UNIVERSITY OF GOTHENBURG

DEPARTMENT OF PHILOSOPHY,
LINGUISTICS AND THEORY OF SCIENCE

SPATIAL RELATIONS IN EMERGENT LANGUAGES

This is the subtitle

Dominik Künkele

Master's Thesis:	30 credits
Programme:	Master's Programme in Language Technology
Level:	Advanced level
Semester and year:	Spring, 2023
Supervisor:	Simon Dobnik
Examiner:	Asad Sayeed
Keywords:	keyword 1, keyword 2, keyword 3

Abstract

Preface

Contents

1 Introduction

[Dominik Künkele] 2 pages

1.1 Research Question

1.2 Motivation

1.3 Contribution

1.4 Scope

2 Background and Related Work

[Dominik Künkele] 7 pages

2.1 Language Games

[Dominik Künkele]
- EGG
- Resnet

2.2 Dataset

[Dominik Künkele]
- CLEVR
—– 3D objects
—– simple objects
- CLEVR baselines

3 Experimental Setup

[Dominik Künkele] 15 pages

In this section I will describe which experiments were conducted to answer the questions from the previous sections as well as the setup of these experiments. This will be done in two parts. The first part contains the creation of the datasets that will be used in the experiments. In the second part I will go deeper into discussing each experiment, including the necessity of the experiment, the specifities of the dataset, as well as the final architecture of the models.

[Dominik Künkele] ?

3.1 Creation of the dataset

The basis for my datasets is the CLEVR dataset (Johnson et al. (2016)). This dataset includes 3D-generated images depicting scenes with different kinds of objects. Each of these objects has different combinations attributes, such as *shape*, *color*, *size* and *material*. The possible values of these attributes are listed in Table 1. Three to ten objects are randomly placed into the scene and assigned with random attributes. To enhance realism and reduce ambiguity, objects do not intersect, have a certain distance from each other, and are at least partially visible. Figure 1 shows an example of a generated image in the CLEVR dataset.

shape	**color**	**size**	**material**
cube	gray	small	rubber
sphere	red	large	metal
cylinder	blue		
	green		
	brown		
	purple		
	cyan		
	yellow		

Table 1: Attributes of objects in the CLEVR dataset

Furthermore, the dataset contains information about each scene. This includes all selected attributes for each object as well as the exact position of the centers of all the objects, both 3D-coordinates in the 3D scene and 2D-coordinates in the final rendered image. In addition, simple spatial relations (in front of, behind, left, right) between the objects are calculated and stored. These are simply based on the 3D-coordinates of the objects in relation to the position of the camera.

[Dominik Künkele] ambiguity

This research investigates how agents communicate about the relations of objects seen in images. For that reason, the original CLEVR dataset offers too little control over how the objects are created and in which relation they stand to each other. Following, I extended the source code to generate images for the CLEVR dataset.[1] To simplify the generation and the later learning of the models, I only focused on the attributes with a high impact on the appearance of the object, namely the *shape*, *size* and *color*. The *material* is always the same for all objects in a generated image. There were three main extensions to the code:

First, objects in the scene were separated into three categories: one *target object*, objects in a *target group*

[1] https://github.com/DominikKuenkele/MLT_Master-Thesis_clevr-dataset-gen

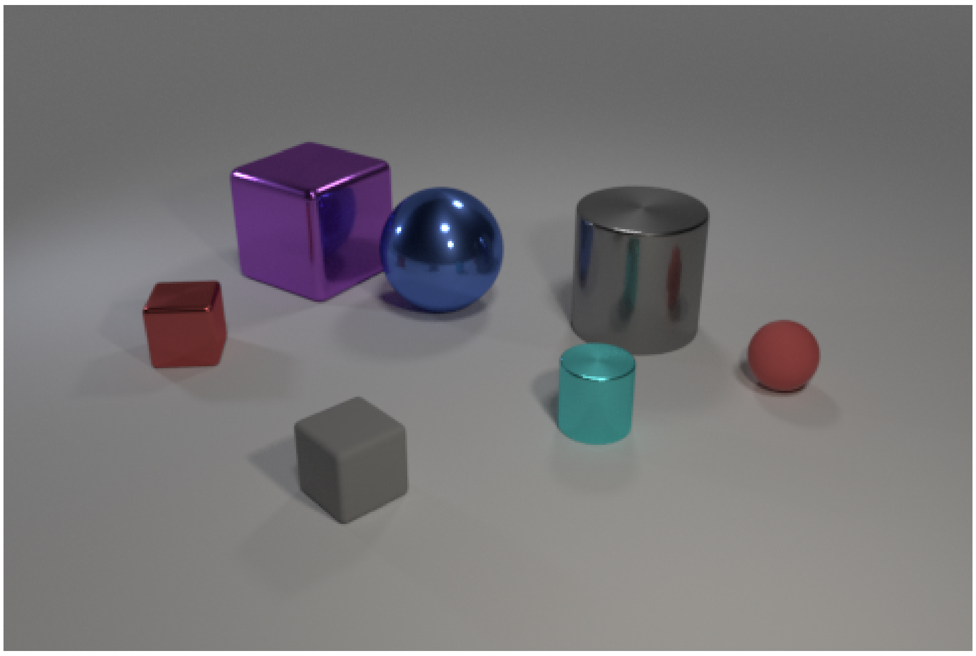

Figure 1: Example of a generated image in the CLEVR dataset

and *distractor* objects. The target object is the main object in the scene that should be identified and communicated by the agents. All other objects and their relations are based on this target object. The target group contains similar objects to the target object. These are objects that the agents need to discriminate the target object from. Finally, the distractors are objects that add noise to the scene and should make it more complex. They expected to teach the agents more precise descriptions of the target object. The number of the objects in both group can be controlled.

In a second step, when generating the images it is possible to define the relations between *target object/target group* and *target object/distractors*. The relation is defined as **how many** attributes of the target object are identical with the attributes of a single object in the target group and distractors respectively. For example the target object is a *small red cube*. If two attributes are shared between target object and target group, objects in the target group could include *small blue cube*, *big red cube* or *small red sphere*, but couldn't include another *small red cube* or a *small blue cylinder*. The number of shared attributes can also be set to a range to make the discrimination task more challenging.

Lastly, it is also possible to define exactly **which** attributes should be shared between the target object and the groups. For example, it can be defined to have the same size for objects in the target group, but have different, randomly selected shapes and colors. This allows for a very controlled generation of relations between the objects in the scene. Figure 2 shows one generated image with this extended source code. Here, the target object is the large purple cylinder. The target group contains four objects that share zero to a maximum of two attributes. It is not controlled, which attributes are shared (they are selected randomly). The large purple cylinder shares the same color and size with the large purple spherem, the same size with both cubes and no attribute with the small turquoise sphere. There are no distractor objects.

For all generated datasets in the following sections, the general constraints and settings are as close as possible to the original CLEVR dataset. The size of the generated images is 480x320 pixels. 10.000 images are created for each of the datasets. Each image contains a maximum of 10 objects, that are not intersecting, have the same minimum distance between objects and are partially visible from the camera.

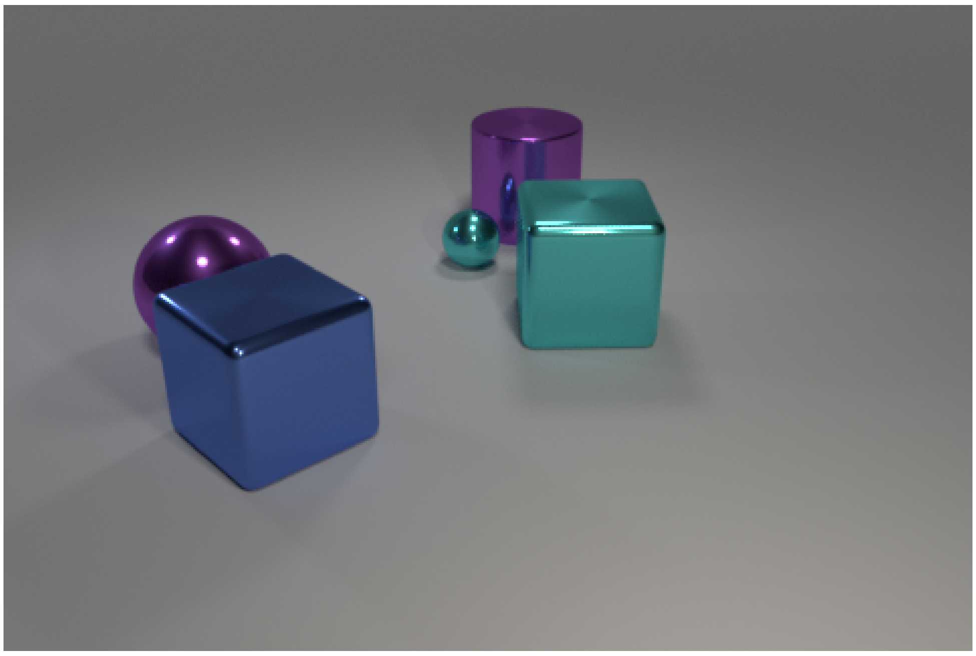

Figure 2: Example of a generated image with the extended code

3.2 Description of experiments

Language games are very complex setups for machine learning models. The models need to solve multiple tasks at the same time to solve the overall problem. For instance, in a simple setup of a game two agents are involved. The first agent, the sender, is shown a scene with objects and needs to communicate one target object to the other agent, the receiver. The receiver is shown the same scene and needs to identify the target object with respect to the message of the sender. In this case, the sender first needs to learn to encode the scene, all objects and their attributes, as well as the information about the target object into its own space. In a next step it needs to learn how to translate this encoding into a message that is sent to the receiver. The receiver then needs to learn to decode this message, after which it needs to learn how to combine the decoded message with its own encoding of the scene and objects. And finally it needs to learn how to identify the target object with this information.

[Dominik Künkele] ?

For this reason, I decided to divide the main problem and let the models learn simpler subtasks and increase the complexity step by step. [2] This will give me a very detailed overview, where the models struggle to learn and in which ways they can be improved. Mainly, I separated the tasks into language games with two agents (final experiments) and classical machine learning tasks without any communication, namely only one 'agent' that solves the task alone (pre-experiments). With this division, I can analyze the learning of the encodings of the scenes separately from the learning of producing and decoding messages.

3.2.1 Feature extractors

To extract features from the images, I make use of different deep neural network architectures that are developed for this task. First, I use the VGG19 (Simonyan & Zisserman (2015)) which is an architecture based on many convolutional layers. After the convolutional layers, the data is passed first through an average pooling layer which outputs 512x7x7 dimensions. Next follow three linear layers with *ReLU* non-linearities in between. After flattening the input, these classification layers output 4069, 4069 and 1000 dimensions

[Dominik Künkele] why ResNet/VGG

[2] https://github.com/DominikKuenkele/MLT_Master-Thesis

respectively.

Secondly, I include the ResNet-101 (He et al. (2016)) that tries to overcome problems of very deep networks, using residual blocks. There are four blocks that output 256x56x56, 512x28x28, 1024x14x14 and 2048x1x1 dimensions. A following average pooling layer outputs 2048x1x1 dimensions as well. The final linear layer reduces the flattened data to 1000 dimensions, corresponding to the ImageNet classes.

[Dominik Künkele] needs to be explained deeper?

Both architectures are available pretrained on an image classification task on the ImageNet dataset. Since the task in this research is very different from a classification, multiple different adaptions of these architectures are compared. Table 2 lists the different adaptions for both VGG19 and ResNet-101 that will be used in this research. In the later chapters, it will be referred to these adaptions using the name in the table.

[Dominik Künkele] deeper discussion about different layers of these networks

	description	**output dimensions**
VGG-0	contains only the convolutional layers	512x7x7
VGG-avg	contains an additional average pooling layer	512x7x7
VGG-cls1	contains an additional one classification layer, including its non-linearity	4069
VGG-cls2	contains another additional classification layer, including its non-linearity	4069
VGG-cls3	the original VGG19 architecture	1000
ResNet-1	contains one residual block	256x56x56
ResNet-2	contains two residual blocks	512x28x28
ResNet-3	contains three residual blocks	1024x14x14
ResNet-4	contains four residual blocks	2048x1x1
ResNet-avg	contains an additional average pooling layer	2048x1x1
ResNet-cls	the original ResNet-101 architecture	1000

Table 2: Different adaptions of VGG19 and ResNet-101 used in this research

Furthermore, I experimented with both the pretrained models as well as with the architectures trained from scratch with a random initialization for the weights. This reason for this was to test if the success of an experiment was actually making use of the pretrained knowledge incorporated in the models. If that was not the case, the agents were likely not using image features, but instead relying on some other underlying patterns to solve the task. Basically, this approach works as an indicator to determine the actual success of the agents aside from measures as the accuracy or precision.

3.2.2 Pre-experiments without language games

The final objective of this thesis is to find out, how agents can communicate about relations of objects spatially as well as based on their attributes. Because of that, the first experiments focus on extracting information from images and combining them with structured knowledge. Here, I structured the experiments into three levels. In the first level, the model are trained to learn the position of objects in the image and attend to specific regions of the image. In the second level, the models are trained to differentiate objects in the scene from each other. The last level adds language to the experiment, more specifically the models learn to caption and describe objects in the image. These combined experiments should lay the basis for how to build up the agents in the language games.

Level 1 - coordinate predictor: This level should help to analyze, how the final task of the language game should look like, in especially what the receiver is tasked to predict. As described before, the sender should communicate an object in the image and the receiver needs to identify it. The challenge lies in how the receiver refers to the identified object. There are multiple possibilities, how it can be done. One of them could be to describe the target object with human language, using the attributes. The main goal however is to let the language of the agents emerge as natural as possible. Including human knowledge into the task would bias also the emerged language towards attributes and words, humans are using. For this reason, the final task of the receiver will be to 'point' to the target object. Since also positional information of the objects should be part of the discrimination, the image will not be separated into several bounding boxes around the objects, but instead the whole image is the basis. The models are therefore tasked to predict the center coordinates of the target object. With this approach, the models receive few human knowledge, but are still able to rely on all information present in the image to discriminate the objects.

To achieve this goal, multiple setups of models are tested. In the simplest setup, the model receives only the image as an input and produces two numbers as an output, the predicted x- and y-coordinate of the target object. Here the image is first passed through one of the *feature extractors*. Next, the extracted feature vector is flattened and passed through two linear layers with a *ReLU* non-linearity between. These reduce the dimensions first to 1024 and finally to 2.

[Dominik Künkele] preprocess image

To determine the loss, the euclidean distance between the resulting predicted point on the image and the ground truth point are calculated. This distance is learned to be minimized. By doing that, the model learns to focus and attend on a specific part in the image, in a perfect model the center of the target object.

With this simple setup, the model is in theory able to focus on an object in the image. The problem arises as soon as multiple objects are present in the image. There is no information available for the model to understand which one of these objects is the actual target object, except for the final calculation of the loss. Since there is not necessarily a pattern for which object in the image is the correct target object over the whole dataset, the models will likely fail to generalize. Therefore, the models need to receive more information. Here, I try out four different ways to encode the target object.

[Dominik Künkele] go deeper, why

In the first method, I encode the attributes of the target object as **one-hot encodings**. There is a three-dimensional vector encoding the *shape*, an eight-dimensional vector encoding the color and a two-dimensional vector encoding the two different sizes. The dimensions of these vectors can either be zero or one, depending on the attributes of the target object. These three encodings of the attributes are then concatenated. The image is again passed through a feature extractor, before the flattened vector is reduced to 2048 dimensions with a linear layer. The result is concatenated with the encoded attributes and passed to a final linear layer to predict the coordinates.

In an extension to this method, I also include the **center coordinates of all objects** in the image. This should help the model to identify all possible options to chose from, when predicting the target object. All the center coordinates are simply extracted and shuffled. Since there are varying numbers of objects in the image, this vector of variable length is padded to 10 objects, resulting in a vector with a length of 20. The padding objects have zeros for both coordinates, which no other real object can have. For this model, I also made use of a more complex way to encode the image. Here, I based the approach on code that implements baselines for the CLEVR dataset (Johnson et al. (2017)). The image is first passed through the feature extractor. Afterwards, a 2-dimensional convolutional layer, reduces the channels from 2048 to 512 channels with a kernel size of 1. After applying the *ReLU* function, the resulting matrix is max pooled over two dimensions with both a kernel size and a stride of 2. The resulting matrix is flattened and concatenated with both the attribute encodings and the shuffled coordinates of all objects. This is then passed again through a final linear layer to predict x- and y-coordinates.

[Dominik Künkele] why?

The third method encodes the attributes of the target object with human language using the **incremental algorithm for the Generation of Referring Expressions (GRE)** described in Dale & Reiter (1995). This opposes the idea described before, to share as few human knowledge as possible with the model. Still, this approach can help to understand and analyze if the model was able to extract information about the objects and more specifically their attributes from the image. If the model is able to match parts of the image with human words it would show that the model learned this attribute. If the model in a next step can learn this over the whole dataset, this would mean that it could generalize over these attributes and assign them to certain regions in an image. This insight could help then to design another model that makes use of these learnings without human language.

Using the algorithm, one can describe an object using its attributes to discriminate it from other objects as efficiently as possible. In other words, the object is described unambiguously using the lowest number of words. The algorithm assumes that there is an order of importance for attributes, such as shape, color and size. This order defines, which attributes can be left, while still identifying the object uniquely. For our research, we rely on the following order from most important to least important: shape, color, size Given for example the scene from 2 with the target object being the *big purple cylinder*. Using all three attributes, this description identifies the object perfectly and uniquely. Following the algorithm, we could make the description shorter by removing the least important attribute *size* without loosing unambiguity, describing it as the *purple cylinder*. This can be taken even one step further by removing also the *color*. Describing it as the *cylinder* still doesn't describe any other object, since the target object is the only cylinder in the scene.

For the experiments, each every image is captioned, by a description of the target object using the described algorithm. To use it in the model, the caption needs to be padded to an equal length, in this case to a length of 3, which is the maximum number of attributes that can be used. For this, as standard practice in captioning tasks, the captions are padded at the end with a specified padding token.

The model is made up of three parts: The first part extracts features from the image. Here, the setup is similar to the previous model. The image is passed through a feature extractor, before it is passed through two 2d convolutional layers, both reducing the channels to 128 with a kernel size of 1. A *ReLU* function is applied, after both convolutional layers. As before, the resulting vector is pooled, using max pooling with both a kernel size and stride of 2. In the second part, the caption is encoded, using an LSTM. Here, the learned embeddings of each token are parsed by the LSTM and its final hidden state is then used as a summary of the complete caption. The third part is again the predictor of the coordinates of the target object. Both the processed image and the final hidden state of the LSTM are flattened and concatenated. The resulting vector is passed through three linear layers reducing to 1024, 1024 and 2 dimensions respectively. Between each linear layer, the *ReLU* function is applied. This architecture is also inspired by the baselines described in Johnson et al. (2017).

The forth method, to encode the target object utilizes **masking** of the image. For this, the image is separated into a fixed size squared area containing the target object and the rest of the image. The side of the square is always two fifth of the image width, in this case 192 pixels. This size always includes the whole object if it is large or small, while not being cut of if the target object is too close to the border of the image. The square is filled in white, while the rest of the image is filled in black. This approach has the advantage of providing as few as possible human bias to the model. While even the one-hot encodings contain human knowledge by explicitly encoding human chosen attributes, masking the image will only point the model towards the target object without giving more information. It therefore can only rely on its own extracted visual features when looking at masked images.

The model is presented with both the original image and the masked image. Both are passed through a feature extractor and afterwards passed through a linear layer, reducing the dimensions to 2048. The resulting vectors are concatenated and passed through another linear layer that predicts the coordinates.

[Dominik Künkele] can just rely on masked image when predicting coords, more useful for cap-

Level 2 - bounding box classifier: One problem that arises, when only predicting the coordinates is that the models may not be able to predict locations of objects from just visual features. Feature extractors like VGG or ResNet are trained on extracting visual information. In this process, absolute locations of these features in the images may get lost in the resulting vectors, since they are not important for this task. To address this challenge, I restructured the task in this experiment. Instead of predicting two coordinates of a target object, the model now needs to classify between all available objects. This will take all positional information out and just focus on the attributes and discriminating factors between the objects. To do this, fixed-size bounding boxes are extracted around each of the object in the image. As for the masking, the bounding boxes are a square of 192x192 pixels, to capture the complete object. The bounding boxes of all objects are padded to a number of 10 bounding boxes, the maximum number of objects in an image with a matrix of zeros. These bounding boxes are shuffled.

[Dominik Künkele] pre-process bounding box

The model passes each of the bounding boxes through a feature extractor. The resulting vectors are concatenated, which basically is a concatenation of the extracted features of each object. This long vector is passed through a linear layer with 4096 with a subsequent *ReLU* function. Afterwards, a linear layer reduces the dimensions to 10, corresponding to the 10 bounding boxes in the input. With a final *softmax* function, the model then points to one of the bounding boxes as the target bounding box. The loss is calculated using cross entropy.

Level 3 - caption generator: As the bounding box classifier, the caption generator only acts as a learning step towards understanding, how the model can learn the attributes of objects. It is used the same way to create the captions for each image as in the section above with the GRE-algorithm. This method focuses even more on human knowledge and structure of language than the experiment above, since it is even the final task of the model. There are some minor additions concerning the padding of the caption. As before, the caption is padded to a number of three tokens, corresponding to the maximum of three attributes.

However, there are three different ways how the padding is applied. First, the captions are, as usual in captioning tasks padded at the end with a specified padding token. A problem could arise when the caption is not viewed as a natural language sentence, but as slots filled with tokens. More specifically, following the GRE-algorithm, the last token in the caption is always the shape. The second last token if existing describes the color, while the third last token if existing describes the size. As soon as this sequence is padded at the end, these slots suddenly disappear. A caption that only describes the shape, such as *cube* will be padded to *cube <pad> <pad>*, where the third last slot is filled up with the shape instead of the size. Since in this task we are not focussing on producing natural language with a correct grammar, but focus instead of extracting attributes, having a slot structure could help the model to express the extracted attributes correctly. For this reason, the second method of padding the caption is prepending the caption with padding tokens. By this, the positions of the slots are preserved and if not specified just filled with a padding token. The last variation concerns the order of producing each token. When the captions are prepended, the model would need first produce two padding tokens, before it finally can produce a much more meaningful token for the shape. This could be difficult to learn for a model, as the longer a sequence of tokens is, the more information about the beginning of the sequence gets lost. Even though a sequence of just three tokens may not be long enough for this factor to be a problem, I experimented to reverse the caption. Instead of producing for instance *<pad> green sphere* as correct in English, the model would now need to produce *sphere green <pad>*. Notice that the padding token is again at the end of the generation, but the order of slots as well as the amount of information in the caption are still preserved.

Nonetheless, this helps to understand more detailed if and how the model discriminates objects. Does it rely on the same attributes, humans are using, or does it find other important differences, or is it able to solve the task at all?

I compare two different models against each other. The first model just takes the image as the input. During training, also the ground truth caption is passed to the model. The image is processed as in the section before. First, it is passed through a feature extractor with two following 2d convolutional layers, reducing the channels to 128. After each convolutional layer the *ReLU* function is applied. The result is pooled using max pooling and then reduced to 1024 dimension using a linear layer. The such encoded image serves as the initial hidden state of an LSTM, which generates the caption. During training the input sequence for the LSTM at each time step are embeddings of the tokens in the ground truth caption. No teacher forcing is applied. The output of the LSTM is passed through a linear layer at each step to determine logits over the symbols of the vocabulary. The loss is calculated using cross entropy. When testing, the LSTM always generates three tokens, with a start-of-sequence token as first input to the LSTM. Each token in the sequence is determined, by selecting the highest logit in the output of each step in the LSTM.

Since with this approach, the model doesn't have any information about the target object, I extend it with a masked image. As in the section above, a masked version of the image is created and passed to the model. The model works exactly the same, except for the additional separate image encoder that encodes the masked image in the same way as the original image. Both encodings are concatenated and then used as the initial hidden state for the LSTM. This should point the model to which object to describe and discriminate from the other objects.

To test the success of the model, three measures are calculated. The first measure is the **accuracy** if the model predicted every word in the caption correctly. This gives a hint, how the model fares in general and also able to predict actually any of the attributes at all. However, a 'false' prediction doesn't give much insight into why the model predicted a wrong caption.

It could be the case that the model predicted the correct shape, but wrong color. Even worse, the model could have predicted more attributes than necessarily to uniquely identify the target object and didn't follow the rules of the GRE-algorithm. For instance, consider the scene in Figure 2. The correct caption is *cylinder*. If the model would predict *purple cylinder*, the accuracy determines it as false as captioning *large purple cylinder* as well *small green cube*. The first two descriptions identify the target object perfectly, but the model only didn't learn yet to leave unnecessary attributes out. To mitigate this, I also include a **word-by-word accuracy**. This measure calculates the accuracy not on sentence level, but on word level. The first predicted caption of the model would yield a word-by-word accuracy of 66% (including padding tokens), the second 33%, while the third prediction would yield 0%. This can give a better understanding of the errors the model makes.

With the **non-target accuracy**, I identify if the model described another object, which is not the target image. This is basically an inverted accuracy score; the lower the score, the better the model fares. For this I generated captions for all the non-target objects and distractors in the images using the GRE-algorithm. If the generated description of the model describes an object that is not the target object, it gets assigned 100%. If not, independently of describing the target object, no object, or one of the objects insufficiently, it gets assigned 0%. Using this measure, I can get a quick overview if the model's problem lies in extracting and relating attributes or in understanding which of the presented objects is the target object.

3.2.3 Language games

4 Results and Discussion

[Dominik Künkele] 15 pages

5 Ethical Considerations

[Dominik Künkele] 1 pages

[Dominik Künkele] 1 big model vs. many task specific models

6 Critiques and Limitations

[Dominik Künkele] 3 pages

[Dominik Künkele] combine with future work

7 Future Work

[Dominik Künkele] 2 pages

8 Conclusion

[Dominik Künkele] 2 pages

References

Dale, R. & Reiter, E. (1995). Computational interpretations of the gricean maxims in the generation of referring expressions.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 770–778).

Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., & Girshick, R. (2016). Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. *CoRR*, abs/1612.06890.

Johnson, J., Hariharan, B., van der Maaten, L., Hoffman, J., Fei-Fei, L., Zitnick, C. L., & Girshick, R. (2017). Inferring and executing programs for visual reasoning. In *ICCV*.

Simonyan, K. & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In Y. Bengio & Y. LeCun (Eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

A Resources