_i = 0$.

We also report this metric at a specific rank. When reporting the MRR at 10, we retrieve

10 videos and use these to calculate MRR.

Normalized Discounted Cumulative Gain (nDCG)

The normalized discounted cumulative gain (nDCG) is a retrieval metric which adds up the relevance of retrieved documents after discounting their relevance value based on their rank, and divides the result by the maximum possible value. To understand nDCG we must first look at the discounted cumulative gain (DCG). The DCG at rank k is

$$\text{DCG@}k = \sum_{i=1}^{k} \frac{2^{\text{rel}_i} - 1}{\log_2(i+1)},$$

where rel_i is the relevance of document i and k is the number of documents retrieved. The DCG adds up the gain in usefulness (relevance) for each retrieved document, and discounts the usefulness of a document by the position in which it is ranked. The more documents we retrieve, the higher the maximum DCG. We therefore create a normalized metric, normalized DCG (nDCG), by dividing by the ideal DCG score. We find the ideal DCG score by sorting the documents by relevance and using the top k relevant documents in the DCG calculation. The ideal DCG is then

$$\text{IDCG@}k = \sum_{i=1}^{N_{\text{rel},k}} \frac{2^{\text{rel}_i} - 1}{\log_2(i+1)},$$

where $N_{\text{rel},k}$ is the number of relevant documents that can be retrieved at rank k.

Now the nDCG is

$$\text{nDCG@}k = \frac{\text{DCG@}k}{\text{IDCG@}k}.$$

Normalized Discounted Cumulative Gain (nDCG) is a commonly used metric for web search, where a relevant retrieved document is deemed relatively more useful when ranked highly. nDCG is created to also handle non-binary relevance scores, but in this report we will only be using binary relevance.

2.2 Deep Learning

Deep learning is a branch of machine learning characterized by artificial neural networks consisting of several layers, allowing for complex models. We will now go over some deep learning models and concepts. What they have in common is that our objective when using them is to generate a useful representation of text or video data for search.

Inspired by the network of neurons in the human and animal brains, artificial neural networks consist of nodes called neurons stacked in layers. This simple idea has inspired a variety of different artificial neural networks, spanning from basic multilayer per-

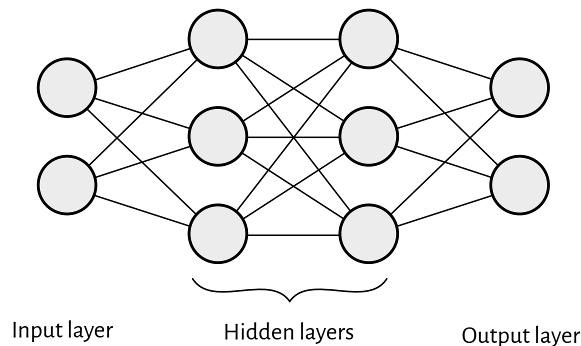


Figure 2.2: An MLP with two hidden layers.

ceptrons (MLPs) to convolutional neural networks (CNNs), recurrent neural networks (RNNs) and transformer models.

The idea behind these machine learning models is to create a model structure with a set of learnable parameters which are optimized through training on data. The model structure defines the restrictions and possibilities of the resulting function, and training using data updates the parameters of the model, creating a function which is useful for the task at hand.

2.2.1 Multilayer Perceptron (MLP)

A multilayer perceptron (MLP) is a feedforward neural network consisting of nodes called neurons connected in layers. The network consists of an input layer, and an output layer with one or more hidden layers in between. Figure 2.2 shows an MLP with two hidden layers. The illustration shows how each neuron is connected to all neurons in the previous layer and the next layer.

Each neuron in a layer takes as input a weighted sum of outputs from all neurons in the previous layers, possibly with an added bias. The neuron applies an activation function and outputs the resulting activation. Letting the outputs from neurons in the previous layer be $\boldsymbol{x} \in \mathbb{R}^n$, the output from a neuron i in layer l is the activation

$$a_{i,l}(\boldsymbol{x}) = g(W_{l,i}\boldsymbol{x} + b_{l,i}),$$

where $W_{l,i} \in \mathbb{R}^n$ gives the weights of the neuron, and $b_{l,i} \in \mathbb{R}$ is the bias parameter. $g(\cdot)$ is the activation function.

The trainable parameters of such a model are the weights and biases of each neuron.

2.2.2 Activation Functions

Applying non-linear activation functions throughout the network is vital for the representational power of the network, and ensures that the network can represent non-linear functions. There are a range of commonly used activation functions. The models in this report use the ReLU and GELU activation functions.

ReLU The rectified linear unit (ReLU) function is a common activation function in neural networks. The function is defined as

$$\mathrm{ReLU}(x) = \max\{0, x\},$$

and makes for cheap evaluation and gradient calculations. The derivative of the ReLU function is simply

$$\mathrm{ReLU}'(x) = \begin{cases} 0, & x < 0, \\ 1, & x > 0. \end{cases}$$

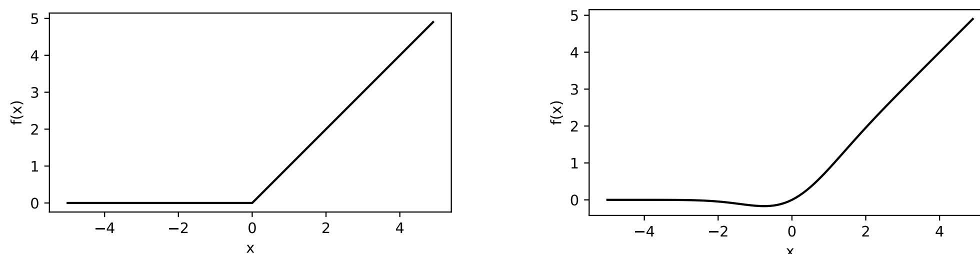
GELU An activation function commonly used in transformer models is the Gaussian Error Linear Unit (GELU) function [14], which was introduced in 2016. This function is defined as

$$\mathrm{GELU}(x) = x \cdot \Phi(x),$$

where Φ is the cumulative distribution function of the standard normal distribution. The graphs in Figure 2.3 display the ReLU and GELU activation functions. We can see that they are quite similar, but while ReLU gives zero activation for all negative inputs, GELU gives slightly negative activations for small negative inputs. Because $\Phi(x) \to 0$ as $x \to -\infty$ and $\Phi(x) \to 1$ as $x \to +\infty$, the GELU function will approach the RELU function for $x \to \pm\infty$. The GELU can be seen as a smoothing of the ReLU. In the paper introducing GELU [14], the GELU outperformed the ReLU in several classification experiments.

2.3 Text Attention Models

The Transformer model [8] was published in 2017, and has caused a small revolution within text models since. Before this introduction, common architectures used within text modeling were RNNs and even CNNs. Central to the Transformer architecture is the attention mechanism. Within the last years, we have seen attention-based language models such as BERT [15] and GPT-3 [16] performing very well at many different tasks such as translation, text representation, question answering and text generation. In this section we explain the structure of the Transformer model, starting with the text preprocessing. Note that when referring to the specific Transformer model published



(a) The ReLU activation function.

(b) The GELU activation function.

Figure 2.3: The ReLU and GELU activation functions.

in 2017, we will be writing the *Transformer*, while for the general model category this belongs to, we will be writing *transformer* models.

2.3.1 Tokenization

Before being sent to a transformer model, text data must be represented as vectors. This is done through tokenization and embedding. A tokenizer splits text into building blocks, and defines a vocabulary with which all input text in the target language can be expressed.

A tokenizer learns a vocabulary of tokens from example text. The simplest building blocks which can make up a sentence are single letters, but after seeing example text, a tokenizer with a large vocabulary will create tokens corresponding to whole words and parts of words. For example, common words such as "the" or "are" will be given their own token. In a small vocabulary, the tokenizer may not have room for all common words, and a word may be split into parts. For example, "walking" could be expressed as "walk" + "ing".

There are also some special tokens in a text transformer. There is the `[UNK]` token, which is used for words that cannot be expressed with the vocabulary of the tokenizer. There is also the `[PAD]` token which is simply used to pad sequences which do not fill the entire context length used. By context length we mean the number of tokens the input to the encoder is set to contain. Finally, there is the important `[CLS]` token. This is a learnable embedding which is used when we want the transformer model to output a single representation for the entire input. It is explained in more detail later.

In order to input the tokens to a transformer encoder, the tokens are numbered and embedded using a mapping which simply maps each token id number to a vector of the correct input dimension. This embedding is a learnable part of the model. Figure 2.4 illustrates the process of tokenizing text into embedding vectors. The figure shows an input sentence being split into tokens, which are then translated to a list of token

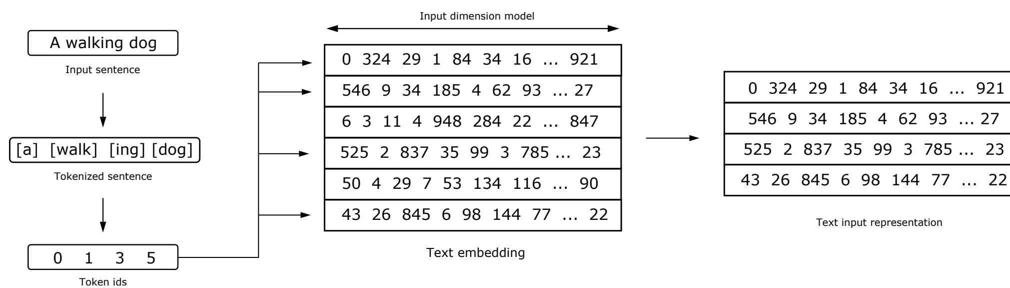


Figure 2.4: An illustration of the tokenization process for a sentence input to a text encoder.

id numbers. Using the token id numbers as indices, text representation vectors are retrieved from a learnable embedding. The result is that the input sentence has been represented as a list of vectors.

2.3.2 The Transformer

The Transformer Architecture

The original Transformer consists of both an encoder and a decoder. These have several layers containing multi-head attention mechanisms and MLP blocks. While the encoder attention mechanisms consist of tokens paying attention to themselves using self-attention, the decoder pays attention to the outputs of the encoder. This architecture is especially useful for translation tasks. For text representation in this report, we will only be using encoder models. A simplified version of the Transformer architecture is shown in Figure 2.5, with the encoder shown to the left. The figure only shows a single encoder layer, but the base model of the Transformer has six encoder layers and six decoder layers. We will not go into any details about the decoder architecture. Each encoder layer contains the multihead self-attention mechanisms which are the core of an encoder, as well as a feed-forward network, residual connections and layer normalization. The feed-forward networks use a GELU activation function. We will now explain the building blocks of the encoder in more detail.

Attention Mechanisms

The attention function takes three vector inputs: the query, key and value. We let several such vectors be stacked together and denote the resulting query, key and value matrices by Q, K and V respectively. Then $Q, K, V \in \mathbb{R}^{T \times d_{\text{model}}}$. Here, T is the number of tokens input to the encoder, and d_{model} is the dimension of vectors input to the encoder.

A few variations on the attention mechanism exist, and we will be using the scaled dot

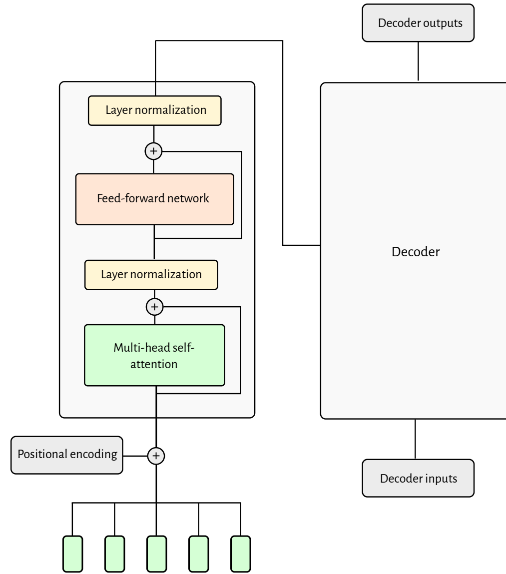


Figure 2.5: Transformer architecture overview, showing the structure of the encoder layer. Only one encoder and decoder layer shown.

product attention function, which is

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_{\text{model}}}}\right)V$$

where $1/\sqrt{d_{\text{model}}}$ is the scaling factor. The dot product is scaled to attain a variance of 1, in the case where the components of Q and K are independent random variables with variance 1. Without this scaling, the absolute value of the argument of the softmax function can become very large, making the gradient of the softmax very small [8]. The softmax function is

$$\text{Softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)},$$

and we take the softmax over the rows.

Attention mechanisms resemble how humans pay attention to different parts of a text. To understand the content of a sentence, humans may pay attention to a few key words, and place less weight on others. This is similar to self-attention. The aim of the encoder is to create a representation of the input data. And using attention mechanisms, the encoder will pay more attention to the features of the data that are important for the task it is trained at. Similarly, when reading a sentence a human might mostly pay attention to the verb, subject and object of the sentence, and pay less attention to for example conjunctions, which may not be as central to the meaning of the sentence. In the case of an image, one might pay more attention to people and objects in the foreground, and less to the background. The model learns what pieces of the data it

should pay attention to and what interactions between different parts of the data are important through training.

When used for translation tasks, attention can be calculated across sentences in the origin language and target language. Then, when predicting a word in the target language, the model mostly pays attention to one or a few words in the original sentence which are important for deciding this particular word.

The attention mechanisms allow the Transformer to use interactions between the entire input sequence, and the model does not forget or discount parts of the sequence. This is an advantage the attention mechanism, through its parallel treatment of input data, gives Transformer models over RNNs.

Multi-Head Attention

Multi-head attention performs several attention operations in parallel. In the encoder architectures described in this report, multi-head *self*-attention is used. As the name suggests, in self-attention, the queries, keys and values are equal. The idea behind multi-head attention is to let the model pay attention to different aspects of the input within one layer. Each head can be seen as extracting a different feature from the inputs. To make the attention computation cheaper, the inputs to each attention head are projected to a lower dimension. With a model dimension d_{model}, and h heads in a layer, the dimension for each head is d_{model}/h. Thus the number of heads must be chosen such that d_{model} is divisible by h.

Each head performs scaled dot product attention, and the output from the multi-head attention function is a projected concatenation of the outputs from each attention head:

$$\text{MultiHeadAttention}(Q, K, V) = [\text{head}_1, \ldots, \text{head}_h]W,$$

where $W \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ is a learnable projection matrix.

Letting the number of input tokens be T, the dimensions of the input matrices Q, K and V are $(T \times d_{\text{model}})$.

There are three learnable projection matrices W_i^Q, W_i^K and W_i^V belonging to each head, used for projecting each of the three inputs, and the output from head i is

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V),$$

where $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}/h}$.

Using multi head attention in place of single head attention allows the models to extract different features from the input vector in parallel, by paying attention to different aspects of the input. Multi head attention could perhaps be seen as an ensemble of smaller single head attention mechanisms.

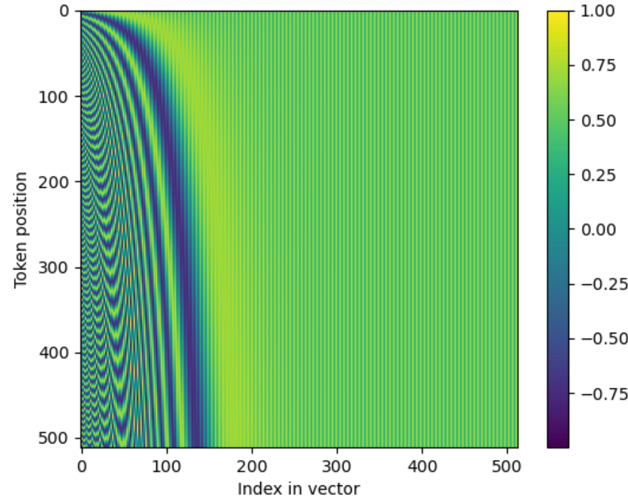


Figure 2.6: Values of the sinusoidal positional embedding.

Positional Encoding

A transformer model takes in input tokens in parallel and the architecture does not take into account the order of the tokens. But transformers are often used for sequential data, such as text, images or video. Text is sequential as the order of words have significance for the meaning of a sentence. Images may not usually be categorized as sequential data, but when dividing an image into smaller patches, the relative position of patches is important. In the case of video, the order of frames matter for the content of the video.

Because of the non-sequential nature of transformers, it is common to add a positional encoding to inputs as a way to make it easier for the model to learn the relative order or position of tokens. Such positional encodings may be learnable or not.

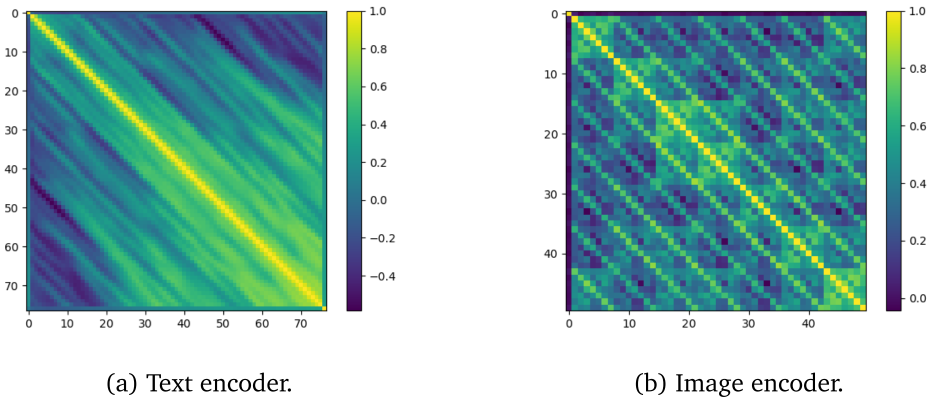
The original Transformer [8] used a non-learnable sinusoidal positional encoding given by

$$P_{pos,2i} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{2.1}$$

$$P_{pos,2i+1} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right), \tag{2.2}$$

where pos is the token position and i is the index within a token vector. Figure 2.6 shows how the values of the sinusoidal positional embedding varies with the token position as well as the index in the positional embedding vector.

The positional encoding may also simply be a learnable parameter which is initialized randomly. This approach is taken by ViT [17], which is used in CLIP [1]. This will be similar to the learnable text token embedding. There will still be patterns in the learned positional encoding vectors, but they are not as obvious to the human eye. However, if



(a) Text encoder. (b) Image encoder.

Figure 2.7: Cosine similarity of positional embedding vectors in different positions in the CLIP text and image encoders.

we plot the cosine similarity between position embedding vectors in different positions, we are able to see patterns. Figure 2.7a shows the cosine similarities of positional embeddings in the CLIP text encoder. It shows some periodic patterns, such that tokens with the same distance to each other have similar relations between their positional embeddings.

Figure 2.7b shows the cosine similarities for the position embeddings in the CLIP image encoder. Here we see clear periodic patterns, and we see that every seventh embedding is similar. This is because with input images of size 224×224 and patches of size 32×32, each input image is divided into patches forming a 7×7 grid. The cosine similarities in the figure show that the encoder learns the relative positions even in a 2D grid.

It has also been shown that even without position encodings, transformer models learn the relative position of tokens [18]. We still choose to use positional embeddings in our models later.

2.3.3 Residual Connections

The transformer models used in this report all contain an architectural feature called a residual connection, also known as a skip connection. A standard transformer encoder layer has two residual connections. The motivation behind residual connections is to combat what is known as the vanishing gradient problem. As neural networks become deeper, the gradients of the loss with respect to the weights in the first layers become small, harming the training of these weights. Residual connections allow gradients to flow more directly through the network, allowing for training of very deep neural networks [7].

The output from a block with a residual connection is

$$\mathcal{H}(x) = \mathcal{F}(x) + x,$$

such that the function the block needs to approximate is $\mathscr{F}(x) = \mathscr{H}(x) - x$. Some of the premise of the residual connection is the hypothesis that $\mathscr{F}(x)$ is more easily approximated than $\mathscr{H}(x)$.

Without introducing any extra parameters, these types of connections enabled the creation and training of networks such as the ResNet models with 50 and more convolutional layers [7].

2.3.4 Layer Normalization

The encoder models in this report have two layer normalizations [19] in each layer. Layer normalization is used to speed up training of neural networks, while overcoming drawbacks of the common batch normalization [20] method.

Normalization techniques are used to overcome the internal covariate shift problem. Internal covariate shift is the phenomenon that the distribution of outputs from a layer changes as the network is trained. Because the function that each layer is trying to learn thus changes as the network is trained, training takes longer. By using normalization, this problem is remedied.

The layer normalization operation normalizes the inputs to a layer. The statistics used for the normalization are the mean and standard deviations of inputs to all hidden units in the layer, for a single training example.

Letting $\boldsymbol{x}_l = [x_{l,1}, \ldots, x_{l,H}]$ be the activations of neurons in layer l, the statistics used for normalization are

$$\mu_l = \frac{1}{H}\sum_{i=1}^{H} x_{l,i} \qquad \sigma_l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(x_{l,i} - \mu_l)^2},$$

where H is the number of neurons in layer l.

The normalized inputs to layer $l+1$ are

$$x_{l,i}^{n} = \frac{x_{l,i} - \mu_l}{\sigma_l}, \quad i = 0, \ldots, H.$$

In addition to being normalized, the inputs are also scaled using learnable parameters β and γ. The final inputs become

$$\boldsymbol{x}_l^{\mathrm{LN}} = \gamma \cdot \boldsymbol{x}_l^{n} + \beta.$$

Layer normalization computes normalization statistics using a single training example, and can thus work with all batch sizes, even a batch size of one. This is unlike batch normalization, which computes normalization statistics over a batch of training examples, excluding batch size one as a possibility.

2.4 Vision Attention Models

In this section we will go over the theory of the visual models used in this report. Like the text models discussed earlier, these are based on the transformer encoder architecture.

2.4.1 The Vision Transformer (ViT)

The Vision Transformer (ViT) [17], published in 2020 by Google Research, brought the transformer architecture to a visual context. It follows the original Transformer architecture closely, but makes some architectural adjustments to allow for image data. Equivalently to the tokenization of input text to the Transformer, input images to ViT are split into patches which are embedded. These patches, consisting of areas of 32×32 pixels, are the equivalent of single tokens in the text model. The image patches are embedded using a learnable embedding in the form of a linear projection. Figure 2.8 illustrates how an input image is split into patches which are projected. It also shows clearly how the different positional embeddings (in gray) are added to each token. The output from the `[CLS]` token is taken as the image representation.

Let the input image be $\boldsymbol{x} \in \mathbb{R}^{H \times W \times C}$, where C is the number of channels. With a patch length p, the image is divided into patches $\boldsymbol{x}_P \in \mathbb{R}^{p \times p \times C}$. These are then flattened and projected by a learnable linear projection $\boldsymbol{E}$ to

$$\boldsymbol{x}_{P,\text{in}} = \boldsymbol{x}_P \boldsymbol{E}.$$

The patch embeddings are concatenated. Then, a learnable `[CLS]`-token embedding is prepended and a learnable positional embedding like the ones described in Section 2.3.2 is added, such that the final input to the vision encoder becomes

$$\boldsymbol{x}_{\text{in}} = \begin{bmatrix} \boldsymbol{x}_{\text{CLS}} \\ \boldsymbol{x}_1 \boldsymbol{E} \\ \boldsymbol{x}_2 \boldsymbol{E} \\ \vdots \\ \boldsymbol{x}_{N_p} \boldsymbol{E} \end{bmatrix} + \boldsymbol{E}_{\text{pos}}$$

A few variations of the ViT exist. The ViT-Base model has 12 encoder layers with 12 attention heads each. The model takes the output of the `[CLS]` token as the representation of the input image when ViT is used for eg. image classification, which the original paper focuses on.

On Representation Learning and Comparison with CNNs

Representation learning is a field of machine learning which is concerned with creating useful learned representations of data. The objective when using ViT, be it for image

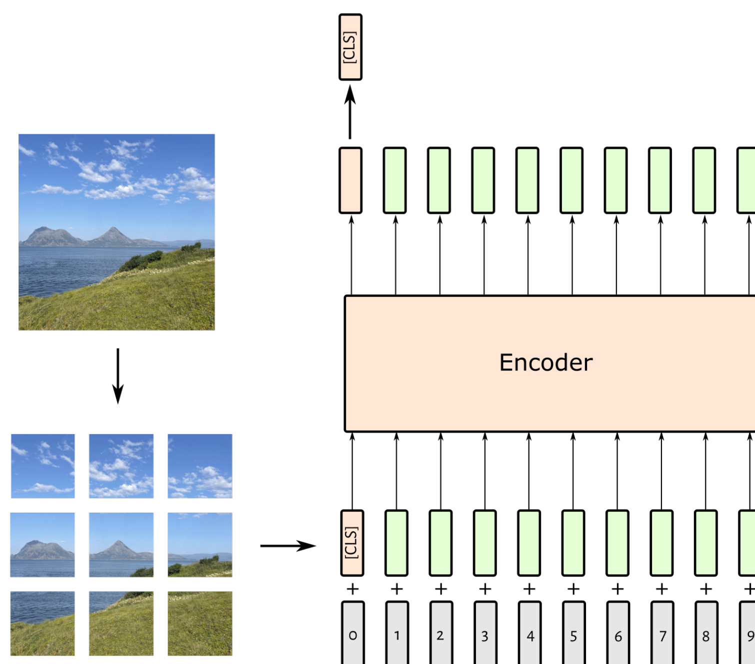


Figure 2.8: Illustration of the image encoder architecture. Image patches are projected to vectors and positional encodings are added.

classification or retrieval, is to create a lower dimensional representation of the input image. The input image to ViT has dimensions $224 \times 224 \times 3$, and the encoder creates a much smaller representation of the image with only 512 dimensions.

What makes a useful representation will depend on the application in question. A classic example of how representation affects results involves a classification problem with a set of data points in the plane, belonging to two classes. The points in the first class are spread out on a disk, while the second group is spread out in an annulus around the disk. Say we want to classify the points into the two classes, using a model with a linear decision boundary. If we represent our points using Cartesian coordinates, separating the two classes is impossible. But when represented using polar coordinates, the data points can be perfectly classified into the two classes.

When using models such as ViT for representation learning, the goal is dimension reduction through extraction of the most important data features. It is natural to compare this method of feature extraction to a CNN, which is probably the most known type of image model. The feature extraction in a CNN is more intuitive than in an encoder. Each layer in a CNN contains filters, or kernels, which extract features from the image, starting with low level features such as edges in the first layers and registering more and more abstract features in later layers.

One difference between a CNN and a ViT is in what is known as the receptive field. The receptive field for a neuron in a CNN is the area of the original input image which affects the output of the neuron. The neurons in early layers have small receptive fields, focusing on local features, and the receptive field increases as we move to later layers. The CNN builds understanding of an image by building global abstract features from local features. Because self-attention is calculated across all tokens in each layer of ViT, the receptive field of attention heads in early layers of ViT are not restricted in the same way as in a CNN. Because of this, a ViT can utilize both global and local connections throughout all layers.

CNNs have some inductive biases which make them suitable for vision tasks. This means that the nature of the architecture corresponds well with assumptions we make about the vision task. For example, we assume translational invariance, that a filter extracting a useful feature from one part of the image will also be useful for another part of the image. This parameter sharing between different parts of the image is central to CNNs. Because transformers lack the inductive biases of CNNs, they may need larger amounts of data for training, and may respond well to data augmentation.

2.5 Dual Encoders and the CLIP Model

The main part of a dual encoder, sometimes called a bi-encoder, two towers model or a dual tower model, is its two encoders. Often used for multi modal models, a dual encoder sends inputs through two parallel encoders, possibly with additional embeddings in either end. Each encoder takes inputs of one modality, so we might have one image encoder and one text encoder. While these can be treated as two independent models,

the encoders are trained together to create a common understanding of the inputs. After training the encoders together, they can be used separately to create text and image features independently. Figure 2.10a shows the overall architecture of a dual encoder model.

One of the most well-known and successful dual encoder models is the CLIP model. Its extensive knowledge about the connections between images and text comes from training on a set of 400 million image-caption pairs collected from the web. The fact that CLIP learns from image-caption pairs makes it easier to construct a large training set for it, as collecting such data is cheaper than having humans classify images into categories. It also makes the model more versatile than pure image classification models. A classification model which is able to output 1000 categories from the famous ImageNet [21] dataset cannot handle new categories. However, CLIP can use its language vocabulary to describe new types of images and is not restricted to a fixed number of classes.

The loss function which enables a dual encoder to learn both text and image embeddings is some form of contrastive loss.

2.5.1 Contrastive Loss

In order to make a model learn a common embedding space for two modalities, such as text and image or text and video, contrastive loss is used. The objective of the training is to push similar data points close together in the common embedding space, and to push unrelated data points far apart. This idea is illustrated in Figure 2.9.

We use the same loss function that was used to train CLIP, the InfoNCE loss function [22].

The InfoNCE loss function is

$$\mathcal{L} = -\frac{1}{B}\sum_{i=1}^{B}\log\frac{f(v_i,t)}{\sum_{j=1}^{B} f(v_j,t)},$$

where $\{v_1,\ldots,v_B\}$ is a batch of videos and t a caption. $f(v_i,t)$ is a probability that v_i and t match. $f(v,t)$ is taken to be $\exp(\text{sim}(v,t)/\tau)$ such that the loss function becomes

$$\mathcal{L} = -\frac{1}{B}\sum_{i=1}^{B}\log\frac{\exp(\text{sim}(v_i,t)/\tau)}{\sum_{j=1}^{B}\exp(\text{sim}(v_j,t)/\tau)},$$

where τ is a temperature parameter. B is the batch size. This is equivalent to cross entropy loss.

In our implementation we follow what is done in the CLIP paper and reformulate the loss function in terms of a new parameter T to get

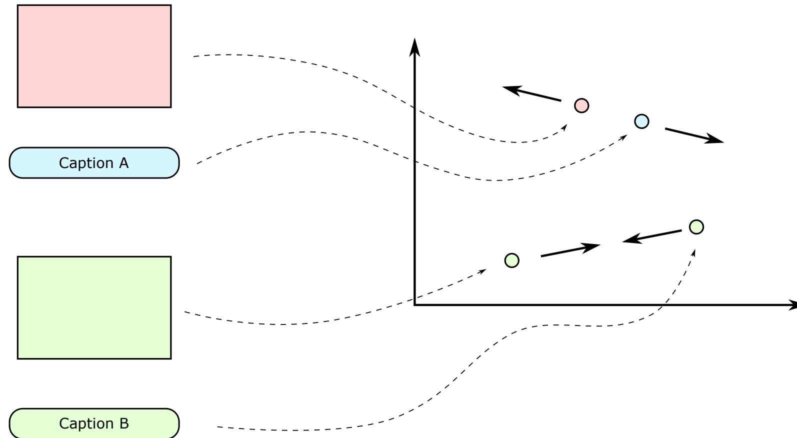


Figure 2.9: Illustration of training using contrastive loss. The representations of a negative pairing (top) are pushed further from each other in the feature space. The representations of a positive pairing (bottom) are pushed closer in feature space.

$$\mathscr{L} = -\frac{1}{B}\sum_{i=1}^{B}\log\frac{\exp(\mathrm{sim}(v_i, t_i)\cdot e^{T})}{\sum_{j=1}^{B}\exp(\mathrm{sim}(v_i, t_j)\cdot e^{T})}. \tag{2.3}$$

In practice, when training the model, we input a batch of B videos and a batch of B captions such that each video belongs to exactly one caption and vice versa. The model outputs embeddings which we turn into a similarity matrix of size $B \times B$. We then divide by the temperature parameter and calculate cross entropy loss over both axes. The resulting two loss values are averaged to get the total loss.

2.6 Cross Encoder Models

A cross encoder is an encoder model which in some way takes in two pieces of different data. In the case of a text encoder, it could be two different sentences. In the case of multi-modal models it could be an image and a sentence. The cross encoder is often used to determine whether the two input data points belong together or to calculate a similarity between the two.

So the task it performs is similar to that of a dual encoder. The dual encoder also takes in two data points and creates a feature representation for each of them, and calculates a similarity score for example using cosine similarity. The main difference between cross encoders and dual encoders is the amount of interaction between the data points they allow. Let us take the example of one image and one caption being input. The dual

encoder treats each of them independently and there is no interaction between them until the cosine similarity is calculated. The cross encoder however, performs cross attention between the two. The attention mechanisms in the cross encoder calculates attention across tokens in both the image and the caption. This allows much more complex interactions to take place.

We are interested in using the cross encoder in a retrieval context, and the two pieces of data we will input are the query and a document. The cross encoder then outputs a similarity between the two. In our case, the query is a text caption, and the document is a video. A cross encoder model takes in both the query and a document at the same time, and outputs a prediction of the relevance of the document to the query. The cross encoder model has the advantage of having access to the query when processing videos. Rather than having to create a representation which should capture as much information about the video as possible, not knowing what features will be useful later, the model can use the query to judge what video content is relevant. Figure 2.10b shows the structure of a cross encoder. Text and video are input to the same encoder, and the output is sent through a similarity head which predicts a similarity score.

In this report, we will combine a dual encoder and a cross encoder by stacking them after each other. The dual encoder is used to extract feature representations of the text query and the frames of the video. The cross encoder is then used to calculate a similarity or relevance score between the text and video representations.

Let the video representation be a matrix of frame feature vectors which have been extracted using the text-image dual encoder

$$V = [v_1, v_2, \ldots, v_{N_F}]^\top \in \mathbb{R}^{N_F \times d_{\text{model}}},$$

and let $Q \in \mathbb{R}^{1 \times d_{\text{model}}}$ be the text query representation. N_F is the number of frames sampled from the video.

The fused features are then

$$U = \begin{bmatrix} Q \\ V \end{bmatrix}.$$

We add a type embedding as well as a positional embedding to the features. The type embedding tells the model what tokens belong to what modality. In our case, we have only two types or categories that we wish to distinguish from each other: the text query and the video document. So, to the text token we add a vector of zeros, while to the video tokens we add a vector of ones. Similarly to the positional encoding, this aids the model in learning the difference between the two types.

The positional encoding in the cross encoder is the same concept as for the plain sequence encoders. We add different positional vectors to each token to help the model understand their relative position. These can take the form of non-learnable vectors such as the ones in Equation (2.1), or learnable vectors. In the case of learnable vectors, the position vectors are learnable parameters of the model and are randomly initialized.

Adding the positional encoding and the type embedding we get

$$U = \begin{bmatrix} Q \\ V \end{bmatrix} + P + T.$$

This is then fed to an encoder and the output is taken as the query token representation in the last layer. This vector is then sent through a similarity function which outputs a similarity score for the query and document. For the similarity calculation, we use a feed forward network with two layers and a ReLU activation function.

Different loss functions can be chosen when training the cross encoder. We will consider two options in addition to the contrastive loss introduced earlier: a pointwise loss function and a listwise loss function.

Pointwise Cross Encoder Loss Function

One option for training the cross encoder is a pointwise loss function. By this we mean that we only use a single text-video pair and a label to calculate loss.

The loss function we use is binary cross entropy.

$$\begin{aligned} z_{\text{out}} &= \text{CrossEncoder}(\text{video}, \text{query}) \\ \text{loss} &= \text{BinaryCrossEntopy}(\text{Sigmoid}(z_{\text{out}}), \text{true label}) \end{aligned}$$

where the binary cross entropy function is

$$\text{BinaryCrossEntopy}(\sigma(\hat{y}), y) = y \cdot \log \sigma(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y}),$$

where $y \in \{0, 1\}$ are the true labels and $\hat{y}$ are the predicted similarities. $\sigma(\cdot)$ is the sigmoid function.

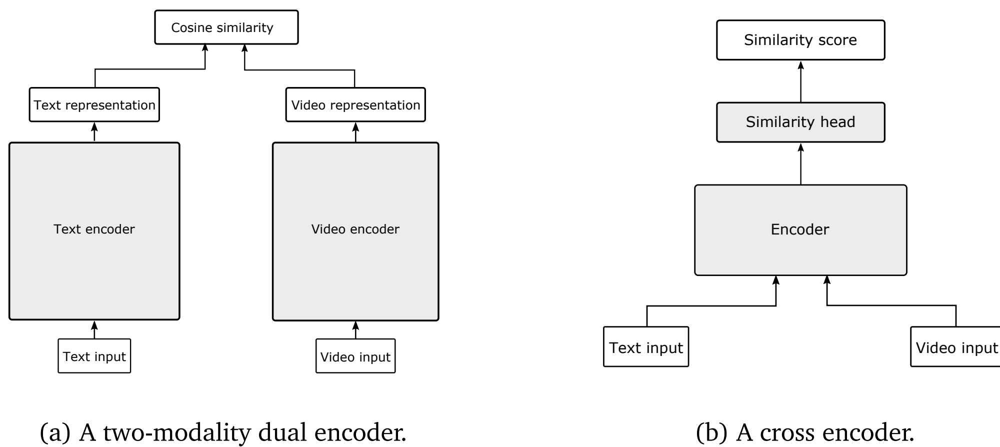
Listwise Cross Encoder Loss Function

Another loss function for the cross encoder which involves more negative examples, is a listwise loss function. When training using a listwise loss function, we show the model a list of negatives for each positive example. The number of negatives per positive can vary.

The loss function is

$$\text{loss} = -\frac{1}{B} \sum_{i=1}^{B} \sum_{j=1}^{n} y_j \log \left(\frac{\exp(\hat{y}_{ij})}{\sum_{k=1}^{n} \exp(\hat{y}_{ik})} \right),$$

where B is batch size, n is the length of the list containing one positive and $(n-1)$ negatives. $\hat{y}_{ij}$ is the predicted similarity for example j in list (data point) i.



(a) A two-modality dual encoder.

(b) A cross encoder.

Figure 2.10: The difference between a dual and cross encoder illustrated.

Chapter 3

Retrieval Models

Here we briefly go over the different models used in later experiments, and explain more concretely how the deep learning models presented earlier can be used for video retrieval. We will test two main categories of models: models for aggregating information across the temporal axis of a video to get a representation for the video as a whole, and cross encoders for predicting a relevance given a query and a video representation.

3.1 Non-Temporal Models

3.1.1 Pooling Aggregation

The main issue we are interested in is how frame features can be aggregated into a useful representation for video retrieval. The simplest aggregation functions we will test are mean and max pooling. These involve taking the mean or the maximum of each dimension across the frame feature vectors. Let V be a matrix of frame feature vectors extracted using CLIP,

$$V = [v_1, v_2, \dots, v_{N_F}]^\top \in \mathbb{R}^{N_F \times d_{\text{model}}},$$

where N_F is the number of frames used. We then use mean or max pooling to aggregate these into a video representation $v_{\text{pool}} \in \mathbb{R}^{d_{\text{model}}}$. When we have representation of dimension d_{model}, we can search directly using ANN search and the CLIP text representations, which also have dimension $d_{\text{model}} = 512$.

3.1.2 A Frame Model

We will call the next search method that we are testing the max frame approach. Here we search among a number of extracted frames in a video dataset, and rank videos based

on the maximum similarity among its frames. We sample N_F frames from each video and represent each frame by a vector $f_i \in \mathbb{R}^{d_{\text{model}}}$. The similarity score of a query and a video is

$$\begin{aligned}\text{sim}(q, v) &= \max\{\text{sim}(q, f) \mid f \in v\} \\ &= \max\{\text{sim}(q, f) \mid f \in \{f_1, \ldots, f_{N_F}\}\},\end{aligned}$$

where q is the query representation, v is a video and f_i are frame representations.

Unlike the other models in this report, this frame model can be a starting point for not only retrieving a video, but also finding the most relevant segment of the video. While the training dataset used in this report contains short videos with a duration of around 15 seconds, segment retrieval is interesting especially for longer videos. This model also has an advantage with videos with changing topics, people or scenes. A longer video containing several scenes or activities will be more difficult to summarize into one feature vector than a short, homogeneous video. Then being able to search among frames might be an advantage over aggregation methods.

3.2 Encoder Models for Temporal Aggregation

The next category of models is encoder models for temporal aggregation. Again, we use CLIP to extract features from frames in the video, giving

$$V = [v_1, v_2, \ldots, v_{N_F}]^\top \in \mathbb{R}^{N_F \times d_{\text{model}}}$$

for each video. We then use each frame representation as a token and input all N_F frames from a single video into an encoder. A positional encoding is also added. The encoder is trained to aggregate the frame features into a single video level representation $v \in \mathbb{R}^{d_{\text{model}}}$. When a d_{model}-dimensional video representation is achieved we can use this for ANN search, with the query representation being taken from the CLIP text encoder. The task of the encoder will in other words be to summarize the frame representations in one vector, reducing the dimensionality of the overall video representation from $N_F \times d_{\text{model}}$ to d_{model}.

We test different encoder architectures, consisting of different numbers of layers and different numbers of attention heads. The loss function used is contrastive loss. We set d_{model} to 512 and the dimensions of the hidden layers of the feed forward networks to 2048.

3.3 Cross Encoder Models

The final model category is different from the aggregation models. Rather than aiming at creating a new video feature from CLIP frame features, the cross encoders take in both video and text and give a similarity score.

In the last few paragraphs, we have seen that a video can be represented in several ways, among them a video level representation $v \in \mathbb{R}^{d_{\text{model}}}$ or a frame level representation $v = [v_1, v_2, \ldots, v_{N_F}]^\top \in \mathbb{R}^{N_F \times d_{\text{model}}}$. We will keep this in mind in the cross encoder section as well, testing the use of both video level and frame level representations.

The cross encoders we create will take in a query token $q \in \mathbb{R}^{d_{\text{model}}}$ and video tokens, either a single token for video level representation or N_F tokens for frame level representation. We then add a positional encoding as well as a type encoding and send query and video through an encoder. The representation for the query-video pairing is taken as the output of the query token in the last layer of the encoder. Then, this d_{model} dimensional vector is sent through a similarity head consisting of a feed forward network which outputs a similarity score.

We have gone over several loss functions which can be used with cross encoders, and we will test pointwise loss, listwise loss and also the contrastive loss function to train our cross encoders.

Chapter 4

Related Work

Here we present some different solutions to the problem of video retrieval found in recent papers. We present a few models and what innovation or idea they are characterized by. Many recent text-video retrieval models use CLIP as a basis due to its demonstrated abilities within text-vision tasks. Several of the models presented here do too. Some of these are alternatives to what we will be doing, while others contain features which could be added for improvement.

4.1 CLIP4Clip: Sequence Models on Top of CLIP

CLIP4Clip [23] are a set of models published in 2021, some of which are very similar to the models described in this report. CLIP4Clip builds different sequential models on top of CLIP, either using an LSTM, an encoder or a cross encoder. In the CLIP4Clip paper, the entire model is trained, including the CLIP weights. Due to time and compute restraints, we will not be training the CLIP model itself.

The models we test are inspired by CLIP4Clip, but we will try more variations of the sequential encoder model, and different cross encoder training variations.

4.2 CLIP2TV: Using Postprocessing for Search

CLIP2TV [24] is a video retrieval model published in 2021 which also employs a video encoder, a text encoder and contrastive loss to implement text-video retrieval.

CLIP2TV also uses dual softmax at inference, which is an example of a technique which is applied during inference. Dual softmax was introduced by the creators of CAMoE [25]. Dual softmax can be seen as a postprocessing technique, as it alters the similarity matrix after it has been calculated, and is used at inference or during both inference and

training.

Given n captions and n videos, we can create an $n \times n$ similarity matrix. Say we want to assign each caption to a video, such that for every caption we want to calculate the probability that it belongs to each of the n videos. We can convert the similarity values into estimated probabilities by taking the softmax over the caption axis and then assign each caption to the video with the highest estimated probability. With this approach we might often end up in a situation where several captions are assigned to the same video, perhaps if the topic of the video is quite general and can fit well with several captions.

However, with the dual softmax method we create a matrix of estimated prior probabilities which allows information exchange between the captions, to adjust for imbalances where some videos are assigned a disproportionate number of captions. The prior probabilities are found by taking the softmax over the video dimension, and the original similarities are weighted by the prior probabilities using the dot product. The final probability matrix is created by taking the softmax over the caption dimension.

CLIP2TV used a symmetrical version of dual softmax loss, where they took the softmax over each axis of the similarity matrix and let the final probability matrix be the elementwise product of these two matrices.

Both CLIP2TV and CAMoE see retrieval improvements using dual softmax. Because we are using the Vespa Engine for retrieval and are restricted to their ANN search, we will not be implementing dual softmax at inference.

4.3 Querybank Normalization

Querybank normalization was introduced in 2021 in [26] to remedy a problem known as hubness. Hubness is the tendency of a few documents being overrepresented in retrieval results. A hub is a point which appears frequently in the k nearest neighbors of other points in the data set, and is common with high-dimensional data [27]. When using querybank normalization, a set of sample queries are collected in a querybank, and their similarity with each document in the corpus is calculated to survey the hubness of the dataset. This information is then used to adjust similarity scores to reduce the effect of hubness during retrieval.

4.4 ViViT: A Pure Video Encoder

The Video Vision Transformer (ViViT) was published in 2021 by Google Research [28]. This is an example of a model which is created for video, rather than adapting an image model. While heavily inspired by ViT [17], the ViViT architecture is video specific and created to extract both temporal and spatial information from videos.

There are a few variations of the ViViT architecture. The simplest version is a video

encoder which takes in 3D tubelets from a video as tokens. This is equivalent to the image patches used as tokens in the CLIP image encoder, but with each token also having a time dimension.

This approach of essentially creating a new architecture limits the possibility of using weights from other pre-trained models, and causes ViViT to be a compute demanding model to train. Still, the ViViT authors were able to use pre-trained weights from ViT to help initialize the video encoder.

ViViT is trained on classification tasks, and does not contain a dual encoder setup with a text encoder, but is an example of how the vision encoder in a dual encoder video retrieval model could have been.

4.5 Mixture of Embedding Experts: Using Several Modalities

The model called mixture of embedding experts (MEE) [29] is a video retrieval model which utilizes multiple modalities and is able to handle missing modalities. The model uses video appearance, video motion, face features and audio to calculate a similarity between a text query and a video.

MEE is a type of mixture of experts model. As the name suggests, this is a kind of model which is built by component models which are experts in one topic or modality. The dual encoder setup is reminiscent of this as it combines a text and an image model, but the MEE setup of course utilizes more of the input data. Adding experts such as a face model in addition to the vision model might seem superfluous, but as face features and movements are very important for what meaning humans give a video, such additional experts might extract important meaning from a video. MEE is an example of how a video retrieval model could be extended to use new aspects of the input data.

Chapter 5

Data

5.1 The MSR-VTT Dataset

We use the MSR-VTT dataset [2] for training and testing. The dataset contains 10000 videos, each with 20 captions, for a total of 200000 video-caption pairs. We use this dataset because it is extensive for a video dataset, and because it is one of the most commonly used benchmarks in the video retrieval literature.

There is a total of 41.2 hours of video in the dataset, giving an average video clip length of just under 15 seconds. The videos cover a range of topics. The videos were collected from the web using 257 different queries. Collecting videos from the web gives different and more varied and non-homogeneous videos than using actors in a controlled environment, as some datasets do. Figure 5.1 shows frames from a few videos from the dataset with sample captions.

It is worth noting that the captions do contain some noise in the form of repeated captions and grammatical errors. Also some captions seem to describe audio content, which is not available to the model. Table 5.1 shows some examples of problematic captions from the dataset and lists the problems the captions exemplify.

Because the MSR-VTT training and test sets are frequently used in the video retrieval community, we do not perform any data cleaning to remedy the mentioned problems, as we want to be able to make fair comparisons with other models.

Caption	Problem
"people wearing headphones and tlaking"	Spelling error.
"a young woman is discussing the social pressure she s under"	Describes audio content.
"a person is playing"	Very general, can fit many videos.

Table 5.1: Examples of problem captions.

Figure 5.1: Examples of videos and captions from the MSR-VTT dataset.

We will be referring to different splits of the dataset. The *train 7k* and *val* splits are the training and validation split published by the creators of MSR-VTT. We will be using these for tuning hyperparameters. For training and testing, we will be using the *train 9k* and *test 1k* splits published by Yu et al. [30].

5.2 Data Preprocessing

The CLIP image encoder takes inputs of size (224, 224), and thus the frames sampled from each video need to be rescaled to this size.

There are several ways in which this can be done. Three options are cropping, padding and stretching.

Cropping In this option we crop the video frame to a centered square. The strength of this method is of course that it keeps the proportions of the video content unchanged, and that it does not introduce potential misleading information. The weakness is that it removes information from the frame. When using the center crop, we rely on the assumption that most of the important information is contained in the center of the frame.

Padding This option involves padding the smallest dimension to match the size of the largest dimension. We would like to pad without adding any information to the frame, and can for example pad using black or white pixels. This option keeps the proportions of the video unchanged, but could potentially mislead the model as it is not trained on such data, and new edges could be interpreted as having a meaning or carrying information.

Figure 5.2: Example of a video frame which has been cropped and resized.

Stretching The option of stretching means to stretch the smallest dimension out to match the largest one. This keeps all the information in the frame, but changes the proportions. With varying dimensions from video to video, the proportions will change in different ways for different videos.

We choose to use the cropping option.

To get an impression of the information available to the model, an example cropped and resized frame is shown in Figure 5.2. This might give us an idea of the difficulty level of the image understanding task. As the example frame shows, the image is pixelated, but is clear enough that both a human and a machine learning image model should be able to capture the meaning of the scene. In this specific case, it seems that the center crop has not left important information out.

Chapter 6

Method

Here we briefly detail the setup with which experiments were carried out. Because we consider reasonably small models and use a small dataset, the training was mostly not very time consuming and the models could be trained on a laptop with an NVIDIA Geforce RTX GPU. The time taken for training the smaller models varied from minutes to a few hours. Some of the larger cross encoder models took several hours and were trained on the IDUN HPC Cluster [3], where NVIDIA V100 or A100 GPUs were used. The search experiments are carried out using the Vespa search engine [12]. The Python library pyvespa [31] provides a Python API and was used for creating and evaluating search applications using different models.

6.1 Optimization

6.1.1 Optimizer and Hyperparameters

The models were trained using the Adam optimizer with a cosine learning rate schedule. The Adam epsilon value was chosen to be 10^{-6}, which is the same as used in training CLIP. When training cross encoders, the epsilon value was increased to 10^{-5} for stability.

A batch size of 128 was chosen. We want a large batch size because the larger the batch size, the better and more varied negative examples the model is likely to be exposed to. In the training of the CLIP model, a very large batch size of 32768 was used [1]. The authors of CLIP4Clip also demonstrated that a larger batch size was to be preferred over a very small batch size [23].

Because hyperparameter tuning using an extensive grid search would be very time consuming, we choose hyperparameters based on a few smaller search experiments. This assumes that the effect of one hyperparameter does not depend very much on the value of another hyperparameter.

T	R@1	R@5	R@10	P@10	MRR@10	nDCG@10
0	0.361	0.643	0.739	0.074	0.482	0.543
0.1	0.364	0.647	0.748	0.074	0.485	0.547
1	0.382	0.668	0.759	0.076	0.504	0.565
5	**0.410**	**0.712**	**0.802**	**0.080**	**0.540**	**0.603**
10	0.380	0.699	0.800	**0.080**	0.515	0.584

Table 6.1: Search results on the validation set using different temperature values and a single head, single layer video encoder on top of CLIP. Models trained for 10 epochs.

Temperature

The loss function contains a temperature parameter. In order to decide the value of this parameter, we test a grid of temperature values on a single model. We train an encoder with one layer and one attention head for 10 epochs on the MSR-VTT train-7k split. The encoder weights are initialized randomly. We set both the weight decay and the initial learning rate to 10^{-4}. We then evaluate search on a validation set, giving the results in Table 6.1. The temperature is given in terms of a parameter T, which we introduced in Equation (2.3). We choose the best performing value, and set the T parameter to 5.

We also note that the temperature parameter affects the search performance quite a lot. The fact that the results are so sensitive to changes in temperature, reflects the fact that the choice of loss function is crucial for search performance. The large variations in retrieval metrics suggest that there might be room for innovation and improvement of the loss function. We are interested in maximizing metrics such as recall and MRR, but we are not able to use these explicitly in the loss function as they are not differentiable. The loss function is an attempt at creating a differentiable function which will be minimal when search metrics are optimal.

The temperature parameter changes how the function penalizes negatives. A small temperature value penalizes the most difficult negatives heavily, such that quite similar data points become more spread out in feature space [32]. A large temperature will create a loss function which prioritizes increasing the distance between very different data points, lumping more similar ones closer together. In Equation (2.3) we reparameterized the loss function in terms of the parameter T, and we give our results in terms of this new parameter. A small temperature will lead to a large T value and vice versa. From Table 6.1, we can see that a small temperature (large T) gave the best results in our tests. For a classification problem, it would be beneficial to prioritize separation between very different points, creating clusters of similar points. However, we are not interested in a classification problem, rather we are interested in being able to discriminate between very similar points, and thus we are interested in a quite uniform distribution of points in feature space. This could be the reason why a quite low temperature value was best for performance.

Note that we use the entire train-9k training set in our actual experiments. For deciding hyperparameters, we trained on the train-7k set and tested on the validation set. For the final evaluation, we will train on train-9k and test on the test-1k set.

Weight Decay

Next, we test different values of optimizer weight decay. Again we use a single layer, single head encoder, an initial learning rate of 10^{-4} and a cosine learning rate schedule. Search metric results are shown in Table 6.2. We set the T parameter value to 5, the best performing value from the last section. Based on the values in the table, we choose a weight decay value of 10^{-4}.

Weight decay	R@1	R@5	R@10	P@10	MRR@10	nDCG@10
0	0.408	0.711	**0.802**	**0.080**	0.538	0.602
10^{-4}	**0.410**	**0.712**	**0.802**	**0.080**	**0.540**	**0.603**
0.1	0.408	0.711	0.801	**0.080**	0.538	0.602
0.2	0.405	0.681	0.765	0.077	0.523	0.581
0.5	0.398	0.677	0.762	0.076	0.517	0.576

Table 6.2: Search results on the validation set using different weight decay values and a single head, single layer video encoder on top of CLIP. T is set to 5. Models trained for 10 epochs.

Learning Rate

Finally, we choose a value for the initial learning rate. We use a single head, single layer encoder like before, and set weight decay to 10^{-4} and T to 5. The search results on the validation set are shown in Table 6.3. Based on these results, we choose a learning rate of 10^{-5}. The learning rate does have some effect on the results, but none of the learning rates tested were detrimental to performance. Choosing a too large learning rate was the worst option tested, and when training for 10 epochs, a learning rate as small as 10^{-6} gives good performance.

Learning rate	R@1	R@5	R@10	P@10	MRR@10	nDCG@10
$1 \cdot 10^{-6}$	0.432	0.723	0.809	**0.081**	0.557	0.618
$5 \cdot 10^{-6}$	0.432	0.723	0.809	**0.081**	0.557	0.618
$1 \cdot 10^{-5}$	**0.433**	**0.731**	**0.814**	**0.081**	**0.560**	**0.622**
$5 \cdot 10^{-5}$	0.428	0.725	**0.814**	**0.081**	0.555	0.618
$1 \cdot 10^{-4}$	0.410	0.712	0.802	0.080	0.540	0.603

Table 6.3: Search results on the validation set using different learning rate values and a single head, single layer video encoder on top of CLIP. T is set to 5. Models trained for 10 epochs.

6.2 Evaluation Method

We report recall at 1, 5 and 10, as well as precision, MRR and nDCG at 10. We choose to limit the number of retrieved documents to 10 as the test dataset is quite small,

consisting of only 1000 videos, and there is only one relevant video per test query. For the search result to have any value or utility, the one relevant video should be ranked at 10 or higher. If ranked lower than 10, the exact ranking is not as relevant. MRR and nDCG are good metrics for the overall performance of a model, while the recall metrics give a more intuitive way of imagining an average search result. Recall is also the main metric reported in the recent video retrieval literature, and is thus useful for comparison with models created by others. Because of the amount of noise in the dataset, $R@5$ might be a better metric than $R@1$.

The evaluation of the models is carried out by querying the search application using a set of test queries. For each test query, we also need a list of relevant videos. We use the MSR-VTT test-1k dataset for evaluation. This consists of 1000 videos with 20 captions describing each video. We create our list of test queries by collecting all captions in the dataset. For each query, we define the video it described as being the only relevant video.

Chapter 7

Experiments and Results

In this section we will test the different models described in Chapter 3. We are interested in how the different methods compare, as well as how each method can be optimized in terms of architecture and training. We will also attempt to explain differences in performance between the models. Finally, we will connect the results from our experiments to the research questions posed in Chapter 1 and attempt to answer these. The models are trained for 10 epochs using the hyperparameters selected in Chapter 6 unless otherwise stated.

7.1 Pooling Aggregation

To create a baseline to compare our sequential models to, we first test the aggregation methods that do not add any parameters and thus do not require any further training. In this category, we test mean and max pooling.

We embed text queries using the CLIP text encoder. Frame representations are created by sampling 12 frames uniformly from each video and embedding each frame using CLIP. Then, video level representations are taken as the mean and max pooling aggregations of these.

The search metrics for these aggregations are reported in Table 7.1. As we can see, the CLIP model applied in this zero-shot manner gives quite good retrieval results, with mean pooling achieving an R@1 of 0.317, an R@5 of 0.552 and an R@10 of 0.649. Mean pooling performs better than max pooling, and will be the baseline with which we compare our sequential models. While these pooling representations do aggregate information over the temporal axis of the videos, they do not contain any information about the order of frames.

The mean length of videos in the dataset is around 15 seconds. Based on a few random samples, there are both quite homogeneous videos and more varied videos in terms of

their visual content. The parameter-less pooling methods would probably work best for short, homogeneous videos where sequential information is secondary. Although they are not able to take sequential information into account, they may understand a video quite well based on scenes, people and objects. Ideally, we would test a collection of different datasets and investigated how the type, length or homogeneity and videos affects how the different models and aggregations compare.

Model	R@1	R@5	R@10	P@10	MRR@10	nDCG@10
Mean pooling	0.317	0.552	0.649	0.065	0.418	0.473
Max pooling	0.195	0.398	0.489	0.049	0.281	0.330

Table 7.1: Search metric results for non trainable aggregation methods.

7.2 The Max Frame Model

Although mean and max pooling do not introduce new parameters or consider the sequence of frames in a video, they do create video level features by aggregating information over the temporal dimension. The next model we are testing does not use video level features, but rather searches among the frame level features and scores the relevance of each video using its most relevant frame.

We test the max frame model using different numbers of frames per video. The retrieval results are shown in Table 7.2. With enough frames, the max frame method performs on par with mean pooling. It does outperform the max pooling strategy, confirming that an element-wise max pooling operation is not a good aggregation method in this setting. The retrieval metrics show that a lot of value can be added by adding a second frame instead of using only the middle frame of the video.

No. frames	R@1	R@5	R@10	P@10	MRR@10	nDCG@10
1	0.240	0.453	0.545	0.055	0.331	0.382
2	0.280	0.506	0.606	0.061	0.377	0.432
4	0.300	0.533	0.632	0.063	0.400	0.455
12	0.314	0.548	0.643	0.064	0.413	0.468

Table 7.2: Search metric results for the max frame model.

7.3 Temporal Encoder Models

Now we move on to testing the first type of sequential model we are using, namely video encoders. Unlike the pooling and max frame methods, our video encoders are able to access and use the order of frames in a video. Here we are interested in investigating whether using this sequence information improves search, and what encoder architecture is suitable for the task. When listing results we will refer to model structure by names such as 1L8H, which denotes a model with one layer and eight attention heads.

7.3.1 Single-Head Attention Models

We test a range of different transformer encoder architectures as the sequential model on top of the CLIP dual encoder, and begin with one of the simplest possible architectures. We test single-head encoders with varying depth to check whether such models can outperform the mean pooling aggregation, and how the depth affects performance. The results are shown in Table 7.3.

The search metrics show a large improvement over the baseline methods, with MRR@10 improving from 0.418 in the mean pooling model to 0.502 in the best single-head encoder model. We notice that the number of layers has a large effect on performance, with each additional layer reducing performance on all metrics.

Model	R@1	R@5	R@10	P@10	MRR@10	nDCG@10
1L1H	0.389	0.653	0.752	0.075	0.502	0.562
2L1H	0.378	0.647	0.746	0.075	0.493	0.553
4L1H	0.363	0.637	0.740	0.074	0.479	0.542
6L1H	0.351	0.629	0.733	0.073	0.469	0.532
8L1H	0.337	0.618	0.729	0.073	0.457	0.522
16L1H	0.318	0.602	0.720	0.072	0.438	0.505

Table 7.3: Retrieval results on the MSR-VTT test-1k dataset. Models trained on the 9k training split for 10 epochs.

7.3.2 Multi-Head Attention Models

Next we move our focus to multi-head attention models. The rationale behind using multiple attention heads is that several types of attention or feature extraction can be carried out in parallel. We investigate whether this possibility improves ranking.

Single Layer

We investigate the effect of adding more attention heads to a single layer encoder model. The results of testing search using different single layer, multi-head architectures are shown in Table 7.4.

The results of the different architectures are very similar, and it is not possible to detect any trend. It seems that for this task, the encoder is able to extract approximately the same information with different numbers of heads. The usage of several different linear projections in parallel prior to the attention mechanism being applied in the models with more heads has not given the model an advantage over the models with fewer heads.

Model	R@1	R@5	R@10	P@10	MRR@10	nDCG@10
1L1H	0.389	0.653	0.752	0.075	0.502	0.562
1L2H	0.391	0.652	0.753	0.075	0.503	0.563
1L4H	0.389	0.654	0.753	0.075	0.503	0.563
1L8H	0.390	0.654	0.754	0.075	0.502	0.563
1L16H	0.388	0.654	0.753	0.075	0.502	0.562

Table 7.4: Retrieval results on the MSR-VTT test-1k dataset. Models trained on the 9k training split for 10 epochs.

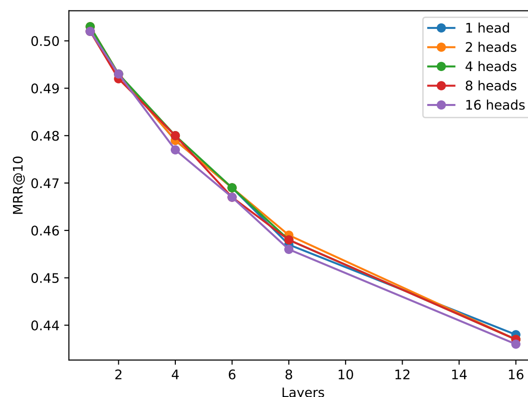


Figure 7.1: MRR@10 for encoders with different numbers of layers. Note that the vertical axis does not start at zero.

More Multiple Layer Models

We test an array of different encoder architectures, including both multiple heads and multiple layers. We plot the MRR@10 for models with different numbers of layers and heads in Figure 7.1. The figure confirms what the single-layer data suggested: the number of heads does not influence performance in any meaningful way. The minimal single-head, single-layer encoder performs as well as multi-head, single-layer models, and better than all tested multi-layer models. Again, we also see that adding more layers consistently decreases performance.

The poorer performance of larger models could be caused by a lack of data, as larger models have more parameters, and need more data for training. We therefore move on to testing the effect of weight initialization.

The Effect of Initializing Encoder Weights

If the poor performance of the large encoder models is caused by a lack of training data, we hypothesize that initializing their weights using a similar model might improve search. We do of course not have any video encoders which we can borrow weights from,

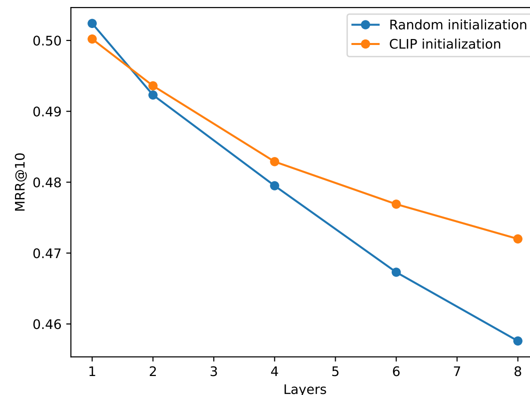


Figure 7.2: MRR@10 for models with random initialization or CLIP text encoder initialization with different numbers of layers.

but we do have an encoder which is trained on sequential data: the CLIP text encoder. This text encoder has 8 heads in each layer, and 12 layers. We train a set of 8 head models initialized from the first layers of the CLIP text encoder.

The difference between the performance of random initialization encoders and the CLIP text encoder initialized encoders is clearly seen in Figure 7.2, showing MRR@10 for both sets of models. For all multi-layer models, the weight initialization improves performance. The deeper the model, the more pronounced the effect is.

This supports the hypothesis that the training set is not large enough for the deeper models, as the head start provided by weight initialization improves performance.

7.4 Re-ranking Using a Cross Encoder

In this section we will test whether re-ranking our search results using a cross encoder improves ranking.

7.4.1 Training Cross Encoders

Just like the video encoder models, the cross encoder has to be trained on another task than actual retrieval. We train cross encoders to take in a text representation and a video representation and output a similarity score between the two. An important part of training any machine learning model is choosing an appropriate loss function. For the cross encoder, we consider three different loss functions: pointwise loss, listwise loss and contrastive loss. We test both cross encoders taking in video level features and cross encoders taking in frame level features.

Pointwise Loss

In pointwise loss training, the model is shown one pair of text and video representations at a time. We can use different ratios of positive to negative examples in the training set. When training, we set a rate of positive examples to show the model, and randomly feed the model matching or non-matching queries and videos during training according to this rate.

A range of rates of positive examples is tested to choose an appropriate rate. To evaluate the different rates, we look at the separation between the similarity scores predicted for true negatives and true positives.

Figure 7.3 shows histograms of sigmoids of predicted similarities for true positives and true negatives on the validation set using video level models. The graphs show that the separation between the two groups is highly dependent on the positive rate. A 0.7 positive rate gives quite good separation with few true negatives being predicted high similarities, and we choose to use this for training.

Figure 7.4 shows histograms of sigmoids of predicted similarities for true positives and true negatives on the validation set using frame level models. The performances of the 0.5 and 0.7 rates are quite similar. We choose to use 0.7 for the frame level models as well. Choosing the same rate as for the video level models also gives consistency in the training, and a fairer comparison between video level and frame level.

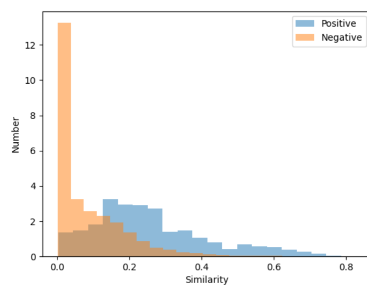
7.4.2 Re-Ranking with Cross Encoders

We test two variations of the cross encoder. The video level version takes in the video representations from one of the video encoders created earlier. The frame level version takes in frame representations directly from CLIP. For the video level cross encoder, we use the single-head, single-layer encoder to create the features the cross encoder takes in.

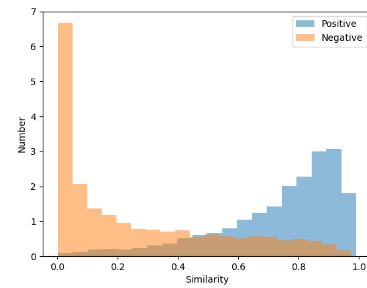
Pointwise Loss

We use the trained cross encoders to re-rank the top 10 results from the first phase ANN search. The results for the video level and frame level models are shown in Table 7.5. The results are very poor. In fact, they are only slightly better than a random re-ranking would be. The video and frame level models perform similarly, with the frame level model having a slight edge.

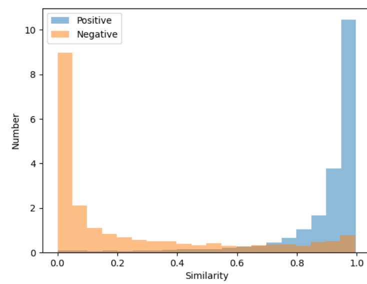
The clear separation between the similarity scores of the true positives and true negatives shown in Figures 7.3 and 7.4 shows that the cross encoders do learn something from the training. And with a suitable ratio of positive to negative examples, the pointwise cross encoders perform well at the task which they are trained at. Because of this large discrepancy in performance between the classification task and the retrieval task, we



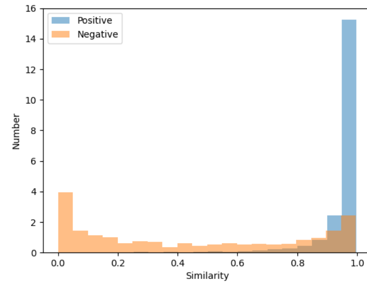
(a) 0.1 positive rate.



(b) 0.5 positive rate.

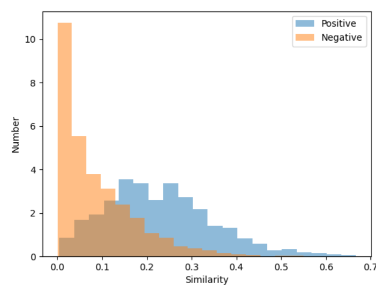


(c) 0.7 positive rate.

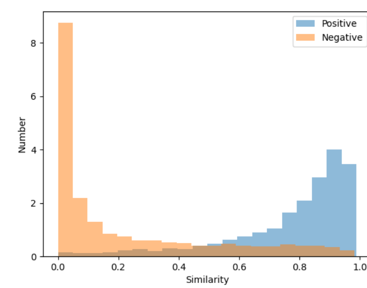


(d) 0.9 positive rate.

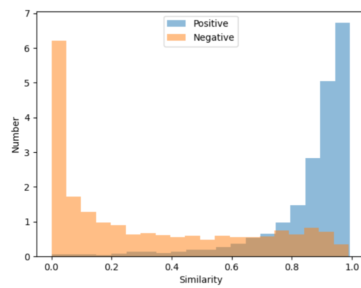
Figure 7.3: Histogram over sigmoid of similarities for true positive and negative samples, for different rates of positive examples. The vertical axis shows a density number. Validation set used. Video level models trained for 10 epochs on train-7k.



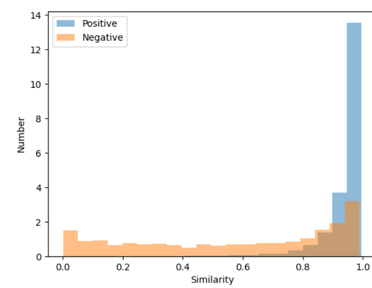
(a) 0.1 positive rate.



(b) 0.5 positive rate.



(c) 0.7 positive rate.



(d) 0.9 positive rate.

Figure 7.4: Histogram over sigmoid of similarities for true positive and negative samples, for different rates of positive examples. Validation set used. Frame level models trained for 10 epochs on train-7k.

hypothesize that we have chosen a poor training strategy for the cross encoders. In fact, the two tasks are quite different in terms of the challenges the data poses. In an attempt to improve the re-ranking capabilities of our cross encoder models, we train them using a new strategy for choosing negative examples in a later section.

Model	R@1	R@5	R@10	P@10	MRR@10	nDCG@10
Video 1L1H	0.102	0.456	0.751	0.075	0.257	0.372
Frame 1L1H	0.118	0.459	0.751	0.075	0.270	0.381

Table 7.5: Retrieval results using cross encoders for re-ranking. Pointwise loss functions.

Listwise Loss

A pointwise loss function only gives the model one example at a time, making the information provided to the model during training limited. We test whether performance can be improved by introducing a listwise loss function. A listwise loss function takes in a text caption and the video belonging to the caption, along with a list of non-relevant videos. We test lists of different lengths. Re-ranking results are shown in Table 7.6.

The table shows that the listwise loss function leads to a great improvement from the pointwise loss function, but without achieving the same level of performance as the first phase ranking. The longer lists of 30 negative examples give better performance than using only 10. The video level models perform much better than the frame level models. One reason for this could be that the frame level models have too many learnable parameters compared to the dataset size. Another possibility is that the architecture is too small for the frame level approach. After all, the video level features are the output from another encoder, so they are the result of a larger model.

Model	List len.	R@1	R@5	R@10	P@10	MRR@10	nDCG@10
Video 1L1H	10	0.258	0.592	0.751	0.075	0.400	0.484
Frame 1L1H	10	0.179	0.502	0.751	0.075	0.324	0.424
Video 1L1H	30	0.274	0.607	0.751	0.075	0.415	0.495
Frame 1L1H	30	0.195	0.510	0.751	0.075	0.336	0.432

Table 7.6: Retrieval results using cross encoders for re-ranking. Listwise loss functions.

7.4.3 Using a Contrastive Loss Function

The next loss function we are testing is the same loss function used to train the encoder models in Section 7.3, contrastive loss. Retrieval results are shown in Table 7.7. We test two 1L1H encoders, with differing batch sizes to see if a larger batch size helps, but the batch size of 256 actually performs worse than the batch size of 128. We also test 4L8H models, both with random initialization and initialized from the CLIP text encoder. The 4L8H model performs slightly better than the 1L1H cross encoder, suggesting that the

1L1H architecture might be too small for the task. The 4L8H cross encoder with weights initialized from the CLIP text encoder is our best performing cross encoder. But with an R@1 of 0.341 and an MRR@10 of 0.468, it still does not reach the ranking performance of the first phase ranking.

Model	Batch size	R@1	R@5	R@10	P@10	MRR@10	nDCG@10
Video 1L1H	128	0.301	0.615	0.751	0.075	0.434	0.509
Video 1L1H	256	0.243	0.582	0.751	0.075	0.387	0.473
Video 4L8H	128	0.306	0.630	0.752	0.075	0.443	0.517
Video 4L8H, init.	128	0.341	0.641	0.752	0.075	0.468	0.536

Table 7.7: Retrieval results using cross encoders for re-ranking, contrastive loss.

7.4.4 Training Re-Ranking Models Using Mined Negatives

The re-ranking capabilities of some of the cross encoders proved quite poor, especially for the models trained with a pointwise loss function. As the training achieved low loss on both training and validation sets, we change our training to better capture the actual problem the model will be tasked with. Re-ranking requires the model to be able to capture fine nuances of the connection between text and video. Being provided with the top k documents from the first phase, deciding the detailed ranking of very similar videos is demanding. We therefore create a training set which is more suitable to the task. We still use the same data set, but pick the negative examples more intelligently, and no longer at random. The negative examples are picked among the top 10 videos returned by the phase one search for each query.

The retrieval results for the pointwise loss function with mined negatives are shown in Table 7.8. The results when using mined negatives are worse than when using random negatives.

Model	R@1	R@5	R@10	P@10	MRR@10	nDCG@10
Video level	0.079	0.406	0.751	0.075	0.231	0.351
Frame level	0.083	0.411	0.752	0.075	0.233	0.352

Table 7.8: Retrieval results using cross encoders for re-ranking, with adjusted training set. Pointwise loss.

Next, we test the listwise loss function. The retrieval metric results are shown in Table 7.9. Again, the mined negatives lead to much worse retrieval results. This is even more apparent here with the listwise loss which performed quite well using random negatives.

Model	R@1	R@5	R@10	P@10	MRR@10	nDCG@10
Video level	0.107	0.436	0.752	0.075	0.257	0.371
Frame level	0.091	0.405	0.751	0.075	0.239	0.356

Table 7.9: Retrieval results using cross encoders for re-ranking, with adjusted training set. Listwise loss.

7.5 Discussion of Results and Research Questions

The first research question we posed in the introduction of this report was whether pre-trained image models could be useful for video search. And our experiments show that even when used in a zero shot manner with mean pooling, the CLIP model performs well on video retrieval. The later encoder experiments also show that by adding additional model architecture and training, the potential of a pre-trained image-text model can be utilized even better. Mean pooling seemed to summarize the contents of the video well. Max pooling, however, led to poor performance. The cause of this could be that while mean pooling summarizes every frame in the video, each element of the max pooling vector mostly reflects a single frame. All other frames must have a lower value for this index, but max pooling incorporates very little information about other frames. This could explain why it does not perform as well as mean pooling. A method involving a max operation which did work well, was the max frame approach, in which we searched among frames. As expected, the number of frames available to the search engine affected performance greatly. The biggest increase in performance was seen when going from using only one frame per video to using two. When adding more frames, performance continued to improve, and at 12 frames per video, the max frame method matched mean pooling in terms of search performance. The mean and max pooling aggregations were also based on 12 frames per video.

The next research question we set out to answer was whether we could improve on the trivial models by using encoders for temporal aggregation. And the numbers clearly show that performance indeed can be improved by adding a sequence model with learnable parameters. We hypothesized that for this relatively simple task, a simple model would be sufficient. And this was confirmed by our experiments in which the single layer encoders were the best performers. The performance of the encoders decreased as more layers were added. This could be because of a lack of data, or could indicate that the single layer encoders are sufficiently complex for the task. When initializing encoders using weights from the CLIP text encoder, performance improved for the multi layer models, indicating that a lack of data might be an issue. So, without a larger training set, we are not able to determine whether a single layer model is too simple.

The results were not affected much by the number of heads in the encoder models. In [33], Michel et. al. studied the effect of removing attention heads from already trained multi head models and found that a large percentage of attention heads could be removed without much effect on performance. In the case of our models, it would be interesting to do something similar and investigate the weighting of different attention heads in our multi head models. It would be interesting to see whether the multi head models utilize all heads or if one or a few heads are given most of the weight, almost creating a single head model in disguise.

Finally, we were interested in investigating whether re-ranking with a cross encoder could improve ranking results. Here, our hypothesis was that due to being able to contain more complex query-video interactions, the cross encoders would be able to rank videos more precisely. When it came to training cross encoders, we tested several variations. The loss function chosen had a great impact on performance. While the pointwise

loss function was able to produce similarity scores which separated true positive and true negative videos quite well when considering the whole dataset, this did not translate into good search performance. The cause for this is probably that when trained using the pointwise loss function, the model is simply not getting enough information to perform the task well. Perhaps a larger training set or training for longer could remedy this to some extent, but the pointwise loss function is not able to compete with the listwise and contrastive loss functions which feed the model with more information in each training step. The listwise loss function and the contrastive loss function performed much better than the pointwise loss function, but did not quite reach the level of the first phase ranking. These loss function show the model several videos at a time during training. Intuitively, when using the listwise function the model is tasked with finding the positive among a list of negatives, while with the contrastive loss function the model has to match the correct video to the correct caption. This leads to a better fine-tuning of the model's capabilities that with the pointwise loss function.

We attempted to tailor the model training to the re-ranking task by creating a dataset of mined negatives found using the first phase search ranking. However, using mined negatives actually led to worse performance overall. A reason for this could be that the task of separating the mined negatives from the positive is a too difficult task, causing the model to learn slowly and not enough. When using random negatives, the model is exposed to a much larger range of examples giving a more general knowledge, and evidently this makes the model better prepared even for re-ranking.

One reason why the cross encoders underperformed in comparison with the simplest sequence encoder models could be that the cross encoders introduce more learnable parameters, which requires more data.

Also, the cross encoder task is more complex than the task of the sequential encoders, and they might require a larger architecture. For the temporal encoders, we saw that the single layer models were the best performers, and even when initializing weights the larger models did not reach the same performance. In the cross encoder case, the situation was different. The best cross encoder was the initialized 4L8H model, suggesting both that the cross encoder task requires a larger model and that the training set is not large enough to utilize the full potential of the cross encoder architecture.

Chapter 8

Conclusion

8.1 Summary

In this report we have used a pre-trained image-text model to investigate models for video retrieval. Our base model, CLIP, proved its extensive image-text understanding by giving good video retrieval results even when mean pooling was used and no further training performed.

We were able to improve on the trivial mean pooling by using encoder models for temporal aggregation. MRR@10 was improved from 0.42 for mean pooling to 0.50 for the single layer encoders. As hypothesized, a small encoder model was sufficient for achieving good results. However, we also found that the smaller encoder model outperformed multi-layer encoders, likely due to a difference in need for training data. Initializing the multilayer encoders using weights from the CLIP text encoder improved performance, demonstrating that borrowing weights from a related model can be useful and make up for some lack of data. Encoder models, despite not having a sequential nature, yet again proved successful in a sequential task, as they have done for example for text understanding.

A selection of cross encoder architectures and training variations were also tested. There were large performance differences among the cross encoders, and none reached the ranking level of the sequential encoder first phase ranking. Still, we were able to investigate how some variations in training and data affected the result. In conclusion, we found that listwise loss and contrastive loss far outperformed pointwise loss. As for the sequential encoder models, the training data set was not sufficient to realize the full potential of multilayer encoder models. The cross encoder video models performed better than or on par with the frame level models, while being much faster to train and faster during inference due to their relative simplicity.

The best results achieved in this report were with the single layer encoders. Their performance was almost indistinguishable from each other. One of the main takeaways from

this report is just how powerful the single head, single layer encoder is. The best model measured by R@1 achieved an R@1 of 0.391, an R@5 of 0.652, an R@10 of 0.753 and an MRR@10 of 0.503. The current state-of-the-art result to our best knowledge, is an R@1 of 0.550, an R@5 of 0.804 and an R@10 of 0.868, achieved by the model HunYuan_tvr [9] in 2022.

8.2 Further Research

There are several possibilities for further exploration within the topic of encoder models for text-video retrieval.

A natural extension of this project would be to try data augmentation techniques. This might improve performance of the models which did not realize their full potential using only the MSR-VTT train-9k dataset. Still, data augmentation might not be the best choice in this case, as it might not a have a large effect on the outputs from CLIP. CLIP has been thoroughly trained on a very large dataset, and thus small augmentations of inputs are not likely to translate into very different outputs. This might mean that the inputs to our temporal encoders would not benefit from the data augmentation.

Another possibility is to use a larger dataset to see what models are restricted by data and what models are restricted by architecture or training method. Another interesting data-related test would be to use different types of datasets to see whether the models are suited to different types of data. For instance, using longer videos or videos relating to specific applications.

Several of the techniques mentioned in Chapter 4, such as dual softmax or querybank normalization could be used to try and improve results.

It is also likely that results could be improved by using the CLIP ViT-B/16 model instead of the CLIP ViT-B/32, giving a patch size of 16×16 instead of 32×32, provided overfitting could be avoided. Additionally, one could train the entire model. In this report, we did not fine-tune the CLIP weights, but rather used the CLIP image and text encoders as is. Training either the CLIP image encoder only or both the image and the text encoder would be very time consuming, but is likely to lead to better retrieval.

Additionally, it would be interesting to investigate the effect of loss functions further. For the encoder models, the temperature parameter changes the loss function, and has a very large effect on results. For the cross encoder, different loss functions were tested and led to even more variations in performance. This shows that there is much to gain in choosing appropriate loss functions for retrieval, and further investigation into how loss functions should be chosen and optimized could lead to useful insights.

Another extension of this project could also be segment retrieval. We only retrieved entire videos in this report, but the same techniques could be applied for retrieving relevant segments or moments from videos.

Bibliography

[1] Alec Radford et al. "Learning transferable visual models from natural language supervision". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8748–8763.

[2] Jun Xu et al. "Msr-vtt: A large video description dataset for bridging video and language". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 5288–5296.

[3] Magnus Själander et al. *EPIC: An Energy-Efficient, High-Performance GPGPU Computing Research Infrastructure*. 2019. arXiv: `1912.05848 [cs.DC]`.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[5] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[6] Christian Szegedy et al. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[7] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[8] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[9] Shaobo Min et al. "HunYuan_tvr for Text-Video Retrivial". In: *arXiv preprint - arXiv:2204.03382* (2022).

[10] Qiang Wang et al. "Disentangled Representation Learning for Text-Video Retrieval". In: *arXiv preprint arXiv:2203.07111* (2022).

[11] Nick Craswell et al. "Overview of the TREC 2021 deep learning track". In: (2022).

[12] *The Vespa Engine*. URL: `https://vespa.ai/`.

[13] Yu A Malkov and Dmitry A Yashunin. "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs". In: *IEEE transactions on pattern analysis and machine intelligence* 42.4 (2018), pp. 824–836.

[14] Dan Hendrycks and Kevin Gimpel. “Gaussian error linear units (gelus)”. In: *arXiv preprint arXiv:1606.08415* (2016).

[15] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[16] Tom Brown et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[17] Alexey Dosovitskiy et al. “An image is worth 16x16 words: Transformers for image recognition at scale”. In: *arXiv preprint arXiv:2010.11929* (2020).

[18] Adi Haviv et al. “Transformer Language Models without Positional Encodings Still Learn Positional Information”. In: *arXiv preprint arXiv:2203.16634* (2022).

[19] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. “Layer normalization”. In: *arXiv preprint arXiv:1607.06450* (2016).

[20] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[21] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: `10.1109/CVPR.2009.5206848`.

[22] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. “Representation learning with contrastive predictive coding”. In: *arXiv preprint arXiv:1807.03748* (2018).

[23] Huaishao Luo et al. “Clip4clip: An empirical study of clip for end to end video clip retrieval”. In: *arXiv preprint arXiv:2104.08860* (2021).

[24] Zijian Gao et al. “CLIP2TV: An Empirical Study on Transformer-based Methods for Video-Text Retrieval”. In: *arXiv preprint arXiv:2111.05610* (2021).

[25] Xing Cheng et al. “Improving video-text retrieval by multi-stream corpus alignment and dual softmax loss”. In: *arXiv preprint arXiv:2109.04290* (2021).

[26] Simion-Vlad Bogolin et al. “Cross Modal Retrieval with Querybank Normalisation”. In: *arXiv preprint arXiv:2112.12777* (2021).

[27] Milos Radovanovic, Alexandros Nanopoulos, and Mirjana Ivanovic. “Hubs in space: Popular nearest neighbors in high-dimensional data”. In: *Journal of Machine Learning Research* 11.sept (2010), pp. 2487–2531.

[28] Anurag Arnab et al. “Vivit: A video vision transformer”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 6836–6846.

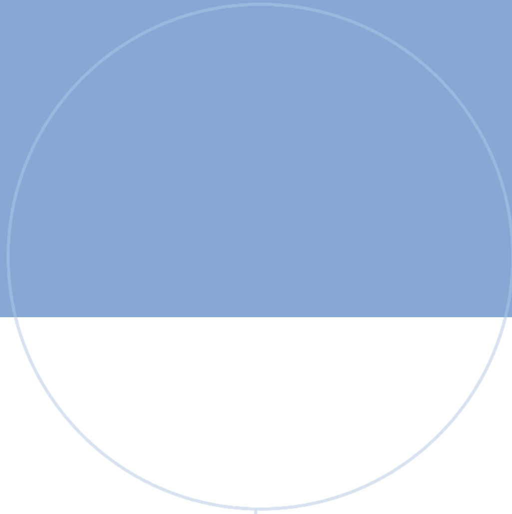
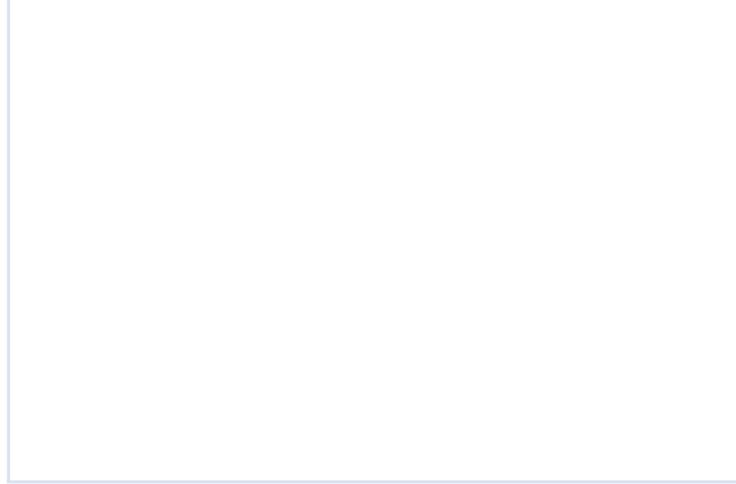
[29] Antoine Miech, Ivan Laptev, and Josef Sivic. “Learning a text-video embedding from incomplete and heterogeneous data”. In: *arXiv preprint arXiv:1804.02516* (2018).

[30] Youngjae Yu, Jongseok Kim, and Gunhee Kim. “A joint sequence fusion model for video question answering and retrieval”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 471–487.

[31] *pyvespa, Vespa Python API*. URL: `https://pyvespa.readthedocs.io/en/latest/index.html`.

[32] Feng Wang and Huaping Liu. "Understanding the behaviour of contrastive loss". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 2495–2504.

[33] Paul Michel, Omer Levy, and Graham Neubig. "Are sixteen heads really better than one?" In: *Advances in neural information processing systems* 32 (2019).

NTNU
Norwegian University of
Science and Technology