# Automatic Video Captioning Using Spatiotemporal Convolutions on Temporally Sampled Frames

by

Simbarashe Linval Nyatsanga

*Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Applied Mathematics in the Faculty of Science at Stellenbosch University*

Supervisor: Prof. Willie Brink

March 2020

# Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: ..........March 2020..........

# Abstract

Being able to concisely describe content in a video has tremendous potential to enable better categorisation, indexed based-search and fast content-based retrieval from large video databases. Automatic video captioning requires the simultaneous detection of local and global motion dynamics of objects, scenes and events, to summarise them into a single coherent natural language description. Given the size and complexity of video data, it is important to understand how much temporally coherent visual information is required to adequately describe the video. In order to understand the association between video frames and sentence descriptions, we carry out a systematic study to determine how the quality of generated captions changes with respect to densely or sparsely sampling video frames in the temporal dimension. We conduct a detailed literature review to better understand the background work in image and video captioning. We describe our methodology for building a video caption generator, which is based on deep neural networks called encoder-decoders. We then outline the implementation details of our video caption generator and our experimental setup. In our experimental setup, we explore the role of word embeddings for generating sensible captions with pretrained, jointly trained and finetuned embeddings. We train and evaluate our caption generator on the Microsoft Video Description (MSVD) dataset. Using the standard caption generation evaluation metrics, namely BLEU, METEOR, CIDEr and ROUGE, our experimental results show that sparsely sampling video frames with either finetuned or jointly trained embeddings, results in the best caption quality. Our results are promising in the sense that high quality videos with a large memory footprint could be categorised through a sensible description obtained through sampling a few frames. Finally, our method can be extended such that the sampling rate adapts according to the quality of the video.

# Opsomming

Die vermoë om 'n video se inhoud bondig te beskryf, het geweldige potensiaal vir beter kategorisering, indeksgebaseerde soektogte, en vinnige inhoudgebaseerde ontrekking uit groot video databasisse. Die outomatiese generering van video-onderskrifte vereis die gelyktydige opsporing van lokale en globale bewegingsdinamika van voorwerpe, tonele en gebeure, om in 'n enkele, samehangende, natuurlike taalbeskrywing opgesom te word. Vanweë die grootte en kompleksiteit van video data is dit belangrik om te verstaan hoeveel tyd-samehangende visuele inligting nodig is om die video voldoende te beskryf. Ten einde die verband tussen video-rame en sinbeskrywings te verstaan, voer ons 'n sistematiese studie uit om te bepaal hoe die gehalte van gegenereerde onderskrifte verander soos video-rame digter of yler in die tyd-dimensie gemonster word. Ons voer 'n gedetailleerde literatuurstudie uit om bestaande werk in die generering van beeld- en video-onderskrifte beter te verstaan. Ons beskryf ons metodologie vir die bou van 'n video-onderskrifgenerator, wat gebaseer is op diep neurale netwerke wat enkodeerder-dekodeerders genoem word. Ons gee dan 'n uiteensetting van die implementeringsbesonderhede van ons video-onderskrifgenerator en ons eksperimentele opstelling. In ons eksperimentele opstelling ondersoek ons die rol van woordinbeddings vir die generering van sinvolle onderskrifte met vooraf-afgerigte, gesamentlik-afgerigte, en verfynde inbeddings. Ons onderskrifgenerator word afgerig en evalueer op die Microsoft Video Description (MSVD) datastel. Deur gebruik te maak van standaard evalueringsmaatstawwe, naamlik BLEU, METEOR, CIDEr en ROUGE, toon ons eksperimentele resultate dat yl gemonsterde video-rame, met verfynde of gesamentlik-afgerigte inbeddings, die beste onderskrifkwaliteit lewer. Ons resultate is belowend in die sin dat hoë gehalte video's met groot geheue-vereistes gekategoriseer kan word, deur middel van sinvolle beskrywings vanaf enkele rame. Ons metode kan ook uitgebrei word deur die monstertempo aan te pas volgens die kwaliteit van die video.

# Contents

# 1 Introduction

Deep learning has become a popular solution in multiple disciplines, unifying seemingly disparate domains including healthcare, transportation, computational simulation and language modelling. It has also revolutionised computer vision in many applications. Using an end-to-end trainable model, a machine can now perform image classification, instance segmentation, object detection and natural language understanding in the form of translation, comprehension and question answering. Furthermore, and of interest to this study, is caption based visual understanding where a machine can take as input an image or a video and produce a natural language description of the input's content. Although a challenging task, deep learning architectures have made it possible to incorporate computer vision and natural language processing models into a single architecture, allowing a model to not only encode the visual features in the input but be able to "translate" them into a natural language description.

In image captioning the model's objective is to find a mapping from the image's visual representation to a natural language sentence that concisely describes the content. In mathematical terms, this means finding a smooth, differentiable function mapping from a *source probability distribution* over raw pixel intensities to an *output probability distribution* over a sequence of words. In video captioning, the problem is further compounded by the sheer size and complexity of the input data that needs to be handled. The nature of a video stream with high temporal dependencies, complex actions and object interactions as well as variable length, makes it a much more challenging problem. Despite the challenge, many architectures have been proposed resulting in substantial progress in both image and video captioning research. At the heart of the majority of these architectures is the so-called *encoder-decoder* architecture, a unified end-to-end model composed of an encoder and a decoder inspired by work in neural machine translation [1]. The encoder converts the input into a compact vector representation and the decoder takes as input the set of previously generated words, an internal state and the vector representation of the image, to generate the next word in a caption.

While there has been significant progress in both image and video captioning, the former has seen relatively more progress because of the availability of large, diverse and well captioned image datasets, e.g. Flickr8k, Flickr30k and MSCOCO, upon which models can be trained [2–4]. Furthermore, transfer learning, where model parameters learnt in one domain can be applied in a different domain, has allowed the image captioning task to take advantage of state-of-the-art models in image classification and object detection [5–8]. This is possible because the relative simplicity of image data compared to video data has given rise to a generic image descriptor that can be used in other related tasks. Image captioning is one of these tasks, where the image descriptor features are used to influence the prediction of words, generating a sentence that correctly describes the image's content. Despite the difference in pace of progress, several fundamental mechanisms are either shared or transferrable from the image to the video captioning task, including the use of encode-decoder architectures [9, 10], devising a common embedding space between the visual and semantic modalities [11–13] and the use of attention mechanisms for fine-grained captioning [10, 14].

For encoding visual information, convolutional neural networks (CNNs) have been extremely successful at visual representation in practical applications including image and video classification, object detection and instance segmentation [5,6,8,15], making them a suitable choice for image and video captioning. For generating sequences, recurrent neural networks (RNNs) have been found to be effective for sequence modelling in machine translation and sentiment analysis [1, 16–18]. Therefore the majority of model architectures in image or video captioning use a combination of a CNN encoder and RNN decoder to generate descriptions of visual content [9, 10, 12, 13, 19–22], although RNNs have also been effectively used as an encoder for videos given their ability to process temporal data [20, 23, 24].

Building upon these established successes in image and video captioning, this study is interested in further progress in video captioning of open domain videos. An important question in that regard is *"How far apart do frames of a video need to be, in order to adequately describe its content?"* Particularly, we aim to understand how the sampling rate, when preprocessing videos, affects the quality of the generated captions. Moreover, just as a generic image descriptor has been found to be effective for image captioning, a generic video descriptor called spatiotemporal convolutions (3D CNN), was proposed by Tran et al. [25] which preserves a video's temporal coherence. A video's temporal coherence is important because understanding the order in which objects interact and when events occur, leads to better video recognition as evidenced by impressive results in video classification and action detection tasks [25, 26]. In video captioning, temporal coherence can lead to more sensible natural language descriptions. Therefore in this study, we intuit that a generic video descriptor, constructed using an optimal sampling rate that maintains temporal coherence, is an important constituent for a video caption generator that can generalise to open domain videos. We aim to verify this suggestion by training and evaluating our caption generator on an open domain video dataset [27]. Additionally, we suggest that a sufficiently low sampling rate that maintains temporal coherence for quality caption generation has far reaching implications for efficient processing of typically large video datasets.

An equally important issue in our study is the role of word embeddings when generating natural language descriptions that are specifically meant to be visually informative. Word embeddings encode the meaning of words in a geometric space where distance can be interpreted as how semantically close any two words are to each other. The intuition is informed by the distributional hypothesis which states that words that are frequently used or appear in similar contexts have similar meaning [28]. Word embeddings have become an integral part of generative models in natural language processing because they can encode vital co-occurence context leading to signficantly higher quality and sensible sentences [29–32]. Moreover, word embeddings have contributed significantly to transfer learning in natural language processing because word embeddings trained in one task, e.g. text classification, can be applied in a different task, e.g. caption generation [33]. Word embeddings can either be pretrained on large text corpora, jointly trained as part of a primary task, or they can be used as a primer for a neural network layer and then finetuned for the desired task. In our study, we are interested in comparing the efficacy of these three embedding configurations in order to generate sensible and visually informative video captions.

We carry out a systematic evaluation to determine what frame sampling rate is required in order to adequately describe a video for applications in automatic description, cataloguing and content based retrieval. The work of Tran et al. [25] extracts only 16 frames from every variable length video in order to create spatiotemporal features for video classification. We also use 16 frame long spatiotemporal features, but vary the sampling rate from temporally close frames to temporally distant frames. We also devise three model configurations where we use pretrained GLOVE embeddings [31], jointly trained embeddings and finetuned GLOVE embeddings, and compare the quality of generated captions. For all model configurations, we use the BLEU, METEOR, ROUGE and CIDEr [34–38] caption quality metrics to determine the best configuration of spatiotemporal features and word embeddings. Firstly, since temporal coherence is important for both video classification and captioning [25, 26], we study the inherent trade-off between densely sampled spatiotemporal features that will result in more accurate captions but potentially miss significant video sections, and sparsely sampled features that cover longer video sections but potentially start to lose temporal coherence. Secondly, we did a comparative study of the effects on captioning quality when using word embeddings that are pretrained on a large text corpus, versus those jointly trained or finetuned on a much smaller task specific dataset.

Automatic video captioning has many important practical applications. As of 2019, about 500 hours of video content are uploaded to YouTube every minute [39]. However poorly tagged videos can make search difficult and lead to poor search results [14]. Video captioning can provide a robust solution to improve cataloguing and indexing for better search results as well as content based retrieval. In conjuction with speech synthesis technology, annotating videos with natural language descriptions can potentially also improve accessibility of videos for the visually impaired.

Our study is structured into five major chapters. In the Literature Review, we give a detailed review of the background work that has been done in image and video caption generation. We delve into different techniques that have become prominent in both image and video captioning and highlight the shared ideas between the two. In the Technical Background, we discuss the foundational mechanisms of neural networks that form the basis of our deep learning based approach to video caption generation. We detail the anatomy of neural networks, how they are trained and the specific optimisation based techniques used to obtain task specific weights. We also give a detailed review of CNNs and RNNs, the constituents of our encoder-decoder caption generator. In the Proposed Approach, we discuss the implementation details of our encoder-decoder architecture. We give a detailed architectural overview of our spatiotemporal convolutions and the merge strategy used to incorporate visual features in to the caption generation process. We give a theoretical overview of the evaluation metrics we chose to implement for measuring the caption quality of our model. In the Experiments and Results, we discuss the experimental setup for comparing differen spatiotemporal strides as well as word embedding configurations. We present all quantitative results according to BLEU, METEOR, ROUGE and CIDEr. For qualitative results, we give illustrative examples of the captions generated by our model. We conclude the chapter with a short discussion to explain our observed results. The Conclusion and Future Work completes our study by discussing the limitations of our study as well as highlighting potentially important areas that can be improved within our study and in video caption generation as a whole.

# 2 Literature Review

The success of deep learning models in many machine learning tasks [40] has led to the dominance of CNNs in computer vision tasks such as image classification [5,6,15,41], object recognition [7,8,42,43], speech recognition [44], and video classification [26]. Meanwhile, RNNs have dominated sequence modelling tasks such as neural machine translation [1,18], sentiment analysis [16,17] and modelling time series data [45], and have been found to be surprisingly effective for image and video recognition [20]. The success of these two classes of models has led to substantial interest and research in solving the problem of automatically describing visual data using natural language. In deep learning approaches, a model can understand visual input in the form of an image or video and be able to articulate this understanding in the form of a single or multiple natural language sentences. This has advanced both image and video caption generation research where different configurations of CNNs, RNNs or both have been used. In this chapter, we give a detailed review of research in image and video captioning, and highlight their close relationship. In particular, the literature shows that several mechanisms are shared between the two captioning problems, including multimodal learning, attention and encoder-decoder architectures. We first discuss the mechanisms in the context of image captioning, then discuss how video captioning also utilises them.

## 2.1 Image Captioning

Multiple approaches have been proposed to tackle the image captioning problem. We describe three broad deep learning based approaches, namely multimodal representations, encoder-decoder representations and dense image captioning.

### Multimodal Representations

A significant amount of research has reframed the problem of image captioning as multimodal with the intuition that the semantic data, to some degree, identifies and describes the entire image or some region of it. This entails devising a common multimodal space that associates and encapsulates the similarities between the image representation and the semantic representation, thus associating the image with its sentence description. Practically, this is done by learning a common visual-semantic embedding in order to directly associate image segments with natural language phrases.

Kiros et al. [46] introduced multimodal neural language models, which are adaptions of neural language models that can be influenced by other modalities including images, speech and video. The motivation is that descriptive language almost always appears within the context of other modalities, e.g. images accompanying a textual advertisement or natural language descriptions of image or video content. Hence by jointly learning an image-text representation, a neural language model can be used to generate descriptions of images in addition to enabling the retrieval of an image given a natural language query. They improved on a probabilistic language model [47] by either injecting the image context vector as an additive bias to the word prediction, or using the image context vector to modulate the word representation matrix in order to influence the conditional probability scoring towards the correct word. This approach generates full sentence descriptions but uses a fixed context window as opposed to an RNN which typically has access to all previously generated words.

Kiros et al. [48] directly built on the multimodal neural language models by introducing an encoder-decoder architecture that unifies the task of first learning a joint image-text representation through an encoder and then a neural language model as a decoder to generate the description. Importantly, this work represents the first time the encoder-decoder architectures were adapted from work in neural machine translation [1, 18] for image caption generation. In addition to learning a common image-text representation, image caption generation can be thought of as the task of translating an image into a description, analogous to translating from a source language to a target language. At the encoding stage the image feature from a CNN [5, 15] is projected on the embedding space of the hidden states of an LSTM [49] and at the decoding stage, a neural language model decodes the encoding into a natural language description that is of signficantly better quality compared to previous work [46]. The improvement in quality is primarily due to the neural language model being able to decompose the sentence structure to its content, influenced by representations produced by the encoder. While this constitutes an improvement on previous solutions, the approach still generates fairly simplistic captions that do not adequately capture various details in images. Attention based models, in which the decoder learns to focus on certain image regions while generating a sentence, have been shown to generate more descriptive captions [10].

Karpathy et al. [11] improved the performance of multimodal representations by formulating a structured max-margin objective for a deep neural network that learns to embed both visual and language data in a common multi-modal space. They do so by not only learning the image-text embeddings, but also learning a fine-grained embedding space between image fragments (objects) and sentence fragments (phrases) which are represented as dependency tree relations [50]. Using a Region-CNN for object detection [7] pretrained on the ImageNet dataset [51], the model is able to break down an image into fragments and explicitly associate them with specific sentence fragments, resulting in more interpretable image-phrase associations in the generated captions. The model learns a latent association between the image and sentence fragments by computing an inner product between their embedding vectors, as illustrated in Figure 1.

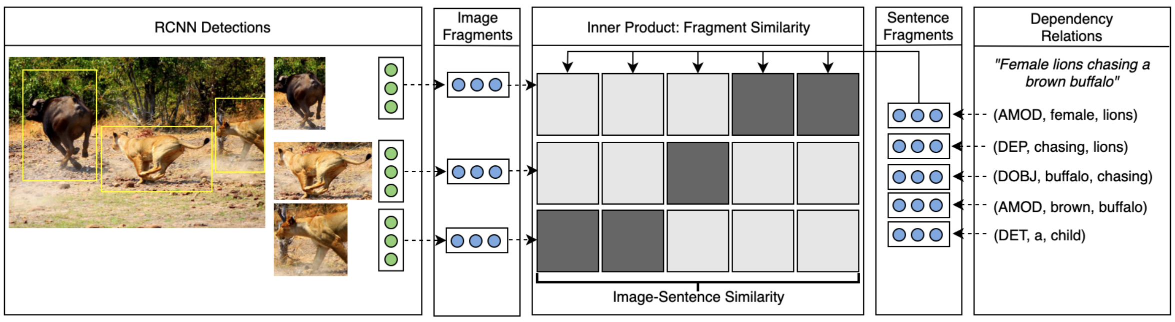


Figure 1: Computing the respective fragments and image-sentence similarities. **Left:** CNN representations of detected objects (green) are mapped into the fragment embedding space (blue). **Right:** Dependency tree relations in the caption are embedded. **Centre:** The model then interprets the individual inner products as a similarity score. The shaded boxes signify the degree of cross fragment alignment. Image inspired by [11].

The model then computes a global image-sentence score as a fixed function of the scores of their respective fragments. Intuitively, an image-sentence pair will obtain a high score if the respective fragments are highly compatible. The result is that given an image, the model can retrieve the highest scoring corresponding sentences (image captioning) and conversely given a sentence, the model retrieves the highest scoring image (image retrieval). Overall this use of typed tree dependencies offers a rich set of relationships for explicitly identifying objects in images. While the use of dependency trees provides an efficient representation of sentences, it is somewhat rigid and not always approriate because certain complex sentences that describe a single object can be parsed as multiple fragments, for example *"white and brown hamster"* can be parsed as two relations *(and, white, brown)* and *(attr, white, hamster)* leading to two fragments that neither fully identify the object in the image. Additionally there are many dependency relations that do not identify any objects in images, for example the phrase *"two dogs barking at each other"* will have an ambiguous relation *"each other"* which does not identify any of the subjects (i.e. the dogs).

Mao et al. [13] proposed a multimodal recurrent neural network (mRNN) which encapsulates a CNN for image processing, an RNN for language modelling and a multimodal layer that connects the CNN and the RNN. Previous multimodal approaches framed the image captioning task as a retrieval problem [11, 52]. These approaches first extract the word, sentence and image features and then, through optimising a ranking cost, learn an embedding model that maps both modalities (image and sentence) into a common semantic feature space. In this way, they can directly calculate the association between images and sentences using for example the inner product of the embedding vectors [11]. This however means that these approaches generate a caption for a given image by retrieving sentences that are most associated with the image and therefore somewhat lack the ability to generate novel sentences or describe images that contain unseen combinations of

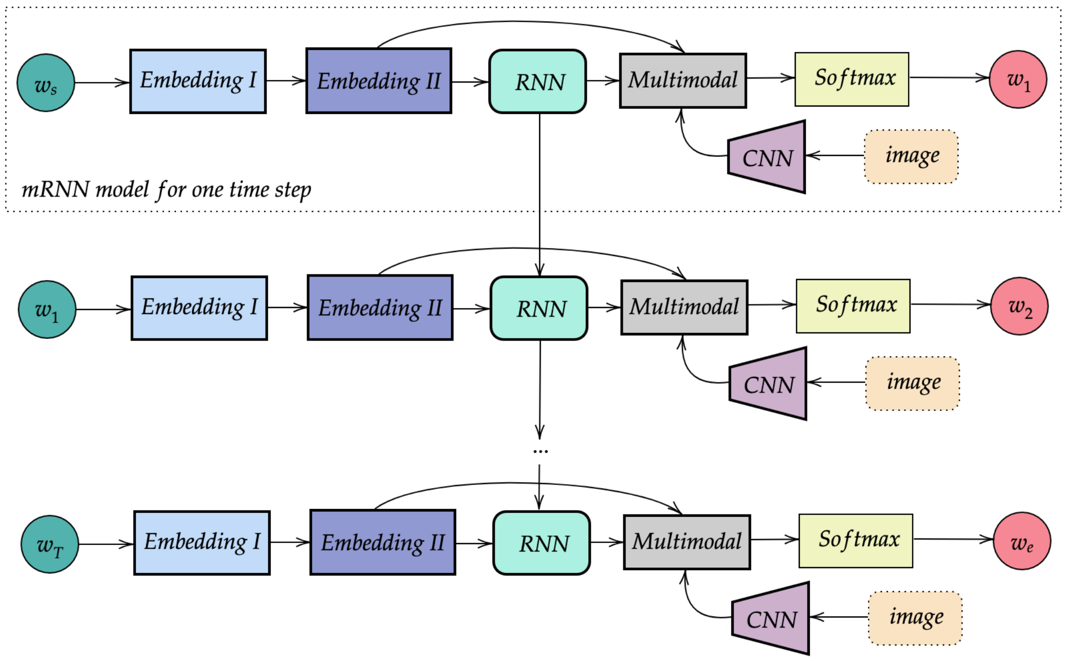


Figure 2: **Multimodal Recurrent Neural Network (mRNN)**. $w_1, w_2, \ldots, w_L$ represent the word sequence, and $w_s, w_e$ are the START and END tokens. Unlike the standard RNN, mRNN is composed of two embedding layers, a custom RNN unit and a multimodal layer that takes in the vectorised word, from the last embedding layer, and the image feature from the CNN. The model then predicts the probability over all the words for the next word in the caption. Image inspired by [13].

objects and scenes.

Without explicitly framing the image captioning problem as a retrieval task, the mRNN approach of Mao et al. [13] is able to not only generate novel sentence descriptions of images but is able to support image and sentence retrieval, illustrated in Figure 2. The approach achieves this by learning a probability density over the common space of multimodal inputs (images and sentences). In this regard it is closely related to the approach of Kiros et al. [46] which uses a probabilistic language model for sentence generation. Where the approach of Kiros et al. [46] is restricted to a fixed context window, this one stores the temporal context within the RNN architecture and thus can generate variable length sentences.

Several works have demonstrated the effectiveness of storing context information within the RNN structure, producing signficantly improved results in the image captioning and retrieval tasks [9, 12, 20, 48]. The mRNN of Mao et al. [13] differs from these approaches in that it employs two word embedding layers that learn word representations more efficiently, and instead of injecting the image feature directly either as the first word or into the recurrent layer, it injects it directly into the common multimodal layer at every time step together with the respective word from the sentence. This approach efficiently utilises the capacity of the recurrent layer, allowing improved image captioning results while using a relatively low-dimensional RNN. At a given time step, the mRNN first ingests the current word vector, and converts it into a dense word representation within the two embedding layers. Notably, instead of initialising the word embedding layers with precomputed vectors as in other image-sentence multimodal models [11, 46, 52, 53], they are randomly initialised and learned from the training data.

The resulting dense word representation is passed into the recurrent layer where the recurrent activation from the previous time step is mapped into the vector space of word representation, and the two are added together with an element-wise operation. The resulting recurrent activation is then passed to the multimodal layer along with the original word representation from the last embedding layer and the image feature is extracted through two CNN variants [5,15] pretrained on ImageNet [51]. These inputs are mapped to the same multimodal vector space and added together to obtain the final activation. The final activation is passed to a softmax layer to generate a probability distribution over all words in the relevant dataset's vocabulary. The mRNN is trained using the log-likehood loss function and the resulting model can support sentence generation, image retrieval using a query sentence and sentence retrieval using an image query. This model produced significant improvements on the IAPR TC-12, Flickr8K, Flickr30K, MSCOCO [2–4, 54] benchmark datasets, according to the BLEU-4 score [34] for sentence generation and the recall rate. The recall rate, R@K, measures the percentage of correct images or sentences in the top $K$ retrieved candidates for an image or sentence query [52, 53].

**Dense Image Captioning**

Dense image captioning reframes the problem as a generalisation of both object detection, where an image region is described by one label, and image captioning where one or more sentences are generated to describe a whole image. Dense image captioning attempts to generate one or more sentence descriptions for multiple sub-regions corresponding to multiple objects in a single image, as illustrated in Figure 3.

Karpathy et al. [12] learned a multimodal embedding space between visual data in the images and semantic data in the text descriptions, which results in model configurations that can generate both full image level captions and region level captions. They directly build upon a previous approach of Karpathy et al. [11], by replacing the dependency tree relations with a bi-directional RNN to compute word representations in the sentence. Unlike dependency tree relations, the bi-directional RNN is more suitable for modelling word sequences and allows for unbounded interactions of words and their context in the sentence. The key insight with this approach is that sentence descriptions can be thought of as weak labels in which contiguous word segments correspond to some particular but unknown location in the image. The solution in this approach has two steps: (i) using

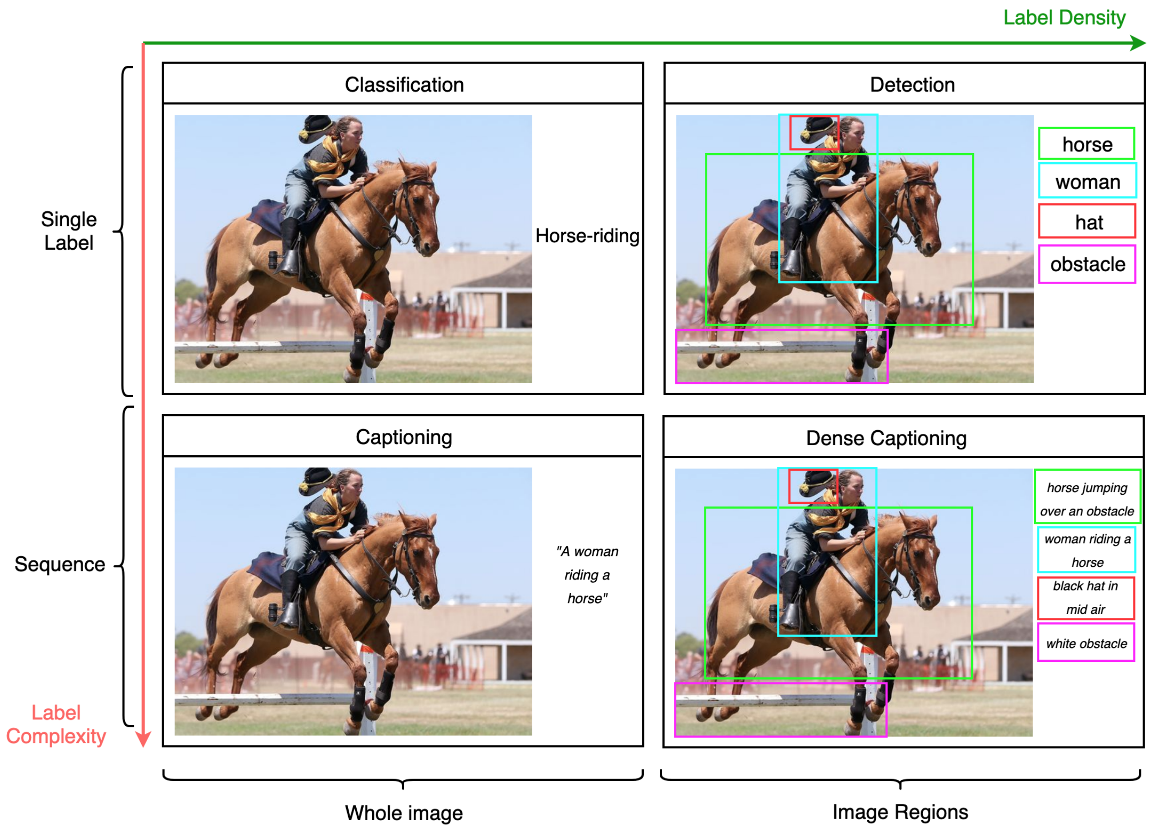


Figure 3: Taxonomy of image description tasks. Dense captioning generalises both image captioning and region-level detection. Image inspired by [55].

images and corresponding descriptions to learn a model that aligns image regions with sentence snippets, and (ii) using the image-text alignments to train a multimodal RNN to generate sentences corresponding to the image regions. The resulting model was evaluated for full image captioning and also on region level captioning. In the former case, the model was trained and evaluated on full image and caption pairs from the MSCOCO dataset [4], while in the later case, evaluation was done on a subset of the MSCOCO test data comprised of image regions and region level captions. The region level model was shown to not only generate sensible full image level captions but also generate fine-grained region level captions that capture the detail of multiple objects which would otherwise not be captured in one sentence. Overall, the approach of Karpathy et al. [12] has a few limitations. Notably, the model can only generate a caption of one input array of region pixels at a fixed resolution. It consists of two models that are trained separately as opposed to an end-to-end solution that can be trained jointly. Additionally, the caption generator uses a bi-directional RNN which is known to have the vanishing or exploding gradient problem, making it not only unable to retain long-term dependencies in sentences but also difficult to train [56, 57].

Inspired by Karpathy et al. [12], Johnson et al. [55] formally introduced the dense image captioning task, unifying the then orthogonal tasks of image captioning with complex sentences as labels, and object detection for identifiying multiple salient image regions. In order to jointly address the localisation and description task, they proposed a fully convolutional localisation network (FCLN) architecture that can process an image with a single efficient forward pass, requiring no external image region proposals and can be trained end-to-end with a single objective function. The architecture is composed of a CNN [5], a novel dense layer for object localisation and an LSTM [49] for generating descriptions. In particular, the dense localisation layer is fully differentiable and is portable to any neural network for region-level training and inference tasks. The CNN receives an image as input and generates an output which encodes the appearance of the image as a set of uniformly sampled image locations. This encoding forms the input into the localisation layer.

The localisation layer identifies spatial regions of interest and smoothly extracts a fixed-sized representation from each region. Similar to Ren et al. [43], the localisation layer predicts a set of variably-sized region proposals which include scalar bounding-box coordinates, region features and confidence scores indicating whether the region contains a desired object. The set of region proposals are then subsampled using an intersection over union (IoU) threshold [7, 42, 43] which scores how well a region intersects with any ground-truth region. In order to interface with the fully-connnected recognition network and the LSTM, the variably-sized region proposals need to be transformed into fixed-sized inputs. The model does this using bilinear interpolation, replacing Ren et al.'s use of a region-of-interest (RoI) pooling operation [43]. Unlike the RoI pooling operation, bilinear interpolation is fully differentiable meaning gradients can be backpropagated from the output features, through the input features all the way to the region proposal coordinates. Features from each region are flattened and passed into the recognition network, each resulting in a 4096-dimensional encoding. These encodings are then passed to the LSTM decoder. Using each region encoding as the first word, the LSTM decoder is conditioned to generate the corresponding caption per region. The FCLN model is trained and evaluated on the Visual Genome dataset [58, 59], a specialised intersection of MSCOCO and YFCC100M [4] where region-level captions were collected through Amazon Mechanical Turk [60]. During evaluation, the model is tested against IoU thresholds for object localisation and the METEOR metric [35] for language modelling. The FCLN model performs well against full image captioning and previous region-captioning baselines [12] in both localisation and captioning tasks. In addition, the model supports image retrieval given a set of random region-level query captions and tasked with retrieving images and localising the objects described in the query captions.

### Encoder-Decoder Representations

Encoder-decoder representations are inspired by successes in neural machine translation [1, 18] where the task is to encode the word representations of a sentence from a

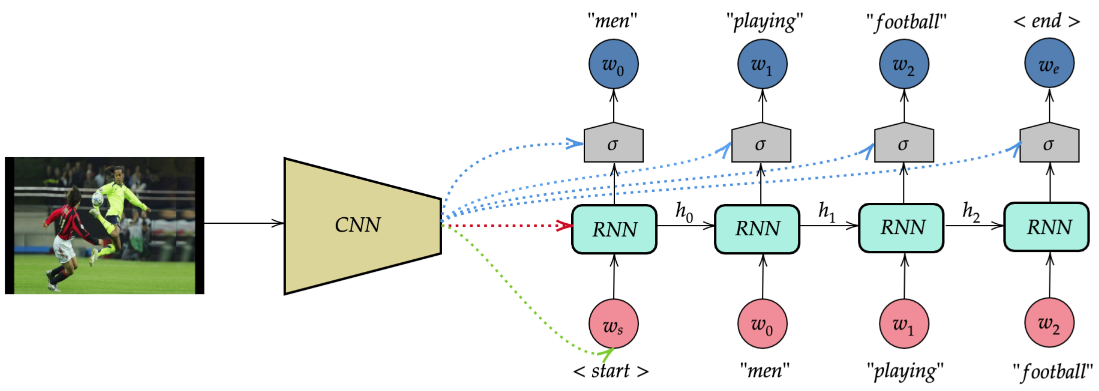


Figure 4: Illustration of encoder-decoder architectures where the CNN image feature is used to condition the text generation. The green path is when the image feature is input as the first word. The red path is when the image feature is used to initialise the hidden state. The blue paths are when the image feature is merged with the vectorised activation emerging from the RNN unit.

source language, and then decode that into a sentence in the target language. In image captioning, the idea is to encode the image representation and, using this representation, decode into a target sentence describing the image. The models usually constitute a CNN for image encoding and an RNN for decoding to a sentence, because of their respective strengths in image feature extraction and sequence modelling. The two components can be pretrained on large image datasets [51] and large sentence corpora [61], and they are then usually jointly trained or finetuned for the image captioning task using image-sentence pairs. An archetypal encode-decoder architecture is illustrated in Figure 4.

Vinyals et al. [9] proposed the neural image caption generator (NIC), the first joint model trainable through stochastic gradient descent [62–64]. The NIC model simultaneously learns the visual and semantic information in order to maximise the likehood of a target caption given an image. The model employs a state-of-the-art CNN [41] that is pre-trained for image classification, to encode images. Unlike Karpathy et al. [12] who use a multimodal RNN for decoding, they use an LSTM architecture [49] for decoding to a natural language caption. The NIC model also represents a direct improvement on the encoder-decoder approach first used by Kiros et al. [48]. While Kiros et al. [48] use a separate encoder and decoder to define a joint embedding space, the NIC model simultaneously defines this space by jointly training the two sub-networks. Additionally, even though the approach of Kiros et al. [48] can generate sentences, it is tuned for multimodal image-sentence ranking. An advantage of the use of two sub-networks that are suitable for processing visual and semantic information, is that they can be pretrained separately on large image datasets and text corpora respectively and enable the advantages of transfer learning. In this approach, the extracted image feature vector is provided as the first word to the LSTM decoder, as opposed to Karpathy et al. [12] who provide the image feature as part of the first hidden state of a multimodal RNN. Additionally, Vinyals et al. [9] empirically verified that injecting the image feature as the first word works better than injecting it at every LSTM time step. This is because when the LSTM attempts to jointly model the image and semantic information, it can learn to memorise the association between image-sentence pairs which can lead to overfitting. The image and the words are mapped to the same embedding space, via the image feature from the CNN and the LSTM's embedding layer which is initialised randomly for simplicity.

During training the objective function is the sum of the negative log-likehood of the correct word at each time step, minimised with respect to all the parameters of the LSTM, the top layer of the CNN and the word embeddings. During inference on a given image, the model generates a sentence by sampling the first word, according to maximum probability, then provides the corresponding embedding as input for the next time step, repeatedly until the end token is sampled. A better performing approach is to use BeamSearch [65] which iteratively considers a set of $k$ sentences up to a time step $t$ as candidates to generate sentences of size $t + 1$. Overall, the NIC model generates high quality sentences that not only correlate highly with human judgement but are diverse and novel, meaning they are not always found in the training set. However, NIC represents the entire image as a single static feature vector which works well for classification, moderately well for detection, but not necessarily so for captioning which usually requires information about the spatial relationships of multiple objects in the image.

Xu et al. [10] proposed an attention based model that improves on NIC by learning to dynamically focus on salient sub-regions of an image as and when they are needed during caption generation. The idea is inspired by research in neuroscience [66, 67] which illustrates the effective use of attention in the human visual system to compress huge amounts of visual information into a natural language sentence. Instead of representing an image as a single static feature vector at the encoding stage, Xu et al. [10] represent the image as multiple features called annotation vectors. The annotation vectors, extracted using a CNN encoder, correspond to certain image regions. In order to obtain correspondence between the 2D portions of the image and the annotation vectors, image features are extracted from an early convolutional layer whose kernels have small receptive fields that cover only sub-regions of the image. Using these annotation vectors allows the model to selectively focus on certain salient features of the image by weighting the corresponding annotation vectors based on the word being generated.

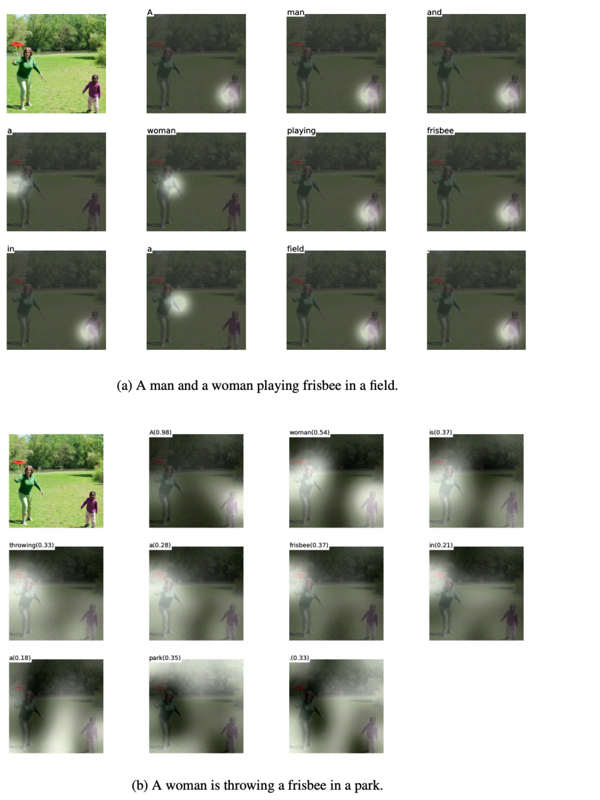
The model extends the soft attention mechanism found by Bahdanau et al. [18] to be effective in neural machine translation. The attention mechanism is formulated in two ways: (i) soft deterministic attention, trainable by stochastic gradient descent [62–64], and (ii) hard stochastic attention, trainable by maximising an approximate variational lower bound or equivalently using the REINFORCE algorithm [68]. In the case of hard stochastic attention, the model generates weights associated with the annotation vectors such that the weight for every region is the probability this it is the correct place to focus on for generating the next word. The formulated objective function is a variational lower bound on the marginal log-likelihood of observing a certain sequence of words given an annotation vector. Maximising this function is equivalent to the REINFORCE learning rule where the reward for the attention resulting in the correct sequence of words is proportional to the log-likelihood of the target sentence. Alternatively for soft deterministic attention, the weights represent the relative importance of a region when blending all the image locations together. The formulated objective function is an expectation on the resulting weighted annotation vector, which is smooth and differentiable, and thus end-to-end training of the entire model can be done with standard backpropagation and stochastic gradient descent [62, 69].

In both attention mechanisms, a multilayer perceptron [69] conditioned on the last hidden state of the decoder is used to generate the attention weights. Intuitively, this means the computed weights at every time step are highly dependent on the previously generated words, i.e. where the model should be looking next depends on the previously generated words. An important attribute of these attention mechanisms is that the attention weights can be visualised at every time step in order to understand where the model was looking when it generated a certain word, which makes for an interpretable model. A visual illustration of the two attention mechanisms is shown in Figure 5.

There is an inherent trade-off between soft and hard attention mechanisms. Soft attention's smooth objective function makes it easier to optimise using stochastic gradient descent but its blended attention maps make it less obvious to distinguish which region the model was looking at when generating a word. Hard attention's objective function is more difficult to estimate but results in sharper attention maps showing exactly what the model was looking at when it generated a particular word. Overall, the attention based approach [10] produced state-of-the-art performance on the BLEU and METEOR evaulation metrics [34, 35] while also providing greater interpretability. Particulary, hard attention performed better than soft attention in generating fine-grained image captions.

## 2.2 Strategies for jointly encoding visual and semantic modalities

Usually when an RNN is used for image caption generation in encoder-decoder architectures, the image features are either directly injected into the RNN as a first word, as part of the initial hidden state, or merged together with the final hidden state of the RNN, right before the word prediction. However there has not been a systematic evaluation of which of these strategies is the most effective way to inject image features and thus condition sentence generation. To address this issue, Tanti et al. [70] conducted an empirical study of the different methods, placing the image captioning architectures into two broad



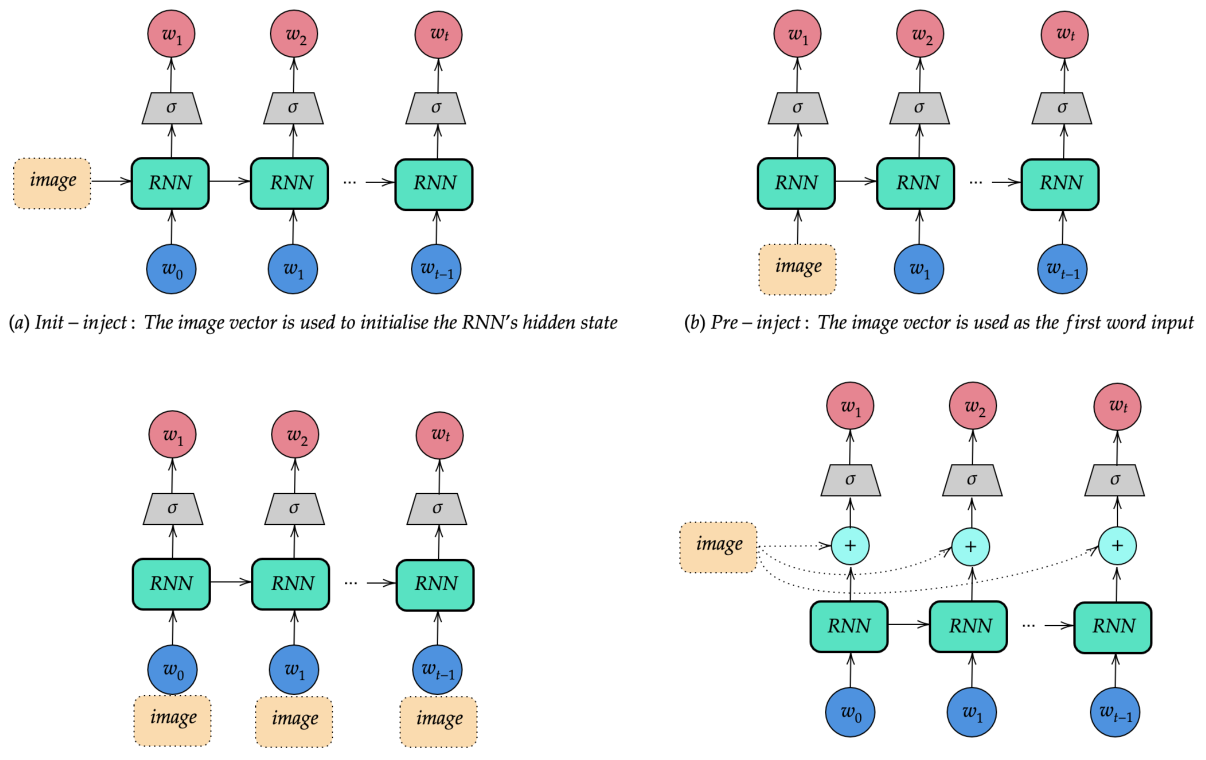
(a) A man and a woman playing frisbee in a field.

(b) A woman is throwing a frisbee in a park.

Figure 5: Visualisation of the difference between *hard* attention (**top**) and *soft* attention (**bottom**). Images by Xu et al. [10].

categories. *Inject architectures* are where the image features are included either early in the generation process or during the whole generation process. In this case, the internal hidden state will modify the image feature at each generation step. *Merge architectures* are when the image feature is only included after the hidden state has been produced. In this case, the same image feature is preserved and used to condition the word prediction at every generation step. From a theoretical perspective, understanding where to introduce the image features during image caption generation has profound implications for insights into how language can be associated with visual information. From a model engineering perspective, the relative performance of the different architectures can provide useful heuristics for selecting an architecture applicable in image captioning and other related but more challenging tasks, including video captioning and paragraph generation. The distinction between inject and merge architectures is further delineated in Figure 6.

- *Init-inject:* The image feature vector is used to initialise the RNN's hidden state. In this case the image vector needs to be the same size as the RNN's hidden state vector and will be modified during generation. Multiple works have employed this architecture in image captioning [71–73] and neural machine translation [74]. Additionaly, Xu et al.'s attention based method [10] uses init-inject to provide the image vector as a representation of the whole image while salient image regions are



(a) *Init – inject : The image vector is used to initialise the RNN's hidden state*

(b) *Pre – inject : The image vector is used as the first word input*

(c) *Par – inject : The image and word are combined as input at every time step*

(d) *Merge : The image vector is merged with the hidden state*

Figure 6: Taxonomy of the inject and merge architectures.

provided via par-inject (described below) at each time step.

- *Pre-inject:* The image feature vector is treated as the first word into the RNN followed subsequently by the word vector inputs [9, 75, 76]. The image vector will be modified during the generation steps and needs to be the same size as the input word vectors. Alternatively, as a way to improve image caption quality, the RNN takes image attributes as the first word or the first two words consisting of the image vector followed by the image attributes [77, 78].

- *Par-inject:* The image and word vectors are either passed as separate inputs or combined into a single input for the RNN, at every step. It is the most commonly used architecture with the largest variety of implementations. Most common is par-injecting the image through all the time steps in the RNN [20] while also init-injecting the image attributes [72, 78]. Other less common implementations involve par-injecting the image but only in the first word [12, 79]. Because the par-inject architecture modifies the image representation, it is commonly used to process differing representations of the same image with every word, so that visual information changes depending on the word being generated. Zhou et al. [80] perform an element-wise multiplication of the image vector and the last generated word's embedding vector in order to attend to different image regions. In general, modified image representations are most effectively used in attention mechanisms for image caption generation [10, 76, 81].

- *Merge:* The image vector is only merged with the output of the hidden state right before word prediction at every step. In this case the RNN's capacity is only utilised for the language modelling and the image feature's representation is preserved [82, 83]. Hendricks et al. [19] use the merge architecture in order to separately train the CNN image encoder and the RNN language decoder using unpaired images and sentences respectively. At every time step, they then combine the two resultant vectors in a multimodal layer which uses an inner-product to predict the next word. This not only makes the two model components independent and portable, but it also allows the model to generate sentences using unseen words as opposed to when using paired image-sentence datasets. Some attention-based architectures utilise merge architectures by merging different image representations at every time step. One implementation is to merge as well as par-inject attended image regions [10, 81], and another is to only merge the regions while par-injecting a fixed image representation [84]. Multimodal representation models that do not use RNNs, but instead use probabilistic language models [47], also utilise merge architectures to merge the image representation and the word presentation [46, 48, 85].

To evaluate the performance of the inject and merge architectures, Tanti et al. [70] measured the quality of captions using four broad criteria.

- *Generation quality* quantifies the degree of overlap and correlation between generated captions and the ground-truth captions. The metrics used were BLEU [34], METEOR [35], ROUGE-L [36, 37] and CIDEr [38].

- *Caption diversity* is based on the entropy of unigram and bigram frequencies in the generated captions. A high entropy yields a more uniform frequency distribution of the unigrams or bigrams, thus indicating the model equally uses all words in the vocabulary leading to diverse captions.

- *Retrieval* quantifies how well a model performs at retrieving the most relevant image given a caption as a query. Relevance is measured as the probability of the whole sequence of words in a caption occurring, given the image. The standard R@K metric [2] was used, which is a percentage of captions such that the image each caption correctly describes is among the $K$ retrieved images.

- *Visual information retention* quantifies how much the image representation influences the caption generation over the entire sequence. This is done by recording the multimodal vector at every time step of the RNN decoder for an image-caption pair. The caption is then paired with a randomly chosen image to create an image-caption mismatch and the resulting multimodal vector is recorded at every time step. The mean absolute difference between the original multimodal vector and the mismatched multimodal vector is measured at every time step. The convergence of this difference over time implies that the vectors are becoming more similar which indicates that the RNN is losing image information and being influenced more by the previously generated words in the caption. By contrast, a constant distance indicates that the different image used in the image-caption mismatch will result in significantly different multimodal vectors than those from the original matching image-caption pair.

For the *generation quality* and *retrieval* criteria, Tanti et al. [70] demonstrated that although the par-inject and init-inject performed best [9, 10, 20, 48, 55], it is not especially detrimental to performance to merge the image features after the recurrent layer of the RNN as in merge architectures. Additionally, merge architectures exhibited comparable performance, while using a considerably lower hidden state dimension for the RNN decoder since they do not have to simultaneously encode visual and semantic information. From a model engineering perspective, this leads to a much smaller and memory efficient RNN whose capacity is fully utilised for language modelling, in addition to exhibiting more stable training dynamics.

Pertaining to *caption diversity*, merge architectures performed best showing higher vocabulary variation and also exhibiting the smallest percentage of generated captions that appear in the training set. An intuitive interpretation of this result is that inject architectures jointly encode the words and image representations in the RNN hidden state, which can lead to a tight coupling of image-caption pairs from the training set. This results in a tendency to reuse captions from the training set. In contrast, merge architectures decouple the image and linguistic modelling, leading to a looser image-caption association that manifests in more diverse captions.

Lastly, for the *visual information retention* criterion, the merge architectures showed the best retention since the image information is not handled or modified by the RNN's hidden state. In contrast, init-inject and par-inject had significantly less retention. This result demonstrated that for generating long captions, and given the finite memory of RNNs, the merge architectures will maximally retain image information and thus generate captions that are most influenced by the image being described.

Overall, our analysis of the various image caption generation implementations can be distilled down to whether visual information is injected into the RNN decoder or merged with the RNN hidden state right before word prediction. From the literature it is evident that the inject architectures are the dominant variation and tend to perform best in corpus based metrics like BLEU and CIDEr [34, 38]. Among these, the encoder-decoder sub-category has proved to be the most popular and performant way to translate visual information to text, and we therefore adopt this as our foundational architecture for video caption generation. Merge architectures however are interesting as they have been shown to perform better for caption diversity, have a significantly smaller RNN memory footprint and have better image information retention [70]. Crucially, they decouple the visual and language modelling meaning they tend to generate less generic and stereotypical captions by exploiting the multimodal information more effectively. Therefore our model is closely related to merge architectures, because as explained later, we modify the encoder-decoder architecture to instead merge the video information with the RNN output right before word prediction.

## 2.3 Video Captioning

In light of the discussion on architectures used in image captioning, CNNs and RNNs have become the de facto solution for many visual feature extraction and language modelling problems respectively. By extension, leveraging the strengths of these two models has been identified as an effective way to tackle the video captioning problem. Traditional video captioning approaches like the *subject-verb-object* [86–88] first perform visual recognition with handcrafted features and then generate lexical tokens that form the caption. Deep learning methods automate the process of feature extraction and caption generation by using CNNs to learn video features to be used for influencing the prediction of a sentence description in the RNN. Therefore, the encoder-decoder architecture is widely adopted as a solution for video caption generation. The crucial difference however is that the complex and temporal nature of videos makes video description a more challenging problem, in addition to the fact that a typical video has a larger memory and storage footprint than its image counterpart.

Donahue et al. [20] proposed the first deep learning solution to video captioning in the form of a long-term recurrent convolutional network in three configurations, all of which take in visual predictions of the video from a conditional random field (CRF) [89]. This allows the model to observe the entire video at every time step, instead of incrementally frame by frame. The first of these configurations uses a two-layer LSTM encoder-decoder with CRF max predictions inspired by statistical machine translation (SMT) based video captioning [90]. The first LSTM layer encodes a one-hot vector of the input sentence, allowing for variable length inputs, and the final hidden representation of the first layer is fed into the second layer which decodes the input into a sentence one word at each time step. The second configuration of the model uses an LSTM decoder with CRF max predictions and encodes the semantic representation as a single fixed length vector for input to the LSTM at each time step, effectively collapsing the problem into an image captioning task. The third and final configuration uses an LSTM decoder with CRF probabilities, since LSTMs can naturally incorporate probability vectors during training and testing time allowing them to learn uncertainties in caption generation. Although all these configurations outperformed the SMT approach, they are not end-to-end trainable.

Venugopalan et al. [23] proposed the first end-to-end trainable deep network. The model simultaneously learns a latent semantic state and a fluent grammatical model of the associated language. An LSTM is used to model sequence dynamics and, to avoid intermediate representations as in Donahue et al. [20], the LSTM is directly connected to a deep CNN which processes input video frames. Additionally, unlike Donahue et al. [20] who showed results on a limited cooking dataset [91], Venugopalan et al. [23] showed results on YouTube Clips [27] which is an open-domain video dataset. In order to effectively summarise an input video representation, one in every ten frames is sampled, and processed by a CNN resulting in a feature vector. A mean pooling operation is then performed on a collection of these feature vectors, resulting in a single fixed length vector which is passed to a two-layer LSTM. Aggregating multiple frames into a single vector representation reduces the problem into an image captioning task and works better than previous approaches. However, the indiscriminate averaging of frames results in loss of valuable temporal information of the video such as the order in which objects appear and interact, which adversely affects the quality of the captions for long form videos and makes this approach only suitable for short clips with a single major event.

Open-domain videos comprise of complex interactions among actors and actions, therefore aggregated features might not adequately capture such temporal dynamics. Yao et al. [14] proposed a spatiotemporal 3D CNN based encoder, trained on the UCF-101 activity recognition dataset [92] and using an LSTM as a decoder, that is able to capture global temporal structure and fine-grained motion information between consecutive frames. The local motion is summarised and preserved by dividing the video into groups of 16 frames which are each encoded into a 3D spatiotemporal cuboid represented by concatenating its histogram of oriented gradients, oriented flow, and motion boundary [93, 94]. Global temporal structure is preserved by employing a temporal attention mechanism adapted from soft attention, first devised in neural machine translation work by Bahdanau et al. [18]. Crucially, using the attention mechanism, the decoder assigns a relevance weight to every cuboid at every caption generation time step which steers the model towards the most relevant video segment and results in fine-grained captions that can capture complex dynamics in videos with more than one major event. Similarly, Pu et al. [21] proposed a spatiotemporal 3D CNN and LSTM based model which uses a temporal attention mechanism to selectively and sequentially focus on different convolutional layers, taking advantage of the fact that low level layers capture fine-grain details while higher level layers have a larger receptive field and therefore capture the global temporal structure of the video.

However, all the discussed approaches rely on fixed video representations which can be limited in modelling various temporal dynamics. Models that rely on variable visual representations can directly map variable length inputs (video frames) into variable length outputs (words or sentences). Venugopalan et al. [24] proposed an LSTM based architecture that addresses the variable length problem for both the encoding and decoding stages. The encoder LSTM processes the sequence of video frames and the decoder LSTM uses the last hidden state of the encoder to sequentially decode the caption one word at a time. This approach represents the first time that sequence-to-sequence modelling was adapted from neural machine translation for the video captioning task.

Recently more attention is being put towards video dense captioning or video paragraph generation. The previously discussed approaches focus on summarising the content of a video in one or two sentences, making them mostly suitable for short videos with one major event. These approaches result in oversimplified captions for long videos with multiple, sometimes overlapping, events. Yu et al. [95] proposed a hierarchical RNN model that can exploit both spatial and temporal attention. The model's decoder is comprised of a gated recurrent unit (GRU) [96] for generating short sentences associated with an event and a higher-level recurrent layer responsible for combining the sentence vectors from the GRU. This higher-level recurrent layer can thus capture inter-sentence dependencies resulting in a sequence of relevant consecutive sentences. Closely related is work by Krishna et al. [22] which proposes a new dense captioning task using an event proposal technique that predicts the start and end of an event. The proposed model can then simultaneously detect multiple events in a single pass and attempt to describe them. This work represents the first attempt to detect and caption multiple and overlapping events in a video using the ActivityNet Captions dataset [97, 98].

As noted earlier, various forms of RNNs including vanilla RNNs, LSTMs and GRUs have been extensively utilised for the video captioning task because of their effectiveness in sequence modelling. However, the recurrent units make them particularly difficult to train because of the vanishing or exploding gradient problem and finite memory resources [56, 57]. Moreover, in order to predict the next word, a typical RNN only has available to it the current hidden state which is a cumulative result of all previous states and the input word from the current hidden time step. For long sequences and due to memory constraints, this results in short term memory which is not ideal for modelling long term semantic dependencies. Vaswani et al. [99] proposed the Transformer model, which is solely based on attention mechanisms that completely replace recurrent units in a decoder. Unlike recurrent models that learn a conditional probability of the occurence of words in a sequence, the Transformer model uses self-attention to learn a sequence representation such that every word's probability of occurring is directly influenced by all preceeding words through weighted relevance. Naturally, the Transformer model presents a compelling alternative to the recurrent based models for the video captioning task. Zhou et al. [100] proposed an adapted Transformer model for the video captioning task. The model extends on the work of Krishna et al. [22], consisting of a video sequence encoder, a proposal decoder that converts the video representation into a set of different anchors to form video event proposals, and a captioning decoder that decodes the event proposals into captions. Crucially, unlike the work of Krishna et al. [22] where the event proposal and captioning are separate, Zhou et al. [100] connect the two components with a network that produces a differentiable mask, allowing the caption generation to directly influence the event proposal for better quality captions.

## 2.4 Summary

In this literature review we discussed the various approaches that have been employed in tackling the closely related problems of image and video caption generation. Their close relationship stems from the fact that both problems can be broadly thought of as visual-semantic processing with the goal to ground natural language descriptions in visual information. A common theme in the literature is to establish a multimodal space that associates the visual features with semantic information in order for the former to influence the generation of the latter. Due to the success of CNNs for visual feature extraction and RNNs for language modelling, encoder-decoder architectures have emerged as a prominent solution for both image and video caption generation. In these architectures, and inspired by work in neural machine translation, the CNN and RNN work in unison to translate an image or video into a natural language description. Crucially, during training, the language model can share gradients with the CNN, thereby optimising the CNN's feature extraction in order to maximise the likelihood of generating the correct caption. We therefore use the encoder-decoder architecture as the foundation of our model for video caption generation.

Most CNN implementations operate on a collection of video frames with kernels that collapse the spatiotemporal information, leading to loss of the video's temporal stucture. This effectively collapses the video captioning problem into an image captioning problem. However the temporal structure of videos has been shown to be important for accurate classification and fine-grained description [14, 21, 26]. Tran et al. [25] proposed a 3D CNN which operates on the video spatially and temporally, preserving a video's temporal dynamics. Attention based approaches have been proposed where the CNN, using spatiotemporal features, can dynamically focus on different sections of the video depending on the word that is about to be generated [14, 21]. We adopt such spatiotemporal video features when processing videos for input into our video caption generator.

In establishing a multimodal space, where visual features and words can be associated, the literature has a diversity of approaches on how to incorporate visual features in order to condition the caption generation. Particulary in image captioning, directly injecting the image feature as an initial hidden state of the RNN (*init-inject*) or combining it with the word at every generation step (*par-inject*), has emerged as the most performant and popular approach [70]. Tanti et al. [70] carried out a systematic evaluation of the different image feature injection configurations and found that merging the image feature with the output of the RNN, right before word prediction, has comparable performance to inject approaches. Merging the image feature and the output of the RNN, and not injecting it in the RNN, preserves the image features and ensures that they maximally influence the word prediction. Since the RNN is not responsible for jointly modelling visual and semantic information, its capacity can be effectively used for language modelling, resulting in a significantly lower memory footprint which is critical for model engineering. Therefore, in addition to spatiotemporal video features, we use the merge approach where the video feature is merged with the output of the RNN unit, right before word prediction. These ideas form the foundation of our study.

# 3 Technical Background

At the heart of the image and video caption generation tasks is the fundamental problem of multimodal learning, where a model has to learn to associate visual and semantic information. Deep learning models have showcased great success in visual-semantic association, particularly due to advances in convolutional neural networks (CNNs) and recurrent neural networks (RNNs) [15,40,49]. These two models have been successful primarily because of the following factors: (i) the availability of large quantities of annotated visual and semantic data upon which the models' data driven parameter optimisation can outperform traditional strategies, (ii) increasingly powerful compute resources in the form of graphics processing units (GPUs) [101] and application specific integrated circuits (ASIC) hardware [102], and (iii) increasingly complex and expressive models that can learn hierarchical features. Encoder-decoder architectures that combine CNNs and RNNs, leverage the two models' respective strengths in order to translate visual information into natural language descriptions. While a CNN encodes visual information into a vector representation, an RNN leverages the representation and word embeddings to generate the description. In this chapter we focus on the mathematical theory that underlies neural networks, how they learn from data and then, particularly, how CNNs and RNNs extract salient features from visual and semantic data, respectively. We then discuss how word embeddings, that inscribe semantic meaning in vectorised form, are utilised in encoder-decoder architectures to generate sensible descriptions for visual information.

## 3.1 Artificial Neural Networks

Artificial neural networks (ANNs) belong to a computing paradigm inspired by, and loosely modelled on biological neural networks. Unlike programmable computing systems, ANNs learn to perform a specific task by observing example inputs and desired outcomes. Without any prior knowledge about the task, ANNs can in principle extract salient characteristics from examples and learn to use these to output the desired outcome on new examples. In our case, a specialised ANN should learn to process a video signal and translate it into a natural language description. An ANN is comprised of artificial neurons or nodes, connected together by artificial synapses that constitute adjustable weights to modulate the strength of the input signal. Typically, artificial neurons are grouped into layers, and the input signal travels from the first layer to the last layer with multiple signal transformations performed at every layer. Figure 7 is an illustration of a simple ANN with three layers.

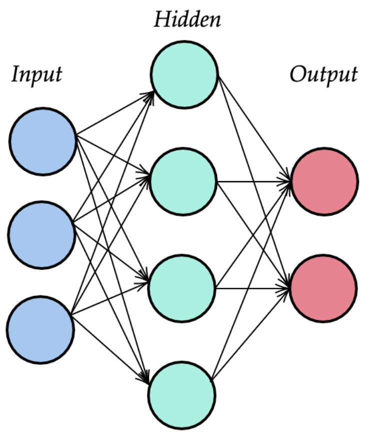


Figure 7: Illustration of a 3-layer artificial neural network. The nodes are interconnected with artificial weighted connections.

### 3.1.1 Artificial Neural Network Operations

In order to produce desired outcomes, e.g. a natural language description, an ANN's layers will typically have to transform a high-dimensional input signal like an image or a video into lower-dimensional characteristics, called features, that can be used to generate the output. Typically, a matrix multiplication is performed between the signal and the connection weights at a particular layer, and the result is passed on to the next layer of neurons until the final output is realised. Figure 8 is an illustration of this operation.

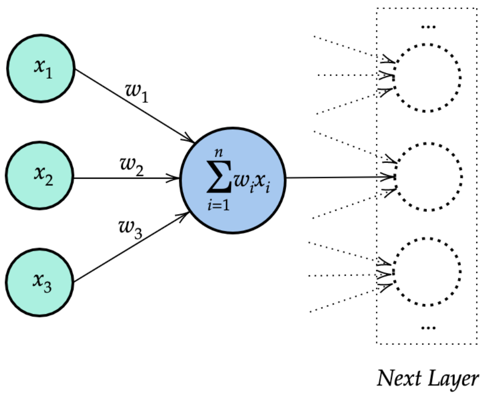


Figure 8: Illustration of the linear algebraic operation between a neuron's weighted connections and its inputs.

It is common to append the input signal with an additional value of 1, for an additional bias term in the summation at every neuron, to give the network greater expressibility. The combined operation at all the neurons of a particular layer can be expressed as a matrix multiplication. Given this matrix formulation, batched mathematical expressions involving multi-dimensional arrays can be evaluated in parallel using efficient vectorised

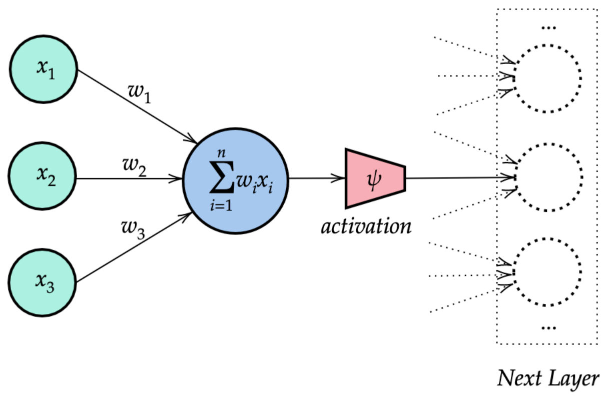


Figure 9: Illustration of the linear transformation followed by a nonlinear activation.

operations. This forms the basis of deep learning frameworks like TensorFlow, PyTorch and Theano [103–105] supported by *single instruction, multiple data* (SIMD) hardware, e.g. GPUs and TPUs [101, 102]. Consequently, using a composition of these operations across all layers, the ANN can be generally represented as a sequence of linear transformations.

### 3.1.2 Limitations of Linear Operations

The aim of a neural network is to approximate some unknown function between given inputs and outputs, and most datasets like images and text cannot be modelled sufficiently well with a linear neural network. In order for ANNs to be more expressive and be capable of modelling nonlinearities, an additional nonlinear operation can be performed on the result of the linear operation at a neuron, just before it is passed to the next layer of neurons. These operations are referred to as activation functions. Figure 9 shows the linear transformation, now modified to include an activation function. The activation can work as a threshold on the signal depending on the magnitude of the weighted sum in the preceding matrix operation. Intuitively, it is loosely analogous to the synaptic activation that occurs in biological neural networks, where an incident electrical signal activates the next neuron depending on the signal's intensity [106]. In artificial neural networks this means neurons can be specialised to activate when stimulated by specific features of the signal.

### 3.1.3 Activation Functions

Activation functions take a real number and perform a fixed mathematical operation on it. Crucially, activation functions also need to be at least partially differentiable for the gradient based learning dynamics of neural networks, as we discuss later on. In this section we discuss commonly used activation functions, their motivations and their

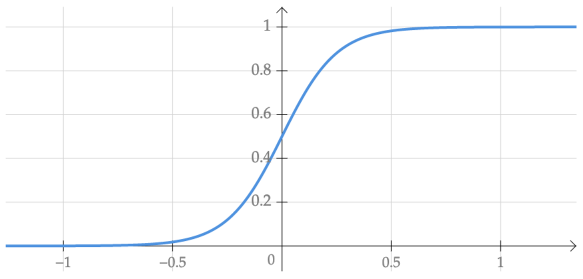


Figure 10: Illustration of the sigmoid function.

respective limitations.

### Sigmoid

The sigmoid function takes a real-valued number and returns a number in the range $[0, 1]$. The function is illustrated in Figure 10, and takes the mathematical form

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{1}$$

Note that large negative numbers result in outputs close to 0 and large positive numbers to outputs close to 1. The function has seen frequent use due to its intuitive interpretation as the activation rate of a neuron, ranging from not activating at all at 0 to fully activated at 1. While historically a popular choice in practice, the sigmoid function has the following drawbacks.

- *Saturated neurons kill gradients.* When a neuron's activation saturates at either 0 or 1, the gradient is almost zero. Because the network learns by recursively multiplying and propagating back these gradients to update the network weights, a small to zero gradient effectively kills the gradient at that neuron, preventing the flow of the gradient signal to preceding neurons. When using the sigmoid, weight initialisation is particularly important because very large positive or negative weights will result in immediate saturation of neurons and the network will not learn.

- *Sigmoid outputs are not zero-centered.* The sigmoid will always propagate positive activations. This means that depending on the sign of the incoming gradient signal from all the subsquent layers up to the sigmoid, the gradient into the weights preceding the sigmoid will either all be negative or all be positive and this can prevent the network from converging towards optimal weights.

- *Expensive to evaluate.* From a model engineering perspective, the exponent term in the sigmoid is computationally expensive to calculate and even more so for very large neural networks with hundreds of millions of parameters.

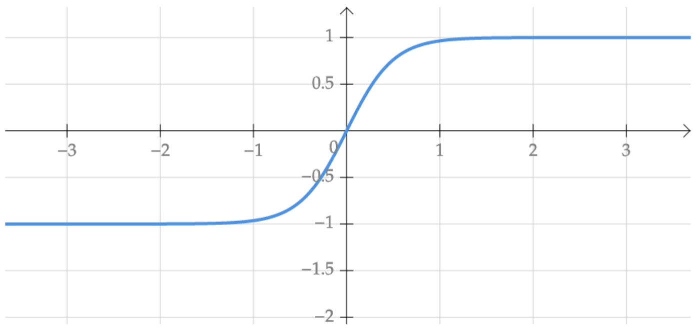


Figure 11: Illustration of the tanh function.

### Tanh (Hyperbolic Tangent)

The tanh function takes a real-valued number and outputs a number in the range $[-1, 1]$. The function is illustrated in Figure 11, and takes the mathematical form

$$\tanh(z) = \frac{e^{2z} - 1}{e^{2z} + 1}. \tag{2}$$

Analogous to the sigmoid function, tanh has a constrained range meaning large negative inputs will be close to $-1$ and large positive inputs will be close to 1. It also shares the advantage of the sigmoid of being interpretable as an activation rate. In fact, the tanh is simply a scaled sigmoid:

$$\tanh(z) = 2\sigma(2z) - 1. \tag{3}$$

The formulation in Equation 3 ensures that it has zero-centered outputs which results in symmetrical weight updates. For this reason the tanh is usually preferred over the sigmoid for its better training dynamics. However it also shares in the drawback that activations close to $-1$ or 1 can kill the gradient signal. Additionally the exponent terms are computationally expensive to evaluate.

### ReLU (Rectified Linear Unit)

The ReLU function uses a threshold in which real values less than 0 are set to 0, otherwise the output is the identity of the input. The function is illustrated in Figure 12, and takes the mathematical form

$$f(z) = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}. \tag{4}$$

The function has gained popularity because of its speed of convergence towards optimal

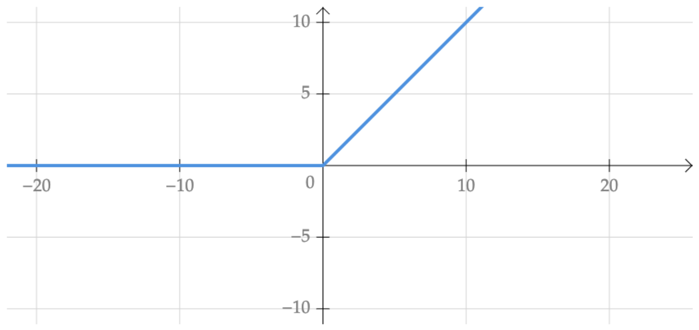


Figure 12: Illustration of the ReLU function.

weights. Its effective use by Krizhevsky et al. [15] showcased significant accuracy improvements in the ImageNet classification task [51, 107]. The function has several advantages over the sigmoid and tanh functions.

- *Avoids neuron saturation.* The ReLU function has a linear output for inputs greater than zero, ensuring no saturation of any arbitrarily strong signal. This also means constant gradient of 1 for values greater than 0, which allows for gradients to flow rapidly through the network to all the preceding weights, resulting in accelerated convergence.
- *Computationally inexpensive.* The ReLU function is computationally simple to implement with mathematical operations that simply change negative signals to 0.

The function has one major drawback, which is that neurons with 0 or negative outputs will effectively become inactive. This is an irreversible state because, in gradient-based optimisation, only neurons that contribute to the incident signal will have a non-zero gradient and hence will be updated and thus continue to contribute to the signal. Therefore the inactive neurons become permanently unactivated which ultimately reduces the learning capacity of the neural network. Careful weight initialisation and choosing an appropriate learning rate for updating the weights, are therefore critical.

### Leaky ReLU, Parametric ReLU and ELU

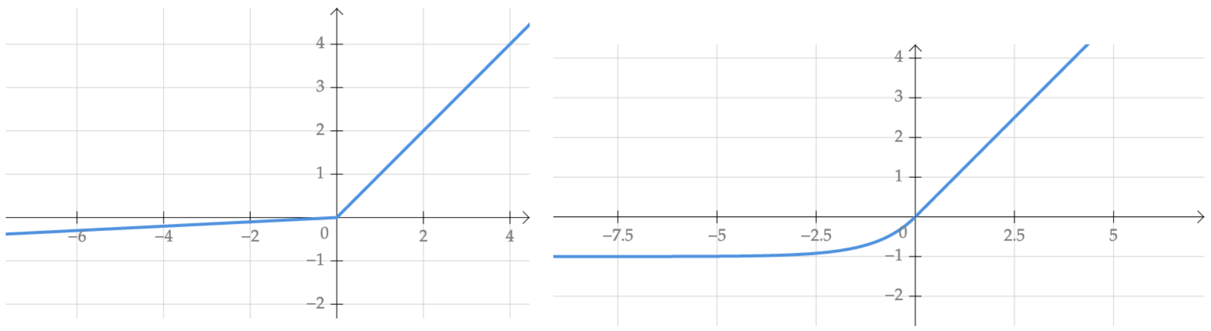


Figure 13: Illustration of the Leaky ReLU (**left**) and ELU (**right**) functions.

Leaky ReLU, Parametric ReLU and ELU (Exponential Linear Unit) are variants of ReLU that try to address the problem of inactive neurons. Leaky ReLU (left in Figure 13) allows a small gradient for values less than 0, which allows potentially inactive neurons to recover during training and be able to contribute to the incident signal [108]. Parametric ReLU extends this idea and parameterises the coefficient of leakage, such that it can be learned with other neural network parameters [109]. ELU (right in Figure 13) is an identity for values greater than 0, just like the ReLU, but smoothly tends to a small gradient for values less than 0 [110]. Leaky ReLU, Parametric ReLU and ELU are mathematically formulated as follows:

$$f(z) = \begin{cases} z & \text{if } z > 0 \\ 0.01z & \text{if } z \leq 0 \end{cases} \tag{5}$$

$$g(z) = \begin{cases} x & \text{if } z > 0 \\ \alpha z & \text{if } z \leq 0 \end{cases} \tag{6}$$

$$h(z) = \begin{cases} z & \text{if } z > 0 \\ a(e^{z} - 1) & \text{if } z \leq 0 \end{cases} \tag{7}$$

Overall, activation functions have become an integral part of deep neural networks in tasks like computer vision, language modelling and speech recognition, where linear models are insufficient to model the data. In the context of our work in automatic video captioning, we choose to use the standard ReLU activation function for both video feature extraction and natural language generation, because it is computationally cheap and, with careful weight initialisation, allows for acceptable network convergence.

### 3.1.4 Prediction Loss

In the previous section we discussed how artificial neural networks composed of simple linear transformations, can be extended with nonlinearities to increase their expressive capacity. In essence, these neural networks perform a sequence of linear tranformations with interwoven nonlinearities in order map input data to the desired output. We want to measure and adjust the influence of the neural network's weights so that they transform the input into the correct output. In order to achieve this, we need to include a special output layer that will convert output from the network's last hidden layer to scores. Together, the linear transformations, nonlinearities and output layer form a score function that outputs a set of scores associated with all possible outputs, called class labels, where the objective is to have the highest score associated with the correct class label.

In order to quantify and optimise how consistent the class scores are with the given class labels in the training data, an additional function is needed. This is called an objective, cost or more commonly a loss function and it quantifies how much the neural network's prediction deviates from the correct prediction. The loss function is usually constituted of (i) a data loss based on the training data and (ii) regularisation loss, to control the neural network's complexity. Optimisation in the context of neural networks is the process of minimising this loss. In this section we discuss various common loss functions and highlight their different characteristics. In general, any loss function can be expressed as a function of the weights of the neural network, the input data and the predicted score. For example, given a data point $x_i$, the loss $L_i$ is a function of the final output of the neural network $\phi$ and the corresponding class label $y_i$. The neural network is itself a function of weights $W^l$ and nonlinearities $\psi^l$. $l \in \{1, \ldots, k\}$ represents the $l$-th layer of the neural network. Given a training dataset of $N$ examples, the overall loss can be expressed as the numerical average of all the individual losses calculated for every input data point:

$$Loss = \frac{1}{N}\left(\sum_{i=1}^{N} L_i\right). \tag{8}$$

**Multiclass Support Vector Machine Loss / Hinge Loss**

This loss formulation is usually used and associated with the support vector machine, a type of classification model that transforms the final feature activations of a neural network into predictions, in the form of classification scores [111]. The formulation is such that the score for the correct class label should be higher than all the other incorrect classes by at least some fixed margin. We illustrate this in the following equations:

$$S = \phi(W^1 \ldots W^k, \psi^1 \ldots \psi^k, x_i), \tag{9}$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \delta). \tag{10}$$

Given a set of class scores $S$ predicted for an example $x_i$, the loss $L_i$ is the summation over the differences between the every individual class score $s_j$ and the score of the correct class label $s_{y_i}$. In order to minimise the loss, the score $s_{y_i}$ must be large enough such that the difference $s_j - s_{y_i}$ is so small that adding a margin $\delta$ results in a value still less than or equal to 0. The result $s_j - s_{y_i} + \delta$ is clamped to 0 by the max operation. The multiclass SVM loss is sometimes referred to as the *hinge loss*. The *squared hinge loss* is a variant of the hinge loss which instead penalises the violation of the $\delta$ margin quadratically, as follows:

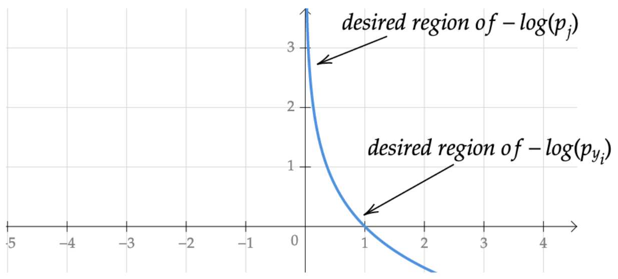


Figure 14: Illustration of the negative log-likelihood objective of a softmax loss. Minimising the loss implies that $-log(p_{y_i})$ for the correct class approaches 0 and $-log(p_j)$ for an incorrect class is asymptotically large.

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \delta)^2. \tag{11}$$

## Softmax Loss

The softmax loss is associated with the softmax classifier, which is another commonly used classification model [112]. Unlike the previously described hinge loss which treats the outputs of the neural network as uncalibrated and possibly difficult to intepret class scores, the softmax classifier takes real-valued outputs and normalises them to values in the range $[0, 1]$, such that they sum to 1. Intuitively, these values can then be interpreted as class probabilities. Given a set of uncalibrated class scores $S$, the softmax classifier is as follows:

$$\mu = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \tag{12}$$

In Equation 12, $s_{y_i}$ is the class score of the correct class. The softmax classifier is set up such that the correct class score is as high as possible (close to 1) and the other incorrect classes are as low as possible (close to 0). This is in contrast to the SVM which only requires the correct class score to be high by a margin $\delta$. Using the softmax function $\mu$, the *softmax loss* is then formulated as follows:

$$L_i = -log(\mu). \tag{13}$$

With the scores converted to class probabilities, the objective of the softmax loss function is to maximise the log-likelihood of the correct class. Since we want to minimise the loss, the softmax loss function is defined as the negative log-likelihood of the correct class. In Figure 14, we illustrate desired loss regions for the correct and incorrect class labels, respectively.

Equation 13 is the softmax loss for one example $x_i$. The full loss across all training examples is a numerical average, as in Equation 8. The softmax loss is also commonly known as the *categorical cross-entropy loss*, which is a combination of the softmax function and cross-entropy loss, motivated by an information theoretic view [112]. In information theory, the cross-entropy between the true distribution $p$ and an estimated distribution $q$, both over a given set (in this case a set of $C$ classes), quantifies the different between these two distributions, and is defined as follows:

$$H(p, q) = -\sum_{i=1}^{C} p(s_{y_i}) \cdot log[q(s_{y_i})]. \tag{14}$$

Using this formulation and given an input example $x_i$, the objective is to minimise the cross-entropy between the estimated class probabilities $q$ resulting from the softmax function in Equation 15, and the true distribution $p$ which can be interpreted as the distribution where all the probability mass is assigned to the correct class label $y_i$ for input $x_i$. In other words, the cross-entropy wants the predicted distribution to have all the probability mass assigned to the correct class label. Combining the softmax function and the cross-entropy implies substituting the former into Equation 14, which leads to

$$H(p, q) = -\sum_{i} p(s_{y_i}) \cdot log\left[\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right]. \tag{15}$$

In the specific case of multi-class classification, the term $p(s_{y_i})$ is 1 only for the correct label and 0 for all the other labels in the “true” distribution $p$. Therefore the summation in Equation 18 collapses to only the term when $p(s_{y_i}) = 1$, resulting in the softmax loss.

The hinge loss and the softmax loss are the two most commonly used loss functions for classification. The hinge loss penalises weights when the score of the correct class is not higher than all the scores of the incorrect classes, by a defined margin $\delta$. Once the margin is satisified all incorrect class scores are simply clampled to 0 by the max operation which makes hinge loss a bounded optimisation function. However it produces class scores that are uncalibrated and difficult to intepret because the scores can be wildly different for different weight initialisations. The softmax loss solves this problem by normalising the scores to a range $[0, 1]$ which makes them intuitively interpretable as confidence scores or probabilities over the classes. In the context of our research in video caption generation we choose to use the softmax loss in order to easily and intuitively interpret class probabilities over a vocabulary of words when predicting the next word in a natural language sentence. Every class in this case is simply a set with one unique word from the chosen vocabulary.

The softmax loss, however, does not have a margin requirement for how far apart the correct class probability needs to be from all the wrong classes which makes it an unbounded optimisation function. This unbounded nature can contribute to overfitting, a scenario where the neural network learns very specific features in the training dataset, preventing it from generalising for new examples. In general, the overfitting problem is alleviated by augmenting the full loss function with a class of functions called regularisers, that we discuss in the next section.

### Regularisation

Regularisation plays an important role in controlling a neural network's capacity in order to prevent overfitting. The primary motivation for this set of techniques is to regulate accuracy on the training data in exchange for a network that will better generalise for new examples. Additionally, from a model engineering perspective, certain weights are more preferrable than others. For a memory efficient model, and for robustness, regularisation functions would encode a preference for small weights over large weights. Such a preference would be encoded and adjusted as part of the optimisation process, generally represented as an additional term in the full loss function as follows:

$$Loss = \underbrace{\frac{1}{N}\left(\sum_{i=1}^{N} L_i\right)}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularisation loss}} \tag{16}$$

where $\lambda$ is a hyperparameter which in this case is used to modulate the strength of the regularisation. The data loss is a function of the weights and data, while the regularisation loss is only a function of the weights. We briefly discuss some commonly used regularisation functions and highlight their different characteristics.

### L2 Regularisation

The L2 regularisation is a popular formulation, and it discourages large weights through an elementwise quadratic penalty [113]. The formulation is as follows:

$$R(W) = \sum_{j}\sum_{k} W_{j,k}^2 \tag{17}$$

where $j$ is the row index and $k$ is the column index in $W$. A compelling example of the effectiveness of L2 regularisation is that penalising large weights tends to improve generalisation. This is because it discourages any arbitrarily large weight dimension from having a disproportionately large influence on the final class scores. For example given an input vector $x = [1, 1, 1, 1]$ and two candidate weigths $w_1 = [1, 0, 0, 0]$ and $w_2 = [0.25, 0.25, 0.25, 0.25]$. In both cases the dot product is 1. However the L2 penalty for $w_1$ is 1, while for $w_2$ is 0.25, so to minimise the overall loss the neural network will prefer to use

$w_2$. Intuitively, by using $w_2$ the neural network is encouraged to take into account diffuse signals from all input dimensions instead of strong signals from a few input dimensions. Notably from Equation 20, in order to minimise the overall loss, the regularisation loss will want the weights to all be 0. However, because the weights have to be non-zero in order for the neural network to learn, the data loss term and the regularisation term will pull the neural network between the two extremes of large capacity and small weights until an equillibrium is reached.

### L1 Regularisation

L1 regularisation discourages large weights through an element-wise absolute value penalty [113]. The formulation is

$$R(W) = \sum_j \sum_k |W|_{j,k} \tag{18}$$

where $j$ is the row index and $k$ is the column index in $W$. An intriguing property of L1 regularisation is that it tends to lead to sparse weight vectors during optimisation. Intuitively, neurons with L1 regularisation end up using a sparse subset of their most important inputs and thus become nearly invariant to “noisy” inputs. In contrast, neurons with L2 regularisation have small diffuse weights which try to use all their inputs. In practice this means L2 regularisation is more suitable if explicit feature selection is needed, whereas L1 regularisation would be a suitable guard against noisy signals if no explicit feature selection is performed.

### Elastic Net Regularisation

Elastic Net Regularisation additively combines L1 and L2 regularisation, while modulating the strength of the L2 term [114]. The formulation is

$$R(W) = \sum_j \sum_k \beta W_{j,k}^2 + |W_{j,k}|. \tag{19}$$

The hyperparameter $\beta$ is trainable and can be adjusted during model validation.

### Max Norm Regularisation

The max norm penalises large weights by enforcing an absolute upper bound on the Euclidean norm of the weight vector of every neural connection [115]. Given a weight vector $W = (w_1, w_2, \ldots, w_n)$, the following constraint is placed on the Euclidean norm:

$$||W||_2 = \sqrt{w_1^2 + w_2^2 + \cdots + w_n^2} < c. \tag{20}$$

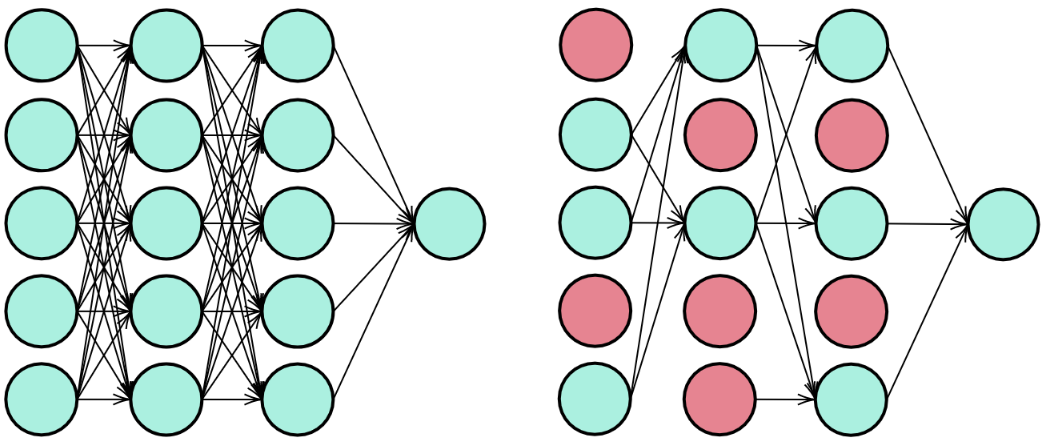


Figure 15: Illustration of dropout. Left is a standard neural network with dense connectivity. On the right is a neural network with a random set of neurons and connections set to 0.

In this expression, the Euclidean norm is bound to be less than a constraint $c$. During optimisation, the weights are adjusted normally, but the constraint is satisfied by clamping the weight vector. A desirable property of this regularisation is that the neural network weights have an absolute upper bound even when an aggressive coefficient of weight update (learning rate) is applied.

**Dropout Regularisation**

Dropout is a simple but effective form of regularisation that specifically alleviates overfitting by randomly setting a collection of neurons and their respective connections to 0 during training time [116]. An illustration of the technique is shown in Figure 15.

The intuition with this technique is that, in large neural networks, neurons tend to co-adapt towards the learning goal. Roughly speaking, certain connections can conspire to produce the desired result and this leads to overfitting. Randomly dropping a subset of the neurons and their connections discourages this tendency and thus reduces the network's potential capacity to learn specific features or noise in the data. A neural network $\Phi$ with $n$ units can be seen as a collection of $2^n$ possible "thinned" neural networks. During each training cycle, a new thinned neural network $\phi$ with $m < n$ units is sampled from $\Phi$ and then, after the loss is computed, only the $m$ units are updated in $\Phi$. In practice, dropping units is controlled by a hyperparameter $p$ where $0 < p < 1$, representing the probability of the unit being retained. During test time, dropout is not applied but the units' outputs are scaled by a factor of $p$ to match the signal intensity that was observed during training with a lower capacity network $\phi$. The scaling can be interpreted as taking the average of final outputs that were each sampled from the exponential combination of thinned neural networks. Model combination, by averaging predictions from multiple neural network configurations, has been shown to be largely beneficial to the performance of machine learning methods [116]. However, for large neural networks, it is prohibitively expensive to train a large number of independent model configurations and then combine them during test time where inferencing speed is critical. In light of this, dropout is effective at simulating the training of multiple neural network configurations, simulating the combination of multiple neural networks' predictions at test time, all while reducing the network's capacity to overfit. Morever it has been found to improve performance on supervised learning tasks in compution vision, speech recognition, document classification and computational biology [116]. In the context of our research in video caption generation, we choose to use dropout regularisation on two units in our caption generator.

- *Video feature:* We apply dropout to the video feature that comes from the video encoder in order to reduce the chance of overfitting on specific salient visual features. Additionally, it allows the caption generation to be robust for different "thinned" video feature vectors.

- *Embedding vector:* We apply dropout to the embedding vector generated from the caption prefix, before passing it to the recurrent unit. Similar to the video feature, it is useful to ensure the recurrent unit does not overfit on specific semantic features encoded in the embedding vector, discouraging the caption generator from effectively memorising and reciting word sequences. We train our embeddings from scratch, and dropout is effective at discouraging co-adaption of the layer's neurons.

### 3.1.5 Optimisation

In the previous sections we discussed the components of a loss function that quantify how far a prediction is from the ground truth. The data, the neural network's architecture, its output function and the loss function can be thought of as the static parts of neural network that generally do not change during training. The dynamic parts are the network weights that are changed during training in order to minimise the loss. Optimisation is the process of iteratively refining weights to accomplish that minimisation. A helpful intuition is to think of the neural network starting from a certain altitude on a high-dimensional loss landscape where the objective is to descend towards the area of lowest altitude which corresponds to the minimum loss. The process of approaching the point of minimum loss is also called convergence. A naive optimisation method would be to repeatedly and randomly sample weights, compute the loss and then keep track of the weights which result in minimum loss. This however is not scalable for large neural networks with hundreds of millions of weights. A more systematic solution is to follow the negative of the slope of the loss function. The slope indicates the rate of change of the loss with respect to the weights. For this, the loss function should be at least partially differentiable with respect to the weights. Using the gradient of the loss function, we are guaranteed to descend towards the minimum loss of a convex loss function.

An example is using the finite difference approximation to estimate the derivative of a 1-D loss function. Given a 1-D loss $f$ which is a function of weight $x$, the finite difference involves applying a small finite change $\delta$ to $x$ and computing the resulting gradient. A positive gradient means the loss will increase and therefore the weight's influence needs to be diminished. A negative gradient means the loss will decrease and therefore the weight's influence must be magnified. In practice, weights are multi-dimensional vectors and therefore the gradient of $f$ with respect to $x$ is a vector of partial derivatives. For brevity, weight vector components are also referred to as parameters. There are three ways to compute the gradient of $f$:

- *Numerical method:* Given a weight vector $x$, add a small change $\delta$ for all vector components and calculate the gradient using the difference between $f(x + \delta)$ and $f(x)$. This method is an easy to implement approximation of the gradient vector at any point of the loss function. However, it is slow and scales linearly as the number of parameters increase. For large neural networks with hundreds of millions of parameters it would be computationally very expensive.

- *Analytical method:* Given the full loss function, a more accurate and fast way to compute the gradient is to evaluate the derivative of the function using differential calculus. Using the analytical solution, the gradient with respect to an entire weight vector can be evaluated in closed form. Because of function composition, the chain rule can be used in order to determine the gradient of the loss with respect to every weight. The analytical method is more error prone especially for complex loss functions where determining the derivate by hand is difficult.

- *Automatic differentiation* is a set of techniques for numerically evaluating the derivative of a function specified by a computer program [117]. Automatic differentiation exploits the fact that any arbitrarily complex computer program is composed of elementary arithmetic operations (addition, subtraction, division) and elementary functions (sin, cos, exp, log). By applying the chain rule repeately to these operations, the derivatives of arbitrary order can be evaluated automatically and accurately. In practice, the derivatives are progressively evaluated through the depth of the network in a method called backpropagation [69]. Automatic differentiation forms the foundation of gradient based optimisation in deep learning frameworks [103–105].

### Gradient Descent

Once the gradients have been computed for the loss function, gradient descent is the process of iteratively refining the network weights in order to minimise the prediction loss by descending on the loss landscape. The general algorithm for performing this process is shown in Listing 1.

```
    while training:
        weights_grad = evaluate_gradient(loss_func, data, weights)
        weights += - learning_rate * weights_grad # parameter update
```

Listing 1: Gradient descent algorithm for iteratively evaluating gradients and updating weights [118].

The weights are updated by the gradients scaled by a coeffient called the learning rate. While the gradient determines the direction of descent, the learning rate is a hyperparameter that determines the magnitude of descent in that direction, i.e. how much the weights need to be adjusted in that direction. This iterative algorithm is the foundation for training neural networks and our learning algorithm for the video caption generator is derived from it.

In practice, training datasets should typically have thousand or even millions of examples in order for the neural network to learn effectively. Therefore gradient descent can be done in one of the following ways.

- *Batch gradient descent:* At every step, the loss and gradients are evaluated on the entired training set. This results in a stable gradient estimation on the training set, but might be computationally wasteful to evaluate in order to adjust the weights once. Moreover, because of hardware memory constraints, it is difficult to fit the typically large training set in memory in order to perform the full evaluation.

- *Stochastic gradient descent:* At every step, the loss and gradients are evaluated on one randomly selected training example. It is much faster to evaluate than batch gradient descent since only a single example is needed. However, it is also computationally inefficient to update all network weights for one example. Additionally, it can result in unstable gradient estimates on the training set that not only significantly impact the speed of convergence, but are difficult to interpret.

- *Mini-batch gradient descent:* At every step, the loss and gradients are evaluated on a small subset of the training set. This is an efficient, relatively stable, approximation of the gradient on the train set, fast to evaluate and memory efficient since only a small batch is needed at every step. The size of the mini-batch is a hyperparameter that can be cross-validated, although usually set to a power of 2 for fast vectorised operations on parallel compute hardware [101, 102]. Notably, mini-batch gradient descent is usually referred to as stochastic gradient descent (SGD) in deep learning frameworks, even though it uses mini-batches. In the context of our research, we use this method of gradient descent especially because of our limited compute resources.

### Optimisers

The previously discussed forms of gradient descent are all based on the simple algorithm of updating weights using the product of the gradients and a constant learning rate as in Listing 1. If the learning rate is constant, the algorithm can be slow to converge. Optimisers are more sophisticated algorithms that accelerate the rate of convergence by dynamically adapting the learning rate. A faster rate of convergence is critical for training neural networks because it allows rapid experimentation and refinement of the model. In this section we discuss a number of prominent optimisers.

### Momentum

Momentum [119] utilises the idea of physical momentum in which the weight updates are accelerated based on consistent gradients over time. The algorithm is given in Listing 2.

```
v = 0
mu = 0.9
while training:
    weights_grad = evaluate_gradient(loss_func, data, weights)
    v = mu * v - learning_rate * weights_grad
    weights += v # parameter update
```

Listing 2: Gradient descent algorithm using momentum update [118].

The momentum update involves two notable terms, namely `v` which can be thought of as velocity and `mu` which can be thought of as a dampening term or coefficient of friction. In reference to an earlier intuition, the neural network starts at some loss altitude, where `v = 0`, with the objective to descend to the minimum loss. As the neural network descends towards a minimum, its velocity `v` accumulates for consistently negative or positive gradients. In other words, it corresponds to accelerated diminishing of weights that produce positive gradients, and accelerated magnification of weights that produce negative gradients. The accumulated velocity needs to be regulated, so `mu` dampens the velocity to ensure stable convergence. In pratice, the value of `mu` can be steadily increased for more regulation as the neural network approaches a minimum loss. However, the neural network can accumulate enough velocity to overshoot the minimum loss, even though the `mu` term acts as a dampener. Nesterov accelerated gradient, or Nesterov momentum [120] improves upon regular momentum by looking ahead in the direction of descent in order to determine when to decelerate before the gradients become positive, or in other words before the minimum is overshot. The main idea with Nesterov momentum is that given a weight vector at some position `x` and disregarding the second term `-learning_rate * weights_grad` in Listing 2, the dampening term alone will nudge the vector to `x + mu * v`. Therefore, since `x + mu * v` is the approximate future position, the gradients can be computed against this position instead of the original `x`, as shown in Listing 3.

```
v = 0
mu = 0.9
weights = random.rand(4,) # random starting position
while training:
    weights_ahead = weights + mu * v
    weights_grad_ahead = evaluate_gradient(data,
                                           loss_func,
                                           weights_ahead)
    v = mu * v - learning_rate * weights_grad_ahead
    weights += v # parameter update
```

Listing 3: Gradient descent algorithm using Nesterov momentum update [118].

Using this approximate look-ahead position results in better convergence rates than regular momentum, in addition to leveraging the future position to determine which direction to decelerate in. Both regular and Nesterov momentum are able to adapt the weight updates (or learning rate) depending on the gradients of the loss function, thus accelerating standard gradient descent. Instead of manipulating the learning rate globally and equally for all parameters, adaptive gradient algorithms can adapt the updates to each individual parameter depending on its influence. This effectively means every parameter can perform a larger or smaller update in the direction of descent depending on its influence, using its own learning rate. We briefly discuss adaptive gradient algorithms next.

### Adagrad

Adagrad adaptively updates the parameters using an accumulated cache of all previously observed gradients, per parameter.

```
eps = 1e-6
while training:
    weights_grad = evaluate_gradient(loss_func, data, weights)
    cache += weights_grad**2
    # parameter update
    weights += -learning_rate * weights_grad / (sqrt(cache) + eps)
```

Listing 4: Gradient descent algorithm using Adagrad [118, 121].

In Listing 4, `cache` is an accumulated vector such that every parameter corresponds to a sum of the squares of the gradients observed, so far, for that parameter in the weight vector, `weights`. This implies that `cache`, `weights` and `weights_grad` are vectors of the same dimension with element-wise correspondence. Because `cache` is an element-wise sum of squares that appears in the denominator, it acts as normalisation for the element-wise parameter update. The normalisation effectively implies that elements of `weights` that accumulate large gradients result in progressively smaller update steps, and conversely those that accumulate small gradients result in progressively larger update steps. This makes Adagrad well suited for sparse data in training large scale neural networks and word embeddings where words have a varying frequency of occurrence [119]. Over long training periods the cache builds up, meaning the learning rate completely decays for parameters with large gradients and thus the neural network eventually stops learning.

### RMSProp

RMSProp (root mean square propagation) improves on Adagrad by decaying some of the accumulated cache of gradients, thus preserving the equalising effect of Adagrad but ensuring that learning rates do not completely decay for parameters with large gradients.

```
eps = 1e-6
decay_rate = 0.99
while training:
    weights_grad = evaluate_gradient(loss_func, data, weights)
    cache = decay_rate * cache + (1 - decay_rate) * weights_grad**2
    # parameter update
    weights += -learning_rate * weights_grad / (sqrt(cache) + eps)
```

Listing 5: Gradient descent algorithm using RMSProp [118, 122].

Notably, because the `cache` is consistently modulated by `decay_rate`, it is effectively a moving average of squared gradients from within a certain window of recently observed gradients. Intuitively this means that, for every parameter, the `cache` forgets the gradients from early training loops and only keeps track of a portion of the most recent gradients.

### Adam

Adam combines the adaptive learning rates of the parameters, using the moving average of squared recent gradients as in RMSProp, and the decaying average of past gradients akin to momentum [119].

```
eps = 1e-8
beta1 = 0.9
beta2 = 0.999
while training:
    weights_grad = evaluate_gradient(loss_func, data, weights)
    m = beta1 * m + (1 - beta1) * weights_grad # like Momentum
    v = beta2 * v + (1 - beta2) * (weights_grad**2) # like RMSProp
    # parameter update
    weights += -learning_rate * m / (sqrt(v) + eps)
```

Listing 6: Gradient descent algorithm using Adam [118, 123].

Notably, the parameter update for Adam is similar to that of RMSProp except that a decaying average of past gradients `m` is used as the enumerator instead of the original gradients `weights_grad`. This is an important distinction because `weights_grad` varies between sampled batches while `m` is more stable and thus results in more stable parameter updates. Additionally, the `m` and `v` hyperparameters are initialised as 0-vectors and therefore are biased towards 0 in early training loops. The parameter update with bias correction is show in Listing 7.

```
eps = 1e-8
beta1 = 0.9
beta2 = 0.999
while training:
    weights_grad = evaluate_gradient(loss_func, data, weights)
    m = beta1 * m + (1 - beta1) * weights_grad # like Momentum
    mt = m / (1 - beta1**t)
    v = beta2 * v + (1 - beta2) * (weights_grad**2) # like RMSProp
    vt = v / (1 - beta2**t)
    # parameter update
    weights += -learning_rate * mt / (sqrt(vt) + eps)
```

Listing 7: Gradient descent algorithm using Adam with bias correction [118, 123].

Adaptive gradient algorithms have an advantage over regular gradient descent and momentum updates, because they can scale learning rates for individual weights. This adaptive learning is especially useful because in practice, manipulating the global learning rate is computationally expensive [118]. In the context of our research, we choose to use Adam as the optimiser when training our caption generator. We found it favourable because of its stable gradients across batches of $(video, caption)$ pairs. Additionally, its adaptive parameter updates, found to be effective for training embeddings of frequently and infrequently occuring words [119], were equally effective when we randomly initialised and trained word embeddings in conjuction with other parameters of the LSTM caption generator.

Deep neural networks are large scale artificial neural networks, motivated by the notion that neural networks with many layers of neurons are more efficient at capturing hierarchical features with multiple levels of abstraction, than ones with few but very wide layers [40]. Convolutional neural networks (CNNs) have demostrated state-of-the-art results in computer vision, speech and audio recognition. Recurrent neural networks (RNNs) have proven effective at processing sequential data including language, time series and videos. The two networks are the foundation of our video caption generator, where the CNN processes the video into a video feature and the RNN uses said feature to generate a natural language description, in a so-called encoder-decoder architecture [9, 10]. In the following sections we discuss details of these neural networks and highlight their effectiveness in visual and language tasks required for video caption generation.

## 3.2 Convolutional Neural Networks

Convolutional neural networks are inspired by a seminal neurological study by Hubel and Weisel on the anatomy and function of the visual cortex [124]. The study hypothesised the topographical mapping of neurons in the cortex, asserting that locality of the visual field being mapped onto the retina is preserved in neighbouring neurons of the cortex. Additionally, the study showed that neurons are hierarchically organised from simple neurons to complex neurons. This structure is replicated in CNN architectures in order to capture local spatial features, where increasing levels of abstraction are represented by deeper layers. CNNs are effective on images and videos because the spatial distribution and locality of pixels are critical for recognising visual concepts. The architecture was popularised by LeCun et al. [125] for digit recognition, and more recently by Krizhevsky et al. [15] for large-scale image classification.

### 3.2.1 Architecture Overview

CNNs take advantage of the spatial dimensions of an image, with the input being a volume $V_i$ of dimension $w_i \times h_i \times d_i$, where $w_i$ and $h_i$ are the spatial dimensions of the image and $d_i$ is number of colour channels. The architecture is chosen to be more suitable for processing the 2D grids of image pixels in a way that preserves spatial locality, unlike a regular neural network that requires a stretched out $1 \times (w_i \cdot h_i \cdot d_i)$ dimensional input. The CNN architecture is composed of a sequence of interleaving convolutional and pooling layers, followed finally by fully connected layers. The $w_i \times h_i \times d_i$ image volume is

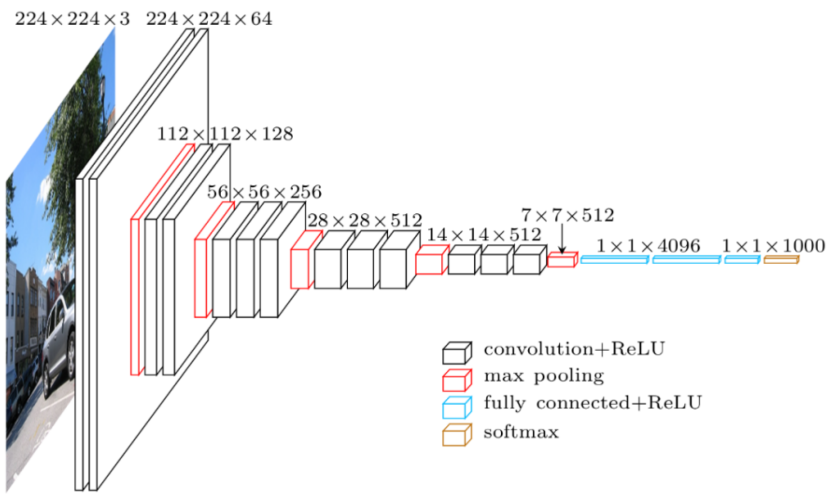


Figure 16: CNN architecture with convolutional, pooling and fully connected layers as arranged in the VGGNet-16 architecture [5].

processed by the first CNN layer which produces an activation volume. Each subsequent layer transforms the activation volume through a differentiable function until a flattened volume is produced for classification. A typical CNN architecture is illustrated in Figure 16.

## Convolutional Layer

The convolutional layer is the foundational and typically computationally heavy unit in a CNN. Unlike a regular neural network where every neuron in a layer is connected to every output of the previous layer, a convolutional layer consists of neurons arranged in a $w_f \times h_f \times d_f$ volume, $V_f$, where $w_f < w_i$ and $h_f < h_i$ and $d_f$ is usually a power of 2. All neurons at a depth-level or "slice" of $V_f$ only connect to a small region within the 2D $w_i \times h_i$ spatial dimension of the image, but extend the full depth $d_i$ of the image volume. More importantly, all the neurons at this slice share the same weights, collectively called a filter, across the $w_i \times h_i$ spatial dimension of the image. Intuitively, a slice in $V_f$ can be visualised as one filter sliding across the $w_i \times h_i$ space and interacting with the image input volume using the same weights at every point. This arrangement significantly reduces the number of weights needed to process an image because the same weights are used to process different image regions. Sharing weights at a slice is analogous to searching for a particular feature across all image regions. Along the full depth $d_f$ in $V_f$, all the neurons are connected to the same image region but with different slice-level weights. The sliding operation involves a dot-product between the image region and one slice of the filter volume $V_f$, followed by a nonlinear transformation such as ReLU, resulting in a 2D $w_j \times h_j$ activation map. The sliding operation is illustrated in Figure 17.

The activation maps for all the filters are stacked together to produce the final activation volume. If for example $d_f = 56$, the final activation is $w_j \times h_j \times 56$. This volume is used as

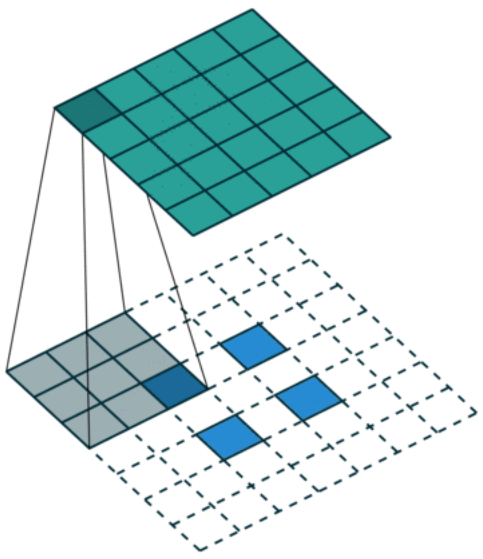


Figure 17: Illustration of the convolutional operation. The green square represents all the possible positions of filter. The grey area (including one blue square) is the spatial extent of the filter on the input image. The blue squares are an example feature in the image that filter might detect. The filter slides and performs a dot-product operation against every image region, each time producing a scalar value, eventually accumulating into a $w_j \times h_j$ activation map [126].

input in the next CNN layer. There are some hyperparameters that determine the reach of the filters and the dimensions of the output activation volume.

- *Receptive field / Filter size:* Filters in initial layers typically have small receptive fields for detecting simple features, while in deeper layers they may have larger receptive fields for detecting composite features.
- *Depth:* Previously noted as the $d_f$ dimension of $V_f$, the depth corresponds to the number of filters used at that convolutional layer. Since each filter has different weights, it will extract a different feature from the same image region. Altogether, the filters decompose the image volume into $d_f$ seperate activation maps.
- *Stride:* The step size by which the filter slides across the image volume, when performing the dot-products. For example, when the stride is 1 the filter moves by 1 pixel at a time.
- *Zero padding:* Since the spatial dimensions of the filter volume $V_f$ are less than the image volume $V_i$, the convolutional operation automatically shrinks the output volume. This leads to rapid loss of spatial features over repeated convolutional operations, through the depth of the CNN. Zero padding counteracts this tendency to lose spatial features too early in the CNN, in order that as many spatial features are preserved for the classification layers. Given an image volume of size $w_i \times h_i \times d_i$,

filter size $F$, zero padding $P$ and stride $S$, the output volume dimensions can be determined by

$$\begin{aligned} w_o &= \frac{w_i - F + 2P}{S} + 1, \\ h_o &= \frac{h_i - F + 2P}{S} + 1, \\ d_o &= d_i. \end{aligned} \tag{21}$$

If the image volume is $224 \times 224 \times 3$, the number of filters is 56, $F = 5$ and $S = 1$, then $P$ needs to be 2 in order to have an output volume $V_o = 224 \times 224 \times 56$ that preserves the original spatial dimensions.

Throughout the convolutional layer, all linear and nonlinear transformations performed on the image volume are differentiable, and all the shared depth-slice weights are trainable through backpropagation and gradient descent.

## Pooling Layer

Pooling layers are usually interleaved with convolutional layers in a CNN. While zero padding counteracts against rapid loss of spatial features, a pooling layer progressively reduces the spatial size of the activation volume, in a deliberate and controlled manner, in order to reduce the number of parameters and computation, hence regulating overfitting. Additionally, the pooling operation improves invariance to certain distortions in the input and increases the receptive field for subsequent convolutional layers. Using fixed filters, the pooling layer independently operates on every depth-slice of the input volume, reducing its spatial dimensions while maintaining the depth. The max pooling operation is a common operation in CNNs [127], where $2 \times 2$ filters with a stride of 2 uniformly reduce every depth-slice by computing the maximum value for every $2 \times 2$ region. This is illustrated in Figure 18.

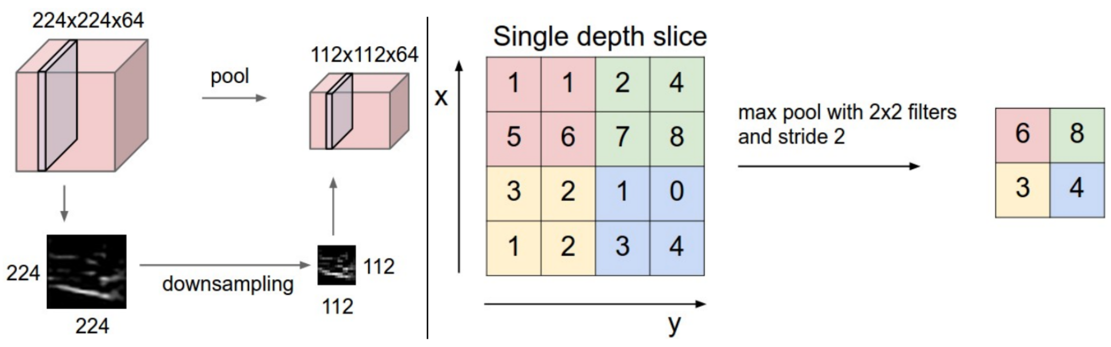


Figure 18: Illustration of the max pooling operation [128].

Given a volume $w_o \times h_o \times d_o$ from the previous convolutional layer, pool filter size $F$ and stride $S$, the output volume dimensions can be determined by

$$\begin{aligned} w_p &= \frac{w_o - F}{S} + 1, \\ h_p &= \frac{h_o - F}{S} + 1, \\ d_p &= d_o. \end{aligned} \tag{22}$$

The pooling layer introduces no trainable parameters since it computes a fixed function on its input. Other less common pooling operations are average pooling which computes average of the $2 \times 2$ region [129], and L2-norm pooling which computes a sum of squares for the $2 \times 2$ region [130].

**Fully Connected Layer**

The fully connected layer has neurons arranged like a regular neural network, where every neuron is connected to every input in the incoming activation volume. Therefore, a combination of the linear transformations and nonlinearities like ReLU are used to compute the final activation of the layer. Normally for a task like image classification, the final activation would have dimension equal to the number of possible classes, and the elements would be un-normalised class scores that can be passed to a softmax function to compute the final class probabilities. In the context of video caption generation, we do not add the softmax layer and instead pass the final activation to the caption generator as a conditioning vector.

## 3.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) can be effective at modelling sequential data. They utilise the neurological concept of working memory [131]. The output not only depends on the input but also on the previous outputs that are internally represented as context by the recurrent unit. Unlike regular neural networks and CNNs, RNNs do not require a fixed length input and, in theory, can produce an arbitrarily long sequence of outputs given a variable length sequence of inputs. RNNs have demonstrated their effectiveness in tasks such as machine translation [1, 18], processing time series [45, 132], processing videos [20] and caption or paragraph generation [95].

### 3.3.1 Architecture Overview

RNNs consist of a recurrent unit that takes in an input and produces an output just like a regular neural network unit. Additionally and more importantly, the activation that results from the input is stored as internal context of the unit. Future predictions will depend on the inputs at that time and also the stored context. The recurrent unit is illustrated in Figure 19.

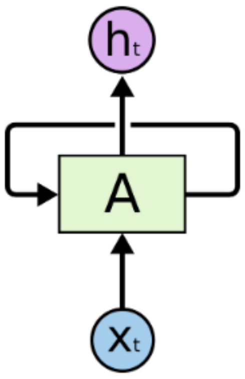


Figure 19: Illustration of the recurrent unit [133].

When processing a sequence with multiple inputs, e.g. translating a sentence from one language to another, the recurrent unit sequentially processes a source sequence $X = (x_0, x_1, \ldots, x_T)$ to produce a target sequence $Y = (y_0, y_1, \ldots, y_T)$. Given the input $x_t$ at time $t$, the unit will produce an activation $h_t$ which can be used to predict the output $y_t$. Crucially, the context $h_t$ is stored and used at time $t + 1$ when processing $x_{t+1}$. The sequential process is better illustrated in Figure 20, where each time step is shown separately.

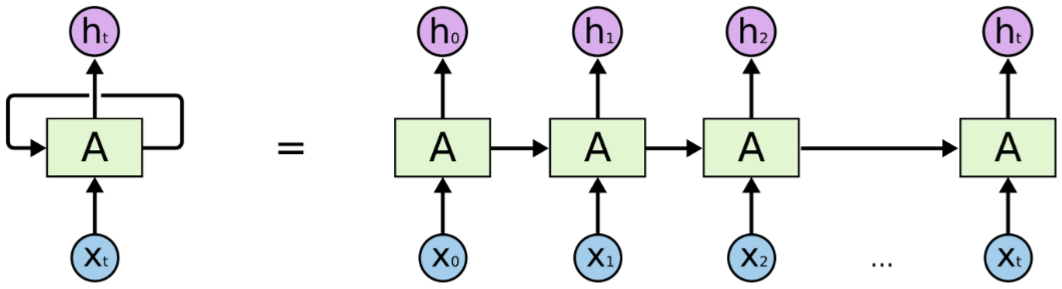


Figure 20: Illustration of the unrolled recurrent unit [133].

At any given time $t$, the activation $h_t$ depends on the input $x_t$ and the previous internal context $h_{t-1}$. More formally, this dependency can be expressed as

$$h_t = \phi(W, h_{t-1}, x_t). \tag{23}$$

where $\phi$ is the RNN and $W$ contains the RNN weights. Given an input $x_t$, the full expression for the predicting $y_t$ is as follows:

$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t), \\ y_t &= \mu(W_{hy}h_t). \end{aligned} \tag{24}$$

$W_{hh}$, $W_{xh}$ and $W_{hy}$ are learnt projection matrices or weights on $h_{t-1}$, $x_t$ and $h_t$ respectively, and $\mu$ is the softmax fuction. Given an input sequence $X = (x_0, x_1, \ldots, x_T)$, the same weights are applied at every time step for every input $x_t$. Similarly, the tanh activation is used to produce $h_t$ and the softmax function then produces the prediction $y_t$ at every step. During backpropagation, the gradients for all prediction losses accumulate into updates for $W_{hh}$, $W_{xh}$ and $W_{hy}$ since they contribute to the loss at every step. Backpropagation is performed through all the time steps of the unrolled RNN using a backpropagation-through-time (BPTT) approach [134].

### 3.3.2 Limitations of Recurrent Neural Networks

Recurrent neural networks are effective and flexible at handling sequential data with long term dependencies [135], but they do have some notable limitations that we discuss below.

- *Input sequence limit:* In theory, the recurrence function in Equation 23 can be applied over very long sequences, e.g. a large sentence corpus. However, in practice, it is infeasible to load the entire sentence corpus in memory, store all context vectors and all losses in order to perform backpropagation. Instead the recurrence function is applied on a subsequence with a predefined length, similar to batches in mini-batch gradient descent, where the final context $h_t$ of one subsequence becomes the initial context for the next subsequence. Therefore, RNNs usually have a predefined sequence length or number of time steps at initialisation.

- *Long term dependencies:* RNNs apply the same fixed recurrence function in Equation 29 across all time steps in a sequence. This repeated operation results in multiple transformations of the input $x_t$ and the context $h_t$, which can lead to gradual loss of relevant context over very long sequences [56]. Consider trying to predict the last word in the text "I grew up in France in the small town of Carbonne... I speak fluent *French*". In order to narrow down the language, the RNN needs the context of "France", but the distance between the current step and the relevant context is large and repeated transformations by the recurrent unit might have completely diminished the context for "France" in favour of context from the most recent words.

- *Vanishing or exploding gradients problem:* Directly and repeatedly transforming the context over long sequences can have profound effects on the gradient flow dynamics and trainability of RNNs [56, 57, 136]. Considering a long sequence $X =$

$(x_0, x_1, \ldots, x_T)$ and the recurrence formula $h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$, the gradients of the loss function with respect to all the contexts look as follows:

$$\begin{aligned} \frac{\partial L}{\partial h_t} &= \frac{\partial h_t}{\partial h_{t-1}} \cdot \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdot \frac{\partial h_{t-2}}{\partial h_{t-3}} \cdots \frac{\partial h_1}{\partial h_0}, \\ \frac{\partial L}{\partial h_t} &= W_{hh} \cdot \frac{d}{dh}(h_{t-1}) \cdot W_{hh} \cdot \frac{d}{dh}(h_{t-2}) \cdot W_{hh} \cdot \frac{d}{dh}(h_{t-3}) \ldots W_{hh} \cdot \frac{d}{dh}(h_0). \end{aligned} \tag{25}$$

Given gradient of the loss function at time $t$ and using the chain rule, backpropagation recursively computes gradients through the tanh and then through the term $W_{hh}h_{t-1}$, for every time step. Gradients are continuously multiplied by $W_{hh}$, from time $t$ to time 0, since the weight is repeatedly applied at every time step. If the $W_{hh}$ matrix is initialised with small numbers the gradient will vanish because of continuous multiplication with a small numbers. This means the RNN cannot learn long term dependencies if the gradient information cannot reach distant time steps. Similarly if the $W_{hh}$ matrix is initialised with large numbers, the gradient will explode because of continuous multiplication with large numbers, leading to infinite loss and the RNN being unable to converge to a minimum loss.

The vanishing or exploding gradient problem makes it particularly difficult for RNNs to learn long term dependencies since any long term corrections are gradually decomposed [56,57,136]. In practice, exploding gradients can be mitigated with gradient norm clipping [136] while vanishing gradients can be alleviated by a soft constraint [136]. However, these mitigations do not alleviate the long term dependency problem caused by direct and complete transformation of the context $h_t$ at every time step. In the following section, we discuss long short-term memory (LSTM) networks that alleviate this problem.

## 3.4 Long Short-Term Memory (LSTM)

Long short-term memory (LSTM) networks [49] are a variant of RNNs that are specifically designed to more effectively learn long-term dependencies. Similar to a regular RNN, the LSTM retains context with $h_t$ but also retains a cell state from which information can be added and removed in a manner regulated by a set of neural network structures called gates. Crucially, the cell state allows information to flow through time steps without being significantly transformed.

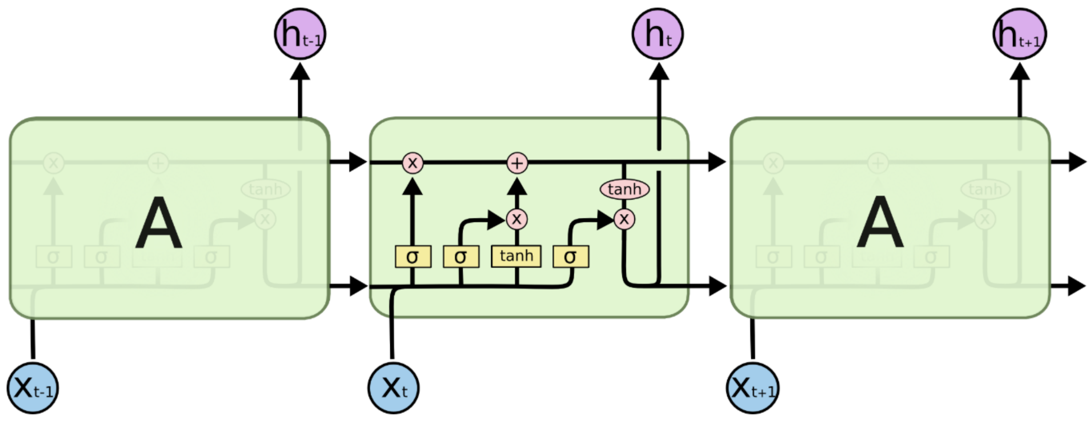


Figure 21: Illustration of the LSTM unit showing the gates and interactions contained in one time step [133].

### 3.4.1 Architecture Overview

The LSTM network is composed of multiple gates that are used to optionally let information through the unit. The mathematical formulation of the gates is as follows:

$$\begin{aligned} c_t = f(W_f, x_t, h_{t-1}) \odot c_{t-1} + i(W_i, x_t, h_{t-1}) \odot g(W_g, x_t, h_{t-1}), \\ h_t = o(W_o, x_t, h_{t-1}) \odot \tanh(c_t). \end{aligned} \tag{26}$$

Symbols $i$, $f$, $o$ indicate the input, forget and output gates. The input and forget gates use the previous context $h_t$ and $x_t$ to determine how much information to add or remove from the cell state $c_{t-1}$. The output gate determines how much information filters into $h_t$ and continues to the next time step. The $\odot$ symbol signifies element-wise multiplication between vectors. We discuss details of the interactions of these gates next.

#### Forget Gate

The forget gate decides which information to remove from the cell state, and is formulated as follows:

$$\begin{aligned} f = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \\ f \odot c_{t-1}. \end{aligned} \tag{27}$$

The gate considers the concatenation of the input $x_t$ and previous context $h_{t-1}$, and using the sigmoid activation, outputs a value between 0 and 1 for every element of the cell state vector $c_{t-1}$. A value of 1 indicates that the element must be retained while 0 indicates that it must be discarded. This vector of activations is then element-wise multiplied with the cell state $c_{t-1}$ to apply the updates. Consider trying to predict the correct pronoun in the text “Anna gave Simba a pass mark and _ was happy”. In order to correctly predict “he”, the LSTM needs to remove any context related to “Anna” and add context related to “Simba”.

### Input Gate

While the forget gate determines what information to remove, the input gate decides what information to store in the cell state. This is done in two parts, first deciding what values to update and then computing a candidate vector that could be added to the state. These operations are as follows:

$$\begin{aligned} i &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \\ g &= \tanh(W_g \cdot [h_{t-1}, x_t] + b_g), \\ & i \odot g. \end{aligned} \tag{28}$$

Similar to the forget gate, the input gate considers the concatenation of the input $x_t$ and previous context $h_{t-1}$, and using the sigmoid activation, outputs a value between 0 and 1 for every element of the candidate cell state vector $g$. A value of 1 indicates that the element must be updated while 0 indicates that it must not be updated. $g$ is a vector of new candidate values that can be added to the state. It is computed by also considering the concatenation of $x_t$ and $h_t$, and producing a vector of elements between $-1$ and 1 using a tanh activation. Finally the cell state update is computed by element-wise multiplication between $i$ and $g$. If $i$ has elements $e_i$ such that $0 < e_i < 1$ and $g$ has elements $e_g$ such that $-1 < e_g < 1$, $i \odot g$ can be thought of as coupling two concepts at play when determining which elements of $c_{t-1}$ to update:

- $0 < e_i < 1$: indicates *whether* the corresponding element $e_g$ from the candidate $g$ must be added to the cell state. In the case of the previous pronoun prediction example, the LSTM needs to add the element for “Simba”.
- $-1 < e_g < 1$: indicates *how much* of the element $e_g$ from the candidate $g$ must be added or subtracted to the cell state.

Once the forget gate is computed for information to be removed and the input gate has been computed for information to be updated in the cell state, the two vectors are element-wise added to produce the new cell state as follows:

$$c_t = f \odot c_{t-1} + i \odot g. \tag{29}$$

Crucially, this regulated addition and subtraction from the cell state is how the LSTM ensures that the internal contexts $c_t$ and $h_t$ are not completely transformed in a way that diminishes relevant information over long dependencies, as is often observed with regular RNNs. Additionally, the additive interaction between $f \odot c_{t-1}$ and $i \odot g$ ensures that, during backpropagation, gradients are evenly distributed across these units' weights $W_f$,

$W_i$ and $W_g$ all the way through the time steps, thus preventing the vanishing or exploding gradient problem. The additive operation also serves a similar purpose in residual units of residual neural networks (ResNets) [6], a very deep variant of CNNs that use additive connections to not only prevent completely transforming the input, but also as "gradient highways" during backpropagation.

### Output Gate

Once the new cell state $c_t$ is computed, the output gate determines how much of it filters through to the next time step as $h_t$. While $c_t$ is fully retained in the next time step, $h_t$ is passed to next time step while also being used as input for prediction, e.g. through a softmax classifier at the current time step. The output gate is formulated as follows:

$$\begin{aligned} o &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \\ h_t &= o \odot \tanh(c_t). \end{aligned} \tag{30}$$

The output gate considers the concatenation of the input $x_t$ and previous context $h_{t-1}$, and through the sigmoid activation, produces values between 0 and 1 for every corresponding element of the cell state $c_t$. This determines which elements of the cell state filter through to the next step as part of $h_t$. The cell state is subjected to a tanh activation, producing values between $-1$ and 1 and thus determining how much of each element filters through to the next step in $h_t$. Similar to the modulating operands of $i \odot g$ in the input gate, the operands of the $o \odot \tanh(c_t)$ operation decide *which* elements of $c_t$ filter through to $h_t$ and *how much* of those elements filter through to $h_t$.

LSTMs are more robust at learning long-term dependencies compared to regular RNNs and have thus become the de facto standard for modelling sequential data in machine translation [1, 18], time series [45, 132], videos [20] and caption or paragraph generation [9, 10, 95]. The LSTM does have notable variants that make architectural changes to the gates. Gers and Schmidhuber [137] add "peep hole" connections that allow the forget, input and output gates to also consider the current cell state $c_t$ in addition to $x_t$ and $h_t$. Another popular variant is the gated recurrent unit (GRU) [96], which combines the forget and input gate into a single update gate. It also merges the cell state $c_t$ and internal context $h_t$ resulting in a simpler model than the standard LSTM.

## 3.5 Encoder-Decoder Architectures

We have discussed the architectural details of CNNs and RNNs, and showed the characteristics that have made them effective solutions for modelling visual and sequential data, respectively. Research in image and video caption generation has been able to leverage the strengths of these two models, and has been inspired by work in neural machine translation [1,18] in order to translate image or video data into natural language descriptions [9,10,20,24]. The main idea behind encoder-decoder architectures is to inject visual context into an RNN to act as a conditioning vector when generating words. Therefore, given an image, the CNN encodes the image into a representation and the RNN decodes this representation into a natural language sentence. In an RNN unit, the mathematical formulation can be as follows:

$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t + W_{vh}v), \\ y_t &= \mu(W_{hy}h_t). \end{aligned} \tag{31}$$

The term $v$ is the activation vector from the last fully-connected layer of a CNN, right before the classifier. The activation vector interacts with an additional weight $W_{vh}$. Similar to weights $W_{hh}$ and $W_{xh}$, the visual weight matrix $W_{vh}$ is repeatedly applied across all time steps, and all gradients from the accumulated loss will be updated during backpropagation. As was prevously noted from image and video captioning literature, the LSTM is usually chosen as the decoder, where the visual activation is injected into the LSTM cell in one of several possible ways.

- Xu et al. [10] and Devlin et al. [71] initialise the internal context $h_t$ with the image vector, in a method called "init-inject" [70], as follows:

$$\begin{aligned} h_t &= v, \\ c_t = f(W_f, x_t, h_{t-1}) \odot c_{t-1} + i(W_i, x_t, h_{t-1}) &\odot g(W_g, x_t, h_{t-1}), \\ h_t &= o(W_o, x_t, h_{t-1}) \odot \tanh(c_t). \end{aligned} \tag{32}$$

- Vinyals et al. [9] and Nina and Rodriguez [75] input the image vector as the first word in the LSTM, in a method called "pre-inject" [70]:

$$\begin{aligned} x_t &= v, \\ c_t = f(W_f, x_t, h_{t-1}) \odot c_{t-1} + i(W_i, x_t, h_{t-1}) &\odot g(W_g, x_t, h_{t-1}), \\ h_t &= o(W_o, x_t, h_{t-1}) \odot \tanh(c_t). \end{aligned} \tag{33}$$

- Donahue et al. [20] and Yao et al. [78] combine the image vector and word at every time step, in a method called "par-inject" [70]:

$$\begin{aligned} z_t &= v + x_t, \\ c_t = f(W_f, z_t, h_{t-1}) \odot c_{t-1} + i(W_i, z_t, h_{t-1}) &\odot g(W_g, z_t, h_{t-1}), \\ h_t &= o(W_o, z_t, h_{t-1}) \odot \tanh(c_t). \end{aligned} \tag{34}$$

The encoder-decoder architecture is particularly effective because the two models can share gradients and thus be jointly trained for the best weight configurations for visual recognition and caption generation. Additionally, the CNN encoder can be pretrained on large visual datasets such as ImageNet, UCF-101 or Sports1M [26,51,92], while the LSTM decoder can be pretrained on large word corpora like Wikipedia [61]. In pratice, the two models are initialised with pretrained weights to leverage transfer learning, where weights learnt from one task with a large data set are reused in a different but related task [138]. The CNN encoder's fully-connected layer weights can be finetuned for the captioning task, while the LSTM decoder can be initialised with trained word embeddings [29, 31, 32]. In our video caption generator, we use a CNN encoder pretrained on Sports1M and our LSTM decoder is trained in multiple configurations with jointly trained word embeddings and finetuned word embeddings.

## 3.6 Word Embeddings

Word embeddings are a set of techniques that aim to encode the semantic meaning of words into a geometric space. This is done by associating an $n$-dimensional vector, called an embedding, with every word in the defined vocabulary. The relationships between words can thus be represented by comparing the words' embeddings. Intuitively, the L2 or cosine distance between any two embeddings captures their semantic relationship, i.e. a short distance implies closely related words while a long distance implies relatively unrelated words. Word embeddings are computed by applying dimensionality reduction techniques to datasets of co-occurence statistics between words in a large corpus of text [139]. For image or video caption generation, word embeddings are crucial for generating sensible natural language descriptions, because they are optimised to predict words that appear in their respective contexts.

### 3.6.1 Pretrained Word Embeddings

Pretrained embeddings are obtained either via neural networks or matrix factorisation based methods. The method of Mikolov et al. [29, 30] uses an unsupervised algorithm to learn word representations in a continuous vector space from large unlabelled text corpora. The algorithm relies on the distributional hypothesis which posits that words that are used and occur in the same contexts have similar meanings [28]. The learnt representations preserve linear regularities such as differences in syntax and semantics. This enables computation of analogies via vector addition and cosine similarity, e.g. `king` - `man` + `woman` = `queen`. Two simple neural network based models are proposed for computing embeddings. The continuous bag of words (CBOW) produces embeddings as a byproduct of predicting a central word given surrounding context words that occur before and after the central word. The continuous skip-gram is similar to CBOW but tries to predict a range of surrounding context words given a central word. Due the simplicity of these models, very accurate high-dimensional embeddings are produced using large corpora in the order of billions of words.

The approach of Pennington et al. [31], called GLOVE, improves on the skip-gram model by observing that while skip-gram produces fine-grained structure in the vector space, demonstrated by computable analogies via vector arithmetic, it poorly utilises the global word-word co-occurence statistics of corpora since it is trained on local context windows instead of global co-occurence counts. The resultant model combines the advantages of both global matrix factorisation and local context window methods. GLOVE leverages the statistical information by training only on non-zero elements in a word-word co-occurence matrix rather than the entire sparse matrix or on individual context windows in a large corpus.

These methods for learning word representations assign distinct word vectors for words in the vocabulary. Therefore, they ignore the morphology of words, i.e. words that take different forms depending on a change of a few characters. This is a limitation for morphologically rich languages like Turkish or Finnish where some forms rarely occur in the training corpus making it difficult to learn robust representations. Bojanowski et al. [32] tackle this problem by directly improving on the skip-gram model, where each word is represented as a bag of $n$-grams. An $n$-gram is an ordered set of $n$ items from a given piece of text or speech. Each $n$-gram is associated with a vector representation and words are represented as a sum of these representations. This approach takes into account the internal structure of words, resulting in representations that capture words in different forms.

### 3.6.2 Jointly Trained Word Embeddings

Word embeddings are computed as a byproduct of a specific language modelling task. Embeddings are usually parameters in an embedding matrix that is part of a hidden layer in a neural network. The word representations are confined to the vocabulary of the task, e.g. words in the English-French translations or descriptions in an image or video caption generation task. The word embeddings are thus optimised as part of maximising the correct word prediction when translating a sentence or generating a caption. There is an inherent trade-off between pretrained embeddings and jointly trained embeddings. Pretrained embeddings are learnt from significantly large text corpora and therefore exposed to many more contexts, but they might not necessarily be immediately transferrable to a specific task. Jointly trained embeddings are typically exposed to a significantly smaller vocabulary and thus fewer contexts, but are closely optimised for the desired task. In our experiments (Chapter 5) we find that either jointly training embeddings or finetuning pretrained embeddings, seems to be advantageous in getting the best results.

# 4 Proposed Approach

In the previous chapter we discussed in detail the foundational mechanisms of neural networks in deep learning. We described how optimisation based techniques are used to train CNNs and RNNs for visual and sequence modelling tasks, respectively. Encoder-decoder architectures, that combine CNNs and RNNs, have become a popular choice for jointly modelling visual and textual information [140], thus pertinent to our study in video caption generation. Merge architectures, that efficiently incorporate visual information when generating captions, have been shown to generate similarly sensible captions, compared to the more popular inject architectures in image captioning research [70]. In this chapter we discuss the overall architecture for our encoder-decoder based video caption generator, that adapts an image captioning based merge architecture for generating video captions. Additionally, given this architecture, we choose the evaluation metrics that have become standard benchmarks for natural language generation, namely BLEU, METEOR, CIDEr and ROUGE. We discuss the theory and intuition behind these metrics and motivate our choice for using them in our study.

## 4.1 Encoder

### 4.1.1 Architecture Overview

The encoder in our video caption generator is a 3D CNN with spatiotemporal convolutions. Spatiotemporal convolutional operations differ from normal spatial convolutions in that they can operate on a video in both the spatial and temporal dimensions.

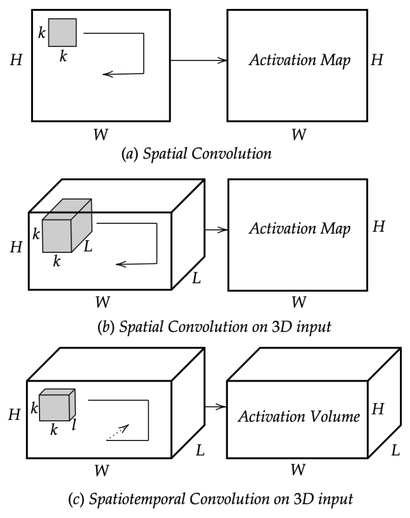


Figure 22: Illustration of different convolutional operations on image and video input. Image inspired by [25].

- *Spatial convolutions on 2D input:* An example input is an image which in fact is a 3D input with a *width*, *height* and *depth* for the RGB colour channels. A regular CNN filter of dimensions $depth \times k \times k$ extending the entire depth of the input, spatially operates on the input and typically produces a $width \times height$ activation map as in Figure 23(a).
- *Spatial convolutions on 3D input:* An example input is a video which is a collection of image frames, where every frame is $depth \times width \times height$. Therefore the full dimensionality of the video volume is $(L \cdot depth) \times width \times height$, where $L$ is the number of frames or the temporal dimension of the video. A regular CNN filter of dimensions $(L \cdot depth) \times k \times k$ spatially operates on the video volume and, because of the dot-product operation between the kernel and the input, collapses the input volume in the $L \cdot depth$ dimension to produce a $width \times height$ activation map as in Figure 23(b). This spatial operation may lose temporal information that is critical for video classification [26], action recognition [25] and caption generation [14].
- *Spatiotemporal convolutions on 3D input:* Given a video input volume of dimensions $(L \cdot depth) \times width \times height$, the 3D CNN filter is of dimensions $(l \cdot depth) \times k \times k$ where $l < L$, and uses a stride of 1 both spatially and temporally. Since the kernel extends only a depth $l$ in the video volume, it can operate spatially and temporally through the $L$ frames, producing an $(L \cdot depth) \times width \times height$ activation volume as in Figure 23(c). Given such an activation volume, a pooling filter also operates spatially and temporally to produce an $(L \cdot depth) \times \frac{width}{2} \times \frac{height}{2}$ activation volume. Spatiotemporal convolutions and pooling operations therefore preserve some temporal information that can be critical for video classification, action recognition and caption generation. A 3D CNN is effective at producing a feature vector that encodes temporal information, critical in order to describe actions, scenes and objects in video, and thus pertinent for our video caption generator.

### 4.1.2 Training

We initialise the 3D CNN encoder and, for every network layer, transplant weights trained on the Sports1M dataset for sports video classification [25]. The encoder is automatically able to extract and preserve generic spatiotemporal features in open domain videos that are not in the sports category. For every video in the MSVD dataset [27], the encoder takes $16 \times 171 \times 128 \times 3$ video segments as input, where 16 is the number of frames, 171 is the width, 128 is the height and 3 is the colour channel depth. The encoder transforms the video segments into 4096-dimensional feature vectors used as visual information for training the decoder. We elaborate on the cadence with which we sample the 16 frames per video, as part of the next chapter.

## 4.2 Decoder

### 4.2.1 Architecture Overview

The decoder is composed of a $256 \times V$ embedding layer, where $V$ is the vocabulary size of the most frequently occurring words in the captions across the dataset. The embedding layer is comprised of an embedding matrix, which is a set of trainable parameters used to encode the meaning of words in vectorised form [29, 31, 32]. Given a 10000 word vocabulary, i.e. $V = 10000$, every word is first represented as a 10000 length one-hot vector where the 1 is an arbitrary position assigned to that word in the vocabulary. This lookup position is later used to reference the word during predictions by the decoder. The word's embedding is computed by multiplying the embedding matrix, which is randomly initialised, and the one-hot vector representation of the word. The process is illustrated in the following equation:

$$\underbrace{\begin{bmatrix} 0.2 & 0.3 & \dots \\ \vdots & \ddots & \\ 0.3 & & -0.3 \end{bmatrix}}_{\text{[256, 10000] word embedding matrix}} \times \underbrace{\begin{bmatrix} 1 \\ \vdots \\ 0 \end{bmatrix}}_{\text{[10000, 1] one-hot word input}} = \underbrace{\begin{bmatrix} 0.2 \\ \vdots \\ 0.3 \end{bmatrix}}_{\text{[256, 1] word embedding output}} \tag{35}$$

The $256 \times 1$ word embedding is used as input, $x_t$, into a 256-dimensional LSTM unit, that updates the cell state $c_t$ and computes the hidden state $h_t$, both of which are 256-dimensional. The 4096-dimensional video feature vector from the encoder is condensed into a 256-dimensional vector through a trainable fully-connected layer. Inspired by the merge architectures described by Tanti et al. [70], the condensed video feature is merged with the LSTM output $h_t$ via element-wise addition. The resulting 256-dimensional vector is passed through a fully-connected layer, and then passed into a softmax classifier. The softmax classifier produces a probability distribution over all the 10000 words, where the most probable word's position is used to look up the original word in text form.

### 4.2.2 Training

The caption generator is trained using the MSVD dataset [27] which is comprised of 1970 open domain videos, each of which having an average of 41 captions in multiple languages, e.g. English, German, Tamil and Arabic. However, for simplicity, training is done using only English language captions. The entire dataset is split into 70% for training, 20% for validation and 10% for testing. Before training, we preprocess every video through the 3D CNN encoder to produce a video feature vector. These feature vectors and captions form $(video, caption)$ pairs that we use for training the decoder. For every $(video, caption)$ pair, the decoder begins with the start token *startseq* as $x_t$ and, through the embedding layer and LSTM unit, predicts the hidden state $h_t$. The softmax classifier then predicts the

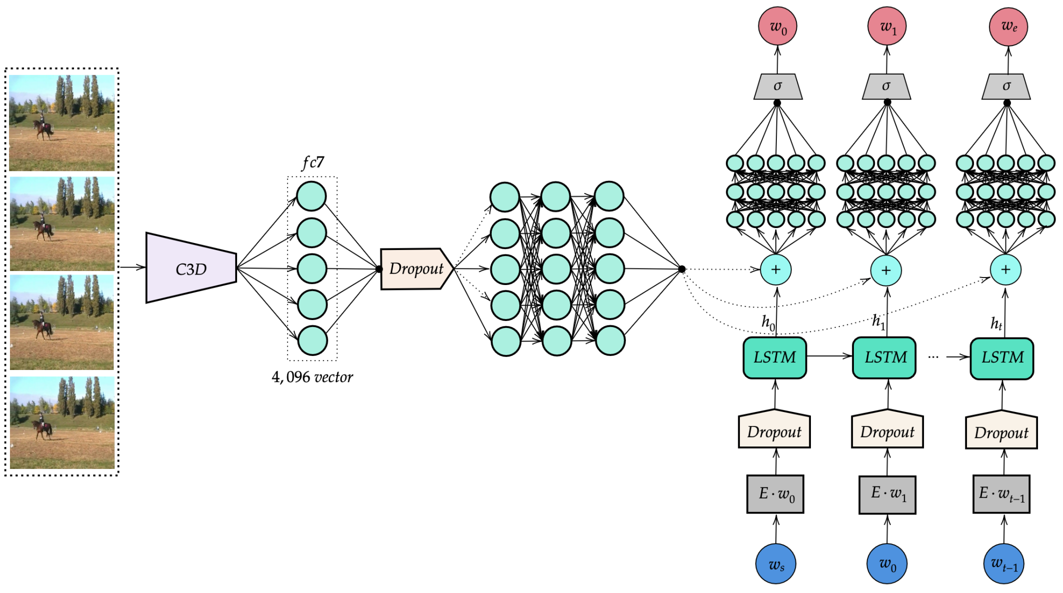


Figure 23: Illustration of our full encoder-decoder architecture.

next word, $y_t$, using the merged result $video + h_t$. The next input $x_{t+1}$ is a concatenation of the first two words *startseq* and $y_t$ and this, along with the video, is used to generate the next word $y_{t+1}$. The decoder repeats this recurrent prediction cycle until the end token *endseq* is generated. The caption generation process is illustrated in Listing 8.

```
    vid = 3d_cnn(video) # extracting the video feature
    # Input                                   # Output
    (vid, startseq)                             a
    (vid, startseq a)                           lion
    (vid, startseq a lion)                      chasing
    (vid, startseq a lion chasing)              a
    (vid, startseq a lion chasing a)            gazelle
    (vid, startseq a lion chasing a gazelle)    endseq
```

Listing 8: Example caption generation by the decoder through multiple time steps.

At every time step $t$, the decoder predicts the next word using the embedding matrix and LSTM weights. After the *endseq* has been generated, all the prediction losses at every time step are summed up to update the weights. Backpropagation through time (BPTT) is applied to the LSTM weights and the embedding weights since both sets of parameters contribute to the prediction loss at every time step. The overall encoder-decoder architecture is fully illustrated in Figure 23.

### 4.2.3 Design Motivations

In designing the caption generator, there are several considerations in order to best support our experiments.

- *Merging visual and semantic information:* Our decoder merges the video feature vector and the hidden state $h_t$ from the LSTM unit. Unlike the more prominent inject architectures that incorporate visual information either through the hidden state [71] or as the first word in the LSTM [9, 20, 75, 78], merge architectures separately handle the two streams of information which simulatenously preserves visual information, critical for influencing word predictions in the video description, and reserves all LSTM capacity for modelling semantic information. Literature on image captioning [70] has shown that merge archictectures maintain comparable performance against inject archictectures while using significantly smaller LSTM dimensions, which has important model engineering implications. We adapt this technique in video captioning, replacing the image features with video features.

- *Jointly training embeddings:* Instead of using pretrained word embeddings, we train our word embedding matrix as a by-product of optimising all decoder parameters for accurate caption predictions. As a rule-of-thumb, pretrained word embeddings result in a more expressive decoder that generates diverse captions because the embedding matrix is trained on a large and diverse vocabulary. Therefore pretrained embeddings are suitable for tasks with relatively small datasets, while jointly training them is suitable when the task's training set is sufficiently large. There is an inherent trade-off between the two strategies because, although pretrained embeddings can produce more diverse captions, they are likely to be biased towards the word distribution in the large dataset of the pretraining task. Conversely, while jointly training embeddings with the decoder will lead to less diverse captions, it simplifies the decoder, making it end-to-end trainable with only $(video, caption)$ pairs. This may also result in optimal embedding parameters for the caption generation task as opposed to parameters trained on a possibly different task. In terms of caption generation, we compare in the next chapter the performance difference between pretrained embeddings, a trained embedding layer and finetuned embeddings, according to the BLEU, METEOR and CIDEr benchmarks.

- *Dropout as a regulariser:* We use dropout regularisation [116] on the video features from the encoder and on the vectorised output of the embedding layer. Dropout discourages neural network parameters from co-adapting towards the optimisation goal which can lead to overfitting. It also efficiently simulates ensemble training where different thinned networks are sampled during different training steps, leading to better performance. It regulates tight coupling between static visual information and semantic information, thus also reducing overfitting.

### 4.2.4 Model Implementation Details

We implement our model using components in the Keras library with Tensorflow as the backend [141]. The 3D CNN encoder's architecture is inspired by the work of Tran et al. [25], but with its parameters set as weights pretrained on the Sports1M video classification dataset [26]. We generate 16 frame video clips, for the all sampling strides, using the OpenCV library [142]. The encoder takes the video clips as input and generates a 4096-dimensional representation which is stored in the serialised `pkl` format. Additionally, we use Keras to implement the decoder, consisting of a dense layer to condense the video representation from 4096 to 256 dimensions, an embedding layer, an LSTM cell, and another dense layer before the final softmax activation. For our embeddings, we either randomly initialise the embedding layer and jointly train with the rest of the decoder parameters, transplant GLOVE embeddings pretrained on the English Wikipedia dataset [139], or transplant and finetune the pretrained GLOVE embeddings with the rest of the decoder parameters.

## 4.3 Evaluation Metrics

Evaluation of generated video captions is a challenging task because there is no specific ground-truth or right answer that can be used as reference for benchmarking accuracy. A video can be correctly described in multiple varied sentences that differ syntatically and semantically. This is particularly the case with the MSVD dataset we use in our study, where multiple ground-truth captions are available for the same video. Several evaluation metrics have been proposed in order to not only tackle this ambiguity, but also ensure that a performant caption generator produces sentences that are highly correlated with human evaluations. For automatic evaluation, three evaluation metrics are adopted from machine translation, namely bilingual evaluation understudy (BLEU) [34], recall oriented understudy for gisting evaluation (ROUGE) [36, 37] and metric for evaluation of translation with explicit ordering (METEOR) [35]. Additionally, consensus based image description evaluation (CIDEr) [38] and semantic propositional image captioning evaluation (SPICE) [143] were proposed fairly recently. These were designed specifically for image captioning and are also being used in video captioning tasks [140]. We briefly discuss each of these and highlight their respective merits and demerits.

### 4.3.1 Bilingual Evaluation Understudy (BLEU)

BLEU [34] is a widely adopted algorithm used to quantify the quality of machine generated text. It measures the correspondence between a candidate sentence and ground-truth sentences. The approach works by counting matching $n$-grams in the candidate sentence and ground-truth sentences. The score varies from 0, where $n$-grams of the candidate sentence do not overlap with any $n$-grams in any of the ground-truth sentences, to 1, where they exactly match ground-truth $n$-grams. A high-scoring sentence should match a ground-truth sentence in length, word choice and word order. BLEU can have many $n$-gram length based variants, e.g. BLEU-1 evaluates unigram matches. In practice BLEU-4 is the $n$-gram length most correlated with human judgements [34]. BLEU is inexpensive to calculate, language independent and has been found to be well correlated with human judgement [140]. However, it does not directly account for recall in its evaluation. Recall is important because it measures the proportion of $n$-grams in the candidate sentence that are in the ground-truth sentences and therefore a measure of how much content was retained [35]. Additionally, it was primarily designed for evaluating translations at a corpus level and therefore not fully suitable for evaluation over single sentences.

### 4.3.2 Recall Oriented Understudy for Gisting Evaluation (ROUGE)

ROUGE [36] is a metric for evaluating text summaries and machine translations. Similar to BLEU, ROUGE is also computed by varying the $n$-gram count, i.e. unigrams, bigrams and higher order $n$-grams. Unlike BLEU which is precision based by matching $n$-grams between candidate and ground-truth sentences, ROUGE also includes a recall measure which measures how much of the ground-truth is recovered or captured by the candidate sentence. For a given a candidate and ground-truth sentence, ROUGE-1 recall which is the measure of overlap for unigrams would be computed as:

$$recall = \frac{n_o}{n_{ref}}, \tag{36}$$

where $n_o$ is the count of overlapping unigrams between the candidate and the ground-truth and $n_{ref}$ is the total number of words in the ground-truth. However, in order to ensure that the candidate sentence is concise, ROUGE has an additional precision term which measures how much of the candidate is relevant or needed to match the ground-truth. Assuming ROUGE-1 again, the precision term is computed as:

$$precision = \frac{n_o}{n_{cand}}, \tag{37}$$

where $n_o$ is the count of overlapping unigrams between the candidate and the ground-truth and $n_{cand}$ is the total number of words in the candidate. For sentences that are both concise and overlap with the ground-truth, the recall and precision terms are combined to produce an F-measure as follows:

$$f = 2 \cdot \left( \frac{precision \cdot recall}{precision + recall} \right). \tag{38}$$

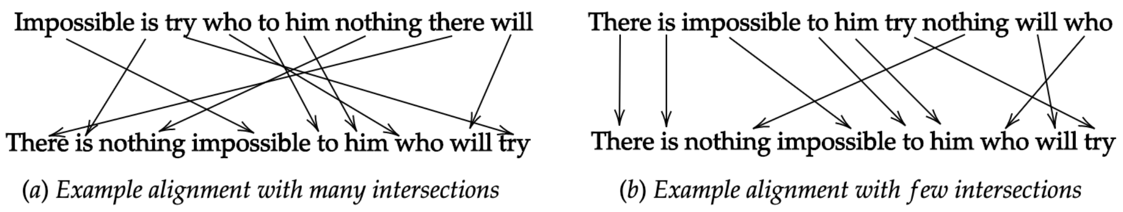

Impossible is try who to him nothing there will
There is nothing impossible to him who will try
(a) Example alignment with many intersections
There is impossible to him try nothing will who
There is nothing impossible to him who will try
(b) Example alignment with few intersections

Figure 24: Illustration of unigram alignments between a candidate sentence and a ground-truth sentence.

ROUGE-L [37] is a variant of ROUGE that is usually used in image and video captioning evaluation. The metric computes the precision and recall scores of the longest common sub-sequence (LCS) between candidate and ground-truth sentences. The intuition with this metric is that the longer the common sub-sequence, the higher the similarity between the candidate and ground-truth sentences. ROUGE-L does not require consecutive matches but rather considers in-sequence matches that reflect sentence level order in $n$-grams. It also automatically includes the longest in-sequence common $n$-grams and therefore, unlike BLEU-$n$ discussed above, predefining $n$-gram length is not required.

### 4.3.3 Metric for Evaluation of Translation with Explicit Ordering (METEOR)

METEOR [35] is designed to alleviate the drawbacks of BLEU of exact lexical matching, by introducing semantic matching. The algorithm leverages WordNet [144], a lexical database for the English language, in order to accomplish multiple matching levels, namely exact token, stemmed token, synonymy and paraphrase matching. The score is computed based on the alignment between a candidate sentence and ground-truth sentences. During the process of alignment, every sentence is taken as set of unigrams with each unigram being mapped to zero or one unigram in a ground-truth sentence. Given one candidate sentence and a ground-truth sentence, if there exists more than one alignment with the same number of mappings, e.g. Figure 24, then the alignment with the smallest number of intersections is chosen, in this case alignment (b).

The final score is a function of unigram recall and precision measures which are calculated independently. Similar to ROUGE [36], the precision measures how many of the unigrams are relevant for a correct candidate sentence, and recall measures how much of the ground-truth's content is retained in the candidate sentence. The unigram precision is calculated as follows:

$$P = \frac{m_o}{m_{cand}}, \tag{39}$$

where $m_o$ is the number of unigrams in the candidate sentence that are found in the ground-truth sentence, and $m_{cand}$ is the number of unigrams in the candidate sentence.

The unigram recall is calculated as follows:

$$R = \frac{m_o}{m_{ref}}, \tag{40}$$

where $m_o$, similar to unigram precision, is the number of unigrams in the candidate sentence that appear in the ground-truth and $m_{ref}$ is the number of unigrams in the ground-truth sentence. The precision and recall are combined using a harmonic mean where the recall is weighted 9 times more than precision:

$$F_{mean} = \frac{10PR}{R + 9P}. \tag{41}$$

However, the $F_{mean}$ score only ensures congruency of individual words between candidate and ground-truth sentences. In order to ensure congruency for longer sentence segments, longer $n$-gram matches are used to compute an alignment penalty. First the candidate sentence's unigrams are grouped into the fewest possible chunks that, when mapped, are adjacent in the ground-truth sentence. The alignment penalty is directly proportional the number of chunks, for example more chunks imply the candidate sentence is fragmented into multiple chunks that are adjacent to corresponding chunks in the ground-truth. The penalty is computed as:

$$p = 0.5 \cdot \left(\frac{c}{U_{map}}\right)^2, \tag{42}$$

where $c$ is the number of chunks and $U_{map}$ is the number unigrams mapped from the candidate to the ground-truth sentence. The METEOR score for a segment is calculated as follows:

$$F_{meteor} = F_{mean} \cdot (1 - p). \tag{43}$$

To compute the score over a whole corpus or multiple sentence segments, the aggregate values of $P$, $R$ and $p$ are combined using the same formula in Equation 47. In the case where there are multiple ground-truth sentences for a given candidate sentence, the highest METEOR score between a candidate and ground-truth sentence is adopted as the final score. METEOR has demonstrated better correlation with human judgements compared to BLEU, and performs better than other contemporary metrics for image captioning [145].

### 4.3.4 Consensus Based Image Description Evaluation (CIDEr)

CIDEr [38] is a metric designed for evaluating image captions based on a measure of consensus between candidate and ground-truth sentences. Given an image, CIDEr evaluates how well a candidate sentence matches the consensus of a set of ground-truth sentences.

The approach first stems off all words in the candidate and ground-truth sentences, for example "work", "worked" and "working" are converted to the root "work". Every sentence is subsequently represented as a set of 1 to 4 length $n$-grams. Consensus encodes how often $n$-grams in the candidate sentence occur in the ground-truth sentences. Conversely, $n$-grams not present in the ground-truth are not expected to be in the candidate. Finally, $n$-grams that commonly occur across all ground-truth sentences for all images in the dataset should be disregarded as they are likely to be less visually informative. In order to encode this intuition, the $n$-grams are assigned weights using the using term frequency inverse document frequency (TF-IDF) [146]. The "term frequency" term assigns large weights for $n$-grams that occur frequently in the ground-truth sentence of an image, whereas the "inverse document frequency" term assigns small weights on $n$-grams that frequently occur across the whole dataset. Given these weights, a candidate sentence $c_i$ and set of ground-truth sentences $S$ for image $i$, the CIDEr$_k$ score for $k$-length $n$-grams is computed using the average cosine similarity between candidate and ground-truth sentences, as follows:

$$CIDEr_n(c_i, S) = \frac{1}{m} \sum_{j=1}^{m} \frac{\mathbf{g^k}(c_i) \cdot \mathbf{g^k}(s_j)}{\|\mathbf{g^k}(c_i)\| \|\mathbf{g^k}(s_j)\|}, \tag{44}$$

where $\mathbf{g^k}(c_i)$ and $\mathbf{g^k}(s_j)$ are vectors formed by $g^k(c_i)$ and $g^k(s_i)$, corresponding to weights encoding the frequency of $k$-length $n$-grams in the candidate and ground-truth sentences, respectively. $m$ is the number of ground-truth sentences describing the image $i$. In order to capture the grammatical properties and richer semantics of the text, CIDEr uses higher order $n$-grams in its final score by combining scores from $n$-grams of varying length as follows:

$$CIDEr(c_i, S) = \sum_{n=1}^{N} w_n CIDEr_n(c_i, S), \tag{45}$$

where $N = 4$ and $w_n$ being uniformly $\frac{1}{N}$ having been found to work well [38].

### 4.3.5 Semantic Propositional Image Captioning Evaluation (SPICE)

SPICE [143] is the most recently proposed metric for image and video captioning. The approach is based on decomposing the candidate sentence into a scene graph and then grouping the nodes into tuples of the candidate and ground-truth sentences. This intermediate representation of the candidate and ground-truth sentences encodes the semantic propositional content they are describing. The scene graph is used to parse the candidate sentence into semantic tokens such as object classes, relation types and attribute types. For example a SPICE scene graph parses a candidate sentence, $c$, into a scene graph tuple using the following:

$$G(c) = [O(c), E(c), K(c)], \tag{46}$$

where $G(c)$ denotes the scene graph, $O(c)$ is the set of objects, $E(c)$ denotes relationships between objects and $K(c)$ denotes objects' attributes. Using these tuples, recall and precision measures are calculated and the final score for SPICE is calculated using a similar F-measure defined for ROUGE in Equation 38. For our study we choose to evaluate video captions using BLEU, METEOR, CIDEr and ROUGE metrics because of their complementary qualities. While BLEU is widely adopted in literature, we noted its deficiencies that are alleviated by taking recall into account using ROUGE. METEOR extends precision and recall beyond individual words into longer sentence segments while introduction synonym matching. We incorporated CIDEr because it is one of the metrics specifically designed for the captioning task unlike BLEU and METEOR that are adopted from machine translation. We do not include SPICE in our experiments for practical reasons because we found computing and caching the scene graphs for the ground-truth sentences to be too expensive on our limited compute resources.

# 5 Experiments and Results

Generic image descriptors, learnt in image classification tasks, have been used effectively in other related tasks e.g. object detection [7, 8, 42, 43], semantic segmentation and image caption generation [9, 10]. In the same vein, generic video descriptors have been developed using spatiotemporal convolutions (3D CNNs) for video classification [25, 26] and pose estimation [147]. Using a generic video descriptor trained on a video classification task [26], we train a video caption generator with multiple video features obtained using temporal sampling. We perform the temporal sampling with a constant temporal stride between sampled frames in a video. Since our encoder-decoder model is pretrained to sample 16 frames for every video, following the approach by Tran et al. [25], we devise experiments to determine how temporal sampling affects the quality of the generated captions. In this chapter we discuss the experiments and results of our encoder-decoder architecture using pretrained and jointly trained semantic embeddings.

## 5.1 Experimental Setup

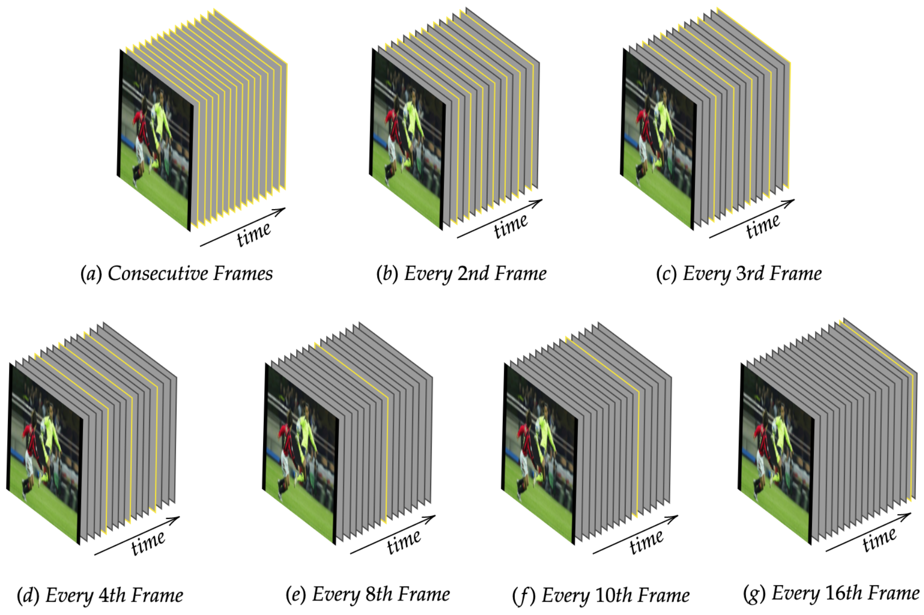


Figure 25: Illustration of sampling stride used during experiments. The caption generator is separately trained with video features resulting from every sampling configuration.

For the entire video dataset, we sample video features that will be used as visual features in our caption generator. The sampling configurations are as illustrated in Figure 25. We use video features in the validation and test sets for evaluating accuracy and generalisation. We outline the steps taken to convert the input videos into video features below.

- We extract video volumes that will be used to condition the caption generation, from the MSVD open domain video dataset [27]. For every video we create 7 separate video volumes, by sampling 16 frames with a temporal stride of $i$, where $i \in \{1, 2, 3, 4, 8, 10, 16\}$. For example, for $i = 2$, we sample every second frame until we have sampled 16 frames. $\mathcal{D}$ is $3 \times 16 \times 171 \times 128$, the dimensions of the video volume; 3 is the RGB colour dimension, 16 is the number of frames and $171 \times 128$ is the spatial dimension of every frame.

- We also generate a set of video features corresponding to every video volume. Each of these features is a 4096-dimensional activation from the 3D CNN encoder, which is pretrained on the Sports1M dataset with input dimension $\mathcal{D}$ [26]. The features are associated with a list of human generated captions linked to the videos, and we train the caption generator using the (*video feature*, *captions*) pairs. Every caption's words are vectorised through the embedding layer and passed into the LSTM unit. We concatenate the hidden state $h_t$ with the video feature via element-wise addition before the final word prediction [70].

- We train the decoder by minimising the categorical cross-entropy loss on predicting the corresponding ground-truth word, at every time step. The loss function is as follows:

$$J = -\sum_{i=1}^{W} y_i log(f(y)_i), \tag{47}$$

  where $y_i$ is the ground truth in the vocabulary of size $W$, and $f(y)_i$ is the softmax classification score of the target word $y_i$:

$$f(y)_i = \frac{e^{s_{y_i}}}{\sum_{j=1}^{W} e^{s_{y_j}}} \tag{48}$$

For every sampling stride $i \in \{2, 3, 4, 8, 10, 16\}$, we train a decoder for 100 epochs using a single Nvidia GTX 1060 GPU. One epoch represents a time period in which the model has observed every (*video feature*, *captions*) in the training set. After observing the model performance over multiple experiments, we include early stopping to speed up experimentation. Early stopping terminates training when the model's validation loss does not improve for a preset number of epochs. This not only helps to speed up experimentation but prevents the model from overfitting on the training set which will regress performance on the test set.

## 5.2 Dataset

We use the Microsoft Video Description (MSVD) [27] open doman video dataset for all our experiments. The dataset consists of 1970 variable length videos, each between 10 and 60 seconds long, drawn from diverse categories including sports, cooking, animals, movie clips and music. Each video depicts a single, fairly unambiguous action or event. Additionally, each video is annotated with about 41 natural language descriptions, across 35 languages including English, German, Hindi, Tamil and Mandarin. In addition to being useful for video caption generation, the dataset effectively provides parallel translations which are useful in machine translation tasks. For simplicity in our experiments, we utilise only the English captions. Due to the variable length and disparate quality of videos, we studied some descriptive statistics of the dataset in order to inform our choices for frame sampling and facilitate fair comparison between high framerate and low framerate videos. The statistics are given in Table 1.

| Statistic | Frame count | Duration (seconds) | Frames per second |
|---|---|---|---|
| Min | 41 | 1.74 | 6.00 |
| Max | 1799 | 60.03 | 60.00 |
| Mean | 275 | 9.65 | 28.98 |
| Median | 240 | 8.02 | 29.95 |
| Standard deviation | 192 | 6.18 | 8.56 |
| 25th percentile | 150 | 6.01 | 25.00 |
| 50th percentile | 240 | 8.02 | 29.95 |
| 75th percentile | 326 | 11.01 | 29.98 |

Table 1: Descriptive statistics in the MSVD dataset.

When preprocessing every video, we sample 16 frames due to the input dimension $\mathcal{D} = 3 \times 16 \times 171 \times 128$ upon which our 3D CNN is pretrained. This means that in some of our sampling configurations (shown in Figure 25), some videos might not have sufficiently many frames to add up to 16. In such cases we pad the volume with all zero frames until we have 16 frames. Therefore large strides can discriminate against videos with low frame counts or frame rates. However, there is an inherent trade-off between large strides that cover longer portions of a video, versus shorter strides that cover shorter portions of the video but ensure more image frames are likely to be sampled for the video's resulting visual feature. For example, 41 is the lowest frame count and therefore 1 or 2 stride configurations would ensure that for all videos, there will be enough image frames to sample. On the other hand, when using the 16 stride configuration, about 41% of the videos have 256 or more frames in order to avoid the zero padding. This trade-off is balanced by the 8 and 10 stride configurations which simultaneously cover longer portions of videos while ensuring that most videos have enough frames to sample, as illustrated in Figure 26.

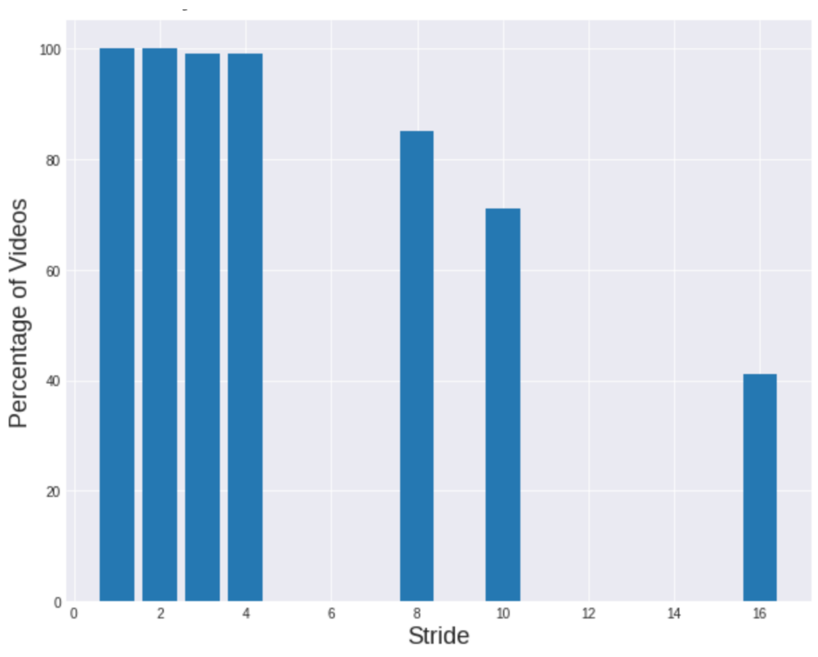


Figure 26: Comparison between the stride configurations and the percentage of videos with enough frames to sample.

We split our data into 70% for training, 20% for validation and 10% for the testing. We also preprocess the associated captions by first stripping all punctuation and building a vocabulary of 10452 most frequently occurring words. Every word is assigned a uniquely identifiable integer index, which the decoder and softmax classifier will attempt to predict through the assigning of highest probability. The predicted index is then used to look up the associated word, adding to the natural language description. Therefore the embedding matrix that encodes the semantic representation of every word in the vocabulary, is of dimension $256 \times 10452$.

## 5.3 Quantitative Results

We train the video caption generator using the seven different sampling configurations outlined in the Section 5.1. For every one of these configurations, we evaluate the resulting model's caption generation capability using the BLEU, METEOR, CIDEr and ROUGE-L metrics [34, 35, 37, 38] that we discussed in the previous chapter. In addition to these sampling configurations, we train our models using jointly trained embeddings, pretrained GLOVE embeddings [31] and finetune GLOVE embeddings [31] to determine which of those embedding configurations might result in the best quality captions.

### 5.3.1 Jointly Training Word Embeddings

In this configuration, the embedding layer of the caption generator is randomly initialised as a trainable set of parameters. During backpropagation through time, the embedding matrix is updated along with the parameters of the LSTM unit. In particular, only the embeddings for the words that were predicted are updated instead of the entire embedding matrix, since they are responsible for the evaluated loss. In this experiment, the embedding layer only learns the context and semantics of the words that appear in the vocabulary of the ground-truth captions. With this model, we present the results of evaluating caption generation quality, according to the aforementioned benchmarks, in Table 2 and Figure 27.

| Stride | BLUE-1 | BLEU-2 | BLEU-3 | BLEU-4 | METEOR | ROGUE-L | CIDEr |
|---|---|---|---|---|---|---|---|
| 1 | 0.621 | 0.481 | 0.331 | 0.208 | 0.239 | 0.529 | 0.297 |
| 2 | 0.614 | 0.461 | 0.298 | 0.189 | 0.226 | 0.526 | 0.257 |
| 3 | 0.620 | 0.469 | 0.317 | 0.190 | 0.233 | 0.541 | 0.275 |
| 4 | **0.661** | **0.513** | **0.353** | 0.218 | **0.251** | **0.549** | 0.318 |
| 8 | 0.629 | 0.482 | 0.332 | **0.219** | 0.244 | 0.545 | **0.320** |
| 10 | 0.641 | 0.502 | 0.340 | 0.212 | 0.250 | 0.539 | 0.280 |
| 16 | 0.636 | 0.474 | 0.320 | 0.212 | 0.242 | 0.540 | 0.291 |

Table 2: Experimental test results using jointly trained embeddings.

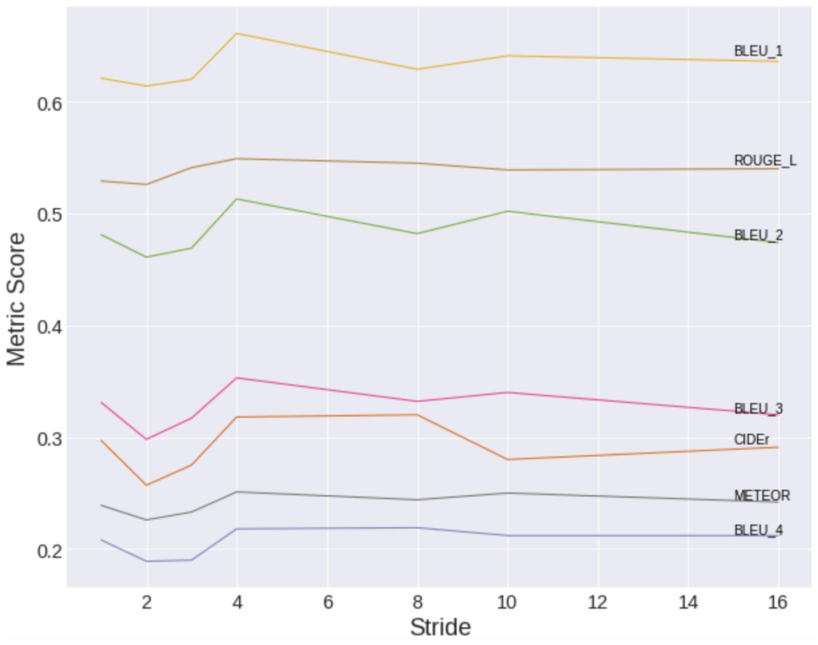


Figure 27: Illustration of experimental test results using jointly trained embeddings.

From our experiments, we obtain the best caption quality for the 4 and 8 strides that cover relatively long portions of the video. However the performance diminishes slightly for large strides because video volumes are padded with more all zero frames, which translates to loss of visual data that is otherwise useful for caption generation. Although the jointly trained embeddings are closely optimised for the caption generation task, they are fairly limited to a small vocabulary of 10452 words which can lead to the caption generator reusing more captions from the training data. In pursuit of more diverse captions, we introduce pretrained GLOVE embeddings that are based on a much larger word corpus. We elaborate on this addition in the following section.

### 5.3.2 Transplanting Pretrained Word Embeddings

In this configuration, the embedding layer and the LSTM unit are jointly trained similar to the previous section. However, at evaluation time, we replace the embedding layer with GLOVE word embeddings pretrained on a large Wikipedia dataset [31,61]. GLOVE is an unsupervised learning algorithm that produces vector representations of words by using co-occurrence statistics of words in a large corpus [148]. We first initialise an embedding matrix of zeros, then from the GLOVE embeddings we select the set of embeddings corresponding to the words in the vocabulary of the ground-truth captions [139]. Each selected word embedding is a 200-element vector that encodes the associated word's semantic meaning, learnt during pretraining. To conform with the $256 \times 10452$ dimension of the embedding layer, we zero pad every 200-element vector with 56 additional elements to be used with a $256 \times 10452$ matrix that we set as weights for the embedding layer.

The intuition with this strategy is that word representations from a pretrained embedding encapsulate much richer semantics since they are trained on a much larger dataset and therefore encounter many more word co-occurrences across more contexts, as opposed to the limited set of ground-truth captions. However, this means that words that appear in the captions but not in the GLOVE embeddings will have a 256-vector of zeros and will remain without any semantic meaning encoded in the embedding matrix's geometric space. However, we found that this only affected words that were misspelled in the vocabulary. We report results of running the aforementioned benchmarks using pretrained embeddings in Table 3 and Figure 28.

| Stride | BLUE-1 | BLEU-2 | BLEU-3 | BLEU-4 | METEOR | ROGUE-L | CIDEr |
|---|---|---|---|---|---|---|---|
| 1 | **0.231** | **0.087** | **0.018** | 0.00 | **0.108** | **0.374** | **0.075** |
| 2 | 0.056 | 0.020 | 0.004 | 0.00 | 0.069 | 0.196 | 0.041 |
| 3 | 0.139 | 0.041 | 0.013 | 0.00 | 0.099 | 0.303 | 0.072 |
| 4 | 0.094 | 0.018 | 0.000 | 0.00 | 0.082 | 0.288 | 0.068 |
| 8 | 0.129 | 0.033 | 0.006 | 0.00 | 0.075 | 0.265 | 0.056 |
| 10 | 0.179 | 0.053 | 0.009 | 0.00 | 0.082 | 0.308 | 0.053 |
| 16 | 0.189 | 0.055 | 0.000 | 0.00 | 0.088 | 0.330 | 0.060 |

Table 3: Experimental test results using pretrained GLOVE embeddings.

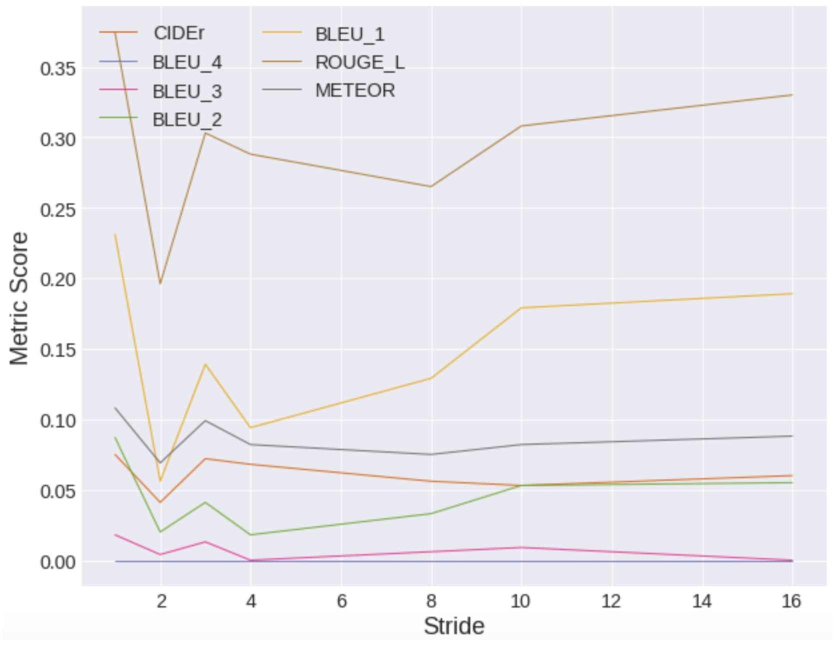


Figure 28: Illustration of experimental test results using GLOVE embeddings.

Across all metrics, we obtain the best caption quality when using the shortest stride. Notably the overall scores are significantly worse than when we use a jointly trained embedding layer. In particular, the BLEU score deteriorates significantly to 0 for BLEU-4. BLEU-4 is the most challenging BLEU metric in our experiments, requiring the model to generate a caption that retains at least four ordered words or quad-grams from a ground-truth caption. In this case the generated captions have no quad-grams that overlap with any of the ground-truth captions, and our qualitative results show that the model generates unintelligible captions. An explanation for this result is that, although pretrained embeddings originate from a large word corpus, they are trained on a different learning task that is not necessarily optimised for caption generation. Therefore simply using these embeddings without finetuning is similar to randomly initialising the embedding matrix and freezing the embedding weights. Another explanation for the markedly worse performance is that the particularly important words `startseq` and `endseq`, that prime the decoder to start and stop generating sentences, are not present in the GLOVE embeddings and therefore are missing vital co-occurrence context that they would otherwise obtain when jointly trained with the rest of the decoder's weights. We contrast the difference in semantic encoding between the pretrained embeddings and the jointly trained embeddings, where the first two vectors of the $256 \times 10452$ matrix represent the `startseq` and `endseq` tokens in Listing 9.

```
# (2 X 256) startseq and endseq representations using
# pretrained embeddings
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0. ...],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0. ...]])

# (2 X 256) startseq and endseq representations using
# jointly trained embeddings
array([[ 0.02508733,  0.04892572,  0.04317002,  0.02298352, -0.03714135,
        -0.00219072, -0.04738931,  0.03398378, -0.0405638 ,  0.00202782,
         0.00091969,  0.00147787, -0.01984164, ...],
       [ 0.03953084, -0.04639492, -0.03263773,  0.01540521, -0.00485314,
         0.11522665, -0.08679304,  0.09251759, ...]])
```

Listing 9: Comparison of embeddings for the `startseq` and `endseq` words when using pretrained embeddings versus a jointly trained embedding layer. The embeddings for `startseq` and `endseq` are set to zero vectors in the GLOVE embeddings, versus non-zero vectors in jointly trained embeddings.

For example given `startseq` at $t = 0$, $x_0$ is a $1 \times 256$ vector of all zeros that interacts with the LSTM unit's input, forget and output gates as follows:

$$\begin{aligned} c_1 &= f(W_f, x_0, h_0) \odot c_0 + i(W_i, x_0, h_0) \odot g(W_g, x_0, h_0) \\ h_1 &= o(W_o, x_0, h_0) \odot \tanh(c_1) \end{aligned} \tag{49}$$

$x_0$ is concatenated with $h_0$, via an addition operation, then multiplied by the input, forget and output gates' weights. $x_0$ is not co-adapted with the hidden state $h_0$ or cell state $c_0$ and thus adds no additional semantic context. Consequently this inexpressive interaction does not appropriately influence the LSTM unit to generate a sensible word to follow `startseq`. In the case of jointly trained embeddings, the `startseq` and `endseq` embeddings are randomly initialised, and then updated during backprogation through time, as illustrated by the non-zero embedding vectors in Listing 9. Notably, only the `endseq` embedding is updated because its occurrence depends on the variable length of ground-truth captions. This can lead to the example predicted and ground-truth captions in Listing 10.

```
    # Predicted caption
    "startseq Two men are playing a piano endseq"
    # Ground-truth caption
    "startseq Two men are playing a black piano endseq"
```

Listing 10: Example of a predicted and ground-truth caption.

Cross-entropy loss will be registered between the predicted subsequence `"piano endseq"` versus the expected subsequence `"black piano endseq"`, which leads to an update of the `endseq`'s embedding to discourage the co-occurrence of the predicted subsequence while encouraging that of the ground-truth subsequence, when given the caption prefix. Since there are several ground-truth examples of where the `endseq` word should occur given the same video, its resulting embedding encodes a sort of average that most likely results in a predicted caption that exactly matches any ground-truth caption. In contrast, `startseq` is always observed at the beginning of both predicted and ground-truth captions, hence registering zero loss and thus not needing an update.

Besides `startseq` and `endseq`, we also observed that 1884 words from the 10452 vocabulary where missing in the GLOVE embeddings and therefore remained with no meaningful semantic representation. However, almost all of these words turned out to be mispellings in the captions that would otherwise result in unintelligible captions in our qualitative results. Overall the qualitative results of using the pretrained embeddings configuration were also worse, with the decoder generating significantly longer and non-sensible sentences because of missing co-occurrence context for the `startseq` and `endseq` words. With this result in mind we noted the need to not only incorporate pretrained embeddings that were trained on a large corpus, but to optimise these embeddings specifically for the caption generation task.

### 5.3.3 Finetuned Word Embeddings

In the previous section we demonstrated that replacing the embedding layer's matrix with pretrained GLOVE embeddings results in poor caption quality. While pretrained embeddings theoretically encapsulate more semantic meaning, they are likely to bias the caption generator towards generating word sequences that reflect the co-occurrences encountered in the large Wikipedia corpus. Moreover, the words encountered in the captions but not in the pretrained embeddings are left devoid of co-occurrence context that is vital for generating sensible captions, resulting in poor performance as was demonstrated in the previous section. Since the pretrained embeddings are trained on a different task and therefore not necessarily optimised for the caption generation task, we finetune the embeddings for caption generation. To achieve this, we prime the embedding layer with the pretrained GLOVE embeddings and then train the caption generator in order to not only optimise the embeddings for captioning but also fill in the missing context of words that were not present in the Wikipedia corpus upon which the GLOVE embeddings were trained. This configuration is similar to jointly training the embedding layer from scratch, with the only difference being in the initialisation. In Listing 11, we compare the embeddings for `startseq` and `endseq` words from pretrained GLOVE embeddings versus finetuned GLOVE embeddings, after training.

```
# (2 X 256) startseq and endseq representations using
# pretrained embeddings
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0. ...],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0. ...]])

# (2 X 256) startseq and endseq representations using
# after finetuning pretrained embeddings
array([[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00, ...],
       [ 2.90071443e-02, -3.23250405e-02,  3.09367999e-02,
        -3.57646942e-02,  7.82455876e-02, -7.57628866e-03,
         6.80529401e-02, -1.07432917e-01,  3.32835279e-02,
        -7.29207993e-02, ...]])
```

Listing 11: Comparison of embeddings for the `startseq` and `endseq` words when using pretrained embeddings versus finetuned embeddings.

As we established earlier, the embedding for `startseq` remains the same since it is always observed at the same position of every ground-truth caption versus a generated caption, and therefore generates no cross-entropy loss during training. However the `endseq`'s embedding is jointly optimised with other decoder parameters for the caption generation task. The result is more sensible captions that are more or less the length of those found in the ground-truth. We present the results of the caption generator using finetuned embeddings in Table 4 and Figure 28.

| Stride | BLUE-1 | BLEU-2 | BLEU-3 | BLEU-4 | METEOR | ROGUE-L | CIDEr |
|---|---|---|---|---|---|---|---|
| 1 | 0.624 | 0.479 | 0.334 | 0.222 | 0.240 | 0.540 | **0.321** |
| 2 | 0.617 | 0.475 | 0.321 | 0.216 | 0.241 | 0.535 | 0.300 |
| 3 | 0.527 | 0.374 | 0.244 | 0.155 | 0.221 | 0.526 | 0.287 |
| 4 | 0.579 | 0.435 | 0.289 | 0.179 | 0.240 | 0.537 | 0.249 |
| 8 | 0.566 | 0.434 | 0.290 | 0.185 | 0.234 | 0.528 | 0.296 |
| 10 | **0.650** | **0.502** | 0.343 | 0.228 | **0.248** | **0.560** | 0.301 |
| 16 | 0.632 | 0.498 | **0.350** | **0.229** | 0.245 | 0.548 | 0.315 |

Table 4: Experimental test results using finetuned GLOVE embeddings.

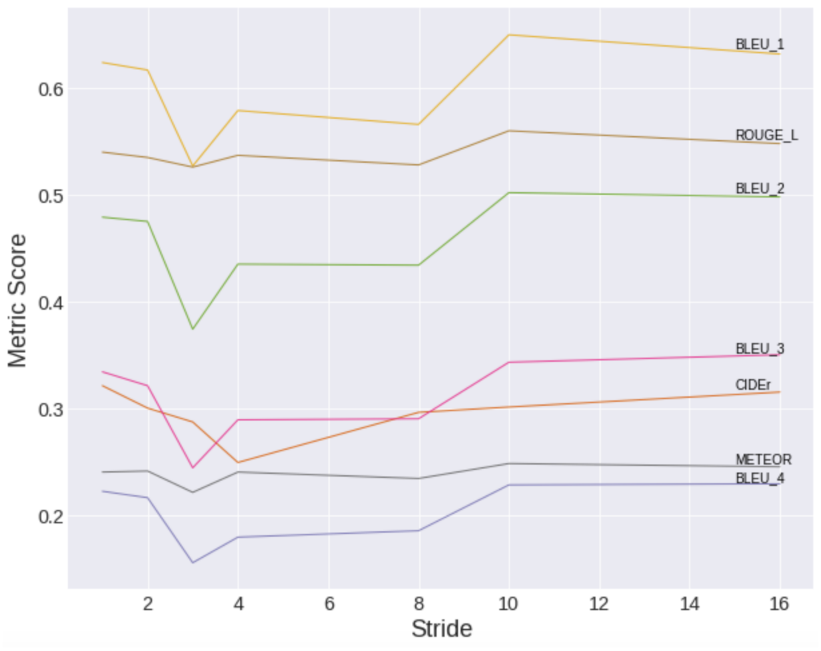


Figure 29: Illustration of experimental test results using finetuned GLOVE embeddings.

Similar to experimental results when using jointly trained embeddings, we find that sparsely sampling frames generally performs better since a longer portion of the video is included. In this case we find that the 10 and 16 strides balance the trade-off of the largest stride while ensuring to have enough non-zero video frames to influence the video captioning. Additionally, randomly initialising and jointly training the embedding layer, and finetuning GLOVE embeddings are the best performing strategies, although there is no clear winner among the two. Table 5 illustrates the best scores on the aforementioned benchmarks for the two strategies.

| Strategy | BLUE-1 | BLEU-2 | BLEU-3 | BLEU-4 | METEOR | ROGUE-L | CIDEr |
|---|---|---|---|---|---|---|---|
| J.T. | **0.661** | **0.513** | **0.353** | 0.219 | **0.251** | 0.549 | 0.320 |
| F.G. | 0.650 | 0.502 | 0.350 | **0.229** | 0.248 | **0.560** | **0.321** |

Table 5: Experimental test results using jointly trained embeddings (J.T.) versus finetuned GLOVE embeddings (F.G.).

## 5.4 Qualitative Results

Next we present examples of generated captions obtained through the three embedding configurations, each using the model checkpoint with the best performing stride. We selected a set of videos for which we generate captions with all the embedding configurations, and qualitatively divide results into good, average and bad. We use the jointly trained embedding configuration's results as a benchmark against the other two configurations that utilise pretrained embeddings. For the jointly trained and finetuned embedding configurations, we report and discuss both positive and negative qualitative results using the same set of videos. For the pretrained embeddings, we present results that substantiate the poor performance we noted in the quantitative results.

### 5.4.1 Jointly Trained Word Embeddings

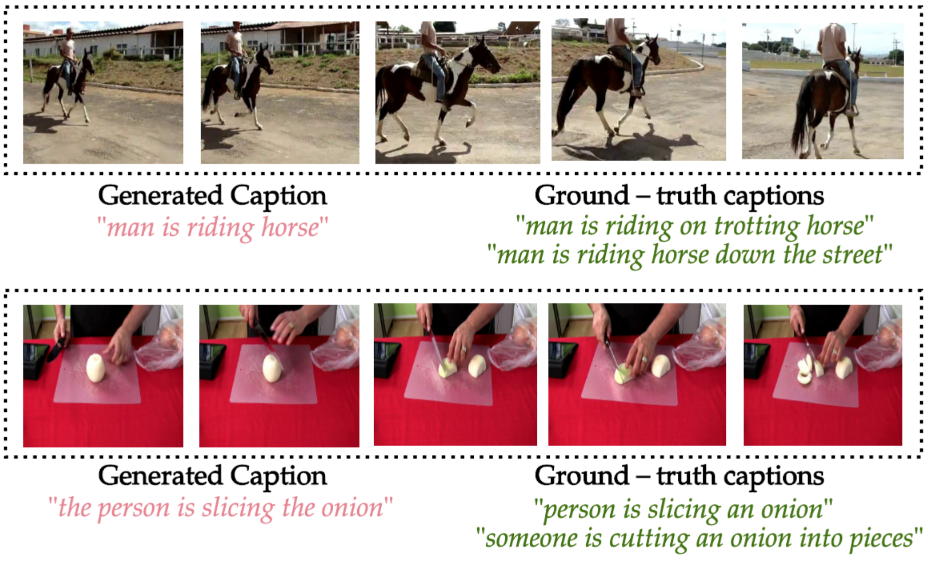

Generated Caption
"man is riding horse"
Ground – truth captions
"man is riding on trotting horse"
"man is riding horse down the street"
Generated Caption
"the person is slicing the onion"
Ground – truth captions
"person is slicing an onion"
"someone is cutting an onion into pieces"

Figure 30: Experimental test results from using jointly trained embeddings, showing good caption predictions.

Figure 30 illustrates good captions generated using the model checkpoint trained with randomly initialised and jointly trained embeddings. The model generates sensible captions that adequately capture the event or action in the video. Moreover the generated captions overlap with some that are found in the training dataset.

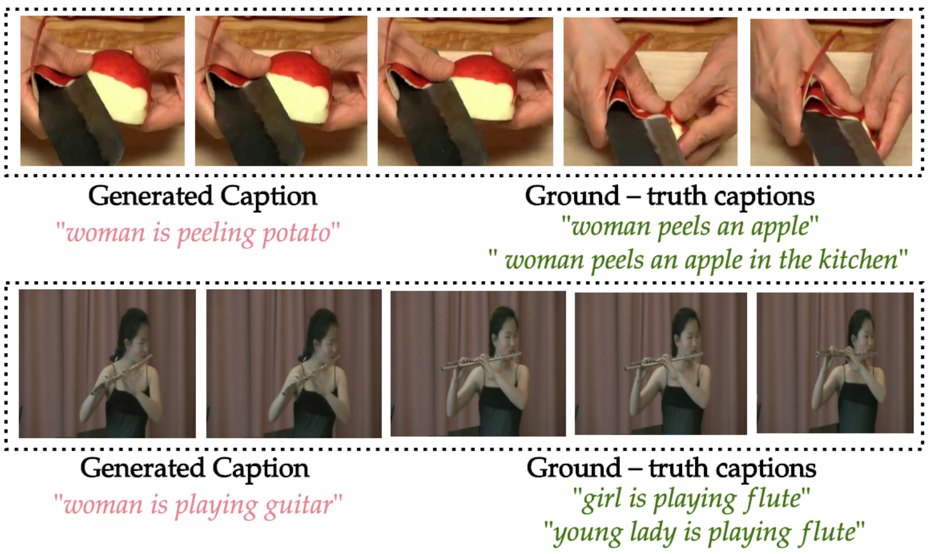


Figure 31: Experimental test results from using jointly trained embeddings, showing average caption predictions.

Figure 31 illustrates average captions using the same model checkpoint. We define average predictions as instances where the model is able to recognise an action or event but fails to identify the object or subject. In the first video, the model correctly identifies "peeling", but incorrectly identifies an apple as a potato. Similarly, for the second video, the model identifies the "playing" of an instrument but states the wrong instrument.

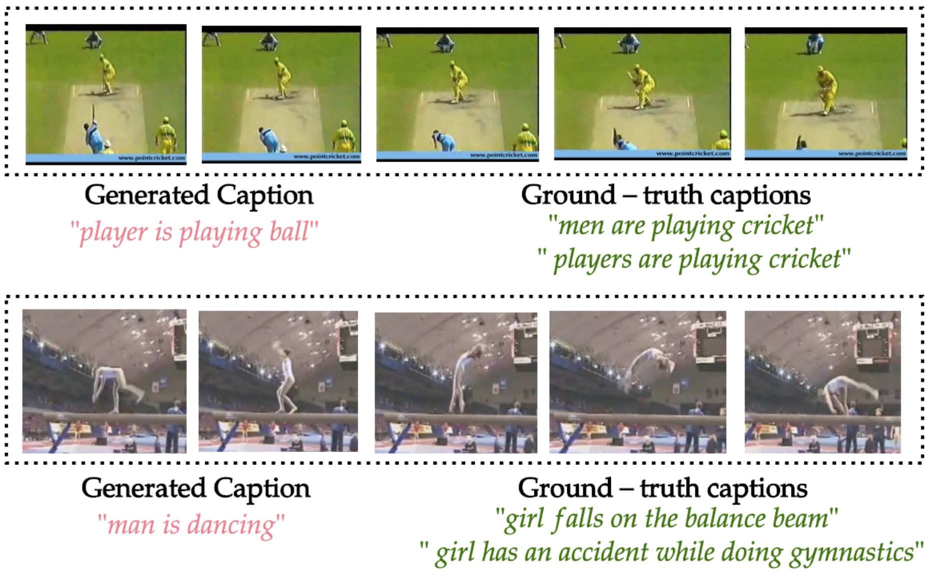

Generated Caption
"player is playing ball"
Ground – truth captions
"men are playing cricket"
" players are playing cricket"
Generated Caption
"man is dancing"
Ground – truth captions
"girl falls on the balance beam"
" girl has an accident while doing gymnastics"

Figure 32: Experimental test results from using jointly trained embeddings, showing bad caption predictions.

Figure 32 illustrates bad captions using the same model checkpoint. We define bad predictions as instances where the model fails to identify the action, objects and subjects in the video. In the first video the model is able to understand the event as a kind of sport, but is unable to identify cricket or the multiple players. Similarly, in the second video, the model incorrectly identifies a woman as a man and misidentifies gymanstics as dancing.

### 5.4.2 Finetuned Word Embeddings

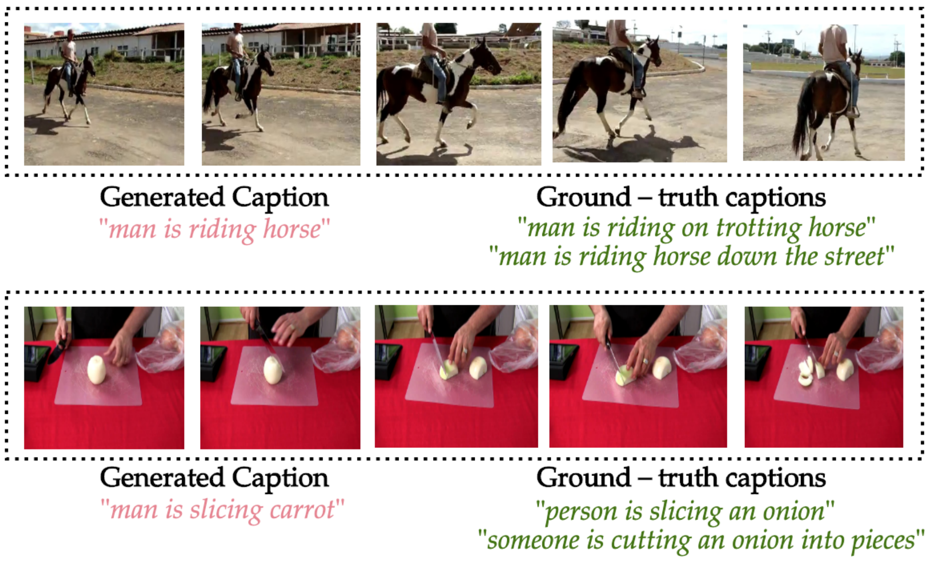

Generated Caption
"man is riding horse"
Ground – truth captions
"man is riding on trotting horse"
"man is riding horse down the street"
Generated Caption
"man is slicing carrot"
Ground – truth captions
"person is slicing an onion"
"someone is cutting an onion into pieces"

Figure 33: Experimental test results from using finetuned embeddings, showing good caption predictions.

Figure 33 illustrates captions generated for the same videos used as a benchmark for good captions in the jointly trained embeddings configuration. The model correctly captions the first video, it correctly identifies the "slicing" action in the second video but misidentifies the onion as a carrot. These experimental results show that the jointly trained embedding configuration can perform slightly better than finetuned embeddings.

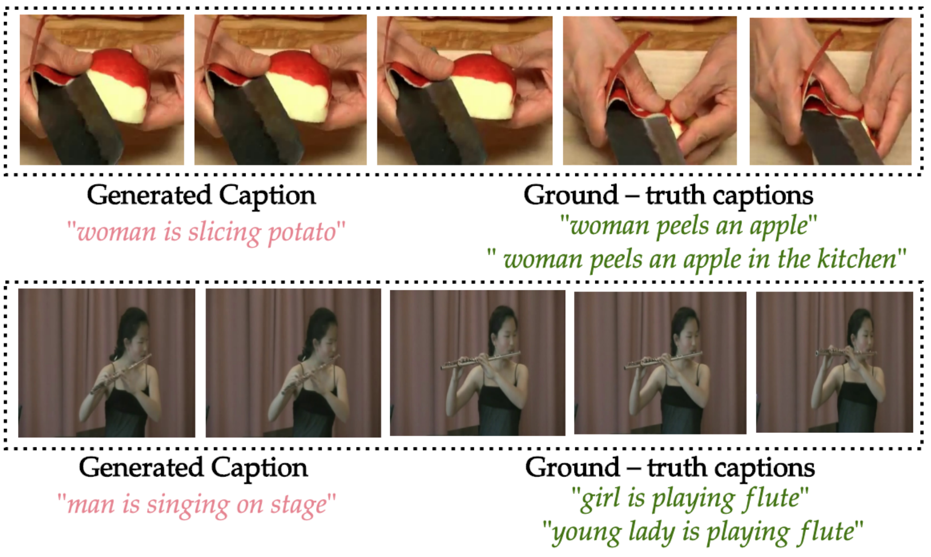


Figure 34: Experimental test results from using finetuned embeddings, showing average caption predictions.

Figure 34 illustrates captions generated for videos used as a benchmark for average captions in the jointly trained embeddings configuration. The model is able to identify the subject and "slicing" action in the first video, although it misidentifies the object. In the second video, it neither identifies the action, subject nor object. For these videos, the jointly trained embeddings configuration performs better than finetuned embeddings.

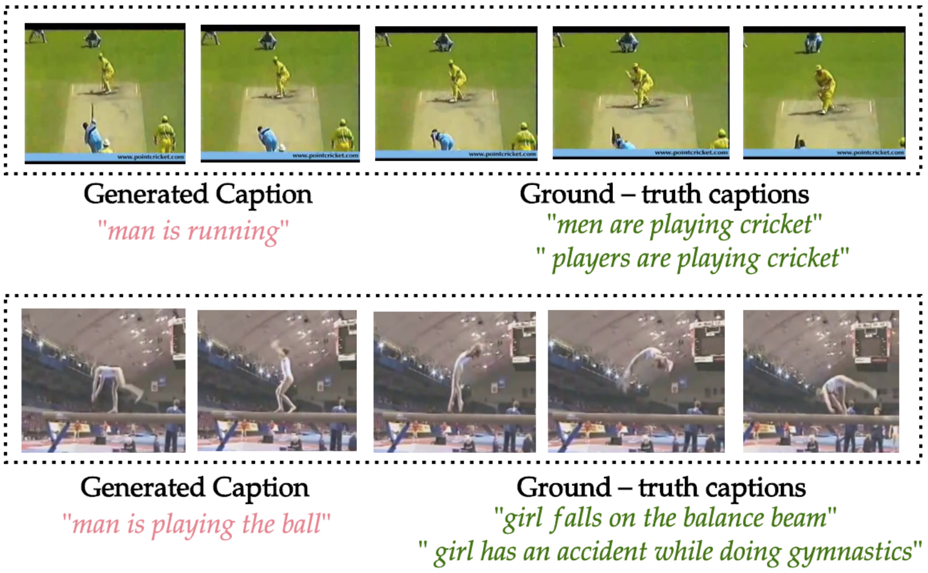

Generated Caption
"man is running"
Ground – truth captions
"men are playing cricket"
" players are playing cricket"
Generated Caption
"man is playing the ball"
Ground – truth captions
"girl falls on the balance beam"
" girl has an accident while doing gymnastics"

Figure 35: Experimental test results from using finetuned embeddings, showing bad caption predictions.

Figure 35 illustrates captions generated for videos used as a benchmark for bad captions in the jointly trained embeddings configuration. Similar to the jointly trained embedding model, this model performs poorly on the given videos. In the first video, it generally understands the "running" but not the overall context of the cricket match within which it happens. For the second video it fails to identify the action, subject and object.

### 5.4.3 Pretrained Word Embeddings

From our quantitative results, we noted the significant regression in performance when comparing caption quality between jointly trained embeddings and transplanting pretrained GLOVE embeddings without finetuning. We further qualified this result by noting that the pretrained embeddings are not optimised for the caption generation task. In this section we illustrate how the qualititative results are equally poor when using this embedding configuration against all the benchmark videos. In all but the first video of Figure 36, the model generates non-sensible captions that poorly describe the events in the videos.

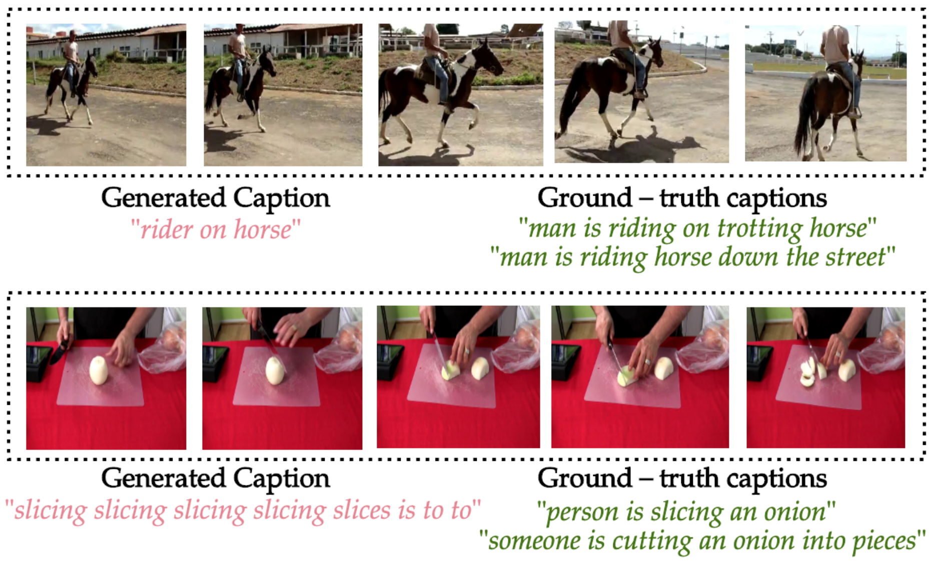

Generated Caption
"rider on horse"
Ground – truth captions
"man is riding on trotting horse"
"man is riding horse down the street"
Generated Caption
"slicing slicing slicing slicing slices is to to"
Ground – truth captions
"person is slicing an onion"
"someone is cutting an onion into pieces"

Figure 36: Experimental test results from using pretrained embeddings.

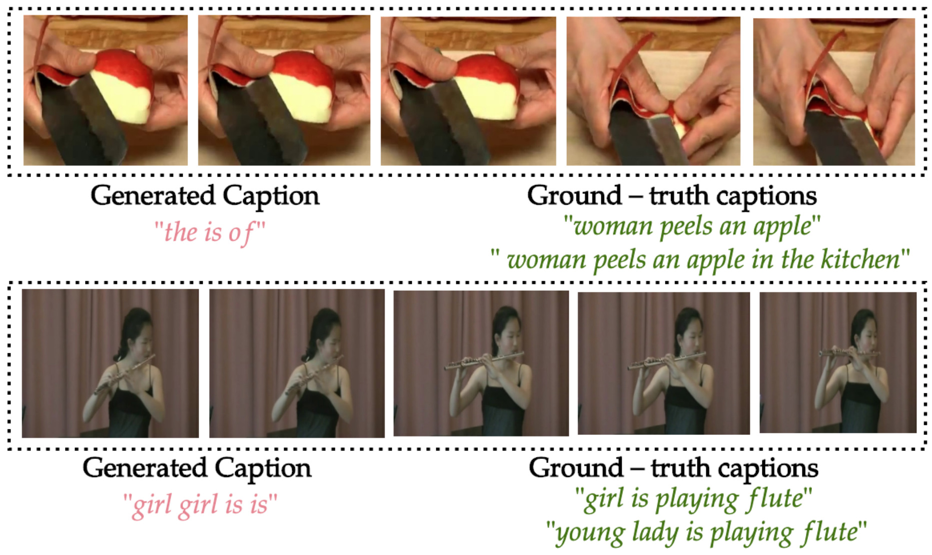


Figure 37: Experimental test results from using pretrained embeddings.

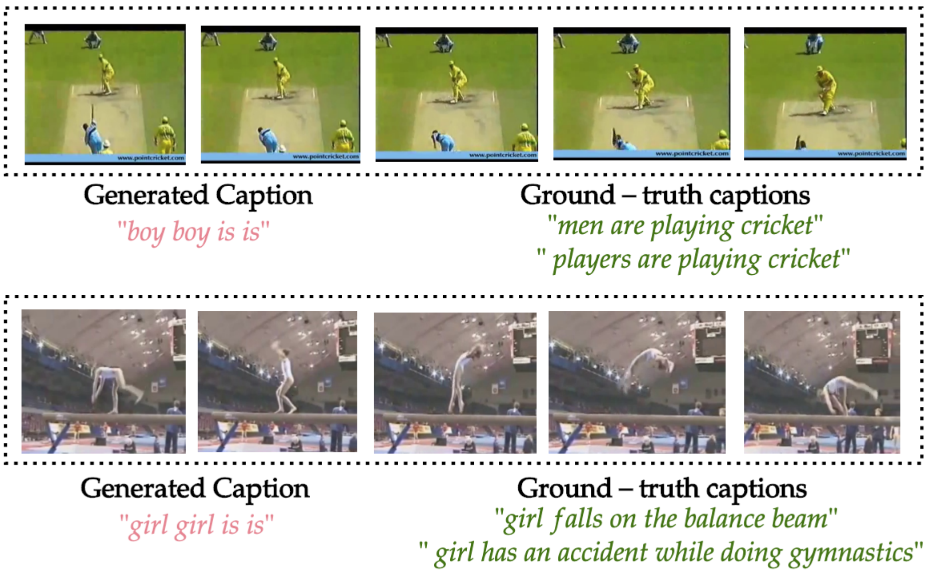

Generated Caption
"boy boy is is"
Ground – truth captions
"men are playing cricket"
" players are playing cricket"
Generated Caption
"girl girl is is"
Ground – truth captions
"girl falls on the balance beam"
" girl has an accident while doing gymnastics"

Figure 38: Experimental test results from using pretrained embeddings.

## 5.5 Discussion

In this chapter we elaborated on the experiments conducted and the results obtained when trying to determine the most efficient way to sample video frames in order to adequately describe its content. We trained several model configurations using different sampling strides. In addition to these stride configurations, we trained the models on three word embedding configurations, namely jointly trained, pretrained and finetuned embeddings. Our results generally showed that sparsely sampling frames results in better caption generation quality since a longer portion of the video is covered and therefore a longer portion of the event in question is captured. However there is an inherent trade-off between using a short stride that captures a short portion of the video versus a longer stride which potentially discriminates against short videos which will result in inadequate video volumes being augemented by zero value frames. The statistics of the data showed that using 8 or 10 strides simultaneously covers longer portions of videos while ensuring that most videos have enough image frames to build a feature rich presentation. Equally important are our findings from experiments on the viability of word embeddings. We found that randomly initialising and jointly training word embeddings, using the vocabulary of the ground-truth captions in the training set, is effective for generating sensible captions at test time. We introduced pretrained GLOVE embeddings and found that they significantly regress the performance if they are not finetuned for the aption generation task, having been trained on a different task. Upon finetuning, we found them to be more or less equally effective as compared to the jointly trained embeddings. Our results show that pretrained embeddings can be a good initialisation upon which further task-specific optimisation can be done.

# 6 Conclusion and Future Work

Given the ubiquity of video data being continuously generated on multimedia platforms, automatic caption generation has far reaching applications in terms of organising this data for cataloging, indexed searches and content based retrieval purposes. The temporal nature of videos makes them more complex than images and caption generation algorithms need to be able to efficiently select features needed to adequately describe events in the video. In our study we discussed how encoder-decoder architectures have been effectively used in the literature for image caption generation. Inspired by this, we adopted encoder-decoder architectures for our video caption generator. Just as generic image descriptors can be effective for image caption generation, spatiotemporal convolutions as generic video descriptors, can be effective for video understanding tasks. We therefore incorporated spatiotemporal convolutions in our encoder-decoder architecture for video caption generation. We devised experiments to determine an efficient temporal sampling rate for selecting a small number of frames needed to adequately describe a video.

Our results generally showed that sparsely sampled video features results in better caption quality since longer portions of the video are included and thus a longer portion of the event in question is covered. Pretrained word embeddings have been proven to be effective priors for encoding word representations in natural language processing tasks [31,32]. With this in mind, we designed experiments to compare the performance of our caption generator using jointly trained embeddings versus pretrained embeddings. We found that the model performs more or less equally effectively when using either jointly trained embeddings or using GLOVE embeddings that are finetuned for the caption generation task. We found that simply incorporating pretrained embeddings resulted in poor performance since the embeddings are not co-adapted with other caption generator parameters for the captioning task.

Our models have limitations that can be tackled as part of future work. In the previous chapter we showed that videos in the MSVD dataset not only have variable length but have different frame counts and frame rates. We manually selected the sampling strides that we thought could be effective for extracting salient features in the majority of videos. An improvement would be to train the sampling rate as an additional parameter to dynamically adapt between videos of disparate frame count and frame rate. We only tested our temporal sampling hypothesis on the MSVD dataset and therefore the models where optimised for the visual-semantic co-occurrence statistics of one dataset. For better generalisation, we would like to test this hypothesis on other video datasets including MSRVTT [149], a larger dataset with 10000 video clips and 260000 video-caption pairs.

Our models, just like the majority of encoder-decoder architectures, utilise some form of a recurrent neural network (RNN) for generating natural language descriptions. However, recently, attention based models have been developed with the aim to replace recurrent units which are inherently serial and therefore not parallelisable during training. Attention based models like the Transformer [99] are based fundamentally on attention mechanisms where the probability of a word being the next in a sequence is directly influenced by a weighted attention of all the preceeding words. This is unlike an RNN where the probability is conditionally dependent on recursively evaluating the hidden state of all the preceeding words. Transformer based models are quickly becoming a dominant strategy for language modelling tasks [150–152] and have been employed recently for dense video caption generation in videos with multiple overlapping events [100]. We would like to pursue the use of attention based caption generators with a view to make them more widely used for video understanding tasks that invariably involve a language modelling subtask.

Dense video captioning work by Krishna et al. [22] showed that there is a high agreement in the temporal event segments among human subjects. This is in line with research in neuroscience which suggests that during narrative perception, brain activity is naturally structured into semantically meaniful events [153]. This makes dense video captioning a more well-defined task where the video can be decomposed into discrete events, unlike general video captioning. The ActivityNet [22] dataset provides an opportunity to train video caption generators using videos with multiple overlapping events. The resulting video caption generator would be much more robust to complex open domain videos, as opposed to those trained on single event video datasets like MSVD and MSRVTT.

Ultimately video caption generation is a subset of the general task of learning the latent relationship between multiple modalities in order to describe the visual modality using the textual modality. Besides captions, videos are usually accompanied by audio and speech data and these can be integrated into an encoder to create a multi-modal feature that is used as input by the decoder [154]. Experimental results have shown the effectiveness of using a multi-modal fusion encoder, with significant performance gains on the BLEU, METEOR, ROUGE and CIDEr benchmarks. In the future we would like to explore incorporating the audio modality in order to understand how that can improve caption generation performance.

# References

[1] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *Association of Computational Linguistics (ACL)*, 2014.

[2] M. Hodosh, P. Young, and J. Hocknmaier, "Framing image description as a ranking task: Data, models and evaluation metrics," *International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[3] P. Young, A. Lai, M. Hodosh, and J. Hocknmaier, "From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions," *Association for Computational Linguistics (ACL)*, 2014.

[4] T. Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollar, "Microsoft COCO: Common objects in context," *European Conference on Computer Vision (ECCV)*, 2014.

[5] K. Simonyan and A. Zisserman, "Very deep convolulations networks for large-scale image recognition," *International Conference on Learning Representations (ICLR)*, 2015.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *Computer Vision and Pattern Recognition (CVPR)*, 2014.

[8] K. He, G. Gkioxari, and R. Girshick, "Mask R-CNN," *International Conference on Computer Vision (ICCV)*, 2017.

[9] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[10] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generator with visual attention," *International Conference on Machine Learning (ICML)*, 2015.

[11] A. Karpathy, A. Joulin, and L. Fei-Fei, "Deep fragment embeddings for bidirectional image sentence mapping," *Neural Information Processing Systems (NeurIPS)*, 2014.

[12] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[13] J. Mao, W. Xu, Y. Yang, J. Wang, Z. Huang, and A. Yuille, “Deep captioning with multimodal recurrent neural networks (m-RNN),” *International Conference on Learning Representations (ICLR)*, 2015.

[14] L. Yao, A. Torabi, K. Cho, N. Ballas, C. Pal, H. Larochelle, and A. Courville, “Describing videos by exploiting temporal structure,” *International Conference on Computer Vision (ICCV)*, 2015.

[15] A. Krizhevsky, I. Sutskever, and G. Hinton, “ImageNet classification with deep convolulational networks,” *Neural Information Processing Systems (NeurIPS)*, 2012.

[16] X. Glorot, A. Bordes, and Y. Bengio, “Domain adaption for large-scale sentiment classification: A deep learning approach,” *International Conference on Machine Learning (ICML)*, 2011.

[17] L. Zhang, S. Wang, and B. Liu, “Deep learning for sentiment analysis: A survey.” arXiv preprint arXiv:1801.07883 (2018).

[18] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *International Conference on Learning Representations (ICLR)*, 2015.

[19] L. A. Hendricks, S. Venugopalan, M. Rohrbach, R. Mooney, K. Saenko, and T. Darrell, “Deep compositional captioning: Describing novel object categories without paired training data,” *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[20] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, “Long-term recurrent convolutional networks for visual recognition and description,” *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[21] Y. Pu, M. R. Min, Z. Gan, and L. Carin, “Adaptive feature abstraction for translating video to text,” *Association for the Advancement of Artificial Intelligence (AAAI)*, 2018.

[22] R. Krishna, K. Hata, F. Ren, L. Fei-Fei, and J. C. Niebles, “Dense captioning events in videos,” *International Conference on Computer Vision (ICCV)*, 2017.

[23] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney, and K. Saenko, “Translating videos to natural language using deep recurrent neural networks,” *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2015.

[24] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko, "Sequence to sequence: Video to text," *International Conference on Computer Vision (ICCV)*, 2015.

[25] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3D convolulations networks," *International Conference on Computer Vision (ICCV)*, 2015.

[26] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and F.-F. L., "Large-scale video classification with convolutional neural networks," *Computer Vision and Pattern Recognition (CVPR)*, 2014.

[27] D. L. Chen and W. B. Dolan, "Collecting highly parallel data for paraphrase evaluation," in *Association for Computational Linguistics (ACL)*, 2011.

[28] J. R. Firth, "A synopsis of linguistic theory 1930-55.," vol. 1952-59, pp. 1–32, 1957.

[29] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *Neural Information Processing Systems (NeurIPS)*, 2013.

[30] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Neural Information Processing Systems (NeurIPS)*, 2013.

[31] J. Pennington, R. Socher, and C. Manning, "GLOVE: Global vectors for word representation," *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[32] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Association for Computational Linguistics (ACL)*, 2017.

[33] S. Ruder, M. E. Peters, S. Swayamdipta, and T. Wolf, "Transfer learning in natural language processing," in *North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 15–18, 2019.

[34] K. Papineni, S. Roukos, T. Ward, and W. Zhu, "BLEU: A method for automatic evaluation of machine translation," *Association of Computational Linguistics (ACL)*, 2002.

[35] A. Banerjee and A. Lavie, "METEOR: An automatic metric for mt evaluation with improved correlation with human judgements," *Worshop on Intrinsic Evaluation Measures for MT and/or Summarization at Association for Computational Linguistics (ACL)*, 2005.

[36] C. Lin, "ROUGE: A package for automatic evaluation of summaries," *Association of Computational Linguistics (ACL)*, 2004.

[37] C. Lin and F. J. Och, "Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics," *Association of Computational Linguistics (ACL)*, 2004.

[38] R. Vedantam, C. L. Zitnick, and D. Parikh, "CIDEr: Consensus-based image description evaluation," *Computer Vision and Pateern Recognition (CVPR)*, 2015.

[39] "Youtube by the numbers stats, demographics and fun facts." `https://www.omnicoreagency.com/youtube-statistics/`. Accessed: 2020-01-30.

[40] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, 2015.

[41] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, and D. Erhan, "Going deeper with convolulations," *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[42] R. Girshick, "Fast R-CNN," *International Conference for Computer Vision(ICCV)*, 2015.

[43] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *Neural Information Processing Systems(NeurIPS)*, 2015.

[44] O. Abdel-Hamid, A. Mohammed, J. Hui, L. Deng, G. Penn, and D. Yu, "Convolulational neural networks for speech recognition," *ACM Transactions on Audio, Speech and Language Processing (ACM/TASLP)*, vol. 22, No. 10, pp. 1533–1545, October 2014.

[45] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu, "Recurrent neural networks for multivariate time series with missing values," *International Conference on Learning Representations (ICLR)*, 2017.

[46] R. Kiros, R. S. Zemel, and R. Salakhutdinov, "Multimodal neural language models," *International Conference on Machine Learning (ICML)*, 2014.

[47] A. Mnih and G. Hinton, "Three new graphical models for statistical language modelling," *International Conference on Machine Learning*, 2007.

[48] R. Kiros, R. Salakhutdinov, and R. S. Zemel, "Unifying visual-semantic embeddings with multimodal neural language models," *Neural Information Processing Systems (NeurIPS): Deep Learning Worshop*, 2014.

[49] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, 1997.

[50] M. de Marneffe, B. MacCartney, and C. D. Manning, "Generating type dependency parses from phrase structure parses," *International Conference on Language Resources and Evaluation (LREC)*, 2006.

[51] J. Deng, W. Dong, R. Socher, L. L. J., and F.-F. L., "ImageNet: A large-scale hierarchical image database," 2009.

[52] R. Socher, A. Karpathy, Q. V. Le, C. D. Manning, and A. Y. Ng, "Grounded compositional semantics for finding and describing images with sentences," *Association for Computational Linguistics (TACL)*, 2014.

[53] A. Frome, G. S. Corrado, J. Shlens, S. Bengio, J. Dean, M. Ranzato, and T. Mikolov, "DeViSE: A deep visual-semantic embedding model," *International Conference on Machine Learning (ICML)*, 2014.

[54] M. Grubinger, P. Clough, H. Muller, and T. Deselaers, "IAPR TC-12: A new evaluation resource for visual information systems," *International Worshop OntoImage*, pp. 13–23, May 2006.

[55] J. Johnson, A. Karpathy, and L. Fei-Fei, "DenseCap: Fully convolutional localization networks for dense captioning," *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[56] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEE Transactions on Neural Networks*, 1994.

[57] D. Gilbao, B. Chang, M. Chen, G. Yang, S. S. Schoenholz, H. E. Chi, and J. Pennington, "Dynamical isometry and a mean field theory of lstms and grus," *International Conference on Machine Learning (ICML)*, 2018.

[58] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L. Li, D. A. Shamma, M. S. Bernstein, and L. Fei-Fei, "Visual genome: Connecting language and vision using crowdsourced dense image annotations," *International Journal of Computer Vision (IJCV)*, 2016.

[59] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L. L., "YFCC100M: The new data in multimedia research," *Communications of the ACM*, vol. 59, No. 2., pp. 64–73, February 2016.

[60] "Amazon Mechanical Turk." `https://www.mturk.com/`. Accessed: 2019-05-04.

[61] "Wikipedia multilingual corpus." https://dumps.wikimedia.org/enwiki/latest/. Accessed: 2019-04-28.

[62] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, vol. 22, No. 3., pp. 400–407, September 1951.

[63] J. Keifer and J. Wolfowitz, "Estimation of the maximum of a regression function," *The Annals of Mathematical Statistics*, vol. 23, No. 3., pp. 462–466, 1952.

[64] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large scale machine learning," *Society of Industrial and Applied Mathematics (SIAM)*, vol. 60, No. 2., pp. 223–311, 2018.

[65] A. Graves, "Sequence transduction with recurrent neural networks," *International Conference on Machine Learning(ICML)*, 2012.

[66] M. Corbetta and G. L. Shulman, "Control of goal-directed and stimulus-driven attention in the brain," *Nature Reviews: NeuroScience*, 2002.

[67] R. A. Rensik, "Dynamic representation of scenes," *Visual Recognition*, 2000.

[68] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, May 2017.

[69] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1., pp. 318–362, 1986.

[70] M. Tanti, A. Gatt, and K. P. Camilleri, "Where to put the image in an image caption generator." arXiv preprint arXiv:1703.09137 (2017).

[71] J. Devlin, H. Cheng, H. Fang, S. Gupta, L. Deng, X. He, G. Zweig, and M. Mitchell, "Language models for image captioning: The quirks and what works," *Association of Computational Linguistics (ACL)*, 2015.

[72] S. Liu, Z. Zhu, N. Ye, S. Guadarrama, and K. Murphy, "Improved image captioning via policy gradient optimization of spider," *International Conference on Computer Vision (ICCV)*, 2017.

[73] M. Wang, L. Song, X. Yang, and C. Luo, "A parallel fusion RNN-LSTM architecture for image caption generation," *International Conference on Image Processing (ICIP)*, 2016.

[74] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *International Conference on Image Processing (ICIP)*, 2014.

[75] O. Nina and A. Rodriguez, "Simplified LSTM unit and search space probability exploration for image description," *International Conference on Information, Communications and Signal Processing (ICICS)*, 2015.

[76] S. J. Rennie, E. Marcheret, Y. Mroueh, J. Ross, and V. Goel, "Self-critical sequence training for image captioning," *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[77] Q. Wu, C. Shen, A. van den Hengel, L. Liu, and A. Dick, "Image captioning with an intermediate attributes layer." arXiv preprint arXiv:1506.01144v1 (2015).

[78] T. Yao, Y. Pan, Y. Li, Z. Qiu, and T. Mei, "Boosting image captioning with attributes," *International Conference on Computer Vision (ICCV)*, 2017.

[79] J. Hessel, N. Savva, and M. J. Wilber, "Image representations and new domains in neural image captioning," *Empirical Methods in Natural Language Processing Vision + Language Worshop (EMNLP)*, 2015.

[80] L. Zhou, C. Xu, P. Koch, and J. J. Corso, "Image caption generation with text-conditional semantic attention." arXiv preprint arXiv:1606.04621 (2016).

[81] Q. You, H. Jin, Z. Wang, C. Fang, and J. Luo, "Image captioning with semantic attention," *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[82] J. Mao, W. Xu, Y. Yang, J. Wang, Z. Huang, and A. Yuille, "Learning like a child: Fast novel visual concept learning from sentence descriptions of images," *International Conference on Computer Vision (ICCV)*, 2015.

[83] J. Mao, W. Xu, Y. Yang, J. Wang, and A. Yuille, "Explain images with multimodal recurrent neural networks (m-RNN)," *Neural Information Processing Systems (NeurIPS)*, 2014.

[84] J. Lu, C. Xiong, D. Parikh, and R. Socher, "Knowing when to look: Adaptive attention via a visual sentinel for image captioning," *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[85] M. Song and C. D. Yoo, "Multimodal representation: Kneser-ney smoothing/skip-gram based neural language model," *International Conference on Image Processing (ICIP)*, 2016.

[86] J. Thomason, S. Venugopalan, S. Guadarrama, K. Saenko, and R. Mooney, "Integrating language and vision to generate natural language descriptions of videos in the wild," *International Conference on Computational Linguistics: Technical Papers (COLING)*, 2014.

[87] N. Krishnamoorthy, G. Malkarnenkar, R. J. Mooney, K. Saenko, and S. Guadarrama, "Generating natural-language video descriptions using text-mined knowledge," *Association for the Advancement of Artificial Intelligence (AAAI)*, 2013.

[88] S. Guadarrama, N. Krishnamoorthy, G. Malkarnenkar, S. Venugopalan, R. J. Mooney, T. Darrell, and K. Saenko, "Recognizing and describing activities using semantic hierarchies and zero-shot recognition," *International Conference on Computer Vision (ICCV)*, 2013.

[89] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," *International Conference on Machine Learning (ICML)*, 2001.

[90] M. Rohrbach, W. Qiu, I. Titov, S. Thater, M. Pinkal, and B. Schiele, "Translating video content to natural language descriptions," *International Conference on Computer Vision (ICCV)*, 2013.

[91] A. Rohrbach, M. Rohrbach, Q. W., A. Friedrich, M. Pinkal, and B. Schiele, "Coherent multi-sentence video description with variable level of detail," in *German Conference on Pattern Recognition (GCPR)*, 2014.

[92] K. Soomro, A. R. Zamir, and M. Shah, "UCF-101: A dataset of 101 human action classes from videos in the wild." arXiv preprint arXiv:1212.0402 (2012).

[93] N. Dalal, B. Triggs, and C. Schmid, "Human detection using oriented histograms of flow and appearance," *European Conference on Computer Vision (ECCV)*, 2006.

[94] H. Wang, M. M. Ullah, A. Klaser, I. Laptev, and C. Schmid, "Evaluation of local spatiotemporal features for action recognition," *British Machine Vision Conference (BMVC)*, 2009.

[95] H. Yu, J. Wang, Z. Huang, Y. Yang, and W. Xu, "Video paragraph captioning using hierarchical recurrent neural networks," *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[96] K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, D. Bahdanau, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[97] F. C. Heilbron, V. Escorcia, B. Ghanem, and J. C. Niebles, "ActivityNet: A large scale video benchmark for human activity understanding," *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[98] "ActivityNet Captions Challenge, Task 5: Dense captioning events in videos." `http://activity-net.org/challenges/2017/index.html`. Accessed: 2019-04-21.

[99] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Poloshukhin, "Attention is all you need," *Neural Information Processing Systems (NeurIPS)*, 2017.

[100] L. Zhou, Y. Zhou, J. J. Corso, R. Socher, L. Jones, and C. Xiong, "End-to-end dense video captioning with masked transformer," *Computer Vision and Pateern Recognition (CVPR)*, 2018.

[101] "What's the difference between a CPU and a GPU?." https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/. Accessed: 2019-06-01.

[102] "An in-depth look at Google's first Tensor Processing Unit (TPU)." https://cloud.google.com/blog/products/gcp/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu. Accessed: 2019-06-01.

[103] "TensorFlow." https://www.tensorflow.org/. Accessed: 2019-06-04.

[104] "PyTorch." https://pytorch.org/. Accessed: 2019-06-04.

[105] T. D. Team, "Theano: A python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016.

[106] M. London and M. Hausser, "Dentritic computation," *Annual Review of Neuro-Science*, vol. 28, pp. 503–532, July 2005.

[107] O. Russakovsky, J. Deng, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, A. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, vol. 115, No. 3., pp. 211–252, April 2015.

[108] A. L. Maas, A. Y. Hannum, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," *International Conference on Machine Learning (ICML)*, 2013.

[109] K. He, X. Zhang, S. Ren, and H. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," *International Conference on Learning Representations (ICLR)*, 2016.

[110] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units," *International Conference on Learning Representations (ICLR)*, 2016.

[111] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, No. 3., pp. 273–297, September 1995.

[112] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[113] A. Y. Ng, "Feature selection, L1 vs L2 regularization, and rotational invariance," *International Conference on Machine Learning (ICML)*, 2004.

[114] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society*, 2003.

[115] N. Srebro and A. Shraibman, "Rank, Trace-Norm and Max-Norm," *In Proceedings of the 18th Annual Conference on Learning Theory*, 2005.

[116] N. Srivastava, G. Hinton, A. Krizhevsky, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research (JMLR)*, vol. 15, pp. 1929–1958, 2014.

[117] A. Baydin, B. Pearlmutter, A. Radul, and J. Siskind, "Automatic differential in machine learning: A survey," *Journal for Machine Learning Research*, vol. 18., pp. 1–43, 2017.

[118] "Cs231n: Convolutional neural networks for visual recognition - learning." `http://cs231n.github.io/neural-networks-3/`. Accessed: 2019-06-16.

[119] S. Ruder, "An overview of gradient descent optimisation algorithms," arXiv preprint arXiv:1609.04747 (2017).

[120] Y. Nesterov, "A method for solving a convex programming problem with convergence rate o$(1/k^2)$," *Soviet Mathematics Doklady*, vol. 27, pp. 327–376, 1983.

[121] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimisation," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[122] "RMSProp: Divide the gradient by a running average of its recent magnitude." `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`. Accessed: 2019-04-23.

[123] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimisation," *International Conference in Learning Representations (ICLR)*, 2015.

[124] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of Physiology*, vol. 160, No. 1., pp. 106–154, September 1962.

[125] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the Institute of Electrical and Electronics Engineers (IEEE)*, vol. 86, pp. 2278–2324, November 1998.

[126] "Theano: Convolutional arithmetic tutorial." `http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html`. Accessed: 2019-06-16.

[127] M. Ranzato, F. Huang, Y. Boureau, and Y. LeCun, "Unsupervised learning of invariant feature hierarchies with applications of object recognition," *Computer Vision and Pattern Recognition (CVPR)*, 2007.

[128] "Cs231n: Convolutional neural networks for visual recognition - cnns." `http://cs231n.github.io/convolutional-networks/`. Accessed: 2019-06-16.

[129] C. Lee, P. W. Gallagher, and Z. Tu, "Generalizing pooling functions in convolutional neural networks: Mixed, gated and tree," *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

[130] "Convolutional neural networks." `http://cbl.eng.cam.ac.uk/pub/Intranet/MLG/ReadingGroup/cnn_basics.pdf`. Accessed: 2019-04-27.

[131] A. Miyake and P. Shah, "Models of working memory: Mechanisms of active maintenance and executive control," 1999.

[132] J. Gamboa, "Deep learning for time-series analysis." arXiv preprint arXiv:1701.01887 (2016).

[133] "Understanding LSTM networks." `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`. Accessed: 2019-06-16.

[134] P. J. Werbos, "Generalization of backpropagation with application or recurrent gas market model," *Neural Networks*, vol. 1., pp. 339–356, 1988.

[135] "The unreasonable effectiveness of recurrent nueral networks." `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`. Accessed: 2019-07-04.

[136] R. Pascanau, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," *International Conference on Machine Learning (ICML)*, 2013.

[137] F. A. Gers and J. Schmidhuber, "Recurrent nets that time and count," *International Joint Conference on Neural Networks*, 2000.

[138] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[139] "Using pre-trained word embeddings in a Keras model." `https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html`. Accessed: 2019-07-27.

[140] N. Aafaq, S. Z. Gilani, W. Liu, and A. Mian, "Video description: A survey of methods, datasets and evaluation methods." arXiv preprint arXiv:1806.00186 (2018).

[141] "Keras." `https://keras.io/`. Accessed: 2020-01-28.

[142] "OpenCV." `https://opencv.org/`. Accessed: 2020-01-28.

[143] P. Anderson, B. L. Fernando, M. Johnson, and S. Gould, "SPICE: Semantic propositional image caption evaluation," *European Conference on Computer Vision (ECCV)*, 2016.

[144] G. A. Miller, "WordNet: A lexical database for english," vol. 38, No. 11, pp. 39–41, Nov 1995.

[145] D. Elliot and F. Keller, "Comparing automatic evaluation measures for image description," *Association for Computational Linguistics (ACL)*, 2014.

[146] S. Robertson, "Understanding inverse document frequency: On theoretical arguments for idf," *Journal of Documentation*, vol. 60, No. 5., pp. 503–520, October 2014.

[147] A. Grinciunaite, A. Gudi, E. Tasli, and M. den Uyl, "Human pose estimation in space and time using 3D CNN," *European Conference on Computer Vision Workshops (ECCV)*, 2016.

[148] "GLOVE: Global vectors for word representation." `https://nlp.stanford.edu/projects/glove/`. Accessed: 2019-08-2.

[149] J. Xu, T. Mei, T. Yao, and Y. Rui, "MSR-VTT: A large video description dataset for bridging video and language," in *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[150] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[151] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. Le, and R. Salakhutdinov, "Transformer-Xl: Attentive language models beyond a fixed-length context," *https://arxiv.org/abs/1901.02860*, 2019.

[152] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[153] C. Baldassano, J. Chen, A. Zadbood, J. Pillow, U. Hasson, and K. Norman, "Discovering event structure in continuous narrative perception and memory," *Neuron*, vol. 95, No. 3., pp. 709–721, August 2017.

[154] Q. Jin, J. Chen, S. Chen, Y. Xiong, and A. Hauptmann, "Describing videos using multi-modal fusion," *ACM International Conference on Multimedia*, vol. 16, pp. 1087–1091, October 2016.