Project Dissertation

Trinity Term 2022

Title:

Video Annotation with Knowledge Graphs, using Deep Learning and Background Knowledge

Candidate number: 1057791

Degree: MSc in Advanced Computer Science

Word Count: 28342*

*using Overleaf's built-in TeXcount utility, it excludes the table of contents, all mathematical equations and symbols, diagrams, tables and the bibliography.

Abstract

Recent advances in deep learning have motivated researchers to tackle more complex problems in the area of visual understanding. One of the challenging topics in this field is extracting semantic content from visual inputs, such as images and videos. In the past, this task has often been combined with language models to effectively generate annotation for images and videos, which could express the semantic content with natural language sentences. One major shortcoming with this framework is that the language models encode statistical irregularities in natural language text, which leaves the deep learning model prone to exploiting such irregularities rather than drawing on the visual contents. Additionally, these systems could not incorporate background or commonsense knowledge and have poor interpretability.

This thesis proposes a deep learning system that can extract the semantic content of videos and expresses it as a knowledge graph. The video annotation system is a deep neural network consisting of a combination of CNNs and an RNN, encoding the video into a feature vector. Together with a combination of several multi-classifiers and MLPs, it produces the first set of facts. For the second set of predictions, an independent dataset is used to impart commonsense knowledge, which is then combined with the first set of predictions to produce structured captions in the form of a knowledge graph. The model can no longer exploit the natural language patterns and is forced to base its output solely on the relevant semantic information.

I chose to annotate videos because structured annotation of videos is a less researched task than structured annotation tasks for images. There is a group of work which aims to describe images using structured representation known as scene graphs. Limited work on the extraction of scene graphs from videos has also been attempted but has faced some shortcomings. A knowledge graph is similar to a scene graph, but unlike scene graphs, they can represent any sort of knowledge and do not require to be visuospatial. For video annotation tasks, this is desirable.

To my knowledge, only two existing systems [1, 2] have attempted video annotation using knowledge graphs, and my model significantly outperforms both systems while offering several other benefits. Thus, this thesis advances the state-of-the-art video annotation using knowledge graphs and offers additional advantages of incorporating background knowledge and injecting commonsense knowledge, thereby producing a model with better generalization capability and interpretability.

Contents

1 Introduction

In recent years, artificial intelligence (AI) has gained a lot of popularity and is a thriving field with many practical applications and active research topics. AI enables machines to perform tasks that usually require human intelligence. In fact, some abstract and formal tasks that are among the most difficult mental undertakings for a human being are among the easiest for a computer. However, for a computer to behave in an intelligent way, it needs to capture knowledge which is subjective and intuitive and so is difficult to articulate in a formal way. And this task of getting informal knowledge into computers has been one of the challenges in AI.

Several AI projects have sought to hard-coding knowledge about the world in formal languages, on which a computer can reason and apply logical inference rules (more discussion on logic and logical inference is given in Section 2.3). This started to be known as the knowledge base approach to AI (or knowledge base system (KBS). One of the most notable KBS was Cyc [3]. Cyc was used for logical inferences using a database of statements in a language called CycL [4]). These statements were entered by human engineers. The project did witness success in some applications [5]. However, on its own, it was not enough for many applications because it was difficult to devise formal rules with enough complexity to describe the world accurately. For instance, Cyc failed to understand the story of a person named Fred, who was shaving. The inference engine detected an inconsistency in the story: the engine knew that humans do not have electrical parts, but Fred was holding an electric razor, and so it believed the entity "FredWhileShaving", contained electrical part. This made the engine question if Fred was still a human while he was shaving [6].

Understandably, these KBSs' reliance on hard-coded knowledge suggested that AI systems need something more; they need the ability to acquire their own knowledge by extracting patterns from raw data. This capability, known as machine learning (ML), which was already present before the above AI systems, now started getting more focus from the researchers. ML is a sub-field of AI where the systems learn how to perform various tasks from their experience.

However, the ML system's performance still depended heavily on the representation of the data they were given. That is, a human is required to tell the system several pieces of relevant information, which represent the data, for it to work or make decisions. These pieces of information included in the data representation are known as **features**. Manually selecting and feeding these features is only possible in simpler scenarios. Extraction of high-level abstract features directly from the raw data without human intervention proved to be a major challenge with ML systems.

To solve this central problem with ML, an old idea of the artificial neural network started receiving more attention. The systems based on artificial neural network is often referred to as deep learning (DL) [7] and is considered a subdomain of ML. In recent years, DL has produced impressive results in several key areas which have previously proved to be unfeasible for earlier AL and ML systems. This made DL receive intense interest and enthusiasm from the research community. DL has the capability to extract the features that are most relevant to the task at hand on their own (more discussion on this and various deep learning algorithms are covered in Section 2.1).

Coming back to our initial discussion, which started in the initial era of AI, we want our computers to capture knowledge which is subjective and intuitive, which cannot be articulated in a formal way. For us humans to understand the world, it was critical to develop an understanding of our surroundings by seeing them. In fact, we use about 27% of our precious brain power for visual processing[8, 9], which is a fundamental task. Our visual perception of the world is reflected in our ability to make decisions through what we see, and providing such capabilities to computers will make it possible to design remarkable applications that will enhance our lives. And so, we want the DL systems to develop such abilities as understanding scenes, human behaviours, relationships, and reasoning. Giving the machines the ability of visual understanding will help doctors and nurses to have an extra pair of tireless eyes for diagnosis and patient care, cars will run smarter and safer on the road, robots will help brave disaster zones to save the trapped and wounded, we will be able to explore unseen frontiers, among several tasks which will be made possible. Hence, the area of visual and cognitive understanding is extremely important.

One of the distinguished researchers in this area, Fei Fei Li, rightly said [10]:

> "If we want our machines to think, we need to teach them to see. And by see I mean to understand and not just record pixels."

This idea has been a research topic in a field of AI known as computer vision. Computer vision enables computers and systems to derive meaningful information from images, videos and other visual inputs - and take actions or make recommendations based on that information. In 2014 Li, along with her student, produced one such computer vision model capable of generating human-like sentences to describe an image [11].

This human-like sentence generation has been a major part of existing research in another field of AI, known as natural language processing (NLP), which deals with interactions between computers and human language. Here natural language means human language like English, Russian, German etc., as opposed to artificial commands or computer-programming languages.

Hence, visual understanding has long been a desire of humans. In the recent past, the emergence of DL compiled with better computing power has majorly contributed to the development of many such tasks [12–15]. Visual understanding can be understood as the task of automatically detecting and identifying objects in a scene, their attributes and the relationship existing between them. This would require the machines to develop cognition and extract semantic information from the visual input. In this context, one of the important sub-problems and tasks includes image and video annotation. The goal of the image/video annotation (or referred image/video captioning in the NLP domain) task is to capture the visual input's semantics by generating a summary of them in a humanly understandable form. This is a highly demanding task, and advancement of this process will open up enormous opportunities in many application domains, such as serving aid to people who suffer from various degrees of visual impairments by transforming visual signals into information that can be communicated to them, facilitate automatic sign-language translation,

advance automatic video/image indexing or retrieval thereby enhancing search engines and platforms like Google photos, help in surveillance and military, improve human-robot interaction, enhance the optical system analysis of self-driving vehicles, and perform numerous such tasks of importance [16].

In visual input annotation systems, the main challenge is to extract visual information from the picture and transform this information into a proper and meaningful language. This challenge has often been considered a vision to language problem, where the content of the visual input is represented in a natural language sentence. Recently, significant progress has been observed in this task of understanding images, and videos using various DL techniques, with current state-of-the-art models able to caption them in NL [17–19] and classify objects close to human-level precision but on certain datasets only [20–22]. However, this problem is far from being considered a solved task. This is because NL captioning systems suffer from major shortcomings such as:

- Firstly, the generation of NL caption for a visual input requires the system to do two tasks - understand the semantic content of the visual input and determine how to say that in a grammatically correct NL sentence [23, 24]. For this, the system has to simultaneously learn both tasks at the same time. However, the former task is what our aim is, the latter is irrelevant to the goal of annotation, and unsurprisingly it makes the overall task more difficult.

- Secondly, NL annotations are in the form of a sentence, which makes them unsuitable for logical inference (Section 2.3.3). That is, it can identify a man in the image but cannot conclude that "all men are males" or "all men are humans". Inferences like this come so naturally to us humans and cannot be easily performed for NL tasks [25, 26]. Also, we cannot write rules like KBS or perform string matching because words have different contexts. For example, the word "man" has a different context in both these sentences: "sailors man the ship", "A man is running", so hard-coded rules or string matching will not make sense here [2].

- Another drawback is that NL captions are single-language specific. So a model trained to produce captions in English cannot produce annotation in German, and it is difficult to translate

the English caption into German ones, as different languages have different grammatical and structural rules.

- Lastly, there is a practical problem with respect to NL annotations. They are not easy to interpret or quantitatively measure. This is because the closeness between the model-generated and actual sentences cannot be directly measured using standard metrics used in the ML domain (more discussion on this in Section 3.1). Instead, they require specifically designed ones like, BLEU [27], METEOR [28], and LEPOR [29]. Typically, scores are reported on multiple metrics as none of them is perfect, and so it lacks simple, intuitive interpretation.

All these reasons compelled the researchers to look for an alternative solution. Recent research in computer vision has advocated for logically structured representation of semantic information from a visual scene [1, 2, 19, 30]. Using structured representation rather than natural language forces the model to base its output only on the relevant semantic information instead of exploiting natural language patterns. A simpler, more abstract and structured representation of the semantics in a visual scene may help avoid the complexity of natural language when learning such models and could improve reasoning abilities.

In this context, the initial attempts in the image annotation domain gave birth to the emergence of scene graphs. Scene graphs were first proposed in [31] as a data structure that describes the object instances in an image scene and their relationship. A complete scene graph can potentially represent the detailed semantics of a dataset of scenes. A scene graph represents the scene using a set of $\langle subject, predicate, object \rangle$ triplets, where each of the connections uses directed graphs (more discussion on this in Section 3). This representation proved to be more successful and hence attracted a lot of attention from a large number of researchers [19].

Since scene graphs were proving highly successful in image annotation tasks, this also inspired researchers to employ them for video-based tasks. Scene graphs are often considered a visuospatial representation of a scene; that is, it shows what is where in a scene. For image annotation, the

nodes in the scene graph are often associated with a bounding box (coordinates of that entity) in the image and, through this, show what is where in a scene [32]. Some researchers tried to implement a scene graph for video annotation where each frame had an associated scene graph, thus implementing the visuospatial requirement. However, a scene graph for every frame was redundant and computationally expensive, and it was difficult to predict relationships which spanned through multiple frames [32].

Hence while structured annotation for videos is a useful task, scene graphs might not be a suitable choice for them. In [1] the authors made an initial attempt to logically annotate videos using another representation called knowledge graph (KG). KGs are similar to scene graphs in the sense that it also represents the objects present in the video and the facts that hold true of them, but they can represent any type of information and do not need to show what is where in the video. This means that a KG has the potential to successfully extract the semantic content of a video spanned across multiple frames. A KG often contains a large number of fact triples and multi-relational data, denoted in the form of $\langle subject, predicate, object \rangle$. A fact expresses a relation between two objects (e.g., $fold(person, paper)$ or an attribute of an individual (eg. $white(paper)$). Discussion on this notation is given in Section 2.3.3.

Since knowledge graphs are structured representations of semantic content, they avoid all the problems previously seen in NL annotation, as alluded above. One major advantage of using KG is that they have been shown as a graph-variant of first-order logic, which is equivalent to standard first-order logic [33], which makes it possible to do seamless logical deduction and inference using KGs. And so, applying inference rules like "all men are also males" is trivial here. Further, KGs are machine-readable, so we can perform automated data processing, such as searching all videos depicting a "man". Next, they are generated only using the semantics of the video content and we do not need to worry about generating syntactically and semantically correct NL sentences. They are also language agnostic, as they can be trivially translated to another language by translating one component of the triple at a time. Lastly, their performance can easily be measured quantitatively

using standard metrics such as F1-score and accuracy.

While most of the previous work focused on structurally annotating images by generating scene graphs, structured video annotation is a less researched task. Video annotation using KG has only been explored by two papers before (as per my knowledge) [1, 2], and has shown convincing results and promises a new research direction for us. Hence, designing a solution to comprehensively and formally annotate video content, particularly using knowledge graphs, could significantly impact the video understanding domain. This makes it an exciting and promising research area. Hence in this thesis, I chose to work on structured video annotation using knowledge graphs.

This thesis proposes a novel deep-learning-based framework for annotating videos with knowledge graphs. I have leveraged KGs to explicitly capture the relationship information between objects and their attributes in a video. Inspired by the promising results in [2], the encoder proposed by them [2] is used to extract spatio-temporal video features. Further, I have employed three conditionally independent multi-classifiers to extract the objects, the relationship between them, and their attributes. These three components are intelligently combined to filter out irrelevant facts and only keep possible facts. The possible facts are later passed through another binary classifier to classify them as true or false facts with respect to the video. This produces the first set of predictions. Next, I recognised the need to make the model dataset agnostic; that is, the model needs to give good results on videos from any domain. To satisfy this need, I introduce a new use of the existing independent dataset of images annotated with scene graphs: Visual Genome [30]. Further, I have also devised a method to extract statistics from this dataset and apply Bayesian probability rules to get predictions on input data. Both the separate predictions are finally combined using a machine learning model called logistic regressor to produce the final KG for a video. The results are evaluated using two automatically generated datasets, MSVD* and MSR-VTT* (taken from [2]). I receive an F1 score of **29.03** for MSVD* dataset, and **37.23** for MSR-VTT* dataset, which is significantly better than the previous models [1, 2]. The results show the effectiveness of the proposed model. The main contributions of this work are thus briefly summarized below:

- I describe the use of attribute and relation multi-classifiers to predict the predicates (attributes and relations) in a video. While [2] was only using an individual multi-classifier to identify objects in the image, which forced them to combine the identified objects with every possible predicate in the vocabulary and generated many irrelevant facts, on the other hand, the introduction of predicate multi-classifiers on top of individual multi-classifier in this work avoids this issue in the existing model.

- The above three multi-classifiers proposed in this work produce separate predictions for individuals, attributes and relations, which need to be combined efficiently to filter irrelevant facts. In Section 4.3.1, I have proposed a framework with mathematical foundations to perform this task, which has shown to significantly improve the model's performance.

- Next, I have also proposed a novel framework for utilising an independent dataset (Visual Genome in this work) to inject" commonsense knowledge of which predicates are likely to hold true of which individuals in general. As per my knowledge, this work is the first to explore the impact of an independent dataset for helping the prediction of the DL-based model. This is executed by first extracting various statistics from the Visual Genome dataset and followed by generation of prediction using the Bayesian framework.

- Secondly, I have also performed an investigative study in Section 8.2 to explore the impact of using an independent dataset for injecting commonsense knowledge. The results signify that this step has given my model the ability to produce good predictions even when the model is not at all trained and is initialised with random values. This step paves the way for overcoming the issue with the availability of fewer data for training the model and tries to make the model data agnostic, such that it could perform equally well across various datasets without any bias.

- I have also quantitatively measured the performance of the proposed model on standard metrics and shown that the results are substantially better while conquering some major problems in the baseline models.

- Next, several investigative studies have also been performed, which show that controlling

certain hyper-parameter in the model could in fact produce even better results. I also studied how this improvement in results comes at the cost of the requirement of more processing power. This trade-off is plotted indicating better performance with improvement in processing power. Lastly, I have also performed ablation study on various components of the network to study their effectiveness.

Finally, the dissertation is structured as follows:

- In Chapter 2, I discuss the background necessary to understand the work presented in this thesis. This includes discussion on deep learning, its various algorithms and concepts. This is followed by a discussion on logic, logical inference, knowledge graphs, WordNet and local-closed world assumption.
- In Chapter 3, I give a brief overview of the related work done in the field of video annotation and the various components which inspired this work.
- In Chapter 4, I present the proposed system, where the formal problem statement is presented, followed by a formal description of the system and a detailed discussion on the approach used in this work.
- In Chapter 5, I present the training and implementational details related to the proposed system. Here the training procedure is explained, along with the loss functions used. I also discuss the hyper-parameter search procedure and values of various parameters. Lastly, I also include a discussion on the choice of the encoder used by the proposed model, which is an essential component.
- In Chapter 6, I discuss the datasets that were used for evaluation followed by the method used for automatically generating the datasets. It also presents a discussion on the Visual Genome dataset and extraction of statistics from it for injection of commonsense knowledge.
- In Chapter 7, I start by discussing the choice of evaluation metrics, followed by the quantitative

and qualitative results. The results are further analysed to extract important conclusions.

- In Chapter 8, I present a discussion on some investigative studies and present results from further experiments and ablation studies.

- Lastly, in Chapter 9, I conclude the thesis by explaining the main contributions of this work and suggest some possible future directions for this project.

2 Background and State of the Art

This chapter provides an overview of the fundamentals of various concepts relevant to this thesis. I start with a discussion on deep learning, its building blocks, and pertinent algorithms. I then present a brief discussion on concepts related to training, regularisation and hyper-parameter search useful to this work. After this, I move on to the discussion on logic and knowledge graphs. This is followed by a presentation of WordNet and local closed-world assumption concepts.

2.1 Deep Learning

Deep learning (DL), a subfield of machine learning which attempts to learn high-level abstractions in data by utilising hierarchical architectures. It is an emerging approach which has been largely applied to many traditional AI domains [34]. The boom of DL is often considered the outcome of three things: (1) a drastic increase in chip processing abilities (e.g. GPU units), (2) a significantly lowered cost of computing hardware and (3) considerable advances in ML and DL algorithms [35].

Some of the earliest learning algorithms we recognise today were intended to be thecomputational models of biological learning, that is, models of how learning happens or could happen in the brain [6]. This inspired the formation of a model called artificial neural network (ANN) [36]. DL models are largely based on ANNs. In the following sections, I discuss some of the building blocks of ANNs.

2.1.1 Neural Networks

Artificial neural networks or popularly known as neural networks (NNs) in the field of deep learning are a class of learning algorithms which are composed of several simple, connected processors called neurons. The behaviour of a NN is determined by the properties of these individual neurons and the connection between them.

In this section, I start by discussing the building block of NNs, the neurons. Next, I present the first type of NN, known as multi-layer perceptions, followed by a discussion on convolutional neural networks and recurrent neural networks.

Neuron The fundamental unit of any NN is the neuron or an artificial neuron. It was first introduced by Warren McCulloch (neuroscientist) and Walter Pitts (logician) in 1943 as a computational model of the biological neuron.

An artificial neuron is a mathematical function based on the model of a biological neuron, where each neuron receives input from other neurons, performs some processing and produces an output in a form of a single real number.

However, the McCulloch-Pitts neurons suffered from some limitations. It only allowed binary inputs and outputs, assigned equal weights to inputs and parameters like thresholds needed to be hand-coded (these parameters are discussed in the context of another modified model below).

This inspired the work of Frank Rosenblatt, who in 1958 modified the McCulloch-Pitts neuron to overcome some limitations by making weights and thresholds to be learned over time instead of hand-coding them. This model is now known as the classical perceptron model, where the neurons were called perceptrons [37]. Further, in 1969, Minsky and Papert [38] refined and carefully analysed this work, and their model is referred to as the perceptron model. The perceptron model is a more general computational model and overcomes some of the limitations in the above models.

A perceptron takes as input any real values $[x_1, x_2, \ldots, x_m]$, where m can be any finite natural number. Then it calculates a weighted sum of the input and sets the output as one only when the sum is more than an arbitrary threshold (θ). This is shown in Equation 1.

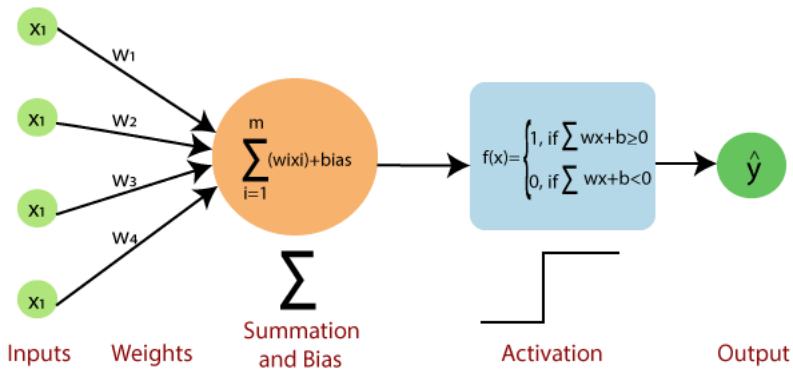

Figure 1: A simple perceptron model, taken from [39]

$$Output = \begin{cases} 1 \; if \; \sum_i^n w_i x_i \geq \theta \, , \\ 0 \; if \; \sum_i^n w_i x_i < \theta \, , \end{cases} \tag{1}$$

where each input x_i is assigned a weight w_i, and θ is the given threshold value where the output of the neuron changes. While the classical perceptron model introduced by Rosenblatt [37] only allowed binary values as inputs and equal weights to all the features, Minsky and Papert [38] allowed different weights to different inputs and inputs were not required to be binary values.

Now, if we look closely at equation 1, we can see that the output changes from 0 to 1 abruptly. That is, let θ be 0.5, and the value of $\sum_{i=0}^n w_i x_i$ is 0.49, then the output would be 0, however if the value of $\sum_{i=0}^n w_i x_i$ is 5.01, then the output will be 1. These sudden change from 0 to 1 when $\sum_{i=0}^n w_i x_i$ crosses the threshold is sometimes not expected in some applications and, requires a smoother change which gradually changes from 0 to 1.

So we need to modify Equation 1 such that it could accommodate both scenarios - (1) abrupt change from 0 to 1, (2) gradually changes from 0 to 1, in output. One way to do this is by using the activation function which introduces non-linearity. These types of perceptrons are called non-linear neurons. With this addition, the output is given by Equation 2. This perceptron model can be

visualized in Figure 1, which is taken from [39].

$$Output = f(\sum_{i=1}^{m} w_i x_i)\,, \tag{2}$$

where f is the activation function.

A few commonly used activation functions are:

- Sigmoid: The sigmoid function is given by $\sigma(x) = \frac{1}{1+e^{-x}}$. The sigmoid function only ranges from 0 to 1.
- Tanh / Hyperbolic tangent activation function: Tanh is a lot similar to the sigmoid function, where the function also has a sigmoidal "S" shape, but the difference is that it ranges from -1 to 1. It is given by: $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.
- ReLU: ReLU stands for Rectified Linear Unit and is a popularly used activate function in neural networks. It is given by: $f(x) = max(0, x)$ [40]. It ranges from 0 to ∞. The ReLU function suffers from a problem called, 'dying ReLU', where the output is value 0 for any input value. This could happen when the model learns highly negative values for biased terms. This means that the model did not learn anything from this neuron.
- Leaky ReLU: To overcome the 'dying ReLU' problem, leaky ReLU was introduced [41]. It does not set negative values to 0, instead, it follows a linear equation with gradient α, where α can be a small positive real number, like 0.01. It is given by Equation 3:

$$LeakyReLU(x) = \begin{cases} x \;\; if\; x \geq 0\,. \\ \alpha x \;\; if\; x < 0\,. \end{cases} \tag{3}$$

  Hence, in LeakyReLU, even if the input is negative, the output is a small negative value and not 0.

- Softmax function: It calculates the relative probability and is among the most commonly used activation function in the case of multi-class classification. It is given by:

$$softmax(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \,. \tag{4}$$

2.1.2 Multi-Layer Perceptron (or Feedforward Neural Network)

The principle weakness of the single-neuron perceptron model was that it could only solve problems that were linearly separable [42]. To learn more complex functions, we need to use many such perceptrons connected together. Such a setting is called a multi-layer perceptron (MLP) or feed-forward neural network (FFNN). These networks are also known as feedforward networks because all information is passed forward from one layer to another. It is the simplest form of neural network.

Figure 2 shows an example of MLP. In this network, the set of neurons can be thought of as divided into sequences of distinct layers. The leftmost layer is known as the input layer, while the rightmost layer is the output layer, and all the intermediate layers are the hidden layers with many neurons stacked together. There can be one or more hidden layers in an MLP, allowing the computation of more complex functions. It can be thought that the input to each neuron in layer i is the output of every neuron in layer $i-1$. A neural network with total n layers is called n-layer FFNN. Each layer in the NN is numbered sequentially from 1 to n, containing $n-2$ hidden layers.

Let us understand the feed-forward procedure of the n-layer MLP. Let the output of each neuron j in layer i is given by $a_{i,j}$. Let there are m neurons in the $i-1^{th}$ layer, then

$$a_{i,j} = f_{i,j}(\sum_{k=1}^{n} w_{i,j,k} a_{i-1,k}) \,, \tag{5}$$

where $w_{i,j,k} \dots w_{i,j,n}$ and $f_{i,j}$ are respectively the weights and activation function of j^{th} neuron in the i^{th} layer. $a_{i-1,k}$ denotes the output of the k^{th} neuron at $i-1$ layer, which is the used as input

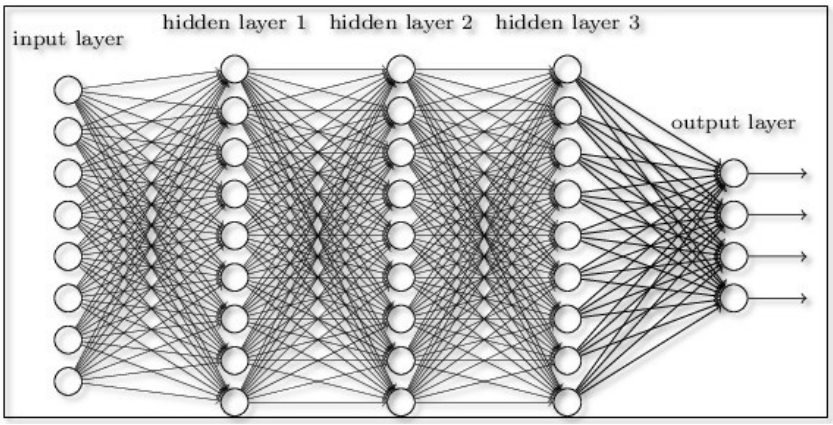

Figure 2: An example architecture of the multi-layer perceptron, taken from [43]

to j^{th} neuron in i^{th} layer.

Using vectorized notation, equation 5 can be re-written as:

$$a_i = f(W a_{i-1}), \tag{6}$$

where f is applied to a vector of inputs (i.e. the output of all neurons in layer $i-1$) and produces a vector of outputs (i.e. the output of all neurons in layer i). Here the activation is being applied to all the neurons in layer i to produce a vector of output. The activation function with scalar input and output can also be applied element-wise. Lastly, as we will see in Section 4.2, the proposed model effectively uses MLPs for the classification tasks.

2.1.3 Training Neural Networks

The process of learning from data is referred to as training. Like other deep learning models, NNs require training during which the value of loss function is optimised. This is done by optimizing the network parameters and weights W in the case of NNs.

The training starts off with the neutral network initialized to random weights. Hence the training process might start off with a high loss value and subsequently decreases to much lower at the end

of training. This would mean that the model output is becoming close to the actual output, which we want.

There are many commonly used training algorithms, but they all are variants of gradient descent, depending on the chosen loss function. The gradients are calculated using a technique called back-propagation.

Backpropagation Backpropagation is the learning mechanism that allows the iterative adjustment of weights in the network, with the goal of minimizing the cost function [44].

In each iteration, the gradient for all neurons in the final layer is calculated. Let the total number of layers be m. The gradient is the derivative of the loss function. Hence, there is a hard requirement - the weighted sum and the activation function must be differentiable. Then, the calculated gradients are stored, and the algorithm moves to the penultimate layer. Now, for each i^{th} neuron, x_i^{m-1}, in layer $m-1$, the gradient of each unit x_j^m in the final layer is calculated with respect to x_i^{m-1}, which is computed by taking the derivative of activation function of x_j^m, multiplied by the weights connecting x_i^{m-1} and x_j^m. Finally, the total gradient for x_i^{m-1} is the summation of the derivative of the loss function with respect to that unit times the derivative of that unit with respect to x_i^{m-1}, across all units in the following layer m. Hence, the gradient calculation is the basically successive application of the chain rule. The algorithm is given by:

$$\begin{aligned} \frac{\partial L}{\partial x_i^{m-1}} &= \sum_{x_j^m \in \text{layer } m} \frac{\partial L}{\partial x_j^m} \times \frac{\partial x_j^m}{\partial x_i^{m-1}}\,, \\ &= \sum_{x_j^m \in \text{layer } m} \frac{\partial L}{\partial x_j^{(m)}} \times \frac{da_j^m(t)}{dt} \times w_{ij}^{(m-1)}\,, \end{aligned} \tag{7}$$

where a_j^m is the activation function of x_j^m, calculated at previous iteration and w_{ij}^{m-1} is the weight

connecting x_i^{m-1} to x_j^m. t is the pre-activation received by $a_j^m(t)$ as input ($t \in \mathbb{R}$). L is the loss function.

Loss Function The loss function, also known as the cost function (sometimes referred to as the error function), denotes how good the neural network is for a certain task. This function maps the value of one or more variables onto a real number. Commonly a loss function is calculated for each data point using the network's output (denoted by $\hat{y}$), and its expected output (also known as ground truth, denoted by y), averaged over a large subset of the available data, known as the training set. There are several possible loss functions. The loss function used in this work (see 5.1) is discussed here below.

A common loss function used for binary classification tasks is the binary cross entropy (BCE) loss (also known as log-loss). BCE loss function measures the performance of the classification model whose output is a probability value between 0 and 1. The BCE loss function is given by:

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(p(y_i)) + (1 - y_i)\log(1 - p(y_i))\,, \tag{8}$$

where y_i stands for the actual observation value, $p(y_i)$ denotes the probability output for the binary classification task by the network for i^{th} input and n stands for the number of examples in the data.

As we can see from equation 8, the loss increases as the predicted probability diverge from the actual label. For example, predicting a probability of 0.0001, when the actual observation label (ground truth), 1, would be bad and result in a high loss value. As we will see in Section 5.1, all the loss functions used by different types of networks in this project are BCE loss.

Another example of the loss function is the average of the square of euclidean distance between y and $\hat{y}$, over n training examples (this is also known as mean squared error (MSE) loss), given by L:

$$L_{MSE} = \frac{1}{2n} \parallel \sum_{i=1}^{n} (y_i - \hat{y}_i) \parallel^2 , \tag{9}$$

where y stands for the actual output, and $\hat{y}$ stands for the output we got using the input x_i in NN.

Gradient Descent Gradient Descent (GD) is a commonly used optimization algorithm for training ML models. GD uses the update rule, given by Equation 10, to iteratively update each parameter in a network (weights w in this case).

$$w^{t+1} = w^t - \epsilon \frac{\partial L^t}{\partial w} , \tag{10}$$

where w^{t+1} is the updated parameter at update step $t+1$, using the parameter value w^t and loss function L^t calculated using all network parameters, at step t. ϵ is a hyper-parameter, known as learning rate, which is hard-coded before starting the training process. This is done using hyper-parameter search (see Section 5.2). The learning rate controls how quickly the model can be adapted for the problem at hand.

Calculation of Equation 10 requires the calculation of loss function L, which in turn requires iterating through the entire training set at each update step. This process is called Batch gradient descent (BGD). However, for a large number of training data (which is usually the case), BGD is slow and computationally expensive. Alternatively, a more efficient way used in this thesis is to calculate the gradient for each small mini-batch of training data. That is, the training data is divided into many tiny groups (i.e. mini-batches) initially, and each mini-batch receives one update. Hence is known as mini-batch gradient descent. Consequently, a smaller value of min-batch size results in faster convergence, at the cost of noise in the training process. On the other hand, a larger value of the mini-batch size converges slower but has less noise in the process.

This gives rise to the challenge of choosing another hyper-parameter, the appropriate mini-batch size,

according to the data and the model. Larger batch sizes require fewer updates and give overall results in less time. This is because the modern-day training process is carried out in GPUs (including my model), compared to Central Processing Units (CPUs) which usually execute software applications. GPUs can perform numerous small tasks in parallel. So for each batch update, multiple batch elements can be operated on in parallel. Further, the GPUs vectorise the mini-batches, which tend to be faster than looping through each batch element individually.

Therefore, the above reasons generally make larger batch sizes more efficient. The mini-batch size is usually selected as the one which fits the memory requirement of the GPU used. In our case, we select a large size that fits the GPU memory [45]. This simplifies the tuning of this hyper-parameter and is sufficient for most of my use cases.

Next, as mentioned earlier, choosing the correct learning rate is important. A large learning rate may cause the model to converge too quickly to a sub-optimal solution, while a too small value will result in slow convergence or the process getting stuck. There are several ways to select a learning rate, which can be theoretically sound and practically heuristic. The learning rate can be fixed or adaptive, gradually changing through the learning process. In this project, a fixed learning rate $\epsilon = 10^{-3}$ is selected. It is a standard choice for learning rate and works well for many cases. More discussion on this presented in Section 5.2.

Lastly, in this thesis, I have used adaptive movement estimation algorithm (or Adam) [46], for training setting-1 (explained in Section 5.1). It is an extension of the gradient descent optimization algorithm. It is an efficient stochastic optimization method, only requiring first-order gradients, with little memory requirement.

2.1.4 Overfitting and Regularization

After the network is trained, it is also necessary to analyse the training and the performance of the deep learning model. In this context, there are two very fundamental and important concepts - bias

and variance. These two terms are also related to the two pitfalls which must be avoided during model training which are - overfitting and underfitting. I will discuss these terms and how to avoid them in this section.

Bias and variance are numeric quantities defined using model accuracy for understanding and optimizing the training procedure. Let the target the model tries to predict is y when input x is fed. That is, the each datapoint is a set $(x_i, y_i)_{i\geq 1},\ i \in \mathbb{R})$. As seen above during the training model will try to minimize the average loss/error of the model. In the case of regression tasks, it tries to minimize the mean squared error given by:

$$MSE(f) = \mathbb{E}[(f(x) - y)^2]\,, \tag{11}$$

where $f(x)$ is the model's prediction conditioned on input x, and MSE_f is the mean squared error.

During model training, we only have a finite amount of data available. However, we aim to learn the data distribution $\tilde{f}(x)$, and not the data itself. Therefore, usually, the data is divided into two parts, one part is used to train the model, while the other part is unseen to the model and is used for evaluating the performance of the trained model. The latter is known as the test set, while the former is known as the train set. In practice, it is standard to divide data into three parts instead - train, validation, and test set. While train data (or training set) functions as mentioned above, validation data (or validation set) is also used during the training process, which is used to evaluate model fit on the training data while hyper-parameters are tuned. This is also done in this project (Section 6 gives the ratio of split for the datasets used in this project.).

We may assume that the data is the sum of a value from this generated distribution and a noise term, taken from a Gaussian distribution with 0 mean and σ^2 variance, where the probability of a value x is given by:

$$N(x; \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}}\,. \tag{12}$$

Thus, given the input x, the output from the model is given by:

$$\hat{y} = \tilde{f}(x) + \epsilon, \;\; \epsilon \sim N(0, \sigma^2)\,. \tag{13}$$

The variance σ^2 represents the noise in the data and denotes how difficult it is to learn the data distribution. It is also known as irreducible error, as the value of the error function would not drop below the value of σ, even when the generating function perfectly learns the data distribution. Using the notations used here, the bias and variance are formally defined using Equation 14.

$$\begin{aligned} bias(f) &= \mathbb{E}[f(x) - y]\,, \\ variance(f) &= \mathbb{E}[(x - \mathbb{E}[f(x)])^2]\,. \end{aligned} \tag{14}$$

Hence, bias is the expectation of a difference between the model's prediction and the correct value that the model is trying to predict. While the variance is the variability of the model prediction, which is the square of the expectation of the deviation of model prediction from its mean. If it is assumed that the targets are the result of summation of the true underlying function $\tilde{f}(x)$ and the degree of random noise. Then the mean squared error can be expressed as:

$$MSE(f) = bias(f) + variance(f) + \sigma^2\,. \tag{15}$$

From equation 15, we can observe that the MSE will be high if the bias and variance are high, meaning that the model is performing poorly.

To simply put, a high-bias model pays very little attention to the training data and oversimplifies the model. Such a model is said to be 'underfitting'. This would produce high MSE in both train and test sets. On the other hand, if the model has high variance, it pays a lot of attention to the

training data and does not generalize on the data which is unseen to it. It has actually learnt the noise present in the training set. This means it can achieve low MSE when calculated on the training set but will not be able to transfer what is learnt to the test set. Because of this, the MSE on the training set would be very low and very high in the test set. Such a model is known as 'overfitting'. Unfortunately, there is a trade-off between bias and variance.

Commonly, in literature, powerful models are used to achieve low bias, and then additional techniques are employed to minimize the variance explicitly. This additional technique is called regularization. In this project, when training the model, I have employed two regularization techniques in the two training settings in the proposed framework (see 2.1.5 and 2.2). The proof for Equation 15 along with a detailed discussion on bias and variance and the trade-off between them, can be found in [47].

2.1.5 Early stopping

Early stopping is a form of regularization technique used to avoid overfitting when training a model with an iterative method like gradient descent 2.1.3. As seen from Equations 10, the training will only stop when the model parameters are precisely on a local extremum. Practically, this is not likely to occur. Further, because of using mini-batch gradient descent (Section 2.1.3), every batch update has some amount of randomness. Even if the loss reaches an extremum for one update, it is not guaranteed to stay there for the next update.

An alternate approach is to put a stopping criterion during training. We want the model to train as long as possible until the performance starts to plateau. We can use the loss as the metric to monitor the model's performance at each epoch and apply an explicit stopping criterion. So, an obvious choice for the stopping criterion is to stop training when training loss stops decreasing.

This method involves setting aside a fragment of the dataset, called the validation set (as discussed in Section 2.1.4), which is not trained on but is instead used for the purpose of early stopping. So the stopping criterion of the model is the loss on the validation set. Particularly, after each training

epoch, the following steps are followed: (1) The average loss value over the entire validation set is computed. (2) The model parameters are not calculated, and gradients aren't updated. Now, if the validation loss has decreased compared to the previous lowest loss value, then it is concluded that the model has improved; else, it is concluded that the model has not improved. In practice, we stop training only when it has failed to improve the loss for a certain number of epochs. For example, if epoch 40 produced a validation loss of 0.5, and the epochs $41 - 47$ all produce loss > 0.5, then we would stop the training, and the network parameters (weights in this project) are set to values received right after epoch 40.

Lastly, this is an effective regularization method because, if in an epoch the model's train loss is decreased but does not generalize well, i.e. the validation loss is not decreased, it would mean the model is overfitting, which triggers the stopping criterion. Since the network parameters are reloaded in from the last epoch where validation loss has decreased, the effect of all intervening epochs, which could have decreased the training loss but not the validation loss, is omitted, which may have caused model overfitting.

2.1.6 Hyper-parameter Search

In deep learning, hyper-parameters are the model parameters that are external to the model and control the model's learning process, unlike non-hyper-parameters, which are internal to the model (such as network weights), they are not derived via training. The value of the hyper-parameter needs to be set manually before training begins by the engineer. This is usually done based on expert knowledge or hyper-parameter search.

Hyper-parameter search (or hyper-parameter tuning) [48], refers to the process of searching for optimal values of the model hyper-parameters that could result in the most skilful predictions by the model. Hence, hyper-parameter search requires a separate optimization procedure. Understandably more the hyper-parameters of an algorithm that need to be tuned, the slower will be the tuning

process. Therefore it is desirable to select a minimum subset of model hyper-parameters to search or tune. It is also important to note that not all hyper-parameters are equally important. Some hyper-parameters have an out-sized effect on the performance of the model compared to others. Section 5.2 discusses hyper-parameters and the search procedure relevant to the work in this thesis.

2.1.7 Convolutional Neural Networks

A convolutional neural network (CNN), or covnets for short, is a variant of the FFNN conventionally used for image recognition tasks. It exploits temporal correlations in the input data and performs multiple convolutional transformations and non-linear activations to hierarchically extract abstract high-dimensional global features from spatially-local data points [49].

It contains a three-dimensional arrangement of neurons instead of the standard two-dimensional array used in the above variant. Here every neuron in layer i receives input from a subset of neurons from layer $i-1$ which are 'close' to each other, as opposed to the case of FFNN where input from every neuron in layer $i-1$ is received. Further, unlike FFNN, each neuron in layer i uses the same set of weights. Each neuron in layer i gets input which are different regions from layer $i-1$. Hence, it has constant weights equal to to the region size in the previous layer. Consequently, it requires making the following assumptions:

- The relevant features are entirely contained within these regions, which are 'close' to each other. This lets CNN only input a fixed number of neurons from layer $i-1$ to layer i.
- Since features in a region depends only on the neuron in that region, allowing the use of fixed weights by all the neurons in layer i. So the weights are shared across different regions.

Because of these assumptions, the number of weights that needs to be calculated and the connections are reduced, thereby making CNNs computationally more efficient. This is because, as seen in Section 2.1.3, the forward and backward passes during the training of NN require linear time with respect to

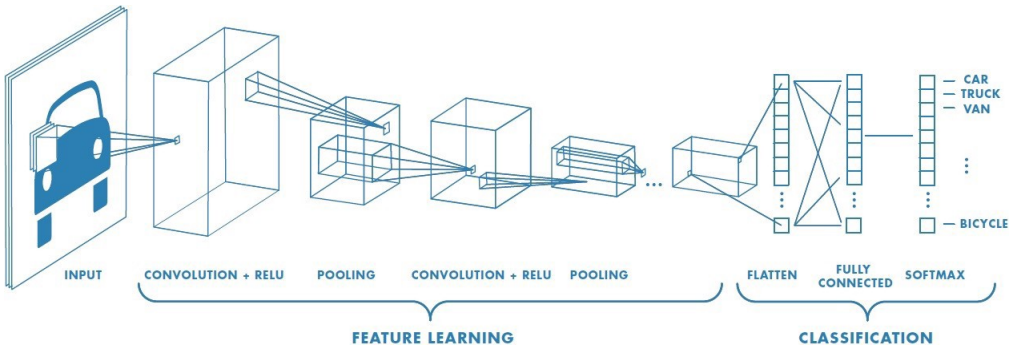

Figure 3: General architecture of Convolutional Neural Networks. The input is a set of points in space. Here it is a two-dimensional grid with each point having a scalar feature value (it is also possible to have three-dimensional input in advanced architectures). Each convolutional layer produces a feature matrix, which increases the number of features per spatial point, and the pooling layer decreases the number of spatial points. The output of the network is produced by a fully connected layer, which is a form of a single feature vector with no spatial structure. Image is taken from [50]

the model size. CNN is more compact, thereby reducing the time taken for training and prediction from trained models. As we have seen in Section 2.1.4, a more compact model is more likely to avoid the pitfall of overfitting the training data, so CNN is more resistant to overfitting.

CNN also reduces the chance of overfitting during the training process. Practically, several sets of weights could be applied to each region. Each set of weights is called a filter in the CNNs context. Hence, a filter has a 2-dimensional shape. A general model of CNN contains four components: (1) convolution layer, (2) pooling layer, (3) activation function and (4) fully connected layer, as shown in Figure 3.

The convolutional layer is central to the CNN, which gives the model its name. In CNNs' context, convolution is the dot product between a filter-sized input region and the filter, which is summed, resulting in a single value. Specifically, the filter is applied systematically to each overlapping part of the filter-sized region of the input data, from left-to-right and top-to-bottom. This is shown in Figure 4. Hence if we have k filters, then each region in layer $i-1$ gives rise to a k-dimensional vector as output.

Often convolutional layer's output is passed through an activation function to introduce non-linearity.

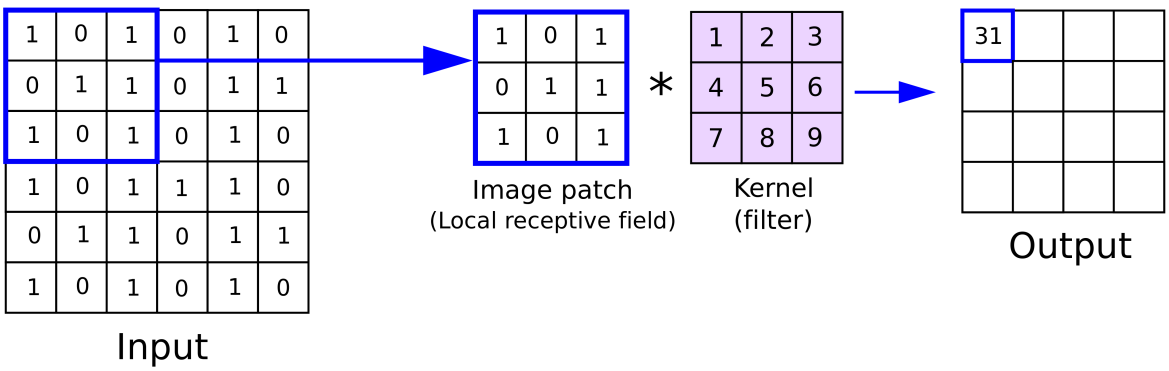

Figure 4: An example CNN convolution process on an input image matrix. At each application of the filter (also known as the kernel) to a filter-sized region, the filter is convolved with a filter-sized region; the received values are summed to produce a single value output. Then the filter is slid by one square over the image from left to right, from top to bottom, and the same calculations are performed to finally produce the output of the appropriate size. In this example, a 6×6 image matrix is convolved with a 3×3 filter to produce a 4×4 output matrix. Figure is taken from [51]

ReLU (discussed in Section 2.1.1) is a popular choice for this case.

The next component is the pooling layers. The pooling layer performs the down-sampling operation, which is applied after the convolution layer, thereby reducing the spatial dimension. The spatial dimension defines the input regions, like x and y coordinates of a pixel in an image, rather than the dimension that defines features for a region, like colour channels in the images.

Hence, the convolutional layer usually increases the size of the feature dimension (depending on the number of filters), which is often followed by a pooling layer reducing the spatial dimension. This way, a spatially structured input after multiple passes in the convolutional layer and pooling layer squeezes into the output feature vector without spatial structure. The pooling layer can be of different types: max pooling, average pooling and sum pooling. Max pooling is a common choice for the pooling layer where the largest element from the region is the output.

Lastly, the fully connected layers operate on a flattened input where each input is connected to all the neurons. They are commonly present towards the end of the CNN network and are used to optimize objectives. For example, in the case of image classification, the CNN can be viewed as a

combination of two parts: (1) feature extraction and (2) classification. The convolutional layer and pooling layer together perform the feature extraction task, while the fully connected layer performs classification and generates the final output.

The CNNs are inspired by brain research in the 1950s and 1960s by D.H. Hubel and T.N Wiesel on the brain of mammals suggesting a new model for how mammals perceive the world visually [52]. Inspired by this and consequent research works, Bengio, Le Cun, Bottou, and Haffner introduced the first CNN in 1998 and called it LeNet-5 [53]. The model was able to classify hand-written digits. Later on, after the success of another CNN network called AlexNet [54], CNNs got huge popularity in three major fields - (1) Image classification [55], (2) object detection [56] and (3) segmentation [57], and many advanced models have been proposed over the years. In successive years, many advanced CNN models have been proposed, which have given state-of-the-art performance in many areas, including image classification, object tracking, object detection, segmentation, human pose estimation, text detection, visual saliency detection, action recognition, scene labelling, visual question answering, speech and natural language processing, etc. [58].

The efficiency of CNNs compared to FFNN and the ability to avoid overfitting has been a major contributor to the success of NNs and deep learning in general. Hence, CNNs are likely to remain a central part of NN research in the foreseeable future.

In this project, the encoder has employed two different advanced architectures of CNN. The first is VGG19 [59], used in our encoder (discussed in 3.3.1), which uses max-pooling in all the pooling layers. It is an advanced CNN model, which is 19 layers deep, and pre-trained on more than a million images from the ImageNet database [60]. Because of this, the network has learned rich feature representations of a wide range of images and its frozen copy is used in this project.

Secondly, the I3D network [61] is also used in the encoder in the proposed model (discussed in Section 3.3.1), which is a two-stream inflated 3D CovNet based on 2D ConvNet inflation. That is, its filters and pooling layers are expanded into 3D. This leads to seamless spatio-temporal features

being learnt by the network. This is significant to the case of video annotation as, unlike images, video feature extraction will need to carry out for both spatial and temporal dimensions (more discussion on this is given in Sections 5.3 and 3.1.2).

2.1.8 Recurrent Neural Networks

Recurrent neural networks (RNNs) are another variant of FFNN (described in 2.1.2), adapted to work for time series data or data that involve sequences. While ordinary FFNN are only meant for independent data points, they are not suitable for sequences where one data point depends on the previous data point. That is, each input (x_i) depends on those that came before ($x_{<i}$). Hence the FFNNs were modified to incorporate the dependencies between the data points. RNNs use a concept of memory that helps them store the state or information of previous input to generate the next output of the sequence. That is, at each time step, it processes the input, and the output is produced conditioned on both the input and the internal state. It is then used to update the internal state. The idea is that the internal state encodes the relevant contextual information. This allows its output at each step to use information not only from the input but also the history of inputs up to that point.

Let us understand the training procedure in RNN. The RNN is trained using the backpropagation algorithm explained in Section 2.1.3. The RNN persist information using loops, where information from the previous time step is carried forward in future time steps. The right figure in 5 shows an unrolled version of the one-unit RNN for a better understanding of the passage of information. Each cell in RNN is a simple FFNN (described in Section 2.1.2), also known as the vanilla RNN. Consider a vanilla RNN similar to the one in Figure 5; the input to the unrolled version is the concatenation of all elements in the input sequence, and the output is the concatenation of all elements in the output sequence.

In the case of an FFNN, each sequence will be treated as a vector by concatenating all its elements,

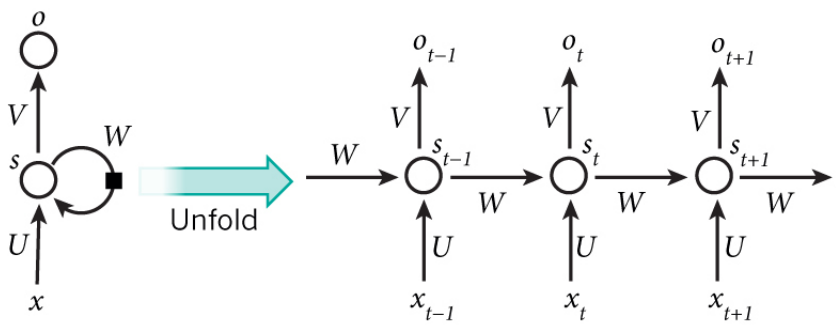

Figure 5: Architecture of one-unit Recurrent Neural Network. From bottom to top: first is the input state, followed by the hidden state, and lastly, the output state. The diagram on the left shows the compressed version of RNN, while the diagram on the rights shows unfolded version of RNN. W, U and V are all network parameters which can be trained. Taken from [62].

but RNN offers several advantages over this. Firstly we can feed sequences of arbitrary length in it, whereas FFNN would require each different network for different concatenated sequences. Secondly, it reduces the model size by reusing the same weights for each input step. This reduction in model size offers benefits the same as those mentioned in Section 2.1.7. Lastly, the RNN is more suitable for cases where data points aren't independent of each other.

Similar to the case of the CNN, the RNN makes some assumptions about the structure of the data, which is - each sequence element is of the same type and so can be processed by the same set of weights. This allows us usage of a constant number of weights for all the sequence lengths.

There are multiple common RNN architectures. Long-short term memory (LSTM) [63] and gated recurrent units (GRU) [64], are two popular architectures. The architecture used in the encoder in this project (discussed in Section 3.3.1) is the GRU. More discussion on GRU is presented below in Section 2.1.9.

Lastly, due to several advantages RNN offers, as listed above, they have contributed to a significant number of domains. Perhaps most notable is their applicability in natural language processing (NLP) tasks. Whether language is represented as words, letters or speech acoustic signals, it is natural to interpret them as sequences. Thus, this sequence can be very long and of variable length [65].

2.1.9 Gated Recurrent Units

Gated Recurrent Units (GRsU) are a type of RNN that help retain the information longer, which was first introduced in [64]. They use an update gate and reset gate, which helps decide what information should be kept from the past and what should be discarded. Figure 6 shows an example architecture of a single-GRU unit.

Update gate: It controls how much information from the previous step should be passed to the next step. This has proved to be highly useful as it can theoretically send all the information to the next step. This is given by:

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \,, \tag{16}$$

where x_t denotes the input at time step t, h_t denotes the hidden state which holds the information of the previous $t-1$ time step, W_z and U_z are the trainable weights and σ is the sigmoid activation function.

Reset Gate (Short term memory): This gate controls how much information from the previous step will be discarded, given by:

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \,, \tag{17}$$

where the notations are the same as above. The calculation for the reset gate is the same as the update gate; only the trained weight parameters are different from the update gates.

Current memory content (h'_t): The reset gate is now used to decide the amount of memory from the past which should be retained, given by the equation 18.

$$h'_t = \tanh(W x_t + r_t \bigodot U h_{t-1}) \,, \tag{18}$$

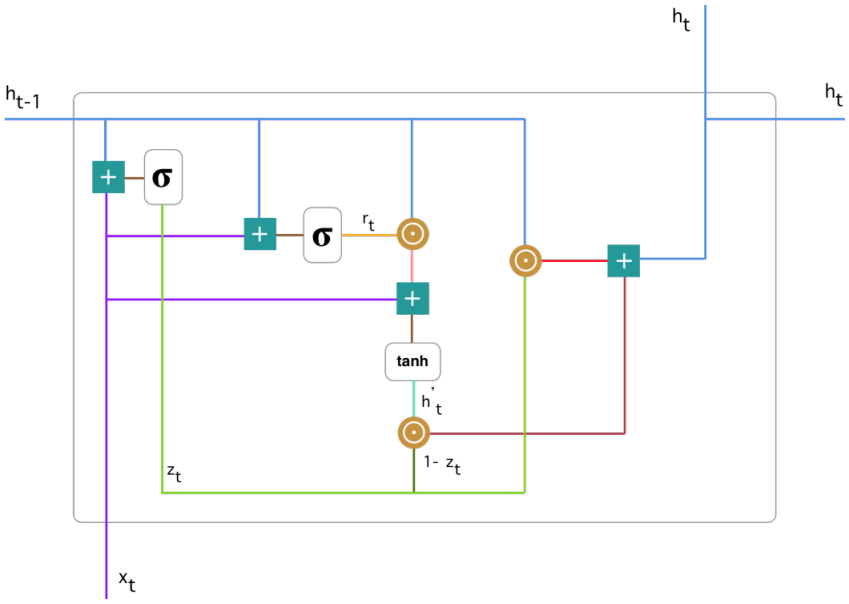

Figure 6: A gated recurrent unit cell. Taken from [66].

where W and U are the trainable weight matrices, and $\bigodot$ is the Hadamard product. $r_t \bigodot U h_{t-1}$ contributes to deciding how much previous memory should be retained. This is added to the current input, and memory content is calculated.

Final memory at the current time step: Lastly, the memory content which needs to be passed to the next cell is given by:

$$h_t = \sigma(z_t \bigodot h_{t-1} + (1 - z_t) \bigodot h'_t)\,, \tag{19}$$

where z_t is the output of the update gate calculated in equation 16.

2.2 Logistic Regression

Logistic regression is a powerful statistical way of modelling a binomial outcome with one or more explanatory variables, which is borrowed by machine learning from the field of statistics. In its most basic form, logistic regression is a method for binary classification, where classes are usually labelled as 0 and 1, or -1 and 1, for mathematical convenience. In the proposed work, binary logistic

regression is used for assigning each fact to positive or negative class, given the input video and model parameters (see 4.2).

Hence, it models the conditional distribution over the output Y, given the inputs X and model parameters W, where Y (in binary classification case, Y is either 1 or 0) is a discrete value, and $X = \{x_1, x_2, \ldots, x_n\}$ is a vector containing discrete or continuous values. Now, the probability of input X belonging to the positive class ($P(Y = 1|X)$), and negative class ($P(Y = 0|X)$) is given by:

$$\begin{aligned} P(Y = 1|X) &= \frac{1}{1 + \exp(-(w_0 + \sum_{i=1}^{n} w_i x_i))}\,, \\ P(Y = 0|X) &= 1 - P(Y = 1|X)\,, \\ &= \frac{\exp(-(w_0 + \sum_{i=1}^{n} w_i x_i))}{1 + \exp(-(w_0 + \sum_{i=1}^{n} w_i x_i))}\,, \end{aligned} \tag{20}$$

where the parameter $W = \{w_0, w_1, \ldots, w_n\}$ are calculated during training by maximising the cost function (see 2.1.3).

Training and Regularization As discussed in 2.1.3, the logistic regressor is also trained and requires regularization to reduce the variance of the model and avoid model overfitting (see 2.1.4). The cost function for the logistic regressor is given by:

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}(y_i \log(p(y_i)) + (1 - y_i)\log(1 - p(y_i)))\,, \tag{21}$$

where $p(y_i)$ is the prediction from the logistic regressor, while y_i is the actual output, corresponding to input x_i.

In the context of logistic regression, there are two popular choices for regularization - ridge regression

and lasso regression. Lasso regression (also known as l1 regression) is used in this thesis (Section 5.1.2, which adds an extra term to the cost function, which is the sum of magnitudes of all the coefficients in the model, given by:

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}(y_i \log(p(y_i)) + (1-y_i)\log(1-p(y_i))) + \lambda \sum_{j}^{p} \parallel w_j \parallel_1 , \tag{22}$$

where the first term is the same as the logistic regressor's loss term, and the second term is a penalty whose size depends on the total magnitude of all the network parameters. The term λ is known as the tuning parameter that adjusts how large a penalty term will be. If $\lambda = 0$, we end up with no regularization.

2.3 Logic

In this thesis, I aim to annotate videos with knowledge graphs. Knowledge graphs are a knowledge representation formalism based on graphs, which can be used to model natural language understanding. Though they are not completely based on logic, it has been shown that this is a graph-variant of first-order logic, which is equivalent to standard first-order logic [33, 67]. This equivalence is useful to handle logical deduction and inference, which is useful in the case of video annotation tasks. Below I discuss these concepts.

Logic is the study of correct reasoning or good arguments, which is often defined in a more narrow sense as the science of deductively valid inferences or logical truths.

Logical Notations The way in which logical concepts and their interpretations are expressed in natural languages is often very complicated. Hence, a system of streamlined notations is developed to express the logical information using logical symbols. There exist many logical notations with varying syntax and semantics. The one which is relevant to this project is a subset of the system

known as predicate logic, which is generally considered an extension of a simpler system called propositional logic.

2.3.1 Propositional Logic

A propositional logic statement is formed using a combination of propositions, which is written using upper case letters, and connectives, each of which has its own special symbol. So, the vocabulary of propositional logic is a set of propositions and the set of logical connections. The propositions consist of statements that can either be true or false, and the connectives are the logical relations between these statements. If we label each proposition as true or false (or formally as 1 or 0), it also assigns a truth value to the statement itself. That is, the truth value of a statement of propositional logic is completely determined by the truth values of the propositions it constitutes. Formally, the propositional logic statements are functions from the truth of their constituent propositions to $\{0, 1\}$. The structure of this function is determined by the connectives in the statement, which is a binary truth function. Commonly used 5 connectives are: $\wedge$ expresses 'and', $\vee$ expresses 'or', $\neg$ expresses "not", $\Rightarrow$ expresses 'entails', and $\Longleftrightarrow$ expresses 'is equivalent to'. Brackets can also be used in the same way as algebra to show the order in composing the connectives. For instance, the statement "If Mary is outside, and it's sunny outside, then Mary will be using sun protection" can be expressed as:

$$\begin{aligned} &(P \wedge Q) \Longleftrightarrow R \\ &P := Mary\, is\, outside \\ &Q := It\, is\, sunny\, outside \\ &R := Mary\, using\, sun\, projection \end{aligned} \tag{23}$$

Let $P = 0$, $Q = 1$ and $R = 0$, then the statement is evaluated as true:

$$((0 \wedge 1) \iff 0) = (0 \iff 0) = 1 \tag{24}$$

2.3.2 Predicate Logic

The propositions in propositional logic have no internal structure. Predicate logic replaces propositions with logical atoms. So, like propositional logic, logical atoms have a truth value, but they also have an internal structure. This internal structure allows predicate logic to express certain statements which cannot be expressed in propositional logic. The vocabulary of predicate logic is a set of predicates, a set of constants (together forming a logical atom), and a set of logical connectives. Each predicate has a fixed number (or arity of the predicate) of constants with which it can form an atom. A $n-$ary predicate (i.e. predicate with arity n) can be considered as a function from $n-$tuples of constants to $\{0, 1\}$. For instance, the statement "if Mary doesn't like Sheldon and Sheldon doesn't like George, then Mary likes George" speaks about the same relation holding (or not holding) between 3 different pairs of people. This statement cannot be expressed in propositional logic alone but can be in predicate logic as follows:

$$\neg likes(Mary, Sheldon) \wedge \neg likes(Sheldon, George) \Rightarrow likes(Mary, George) \tag{25}$$

Here the atoms (or negated atoms) are $\neg likes(Mary, Sheldon)$, $\neg likes(Sheldon, George)$ and $likes(Mary, George)$, also known as literals.

The term 'logical caption' used in this thesis refers to the conjunction of closed literals formed from either binary or a unary predicate. The former indicates class membership, as $P(a)$ can be interpreted to mean that individual a belongs to the class P, and the latter are called relations.

Class membership is expressed as triple in this thesis, such that logical captions is a set of logical triples. Such expression is obtained by defining classes as special individuals and introducing a special membership predicate isA, which is binary. Then, instead of writing $P(a)$, we can write $isA(a, P) >$.

When class predicates are represented using the isA relation and no predicates are used with arity > 2, a logical caption can then be seen as a graph whose edges are relations and nodes are individuals. This is termed as **knowledge graph**. No caption in the dataset used in this work (see 6) contains a predicate of arity > 2, and so every caption in the dataset only admits an alternative interpretation as a knowledge graph. As mentioned above, it is occasionally useful to consider logical captions in this form.

2.3.3 Knowledge Graph

The term knowledge graph was introduced by Google in 2012 to refer to its general-knowledge base, though similar approaches have been around since the beginning of modern AI in areas such as knowledge representation, knowledge acquisition, natural language processing, ontology engineering, and the semantic web [68].

Knowledge graphs are simplified semantic networks. As seen in Section 2.3.2, a knowledge graph (KG) specifies the individuals (i.e. objects) present in the video and the facts that hold true of these individuals. Each fact contains a predicate and the corresponding arguments: $\langle subject, predicate, object \rangle$ triples (a.k.a binary relations) in the case of binary predicates, and $\langle subject, predicate \rangle$ pairs (a.k.a unary relations) in the case of unary predicates. Hence, a fact expresses a relation between two individuals (like $play(girl, music)$) or an attribute of an individual (like $little(girl)$). Remember from from Section 2.3.2, a knowledge graph contains predicates with arity≤ 2. A visual representation of a KG is shown in Figure 12, where nodes representing individuals are shown in blue, nodes representing attributes are shown in orange, and nodes representing relations are shown in red colour.

So, in this thesis, the focus is restricted to unary and binary relations only.

Knowledge Graphs and Logical reasoning Logical Reasoning helps in establishing mathematical arguments as valid or invalid. Therefore it is advantageous to use logical captions compared to natural language captions, as the former enables logical reasoning. Using logical reasoning, a set of true logical atoms can be used to infer additional atoms to be true using a set of rules. For example, given the set of formulae : $\{f(s), g(a)\}$, and the rule $k(X) \leftarrow f(X)$, then we can infer additional true atom $k(a)$.

A semantic ontology is a set of such rules. So, semantic ontologies can augment a given amount of information with background knowledge. This makes logical inference possible when required. For example, if a video depicts "a man playing the guitar", i.e. the logical annotation for the video is $play(man, guitar)$, then using semantic ontology rules, it is possible to make inferences like $(man(X) \leftarrow person(X))$, i.e. it captures commonsense of human reasoning that all men are persons, and so the caption $play(man, music)$ is interpreted in a more holistic, human-like sense. Further, linking nodes from the KG to a semantic ontology (or knowledge base) facilitate automatic word-sense disambiguation, which is necessary in the case of polysemous words, which are words with multi distinct senses (see 6.4). Though we do not perform logical inference in this work, it can very easily be performed by extending the model. The knowledge base used here is WordNet [69] described in Section 2.4.

2.4 WordNet

WordNet [69] is a large lexical database of English. In the database, all the nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (**synsets**). Each synset expresses a distinct concept. The synsets are interlinked through conceptual semantic and lexical relations. Hence, it semantically links all words in the English language.

Synonyms (like between words shut and close or car and automobile) are grouped into the unordered set (synsets). Each WordNet's 117000 synsets are linked to other synsets utilizing a small number of conceptual relations. The synsets are also accompanied by a brief definition illustrating the use of the synset member. Polysemous words (word forms with several different meanings) have distinct synset, and so each form-meaning pair in WordNet is unique [70]. In Section 6.4, a discussion on how WordNet is used to link to the nodes in KG is presented.

2.5 Local Closed-World Assumption

The ground truth obtained from the dataset only contains positive facts. In a naive implementation where only the positive facts are used for training, the model will predict everything to be true. Under this implementation, the model would produce a very low loss value and very high probability for every possible atom, giving perfect accuracy. This is, of course, irrational as the model is expected to predict negative facts where appropriate. Hence, we are also required to train the network to produce low probability values for all those atoms which do not hold true of the given video.

However, making a closed-world assumption (CWA) on the dataset, i.e., assuming all the atoms which are not in the dataset are false, is incorrect in my case. This is because CWA is only applicable when the dataset has complete knowledge (about the domain), which is not the case with the dataset I am using. A suitable caption for a video contains only the most relevant information about the video, but it does not contain every possible truth about the video, and so there are truths about the video which is not present in the captions.

For example, a video depicts a man eating a carrot in a big kitchen. The caption may just only say "A man is eating" and not include captions like "A man is in a large kitchen", even though the latter is true. That is, the atoms in(man, largeKitchen) are true but wouldn't appear in the caption. Hence as illustrated by this example, making a closed world assumption is too strong in our case.

Instead, a weaker assumption known as a local closed-world assumption (LCWA) [71] is more ap-

propriate in my case. Under LCWA, it is assumed that if an atom is not present in the caption of the video but is:

1. sufficiently close to an atom which is present in the caption. Sufficiently close mean differing only by one of the two arguments. Since we are only considering unary and binary predicates (as discussed in 2.3.3), all facts will ≤ 3 components, which is the predicate and up to 2 arguments.

2. negation of this atom does not contradict the logical caption itself.

Then this atom is false. Hence, the literal formed by negating the atom can be added in the caption. Therefore, if an atom $P(a.b)$ is present in the caption, then for each constant $a' \neq a$, such that $\neg P(a', b)$ is not in contradiction with the logical caption itself, we can add the negative literal $\neg P(a', b)$ and similarly, $\forall b' \neq b$.

Let's consider the above example again; here, we assume that it is false that "a man is eating mango" or "a woman is eating carrot", but we don't make any assumption about "a woman eating mango". LCWA relies on the assumption that any 'relevant' atom will be contained in the caption if it holds true for the video. Here 'relevant' refers to the predicate and individual that are present elsewhere in the video caption.

For instance, if a video depicts "a man and women both eating carrot", we will assume that the literal $eats(man, carrot)$ and $eats(woman, carrot)$ will appear in the caption. That is, the natural language caption would be something like this, "a man and woman are eating carrot", instead of only "a man is eating carrot". This means that if $eats(woman, carrot)$ is not present in the caption, then we assume it does not hold true for that video. So, we can add its negation to the ground truth logical caption, $\neg eats(woman, carrot)$. Similarly, we can also add $eats(child, carrot)$, $\neg(woman, mango)$, etc. However, we cannot add $\neg in(man, bigRoom)$ because it differs from the original atom in both subject and object argument, and so it does not qualify as sufficiently 'relevant'. To put it another

way, it is expected from the natural language caption to reflect the presence of other people in the video, as it has already provided some knowledge about the humans who are present, but it is not necessarily expected for the caption to include information about the layout of the room the man is in because the caption has not mentioned anything about the room layout anywhere.

Though the LCWA assumption is not faultless, it is a standard in this field to evaluate the accuracy of logical predictions [72, 73]. It provides a simple way to prevent the network from learning to predict all atoms as true.

I have used LCWA to create a set of negated facts, referred to as F, which the model also learns to predict. Essentially, to obtain the corrupted version of facts, I replace the predicate with a different predicate from the vocabulary. For example, the fact $eat(man, carrot)$, can be corrupted to $\neg fold(man, carrot)$, which is then added to F. Then while computing the loss function for training the neural model in 4.2, I take the set T of facts to be those given by the logical caption (after it has been passed through the Stanford NLP syntactic parser, see Section 6.4), and take a set of F of negative facts which are received by corrupting the facts as described here.

3 Related Work

3.1 Natural Language Generation from Visual Contents

Natural language models for visual understanding systems have been a popular research paradigm. In this regard, both image and video captioning have received a fair bit of attention, where attempts have been made to develop a semantic understanding of visual input's content and automatically translate them into natural language.

The state-of-the-art models have shown promising results. However, there are several shortcomings to them and they are far from being considered a solved task. Introduction (Section 1) highlights some of the major shortcomings of using natural language for annotation tasks. Adding to that shortcoming, the models are often hard to interpret, that is, it is difficult to know why the particular NL sentence was generated. Existing methods show good performance on the datasets collected from the same domain, but often the generalization capability is not satisfactory.

Nevertheless, the research in this area has produced important results, which have inspired the area of structured representation of annotation data. Here I discuss some of the important research in the area of image captioning and video captioning tasks.

3.1.1 Image Captioning

Image captioning is the task of describing the visual contents of the image in a natural language by employing a visual understanding system and a language model capable of generating meaningful and syntactically correct sentences.

Following an initial impressive result of [74], the vast majority of current approaches use a combined system of CNN and RNN (both architectures are discussed in detail in Sections 2.1.7 and 2.1.8 respectively) both of which have shown very strong results over the past decade. Since then, the

task has been improved upon using several additional techniques. [75, 76] uses an attention-based network where the images were first encoded using CNN. This is followed by a language generation phase, where words or phrases are generated from the encoded output of the CNN. At each step of the language generation, salient regions of the image are focused, and captions are updated dynamically until the end of the state of the language generation phase. Techniques such as deep reinforcement learning (a sub-field of neural network research) have also been proposed for this task [77, 78].

Current state-of-the-art models have shown remarkable progress in recent years, which can produce accurate natural language captions for a diverse range of input images. See [21, 79] for an up-to-date and comprehensive review of the existing deep-learning-based techniques. However, a robust image-captioning method that can generate high-quality captions for nearly all images is yet to be achieved.

3.1.2 Video Captioning

Traditionally, video captioning tasks aimed to collect natural language phrases that could explain the video contents. It is a more complicated task than image captioning, as it needs to capture the motion trajectory of related objects, causality of events, acts, and respective objects to understand the video. Because of this, one needs to capture visual, spatial and temporal features, unlike image captioning, which only requires capturing visual features.

In all these research fields, large datasets have played vital roles in the captioning system. At the beginning of video data collection, datasets were created with video-specific categories such as cooking, makeup, movies etc [80–82]. However, this made the models lack proficiency in operating across all domains. This moved the focus towards open-domain datasets, among them MSVD [83], and MSR-VTT [84] have been the most prominent [85–87]. Hence, in this work, I have also utilised these two datasets (detailed in Section 6).

The task of video captioning in recent years mainly adopts the encoder-decoder architecture: (1)

The encoder first extracts features from video data; (2) In the second stage, a decoder is utilised, which converts the extracted features into NL-descriptions. In the recent past, CNNs and RNNs have come out as the most powerful architectures for these tasks [87] (both the architectures are discussed in detail in Sections 2.1.7 and 2.1.8 respectively). Commonly, the encoder consists of 2D- or 3D-convolutional neural network that could convert the input into a fixed-length vector representation, which is then decoded by an RNN-based decoder (such as LSTM or GRU). Venugopalan et al. [88] first proposed such a framework, where a CNN called AlexNet [89] was used to extract visual features for each frame, and an RNN (LSTM) architecture was used as a decoder to encode visual features as a sequence of words. However, the work did not consider the temporal domain in the encoder. The work tried to represent the information using a single, temporarily folded feature vector which failed to exploit the temporal structure in the video. This lead to wrong sentence generation, as different event and objects in temporal sequence may fuse incoherently.

Later on, several attempts were made to get rid of this bottleneck. The attention mechanism which was popular in other areas was also applied here, which led to improved performance. An early example includes the work by Yoa et al. [90], where the encoder first fed the video through a 3D-CNN followed by an RNN to produce an encoded sequence of feature vectors produced by the CNN. The decoder then took the attention-weighted sum over these feature vectors and used a language model to produce a natural language caption. Several authors such as [91–93] also suggested using both 2D and 3D-CNN for encoding the spatio-temporal features during the encoding phase. The current state-of-the-art approaches are often roughly divided into two categories: (1) pre-trained CNN networks like VGG (for RGB frames) [59] are usually used to extract visual features followed by an RNN (usually an LSTM or GRU) for extracting temporal information (examples include - [94–96]). Here the video is treated as a sequence of visual features, where the RNN tries to encapsulate the context information of a sequence of data into the hidden state of the RNN layer at the final step; (2) In the second method, the models extract visual features using 2D-CNN but mainly rely on the 3D CNN to capture the temporal information in the encoder. Examples include [90, 97].

As we will see in Section 3.3.1, the encoder adpoted in the proposed model in this work is based on a combination of pre-trained 2D-CNN, RNN and 3D-CNN is used for encoding the visual, spatial and temporal features from the video.

Lastly, the quantitative evaluation of captions generated for video or image captioning is a difficult task [98]. They cannot be measured by standard ML evaluation metrics like accuracy and F1-score. Instead specially designed metrics like BLEU [27], METEOR [28], LEPOR [29] etc., are required. Recognising the imperfections of each of these, authors typically report scores on multiple metrics, none of which has a simple and intuitive interpretation, which makes the performance evaluation difficult [2, 21].

3.2 Incorporating structured knowledge in annotation task

As outlined in the introduction, there are strong theoretical and practical reasons why the structured formal representation of knowledge in deep learning is a promising avenue of research. There are notable shortcomings to current annotation methods, which can be efficiently re-explored using formal, structured knowledge. Recognizing this potential, the incorporation of symbolic knowledge has been explored by numerous authors, across a wide range of deep learning tasks, with proposals of several methods to bridge the gap between deep learning and symbolic knowledge [99–103].

Apart from the benefits mentioned in 1, combining symbolic knowledge and deep learning also allows neural networks to learn from existing symbolic techniques instead of having to learn their task from scratch. This got recognised in the context of visual question answering (VQA) in [104] where the DL model was augmented with symbolic techniques for image extraction. VQA is often understood as a computer vision task where a system is given a text-based question about an image, and it must infer the answer [105]. The work in [104] produced the state-of-the-art performance on the Visual Genome dataset. This model used computer vision algorithms to detect visual elements like objects, persons, and relations between them and built the VQA model on top of this. Thereby, their VQA

model only had to reason over this explicit high-level representation of the contents of the image.

Adding to the above advantage, it also makes the deep neural network more interpretable. Being a black box has been a well-known shortcoming of a typical deep learning architecture [106–108]. In these models, it is hard to understand the inner working, and so it is difficult to diagnose the problem when the model fails or when it is behaving unpredictably.

Since a visual input often denotes complex relationships between objects, a structured graphical representation has often been advocated in literature [1, 31, 109, 110]. This is because graphical representation can encode abstract semantic elements, i.e. objects and their interactions, in nodes and edges, respectively, and are visually comprehensible, and it is possible to apply explainable reasoning to them.

This idea started to be explored in image annotation tasks using scene graphs. Johnson et al. first proposed the idea of scene graphs [31] and Real-World Scene Graphs Dataset (RW-SGD), which became the first dataset to be created for scene graph generation and application (image retrieval). A scene graph G is a data structure which consists of directed graphs, defined as a tuple $G = (O, E)$, where $O = \{O_1, O_2, \ldots, O_n\}$ is a set of objects. Each object has the form $o_i = (c_i, A_i)$, where c_i is the category of the object and A_i are the attributes of the object. $E \subseteq O \times R \times O$ is a set of directed edges which are the relationships between objects [111]. Scene graphs became popular with the Visual Genome dataset in 2016 [30], where the authors recognised while existing systems could perform well in perception tasks like object segmentation and classification, they lack the cognitive ability to correctly identify attributes of an object or the relationship between them. The dataset contained 100,000 images densely annotated with scene graphs. Due to the lack of any such structurally annotated video dataset, this project has also leveraged Visual Genome, but for a different task, to incorporate commonsense knowledge in the proposed model. A thorough discussion on this dataset and my approach of using it is given in Section 6.3 and 4.3.3 respectively.

Since then, scene graphs have proved highly successful in tasks such as image captioning [109, 112]

and visual question answering (VQA) over images [110, 113, 114], image retrieval [15, 31] and image synthesis [115]. For example, [109] used scene graph representation of an image to encode semantic information. It developed semantic and spatial graphs and encoded them in feature vectors using Graph Convolutional Network (GCN).

This also inspired the work in video-based tasks, where authors tried to extract structured semantic content. Several researchers such as [116–118] constructed a scene-graph for every video frame followed by inter-frame representation learning to produce holistic video-level features for tasks like reasoning. In [119], the authors proposed VidSGG for producing a structured representation of video for high-level understanding tasks, where they too constructed scene graphs for every video frame. However, a scene graph for every frame may be redundant and may even be computationally expensive for longer video sequences.

Particularly towards the video annotation task, in [1], the authors made the first initial attempt to generate structured descriptions for videos for the automated video annotation tasks. They leveraged knowledge graphs and outside knowledge using linked data ontology (see 2.4) and tested their model on a formally annotated dataset generated using the MSVD dataset.

Knowledge graphs are another form of structured representation, which is of particular importance in this project. There are similarities between a knowledge graph and a scene graph, such as their graphical structure and constituent elements. However, scene graphs are often understood as being visuospatial, i.e., they say what is where in the scene. This is often done by associating the nodes with a bounding box to various entities in a scene in the visual input[30, 120, 121]. To generate the bounding box, the model also needs to predict the coordinates of the entities in the visual input. On the other hand, knowledge graphs can represent any sort of knowledge, so there is no requirement for its node to have a bounding box [122]. This property of a KG is highly desirable because we no more need to make a scene graph for every video frame and can associate only one knowledge graph with one video.

Next, in a structured annotation such as a scene graph or knowledge graph, it is also possible to link nodes to a knowledge base which makes them incorporate commonsense knowledge about the world, which is very helpful while identifying complex situations [1, 2, 30]. For example, if we know a *man* is a sub-type of a person, so only training the model on $man - riding - Bike$ is sufficient, and the model can generalise it to $person - riding - Bike$ on its own. This idea is intuitively promising, and hence in this work, a knowledge graph is utilised to annotate the videos structurally.

In [2], the authors followed the work in [1] and made another attempt to extract KG from a video for the video annotation task. The knowledge graphs' nodes were linked to ontology (WordNet [123]), which made logical inferences like the above example possible. Due to the lack of any suitable dataset, they created a KG-annotated video dataset from MSVD and MSR-VTT datasets as well.

Hence, the work in this thesis is inspired by the above two works [1, 2] and attempts to annotate videos using KGs. [2] has also been used as a baseline for my work. There is still a lack of any KG-annotated manually constructed datasets, due to which the dataset used in this thesis is generated using the framework proposed in [2]. In their implementation, an encoder is utilised for encoding the video. As discussed in 3.1, the state-of-the-art encoders often use a combination of CNNs and RNNs, which is also been used in [2]. Due to its success, my model also adopted this encoder architecture discussed in Section 3.3.1. The encoded video is then fed to a multi-classifier (called an individual-multi-classifier) to identify objects in the video (it uses an MLP (2.1.2) for this task, discussed in 3.3.2). These detected objects were combined with every possible combination of attributes and relationships from their vocabulary to generate a possible set of facts and were fed to another classifier (called predicate-MLP, discussed in 3.3.3) which could classify if the fact is true for the given video or not. The results shown were promising.

Hence, the approach proposed in this work is more close to the baseline model [2], the significant difference being - the possible set of facts is generated using a combination of three conditionally independent multi-classifiers followed by a framework to combine them, which could filter irrelevant

facts at before the next stage. While the linking of nodes to WordNet [123], made logical inference possible, I also introduced a novel framework to use an independent dataset to impart commonsense knowledge in the model output. This also made the model generalize well on unseen data compared to [2]. The work in this thesis has shown significant improvement over the baseline model over several different dimensions (see 7).

3.3 Baseline Neural Model Components

In this section, I discuss three components of the baseline model [2], which has been adopted in the proposed system in this thesis due to compelling reasons discussed in Sections 3.2 and 5.3.

3.3.1 The Encoder

The encoder takes in a sequence of raw and unprocessed video frames as input and outputs an identical-length sequence of encoded video frames. The encoded video frames have much smaller dimensions than the original input frames.

Let the input X be a set of videos, where each video, $x = [x_1, x_2, \ldots, x_n]$ is a set sequence of 3-dimensional video frames forming a 4-dimensional video tensor. In practice, all the operations are batched, so each video tensor contains an additional batch dimension. The encoder f produces a fixed length feature vector $f(x)$ for each $x \in X$.

For a given video $x = [x_1, x_2, \ldots, x_n]$, the encoder first applies a pre-trained version of VGG19 [59] model to each frame x_i, i.e., $vgg(x) = [vgg(x_1), vgg(x_2), \ldots, vgg(x_n)]$, where $vgg(x_i) \in \mathbb{R}^{4096}$. Then the output of the 3^{rd} last layer is taken as the i^{th} frame encoding, ζ_i, as the output.

This is because VGG19 is a CNN trained for image classification of over 1000 possible objects [59]. Hence, the final layer is a vector of size 1000, which is designed to select the most suitable of 1000 possible image classes. So, instead of using this as the output layer, the encoder uses the activations

from the 3^{rd} last layer. This is done as these activations would encode more general features, while the last two layers would be specialised to the specifics of the classification task VGG19 was designed for. This is a standard approach taken by others in this field as well [124–126].

Next, the sequence of frame encodings $\zeta_1, \zeta_2, \ldots, \zeta_n$ is passed through a 3-layer gated recurrent unit (GRU) [64] (i.e. the Encoder RNN), producing a sequence $\overline{\zeta_1}, \overline{\zeta_2}, \ldots, \overline{\zeta_n}$. This makes the first stream of the encoding, given as $f_a(x)$,

$$\begin{aligned} &f_a(x) = [\overline{\zeta_1}, \overline{\zeta_2}, \ldots, \overline{\zeta_n}]\,, \\ &where\ \overline{\zeta_i} = [GRU_{enc}(\overline{\zeta_{i-1}},\ \zeta_i)] = [GRU_{enc}(\overline{\zeta_{i-1}}, vgg(x_i)]\,. \end{aligned} \tag{26}$$

As a second stream, the feature vector is computed from a frozen copy of the I3D network [61]. I3D network is a two-stream Inflated 3D ConvNet. It is based on 2D ConvNet inflation, where filters and pooling kernels of very deep image classification CovNets are expanded into 3D. This makes it possible to extract spatio-temporal features from videos. The encoder utilises the pre-trained checkpoint of RGB NumPy arrays as inputs (trained from scratch on Kinetics-600). This is chosen as its state-of-art action recognition model and is commonly utilised in the task of video captioning.

Finally, the output of the encoder is a concatenation of this I3D feature vector and a weighted sum of the $\overline{\zeta_i}$s', weighted by a learnable n-dimensional vector w. The encoder is defined as:

$$\begin{aligned} f(x) &= w.f_a(x).I3D(x) \\ &= w.GRU_{enc}(VGG(x)).I3D(x)\,, \\ &where\ I3D(x_i) \in \mathbb{R}^{1024} \end{aligned} \tag{27}$$

The value of w is chosen as the network hyper-parameter. Hence the encoder defined by Equation 27, has dimension:

$$dim(f(x)) = 4069 + 1024 = 5120\,. \tag{28}$$

The output from the encoder is referred to as frame encodings.

3.3.2 The Individual Multi-classifiers

The individual multi-classifiers are used for predicting the individuals present in the input video. They are multi-layer perceptrons, which take the video encoding as input and output the probability of the presence of an individual from the vocabulary in video. Hence, the input size of the multi-classifier is equivalent to the video encoding, given by $dim(f(x)) = 5120$ (using encoder defined by Equation 27). It has one hidden layer.

3.3.3 Predicate MLPs

Each predicate is assigned separate a MLP, which contain one hidden layer. The input size is $dim(f(x)) + D$, in case of unary facts and $dim(f(x)) + 2D$ in case of binary facts. Here D is the size of individual(s) vectors (300 in my case).

4 Proposed System

In this chapter, I will formally explain the problem statement, followed by a discussion on the proposed approach and the DL-model proposed to annotate videos using knowledge graphs.

4.1 Problem Formulation

The goal of the proposed network is to annotate the semantic information in input videos with knowledge graphs. Formally, given a video V to the network and an ontological knowledge base, KB (consisting of domain knowledge about V), the network should compute some representation, S_V, of its semantic content. The representation should be such that it can be queried by the user to see what facts hold true of the video. The representation S_V is a knowledge graph in this work, which has been discussed in detail in Section 2.3.3. The KG contains the individuals present in the video (or the objects) and the facts that hold true of the individuals. A $fact$ expresses relation between two individuals (for example, $walk(man, dog)$), or an attribute of the individual (for example, $black(dog)$). In this work the unary facts are often denoted as $\langle subject, predicate \rangle$, where the associated predicate is also referred as attribute in this work. Similarly, the binary fact is denoted as $\langle subject, predicate, object \rangle$, and the associated predicate is referred to as relation in this work (these notations are discussed in Section 2.3). Attributes and relations together constitute the predicate component, while the subject and object are the individuals in the video.

4.2 Deep Learning Model Architecture

In this section, the deep learning model used for annotating videos using knowledge graphs is formally described.

Let X be the possible set of input videos and KB be a knowledge base containing ontological domain knowledge about X (see 2.4). Let our vocabulary consist of a set of I of all individuals and P a set

of all predicates that appear in KB. Further, P comprises the set C of all the attributes and set R of all the relations in the vocabulary. That is, $P = C \cup R$, where $\cup$ is the set union. Let the set of statistics extracted from the Visual Genome dataset be given by VG.

Then, the neural architecture for annotating videos with KG's consists of the following eight components:

1. An encoder $f : X \rightarrow \mathrm{Z}$, where Z is the space of extracted feature vectors. The function f encodes raw unprocessed video frames to fixed length feature vector Z.

2. A multi-classifier for individuals, $g : Z \rightarrow (0,1)^{|I|}$. The multi-classifier returns the probability that each individual in vocabulary being present in x.

3. A multi-classifier for attributes, $h : Z \rightarrow (0,1)^{|C|}$. The multi-classifier returns the probability if each attribute in vocabulary being present in x.

4. A multi-classifier for relations $k : Z \rightarrow (0,1)^{|R|}$. The multi-classifier returns the probability if each relation in vocabulary being present in x.

5. A set of predicate multilayer perceptrons (MLPs), $\{m_p | p \in P\}$, corresponding to all the attributes in C and relations in R. They take individual vectors and video encoding as input and output a single probability if the corresponding fact ($< subject, predicate, object >$ or $< subject, predicate >$) is present in x or not.

6. A set of trainable individual vectors: $\{v_i | i \in I\}$.

7. A set of predictions produced using the Visual Genome dataset, by utilizing Bayesian framework and statistics VG extracted from the Visual Genome dataset.

8. A Logistic Regressor, $lr : [0,1]^2 \rightarrow [0,1]$, where the input is $\{p_{m_p}, p_{VG}\}$. $p_{m_p} \in [0,1]$ and $p_{VG} \in [0,1]$ is the probability for each fact a hold true of the input video x, produced by m_p and VG respectively.

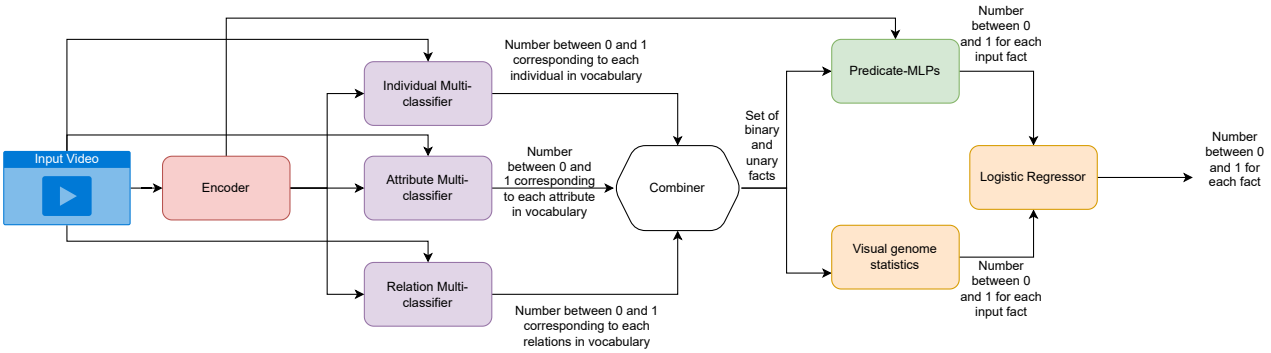

Figure 7: Description of the proposed approach for annotating a video input with knowledge graph, as explained in Section 4.3

4.2.1 Attributes and Relations Multi-classifiers

The attribute multi-classifier h and relation multi-classifier k are used for predicting the attributes and relations present in video x, respectively. They both take the video encoding as input and output the probability of the presence of attributes and relations from the vocabulary in the video, respectively. The multi-classifiers are independent of each other.

Similar to individual multi-classifier (3.3.2), the input size of both the multi-classifiers is equivalent to the video encoding, which is $dim(f(x)) = 5120$. They both have only one hidden layer. The output of the attribute multi-classifier is the multi-class probability of each attribute from the vocabulary being present in the video. Similarly, the output of the relation multi-classifier is the multi-class probability of each relation from the vocabulary being present in the video.

4.3 Proposed Approach

To achieve the proposed task of video annotation using KG, I begin with a dataset of videos, synthetically annotated with corresponding KGs, using the dataset generation steps discussed in section 6.4, an ontological knowledge-base (KB) (discussed in Section 2.4) and a set of statistics extracted from Visual Genome dataset (see Section 6.3.1). The goal is achieved in the following phases, as shown in Figure 7.

Step 1: First, the input video x is encoded using the encoder f (see Section 3.3.1) to produce the video encoding e. The encoder architecture is adapted from the work [2] because of its success reported in the paper. Section 5.3 further elaborates on the reasons for choosing this encoder for my work.

Step 2: The video encoding e is then fed to the individual multi-classifier g (see 3.3.2). The outputs from the multi-classifier is a set of probabilities corresponding to each component in the vocabulary I, given by $P_I = \{g(e)_i | i \in I\}$. The probability signifies if the corresponding individual is present in x.

Step 3: Similar to Step-2, video encoding e is fed to attribute multiclassifier h (see 4.2.1). The outputs from the multi-classifier is a set of probabilities corresponding to each component in the vocabulary C, given by $P_C = \{h(e)_c | c \in C\}$. The probability signifies if the corresponding attribute is present in x.

Step 4: Again, encoding e is fed to relation multiclassifier k (see 4.2.1). The outputs from the multi-classifier is a set of probabilities corresponding to each component in the vocabulary R, given by $P_R = \{k(e)_r | r \in R\}$. The probability signifies if the corresponding relation is present in x.

The attributes and relations together constitute the predicates for the given video. All three multi-classifiers, i.e. g, h and k are conditionally independent, given encoding e.

While the baseline work [2] did use individual multi-classifier to predict individuals in the video, it did not use attribute or relations multi-classifiers. This resulted in using predicted individuals with every possible combination of attributes and relations in the vocabulary. This is in-efficient as running predicate MLPs on all possible combinations for unary and binary facts is very expensive. To improve this, I have introduced multi-classifiers for attributes and relations here. Using them, I now only need to use a subset of attributes and relations instead of all of them for every video. This approach also has several other advantages, including superior quantitative results and decreased time complexity, which is

discussed in detail in section 7.

Step 5: Now, from the output of the individual, attribute and relation multi-classifiers (P_I, P_C and P_R respectively), a set containing a combination of individuals and predicates is chosen to build a set of candidate unary and binary facts, for the given video. Section 4.3.1 discusses our method for producing candidate unary and binary facts. This is an important step, as a poor selection of facts will degrade the model's performance (as seen in the ablation study in 8.3).

Step 6: Then, a set of MLPs (referred to as predicate-MLPs, discussed in 3.3.3), corresponding to a subset of predicates in the vocabulary are used to get the predictions for facts for each video. The candidate set of facts received from Step-5 is used for this step. Then, for each fact in the candidate set, predicate-MLP corresponding to the predicate in the fact is fed with encoding e, and individual vector(s) corresponding to the predicted individual(s) in the fact. That is, the predicate-MLP takes encoding and an individual vector v_i (for unary facts), or two individual vectors v_i and v_j (for binary facts), and produces a single probability output denoting if the fact is true of that video x. This produces the first set of predictions.

Step 7: Next, the second set of predictions is also produced for the candidate facts produced in Step-5. This is because, while the predicate-MLPs can predict the truth value of a fact for a given input, but it lacks commonsense knowledge, such as a fact $drive(man, car)$ is possible in the real world, but a fact $swing(man, car)$ is not possible. So this step accounts for such cases. Further, predicate-MLP might not produce correct facts on its own when it has less particular information about the video domain or the dataset. This is particularly common in our case, as the KG-annotated video dataset is not available in the literature (as per my knowledge). As we will see in 8.2, this step also increases the generalization capability of the model.

To execute this step, statistics from a large image dataset, Visual Genome [30] (see in 6.3) is built and used. Visual Genome, being much larger than other data sets used here, could

encode more general world information in its statistics and is independent of the above model (explained in section 4.3.3). The second set of predictions will be used to modify the predictions given by predicate-MLPs, in Step-6. As we will see in section 8.2, it benefits the most in untrained/not fully trained networks, signifying not having particular information for the input dataset. Information from Visual Genome is leveraged in two steps:

(a) A set of statistics which describes the individuals, attributes, relations and facts in the images are extracted from the whole Visual Genome dataset using the pipeline discussed in section 6.3.1.

(b) Predictions are given by calculating the expected value of the fact a under the Bayesian posterior of fact being true given the statistics. This step is repeated for each candidate fact from Step-5 to produce a second set of predictions. This discussed in Section 4.3.3.

Step 8: Now, for each fact in the candidate set (Step-5), there are two predictions, produced in Step-7 by using Visual Genome statistics and Step-6 given by predicate-MLPs.

These two probability values are input to a logistic regressor, and the target is to classify the corresponding fact as true or false for the given input x (i.e. perform a binary classification task).

Step 9: Finally, all the true facts received from Step-8 are combined to produce the knowledge graph for the input video x. This step is discussed in Section 4.3.4.

The resultant network takes raw video as input x and produces a set of facts (and negated facts) representing that video's semantic content. Concisely, the proposed approach is shown in Figure 7. Below we describe various steps taken in the proposed approach in detail.

4.3.1 Combination of predictions from three multi-classifiers to get candidate facts

Here we elaborate Step-5 in the proposed approach (see Section 4.3). After receiving the predictions P_I, P_C and P_R corresponding to the set of individuals I, attributes C and relations R from respective

classifiers, for the video x, we need to combine respective predictions to get a set of candidate facts. This task can be approached in broadly two following ways:

Naive Approach: The probabilities from each of the multi-classifiers are thresholded at $threshold = 0.5$ or using an adaptive threshold (adaptive threshold could be calculated using criteria such as - choosing threshold which increases overall accuracy in training dataset). Let the sets P_I, P_C and P_R be thresholded, and we receive the sets I', C' and R' of individuals, attributes and relations respectively, for the input video x. That is, $I' = \{i \in I | g(e)_i > threshold\}$, $C' = \{c \in C | h(e)_c > threshold\}$, $R' = \{r \in R | k(e)_r > threshold\}$, where $g(e)_i$ is the probability output for individual i in vocabulary produced by individual multi-classifier when fed with encoding e. $h(e)_c$ and $k(e)_r$ is similary defined for respective multi-classifiers.

Then let the set of all candidate unary facts be given by $U = I' \times C'$ for x. Similarly, let the set of all candidate binary facts is given by $B = I' \times I' \times R'$. Later on, all the facts from U and B are tested using predicate-MLPs and Visual Genome to get the prediction if the facts hold true of the given input video encoding e, as explained in Steps-6 and 7.

Hence, the set of all predicate-MLPs that I will need to run will be $\{m_p | p \in P, P = C' \cup R'\}$. For each fact a, a pair $a = \langle subject, predicate \rangle = \langle i, c \rangle$ in set $U = \{a_1, a_2, \ldots, a_c\}$, individual vector v_i corresponding to the i, along with video encoding e, is fed to the predicate-MLP m_{p_c} corresponding to the attribute c in the fact. Similarly, for each binary fact in set B, individual vectors corresponding to subject (v_s) and object (v_o), along with video encoding is fed to the predicate-MLP m_{p_r}, corresponding to relation r in the binary fact.

This would require $|P| \times |n|$ passes, where $|n|$ is the number of individuals predicted by the individual-multi-classifier for the input video, and $|P|$ is the combined number of predicates in $C' \cup R'$. However, this naive approach has the following shortcomings:

1. The threshold is constant among all the inputs for the same component. However, depending on video quality, the size of I', C', and R' could vary greatly and could potentially lead to an empty set ($\emptyset$). This might not be true in many cases.

2. Also, using this approach would mean treating individuals, attributes and relations separately. However, our goal is to infer facts, and so we need to consider them together to produce the set of candidate facts.

Alternate Approach: Alternatively, a better approach would be to account for the fact that individuals should be considered in conjunction with associated attributes and relations. for each given input video.

All the multi-classifiers have been trained independently, using the encoding. Hence, the multi-classifiers g, h and k are independent given the encoding e. In literature there exist different possible combination methods for classifier output such as minimum rule, maximum rule, weighted average, mean, product rule etc. [127], which works well in practice. However, it would be beneficial to devise a combination strategy on a fundamental basis, which is taken here.

Formally, the probability of a fact a occurring is actually the joint probability of (*subject* and *attribute*) or (*subject*, *object* and *relation*) occurring together, given the input video encoding. Let the probability of a fact a occurring be $P(a \in A|e)$, where e is the video encoding of the corresponding input video x. Then,

$$P(a \in A|e) = \begin{cases} P((s \in I \cap c \in C)|e), \text{ if } Size(a) = 2 \ (unaryfact) \\ P((s \in I \cap o \in I \cap r \in R)|e), \text{ if } Size(a) = 3 \ (binaryfact), \end{cases} \tag{29}$$

where $Size(a)$ denotes number of components/atoms [1] in fact a.

[1] In this thesis, each constitutent elements of the facts have often been called as atom

Since, the classifiers are conditionally independent, i.e. independent given the input video encoding, we get,

$$\begin{aligned} P((s \in I \cap c \in C)|e) &= P(s \in I|e) \times P(c \in C|e)\,, \\ P((s \in I \cap o \in I \cap r \in R)|e) &= P(s \in I \cap o \in I|e) \times P(r \in R|e)\,. \end{aligned} \tag{30}$$

Using equation 30, the probability of a fact a occurring given the video encoding e.

$$P(a \in A|e) = \begin{cases} P(s \in I|e) \times P(c \in C|e), \text{ if } Size(a) = 2\ (unaryfact)\,. \\ P(s \in I \cap o \in I|e) \times P(r \in R|e), \text{ if } Size(a) = 3\ (binaryfact)\,. \end{cases} \tag{31}$$

Equation 31 for binary facts can be further simplified. Let the hidden layer in the individual multiclassifier is denoted by l (architecture of individual multi-classifier is given in Section 3.3.2). Then we can assume that, $P(s \in I)$ and $P(o \in I)$ are conditionally independent given the e and l. That is,

$$P(s \in I \cap o \in I|e) = P(s \in I|l, e) \times P(o \in I|l, e), \;\; s \perp\!\!\!\perp o|l, e\,. \tag{32}$$

Using Equation 32, we get:

$$P(a \in A|e) = \begin{cases} P(s \in I|e) \times P(c \in C|e), \text{ if } Size(a) = 2\ (unaryfact)\,. \\ P(s \in I|e, l) \times P(o \in I|e, l) \times P(r \in R|e), \text{ if } Size(a) = 3\ (binaryfact)\,. \end{cases} \tag{33}$$

Equation 33 simply means that the joint probability of fact occurring is given by the multiplication

of probabilities of each component in the fact. This is also known as the product combination rule, which has often shown superior results in various cases [127].

Now, to do this, the multi-classifier's outputs are multiplied using the following approach. The probabilities received from the multi-classifier (for each component) in the fact are multiplied and saved in the matrix.

For unary facys, a 2d-matrix $U'^{|I|\times|C|}$ is formed.

$$U' = \begin{bmatrix} p_{11} & p_{12} & p_{13} & \cdots & p_{1n} \\ p_{21} & p_{22} & p_{23} & \cdots & p_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{d1} & p_{d2} & p_{d3} & \cdots & p_{dn} \end{bmatrix}_{|I|\times|C|}$$

, where p_{ij} denotes the multiplication of probability of individual i in the vocabulary, and attribute j in the vocabulary, being present in x, received from respective multi-classifiers, i.e., $p_{ij} = \{g(e)_i \times h(e)_j \mid i \in I\, j \in C\}$.

Similarly for binary relations, a 3d-tensor $B'^{|I|\times|I|\times|R|}$ is formed.

$$B' = \begin{bmatrix} \begin{bmatrix} p_{111} & p_{112} & p_{113} & \cdots & p_{11m} \\ p_{121} & p_{122} & p_{123} & \cdots & p_{12m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{1d1} & p_{1d2} & p_{1d3} & \cdots & p_{1dm} \end{bmatrix} & \cdots & \begin{bmatrix} p_{d11} & p_{d12} & p_{d13} & \cdots & p_{d1m} \\ p_{d21} & p_{d22} & p_{d23} & \cdots & p_{d2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{dd1} & p_{dd2} & p_{dd3} & \cdots & p_{ddm} \end{bmatrix} \end{bmatrix}_{|I|\times|I|\times|R|}$$

,where p_{ijl} denotes the multiplication of probability of subject i, object j and relation l from the vocabulary I, I and R respectively, being present in x, received from respective multi-classifiers, $p_{ijk} = \{g(e)_i \times g(e)_j \times k(e)_l \mid i, j \in I\, i \neq j, l \in R\}$. Hence, all the values, when $i = j$, is set to 0. That is,

$$B' = \left[\left[\begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ p_{121} & p_{122} & p_{123} & \cdots & p_{12m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{1d1} & p_{1d2} & p_{1d3} & \cdots & p_{1dm} \end{bmatrix} \dots \begin{bmatrix} p_{d11} & p_{d12} & p_{d13} & \cdots & p_{d1m} \\ p_{d21} & p_{d22} & p_{d23} & \cdots & p_{d2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}\right]\right]_{|I|\times|I|\times|R|}$$

From the U' matrix, top q values are selected and saved in set U, as the combined prediction from multi-classifiers for candidate unary facts. That is, $U = \{a_1, a_2, \dots a_q\}$, where each fact a_l is a unary fact, $a_l = \langle s, c\rangle$[2], such that the probability of it occurring $p_{sc} \in U'$ is in top q values, for some $s \in I$ and $c \in C$. Here s is the individual in vocabulary and c is the attribute in vocabulary.

Similarly, from the B' matrix, top q values are selected and saved in set B, as the combined prediction from multi-classifiers for candidate binary facts. That is, $B = \{a_1, a_2, \dots a_q\}$, where each fact a_l is a binary fact, $a_l = \langle s, r, o\rangle$[3], such that the probability of it occurring $p_{sor} \in B'$ is in top q values, for some $s, o \in I$, $s \neq o$ and $r \in R$. The value of q is chosen according to the method described in Section 5.2.2.

This approach offers several advantages. As we know, mathematical operations like multiplication in matrix form are quick and computationally inexpensive compared to the earlier approaches. Further, this ensures that the components in a fact are chosen by considering them together. This drastically affects the overall accuracy, which is evident from our results in Section 8.3. Lastly, this method can now naturally extend to > 3 components in a fact, without the need to fine-tune the threshold for all of them separately. We can use the same value of q for datasets with facts having more than three components.

Note: In my case, the size of vocabulary is $|I| = 285$, $|C| = 150$, $|R| = 129$.

[2]Following the convention used in this thesis an unary fact is denoted as $\langle subject, predicate\rangle$

[3]Following the convention used in this thesis a binary fact is denoted as $\langle subject, predicate, object\rangle$

4.3.2 Predictions from Predicate-MLPs

The Step-6 in the proposed approach (Section 4.3) is discussed in detail here. The set of candidate unary U and binary facts B received by combining the results from the three multi-classifiers (see Section 4.3.1) is used here. Since the candidate facts only contain a subset of the predicates from the vocabulary, I now no more need to run all the predicate-MLPs from the vocabulary with each predicted individual (this was one of the shortcomings of the baseline model [2], as it required the model to run all predicate-MLPs in the vocabulary with each predicted individual).

For each fact a_l in set $U = \{a_1, a_2, \ldots, a_q\}$, where $a_l = \langle subject, predicate \rangle = \langle s, c \rangle$, for some $s \in I$ and $c \in C$. The individual vector v_s corresponding to individual s, along with input video encoding e, is fed to the predicate-MLP m_{p_c} corresponding to the attribute c. We save the output of predicate-MLP corresponding to each candidate unary fact from U, for a given input video. The set of predictions from predicate-MLPs is given by $U_{p-mlp} = \{m_{p_c}(v_s, e) \mid \langle s, c \rangle \in U\}$, where $m_{p_c}(v_s, e)$ is the probability output given by the predicate-MLP for fact $\langle s, c \rangle$. This value is calculated by feeding predicate-MLP m_{p_c} corresponding to predicate c, with individual vector v_s corresponding to individual s and video encoding e.

Similarly, each candidate fact a_l in set $B = \{a_1, a_2, \ldots, a_q\}$, where $a_l = \langle subject, predicate, object \rangle = \langle s, r, o \rangle$, individual vectors v_s, v_o corresponding to the s and o, along with video encoding e, is fed into the predicate-MLP $m_{p_r} \in P$ corresponding to the attribute r. The set of predictions from predicate-MLPs for all the candidate binary facts is given by $B_{p-mlp} = \{m_{p_r}(v_s, v_o, e) \mid \langle s, r, o \rangle \in B\}$, where $m_{p_r}(v_s, v_o, e)$ denotes the prediction out from predicate-MLP, when it is operated with input (v_s, v_o, e).

Finally, the set of predictions produced by the predicate-MLPs for the candidate facts for video x is given by T_{p-mlp}:

$$T_{p-mlp} = U_{p-mlp} \cup B_{p-mlp}\,. \tag{34}$$

4.3.3 Predictions using Visual Genome dataset

Next, I explain the proposed approach for producing the second set of predictions, as discussed in Step-7 in Section 4.3. For each candidate unary fact in U and candidate binary fact in B for an input video x, (see 4.3.1), a second prediction is produced using the statistics VG, which is extracted from the Visual Genome dataset. Details about the dataset and extraction of statistics VG, are given in Section 6.

The prediction from VG will not depend on the video itself; rather, it represents commonsense knowledge of which predicates are likely to hold true of which individuals in general. That is, for each potentially true fact, we want to estimate the probability that it holds true of a randomly chosen video. This is essentially useful to avoid the prediction of facts which are unlikely to occur in the real world. For example, $drive(man, car)$ is possible, however $swing(man, car)$ is never possible. In this case, for the (man, car) pair, the VG will produce very high confidence for the former fact while very low confidence for the latter fact. This would mean that there is a possibility of fact $drive(man, car)$ occurring in the real world, but the same cannot be said for $swing(man, car)$.

This is an important problem to solve in semantic understanding work because, along with producing correct facts, we also need to avoid impossible scenarios. As we will see in Section 8.2, predictions using VG also help the model most when it has less information about the data.

Now the task of calculation of prediction using the Visual Genome dataset can be approached in broadly two following ways:

Naive Approach A naive approach to calculate the predictions from the statistics from the dataset would be to calculate the probability for each fact a using:

$$\begin{aligned} Pr(a|VG) = Pr(\langle s, c\rangle|VG) &= \frac{\#(c,s) - in - VG}{\#s - per - image - in - VG}\,. \\ Pr(a|VG) = Pr(\langle s, r, o\rangle|VG) &= \frac{\#(s,r,o) - in - VG}{\#(s,o) - per - image - in - VG}\,. \end{aligned} \tag{35}$$

That is, $Pr(a|VG) = Pr(\langle s, c\rangle|VG)$ gives the value of the count of the number of occurrences of unary fact a in the VG divided by the count of the number of occurrences of individuals s in VG. This would signify the probability of individual s having that attribute c. The higher the probability, the greater the confidence that subject s and attribute c, can appear together. For example, if the probability of fact $brown(dog)$ has a high probability $Pr(\langle dog, brown\rangle|VG) = 0.9$, then there is a higher chance that any appearance of a dog in the output of predicate-MLP, will have an attribute *brown* associated to it. Similarly, $Pr(a|VG) = Pr(\langle s, r, o\rangle|VG)$ signifies the probability of having relation r, if (s, o) appears in VG.

However, this simple method suffers from several issues. As we can see in Table 1 (Section 6.3.1), the VG and our dataset do not have high overlap in vocabulary (particularly, individuals have higher overlap, followed by attributes and relations). This means there is a chance many of the candidate facts are not present in the Visual Genome dataset, even when the *individual* (or $(subject, object)$) is present in the dataset. We will receive $P_{VG} = 0$, for many of them. This is because $\#(c, s)$ (or $\#(s, r, o)$) is equal to zero when the fact is not present in the dataset. This will decrease the usefulness of this method if predictions from VG are not being used in most of the cases.

Further, it was also observed that many images in Visual Genome contain repetition of the same fact more than once, though the individual (or (subject, object)) has only appeared once in the dataset. This would produce a probability $P_{VG} > 1$, which is not true.

Lastly, if the individual and the fact only appear once in the VG, then $Pr = 1$, assigning a perfect probability of its occurrence might not be true, as the fact only appeared once in the whole dataset.

Better Approach: Alternatively, it would be wiser to calculate the expected value of the fact a under the Bayesian posterior $P(a \in A|D = d)$, given the statistics VG from the Visual Genome dataset. That is,

$$\mathbb{E}(a) = \int_0^1 a \; P(a \in A|D = d) \; da \, . \tag{36}$$

,where $P(a \in A|D = d)$ is the posterior probability of fact a given it's number of occurrence $d \in D$, in VG. A is the set of all the facts in the Visual Genome dataset. The prediction produced by the Visual Genome dataset for fact a is given by $\mathbb{E}(a)$.

Using Bayes' theorem [128], the posterior probability of fact a is given by,

$$P(a \in A|D = d) = \frac{P(D = d|a \in A) \times P(a \in A)}{P(D = d)} \, , \tag{37}$$

where $P(a \in A)$ is the Bayesian prior, $P(D = d|a \in A)$ is the likelihood and $P(D = d)$ is the evidence.

For simplicity, let us assume a uniform prior, i.e. $P(a \in A) = 1$, such that $\int_0^1 \; P(a \in A) \; da = 1$.

Likelihood: Now, the likelihood is given by:

$$P(D = d|a \in A) = a^d \; (1 - a)^{N-d} \, , \tag{38}$$

where N is given by Equation 40.

Calculation of evidence:

$$P(D = d) = \int_0^1 x^d (1 - x)^{N-d} \, dx \,, \tag{39}$$

where N is the total number of images in the Visual Genome dataset containing the *individual* (or $(subject, object)$ for a binary fact) for fact a. That is,

$$N = \begin{cases} \#(\_, i) \text{if} \,, len(a) = 2 \\ \\ \#(\_, s, o) \text{if} \,, len(a) = 3 \,, \end{cases} \tag{40}$$

where i is the individual in unary fact a, and (s, o) is the subject-object pair in fact binary fact a.

Coming back to equation 39:

$$P(D=d) = \int_0^1 x^d (1-x)^{N-d}\, dx$$

Using integration by parts, given by $\int_a^b u\frac{dv}{dx}dx = uv - \int v\frac{du}{dx}dx$,

Substituting $u = (1-x)^{N-d}$ and $\frac{dv}{dx} = x^d \Rightarrow v = \frac{x^{d+1}}{d+1} + constant$,

$$= (\frac{x^{d+1}(1-x)^{N-d}}{d+1} + constant)\Big|_0^1 + \frac{N-d}{d+1}\int_0^1 x^{d+1}(1-x)^{N-d-1}dx \text{ , for } Re(d-N) < 1 \wedge Re(d) > -2$$

As $d \in \mathbb{Z}_{\geq 0}$, $d \leq N$, $Re(d-N) < 1 \wedge Re(d) > -2$ is always satisfied,

$$= 0 + \frac{N-d}{d+1}\int_0^1 x^{d+1}(1-x)^{N-d-1}dx$$

$$= \frac{N-d}{d+1}\cdot\frac{N-d-1}{d+2}\cdot\int_0^1 x^{d+2}(1-x)^{N-d-2}dx$$

Repeating integrating by parts several times, we get,

$$= \frac{(N-d)\cdot(N-d-1)\cdot(N-d-2)\ldots 2\cdot 1}{(d+1)\cdot(d+2)\ldots N}\int_0^1 x^N(1-x)^0 dx$$

$$= \frac{d!(N-d)!}{N!}\int_0^1 x^N dx$$

$$= \frac{d!(N-d)!}{N!}\cdot\frac{1}{N+1},$$

$$= \frac{d!(N-d)!}{(N+1)!}$$

Using Fact 1, we get,

$$= \frac{\Gamma(d+1)\times\Gamma(-d+N+1)}{\Gamma(N+2)}, \tag{41}$$

where Γ is the gamma function, $Re(z)$ is the real part of z, $e_1 \wedge e_2 \wedge \ldots$ is the logical AND function, and $\mathbb{Z}_{\geq 0} = \{0, 1, 2\ldots\}$ denotes the set of whole numbers.

Calculation of Posterior: Finally, using equation 37-38, the posterior probability is given by:

$$
\begin{aligned}
P(a \in A|D = d) &= \frac{P(D = d|a \in A) \times P(a \in A)}{P(D = d)}\,, \\
&= \frac{a^d\ (1-a)^{N-d} \times 1}{\frac{\Gamma(d+1)\times\Gamma(-d+N+1)}{\Gamma(N+2)}}\,, \\
&= \frac{a^d\ (1-a)^{N-d}}{\frac{\Gamma(d+1)\times\Gamma(-d+N+1)}{\Gamma(N+2)}}\,, \\
&= \frac{a^d\ (1-a)^{N-d}}{\frac{d!\times(-d+N)!}{(N+1)!}}\,.
\end{aligned}
\tag{42}
$$

Finally, using the equations 36-42, the expected value of fact a is calculated as,

$$
\begin{aligned}
\mathbb{E}(a) &= \int_0^1 a\ P(a \in A|D = d)\ da \\
&= \int_0^1 \frac{a \times (a^d\ (1-a)^{N-d})}{\frac{\Gamma(d+1)\times\Gamma(-d+N+1)}{\Gamma(N+2)}}\ da \\
&= \frac{1}{\frac{\Gamma(d+1)\times\Gamma(-d+N+1)}{\Gamma(N+2)}} \times \int_0^1 a \times (a^d\ (1-a)^{N-d})\ da \\
&= \frac{\Gamma(N+2)}{\Gamma(d+1)\times\Gamma(-d+N+1)} \times \int_0^1 a^{d+1}\ (1-a)^{N-d})\ da \\
&\text{Substituting solution of integral } \int_0^1 a^{d+1}\ (1-a)^{N-d}) \text{ using Equation 44, we get,} \\
&= \frac{\Gamma(N+2)}{\Gamma(d+1)\times\Gamma(-d+N+1)} \times \frac{\Gamma(d+2)\times\Gamma(-d+N+1)}{\Gamma(N+3)}
\end{aligned}
$$

Using Fact 1 ,

$$
\begin{aligned}
&= \frac{\Gamma(d+2)}{\Gamma(d+1)} \times \frac{\Gamma(N+2)}{\Gamma(N+3)} \\
&= \frac{(d+1)!}{d!} \times \frac{(N+1)!}{N+2!} \\
&= \frac{d+1}{N+2}\,.
\end{aligned}
\tag{43}
$$

Fact 1. Γ function is a generalized factorial function, i.e. $\Gamma(n+1) = n!$, for all non-negative whole numbers n. Since d and N are the counts they are always non-negative whole numbers, this condition applies in our case.

$$
\begin{aligned}
&Int = \int_0^1 a^{d+1}(1-a)^{N-d}\,dx\,,\\
&\quad\text{Using integration by parts, given by } \int_a^b u\frac{dv}{dx}dx = uv - \int v\frac{du}{dx}dx\,,\\
&\quad\text{Substituting } u = a^{d+1} \text{ and } \frac{dv}{da} = (1-a)^{N-d} \Rightarrow v = \frac{(1-a)^{N-d+1}}{d-N-1} + constant\,,\\
&\quad= (\frac{a^{d+1}(1-a)^{N-d+1}}{d-N-1} + constant)\Big|_0^1 + \frac{d+1}{N-d+1}\int_0^1 a^d(1-a)^{N-d+1}da\,, \text{for } Re(d-N) < 1 \wedge Re(d) > -2\,,\\
&\quad\text{As } d \in \mathbb{Z}_{\geq 0},\ d \leq N,\ Re(d-N) < 1 \wedge Re(d) > -2 \text{ is always satisfied}\,,\\
&\quad= 0 + \frac{d+1}{N-d+1}\int_0^1 a^d(1-a)^{N-d+1}da\,,\\
&\quad= \frac{d+1}{N-d+1}\cdot\frac{d}{N-d+2}\cdot\int_0^1 a^{d-1}(1-a)^{N-d+2}da\,,\\
&\quad\text{Repeating integrating by parts several times, we get}\,,\\
&\quad= \frac{(d+1)\cdot(d)\cdot(d-1)\ldots 2\cdot 1}{(N-d+1)\cdot(N-d+2)\ldots(N+1)}\int_0^1 a^0(1-a)^{N+1}da\,,\\
&\quad= \frac{(d+1)!(N-d)!}{(N+1)!}\int_0^1 a^{N+1}dx\,,\\
&\quad= \frac{(d+1)!(N-d)!}{(N+1)!}\cdot\frac{1}{N+2}\,,\\
&\quad= \frac{(d+1)!(N-d)!}{(N+2)!}\,,\\
&\quad\text{Using Fact 1, we get}\,,\\
&\quad= \frac{\Gamma(d+2)\times\Gamma(-d+N+1)}{\Gamma(N+3)}\,.
\end{aligned}
\tag{44}
$$

Finally, for each fact, a, the predictions produced using VG are put into matrix U_{VG} and B_{VG}, corresponding to the set of candidate facts in U, and B respectively. The final set of predictions from VG is given by T_{VG}:

$$
\begin{aligned}
U_{VG} &= \{\frac{d_{(s,c)}+1}{N_s+1} \mid d_{(s,c)}, N_s \in VG\}\,, \\
B_{VG} &= \{\frac{d_{(s,r,o)}+1}{N_{(s,o)}+1} \mid d_{(s,r,o)}, N_{(s,o)} \in VG\}\,, \\
T_{VG} &= U_{VG} \cup B_{VG}\,.
\end{aligned}
\tag{45}
$$

4.3.4 Final Prediction from the model

After receiving predictions from predicate-MLP (explained in 4.3.2), and Visual Genome (Section 4.3.3), both the outputs are combined to produce the final semantic representation S_V, in form of knowledge graph for the given input, x.

To combine predictions from predicate-MLPs and Visual Genome, several approaches can be taken. One of such method would be to combine both the outputs by taking an average, i.e., $P(a) = \frac{T_{p-mlp}(a)+T_{VG}(a)}{2}$, where $T_{p-mlp}(a)$ and $T_{VG}(a)$ is the predicate-MLP output and Visual Genome output for fact a, received from T_{p-mlp} and T_{VG}, respectively.

However, this gives equal weight to both outputs. While in my case $T_{p-mlp}(a)$ might be a more relevant prediction than $T_{VG}(a)$ because the vocabulary of the datasets does not 100% overlap. Further, we only aim to use VG output for injecting commonsense in the predicate-MLP output.

On the other hand, a weighted sum makes more sense, depending on how much importance should be given to each of the predictions. We can train a simple logistic regressor lr, to input both probabilities $\{T_{p-mlp}(a), T_{VG}(a)\}$ for a given fact a, and output a classification result $P(a|T_{pred-MLP}, T_{VG})$. This way weights are automatically assigned to each probability, depending on how much data of VG is relevant in cases of different datasets.

Finally, there could be the case that *subject* (or $(subject, object)$) for a fact a is not present in the Visual Genome vocabulary. In this case, Visual Genome output is irrelevant to us, and instead of

assigning a probability of zero (which is untrue as the VG dataset is not exhaustive of all the world knowledge), we ignore it and the final output from predicate-MLPs is used to decide if the fact holds true of the input video.

The two probabilities (received from predicate-MLP and VG) are then fed to a logistic regressor, which produces a single probability for a fact a. This final probability is then compared with a threshold to determine if it's a true or false fact. The set of true predicted facts is represented as $\hat{T}$, while the set of false predicted facts is represented as $\hat{F}$. The choice of threshold is discussed in section 5.2.1.

In other words, the above steps can be represented as the following algorithm,1,

Algorithm 1 Final Prediction for x

for **do** a, $\forall a \in T_{p-mlp}$
  if $(|a| = 2 \wedge s \in T_{VG})$ or $(|a| = 3 \wedge (s, o) \in T_{VG})$ **then**
    $P(a|T_{p-mlp}, T_{VG}) \leftarrow lr([T_{p-mlp}(a), T_{VG}(a)])$
    if $P(a|T_{p-mlp}, T_{VG}) > threshold$ **then**
      $\hat{T} \leftarrow \hat{T} \cup \{a\}$
    else
      $\hat{F} \leftarrow \hat{F} \cup \{a\}$
  else
    $P(a|T_{p-mlp}) \leftarrow T_{p-mlp}(a)$
    if $P(a|T_{p-mlp}) > threshold$ **then**
      $\hat{T} \leftarrow \hat{T} \cup \{a\}$
    else
      $\hat{F} \leftarrow \hat{F} \cup \{a\}$

All the predicted true facts for video x, are saved in $\hat{T}$, and all the negative facts are saved in $\hat{F}$. The final set of predictions from the model for input x is given by $\hat{T} \cup \hat{F}$. The union of facts in $\hat{T}$ forms the knowledge graph.

4.4 Training Strategy

As described in section 2.1.3, the proposed deep learning model is trained to learn from the given data, during which the model's parameters are updated in the direction of decreasing loss value. During training, I have access to the ground truth data for the train and validation set (dataset split

is given in Section 6).

I have employed two training settings. During the training setting-1, I train the encoder f, individual multiclassifier g, attribute multiclassifier h, relation multiclassifier k and predicate-MLPs m_p corresponding to all predicates in the vocabulary[4]. The pipeline for training setting-1 is shown in Figure 8.

The training on setting-1 is completed to convergence. After which, the model is reloaded for setting-2 training where logistic regressor lr training is carried out (referred to as train setting-2). For setting-2, a different dataset is built from components fully trained during setting-1. Figure 9 shows the pipeline for train setting-2.

The training steps, along with loss calculations and other implementational details are discussed in detail in section 5.1.

Example 1. Consider the video in Figure 10, and the pipeline of the model given in Figure 7. Following the steps in Figure 7, the KG for the video in Figure 10 is produced in the following steps:

1. The input video x is mapped into a fixed-length feature vector e using the encoder f.
2. The encoding e is fed into individual multi-classifier g, which returns the probability that each individual in the vocabulary being present in x, given by P_I.
3. The encoding e is also fed into attribute multi-classifier h, which returns the probability of each attribute in the vocabulary being present in x, given by P_C.
4. Similarly, the encoding e is also fed into relation multi-classifier k, which returns the probability of each relation in the vocabulary being present in x, given by P_R.
5. After receiving the sets P_I, P_C and P_R, the set of unary facts U' and binary fact B' if formed,

[4]there are 129 attributes and 150 relations in the vocabulary, hence we train 279 predicate-MLPs corresponding to each of them

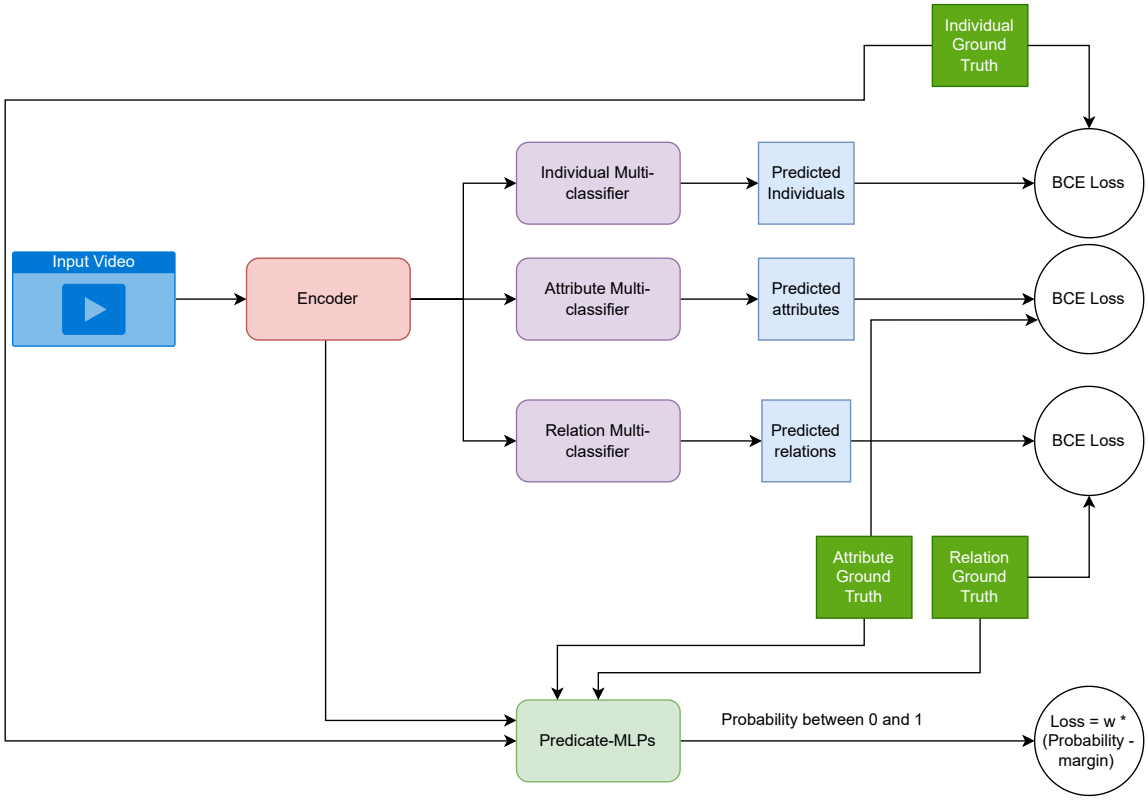

Figure 8: Pipeline for train setting-1, discussed in Section 5.1.1

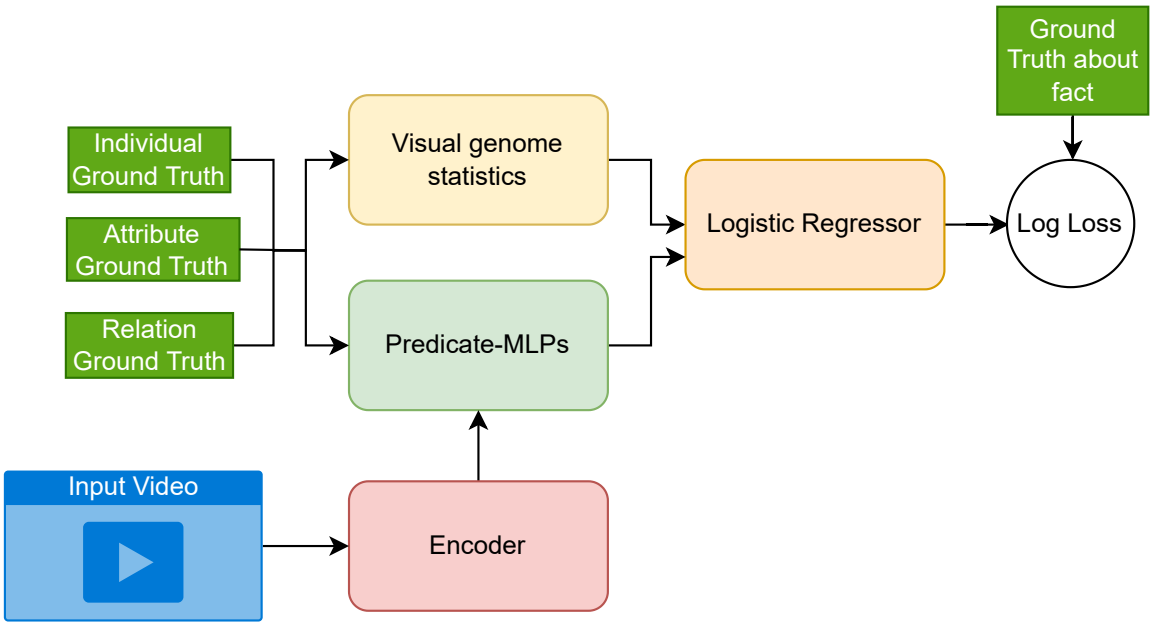

Figure 9: Pipeline for train setting-2, discussed in 5.1.2

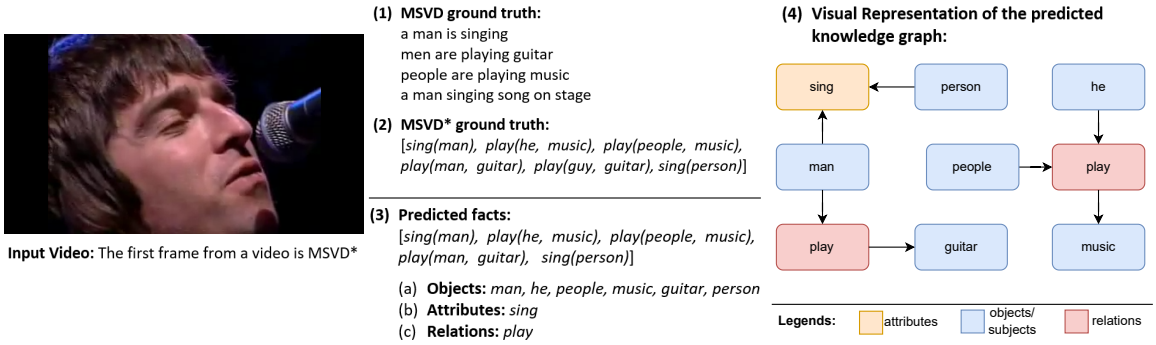

Figure 10: The first frame from MSVD*, with (1) ground-truth natural language captions in MSVD, (2) the ground-truth set of facts in MSVD*, (3)the facts predicted by our model, with (a)objects/subjects present, (b)attributes predicted, (c)relations predicted, and (4) visual representation of the knowledge graph produced

using the method discussed in Section 4.3.1. For each unary and binary fact, top q results are picked. The value of $q = 1000$, which is decided in Section 8.1.

6. For each candidate fact a in U and B, predictions using predicate-MLPs are produced, and we get the set $T_{pred-mlp}$, as given in Section 4.3.2.

7. For each candidate fact a in U and B, predictions using Visual Genome are calculated, and we get the set T_{VG}, as discussed in Section 4.3.3.

8. Final predictions for each video are produced using the algorithm 1. In this example, we will get, $\hat{T} = \{sing(man),\ play(he, music),\ play(people, music),\ play(man, guitar),\ sing(person)\}$. KG is the combination of all facts in $\hat{T}$ as shown in Figure 10.

5 Model Training and Implementational Details

5.1 Training Details

Here I elaborate on the training approach, briefly discussed in section 4.4.

I use two different settings for training the complete deep learning model from section 4.2. In the first setting, the encoder, individual multi-classifiers, attribute multi-classifiers, relation multi-classifiers, individual vectors corresponding to each individual in vocabulary, and predicate-MLPs corresponding to each predicate in vocabulary, are trained.

In the second setting, the logistic regressor lr is trained using the predictions from the fully-trained network received from training setting-1. The pipeline to build the dataset for training lr is explained in section 5.1.2.

As explained in Section 2.5, using the LCWA assumption the dataset contains many negated facts compared to positive facts. To avoid the model predicting everything false, the number of negated facts used for training is limited to the number of positive facts. This is done by randomly sampling an equal number of negated facts for each video. This is done for both training settings.

5.1.1 Training setting-1

The pipeline for training setting-1 is shown in Figure 8.

The data for training for the neural architecture consists of 6 tuples (x, i, c, r, T, F), where $x \in X$ is the input video to be annotated (a 4D video tensor), $i \subseteq I$ is the set of individuals present in the input, $c \subseteq C$ is the set of attributes present in the input, $r \subseteq R$ is the set of relations present in the input, and T is a set of ground-truth facts containing a combination of these individuals, attributes and relations, while F is a set of ground truth negated facts (generated using LCWA assumption,

2.5) containing a combination of these individuals, attributes and relations.

For each input video in the training set, the following steps are followed, as shown in figure 8:

1. First the feature vector for the x is computed, $f(x) = e, e \in Z$, is computed.

2. The feature vector e is then fed into the individual multiclassifier to predict the individuals present in x, $g(e) = \widehat{i}$. Then the binary cross-entropy loss (BCE) for each individual in the vocabulary is computed, which given by equation 46:

$$L_i = \Sigma_{j=0}{}^{|I|} L_{bce}(\widehat{i_j}, i_j)\,, \tag{46}$$

   where $i_j \in \{0, 1\}$ and $\widehat{i_j} \in [0, 1]$ are the ground truth and the prediction if the j^{th} individual in vocabulary is present in x, respectively. That is, $i_j = 1$ would mean that the j^{th} individual is present in x, while $i_j = 0$ would mean that it is not present, given by the ground truth. $\widehat{i_j}$ gives the probability of j^{th} individual in the vocabulary being present in x produced by the individual multiclassifier. L_{bce} denotes the BCE, as discussed in Section 2.1.3.

3. As above, the feature vector e is fed into the attribute multiclassifier to predict attributes present in x, $\widehat{c} = h(e)$. Similar to individual multiclassifier, the BCE loss for each attribute in vocabulary is given by:

$$L_c = \Sigma_{j=0}{}^{|C|} L_{bce}(\widehat{c_j}, c_j)\,, \tag{47}$$

   where $c_j \in \{0, 1\}$ and $\widehat{c_j} \in [0, 1]$ are the ground truth and the prediction if the j^{th} attribute in vocabulary is present in x, respectively. L_{bce} denotes the BCE (see 2.1.3).

4. The feature vector e is also fed into the relation multiclassifier to predict relations present in x, $\widehat{r} = k(e)$.The BCE loss for each relation in vocabulary is given by:

$$L_r = \Sigma_{j=0}{}^{|R|} L_{bce}(\widehat{r_j}, r_j)\,, \tag{48}$$

where $r_j \in \{0,1\}$ and $\widehat{r_j} \in [0,1]$ are the ground truth and the prediction if the j^{th} binary relation in vocabulary is present in x, respectively. L_{bce} denotes the BCE (see 2.1.3).

5. Next, the predicate-MLPs are trained using ground-truth facts T and negated facts F. For each fact in the ground truth, The predicate-MLPs corresponding to predicates in the fact is fed with individual vector(s) selected from the individual(s) in the fact along with the video encoding. In other words, for each binary fact $t \in T$, where $t = \langle i, p, j \rangle$, for some $i, j \in I, i \neq j$ and $p \in P$, a prediction $\widehat{t} = m_p(e, v_i, v_j)$ is computed. Similarly, for each negated binary fact $f \in F$, where $f = \langle i, p, j \rangle$, for some $i, j \in I, i \neq j$ and $p \in P$, a prediction $\widehat{f} = m_p(e, v_i, v_j)$ is computed. v_i and v_j are the respective individual vectors. The same procedure is repeated for each unary fact in T and negated fact in F.

   Now, to train the predicate MLPs two ways are possible:

   (a) An naive brute-force method is to run every predicate-MLP on every individual vector present in the vocabulary. However, this would take $|P| \times |I|$ forward passes for every video.

   (b) Alternatively, I can only select individuals present in the video and run predicate-MLPs only on them. This would reduce the number of forward passes to $|P| \times |n|$, where $|n|$ is the number of individuals present in input video x. $|n|$ is less than 3 in most cases in our dataset. Hence, this method is chosen to train predicate MLPs.

   The prediction loss is again calculated using BCE loss (see Section 2.1.3). However, it is applied to each fact (respectively to negated facts) now. That is,

$$L_p = -\frac{1}{2}\Sigma_{t \in T} \log(\widehat{t}) - \frac{1}{2}\Sigma_{f \in F} \log(1 - \widehat{f})\,. \tag{49}$$

6. The individual vectors v_i and v_j are initialized to the corresponding word2vec word vectors [129]. For example, for an individual "person", v_{person} is initialised to the word2vec vector for the word "person". These vectors are also updated during the training along with the network

weights.

7. Lastly, the total back propagated loss is the summation of the above losses, given as:

$$L = L_i + L_c + L_r + L_p \,. \tag{50}$$

The final loss L is decreased during training, by updating the network parameters of the encoder, all three multi-classifiers, predicate-MLPs and individual vectors.

5.1.2 Training Setting-2

Building secondary dataset for training setting-2 First, I built a secondary training and validation set for training the logistic regressor.

The fully trained network from setting-1 5.1.1 and Visual Genome statistics built-in 6.3.1 are utilised and the following steps are followed:

1. For each video x, I compute the encoding e.

2. Next, ground truth facts T and an equal number of randomly sampled negated facts F are used to get the predictions from predicate-MLPs. That is, for each ground-truth fact $t \in T$, a prediction $\widehat{t} = m_p(e, v_i, v_j)$ is computed, such that $\widehat{t} \in [0, 1]$. Similarly, for each ground-truth negated fact $f \in F$, where a prediction $\widehat{f} = m_p(e, v_i, v_j)$ is computed, such that $\widehat{f} \in [0, 1]$. v_i and v_j are the individual vectors. The same steps are followed for each unary fact in the ground truth of video x. Let the each prediction from predicate-MLPs $\widehat{t}$ (or $\widehat{f}$) is referred as p_{mlp} here.

3. Similarly, using ground-truth T and F, corresponding predictions from Visual Genome statistics VG are calculated using the steps in 4.3.3. For each fact $t \in T$ and negated fact $f \in F$, I get corresponding prediction p_{VG}.

4. Finally, each training example consists of $\{p_{mlp}, p_{VG}, y\}$, corresponding facts (or negated facts) in the ground truth of x. y is the target for training, $y = 1$ for fact and $y = 0$ for a negated fact.

5. Lastly, all the facts (resp. negated facts), corresponding to which no prediction is produced by VG, due to reasons explained in Section 4.3.4, are removed from the training/validation set.

Training Logistic Regressor The dataset built in Section 5.1.2, is used to train the Logistic Regressor.

Let a weight matrix w be created with random initialization. The logistic regressor learns binary classification, where every training example fed to it is classified as a positive or negative fact for an input x. The target y takes value in the set $\{0, 1\}$ for each training example i.

For training example x_i, the loss function is a log-loss given by J (51):

$$\begin{aligned} J =& \min_{w} C \sum_{i=1}^{n} (-y_i log(\hat{p}(x_i)) - (1 - y_i) log(1 - \hat{p}(x_i))) + r(w) \\ =& \min_{w} C \sum_{i=1}^{n} (-y_i log(\hat{p}(x_i)) - (1 - y_i) log(1 - \hat{p}(x_i))) + \|w\|_1 \,, \end{aligned} \tag{51}$$

where $\hat{p}(x_i)$ is the probability of positive fact, i.e. $P(y_i = 1|x_i, w) = \hat{p}(x_i)$, $r(w)$ is the regularization function, which is l_1 in our case, C is the inverse of regularization strength (higher the value weaker is the regularization strength). The lr during training minimises the cost function J, by adjusting the weights w.

Note: The loss J is actually the BCE loss with regularization which is used everywhere else in this work (discussed in Section 2.1.3).

In our case, the hyper-parameters are: $r(w) = \|w\|_1$, $C = 1.2$, and *saga* solver, with $maximum\ iteration = 200$ is used. The hyper-parameters are chosen by hyper-parameter search explained in section 5.2.

5.1.3 Early Stopping

In training setting-1, I have employed early stopping during training for regularizing the model, and to avoid overfitting, as discussed in detail in Section 2.1.5. In this project, the patience is set to 7 for all training settings.

5.2 Hyper-parameter Search

As discussed in Section 2.1.6, hyper-parameter search is an important task, which can affect the model performance. The full list of hyper-parameters for which I have employed hyper-parameter search are as follows:

- Regulariser strength for the logistic regressor (lr), given by C in 5.1.2.
- Penalty term for the logistic regressor.
- The solver for train setting-2 5.1.2.
- Maximum number of iterations for the solver in 5.1.2, referred as *maximum iterations*.
- Value of q in 4.3.1.
- The value of the threshold for classifying output of predicate-MLPs in positive or negative facts.
- The value of the threshold for classifying output of logistic regressor (lr), as positive or negative fact.
- The learning rate.

There are several hyper-parameters which are applied to either one of the training settings or both of them. As mentioned in section 5.1, I first train in setting-1 and reload the trained model for

setting-2 training for convergence. For this, hyper-parameters used in setting-1 are reloaded as they were on the initial training of setting-1. There are many methods available for searching the optimal hyper-parameters. Search for hyper-parameter q and both *thresholds* are discussed below in Section 5.2.2, and 5.2.1 respectively. Some hyper-parameters were also selected by adopting the standard values used in deep learning literature.

The search space for the remaining hyper-parameters to be optimized is small, hence I can apply the simplest but most reliable method, known as grid search (in deep learning context). I start with a set of values for each hyper-parameter, and then train separate networks using hyper-parameters received by permuting between these values, evaluate the performance, and finally chose the set of values which produces the highest accuracy on the validation set.

5.2.1 Fine Tuning Threshold

To decide if a fact hold true o the given video, the final probability produced by the network in Section 4.2, is compared with a threshold.

It is necessary to fine-tune the threshold value, so as to improve the model's performance. For my work, I am required to fine-tune two different thresholds for each dataset. The first threshold would be used to compare the logistic regressor output. A different threshold is used in cases, where Visual Genome statistics is not used for final prediction, as discussed in Section 4.3.4. In those cases, the output of the predicate-MLPs is compared with the threshold.

I can simply choose a threshold of 0.5, as the commonly used default threshold in literature, [2, 130, 131]. However, a better approach would be to select the threshold based on the performance of the model on the validation set, during inference. This is done as follows:

The inference is performed on the validation data first. For a threshold, $0 \leq threshold \leq 1$, with step of size 0.05, $F1$ score received is compared. The threshold giving the best $F1$ score is chosen

as the final *threshold*.

This is only performed once for each dataset. The values received in our case are:

- For MSVD dataset, threshold = 0.45
- For MSR-VTT dataset, threshold = 0.55

5.2.2 Choosing value of q

As discussed in Section 4.3.1, I choose top q number of facts from U' and B'. Several values are possible for q. As we know from section 4, the vocabulary consists of 285 individuals, 129 attributes and 150 relations. As discussed in Section 4.3.1, there are $\binom{258}{1} \times \binom{129}{1} = 33,282$ candidate unary facts, and $P_2^{258} \times \binom{150}{1} = 9,945,900$ candidate binary facts, which can be tested with predicate-MLPs and VG.

As we know, typically each video has ≤ 3 individuals in it (see 6, [2]), using our vocabulary it would equate to, $3 \times \binom{129}{1}$ pairs + $P_2^3 \times 150$ triples = 1287 possible unary and binary facts. Hence, using this, I roughly choose a value of $q = 1000$ experimentally. A value of < 1287 is used to keep the inference time low.

Further, we will see in the study in Section 8.1, increasing the value of q also increases the performance of our model, but it also increases the overall inference time, as more number of predicate-MLPs need to run corresponding to the attributes and relations.

Below, I also visualized when a correct fact is received from the set of candidate facts from U' and B' (see 4.3.1), with $q = 1000$. For this exploration, the train set is used, and following steps are followed (note: first 2 steps represent Step-1 to 4):

1. The video is encoded to e.

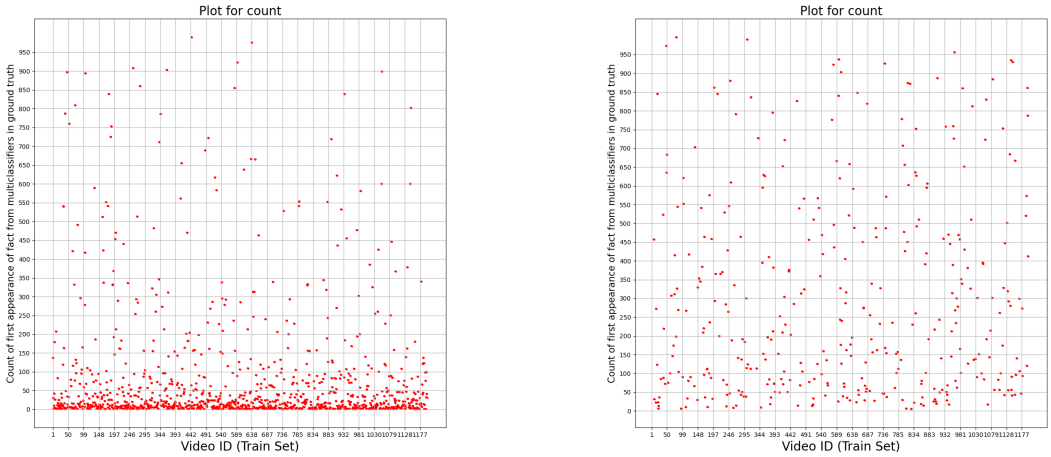

Figure 11: Count for attributes and relations

2. The encoding is fed into g, h and k, to produce set of probabilities P_I, P_C and P_R respectively.

3. Now, the predictions are combined using the framework in 4.3.1, producing U' and B'. Both matrices containing probability values are then sorted in descending order to form an array. From both sorted arrays, I pick facts one by one and check if they are present in the ground truth. If it is, then the count up to which I have to go to get the first true fact is saved. This is done for up to $q = 1000$ only, i.e. the count is only carried till 1000^{th} index in the list. If none of the facts in the first 1000 count is true, then the count is undefined.

The count-value corresponding to each video in the train set is plotted. The x-axis consists of video ID, which is simply the index number of the input video. The left figure in Figure 11 visualises this count-value for unary facts, while the right figure visualises this value for binary facts. As we can see visually, it covers a good range for most of the videos in the training set and so it could be a good choice (it may not be optimal). For undefined count value, nothing is plotted against that Video ID. More discussion on the tradeoff when choosing q value is given in 8.3.

5.3 Choice of Encoder

The pre-trained VGG-CNN has been a popular choice for the encoder in several visual understanding tasks such as visual-question answering, scene graph generation, image captioning [30, 105, 132–134], etc.

As we see, in Section 3.1, image and video captioning literature, over the years, have investigated that a CNN + RNN should be adopted as a standard pipeline for such tasks as encoding an image input [11, 75, 95, 135].

Though the choice of CNN + RNN is popular in image captioning, for video captioning tasks, tne encoder is also required to encode the temporal dimension of the video. Verb prediction tasks popularly employ I3D [61], or SlowFast [136] networks, which have also been used in many video captioning tasks [137].

Hence I could conclude that an encoder with a combination of CNN, RNN and I3D could be a suitable choice for our video annotation task. This framework has also been employed by the baseline model, which is adopted in this work. The ablation study conducted in [2] also shows the usefulness of this encoder.

6 Datasets

6.1 Microsoft Video Dataset

The MSVD dataset is processed to generate MSVD* dataset, which is later used to train and test the proposed deep learning model in Section 4.2. The dataset preprocessing (see 6.1.1) and generation (see 6.4) steps are taken from the baseline work [2], to maintain consistency with the previous baseline work involving this dataset. This is to make the results produced by the proposed model directly comparable with baseline work[2].

Microsoft Video Dataset (MSVD) created by Microsoft Research, consists of 1970 videos in downloadable version (originally 2089 videos were collected using this work, however, some of the videos were removed from YouTube before they could be archived, and so are lost).

Each of the videos is marked with about 40 natural language captions. Both the videos and captions are obtained via Amazon Mechanical Turk workers. To obtain the videos the workers were asked to select an appropriate clip from a YouTube video, by specifying its start- and endpoints. Typically the clips were 4-10 seconds long and depict a single, unambiguous event. For annotation, the workers were asked to describe the main event in each video clip using a single sentence. Lastly, for each given video, each caption was provided by a different worker, for instance, if a given video contains 30 captions, then these were provided by 30 different workers. In this project, for each video, I combine all captions by taking the union of all sets of facts. Further details on the dataset are given in [83].

6.1.1 MSVD data preprocessing

The MSVD data is pre-processed before using it. This is done to make the shapes of each video tensor uniform so that they could be batched and processed by the model more easily. The following

steps are followed to pre-process the MSVD dataset:

1. Each video in the dataset is down-sampled to 8 frames per video frame rate.
2. Next, for each video, all the frames are resized to 256×256 using OpenCV's "resize" function [138].

The standard split for MSVD dataset is 1,200 videos for training, 100 for validation, and 670 for testing, as suggested in [139].

6.2 Microsoft Video to Text Dataset

MSR-VTT is another dataset, which is used in this thesis. It is first processed to generate MSR-VTT* dataset, which is later used to train and test the proposed deep learning model in Section 4.2. The dataset preprocessing and generation (see 6.4) steps are taken from the baseline work [2] because of the reasons explained above.

Microsoft Research also created the Microsoft Video to Text dataset (MSR-VTT). MSR-VTT is a more exhaustive and recent video captioning dataset than MSVD. This is because the videos are collected from 257 popular queries from a commercial video search engine, with 118 videos for each query, spanning across 20 categories. The downloadable version consists of $10,000$ video clips. Each video clip is annotated with 20 English sentences by Amazon Mechanical Turks. For annotation purposes, the workers were instructed to select at most 3 clips from each video. This is because the crawled videos were spanned with several shots, but for annotation purpose, short video clips comprising of a single shot is required. Each clip is about 10-30 seconds long with a mean length of 14.8s, amounting to a total of 41.2 hours of footage. After considering the overlap in search results, about 30,000 clips were received, from which 10,000 were randomly selected for annotation. These 10,000 are drawn from 7,180 different videos. Similar to MSVD, the natural language captions were given by the workers from Amazon Mechanical Turk. Each of the 20 captions was supplied by 20

different workers. Any caption which is too short or duplicated is removed, leaving 20 cations for each video. The complete detail on the dataset can be found in [84].

Similar to MSVD, all the generated captions are combined for each video. The videos are then pre-processed using the same steps discussed in Section 6.1.1, where each video is downsampled to 8 evenly spaced frames and cropped to 256×256 pixels.

Lastly, for splitting the dataset into training, validation and test set, the standard given by authors [84] is followed. There are 6,513 videos for training, 2,990 for validation, and 497 for testing.

6.3 Visual Genome Dataset

One of the novelties of the work in this thesis is the use of an independent dataset, Visual Genome in this case, for producing predictions. As per my knowledge, this thesis is the only work which has attempted to use an external dataset for injecting commonsense knowledge using the novel framework proposed in Section 4.3.3. Further, as per my research, this is also the only work which has attempted this task using the Visual Genome dataset. The model 4.2 is not trained on this dataset. Hence it is independent of it. Below, the dataset, as well as the preprocessing steps, are discussed.

Visual Genome [30] is a very large dataset of images built with the aim to achieve success at cognitive tasks and understanding relationships between objects in an image. The downloadable version has $108,000$ images, where each image has, on average, 35 individuals, 26 attributes, and 21 relations.

The Visual Genome data was collected and verified by workers from Amazon Mechanical Turk (AMT). The dataset contains multiple components for each image, which includes region descriptions, individuals, attributes, relations, region graph, scene graph [31], and question and answers [105], built in multiple steps.

Firstly, the AMT workers are instructed to annotate images using natural language sentences. In each image, the worker is asked to draw three bounding boxes and write three descriptions for the region enclosed in each box. The bounding box needs to cover all the objects mentioned in the description. Once 50 region descriptions is been received for each image, the description is sent to one of the workers, who extracts all the objects from it and grounds the object as a bounding box in the image. For example, if the region description is "woman in shorts is standing behind the man", the objects extracted would be *woman*, *shorts*, and *man*. Once all the objects have been extracted for each region description, the attributes and relations are extracted similarly. The workers are provided with region descriptions and objects and asked to add attributes to objects or connect pairs of objects with relations. Continuing the above example, the attributes extracted will be $standing(woman)$, and the relations would be $in(woman, shorts)$ and $behind(woman, man)$. After this, the generated annotations are verified to eliminate incorrect or vaguely labelled objects or predicates, carried out by workers again. A similar approach is used for building scene graphs and VQA, which are irrelevant to my work and left from the discussions present here.

Finally, the annotations extracted by workers are not constrained by limitations, such as forcing them to refer to a "man" in the image as *man* instead of *person*, *boy*, etc. To remove ambiguity from the dataset, simple heuristics and hand-crafted mapping rules (to avoid common failures) are used to map them to synsets in WordNet (see 2.4) according to WordNet lexeme counts. For example, an entity *man* will be mapped to *man.n.*03 (which in the WordNet database corresponds to a generic use to refer to any human being). To make these mappings more perfect, they are sent to AMT workers along with the top four alternative synsets for each term. The workers verify if the mapping is accurate or change it to an alternative one if it better fits.

The images consists of 108,077 most creative common images from the intersection of Microsoft's MS-COCO's 328,000 images [140] and YFCC100M's 100 million images [141]. Real-world non-iconic images that were uploaded on Flickr by users. However, the dataset is somewhat biased towards images of people but quite diverse overall. For more information about the dataset, please refer to

[30].

Due to the absence of any such dataset for video annotation, an image dataset is chosen for my purpose. Particularly, the Visual Genome dataset was chosen because of the following reasons:

1. It is a very large image dataset (108,077 images) and considerably larger than MSVD (has 1970 videos), and MSR-VTT (has 10,000 videos), hence it increases the chances of more common atoms between Visual Genome and other datasets used here (see Table 2).

2. Further, in the proposed pipeline, I need to extract various statistics from the independent dataset. Using a large dataset is beneficial as the statistics extracted would be more accurate and would reflect real-world concepts.

3. The images in Visual Genome are annotated with scene graphs (where facts have the same structure as a knowledge graph, i.e. $\langle subject, predicate \rangle$ or $\langle subject, predicate, object, \rangle$, which makes them a suitable dataset for the proposed system. Since the dataset is manually generated, there is less chance of error than an automatically generated dataset. This is important to improve the model output by working at the bottleneck of not having a more accurate dataset that the model can look at.

Visual Genome extracted statistics are later used in the proposed model to generate a set of predictions (discussed in Section 4.3.3).

Visual Genome Data Description To get more accurate results, it is essential that many of the atoms from my vocabulary are also present in the Visual Genome dataset. In the absence of the atom predicted by the predicate-MLP, VG statistics are ignored, as using a constant or "0" value in place would be naive. The absence of an atom would not directly mean it is impossible to exist, as the dataset is not completely exhaustive/universal. Table 1 shows the overlap of vocabulary between the Visual Genome dataset and MSVD, MSR-VTT dataset.

Dataset	% Overlap between individuals	% Overlap between attributes	% Overlap between relations
MSVD*	85.61	65.89	30.67
MSR-VTT*	88.77	63.76	33.33

Table 1: Percentage of number of individuals, attributes and relations present in the Visual Genome dataset compared to MSVD* and MSR-VTT*

In Table 2 shows another set of statistics for the Visual Genome dataset. It shows the number of distinct individuals, predicates and data points with at least 1 fact in the dataset.

6.3.1 Extraction of statistics from Visual Genome dataset

Before the Visual Genome dataset can be used to make predictions about facts using the algorithm in 4.3.3, I first need to extract various statistics from it that can represent the distribution of the data in it. Since the ontology used by Visual Genome for annotation is the same as the proposed model (WordNet 2.4), it removes the possibility of any ambiguity in the linking stage. The following steps are used to process it:

For each image in the dataset:

1. All attributes and relations present in it are extracted. The synsets (see 2.4) are converted to *synset_id*, to compare with the dataset used here.

2. All the unary facts $\langle subject, attribute \rangle$ and binary facts $\langle subject, predicate, object \rangle$ are saved along with their count in the dataset. The count signifies, how often that fact appears in the dataset. Additionally, the number of each *subject* and $(subject, object)$ pairs in the dataset are also counted and saved. This count signifies how often *subject* (or $(subject, object)$ pair) can appear (respectively appear together) in the dataset.

3. All the objects with no attribute or relation are skipped. All atoms without equivalent *synset_id* are also ignored.

After the process, I receive four separate files containing statistics related to unary facts, binary facts, count of $subjects$ and count of $(subject, object)$ pairs in the dataset. All the statistics generated here are collectively referred to as VG throughout the thesis, which is used in later stages to calculate prediction by applying the Bayesian framework as discussed in Section 4.3.3.

6.4 Dataset Generation

As discussed in section 4, the proposed neural model requires the video captions to be a set of facts (which are equivalent to KGs (2.3.3). There are several existing video-captioning datasets, but the videos are annotated with natural language captions. In the absence of any appropriate dataset, an automatic generation method given in [2] is adopted with minor modifications. Two well-known video-captioning datasets are used: MSVD [83] and MSR-VTT [84], to create MSVD* and MSR-VTT* datasets suitable to train and test the deep learning model in 4.2. The generation method is described below.

1. First, the NL sentences in the caption are parsed using a rule-based parser based on the Stanford NLP syntactic parser [142]. This will produce a dependency parse of the sentence, where the part of speech for each word and the syntactic relations that hold true of them will be identified.

2. Then, from the above syntactic parse, atoms are formed by applying a sequence of rules. To find the rules and algorithm used for this step, please refer to [2]. The atoms are then used to form facts. Each fact contains a predicate and the corresponding arguments. I am only considering two cases here: $\langle subject, predicate, object \rangle$ triples in case of binary facts, and $\langle subject, predicate \rangle$ pairs in case of unary facts (see Figures 10, 12, 13).

3. Next, it is also important to perform automatic word-sense disambiguation by linking all the predicates and individuals to a structured knowledge base (ontology). Word-sense disambiguation is necessary for polysemous words, which are words with more than 1 distinct sense. To

return to the example, in Figure 10, the word 'man' can mean anything among these: males of the human species or (a verb) to operate or constitute a vehicle or machine (example: To man a ship). The output of the parser would indicate the former over the latter. The ontology used for linking is WordNet [123]. Throughout the thesis, WordNet is also referred to as the ontological knowledge base (KB), as often done in the literature. More discussion on WordNet is given in 2.4.

Since the linking method needs to be context-sensitive, i.e. a word's contribution in the sentence depends on the context in which the word is uttered, the linking method needs to find the most suitable WordNet synset. This is done by 1) computing doc2vec [129] representation of the original NL sentence from which the word w was taken; 2) doc2vec representation of the definition of each WordNet synset containing the word w is taken; 3) then, the synset which has the most similar vector representation is picked as the synset for the word w. Formally,

$$syn = \underset{\{syn' | w \in syn'\}}{\operatorname{argmax}} \; doc2vec(def(syn'))doc2vec(s)\,, \tag{52}$$

where $def(syn')$ corresponds to the definition of synset syn', s is the original natural language annotation sentence from which word w is taken.

This formalisation of vocabulary by linking it to an ontology (WordNet, in this case) allows us to apply inference rules to supplement the annotations produced by the network.

4. Now, the set of atoms (linked to WordNet) forms the corresponding logical annotation. All the logical annotations for each video are merged by taking the union of all the constituent atoms.

5. Lastly, semantically weak verbs and all the words that appear less than 50 times across that dataset are excluded. This is because words appearing a few times would not produce a good performance. Verbs such as "do", "take", "be", and "have" are considered semantically weak verbs. They generally function as copulas or syntactic operators and do not convey relevant information about the video. For instance, the parse of the caption "man is climbing

a mountain" should not include the atom $be(man)$ [2].

6. The method described here only produces non-negated facts, also referred to as T in this thesis. This would imply that the model trained on this data could simply learn to predict every potential fact to be true. To solve this, local closed-world assumption (LCWA) [143] is used to create a set of negated facts (referred to as F here). For example, the fact $play(man, guitar)$, which appears in Figure 10 could be corrupted to get a negated fact $\neg throw(man, guitar)$. This negated fact is then added to F. More discussion of LCWA and the generation of negated facts is presented in Section 2.5.

After all the logical captions are generated and linked to WordNet, another step is taken to convert the logical captions in the dataset into a set of real-valued vectors, which can be used by the deep learning model 4.2. This is done by converting them into multi-categorical one hot encoding vector, i.e. the categorical variables are represented in a multidimensional space by encoding them into a numerical value i.

Finally, all the KGs for each video caption are merged by taking a union of facts and their parses. In MSVD and MSR-VTT, each video has multiple captions and produces a separate training example. In MSVD* and MSR-VTT*, each video only appears in one training example, annotated with a merging of all captions. For detailed information about the formation of the MSVD* and MSR-VTT* dataset, please refer to [2]. After the above merge and exclude steps, the statistics about the MSVD* and MSR-VTT* datasets are shown in Table 2. It shows the number of distinct individuals, predicates and data points with at least 1 fact or negated fact. The column "Num Facts" indicates the number of unique facts in the dataset. Additionally, it also shows the corresponding values for the Visual Genome dataset discussed in 6.3.

Dataset	Number of Training Example	Number of Individuals	Number of Attributes	Number of Relations	Number of Facts	Number of Non-empty Training examples
MSVD*	1970	122	48	69	117	1800
MSR-VTT*	10000	372	235	113	348	9802
Visual Genome (Image dataset) [30]	108,077	75,729	40,513	40,480	-	80,993

Table 2: Dataset statistics showing: number of distinct training example in each dataset, number of distict individual, number of distict attributes and number of distict relations, followed by number of unique facts in the dataset. The final column shows the number of of data points that remained with non-empty captions after infrequently occuring words were excluded (see Section 6.4) .

7 Results and Analysis

7.1 Evaluation Metrics

The training aims to produce accurate logical descriptions of the video when the video is fed into the proposed system (see 4). This work aims to build a system which can input a video and produce a representation of it in a form of a knowledge graph. So the model's performance is evaluated in terms of accuracy scores on the truth of logical atoms.

As described in section 6.4, the model is trained with manufactured negated facts F as well. Because F is produced artificially, the number of data points in it is much larger than the original set of facts T. Therefore, it is misleading to simply report the global accuracy of the proposed model. This is because, even if the model predicts every atom as negative, it will still has high accuracy, as there are many negated facts in the augmented ground truth caption.

Instead, separate accuracy on positive and negative facts are reported here. I also calculate the F1 score, which is a combination of precision and recall scores. Precision is the proportion of atoms which are predicted to be true and are actually true in the ground truth. Whereas recall gives the proportion of a number of true atoms in the ground truth predicted to be true. F1 is then computed using a harmonic mean of precision and recall scores. F1 score is an evaluation metric to measure the model's performance on the dataset, commonly used for evaluating classification systems.

Formally,

$$
\begin{aligned}
true\ positive\ (t_p) &= \{i | y_i = 1, \hat{y}_i = 1\}\,, \\
false\ positives\ (f_p) &= \{i | y_i = 0, \hat{y}_i = 1\}\,, \\
true\ negatives\ (t_n) &= \{i | y_i = 0, \hat{y}_i = 0\}\,, \\
false\ negatives\ (f_n) &= \{i | y_i = 1, \hat{y}_i = 0\}\,, \\
precision &= \frac{\#t_p}{\#t_p + \#f_p}\,, \\
recall &= \frac{\#t_p}{\#t_p + \#f_n}\,, \\
F1 &= \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{2(precision \times recall)}{precision + recall}\,,
\end{aligned}
\tag{53}
$$

where # denotes the set cardinality. Please note that recall is equivalent to positive accuracy, which is directly reported here.

7.2 Quantitative Results

In this section, the quantitative experimental results for MSVD* and MSR-VTT* datasets are reported. The results are further analysed and compared with other existing works. The hyper-parameters and implementational details for the experiments performed is discussed in Sections 5.2 and 5 respectively. All results are taken from a held-out test-set, using the train, validation, and test splits of the datasets (defined in Sections 6.1 and 6.2 for MSVD* and MSR-VTT* dataset, respectively). To the best of my knowledge, there are only two existing works that previously attempted the task of logical video annotation, which is been used to benchmark the performance of the proposed model[1, 2].

The F1, positive, negative and total accuracy scores for MSVD* and MSR-VTT* datasets are given in Table 3 and Table 4 respectively. It also shows the respective results from the baseline model reported in [2], called "LM 2020". Table 3 also includes result from another model, called as "VL

2018" [1].

As we can see, the proposed model significantly outperforms both baseline models in f1 score, positive and total accuracy. Importantly, it gives superior positive accuracy, the most difficult metric to score highly on.

Further, the problem with earlier works, and particularly in the baseline model [2] was that the model was predicting most of the facts as negative. As we know from section 6.4, the artificially constructed datasets have a higher percentage of negative facts than positive ones. This means that even if the model predicts everything as false, the negative accuracy would be very high, while the positive accuracy would be nearly 0. However, the proposed model is not doing this because of which the positive accuracy is far better than the baseline, while the negative accuracy is lower compared to the baselines. This is highly desirable.

Table 3: Results on the MSVD* video dataset annotation with KG. As described in section 4. The best results are in bold.

	F1-score	Positive Accuracy (%)	Negative Accuracy (%)	Total Accuracy
Proposed Model	**29.03**	**28.07**	91.74	**81.05**
Baseline (LM 2020)	13.99	12.65	**99.20**	22.16
VL 2018	6.11	3.36	-	-

Table 4: Results on the MSR-VTT* video dataset annotation with KG. As described in section 4. The best results are in bold.

	F1-score	Positive Accuracy (%)	Negative Accuracy (%)	Total Accuracy
Proposed Model	**37.23**	**34.07**	94.15	**84.08**
Baseline (LM 2020)	11.83	6.76	**99.96**	83.01

7.2.1 Comparison of MSVD and MSR-VTT Results

As we can see, results for MSR-VTT* are significantly better than those for MSVD* dataset, with differences being found among all the metrics reported.

Surprisingly, it is opposed to the results reported using the existing models in the literature, including the baselines [1, 2, 126, 144–147], where lower video captioning / annotation performance on MSR-

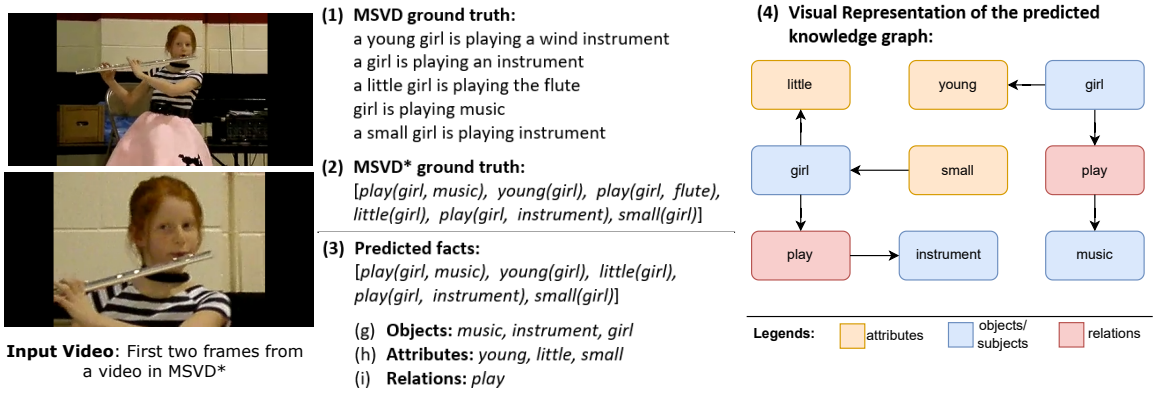

Figure 12: The first two frames from MSVD*, with (1) ground-truth natural language captions in MSVD, (2) the ground-truth set of facts in MSVD*, (3)the facts predicted by the proposed model, with (a)objects/subjects present, (b)attributes predicted, (c)relations predicted, and (4) visual representation of the knowledge graph produced

VTT is received compared to MSVD. The reason is the greater size of MSR-VTT vocabulary: 29316 compared to $13,0101$ for MSVD (figures taken from [2]), which makes the task more difficult.

However, from the results produced by the proposed model, we can see that the proposed model could use the fact that MSR-VTT has a larger size to its advantage. MSR-VTT* has 6517 training examples, as compared to 1200 in MSVD* (given in section 6). This meant that the model could see more examples and optimise the parameters, because of which we could see an improvement in my results. Unlike the work in [2], the model is no more only predicting most facts as false, hence more training data is actually helping in improving the result in my case.

Further, because of using separate multi-classifiers for predicting all the components in fact and combining them using the methods in section 4.3.1, I am are already filtering irrelevant unary and binary facts, because of which when facts are fed in predicate-MLPs more accurate result is produced. This is opposed to the work in "LM 2020" [2], where every combination of individuals with all the predicates in vocabulary is fed into predicate-MLPs.

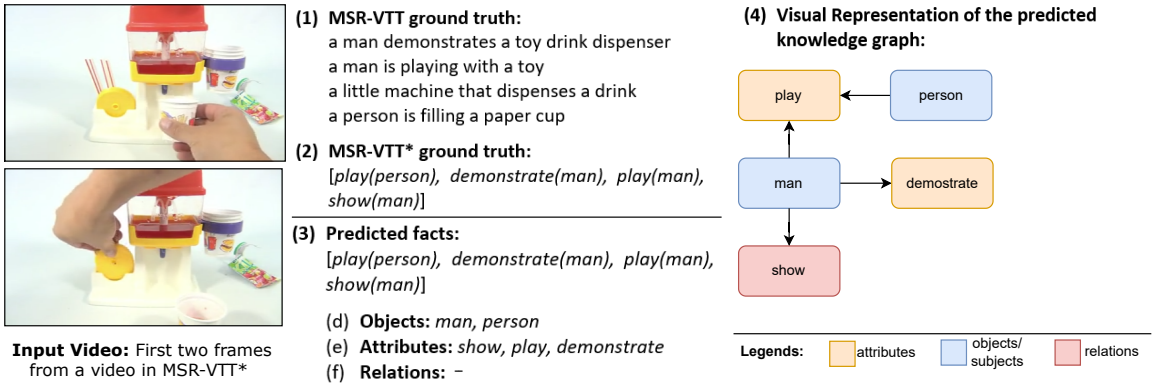

Figure 13: The first two frames from MSR-VTT*, with (1) ground-truth natural language captions in MSR-VTT, (2) the ground-truth set of facts in MSR-VTT*, (3)the facts predicted by the proposed model, with (a)objects/subjects present, (b)attributes predicted, (c)relations predicted, and (4) visual representation of the knowledge graph produced

7.3 Qualitative Results

To further evaluate the quality of the KGs produced for the video by the proposed model, manual inspection of videos and predicted facts is carried out. Figure 12 shows the first two frames from a video from MSVD* with the facts predicted by the model (Figure 10 shows another example for MSVD*.). The same for a video in MSR-VTT* is shown in 13.

The qualitative examples show the limitations imposed by the smaller vocabulary size in MSVD*. As discussed in Section 6.4, the individuals, attributes and relations which appear fewer than 50 times are excluded from the dataset. This implies that at times there could insufficient material to describe a video fully. For example, the girl in the video in Figure 12 is playing the flute, which is also expressed in one of the MSVD captions. However, $flute$ appears less than 50 times and so it cannot predict $play(girl, flute)$. Though the model is correctly able to identify it as an *instrument*. It could also correctly identify other attributes, relations and individuals present in the video. Recall from Section 6.4 that there are multiple annotations for each video which are merged in the dataset used here.

Next, Figure 13 shows that the model can predict all the facts correctly for the video. However, it shows another limitation, which was beyond the scope of this work. The automatic dataset

generation process in 6.4, the ground truth annotation could not expresses some facts about the video such as objects *paper_cup*, *toy* etc. They are excluded from the ground truth, so the model is not trained on them and cannot predict them. This emphasises the need for a manually generated structured video annotation dataset to avoid such cases.

Further, during qualitative analysis, it is observed that the model could correctly predict all the facts present in ground-truth, for more number of videos in the MSR-VTT* dataset, compared to the MSVD* dataset. This also confirms the observation made in 7.2.1, where MSR-VTT* produced better results than MSVD* dataset.

Lastly, as discussed in Section 6.4, because of using an ontological knowledge base like WordNet, inferring additional facts, later on, is trivial. Though additional inferences are not produced in the present work, it can easily be done using the Wordnet knowledge base, as explained here. For example, the class *man* is a subclass of *person* and *male*. So, later on, it is possible to apply inference to all facts mentioning a *man*. If it is needed to determine how many videos in a database depict at least one person, or males, this would be possible here, but not in the case of merely annotating them with NL sentences. This is one of the advantages of this approach of using KGs for annotation.

8 Discussions

8.1 Improvement in model performance by changing the value of q

As we discussed in Section 5.2.2, there are many possible values for q, ranging from $0 < q \leq 33282$. Smaller values of q would mean the number of unary and binary facts fed to predicate-MLPs is small. This would result in a smaller inference time, but the $F1$ score will be inferior. This is because many of the combinations could be false, as the dataset consists of more negative facts than positive facts. Feeding a bigger pool of facts to the predicate-MLPs increases the chances of receiving true facts. Hence, increasing the value of q may increase the chance of getting a true fact.

Here I perform experiments with different values of q to study the effect of changing the value of q on the overall performance of the model. The study is performed on the test set. Here I have omitted the last step of using VG statistics. This is because using VG for inference increases the overall time complexity of the model. Since this is meant to be a comparative study, I use predicate-MLPs probability value to get the final set of facts for an input. The performance as well as time-taken by the model to produce output is noted.

The experiments are performed on both MSVD* and MSRVTT* datasets, where the value of q is varied from $200 \leq q \leq 30000$, with varying step sizes. While smaller steps would result in better-approximated curves, it is expensive to run them. Hence, the experiment with MSVD* has a smaller step size than MSR-VTT* because the MSR-VTT* dataset size is much larger and running multiple experiments on it is expensive.

Further, the time taken to perform the experiments is noted here. As it depends on the processor used and the load on it at the time of the experiment, the time taken reported here is only representative of the actual value. Since the experiments are conducted on shared GPU servers, the load on it has varied throughout the experiment, which affects the overall experiment time.

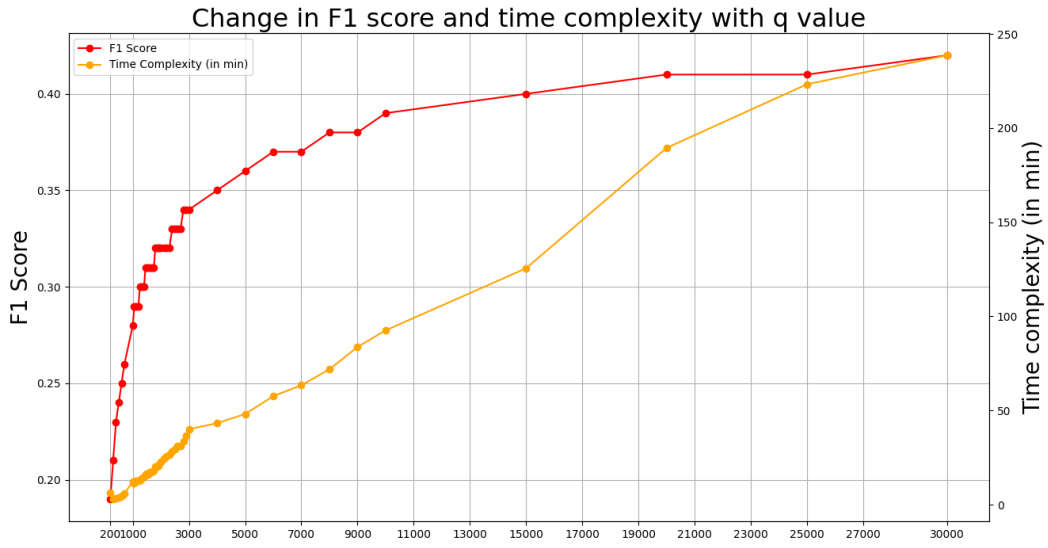

Figure 14: Plot for MSVD* for change in F1-score and time-taken(in min) to perform the experiment with change in value q (q is defined in Section 4.3.1)

Figures 14 and 15 shows the plot for MSVD* and MSRVTT* dataset respectively. The 'red curve' plots the F1 score on the y-axis with the corresponding q value on the x-axis. The 'yellow line' on the plot shows the time taken for the inference (in minutes) on the second y-axis, corresponding to the q value on the x-axis and the F1 score on the first y-axis.

Interestingly, we can see that the F1 score continuously grows, with an almost exponential increase for the first few epochs, while for higher epochs, the growth rate slows down, with a nearly linear increase in time-taken for inference, with an increase in value of q. As expected, the F1 score grows fast initially, with decreasing growth rate with every increase in q. This implies that with better computational resources, the proposed model will be able to achieve ever better performance by using higher value of q. This is superior to the baseline results reported in section 7.2.

Table 5 shows the F1 score for both the datasets on two extreme values of q - 200 and 30000. We can see that even a small value of $q = 200$ produces better result than the baseline model (LM 2020) in both datasets. This shows the superiority of the proposed method of using separate multi-classifiers for all the components in the fact and combining them using the technique mentioned in 4.3.1.

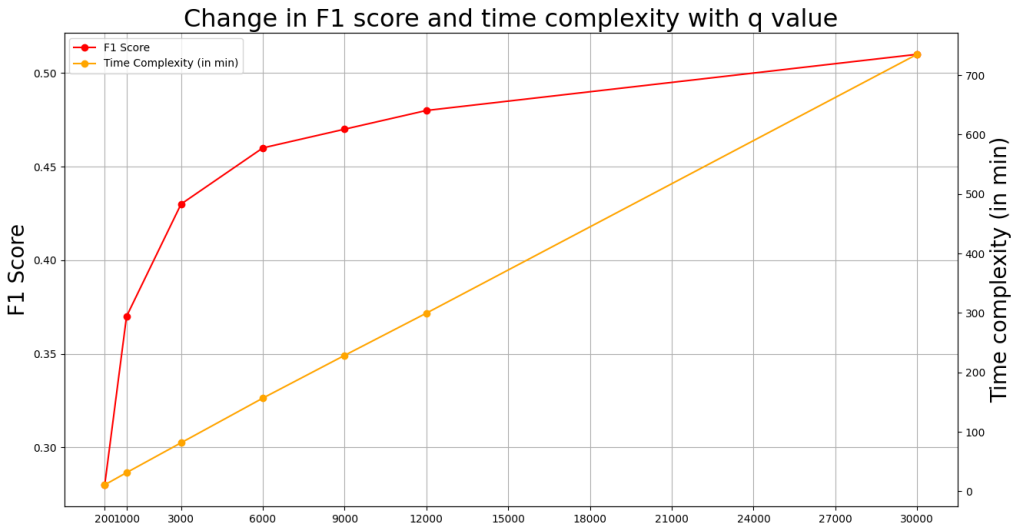

Figure 15: Plot for MSR-VTT* for change in F1 and time-taken(in min) with q to perform the experiment with change in value q (q is defined in Section 4.3.1)

Table 5: F1 score and time-taken (in min) for inference with $q = 200$ and q=30000 by the proposed model, compared with the F1 score generated by the baseline model [2]

Dataset	q	**F1-score**	**Baseline F1**	**Time-Taken (in min)**
MSVD*	200	18.65	13.99	6.41
	30000	41.68		238.68
MSRVTT*	200	28.45	11.83	10.78
	30000	50.79		735.09

8.2 Investigation of the contribution of VG statistics

To investigate the contribution of Visual Genome statistics in the KG generation, I perform an ablation study on it.

The experiments are done for two scenarios:

1. Predictions supported by Visual Genome statistics by following steps - 1 to 8 from Section 4.3.
2. Prediction using only predicate-MLPs by following steps- 1 to 6 in Section 4.3

In both scenarios, predicate-MLPs are trained for different epochs, starting with 0 up to training

convergence. The other components of the network, namely - encoder, individual-, attribute-, and relation-multi-classifiers are reloaded from a fully trained network. For each separately trained predicate-MLP, a dataset for the logistic regressor is built using the pipeline in section 5.1.2, and the logistic regressor is trained to convergence. This is because the logistic regressor training data use prediction from predicate-MLPs, so at different epochs, the predictions would be different, meaning the logistic regressor has to adjust the weight w parameter accordingly.

I only train the predicate-MLPs at different epochs while other components are reloaded from the fully-trained network because the aim is to investigate the effect of using Visual Genome predictions on the final output when predicate-MLPs are trained at different epochs, only.

Figure 16 plots the results for MSVD* dataset. As we can see, at epoch=0, i.e. when the predicate-MLPs are untrained, the F1 score is far better when using second predictions from Visual Genome. This signifies that when the network did not have any information about the dataset, VG statistics representing general world knowledge helped the predictions the most.

As I increase the number of epochs, the network learns more about the particular dataset, and the F1 score in both scenarios becomes close to each other, with ultimately scenario-1 giving a slightly better result on the fully trained network. There is 1% increase in F1 score compared to scenario-2, at training convergence. This is expected because, as we can see, the overlap of components between MSVD* and Visual Genome is quite low (given in Table 1). Hence using the algorithm from section 1 for final prediction, many of the facts predicted do not use Visual Genome prediction. This decreases the overall effectiveness of using the Visual Genome predictions on the final ouput. However, even after this, there is a slight increase in the F1 score at the end using Visual Genome. This signifies that a better dataset, specifically made for videos, can further support the result generated by the proposed model.

Figure 17 shows the output for MSRVTT dataset. Similar to MSVD* dataset, we can see a high F1 score at epoch = 0, for scenario-1, compared to only using predicate-MLPs (scenario-2). It is to be

noted that the F1 score at this epoch is significantly better than the baseline model [2]. This signifies the usefulness of using an independent dataset to support the results in the proposed model. As we would expect, as I train the predicate-MLPs with more epochs, the score gets better. However, as the epochs increases, we can see that the scores using VG are slightly inferior to those without it. This is because, as discussed above, the overlap between its components and MSR-VTT* is quite low. Further, as alluded in section 7.3, the MSR-VTT* contains a larger vocabulary, which resulted in more logical atoms excluded from the captions. This means there are more captions with missing information which leads to a weakening of the semantic relations between the video and the caption. Since overlap with Visual Genome vocabulary is already quite low in this case, VG statistics are not able to model the output properly. Hence, for such cases, a more comprehensive dataset is required (further discussed in 9).

Lastly, the results produced by the proposed model for this experiment signify my purpose of using an independent dataset to support the predictions produced. The use of predictions from the Visual Genome dataset increases the probability of correct prediction. This is because if a fact predicted by the network also occurs in nature (VG in this case), there is more chance of it being true. It also avoids cases which are impossible to occur; for example, $drive(man, car)$ is possible; however, $sing(man, car)$ is never possible. In this case, the statistics would produce very high confidence in the former fact while very very low confidence in the latter fact. This is an important problem to solve in semantic understanding work because, along with correct, I also need to avoid impossible scenarios.

8.3 Ablation on the combining framework in Section 4.3.1

As discussed in 4.3.1, the procedure to combine the output from the individual-, attribute-, and relation-multi-classifiers to produce candidate unary and binary facts is an essential step in filtering irrelevant facts. To investigate the contribution of the combining procedure used, I perform an ablation study on this network component.

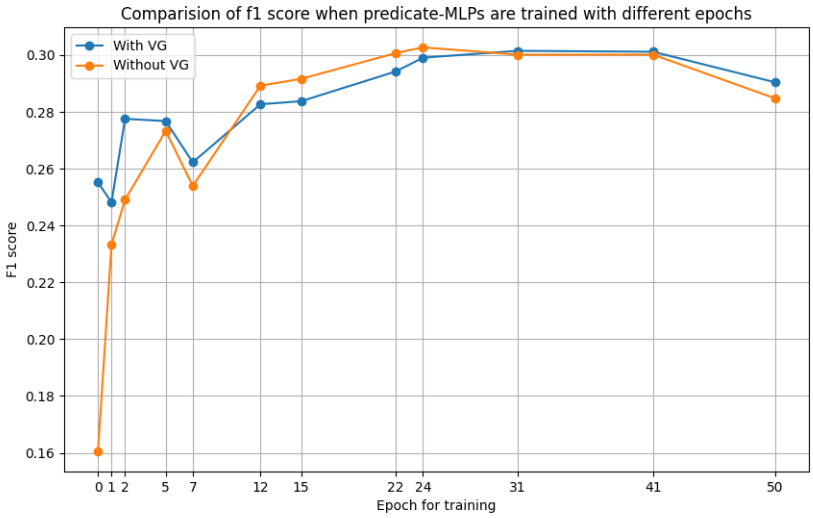

Figure 16: Investigation results for MSVD* dataset. The predicate-MLPs are trained at different epochs while the rest of the model is reloaded from fully trained network. The curves compare results for experiments for scenario-1 (prediction using predicate-MLPs and Visual Genome, shown by blue curve on the plot) and scenario-2 (prediction only using predicate-MLPs, shown by orange curve on the plot). The plot shows change in F1 score with increase in number of epochs during predicate-MLP training. Epoch=0 signifies that the predicate-MLP is untrained and is initialised with random weights.

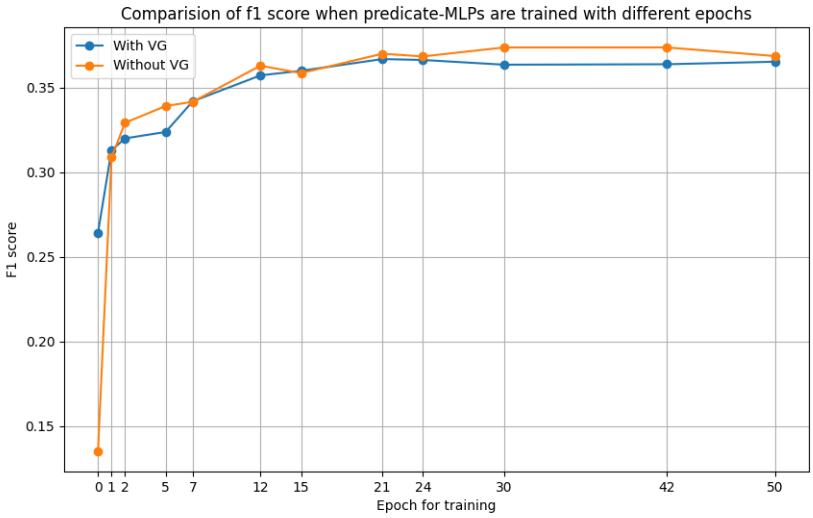

Figure 17: Investigation results for MSR-VTT* dataset. The predicate-MLPs are trained at different epochs while the rest of the model is reloaded from fully trained network. The curves compare results for experiments for scenario-1 (prediction using predicate-MLPs and Visual Genome, shown by blue curve on the plot) and scenario-2 (prediction only using predicate-MLPs, shown by orange curve on the plot). The plot shows change in F1 score with increase in number of epochs during predicate-MLP training. Epoch=0 signifies that the predicate-MLP is untrained and is initialised with random weights.

In "without combiner" setting, the naïve approach given in Section 4.3.1 is followed. The combiner is replaced with a simple thresholding method to get I', C' and R'. Then all possible combinations of them are used to form U' and B'.That is,

That is,

$$
\begin{aligned}
I' &= \{i \in I | g(f(x))_i > threshold\}\,, \\
C' &= \{c \in C | h(f(x))_c > threshold\}\,, \\
R' &= \{r \in R | k(f(x))_r > threshold\}\,, \\
U' &= I' \times C'\,, \\
B' &= I' \times I' \times R'\,, \\
T' &= U' \cup B'\,.
\end{aligned}
\tag{54}
$$

The other components of the network are kept the same. The Step-5 (in Section 4.3) is replaced with the above method, while all the steps are kept same. This tests whether the combining method is actually filtering irrelevant facts or not and further understands how much it is affecting the accuracy scores.

Tables 6 and 7 shows the results for MSVD* and MSR-VTT* datasets respectively. The results in this setting are significantly worse than the proposed model. This is because many of the irrelevant pairs and triples are fed into the predicate-MLPs, without considering the fact that each component should be considered with others rather than individually. This shows the effectiveness of the combining technique 4.3.1.

Table 6: Ablation results on the combining technique given in Section 4.3.1 on MSVD*. The combining method here is replaced by a simple thresholding method. The best results are in bold

	F1-score	Positive Accuracy (%)	Negative Accuracy (%)	Total Accuracy (%)
Proposed Model	**29.03**	**28.07**	91.74	81.05
Without Combiner	10.6	7.75	**98.96**	**83.65**

Table 7: Ablation results on the combining technique given in Section 4.3.1 on MSR-VTT*. The combining method here is replaced by a simple thresholding method. The best results are in bold

	F1-score	Positive Accuracy (%)	Negative Accuracy (%)	Total Accuracy (%)
Proposed Model	**37.23**	**34.07**	94.15	84.08
Without Combiner	13.6	9.01	**99.66**	**84.47**

However, it is to be noted that the results in this ablation setting are better for MSR-VTT* than MSVD*, which agrees with the quantitative results in Section 7.2. This signifies that the model can still use and learn from a larger number of training examples from MSR-VTT*, giving better results for it.

Moreover, because here I am using class- and relation-multi classifiers for predicting attributes and relations, respectively, I only need to use a subset of attributes and relations rather than using all combinations from the vocabulary (as was done in the baseline [2]), the proposed model is still more efficient and filters some of the irrelevant predicates. We can observe that the F1 score and positive accuracy for MSR-VTT* in this setting are better than the baseline [2].

Lastly, it is worth noting that from the experimental results in Section 8.2, we can see that the results for the scenario-2, where KGs are only produced using predicate-MLPs, at epoch = 0 (i.e. when the predicate-MLPs have not been trained at all and is initialised with random weights), when compared with the baseline results [2], the F1 scores are better for both datasets. This further validates my choice of using different multi-classifiers for all the components in a fact and combining them using the method in 4.3.1, due to which many of the inapplicable facts are excluded before they can be fed to the predicate-MLPs. The comparison can be seen in Table 8.

Table 8: F1 score produced by MSVD* and MSR-VTT* dataset using scenario-2 experiment, where predicate-MLPs are untrained (epoch = 0) while the rest components of the model are reloaded from fully trained network. The results are compared with baseline model [2]. The best results are shown in bold

	F1-score	Baseline [2]
MSVD*	**16.05**	**13.99**
MSR-VTT*	13.51	11.83

9 Conclusions and Future Works

This thesis has presented the design, implementation and performance of a system designed to analyse the semantic content of videos. The novel feature of this system is that it extracts the semantic content in the form of knowledge graph (KG) rather than natural language. Thus, it can incorporate background knowledge (by incorporating the knowledge base, WordNet in this case (Section 2.4)). As per my knowledge, there are only two research works that has attempted the same task [1, 2]. My system significantly outperforms both the models in both positive accuracy and F1 score, which are more difficult to achieve. It also has much better generalization capability compared to [1, 2].

In the proposed system, first, three conditionally-independent multi-classifiers are used to predict the individuals, attributes, and relations, respectively. These three components are combined to form the set of candidate facts in a way to avoid the inclusion of irrelevant facts. Next, the facts from the candidate sets are fed into predicate-MLPs to predict if it holds true of the input video. A second set of predictions is also produced using an independent dataset on which the model is not trained to incorporate commonsense knowledge. The two sets of predictions are combined using a binary classifier to get the set of final predicted facts. Finally, the knowledge graph for the input video is formed by taking a union of all the facts in the final predicted set.

The key contributions and novelty of this work is summarized below:

1. I have proposed a framework for utilising an independent dataset to inject "commonsense knowledge" of which predicates are likely to hold true of which individuals in general. That is, for each potentially true fact, I want to estimate the probability that it holds true of a randomly chosen video. This "commonsense knowledge" is essential in avoiding cases which are unlikely to occur. As per my knowledge, this way of injecting commonsense knowledge has never been attempted in literature before.

Particularly, due to the lack of any video dataset which can be utilised here, I have selected the Visual Genome dataset [30], which is a very large image dataset where images are annotated with scene graphs (see 6.3). Next, extracting predictions in a correct way from the independent dataset is also an important step. Instead of relying on simple heuristics, I have given a foundational basis for the proposed framework. The prediction of the presence of a fact in the real world, conditioned on the Visual Genome dataset, is given by calculating the expected value for the fact under the Bayesian posterior of the presence of the fact, given the Visual Genome dataset. In Section 4.3.3, this value is derived and comes out to be a simple fraction of some statistics that can be easily extracted from the dataset (see Equation 43). These statistics only need to be extracted once and thus will not require any extra computational resource during the test time, which is another advantage of this method. The theoretical underpinnings of this step will let me use any such dataset in place of Visual Genome and calculate prediction directly using Equation 43 in future. As per my knowledge, this work is also the first to use scene graphs from Visual Genome for this purpose.

2. From the investigative study in Section 8.2, to explore the impact of using an independent dataset for injecting commonsense knowledge, we can see that this step has given my model the ability to produce good predictions even when the model is not at all trained and is initialised with random values. This step paves the way for overcoming the issue with the availability of fewer data for training the model and tries to make the model data agnostic, such that it could perform equally well across various datasets without any bias. Ofcourse, better the dataset (i.e. more exhaustive) used for incorporating commonsense knowledge, better will the produced results. This overcomes the shortcoming which is highlighted in Introduction (Section 1), where many previously built models have shown good results only on some datasets.

3. The work also proposes an approach for combining the results from three conditionally independent multi-classifiers, namely, individual-, attribute- and relations-multi-classifiers, for building a set of candidate facts. I also utilise this step to filter out irrelevant facts. As we have

seen in Section 8.3, this step has helped in drastically improving the results. I mathematically ground this step using the derivation given in Section 4.3.1. This makes the approach more effective while generalizing its use.

4. In section 8.1, I have also shown how controlling a hyper-parameter q could even further improve the results generated by the proposed model. Another important observation made here is that this improvement only results in a near-linear increase in the time required to build KGs from the videos (The trade-off is plotted in graphs 14 and 15 indicating better performance with improvement in processing power in future). This is because many of the steps are designed to make the overall performance more efficient by exploiting alternative ways, which result in faster operation. I have focused on this as well because it is important if I want the proposed model to operate in the real world.

5. Numerous investigation and ablation studies are conducted in Sections 7 and 8, to study the effectiveness of various components. The model has shown significant improvement compared to the existing models. For MSVD* dataset: $F1 = 29.03$, $positive\,accuracy = 28.07$, $total\,accuracy = 81.05$, (compared to $F1 = 13.99$, $positive\,accuracy = 12.65$ and $total\,accuracy = 22.16$, received by the baseline work [2]). For MSR-VTT* dataset: $F1 = 37.23$, $positive\,accuracy = 34.07$, $total\,accuracy = 84.08$, (compared to $F1 = 11.83$, $positive\,accuracy = 6.76$ and $total\,accuracy = 83.01$, received by the baseline work [2]). My model has outperformed the baseline significantly on metrics such as $F1$ and $positive\,accuracy$ (which is more difficult to perform better on, as seen in [2].) This is because my model could overcome the common obstacle found in this area of work - predicting most of the facts as false because the model sees more negative examples than positive ones.

   Further, as opposed to the literature, where it is difficult to perform better in the MSR-VTT dataset, my model actually performed much better on the MSR-VTT dataset compared to MSVD, which shows that the model has the potential to learn large vocabulary in MSR-VTT.

Apart from these advantages, my proposed framework offers several other benefits and gains over

the common task video annotation (also highlighted in Sections 1 and 3). Firstly, rather than producing a single sentence to describe the video, it produces an abstract representation of the semantic content, making it possible to later query on to recover whatever information is needed. The knowledge graph's nodes are linked to the WordNet (knowledge base), which makes it possible to incorporate background knowledge and get abstract information. One use case of the model would be to automatically tag a database of videos such that the user could query the database to find out information, such as how many videos depict living organisms or depict some social interactions etc., without the need to watch those videos. Another use case could be the interpretation of video content for the visually impaired. The user can recover commonsense inferences that any human will make from the video content (because the nodes in the knowledge graphs are linked to a knowledge base (WordNet)), which will provide a significant advantage over the generation of a single natural language sentence for annotation.

Thus, while annotating a video with a knowledge graph (consisting of a set of logical facts) has been the main focus of my work in this thesis, the proposed system is, in fact, more versatile than that. It produces better interpretable representation of the semantic content of the video. This is because a typical DL architecture often produces vectorised representation which cannot be understood by direct examination. For example, the CNN used in encoder VGG19 (Section 3.3.1), effectively represents the semantic content of an image (evident from the paper [59] and the Section 3) using a feature vector. However, this feature vector is uninterpretable, and humans cannot understand what it says about the image. On the other hand, facts produced by my network are in the form of entity ID from WordNet, which can directly be converted to a word that humans can understand. Secondly, the injection of commonsense knowledge using an independent dataset makes the model effectively generalise its output on videos from larger domain. As seen in Section 3, many video annotation models lack generalisation capability, which was a major shortcoming.

Finally, based on the observations made in this thesis and the fact that the use of knowledge graphs is still in a nascent stage for video annotation, there are a few possible future directions that would

be interesting to explore. These are summarised below:

1. As highlighted throughout the thesis, and especially in 7.3, there is a need for manually constructing a dataset specifically designed for video annotation using KG rather than automatically generated datasets.

2. As we can understand, injecting prior or commonsense knowledge is an essential task because, as humans, machines should not predict something impossible. To this end, Visual Genome was used in this work; however, as we see in Table 1, the overlap between Visual Genome and the MSVD/MSR-VTT vocabulary is very low. Hence, an exhaustive and manually generated dataset for videos is required.

3. Currently, to produce the second prediction, I am utilising statistics from an independent dataset. It would also be interesting to explore the effect of injecting these statistics before training the model. Currently, the predicate MLPs, all three multi-classifiers and the individual vectors are initialised randomly. Instead, I can also pre-train them on the independent dataset and see if the information can be transferred while the model makes predictions from other datasets.

4. As KG offers several advantages, it is also good to explore annotation using KG in other input domains. Such as application to text, where the model could perform a task similar to open information extraction.

I will explore the above options in my future research.

References

[1] D. Vasile and T. Lukasiewicz, "Learning structured video descriptions: Automated video knowledge extraction for video understanding tasks," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, Springer, 2018, pp. 315–332.

[2] L. Mahon, E. Giunchiglia, B. Li, and T. Lukasiewicz, "Knowledge graph extraction from videos," in *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, IEEE, 2020, pp. 25–32.

[3] D. B. Lenat, "Cyc: A large-scale investment in knowledge infrastructure," *Communications of the ACM*, vol. 38, no. 11, pp. 33–38, 1995.

[4] D. B. Lenat and R. V. Guha, "The evolution of cycl, the cyc representation language," *ACM SIGART Bulletin*, vol. 2, no. 3, pp. 84–87, 1991.

[5] A. Nuraliyeva, S. Daukishov, R. Nassyrova, *et al.*, "Comparative analysis of knowledge bases: Conceptnet vs cyc," *Advanced technologies and computer science*, no. 1, pp. 4–14, 2021.

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

[7] Y. Bengio, I. Goodfellow, and A. Courville, *Deep learning*. MIT press Cambridge, MA, USA, 2017, vol. 1.

[8] D. C. Van Essen, "Organization of visual areas in macaque and human cerebral cortex," *The visual neurosciences*, vol. 1, pp. 507–521, 2003.

[9] B. R. Sheth and R. Young, "Two visual pathways in primates based on sampling of space: Exploitation and exploration of visual information," *Frontiers in integrative neuroscience*, vol. 10, p. 37, 2016.

[10] *Fei-fei li: If we want machines to think, we need to teach them to see*, `https://www.wired.com/brandlab/2015/04/fei-fei-li-want-machines-think-need-teach-see/`.

[11] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3128–3137.

[12] X. Liang, L. Lee, and E. P. Xing, "Deep variation-structured reinforcement learning for visual relationship and attribute detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 848–857.

[13] P. Ren *et al.*, "A comprehensive survey of neural architecture search: Challenges and solutions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–34, 2021.

[14] P. Ren *et al.*, "A survey of deep active learning," *ACM computing surveys (CSUR)*, vol. 54, no. 9, pp. 1–40, 2021.

[15] S. Schuster, R. Krishna, A. Chang, L. Fei-Fei, and C. D. Manning, "Generating semantically precise scene graphs from textual descriptions for improved image retrieval," in *Proceedings of the fourth workshop on vision and language*, 2015, pp. 70–80.

[16] S. Amirian, K. Rasheed, T. R. Taha, and H. R. Arabnia, "Automatic image and video caption generation with deep learning: A concise review and algorithmic overlap," *IEEE Access*, vol. 8, pp. 218 386–218 400, 2020.

[17] L. Gao, Z. Guo, H. Zhang, X. Xu, and H. T. Shen, "Video captioning with attention-based lstm and semantic consistency," *IEEE Transactions on Multimedia*, vol. 19, no. 9, pp. 2045–2055, 2017.

[18] L. Zhou, Y. Zhou, J. J. Corso, R. Socher, and C. Xiong, "End-to-end dense video captioning with masked transformer," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8739–8748.

[19] X. Chang, P. Ren, P. Xu, Z. Li, X. Chen, and A. G. Hauptmann, "A comprehensive survey of scene graphs: Generation and application," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[20] Z. Wu, T. Yao, Y. Fu, and Y.-G. Jiang, "Deep learning for video classification and captioning," in *Frontiers of multimedia research*, 2017, pp. 3–29.

[21] M. Z. Hossain, F. Sohel, M. F. Shiratuddin, and H. Laga, "A comprehensive survey of deep learning for image captioning," *ACM Computing Surveys (CsUR)*, vol. 51, no. 6, pp. 1–36, 2019.

[22] M. Stefanini, M. Cornia, L. Baraldi, S. Cascianelli, G. Fiameni, and R. Cucchiara, "From show to tell: A survey on image captioning," *arXiv preprint arXiv:2107.06912*, 2021.

[23] K. R. McKeown, "Discourse strategies for generating natural-language text," *Artificial intelligence*, vol. 27, no. 1, pp. 1–41, 1985.

[24] W. J. Levelt, *Speaking: From intention to articulation*. MIT press, 1993.

[25] Y. Belinkov, A. Poliak, S. M. Shieber, B. Van Durme, and A. M. Rush, "Don't take the premise for granted: Mitigating artifacts in natural language inference," *arXiv preprint arXiv:1907.04380*, 2019.

[26] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning, "A large annotated corpus for learning natural language inference," *arXiv preprint arXiv:1508.05326*, 2015.

[27] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: A method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[28] S. Banerjee and A. Lavie, "Meteor: An automatic metric for mt evaluation with improved correlation with human judgments," in *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, 2005, pp. 65–72.

[29] A. L. Han, D. F. Wong, and L. S. Chao, "Lepor: A robust evaluation metric for machine translation with augmented factors," in *Proceedings of COLING 2012: Posters*, 2012, pp. 441–450.

[30] R. Krishna *et al.*, "Visual genome: Connecting language and vision using crowdsourced dense image annotations," *International journal of computer vision*, vol. 123, no. 1, pp. 32–73, 2017.

[31] J. Johnson *et al.*, "Image retrieval using scene graphs," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3668–3678.

[32] X. Chang, P. Ren, P. Xu, Z. Li, X. Chen, and A. Hauptmann, "Scene graphs: A survey of generations and applications," *arXiv preprint arXiv:2104.01111*, 2021.

[33] H. van den Berg, "First-order logic in knowledge graphs," *Current Issues in Mathematical Linguistics*, vol. 56, pp. 319–328, 1993.

[34] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, "Deep learning for visual understanding: A review," *Neurocomputing*, vol. 187, pp. 27–48, 2016, Recent Developments on Deep Big Vision, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2015.09.116. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231215017634.

[35] L. Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning," *APSIPA transactions on Signal and Information Processing*, vol. 3, 2014.

[36] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[37] C. v. d. Malsburg, "Frank rosenblatt: Principles of neurodynamics: Perceptrons and the theory of brain mechanisms," in *Brain theory*, Springer, 1986, pp. 245–248.

[38] M. L. Minsky and S. A. Papert, *Perceptrons: Expanded edition*, 1988.

[39] *Perceptron: A basic neural network model for deep learning*, https://towardsai.net/p/l/perceptron-a-basic-neural-network-model-for-deep-learning.

[40] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Icml*, 2010.

[41] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.*, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, Citeseer, vol. 30, 2013, p. 3.

[42] L. Noriega, "Multilayer perceptron tutorial," *School of Computing. Staffordshire University*, 2005.

[43] *Introduction to feedforward neural networks*, https://towardsdatascience.com/feed-forward-neural-networks-c503faa46620.

[44] I. Goodfellow, Y. Bengio, and A. Courville, *6.5 back-propagation and other differentiation algorithms. deep learning*, 2016.

[45] Y. Kim, J. Lee, J.-S. Kim, H. Jei, and H. Roh, "Efficient multi-gpu memory management for deep learning acceleration," in *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, IEEE, 2018, pp. 37–43.

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[47] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[48] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization.," *Journal of machine learning research*, vol. 13, no. 2, 2012.

[49] J. Wu, "Introduction to convolutional neural networks," *National Key Lab for Novel Software Technology. Nanjing University. China*, vol. 5, no. 23, p. 495, 2017.

[50] S. Saha, *A comprehensive guide to convolutional neural networks — the eli5 way*, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[51] *Convolutional neural networks (cnns)*, https://anhreynolds.com/blogs/cnn.html.

[52] 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, p. 106, 1962.

[53] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[54] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition (2015). cite," *arXiv preprint arxiv:1512.03385*,

[55] F. Sultana, A. Sufian, and P. Dutta, "Advancements in image classification using convolutional neural network," in *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, IEEE, 2018, pp. 122–129.

[56] ——, "A review of object detection models based on convolutional neural network," *Intelligent Computing: Image Processing Based Applications*, pp. 1–16, 2020.

[57] N. M. Zaitoun and M. J. Aqel, "Survey on image segmentation techniques," *Procedia Computer Science*, vol. 65, pp. 797–806, 2015.

[58] A. Ghosh, A. Sufian, F. Sultana, A. Chakrabarti, and D. De, "Fundamental concepts of convolutional neural network," in *Recent trends and advances in artificial intelligence and Internet of Things*, Springer, 2020, pp. 519–567.

[59] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[60] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[61] J. Carreira and A. Zisserman, "Quo vadis, action recognition? a new model and the kinetics dataset," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017.

[62] *Neural network (2): Rnn and problems of exploding/vanishing gradient*, `https://liyanxu.blog/2018/11/01/rnn-exploding-vanishing-gradient/`.

[63] R. C. Staudemeyer and E. R. Morris, "Understanding lstm–a tutorial into long short-term memory recurrent neural networks," *arXiv preprint arXiv:1909.09586*, 2019.

[64] K. Cho *et al.*, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[65] W. Yin, K. Kann, M. Yu, and H. Schütze, "Comparative study of cnn and rnn for natural language processing," *arXiv preprint arXiv:1702.01923*, 2017.

[66] *Understanding gru networks*, https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be.

[67] P. Wang, D. Dou, F. Wu, N. de Silva, and L. Jin, "Logic rules powered knowledge graph embedding," *arXiv preprint arXiv:1903.03772*, 2019.

[68] D. Fensel *et al.*, "Introduction: What is a knowledge graph?" In *Knowledge Graphs*, Springer, 2020, pp. 1–10.

[69] G. A. Miller, "Wordnet: A lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[70] C. Fellbaum, "Wordnet and wordnets," 2005.

[71] A. Cortés-Calabuig, M. Denecker, O. Arieli, B. Van Nuffelen, and M. Bruynooghe, "On the local closed-world assumption of data-sources," in *Logic Programming and Nonmonotonic Reasoning*, C. Baral, G. Greco, N. Leone, and G. Terracina, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 145–157, ISBN: 978-3-540-31827-9.

[72] L. A. Galárraga, C. Teflioudi, K. Hose, and F. Suchanek, "Amie: Association rule mining under incomplete evidence in ontological knowledge bases," in *Proceedings of the 22nd international conference on World Wide Web*, 2013, pp. 413–422.

[73] X. L. Dong *et al.*, "Knowledge vault: A web-scale approach to probabilistic knowledge fusion," in *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, Evgeniy Gabrilovich Wilko Horn Ni Lao Kevin Murphy Thomas Strohmann Shaohua Sun Wei Zhang Geremy Heitz, 2014, pp. 601–610. [Online]. Available: http://www.cs.cmu.edu/~nlao/publication/2014.kdd.pdf.

[74] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3156–3164.

[75] K. Xu *et al.*, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, PMLR, 2015, pp. 2048–2057.

[76] J. Jin, K. Fu, R. Cui, F. Sha, and C. Zhang, "Aligning where to see and what to tell: Image caption with region-based attention and scene factorization," *arXiv preprint arXiv:1506.06272*, 2015.

[77] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.

[78] S. J. Rennie, E. Marcheret, Y. Mroueh, J. Ross, and V. Goel, "Self-critical sequence training for image captioning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7008–7024.

[79] A. Oluwasammi *et al.*, "Features to text: A comprehensive survey of deep learning on semantic segmentation and image captioning," *Complexity*, vol. 2021, 2021.

[80] A. Torabi, C. Pal, H. Larochelle, and A. Courville, "Using descriptive video services to create a large data source for video annotation research," *arXiv preprint arXiv:1503.01070*, 2015.

[81] A. Rohrbach, M. Rohrbach, N. Tandon, and B. Schiele, "A dataset for movie description," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3202–3212.

[82] L. Zhou, C. Xu, and J. J. Corso, "Towards automatic learning of procedures from web instructional videos," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[83] D. Chen and W. B. Dolan, "Collecting highly parallel data for paraphrase evaluation," in *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*, 2011, pp. 190–200.

[84] J. Xu, T. Mei, T. Yao, and Y. Rui, "Msr-vtt: A large video description dataset for bridging video and language," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 5288–5296.

[85] R. Pasunuru and M. Bansal, "Multi-task video captioning with video and entailment generation," *arXiv preprint arXiv:1704.07489*, 2017.

[86] Z. Shen *et al.*, "Weakly supervised dense video captioning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1916–1924.

[87] S. Islam, A. Dash, A. Seum, A. H. Raj, T. Hossain, and F. M. Shah, "Exploring video captioning techniques: A comprehensive survey on deep learning methods," *SN Computer Science*, vol. 2, no. 2, pp. 1–28, 2021.

[88] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney, and K. Saenko, "Translating videos to natural language using deep recurrent neural networks," *arXiv preprint arXiv:1412.4729*, 2014.

[89] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[90] L. Yao *et al.*, "Describing videos by exploiting temporal structure," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4507–4515.

[91] Z. Guo, L. Gao, J. Song, X. Xu, J. Shao, and H. T. Shen, "Attention-based lstm with semantic consistency for videos captioning," in *Proceedings of the 24th ACM international conference on Multimedia*, 2016, pp. 357–361.

[92] C. C. Shekhar *et al.*, "Domain-specific semantics guided approach to video captioning," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2020, pp. 1587–1596.

[93] K. Hara, H. Kataoka, and Y. Satoh, "Learning spatio-temporal features with 3d residual networks for action recognition," in *Proceedings of the IEEE international conference on computer vision workshops*, 2017, pp. 3154–3160.

[94] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko, "Sequence to sequence-video to text," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4534–4542.

[95] J. Donahue *et al.*, "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 2625–2634.

[96] P. Pan, Z. Xu, Y. Yang, F. Wu, and Y. Zhuang, "Hierarchical recurrent neural encoder for video representation with application to captioning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1029–1038.

[97] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4489–4497.

[98] M. Stefanini, M. Cornia, L. Baraldi, S. Cascianelli, G. Fiameni, and R. Cucchiara, "From show to tell: A survey on deep learning-based image captioning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[99] F. J. Kurfess, "Neural networks and structured knowledge: Knowledge representation and reasoning," *Applied Intelligence*, vol. 11, no. 1, pp. 5–13, 1999.

[100] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[101] H. Poon and P. Domingos, "Sum-product networks: A new deep architecture," in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, IEEE, 2011, pp. 689–690.

[102] A. Santoro *et al.*, "A simple neural network module for relational reasoning," *Advances in neural information processing systems*, vol. 30, 2017.

[103] J. Xu, Z. Zhang, T. Friedman, Y. Liang, and G. Broeck, "A semantic loss function for deep learning with symbolic knowledge," in *International conference on machine learning*, PMLR, 2018, pp. 5502–5511.

[104] P. Wang, Q. Wu, C. Shen, and A. van den Hengel, "The vqa-machine: Learning how to use existing vision algorithms to answer new questions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1173–1182.

[105] S. Antol *et al.*, "Vqa: Visual question answering," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2425–2433.

[106] D. Castelvecchi, "Can we open the black box of ai?" *Nature News*, vol. 538, no. 7623, p. 20, 2016.

[107] D. Lyu, F. Yang, B. Liu, and S. Gustafson, "Sdrl: Interpretable and data-efficient deep reinforcement learning leveraging symbolic planning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 2970–2977.

[108] Z. Ma *et al.*, "Learning symbolic rules for interpretable deep reinforcement learning," *arXiv preprint arXiv:2103.08228*, 2021.

[109] T. Yao, Y. Pan, Y. Li, and T. Mei, "Exploring visual relationship for image captioning," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 684–699.

[110] J. Shi, H. Zhang, and J. Li, "Explainable and explicit visual reasoning over scene graphs," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8376–8384.

[111] P. Xu, X. Chang, L. Guo, P.-Y. Huang, X. Chen, and A. G. Hauptmann, "A survey of scene graph: Generation and application," *EasyChair Preprint*, no. 3385, 2020.

[112] X. Yang, K. Tang, H. Zhang, and J. Cai, "Auto-encoding scene graphs for image captioning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 685–10 694.

[113] D. Teney, L. Liu, and A. van Den Hengel, "Graph-structured representations for visual question answering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1–9.

[114] C. Zhang, W.-L. Chao, and D. Xuan, "An empirical study on leveraging scene graphs for visual question answering," *arXiv preprint arXiv:1907.12133*, 2019.

[115] J. Johnson, A. Gupta, and L. Fei-Fei, "Image generation from scene graphs," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 1219–1228.

[116] S. Geng *et al.*, "Dynamic graph representation learning for video dialog via multi-modal shuffled transformers," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 1415–1423.

[117] M. Chatterjee, J. Le Roux, N. Ahuja, and A. Cherian, "Visual scene graphs for audio source separation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 1204–1213.

[118] B. Pan *et al.*, "Spatio-temporal graph for video captioning with knowledge distillation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 870–10 879.

[119] Y. Teng, L. Wang, Z. Li, and G. Wu, "Target adaptive context aggregation for video scene graph generation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 13 688–13 697.

[120] H. Liu, N. Yan, M. Mortazavi, and B. Bhanu, "Fully convolutional scene graph generation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 11 546–11 556.

[121] D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei, "Scene graph generation by iterative message passing," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 5410–5419.

[122] A. Zareian, S. Karaman, and S.-F. Chang, "Bridging knowledge graphs to generate scene graphs," in *European conference on computer vision*, Springer, 2020, pp. 606–623.

[123] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller, "Introduction to wordnet: An on-line lexical database," *International journal of lexicography*, vol. 3, no. 4, pp. 235–244, 1990.

[124] Y. Pan, T. Yao, H. Li, and T. Mei, "Video captioning with transferred semantic attributes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 6504–6512.

[125] Y. Pan, T. Mei, T. Yao, H. Li, and Y. Rui, "Jointly modeling embedding and translation to bridge video and language," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4594–4602.

[126] X. Zhang, K. Gao, Y. Zhang, D. Zhang, J. Li, and Q. Tian, "Task-driven dynamic fusion: Reducing ambiguity in video description," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3713–3721.

[127] D. M. Tax, M. Van Breukelen, R. P. Duin, and J. Kittler, "Combining multiple classifiers by averaging or by multiplying?" *Pattern recognition*, vol. 33, no. 9, pp. 1475–1485, 2000.

[128] R. Swinburne, "Bayes' theorem," *Revue Philosophique de la France Et de l*, vol. 194, no. 2, 2004.

[129] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International conference on machine learning*, PMLR, 2014, pp. 1188–1196.

[130] S. K. Kauwe, J. Graser, R. Murdock, and T. D. Sparks, "Can machine learning find extraordinary materials?" *Computational Materials Science*, vol. 174, p. 109 498, 2020.

[131] C. Esposito, G. A. Landrum, N. Schneider, N. Stiefl, and S. Riniker, "Ghost: Adjusting the decision threshold to handle imbalanced data in machine learning," *Journal of Chemical Information and Modeling*, vol. 61, no. 6, pp. 2623–2640, 2021.

[132] B. Zhou, Y. Tian, S. Sukhbaatar, A. Szlam, and R. Fergus, "Simple baseline for visual question answering," *arXiv preprint arXiv:1512.02167*, 2015.

[133] Q. Wu, D. Teney, P. Wang, C. Shen, A. Dick, and A. Van Den Hengel, "Visual question answering: A survey of methods and datasets," *Computer Vision and Image Understanding*, vol. 163, pp. 21–40, 2017.

[134] Y. Li, W. Ouyang, B. Zhou, K. Wang, and X. Wang, "Scene graph generation from objects, phrases and region captions," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017.

[135] X. Chen and C. L. Zitnick, "Learning a recurrent visual representation for image caption generation," *arXiv preprint arXiv:1411.5654*, 2014.

[136] C. Feichtenhofer, H. Fan, J. Malik, and K. He, "Slowfast networks for video recognition," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 6202–6211.

[137] A. Sadhu, T. Gupta, M. Yatskar, R. Nevatia, and A. Kembhavi, "Visual semantic role labeling for video understanding," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 5589–5600.

[138] S. Brahmbhatt, *Practical OpenCV*. Apress, 2013.

[139] S. Guadarrama *et al.*, "Youtube2text: Recognizing and describing arbitrary activities using semantic hierarchies and zero-shot recognition," in *Proceedings of the IEEE international conference on computer vision*, 2013, pp. 2712–2719.

[140] T.-Y. Lin *et al.*, "Microsoft coco: Common objects in context," in *European conference on computer vision*, Springer, 2014, pp. 740–755.

[141] B. Thomee *et al.*, "Yfcc100m: The new data in multimedia research," *Communications of the ACM*, vol. 59, no. 2, pp. 64–73, 2016.

[142] P. Qi, T. Dozat, Y. Zhang, and C. D. Manning, "Universal dependency parsing from scratch," *arXiv preprint arXiv:1901.10457*, 2019.

[143] X. Dong *et al.*, "Knowledge vault: A web-scale approach to probabilistic knowledge fusion," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 601–610.

[144] J. Xu, T. Yao, Y. Zhang, and T. Mei, "Learning multimodal attention lstm networks for video captioning," in *Proceedings of the 25th ACM international conference on Multimedia*, 2017, pp. 537–545.

[145] Y. Chen, S. Wang, W. Zhang, and Q. Huang, "Less is more: Picking informative frames for video captioning," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 358–373.

[146] W. Zhang, B. Wang, L. Ma, and W. Liu, "Reconstruct and represent video contents for captioning via reinforcement learning," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 12, pp. 3088–3101, 2019.

[147] S. Olivastri, G. Singh, and F. Cuzzolin, "End-to-end video captioning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019, pp. 1–9.