Inferring and Executing Programs for Visual Reasoning

Published by Justin Johnson, Bharath Hariharan, Lauren van der Maaten, Judy Hoffman, Li Fei-Fei, C. Lawrence Zitnick, Ross Girshick

Stanford University, Facebook AI Research

Explainable Machine Learning

PD Dr. Ullrich Köthe

University of Heidelberg

Summer semester 2018

Author

Hannes Perrot

1 Introduction

In the following, I will summarize and discuss the paper "Inferring and Executing Programs for Visual Reasoning" written by Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Li Fei-Fei, C. Lawrence Zitnick and Ross Girshick from Stanford University and Facebook AI Research [4]. More and more applications in computer vision need to answer sophisticated questions, which require reasoning about the world. Reasoning is defined in the Cambridge Dictionary as "the process of thinking about something in order to make a decision" [1]. A real-world question requiring visual reasoning is shown in Figure 1.1. To answer this question, the terms have to be understood, recognized in the image and related to each other.

Is there a pedestrian in my lane?

Figure 1.1: Example of a visual reasoning problem [4]

The presented paper introduces a new method to model visual reasoning. The results are compared to other methods and tested on the CLEVER and the CLEVER-Humans dataset. The method includes a program generator and an execution engine, which together conduct the visual reasoning process.

1.1 CLEVER Dataset

A dataset specially created for the task of visual reasoning is the CLEVER dataset [3]. The dataset consists of images, a set of questions about each image, an example program, how this question could be answered and the correct answers. The dataset is created algorithmically and the code to create new images and questions is published. The main tasks, the algorithm has to be able to accomplish on the dataset,

are attribute identification, comparison, counting and evaluating spatial relationships and logical operations.

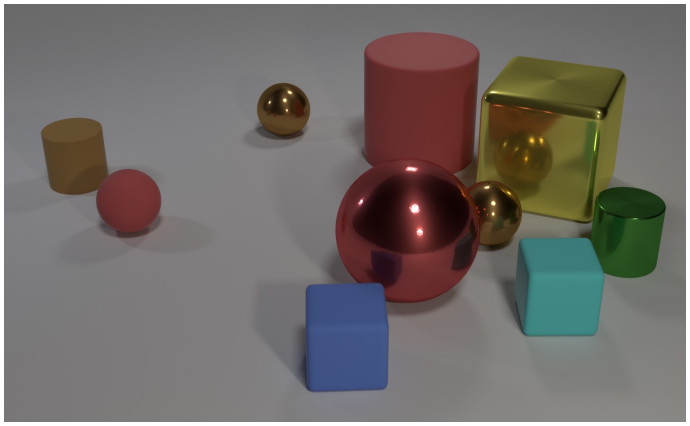

Figure 1.2: Example image of the CLEVER dataset [3]

Figure 1.2 is an example image from the CLEVER dataset. Some questions about this picture could be for example:

- Are there an equal number of large things and metal spheres?
- What size is the cylinder that is left of the brown metal thing that is left of the big sphere?
- There is a sphere with the same size as the metal cube; is it made of the same material as the small red sphere?

2 Method

An overview of the method proposed by the paper from Johnson et al. [4] is shown in Figure 2.1. The system consists mainly of two parts. First based on a question, a program is predicted by the program generator. This program is then executed by the execution engine using the visual features from the image. At the end of the execution engine, a classifier predicts an answer to the question about the image. Both, the program generator and the execution engine are implemented using neural networks. They are trained using back-propagation and REINFORCE.

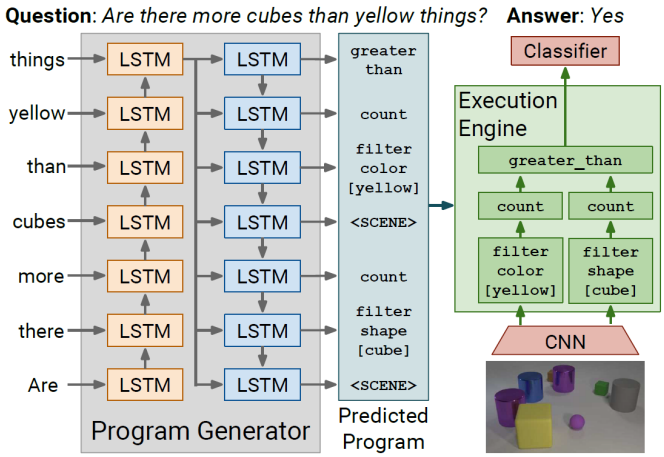

Figure 2.1: Overview of method [4]

2.1 Programs

The programs used to predict the questions' answers are composed of functions with a predefined meaning. The programs are represented by a tree of functions. Each function node gets the output of its children as input. The functions have a predefined arity, which means how many children the function node has. The output of the root node is fed to a classifier, which predicts the answer.

2.2 Functions

The functions are implemented using neural networks. For each function, a neural network module is used. The network to predict the answer to a question is build dynamically from the function modules. Each function has the same output size.

The input size is either of the functions depend on their airty. Basically there are different function types:

SCENE The Scene function returns the visual features of the input image. For this task, the output of the conv4 layer from a pretrained ResNet-101 [2] is used.

Unary functions Unary functions are functions with one input and one output. An example for this function is "count". These functions consist of a single residual block from the ResNet [2]. A residual block is shown in Figure 2.2. It consists of two convolutional layers. The input to the residual block is added to the output of the second convolutional layer. By introducing residual blocks, it was possible to build much deeper networks, without having problems with vanishing gradients. Also only learning the residual seemed to be an easier problem.

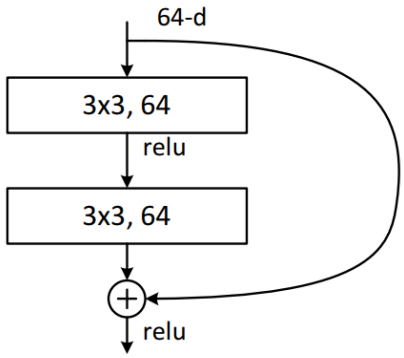

Figure 2.2: Residual block used in functions [2]

Binary functions Binary functions are functions with two inputs, like for example "greater_than". The two inputs are concatenated at their channel dimension and a 1x1 convolution reduces the channels again to their original channel width. A consecutive residual block like for the unary functions implements the actual functionality.

Classifier The output of the last function in the execution tree is flattened and fed to a classifier. This classifier predicts the most probable answer out of a fixed set of possible answers. The classifier is implemented as a multilayer perceptron classifier.

2.3 Program Generator

The program generator predicts a program from a natural language question. It gets the sequence of embeddings of words from the question as input and predicts a sequence of functions. These functions are then parsed to the program tree. The predicted functions are the new nodes, until the SCENE function is predicted. The next predicted functions build a new branch, if there is still a binary function with only one child left. The program generator is implemented using a sequence to sequence model. These kind of models are typically used for language translation [7]. One long short-term memory (LSTM) (or two consecutive) first takes the word embeddings and compresses them to an intermediate representation of the question with a fixed size. A second LSTM then predicts the program from the fixed size representation of the question. It samples from the functions until the program tree is complete. To make the program generator differentiable, they replace the argmax with sampling and use REINFORCE to estimate gradients on the outputs of the program generator [4].

2.4 Execution Engine

The execution engine takes a program and an image as input and predicts an answer. It is implemented using neural networks. For each function node in the program tree, the respective network module is taken. These modules are then composed according to the tree structure. So for each program, the network to predict the answer looks different. Because the interface between the functions has always the same size, all complete program trees can be executed. The output of the last function is then flattened and the classifier predicts an answer.

2.5 Training

For training this network, three approaches are proposed.

Strongly Supervised One possibility is to train the program generator and the execution engine separately. The program generator is trained with the ground truth questions and programs using the standard LSTM sequence to sequence training like for example in language translation. The functions in the execution engine are

trained with the images and their ground truth programs and answers. Since the network is defined by the ground truth program, this is a normal classification training, besides that the network is rearranged for each question. With this supervised training procedure, the best testing performance is achievable. On the other side, for every question, a ground truth program is needed. This is not always possible, for example when training on the CLEVER-Humans dataset (see section 3.5)

Joint REINFORCE Another possibility is to train both networks jointly end to end and back-propagate through both parts of the network. A clear benefit of this training method is, that no ground truth programs are needed to train the program generator. But training without any ground truth programs is hard, since there is the hen and egg problem: The program generator needs to produce programs without understanding what the functions are meant to mean and the execution engine has to produce the correct answers from programs, that are just like randomly generated. Nonetheless, this is a good possibility to fine-tune the network, once a program generator and an execution engine are pretrained.

Combined semi-supervised A third possibility is a combination of both methods:

1. Train program generator on a small subset of ground truth programs
2. Fix program generator and train execution engine using predicted programs on large dataset
3. Use REINFORCE to fine-tune program generator and execution engine

Applying this method, only a small subset of the ground truth programs are used to train the program generator in the beginning. This also allows to fine-tune the model on a dataset without ground truth programs.

3 Experiments and Results

In this chapter some of the experiments conducted by the authors of the paper are presented.

3.1 Comparison of training procedures

In Figure 3.1, the accuracy of different methods is shown. The first five methods are the baselines. Human performance is measured on a subset of the CLEVER dataset on Mechanical Turk. The three rows below show the performance of networks trained with a different amount of ground truth programs. The Ours-strong model was trained strongly supervised exploiting all ground truth programs. The models Ours-semi was trained with the combined semi-supervised training method. The 18k and 9k means the number of ground truth programs used to train the program generator in step 1 of the semi-supervised training.

			Compare Integer			Query				Compare				
Method	Exist	Count	Equal	Less	More	Size	Color	Mat.	Shape	Size	Color	Mat.	Shape	Overall
Q-type mode	50.2	34.6	51.4	51.6	50.5	50.1	13.4	50.8	33.5	50.3	52.5	50.2	51.8	42.1
LSTM	61.8	42.5	63.0	73.2	71.7	49.9	12.2	50.8	33.2	50.5	52.5	49.7	51.8	47.0
CNN+LSTM	68.2	47.8	60.8	74.3	72.5	62.5	22.4	59.9	50.9	56.5	53.0	53.8	55.5	54.3
CNN+LSTM+SA [45]	68.4	57.5	56.8	74.9	68.2	90.1	83.3	89.8	87.6	52.1	55.5	49.7	50.9	69.8
CNN+LSTM+SA+MLP	77.9	59.7	60.3	83.7	76.7	85.4	73.1	84.5	80.7	72.3	71.2	70.1	69.7	73.2
Human† [19]	96.6	86.7	79.0	87.0	91.0	97.0	95.0	94.0	94.0	94.0	98.0	96.0	96.0	92.6
Ours-strong (700K prog.)	**97.1**	**92.7**	**98.0**	**99.0**	**98.9**	**98.8**	**98.4**	**98.1**	**97.3**	**99.8**	**98.5**	**98.9**	**98.4**	**96.9**
Ours-semi (18K prog.)	95.3	90.1	93.9	97.1	97.6	98.1	97.1	97.7	96.6	99.0	97.6	98.0	97.3	95.4
Ours-semi (9K prog.)	89.7	79.7	85.2	76.1	77.9	94.8	93.3	93.1	89.2	97.8	94.5	96.6	95.1	88.6

Figure 3.1: Comparison results on test set different training procedures [4]

One observation is, that the strongly supervised model exceeds human performance in each question category. Also the semi-supervised model with 18k programs exceeds human performance in almost every category, even though only 4 % of the unique questions of the dataset were in the training set. This means, that the model can adapt to new questions.

The program accuracy over the number of training programs is shown in Figure 3.2. The program accuracy is measured as the number of programs, that are predicted by the program generator exactly as they are in the ground truth. This accuracy rises below 10k training programs and converges then to nearly 100 %. 20k training programs seem to be sufficient to have almost exact predicted programs.

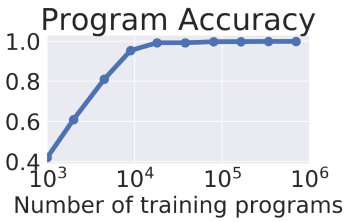

Figure 3.2: Program accuracy over number of training programs in training set [4]

The answer accuracy of the execution engine is shown in Figure 3.3. The dashed line is the accuracy of the strongly supervised model, which used all of the ground truth programs during training. The green line is the answer accuracy when training the execution engine with the predicted programs of the program generator, which itself was trained on a certain number ground truth of programs. This leads to less accurate predicted answers. With the joint fine-tuning (step 3 in the combined semi-supervised training method), some of the reduced accuracy can be eliminated.

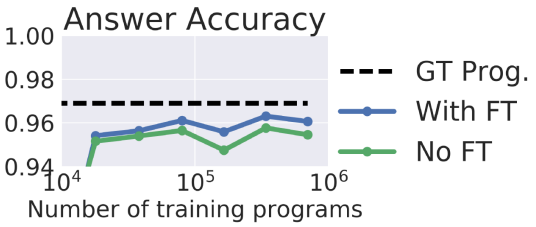

Figure 3.3: Answer accuracy over number of training programs in training set [4]

3.2 What do the modules learn

To check, what the modules learned, they visualized the attention of the network with respect to the input pixels on a series of questions, where always some more or different aspects are asked. In Figure 3.4 one of these series is shown.

Figure 3.4: Attention visualization of the network for slightly different questions [4]

This series shows, that the attention of the network actually is directed to the parts of the image, where the objects from the questions are. The region of attention also changes if single words in the question are exchanged. This seems like, the modules actually learned the meaningful representations.

3.3 Generalizing to new attribute combinations

With a split of the network, generalization to new attirbute combinations can be checked. Therefore the dataset was split, so that split A had cubes in gray, blue, brown, and yellow and cylinders in red, green, purple, and cyan. Split B has the colors exchanged. They first trained on split A of the dataset and then tested on A and B separately. The pretrained network was then fine-tuned on split B and tested again.

	Train A		Finetune B	
Method	A	B	A	B
LSTM	55.2	50.9	51.5	54.9
CNN+LSTM	63.7	57.0	58.3	61.1
CNN+LSTM+SA+MLP	80.3	68.7	75.7	75.8
Ours (18K prog.)	**96.6**	**73.7**	**76.1**	**92.7**

Figure 3.5: Results generalizing to unseen attribute combinations [4]

In Figure 3.5 the results are shown. Still, their method outperforms their baselines, but the split, where the network was not trained on, has significantly worse accuracy. So no complete generalization of attributes is possible, if the network hasn't seen the attribute combinations during training. After fine-tuning on split B, the accuracy on this split is improved, while it "forgets" to answer questions on the split it was originally trained on.

3.4 Generalizing to new question types

In real world scenarios, the program generator should be able to adapt to new question types and program structures without having seen every possibility. To test this, the dataset was split into long and short programs. The program generator was first trained on short examples with the semi-supervised training procedure. Then the program generator was fine-tuned on both types, with a fixed execution engine.

Method	Train Short		Finetune Both	
	Short	Long	Short	Long
LSTM	46.4	48.6	46.5	49.9
CNN+LSTM	54.0	52.8	54.3	54.2
CNN+LSTM+SA+MLP	74.2	**64.3**	74.2	67.8
Ours (25K prog.)	**95.9**	55.3	**95.6**	**77.8**

Figure 3.6: Results generalizing to longer questions [4]

When the execution engine was trained only on the short programs, a "short program bias" was introduced. This can be seen when the program generator is tested on longer questions. The generated programs tend to model a shorter question. With these programs, the model even underperforms one of their baselines. After fine-tuning only the program generator on both question types, this bias can be reduced. The model then again outperforms the baselines. This means, that new program structures can be adapted, if the model is fine-tuned on them. This fine-tuning needs only the correct answers, but no ground truth programs.

3.5 CLEVER-Humans Dataset

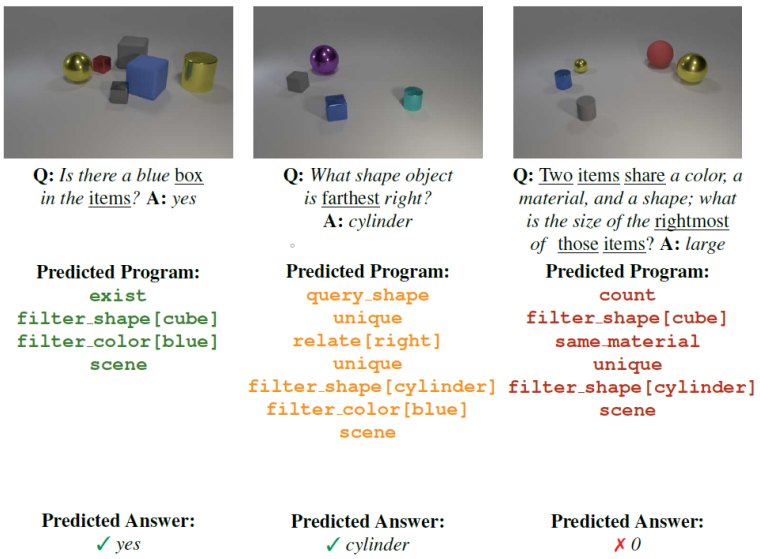

Figure 3.7: Examples of CLEVER-Humans dataset with predicted programs and answers [4]

The CLEVER-Humans dataset is a dataset with linguistically more diverse questions. To create the CLEVER-Humans dataset, workers on Mechanical Turk were

asked to write questions about images from the CLEVER dataset, which are hard for a clever machine to answer. Some other workers were asked to answer these questions. If the answers agreed among each other, the question was added to the CLEVER-Humans dataset. In Figure 3.7 some example questions are shown. The underlined words didn't appear in questions of the original CLEVER dataset. For this dataset, no ground truth programs are available. To train the model on this dataset, it was first trained on the CLEVER dataset, then the program generator was fine-tuned on the CLEVER-Humans dataset. The embeddings of previously unknown words were initialized randomly.

The task of the model is to reuse the learned reasoning capability and adapt to new words and question types. This works, as it still outperforms the baselines. In the left example of Figure 3.7, the program generator for example learned, that the new word "box" has the same meaning than the already known word "cube". In the middle example, the program can approximate the question, and create a reasonable program and answer. In some cases (right example), the predefined functions are just not able to model the question. In these cases, the model often fails.

4 Conclusion

Why is this approach better than other compared attempts? Johnson et al. [4] introduce a novel approach to answer questions, that involve reasoning. The method involves a program generator, that constructs a representation of the reasoning process needed to answer the question and an execution engine, to execute the program and answer the question. Previous methods, that directly tried to answer the question with models, that work as a black box model tend to exploit model biases. By explicitly modeling the reasoning process, this novel approach has the possibility to learn reasoning about the questions rather than exploiting data biases.

To what extent make the functions the approach explainable? The intermediately generated program offers an additionally explanation, why the model came to this specific answer. This makes the approach more explainable than the black box approaches.

What could be issues when trying to exceed toy problems? In real wold problems, the training data cannot be generated as easily as the CLEVER dataset. When labeling this dataset in addition to the questions and answers, an appropriate program has to be generated. This labeling process could be very expensive.

As seen in section 3.3, generalizing to new attribute combinations is not trivial for the model. To overcome this, preferably every attribute combination should be present in the training set. In real world problems, the possible categories could be much more diverse and there could be much more attribute combinations. This would also require a big dataset.

Are there new developments and papers so far? The field of visual reasoning is an open research topic and there are many new papers published, using the CLEVER benchmark. Some of them are also surpassing the results of the presented method. These include for example new attention mechanism [5] and new conditioning methods [6].

5 References

[1] Dictionary, Cambridge: *reasoning Bedeutung im Cambridge Englisch Wörterbuch*. https://dictionary.cambridge.org/de/worterbuch/englisch/reasoning. – visited on 2018-09-06

[2] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Deep Residual Learning for Image Recognition. In: *CoRR* abs/1512.03385 (2015). http://arxiv.org/abs/1512.03385

[3] Johnson, Justin ; Hariharan, Bharath ; Maaten, Laurens van d. ; Fei-Fei, Li ; Zitnick, C. L. ; Girshick, Ross B.: CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning. In: *CoRR* abs/1612.06890 (2016). http://arxiv.org/abs/1612.06890

[4] Johnson, Justin ; Hariharan, Bharath ; Maaten, Laurens van d. ; Hoffman, Judy ; Li, Fei-Fei ; Zitnick, C. L. ; Girshick, Ross B.: Inferring and Executing Programs for Visual Reasoning. In: *CoRR* abs/1705.03633 (2017). http://arxiv.org/abs/1705.03633

[5] Malinowski, M. ; Doersch, C. ; Santoro, A. ; Battaglia, P.: Learning Visual Question Answering by Bootstrapping Hard Attention. In: *ArXiv e-prints* (2018), August. https://arxiv.org/abs/1808.00300

[6] Perez, Ethan ; Strub, Florian ; Vries, Harm de ; Dumoulin, Vincent ; Courville, Aaron C.: FiLM: Visual Reasoning with a General Conditioning Layer. In: *CoRR* abs/1709.07871 (2017). http://arxiv.org/abs/1709.07871

[7] Sutskever, Ilya ; Vinyals, Oriol ; Le, Quoc V.: Sequence to Sequence Learning with Neural Networks. In: *CoRR* abs/1409.3215 (2014). http://arxiv.org/abs/1409.3215