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

The center of this thesis evolves around language games between deep neural models. Hereby, multiple deep neural agents need to solve a task, by communicating with symbols, which initially are not associated with any meaning. By using and interpreting these symbols in there communication, the agents start to give these symbols and their combinations meaning. After this process, a new artificial language emerged, with which the agents can communicate with each other.

In section 2.1.1, I will explain reasons, why research in this field is conducted and how it may help in further research. Section 2.1.2 shows in more detail, how the language games are set up and how exactly agents can communicate and the language emerges. In section 2.1.3, I will discuss, how the emerged languages can be built up. Finally, section 2.1.4 discusses, how the language games will be used in this thesis, to explore how agents can refer to objects in images.

2.1.1 Aims of emerged languages

The setup of language games allows for very controlled rules of how agents can behave.

2.1.2 Setup of language games

The term of 'language games' was first introduced by Wittgenstein (1953). The author describes language games as uses of language between multiple interlocutors. These can be any small parts of conversations, for instance between a teacher and a student teaching a new concept or between two persons, discussing a topic. The rules in each of these situations differ and therefore the meanings and semantics of words and sentences may differ from situation to situation. An interjection 'Water!' may be a warning, an answer to a question, a request or something else, depending on the context. The meanings are bound to the rules of the language games.

This reasoning was taken up, when trying to train artificial entities to produce a language. One of the original games is the signaling game, proposed by Lewis (1969). In this game, a sender needs to send a message to a receiver, based on information only available to the sender. Based on the message, the receiver proposes an action. Both sender and receiver are rewarded in the same way if the proposed action was correct. Hence, both agents need to invent a language together, fit to the conditions of the game.

The entities can be set into real or simulated interactive situation, a language game, that meaning and a language can emerge (Kirby (2002)). Multiple deep neural models, called agents, communicate with a set of symbols to solve a task. Guided by the task and the rules, how agents can generate and interpret symbols, a language can emerge.

2.1.3 Properties of the emerged language

2.1.4 Referring to objects and grounding

2.2 Dataset

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

2.3 Computational methods

3 Methodology

3.1 CLEVR 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

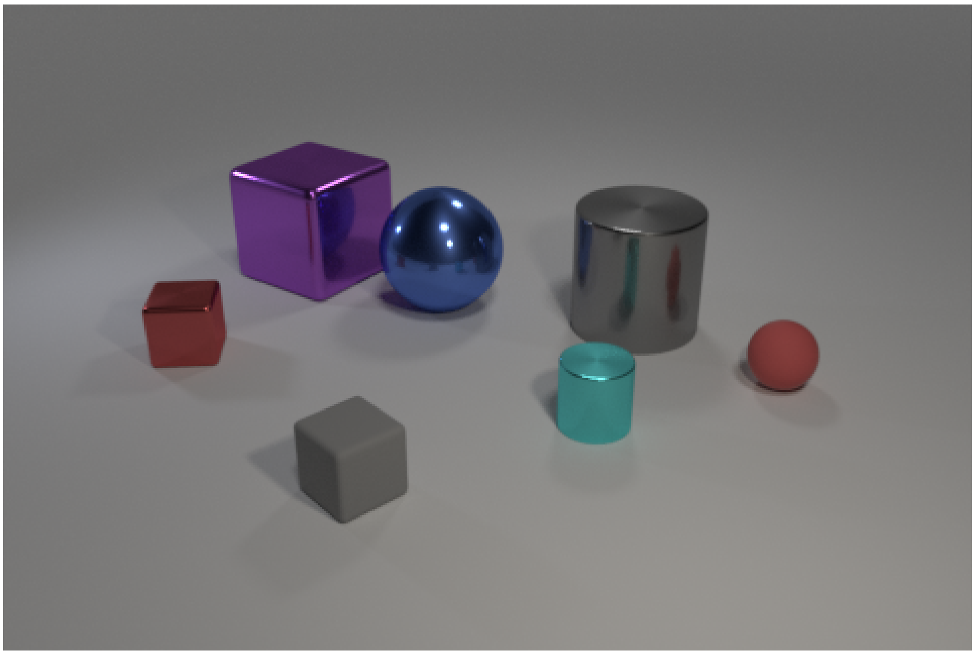


Figure 1: Example of a generated image in the CLEVR dataset

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.

3.2 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 respectively.

[Dominik Künkele] why ResNet/VGG

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.

3.3 EGG

3.4 Ethical considerations

4 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, which datasets the model is trained on, as well as the final architecture of the models.

[Dominik Künkele] ?

4.1 Creation of the dataset

[Dominik Künkele] ambiguity

[Dominik Künkele] referring expressions

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* 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.

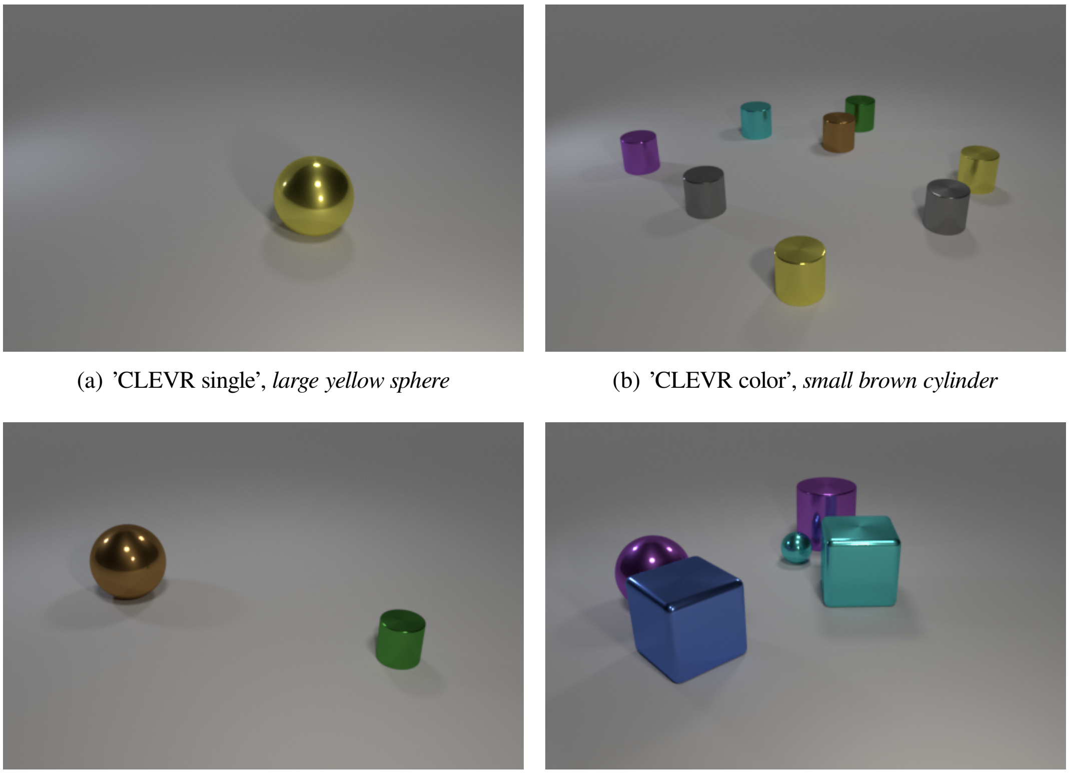
In a second step, when generating the images it is possible to define the relations between *target object/target group* and *target object/distractors*. The relation is defined as **how many** attributes of the target object are identical with the attributes of a single object in the target group and 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(d) 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 sphere, 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

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

intersecting, have the same minimum distance between objects and are partially visible from the camera.



(a) 'CLEVR single', *large yellow sphere*

(b) 'CLEVR color', *small brown cylinder*

(c) 'Dale-2', *small green cylinder*

(d) 'Dale-5', *large purple cylinder*

Figure 2: Example images of each dataset, with the target object specified

CLEVR single

The simplest dataset is called 'CLEVR single'. This is a very simple dataset and has the purpose to simplify the problem the model needs to learn as much as possible. Each scene in the dataset contains only one single object, the target object. There are neither objects in the target nor in the distractor group. All attributes are assigned randomly to the target object. The differences across the whole dataset are the locations and rotations of the objects. With this dataset, neural models can focus on only the features, as well as the locations of this single object. There are no objects that distract the model from extracting features from the target object. This helps to understand if the models are actually able to assign features or learn locations of these features in an image. Figure 2(a) shows an example with the only object being the *large yellow sphere*.

CLEVR color

This task could on the other hand be difficult to learn for neural models. Although the target object is unique in the scene with resect to the distractors, the description isn't. Considering all unordered combinations of the three attributes and a composite referring expression, there are $\binom{3}{3} + \binom{3}{2} + \binom{3}{1} = 7$ possible descriptions of the target object. This can propose a difficulty, when the models try to converge towards one representation. For this reason, I created the 'CLEVR color' dataset. It has a similar setup as the previous dataset, containing between 6 and 9 distractors. But instead of the target object sharing no attribute with the distractors, they share exactly two attributes. Furthermore, to simplify the relation between target object and distractors over the whole dataset, it is also controlled which attributes are shared. The distractors have always the same size and shape as the target object, but the color is always different. The reason for choosing the color as the only discriminating attribute is that it is assumed that the color is easier to learn for neural models as opposed to for instance abstract shapes.

As seen in Figure 2(b), the *small brown cylinder* is unique. The number of possible descriptions is lowered to three, since the color *brown* needs to be part of it. Notice as well that this restriction doesn't apply to the distractors, where multiple objects with the same color are allowed.

CLEVR Dale datasets

Both of the above described datasets are very restrictive on the disambiguating relations between the target object and the rest of the objects. Either no attribute or only one attribute is shared. More realistically would be if objects can share a variable number of attributes. In real situations, there is no restriction at all, how objects or things relate to each other. Natural language emerged that can refer to distinct attributes of these objects to discriminate them from each other. This emergence of referring attributes and their combination is studied deeper in this work.

For this, I created a dataset that allows almost any relation between a target object and the distractors. The relations are only restricted by the incremental algorithm for the Generation of Referring Expressions (GRE) described in Dale & Reiter (1995). This algorithm ensures that every scene contains a unique object in respect to its and the distractors' attributes. Using the algorithm, one can refer to an object using its attributes to discriminate it from all other objects as efficiently as possible. In other words, the object is described unambiguously using the lowest number of words. For the dataset that means that zero, one or two attributes can be shared between the target object and distractor objects. This ensures the uniqueness of the target object. On the other side, it is not controlled which attributes are shared. These are assigned randomly. There is again no control over the relations between distractors, which means that distractors can appear multiple times.

Two datasets, following these rules are created. The Dale-2 dataset contains one target object and one distractor (see Figure 2(c)), while the Dale-5 dataset contains one target object and exactly four distractors. Consider Figure 2(d), with the target object being the *large purple cylinder*. The large purple sphere shares the size and color, the two cubes only share the size, and the small turquoise sphere doesn't share any attribute.

These two datasets allow a more realistic look in how models can acquire knowledge about attributes of objects. More specifically how models learn to discriminate objects from each other, since the model may only need to learn discriminative features of objects and not all features of the whole object.

[Dominik Künkele] probabilities of shared attributes for both Dale-2 and Dale-5

4.2 Generating and understanding referring expressions

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.

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.

4.2.1 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

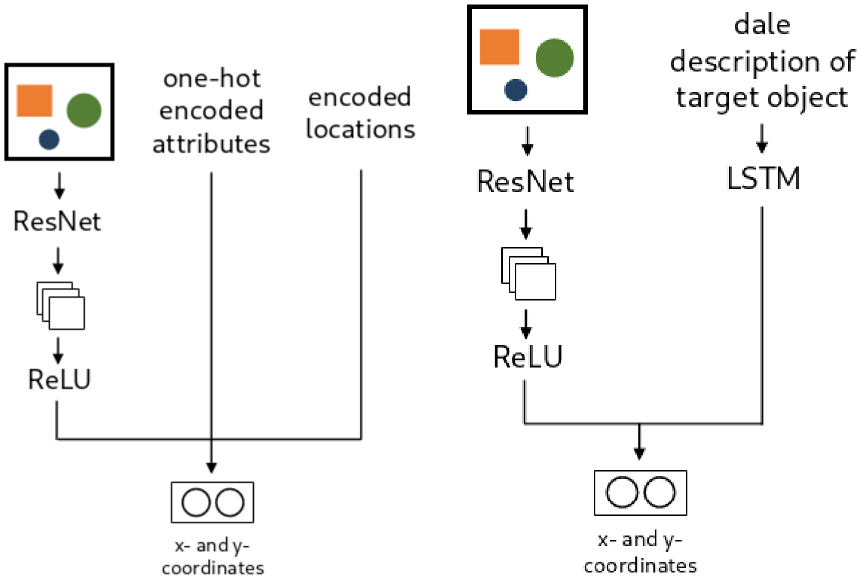
To determine the loss, the euclidean distance between the resulting predicted point on the image and the ground truth point are calculated. This distance is learned to be minimized. By doing that, the model learns to focus and attend on a specific part in the image, in a perfect model the center of the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

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



(a) Model including one-hot encoded attributes and locations

(b) Model including GRE description

Figure 3: Simplified architecture of the coordinate predictors

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(d) 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.

[Dominik Künkele] can just rely on masked image when predicting coords, more useful for captioning or language game

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.

The test dataset is again evaluated on the euclidean distance of the predicted coordinates to the ground truth coordinates. This distance needs to be minimized. The mean of all calculated distances is calculated across the whole epoch, which results in a mean distance score per epoch. Since this score only takes the average of all predictions into account it can't evaluate every prediction individually. If for instance the prediction of one object is getting more precise with growing number of epochs, but the precision of another object gets worse, the mean distance will stay the same. It doesn't reflect this change. For that reason, I also introduced an accuracy score. For that I defined a fixed size circle with a radius of 20 pixels around the center of each object. If the model's prediction lies in this circle, it will be counted as a correct prediction, if it lies outside, it is a false prediction. These scores are averaged for the epoch and result in an accuracy score, where 100%

means that all predictions were very close to the center coordinates and 0% means that no predictions were close to the center coordinates. This of course doesn't give a perfect representation since the size of the objects varies, but it will still show, how precise each individual prediction is. A high accuracy may indicate that the model could identify this specific object better.

The coordinate predictors are trained on the 'CLEVR single' as well as on both 'Dale' datasets. The 'CLEVR single' dataset should test the model if it can actually learn locations of an object. Since the model relies on the extracted features of either VGG or ResNet, locational information about the image could have gone lost. Training on this dataset should make sure that the model can converge towards the correct pixels, utilizing these features. In a next step, the 'Dale' datasets provide the actual problem of discriminating objects from each other and afterwards pointing to the correct one. Here, the models should make use of the additional given information about the scene, as one-hot encodings of the attributes, descriptions using the GRE-algorithm or the encoded locations. 'Dale-2' and 'Dale-5' provide two different difficulties for the model, where it needs to discriminate a target object from one or four distractors. Latter task is assumed to be significantly harder.

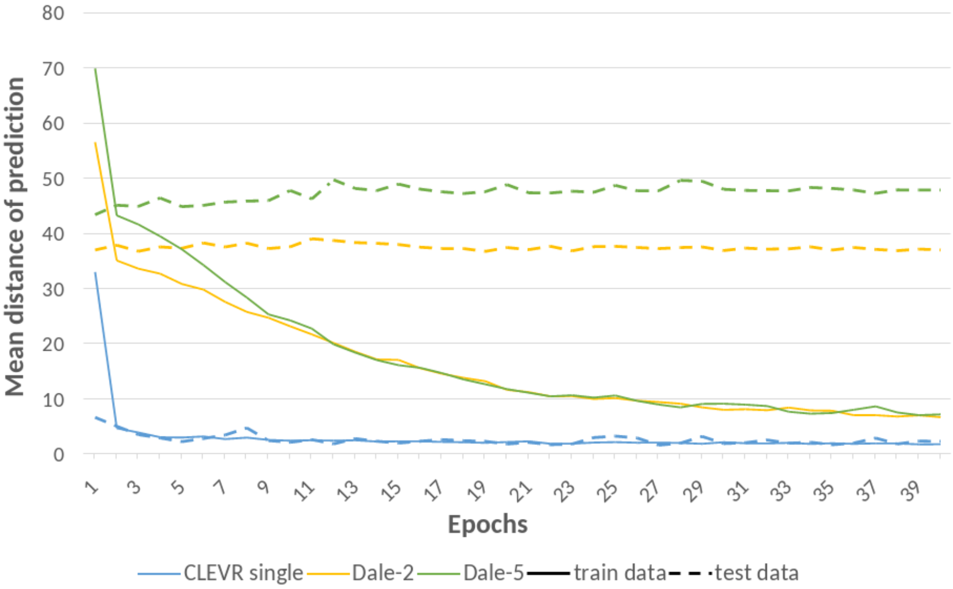


Figure 4: Mean distance between predicted coordinates and ground truth in pixels on different datasets

Results: Figure 4 shows the results of the coordinate predictor that doesn't include any information about the target object. The used feature extractor for these results is *Resnet-3*, but the results don't differ significantly with other feature extractors. As can be seen, the success between the different datasets differ significantly. The more objects are present in an image, the worse the model performs. The model converges for the CLEVR single dataset after around 20 epochs to a mean distance of around 2 pixels. This prediction even though not perfectly on the center point is always on the object. Opposed to that, using the Dale-2 dataset with two objects, the mean distance is from the first epoch between 37 and 38 pixels. With five objects in the Dale-5 dataset, the model only predicts a mean distance of around 45 pixels in the beginning, which worsens with a rising number of epochs to 48 pixels.

An interesting observation is the difference of the mean distances between training and testing data. The training distance is constantly approaching zero, while the testing loss is staying constant or even getting higher. This points to the fact that the model is not generalizing the task and learning abstract patterns that can be applied to unseen data, but instead learning the training data by heart. That is especially visible for the Dale-5 dataset, where learning the patterns of the training data looses even the ability to interpret some patterns in the testing data. Applying a higher dropout didn't have an impact on the results.

This behavior is indeed not very surprising. First, these results show that the model is able to derive geometrical information from abstract feature, extracted by a feature extractor. Geometrical information therefore doesn't get lost during this abstraction, but the model is able to point to a specific object, as long as only one object is part of the image. Secondly, more than one objects present in an image confuse the model, and it is not able to consistently point to one of them. This can have multiple reasons, for instance that models lacks the ability to separate objects in the extracted features. But even given, the model is able to do that and could determine the location of each object in the scene given the feature, it would not be able to tell, which of these is the actual target object. The guess is then more or less random. This especially applies to the Dale-2 dataset, where an identification of the target object just based on one distractor is impossible; both of the present objects are unique. For the Dale-5 dataset, the model could in theory learn that the target object is always the one object which is unique in respect to its and the distractors' attributes. This task on the other hand seems very difficult to learn. In conclusion, the models are able to predict geometrical coordinates, but need more information about the target object to identify it.

When the target object **attributes are encoded as one-hot vectors** and added to the input, the results don't improve. One factor that now has a much higher impact is the feature extractor that is used. Table 3 compares the mean distances the models predict for the different feature extractors. The results are shown for the models trained on the Dale-2 and Dale-5 datasets for training and test data. First, a big difference can be seen between the datasets. The model only converge to a minimum mean distance of around 46 pixels to the correct coordinates for the Dale-5 dataset, looking at the test data. In most cases, it stays above 50 pixels. Using the Dale-2 dataset, the behavior is a little different. All ResNet extractors with four residual blocks and an additional average pooling and optionally a classifier layers reach similar scores as the experiments before without any one-hot encodings. Interestingly, without the classifier layer, the model doesn't converge at all and the mean distances jump up and down between the epochs. This effect also applies when using less residual blocks. Using the VGG, only VGG-cls2 achieve a similar performance, while the others predict coordinates between 43 and 46 pixels away.

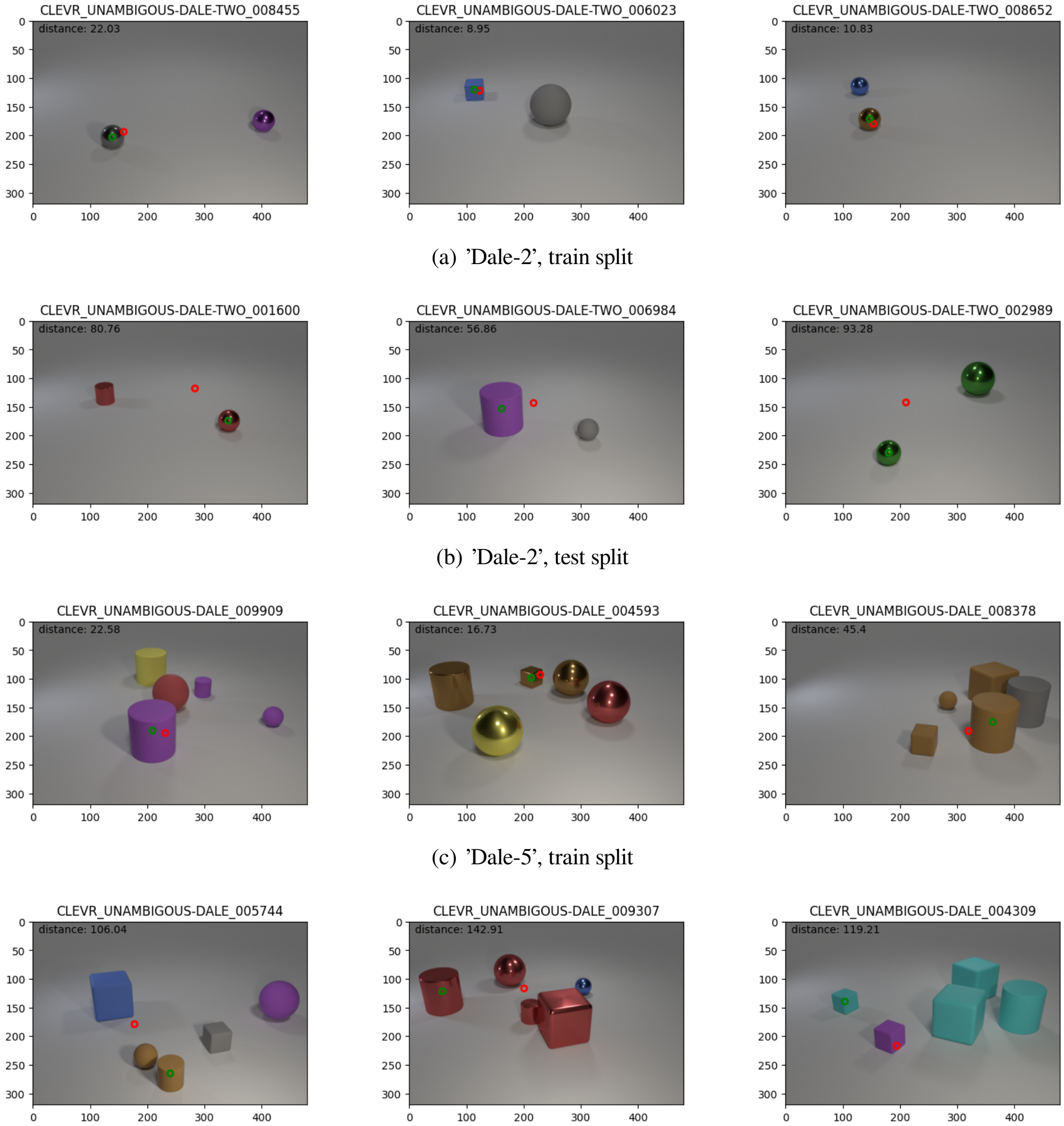
	Dale-2		**Dale-5**	
	train	test	train	test
VGG-0	30,27	46,20	**32,17**	54,40
VGG-avg	**29,99**	45,08	32,32	52,67
VGG-cls1	37,99	43,28	46,57	50,75
VGG-cls2	38,87	**39,02**	47,48	49,91
VGG-cls3	39,99	44,32	47,26	**46,77**
ResNet-3	78,26	65,23	92,07	91,12
ResNet-4	44,14	55,24	**36,48**	58,28
ResNet-avg	**33,06**	**39,18**	47,64	46,38
ResNet-cls	37,57	38,10	44,72	**45,92**

Table 3: Mean test losses for different feature extractors with one-hot attribute encodings after 20 epochs

Secondly, the training loss now looks also different. In almost no cases, the models converge to a lower mean distance than with the test data, meaning a higher precision in their predictions, as they did in the experiment before. The only exception is ResNet-3 as a feature extractor. In other words, the models are again not able to generalize, but in specific cases memorize the patterns in the train data. This hints to the fact that only specific layers of the feature extractors contain information that is generally usable to identify and discriminate objects. Especially the lower layers with fewer residual blocks in the case of ResNet and no classifier layers for the VGG seem to not encode knowledge that can be utilized for this task. Higher layers, with more specific encoded information need to be used for this research. The experiments in the following sections are set up using these higher layers.

Adding **information about the center coordinates** of all objects should have helped the models to get a list of possible predictions. In theory, the model could learn to choose between these coordinates by relating them to the extracted features of the image. This hypothesis doesn't hold. All results for both datasets Dale-2 and Dale-5 are the exact same as without included information about the locations. The problem therefore doesn't seem to lie in predicting coordinates in general, but predicting the coordinates of the target object. The model is still not able to understand, which object is the target object. For that reason, a better representation of the target object is necessary.

In a next step, information about the attributes is included using the ***GRE-algorithm*** from Dale & Reiter (1995). Again, the mean distance of the predictions as well as the accuracy doesn't improve compared to the previous experiments.



(a) 'Dale-2', train split

(b) 'Dale-2', test split

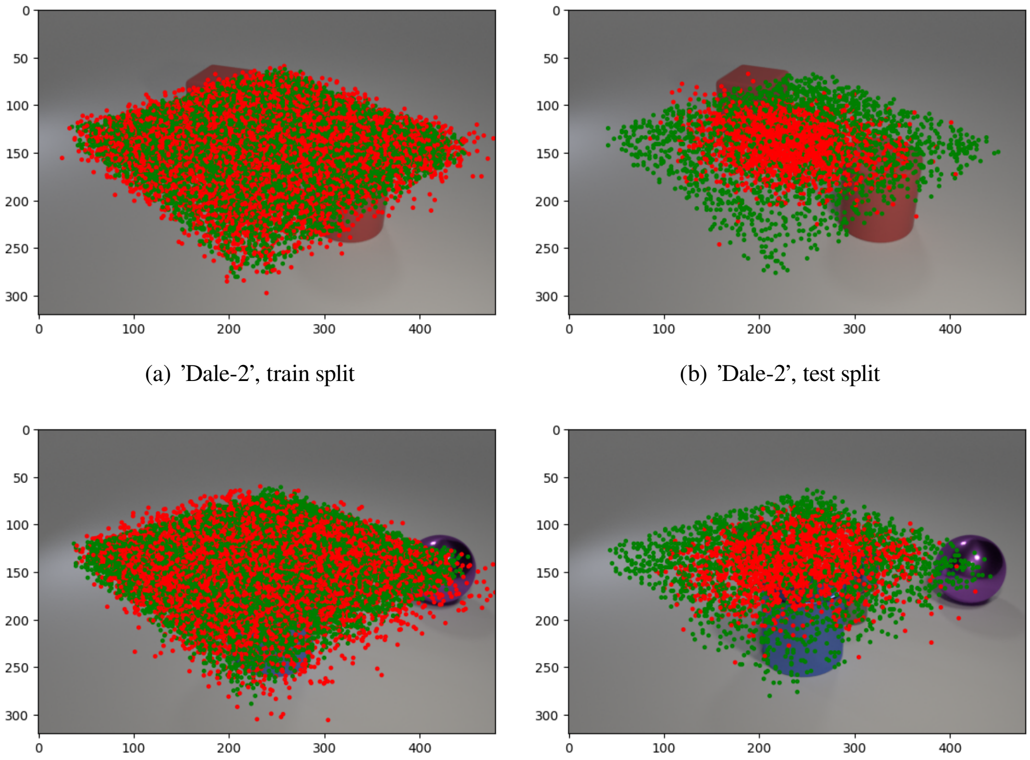
(c) 'Dale-5', train split

(d) 'Dale-5', test split

Figure 5: Visualization of the models' predictions in the 'Dale' datasets

An interesting pattern appears when doing a qualitative analysis of the models' predictions. Here, I visualized the predicted coordinates compared to the ground truth coordinates. Figure 5(a) shows random examples of predictions for images in the train dataset of Dale-2. The green circle shows the ground truth center coordinates of the target object, while the red circle shows the prediction of the model. As can be seen, the prediction are very precise. Figure 6(a) combines the predictions and ground truths across all images in the train dataset. Here, all predicted coordinates are placed as red circles into the image, while all ground truth coordinates are places as green circles. The resulting shape is a rhombus, which reflects that all objects are placed usually central into the scene. As expected the green and red rhombus align mostly in the same area for the train split of the 'Dale-2' dataset.

The results look very different for the test split. As can be seen in Figure 5(b), the three randomly selected predictions don't align with the ground truth coordinates. For all the images, the predictions don't lie on any object. In the left image as well as in the central image, the predictions are closer to the target object than towards the distractor, but are still quite imprecise. These findings align with the mean distance scores, described in the sections before. However, it seems that the model's predictions are all towards the center of the image. This can be seen clearer in Figure 6(b). Again, the green circles form the shape of rhombus. In contrast, the predictions in red almost all cluster in the center of the image. They form roughly the shape of a smaller rhombus. This behavior can be observed for all datasets and architectures of the model. Figures 5(c), 5(d), 6(c) and 6(d) show the results for the 'Dale-5' dataset. Here, the model more likely predicts the center coordinates of a distractor object as seen in the right image, which is also reflected in the lower score of the mean distance. Also the combined visualization shows the same clustering of predictions in the center of the scene, but pattern of the smaller rhombus is better visible.



(a) 'Dale-2', train split

(b) 'Dale-2', test split

(c) 'Dale-5', train split

(d) 'Dale-5', test split

Figure 6: Visualization of the models' predictions in the Dale datasets

These results allow two conclusions. First, the models are biased to predict coordinates in the center of the image. The reason for this is likely that the model can produce a relatively low loss, without relying on many extracted features of the objects. Since all objects are always located in the center of the image and never in its corners, a prediction of any coordinate in the center is on average closer to the target object than any random prediction or predictions of coordinates at the borders of the image. The model therefore learns only, where any object is likely located and can minimize the mean distance to a certain extent with this strategy.

Second, even though the models are biased towards the center of the image, the predictions are still often leaning towards the location where many objects lie. This can be seen for the 'Dale-5' dataset, especially in left and central image in Figure 5(d). Again, by this strategy, the model can minimize the mean distance, since the probability is high that the target object lies in this cluster of objects. Concluding, the model is able to extract, where objects are located in the image, but can't decide, which of these objects is the target object.

4.2.2 Bounding box classifier

[Dominik Künkele] better to call it discriminators?

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 in the experiment before, the bounding boxes are a square of 192x192 pixels, to capture the complete object. The bounding boxes of all objects are padded to the maximum number of object present in an image across the whole dataset with a matrix of zeros. These bounding boxes are shuffled. Furthermore, the attributes of the target object are encoded as on-hot vectors to point the model towards it.

[Dominik Künkele] pre-process bounding box

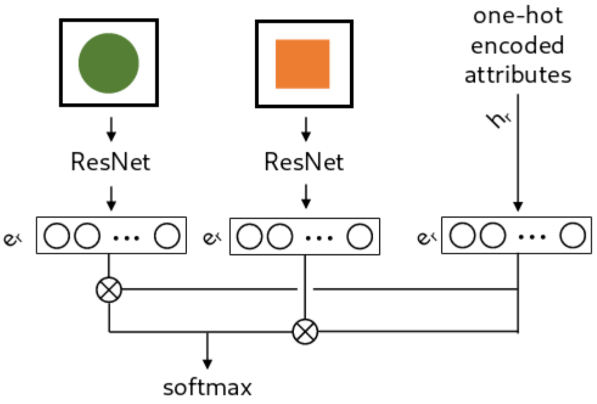


Figure 7: Simplified architecture of the bounding box classifier

The model passes each of the bounding boxes through a linear layer to compress it to an embedding vector of 10 dimensions. The same is done with the one-hot vector of the target attributes. In the next step, the dot product between each embedded image and the embedded attributes are calculated. A high correlation between the attributes and the object should result in a high dot product, while a low correlation results in a lower dot product. The model points then to an object, using the *softmax* function over all concatenated dot products. The loss is calculated using cross entropy.

Furthermore, the model is compared to a baseline that doesn't include any information about the target object. Here, the model passes all flattened bounding box representations through a linear layer resulting in a vector with as many objects as are present in the image. Again, the model uses a *softmax* function to point to one of these objects. Since no information about the target object is included, the guesses are expected to be random.

The bounding box classifier is trained on all datasets excluding the 'CLEVR single'. The 'Dale' datasets are directly comparable to each other for the similar setup of their creation. Especially interesting is the effect of the increasing the number of distractors and the growing number of attributes that are needed to discriminate the objects.

Results: As expected, all baselines achieve a random accuracy for each of the datasets. The accuracy lies at 50% for the 'Dale-2' dataset, 22% for the 'Dale-5' dataset and 12% for the 'CLEVR color' dataset. This changes however, when information about the target object is included. Table 4 lists the accuracies of the models' predictions after 10 epochs.

	Accuracy baseline	**Accuracy**
Dale-2	50%	99%
Dale-5	22%	94%
CLEVR color	12%	93%

Table 4: Results of the bounding box classifier after 10 epochs

The model achieves almost perfect accuracy on the 'Dale-2' dataset with 99%. For both other datasets, the accuracy is slightly lower with 94% and 93% respectively, but still close to perfect, especially compared to the random baseline. First, this shows that the model is able to extract attribute features from the image and manages to combine them with the given information about the target object. This worsens slightly, when more objects are introduced. More objects lead to more needed attributes to discriminate the objects from each other. The model is still able to extract and utilize these additional attributes and combine them with the given attribute encoding of the target object. The model can generate these high results, even though its architecture is very simple. In this architecture, the model doesn't compare the objects directly to each other, but each object is only associated with the attribute encodings. A more complex architecture, in which the model is additionally tasked to discriminate the objects directly from each other might even improve the results.

4.2.3 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?

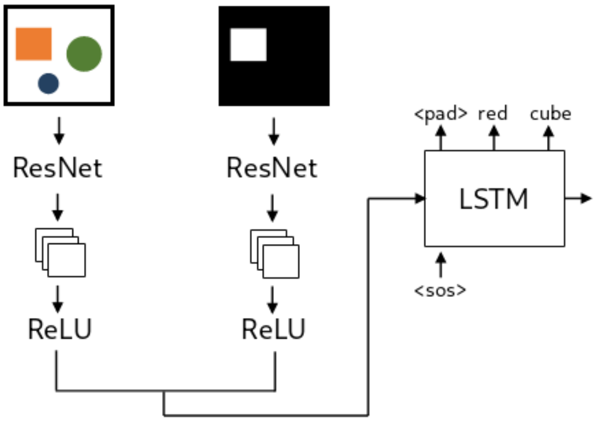


Figure 8: Simplified architecture of the masked caption generator

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. This is shown in Figure 8. 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(d). 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.

The caption generator models are trained on both ’Dale’ datasets. Again, each of these datasets increases the complexity of the description. While the referring expression for the ’Dale-2’ datasets are generally shorter, expressions of the ’Dale-5’ datasets need to be more specific and use more attributes. Furthermore, the model needs to attend to many more locations in the image at the same time to find discriminating factors between those.

[Dominik Künkele] align ’description’, ’caption’ and ’referring expression’ over the whole section

Results: In the next paragraphs, the results of the caption generator will be discussed. Table 5 shows the different scores of the models, when trained on ’Dale-2’ and ’Dale-5’. ResNet-3 produced the best results, when it was used as a feature extractor. The results are shown after 50 epochs, but the scores already start to converge after 20 epochs and changed only very in the following epochs.

	Accuracy	**Word-by-word accuracy**	**Non-target accuracy**
Dale-2	49%	82%	48%
Dale-2 masked	72%	90%	25%
Dale-5	21%	45%	40%
Dale-5 masked	21%	54%	45%

Table 5: Results of the caption generator, based on ResNet-3

The sum of the *accuracy* and the *non-target-accuracy* adds to 97%, when trained on the ’Dale-2’ dataset. This means that the models produce almost always correct descriptions of an object in the image. Moreover, these descriptions are also efficient in the way that they follow the GRE-algorithm of Dale & Reiter (1995) and only use necessary discriminative attributes. Features of both objects are therefore extracted perfectly and related to the vocabulary. As expected, the model has difficulties to decide, which of both objects is the target object, since no information is passed to the model. With 48% of the images describing the target object and 49% of the images describing the distractor, the model uses a random guess. This changes, when also the masked image is presented to the model. Then, the accuracy for the target object is with 72% well above chance. Still, the wrong descriptions describe almost every time the distractor object. This indicates that the problem doesn’t lie in extracting features of objects, but only in deciding which object in the image to describe.

When the model is trained on the ’Dale-5’ dataset, the results are much worse. Now, without masking only in 61% of the images, the model describes any of the objects in the image. With 21% of these descriptions being a description of the target object, the model again uses a random guess. Interestingly, passing the masked image to the model doesn't help it to identify the correct object. The accuracy stays at the same value. In contrast, the non-target accuracy is increased by 5% points. The model is more likely to identify a wrong object. Furthermore, the word-by-word accuracy increases from 45% to 54%. This increase is likely due to the fact that the description of the target object often shares some attributes with distractor objects. For instance an image with a target object being a *small red cube* was identified as a *<pad> <pad> cube* when the model didn't receive the masked image. After also the masked image was passed to the model, it generated the description *small blue cube*. This was the correct description of a distractor and because of the overlap of the attributes, the word-by-word accuracy increased from 33% to 66%.

The approach, how the padding is produced and in which order the attributes are concatenated didn't have an effect on the described metrics. When the order was reversed and the padding appended, it seemed like the model was converging slightly faster and reaching the limit around tow to three epochs earlier. The final peak stayed exactly the same and the effect was therefore not studied deeper if there is an actual significant difference.

There are two main possible explanations for the big difference between the two datasets. First, as already seen in the experiments before, the bigger number of distractors confuses the model more, where to focus on. In the 'Dale-2' dataset, there are only two possibilities, while the four distractors in the 'Dale-5' dataset give the model a bigger choice. The second explanation lies in the used GRE-algorithm. When only two random objects are placed in a scene, the probability that only the shape is enough to discriminate the objects is at 66,6%, namely the caption is only one word long. For shape and color, the probability lies at 29,2% and that all three attributes are necessary is 4,1%. Opposed to that the probabilities with four distractors are 19,8%, 47% and 33,2% respectively. With four distractors the algorithm is much more likely to produce longer captions. These are harder to generate for the model, since it needs to take more attributes into account to discriminate the target object from the distractors.

4.3 Language games

The experiments that are executed using language games have a similar structure as the pre-experiments, since those lied the basis for the language games.

The language games in this research have an asymmetric setup. One agent, the sender is shown some information and needs to generate a message. This message is received by the second agent, the receiver. The receiver needs to parse this message and combine it with the information they are given as well. The receiver then makes a prediction, which is compared to the ground truth. The game is set up prosocial, which means that both agents receive the same loss based on the receiver's prediction. All weights of the agents are adapted in the same way. All language games are set up and run in the EGG framework (Kharitonov et al., 2019), developed in PyTorch. This framework consists of heavy configurable core that controls the generating and parsing of the message, the calculation of the loss and the rules, for how the weights of all neural models are trained. The configuration includes for example a choice between single symbol and sequence messages with varying RNNs, a choice between Gumbel-Softmax relaxation and REINFORCE algorithms to learn neural models containing discrete symbols or an easy switch between different loss functions. Furthermore, runs of games can be saved to analyze the used messages of the agents and how they change over the duration of the learning.

The framework is hereby setup in three levels. Part of the lowest level are the *agents* themselves. The agents are neural models that need to be developed from scratch and define how, the agents process their input and combine it with the message (in case of the receiver) and what they output. The second level are *wrappers* that take care of generating and parsing the message. The sender wrapper uses the output of the sender agent, to produce a message. The receiver on the other hand parses the message received by the sender and passes the result as an additional input to the receiver agent. The third level, the *game* links all described

[Dominik Künkele] Gumbel-Softmax vs REINFORCE

[Dominik Künkele] better in theoretical background?

[Dominik Künkele] vocabulary?

parts together. It provides the agents with the input and passes the message from the sender to the receiver. Furthermore, it uses the output of the receiver and calculates the loss, which is then the basis for the adaption of the weights for both wrappers and agents.

For the following language games, the sender will always produce a sequence of symbols as a message, which the receiver will parse. Gumbel-Softmax relaxation is applied to produce discrete symbols. This is done using two LSTMs, an encoder LSTM in the sender wrapper and a decoder LSTM in the receiver wrapper. The output of the sender agent is used as the initial hidden state for the encoder LSTM. This LSTM is then producing symbols until it generates an end-of-sequence symbol. This sequence is then passed to the receiver wrapper with its decoder LSTM. Its hidden state is initialized randomly. The received message sequence is processed symbol by symbol. After each time, a symbol is processed by the LSTM, the resulting new hidden state is passed to the receiver agent as the parsed message. The receiver agent is combining it with its representation of the image input and predicting an output. The *game* is afterwards calculating a loss for each of these outputs separately. These losses are summed up to a total loss that is used to adapt the weights in both agents as well as in both LSTMs.

[Dominik Künkele] always ResNet-3

First, I will discuss *discrimination games*, because they have the simplest setup. Furthermore, other language games that research this topic use a very similar setup. In the next step, I will look into caption generators that are set up as a language game. Here, the sender describes the scene, while the receiver needs to generate a caption. In the last step, I try to lose as much human bias as possible and the models are trained on just 'pointing' towards the target object, by again predicting its center coordinates.

[Dominik Künkele] look at emerged language: distribution of vocab/ngrams, how do they relate to the images (clustering?)

4.3.1 Discrimination games

In a discrimination game, the agents are presented with two or more images, one of these being the target image. The sender needs to communicate this target image to the receiver by discriminating it from the other distractor images. The receiver then needs to decide based on the message, which of the images is the target image.

The discrimination games in this research have a very similar setup as described in Lazaridou et al. (2016). The agents in this research resemble their *agnostic sender* as well as their *receiver*. One central difference is the production of the message. The main goal of their language game was the identification of the concept that and image was related to. Therefore, the sender communicated only single-symbol messages to the receiver, which should describe the concept of the target image. Opposed to that in this research, the agents are tasked to discriminate objects from each other based on their attributes. It is therefore assumed that the sender will communicate these discriminative attributes. For that reason, the sender is allowed to generate sequences as a message.

Both 'Dale' datasets are used to train the agents in this mode. Similarly to the bounding box classifier, bounding boxes around each of the objects are extracted and fed to the game. As in Lazaridou et al. (2016), the sender receives the bounding box of the target object as the first image, while the rest of the bounding boxes are shuffled. The sender is assumed to learn which of the input images is the target images by this approach. The receiver receives the images in completely shuffled order.

Figure 9 shows, how the sender and the receiver of the discriminator are built up. For the sender, the images are passed through a feature extractor and a following linear layer that reduces the dimensions to an embedding size e_s. All embedded images are concatenated and passed through another linear layer to reduce the dimensions to the hidden size h_s. This is then used as the initial state of the encoder LSTM in the sender wrapper.

The receiver also encodes all images using a feature extractor with a following linear layer, reducing it

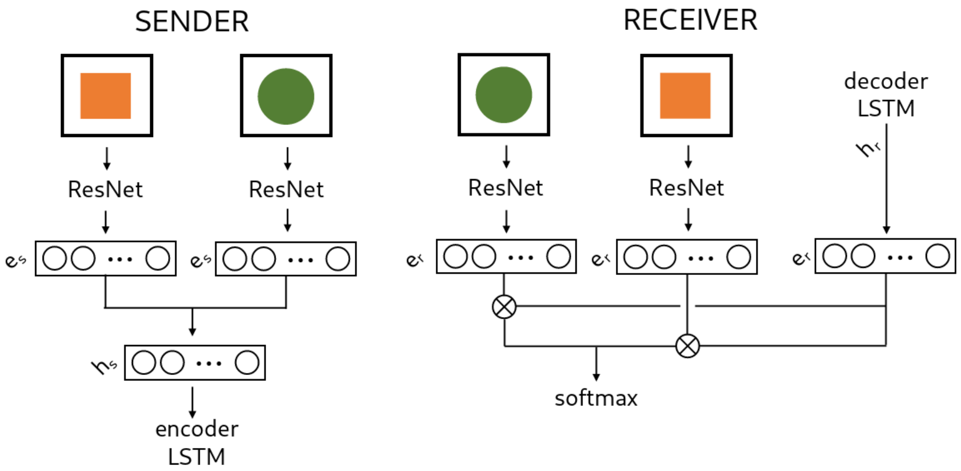


Figure 9: Sender and receiver architectures in the discrimination game

to e_r. The sequence, received by the sender is the input for its decoder LSTM, where the hidden state with a dimension of h_r is randomly initialized. As mentioned above, the receiver combines this image representation with the hidden state of each symbol separately. This is done by passing the hidden state through a linear layer to scale its dimensions to the same e_r. This allows the calculation of the dot product between it and the image representation. If the message describes an object well, the resulting dot product should be higher. The receiver then 'points' to one of the images by applying the *softmax* function over the results of the dot products. The loss is calculated using the NLL-loss.

During the experiments, five variables are adjusted to compare their effects: (1) the image embedding size for the sender e_s, (2) the LSTM hidden size for the sender wrapper h_s, (3) the image/message embedding size for the receiver e_r, (4) the LSTM hidden size for the receiver wrapper h_r and (5) the size of the vocabulary $|V|$.

Results: Table 6 shows the accuracy of the models calculated on the success of communication if the receiver can identify the target object. A random guess corresponds to 50% in the *Dale-2* dataset and 20% in the *Dale-5* dataset. Four different vocabulary sizes $|V|$ are tested. A size of 13 symbols corresponds to the 13 attributes the objects can have and align with human language. If the symbols were similarly used, messages would have lengths between one and three symbols. Opposed to that a slightly smaller vocabulary with 10 symbols is used to create a smaller bottleneck with a higher pressure to condense the information. Similarly, a bigger vocabulary consisting of 20 symbols tests how a bigger bottleneck changes the results. Lastly, a big vocabulary of 100 symbols should give the model all options to encode the information, including one symbol per attribute or one symbol describing a combination of attributes.

The hidden sizes h_s and h_r as well as the embedding sizes e_s and e_r are chosen in alignment with the vocabulary size. The hidden sizes are always smaller or equal to the vocabulary size since the information about each word needs to be compressed in a smaller dimension to learn meaning. Hereby, hidden sizes of 10 and 100 are tested. On the other hand, three different embedding sizes are tested: 10, 50 and 100. The reason for this is to experiment, what is the optimal middle ground between compressing features of an image encoded in high dimension vectors and upscaling encoded messages in low dimension vectors.

For the *Dale-2*, a clear correlation between the hidden sizes, embedding sizes and the size of the vocabulary can be identified. A hidden/embedding size as high as the vocabulary size is beneficial for identifying the correct object. The receiver identifies almost every sample correctly when all sizes are 10. When the hidden and embedding sizes are increased, the guesses by the receiver are random with 50% accuracy. Interestingly,

					Dale-2		**Dale-5**		**CLEVR color**	
$\lvert V\rvert$	h_s	h_r	e_s	e_r	**Accuracy**	**length**	**Accuracy**	**length**	**Accuracy**	**length**
10	10	10	10	10	96,4%	1,99	24,7%	1	17,3%	2
10	50	50	50	50	50%	2	21,4%	2	17,8%	1
13	10	10	10	10	94,9%	2	24,8%	2	17,1%	2
13	10	10	50	50	49,6%	2	21,9%	2	17,9%	1
20	10	10	50	50	50,9%	1	23%	2	15,9%	2
100	10	10	10	10	97,3%	2	24%	2	18,1%	2
100	10	10	50	50	49,9%	2	24,4%	2	15,8%	2
100	100	100	100	100	49%	1	25,3%	2	15,6%	1

Table 6: Results of the discriminators: $|V|$ are different vocabulary sizes, h hidden sizes and e embedding sizes.

a vocabulary size of 10 is enough to communicate a meaningful message when the model is trained on the *Dale-2* dataset.

The results change, when using the *Dale-5* dataset with four distractors. With four distractors and with low hidden, embedding and vocabulary sizes, the agents barely pass the random baseline with 23%. Only increasing the vocabulary size to 100 raises the accuracy by almost 20% points to 43%. This is still considerably lower than the 95% of the *Dale-2* dataset. The same applies to the 'CLEVR color' dataset, where all models achieve a very low accuracy of around 15 to 17%, corresponding to random guesses.

[Dominik Künkele] calculation of loss?

[Dominik Künkele] length

Two conclusions can be drawn. First, the hidden as well as the embedding sizes need to be close to the vocabulary size. This even applies for very low vocabulary sizes, which means that the image encodings need to be compressed to the same low dimensions. The reason for this is very likely that neural models have difficulties to upscale from lower dimensions (e.g. from low h_r to high e_r) as opposed to learn how to extract the important information from a vector with many dimensions.

The second conclusion that can be drawn looks at the differences between the two datasets. Unsurprisingly, the agents have a much higher difficulty to discriminate a target object from four instead of one distractor. Since we discriminate objects based on properties that are also distinguished in human cognition (color, size, shape), we expect that the vocabulary onto which the agents converge reflects these categories and is therefore close to human vocabulary. There are 48 possible combinations of attributes. Still, for Dale-2, a vocabulary size of only 10 is enough for an almost perfect accuracy with two objects. This hints to the fact that the agents don't describe the complete target object, but only rely on discriminative attributes between the objects. The need for a more detailed description of discriminative attributes is higher when more distractors are involved. Therefore, the models need to learn more combinations of symbols in order to attest to this higher level of detail and especially how to relate them to features in the images.

[Dominik Künkele] look at language

4.3.2 Caption generator games

In a next step, it is tested if the agents can learn to extract features of the objects together. For this, the receiver is tasked to describe the target object in natural language, while the sender needs to communicate, which object is the target. Again, the setup is asymmetrical: the sender receives the image and information, which of the objects in the image is the target object in form of a masked image. The receiver only sees the image without additional information.

[Dominik Künkele] similarity to bounding box classifier

[Dominik Künkele] human bias vs emergent

This setup is based on the single neural model described above. The target caption for each image is created using the GRE-algorithm of Dale & Reiter (1995). Since the results of the experiments show that the position of the padding doesn't have an effect on the final converging scores, the padding tokens are only prepended to the caption.

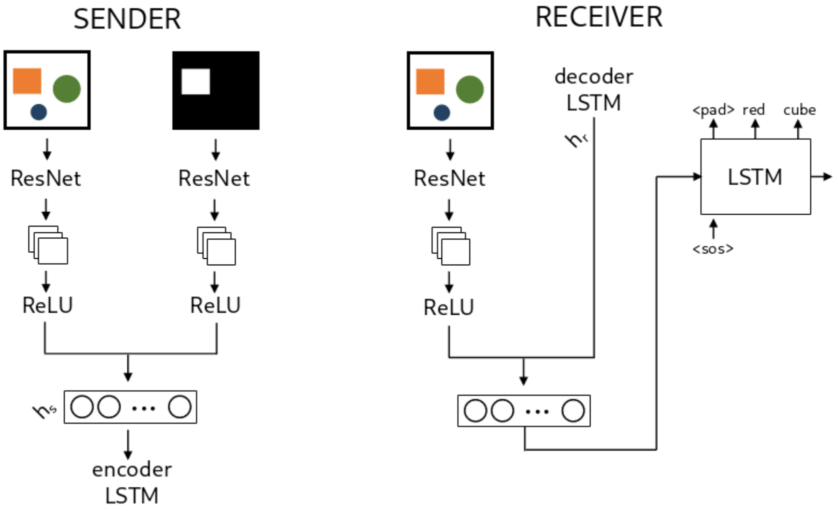


Figure 10: Simplified architecture of the caption generator game

The sender is built up of two image encoders, one encodes the original target image and one encodes the masked image. The image encoders are set up as described in Johnson et al. (2017). A feature extractor extracts basic features of the images. Two following convolutional layers with a *ReLU* function are trained to condense the most important information from the resulting matrices. A max pooling layer and a linear layer reduce the dimensions to 1024 dimensions. This size was found to have the best results during the experiments. The two resulting encoded images are flattened and concatenated. A final linear layer reduces this long vector to the hidden size of the message LSTM h_s, which is used as the initial hidden state of this LSTM.

The receiver uses the same architecture to encode the original image as the sender. The resulting flattened vector is concatenated with the decoded message of hidden size h_r. This is then passed through a linear layer, again reducing it to 1024 dimensions and is then used as the initial state of the captioning LSTM. During training, the ground truth caption is used as the input to the LSTM without teacher forcing. When presented with test data, the LSTM always produces three tokens, by using its own predicted words as the input for the next step. The loss is calculated with cross entropy.

Since the agents are trained to describe the target object discriminatively based on the described GRE-algorithm, they are trained on the 'Dale-2' and 'Dale-5' dataset. The 'Dale-5' should be again much harder to learn, since there are more objects that the agents need to discriminate the target object from.

During the experiments, the following variables are adjusted, and the results are compared: (1) the vocabulary size $|V|$, (2) the LSTM hidden size of the sender wrapper h_s and (3) the LSTM hidden size of the receiver wrapper h_r. The same metrics as in the pre-experiments are used to evaluate the results.

Results: The results of the caption generator game are summarized in Table 7. In general, it can be seen that the agents have much bigger problems, to solve the task together than a single neural network. The highest accuracy for descriptions, the agents manage to predict correctly is at 24,6% for images of the 'Dale-2' dataset. Compared to the (masked) accuracy of the single model with 72%, the agents predict 47,4% points less correct descriptions. A similar worse performance can be seen for the 'Dale-5' dataset. Here, the agents only manage to produce for 7,8% of the images correct descriptions, 13,2% points less than the single neural model. The same effect can be seen for the word-by-word accuracy, which is much

			Dale-2			Dale-5		
$\|V\|$	h_s	h_r	**Acc.**	**word-by-word**	**length**	**Acc.**	**word-by-word**	**length**
10	10	10	22,9%	62,8%	2	7,1%	40%	2
13	10	10	22,8%	62,9%	1	7,3%	38,7%	2
20	10	10	24,6%	64%	2	6,7%	38,7%	2
100	10	10	24,4%	62%	2	7,8%	40%	2
100	100	100	21%	62%	2	6,5%	37,8%	2

Table 7: Results of the caption generator: $|V|$ are different vocabulary sizes and h hidden sizes.

lower than the metric for the single neural model for both datasets.

When looking, how the different variables affect the performance, it can be seen that a bigger vocabulary size tends to help the agents. This is only visible for the 'Dale-2' dataset. With constant hidden sizes of 10, the agents score around 22,9% with only 10 and 13 available symbols. When this is increased to 20 and respectively 100 symbols, the agents can increase their accuracy to around 24,5%. However, the increase is relatively small. Interestingly, this effect only occurs, when the hidden sizes are small with only 10 dimensions. As soon as they are increased to 100 dimensions with a vocabulary size of 100 symbols, the accuracy drops to 21%.

Looking at the 'Dale-5' dataset, the increase is still there, when the vocabulary is increased to 100 symbols. Nonetheless, the difference is with 0,5% points even smaller and the reason may be due to other influences, such as the random initialization of the weights of the agents. This would need to be confirmed in future experiments.

An interesting observation can be made about the length of the messages, the sender generates. In almost all cases, the average length is 2 symbols. Only in one configuration, the sender produces single symbol messages. This happens, when the vocabulary size is 13, the same number as possible features the objects can have. In this case, the accuracy, as well as the word-by-word accuracy stay the same compared to the configuration with a vocabulary of only 10 symbols.

[Dominik Künkele] look at language more detailed

4.3.3 Coordinate predictor games

In the final experiments, it is tried to eliminate as much human knowledge as possible. For that reason, the agents are tasked to only predict the location of the target object. This task is approached in two steps. In the first step the sender is still shown human knowledge in form of a description of the target object, since the previous experiments showed that the models were able to relate them to their own extracted features. The receiver doesn't come in contact with any human knowledge, not as input nor as output. This approach acts as a sanity check if the agents are able to converge together on the correct target object coordinates. In the second step, the caption is also removed from the sender. Instead, a masked image points the sender towards the target object. In this setup, no human knowledge is explicitly present, that can bias the emerged language to form similarly to human language, except for the implicit information in the image itself. With this, it can be analyzed, how the language between the agents emerges and which features or patterns are represented with symbols.

The agents are both set up in the same way as the single neural model, predicting the target object's coordinates. The sender encodes the original image the same as in the previous paragraph. In the first setup, the description of the target object is encoded, using an LSTM. For this, an embedding with emb_{descr} dimensions is learned to represent each word. These embeddings are the input for the LSTM. The final hidden state of $LSTM_{descr}$ dimensions is used as the representation of the whole description. The vocabulary that is used for the descriptions is based on 14 symbols, including the padding token. For the LSTM to learn a representation of each token as well as of the complete description, both emb_{descr} and $LSTM_{descr}$ need to be smaller than the size of this vocabulary. Choosing a size of 10 for both variables proved to give good results in the experiments. In a next step, the image encoding and the final hidden state of the description are concatenated and passed through a linear layer to reduce the dimension to the hidden size h_s. The resulting vector is passed to the sender wrapper LSTM, to generate the message.

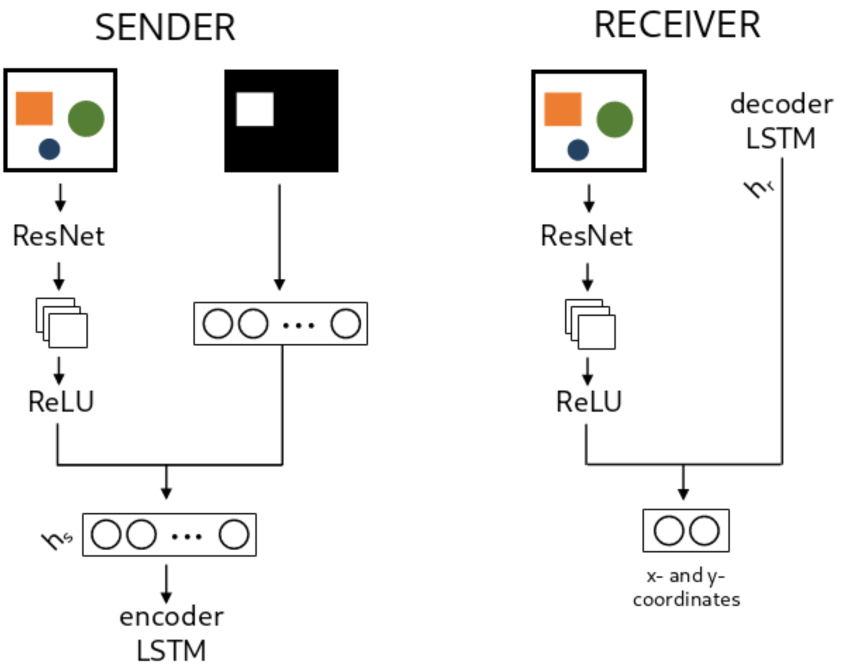


Figure 11: Simplified architecture of the masked coordinate predictor game

For the second approach, the masked image is passed only through a feature extractor. Both the resulting encoding of the masked image and the encoding of the original image are each passed through separate linear layer to adjust their dimensions to the same embedding size e_s. These embeddings are subsequently concatenated and reduced to the hidden size h_s with a final linear layer.

The receiver contains of two parts. First, the original image is encoded with an image encode of the same setup. This encoded image is flattened and concatenated with the final hidden state of the wrapper LSTM encoding the message received by the receiver. The resulting vector is the passed through the second part of the receiver, the predictor. This predictor contains of three linear layers, reducing the dimensions to 1024, 1024 and 2 respectively. A *ReLU* function is applied in between. This setup is trained to combine the important information from both the image and the message of the sender. The euclidean distance between the resulting prediction of the center point and the true center of the target object is calculated and the weights of both agents are adapted accordingly.

As in the pre-experiments, the agents are trained first the 'CLEVR single' dataset to understand if they are capable of predicting locations in an image together. In a next step, the 'Dale-2' and 'Dale-5' datasets are used to test if the agents are able to first communicate a target object and second describe the target object discriminatively with a small vocabulary.

During the experiments, the effects of the following variables are compared: (1) the vocabulary size $|V|$, (2) the LSTM hidden size of the sender wrapper h_s, (3) the LSTM hidden size of the receiver wrapper h_r and (4) the embedding size of the sender e_s. As before, the metrics to evaluate the results are the same as in the pre-experiments.

Results: In the final setup, the agents are tasked with communicating objects with fewer infused human knowledge. Table 8 shows the results for the setup, in which the sender is pointed towards the target object with a human description based on the GRE algorithm. Hereby, the 'CLEVR single' dataset acts as a baseline, to test if the agents are able to predict coordinates of objects at all. In every configuration of the variables, the agents achieve a very high performance. The worst average distance across the test dataset is 10 pixels, which still points on an object. Also the accuracy, which evaluates how many guesses of the receiver were pointing on an object reflects this fact. All configuration achieve an accuracy higher than 96,7%. This aligns also with the results from the single neural models, where the average distance was similarly low. In general, this shows that the agents are able to predict coordinates together. However, the question arises if the message by the sender has actually an effect on the receivers' decision, or if the receiver learns to point towards the target coordinates on his own and the message is ignored. When a different sender is used that is receiving the masked image as input, the results indicate that the message has in fact an influence and the receiver learns how to interpret the message. The results for this setup are summarized in 9. In three of six experiments, the agents predict much higher average distances. In one case, the mean distance lies at 19,8 pixels, while the accuracy is only at 55,1%. Since the receiver is the same in both setups, this observation indicates that a meaningful message is communicated between the agents.

			CLEVR single			**Dale-2**			**Dale-5**		
$\|V\|$	h_s	h_r	**Dist.**	**Acc.**	**length**	**Dist.**	**Acc.**	**length**	**Dist.**	**Acc.**	**length**
10	10	10	10,1	98,5%	2	36,5	19,9%	2	45,7	14,4%	2
13	10	10	6	99%	1	38	20,4%	2	47,3	10,8%	2
20	10	10	9,7	96,7%	2	37,3	21,2%	2	47,3	11,3%	1
100	10	10	7,7	98,4%	2	40,4	21,7%	2	45,4	10,8%	2
100	100	100	7,5	96,9%	2	40,1	17,8%	2	44,3	11,8%	1

Table 8: Results of the description coordinate predictor: $|V|$ are different vocabulary sizes and h hidden sizes.

When the experiments are run, using the 'Dale-2' dataset, the results are much worse. For the *description coordinate predictor*, the average distance ranges from 36,5 pixels to 40,4 pixels. The configuration with a vocabulary size of only 10 symbols fares the best, while a vocabulary of 100 symbols produces the worst results. Still, the accuracy shows that around 19,9% to 21,7% of the guesses are on the target object. Here, the configurations with higher vocabulary sizes fare better, even though the differences are very small and may be due to other factors. Interestingly, when comparing the two configurations with a vocabulary size of 100 symbols one can see that the average distance is similar, but the accuracy is significantly higher when the hidden sizes are lower. In other words, with lower hidden sizes the agents can make more correct guesses. Overall, it can be seen that a higher vocabulary size lets the agents' guesses get in general closer to the target object, but they are less precise on the object itself.

The results for 'Dale-5' dataset are even worse, but are comparable with the results with a single neural model. Apparently, the agents are not able to communicate the target object, and the predictions by the receiver are general in the middle of the image, which results in an average distance of around 45 to 50 pixels. The differences of the mean distances are not very significant in this range, to allow an analysis of the different configurations. The only metric that stands out is the accuracy of the configuration with a vocabulary size of 10 symbols. With 14,4%, it is 2,6% points higher than the next best accuracy. It is able to predict the center coordinates of the target object the most precise.

Finally looking at the

				CLEVR single			**Dale-2**			**Dale-5**		
$\lvert V\rvert$	h_s	h_r	e_s	**Dist.**	**Acc.**	**length**	**Dist.**	**Acc.**	**length**	**Dist.**	**Acc.**	**length**
10	10	10	1024	19,8	55,1%	2	34,8	24,3%	1	44,3	11,8%	2
10	10	10	512	9,3	92%	2	36,3	19,9%	1 ,7	45,9	12,7%	2
13	10	10	1024	7,8	96,8%	2	36,3	20,2%	1	45,4	11,4%	2
20	10	10	1024	6,6	98,3%	2	37,8	16,1%	2	45,2	11%	2
100	10	10	1024	5,2	98,5%	2	37,4	20,1%	2	43,6	16,7%	2
100	100	100	1024	12,5	92,1%	2	36,5	20,7%	2	44,6	12,7%	2

Table 9: Results of the masked coordinate predictor: $|V|$ are different vocabulary sizes, h hidden sizes and e embedding sizes.

5 Discussion

[Dominik Künkele] 4 pages

6 Conclusion and future work

[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*.

Kharitonov, E., Chaabouni, R., Bouchacourt, D., & Baroni, M. (2019). Egg: a toolkit for research on emergence of language in games.

Kirby, S. (2002). Natural language from artificial life. *Artificial Life*, 8(2), 185–215.

Lazaridou, A., Peysakhovich, A., & Baroni, M. (2016). Multi-agent cooperation and the emergence of (natural) language.

Lewis, D. K. (1969). *Convention: A Philosophical Study*. Cambridge, MA, USA: Wiley-Blackwell.

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*.

Wittgenstein, L. (1953). *Philosophische Untersuchungen*. Oxford: Basil Blackwell.

A Resources