CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Interactionwise

Semantic Awareness in Visual Relationship Detection

Master's thesis in computer science

Giovanni Pagliarini
Azfar Imtiaz

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Master's thesis 2020

Interactionwise

Semantic Awareness in Visual Relationship Detection

Giovanni Pagliarini

Azfar Imtiaz

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

Interactionwise
Semantic Awareness in Visual Relationship Detection
Giovanni Pagliarini
Azfar Imtiaz

Supervisor: Richard Johansson, CSE
Advisor: Pedro Custodio, Findwise AB
Examiner: Aarne Ranta, CSE

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2020

Abstract

Visual Relationship Detection (VRD) is a relatively young research area, where the goal is to develop prediction models for detecting the relationships between objects depicted in an image. A relationship is modeled as a *subject-predicate-object* triplet, where the predicate (e.g an action, a spatial relation, etc. such as "eat", "chase" or "next to") describes how the subject and the object are interacting in the given image. VRD can be formulated as a classification problem, but suffers from the effects of having a combinatorial output space; some of the major issues to overcome are long-tail class distribution, class overlapping and intra-class variance. Machine learning models have been found effective for the task and, more specifically, many works proved that combining visual, spatial and semantic features from the detected objects is key to achieving good predictions. This work investigates on the use of distributional embeddings, often used to discover/encode semantic information, in order to improve the results of an existing neural network-based architecture for VRD. Some experiments are performed in order to make the model semantic-aware of the classification output domain, namely, predicate classes. Additionally, different word embedding models are trained from scratch to better account for multi-word objects and predicates, and are then fine-tuned on VRD-related text corpora.
We evaluate our methods on two datasets. Ultimately, we show that, for some set of predicate classes, semantic knowledge of the predicates exported from trained-from-scratch distributional embeddings can be leveraged to greatly improve prediction, and it's especially effective for zero-shot learning.

Keywords: Deep Learning, Natural Language Processing, Computer Vision, Visual Relationship Detection, Object Detection.

Acknowledgements

First and foremost, we would like to thank our our supervisor for this project, Richard Johansson. Throughout the course of this project, he has been extremely helpful with things like conducting weekly meetings, holding discussions on matters both theoretical and technical, providing us with alternate ideas wherever applicable, and delivering detailed feedback on multiple drafts of our report in a timely manner.
We would also like to thank our examiner Aarne Ranta, for helping us proceed through the various stages involved in the thesis process without any roadblocks.
A big shout-out to our advisor from Findwise, Pedro Custodio, for keeping a check on our progress, providing immediate help on any obstacles we may be facing, and compiling ideas and resources that would have been necessary for building an image retrieval platform around this project.
We are grateful to Findwise in general for providing us with a nice and productive atmosphere to conduct the work in, as well as all computing resources required such as servers, GPUs, extra storage etc. They ensured that we have uninterrupted access to them, even during the unprecedented times we saw due to COVID-19, which forced us to work remotely for the second half of the project. Some names to mention from Findwise in particular are Niklas Helmertz and Simon Almgren. We feel that most of our colleagues at Findwise during our time working there deserve a mention here, for maintaining an interest in our work, giving ideas and suggestions, and helping us visualize practical applications of this work.

Azfar Imtiaz & Giovanni Pagliarini,
Gothenburg/Ferrara,
June 2020

Contents

1
Introduction

1.1 See and speak

Over the past few decades, great advances in Computer Vision (CV) and Natural Language Processing (NLP) gave rise to new and interesting interesting problems relating to both fields [2]. The task of automatically generating natural-language image descriptions is perhaps the most popular of these problems, and it is a great example of the high potential of models that combine two different kinds of intelligence, namely the ability to *see*, as in process visual data and identify the content in some form, and the ability to *speak* a language and carry out some form of semantic reasoning. But where, in such multi-intelligent models, is the part that displays a certain *understanding* of a picture?
Considering image captioning, for example, it generally involves extracting visual, *non-semantic* information from the image (such as the output of a CNN feature extractor), and using it as the input to a text generation model. Therefore, the first part of the pipeline only knows how to recognise visual patterns, while the second one takes care of translating them into meaningful sentences. While drawing some inspiration from this fashion of combining visual pattern recognition and semantic reasoning, we choose a slightly different approach for this work: we consider existing work in a task which mainly relates to computer vision, and explore different ways of "injecting" semantic knowledge into a model, using a popular technique from the NLP field.

1.2 Study case

Generally, models that understand the content of an image can be used to tag images with useful metadata, and are therefore of particular interest for the "findability" of the image, which is the focus area of the company we are collaborating with for this thesis project. Imagine we have to index images in a database, such that they are easily searchable by queries: the user might want to type queries like "pictures of dogs", "pictures of gardens" etc. Thus, a first type of metadata that we might want to tag images with might consist of the depicted *objects*, for example "horse", "field", "fence". This requires object detection models, which have been largely improved in the last decade [39]. The model potentially allows the user to look for images containing (sets of) objects, e.g "pictures with a *dog* and a *man*".
However, tagging images with objects only may become a bit restrictive: in fact, what if a user wants to look for pictures with more specific content, like "a dog

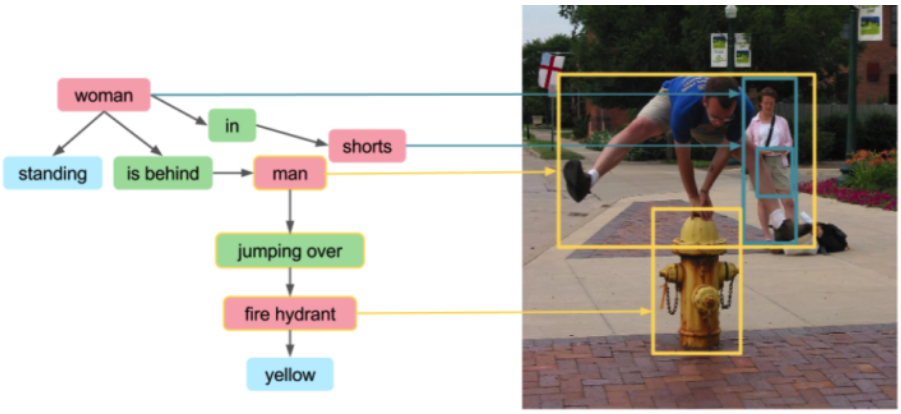


Figure 1.1: Example of a scene graph A scene graph is a labeled graph summarizing the content of an image in terms of objects, object attributes and pairwise interactions between objects. In the example have four objects (red labels), two of which with an attribute (in blue), and three pairs of objects are in a relation/interaction (in red). (Source: [19])

chasing a person"? At best, using object detection only, the search system may preprocess the query, synthetize the desired objects ("dog" and "person"), and base the search on those. This appoach would only return images containing both dogs and men, whereas the interaction between them ("chasing") would be ignored.
At first glance, the difference between an object detector and a detector of objects' *interactions* might not seem substantial; but note that, whereas the first one can only describe the mere *presence* of things, interactions potentially provide much more information about the *meaning* of the overall scene. Therefore, in order to incorporate a little more semantic information to the scene, one can think of describing an image by listing both the objects depicted and the existing interactions between object pairs.

1.3 Visual Relationship Detection (VRD)

The visual content of an image can be summarized in a structured way by means of a "scene graph" (see an example in Fig. 1.1), namely a graph describing the depicted scene in terms of objects, their attributes and their pairwise relationships. In computer vision, the task of predicting an image's scene graph is often referred to *Scene Graph Generation* (SGG). For our purposes, we disregard object attributes and only focus on the recognition of objects and interactions, a task that goes under the name of *Visual Relationship Detection* (VRD) [31].
Understanding object interactions in an image is a crucial step for the understanding of the image, and it's easy to count many salient tasks that can be achieved or improved with these kind of models: image captioning, image-based recommendations, image indexing and retrieval, improved experience for virtual assistants and accessibility for visually impaired users, etc. Recent years have seen a rising interest in these fields, especially using machine learning techniques.
Relationships are modeled as triplets involving two objects and a predicate (consist-

ing of a verb, a preposition, comparative adjective etc.), and VRD models usually build upon an object detection step and focus on identifying, for each pair of objects, the predicates that best describes the relation between them, if any.

1.4 Importance of semantics

Although close to object detection, VRD is heavily hindered by a certain number of problems, which are discussed in greater detail the next chapter. Recent works have shown that VRD models exhibit better performance when, along with the appearance of the detected objects (i.e the visual features), they account for different kinds of information that one can derive from the detected objects. For example, an object detection model doesn't simply localize an object, but also classifies it, and the predicted object label provides useful information for determining which relationships the object might be in. For example, if you know that a certain detected object is a "dog" and a different one is a "person", then you are more likely to be give accurate guesses about what the interaction between the two is.
Class labels are an example of useful additional information that can aid the task of relationship detection, but note that they carry no notion of the objects' *semantics*: in fact, at least in a typical classification setting, a predicted label is no more than one numeric value instead of another. It stands to reason, however, that predicate prediction could benefit from knowing how different object categories relate in terms of their meaning. For example, objects may include "dog", "person" and "child", and a model which knows that a child is more semantically similar to a person than it is to a dog is more likely to give meaningful predictions.
In order to incorporate semantics, we can revisit NLP, given that it provides a method for discovering and encoding semantic information: word embeddings. Using the semantic word embeddings of the detected objects has already been found effective in improving predicate prediction [20, 23], and in this work, we investigate further on the use of embeddings for the predicate labels.
Many works, for example, approach the predicate prediction step as a classification problem, but disregard how different predicates relate with each other, treating each predicate category as an entity completely independent from the others. However, this is hardly the case, and the fact that some predicates may be semantically or visually correlated may be leveraged to improve the quality of the detections. Indeed, here we experiment with different ways of making VRD models aware of the semantics of the concerned classes, and we find room for improvement in the notion that embeddings can be used to also teach models about predicate semantics, alongside object semantics.
In the following chapter we explain the major challenges in VRD and review the tools that are commonly used to tackle the problem; in the "Methods" chapter we present our contributions, together with the implementation tools we used; follows a chapter with a discussion on the results; finally, the last chapter sums up the findings and suggests some ideas for further development of this work.

2
Background

This chapter explains visual relationship detection in detail and discusses the existing approaches to tackle the problem. Most of the work in the field relies on a set of supervised learning techniques that are introduced in detail the first section.

2.1 Underlying theory

In statistics, classification is the task of predicting which of a set of categories (or classes) a certain unseen observation belongs to, given a set of observations labeled with the correct categories. The task is quite relevant for our purposes, in that, as we'll see further on, VRD naturally lends itself to be framed as a classification problem.
Even more relevant to this study is a widespread approach for tackling classification problems, based on a popular technique used in machine learning: artificial neural networks. Artificial neural networks (or, more simply, Neural Networks, NNs), are a well known computational model, at first inspired by a simplistic modeling of neurons in animal brains. In recent years they gained momentum in a large number of tasks, not least of which are classification and object detection.
The following sections give an overview of these concepts.

2.1.1 Artificial neural networks

A neural network is a graph-like structure where the nodes, called *neurons*, represent basic units of computation, and directed edges define how the information flows from one neuron to the others. Generally, a neuron has a certain number of incoming edges, and its job is to apply a simple operation to these, and to pass the output of the operation to the neighboring neurons through its outgoing edges.

2.1.1.1 Feed-forward NNs

Neurons in a network are generally organized into a stack of n layers, with edges only running between consecutive layers and in the same direction. This approach gives a defined order to the computation flow: the output of a neuron layer is input to the next layer. In this context, each layer represents an independent function, and the computation workflow can be explained as follows: the input to the network, represented as a vector $\mathbf{x}$, is fed to the first layer (or "input layer"), and the information is sequentially processed and fed forward by each of the inner layers, until the last layer ultimately holds the model's output. This type of network is

called *feed-forward network* (see Fig. 2.1), and can be seen as the application of a sequence of functions:

$$f(\mathbf{x}) = f_n(\dots f_2(f_1(\mathbf{x}))\dots) = (f_n \circ \dots \circ f_2 \circ f_1)(\mathbf{x})$$

Locally, every layer is defined by a set of parameters that determines how the layer's output is computed, given the input. In the simplest case, every neuron (and thus the layer itself) simply computes an affine transformation of its inputs; in this case, the output of a layer is defined by a matrix of *weights* and a vector of *biases*:

$$f_i(\mathbf{x}) = A_i\mathbf{x} + b_i$$

Note, however, that neural networks in this form are not very "expressive": in fact, passing the information through a sequence of layers in such a linear, feed-forward fashion is equivalent to applying a unique affine transformation:

$$f(\mathbf{x}) = A_n[\dots A_2[A_1\mathbf{x} + b_1] + b_2]\dots + b_n = \tilde{A}\mathbf{x} + \tilde{b}$$

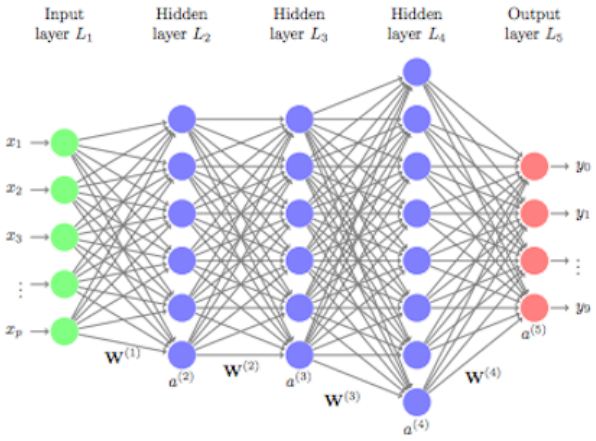


Figure 2.1: Example of a Feed-Forward Neural Network Neurons are organized into layers, and the values at each layer are simply computed from the values of the previous one. The first layer represents the input side of the model, and therefore has no incoming edges; similarly, the output layer, representing the network's output side, has no outgoing edge. (Source: [1])

2.1.1.2 Generalization capability

NNs become powerful with the introduction of *non-linear* computations; specifically, when a non-linear function is applied to the output of each layer:

$$f_i(\mathbf{x}) = \sigma_i(A_i\mathbf{x} + b_i)$$

For each layer i, σ_i is called the layer's *activation function*. This additional detail gives NNs their primary theoretical, important feature: an extraordinary generalization ability. It is in fact proved that any function of any number of variables and

output domain size can be approximated with a "big-enough" neural network that makes use of simple, but *non-linear* activation functions [13].
Although non-linear, activation functions are not necessarily complex; in fact a general requirement is for them to be easily computable. Simple activations (ignoring the trivial identity function) only perform some form of thresholding; examples are the *step* and the *sign* functions, which output values in $\{0, 1\}$ and $\{-1, 0, 1\}$, according to the sign of the neuron's output. The Rectified Linear Unit (ReLU) is another simple one; it almost looks linear but it unlocks more expressiveness, and it's generally the default choice for inner layers. Sigmoidal functions (e.g *logistic*, *hyperbolic tangent*) are more sophisticated functions used map the real axis to normalized ranges (e.g $[0, 1]$). See some common activation functions in Fig. 2.2.

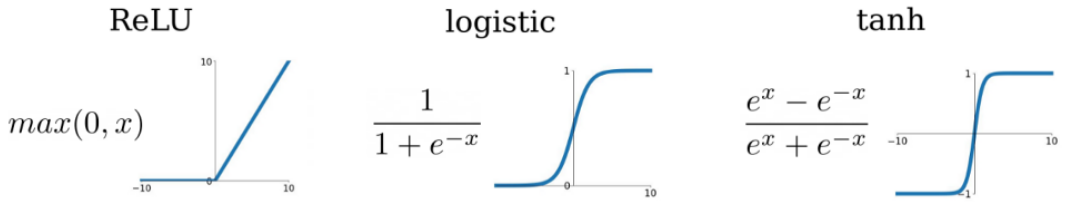


Figure 2.2: Some popular activation functions Rectified Linear Unit (ReLU), logistic and hyperbolic tangent.

2.1.1.3 Training

The possibility of generalizing any function, alone, doesn't straightforwardly make NNs useful. In fact, while it's true that for any given task there exists a network with a correct set of parameters that solves the problem, the existence of an appropriate neural network brings up a hard problem: computing the correct set of parameters defining such a network. Unfortunately, a parameter combination lies in a high-dimensional parameter space (e.g million, billion dimensions), so the problem is not easily solvable (in fact, it's NP-hard [3]).
In supervised learning, the method generally used for deriving the appropriate network for a specific job is called *training*, and comes from the study area of machine learning. The process requires a dataset of "training samples", namely set of pairs $(\mathbf{x}, \mathbf{y})$, where $\mathbf{x}$ is the input to the model, and $\mathbf{y}$ is the desired output; the problem is then approached as an optimization one: find the combination of parameters that maximizes the performance of the network on the available training dataset.
The first step is to decide the "shape" of the chosen model; for neural networks: number of layers, number of neurons per layer, activation functions, etc. The parameters are then initialized according to some fashion, e.g randomly, and an optimization procedure (or *optimizer*) is iteratively applied. The procedure updates the parameters in order to minimize an objective function (or *loss function*), that generally measures some kind of distance between $f(\mathbf{x})$ and $\mathbf{y}$
Note however that training is nowhere near returning the optimal set of parameters. In fact, it tackles a problem that is combinatorial by taking greedy steps, so it is only expected to return local optima. Nonetheless, this tool proved multiple times to yield extremely good results for many different task: in fact, recent decades

have seen an explosion in the applications of machine learning, which is essentially built upon the training procedure. One can also imagine that a globally-optimal answer might not be desirable, as it could lead to the network not being able to correctly generalize the "learned notions" (a potential issue known as *overfitting* on the available dataset).

2.1.2 Classification with neural networks

As mentioned, training Neural Networks has been found effective for many different tasks, and classification is a great example. The typical approach when tackling a multi-class classification problem by neural network training involves the following design choices:

- The network's last layer has K neurons, where K is the number of categories;
- The loss function and training process leads the network to output high values for neurons corresponding to the correct categories.

More specifically, a widespread setup uses a *softmax* activation function for the last layer, and a *cross-entropy* loss function. This coupling is sometimes referred as "categorical cross-entropy loss" [11], and its goal is to teach the network to output a likely probability distribution over the K categories.

The softmax function is conceptually equivalent to the application of a sigmoid function, which squashes the values into the interval $[0, 1]$, followed by a normalization step that makes sure that the output vector sums up to 1. Mathematically, the following form is used:

$$\text{softmax}(\mathbf{x})_k = \frac{\exp(x_k)}{\sum_{i=1}^{K} \exp(x_i)}$$

The application of this function to the output layer produces a valid probability distribution over the K categories, while preserving the rank order of the values: $P(k) = \text{softmax}(f(\mathbf{x}))_k$

In a single-label classification setup, the desired categorical distribution output by the network is one where the correct category y has 100% predicted probability, and the other ones have 0% probability:

$$Q(k) = \begin{cases} 1 & \text{if } k = y \\ 0 & \text{otherwise} \end{cases}$$

Given the output of the network, P, and the desired target, Q, the loss function is now supposed to penalize P being too different from Q, ultimately leading the network to output high values for the correct categories. To this end, cross-entropy is borrowed from information theory, where it is used a measure of distance between

two probability distributions, and it is here used as a loss function:

$$\begin{aligned}\text{CrossEntropyLoss}(f(\mathbf{x}), y) &= -\sum_{k=1}^{K} Q(k) \cdot \log P(k) \\ &= -\log(\text{softmax}\,(f(\mathbf{x})_y)) \\ &= -\log \frac{\exp\left[f(\mathbf{x})_y\right]}{\sum_{k=1}^{K} \exp\left[f(\mathbf{x})_k\right]} \\ &= -f(\mathbf{x})_y + \log \sum_{k=}^{K} \exp\left[f(\mathbf{x})_k\right]\end{aligned}$$

As a side note, for a model that combines softmax with cross-entropy, the minimization of the cross-entropy of P and Q is mathematically equivalent to maximizing the likelihood of the correct labels for the given input.

2.1.2.1 Multi-label classification

A generalization of the multi-class classification problem *multi-label* classification, namely when an observation can belong to more than one category. Arithmetically, the softmax cross-entropy loss can be generalized for multi-label classification [24], but this is generally not the preferred way of going. The reason behind this has to do with the fact that softmax silently assumes that categories are mutually exclusive (i.e the categorization into a given category excludes the categorization into the others), whereas in fact, multi-label classification removes this constraint.
To show this limitation, consider an input vector $\mathbf{x}$ for which two categories are the correct ones: $y = \{i, j\}$ (note how y is now a *set* of ground-truth labels, instead of a single ground-truth label); since the normalization step in the softmax forces the output vector to sum up to 1, the scores for the two labels $f(\mathbf{x})_i$ and $f(\mathbf{x})_j$ will be competing with each other in order to reach high values and, in the best scenario, the output distribution will give a probability of $\frac{1}{2}$ to both labels. One can imagine how, with a possibly higher number of co-occurring labels, this "competition effect" might not be desirable.
A common alternative approach involves considering the output values as the result of K *independent* binary detectors, rather than a unique estimator of a categorical distribution [35]. Similarly to the single-label setup, the desired behavior is for the network to output values close to 1 for the correct categories, and close to 0 for the other categories, so the target output is:

$$Q(k) = \begin{cases} 1 & \text{if } k \in y \\ 0 & \text{otherwise} \end{cases}$$

The softmax step is replaced with a logistic function, which simply maps the values to $[0, 1]$, without normalizing the vector: $P(k) = \text{logistic}(f(\mathbf{x})_k)$. Each score, then, represents an independent estimate of the probability that an observation belongs to the respective category, and training each of the K detectors is considered as

an independent binary classification problem ("belongs/doesn't belong to the category"). To this end, a (binary) cross-entropy loss can be computed for each of the the K neurons, and then averaged:

$$\begin{aligned}\text{BinaryCELoss}(f(\mathbf{x}), y) &= -\frac{1}{K}\sum_{k=1}^{K}\left[Q(k)\cdot\log P(k) + (1-Q(k))\cdot\log(1-P(k))\right] \\ &= -\frac{1}{K}\left[\sum_{k\in y}\log P(k) + \sum_{k\notin y}\log(1-P(k))\right]\end{aligned}$$

Note that this approach makes the assumption that different categories are not correlated with each other, so that in the final layer a detector can predict a score for the respective label independently of the predicted scores for other labels.

2.1.2.2 Convolutional Neural Networks (CNNs)

In recent years, a specific type of feed-forward neural networks, called Convolutional Neural Networks (or CNNs) have shown great performance at *pattern recognition* in data that has a defined sequence or grid-like topology. These have been used for various fields of computer vision and natural language processing, such as image/video classification, multi-media compression, object detection, sentiment analysis, topic categorization [16, 37].
The core feature of CNNs is represented by the presence of a special type of layer, called *convolutional* layer; usually, a CNN is a *deep* network (namely a network with many layers) consisting of repeated blocks of *convolutional*, *pooling* and linear layers [8].

Convolutional layer The reader will recall how a layer is essentially an operation being applied to an input vector, and while affine/linear layers (also called *fully-connected*) are widely used, one can also think of deploying different operations. Specifically, convolutional layers build upon the mathematical operation of *convolution*, that allows a layer to "seek" for patterns in an n-dimensional input.
A convolutional layer consists of a set of *filters*, namely small matrices that are convolved (i.e "slid") across the input, returning a response map (or "feature map"). Each filter is responsible for the detection of a pattern; in the case of image analysis, for example, filters can detect edges, circles, squares, etc, but the learning process can lead a filter to detect more complex and domain-specific patterns.
Note that, unlike fully-connected layers, where each neuron computes a value as a function of the entire previous layer, a neuron in a convolutional layer only processes a fixed-size subarea of the previous layer (known as its "receptive field"). A beneficial side-effect of this fact is a drastically lower number of layer parameters to be learnt.

Pooling Convolutional layers are often coupled with *pooling* operations, introducing some form of data aggregation (e.g max, min, average). The objective of pooling is to down-sample an input representation, reducing its dimensionality and potentially enabling more abstract representations.

Max pooling is perhaps the most popular choice: when convolutional layers are interweaved with max pooling layers, the output identifies whether a certain pattern was found in any region of the previous layer. Max pooling layers act as a *zoom out* operation, summarizing data regions and overall makes the network invariant to minor transformations of the input [17].

2.1.2.3 Object detection with R-CNNs

As mentioned, CNNs have been found useful for computer vision tasks, due to their ability to extrapolate and synthesize useful information from visual data. Nowadays, there are a certain number of known deep CNN architectures that can be used as "feature extractors", which represent the "heavy artillery" with which difficult computer vision tasks can be coped with. VGG-nets [33] or ResNet [12] are two well-known feature extractors, for which pre-trained models are also publicly available.
Two tasks of interest where CNNs have been found extremely effective are image classification and object detection. The two problems are similar but, while the first task is plain classification of images, the second task involves both the classification and *localization* of objects appearing in the images, where an image might contain more than one object.
The natural approach to object detection might involve classifying many areas of the image to check whether or not they contain an object, and eventually categorize them: Region-based CNNs (R-CNNs) are a class of architectures that do something along these lines. In these models, a region is substantially a box defined by four coordinates:

$$\begin{bmatrix} x_i^{min} & y_i^{min} & x_i^{max} & y_i^{max} \end{bmatrix}$$

The latest model of the R-CNN family (Faster R-CNN [29]) generates a fixed number of candidate regions, computes the feature vector for each one, and uses the output to train single-label classifiers, that ultimately predicts the presence of objects and their label.
In order to compute the feature vectors of the regions, a unique convolutional feature map is first computed by feeding the entire image to a CNN feature extractor. Then, the coordinates of the box are used to access a specific area of the feature map, and the features are derived by applying a special pooling operation. The operation is called Region-Of-Interest Pooling (ROI Pooling [9]), and it's substantially a max-pooling operation, adapted to non-uniform inputs (since the region proposals have a variable size).

2.1.3 Word embeddings

Any given task in the field of natural language processing involves dealing with some sort of text corpus. As machine learning algorithms in general are incapable of dealing with raw strings, the text must first be converted into some sort of numerical representation. This is where the concept of word embeddings comes in. Simply put, a word embedding is a numerical vector representation of a word in a given corpus. When all the words in a given corpus are represented as vectors, they can

be used by machine learning algorithms for various tasks in NLP such as sentiment analysis, question answering, chatbots etc.
There are various ways of computing or learning numerical vector representations of different words in a given corpus. For instance, given a supervised learning task, the embeddings for words can be learnt as part of training the whole model. Alternatively, there exist specific methods where dense vector representations of words are pre-trained by accounting for their local and/or global statistics in the corpus. The word embeddings learnt from these methods can then be plugged into various NLP applications to incorporate their semantic meanings.
The next two sections explain two popular word vector learning techniques that are used in this work.

2.1.3.1 Word2vec

Word2vec uses a neural network to predict the output word given an input word, and then adjusts the weights based on the loss computed between the predicted output word and the actual output word. To do this, data tuples of (*input word*, *output word*) are generated from the entire corpus, where each word is represented by a one-hot vector. These tuples are created on the basis of *target* and *context* words. Consider the following sentence:

The boy **picked** up a stone.

If we consider the word in bold here, we can see that the words it is surrounded by; that is, the context words; can give us some meaningful information about it. This is the main concept that Word2vec is based upon: If two words appear in similar contexts, they probably have a similar meaning. For a given word, the number of context words for it depends upon the window size we define. If we have a window size of 1 for each side, then only the immediately preceding word (boy) and immediately following word (up) to the target word will be considered as context words. If we have a window size of 2, then it means that the two immediately preceding words (The, boy) and two immediately following words (up, a) to the target word will be considered as context words. Therefore, the number of context words for a given target word will be ($2 \times$ window size). Window size is a hyperparameter for the Word2vec model, and a larger window size usually guarantees better performance of the model, at the expense of computation time.
Given a context word, we try to predict the target word; this approach is known as the Continuous Bag of Words (CBOW) model, and it models frequent words better. Alternatively, given a target word, we can also predict its context words; this approach is known as the Skipgram model, and it models rare words better. In this project, we used the CBOW model with hierarchical softmax, and the subsequent explanation is for this specific architecture.
As mentioned earlier, (*target word*, *context words*) tuples are created from the text data in the corpus. The input context words are fed through an embedding layer to get their vector representations, the dimensionality of which is (vocabulary size $\times$ embedding size), where the embedding size is the dimensionality we choose for our word vectors. The vectors for all the words in our corpus are initialized randomly, and are then adjusted as part of the learning process. Once we have the vectors of our input context words, they are summed up to get a single dense vector representation.

This is then passed through a softmax layer to predict the most likely target word. The predicted target word is compared with the actual target word to compute the loss, which is then backpropogated through the network to update the weights in the embedding layer. This process is repeated for all (*target word*, *context words*) tuples in our dataset for multiple epochs.
The network for the CBOW architecture looks like this:

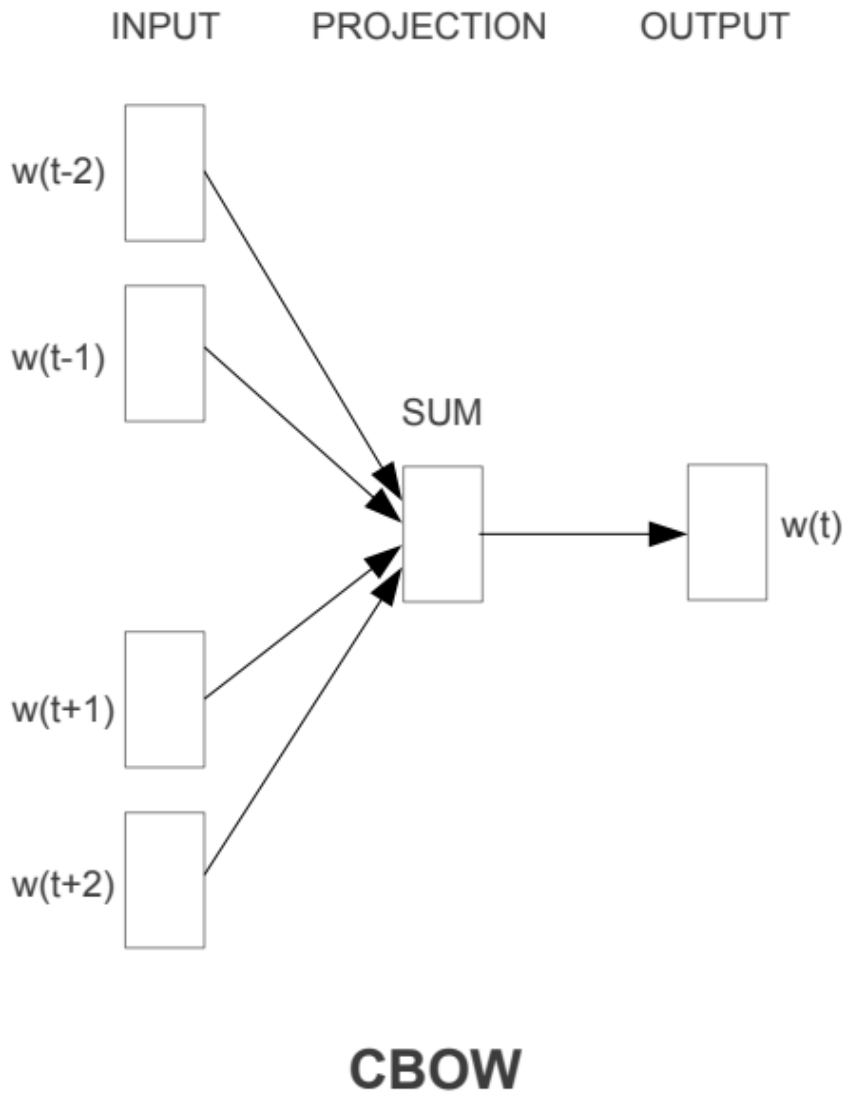


Figure 2.3: Continuous Bag of Words (CBOW) architecture. (Source: [25])

The softmax formulation of the model looks like this:

$$F(w_t, v_c) = P(w_t|v_c) = \frac{\exp(V_{w_t}^T \cdot V'_{v_c})}{\sum_{i=1}^{|V|} \exp(V_{w_{t_i}}^T \cdot V'_{v_c})}$$

where V denotes the embeddings of the target words, and V' denotes the sum of the

embeddings of the context words. The target word embeddings are different from the context word embeddings; the former is denoted by w and the latter is denoted by v.
One key difference to note in the implementation of Word2vec that we used (through the Python library Gensim) is that in place of a regular softmax function, a hierarchical softmax function is used. This is because using regular softmax in a setting where we have thousands or millions of output classes (since the number of classes is equal to vocabulary size, a byproduct of the fact that in order to get good word embeddings, the training is done on huge text corpora), each training sample or batch requires updating the weights for all output classes, and this can be extremely slow. The hierarchical softmax function approximates the softmax function through a Huffman Tree. The Huffman tree in this implementation is a binary tree where the leaves of the tree are words. These words are organized such that infrequent words are present at deeper levels of the tree, and frequent words are found at shallower levels.
The way it works is this: For a given input context vector, it navigates through the tree until it arrives at a leaf node that contains the target word. Starting from the root node, a dot product between the context vector and the vector at the root node is computed, followed by a sigmoid function which gives some value that is mapped either to 0 or 1. Here, 0 means go into the left sub-tree, and 1 means go into the right sub-tree. This process is repeated for each subsequent node until we arrive at the target word leaf node. In this way, considering all possible words is avoided by instead considering a subset of words while trying to compute the probability of the target word.
The basic intuition behind this method is thus: If the model needs to learn a very similar probability distribution for two very similar words like "couch" and "sofa", then it is motivated to learn very similar input vectors for those two words.

2.1.3.2 GloVe

As explained in the previous section, Word2vec captures local statistics by generating (*context words*, *target word*) tuples from the corpus, and then using this data to train a network. In other words, the semantic information contained within the embedding of any word depends upon its surrounding words. For example, consider the following sentence:

He cast his gaze on all his people and his land.

Since Word2vec considers local information only, it cannot identify things such as whether "his" is a special context of the words "gaze", "people" and "land", or whether it is just a stopword.
This is where GloVe [27] comes in. An acronym for Global Vectors, GloVe is an alternate technique for generating word embeddings which considers both local and global statistics of a corpus. For capturing global statistics, GloVe makes use of a co-occurrence matrix. This is a matrix X of dimensionality V x V, where V is the vocabulary size of the corpus. A certain cell, say row i and column j in the co-occurrence matrix denoted by X_{ij} tabulates the number of times word i has occurred with the word j in the corpus.
As an example, consider the following co-occurrence matrix:

	he	cast	his	gaze	on	all	people	and	land
he	0	1	0	0	0	0	0	0	0
cast	1	0	1	0	0	0	0	0	0
his	0	1	0	1	0	1	1	1	1
gaze	0	0	1	0	1	0	0	0	0
on	0	0	0	1	0	1	0	0	0
all	0	0	1	0	1	0	0	0	0
people	0	0	1	0	0	0	0	1	0
and	0	0	1	0	0	0	1	0	0
land	0	0	1	0	0	0	0	0	0

Table 2.1: Co-occurrence matrix for "he cast his gaze on all his people and his land" with window size 1

Let P_{ik} denote the conditional probability of seeing word i and k together. This is computed by dividing the number of times the words i and k have appeared together - that is, X_{ik} - by the number of times that the word i has appeared with any word - that is, X_i.

Probability and Ratio	k = *solid*	k = *gas*	k = *water*	k = *fashion*
$P(k\|ice)$	1.9 x 10^{-4}	6.6 x 10^{-5}	3.0 x 10^{-3}	1.7 x 10^{-5}
$P(k\|steam)$	2.2 x 10^{-5}	7.8 x 10^{-4}	2.2 x 10^{-3}	1.8 x 10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5 x 10^{-2}	1.36	0.96

Table 2.2: Co-occurrence probabilities for target words ice and steam with selected context words (Source: [27])

Keeping this table in mind, we can see that considering the two words *steam* and *ice*, if the third word k (also known as the probe word) is very much related to ice but not so much to steam, then P_{ik}/P_{jk} will be very high. The converse is also true; if the word k is not much related to ice but is very much related to steam, then P_{ik}/P_{jk} will have a very low value. If k is unrelated to both *ice* and *steam*, then P_{ik}/P_{jk} will be quite close to 1. Compared to the raw probabilities, the ratio of probabilities is better able to distinguish between relevant and irrelevant words, and also to differentiate between two relevant words. Due to this, the authors of GloVe [27] use ratios of co-occurrence probabilities. Since this ratio depends upon three words i, j and k, it can be formulated as follows:

$$F(w_i, w_j, v_k) = \frac{P_{ik}}{P_{jk}}$$

where w and v are two separate embedding layers. According to [27], these two layers often perform equivalently; however, having two layers helps the model avoid over-fitting. As for the function F, it is not defined yet, but the possibilities can be narrowed down by enforcing some conditions. For instance, this function should encode the information present in the P_{ik}/P_{jk} ratio, in the word vector space. This

can be done with vector differences, as vector spaces are inherently linear structures. This results in

$$F(w_i - w_j, v_k) = \frac{P_{ik}}{P_{jk}}$$

Since the arguments of the function F are vectors (having dimensionality D x 1) and the right hand side of the equation consists of scalar values, the input of F is finally adjusted by considering the dot product of the arguments:

$$F((w_i - w_j)^T \cdot v_k) = \frac{P_{ik}}{P_{jk}} \tag{2.1}$$

Next, the authors assume that if F has the homomorphism property between the additive and multiplicative groups, the equation then becomes:

$$F((w_i - w_j)^T \cdot v_k) = \frac{F(w_i^T \cdot v_k)}{F(w_j^T \cdot v_k)}$$

This homomorphism ensures that the subtraction $F(A-B)$ can also be represented as a division $F(A)/F(B)$ and guarantee the same result. This is then solved by (2.1), as follows:

$$F(w_i^T \cdot v_k) = P_{ik} = \frac{X_{ik}}{X_i}$$

If it is assumed that $F = \exp$, then the homomorphism property is satisfied.

$$\exp\left(w_i^T \cdot v_k\right) = P_{ik} = \frac{X_{ik}}{X_i}$$

Applying the logarithm, this gives us:

$$w_i^T \cdot v_k = \log(X_{ik}) - log(X_i)$$

Next, the authors note that the term $\log(X_i)$ is independent of the term k, and therefore can be absorbed into a bias b_i for w_i. Furthermore, by adding in a bias b_k for w_k, the equation becomes:

$$w_i^T \cdot v_k + b_i + b_k = \log(X_{ik})$$

One problem that the authors of Pennington et al. [27] had to contend with at this point was the scenario where $X_{ik} = 0$, since $\log(0)$ is undefined. To get around this, the authors proposed a new weighted least squares regression model that addresses these problems. This gives the cost function of the model, expressed as:

$$J = \sum_{i,k=1}^{V} f\Big(X_{ik}\Big)\Big(w_i^T \cdot v_k + b_i + b_k - \log(X_{ik})\Big)^2$$

The squared errors between the log counts and the predicted scores are then minimized.

2.2 Overview of VRD

2.2.1 The problems with VRD

The interest in VRD has been increasing in recent years, but the rise has generally been hindered by major challenges that make this problem much harder than plain object detection. The main problem is given by the fact that the relationships space is much larger than that of objects alone: as previously mentioned, relationships are modeled as a triplet involving two objects and a predicate, so given N object categories and M predicate categories the search space is $\mathcal{O}(N^2M)$.
The combinatorial nature of relationships is a source of many problems. Firstly, unlike for object detection, it's unthinkable to use a standard classification framework, where the network's output is as big in size as the number of available categories. An early work [30] certified how, for a small set of 13 relationship triplets and a balanced dataset, this solution would be effective; However, this approach does not scale well when relationships are generalized to be a combination of two objects and a predicate, since that would require a combinatorial number of output neurons: N^2M.
Secondly, a combinatorial number of categories is particularly troublesome for machine learning techniques, since data availability is crucial to the learning. In fact, even assuming that a model can comprehend a large number of triplet categories, no dataset is big enough to exhaustively cover (as in, give enough training samples for) every possible category.
The need for an appropriate dataset leads to a third problem that has to do with the nature of interactions: dataset imbalance. In general, even considering small subsets of objects and predicates, it is expected that, for example, many object pairs are not going to relate with each other in any (visually) reasonable way. Object categories are in fact often semantically associated, and even semantically associated with certain predicates more than others.
As a result, most of the object-predicate-object combinations are not going to be covered at all, while the covered ones are going to present a "natural" long-tail frequency distribution; namely, exponentially many samples are going to be given for a small subset of triplets, and a lower number of samples is going to be given for most of triplets (e.g see Fig. 3.3).

2.2.2 VRD as a classification task

Different papers describe different architectures for overcoming the problems in VRD; however, the general trend involves using a pipeline with: 1) a region-based *object detection* stage where objects are detected, localized and classified; and 2) a *predicate prediction* model that predicts, for every object pair, the predicate(s) that appropriately describes the relation between pairs of objects.
Let n be the number of detections obtained by the first step, $n(n-1)$ is the number of potential pairs of objects that could be interacting. Note that the number of meaningful pairs is often much more sparse; this is why some architectures introduce an intermediate *pair proposal* module, estimating the relatedness of object pairs and

selecting the meaningful ones for the next stage.
A reasonable critique to using object detection as a first step is made in Sadeghi and Farhadi [30], where the authors argue that the line between the concepts of "object" and "object composite" (i.e two objects interacting in a visual relation) is arbitrary; arguably, most of what are generally considered to be objects (e.g person, dog, table) are in fact composed of simpler objects (e.g hand, torso, eye, dog leg, table leg) and generally, it may be the case that an object composite is detected with more accuracy by a designated detector, rather than using lower-level detectors for the object's "sub-modules".
However, at least three counter arguments defend the use of object detectors: 1) as previously mentioned, training one detector for every single composite is not scalable; 2) generally, pre-trained models detect objects that are the meaningful ones (i.e the ones that one would probably want to see in a scene graph) while disregarding the lower-level objects they are composed of (e.g no use in detecting an eyelash in a full-body photo of a person); and 3) the objects detected by pre-trained detectors arguably show much less variability than relationships among the objects [19]. As a result, relying onto ready-made object detectors may reflect a convenient trade-off between minimizing the intra-variance in the appearance of objects while maximizing their relevance.

2.2.3 Evaluation

Accuracy, precision and recall are the most commonly used metrics for evaluating classification models, and they all essentially measure the percentage of correctly predicted labels. For example, recall is defined as:

$$R = \frac{\text{\# of ground-truth triplets correctly predicted}}{\text{\# of ground-truth triplets}}$$

The current state-of-the-art models for VRD/SGG, however, are not sufficiently performing for these metrics to give appreciable estimates of a model's quality; in fact, due to the combinatorial effects mentioned above, predicting a scene graph in its customary graph form is hard, limitative, and these metrics are currently found to be too pessimistic. Instead, for these tasks "softened" predictions are made and more tolerant metrics are used.
A commonly used framework introduced by Johnson et al. [14] involves, for a given image, sorting all the predicted *subject-predicate-object* triplets by confidence (e.g by the probability/score predicted in predicate prediction) and calculating the recall considering only the top-k predictions:

$$R_k = \frac{\text{\# of ground-truth triplets predicted in the top-}k\text{ confident predictions}}{\text{\# of ground-truth triplets}}$$

This metric is denoted as $R@k$ (read "recall at k"), and two established values of interest are $R@50$ and $R@100$.
It is also common in literature to evaluate different abilities of VRD models: two interesting abilities are *predicate prediction* (PREDCLS) and *relationship detection* (RELCLS) [20, 23, 36, 38]. In predicate prediction, the core classification problem is

isolated by feeding the ground-truth objects (bounding boxes and labels) as object detections; that is, the model has perfect knowledge of the depicted objects, so it only learns to classify their interactions.
This ability alone, however, might not be enough for some real-world scenarios where the objects are not known. In such cases, a model needs to detect the objects first with a dedicated sub-module, which may return inaccurate detections. It is then of interest to measured a model's ability in detecting relationships with no knowledge of the objects, that is, considering the image the sole input to the model. Note that it is possible for a predicate detection model to perform well at predicate detection while, at the same time, to not be able to "endure" imperfect objects detections. Refer to Fig. 2.4 for a visualization of the evaluation pipeline.

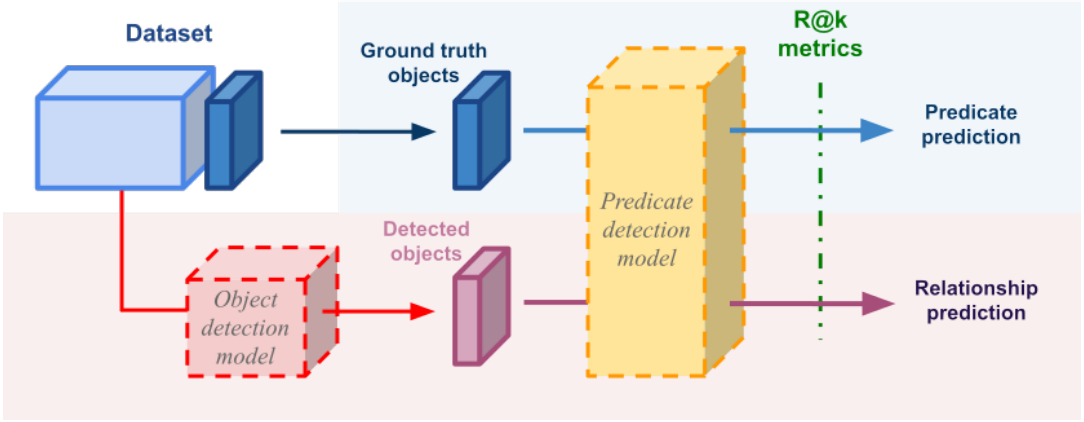


Figure 2.4: Evaluation pipeline Predicate prediction and relationship detection are respectively evaluated by using the ground truth objects and the object detections derived with a pre-trained model. Naturally, due to the error brought by the object detection step, relationship detection is much harder than predicate prediction.

As a final remark on evaluation, many works evaluate their models' ability in detecting unseen relationship triplets ("*zero-shot* learning"); note that this ability is particularly relevant, due to the extreme skew of the relationship space.

2.2.4 Literature overview

As mentioned, VRD is split into two sub-problems: object detection and predicate prediction. The object detection stage is usually achieved through off-the-shelf region-based Convolutional Neural Networks (CNN) (e.g Faster R-CNN [29]). The model outputs a list of detected objects, each with a bounding box, a predicted label (or predicted categorical distribution) and a prediction confidence.
For predicate prediction, different works use different inputs to the model. The primary type of inputs (feature "modalities") that are usually used are the ones derivable from the image. The visual features of two interacting objects are usually derived either by using the computed values from the R-CNN's inner layers, or with a dedicated feature extractor.

However, the visual features alone might not be enough to predict the correct predicate: relationship detection, in fact, does not simply involve some visual pattern recognition, but also requires different kinds of reasoning (such as spatial or semantic). An early work showed how image captioning can be improved with the aid of distributional semantics [7]: the authors introduced a "meaning space of relationships", therefore deriving a model which not only considers the visual appearance of the objects, but also accounts for a semantically-grounded likelihood of relationships. Similarly, many more recent works shows how **multi-modality** is key to predicate detection: different authors factor in additional features, such as semantic information of the objects, and information regarding their spatial arrangement.

2.2.4.1 First use of semantic embeddings

A milestone in VRD is Stanford's paper "Visual Relationship Detection with Language Priors" Lu et al. [23]. After the object detection step (achieved with the original and obsolete R-CNN model [10]), the authors train two different functions in scoring each predicate: one accounting for the visual appeareance, and one accounting for the object's semantics.
The second module involves a function that is learnt by exporting the semantics from word embedding spaces. The model in used is a popular pre-trained Word2vec trained on the Google News dataset [25]). A distance function between relationships is learned, and a ranking loss function used to lead the learning process. Note that the function then scores predicates based on the objects it occurs with in the dataset, but due to the semantics, it can be used to compute a likelihood for any possible relationship; this allows zero-shot learning and alleviates the problems derived from the inevitable imbalance of datasets.
Through ablation studies, the authors highlight how the semantics module was crucial for achieving state-of-the-art results, as it allows to derive a *semantic space for relationships*.

2.2.4.2 Multi-modals extensions

Many works succeeded in improving the baseline set by the Stanford paper by factoring in spatial features from the objects' bounding boxes, or by experimenting with different ways of representing the visual and semantic information.
For instance, Liang et al. [20] experiments with two different ways of incorporating spatial information. The first idea is to represent the relative distance and orientation between the two objects using an 8-dimensional feature vector that captures the relative location of the subject and object boxes, their relative distance, and sizes. The second idea is to create a two-layer binary matrix spatial mask, where the bounding boxes are represented with respect to the image frame; this is then downsampled to a smaller-size square matrix, and compressed into a lower dimensional vector using a (learned) CNN. The authors ultimately show that accounting for the spatial arrangement with any of the two methods improves the model's accuracy in identifying the predicates.
They also try different designs for semantic and visual features. For the semantic features, a pre-trained Word2vec model is used to obtain the semantic embeddings

of the two objects, which are then concatenated into a single vector. For the visual features, features from the union bounding box (with some margin) alone are considered, and experiments show that the model benefits from the additional consideration of features from the individual bounding boxes of the two objects. For feature extraction, a pre-trained VGG16 extractor [32] is used and ROI Pooling [9] is applied to the relevant boxes. The visual, semantic and spatial features for objects are concatenated and fused through two linear layers which finally output non-normalized scores over the predicates. The predicate prediction network is trained end-to-end.
Zhu and Jiang [38] ignores the semantic information and only considers visual and spatial features, in a very similar manner. Instead of using a feature extractor, the visual features are learnt from inner layers of the object detection's model (*conv5_3*), and the multi-modal fusion is done through a fully-connected layer, initialized as another, different layer from the Faster R-CNN. By means of separate linear layers, feature-level prediction scores for object, predicate, and subject are computed, and a unique fully-connected layer is trained to capture the interaction of pairwise labels.
"VRD with Language prior and Softmax" Jung and Park [15] shows how Stanford's paper can be improved with the use of a categorical cross-entropy loss; the language module is framed a similar way, but learnt with cross-entropy loss rather than a custom loss function. The authors compare different ways of combining the information from different modules. The two main models (*V*isual and *L*inguistic) can be trained either separately or together, and can either be augmented using the spatial module or not; when they are trained together, the overall loss function is computed over the element-wise multiplication of the two feature vectors (softmax($V \times L$)). The spatial feature vector is designed considering, among other things, the Intersection over Union (IoU) of the two objects' boxes, and the sizes of the two boxes (relative to the image size). The authors train the predicate prediction model with different loss functions, using different combinations of visual, spatial and linguistic module.
In Dai et al. [5], Johnson et al. [14], Liao et al. [21] a pair proposal module is introduced as an intermediate step between object detection and predicate identification. The idea is that a thresholding method can be applied before predicate prediction, in order to preemptively discard wrong object pairs; as a positive side-effect, this saves some computational cost.

3

Methods

The main focus of this work is in improving predicate detection, with particular focus to teaching the model about the semantics of predicates, as well as improving the models' semantic knowledge about objects. In order to do this, a baseline model based on a pre-existing work is built, and some experiments are performed on it. The experiments follow two main viewpoints: 1) predicate detection can be improved by making the model aware of the semantics of the prediction classes, namely the predicates; 2) word embeddings, the technique in use providing semantic knowledge, can deliver more precise information if pre-emptively specialized for the task at hand. In the following sections, we present the baseline architecture, and the ideas for implementing the two main viewpoints.

3.1 Baseline and implementation details

The baseline architecture is built upon the foundations of Liang et al. [20] (presented in Section 2.2.4.2). The model (see Fig. 3.1) accepts the image and the object detections, and performs predicate prediction for all possible pairs. The predicate prediction module consists of a network that, given an object pair (s, o) ("subject" and "object" of eventual relationships): 1) computes three different kind of features (visual, spatial and semantic features) and concatenates them, 2) fuses them through a fully-connected layer ("fusion layer") into a 256-dimensional fused feature vector, and 3) finally passes the fused features to a fully-connected layer ("classification layer") that predicts probabilities. ReLU activations are used for every layer, except for the last one.

$$f_{\text{pred\_scores}}(\mathbf{I}, (s, o)) = f_{\text{fc\_classification}} \circ f_{\text{fc\_fusion}} \left(\left[\; V_{s,o} \mid S_{s,o} \mid L_{s,o} \; \right] \right)$$

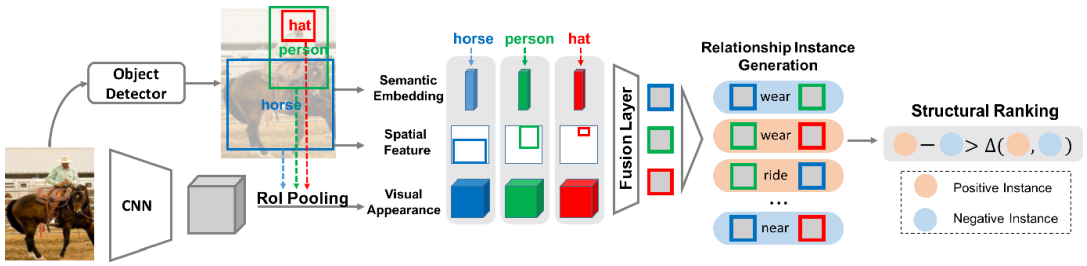


Figure 3.1: Baseline architecture chosen for experimentation. (Image from the original paper [20])

The training framework is similar to the multi-label classification one described in Section 2.1.2.1, with the unique exception that the objective function in use is not a softmax+cross-entropy one, but a margin loss based on Euclidean distance. Rather than penalizing large gap between an output distribution and a target one, this loss penalizes short margin between the output scores for the correct labels and those for incorrect labels. Since it's based on geometric distances, rather than having a rationale in information theory, it also causes the output values of the network to not represent valid probabilities, but generic scores instead.

The model uses a ranking approach described as follows. An image I is fed to a pre-trained Faster R-CNN model, which derives n object detections. In this context, the detection of an object i, yields a bounding box b_i, a label l_i, a prediction confidence c_i. Then, for each $n(n-1)$ candidate pairs (note how it is assumed that objects do not interact with themselves), the features are computed and scores for the M predicates are predicted. Overall, this process yields a score for each of the $n(n-1) \cdot M$ candidate triples; the scores are finally ranked, and the $R@k$ metrics for the image are computed, as described in Section 2.2.3.

The network is trained with the 'Adam' optimizer [18], using a learning rate of 10^{-5}, and an L2 regularization term [4] of 0.0001. The metrics show that the model achieves most of its potential by epoch 4, namely after processing the entire training set for 4 times.

3.1.1 Visual features

Given a pair of objects (s, o), the visual features consist of the concatenation of features extracted from both the individual bounding boxes of subject and object, and from the union bounding box (namely the bounding box of the boxes of subject and object) with a few pixels of margin: $V_{s,o} = \left[\; V^u_{s,o} \;\middle|\; V^{so}_{s,o} \; \right]$. The process for computing them goes as follows.

First, the visual features of the image are extracted, using a sequence of 5 fixed-weights convolutional layers from the backbone of the pre-trained R-CNN model. Then, ROI Pooling is applied to the image's feature map using the boxes of the objects, and the union boxes for each object pair. The results are fed through two learned fully-connected layers, initialized with weights from VGG16, and a randomly initialized fully-connected layer.

The features from the union box finally have this expression:

$$V^u_{s,o} = f_{\text{fc8}} \circ f_{\text{fc7}} \circ f_{\text{fc6}} \circ \text{ROIPool}(f_{\text{VGG-conv}}(\mathbf{I}), \text{union}(b_s, b_o))$$

The features from the individual objects have a similar formulation, but before being used they are again concatenated passed through another fully-connected layer:

$$\begin{aligned} V_i &= f_{\text{fc8}} \circ f_{\text{fc7}} \circ f_{\text{fc6}} \circ \text{ROIPool}(f_{\text{VGG-conv}}(\mathbf{I}), b_i) \\ V^{so}_{s,o} &= f_{\text{fc\_so}} \circ \left[\; V_s \;\middle|\; V_o \; \right] \end{aligned}$$

3.1.2 Spatial features

Let the bounding box of an object i be a vector:

$$b_i = \begin{bmatrix} x_i^{min} & y_i^{min} & x_i^{max} & y_i^{max} \end{bmatrix}$$

The width and height of the box can be computed as:

$$\begin{cases} w_i = x_i^{max} - x_i^{min} \\ h_i = y_i^{max} - y_i^{min} \end{cases}$$

Given the two objects' boxes b_s and b_o, the spatial features are derived from an 8-dimensional vector capturing their relative location, distance, and sizes, fed into a fully-connected layer. Specifically, the feature vector is composed of 4 terms describing the relative position of the boxes, and 4 terms describing their relative size:

$$S_i = f_{\text{fc\_spatial}} \circ \left[\begin{array}{cccc|c} \frac{x_s^{min} - x_o^{min}}{w_o} & \frac{y_s^{min} - y_o^{min}}{h_o} & \frac{x_o^{min} - x_s^{min}}{w_s} & \frac{y_o^{min} - y_s^{min}}{h_s} & \log \begin{bmatrix} \frac{w_s}{w_o} & \frac{h_s}{h_o} & \frac{w_o}{w_s} & \frac{h_o}{h_s} \end{bmatrix} \end{array} \right]$$

3.1.3 Semantic features

The semantic features $L_{s,o}$ from the object pair are computed considering the word embeddings of the labels of subject and object. Specifically, given an embedding model E, the two embedding vectors $E(l_s)$ and $E(l_o)$ are concatenated and fed into a fully-connected layer. Initially, a pre-trained Word2vec model of dimensionality 300 [25] is used. As explained later in this chapter, experiments are carried out exploring alternative choices to concatenation and a pre-trained model.

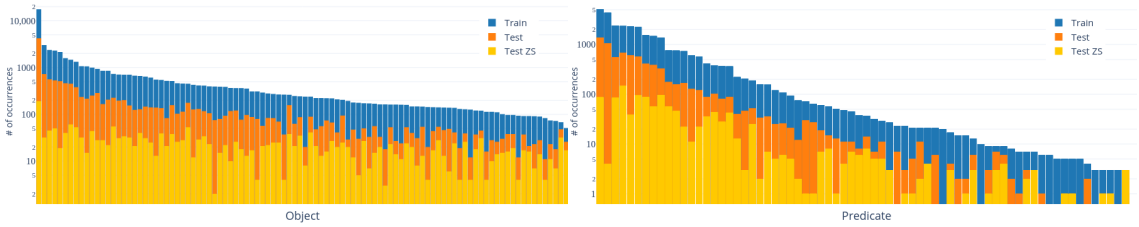
3.1.4 Implementation details

The architecture is implemented in Python 3.6, and makes extensive use of the **PyTorch** library (version 1.4), which is one of the most popular ones for machine learning. Some of the other libraries in use are **Spacy**, **gensim** and **torchvision**. The implementation is publicly available at: `https://github.com/FireSterLine/interactionwise-vrd/`.
The training processes involve a great computational load, but it can be faced through thread-level parallalization, and through vectorization with the help of GPUs. In order to train the networks, a Nvidia GeForce RTX 2080 (provided by Findwise AB) was used, and this allows a training time of 10 to 60 minutes per model (mainly depending on the dataset in use).

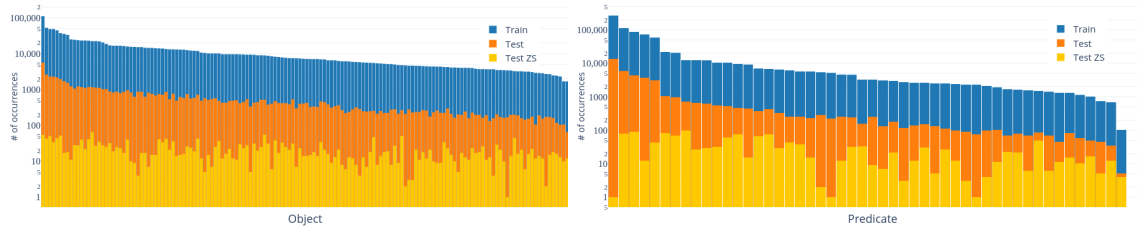
3.2 Datasets

Two popular datasets are used: the "Visual Relationship Dataset" (VRD) and Visual Genome. The first one, introduced by Lu et al. [23], counts $5K$ images with $38K$ annotated relationships and $7K$ different relationship triplets, considering a set of 100 object categories and a set of 70 predicate categories.

Note how the number of triplets covered by the dataset is just around 1% of the number of potential relationships, namely $100^2 \cdot 70 = 700K$ and, unsurprisingly, the dataset displays long tails when considering the frequency distribution of both objects, predicates, and triplets (see Figs. 3.3 and 3.2a). These features refer back to the combinatorial effects discussed in Sec. 2.2.



(a) Visual Relationship Dataset (VRD)



(b) Visual Genome, 150 50 cleaned cut ($VG_{150,50}$)

Figure 3.2: Classwise distributions of the samples by objects (left) and predicates (right) for the two datasets in use The annotated triplets seems to show great imbalances across both object and predicate categories. Note how the VG cut shows less skew than VRD.

Although this dataset is a widespread reference, in more recent years the much bigger Visual Genome dataset [19] gained success, as it did in several other areas of computer vision. Rather than being tailored for a single task, VG was designed to be a "general-purpose representation of the visual world". It contains more than $100K$ images with exhaustive descriptions of the scenes by means of scene graphs, region captions, and other structures.
A note should be made about these datasets. The annotations are manually made by humans, and they supposedly cover *relevant* objects and relationships; however, the definition of "relevant" in this context relies on common sense, and often the annotated graphs can not extensively cover the whole content of an image. This incompleteness of the annotations is also due by the fact that predicate classes of these datasets belong to different types (consider the following predicates from VRD: "next to", "taller than", "play with", "walk past"), and even if multi-predicate relationships are contemplated in both VG and VRD, in reality they are not as frequent as one would expect them to be.

3.2.1 Clean cut of Visual Genome

Visual Genome is widely regarded as much more noisy and less straightforward to operate with; suffice it to say that the object and predicate annotations are not in a

categorical form, but rather strings typed by crowd workers, sometimes containing spelling mistakes. However, it potentially allows for detection on a larger scale, and it still is a useful source of visual data.

A popular choice is in fact to derive cleaned versions of the dataset. For our purposes, we chose to clean the objects and predicates by performing routine text processing tasks; this involves converting to lowercase, removing hyphens and full stops, and removing leading and trailing whitespaces. Additionally, we lemmatized the objects (by treating them as nouns) and predicates (by treating them as verbs) as well.

Additionally, for multi-word objects, there are some cases in the Visual Genome dataset where attributes and objects are joined together. This results in objects like "black shoe" or "yellow shed". To get around this, we created a list of the 19 most common attributes, and then for each multi-word object, we simply check if its first word belongs to this list of common attributes. If so, it is removed, thereby helping us retain just the object itself.

Finally, we create a "split" of the VG dataset by first creating a list of the 150 most common objects, and 50 most common predicates. To generate data according to this split, we then consider only those relationships where the subject and object are found in the 150 most common objects list, and the predicate is found in the 50 most common predicates list. The outcome is a dataset which is slightly less skewed than the VRD dataset (see Fig. 3.2).

Refer to Table 3.1 for a size comparison between the two datasets.

	VRD	**VG**$_{150,50}$	**VG**
# images	5,000	103,077	∼108,000
# unique objects	100	150	∼75,000
# unique predicates	70	50	∼13,000
# annotated rels.	37,993	862,490	∼1,800,000
# unique rels.	7,701	48,663	∼40,000

Table 3.1: Overview of the size of the datasets, namely Visual Relationship Dataset (VRD), Visual Genome (VG) and our cleaned version of Visual Genome. (Data for VG from Krishna et al. [19])

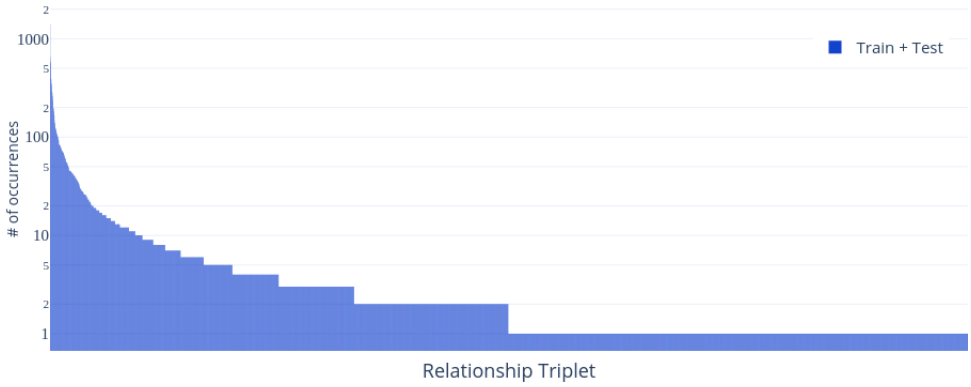


Figure 3.3: Distribution of annotated relationship triplets in the VRD dataset The triplet distribution is extremely skewed; as explained in 2.2, this is caused by the skeweness of the distributions of objects and predicates (see Fig. 3.2a), aggravated by combinatorial effects. Note that this figure only shows the frequenncy for the 7701 unique triplets that the dataset covers, whereas the actual triplet space is much bigger: $100 \times 100 \times 70 = 700K$. The dataset is thus representing a very small portion of the possible triplets.

3.2.2 Predicate subsets

Given that the set of predicates is quite diverse (in that predicates can be of very different types), for this study, two subsets of the available predicates are considered: *spatial* predicates (e.g "next to", "on", etc.) and *activity* predicates (e.g "play with", "eat", "wear", etc.). More specifically, for each of the two datasets in use, namely VRD and $\text{VG}_{150,50}$, the results are measured for:

1. The original dataset, with all the predicate categories it provides;
2. A derived version of the dataset considering only a subset of spatial predicates;
3. A derived version of the dataset considering only a subset of activity predicates;

Ultimately, the derived dataset cuts are denoted as *VRD/all*, *VRD/spatial*, *VRD/activities*, *VG/all*, *VG/spatial*, and *VG/activities*. Table 3.2 shows the number of predicate categories in the datasets; note that VRD and VG also contain other predicates that are not spatial nor denoting activities (e.g "have", "taller than", etc.).

	all	**spatial**	**activities**
VRD	70	21	27
VG	50	19	9

Table 3.2: Number of predicate categories for the datasets in use: VRD/all, VRD/spatial, VRD/activities, VG/all, VG/spatial, VG/activities

3.3 Class semantic awareness

As explained in Section 2.1.2, classification neural networks are trained to output a vector $f(\mathbf{x})$ representing categorical scores, with high values in correspondence of the correct classes. This approach treats each category as an completely and equally different entity from the others. It is often the case, however, that prediction classes are actually entities with meaning, and that their meaning induces some interrelation between them.
For example, the VRD dataset contains the predicates "above" and "under", and clearly these two are related to each other in a very specific way. A first trivial observation is that one of the two excludes the other, hence a model should never output high values for both of them. A second more specific observation is that they actually denote two opposite concepts; from a certain perspective, one might perceive them as two opposite "poles" of a range describing the "verticalness" of the spatial arrangement of the two objects.
In a similar way, certain predicates nearly mean the same thing, despite being considered as two completely different categories, e.g "below" and "under", "near" and "next to": this feature goes under the name of class overlapping, and leads to a series of subtle problems [15]. For pairs of similar predicates such as the above-mentioned ones, it is expected that available samples for one of the predicates can also be safely classified with the other predicate. Additionally, the individual training samples for the two predicates could be used to teach the model something about *both* classes.
In general, the fact that a certain predicate i holds between an object pair may tell *something* about the score of a different predicate j holding as well for the same pair, and this influence has to do with how the two predicates relate to each other semantically and visually. In this work, we only focus on the semantic relations, and choose text-based distributional embeddings as a method for estimating them.
There are some reasons supporting this choice, but also some counterarguments to it. The known "distributional hypothesis" states that the semantic similarity of two words is correlated with their occurrence in similar contexts, and this makes distributional embeddings a good technique for discovering semantic knowledge. However, this also means that the semantic similarity of two predicates, in these terms, is essentially estimating how much two predicates are *of the same kind*, and this probably not desired. As an example, despite the fact that "above" and "under" represent two opposite concepts, they do belong to the same semantic area of "spatial prepositions", and in fact they are often used in similar contexts (as in, surrounded by similar words); therefore a distributional embedding model would consider these two as words with similar meaning.
In the following sections, we propose three ways of using text-based embeddings to make the model aware of the semantics of the classes, right after giving a visual insight from a geometric perspective.

3.3.1 Geometric perspective

As describred in 2.1.2, standard classification neural network learns intermediate representations of the data, until the very last layer, which finally produces cate-

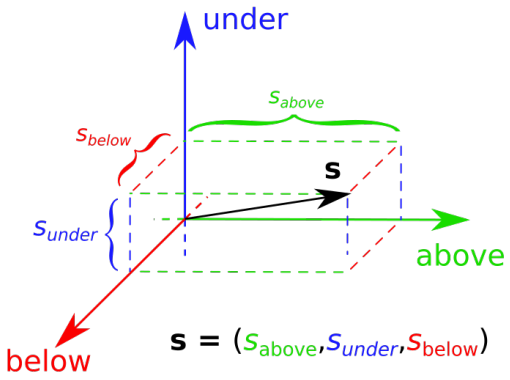


Figure 3.4: Geometric visualization of a categorical score vector The example shows a vector s of three categorical scores, i.e one for each predicate category. The axes are orthogonal, thus each score is an independent variable: one can increase the score for "under" without affecting the scores for "above" or "below".

gorical scores as a vector $f(\mathbf{x})$. From a geometric viewpoint, one can picture $f(\mathbf{x})$ as lying in a K-dimensional space, where each of the axis u_k represents one of the categories: in this space, the expected behavior is for $f(\mathbf{x})$ to be "stretched" along the axes corresponding to the correct prediction classes (see Fig. 3.4).
Note how this orthogonal space sees its axes as linearly independent vectors: in fact, taking one step along one axis (i.e getting closer to representing a specific predicate) causes no change in the values of the coordinates on other axes. This is certainly desired, as it allows for more expressive power for the output vector; however, this viewpoint reveals the naive assumption that predicates are all equally different entities, and suggests that the interrelation of the classes might be factored in by remodeling the axis according to the semantics of what they represent.
For example, considering again the two predicates "below" and "under", a more appropriate representation for the two would see them as two similar vectors: $u_{\text{below}} \simeq u_{\text{under}}$. This would positively correlate the scores for the two predicates, ultimately preventing undesired situations where a high score is predicted for one and a low score for the other.
In a similar way, it might be desirable for the axes representing synonymous predicates to point to opposite directions at least in a subspace or along a certain direction: this would cause the two corresponding scores to be negatively correlated, preventing that a high score is predicted for both. Fig. 3.5 gives an intuitive visualization of these considerations for the set of these three predicates.

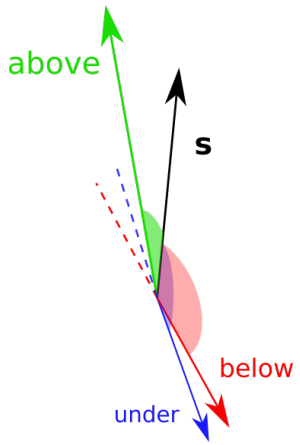


Figure 3.5: Intuitive visualization of a semantic-aware space for a categorical score vector With some predicates being semantically/visually similar or dissimilar to others, it might be desirable for a score vector to correlate the scores of some predicates according to their similarity. In the example, increasing the score for "under" will cause a decrement of the score for "above", and an increment of the score of "below". This might be a more appropriate structured represention.

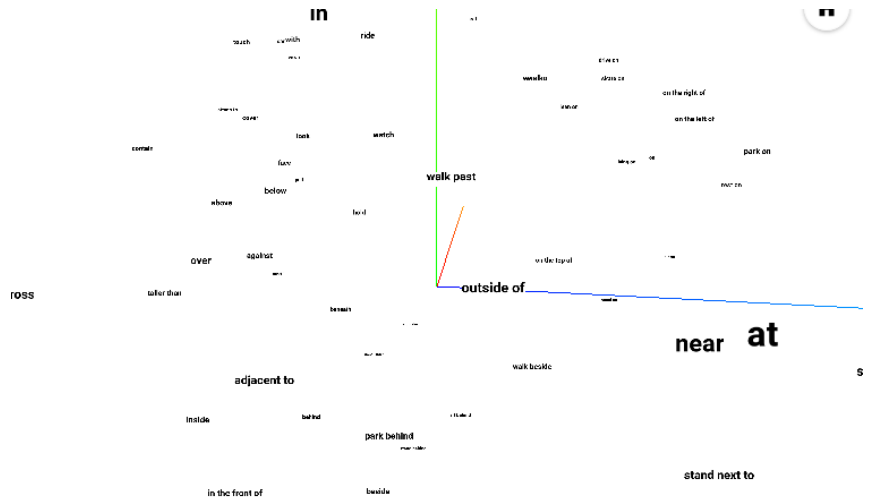


Figure 3.6: Visualization of predicate semantic space The vectors actually lie in a high-dimensional space, but here they are projected into three-dimensions with dimensionality reduction techniques. [Created with TensorFlow Embedding Projector [34]]

3.3.2 Semantic similarity layer (SemSim)

A good embedding model for predicates consists of a set of K vectors, one for each predicate, arranged in a space according to their semantics (e.g see Fig 3.6). Predicates of similar semantics, for example, are pointing to the same spatial direction, whereas predicates belonging to different semantic areas are going to be farther apart.

The first of these two features is particularly interesting for our purposes (see previous section), and sparks the idea of predicting an inner vector representation as a *predicate embedding*, which can be achieved through a layer where an embedding vector $\mathbf{p}$ is predicted, and the K scores are reassigned according to the similarity to this vector.
The fusion and classification layers are therefore reshaped to flow the information through this new semantic space. Specifically, the fusion layer is replaced with a linear layer with no activation, predicting an embedding vector $\mathbf{p}$.

$$\mathbf{p} = f_{\text{fc\_fusion}}\left(\left[\ V_{s,o} \mid S_{s,o} \mid L_{s,o}\ \right]\right)$$

Note that this vector now represents a whole "point of meaning" in the predicates' space, which carries more semantically-relevant information than categorical scores. From this vector, the classification layer then derives categorical scores by measuring the similarity between $\mathbf{p}$ and each of the K predicate class embeddings $\mathbf{E}_k$:

$$f_{\text{fc\_classification}}(\mathbf{p}) = \left[\ \text{sim}(\mathbf{p}, \mathbf{E}_1) \quad \text{sim}(\mathbf{p}, \mathbf{E}_2) \quad \cdots \quad \text{sim}(\mathbf{p}, \mathbf{E}_k)\ \right]$$

In order to compute embedding similarity, a commonly used measure is the cosine of the angle between the two vectors. This metric, referred as "cosine similarity", is essentially a $[-1, 1]$ value measuring how strongly two vectors are pointing in the same direction:

$$\text{sim}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}|| \times ||\mathbf{y}||}$$

The proposed idea is called "semantic similarity layer", or *SemSim* layer.

3.3.3 Semantic rescoring layer

With the previous method, the similarities of the predicate classes are not directly derived from the available embeddings, but rather show up implicitly when rescoring the predicates by how similar they are to the predicted predicate vector $\mathbf{p}$. In fact, when $\mathbf{p}$ is cosine-close to the embedding $\mathbf{E}_y$ for the correct predicate, the correct predicate together with all of the predicates with similar semantics are going to be ranked higher.
A more straightforward way of obtaining semantic information from word embeddings is to directly measure the similarities between the predicate class embeddings. From the set of K embeddings, a "predicate-to-predicate similarity matrix" M can be derived, for example using cosine similarity (see example in Fig. 3.7), and it can be used for reassigning the scores.
Similarly to the previous method, the fusion of the features consists of a linear layer with no activation, but it generates a vector of categorical scores $\mathbf{s}$:

$$\mathbf{s} = f_{\text{fc\_fusion}}\left(\left[\ V_{s,o} \mid S_{s,o} \mid L_{s,o}\ \right]\right)$$

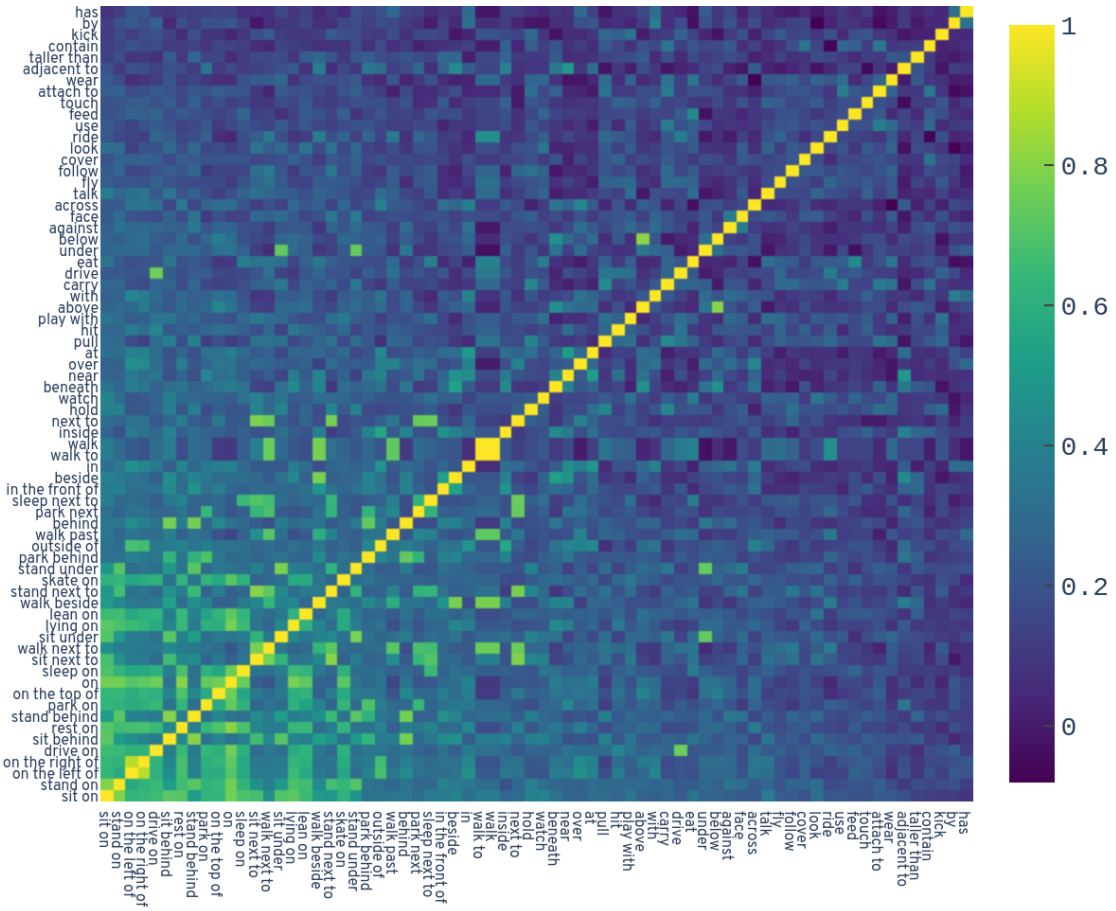


Figure 3.7: Predicate similarities (VRD) according to pre-trained world2vec model Predicate similarities are given by cosine distance between the predicate embedding vectors. To facilitate the interpretation, the rows of matrix are sorted by sum, which causes some hotspots to come close to each other.

The similarity matrix $K \times K$ is then applied to these scores:

$$f_{\text{fc\_classification}}(\mathbf{s}) = \mathbf{M} \cdot \mathbf{s}^T = \begin{bmatrix} \text{sim}(\mathbf{E}_1, \mathbf{E}_1) & \text{sim}(\mathbf{E}_1, \mathbf{E}_2) & \cdots & \text{sim}(\mathbf{E}_1, \mathbf{E}_k) \\ \vdots & & & \vdots \\ \vdots & & \ddots & \vdots \\ \vdots & & & \vdots \\ \text{sim}(\mathbf{E}_k, \mathbf{E}_1) & \text{sim}(\mathbf{E}_k, \mathbf{E}_2) & \cdots & \text{sim}(\mathbf{E}_k, \mathbf{E}_k) \end{bmatrix} \cdot \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ \vdots \\ s_k \end{bmatrix}$$

The rationale behind this is that the scores obtained are somehow "enhanced" by the semantic dependencies that the predicates have with each other. Each score is in fact recomputed as a linear combination of all scores with the the cosine similarities of the predicate class embeddings:

$$s'_k = \sum_{i=1}^{K} \text{sim}(\mathbf{E}_k, \mathbf{E}_i) \cdot s_i$$

Note that this approach is potentially unfair: a high score can be assigned to a predicate just because it is generally more similar to the others, and thus generally higher weights are used in the sum. Therefore, the rows of the matrix are L2-normalized beforehand, which makes the linear combination become a simple weighted average: this prevents a given score from blowing up just because the respective predicate has high similarity to all the other predicates.

3.3.4 Soft embedding rescoring layer

The idea of reassigning scores according to semantic-agnostic scores, together with the idea of using the original embedding space are now mixed into a hybrid method. Similarly to the previous method, the fusion layer produces a semantic-agnostic score vector. A "soft embedding" predicate vector is then computed as the linear combination of the K class embeddings and the semantic-agnostic scores.

$$\mathbf{p} = \sum_{k=1}^{K} \mathbf{E}_k s_k \tag{3.1}$$

Note that the predicate embeddings E_i can be thought of as a set of vectors defining a space of "predicate meaning", and this operation is equivalent to using them as a basis; the predicate vector p is therefore built by considering contributes from each predicate embedding, according to the weights s. Finally, the classification layer recomputes each score as the cosine similarity of the soft embedding with the class embeddings.

$$f_{\text{fc\_classification}}(\mathbf{p}) = \begin{bmatrix} \text{sim}(\mathbf{p}, \mathbf{E}_1) & \text{sim}(\mathbf{p}, \mathbf{E}_2) & \cdots & \text{sim}(\mathbf{p}, \mathbf{E}_k) \end{bmatrix}$$

This method is essentially equivalent to a SemSim layer, with the exception that the predicate embedding is not "freely" predicted by the fusion layer, but rather computed as a linear combination of the class embeddings.

3.4 Specialized embeddings

Given the extensive use of embeddings, it is critical for our methods to have a good embedding representation of objects and predicates. In order to get good vector representations of words, it is generally required for the embeddings to be trained on a huge text corpus. In VRD and in fact in many NLP applications, a popular choice when using word embeddings is to opt for one of the publicly available embedding models, that have been pre-trained on huge general-purpose text corpora, such as the Google News dataset or the Wikipedia dataset. While this is certainly a valid approach, we identify two main problems:

- **Approximate representations of multi-word phrases** A key limitation of of word embedding models is that they deal with text at a word level. Therefore, multi-word phrases such as "trash can" or "traffic light" do not have a dedicated embedding representation. The individual constituents of a multi-word phrase have embedding representations however; so we have embeddings for "trash" and "can", but not "trash can". Therefore the combined meaning of "trash can" as an entity somewhat different from "trash" and "can" is lost. To make up for this, some existing works in VRD compute the mean of the embeddings of all words involved in a multi-word object or predicate, to get an average representation of the entity using its constituents. For example, the embedding of "trash can" would be computed by averaging the embeddings of the words "trash" and "can". While this is quite a widely used approach to get embeddings for multi-word phrases, it is not generally recommended, as averaging word vectors could result in a completely different vector altogether, perhaps then drawing similarity to something it is not similar to at all. Moreover, this approach does not take word order into account, and two completely different multi-word phrases could end up with similar averaged embeddings.
- **General purpose vs. domain-specific embeddings** Many experiments in the field of Natural Language Processing have shown that using domain-specific word embeddings instead of general-purpose word embeddings can often greatly improve results. A primary example of this is the biomedical field; using embeddings which have been trained over biomedical text, containing multiple mentions of various types of diseases and medications, would be a lot more effective in a task revolving around this field as opposed to using embeddings trained over a more generic dataset, such as Wikipedia or Google News. The same concept can be applied to the case of visual relationship detection too. If a word embedding model is trained over a dataset that is more specific to objects and their interactions, the resulting embeddings might be better modeled than the embeddings obtained from models pre-trained on general text corpora.

Effective embedding representations are important for this work as the resulting predicate vectors should be similar or different in the prediction space as per their semantic meaning. For example, embeddings of spatial directions such as "above" and "up" should ideally be closer to each and farther away from something unrelated, such as an activity predicate like "skate". The following sections highlight how the aforementioned issues were tackled.

3.4.1 Multi-word expressions

The Visual Relationship Dataset and the Visual Genome dataset both contain several multi-word objects and predicates. In order to get more precise embeddings for them, we train a word embedding model from scratch on a slightly altered version of the Wikipedia dataset. This alteration was done such that all instances of the multi-word objects and predicates from the the VRD and VG datasets that occur in these Wikipedia articles are joined together into single tokens; so for example, "next to" becomes "next_to", and "traffic light" becomes "traffic_light". This way, they are treated as individual words, and their semantic vectors are computed accordingly, instead of having to average any embeddings.

3.4.2 Embedding models

In this work, two different techniques for training word embedding models are experimented with: namely, Word2vec [25] and GloVe [27]. Both models are trained on the altered version of the Wikipedia dataset (to account for multi-word objects and predicates, as discussed in the previous section) for 5 epochs.
In Word2vec, there are two different algorithms for training the embedding model: Continuous Bag of Words (CBOW) and Skipgram. Additionally, there are two options available for training the model; either by using hierarchical softmax, or negative sampling. In this work, the Word2vec model was trained using CBOW and hierarchical softmax with a window size of 5. This was done using the popular Python library Gensim [28].
Word2vec only incorporates local statistics from the corpus for training the word embeddings. In order to account for global statistics, a GloVe word embedding model was also trained on the same corpus, with a window size of 5. This was done using an open-source Python implementation of GloVe.

3.4.3 Domain-specific word embeddings

Visual relationship detection revolves around objects and how they interact with one another. As such, it cannot really be considered an individual domain (e.g medicine or finance), so it is hard to find a text corpus which focuses solely or heavily on relationships between different objects. Therefore, we looked to utilize existing data in the realm of image captioning or VRD. One such dataset is the image captions from the MS-COCO [22] dataset. MS-COCO is a multi-purpose dataset, and one of the tasks that it has designed for is multi-sentence image captioning. Every image in this dataset is annotated with five captions, each describing the scene a little differently. This dataset was generated by extracting the text of all image captions from the MS-COCO dataset, and applying the same joining of multi-word objects and predicates from VRD and VG into single tokens as done previously with the Wikipedia dataset. This resulted in a total of 203,450 text pre-processed captions.
The models that were chosen to be fine-tuned on the MS-COCO image captions were the Word2vec and GloVe models that were previously trained on the altered version of the Wikipedia dataset. For Word2vec, the training is simply resumed on the MS-COCO dataset for fine-tuning. It was a little unclear how many epochs

the fine-tuning should be done for, since the captions dataset is much smaller than the Wikipedia dataset. For a clearer idea, three different Word2vec models were fine-tuned on the MS-COCO captions: one for 20 epochs, one for 50 epochs, and one for 100 epochs.
The fine-tuning of the GloVe model was a little more complicated, as the Python library we made use of (called glove_python) does not seem to have support for fine-tuning the embeddings. We found an existing work called Mittens [26], which is an extension of GloVe for learning domain specific representations. Their open-source implementation in Python was then made use of for fine-tuning the GloVe embeddings on the MS-COCO captions. As before with Word2vec, three different GloVe models were fine-tuned: one for 20 epochs, one for 50 epochs, and one for 100 epochs.

4
Results

This chapter illustrates the results achieved by factoring in predicate semantics and the specialized embeddings. The main focus is on the core problem of predicate prediction (PREDCLS).

Roadmap

The results shown are mostly given in terms of percentage values of R@50 and R@20, and zero-shot detection is evaluated with the same metrics over a subset of test triplets that are are not seen in the training (refer to Sec. 2.2.3). Results are made robust by repeating each run from 3 to 5 times, and only considering the average performance. In support of the stability of the results, and unless otherwise specified, the reader may consider a standard deviation < 1.5 for the recall values shown.
Given the baseline model described in the previous chapter, different experiments are performed by considering semantic features only (denoted as *sem* in the result tables) and the full combination of spatial, semantic and visual features (*all*). For comparison purposes, results are also given for an *untrained* model (one that outputs random predicate scores), and for models that use no features at all, that is, *blind* models that maximize the objective function by simply learning to output high scores for the most frequently occurring predicates.
Result tables show for the baseline model, and for modified versions of it implementing the ideas proposed in Sec. 3.3:

- **SemSim**: predict a representation of the predicate as a semantic vector in an embedding space;
- **Rescore**: apply a normalized predicate-to-predicate similarity matrix to the network's output, consisting of semantic-agnostic scores;
- **SoftEmb**: hybrid approach that rescores the predicates according to the average of the class embeddings, weighted on semantic-agnostic scores.

In Section 4.1, where the effect of methods for predicate semantics are studied, the embedding model in use is the pre-trained Word2vec on Google News [25] (*gnews*), whereas Section 4.2 shows a comparison of the result using different embeddings. First, the pre-trained model is compared with a Word2vec and a GloVe model, trained to account for multi-word expressions, as described in Section. 3.4.2. Follows a comparison showing the effects of fine-tuning the two models on the MS-COCO image captions for 20, 50, and 100 epochs, with methods described in Sec. 3.4.3.

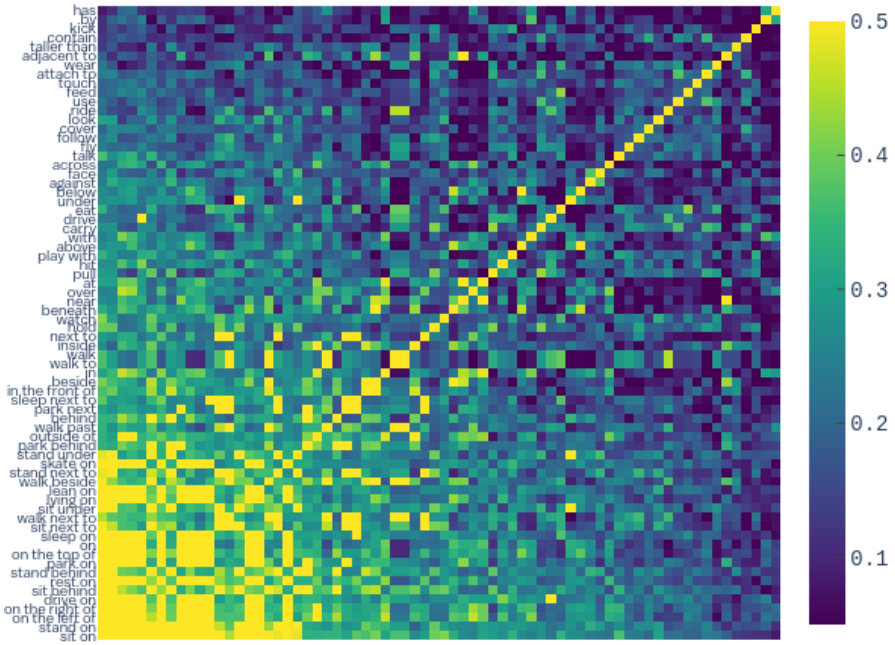


Figure 4.1: VRD/all, predicate similarities according to gnews Note how multi-word predicates are represented by a set of embedding vectors all close to each other (bottom-left corner).

4.1 Predicate semantics methods

This first section of results discusses the three predicate semantics methods, using the general-purpose embedding model pre-trained on Google News. We first show the results for the subset of all predicates, then for the two subsets of spatial and activity predicates.

4.1.1 All predicates

Table 4.1a shows the results on the Visual Relationship Dataset for the baseline model and the three methods in use (SoftEmb, SemSim and Rescore), using three variations of features (blind, sem, all). We observe that across all feature variations, the performances of the methods that we introduce go down, for both zero-shot and non-zero-shot learning. While Rescore still manages to produce scores that are somewhat close to the baseline scores, the approaches which suffer the most are SoftEmb and SemSim. These two generate worse results for all feature variants, and almost seem equivalent, as they yield recall scores that are always close to each other. Results on Visual Genome, shown in Table 4.1b, show a similar trend, with the only difference that using semantic features only leads to slightly better results, as compared to using all feature kinds.

Model type	R@50	R@50 ZS	R@20	R@20 ZS	Avg.
untrained	7.64	11.58	2.78	5.08	6.77
blind (baseline)	61.12	41.97	35.99	15.77	38.71
blind+SoftEmb	39.57	30.80	18.40	15.80	26.14
blind+SemSim	39.48	30.48	18.36	15.85	26.05
blind+Rescore	52.85	33.42	29.95	16.20	33.10
sem (baseline)	89.93	66.41	73.65	37.87	66.96
sem+SoftEmb	69.27	36.47	57.51	22.27	46.38
sem+SemSim	70.11	37.75	58.06	21.19	46.78
sem+Rescore	85.97	62.08	68.98	34.84	62.97
all (baseline)	91.84	75.51	76.46	48.59	73.10
all+SoftEmb	71.64	47.45	61.13	33.22	53.36
all+SemSim	72.34	46.22	60.84	30.51	52.48
all+Rescore	89.52	71.60	72.39	43.43	69.23

(a) PREDCLS, VRD/all

Model type	R@50	R@50 ZS	R@20	R@20 ZS	Avg.
untrained	4.76	12.15	1.41	5.51	5.96
blind (baseline)	63.16	28.75	41.66	16.34	37.47
blind+SoftEmb	37.21	19.70	26.26	11.82	23.75
blind+SemSim	38.93	20.53	26.08	12.01	24.39
blind+Rescore	62.89	28.22	41.80	16.73	37.41
sem (baseline)	89.64	45.94	73.82	22.44	57.96
sem+SoftEmb	76.84	25.54	55.85	12.81	42.76
sem+SemSim	76.65	24.69	55.65	12.61	42.40
sem+Rescore	88.51	41.58	72.72	21.09	55.98
all (baseline)	88.53	43.86	72.70	20.99	56.52

(b) PREDCLS, VG/all

Table 4.1

It is to be noted that that the peculiarity of SoftEmb and SemSim is that they involve a hidden representation being in the form of a predicate embedding. This does not seem to work well, and could be due to the fact that, as previously mentioned, predicates can be of very different types, and applying a unique predicate semantics method might not be desirable. For example, the semantic difference between "above" and "below" perhaps carries a different type of information compared to the difference between two activity predicates, and should therefore be used differently, while the network uses all embedding vectors in the same way.
Figure 4.1 shows a colored heatmap of the cosine similarity of the embeddings, and reveals an imbalance that the word embedding space in use encloses. The predicates in the heatmap are sorted by sum of the cosine similarities with the other predicates, which causes some hotspots to get closer. The bottom-left corner in the image ultimately shows how there exists a cluster of predicates quite close to each other (similarity ≥ 0.5), while other predicates have lower cosine similarities.
This is due to the fact that embeddings for multi-word predicates are computed as the mean of the vectors of the words they contain, therefore end up being cosine-similar to them; the problem will be addressed in Sec. 4.2. In the meantime, we suspect that this imbalance, together the presence of a heterogeneous set of predicates, could prevent the model to make a good use of predicate semantics. In the next sections will isolate predicates belonging to two different domains, namely spatiality and activity.

4.1.2 Spatial predicates

Model type	R@50	R@50 ZS	R@20	R@20 ZS	Avg.
untrained	32.27	33.60	12.14	13.52	22.88
sem (baseline)	92.18	72.89	75.77	42.70	70.88
sem+SoftEmb	84.45	59.36	70.29	40.64	63.68
sem+SemSim	85.20	58.79	70.84	39.62	63.61
sem+Rescore	90.02	71.33	73.83	45.23	70.10
all (baseline)	94.68	83.91	79.74	55.04	78.34
all+SoftEmb	86.24	67.65	73.09	48.88	68.97
all+SemSim	87.90	69.20	73.93	49.80	70.21
all+Rescore	93.64	82.52	78.50	57.00	77.91

Table 4.2: PREDCLS, VRD/spatial

Table 4.2 shows the results of models using semantic features only and all feature kinds, on a subset of VRD which only considers spatial predicates. Since the number of predicates is lower than before (21 predicate categories instead of 70), the task is slightly easier, thus the scores are generally higher. Other than this, patterns similar to the previous case can be seen: the baseline model still outperforms all other models; Rescore is still the method that doesn't heavily impact the performances, while SoftEmb and SemSim still perform the worst, and their scores are still very similar.

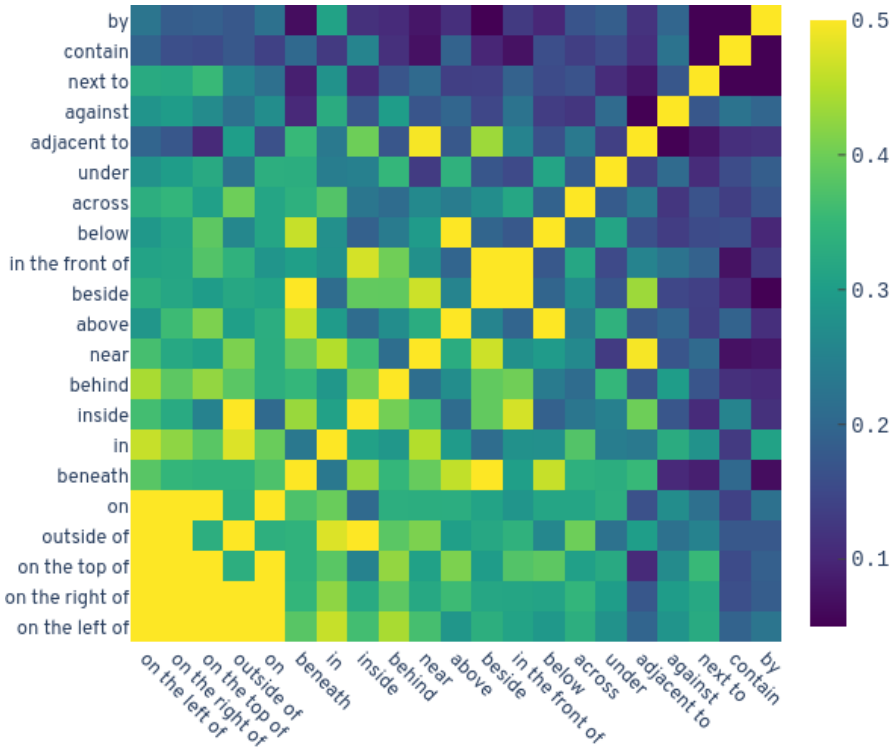


Figure 4.2: VRD/spatial, predicate similarities according to gnews

By inspecting the cosine similarities of the embedding model in use (Fig. 4.2), we once again notice the presence of clusters, but a closer look reveals that the structure of the similarity matrix doesn't match the desired structure explained in 3.3.1. As an example, the semantic similarity between opposite predicates is not at all close to -1, which means their embeddings are far from being opposite vectors: *below*/*above* have a similarity ~ 0.8, *behind*/*in front of* have a similarity ~ 0.4.

In fact, high values for these predicate pairs are justified by one key observation: antonyms often belong to the same semantic area, so we can expect them to be cosine similar, much like synonymous words. This suggests that semantic embeddings might not be the correct estimator of the desired structure, at least not in the case of spatial predicates.

As a final note, it appears to be the case that the semantic embeddings for spatial predicates are quite close to each other (see how Fig. 4.2 is brighter than Fig. 4.1), and this is simply due to the fact that spatial predicates all serve the same purpose of describing a spatial relation. However, this general similarity of the embeddings might get in the way of learning their differences, as it makes it harder for the model to discern between them.

Table 4.3 shows the results on the set of spatial predicates in Visual Genome, and the situation seems to be different: each of our methods seem to be improving the baseline. We can't quite explain this behavior.

Model type	R@50	R@50 ZS	R@20	R@20 ZS	Avg.
sem (baseline)	89.31	58.15	69.61	24.59	60.41
sem+SoftEmb	91.31	46.57	82.98	30.07	62.73
sem+SemSim	91.46	46.16	83.21	31.30	63.03
sem+Rescore	95.32	66.80	86.25	41.29	72.41

Table 4.3: PREDCLS, VG/spatial

4.1.3 Activity predicates

Table 4.4 shows results of different models using a variation of features (all, sem) on a subset of VRD only considering 27 activity predicates. Activity predicates in VRD are the ones with less data samples, suffice it to say that the zero-shot set of test triplets only counts 75 triplets (namely an average of $75/27 \simeq 3$ samples per predicate class). This leads to zero-shot recall values to be less stable.
For activity predicates, it seems like the baseline model here is able to learn non-zero-shot prediction quite accurately (R@20 $> 90\%$), but cannot abstract the learnt dependencies enough to be able to predict unseen triplets: in fact, the R@20 ZS is just 4 recall perc. points better than random guess, independently of whether visual and spatial features are used (see highlighted scores in the table). Zero-shot prediction is also worse compared with models for all predicates (Table 4.1a) and spatial predicates only (Table 4.2). In addition, compared with the previous cases, the use of semantic features alone seems to lead to the best results, and this is probably due to activities being less "visual", and more about what kinds of objects are involved.

Model type	R@50	R@50 ZS	R@20	R@20 ZS	Avg.
untrained	13.86	46.22	9.18	**32.44**	25.43
all (baseline)	95.89	62.67	91.07	**37.78**	71.85
all+SoftEmb	98.11	84.89	96.35	71.56	87.72
all+SemSim	98.49	84.44	97.21	**73.33**	88.37
all+Rescore	98.74	85.33	96.78	70.67	87.88
sem (baseline)	96.38	64.53	92.05	**37.87**	72.71
sem+SoftEmb	98.70	86.67	97.33	73.33	89.01
sem+SemSim	98.36	81.33	97.43	**74.67**	87.95
sem+Rescore	98.25	83.33	97.12	74.00	88.18

Table 4.4: PREDCLS, VRD/activities Note that, due to a low number (75) of zero-shot test triplets, zero-shot prediction recalls are less stable: consider a standard deviation of 4.5. Regardless, the highlighted values show that the baseline model is not accurate at zero-shot prediction, while all three predicate semantics methods greatly improve this ability.

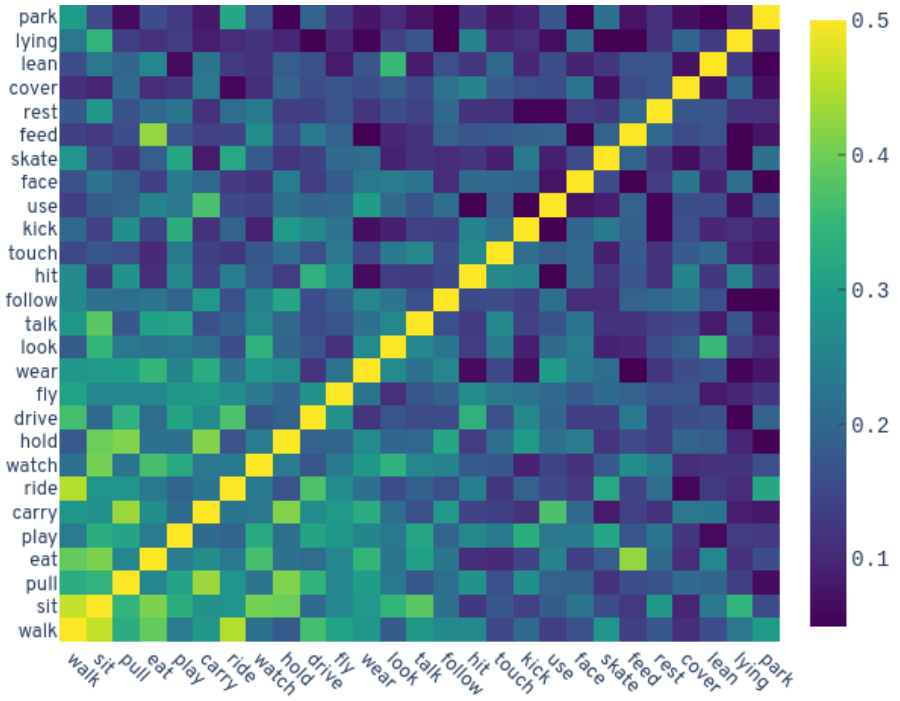


Figure 4.3: VRD/activities, predicate similarities according to gnews

Model type	R@50	R@50 ZS	R@20	R@20 ZS	Avg.
all (baseline)	75.25	18.42	68.15	**12.11**	43.48
sem (baseline)	87.31	31.05	83.69	**23.68**	56.44
sem+SoftEmb	98.39	80.53	97.65	80.00	89.14
sem+SemSim	98.42	77.89	97.71	77.89	87.98
sem+Rescore	99.15	82.63	98.33	80.00	90.03

Table 4.5: PREDCLS, VG/activities Again, when considering only activity predicates, predicate semantics models massively improve zero-shot prediction, as well as non-zero-shot. Note how, differently from previous cases, models using semantic features only perform better than models using of all feature kinds.

What is of more interest, however, is that all the three methods equally and substantially improve the baseline model; this happens for all recall scores, and especially with zero-shot recall scores, where there is much room for improvement. In fact, the results are surprisingly good, as zero-shot scores are normally the hardest to accommodate.
Upon examining the predicate similarity heatmap (Fig. 4.3), we first notice how activity predicates are much less similar to one another as compared to the spatial predicates, and thus more easily differentiable from one another. The similarities are more modest, and this points out that "activities" consists of a broader semantic area compared to the spatial one.
In order to check how the improvements are distributed for each of the predicate

classes, we report the difference in the number of correctly identified triplets: Table. 4.6, for each of the 27 activity predicates, shows the improvements achieved when using SemSim on a model using semantic features. Considering the number of zero-shot triplets for recall @ 20, we witness an average of 24.80 more triplets correctly predicted, which makes up about a third of the zero-shot test triplets (remember the number of zero-shot triplets is 75); this is what causes R@20 ZS to go from 37.78 to 72.27 in the previous table.
The improvement seems to be uniformly distributed across frequent and infrequent predicates, but there are some extreme values: specifically, predicates that are positively affected are “hold”, “carry”, “look (at)”, “watch”, “lying (on)”, “rest (on)”. Note that these 6 happen to be three synonymous predicate pairs, and this suggests that the model was able to pick up some predicate correlations; arguably, the cosine similarity for these pairs may not be substantially high to support the hypothesis: 0.41 for carry/hold, 0.33 for watch/look, 0.11 for lying/rest. Predicates which prediction is instead improved are “ride”, “touch” and “play (with)”.

		Counts		Improvement				
#	**Predicate**	**Test**	**ZS**	**#@50**	**#@50 ZS**	**#@20**	**#@20 ZS**	**AVG**
-	*Total*	1460	75	+27.20	+12.80	+74.63	+24.80	+33.76
1	wear	1056	4	+0.21	+0.20	+2.43	+1.00	+0.96
2	hold	127	11	+5.99	+3.20	+9.80	+6.80	+6.45
3	sit	46	3	+0.60	+0.60	+0.60	-0.80	+0.25
4	ride	34	2	-1.80	-0.80	-0.60	-2.00	-1.30
5	carry	36	7	+3.40	+1.80	+8.80	+4.20	+4.55
6	look	25	5	+7.00	+3.00	+3.20	+3.20	+6.60
7	use	30	1	+2.80	+0.80	+3.80	+0.80	+2.05
8	cover	19	7	+1.20	+1.20	+2.80	+2.80	+2.00
9	touch	15	8	-4.80	-3.60	-2.40	-2.40	-3.30
10	watch	11	4	+6.00	+2.00	+9.80	+3.60	+5.35
11	talk	7	0	+0.00	-	+5.00	-	-
12	fly	3	1	+2.00	+0.00	+2.00	+0.00	+1.00
13	pull	6	0	+0.00	-	+2.00	-	-
14	park	4	4	+1.40	+1.40	+1.40	+1.40	+1.40
15	drive	11	2	+1.40	+0.00	+7.00	+1.00	+2.35
16	walk	1	1	+0.00	+0.00	+1.00	+1.00	+0.50
17	lean	4	3	+1.20	+1.00	+1.20	+0.20	+0.90
18	eat	2	0	+0.00	-	+0.00	-	-
19	face	2	1	+0.20	+0.20	+1.00	+0.00	+0.35
20	lying	6	3	+2.20	+1.60	+3.20	+2.20	+2.30
21	hit	0	0	-	-	-	-	-
22	play	7	3	-3.60	-2.20	+0.60	-0.60	-1.45
23	rest	3	2	+3.00	+2.00	+2.60	+2.00	+2.40
24	follow	3	3	+0.40	+0.40	+0.40	+0.40	+0.40
25	feed	0	0	-	-	-	-	-
26	kick	0	0	-	-	-	-	-
27	skate	2	0	-1.60	-	-1.00	-	-

Table 4.6: PREDCLS, VRD/activities, sem: per-predicate improvements when using SemSim Note that the results here are shown in terms of difference in the number of triplets predicted, instead of recall values. This is because some predicates have a low number of test samples, and the recall value cannot be used to compare the scores of different predicates.

4.2 Specialized embeddings

4.2.1 Embeddings tailored on our vocabulary

This section compares the use of embeddings obtained from the pre-trained embedding model (denoted as *gnews*) with the embeddings obtained from the trained-from-scratch Word2vec and GloVe models (denoted as *w2v* and *glove*, respectively).

4.2.1.1 All predicates

Model type	R@50	R@50 ZS	R@20	R@20 ZS	Avg.
gnews (baseline)	91.84	75.51	76.46	48.59	73.10
gnews+SoftEmb	71.64	47.45	61.13	33.22	53.36
gnews+SemSim	72.34	46.22	60.84	30.51	52.48
gnews+Rescore	89.52	71.60	72.39	43.43	69.23
w2v (baseline)	91.99	75.58	76.56	47.60	72.94
w2v+SoftEmb	80.73	61.55	67.21	41.87	62.84
w2v+SemSim	81.23	62.66	67.33	39.99	62.80
w2v+Rescore	89.38	71.26	72.56	43.76	69.24
glove (baseline)	91.59	76.39	76.12	49.49	73.40
glove+SoftEmb	77.68	57.49	64.90	38.58	59.66
glove+SemSim	78.55	58.04	64.62	35.12	59.08
glove+Rescore	88.67	70.74	71.61	42.69	68.43

Table 4.7: PREDCLS, VRD/all, all features Comparison of different general-purpose embeddings

Table 4.7 shows the recall scores achieved on VRD using the pre-trained gnews embedding model, the w2v model and the glove model. Embeddings that are trained from scratch do not seem to affect the baseline and Rescore models; on the other hand, they improve SemSim and SoftEmb, narrowing the performance gap between these two models and the other two. Note that the baseline model only uses embeddings for the semantic features of the object pair, and disregards the embedding vectors for predicates; on the other hand, SemSim and SoftEmb also involve a hidden representation of the predicate in the form of a high-dimensional embedding vector, so it probably stands to reason that these two rely more strongly on effective embedding representations.
It is also observed that, for all models, w2v embeddings give better scores than glove embeddings; a little surprising, perhaps, as one would have expected GloVe's incorporation of global statistics to match Word2vec's performance, if not improve upon it. By inspection, we verify that, for SoftEmb and SemSim, the w2v model improves the prediction of some multi-word predicates, such as "next to", "in the front of", "on the left of", "on the right of", etc. but also synonymous predicates of these, such as "near", "beside" and "on".

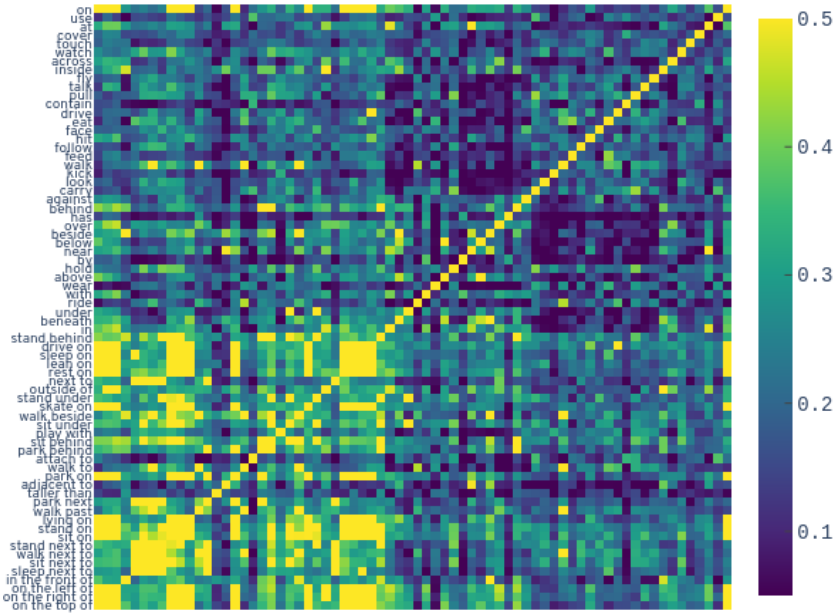


Figure 4.4: VRD/all, predicate similarities according to gnews (Word2vec embeddings pre-trained on Google News)

Ultimately, however, we still observe that the baseline model is the preferred choice when considering a heterogeneous set of predicates; it seems like none of the methods in use can deal effectively with every predicate.

Figures 4.4, 4.5 and 4.6 show the predicate-to-predicate similarities for VRD/all according to the three embedding models. As mentioned previously, the w2v and glove models that have been trained on our vocabulary deal with multi-word predicates differently; they are joined by underscore to transform them into individual tokens. This helps the models treat them as individual tokens and learn unique embeddings for them. This is in contrast to the gnews model, where the embeddings for multi-word predicates are computed by simply averaging the embeddings of all individual words in the phrase.

Upon an initial glance, it seems that similar patterns can be observed for gnews and w2v, with w2v enforcing the similarity scores for many predicate pairs. So for example, in w2v predicate pairs that should be similar get a similarity score that is the same as or higher than the ones in gnews. But perhaps more noticeably, the w2v embeddings also decrease the similarity scores for many predicate pairs. As such, the w2v embeddings seem to a more easily discernible representation of the predicates than the gnews ones. The glove embeddings seem to go on a different tangent altogether however, and contain high similarity scores for many predicate pairs.

It should be noted however, that this is just a preliminary observation; the predicate pair similarities are analyzed in greater detail for the spatial and activity predicates, followed by some discussion points.

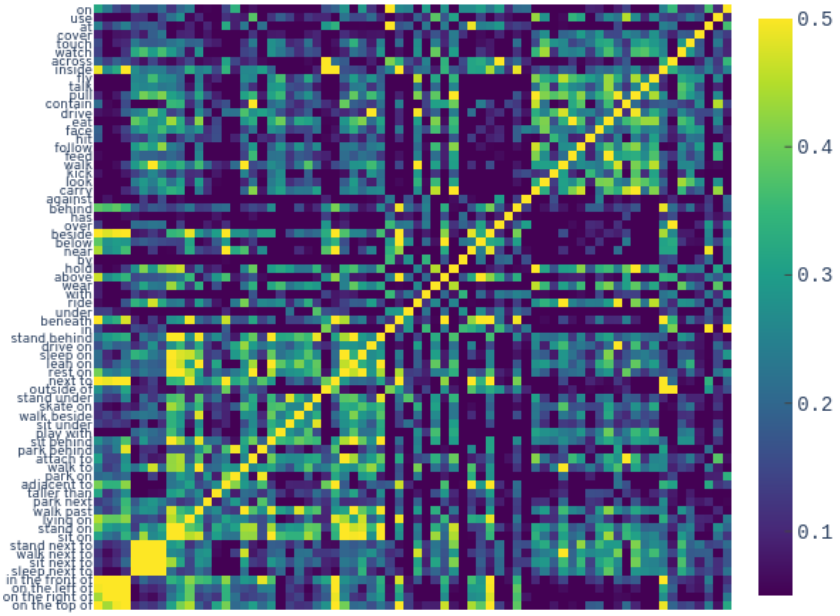


Figure 4.5: VRD/all, predicate similarities according to w2v (Word2vec embeddings tailored on our vocabulary)

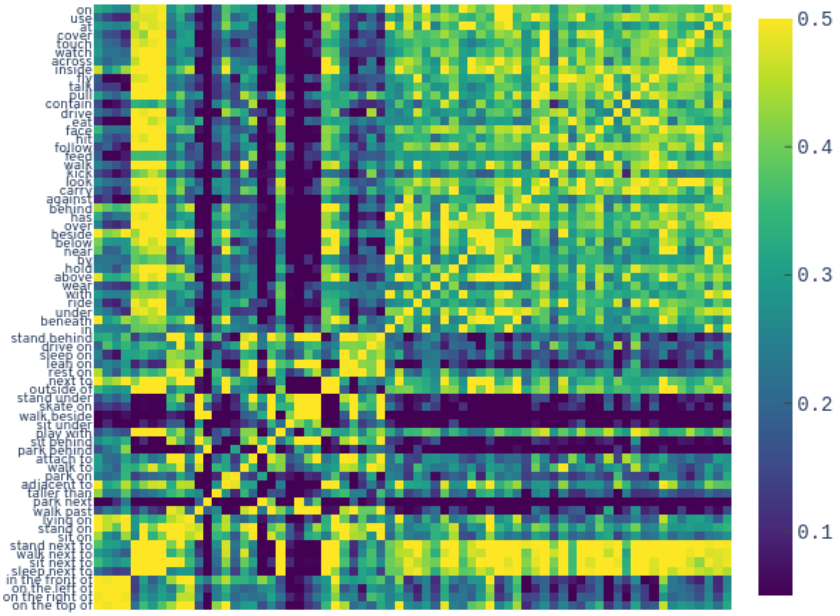


Figure 4.6: VRD/all, predicate similarities according to glove (GloVe embeddings tailored on our vocabulary)

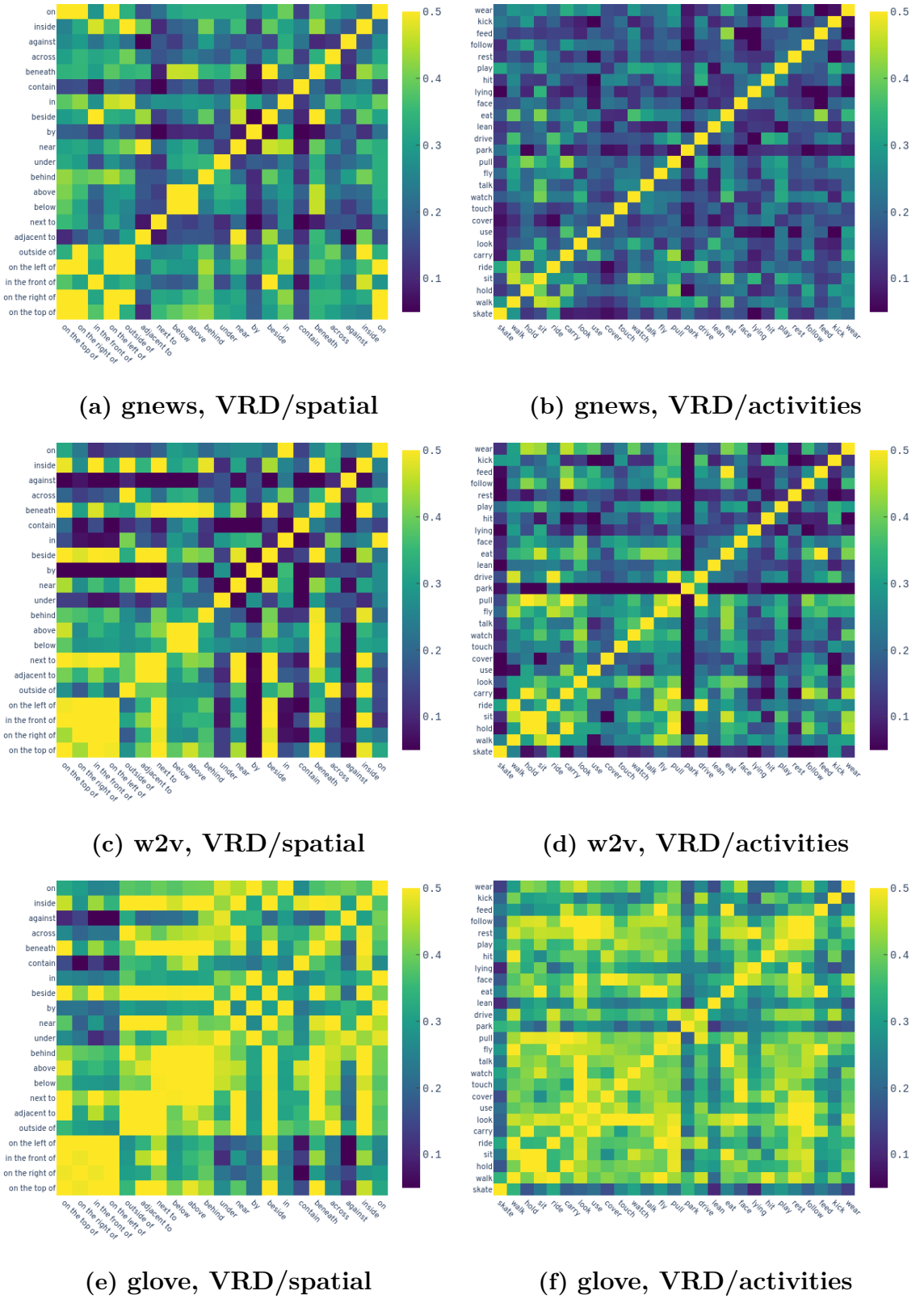
4.2.1.2 Spatial predicates

Model type	R@50	R@50 ZS	R@20	R@20 ZS	Avg.
gnews (baseline)	92.18	72.89	75.77	42.70	70.88
gnews+SoftEmb	84.45	59.36	70.29	40.64	63.68
gnews+SemSim	85.20	58.79	70.84	39.62	63.61
gnews+Rescore	90.02	71.33	73.83	45.23	70.10
w2v (baseline)	91.92	71.33	75.64	41.78	70.17
w2v+SoftEmb	83.78	59.43	71.35	39.79	63.59
w2v+SemSim	84.67	60.62	72.17	39.93	64.35
w2v+Rescore	90.63	72.55	74.43	46.45	71.02
glove (baseline)	91.80	72.38	75.40	43.04	70.66
glove+SoftEmb	82.83	57.13	68.87	36.98	61.45
glove+SemSim	83.66	59.53	69.37	37.32	62.47
glove+Rescore	89.81	72.11	73.30	45.37	70.15

Table 4.8: PREDCLS, VRD/spatial, semantic features Comparison of different general-purpose embeddings

With spatial predicates (see Table 4.8) the w2v and glove models do not seem to affect the models for the most part. The noteworthy changes include how in R@50 zero-shot using w2v embeddings help the Rescore model beat the baseline (by a very small margin), and glove embeddings help the Rescore model perform almost equally well as the baseline (very small difference). Additionally, the SoftEmb and SemSim models perform equivalently for gnews and w2v embeddings, but their performance noticeably decreases when using glove.
If the average recall scores are considered, however, it is noticed that very similar patterns are observed throughout irrespective of the embeddings in use: the baseline and Rescore models perform very similar to each other, just as the SoftEmb and SemSim models do.
There are several observations to be made in the predicate similarity heatmaps in Figures 4.7a, 4.7c and 4.7e. Firstly, when comparing the gnews embeddings with w2v and glove embeddings, it is easy to note the difference of similarities between multi-word predicates and others. If we consider the first four multi-word predicates at the bottom left of the figure; namely, “on the top of”, “on the right of”, “in the front of”, and “on the left of”; one would expect their embeddings to be very similar to one another since these terms occur in very similar contexts frequently. While that’s true for the multi-word predicates starting with “on”, “in the front of” is quite dissimilar from them. Moreover, these multi-word predicates do not seem to have a high similarity with any other spatial predicate.
The w2v and glove embeddings deal with multi-word predicates differently as compared to gnews, as they consider them as single entities with their own unique embeddings. In Figures 4.7c and 4.7e, it can be seen that the multi-word predicates starting with “on” and “in” are similar not just to one another, but also other spatial predicates that occur in similar contexts such as “inside”, “beneath”, “next to” and

"beside". This reinforces our belief in the fact that treating multi-word predicates as individual tokens with their own embeddings is a better representation of them as opposed to simply averaging the embeddings of all individual words involved in the multi-word predicate. Since the w2v and glove embedding models adopt the former approach, we see high similarity scores between multi-word predicates, as well as other predicates that should in fact have high similarity scores, such as (adjacent to, next to), (beside, next to), and (outside of, across). There seems to be quite a high level of agreement between the w2v and glove embeddings heatmaps, apart from the obvious fact that the glove embeddings have much higher similarity scores between predicates than the w2v embeddings. However, there are differences between these two embeddings too; for instance, (under, contain) seems to have somewhat of a high similarity score in glove but practically no similarity in w2v.



(a) gnews, VRD/spatial

(b) gnews, VRD/activities

(c) w2v, VRD/spatial

(d) w2v, VRD/activities

(e) glove, VRD/spatial

(f) glove, VRD/activities

Figure 4.7: Predicate similarities according to semantic word embeddings Here we compare three models: a pre-trained Word2vec (gnews), a trained-from-scratch Word2vec (w2v) and trained-from-scratch GloVe (glove) model

4.2.1.3 Activity predicates

Table 4.9 shows the results on activity predicates. Several observations can be made here. Firstly, independently of the embeddings in use, SemSim always outperforms SoftEmb by a few recall points, and given that, with activity predicates, the two models have high recall scores, this might reveal how the first one is actually a better version of the other. Indeed, the only difference between the two is how the predicate embedding vector $\mathbf{p}$ is computed: whether through a linear combinations with initial scores, or predicted from the (linear) fusion layer. However, note that there is no loss function driving the network to learn appropriate semantic-agnostic scores s_k, so computing $\mathbf{p}$ as a linear combination of s_k and the class embeddings as in Eq. 3.1 can become a redundant, unnecessary step.
Ignoring SoftEmb for a moment, the other two methods seem to benefit from the use of w2v and glove embeddings. This is mostly the case for zero-shot prediction, which is, again, a less stable measure, given the low number of zero-shot test triplets. However, w2v embeddings greatly improve the scores for the baseline and Rescore models for both R@50 ZS and R@20 ZS, and, similarly, the use of glove embeddings results in quite a jump for SemSim, resulting in the highest recall scores achieved on VRD activities so far. Refer to Table A.1 in Appendix A for the results on VG/activities, showing very similar patterns.

Model type	R@50	R@50 ZS	R@20	R@20 ZS	Avg.
gnews (baseline)	95.89	62.67	91.07	37.78	71.85
gnews+SoftEmb	98.11	84.89	96.35	71.56	87.72
gnews+SemSim	98.49	84.44	97.21	73.33	88.37
gnews+Rescore	98.74	85.33	96.78	70.67	87.88
w2v (baseline)	96.20	64.67	91.68	41.33	73.47
w2v+SoftEmb	98.25	85.33	96.10	67.33	86.75
w2v+SemSim	98.70	85.33	97.02	73.33	88.60
w2v+Rescore	99.08	88.67	97.64	76.00	90.34
glove (baseline)	95.51	64.00	90.07	34.67	71.06
glove+SoftEmb	98.77	88.67	96.92	70.00	88.59
glove+SemSim	99.01	90.00	97.88	79.33	91.55
glove+Rescore	98.63	86.67	97.09	72.67	88.76

Table 4.9: PREDCLS, VRD/activities, all features Note that, due to a low number (75) of zero-shot test triplets, zero-shot prediction recalls are less stable: consider a standard deviation of 4.5.

Analyzing the cosine similarities (heatmaps in Figs. 4.7b, 4.7d, and 4.7f) makes for some very interesting insights. The embeddings from w2v seem to support a lot of the hypotheses made by the gnews embeddings, and seems to make them clearer. So for example, "park" does not seem to have a particularly high similarity with any predicate other than "drive", "ride" and "walk". The w2v embeddings seem to bolster this hypothesis. Similarly, predicate pairs with high similarity such as (carry, hold), (drive, ride), and (feed, eat) in gnews see a higher similarity in w2v

embeddings. The w2v embeddings also seem to more disassociate some predicate pairs that should not be similar but have somewhat of a high similarity score in gnews, such as (watch, park).
The heatmap for glove embeddings tells an altogether different story. Here, high similarity scores for a lot of the activity predicate pairs are observed. Some are sensible, such as (rest, lying) and (feed, eat), but some are perhaps not so sensible, such as (fly, eat). Perhaps a higher max similarity score would help see the differences in pred-pred similarities better. These embeddings do not seem to adversely impact the performance of the models as compared to the gnews or w2v embeddings, however.

4.2.2 Fine-tuned embeddings

This section compares the use of embeddings obtained from the trained-from-scratch Word2vec and GloVe models (w2v and glove) with the embeddings obtained from the fine-tuned versions of these models. The fine-tuned models have been trained on the COCO image captions for 20, 50, and 100 epochs (e.g w2v+coco-20 refers to the w2v model, fine-tuned for 20 epochs).
Note that, since there are many models involved in the following experiments, bar plots have been made use of to highlight the most interesting results and draw comparisons. The respective tables can be found in Appendix A.

4.2.2.1 All predicates

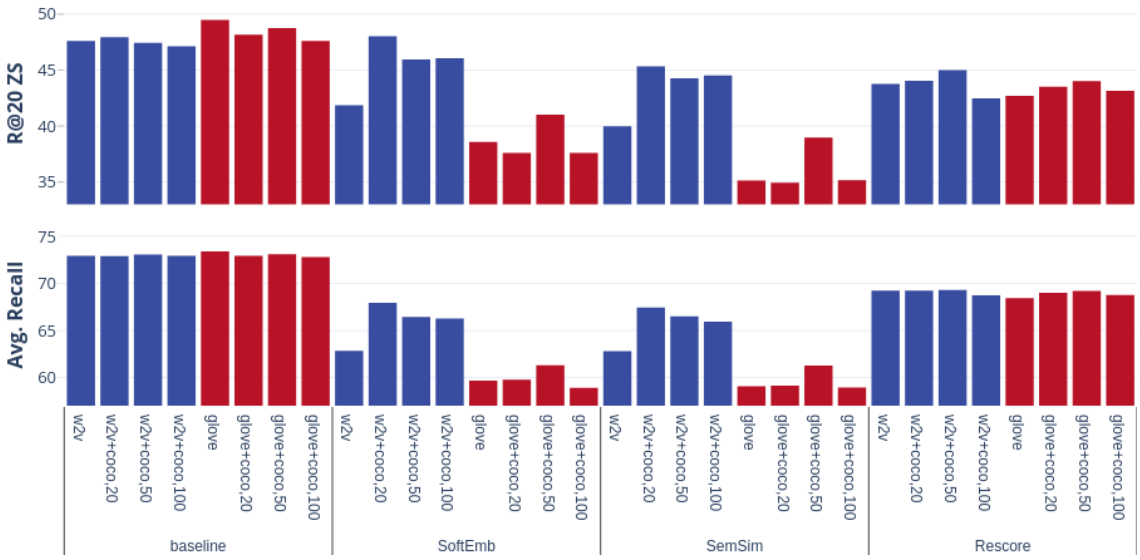


Figure 4.8: PREDCLS, VRD/all, all features The plot shows R@20 Zero-Shot and the average of the recall values of interest (R@50, R@50 ZS, R@20, R@20 ZS). See Table A.2 in Appendix A for a more detailed overview of the recall values

Figure 4.8 show the results of fine-tuning on VRD/all in terms of R@20 zero-shot and the average of recall scores. The first thing to observe right away is that fine-tuning w2v and glove embeddings have little to no effect on the baseline model.

Again, note that the baseline model makes use only of object embeddings, and not predicate embeddings. The fine-tuned embeddings show a small improvement for Rescore for R@20 zero-shot. If the average recall score is considered, however, the performance of the fine-tuned embeddings for Rescore is equivalent to that of their trained-from-scratch counterparts, for both w2v and glove.

The plots tell quite a different story for the SoftEmb and SemSim, however: both methods show considerable improvement when fine-tuned embeddings are used, for both R@20 zero-shot and average recall score. There are two interesting things of note here:

- The fine-tuned w2v embeddings give much higher scores for the SemSim and SoftEmb models as compared to the fine-tuned glove embeddings for all epochs, for both R@20 zero-shot and average recall score. A similar pattern was observed earlier as well, when the w2v model noticeably outperformed the glove model.
- The performance of the fine-tuned w2v embeddings is fairly consistent across the different epochs, with coco-20 usually getting the best score. However in the case of fine-tuned glove embeddings, it seems that only the coco-50 embeddings manage to outperform the trained-from-scratch glove model; the other fine-tuned embeddings perform mostly equivalent to or worse than the trained-from-scratch ones. We hypothesize that coco-50 seems to be a good fit between underfit and overfit embeddings generated by coco-20 and coco-50 respectively; however, this is just a speculation on our part.

4.2.3 Spatial predicates

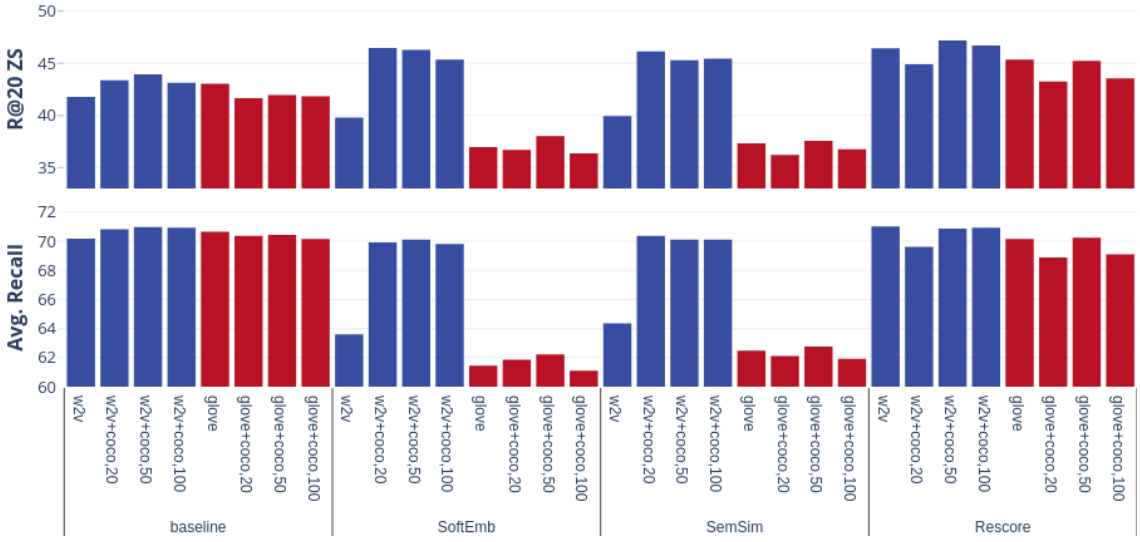


Figure 4.9: PREDCLS, VRD/spat, all features The plot shows R@20 Zero-Shot and the average of the recall values of interest (R@50, R@50 ZS, R@20, R@20 ZS). See Table A.3 in Appendix A for a more detailed overview of the recall values.

The fine-tuned embeddings have different impacts for w2v and glove. For w2v, the scores for SemSim and SoftEmb go up when using fine-tuned embeddings, whereas the baseline and Rescore models do not seem to be affected majorly. Once more, this is reasonable as the approaches showing improvement are those which predict the predicate as a vector. R@20 zero-shot shows the most improvement with the coco embeddings, with all fine-tuned models beating their corresponding baseline. Rescore actually achieved that with just trained-from-scratch w2v too, but the coco embeddings help the SimSim and SoftEmb models achieve that too. Finally, if the average recall scores are considered, the use of fine-tuned embeddings results in all models performing equivalently.
Fine-tuned glove embeddings tell a very different story, however; they do not seem to help any of the models in particular. There is not much difference between trained-from-scratch glove embeddings vs. glove+coco embeddings, for any number of epochs. If R@20 is considered, glove+coco-50 seems to show some promise for SemSim and SoftEmb. But for the most part, the fine-tuning glove embeddings does not seem to hold much promise for spatial predicates, for any model.

4.2.4 Activity predicates

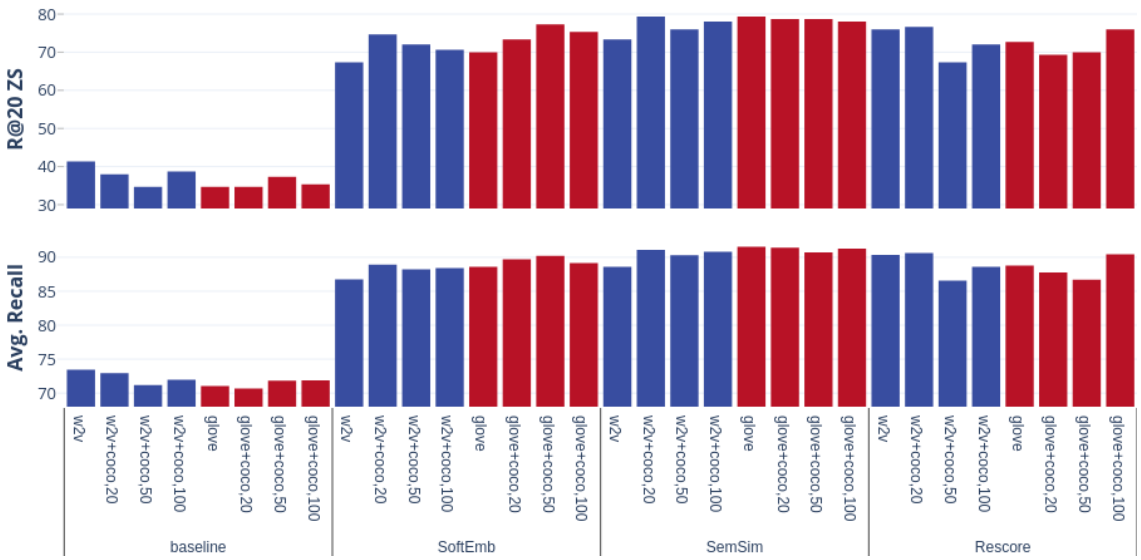


Figure 4.10: PREDCLS, VRD/activities, all features, Avg. of R@50, R@50 ZS, R@20, R@20 ZS. See Table A.4 in Appendix A for a more detailed overview of the recall values

Through these graphs, it is observed that the baseline models show no improvement for activity predicates by use of the fine-tuned embeddings. This was observed previously too and therefore expected. What was also observed previously was that the SoftEmb and SemSim models outperform all other models for VRD/activities. Moreover, SemSim is still better than SoftEmb.
Furthermore, it is observed that for R@20 zero-shot and average recall scores, the fine-tuned w2v embeddings improve both SoftEmb and SemSim. However, for the

same evaluation metrics, the fine-tuned glove embeddings seem to help SoftEmb but not SemSim.
Finally, the Rescore model shows a very similar pattern for trained-from-scratch vs. fine-tuned embeddings for both w2v and glove. The w2v model and glove+coco-100 perform almost equally, with w2v+coco-20 slightly outperforming them both.

Final considerations

Ultimately, in this section we find that the two vocabulary-specific embedding models can improve SemSim and SoftEmb more than the other two methods, and that, as a general pattern, Word2vec embeddings seem to help the models more than GloVe embeddings. This can be due to the fact that the glove model is more "generous", in the sense that it generally gives higher similarity scores between all predicates, and this can make it difficult for the network to distinguish between them. As for fine-tuning the embeddings on the MS-COCO dataset, it seems to help the w2v model, leading to significantly improved performance for the SemSim and SoftEmb models in particular, especially for all predicates and spatial predicates. w2v+coco-20 seems to be the most promising of the fine-tuned w2v models, with almost consistently good performance throughout, and the general trend may suggest that w2v models do not need to be fine-tuned for many epochs over a domain-specific dataset. Finally, fine-tuning the GloVe embeddings shows some promise too, however there does not seem to be a consistent pattern suggesting how to exactly benefit from it. Overall, though, fine-tuning the glove model does not help in getting a significant performance gain over the w2v models.

5
Conclusion

The main finding of this thesis work is that a classification model can leverage some kind of knowledge of the classes involved, and such knowledge can be derived through the training of text-based distributional embedding models, a process able to discover semantic similarities between words/expressions. In order to improve the quality of the embedding representations, wherever some multi-word class labels are involved, embeddings for these can be computed by considering the labels as atomic entities (with multi-word expressions on the same level as words); additionally, fine-tuning general-purpose embedding models on text corpora that more strictly relates to the domain at hand ("domain fine-tuning"), can potentially further improve the representations.
In the case of visual relationship detection, the class domain is extremely diverse; in fact, predicates can be of very different types, denoting different kinds of object-to-object relationships and as such, semantic embeddings might behave in unexpected ways. Here we show that, for both of the datasets in use, while the three proposed methods for *class semantics* do not seem to work for the entire set of predicate categories, they can lead to major improvements when a subset of classes having certain characteristics is considered.
Our experiments show that these improvements are enabled when the classes do not all belong to the same semantic area (e.g the area of "spatiality") and, instead, cover a wider range of "contexts"; this makes the respective embedding vectors more spread out, and more meaningful to each other.
Ultimately, we find that in the context of VRD, semantic rescoring (described in Sec. 3.3.3) is the safest of the presented methods for predicate semantics; on one hand, when the set of predicates is not appropriate for a meaningful representation with distributional embedding, semantic rescoring seems to yield results that are similar to the baseline; on the other hand, when the set of predicates is appropriate (e.g the set of activity predicates), the improvement it enables is comparable with that brought by the two other methods. Incidentally, the method also simply consists of a matrix multiplication, so it is the one method with the lowest computational overhead.

5.1 Limitations and future work

The major limitation of the described methods for class semantics is that they only work with a set of classes with certain, partially unclear, characteristics. Specifically, unless a different kind of embedding is used, the methods can only be applied on a set

of classes that are represented in an *appropriate* way by distributional embeddings. If a classification problem has a certain number of categories, but only a few of them are appropriately represented by distributional embeddings, then this opens up the question whether it is possible to apply this method in a selective fashion. In our case, for example, can we leverage the improvement achieved on VRD/activities for improving the classification of the same predicates on the original dataset (VRD/all)? Perhaps it is possible to apply the methods to only a subset of classes involved; for example, since semantic rescoring (*Rescore*, in the result tables) is a simple matrix multiplication, an idea could be to tweak the matrix M to only affect the scores for specific classes.

One important issue we identified with using distributional embeddings is that semantic similarity does not seem to simulate the difference between synonyms and antonyms too well: consider once again *above*/*below*, which are antonymous to each other, and yet they occur in very similar contexts, and are thus represented by similar vectors. Keeping this in mind, Dou et al. [6] is an interesting work in which the authors attempt to train word embeddings such that they can identify and deal with antonyms in a way that would be more suited here. We did not get the opportunity to experiment with this work, but it could be a promising venture to learn embedding representations for predicates using a word embedding training technique that keeps antonyms in mind, and observe the impact of using such embeddings for predicate prediction, especially for methods like SemSim and SoftEmb.

Finally, we suggest one last cue for future work, that is, to investigate on similarity measures for embeddings other than cosine. Another common choice is in fact the L2 distance (or Mean Squared Error, MSE) and, interestingly, using it together with a SemSim layer would have an additional mathematical interpretation, that has to do with a known probabilistic model (Gaussian Mixture Model, GMM).

Bibliography

[1] UC Business Analytics R Programming Guide – Feedforward Deep Learning Models, 2020. URL `http://uc-r.github.io/feedforward_DNN`.

[2] Raffaella Bernardi, Ruket Cakici, Desmond Elliott, Aykut Erdem, Erkut Erdem, Nazli Ikizler-Cinbis, Frank Keller, Adrian Muscat, and Barbara Plank. Automatic description generation from images: A survey of models, datasets, and evaluation measures. *Journal of Artificial Intelligence Research*, 55:409–442, 2016.

[3] Avrim Blum and Ronald L Rivest. Training a 3-node neural network is np-complete. In *Advances in neural information processing systems*, pages 494–501, 1989.

[4] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. L2 regularization for learning kernels, 2012.

[5] Bo Dai, Yuqi Zhang, and Dahua Lin. Detecting visual relationships with deep relational networks, 2017.

[6] Zehao Dou, Wei Wei, and Xiaojun Wan. *Improving Word Embeddings for Antonym Detection Using Thesauri and SentiWordNet: 7th CCF International Conference, NLPCC 2018, Hohhot, China, August 26–30, 2018, Proceedings, Part II*, pages 67–79. 08 2018. ISBN 978-3-319-99500-7. doi: 10.1007/978-3-319-99501-4_6.

[7] Ali Farhadi, Mohsen Hejrati, Mohammad Amin Sadeghi, Peter Young, Cyrus Rashtchian, Julia Hockenmaier, and David Forsyth. Every picture tells a story: Generating sentences from images. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, page 15–29, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-15561-1.

[8] Li Fei-Fei, Ranjay Krishna, Danfei Xu, and Amelie Byun. CS231n: Convolutional Neural Networks for Visual Recognition, 2020. URL `http://cs231n.stanford.edu/`.

[9] Ross Girshick. Fast R-CNN, 2015.

[10] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2013.

[11] Raúl Gombru. Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those con-

fusing names, 2018. URL https://gombru.github.io/2018/05/23/cross_entropy_loss/.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[13] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators.

[14] Justin Johnson, Ranjay Krishna, Michael Stark, Li-Jia Li, David Shamma, Michael Bernstein, and Li Fei-Fei. Image retrieval using scene graphs. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[15] Jaewon Jung and Jongyoul Park. Visual relationship detection with language prior and softmax, 2019.

[16] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, Apr 2020. ISSN 1573-7462. doi: 10.1007/s10462-020-09825-6. URL http://dx.doi.org/10.1007/s10462-020-09825-6.

[17] Fredrik Kindström, Alexander Håkansson, Shruthi Dinakaran, Giovanni Pagliarini, and Raya Altarabulsi. Classifying electrocardiograms using deep learning. 2019.

[18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[19] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Fei-Fei Li. Visual genome: Connecting language and vision using crowdsourced dense image annotations, 2016.

[20] Kongming Liang, Yuhong Guo, Hong Chang, and Xilin Chen. Visual relationship detection with deep structural ranking. 2018.

[21] Wentong Liao, Cuiling Lan, Wenjun Zeng, Michael Ying Yang, and Bodo Rosenhahn. Exploring the semantics for visual relationship detection. *CoRR*, abs/1904.02104, 2019. URL http://arxiv.org/abs/1904.02104.

[22] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. URL http://arxiv.org/abs/1405.0312.

[23] Cewu Lu, Ranjay Krishna, Michael Bernstein, and Li Fei-Fei. Visual relationship detection with language priors. In *European Conference on Computer Vision*, 2016.

[24] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar

Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining, 2018.

[25] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[26] Christopher Potts Nicholas Dingwall. Mittens: An extension of glove for learning domain-specialized representations, 2018.

[27] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL `http://www.aclweb.org/anthology/D14-1162`.

[28] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[29] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015.

[30] Mohammad Amin Sadeghi and Ali Farhadi. Recognition using visual phrases. 2011.

[31] Google Scholar. Visual relationship detection — google scholar. URL `https://scholar.google.se/scholar?hl=en&as_sdt=0,5&as_vis=1&q=visual+relationship+detection`.

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[34] TensorFlow. Embedding projector - visualization of high-dimensional data. URL `https://projector.tensorflow.org/`.

[35] Shiva Verma. Multi-Label Image Classification with Neural Network | Keras, 2019. URL `https://towardsdatascience.com/multi-label-image-classification-with-neural-network-keras-ddc1ab1afede`.

[36] Rowan Zellers, Mark Yatskar, Sam Thomson, and Yejin Choi. Neural motifs: Scene graph parsing with global context. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5831–5840, 2018.

[37] Qianru Zhang, Meng Zhang, Tinghuan Chen, Zhifei Sun, Yuzhe Ma, and Bei Yu. Recent advances in convolutional neural network acceleration, 2018.

[38] Yaohui Zhu and Shuqiang Jiang. Deep structured learning for visual relationship detection. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[39] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey, 2019.

A

Appendix I

A.1 Embeddings tailored on our vocabulary for activity predicates

Model type	R@50	R@50 ZS	R@20	R@20 ZS	Avg.
gnews (baseline)	87.31	31.05	83.69	23.68	56.44
gnews+SoftEmb	98.39	80.53	97.65	80.00	89.14
gnews+SemSim	98.42	77.89	97.71	77.89	87.98
gnews+Rescore	99.15	82.63	98.33	80.00	90.03
w2v (baseline)	81.80	23.51	76.46	16.84	49.65
w2v+SoftEmb	98.93	77.89	98.21	76.14	87.80
w2v+SemSim	98.92	80.00	98.18	78.60	88.92
w2v+Rescore	99.21	82.46	98.45	81.40	90.38
glove (baseline)	86.79	27.72	83.00	22.46	54.99
glove+SoftEmb	98.52	77.54	97.81	76.84	87.68
glove+SemSim	98.15	79.65	97.39	78.60	88.44
glove+Rescore	99.27	84.21	98.24	80.70	90.61

Table A.1: PREDCLS, VG/activities, semantic features

A.2 Fine-tuned embeddings for all predicates

Model type	R@50	R@50 ZS	R@20	R@20 ZS	Avg.
w2v (baseline)	91.99	75.58	76.56	47.60	72.94
w2v+SoftEmb	80.73	61.55	67.21	41.87	62.84
w2v+SemSim	81.23	62.66	67.33	39.99	62.80
w2v+Rescore	89.38	71.26	72.56	43.76	69.24
coco20 (baseline)	91.89	75.45	76.36	47.95	72.91
coco20+SoftEmb	84.68	67.92	71.16	48.03	67.95
coco20+SemSim	85.51	67.58	71.28	45.34	67.43
coco20+Rescore	89.51	70.96	72.50	44.05	69.25
coco50 (baseline)	92.08	76.30	76.51	47.43	73.08
coco50+SoftEmb	83.83	65.83	70.08	45.94	66.42
coco50+SemSim	84.68	66.51	70.50	44.27	66.49
coco50+Rescore	88.98	70.79	72.40	45.00	69.29
coco100 (baseline)	92.07	76.22	76.38	47.13	72.95
coco100+SoftEmb	83.45	65.27	70.29	46.07	66.27
coco100+SemSim	83.56	65.36	70.23	44.53	65.92
coco100+Rescore	89.22	71.04	72.27	42.47	68.75

(a)

Model type	R@50	R@50 ZS	R@20	R@20 ZS	Avg.
glove (baseline)	91.59	76.39	76.12	49.49	73.40
glove+SoftEmb	77.68	57.49	64.90	38.58	59.66
glove+SemSim	78.55	58.04	64.62	35.12	59.08
glove+Rescore	88.67	70.74	71.61	42.69	68.43
coco20 (baseline)	91.75	75.41	76.40	48.16	72.93
coco20+SoftEmb	78.01	58.73	64.79	37.60	59.78
coco20+SemSim	78.73	58.25	64.63	34.94	59.14
coco20+Rescore	88.83	71.94	71.68	43.50	68.99
coco50 (baseline)	91.73	75.36	76.58	48.76	73.11
coco50+SoftEmb	77.91	59.50	66.72	41.02	61.29
coco50+SemSim	79.14	60.05	66.87	38.96	61.26
coco50+Rescore	89.05	71.69	72.01	44.01	69.19
coco100 (baseline)	91.75	75.71	76.10	47.60	72.79
coco100+SoftEmb	76.93	56.59	64.51	37.60	58.91
coco100+SemSim	78.28	57.53	64.72	35.16	58.92
coco100+Rescore	88.99	71.64	71.30	43.16	68.77

(b)

Table A.2: PREDCLS, VRD/all, all features Comparison of fine-tuned embeddings for Word2Vec and GloVe

A.3 Fine-tuned embeddings for spatial predicates

Model type	R@50	R@50 ZS	R@20	R@20 ZS	Avg.
w2v (baseline)	91.92	71.33	75.64	41.78	70.17
w2v+SoftEmb	83.78	59.43	71.35	39.79	63.59
w2v+SemSim	84.67	60.62	72.17	39.93	64.35
w2v+Rescore	90.63	72.55	74.43	46.45	71.02
coco20 (baseline)	91.96	72.11	75.84	43.37	70.82
coco20+SoftEmb	88.82	68.36	75.98	46.48	69.91
coco20+SemSim	89.54	69.24	76.54	46.15	70.37
coco20+Rescore	89.37	70.79	73.43	44.90	69.62
coco50 (baseline)	91.99	72.08	75.92	43.95	70.98
coco50+SoftEmb	89.12	69.27	75.77	46.28	70.11
coco50+SemSim	89.66	69.27	76.23	45.30	70.12
coco50+Rescore	90.12	71.33	74.83	47.19	70.87
coco100 (baseline)	92.12	72.41	76.00	43.14	70.92
coco100+SoftEmb	88.97	69.17	75.68	45.37	69.80
coco100+SemSim	89.31	69.68	76.02	45.44	70.11
coco100+Rescore	90.06	72.28	74.70	46.72	70.94

Model type	R@50	R@50 ZS	R@20	R@20 ZS	Avg.
glove (baseline)	91.80	72.38	75.40	43.04	70.66
glove+SoftEmb	82.83	57.13	68.87	36.98	61.45
glove+SemSim	83.66	59.53	69.37	37.32	62.47
glove+Rescore	89.81	72.11	73.30	45.37	70.15
coco20 (baseline)	91.94	72.72	75.19	41.65	70.37
coco20+SoftEmb	83.25	58.59	68.88	36.71	61.86
coco20+SemSim	84.12	58.72	69.37	36.21	62.11
coco20+Rescore	89.17	70.79	72.34	43.24	68.89
coco50 (baseline)	91.93	72.41	75.42	41.95	70.43
coco50+SoftEmb	82.77	57.74	70.37	38.03	62.23
coco50+SemSim	83.44	59.13	70.88	37.56	62.75
coco50+Rescore	90.02	72.62	73.09	45.23	70.24
coco100 (baseline)	91.70	71.87	75.16	41.85	70.15
coco100+SoftEmb	82.59	56.69	68.75	36.38	61.10
coco100+SemSim	83.56	57.81	69.54	36.75	61.91
coco100+Rescore	89.31	70.72	72.84	43.54	69.10

Table A.3: PREDCLS, VRD/spatial, semantic features Comparison of fine-tuned embeddings for Word2Vec and GloVe

A.4 Fine-tuned embeddings for activity predicates

Model type	R@50	R@50 ZS	R@20	R@20 ZS	Avg.
w2v (baseline)	96.20	64.67	91.68	41.33	73.47
w2v+SoftEmb	98.25	85.33	96.10	67.33	86.75
w2v+SemSim	98.70	85.33	97.02	73.33	88.60
w2v+Rescore	99.08	88.67	97.64	76.00	90.34
coco20 (baseline)	96.20	66.00	91.51	38.00	72.93
coco20+SoftEmb	98.60	85.33	96.99	74.67	88.90
coco20+SemSim	99.04	88.00	98.01	79.33	91.10
coco20+Rescore	98.94	89.33	97.64	76.67	90.64
coco50 (baseline)	95.65	64.00	90.48	34.67	71.20
coco50+SoftEmb	98.77	84.67	97.47	72.00	88.22
coco50+SemSim	99.04	88.00	98.08	76.00	90.28
coco50+Rescore	98.56	83.33	96.92	67.33	86.54
coco100 (baseline)	95.86	62.00	91.44	38.67	71.99
coco100+SoftEmb	99.01	86.67	97.36	70.67	88.43
coco100+SemSim	99.08	88.00	98.25	78.00	90.83
coco100+Rescore	98.56	87.33	96.47	72.00	88.59

Model type	R@50	R@50 ZS	R@20	R@20 ZS	Avg.
glove (baseline)	95.51	64.00	90.07	34.67	71.06
glove+SoftEmb	98.77	88.67	96.92	70.00	88.59
glove+SemSim	99.01	90.00	97.88	79.33	91.55
glove+Rescore	98.63	86.67	97.09	72.67	88.76
coco20 (baseline)	95.27	62.67	90.14	34.67	70.69
coco20+SoftEmb	98.90	89.33	97.19	73.33	89.69
coco20+SemSim	99.21	90.00	97.74	78.67	91.40
coco20+Rescore	98.73	86.00	96.99	69.33	87.76
coco50 (baseline)	95.86	63.33	90.82	37.33	71.84
coco50+SoftEmb	98.12	88.67	96.71	77.33	90.21
coco50+SemSim	98.70	88.00	97.57	78.67	90.73
coco50+Rescore	98.12	82.00	96.64	70.00	86.69
coco100 (baseline)	96.10	64.67	91.37	35.33	71.87
coco100+SoftEmb	98.60	85.33	97.33	75.33	89.15
coco100+SemSim	99.11	90.00	97.91	78.00	91.26
coco100+Rescore	98.90	90.00	96.95	76.00	90.46

Table A.4: PREDCLS, VRD/activities, all features Comparison of fine-tuned embeddings for Word2Vec and GloVe