POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Politecnico
di Torino

Master's Degree Thesis

Large-scale video scene retrieval through Transformer Encoder

Supervisors

Prof. Andrea CALIMERA

Eng. Walter MAFFIONE

Dr. Rosalia TATANO

Candidate

Lorenzo DE NISI

October 2021

Abstract

Over the last few years the production of multimedia content has experienced a rapid growth. Such data constitutes a valuable source of information, but to leverage that great potential, automating human processes is crucial. A good portion of multimedia data is represented by video data. From social media and streaming services to security, videos constitute one of the most immediate mediums to convey information. Combining the great expressivity of written text with vision is the foundation of Vision-Language understanding, often employed to perform automatic supervision, moderation and anomaly detection. The Thesis goes in this direction, investigating different solutions for an application capable of performing retrieval and detection on a video, starting from a textual description of the desired scene. Experiments have been conducted with Transformer-based architectures and particular attention is given to scale efficiency and real-time capabilities, analyzing the trade-off between latency and precision, increasing input resolution and altering the architectures. Different approaches are considered. Firstly, dealing with single frames as input data, image retrieval is performed using TERN architecture. Aiming at real-time inference, a faster single-stage object detector is proposed instead of the original two-stage model. Secondly, processing short video windows instead of frames, video retrieval is performed using CLIP4Clip architecture, with a study on the impact of different input resolutions. For both approaches, real-time capabilities are evaluated. Lastly, to test image and video retrieval models on a different domain, a common retrieval dataset is created starting from security camera recordings, annotated with a self-labelling approach by a captioning model. The results show how, switching to a single-stage detector, TERN inference time is reduced by 10 times, at the cost of a noticeable drop in metrics. For video retrieval solution, the experiments demonstrate that increasing input size is beneficial for precision up to a certain resolution, at the cost of higher inference time. On the additional dataset, original TERN architecture achieved the best results, far ahead of the modified single-stage version, which pays the price for higher speed. CLIP4Clip models performed closely to the original TERN, with the potential advantage of exploiting temporal dimension to recognize more actions. The overall experiments testify the suitability of both approaches. Switching to a single-stage object detector is an effective way to speed up inference but can also lead to performance degradation. Increasing resolution is costly, especially in terms of inference and training time, but the benefits are noticeable. Lastly, considering the additional dataset, the weights of the models demonstrate the ability to easily generalize on a new domain, without needing a specific fine-tuning, although this would still lead to better performance.

Summary

The ever increasing amount of multimedia data produced every day constitutes a very valuable source of information. Unfortunately this opportunity does not come at no cost. Large-scale data brings in new challenges and automating human processes involved is, now more than ever, crucial to leverage the enormous potential that resides in such data. Video data represents a good portion of multimedia data, considering the different applications that rely on them. From social media to streaming services, from communication to security cameras. Establishing a semantic link between vision and language is the foundation of Vision-Language understanding. Such interoperability is the key to handle multimedia data, exploiting the great expressivity of written text to drive supervision. In fact, rapid growth of such data led to the need of automated supervision, moderation and anomaly detection, previously done by humans. Anomalies represent something unexpected that deviates from normal behaviour. What is considered anomaly heavily depends on the context. A car crossing a sidewalk, a person lying on the ground or a big crowd on a square represent some examples of anomalies from the perspective of a security camera. This work goes in this direction, conceiving an application capable of performing retrieval and detection of a particular scene frame on a video, starting from a textual description of the required scene. Such descriptions can be related to anomalous scenes, allowing the model to perform context-specific anomaly detection based on the user needs. One of the main challenges of this approach concerns the multi-modality. In fact, the model is required to learn an alignment between the same concepts expressed with language and with vision. Over the years, a lot of effort has been put to tackle this challenge, from hand-crafted features, as Scale-Invariant Feature Transform [1], up to recent deep learning solutions based on of Convolutional Neural Networks and Recurrent Neural Networks, such as selective multimodal LSTM (smLSTM) [2] and Dual Attention Networks (DANs) [3] or Graph Neural Networks such as Visual Semantic Reasoning Network (VSRN) [4]. In 2017, Transformer architecture has been proposed by Vaswani et al. [5] and, in the following years, different works have demonstrated the effectiveness of such architecture on Visual-Language tasks. For this reason, the focus will be on deep learning models based on the Transformer architecture,

more specifically on the Transformer Encoder. Such module, producing a compact, very informative representation of the input sequence, constitutes the core of all the discussed retrieval models. Other problems may arise from the performance side. In the described application, typical input data are raw videos, directly coming from cameras; this kind of data is particularly heavy to be processed due to high input resolution and sampling frequency. Some models are not able to scale efficiently with the amount of data and some others cannot work in real-time. Input resolution can sometimes ease the task, increasing detection capabilities so it's important to find a balance between accuracy and computational cost.
This work aims at studying and testing different Transformer-based models for scene retrieval, increasing the complexity of input data and with particular focus on the study of real-time capabilities of the considered models. To this end, different approaches will be proposed. Firstly, considering single frames as input data, image retrieval, i.e. the task of finding relevant images according to a textual description, will be performed using the model TERN [6]. Aiming at real-time inference, a faster single stage object detector is proposed instead of the original two stage model. The second approach consists in processing short video windows instead of frames. Video retrieval, i.e. the task of finding relevant videos according to a textual description, will be performed with model CLIP4Clip [7]. In addition, in this work, a study on the influence of input data resolution on the performance of the considered models is carried out. To achieve this, finetuning on different input sizes is performed on the Video Retrieval architecture to demonstrate the importance of resolution, especially in cluttered scenes. All the experiments have been conducted in collaboration with Addfor S.p.A.[1]. Finally, for both approaches, real-time capabilities are evaluated along with Recall and NDCG [6] metrics used for retrieval tasks. In addition to the original retrieval datasets, the models are also tested on a common retrieval dataset, created starting from 1 week security cameras recordings made available by Addfor S.p.A.
This thesis is structured as follows. Chapter 1 focuses on Vision-Language understanding and the related tasks. In Chapter 2, the specific tasks of Image and Video retrieval are discussed, with particular emphasis on the Deep Learning strategies before Transformer and on attention mechanisms. Chapter 3 gives an in-depth explanation of original Transformer architecture and the ways such model can be applied in Computer Vision and specifically in Image and Video retrieval. A discussion about efficiency in large-scale settings follows. Chapter 4 is dedicated to the details of the final architectures used for the application. The experiments and the results with different datasets and input resolutions are reported. Finally, in Chapter 5, the conclusions are drawn.

[1] `https://www.add-for.com/`

Acknowledgements

I would first like to thank Addfor S.p.A. for providing me the significant amount of resources needed for the conducted experiments. None of this would have been possible without your availability.

I would like to acknowledge the company supervisors, Eng. Walter Maffione and Dr. Rosalia Tatano that helped, advised and supported me throughout all the stages of this work.

I would also like to thank the academic supervisor, Professor Andrea Calimera for the availability and the precious advices.

I am also grateful to all my past and present colleagues at Politecnico di Torino who inspired and shared with me this amazing journey and to my friends, who contributed to make these years very special.

Last but not least, I would like to express my deep appreciation to my family, for the continuous financial and emotional support that, over the years, gave me the chance to do my best and reminded me every day how lucky I am.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In Chapter 1, the main goal of the Thesis along with the challenges involved will be summarized. Vision-Language understanding will be discussed, followed by a series of related tasks and possible use cases. At the end, a subsection will be dedicated to video understanding and the reasons why is useful to take it in consideration for the final problem.

1.1 Scene retrieval and detection

The final goal of the application is to retrieve the timestamps in which the specified scene description prompted in input happens in the video. In alternative, working on a real-time streaming, after inserting a series of possible scenes, the application notifies the user whenever one of them occurs. **Scene retrieval and detection** is a complex task involving multi-modal models, capable of processing both a textual input and a visual input. Consequently one of the challenges is to make the model learn a semantic alignment between the two modalities, bringing "closer" the similar concepts, regardless of the modality. For this reason, in the next Sections, **Visual-Language understanding** will be discussed. Other challenges concern performance. Operating on a large-scale database requires the model to **scale efficiently** in order to produce an inference in an acceptable amount of time. In addition, working in real-time brings in other limitations on the **inference time**. Then, **input resolution** is another important aspect to be taken into consideration. Higher resolution allows the model to recognize better also the smaller instances, but on the other side, it heavily affects the computational cost. For these reasons, both Image Retrieval and Video retrieval models will be tested on the tasks, considering the scaling capabilities and the inference time. Then, for models working on low resolution (224×224), finetuning on higher resolutions will be performed, studying the impact on metrics and performance cost.

1.2 Vision-Language understanding and generation tasks

Vision-Language tasks are a subset of Computer Vision tasks concerning the interaction of two information modalities: **vision modality**, that consists of images and videos, usually represented by rasters of pixels, and **natural language modality**, referring to written text. Such tasks aim at creating a connection between a set of words, composing a textual description, and a set of pixels representing an image. These two very different sources of information are the main building blocks of human understanding and communications, so it's easy to understand how crucial is to automate the process of linking them and the extraction of the underlying concepts. Some tasks are devoted to the Vision-Language understanding and require the model to be able to deeply understand the modality-agnostic meaning of the inputs. Some of them are **Image-Text retrieval** and **Natural Language for Visual Reasoning** (NLVR). Other tasks go beyond the understanding, requiring the model to **generate** new information based on the input. An example is **Image captioning**, an image-to-text generation task, but also the opposite has been studied, as testified by DALL · E [8], an astonishing Zero-Shot Text-to-Image generator proposed by OpenAI. While those tasks seem to be somehow very different, some of them can be solved by a single architecture through a Vision-Language common pretraining, whose purpose is to make the model learn generic image-text pairs representations, before finetuning. This highlights the importance of Vision-Language understanding, being the essential grounding for every related downstream task.

1.2.1 Image captioning

Image Captioning is a Vision-Language generation task consisting of generating natural language descriptions according to the content of an image. To perform Image Captioning, the **language model**, responsible for text generation, needs to be conditioned on the **image** (Figure 1.1). The resulting model is a multi-modal model, in which the two modalities are combined. The visual information is encoded and, word by word, the natural language description is generated starting from to the aforementioned encoding and the previously generated words.

1.2.2 Image-Text retrieval

Image and Text retrieval are two parallel tasks. In Image retrieval the goal is to find relevant images according to a textual description while in Text retrieval the process is reversed: finding relevant descriptions to an input image (Figure 1.2). Usually, the modality that needs to be retrieved (visual in Image retrieval

Figure 1.1: Image Captioning task example.

and textual in Text retrieval) is represented by a pool of candidate elements (this defines the main difference between Image captioning and Text retrieval). Then, according to a **similarity function**, the elements are sorted and the **top K** are retrieved.

Figure 1.2: **Left**: Image retrieval example. **Right**: Text retrieval example.

1.2.3 Visual question answering

Visual Question answering, as the name suggests, consists in generating or predicting the answer to a question related to the input image. This task is a good benchmark to evaluate the Visual-Language understanding capabilities of the model. The answer could be predicted, choosing from a pool of possible answers, or generated,

the same way captions are generated in Image captioning. The question typology can vary a lot, for example the model could be asked to count how many elements are showed, or where a specific object is located with respect to the others, or again, recognizing the action that a given entity is performing in the image. An example is reported in Figure 1.3 (Left).

1.2.4 Natural Language for Visual Reasoning

Natural Language for Visual Reasoning (NLVR) [9] and Natural Language for Visual Reasoning for Real (NLVR2) [10] are two Vision-Language understanding tasks introduced by Suhr et al. in 2017 and 2019 (Figure 1.3 (Right)). The tasks consist of determining whether a sentence about a visual input is true. In NLVR, the input is represented by the sentence and a synthetic (computer generated) image and the output is a binary classification (true/false). In NLVR2, instead of generating synthetic images, the visual input is composed of a pair of photographs. The sentence is labeled as True if it is coherent with the combined content of the two.

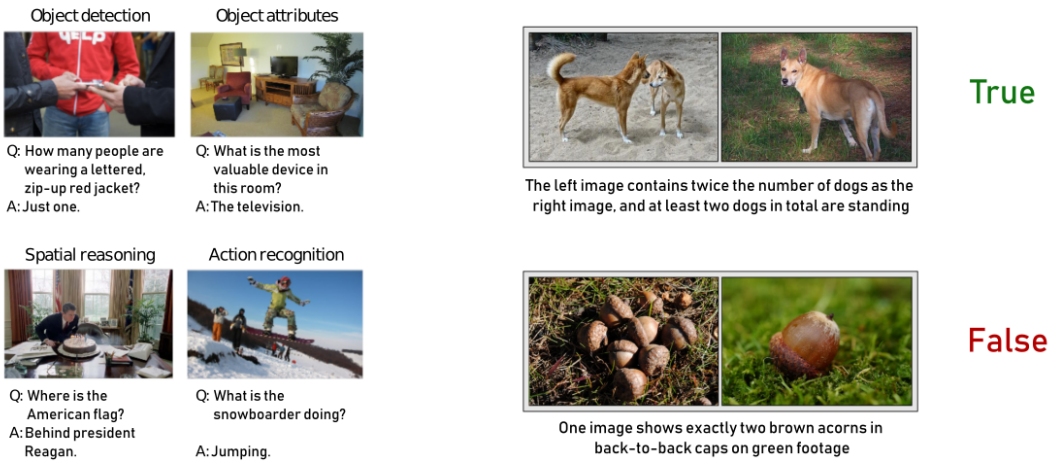


Figure 1.3: **Left**: Visual Question answering examples, questions can vary from counting to action recognition. **Right**: Natural Language for Visual Reasoning for Real (NLVR2)[10]) dataset samples. Labels refer to a pair of pictures. (Sources: Left:[11] , Right: [10])

1.3 Use cases

Vision-Language tasks are particularly handful for several applications. Autonomous vehicles, search engines, accessibility for visually impaired people, medical screening, image indexing and browsing are a few of the fields that benefit from models able to connect visual and language modality. Narrowing the focus on **video frame retrieval**, the ability to find a specific scene inside a very long video could be very useful for CCTV cameras recordings or video-editing applications, thus saving a lot of time otherwise spent watching (and manually annotating) the entire content. The same concept could be extended to real-time streaming, signalling to the user when there is a match between a frame in the last time window and a fixed "anomaly" query. Working with natural language then, widens the accessibility to non-technical users and increase explainability of the algorithm decisions.

1.4 Towards video understanding

As expressed before, human communication heavily relies on vision and language. Although single pictures could be sufficient in many cases, there are some scenes that are quite difficult, if not impossible, to be fully described by a static image. Human actions often depend on the temporal dimension to be disambiguated. Examples are pairs of actions such as *open-close* or *jump-fall* in which the main difference resides in the frame order. It follows that, in such cases, to achieve a complete understanding, models working on images are not enough. The main tasks involving video-language understanding are directly derived from their image understanding counterparts described above: **video captioning**, **video-text retrieval**, **video question answering** are some examples. The main purpose remains the same while the vision input is a sequence of frames for which a single inference is performed. Considering the nature of the problem, described in Sec. 1.1, both **image** and **video retrieval** models can be taken into account. For this reason in the following sections both tasks will be considered.

Chapter 2

Image and Video retrieval

Once established that retrieval tasks are best choice for the Scene retrieval and detection, in this Chapter, Image and Video retrieval will be discussed with particular emphasis on the Deep Learning strategies before Transformer [5] and on attention mechanisms.

2.1 Image retrieval

Image retrieval task, as anticipated in Sec. 1.2.2, aims at finding relevant images according to a textual description. Usually, the textual description is defined as the **query** and is written in **natural language**. The retrieved elements, images in this case, belong to a pool of candidates. All the candidates need to be **ranked** according to a **similarity function** with the input query and the first K are returned. Image retrieval can be widely used for search engines in which, prompting a textual description (or even another image), relevant images are gathered from a large-scale database. Image indexing and browsing software can rely on Image retrieval, recognizing the subjects and the objects in the pictures, being able to label and put them in different folders, simplifying the access by the user. Finally, recognizing the context can also be handful for image editing application, suggesting context specific presets or settings.

2.1.1 Metrics

Evaluating an Image Retrieval system is not trivial. The actual relevance of a proposed image is difficult to assess and there is no common strategy to find the perfect ranking everyone agrees with. The most common metric is the **Recall@K**. Given a dataset of image-caption pairs, all the images are ranked according to one of the captions and the prediction is considered correct if, in the top-K retrieved

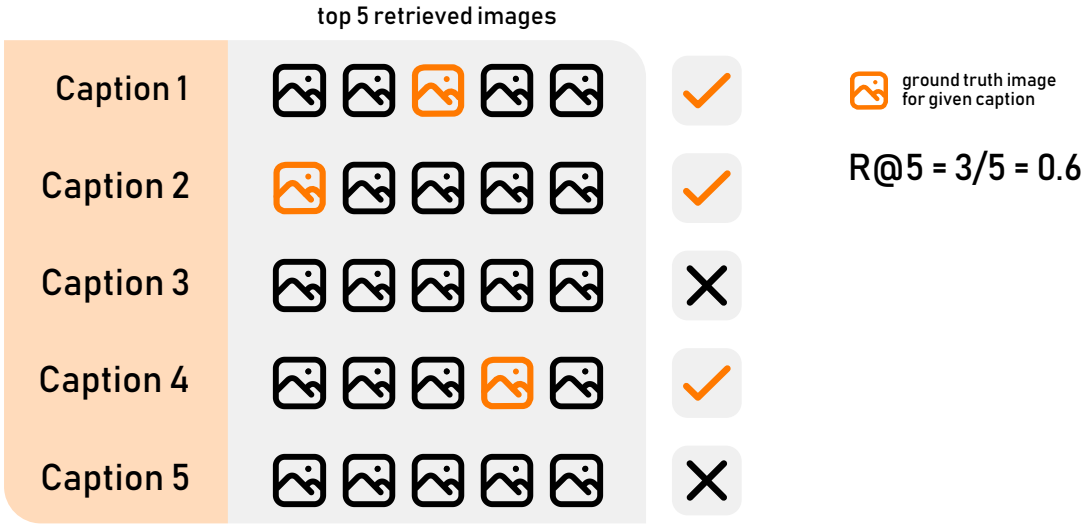


Figure 2.1: Recall metric for $K = 5$ (R@5) evaluated for 5 captions. Each caption is associated with one specific image (highlighted in orange). If the latter is retrieved among the top K (top 5) results, the ranking is considered correct.

images, is found the specific image the caption was coupled with. An example is reported in Figure 2.1. Most common values for K are 1, 5, and 10. Higher K values make the metric less strict, leading to greater recall values while lower K values penalize the model more. Focusing on a search engine application, the user experience is not strictly dependent on the ability of the model to retrieve the exact image as first result. Priority should go to the quality of the retrieved candidates, assessing the relevance with the query. To better fit the user needs, a new metric, the **Normalized Discounted Cumulative Gain** (NDCG) has been introduced by Messina et al. [6] and will be further discussed in Sec. 3.4.3.

2.2 Deep Learning for Image Retrieval

Traditionally, when querying an image database, retrieval can be based on the **content** of the candidate images (content-based) or based on the images **metadata** (concept-based). While the latter heavily relies on annotation quality and coverage, Content Based Image Retrieval (CBIR) allows the application to work even without keywords or metadata in general. Before the advent of Deep Learning models, CBIR relied on hand-crafted features such as Scale-Invariant Feature Transform [1] extracted from the query image and compared with the same features extracted from the candidates. Deep Learning introduced models able to automatically design and extract visual features, leading to a fresh start in computer vision

approaches to solve the task. The main Deep Learning models involved in Image retrieval are the Deep Convolutional Neural Network (DCNN). A famous historical example is AlexNet [12], firstly introduced by Krizhevsky et al. in 2012, showing impressive results on ImageNet[13] contest for the time. Nowadays bigger and more powerful DCNN are used, such as ResNet [14], VGG [15] and ResNeXt [16]. If the query is a natural language text then, Recurrent Neural Networks (RNN) are also involved to sequentially process the words. One famous example is represented by Long Short-Term memory (LSTM) [17] networks, introduced by Hochreiter and Schmidhuber in 1997.

2.3 Attention mechanism

2.3.1 Introduction

Attention is the process of selectively concentrating on some information, ignoring other. This is a typical behaviour of the human brain and a lot of effort has been put to translate it into deep learning and computer vision. Traditional attention methods focus on conditioning some context vector on the input. The first attention mechanism was elaborated in 2014 by Bahdanau et al. [18] that proposed an extension of the encoder-decoder architecture for machine translation that learns the alignments between source and target words. The architecture is composed of an **encoder**, that encodes the source sentence and a **decoder** that, starting from the encoder output and the previous state, generates the translation word by word, autoregressively. Both encoder and decoder are recurrent neural networks (RNN). At each step, a new word is fed into the encoder, producing a new hidden state. In the traditional framework, the output of the encoder is a fixed-length context vector c derived from encoder hidden states (usually c is equal to the last hidden state), that for very long sequences eventually leads to information loss. To solve this issue, Bahdanau et al. proposed to feed the decoder with a weighted sum of the encoder hidden states, where the weights are calculated each decoder step, according to the alignment between source and target words:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{2.1}$$

h_j is the encoder hidden state at step j (hidden state of j-th word), T_x is the input sequence length and α_{ij} are the alignment weights between source word at position j and target word at position i. Note that in this case there is a distinct context vector c_i for each target word y_i. The weights α are obtained by an *alignment*

model a:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \tag{2.2}$$

$$e_{ij} = a\left(s_{i-1}, h_j\right) \tag{2.3}$$

$$\tag{2.4}$$

where s_{i-1} is the decoder hidden state at time $i-1$.
Different *alignment models* can be evaluated. Some of them are reported by Luong et al. [19]:

$$a(s_i, h_j) = \begin{cases} s_i^\top h_j & dot \\ s_i^\top W_a h_j & general \\ v_a^\top \tanh\left(W_a\left[s_i; h_j\right]\right) & concat \end{cases} \tag{2.5}$$

where W_a and v_a are learnable projection matrices and vectors and $[\cdot;\cdot]$ indicates concatenation operation.
This model learns to score the alignment between the output word at a given position i and the input words around position j. In other words, alignment is evaluated between encoder and decoder hidden states respectively at position $i-1$ and j, producing the weights α_{ij}. Once weights are computed for each encoder hidden state, the weighted sum in Eq. (2.1) is performed to obtain the context vector to be fed to the decoder.

2.3.2 Attention methods

Starting from the first implementation in 2014 [18], different variants of the attention mechanism have been introduced. The common ground for all of them is that the final goal is to modulate a context vector, conditioned on the input.

Soft attention

Soft attention refers to the one proposed by Bahdanau et al. [18] and presented above. The attention scores are used as **weights** in the weighted sum context vector. Since α weights are normalized, α_{ij} can be considered as the probability that the context vector is equal to the encoder hidden state at position j:

$$\hat{h} = \{h_1, \ldots, h_{T_x}\} \tag{2.6}$$

$$p(c = h_j | s_{i-1}, \hat{h}) = \alpha_{ij} \tag{2.7}$$

Hence, Eq. (2.1) can be seen as the **expected value of the context vector**:

$$c_i = \mathbb{E}[c | s_{i-1}, \hat{h}] = \sum_{j=1}^{L} \alpha_{ij} h_j \tag{2.8}$$

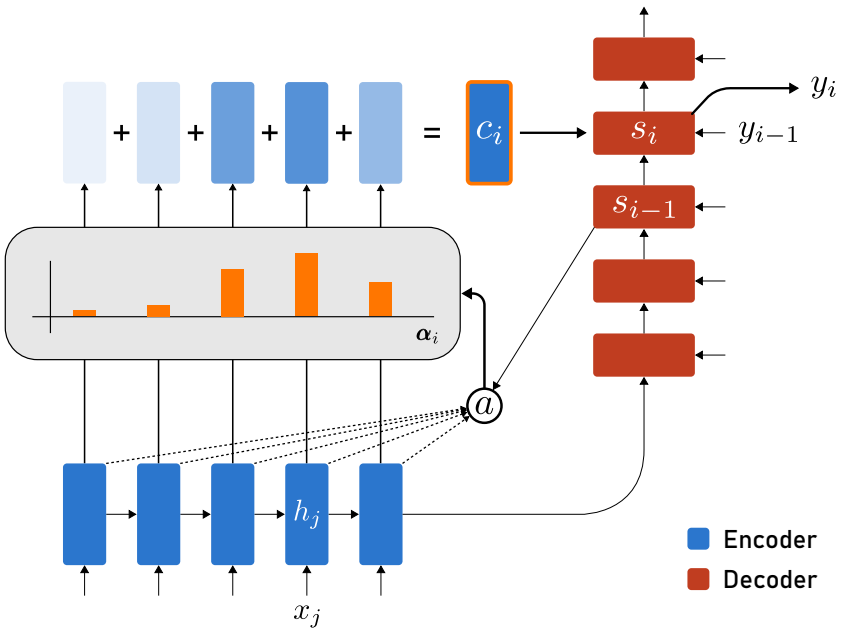


Figure 2.2: Example of soft attention applied on an encoder-decoder model for machine translation. The input words x_j are fed into the encoder (blue) and the hidden states are used in combination with the decoder (red) hidden state s_{i-1} by the *alignment model* a, to compute the scores $\boldsymbol{\alpha}_i$ (orange). Such scores are used as weights for the weighted sum of the encoder hidden states that will constitute the context vector c_i. The context vector, combined with the decoder hidden state at step i and the previously generated word y_{i-1}, generates the next word y_i.

Note that the entire procedure is differentiable, so standard backpropagation can be performed. An example of soft attention is illustrated in Figure 2.2 and Figure 2.4 (left).

Hard attention

Starting from the same considerations of soft attention, considering α weights as probabilities, in the case of hard attention, a Multinuolli distribution is built, parametrized by α values, as described by Xu et al. [20]:

$$\boldsymbol{\alpha}_i = \{\alpha_{i,1}, \ldots, \alpha_{i,T_x}\} \tag{2.9}$$

$$t_i \sim \text{Multinoulli}(\{\boldsymbol{\alpha}_i\}) \tag{2.10}$$

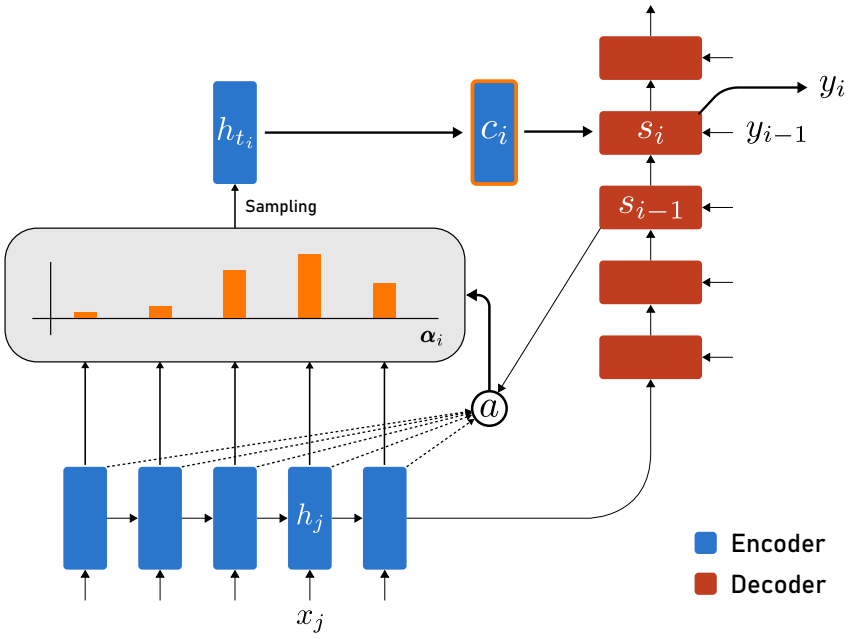


Figure 2.3: Example of hard attention applied on an encoder-decoder model for machine translation. The difference with respect to soft attention, illustrated in Figure 2.2, is that the context vector c_i is not the weighted sum of the encoder hidden states but is equal to the hidden state at position t_i where such position is sampled from a Multinuolli distribution, parametrized by $\boldsymbol{\alpha}_i$ scores.

For the i-th position of the decoder, instead of having a weighted sum of all the encoder hidden states as for soft attention, just one hidden state is taken and its position t_i is **sampled from the Multinoulli distribution**. Very aligned hidden states will have higher probability to be sampled. Using this sampling strategy, a non-differentiable step is introduced. For this reason, standard backpropagation cannot be used. The gradient can be approximated via Monte Carlo method, as shown in [20]. An example of hard attention is illustrated in Figure 2.3 and Figure 2.4 (right).

Global vs Local attention

Attention methods can be divided into two categories, *global* and *local*. The difference resides in where the attention is placed. In *global* attention, the context vector is derived from all the encoder hidden states. It produces scores for each element of the input sequence and then derive c_i from them. The main drawback

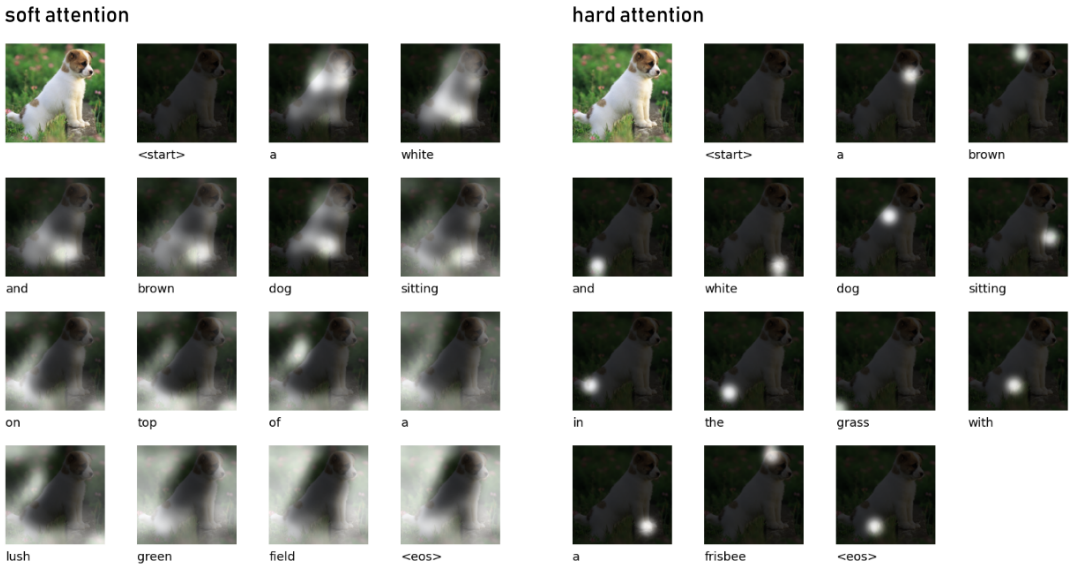


Figure 2.4: An example of soft attention (**Left**) and hard attention (**Right**) applied on image captioning task. Lighter areas have higher attention scores. In the case of hard attention, one sampled region has maximum score while the others have score zero. Inference is performed on the model described by Xu et al. [20] with region features extracted from VGG [15] and fed to an LSTM [17] with attention over the image to generate the description[1]. Note that, in general, hard and soft attention models generate different captions. Other examples are reported in Appendix A.

of this approach is that, for each target word, the model needs to attend to all the input words, and for very long sequences can be expensive or even impractical. To tackle this problem, Luong et al. [19] proposed *local* attention. *Local* attention mechanism chooses to focus only on a small subset of the input sequence positions. It generates an aligned position p_i (for i-th target word) and then the context vector is derived only from the hidden states in positions $[p_i - D, p_i + D]$, with D empirically selected. In this way, the number of required scores is fixed to $2D + 1$ and the amount of computation does not grow with the input sequence length. p_i can be predicted or simply set to i. An example of local attention is illustrated in Figure 2.5.

[1]Visualizations are produced with code provided by AaronCCWong, available at `https://github.com/AaronCCWong/Show-Attend-and-Tell`

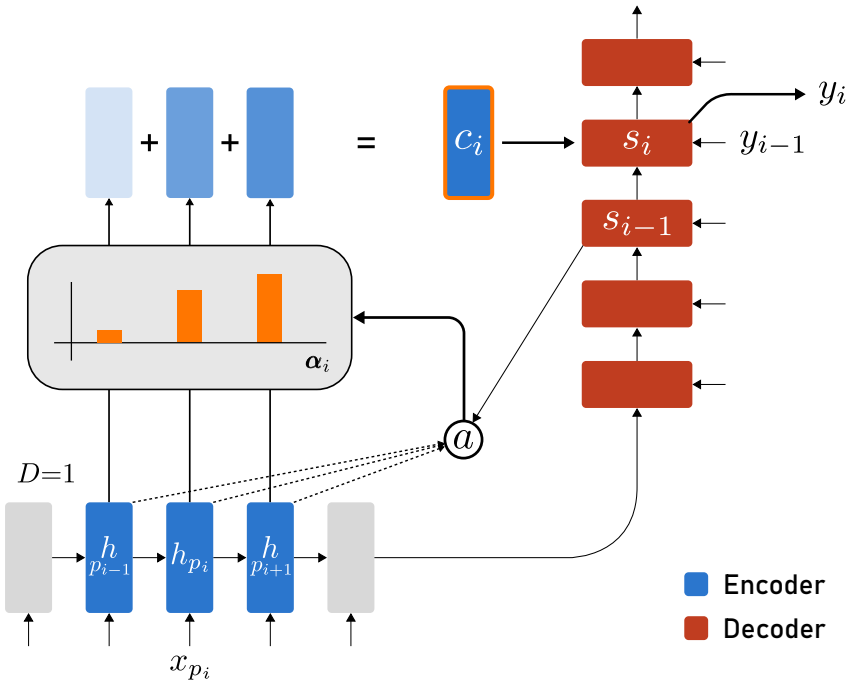


Figure 2.5: Example of local attention applied on an encoder-decoder model for machine translation. The difference with respect to global attention is that the model focuses only on a subset of the input encoder hidden states. In particular it chooses a position p_i and attend to the hidden states at positions $[p_i - D, p_i + D]$. In this case D=1, so only three scores are computed.

Scaled dot-product attention

In general, as described by Vaswani et al. [5], attention inputs can be represented as **key-value pairs**, where the **key** is used to compute the scores with respect to a **query**, while the **value** is used to generate the selected information, applying those scores. The query-key matching can be considered as the word alignments, mentioned for the previous methods. In scaled dot-product attention, the scores are computed with a dot product between query **Q** and key **K**, scaled on the square root of the sequence length d_k. The softmax operation is applied, before multiplying the scores on the value **V** to obtain the attentioned output:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{2.11}$$

An extension of scaled dot-product attention is **Multi-head attention** in which the inputs are projected into multiple Queries, Keys and Values and attention is

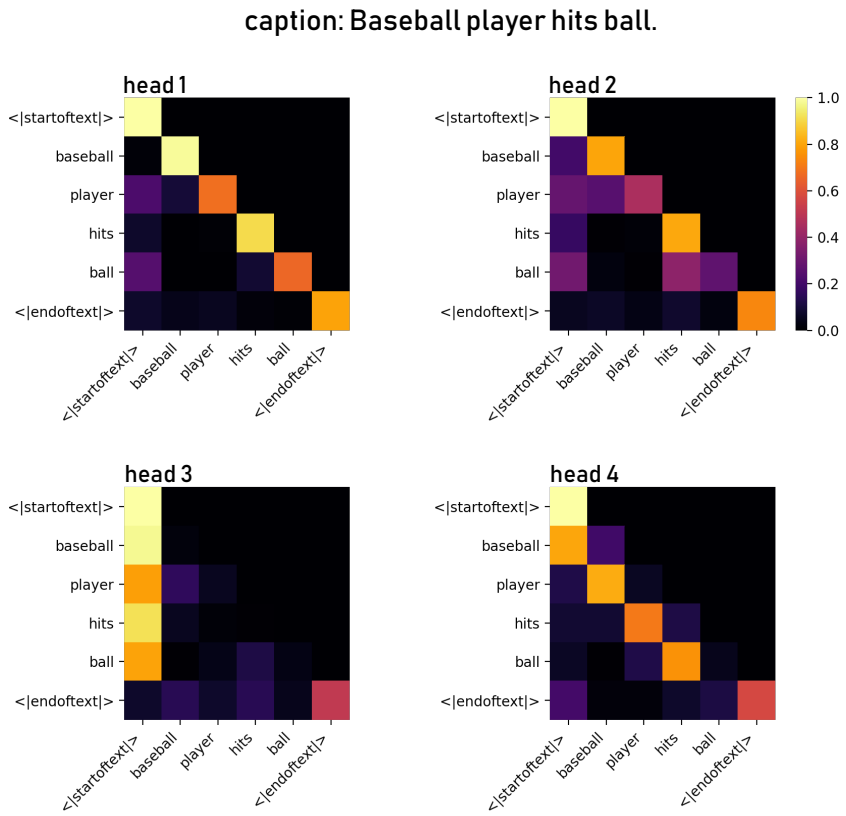


Figure 2.6: Example of multi-head dot product self-attention applied on a caption. Attention scores are calculated for each pair of tokens. The upper right portion of the scores are set to zero to avoid previous words to attend to next ones (masked self-attention, see Sec. 3.1.5 for further details). The scores are extracted from the first Transformer layer of the text branch of CLIP [21]. Each heatmap refers to one of the attention heads. Here the first 4 heads (out of 8) are reported. Attention scores for all the heads can be found in Appendix A. (Figure A.8)

applied in parallel. This allows the model to learn different alignments from the same input.
This attention method has been introduced by Vaswani et al. in 2017 for the Transformer architecture [5]. Examples of multi-head dot-product attention are illustrated in Figure 2.6 and Figure 2.7. Further details about the architecture and these methods are reported in Chap. 3.

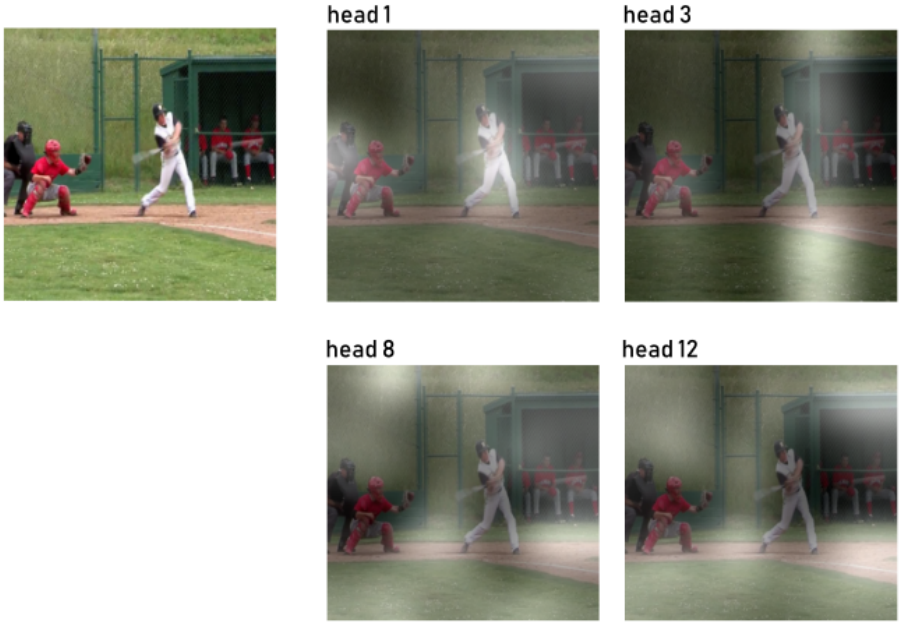


Figure 2.7: Example of multi-head dot product self-attention applied on an image. Attention scores are calculated for each pair of image regions and then summed to obtain the final visualization. The scores are extracted from the first Transformer layer of the visual branch of CLIP [21]. Each heatmap refers to one of the attention heads. Here the first 4 heads (out of 12) are reported. Attention scores for all the heads can be found in Appendix A. (Figure A.9)

2.4 Related works before Transformers

Starting from CNNs and RNNs mentioned before, different architectures have been proposed for Content-Based Image Retrieval (CBIR).
In 2015, Ma et al. proposed multi-modal convolutional neural networks (m-CNNs) to match images and sentences [22]. Their architecture includes three main components. **Image CNN**, a CNN (OverFeat [23] or VGG [15]) responsible for the encoding of image information, **Matching CNN** that combine image encoding and text, with different aggregation levels (word-level, phrase-level and sentence-level), **MLP**, multilayer perceptron that takes the joint representation coming from the Matching CNN as the input and produces the final matching score.
Huang et al., in 2016 proposed a selective multimodal LSTM (smLSTM) [2] that compute the image-text similarity by attending, each timestep, to different image regions and caption words, obtained by an **attention-based modulation**. Image and text representation are extracted by respectively a VGG [15] and a bidirectional

LSTM (BLSTM) [24]. Attention mechanism takes into account both global and local image (and text) representation. Modulated inputs are then fed to a multi layer perceptron (MLP) followed by an LSTM that aggregate the representations at each timestep. The LSTM hidden state conditions the attention for the next step and the last hidden state is used to compute the final matching score.
A relatively simpler setting was proposed by Nam et al. under the name of Dual Attention Networks (DANs) [3] in 2017. Image and text representations are extracted in a similar way to smLSTM and similarity is evaluated starting from memory vectors, updated in a RNN fashion. Image and text memories are independent. Those vectors are recursively updated by a **soft-attention mechanism** applied on the input representation and conditioned on the memory at the previous step. Final score is the inner product between image and text memory vectors.
In 2017, Huang et al. proposed a new method [25] for computing the image embedding based on the extraction of semantic concepts from different regions, through a multi-label CNN (VGG [15]). Concepts are nouns, verbs and adjectives, representing the CNN labels. The model learns how to reorder the concepts via a specific sentence generation supervision. Global context information (again extracted by VGG) is combined with local region semantic concepts via a learnable gating mechanism to produce the final image representation. Such vector is used both for sentence generation task (during training only) and to evaluate the image-text score for the matching task.
A very powerful two-stage object detector, Faster R-CNN [26] was used by Lee et al. for SCAN [27]. **Stacked Cross Attention Network** (SCAN) leveraged on a novel attention method called Stacked Cross Attention, applied over the region features extracted by Faster R-CNN. It's a two stage attention. At stage 1, it attends to words in the sentence with respect to each image region. For each image region it produces a sentence vector that is compared with the region itself on the second stage, obtaining per-region relevances. Relevances are then aggregated with a specific pooling function to compute the final score. This procedure is applied for image-text matching while a similar method is reported for text-image matching.
Aiming at a more relational-aware image embedding method, in 2019, Li et al. proposed **Visual Semantic Reasoning Network** (VSRN) [4], an image-text retrieval model that exploit relations among regions through a Graph Neural network (GNN). Region features are extracted following SCAN [27], using directly Faster R-CNN object detector. On top of that, a Graph Neural Network is applied with the regions as nodes and the pairwise affinity between regions as edge weights. The output nodes of the GNN are then fed to a recurrent neural network (GRU [28]) that perform global reasoning and output the final image representation.

2.5 Video retrieval

Video retrieval is the video counterpart of the image retrieval task presented above. The main idea remains the same: all the candidates inside the pool, that in this case are videos instead of static pictures, are ranked with respect to a **query** according to a **similarity function**. The query is usually written in natural language and the metrics are the same found for image retrieval task. The way visual features are extracted and the way multi-modal information are combined represent the main differences between image retrieval and video retrieval models. The additional challenges come from the need to encode temporal information. Usually this is done by aggregating key frame encodings coming from traditional CNN architectures (VGG [15], ResNet [14], ResNeXT [16]). Aggregation can be performed by a simple average [29], a weighted average based on attention [29] or by using sequential models such as Recurrent Neural Networks [30] [31]. Other methods relies on 3D Convolutional Neural networks (I3D [32], S3D [33]) to extract temporal-aware information from videos. Those methods can even be combined to benefit both from object-based encodings coming from 2D CNNs and action-based encodings extracted 3D CNNs [34] [35] [36] [37]. Besides the ways video are processed, all the other components can be transferred from image retrieval. Once video embedding is calculated, the pipeline can be easily adapted.
Considering the scene retrieval task, video retrieval could be crucial, in particular when the query describes an action, hardly recognizable by a image retrieval model applied on the frames. Being able to exploit the temporal information, video retrieval models can produce more time-aware encodings. For this reason, in the following Chapters both image and video retrieval models will be investigated for the task.

Chapter 3

Transformers

Transformer is a network architecture introduced in 2017 by Vaswani et al. [5] originally conceived to solve the task of machine translation. This task consists into automatically converting source text in one language to text in another language (for example translating an English sentence into German). Transformers rely solely on **attention mechanisms**, that will be covered in details in Sec. 3.1.3, without the need of convolutions or recurrence. Consequently, the amount of computations that can be executed in parallel is remarkably higher with respect to previous architectures proposed for the same task, thus resulting in faster training time and better performances.

3.1 Transformer architecture

The Transformer architecture is composed of two main units, an **encoder** and a **decoder**. The complete architecture is shown in figure 3.1.
The encoder takes the input of the model and, leveraging the attention mechanism, produce an **encoded representation** that will be fed to the decoder. The decoder combines the partial output and the encoded input to generate the next output token. In the first part of the decoder the previously generated decoder output is encoded. Then the, both encoded, input and partial output are **combined** with cross-attention to generate the model output.
In the machine translation setting, the **input sequence** (source text) is fed into the encoder while the decoder outputs the **translated text**, one word each step. The generation of the next word is conditioned both by the input information, coming from the output of the encoder, and the already generated text that is fed into the decoder input (auto-regressive behaviour). This is crucial because, in the context of an English-German translator for instance, the next German word depends, at the same time, on the whole English text and on the partial translation

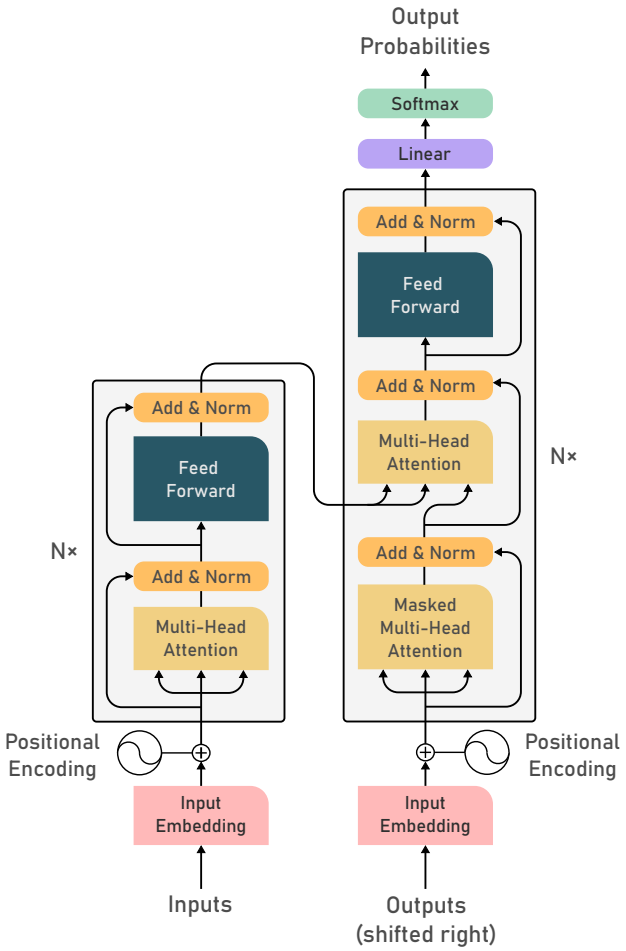


Figure 3.1: The Transformer architecture, with the encoder on the left and the decoder on the right.

already done.
Beside the **attention mechanism**, for which the general idea will be presented in Sec. 3.1.3, other actors are involved. First of all, **word embedding**, used to transform input word tokens into a more convenient representation, and **positional encoding**, useful to inject into token representation the position inside the sentence. Finally, a small subsection will be dedicated to the training technique used in the machine translation setting, **teacher forcing**.

3.1.1 Word embedding

Source text and partial output words needs to be codified into sequence of numbers, in order to be fed into the model. A naive approach could be building a dictionary of all the words that appears in the training set, associating to each entry an index that will be used as an encoding of the word. The problem with this approach is that, encoding words with progressive numbers injects into the input a meaningless ordering. For example, if the dictionary is composed of 4 words: `alpha, beta, delta, epsilon`, the encoding should be `{alpha=0, beta=1, delta=2, epsilon=3}` but the model could wrongly learn that `delta` is "twice" `beta`, `beta` is greater than `alpha` and so on. This sort of ordering is clearly misleading.
To tackle this problem, another solution could be feeding into the model a n-sized vector, where n is the dimension of the dictionary, with a one in the position corresponding to the word index and zeros everywhere else (**one-hot encoding**). So, following the previous example, the encoding should be `{alpha=[1, 0, 0, 0], beta=[0, 1, 0, 0], delta=[0, 0, 1, 0], epsilon=[0, 0, 0, 1]}`. In this way, each input is "orthogonal" to the others, preventing some wrong learning. However, this approach has some drawbacks: the resulting vectors are sparse, huge and the "distance" between two words is always the same, regardless of the semantic meanings.
To solve this problem, **word embeddings** have been introduced. They consist in smaller fixed size vectors containing the encoded meaning of the words. This means that conceptually similar words are closer than unrelated words. Let us assume that N and D are, respectively, the dimension of the dictionary and the dimension of the final word embedding. The word encoding, WE, is obtained by multiplying a one-hot encoded vector OH with a learned weight matrix W, i.e.

$$WE = OH \times W \tag{3.1}$$

where $WE \in \mathbb{R}^D$, $OH \in \mathbb{R}^N$ and $W \in \mathbb{R}^{N \times D}$. In the case of Transformer, the dimension of the final word embedding is 512. These weights can be learned along with the whole model or kept fixed to pre-learned values and accessed as a lookup table. Some examples of pre-learned word embeddings are Word2Vec [38] and GloVe [39]. In the original implementation of the Transformer, embeddings are randomly initialized and refined during training.

3.1.2 Positional encoding

While word embeddings, introduced in the previous subsection, are useful to transfer to the encoding the meaning of the word, they are not able to capture information about the relative order in the sentence. Using recurrent neural networks, processing one word at a time, the order information is automatically codified, while in this

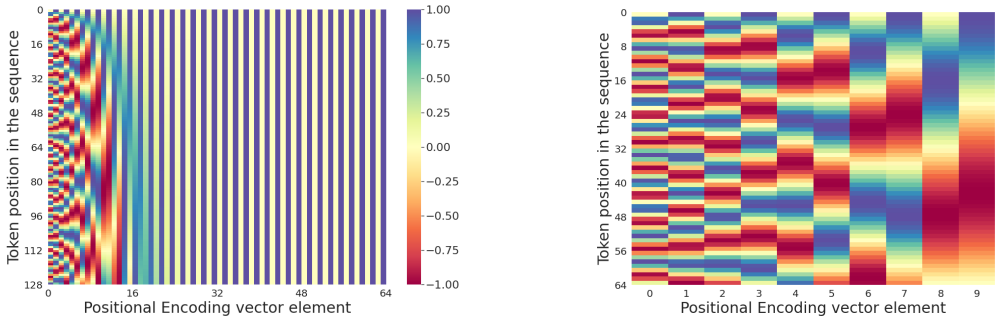


Figure 3.2: Left: Positional encoding vectors for $d_{model} = 64$, for the first 128 positions of the input sequence. **Right**: A closeup for the first 64 positions and 10 vector elements. The i^{th} row represent the encoding vector that is added to the word embedding of the i^{th} word of the sequence.

case, the Transformer processes the entire input simultaneously. The embedding for a given word is always the same, regardless of the position inside the sentence. Lack of such information could be a problem because the context and the meaning of a word significantly depend on the position. Moreover, self-attention is bidirectional, this means that the produced output is not dependent on the order of the input tokens. Shuffling the tokens would produce the same final encoding. To solve this problem, **positional encoding** is introduced to inject positional information. With positional encoding, each word embedding is summed to a positional vector that depends on the position of the word itself inside the sentence. Positional vectors PE are defined as:

$$PE_{(pos,2i)} = sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{3.2}$$

$$PE_{(pos,2i+1)} = cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{3.3}$$

Each positional vector has the same dimension of the word embeddings, equal to d_{model}. For each input token, positional vector is computed starting from the position pos that such token has inside the sentence. Given pos, each element i of the positional vector (from 0 to $d_{model} - 1$) is defined by the functions above. An example of the positional vectors obtained for $d_{model} = 64$ is presented in figure 3.2, for the first 128 positions (pos) of the input sequence.

3.1.3 Attention mechanism

As anticipated in Sec. 2.3, attention mechanisms allow machine learning models to selectively focus on the main information in the input, ignoring the rest. In

the case of the Transformer, **scaled dot-product attention** is implemented. An attention function can be seen as a weighting function that is applied to the input to produce the attentioned output. With **scaled dot-product attention** (also reported in Figure 3.3), three set of vectors are considered, $Q \in \mathbb{R}^{t \times d_k}$, $K \in \mathbb{R}^{s \times d_k}$ and $V \in \mathbb{R}^{s \times d_v}$, respectively **Queries**, **Keys** and **Values**, where t and s are the lengths of the input sequences (so they represent the number of stacked vectors), while d_k and d_v represent the actual dimension of each vector. The output of the attention function is the **Values** vector, weighted by the compatibility between **Queries** and **Keys**. Compatibility is evaluated by a dot product between Q and K, scaled on $\sqrt{d_k}$ to avoid the product to be huge for long sequences and with a **softmax** applied, formally:

$$\text{Attention}(Q, K, V) = \sigma\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{3.4}$$

The **softmax** function $\sigma(\cdot)$ is defined as

$$\sigma(s_{i,j}) = \frac{e^{s_{i,j}}}{\sum_{z=0}^{s} e^{s_{i,z}}} \tag{3.5}$$

for $i \in [0, t-1],\ j \in [0, s-1]$, where

$$\frac{QK^T}{\sqrt{d_k}} = S \in \mathbb{R}^{t \times s}. \tag{3.6}$$

Hence, the sum of the elements is 1 for each row and the dot-product becomes a weighted sum. In the implementation $t = s$.
Inside the Transformer architecture, attention is computed in three different ways in which the main difference is the nature of the input vectors **Queries**, **Keys** and **Values**. Once **Q**, **K** and **V** are obtained, the inner computations presented above are the same.

Encoder self-attention. Inside the encoder, attention inputs Queries, Keys and Values are all coming from the same input sequence, performing so a self-attention on the input itself. This means that the encoder will learn the relations among the input tokens, implicitly learning the language syntax and grammar.

Decoder masked self-attention. The attention mechanism performed inside the first decoder submodule is very similar to the encoder one. The input vectors, Queries, Keys and Values are all coming from the decoder input sequence. Future ground truth words are masked with a triangular matrix that forces to zero the scores related to the words the model is not supposed to see.

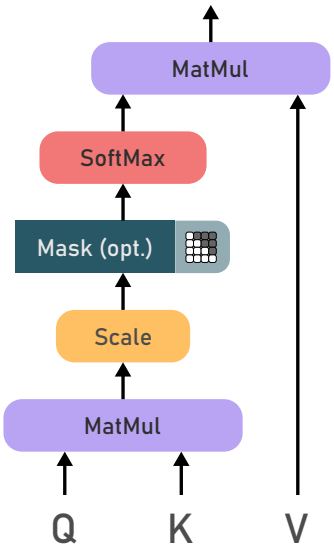


Figure 3.3: Scaled Dot-Product Attention.

Encoder-Decoder cross-attention. In the second decoder submodule, input vectors are coming from different sources, Queries are coming from the encoder output while Keys and Values are coming from the previous decoder submodule, performing so a cross-attention between input and partial output information.

Multi-head attention

Performing a single instance of attention means learning only one type of relationship between input tokens. Therefore, input vectors are linearly projected h times with different projection matrices W^Q, W^K and W^V and attention is performed h times in parallel, obtaining h output vectors that are then stacked again and projected again to be fed to the next layer. The general schema is reported in figure 3.4. Following this approach, with a slight notation change (same notation of [5]), the dimensions of input vectors Q, K, V will be $s \times d_{model}$ while d_k and d_v will represent the dimension after projection. In the original implementation, $d_{model} = 512$, $h = 8$ while $d_k = d_v = d_{model}/h = 64$. In this way, while the attention function is computed 8 times, the dimension of the vectors is 8 times lower, keeping the computational cost similar to single-head attention. This approach is called **multi-head dot-product attention**. Formally:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, ..., head_h)W^O \tag{3.7}$$

$$head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{3.8}$$

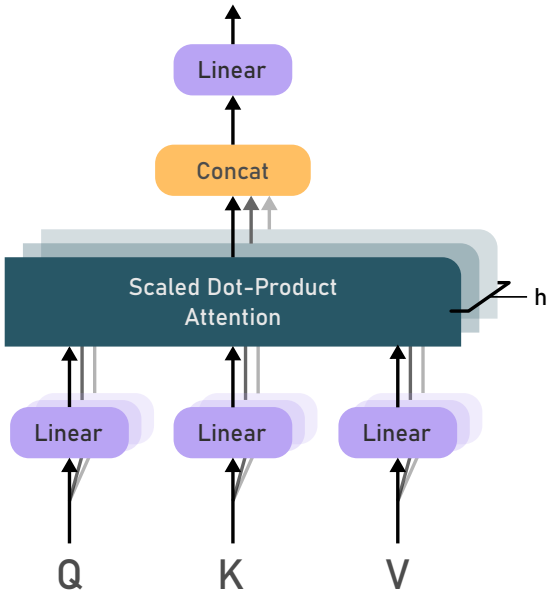


Figure 3.4: Multi-Head Dot-Product Attention.

where $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ are the projection matrices of queries, keys and values for each head and the final projection matrix is $W^O \in \mathbb{R}^{d_v \times d_{model}}$.

Teacher forcing

Teacher forcing is a training technique consisting in feeding the decoder with the partial ground truth sentence instead of the previously generated one. In this way, for instance, an error in the generation of the second word has no repercussions on the generation of the third one and the model is able to converge faster. So, at training time, during the inference of the n^{th} word, the decoder input will be the sequence of the first $(n-1)^{th}$ ground truth word tokens, regardless of the previously generated ones.

3.1.4 Encoder

The encoder is the first building block of the Transformer architecture. The preprocessed input sentence (tokenization, word embedding and positional encoding)

is fed into the first encoder module. One encoder module is composed of two submodules, the first one is the actual multi-head attention and the second one is a feed forward layer. The former has been explained in Sec. 3.1.3 while the latter consists into two linear transformations with a ReLU [40] in between:

$$\text{FFN}(x) = [xW_1 + b_1]_+ W_2 + b_2 \tag{3.9}$$

where $[x]_+ = \max(0, x)$. Both submodules have residual connections, followed by layer normalization, formally:

$$\begin{aligned} \text{EncoderLayer}(x) &= \text{LayerNorm}(\text{Sublayer}(x_{EA}, \text{FFN}(x_{EA}))) && (3.10)\\ x_{EA} &= \text{LayerNorm}(\text{Sublayer}(x, \text{MultiHeadAttention}(x, x, x))) && (3.11)\\ \text{Sublayer}(x, f(x)) &= x + \text{Dropout}(f(\text{Norm}(x)) && (3.12) \end{aligned}$$

where x_{EA} is the encoder self-attention output coming from the first submodule and fed into the second one (FFN). The same module is repeated 6 times, feeding the input of the next one with the output of the previous (the individual head outputs are concatenated to match the input dimensions).

3.1.5 Decoder

The decoder is the second part of the Transformer architecture. The decoder is responsible for the generation of the next token, conditioning the generation process on the source text information, encoded by the encoder (Sec. 3.1.4). The decoder input is represented by the partially generated text; in fact, text generation is performed auto-regressively, generating the next word starting from the previous ones. The module is composed of three submodules. The first one is a **masked multi-head attention** submodule, that is similar to the one present in the encoder architecture, with the addition of a **mask**, as anticipated in Sec. 3.1.3. At inference time, only the previously generated words will be available, so during training, to keep the same auto-regressive setting, the future ground truth words are masked. This means that, tokens at a given position can only attend to previous positions, avoiding to see into the future. In this way, the decoder is forced to rely only on unidirectional (right to left) attention. The actual mask is an upper triangular matrix that is applied to the scores (queries and keys dot-product output), before softmax operation (see Eq. (3.4)). The upper-right part is filled with a very small negative number (-10^9) that will become practically zero after softmax.
The second submodule is a standard multi-head attention module in which decoder information is combined with encoder information while the last submodule is again a feed forward layer (Eq. (3.9)). As inside the encoder module, each submodule has residual connections followed by layer normalization. Formally:

$$\begin{aligned} \text{DecoderLayer}(x, x_E) &= \text{LayerNorm}(\text{Sublayer}(x_{DCA}, \text{FFN}(x_{DCA}))) && (3.13)\\ x_{DCA} &= \text{LayerNorm}(\text{Sublayer}(x_E, \text{MHAttention}(x_{DSA}, x_E, x_E))) && (3.14)\\ x_{DSA} &= \text{LayerNorm}(\text{Sublayer}(x, \text{MHAttention}(x, x, x, mask))) && (3.15) \end{aligned}$$

where x_E is the information coming from the encoder, x_{DSA} is the decoder self-attention output, so the features coming from the first submodule, while x_{DCA} is the decoder cross-attention output, coming from the second submodule. Also in the case of the decoder, the same module is repeated 6 times, feeding the input of the next with the output of the previous (encoder output is fed the same way to all the decoder modules).

3.2 Transformers in Computer Vision

Following the great success of Transformer based architecture on the machine translation task and in general on Natural Language Processing tasks such as question answering, sentence classification, sentiment analysis or text summarization such as BERT [41], in the last few years a lot of effort has been dedicated into **adapting the Transformer architecture to Computer Vision tasks**. The main challenge is represented by the fact that such tasks require a network capable of dealing with **visual modality**, an extremely different modality with respect to language. To tackle this problem, one solution is to extract a set of different feature vectors from the image and to feed them to the Trasformer encoder the same way word tokens are fed in case of language modality. These feature vectors can be generated from dedicated networks that perform **object detection**, i.e. the task of detecting all the object/entities depicted in the image by producing a bounding box (also referred to as bbox) and a label for each of them. The feature extraction process needed for bounding box regression and object classification can be exploited by Transformer encoder networks that can use the intermediate features of such models to draw attention between different image regions, producing very informative encodings, useful for several downstream tasks. Another approach consists in feeding the Trasformer directly with raw or slightly processed image patches, as presented in Vision Transformer (ViT) [42], discussed in Sec 3.5.1.

3.2.1 Object detectors

As anticipated before, object detectors have a key role when using Transformers for Computer Vision. Intermediate per-region features are crucial to adapt the vision modality to the Trasformer input design. The way such features are extracted is highly dependent on the specific object detector architecture. One of the most used model for feature extraction is Faster R-CNN [26]. Faster R-CNN is a **two stage object detector** in which an input image is fed into a backbone network that extracts global feature maps. Then the feature maps are fed into a region proposal network that predicts a set of boxes containing objects. The feature maps extracted from the backbone are then ROI pooled with the coordinates of

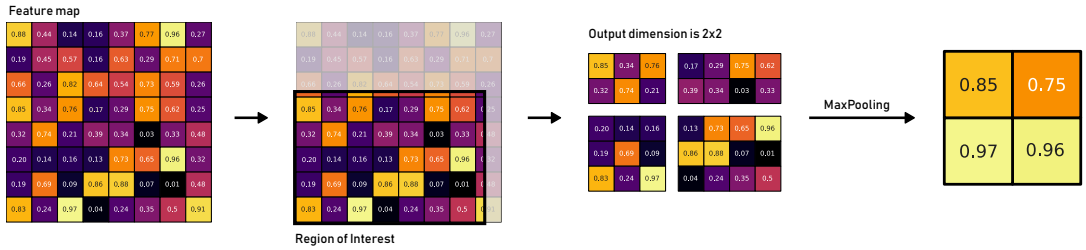


Figure 3.5: Region of Interest (ROI) pooling visualized, for a given feature map and an output dimension of 2×2.

the proposed boxes, obtaining a fixed size feature vector (2048 elements) for each proposal. In the original object detection task, such features are then fed to a classifier to predict the class probabilities. Considering that the fixed-size feature vectors are directly used to predict the class of the region we can argue that those are highly informative about their portion of the image. Therefore the latter can be used as the input of the Trasformer encoder.
The **Backbone network** usually is an image classification network, pretrained on ImageNet [13]. Those networks are extremely powerful and it is possible to exploit their image encoding capabilities by removing the last layer, used to compute the class probabilities, and serving to the rest of the network the global feature maps. In the case of Faster R-CNN, the backbone is a ResNet-101 [14], a very deep convolutional neural network that introduce residual connections to handle the problems of vanishing gradients and training accuracy degradation that arise especially in deeper architectures. Other common choices are VGG [15] and ResNeXt [16].
Region of interest pooling (**ROI pooling**) (originally introduced in [43]) is a specific kind of pooling that allows to obtain fixed size features starting from a feature map and the coordinates of a region. Traditional pooling is an operation widely used to decrease the dimensionality of the feature maps. It usually operates with a fixed sized kernel (ie. 2x2 or 3x3) that slides over the entire feature map, aggregating the results according to the type of pooling (max pooling takes the maximum of the values inside the kernel, average pooling takes the average). It works independently on all the channels. ROI pooling is particularly handful because it can operate on a region of the feature map and produce a fixed sized output regardless of the dimension of the proposed regions, that for an object detector could vary significantly. It works by dividing the region into a grid of subregions of the specified output dimension. Then max-pooling is applied on the feature maps, following the grid subregions previously considered. An example of ROI pooling is visualized in figure 3.5.

Single stage Object detectors

Two stage object detectors like Faster R-CNN [26] perform object detection in multiple steps. Firstly the image is fed into the backbone, then a separate network propose the regions that are used for ROI pooling on the backbone feature maps. This mechanism requires the model to look at the input image multiple times in different regions leading to high inference time. To solve this issue, Redmon et al. proposed, in 2016, You Only Look Once (**YOLO**) [44], a single stage object detector that looks at the image only once, producing bounding boxes and classifications end-to-end, directly from pixels. This mechanism allows the model to work real-time with very low latency. Bounding boxes coordinates are regressed directly from the whole input image along with an objectness score that indicate how much the model is confident that in that box there is actually an object. The image is divided into an $S \times S$ grid. If an object falls into a grid cell, that grid cell is responsible for its detection. Each grid cell regresses B bounding boxes and for each of them predicts also the class probabilities for C object classes ($C = 80$ classes in the original architecture). So the final output is an $B \times S \times S \times (5 + C)$ tensor where the 5 elements are 4 box coordinates plus the objectness score. In the first implementation, a custom network based on Googlenet [45] was used as the base feature extractor. An example about how YOLO works is illustrated in Figure 3.6. Further improvements were introduced with the following YOLO versions.
In **YOLOv2** [46], bounding box inference is based on anchor boxes. The model tries to predict offsets with respect to predefined boxes instead of regressing directly the coordinates. Input resolution is increased from 224×224 to 448×448 and batch normalization is introduced. Feature extraction is now performed by a new classification model called **Darknet-19**, pretrained on ImageNet [13] classification. Other than YOLOv2, a new model called **YOLO9000** is proposed, capable of performing object detection over 9000 classes. The model is trained jointly on classification and detection, combining ImageNet [13] (for classification only samples) and COCO [47] (for detection and classification) through hierarchical labels.
Other improvements and tweaks are reported in **YOLOv3** [48], along with a new bigger version of the feature extractor, **Darknet-53**.
On top of YOLOv3, Bochkovskiy et al. proposed **YOLOv4** [49], applying some modifications to the architecture, data augmentation strategies, losses, activation and so on. Changes are divided into two categories: *Bag of Freebies*, methods to increase accuracy without inference cost such as data augmentation and *Bag of Specials*, plugin modules and post processing methods to increase accuracy with a little inference cost. The resulting model is better and faster, still working real-time.
Finally, in 2021, Wang et al. proposed **Scaled-YOLOv4** [50]. The paper focuses on the design of a model scaling method and a strategy for scaling large object

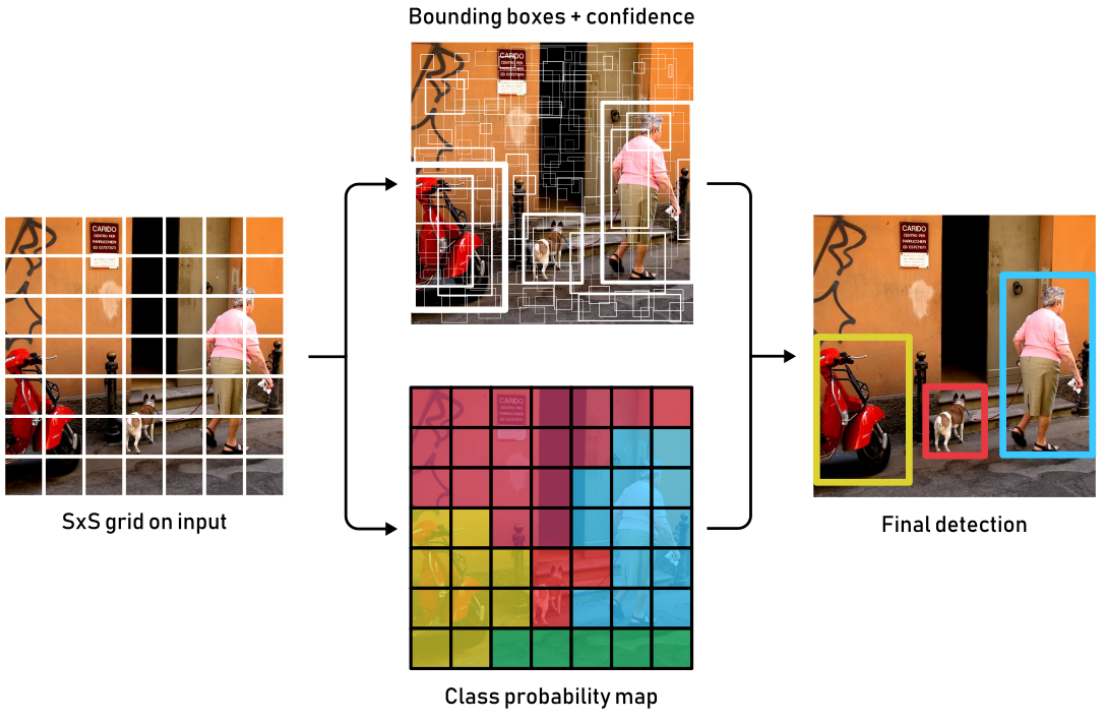


Figure 3.6: YOLO detection pipeline. All the regions are detected in one forward pass. Firstly the input image is divided into an $S \times S$ grid (**left**). Then, each grid cell predicts B bounding boxes (**center-top**) and confidence for those boxes (indicated with box thickness). At the same time, C class probabilities are predicted for each grid (most probable classes are highlighted in **center-bottom** figure). Such predictions are then combined and filtered to produce the final detections (**right**). Visualization inspired by original YOLO paper [44].

detector models, proposing a set of networks for different input resolutions. Starting from YOLOv4, a technique called Cross Stage Partial Network (CSP) [51] is applied to reduce the number of computations. The main idea behind CSP is that the amount of computation of a given CNN stage can be reduced by splitting the feature maps. One part is fed to the actual stage, followed by a transition layer. The transition layer output is then concatenated with the remaining part of the input feature maps. Applying CSP to YOLOv4 Darknet allowed Scaled-YOLO models to achieve SoTA results on COCO [47] minval set.

3.2.2 Computer vision proposed architectures

Following the aforementioned architectures, based on region features extracted from object detectors, different solutions have been proposed to solve Vision-Language

tasks. Some of them are specifically conceived to solve Image Captioning while others aim at obtaining a pre-trained model on Visual-Language understanding that then can be fine-tuned for different downstream tasks.

The image Transformer

The Image Transformer [52] aims at adapting the original Transformer architecture to the image captioning task through a modified encoder and an implicit decoder. The main **encoder** modification is related to the introduction of adjacent matrices. Adjacent matrices model the relationships between the regions according to the overlap between them. For each pair of regions, three possible relationships can exist: neighbourhood, parenthood and childhood. Considering two regions, l and m, if l contains m for the most part (90%), the relationship is parenthood; if instead it is m the region containing the most part of l, the relationship is childhood; neighbourhood if none of the above. Formally:

$$\begin{aligned}
&\Omega_p[l,m] = \begin{cases} 1, & \text{if } \frac{\text{Area}(l \cap m)}{\text{Area}(l)} \geq \epsilon \text{ and } \frac{\text{Area}(l \cap m)}{\text{Area}(l)} > \frac{\text{Area}(l \cap m)}{\text{Area}(m)} \\ 0 & \text{otherwise} \end{cases} \\
&\Omega_c[l,m] = \Omega_p[m,l] \\
&\Omega_n[l,m] + \Omega_p[l,m] + \Omega_c[l,m] = 1
\end{aligned} \tag{3.16}$$

where $\epsilon = 0.9$. Ω_n, Ω_p and Ω_c are respectively neighbourhood, parenthood and childhood matrices, with same number of rows and columns, equal to the number of regions.
Those matrices are then used as spatial hard attention applied to the output of each layer encoder. Inside the encoder, three sets of keys and values are created, and the three outputs are weighted by the three adjacent matrices. Finally, the three weighted outputs are summed. Formally, reformulating Equations (3.4) and (3.11):

$$\begin{gathered}
\text{Attention}(Q, K_i, V_i) = \Omega_i \circ \text{Softmax}\left(\frac{QK_i^T}{\sqrt{d}}\right) V_i \\
\text{with } \; i \in \{p, n, c\}
\end{gathered} \tag{3.17}$$

$$x_{EA} = \text{LayerNorm}\left(\text{SubLayer}\left(x, \sum_{i \in \{p,n,c\}} \text{MHAttention}(Q, K_i, V_i)\right)\right) \tag{3.18}$$

where x_{EA} is the output of the multi-head attention, x is the input, represented by the regions features, MHAttention is the concatenation of the attentions of each head, defined in Eq. (3.7) while SubLayer is a residual connection, defined in Eq.

(3.12).
The Image Transformer **decoder** is composed of an LSTM [17] layer and an implicit Trasformer decoding layer. The LSTM works as a memory, it receives as an input the mean of the encoder output, the previous context vector and the embedding of the current word. The LSTM output is then transformed linearly and used as the query of the implicit decoding layer. Also here, different sub-transformers are added in parallel and the output of them is then averaged and passed through a gated linear layer (GLU) [53] to obtain the new context vector that will be both used for predicting word probabilities and fed to the LSTM at the next timestep.

Meshed-Memory Transformer

Meshed-Memory Transformer [54] is an architecture based on the original Transformer in which the main novelty is related to the introduction of additional memory vectors inside the encoder. The sets of keys and values are extended with additional learnable weights that do not depend on the input. Formally, reformulating Eq. (3.8):

$$\begin{aligned} head_i(\boldsymbol{X}) &= \text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) \\ \boldsymbol{Q} &= \boldsymbol{X}W_i^Q \\ \boldsymbol{K} &= [\boldsymbol{X}W_i^K, \boldsymbol{M}_i^K] \\ \boldsymbol{V} &= [\boldsymbol{X}W_i^V, \boldsymbol{M}_i^V] \end{aligned} \tag{3.19}$$

where $\boldsymbol{M}_i^K$ and $\boldsymbol{M}_i^V$ are matrices, concatenated to the actual Keys and Values, obtained by linearly projecting the input vector $\boldsymbol{X}$ with projection matrices W_i^K and W_i^V where i indicates the head index. Queries $\boldsymbol{Q}$ are the same as the original formula, projecting the input with matrix W_i^V. Those weights act as a sort of **prior knowledge**, not embedded into the input. For example, given two regions depicting one person and a tennis racket, the concept of *match* or *sport* can be learnt by input-independent weights that model the relationships between those two regions encodings. Having traditional Keys and Values only, this relationship is difficult to be handled.
The second major modification is on the decoder side. Instead of feeding each decoder layer only with the output of the last encoder layer, the decoder can take advantage from the multi-level representation of the region relationships by being fed with all the encoder layers outputs. Inside each decoder layer, cross-attention is applied for each set of Keys and Values coming from the different encoder layers and the outputs are summed together after being modulated. Modulation is crucial because at a certain decoding layer, some piece of information could be more useful than others. This operation is called **gated cross-attention** and can be defined

as

$$head_i(\tilde{X}, x_{DSA}) = \sum_{j=1}^{N} \alpha_j \odot \text{Attention}(x_{DSA}W_i^Q, \tilde{X}^j W_i^K, \tilde{X}^j W_i^V) \tag{3.20}$$

where $\tilde{X}^j$ indicates the encoding coming from the j^{th} encoder layer, x_{DSA} is the result of the decoder self-attention defined in Eq. (3.15) and i is the index of the attention head.
α_j is the **modulation parameter** for the j^{th} encoder layer and is computed by considering the relevance between the input query, coming from the decoder self-attention, and the cross-attention performed between the aforementioned query and the j^{th} encoder layer output.

Oscar

Object-Semantics Aligned Pre-training for Vision-Language Tasks (Oscar) [55] is a Vision-Language pretraining model based on the Transformer encoder. Oscar is based on two main stages, the first one is the **Vision-Language pretraining** (VLP), when the model learns generic representations from image-text pairs and develop a so-called Vision-Language understanding. The second stage consists of a **specific fine tuning** that aims at specialize Oscar to solve a specific Vision-Language downstream task. Those tasks can be Image Captioning but also Image and Text retrieval, Visual Question Answering or Novel Image Captioning. The architecture is initialized with weights of the BERT architecture [41], used with textual only modality and then adapted to multimodality tasks. The input is composed of three main parts, **word tokens** coming from the input text, **object tags** and **region features** that are both extracted from the image by object detectors. The introduction of **object tags** is one of the main novelties and is motivated by the fact that, often, the salient objects depicted in the image are mentioned is the corresponding caption. So, object tags act as a sort of *anchor point* between the two modalities, the language modality and the image modality, making easier the learning of image-text alignment. There are two pretraining objectives, the first one is *Masked Token Loss* and consists of predicting randomly chosen word tokens that are masked on the input side, starting from the other tokens and the image features and tags. The second one is *Contrastive Loss* in which, with a certain probability, the image representation is "polluted" by replacing the object tags with other tags from the dataset. Then the model is asked to predict whether or not the tags have been "polluted" with a binary classification. The finetuning procedures are specific for the actual V+L (Vision-Language) task and, according to it, some of the input can be missing (i.e. Language modality is missing in case of Image Captioning).

VinVL

In VinVL [56], the Vision-Language model is defined as the interaction of two modules, the **Vision** module and the cross-modal understanding module (**VL**). **Vision** module is usually an object detector that takes the image as input and outputs region features (and tags in the case of Oscar). The **VL** module, starting from image information extracted by the **Vision** module and a natural language input, produces the task-dependent model output.
VinVL aims at improving the Oscar pretraining model by increasing the amount of data used for training and focusing on the object detector (the **Vision** module), often treated like a blackbox and kept identical while proposing newer architectures. The major novelty is indeed the backbone of the object detector that in the case of VinVL is a ResNeXt-152 C4 [16] and is trained on different datasets (COCO [47], Objects365 [57], OpenImagesV5 [58] and VisualGenome [11]). Another improvement is related to the introduction of Oscar+, a better version of the cross-modal understanding module (**VL** module) in Oscar in which one of the two pretraining objectives, *Contrastive Loss*, is substituted with a *3-way Contrastive Loss*. Two types of training samples are taken into account, one are the caption-image samples while the second are the question-answer-image samples. In Oscar+, "pollution" is done by constructing negative examples (unmatched triplets), that for the caption-image samples consists of wrong captions while for the question-answer-image consists of a wrong answer. As for Oscar, the model is asked to predict whether or not the sample has been polluted with a binary classification.

3.3 Transformers for Image Retrieval

Image Retrieval task, as anticipated in Sec. 1.2.2 and Sec. 2.1 consists in finding relevant images according to a **query**. What happens in practice is that, starting from a pool of images and a query, that can come from a different modality (natural language) or the same modality (another image), a **similarity score** is evaluated for each query-image pair. Then, according to such scores, images are sorted and the top K are returned as a result. The Image Retrieval model acts like a similarity function S that take as input the query-image pair and returns a number between 0 and 1:

$$S : (\Omega_I, \Omega_Q) \rightarrow \{x \in \mathbb{R} \mid 0 < x < 1\} \tag{3.21}$$

where Ω_I is the image sample space and Ω_Q is the query sample space that, according to the modality, could coincide or not with Ω_I.

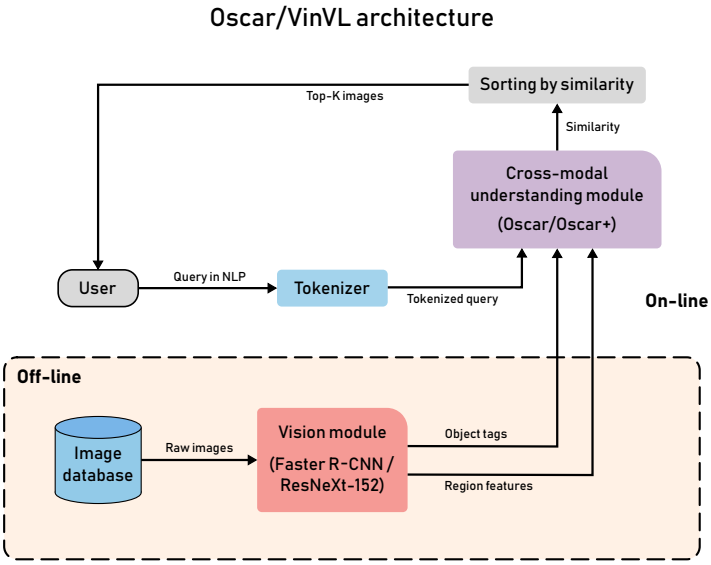


Figure 3.7: Oscar/VinVL architecture.

3.3.1 Oscar/VinVL proposed architecture

Oscar and VinVL are two examples of pretrained models on Vision-Language tasks. It means that, after a specific **finetuning**, those models are able to solve a multitude of Vision-Language tasks, Image Retrieval included. In this case, finetuning consists in a binary classification problem in which the model is asked to predict whether or not a image-caption pair is correct, in which the final representation of the [**CLS**] token is used as input to the classifier. [**CLS**] stands for **classification** token and in the original BERT architecture [41] was a special token used for sentence-level representation added at the beginning of the input sequence. The output of the classifier is a number between 0 and 1 that represent the **similarity** between the image and the query. See Figure 3.7 for the complete architecture.

3.3.2 Efficiency issues in large-scale systems

Image Retrieval is widely used in search engines and the quality of the results is inevitably dependent on the cardinality of the image pool. Large-Scale systems introduce new challenges related to efficiency. For this reason is important to evaluate how well the models **scale with images**. Oscar/VinVL architecture performs Image and Text Retrieval by feeding the model with both the inputs at the same time, leveraging mutual self-attention on the two modalities. This means that, for each query-image pair, a new inference needs to be done. Formally, if Q

is the set of all the queries and I is the set of all the images (the pool):

$$\#\text{inferences} = |Q| \times |I| \tag{3.22}$$

where $|\cdot|$ indicated the cardinality, or if we consider a running application in which queries are prompted by the users, the unit cost for each query is linear to the image pool size:

$$\#\text{inferences}_{query} = |I|. \tag{3.23}$$

Consequently, large-scale image databases make the Image Retrieval task infeasible while using those architectures.

3.4 Transformer Encoder Reasoning Network

Transformer Encoder Reasoning Network (TERN) [6] is a Transformer based architecture designed by Messina et al. in 2021 for image-text matching and retrieval, with a particular focus on efficiency in large-scale contexts.

3.4.1 Architecture

TERN architecture is composed of **two branches**, one for the language modality input and one for the visual modality input. As for previous models, visual input consists of region features, bounding box coordinates and area while language input consists of word tokens. The two branches are initially independent while the last layers are shared. Text processing inside the language branch is done by BERT [41], while for the visual branch, a standard 4-layers Trasformer encoder is used. The goal is to obtain, at the end of the two branches, two representations that are **comparable**. To enforce this constraint, both branches are connected to a 2-layers encoder with shared weights that outputs the final embeddings of 1024 elements. To perform image or text retrieval, the embeddings of all the items in the pool (images or texts according to the task) are computed and compared with the embedding of the query, by a similarity measure defined on the common space. In this specific case, the chosen similarity measure S is the **cosine similarity**, defined as

$$S(i, c) = \frac{\boldsymbol{i} \cdot \boldsymbol{c}}{\|i\|\|c\|} \tag{3.24}$$

where i and c are the pair of image and caption embedding.
The output of **cosine similarity** is a number between -1 and 1 that represents the semantic affinity between the image and the text. In the case of image retrieval, the query will be a textual description while the pool will be the entire set of

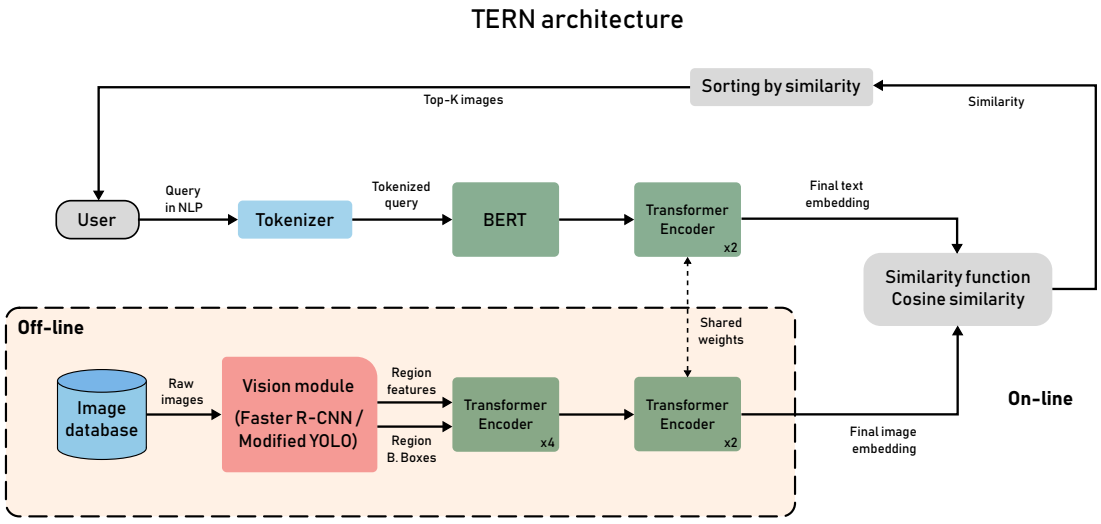


Figure 3.8: TERN architecture.

available images. Sorting images according to the similarity with respect to the query will give us a **ranking**, based on relevance. See Figure 3.8 for the complete architecture.

3.4.2 Shared embedding space

During inference, for a given image-caption input, the trained model should be able to output two representations that are very similar, while uncorrelated pairs should correspond to very different representations. In other words, image and text embedding must share the same **embedding space**. The embedding space can be imagined as a 1024-dimensional space in which every image or text is represented by a point. In such space, images and captions representing the same concept should be very close, regardless of the actual modality. Therefore, the objective of the training is to **align** the representations coming from the two branches. An overview of the general idea is reported in Figure 3.9. With those settings, retrieving the most relevant images according to a textual query corresponds to finding the closest "image points" to the "query point". The concept of distance is modeled by the similarity function mentioned above in Eq. (3.24).

3.4.3 Normalized Discounted Cumulative Gain

The most common metric for Image Retrieval task, as stated in subsection 2.1.1 is **Recall@K** that reports how frequent the original image is found in the top K ranking, given its caption as input query. Considering a search engine application,

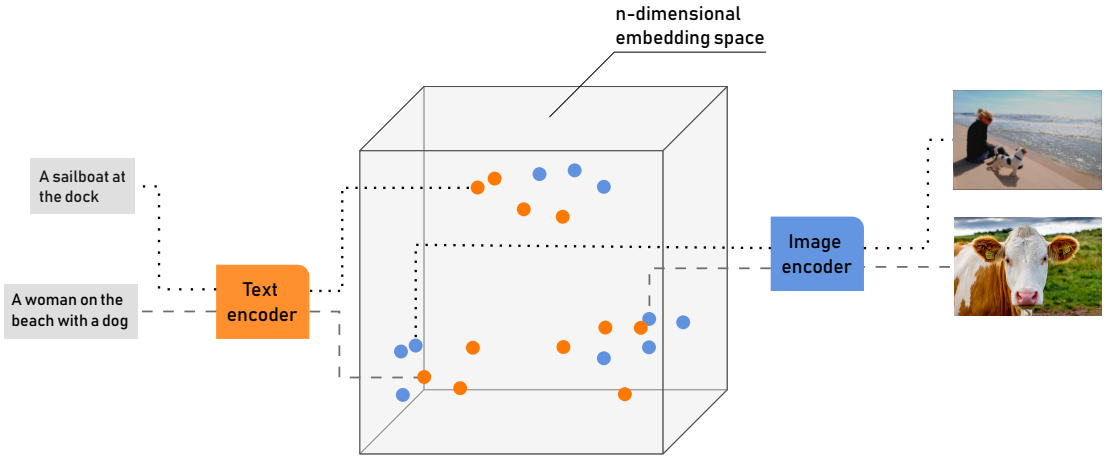


Figure 3.9: Example of n-dimensional shared embedding space (reported here as 3-dimensional to ease visualization). Very similar concepts have very close embeddings, regardless of the input modality, while different concepts tend to be far away. The training objective is to "align" the embeddings for similar concepts while pushing away the others.

the user is interested in finding relevant images and the presence or not of the "exact match" among the first results is not crucial. For this reason, the Recall@K metric often results to be **too rigid**, penalizing too much rankings in which images are very relevant but not "exact matches". To tackle this problem, Normalized Discounted Cumulative Gain (**NDCG**) is introduced. NDCG evaluates the quality of the first p positions of the ranking based on a computed relevance. The non normalized DCG is the sum of the first p images relevances, divided by the logarithm of the ranking position, formally:

$$\mathrm{DCG}_p = \sum_{i=1}^{p} \frac{\mathrm{rel}_i}{\log_2(i+1)} \tag{3.25}$$

where rel_i is the relevance of the i-th position image and can be seen as a sort of *ground truth* similarity. NDCG is obtained by normalizing the DCG by the IDCG, i.e. the Ideal Discounted Cumulative Gain that consists of the DCG of the **best possible ranking**, obtained by sorting the images according to the relevance with the query. Thus, NDCG is defined as

$$\mathrm{NDCG}_p = \frac{\mathrm{DCG}_p}{\mathrm{IDCG}_p} \tag{3.26}$$

The resulting score is constrained in the interval [0, 1]. Relevance can be a specific function of the image-query pair. In this specific case, for efficiency reasons,

relevance is computed by applying a function that takes as input the query and the captions associated with the retrieved image. In the original implementation, both `ROUGE-L` and `SPICE` are used as function of the query-captions pair and the relevance matrices are pre-computed and stored for the entire COCO dataset.

3.4.4 Hinge-based triplet ranking loss

The loss introduced during training is the Hinge-based triplet ranking loss. This is a particular loss that focuses on the hard negatives and is defined as

$$\begin{aligned} L_m(i,c) = &\max_{c'} \left[\alpha + S(i,c') - S(i,c)\right]_+ + \\ &\max_{i'} \left[\alpha + S(i',c) - S(i,c)\right]_+ \end{aligned} \tag{3.27}$$

where $[\cdot]_+ = max(0,x)$, $S(i,j)$ is the similarity function between image i and caption c embeddings and the hard negatives i' and c' are given by

$$\begin{aligned} i' &= \underset{j \neq i}{\text{argmax}}\, S(j,c) \\ c' &= \underset{d \neq c}{\text{argmax}}\, S(i,d) \end{aligned} \tag{3.28}$$

Basically this loss penalizes the cases in which exists at least one other example whose similarity is higher with respect to the input. In fact i' is the most similar image (different from i) to the caption while c' is the most similar caption (different form c) to the image. If instead, i and c are the best matching according to similarity, $S(i,c') - S(i,c) < 0$ and $S(i',c) - S(i,c) < 0$, making the loss lower.

3.5 Contrastive Language-Image Pre-Training

Contrastive Language-Image Pre-Training [21] (CLIP) is an architecture by OpenAI introduced in 2021. The main idea is to train visual models with natural language supervision instead of using fixed classification labels to increase generality and usability. CLIP is trained on a large-scale image-text dataset and the generalization capabilities are verified on different benchmarks in a *zero-shot* setting. This means that at test time, inference is done on classes that the model has never seen before. Natural language descriptions are able to express a wider set of concepts but the nature and the complexity of such labels make it difficult to leverage this supervision. CLIP is trained on the introduced WebImageText (**WIT**), a image-text pairs dataset collected from a variety of publicly available sources on the Internet. Pretraining is done by trying to predict which text *as a whole* is paired with which image, in contrast with another common approach aiming at predicting

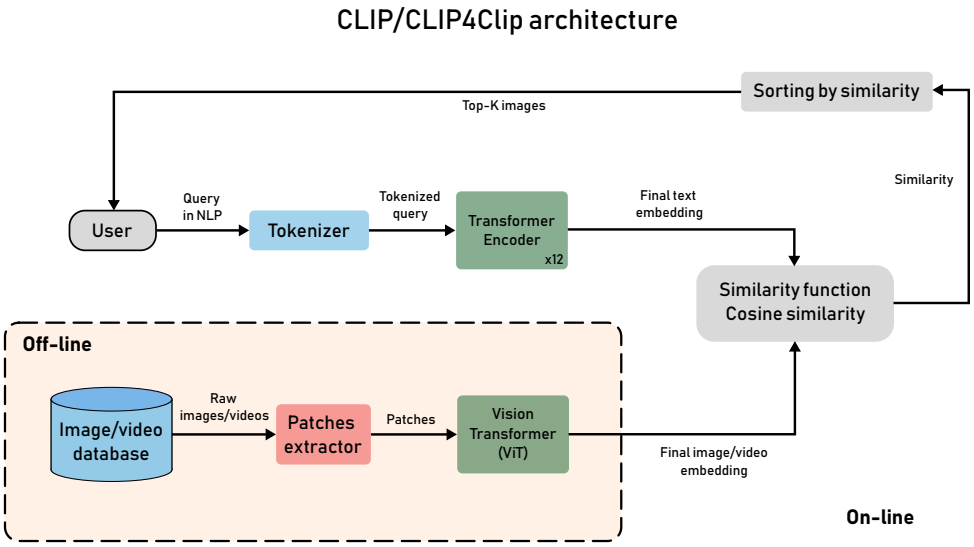


Figure 3.10: CLIP/CLIP4Clip architecture.

the exact words separately. So, considering a batch of N image-text pairs, CLIP is trained to predict which one of the $N \times N$ possible image-text matching actually occurred. This is done by learning a multi-modal embedding space (see Sec. 3.4.2), training simultaneously the image encoder and the text encoder. The objective is maximizing the **cosine similarity** (see Eq. (3.24)) of the embeddings for the actual N correct image-text pairs while minimizing it for the other $N^2 \times N$. The architecture is very similar to the one presented in TERN in Sec. 3.4, with a **text encoder**, responsible for the computation of the query embedding, and an **image encoder**, responsible for the image embedding. The similarity function is applied on the projected embeddings and multiplied by a scale value (temperature) learned during training. In the base implementation the **text encoder** is composed of a 12 layers Transformer encoder [5] with a context vector of 512 elements and 8 attention heads. Regarding the **image encoder**, different experiments have been performed. See Figure 3.10 for the complete architecture.

3.5.1 Vision Transformer

Inside CLIP architecture, a total of eight different image encoders have been tested. All of them can be grouped into two categories: **ResNets** [14] and **Vision Transformers** (ViT) [42]. ResNets have been already mentioned in Sec. 3.2.1 while Vision Transformer is an architecture based on the Transformer encoder [5] adapted for visual input. Vision Transformer aims at solving image classification task and is tested on different datasets. It works by dividing the input image

into a fixed number of **patches** that are then fed to the Transformer encoder as tokens. Patches are extracted by a convolutional layer and the positional encoding, added to the patches, is a learnable vector. An additional token (similar to `[CLS]` token, mentioned in Sec. 3.3.1 for Oscar) is prepended to the input sequence and its state at the output serves as the image representation. In the original paper, different patch sizes are evaluated, but in the base CLIP implementation, 32x32 pixel patches are used. The base architecture is a 12 layers Transformer encoder with a 768 elements context vector and 12 attention heads (**ViT-B/32**). Input resolution can vary but the number of patches changes consequently. Transformer encoder can handle arbitrary sequence length but the positional encoding vector needs to be changed accordingly. (2D interpolation, as explained in CLIP paper [21] and visualized in Figure 4.11).

3.6 Video Retrieval: from CLIP to CLIP4Clip

The impressive performance of CLIP on a variety of different datasets and tasks pushed further studies on the possible application of such architectures in video-text settings. CLIP4Clip [7] is an architecture released in 2021 by Luo et al. aiming at investigating the use of CLIP in a video retrieval setting. The architecture, as the name suggests, is based on the original CLIP implementation, with the introduction of three different **similarity calculators**, that take into account the textual embedding (same as for CLIP) and the visual embeddings (one embedding produced for each input frame).

3.6.1 Similarity calculator

Parameter-free type. In the parameter-free type, frame embeddings are directly aggregated via **mean-pooling** (Eq. (3.30)) and similarity is defined as the **cosine similarity** (Eq. (3.29)) between text embedding $\mathbf{w}_j$ and the aggregated video embedding $\hat{\mathbf{z}}_i$. This is the simplest one since it doesn't require further parameters to train:

$$S(v_i, t_j) = \frac{\mathbf{w}_j^\top \hat{\mathbf{z}}_i}{\|\mathbf{w}_j\| \|\hat{\mathbf{z}}_i\|} \tag{3.29}$$

$$\hat{\mathbf{z}}_i = \texttt{mean-pooling}\left(\mathbf{z}_i^{\mathbf{1}}, \mathbf{z}_i^{\mathbf{2}}, \ldots, \mathbf{z}_i^{|v_i|}\right) \tag{3.30}$$

$\mathbf{z}_i^{\mathbf{1}}, \mathbf{z}_i^{\mathbf{2}}, \ldots, \mathbf{z}_i^{|v_i|}$ are the frame embeddings, extracted from ViT, while $|v_i|$ indicates the total number of sampled frames.

Sequential type. In the sequential type, the main difference with parameter-free type is the way frame embeddings are aggregated. Mean pooling ignores

the sequential information. To inject into the frame embeddings the **temporal information**, two models are experimented for the sequential type. The first one is the LSTM [17] while the second is the Transformer encoder [5]. Frame embeddings are fed to the sequential model and the outputs are then mean-pooled to obtain the final video embedding:

$$\mathbf{Z}_i = \left\{\mathbf{z}_i^{\mathbf{1}}, \mathbf{z}_i^{\mathbf{2}}, \ldots, \mathbf{z}_i^{|v_i|}\right\} \tag{3.31}$$

$$\tilde{\mathbf{Z}}_i = \texttt{LSTM}\left(\mathbf{Z}_i\right) \quad \text{or} \quad \tilde{\mathbf{Z}}_i = \texttt{Transformer-Enc}\left(\mathbf{Z}_i\right) \tag{3.32}$$

$$\hat{\mathbf{z}}_i = \texttt{mean-pooling}\left(\tilde{\mathbf{Z}}_i\right) \tag{3.33}$$

Similarity function is again the cosine similarity.
Tight type. In the tight type, similarity is evaluated in a completely different manner. Text and frame embeddings are concatenated and fed to the same Transformer encoder. Then, two linear projection layers are applied on the first output token to obtain the final score. This is the type that requires the highest amount of uninitialized weights:

$$\mathbf{U}_i = \left[\mathbf{w}_j, \mathbf{z}_i^{\mathbf{1}}, \mathbf{z}_i^{\mathbf{2}}, \ldots, \mathbf{z}_i^{|v_i|}\right] \tag{3.34}$$

$$\tilde{\mathbf{U}}_i = \texttt{Transformer-Enc}\left(\mathbf{U}_i + \mathbf{P} + \mathbf{T}\right) \tag{3.35}$$

where $[,]$ denotes concatenate operation. P represents the positional encoding (as described in Sec. 3.1.2) and T represents the type embedding, similar to Segment embedding in BERT [41]. Type embedding contains two types of embeddings, one for text embedding input token and one for frame embedding input tokens. Then similarity is evaluated in this way:

$$S(v_i, t_j) = \texttt{FC}\left(\texttt{ReLU}\left(\texttt{FC}\left(\tilde{\mathbf{U}}_i[0,:]\right)\right)\right) \tag{3.36}$$

where FC indicates the linear projection while ReLU is the ReLU activation function:

$$\texttt{ReLU}(x) = \max\left(x, 0\right) \tag{3.37}$$

Considering the large-scale setting of the application, the **parameter-free type** seems to be the best choice, considering that no further inference is needed. Moreover, from the results reported in the CLIP4Clip paper, parameter-free similarity achieve SoTA results on the 7k split of MSR-VTT [59] dataset. Therefore, for the following experiments, this similarity method will be used.

3.6.2 Loss function

Considering the similarity function $S(v_i, t_j)$, the model is trained with a symmetric cross entropy loss over the similarity scores:

$$\mathcal{L}_{v2t} = -\frac{1}{B}\sum_{i}^{B}\log\frac{\exp\left(S(v_i,t_i)\right)}{\sum_{j=1}^{B}\exp\left(S(v_i,t_j)\right)} \tag{3.38}$$

$$\mathcal{L}_{t2v} = -\frac{1}{B}\sum_{i}^{B}\log\frac{\exp\left(S(v_i,t_i)\right)}{\sum_{j=1}^{B}\exp\left(S(v_j,t_i)\right)} \tag{3.39}$$

$$\mathcal{L} = \mathcal{L}_{v2t} + \mathcal{L}_{t2v} \tag{3.40}$$

where $\mathcal{L}$ is the final loss, computed by summing the text-to-video loss $\mathcal{L}_{t2v}$ to the video-to-text loss $\mathcal{L}_{v2t}$.

3.6.3 Frame sampling

When dealing with videos, during data preparation and preprocessing, one important step is the sampling strategy. In MSR-VTT [59] dataset, used for CLIP4Clip finetuning, the majority of the videos has a framerate around 30 fps. Feeding the model with all the frames could result in overfitting and a large amount of computation. Moreover, an high sampling frequency can lead to reduntant input information. For these reasons, input clips are sampled at 1 frame per second, and considering the average clip length (from 10 to 30 seconds) the number of input frames will be manageable. The order in which frames are fed to the ViT [42] can be discussed. In CLIP4Clip paper, different options are evaluated.
Head: sample the first $|v_i|$ frames at the beginning of the video, **Tail**: sample the last $|v_i|$ frames at the end of the video, **Uniform**: sample uniformly $|v_i|$ video frames. $|v_i|$ is the notation used in Eq. (3.30) to indicate the total number of sampled frames.

3.6.4 2D/3D patches

Considering that the input is composed of a temporal sequence of frames instead of just one as in the case of CLIP, patches need to be generated starting from a 3D input instead of 2D. Two different methods are discussed in the paper. The first method consists in embedding each frame patch independently, ignoring temporal information. The second method consists in applying a 3D linear projection instead of 2D, enhancing temporal feature extraction. From the reported results, the 2D method seems to be still the best choice, despite the additional temporal information provided by 3D projections. This demonstrates the difficulty in training new parameters from scratch. Therefore in the following experiments, 2D patches will be considered.

3.7 Efficiency in Large-scale retrieval systems

While Oscar/VinVL architectures requires the image-text pairs to be fed together to produce a similarity score, having two different processing pipelines and producing two independent representations, TERN and CLIP are able to **work asynchronously on the modalities**, and similarity can be computed with a simple function applied on **pre-computed embeddings**. This means that a large amount of images can be processed only once, having their embeddings cached and reused each time a new query comes out. The resulting number of inferences, given a set of queries Q and a set of images I, is summarized in Table 3.1, in which are also reported the values for Oscar-VinVL in Equations (3.22) and (3.23).
The clear advantage in terms of efficiency doesn't come at no cost. In fact, isolating

Architecture	#inferences	per-query
Oscar[55]/VinVL[56]	$\|Q\| \times \|I\|$	$\|I\|$
TERN[6]/CLIP[21]	$\|Q\| + \|I\|$	1

Table 3.1: Number of total inferences and per-query inferences for Oscar/VinVL and TERN/CLIP architectures. CLIP values are also valid for CLIP4Clip, given that the architecture is the same

the modalities precludes the mutual self-attention between image and language, present in the Oscar/VinVL architecture. Nevertheless, for TERN architecture, the results on the COCO test-set reported in the paper [6] show that the performances are still comparable or slightly worse than SoTA on Image Retrieval, suggesting that the model is able to compensate for the lack of mutual self-attention. CLIP instead demonstrates that the object detector is not essential, since great results can be obtained by processing image patches directly with the Transformer encoder [42], even with zero-shot transfer only. This, combined with the great scalability and efficiency, makes these architectures a solid choice for the Scene Retrieval task, discussed in detail in Chapter 4.
Regarding CLIP4Clip, same considerations can be done. The architecture is the same presented for CLIP, so retrieval can be done asynchronously, being able to scale efficiently. Lack of cross-modal mutual self-attention could be an issue but the results show that CLIP4Clip achieve SoTA on different Video Retrieval Benchmarks such as MSR-VTT 7k-split[59], MSVD [60], LSMDC [61], ActivityNet [62], and DiDeMo [63]. For this reasons, CLIP4Clip is also a solid choice considering the final application and will be discussed too, in Chapter 4.

Chapter 4

Scene retrieval from video

4.1 Introduction to the task

While in traditional Image Retrieval and Video Retrieval tasks, the goal is to rank a series of heterogeneous candidate images/videos according to a query, the specific scene detection application brings in new settings and requirements. The ever increasing amount of video data available online or produced by security cameras often requires tons of hours of human supervision to be usable and valuable. A valid example is in the surveillance field in which the so called "human in the loop" is crucial to assess safety and effectiveness of the system. Anomaly detection algorithms go in this direction, trying to partially substitute or aid the human intervention in this specific field. On another side, the rapid growth of video contents posted on social-media platforms introduced new challenges related to content moderation that often is carried out by user reports. Being able to automatically analyze and moderate content before publication can be a good way to tackle the problem without relying on the customer's judgment. For this reasons, the final task can be seen as the combination of Video Retrieval and Anomaly Detection, applying the multi-modal Vision-Language setting to a possibly fixed-camera environment with predefined anomaly descriptions. The resulting task can be broken down into two different modes or use cases, **Off-line scene retrieval** and **On-line scene detection**.

4.1.1 Off-line scene retrieval

Off-line scene retrieval is the first use case. Similar to the traditional Image/Video retrieval task, the application allows the user to query a (possibly very long) video with a **scene**, described with natural language. Visual and Language branches need to be independent to ensure the model to scale with the size of video data as discussed in Sec. 3.3.2 and Sec. 3.7. The application returns the top K most

relevant candidates according to the query, along with timestamps. In case of Image retrieval models, a candidate is a frame of the input video, while for Video retrieval models, each candidate is a fixed-size subsection of the video (clip).

4.1.2 On-line scene detection

On-line scene detection is the second use case. The application asks the user to prompt textual descriptions for a series of scenes representing anomalies. For example, if the camera points to a sidewalk, some anomalies could be "a person is laying on the ground", "a person is riding a bike", "a crowded sidewalk" or "a car on the sidewalk". The system notifies the user each time one of the anomalies is triggered. As for Off-line scene retrieval, the visual input depends on the visual modality (image or video) and can be the last captured frame or the last time window. This mode requires the application to work in real-time. For this reason, the proposed models will be evaluated also on the number of inferences per second.

4.1.3 Related experiments

Considering the two previously discussed modes, a series of experiments can be conducted, starting from the original architectures of **TERN** and **CLIP4Clip**. Focusing on **real-time capabilities**, the original TERN object detector (Faster R-CNN [26]) is substituted with a faster one-stage detector, Scaled-YOLOv4 [50]. The goal of the experiment is to **adapt** such architecture to be compatible with TERN, without an excessive metrics drop, while maintaining the inference speed boost. For the video retrieval approach, particular attention will be given to **input resolution**. Firstly the original low resolution model will be trained through the specific finetuning. Then, the architecture will be **altered** to allow the model to be finetuned on higher resolution samples. The goal of the experiments is to investigate the impact of the input size on the metrics and the latency. Lastly, to compare the two approaches, a common dataset will be created starting from security camera recordings and used as a benchmark to test all the previously mentioned models. The goal of this final experiments is to test the behaviour of such models on a different domain, evaluating the possibility for the final application to work in different contexts, **without any finetuning**.

4.2 Datasets

This section reports all the used datasets for image and video retrieval experiments.
MS COCO [47] is the image retrieval dataset used to finetune YOLO after the archiecture modification. The same dataset is then used to train TERN.
MSR-VTT [59] is the video retrieval dataset used to finetune CLIP4Clip model.

(a) **(b)** **(c)** **(d)**

Figure 4.1: Examples of COCO dataset annotations. For each image there are **(a)** 5 captions, **(b)** segmentation coordinates for each instance, **(c)** bounding boxes for each instances and **(d)** keypoint coordinates for each person.

To perform the experiments with higher resolution samples, an HD version of the dataset is created, **MSR-VTT HD**.
Solferino is the common retrieval dataset, created from security camera recordings, used to test all the models, without further training.

4.2.1 MS COCO

Microsoft Common Objects in Contexts (MS COCO) [47] is a large-scale dataset for Image Captioning, Object Segmentation and Object and Keypoint Detection. It contains 2.5 millions of labeled instances and 250000 people with keypoints in a total of 328000 images. There are 5 captions for each image and 91 common object categories. Considering that MS COCO contains both bounding boxes and captions (an example is reported in Figure 4.1), it is used to train both YOLOv4 and TERN. YOLOv4-CSP [50] is the Object detector model used for image feature extraction inside TERN and is finetuned on COCO bounding boxes and classes. Then, starting from features extracted by YOLOv4-CSP, TERN is trained for Image Retrieval on COCO captions. Further details can be found in Sec. 4.3.1.

4.2.2 MSR-VTT

MSR-VTT (**MSR**-**V**ideo **t**o **T**ext) [59] is a large-scale video benchmark for video understanding, and in particular video captioning. It consists of 10000 web video clips (a total of 41.2 hours) annotated with 20 captions each. The framerate is mostly around 30 and 25 frames per second and the resolution is 320×240. As

every video captioning dataset, it can be used to train also video retrieval models. In this work, MSR-VTT is the finetuning dataset for Clip4CLIP [7]. In particular, the 7k split is used as training set and all the metrics are evaluated on the 1k test set split.

MSR-VTT HD

The main drawback of MSR-VTT is the video resolution. It can be sufficient if the subject is centered and on the foreground but can be a problem if the input video is coming from a security camera and in general when the query describes scenes happening in the background. For this reason, effort has been put to create an HD version of MSR-VTT, called MSR-VTT HD. The original videos were available on YouTube and the annotations include both URLs and timestamps to extract each clip. Therefore, the entire dataset has been downloaded directly from YouTube, with the highest possible resolution. Some of the urls were broken so the missing videos have been replaced with the original low resolution ones. The resulting dataset presents nearly the 75% of videos with a resolution higher than 480×360 (mean resolution: 672×504, median resolution 713×534). A breakdown of the resolution is reported in Figure 4.2. Unfortunately, there was a problem with the annotations since the reported start and end timestamps didn't match perfectly with the original clips. The reason could be a different way of rounding fps or number of frames during the cropping. Another source of mismatch could be the fact that over the years, some of the videos could have been edited on YouTube, shifting the timestamps.

Caption analysis

The type of queries that can be processed highly depends on the training data of the language branch of the model. CLIP4Clip language branch is firstly trained on WIP [21] during CLIP training. Then, the same branch is finetuned on MSR-VTT [59] captions. Radford et al. did not make available WIP dataset so the only analysis that can be conducted regards MSR-VTT. In this analysis, stemming and lemmatization are applied before tagging to break down the words to their root form. Stemming can produce words that do not exist (i.e. charachter $\rightarrow$ charact) while the lemma is always a real word. Verbs are also treated differently, lemmatization converts the verb into the infinitive form (i.e. am, is, are $\rightarrow$ be) while stemming only converts it into the stem (i.e. are $\rightarrow$ are; running $\rightarrow$ run). Then, a word tagger is used to categorize words, based on the corresponding part-of-speech (**POS**), inferred by the definition and the context. A **POS** is a category of words that share the same grammatical properties. Some tags are nouns, verbs, adjectives, adverbs, pronouns, prepositions. Then, starting from tags, it is possible to produce rankings of the most frequent words, divided by their POS. Moreover, different

patterns are evaluated such as the sequence of [ADJECTIVE]-[NOUN]-[VERB]. Finally, rankings based on the n-grams (regardless of the tags) are reported. The plots showing the top 20 rankings for each category are reported in Appendix B.1.
Ignoring tags, the most frequent word are mainly articles, prepositions and conjunctions (Figure B.1). The first one is "a", followed by the verb "be". This is not surprising, considering that the sentences have the goal of describing a scene. In fact, often, while describing something, present continuous tense is used. This clearly explains the verb "be" as one of the most frequent terms. This is confirmed by the verb ranking in Figure B.3 and in [VERB]-[VERB] pattern ranking in Figure B.8. This behaviour can be also seen in [NOUN]-[VERB] pattern ranking in Figure B.5, [ADJECTIVE]-[NOUN]-[VERB] in Figure B.6 and [ADJECTIVE]-[NOUN]-[VERB]-[VERB] in Figure B.7. Most common nouns are mainly related to humans: "man", "woman", "person", "girl" and actions related to them: "talk", "game", "play", "show". The presence in the top 20 nouns of "cartoon" suggests that a significant amount of clips is taken from animations (Figure B.2). Most common verbs, excluding "be", are "play", "talk", "show" and "sing". This underlines the consistent presence, inside the dataset, of human interaction scenes, sport games and videogames (Figure B.3). Ranking for adjectives and adverbs is also interesting. Attributes, in fact, represents a crucial component of the query of the final application (Figure B.4). Most common terms are related to colors ("black", "white", "red"), people ages ("young", "old") and dimensions ("small"). An n-gram is a contiguous sequence of n words from a given sentence. For this analysis, n-grams with $n = 2$ (bigrams) and $n = 3$ (trigrams) are considered. Note that $n = 1$ (unigrams) coincides with word occurrences shown in Figure B.1. Considering the vast usage of article "a", a good portion of the most common bigrams and trigrams includes it. "man", "woman" and "person" are also very frequent, often found together with the preceding article or the following verb ("a man", "a woman", "a person", "woman is" for bigrams and "a man is", "a woman is", "a person is" for trigrams). Combinations of conjunctions and articles are also found ("in a", "on a", "of a", "on the", "with a"). There are also some compound nouns such as "video game" and other very common combinations of words such as "a group of", "there is a", "is talking about" and "in front of". Bigrams and trigrams plots are reported respectively in Figure B.9 and Figure B.10.

4.2.3 Solferino dataset

To compare image retrieval and video retrieval approaches, an additional dataset is used to test zero-shot performances of the models. Solferino dataset is composed of 3665 clips taken from a 1 week recording of Solferino square (Turin), kindly

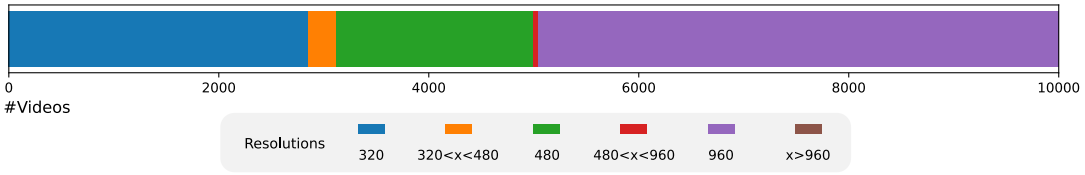


Figure 4.2: A breakdown of the video resolution in MSR-VTT HD dataset. The majority of the samples has a higher resolution than the original dataset.

provided by AddFor S.p.A.[1]. The raw video is recorded in 720p (1280×720) at 29.75 frames per second and, to produce the clips, a crop (360×360) on the center of the square is applied. Clips are divided into three categories: calm, normality and anomaly. **Calm** category contains clips with few movements, people sitting or stop. **Normality** category contains clips with people walking, bicycles or scooters. **Anomaly** category contains clips with anomalies. For example vehicles, a lot of people in small space, flocks of pigeons and more. The duration is fixed to 5 seconds. Considering that Calm/Normality/Anomaly is the only available information for each clip, further annotation is required to allow the models to be tested.

Caption generation

COCO dataset [47] consists of image-captions pairs. MSR-VTT dataset [59] consists of clip-captions pairs. This means that, to be able to test image retrieval and video retrieval models, a set of captions is required for each clip. Even excluding clips labelled as Calm (often with no people or objects in the scene and so not very interesting), there are still 1512 clips to be annotated, making manual captioning very time consuming. A possible workaround is automatic annotation via a **captioning model**. The chosen architecture is VinVL [56], already described in Sec. 3.2.2. Traditional captioning models produce a single caption for each input image. In this case, considering that different events could happen on the scene simultaneously (e.g. a man walking with a dog and a woman on a bike), such caption could not capture all the interesting parts of the frame. To confirm this hypothesis, COCO includes 5 caption for each image while MSR-VTT includes 20 captions for each clip. For this reason, having more than one caption for each sample is preferable. To achieve such result, multiple inferences need to be done on the same input. Firstly, the center frame (2.5 seconds) is extracted from each Anomaly and Normality clip. Then, the resulting keyframes are fed into VinVL backbone, X152-C4 [16], that perform object detection, producing bounding boxes, class labels, confidence scores and region features (2048 elements vector), as

[1] `https://www.add-for.com/`

depicted in Figure 4.3. Then, up to 4 crops are selected, according to the position

Figure 4.3: The keyframe is fed into the object detector that produce bounding boxes, class labels, confidence scores and region features. Here the interesting objects are highlighted in yellow. Details about interesting objects are discussed in Sec. 4.2.3.

of the interesting elements on the frame (see Sec 4.2.3, Sec 4.2.3 and Figure 4.6). Those crops are considered as independent samples that are then fed, together with the whole backbone output, to VinVL. In this way, the captioning model will output up to 5 captions (1 obtained from the whole input frame and up to 4 obtained from the selected crops). An example is displayed in Figure 4.7. To further improve the quality of the annotations, manual correction is applied over all the generated captions to fix the systematic errors of VinVL. Details about manual correction are discussed in Appendix C.1. The resulting retrieval dataset consists in 1512 clips and 3684 captions. The main difference with respect to the other discussed retrieval datasets is that a lot of captions are similar if not identical. This happens because the camera is always fixed to the same point. Therefore the input frames share the vast majority of the details, with some differences that depend on the objects passing through the square. This needs to be kept in mind when looking at the metrics, in particular for the Recall results (see annotation examples in Figure 4.4).

Removing bounding box redundancy

VinVL backbone, X152-C4 [16], returns for each frame, bounding boxes, object classes, confidences and region features. Being trained on COCO [47], Objects365 [57], OpenImagesV5 [58] and VisualGenome [11], there is a wide variety of classes that the model can recognize.

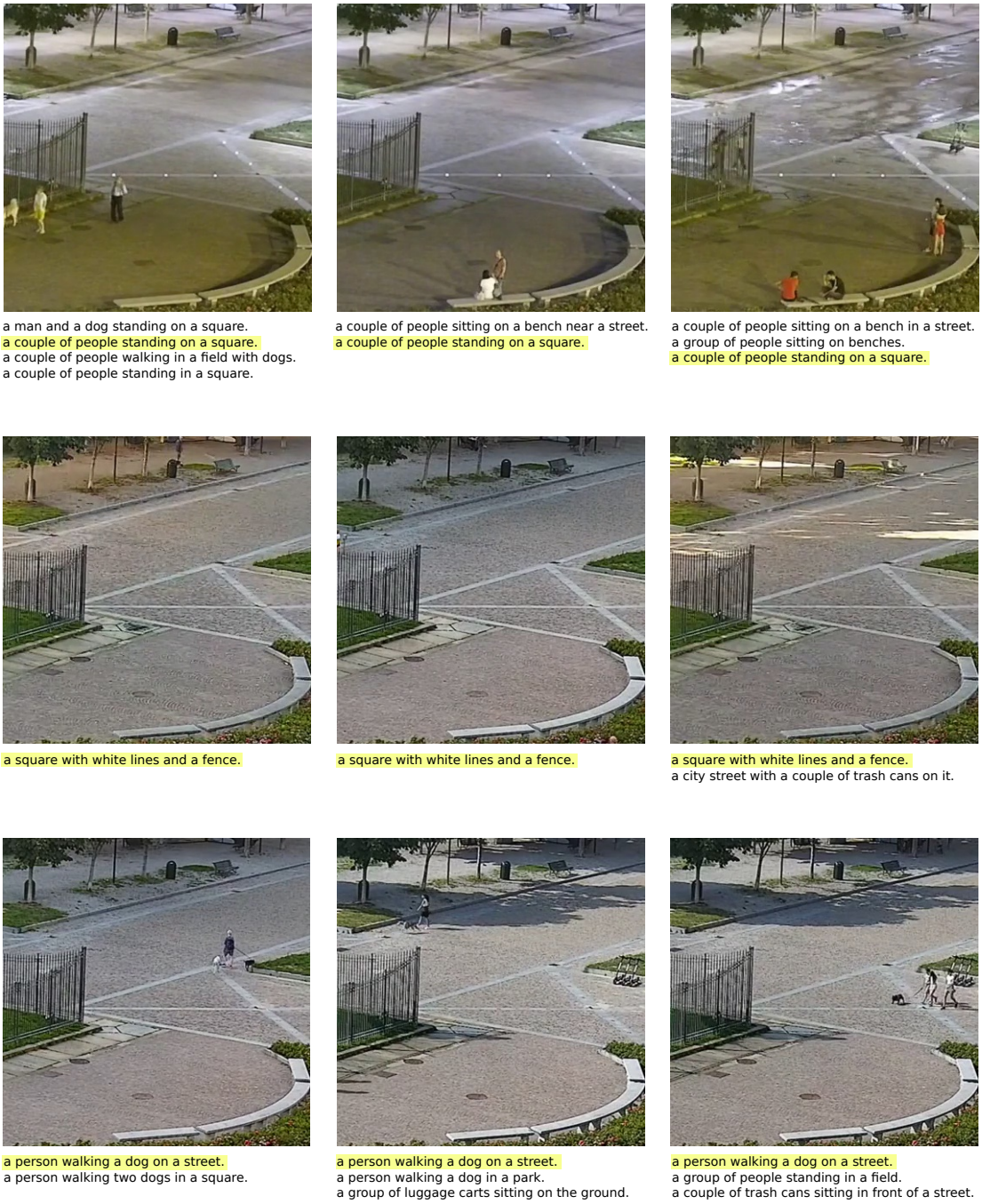


Figure 4.4: Examples of annotations after manual correction. For each row, identical captions are highlighted. During validation, if the input query is the second caption of the first frame (`a couple of people standing on a square`), there are multiple candidates associated with it, but only one (the first frame) is the exact match for that instance of the query. For this reason, Recall metric resulted very low for all the experiments on Solferino dataset.

Some of them are not mutually exclusive, for example there is the class `person` but also `man`, `woman` and `child`. This results in cases in which the same person is detected twice with different labels. Moreover, sometimes, the same object is detected multiple times even with the same class. Once fed into VinVL, those redundant features could trick the model to think that there are actually more than one instance of the detected object. Aiming to solve this issue, a filtering strategy is applied. Firstly, different sets of non-mutual exclusive classes are considered, focusing mainly on people and vehicles. Here the sets are reported:
`Set 1: ["man", "person", "child", "children", "lady", "girl", "boy"]`
`Set 2: ["cart", "truck"]`, `Set 3: ["motorbike", "motorcycle"]`
Then, each pair of bounding boxes is evaluated. If the two region classes are inside the same set of non-mutual exclusive classes, just one of them will remain. If the intersection is above a threshold, the lower confidence proposal is removed. An example is reported in Figure 4.5.

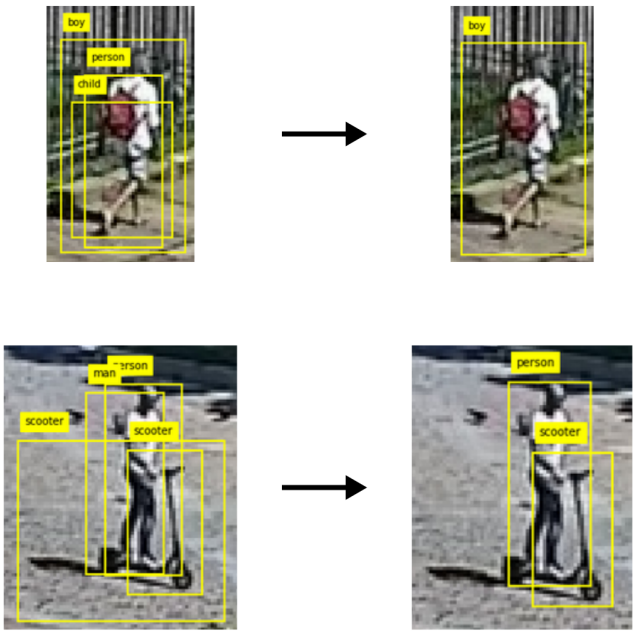


Figure 4.5: Before and after redundancy removal for Solferino dataset

Crop selection

To select the crops, focusing on specific type of classes is needed. In this case, considering the context, the focus is on people, animals and vehicles. The chosen interesting classes are `"man"`, `"woman"`, `"child"`, `"children"`, `"person"`, `"lady"`, `"girl"`,

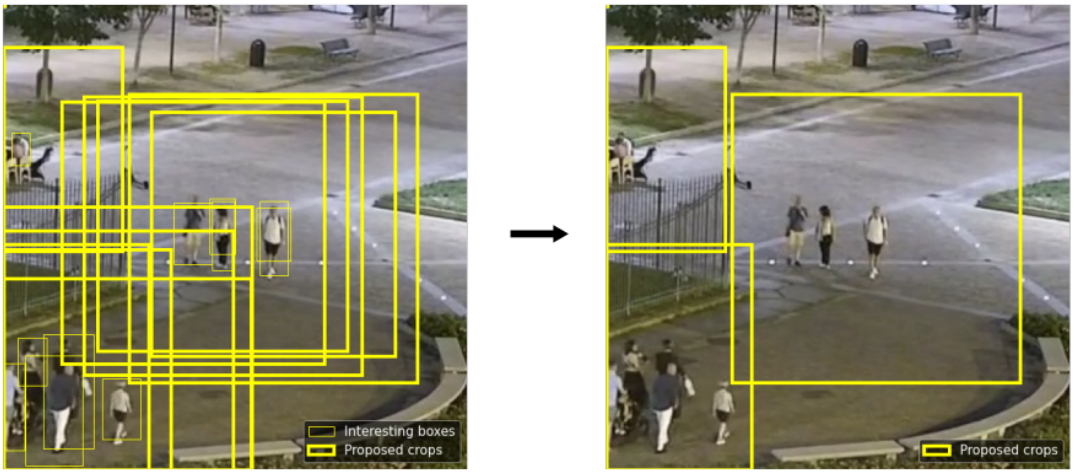


Figure 4.6: One crop for each interesting box is proposed. Then, filtering is applied to reduce the number of crops.

`"boy"`, `"car"`, `"van"`, `"bike"`, `"motorbike"`, `"motorcycle"`, `"bus"`, `"train"`, `"tram"`, `"taxi"`, `"cart"`, `"scooter"`, `"truck"`, `"ball"`, `"dog"`, `"cat"`, `"horse"`. Firstly, all the regions related to non-interesting classes are ignored. Then, starting from the bounding boxes of the interesting instances, the crops are created by expanding the boxes in all directions by a percentage of the image size (15%). The resulting crops could be overlapped. For example, if a group of people is detected, an individual crop proposal for each person is likely to be generated. To remove redundant crops, a suppression strategy is applied. Firstly, the bigger crop is selected. All the other crops belong to a pool of candidates, from which, at each iteration, the best one will be selected until the crop limit (4 crops) is reached. The best crop to be selected is the one that do not overlap the already chosen crops and the distance from them is higher than the other candidates. See Figure 4.6 for a graphic example. The resulting crops will be treated like independent samples and fed to VinVL to get individual captions. Such individual samples have the same structure of the output of the object detector but include only the bounding boxes that are inside the crop boundaries. The inference step is illustrated in Figure 4.7.

4.3 Model architectures

4.3.1 Image retrieval

The proposed model, based on Image Retrieval, is TERN [6]. The details of the original architecture have been discussed in Sec. 3.4. The major problem in this

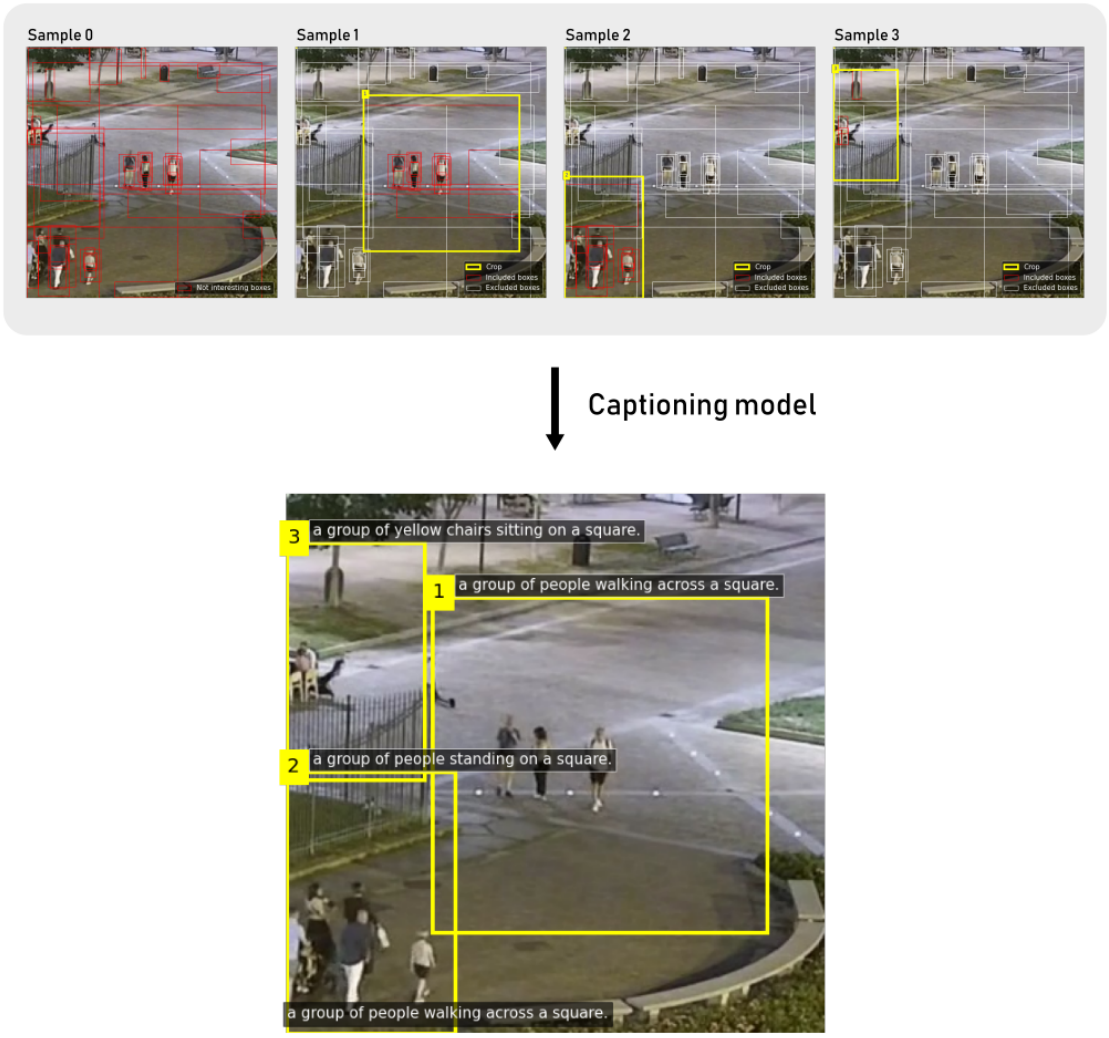


Figure 4.7: For each final crop, an independent sample is produced and fed to the captioning model together with the entire object detection output. Such samples contain only the boxes inside the crop boundaries.

case is the inference time. In fact, the proposed object detector is Faster R-CNN [26], a very accurate detector capable of providing not only the bounding boxes and the classes of the objects in the image, but also a feature vector for each of them. Such features consists in a fixed size vector obtained by ROI pooling the final feature maps with the proposed regions. See Sec. 3.2.1 for details. Those vectors are fed into the vision branch of TERN, that consists in 4 Transformer encoder layers followed by 2 other layers with weights shared with the textual

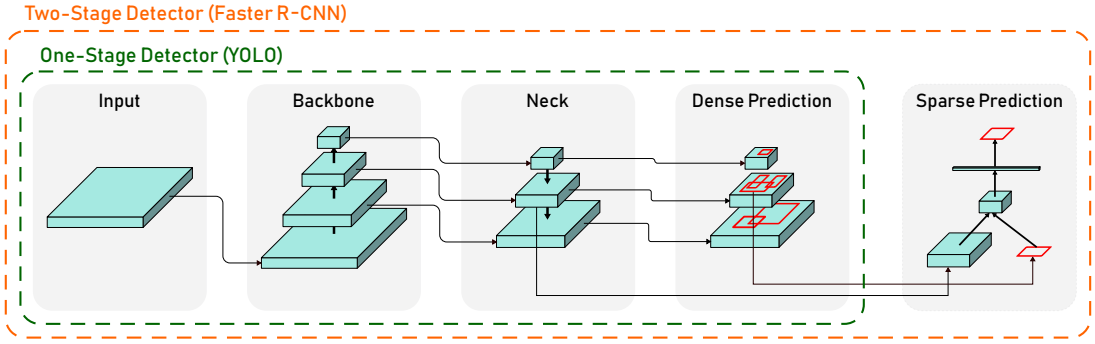


Figure 4.8: General architecture for Single-Stage detectors (e.g. YOLO [44]) and Two-Stage detectors (e.g. Faster R-CNN [26]). Illustration inspired by the one present on YOLOv4 paper [49].

branch to output the final embedding. Having a two stages object detector impacts inference time, since Faster R-CNN needs to look at the image multiple times. To solve this issue, different detectors have been evaluated. One stage detectors produce the entire output with a single pass through the neural network, allowing the models to work in real-time. The chosen architecture is Scaled-YOLOv4 [50], already mentioned in Sec. 3.2.1. The main challenge faced with this detector is that, in the original implementation, only bounding boxes and class probabilities are returned. Macroscopically, Scaled-YOLOv4 is composed of the **backbone**, **Darknet-53**, consisting of 5 downsample blocks, the **neck**, responsible for aggregating features coming from different blocks of the backbone and three **heads**, that, applied at different stages on the network, starting from the neck's output, detect different-sized objects. An illustration of both single-stage and two-stage object detector architectures is reported in Figure 4.8.
Each head returns a single vector containing the coordinates and the class probabilities for each candidate region. As described in Sec. 3.2.1, the image is divided into an $S \times S$ grid and for each grid cell, B bounding boxes are regressed. So, considering C the number of classes, the dimension of the output vector is $B \times S \times S \times (5 + C)$. Working on different stages of the network, each head has a different value of S, while B is fixed to 3 and C is fixed to 80. Fixing the values, the dimensions become $3 \times S \times S \times 85$. To match this shape, the last convolutional layer before the head is designed to produce a $255 \times S \times S$ vector that is then **reshaped**. It follows that producing the entire inference output on a single pass (and not having to apply ROI pooling on feature maps) requires the model to regress the coordinates and probabilities directly from the same aggregated features. For this reason, inside YOLO architecture there is no layer from which is possible to extract per-region features. Since TERN pipeline requires feature vectors for each detected regions,

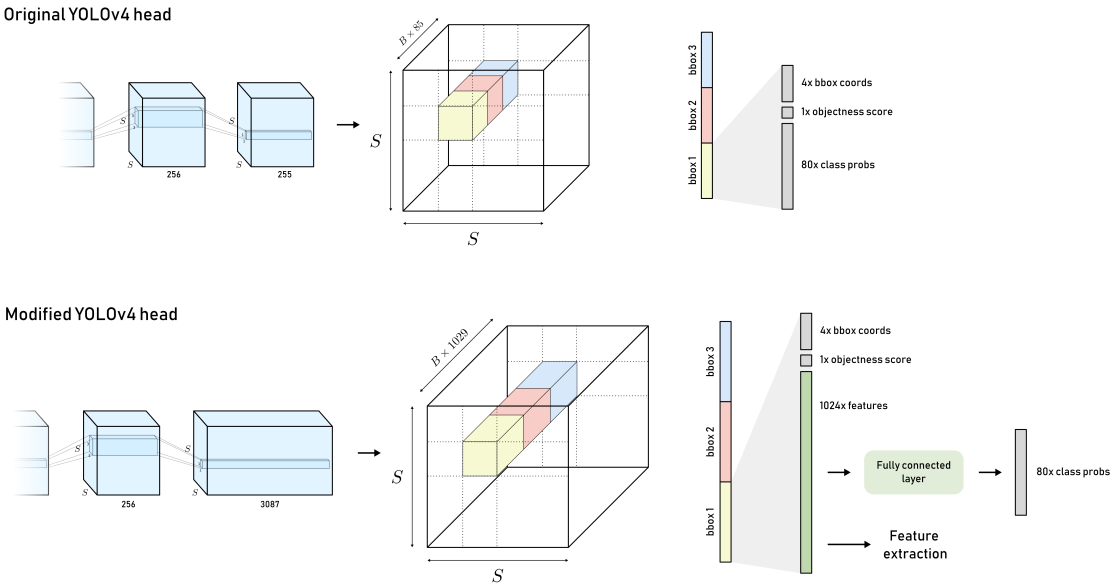


Figure 4.9: Top: Original YOLOv4 head. **Bottom**: Modified YOLOv4 heads. In the original implementation, class probabilities are directly extracted from the last convolutional layer output, while in the modified version, the last conv layer produces 1024 features that are then linearly projected, with a learnable matrix, into the final class probabilities. S is set to 4 to ease visualization and the number of proposed boxes for each grid cell is $B = 3$ (shown in yellow, red and blue).

a **modification** to Scaled-YOLOv4 is applied. The last convolution layer before the head is increased in size to produce a vector of dimension $3087 \times S \times S$, then reshaped into $3 \times S \times S \times 1029$. The first 5 values of the last dimension are **4 coordinates** and **one objectness score** as before, but instead of returning directly the 80 per-class probabilities, **1024 features** are computed. Then those 1024 feature vector is fed to a linear layer responsible for the actual classification. In this way, the model is forced to produce a 1024 elements representation, specific for each region, since the region class is predicted solely on it. Those features are extracted and treated exactly as the Faster R-CNN ones. Original and modified heads are illustrated in Figure 4.9.
Both the object detector and TERN need to be trained and training is done in two stages.
In the first stage, the object detector is finetuned on COCO [47] to make the model adapt to the architecture tweaks. All the weights with exception of the last convolutional layers before the heads and the added linear layers are transferred from the original model. Finetuning consists in 30 epochs with a batch size of 8, maximum image size of 640 pixels, SGD optimizer with an initial learning rate of 0.01, momentum equal to 0.937 and a 5e-4 weight decay.

In the second stage, TERN is trained on COCO, feeding the precomputed feature vectors of the object detector along with the original captions. Notice that modified Scaled-YOLOv4 produces 1024 feature elements in contrast with the 2048 elements of Faster R-CNN. Therefore the first TERN layer of the visual branch is reduced in dimension.
The original TERN [6] was trained on COCO dataset too, so pre-computed weights could still be useful as a starting point. For this reason, multiple experiments have been done, freezing different portions of the architecture while transferring the original weights. Training consists in 30 epochs with a batch size of 80, Adam optimizer with learning rate equal to 2e-6. The experiments have been conducted with a single graphics card. The available cards were an **Nvidia GeForce 1080Ti** and an **Nvidia GeForce Titan X**. The former performs slightly better but the training times are comparable.
Further details on training times and model sizes are reported in Table 4.1. The final architecture consists of 181 Millions parameters, 58 Millions for YOLO and 123 Millions for TERN.

Architecture	**Training**	**Training time** (hours)	**#Parameters** (Millions)
TERN	A	20	123
	V + Sh	15	123
	V	13	123
Modified YOLOv4		44	58
CLIP			151

Table 4.1: Training details for Image Captioning models. Training column indicates which portion of the TERN network is trained (the others are frozen): **A**: All the weights, **V**: Vision branch, **Sh**: Shared Transformer layers.

At inference time the image is fed into the object detector and the output is directly processed by TERN to produce the visual embedding to compare with the query embedding. So, with the combination of the two models, the resulting architecture is capable of performing image retrieval **end to end**, from pixels to similarity score. Considering the great performances of CLIP [21] in zero-shot

predictions, the original model, working on 224×224 pixel resolution, is tested on the same COCO 1k testset without finetuning. The pipeline is similar but there is no need for the object detector. Dealing with video data, frame sampling is essential. Most of the samples have a number of frames per second around 25-30. This means that, processing each frame, results in a very big amount of computation. Moreover, subsequent frames often appear almost identical. For this reason, to avoid redundancy and useless computations, only a subsample of the frames are processed. The interval between a sample and the next one depends on the application. If the scene is static enough, processing a frame each second can be sufficient while, for more dynamic situations, lowering the interval is preferable.

4.3.2 Video retrieval

The proposed model, based on Video Retrieval is Clip4CLIP [7]. The details of the original architecture have been discussed in Sec. 3.6. The major problem resides in the input. In fact, input frames are firstly resized so that the smaller dimension is equal to 224 pixels. Then a **center crop** is applied to obtain the final 224×224 input image. This procedure has two major issues. The first is that 224 pixels could be not enough for very cluttered scenes where there are a lot of elements not always on the foreground. The second issue is related to the cropping operation. Applying cropping in that way, the far-left and far-right portions of the frames are lost. This, again, can be irrelevant if the main subject is on the foreground and occupies a good portion of the scene. Instead, when the information on the side of the frames are important (e.g. security cameras), that crop can be a problem. To solve the second issue, input preprocessing is modified. Instead of resizing to match the smaller dimension to 224 pixels, now the bigger dimension has to match 224 pixels. Then, the same center crop is performed. In this way, no portion of the frame is lost and, to match the input dimension, **padding** is applied (the modification is illustrated in Figure 4.10).
To solve the first issue, finetuning on higher resolution has been experimented. CLIP4Clip is based on the original CLIP [21] weights and the final model is obtained through finetuning on MSR-VTT [59]. Such finetuning can be performed with an higher input resolution, allowing the model to learn to deal with bigger frames. Input frame is divided into patches, that are fed to the Transformer encoder as tokens. Patches are extracted by a convolutional layer applied on the raw pixels. For this reason, if the input size changes, the only difference is the number of patches generated by the convolution.
Since the Transformer encoder can process a virtually unlimited (limited by memory) amount of tokens, the model can adapt quite easily to the new input size. The entire architecture can be kept unaltered with the exception of the visual input positional encoding layer. Positional encoding has been discussed in Sec. 3.1.2. It is a vector

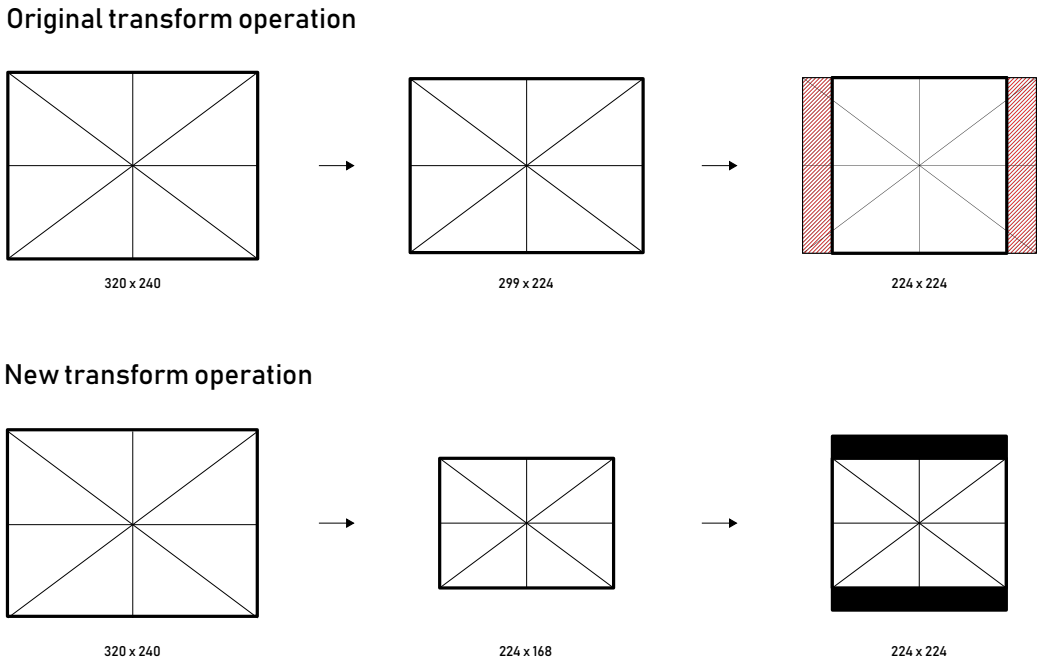


Figure 4.10: Transform operations on input frames. **Top**: Original transform operation. Center of the image is zoomed in while the sides are cropped out. **Bottom**: The proposed transform operation. The whole image is preserved and a padding on the smaller dimension is added.

added to the input to inject positional information that otherwise would be ignored due to dot-product self-attention mechanism. In the case of Vision Transformer (ViT) [42], the block inside CLIP4Clip responsible for frame processing, positional embedding is a learnable vector with a fixed size of 50 elements. 50 is, in fact, the number of input tokens. The convolutional layer that produce the patches has a kernel dimension of 32×32 pixels with a stride of 32. This means that, for a 224×224 pixel input, the number of patches is $7 \times 7 = 49$. An additional token is prepended to the input sequence and its state at the output serves as the image representation. So, increasing the resolution to 320×320 pixels results in an input sequence of $1 + (10 \times 10) = 101$ tokens.
Since the dimension of the positional embedding needs to match the input sequence length, the original one cannot be used. Instead of learning from scratch the new embedding, the 49 positional elements of the input patches can be **2D-interpolated** into the new dimension, as explained in CLIP paper [21](Figure 4.11).
The output of the Vision Transformer consists in one embedding for each frame of the input clip. Different ways for aggregating such embeddings can be evaluated to

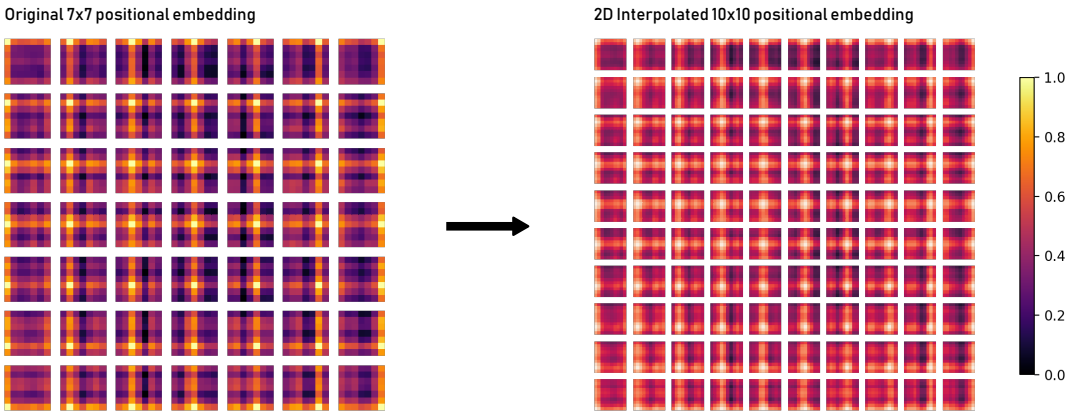


Figure 4.11: Cosine similarity between positional embedding elements. **Left**: Original 7×7 positional embedding elements. **Right**: 2D Interpolated 10×10 positional embedding elements. Tiles show the cosine similarity between the positional embedding of the patch in the indicated row and column and the positional embeddings of all the other patches. Idea taken from Vision Transformer paper [42]. Note that, although cosine similarity ranges from -1 to 1, the colormap interval is reduced to [0,1] for ease of visualization.

compute the similarity score (See Sec. 3.6.1). The chosen type is the **Parameter-free type** in which similarity is evaluated as the **cosine similarity** between the text embedding and the mean-pooled frame embeddings.
The number of processed frames for each clip affects the performances and the computational cost. Different values have been experimented in the original paper [7]. Finetuning is performed with a sample frequency of 1 sample/sec while the maximum clip length is set to 12 or 9 seconds according to the memory and computational requirements. Frames are sampled uniformly (**Uniform** frame sampling, see Sec. 3.6.3) and, during testing, the clip length is fixed to 12 seconds. In some experiments, to speed up training, some of the CLIP layers are frozen. Learning rate for text and video branches is 1e-7, the maximum caption lengths is 32 and the maximum clip length is 12 frames (or 9). The optimizer is Adam, with a weight decay of 0.2 and a warmup proportion of 0.1. Linear patches are set to 2D since they seem to work better [7]. Each finetuning consists in 5 epochs on the 7k split of the MSR-VTT dataset [59] or the MSR-VTT HD dataset if needed (See Sec. 4.2.2). Further details on training times and batch sizes are reported in Table 4.2. Although they have different positional embedding sizes, all the C4C models (CLIP4Clip) have more or less the same amount of parameters, around 151 Millions. At inference time, according to the application, window length, window step and sampling frequency can be changed. Window length is the length of the extracted

Model	Dataset	Batch size	Training time (hours)
C4C-224	MSR-VTT	32	20
C4C-224	MSR-VTT HD	32	22
C4C-320	MSR-VTT	16	41
C4C-320	MSR-VTT HD	24	39
C4C-448	MSR-VTT	10	94
C4C-448	MSR-VTT HD	10	76

Table 4.2: Training details for CLIP4Clip. Note that the machine the models are trained with is shared with other people, so training time can vary according to the load. C4C stands for CLIP4Clip.

clip in seconds while window step is the time interval between the beginning of a window and the beginning of the next one. Windows can be overlapped to ensure to retrieve the exact clip but too long or too overlapped clips increase computations.

4.4 Results

4.4.1 Image retrieval solution

The Image Retrieval model TERN+YOLO is evaluated on the COCO 1k-testset. In Table 4.3 are reported the Recall and NDCG metrics for all the models, CLIP included. Focusing on real-time capabilities, in Table 4.4 are reported the number of inferences per second of the original model with Faster R-CNN [26], the proposed model with YOLOv4 [50] and CLIP [21]. The performance tests have been executed on an **Nvidia 1080Ti** on the same video[2].

Results discussion

Table 4.3 shows the results of the experiments for original TERN architecture [6], modified architecture with YOLO [50] and CLIP[21]. The original model is still the best one. This can be the evidence that the features produced by Faster R-CNN are better and more informative with respect to modified YOLO.

[2] Benchmark video available here: `https://youtu.be/4DKiH4wmQLU`

Architecture	Training	Recall			Median R/ Mean R	NDCG (R/S)
		R@1	R@5	R@10		
TERN^{R-CNN}	-	**51.9**	**85.6**	**93.6**	**1 / 4.6**	**0.7248/0.6534**
TERN^{YOLO}	A	40.6	77.7	89.2	2 / 6.6	0.6997/0.6210
TERN^{YOLO}	V+Sh	40.1	78.0	89.4	2 / 6.6	0.7016/0.6243
TERN^{YOLO}	V	42.5	79.3	90.1	2 / 6.4	0.7059/0.6281
TERN^{YOLO}	V+Sh*	39.9	77.4	89.2	2 / 6.6	0.7021/0.6255
CLIP$_{ZS}$	-	44.0	74.1	84.9	2 / 7.8	0.6947/0.6083

Table 4.3: Results of the experiments on COCO 1k testset [47]. In the first row are reported the metrics for the original TERN [6] model TERN^{R-CNN} while TERN^{YOLO} is the model trained on features extracted by ScaledYOLOv4 [50] with the method described in Sec. 4.3.1. Training column indicates which portion of the TERN network is trained (the others are frozen): **A**: All the weights, **V**: Vision branch, **Sh**: Shared Transformer layers. (**Sh*** means that shared layers are trained from scratch). NDCG score is evaluated with both RougeL (**R**) and Spice (**S**). CLIP$_{ZS}$ is the original CLIP model proposed by Radford et al. [21] working on 224 pixel resolution and is not finetuned on COCO (zero-shot predictions).

Architecture	Resolution	Samples/second
TERN^{R-CNN}	1000	3.20
TERN^{YOLO}	640	29.54
CLIP	224	58.29

Table 4.4: Real-time performances for TERN models. Object detector is responsible for the vast majority of the inference time. Resolution is expressed in pixels of the bigger dimension. The models are tested with their original training resolution to maintain coherence with results reported in Table 4.3.

This is somehow expected since Scaled-YOLO is not designed to produce region features and adapting such architecture resulted in sub-optimal representations. Moreover, YOLO is only trained on COCO while Faster R-CNN is also trained on VisualGenome, so the latter will probably recognize a wider variety of regions with respect to YOLO, limited on the 80 COCO classes. Among $TERN^{YOLO}$ results, better performances seem to be obtained while training only the visual branch, keeping intact the rest of the network. This suggests that TERN original weights were already well trained on the language side and needed no finetuning. CLIP is

trained on WebImageText (WIP) and is not further finetuned on COCO, so the reported results are zero-shot. Therefore the performances were expected to be worse than the other models. Despite that, CLIP demonstrated to be very good at zero-shot, achieving results comparable with $TERN^{YOLO}$ but still far behind $TERN^{R-CNN}$.

Real-time performances

On the real-time performance side, from Table 4.4 it's possible to notice the inference time difference between two-stages object detectors (Faster R-CNN [26]) and one-stage object detectors (Scaled-YOLOv4 [50]). $TERN^{YOLO}$ is able to work around 10 times faster than $TERN^{R-CNN}$. CLIP visual branch does not require an object detector and the real-time performances are even better than $TERN^{YOLO}$.

4.4.2 Video retrieval solution

MSR-VTT 1k testset

The Video Retrieval model CLIP4Clip [7] is evaluated on the MSR-VTT dataset [59]. In Table 4.5 are reported the results of the experiments on the MSR-VTT HD 1k testset. In Table 4.6, original and HD datasets are compared, evaluating the impact on the metrics.
Finetuned models are tested on different resolutions. In Table 4.7 are reported the R@1 and NDCG RougeL scores for each combination of model and input size. Extended results are available in Table 4.8. The impact on the metrics of the alternative image preprocessing, described in Sec. 4.3.2, are available in Table 4.9. Finally, real-time performances are evaluated in Table 4.10. Different combinations of window length and sample frequency have been tested on the same benchmark video[2] used for TERN and executed on an **Nvidia 1080Ti**.

Increasing model resolution

The results in Table 4.5 shows how the architecture struggles when the training resolution is increased too much. With the HD version of the dataset the best results are obtained with an input resolution of 320×320. This could be evidence that, even despite the resolution shift, the performance degradation is compensated by the increased image details. Reaching 448×448 input size, the performance drop is noticeable. One could argue that this is caused by a too big "jump" from the original 224×224 resolution, but the C4C-448^{320} model results demonstrate that, even starting from C4C-320 weights (so gradually increasing resolution), metrics do not improve. Testing is done also on different resolutions to understand how the models are able to deal with input sizes different from the training ones.

Model	Input Size	Training			Recall			Median R/ Mean R	NDCG (RougeL)
		R	F	L	R@1	R@5	R@10		
C4C-224	224 × 224	O	12	0	**42.3**	67.3	**79.0**	2/18.3	0.5179
C4C-320	320 × 320	N	9	3	**42.2**	**69.5**	77.8	2/**17.7**	**0.5248**
C4C-448	448 × 448	N	9	3	40.2	66.8	77.2	2/17.9	0.5045
C4C-448^{320}	448 × 448	N	9	3	40.1	67.0	77.4	2/18.3	0.5041

Table 4.5: The best results of CLIP4Clip models, finetuned with different resolutions, on the MSR-VTT HD dataset. C4C-448^{320} is finetuned on 448 × 448 resolution, starting from C4C-320 weights. Training column indicates different tweaks applied to the model for training: **R**: Resize operation, **O** is the original operation while **N** is the proposed one. See Sec. 4.3.2 for further details. The last two are parameters that can be set to reduce the training time. **F**: Maximum number of frames, **L**: Number of frozen CLIP layers. Note that all the models have been evaluated on a fixed maximum of 12 frames (even if it was 9 during training).

Model	Dataset	Recall			Median R/ Mean R	NDCG (RougeL)
		R@1	R@5	R@10		
C4C-224	Standard	**42.6**	**67.8**	79.0	2/**16.8**	**0.5202**
C4C-224	HD	42.3	67.3	79.0	2/18.3	0.5179
C4C-320	Standard	41.9	67.1	**79.2**	2/17.3	**0.5163**
C4C-320	HD	41.8	**67.6**	78.5	2/**16.2**	0.5132
C4C-448	Standard	**41.1**	**67.2**	**78.7**	**2/16.4**	**0.5067**
C4C-448	HD	40.1	65.4	76.2	2/17.5	0.5019

Table 4.6: Models results on standard and HD MSR-VTT dataset. The standard dataset resolution is 320 × 240 while the HD version resolution is not fixed. Further details are reported in Sec. 4.2.2. All the reported models are trained with the original resize operation.

As expected, results in Tables 4.7 and 4.8 show that, for each input size, best performances are achieved by models trained specifically with them, with lower results for the others.

	Test resolution (R@1 / $\text{NDCG}_{\text{RougeL}}$)		
	224×224	320×320	448×448
C4C-224	**42.3** / **0.5179**	38.9 / 0.4975	34.4 / 0.4450
C4C-320	39.7 / 0.5006	**42.2** / **0.5248**	39.1 / 0.4982
C4C-448	38.4 / 0.4866	41.1 / 0.5075	40.2 / **0.5045**
C4C-448^{320}	38.3 / 0.4784	40.0 / 0.5050	**40.3** / 0.5018

Table 4.7: Results of the models on different test resolutions. This helps to understand how well the model performs on resolutions different from the one used during finetuning. C4C-448^{320} is a model finetuned starting from C4C-320 weights

Model	**Resolution**	**Recall**			**Median R/ Mean R**	**NDCG (RougeL)**
		R@1	R@5	R@10		
C4C-224	320×320	38.9	66.1	75.6	2/21.5	0.4975
C4C-224	448×448	34.4	60.5	72.5	3/26.1	0.4450
C4C-320	224×224	39.7	66.4	76.9	2/18.2	0.5006
C4C-320	448×448	39.1	67.2	76.7	2/20.8	0.4982
C4C-448	224×224	38.4	64.5	75.4	2/18.1	0.4866
C4C-448	320×320	41.1	67.5	78.5	2/16.5	0.5075
C4C-448^{320}	224×224	38.3	65.3	74.4	2/19.2	0.4784
C4C-448^{320}	320×320	40.0	67.2	76.3	2/18.1	0.5050

Table 4.8: Extended results of the models on different test resolutions.

Increasing dataset resolution

The results in Table 4.6 show that, in general, training the models on the HD datasets does not provide benefits, and, especially for C4C-448, the original low resolution videos lead to better performances. This could be caused by some annotation problems, already discussed in Sec. 4.2.2. Despite that, having an original HD dataset (without needing to manually download and crop each video) could lead to better results with respect to the low resolution one, especially for models trained on higher input sizes, where source frame resolution is crucial.

Model	Transform operation	Recall			Median R/ Mean R	NDCG (RougeL)
		R@1	R@5	R@10		
C4C-224	Original	**42.3**	**67.3**	**79.0**	2/**18.3**	**0.5179**
C4C-224	New	41.1	67.0	77.0	2/19.4	0.5076
C4C-320	Original	41.8	67.6	**78.5**	2/**16.2**	0.5132
C4C-320	New	**42.2**	**69.5**	77.8	2/17.7	**0.5248**
C4C-448	Original	40.1	65.4	76.2	2/**17.5**	0.5019
C4C-448	New	40.2	**66.8**	**77.2**	2/17.9	**0.5045**

Table 4.9: Impact of the new frame resize method proposed in Sec. 4.3.2. All the reported models are trained on the MSR-VTT HD dataset.

Input resolution	Window		
	2 sec/6 fps	3 sec/4 fps	6 sec/2 fps
224×224	3.45	2.81	2.24
320×320	3.11	2.60	2.10
448×448	2.49	2.14	1.82

Table 4.10: Real-time performance of the CLIP4Clip models on different combinations of window length and sampling frequency. Note that each combination has the same amount of total frames. The average samples per second are reported for the benchmark video.

Changing resize operation

The original resize operations forces the model to focus only on the center of the frames, ignoring the sides that are cropped out during preprocessing. Providing the model with the entire frame (possibly with padding) could result in better performances. Results in Table 4.9 confirm this hypothesis. Models trained on higher resolution inputs benefit from the change of resize operation, leading to higher scores for the majority of the metrics. On the contrary, the original model, trained on 224×224 input size, seems to perform better with the original crop.

Real-time performances

To test real-time capabilities of the models, different combinations of window length and sampling frequency are evaluated. All the combinations are designed to have a

total number of input frames equal to 12. As expected, lower resolution models have lower inference time. Considering results for the same model, bigger windows require more time to be processed. This happens because, even if the inference time is the same (always 12 frames), reading from disk depends only on the window length. So, reading a 6 seconds window takes longer than a 2 seconds windows, regardless of the frame sampling frequency.[3] Increasing resolution instead affects inference time. In fact, the number of input patches depends on the input size since the first convolution layer has always the same kernel size of 32×32 pixels. The produced patches are 49 for 224×224, 100 for 320×320 and 196 for 448×448. Combining the two effects, longer and high resolution windows result being more expensive than shorter and low resolution ones.

4.4.3 Results on Solferino dataset

As anticipated in Sec. 4.2.3, Solferino dataset is conceived to be a common benchmark for both image and video retrieval models. None of the models are finetuned on it and the domain of the recorded video (fixed camera angle, tiny objects and not on foreground) represents a great challenge. The ability to transfer knowledge to the new domain is assessed by performing zero-shot retrieval on the samples. For image retrieval models each sample is produced by extracting the keyframe, in this case the center frame of the clip (2.5 seconds frame). For video retrieval models, the entire 5 seconds clip is fed into the model. The sampling rate is fixed to 1 frame per second. In Table 4.11 are reported the results for all the discussed models. For CLIP4Clip architecture (C4C), the models are exactly the same showed in Table 4.5.
A required premise to make is that Solferino annotations have been generated by a captioning model and, despite the manual correction, not necessarily represent the ground truth. This means that the best achievable results are upper-bounded by the captioning model itself and not by human-level annotations. Moreover, benchmarks based on manual annotations usually provide more reliable conclusions. From the results it's possible to notice very low values for the Recall metrics while the NDCG scores seem to be comparable with the values obtained with the other experiments. This happens because, as anticipated in Sec. 4.2.3, the majority of the captions are really similar. In fact, the camera angle is fixed to the same spot and the majority of the elements are present in all the clips. Recall metrics measure the ability of the model to detect the exact original sample linked to the caption, regardless of the similarity of the other candidates (see Sec. 2.1.1). Considering the samples, it is practically sure that multiple clips (frames) will be linked to

[3]The benchmark video has 30 frames per seconds. "sampling frequency" refers to the ulterior subsampling applied on all the frames already read from disks.

Model	Recall			Median R	Mean R	NDCG (RougeL)
	R@1	R@5	R@10			
TERN^{R-CNN}	**1.3**	**4.4**	**7.7**	**260**	**387.3**	**0.6274**
TERN^{YOLO}	0.2	0.9	1.4	698	715.1	0.4938
CLIP	0.9	2.9	5.0	353	484.4	0.6048
C4C-224	0.9	3.4	6.2	309	443.6	0.6077
C4C-320	0.9	3.5	5.8	290	433.1	0.6124
C4C-448	1.0	3.6	6.7	265	423.9	0.6246
C4C-448^{320}	0.9	4.0	7.1	308	447.4	0.6143

Table 4.11: Results of image retrieval and video retrieval models on Solferino dataset. The first three models are image retrieval models while the last four are video retrieval models. The training details of the CLIP4Clip models (C4C-*) are the same reported in Table 4.5.

exactly equal captions. Therefore the low values for the Recall metrics are expected. Some annotation examples are reported in Figure 4.4. The Normalized Discounted Cumulative Gain (NDCG) instead, takes into consideration the **relevance** with respect to the query of all the ranked candidates (see Sec. 3.4.3 for further details). For this reason, considering the extreme similarity of some of the samples, a more reliable behaviour of the NDCG score is expected.
From the results, TERN model trained on Faster R-CNN features seems to be the best. Training TERN on YOLO features results in noticeably lower metrics. Focusing on CLIP-based models, the original CLIP architecture achieve very good results, also considering the considerably lower inference time with respect to TERN^{R-CNN} (see Table 4.4). Looking at the results of CLIP, CLIP4Clip models, being derived from it, are expected to be at least as good. The expectations are met, with results that improve as the resolution increases, reaching, for C4C-448 model, metrics that are comparable with TERN^{R-CNN}.
The overall results seem to confirm that the same retrieval task can be effectively performed by image and video retrieval models and the nature of the dataset highlights the limitation of the traditional Recall metric in favour of the NDCG. All the models were consistently able to, at least partially, transfer their knowledge to the new domain allowing a possible application to work even without any finetuning. An inference example for the final application is reported in Figure 4.12, in which a textual query is used to perform scene retrieval on Solferino square recordings, by TERN^{R-CNN} model. Additional inference examples can be found in appendix D.

Figure 4.12: Inference example of Solferino recordings. The top 4 scenes are reported for the query, ordered from top-left to bottom-right, with the corresponding timestamp and confidence score.

Chapter 5

Conclusions

Video data constitutes a valuable source of information but, to leverage such great potential, automating the human processes involved is crucial. Combining the expressivity of written text and vision constitutes the foundation of Vision-Language understanding, often employed to perform automated supervision, that represented the grounding for this work. The goal of this thesis is to investigate different techniques to perform scene retrieval through the Transformer architecture, conceiving an application capable of detecting scenes inside long videos, described by textual queries, in different contexts. Such descriptions can be related to anomalous scenes, allowing the model to perform context-specific anomaly detection. The application can work off-line, on a pre-recorded video, or on-line, performing detection directly from an input streaming. For these reasons, particular attention is given to scale efficiency, real-time capabilities and adaptation to similar but different domains while increasing the complexity of input data. To preserve scale efficiency, the chosen architectures are required to separate the visual and language branch, allowing to compute only once the frame/clip embeddings. To match real-time requirements, inference time is evaluated for each model, altering the architecture if needed. Finally, to test the performances on a different domain, a new dataset is built from security camera recordings. Different approaches are proposed. Firstly, considering single frames as input data, image retrieval is performed using the model TERN [6]. To consistently reduce the inference time, a faster single-stage object detector, Scaled-YOLOv4 [50], is modified to be able to substitute the original two-stage model, Faster R-CNN [26]. Secondly, processing short video windows instead of frames, video retrieval is performed with the model CLIP4Clip [7]. The architecture is altered to allow the input size to grow, studying the influence of input resolution on latency and metrics. CLIP4Clip is based on the image retrieval architecture CLIP, so the results on the datasets are also reported for such model. In addition, a common retrieval dataset has been created starting from 1 week security camera recordings made available by AddFor S.p.A. A self-labelling

approach is used to produce captions for each clip, through a Transformer-based captioning model, VinVL [56]. The results show how, switching to a single-stage detector, TERN inference time is reduced by 10 times. Unfortunately, YOLO original architecture is not designed to produce the features required by TERN to work. Altering such model to produce them leads to a significant drop in metrics. For video retrieval solution, the experiments indicates that increasing input size is beneficial for the metrics up to a certain point. Further increases lead to worse results, demonstrating that, even if CLIP4Clip could potentially deal with higher resolutions, the original weights are well trained for small input sizes, and the finetuning strategy is not enough to tune them properly. Furthermore, higher resolutions lead to higher inference time. Also other parameters, such as sampling frequency and window size demonstrate to have an impact on latency. On the additional self-labelled common dataset, all the discussed models from both approaches are compared. Such dataset is composed of 5 seconds clips, recorded by a security camera over Piazza Solferino in Turin and offers a different domain with respect to original retrieval datasets. The automatically generated captions appear to be similar for the majority of the clips. This happen because the camera is always fixed to the same point. For this reason, the Recall values for all the models result very low, confirming the limitation of such metric in favor of Normalized Discounted Cumulative Gain (NDCG) that instead demonstrates to be more reliable. Original TERN model achieves the best results, far ahead of the modified single-stage version that pays the price for higher speed. CLIP4Clip models perform closely to the original TERN, with the potential advantage of exploiting temporal dimension to recognize more actions. From the overall results, it's possible to conclude that both approaches are valid options for the described application. Switching to single-stage object detector proves to be a good strategy to speed up inference but the compatibility of such architectures needs to be taken into account. Increasing clips resolution is costly in terms of inference and training time but can provide better results in certain conditions. Lastly, considering the additional dataset, the weights of the models demonstrate the ability to easily generalize on a new domain, security camera recordings. Not being trained on such data, the experiments testify the effectiveness of the model even without a specific finetuning, although this would still lead to better performance. This means that the application could work in very different contexts straight out of the box and such flexibility represents the main advantage of this approach.

Appendix A

Attention examples

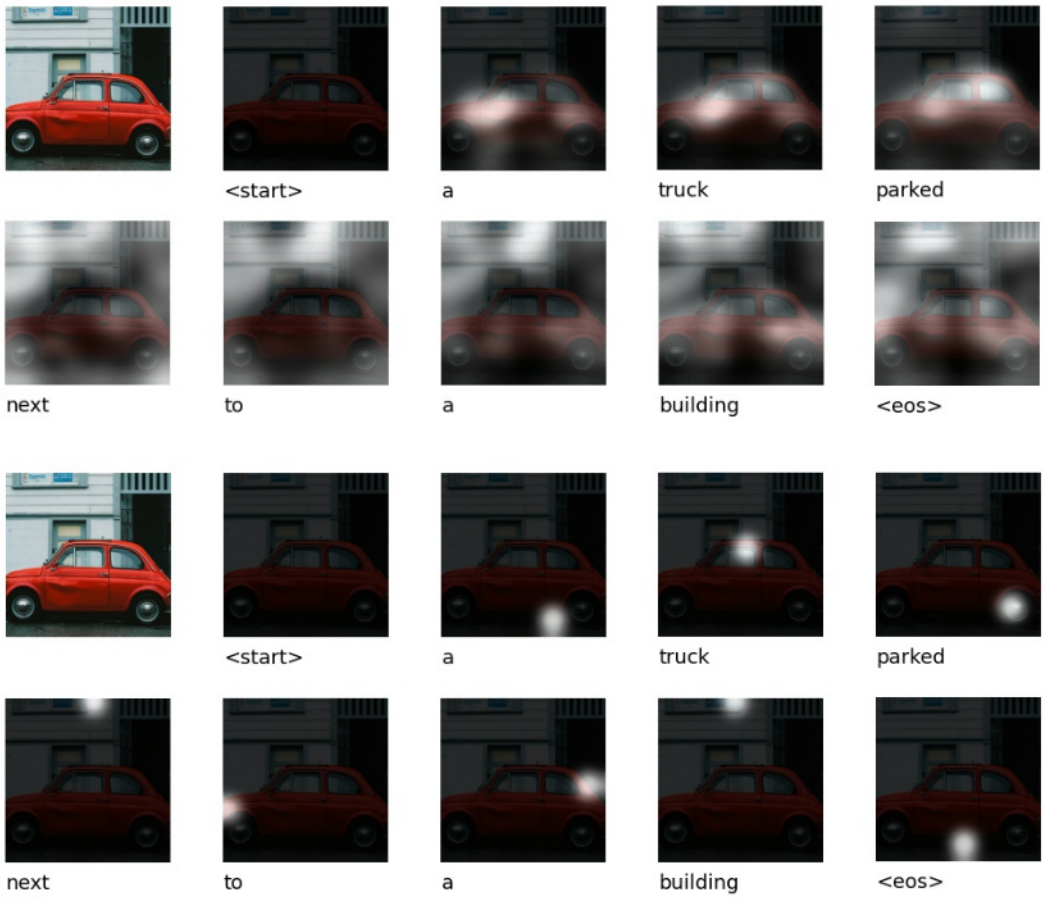


Figure A.1: Example 1. **Top**: Soft attention. **Bottom**: Hard attention. Lighter areas have higher attention scores.

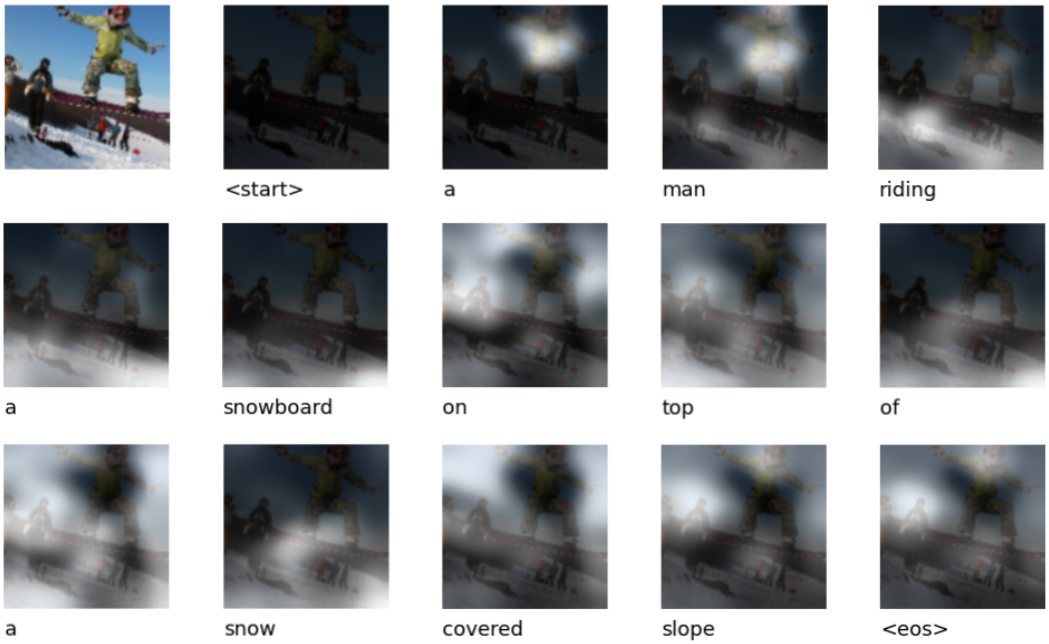


Figure A.2: Example 2 of soft attention.

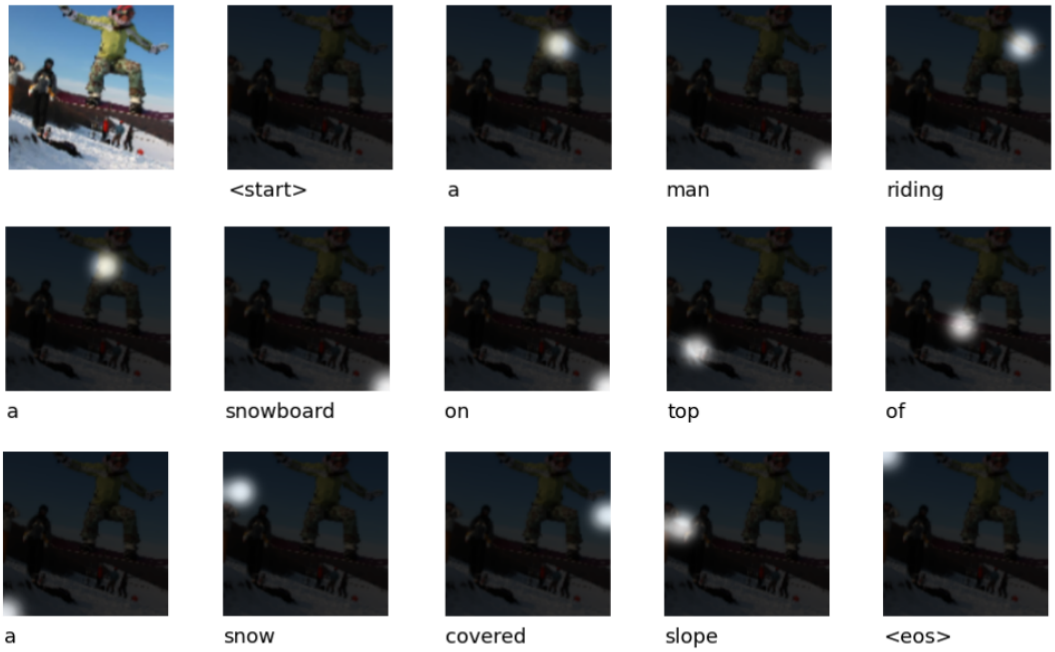


Figure A.3: Example 2 of hard attention.

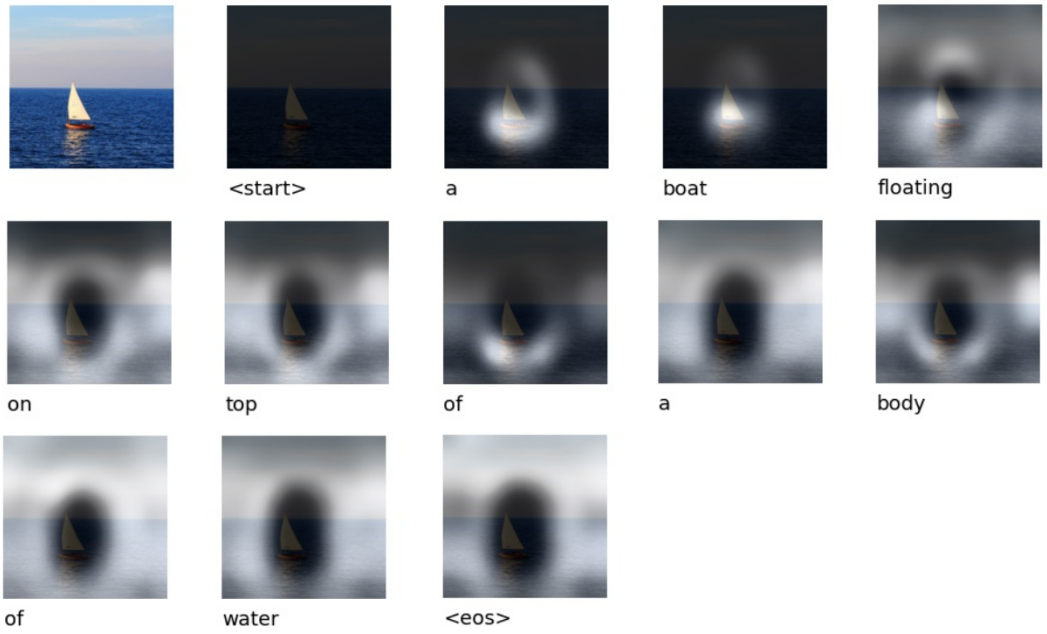


Figure A.4: Example 3 of soft attention.

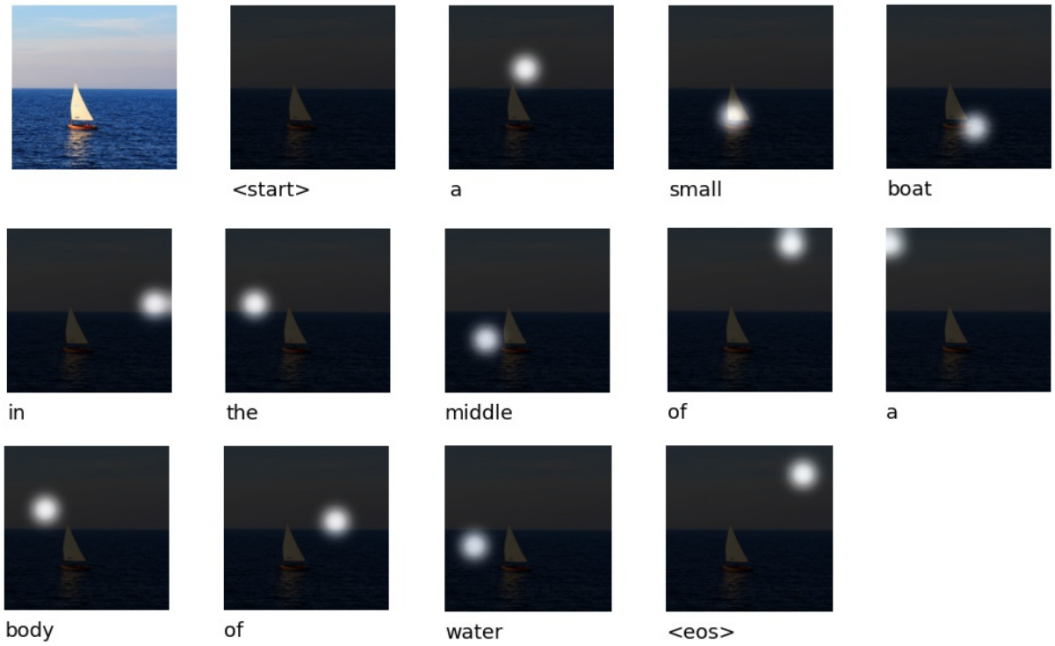


Figure A.5: Example 3 of hard attention.

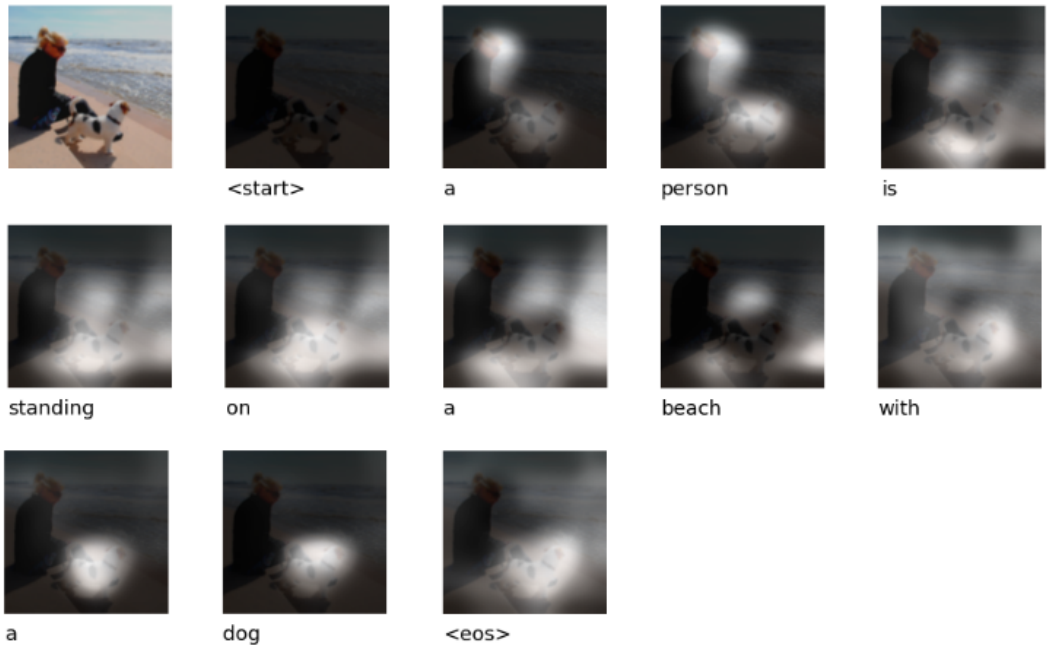


Figure A.6: Example 4 of soft attention.

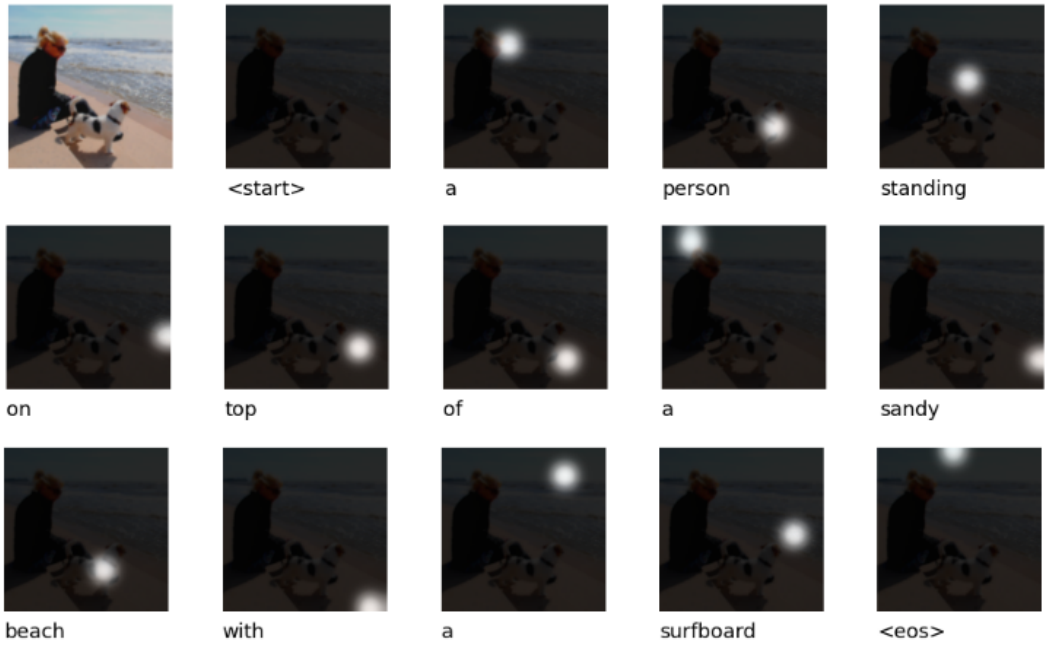


Figure A.7: Example 4 of hard attention.

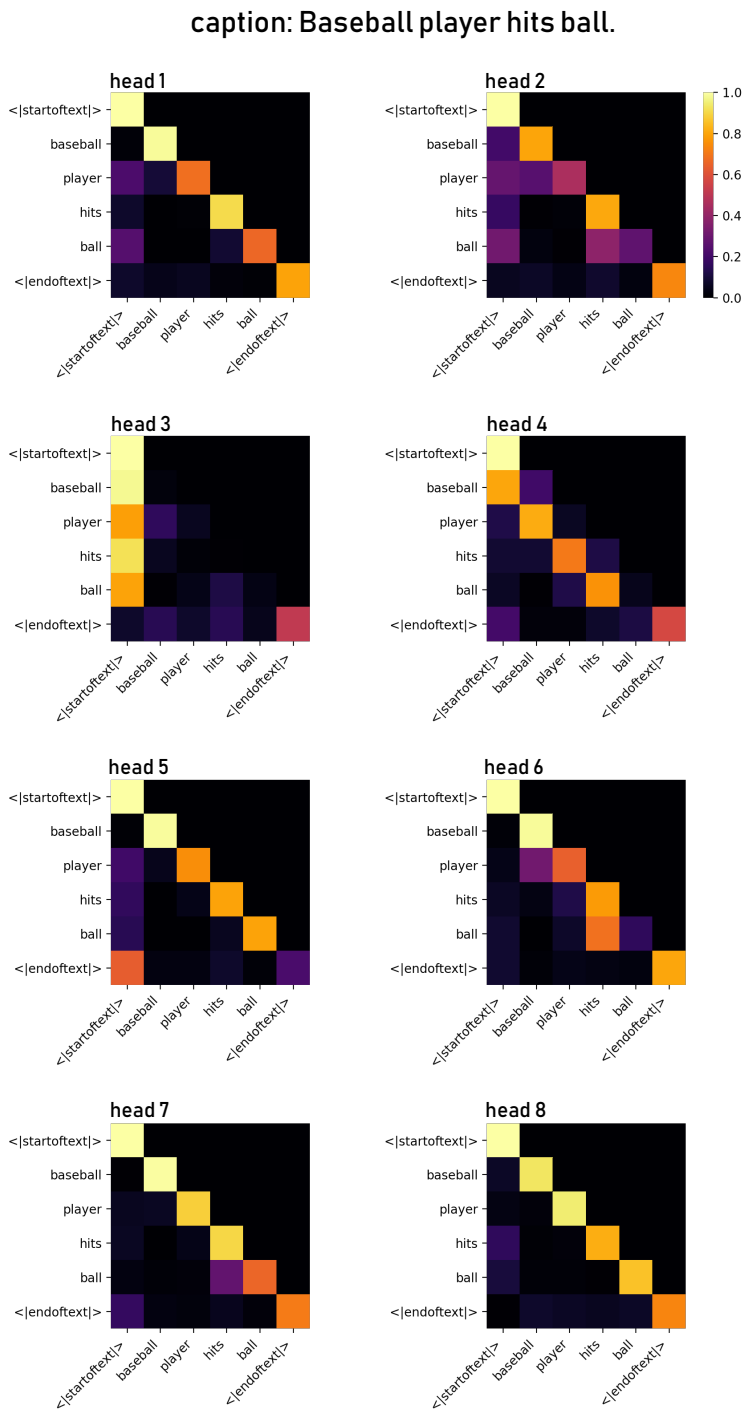


Figure A.8: Example of multi-head dot-product self-attention applied on an a caption. The scores are extracted from the first Transformer layer of the text branch of CLIP [21]. Here are reported the scores produced by each one of the 8 attention heads.

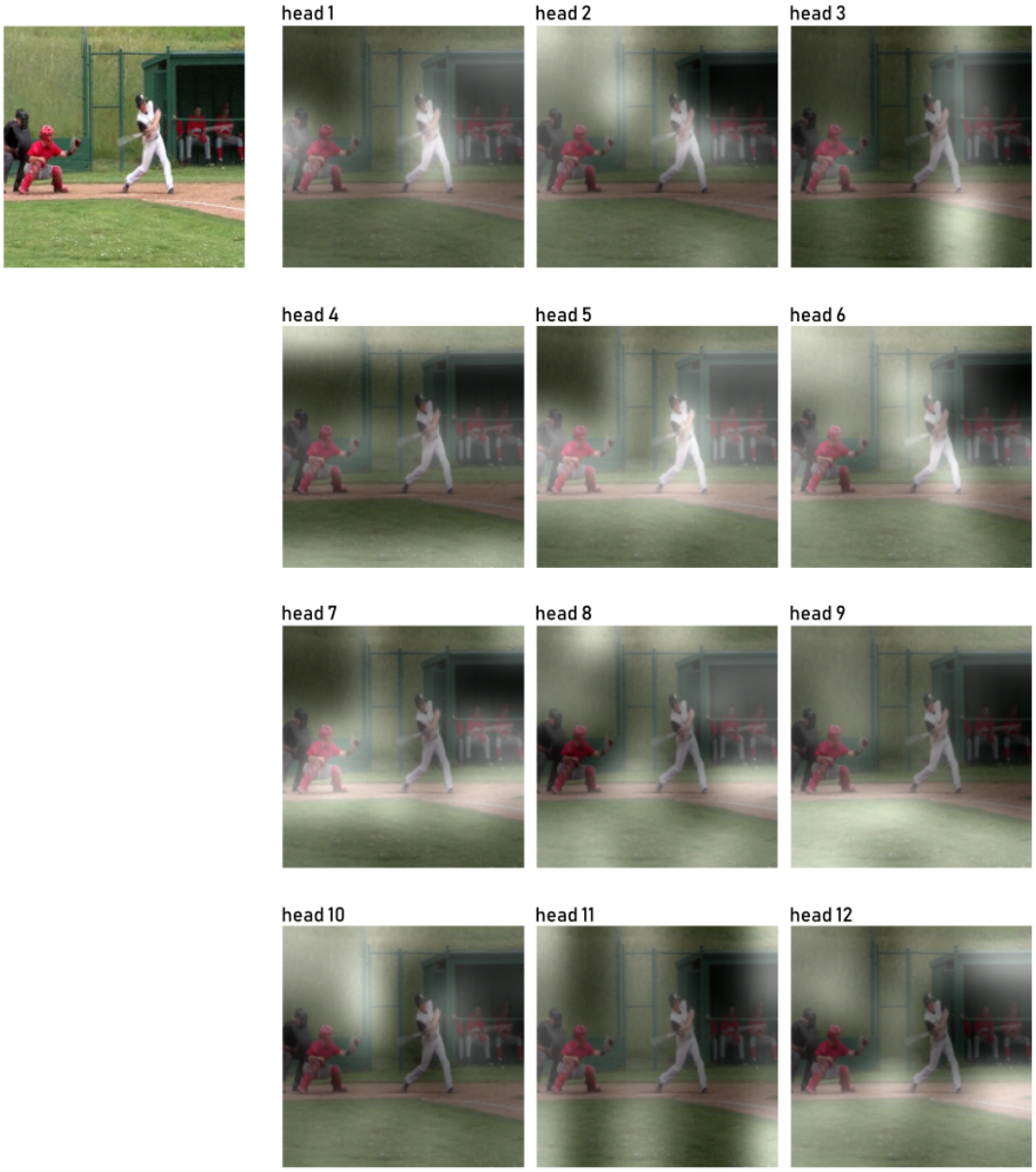


Figure A.9: Example of multi-head dot-product self-attention applied on an an image. Attention scores are calculated for each image region and then summed to obtain the reported visualization. Lighter areas have higher attention scores. The scores are extracted from the first Transformer layer of the text branch of CLIP [21]. Here are reported the scores produced by each one of the 12 attention heads.

Appendix B

MSR-VTT dataset

B.1 Caption analysis

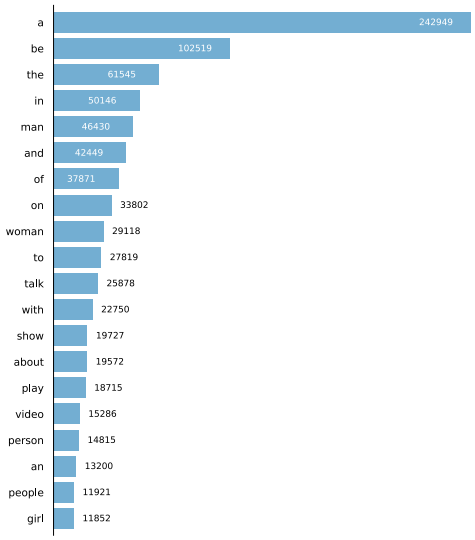


Figure B.1: Number of occurrences for the top 20 words in MSR-VTT [59] captions.

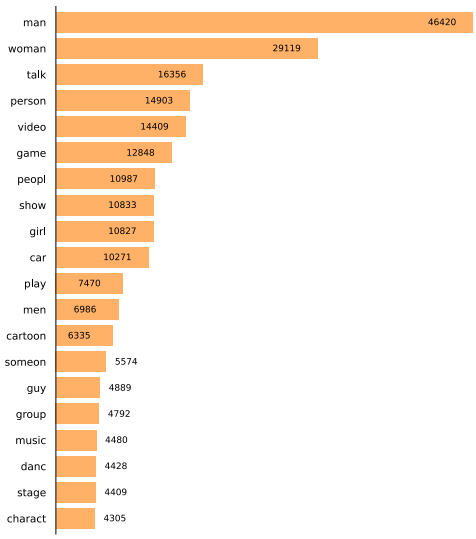


Figure B.2: Number of occurrences for the top 20 nouns in MSR-VTT [59] captions.

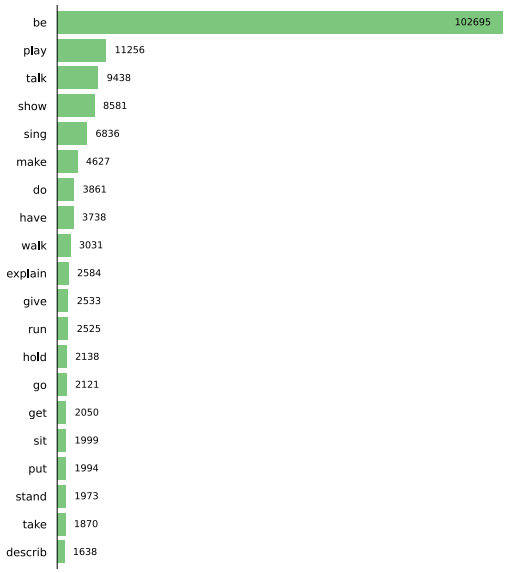


Figure B.3: Number of occurrences for the top 20 verbs in MSR-VTT [59] captions.

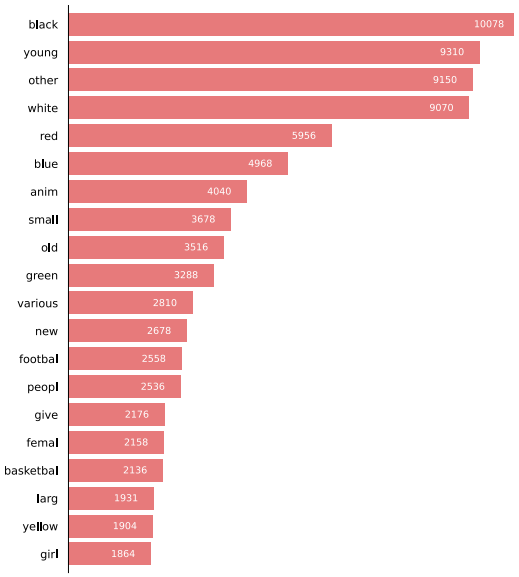


Figure B.4: Number of occurrences for the top 20 adjectives/adverbs in MSR-VTT [59] captions.

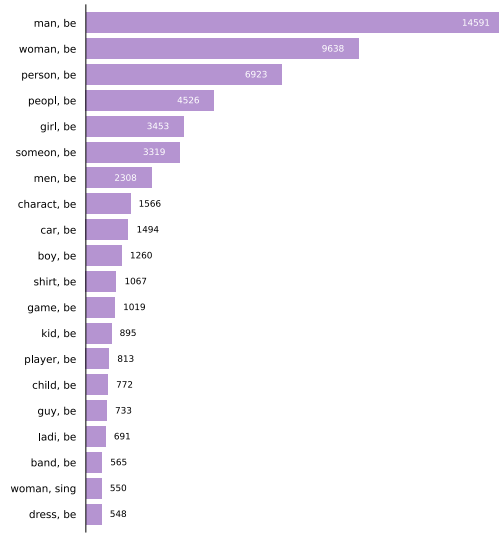


Figure B.5: Number of occurrences for the top 20 match for pattern [NOUN]-[VERB] in MSR-VTT [59] captions.

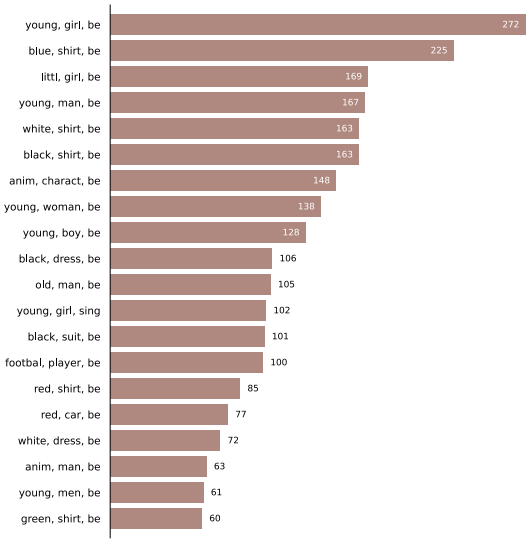


Figure B.6: Number of occurrences for the top 20 match for pattern [ADJECTIVE]-[NOUN]-[VERB] in MSR-VTT [59] captions.

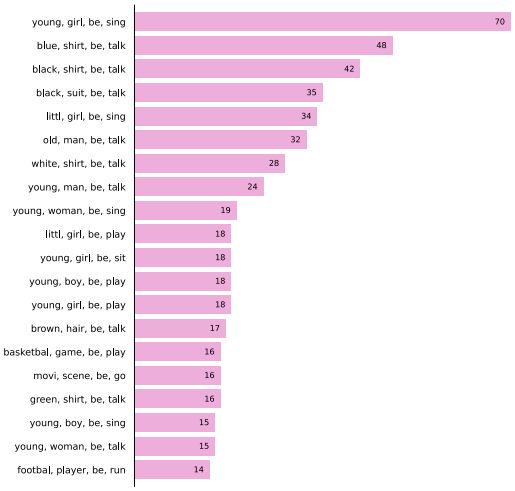


Figure B.7: Number of occurrences for the top 20 match for pattern [ADJECTIVE]-[NOUN]-[VERB]-[VERB] in MSR-VTT [59] captions.

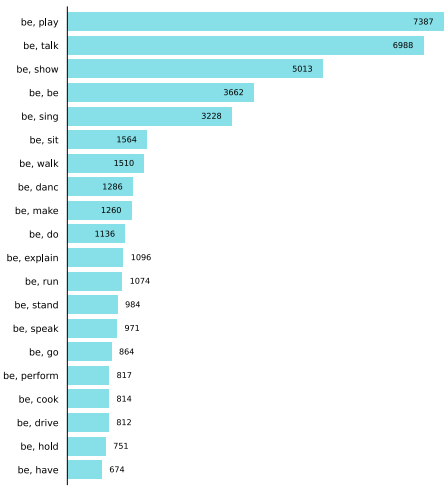


Figure B.8: Number of occurrences for the top 20 match for pattern [VERB]-[VERB] in MSR-VTT [59] captions.

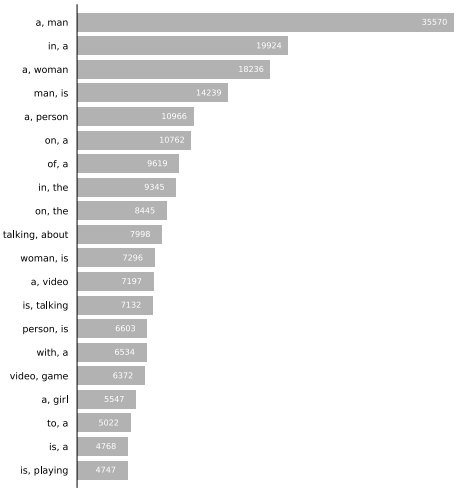


Figure B.9: Number of occurrences for the top 20 bigrams in MSR-VTT [59] captions.

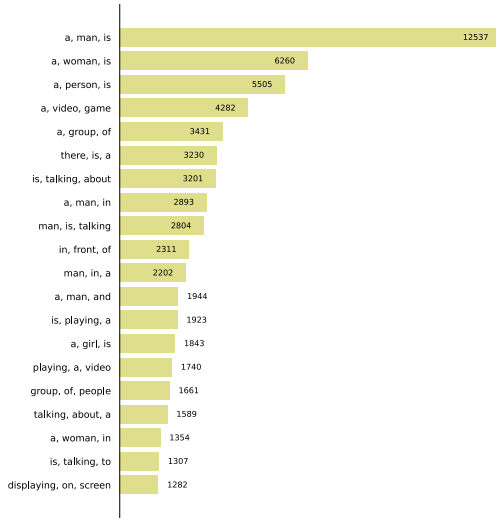


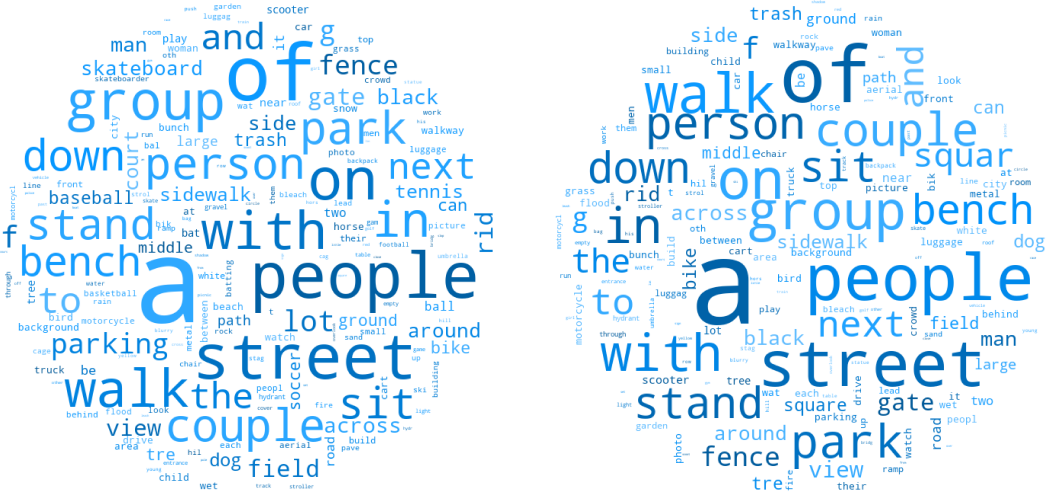
Figure B.10: Number of occurrences for the top 20 trigrams in MSR-VTT [59] captions.

Appendix C

Solferino dataset

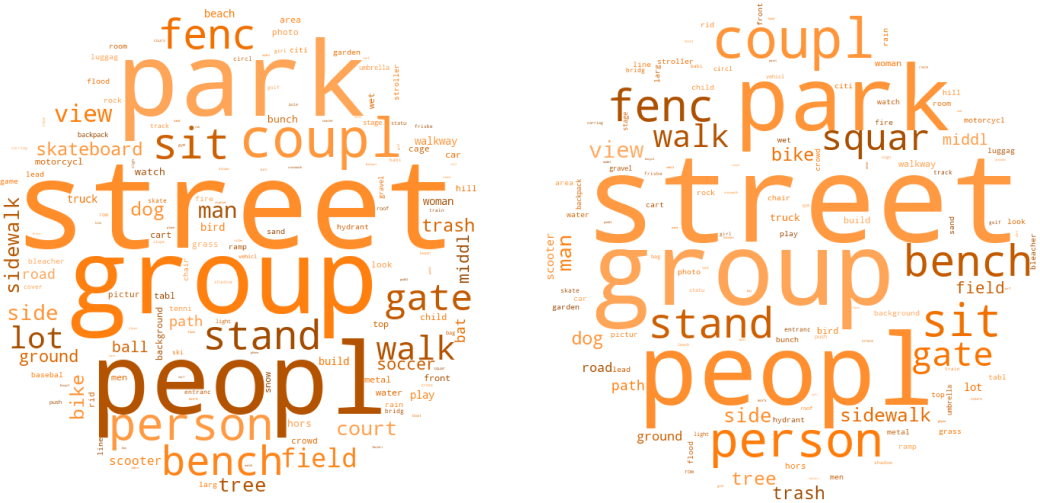
C.1 Manual correction

As previously discussed in Sec. 4.2.3, Solferino dataset relies on a self-labelling approach instead of manual annotation. The output of an object detector is carefully processed, before being fed to the captioning model, to allow the generation of multiple captions for each input image and to correct some common detection errors. Despite removing bounding box redundancy and including specific crops, some of the captions are still not coherent with the scene depicted in the frame. For this reason, a set of wrong patterns and common mistakes are manually fixed for all the original 3767 captions. The most common error is related to the location. Most of the time, the square is confused with a parking lot due to some white lines on the ground. When the scene is particularly clear of instances, the model tends to add fake details to it, starting from small cues. For example, at night, there are some lights on the ground that are mistaken for balls. Starting from it, the model generates captions describing sport matches, baseball bats, soccer players etc. On sunny days instead, the projected shadow of people is detected as a skateboard. All these captions are either removed or manually fixed to match the actual frame scene, resulting in a final set of 3684 captions. The wordclouds before and after manual correction are reported below in Figures C.1, C.2 and C.3. Note that, as for COCO caption analysis, words are lemmatized and stemmed, so some of them are only the roots of the corresponding words. Finally, in Figure 4.4, are reported some examples of annotations after manual correction.



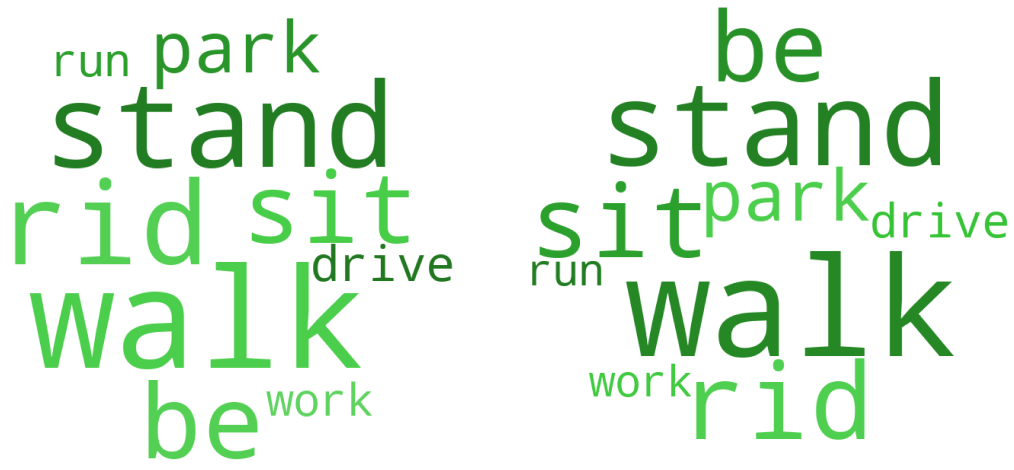
(a) **(b)**

Figure C.1: **(a)**: Wordcloud of captions words before manual correction, **(b)**: Wordcloud of captions words after manual correction.



(a) **(b)**

Figure C.2: **(a)**: Wordcloud of captions nouns before manual correction, **(b)**: Wordcloud of captions nouns after manual correction.



(a) **(b)**

Figure C.3: **(a)**: Wordcloud of captions verbs before manual correction, **(b)**: Wordcloud of captions verbs after manual correction.

Appendix D

Retrieval examples

As anticipated in Sec. 4.4.3, here are reported some inference examples for the final application. For each retrieved scene, timestamp and confidence are reported. Timestamp indicates the number of seconds since the beginning of the video while confidence is the value used to rank the candidates. For each query, the top 4 results are reported, ordered from top-left to bottom-right. Inference is done by the best performing model on Solferino dataset, TERN^{R-CNN}, so the candidates are exactly the plotted frames. Frames are extracted from different videos at different time of the day, recorded by the same security camera over Solferino square. Two different crops are present, one on the square (the one used to build Solferino dataset) and one on the street.

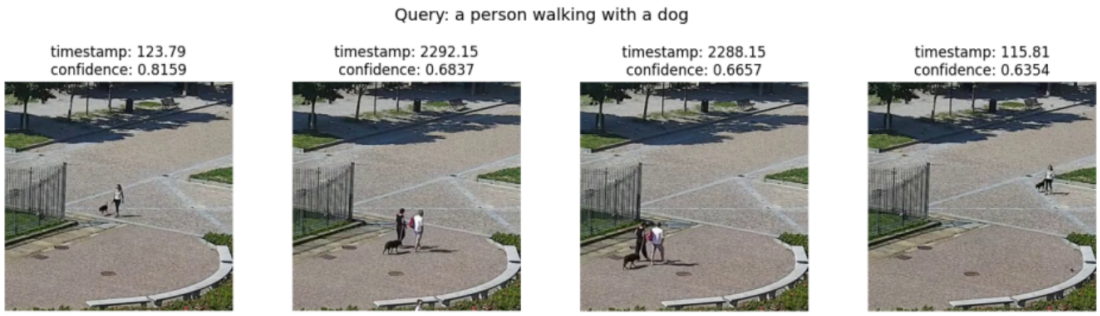


Figure D.1: Inference example 1.

(a)

(b)

Figure D.2: **(a)**: Inference example 2, **(b)**: Inference example 3.

(a)

(b)

Figure D.3: **(a)**: Inference example 4, **(b)**: Inference example 5.

Bibliography

[1] David G Lowe. «Distinctive image features from scale-invariant keypoints». In: *Int. J. Comput. Vis.* 60.2 (2004), pp. 91–110 (cit. on pp. ii, 7).

[2] Yan Huang, Wei Wang, and Liang Wang. *Instance-aware Image and Sentence Matching with Selective Multimodal LSTM*. 2016. arXiv: 1611.05588 [cs.CV] (cit. on pp. ii, 15).

[3] Hyeonseob Nam, Jung-Woo Ha, and Jeonghee Kim. *Dual Attention Networks for Multimodal Reasoning and Matching*. 2017. arXiv: 1611.00471 [cs.CV] (cit. on pp. ii, 16).

[4] Kunpeng Li, Yulun Zhang, Kai Li, Yuanyuan Li, and Yun Fu. *Visual Semantic Reasoning for Image-Text Matching*. 2019. arXiv: 1909.02701 [cs.CV] (cit. on pp. ii, 16).

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL] (cit. on pp. ii, 6, 13, 14, 18, 23, 39, 41).

[6] Nicola Messina, Fabrizio Falchi, Andrea Esuli, and Giuseppe Amato. *Transformer Reasoning Network for Image-Text Matching and Retrieval*. 2021. arXiv: 2004.09144 [cs.CV] (cit. on pp. iii, 7, 35, 43, 53, 57, 61, 62, 70).

[7] Huaishao Luo, Lei Ji, Ming Zhong, Yang Chen, Wen Lei, Nan Duan, and Tianrui Li. *CLIP4Clip: An Empirical Study of CLIP for End to End Video Clip Retrieval*. 2021. arXiv: 2104.08860 [cs.CV] (cit. on pp. iii, 40, 47, 58, 60, 63, 70).

[8] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. *Zero-Shot Text-to-Image Generation*. 2021. arXiv: 2102.12092 [cs.CV] (cit. on p. 2).

[9] Alane Suhr, Mike Lewis, James Yeh, and Yoav Artzi. «A Corpus of Natural Language for Visual Reasoning». In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Vancouver, Canada: Association for Computational Linguistics, July 2017, pp. 217–223. DOI: 10.18653/v1/P17-2034. URL: https://www.aclweb.org/anthology/P17-2034 (cit. on p. 4).

[10] Alane Suhr, Stephanie Zhou, Ally Zhang, Iris Zhang, Huajun Bai, and Yoav Artzi. *A Corpus for Reasoning About Natural Language Grounded in Photographs*. 2019. arXiv: 1811.00491 [cs.CL] (cit. on p. 4).

[11] Ranjay Krishna et al. *Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations*. 2016. arXiv: 1602.07332 [cs.CV] (cit. on pp. 4, 33, 50).

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. «Imagenet classification with deep convolutional neural networks». In: *NeurIPS*. 2012, pp. 1097–1105 (cit. on p. 8).

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. «ImageNet: A large-scale hierarchical image database». In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848 (cit. on pp. 8, 27, 28).

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV] (cit. on pp. 8, 17, 27, 39).

[15] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV] (cit. on pp. 8, 12, 15–17, 27).

[16] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. *Aggregated Residual Transformations for Deep Neural Networks*. 2017. arXiv: 1611.05431 [cs.CV] (cit. on pp. 8, 17, 27, 33, 49, 50).

[17] Sepp Hochreiter and Jürgen Schmidhuber. «Long Short-term Memory». In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735 (cit. on pp. 8, 12, 31, 41).

[18] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL] (cit. on pp. 8, 9).

[19] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation*. 2015. arXiv: 1508.04025 [cs.CL] (cit. on pp. 9, 12).

[20] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*. 2016. arXiv: 1502.03044 [cs.LG] (cit. on pp. 10–12).

[21] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV] (cit. on pp. 14, 15, 38, 40, 43, 47, 57–59, 61, 62, 76, 77).

[22] Lin Ma, Zhengdong Lu, Lifeng Shang, and Hang Li. *Multimodal Convolutional Neural Networks for Matching Image and Sentence*. 2015. arXiv: 1504.06063 [cs.CV] (cit. on p. 15).

[23] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. *OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks*. 2014. arXiv: 1312.6229 [cs.CV] (cit. on p. 15).

[24] M. Schuster and K.K. Paliwal. «Bidirectional Recurrent Neural Networks». In: *Trans. Sig. Proc.* 45.11 (Nov. 1997), pp. 2673–2681. ISSN: 1053-587X. DOI: 10.1109/78.650093. URL: https://doi.org/10.1109/78.650093 (cit. on p. 16).

[25] Yan Huang, Qi Wu, and Liang Wang. *Learning Semantic Concepts and Order for Image and Sentence Matching*. 2017. arXiv: 1712.02036 [cs.CV] (cit. on p. 16).

[26] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV] (cit. on pp. 16, 26, 28, 45, 54, 55, 61, 63, 70).

[27] Kuang-Huei Lee, Xi Chen, Gang Hua, Houdong Hu, and Xiaodong He. *Stacked Cross Attention for Image-Text Matching*. 2018. arXiv: 1803.08024 [cs.CV] (cit. on p. 16).

[28] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014. arXiv: 1412.3555 [cs.NE] (cit. on p. 16).

[29] Atousa Torabi, Niket Tandon, and Leonid Sigal. *Learning Language-Visual Embedding for Movie Understanding with Natural-Language*. 2016. arXiv: 1609.08124 [cs.CV] (cit. on p. 17).

[30] Guy Lev, Gil Sadeh, Benjamin Klein, and Lior Wolf. *RNN Fisher Vectors for Action Recognition and Image Annotation*. 2015. arXiv: 1512.03958 [cs.CV] (cit. on p. 17).

[31] Dotan Kaufman, Gil Levi, Tal Hassner, and Lior Wolf. *Temporal Tessellation: A Unified Approach for Video Analysis*. 2017. arXiv: 1612.06950 [cs.CV] (cit. on p. 17).

[32] Joao Carreira and Andrew Zisserman. *Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset*. 2018. arXiv: 1705.07750 [cs.CV] (cit. on p. 17).

[33] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. *Rethinking Spatiotemporal Feature Learning: Speed-Accuracy Trade-offs in Video Classification*. 2018. arXiv: 1712.04851 [cs.CV] (cit. on p. 17).

[34] Niluthpol Chowdhury Mithun, Juncheng Li, Florian Metze, and Amit K. Roy-Chowdhury. «Learning Joint Embedding with Multimodal Cues for Cross-Modal Video-Text Retrieval». In: *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*. ICMR '18. Yokohama, Japan: Association for Computing Machinery, 2018, pp. 19–27. ISBN: 9781450350464. DOI: 10.1145/3206025.3206064. URL: https://doi.org/10.1145/3206025.3206064 (cit. on p. 17).

[35] Yang Liu, Samuel Albanie, Arsha Nagrani, and Andrew Zisserman. *Use What You Have: Video Retrieval Using Representations From Collaborative Experts*. 2020. arXiv: 1907.13487 [cs.CV] (cit. on p. 17).

[36] Antoine Miech, Dimitri Zhukov, Jean-Baptiste Alayrac, Makarand Tapaswi, Ivan Laptev, and Josef Sivic. *HowTo100M: Learning a Text-Video Embedding by Watching Hundred Million Narrated Video Clips*. 2019. arXiv: 1906.03327 [cs.CV] (cit. on p. 17).

[37] Maksim Dzabraev, Maksim Kalashnikov, Stepan Komkov, and Aleksandr Petiushko. *MDMMT: Multidomain Multimodal Transformer for Video Retrieval*. 2021. arXiv: 2103.10699 [cs.CV] (cit. on p. 17).

[38] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL] (cit. on p. 20).

[39] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. «GloVe: Global Vectors for Word Representation». In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162 (cit. on p. 20).

[40] Vinod Nair and Geoffrey E. Hinton. «Rectified Linear Units Improve Restricted Boltzmann Machines». In: *ICML*. 2010 (cit. on p. 25).

[41] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL] (cit. on pp. 26, 32, 34, 35, 41).

[42] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV] (cit. on pp. 26, 39, 42, 43, 59, 60).

[43] Ross Girshick. *Fast R-CNN*. 2015. arXiv: 1504.08083 [cs.CV] (cit. on p. 27).

[44] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV] (cit. on pp. 28, 29, 55).

[45] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV] (cit. on p. 28).

[46] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. arXiv: 1612.08242 [cs.CV] (cit. on p. 28).

[47] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV] (cit. on pp. 28, 29, 33, 45, 46, 49, 50, 56, 62).

[48] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. arXiv: 1804.02767 [cs.CV] (cit. on p. 28).

[49] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV] (cit. on pp. 28, 55).

[50] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. *Scaled-YOLOv4: Scaling Cross Stage Partial Network*. 2021. arXiv: 2011.08036 [cs.CV] (cit. on pp. 28, 45, 46, 55, 61–63, 70).

[51] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, and Jun-Wei Hsieh. *CSPNet: A New Backbone that can Enhance Learning Capability of CNN*. 2019. arXiv: 1911.11929 [cs.CV] (cit. on p. 29).

[52] Sen He, Wentong Liao, Hamed R. Tavakoli, Michael Yang, Bodo Rosenhahn, and Nicolas Pugeault. *Image Captioning through Image Transformer*. 2020. arXiv: 2004.14231 [cs.CV] (cit. on p. 30).

[53] Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. *Language Modeling with Gated Convolutional Networks*. 2017. arXiv: 1612.08083 [cs.CL] (cit. on p. 31).

[54] Marcella Cornia, Matteo Stefanini, Lorenzo Baraldi, and Rita Cucchiara. *Meshed-Memory Transformer for Image Captioning*. 2020. arXiv: 1912.08226 [cs.CV] (cit. on p. 31).

[55] Xiujun Li et al. *Oscar: Object-Semantics Aligned Pre-training for Vision-Language Tasks*. 2020. arXiv: 2004.06165 [cs.CV] (cit. on pp. 32, 43).

[56] Pengchuan Zhang, Xiujun Li, Xiaowei Hu, Jianwei Yang, Lei Zhang, Lijuan Wang, Yejin Choi, and Jianfeng Gao. *VinVL: Revisiting Visual Representations in Vision-Language Models*. 2021. arXiv: 2101.00529 [cs.CV] (cit. on pp. 33, 43, 49, 71).

[57] Shuai Shao, Zeming Li, Tianyuan Zhang, Chao Peng, Gang Yu, Xiangyu Zhang, Jing Li, and Jian Sun. «Objects365: A Large-Scale, High-Quality Dataset for Object Detection». In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2019 (cit. on pp. 33, 50).

[58] Alina Kuznetsova et al. «The Open Images Dataset V4». In: *International Journal of Computer Vision* 128.7 (Mar. 2020), pp. 1956–1981. ISSN: 1573-1405. DOI: 10.1007/s11263-020-01316-z. URL: http://dx.doi.org/10.1007/s11263-020-01316-z (cit. on pp. 33, 50).

[59] Jun Xu, Tao Mei, Ting Yao, and Yong Rui. «MSR-VTT: A Large Video Description Dataset for Bridging Video and Language». In: IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), June 2016. URL: https://www.microsoft.com/en-us/research/publication/msr-vtt-a-large-video-description-dataset-for-bridging-video-and-language/ (cit. on pp. 41–43, 45–47, 49, 58, 60, 63, 78–80).

[60] David Chen and William Dolan. «Collecting Highly Parallel Data for Paraphrase Evaluation». In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 190–200. URL: https://www.aclweb.org/anthology/P11-1020 (cit. on p. 43).

[61] Anna Rohrbach, Marcus Rohrbach, and Bernt Schiele. *The Long-Short Story of Movie Description*. 2015. arXiv: 1506.01698 [cs.CV] (cit. on p. 43).

[62] Ranjay Krishna, Kenji Hata, Frederic Ren, Li Fei-Fei, and Juan Carlos Niebles. *Dense-Captioning Events in Videos*. 2017. arXiv: 1705.00754 [cs.CV] (cit. on p. 43).

[63] Lisa Anne Hendricks, Oliver Wang, Eli Shechtman, Josef Sivic, Trevor Darrell, and Bryan Russell. *Localizing Moments in Video with Natural Language*. 2017. arXiv: 1708.01641 [cs.CV] (cit. on p. 43).