USING CONCEPT DETECTORS TO IMPROVE PERFORMANCE OF VISUAL QUESTION ANSWERING

S. REITSMA

Master of Science Thesis
Supervised by M.H.T. de Boer, MSc and prof. dr. T.M. Heskes
Computer Science Department
Radboud University Nijmegen

June 28, 2016 – version 1.0
ABSTRACT

In this thesis, the image question answering problem is explored in the context of the Visual Question Answering challenge set up by Microsoft and VirginiaTech. In the image question answering problem, the goal for a system is to answer a question about an image. We explore existing methods and make improvements upon prior work. The main contribution is a trained concept detector that improves performance, especially on questions with numerical answers. The concept detector is trained on the ground truth segmentations in the MSCOCO dataset and during testing time, the detector is run on the full image to obtain activations for each concept. Furthermore, a post-processing repair technique is employed that improves performance even more. An accuracy on numerical answers of 38.09 was reached, a significant improvement over the baseline of 36.92.
CONTENTS

1 INTRODUCTION 1
2 VISUAL QUESTION ANSWERING CHALLENGE 2
3 PRIOR WORK 8
  3.1 Baselines ............................................. 8
  3.2 Methods ................................................. 8
    3.2.1 Image recognition using convolutional neural networks 8
    3.2.2 Understanding questions .............................. 9
    3.2.3 Merging modalities and generating an answer .......... 10
    3.2.4 Other methods ...................................... 11
  3.3 Dynamic Parameter Prediction ................................. 11
    3.3.1 Image understanding: finetuning .................... 11
    3.3.2 Understanding questions using Gated Recurrent Units 11
    3.3.3 Fusing modalities and the hashing trick ............ 12
    3.3.4 Results on the VQA task .............................. 14
4 IMPROVING 15
  4.1 Object proposals ........................................ 15
    4.1.1 Ground truth annotations .......................... 15
    4.1.2 Edgeboxes and Deepboxes .......................... 15
  4.2 Concept detectors ........................................ 16
    4.2.1 Network structure and training method ............ 17
    4.2.2 Testing methodology ................................ 18
  4.3 Regression .............................................. 18
  4.4 Postprocessing repair ................................... 18
5 EXPERIMENTS 19
  5.1 Dataset splits ........................................... 19
  5.2 Experiments ............................................. 19
  5.3 Implementation ......................................... 20
6 RESULTS 21
  6.1 Prior results ............................................ 21
  6.2 Segmentations ........................................... 21
  6.3 Concept detectors and post-processing repair ............ 22
  6.4 Regression .............................................. 23
  6.5 In challenge context ..................................... 23
7 CONCLUSION AND EVALUATION 24
8 FURTHER WORK 26
  8.1 Acknowledgements ....................................... 26

BIBLIOGRAPHY 27
INTRODUCTION

Since a few years, deep learning has been increasingly popular in computer vision, text understanding and speech recognition [12]. It is especially in the areas that were previously thought to be extremely hard for computers to do that deep learning has risen to deliver state-of-the-art performance. In computer vision, one of the larger goals is to have global scene understanding: understanding the contents and activities of an image scene [25]. Over the past few years, significant progress has been made in recognition tasks within computer vision, but it mainly focuses on very specific problems, such as object classification [7], object segmentation [17] and face detection [20]. Ideally, a system would have a much broader understanding of an image scene.

In image question answering, image understanding is brought to a higher level by not focusing on a single computer vision problem, but instead focusing on all aspects of the image scene. The goal of image question answering is for a system to answer natural language questions posed by human users about images. In practice this could take recent developments such as Google Now, Siri and Cortana a step further by not only being able to answer questions on general topics that are searchable on the web, but also on the local user context using e.g. a smartphone’s camera. This could be especially useful for visually impaired users, who can take a picture using their smartphone and ask their device questions about the local scene, such as where is an empty seat in this train? or is there a pedestrian crossing here?.

To enable this kind of applications, a thorough understanding of the image and the question is required. We ideally also have a knowledge base that we can use to help in answering questions. Advances in the deep learning field have increased feature extraction methods for both images – using convolutional neural networks – and natural language – using recurrent neural networks. Combining these extracted features can intuitively generate an answer for the question. If, for example, the question is is there a cat in this scene? and we are presented with an image prominently containing a cat, the feature activation for the concept cat will be high in both the image feature vector and the question feature vector. By combining these features in a clever way, the answer yes could be generated. In this thesis, several existing deep learning solutions [1, 2, 15, 18, 24] to image question answering are explained and improvements are made by using dedicated concept detectors. All results are compared on the VQA dataset as part of the VQA challenge. In Chapter 2 the VQA challenge is explained in more detail. In later sections, methods for solving the image question answering problem and their results are shown.
The Visual Question Answering challenge is set up by VirginiaTech and Microsoft Research after the release of the MSCOCO dataset [2]. The challenge revolves around answering various types of questions on the contents of an image scene. The images are all from the MSCOCO dataset, while the questions and their answers have been gathered specifically for the VQA challenge.

The dataset consists of 204,721 real-world images of various sizes, ranging from 0.01 to 1 megapixels. Each image has 3 questions with 10 ground truth answers per question. The ground truth answers are not necessarily unique and as such, the union of ground truth answers is not necessarily of the same size for each question. The evaluation metric takes the multiple ground truth answers per question into account. If a predicted answer is given by at least three human annotators (of the 10) the answer is marked as correct. If the predicted answer is given by less than three human annotators, the score for that answer is adjusted accordingly.

\[ \text{Acc(\text{ans})} = \min \left( \frac{\# \text{humans that said \text{ans}}}{3}, 1 \right) \]  \tag{1} \]

The reason for this metric is that the inter-human variability in phrasing the answers is rather high. With this metric, even if human annotators are phrasing their answers differently, a predicted answer can still attain a high score. Furthermore, predicted answers are post-processed during evaluation, i.e. by making all characters lowercase and by inserting missing contraction apostrophes.

There are two online leaderboards that are used in this challenge: open-ended and multiple-choice. On the open-ended task, no possible answers are given and the system has to construct the answer by itself. For the multiple-choice task, a set of 18 predefined answers is given from which the correct one has to be picked. Naturally, the multiple-choice task results in a higher answering performance. The 18 possible answers are made up as follows for each question:

1. The most common answer out of the 10 ground truth answers given for the question
2. 3 plausible answers that are answered by a human annotator without seeing the image
3. 10 most popular answers (aggregated over entire dataset) are inserted
4. The rest of the answers are taken randomly from the set of all given answers

Of these answers, only the most common answer given for the question is marked as correct.

An example of some images and their questions is shown in Figure 1.
The dataset is made up of six separate splits: train, validation, test-dev, test-standard, test-challenge and test-reserve. The splits are the same splits as in the MSCOCO dataset itself. The training and validation splits both have their answers publicly available, while the four test splits do not. Participants in the VQA challenge can submit as often as desired to the test-dev split, while the test-standard has a maximum number of five submissions. The test-challenge split is used to determine the winner of the challenge and results are not published until after the deadline. Results on the test-reserve split are not ever made public. This split is to ensure there was no strong overfitting on any of the other splits. Since the results on this dataset are never made public, it can be used for future challenges as well.

The questions and their answers were crowdsourced on Mechanical Turk and can thus vary a lot. They range from concept detection ("how many bikes are in this picture") to knowledge base reasoning ("is this a vegetarian pizza?") to activity recognition ("is this man crying?"). Persons participating in the crowdsourcing were presented an image and were asked to pose a question relevant to the image. Afterwards, ten other crowdsourcers were asked to answer the question. In Figure 2 the distribution can be seen for all question types.

The same type of visualization can be made for the answers to the questions, as shown in Figure 3. Note that the answer space can be reduced immensely by just looking at the question type. For example, if a question starts with is there... the answer is almost always either yes or no.

Some other interesting statistics mentioned by Antol et al. [2] are that 38.37% of all questions are ‘yes/no’ questions and 12.31% are number questions. Furthermore, inter-human agreement is much higher for these types of question. This is likely due to the lack of synonyms and language structure in these types of answers.
Figure 2: Question distribution for all question types. Taken from [2].
Figure 3: Answer distribution per question type. Taken from [2].
In Table 1 human performance is reported for several question types. Human performance was tested in two ways: by showing only the question and by showing the question and the image. Naturally, the performance of the latter is higher. Note that the reported scores are not necessarily equal to the fraction of correctly answered questions due to the soft rejection metric as shown in Equation 1.

<table>
<thead>
<tr>
<th>Question type</th>
<th>% of dataset</th>
<th>Human perf. (Q)</th>
<th>Human perf. (Q + I)</th>
</tr>
</thead>
<tbody>
<tr>
<td>what is</td>
<td>13.84%</td>
<td>16.86</td>
<td>73.68</td>
</tr>
<tr>
<td>what color</td>
<td>8.98%</td>
<td>28.71</td>
<td>86.06</td>
</tr>
<tr>
<td>what kind</td>
<td>2.49%</td>
<td>19.10</td>
<td>70.11</td>
</tr>
<tr>
<td>what are</td>
<td>2.32%</td>
<td>17.72</td>
<td>69.49</td>
</tr>
<tr>
<td>what type</td>
<td>1.78%</td>
<td>19.53</td>
<td>70.65</td>
</tr>
<tr>
<td>is the</td>
<td>10.16%</td>
<td>65.24</td>
<td>95.67</td>
</tr>
<tr>
<td>is this</td>
<td>8.26%</td>
<td>63.35</td>
<td>95.43</td>
</tr>
<tr>
<td>how many</td>
<td>10.28%</td>
<td>30.45</td>
<td>86.32</td>
</tr>
<tr>
<td>are</td>
<td>7.57%</td>
<td>67.10</td>
<td>95.24</td>
</tr>
<tr>
<td>does</td>
<td>2.75%</td>
<td>69.96</td>
<td>95.70</td>
</tr>
<tr>
<td>where</td>
<td>2.90%</td>
<td>11.09</td>
<td>43.56</td>
</tr>
<tr>
<td>is there</td>
<td>3.60%</td>
<td>72.48</td>
<td>96.43</td>
</tr>
<tr>
<td>why</td>
<td>1.20%</td>
<td>11.80</td>
<td>21.50</td>
</tr>
<tr>
<td>which</td>
<td>1.21%</td>
<td>25.64</td>
<td>67.44</td>
</tr>
<tr>
<td>do</td>
<td>1.15%</td>
<td>71.33</td>
<td>95.44</td>
</tr>
<tr>
<td>what does</td>
<td>1.12%</td>
<td>11.12</td>
<td>75.88</td>
</tr>
<tr>
<td>what time</td>
<td>0.67%</td>
<td>7.64</td>
<td>58.98</td>
</tr>
<tr>
<td>who</td>
<td>0.77%</td>
<td>14.69</td>
<td>56.93</td>
</tr>
<tr>
<td>what sport</td>
<td>0.81%</td>
<td>17.86</td>
<td>95.59</td>
</tr>
<tr>
<td>what animal</td>
<td>0.53%</td>
<td>17.67</td>
<td>92.51</td>
</tr>
<tr>
<td>what brand</td>
<td>0.36%</td>
<td>25.34</td>
<td>80.95</td>
</tr>
</tbody>
</table>

Table 1: Human performance for several question types. For human performance, both question-only performance and question and image performance is reported. Taken from [2].

Questions that usually have a yes/no answer naturally have a higher performance. For some question types, such as ‘why’, the difference between the human performance with the question only and the performance with both the question and the image is rather low. This can be explained by the notion that answers to ‘why’ questions can be phrased in a lot of different ways, making it hard to get the answer correct.

In this thesis, a method is proposed that improves performance significantly. The main purpose is to improve performance for counting questions, i.e. questions with numerical answers. Examples of these questions are: How many cars are in this picture? and How many people are sitting? A possible usecase for this is for CCTV cameras, especially in activity detection. The amount of instances of a certain object being present in an image scene has great value for activity detection [9]. Other possible usecases are inventory management (automatically count inventory) and optical sorting. Some of the counting questions can be solved by counting objects in the image, while some others require additional constraints – such as the question How many people are sitting? where you not only need to detect people, but also the activity of sitting. Within the field of counting objects in images a lot of re-
search has been done already [3, 13, 17]. This research has however not yet been applied to visual question answering. In this thesis, several techniques for counting concept instances are tested and compared.

The main research question here is as follows: *Do dedicated concept detectors improve performance for counting questions?* In the next chapter, prior work is explored in more detail, not necessarily focusing on counting questions but on the visual question answering problem as a whole. In Chapter 4, the goal and structure of the employed concept detector is shown, focusing on the performance increase of counting questions.
In this chapter, several existing strategies for tackling the visual question answering problem are shown. While setting up the challenge, Antol et al. [2] implemented a set of baselines and basic methods. These baselines are explained in the following sections in more detail, as well as their components. In later sections, more advanced existing strategies for tackling the problem are shown.

3.1 Baselines

Four baselines were developed and scored on the test-dev dataset: random, prior, Q-type prior and nearest neighbor. The random baseline generates a random answer from the top 1000 answers of the training set. The prior baseline generates the most popular answer (=‘yes’) for every question. The Q-type prior does the same, but chooses the most popular answer per question type instead of for the whole dataset. Finally, the nearest neighbor approach converts the test question to word2vec feature space and finds the nearest training vector using cosine similarity. The ground truth answer for the closest question is chosen. These baseline methods are not designed to attain a high performance but serve as an indicator of what performance is achievable with simple techniques.

3.2 Methods

More interestingly, Antol et al. [2] developed a set of more sophisticated methods for tackling the challenge, built around pretrained convolutional neural networks and a separate question channel. A visual question answering system requires at least three subsystems:

1. Understanding the image
2. Understanding the question
3. Generating an answer

In the following subsections, each of these subsystems is explained for the models Antol et al. [2] developed.

3.2.1 Image recognition using convolutional neural networks

The first subsystem is concerned with understanding the image. Ideally, we would not only gain an understanding of what is in the image, but also the context in which it can be placed. While the latter still seems to be hard, as will be shown later in the results, there are some developments in the field that try to use general knowledge bases to find image contexts [23]. In the baseline method developed by Antol et al. [2], no knowledge bases are used. Instead, a convolutional neural network is used to extract features from the images directly. This network is not trained from scratch, but instead a pretrained model is used. This model is trained on the ImageNet dataset [7] using the VGG16 architecture [19]. Pretrained models are not only easier to use – since they do not require training – but also often reach a higher level of performance. This is especially the case when the problem dataset is smaller than the ImageNet dataset. Since the ImageNet dataset is so large,
the pretrained model can have a very good generalization, much more so than if it were trained on the smaller problem dataset. However, the network will not be as specialized as when it would have been trained on the problem dataset. This is often solved by finetuning the pretrained network by backpropagation with a small learning rate using the previously unseen images from the problem dataset. As previously mentioned, the images in the VQA dataset are taken directly from the MSCOCO dataset, which is very large (over 300,000 images) and also is very much like the ImageNet dataset, as both are real-world photographs of everyday scenes. Finetuning is not strictly necessary in this case, but could increase performance somewhat. In the baseline model [2], no finetuning is employed.

Since the VGG16 model architecture was created for classification into the ImageNet classes, we need to make some minor modifications. The final layer of the model has 1000 units, one for each of the classes of the ImageNet dataset. However, in the case of the VQA problem, we are not interested in these specific classes, but instead in creating an image embedding. Therefore, the final layer of the model is removed and instead the raw 4096-dimensional features are used. Antol et al. [2] employ two strategies for using the resulting features: (1) use them directly or (2) apply L2-normalization before using them. As shown in Table 2, performance is much better for the latter strategy. After obtaining the 4096-dimensional features, they are passed through a fully-connected layer to obtain the final 1024-dimensional embedding of the image.

![Diagram of VGG16 convolutional neural network](image)

Figure 4: A visual representation of the VGG16 convolutional neural network. Note that the final 1 x 1 x 1000 layer and its softmax function are removed. The 4096 preceding features are used as the image embedding.

### 3.2.2 Understanding questions

The second subsystem concerns understanding the question. To this end, the given question has to be converted to an embedding. Antol et al. [2] propose two methods: a simple BoW approach and a more sophisticated LSTM approach. In the BoW approach, the top 1000 words in the questions are used to create a 1000-dimensional bag-of-words representation where:

\[ f_i(q) = \#(w_i(q)) \]  

Thus, feature \( f_i \) is set to the amount of occurrences of word \( w_i \) in the question. Additionally, the first, second and third words of the question generate
an additional top 10 BoW representation. For each question, this generates
an embedding of size 1030. Naturally, this embedding does not capture syntax or word order in any way.

The second approach Antol et al. [2] propose is to use an LSTM [8]. In
this method, each word in the question is converted to a one-hot encoding of the same length as the vocabulary. This encoding is then passed through a fully-connected layer with a tanh non-linearity in order to generate a 300-dimensional word embedding. These embeddings are then fed one-by-one through LSTM units with an output size of 512. Finally, these features are fed into a fully-connected layer with tanh non-linearity to increase the amount of features to 1024, similar to the image embedding. The LSTM model is trained from scratch. A visualization of the LSTM is shown in Figure 5.

![LSTM Visualization](image)

“How many horses are in this image?”

Figure 5: A visualization of the LSTM used to create an embedding of the questions [2].

### 3.2.3 Merging modalities and generating an answer

The third submodule concerns generating an answer. To do this, the two input modalities – the image and the text – need to be merged somehow. For the BoW-model, the BoW embedding is simply concatenated with the image features. For the LSTM model, the image features and the LSTM embedding – both of length 1024 – are fused through element-wise multiplication. The combined input is fed into a set of dense layers followed by a softmax layer to obtain the distribution over the Top 1000 answers. The LSTM model and the final dense layers are trained end-to-end with a cross-entropy loss. Note that the image model is not finetuned, i.e. its parameters are static. In Table 2 the results for these methods are shown.

<table>
<thead>
<tr>
<th>Method</th>
<th>All</th>
<th>Yes/No</th>
<th>Number</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prior ('yes')</td>
<td>29.66</td>
<td>70.81</td>
<td>0.39</td>
<td>1.15</td>
</tr>
<tr>
<td>Per Q-type prior</td>
<td>37.54</td>
<td>71.03</td>
<td>35.77</td>
<td>9.38</td>
</tr>
<tr>
<td>Nearest neighbor</td>
<td>42.70</td>
<td>71.89</td>
<td>24.36</td>
<td>21.94</td>
</tr>
<tr>
<td>BoW Question only</td>
<td>48.09</td>
<td>75.66</td>
<td>36.70</td>
<td>27.14</td>
</tr>
<tr>
<td>Image only</td>
<td>28.13</td>
<td>64.01</td>
<td>0.42</td>
<td>3.77</td>
</tr>
<tr>
<td>BoW Question + Image</td>
<td>52.64</td>
<td>75.55</td>
<td>33.67</td>
<td>37.37</td>
</tr>
<tr>
<td>LSTM Question only</td>
<td>48.76</td>
<td>78.20</td>
<td>35.68</td>
<td>26.59</td>
</tr>
<tr>
<td>LSTM Question + Image</td>
<td>53.74</td>
<td>78.94</td>
<td>35.24</td>
<td>36.42</td>
</tr>
</tbody>
</table>

Table 2: Results for baseline methods on the open-ended task by Antol et al. [2]. Tested on test-dev dataset.

As can be seen from the results, using more sophisticated methods improves performance. However, we can also see that for number questions (questions that have numerical answers), performance does not increase as much when using more advanced techniques. The performance between
using a bag-of-words approach given only the question is 36.70, while the performance for a deep LSTM given the question and the image is 35.24, a decrease in performance.

3.2.4 Other methods

There exist several methods that try to improve upon the result by Antol et al. [2]. Some of these methods are attention-based, such as [1] and [18]. They focus on directing the attention of the system towards a specific region in the image. According to these prior works, this could be especially useful for where questions and what color questions. In [18], for example, a system is developed based on Edgeboxes which tries to find – by end-to-end backpropagation – the relevant areas in an image for a specific question. Other methods are memory or knowledge-based, and try to build and use a knowledge base to answer commonsense questions [23, 24]. In this thesis, DPPnet [15] was chosen to be used as the baseline to improve upon since it achieved the highest result in the previous iteration of the Visual Question Answering challenge. Also, the main contribution of DPPnet is in combining the two modalities of text and image. Since this thesis focuses on counting questions with concept detectors and not on the combining of modalities this can be easily incorporated into DPPnet without changing much of its existing functionality. Furthermore, the implementation of concept detectors can easily be applied to other methods. This is explained in more detail in Chapter 4.

3.3 DYNAMIC PARAMETER PREDICTION

Noh, Seo, and Han [15] used the LSTM and VGGnet baseline to improve performance on the VQA challenge. All three subsystems are improved: question understanding, image understanding and modality fusion. The method is named DPPnet, and in the next sections the method is explained in detail.

3.3.1 Image understanding: finetuning

Instead of taking the raw features from the VGGnet output, the network is finetuned. Finetuning means that the pretrained network is updated using backpropagation on the new dataset. Since the pretrained VGGnet convolutional neural network is trained on ImageNet and the VQA dataset is based on a different dataset (MSCOCO), we can increase the amount of data the network has seen by finetuning. The network architecture needs to be changed, since now we do not have a classification network like in ImageNet. The last fully-connected layer is removed (including its softmax) and three new fully-connected layers are added. Of these layers, the second is a dynamic parameter layer, which properties are explained in Section 3.3.3. Because this is not a classification network and relies on the other parts of the network, the convolutional neural network is not trained separately, but is added to the rest of the network when training. The entire network is then trained end-to-end using backpropagation. Note that finetuning starts only after the validation accuracy of the entire network is saturated. If finetuning is employed from the start of training the gradient will be noisy due to the noisy weights from the hash function (see Section 3.3.3).

3.3.2 Understanding questions using Gated Recurrent Units

In the baseline system [2], the LSTM units are trained from scratch. In DPPnet however, a pretrained model for question understanding is used to in-
crease the amount of data seen. In the VQA dataset, there are only 750,000 questions. This might seem a lot, but due to the high variance this should ideally be a higher number. Therefore, the question understanding model is pre-initialized using the \textit{skip-thought} vector model \cite{10} which is trained on the BookCorpus dataset \cite{26}, containing 74 million sentences. Note that the \textit{skip-thought} model is trained with GRUs instead of with LSTMs units. GRUs have been shown to perform similarly to LSTMs, but with a simpler model \cite{5}. GRUs only have two instead of four gates and do not require the use of a cell state. This simplifies understanding and increases computational performance. The \textit{skip-thought} model can be trained on all types of recurrent units as long as it is possible to backpropagate through it. Like with the image understanding subsystem, the pretrained language model is finetuned using the same end-to-end backpropagation.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6}
\caption{A visualization of the LSTM and GRU cells. These cells are chained and each word is used as input into the cell. Figures taken from Christopher Olah’s blog\cite{1}.}
\end{figure}

3.3.3 \textit{Fusing modalities and the hashing trick}

To combine the image and question modalities, a dynamic parameter layer is used. The dynamic parameter layer is placed right before the output layer and its preceding hidden layer. The input of the dynamic parameter layer is determined solely by the image features. The weights of the dynamic parameter layer are not trained in the usual way, but are instead determined by the question embedding. The process is shown in Figure 7 in a simplified format.

\footnote{http://colah.github.io/posts/2015-08-Understanding-LSTMs/}
The dynamic parameter layer can also be seen as a matrix-vector multiplication between the image features and the question embedding as follows:

\[
f^0 = W_d(q)f^i + b
\]  

(3)

where \(f^0\) is the output of the dynamic parameter layer, \(f^i\) its input, \(b\) the bias and \(W_d(q)\) the matrix constructed by the hashing of the question embedding \(q\). Note that the input of the dynamic parameter layer \(f^i\) is the image feature vector from the pretrained and finetuned convolutional neural network with an additional hidden layer.

Since the image feature vector \(f^i\) and the output \(f^0\) from the dynamic parameter layer are large due to the amount of hidden units used, the matrix \(W_d\) has to be large as well. The trivial way to generate the matrix \(W_d\) is then to have the decoder (fully-connected layer) in the parameter prediction network generate the entire matrix \(W_d\). However, the network may overfit easily when such a large number of hidden units is used in this fully-connected layer, since its input is not as complex and the number of training samples is still limited, even with pretraining. Therefore, Noh, Seo, and Han \[15\] employ a random weight sharing technique based on hashing \[4\]. This hashing method translates the values in the 1D question embedding to the 2D weight matrix. Since the 2D weight matrix \(W_d\) contains more elements than the 1D question embedding \(q\) contains, an element in \(q\) can be shared across multiple positions in \(W_d\). This way, the amount of unique parameters in \(W_d(q)\) is reduced, while still maintaining the needed size of the matrix. As shown in \[4\], using this sparse hashing technique has almost no negative results on the network performance, but greatly decreases overfitting.

Specifically, hashing works as follows. We need to find all values \(w_{mn}\) at positions \((m, n)\) in matrix \(W_d(q)\), which corresponds to the weight between the \(m^{th}\) output and \(n^{th}\) input unit. We define a hash function \(\phi(m, n)\) that maps a tuple \((m, n)\) to a natural number between 1 and \(K\) where \(K\) is the dimensionality of the question embedding \(q\). Thus:

\[
w_{mn} = q_{\phi(m, n)} \times \xi_{\phi(m, n)}
\]  

(4)

where \(\xi(m, n) : \mathbb{N} \times \mathbb{N} \rightarrow \{+1, -1\}\) is another hash function, independent of \(\phi\). It removes the bias of the resulting matrix \(W_d\) being always positive by arbitrarily – albeit deterministically – multiplying some weights by \(-1\) \[4\]. In practice, the implementation xxHash\[^2\] is used as the hashing functions \(\phi\) and \(\xi\). This hashing algorithm is very fast with a maximum hashing speed of

\[^2\] http://cyan4973.github.io/xxHash/
13.8GB/s on consumer hardware. In backpropagation, the hashing function is reversed by memoization during the forward-pass to obtain the original position \((m, n)\).

### 3.3.4 Results on the VQA task

In Table 3 the results for DPPnet are shown. As a comparison, the best result from [2] is shown as well. Several variations of DPPnet are tested:

1. DPPnet with finetuning of the convolutional neural network
2. DPPnet with a fixed convolutional neural network
3. As above, with a downsized dynamic parameter layer

These variations have been tested to show the respective performance of the extra features that DPPnet adds. Furthermore, the use of finetuning and the large dynamic parameter layer require at least 12GB of VRAM (i.e. a GTX Titan X or Tesla M40). When finetuning is removed and the large dynamic parameter layer is reduced, the network fits into 6GB of VRAM, enabling its use on less state-of-the-art hardware. Naturally, this decreases performance somewhat. In the downsized network, the hash size is decreased from 40000 to 10240 and the amount of linear units in the dense part of the network is decreased from 2000 to 1024.

<table>
<thead>
<tr>
<th>Method</th>
<th>All</th>
<th>Yes/No</th>
<th>Number</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM Question + Image</td>
<td>53.74</td>
<td>78.94</td>
<td>35.24</td>
<td>36.42</td>
</tr>
<tr>
<td>DPPnet</td>
<td>57.22</td>
<td>80.71</td>
<td>37.24</td>
<td>41.69</td>
</tr>
<tr>
<td>DPPnet fixed CNN</td>
<td>56.74</td>
<td>80.48</td>
<td>37.20</td>
<td>40.90</td>
</tr>
<tr>
<td>DPPnet fixed CNN, downsized</td>
<td>56.11</td>
<td>79.88</td>
<td>36.84</td>
<td>40.18</td>
</tr>
</tbody>
</table>

Table 3: Results for DPPnet [15] on the open-ended task. Tested on test-dev dataset. As a comparison, the best result from [2] is shown as well.
To improve upon the results obtained by Antol et al. [2] and Noh, Seo, and Han [15], several techniques were used. As mentioned previously, the main focus was to increase performance on counting questions, i.e. questions with numerical answers. Some of the techniques greatly improved performance, while others were not as successful. In this section, all strategies are explained in more detail. Their results are shown in Chapter 6.

4.1 OBJECT PROPOSALS

Most of the questions that have numerical answers are about counting objects in the images. There are exceptions, such as what number is written on the train?, but these questions are not the focus. To improve performance for object counting, an analysis is performed to see whether segmentation pre-processing could be useful. First, ground truth segmentations are used to measure the maximum reachable performance using segmentation techniques. Naturally, these segmentations are not available for the test set and can thus not be used in practice. Next, Edgebox [27] and Deepbox [11] were tested to get a possible performance increase that could also be reached on the test set.

4.1.1 Ground truth annotations

The MSCOCO dataset includes ground truth annotations, an example of which is shown in Figure 8. In theory, these annotations could be useful for counting the amount of a certain object in an image. To test this hypothesis, a masked version of each separate segmentation is fed through the convolutional neural network to obtain its features, similarly to the normal process for the unmasked images. The features for each of the segmentations are concatenated and fed through the normal network as explained in Section 3.3.3, the only difference being the increased size of the output of the convolutional neural network. A fixed amount of segmentations is chosen (25 in our experiments) and if an image has fewer segmentations, the concatenated feature vector is zero-padded. Of course, since the ground truth annotations are only available for the training set, we ideally need to create these segmentations ourselves. The results of using ground truth annotations as an additional input into the classification network are shown in Chapter 6.

4.1.2 Edgeboxes and Deepboxes

Several state-of-the-art techniques exist for segmenting objects in photos, often also called object proposals. One of these techniques is called Edgebox and it is able to generate bounding box object proposals using edge detection [27]. Since Edgebox generates bounding box object proposals, they are especially suited for objects that are rectangular. In some cases, such as in Figure 8, the object is shaped oddly, which is harder to segment with Edgebox. Nevertheless, using Edgebox to segment objects is a proven strategy and works well for natural images in a non-controlled environment such as in the MSCOCO dataset [27].
Often, the Edgebox strategy generates hundreds of object candidates. The algorithm scores and sorts these according to the number of contours that are wholly contained within the image. Deepbox [11] uses a different scoring metric: it trains a convolutional neural network that reranks the proposals that Edgebox made. Using Deepbox, the same recall is achieved with four times less proposals.

Since a lot of the proposals have a large overlap and we want the proposals to be unique – so certain objects are not counted twice – non-maximum suppression is applied to the sorted proposals. A box is removed if its overlap with another box is greater than a certain threshold (in this case, 50%, based on Zitnick and Dollár [27]). Boxes are removed in bottom-up order – i.e. the boxes with the worst score are removed first. After applying non-maximum suppression, the top 25 scored proposals are used. Similarly to the ground truth annotations, object proposal masks are created and used as input for the convolutional neural network. The resulting feature vectors are concatenated for all 25 proposals and the full image. The results of using Edgeboxes and Deepboxes as an additional input into the classification network are shown in Chapter 6.

4.2 Concept Detectors

The ground truth annotations do not only contain the segmentation information, but also the actual label – i.e. what is the object. The MSCOCO dataset contains 80 classes, some of which have a high prevalence (e.g. people) and some of which occur only very rarely (e.g. parking meters). All segmentations have a label that associates it with one of these classes. This information is used to train a concept detector that takes as input the segmentation masks and classifies it as one of the 80 MSCOCO classes. During training and testing of the actual question answering network, the predicted classes of the concept detector network are used as additional input. The inputs that were used previously – activations of the pretrained CNN on the Edgebox segmentations – are not used in this model since they are redundant as we already have the predicted classes. The activations of the pretrained CNN on the full image are still used.
4.2.1 Network structure and training method

The network uses a pretrained GoogLeNet model [22] based on the Inception architecture. GoogLeNet was chosen for its high accuracy and the fact that it uses 12 times fewer parameters and thus fewer VRAM than the next-best ImageNet submission. The last layer of the pretrained network is removed and two new randomly initialized fully-connected layers are attached with 1024 and 80 units respectively. A softmax non-linearity is applied to the last layer. The full network structure is shown in Table 4.

<table>
<thead>
<tr>
<th>type</th>
<th>patch size</th>
<th>stride</th>
<th>output size</th>
<th>params</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>–</td>
<td>–</td>
<td>$224 \times 224 \times 3$</td>
<td>–</td>
</tr>
<tr>
<td>convolution</td>
<td>$7 \times 7$</td>
<td>2</td>
<td>$112 \times 112 \times 64$</td>
<td>2.7K</td>
</tr>
<tr>
<td>max pool</td>
<td>$3 \times 3$</td>
<td>2</td>
<td>$56 \times 56 \times 64$</td>
<td>–</td>
</tr>
<tr>
<td>convolution</td>
<td>$3 \times 3$</td>
<td>1</td>
<td>$56 \times 56 \times 192$</td>
<td>112K</td>
</tr>
<tr>
<td>max pool</td>
<td>$3 \times 3$</td>
<td>2</td>
<td>$28 \times 28 \times 192$</td>
<td>–</td>
</tr>
<tr>
<td>inception</td>
<td>–</td>
<td>–</td>
<td>$28 \times 28 \times 256$</td>
<td>159K</td>
</tr>
<tr>
<td>inception</td>
<td>–</td>
<td>–</td>
<td>$28 \times 28 \times 480$</td>
<td>380K</td>
</tr>
<tr>
<td>max pool</td>
<td>$3 \times 3$</td>
<td>2</td>
<td>$14 \times 14 \times 480$</td>
<td>–</td>
</tr>
<tr>
<td>inception</td>
<td>–</td>
<td>–</td>
<td>$14 \times 14 \times 512$</td>
<td>364K</td>
</tr>
<tr>
<td>inception</td>
<td>–</td>
<td>–</td>
<td>$14 \times 14 \times 512$</td>
<td>437K</td>
</tr>
<tr>
<td>inception</td>
<td>–</td>
<td>–</td>
<td>$14 \times 14 \times 512$</td>
<td>463K</td>
</tr>
<tr>
<td>inception</td>
<td>–</td>
<td>–</td>
<td>$14 \times 14 \times 528$</td>
<td>580K</td>
</tr>
<tr>
<td>inception</td>
<td>–</td>
<td>–</td>
<td>$14 \times 14 \times 832$</td>
<td>840K</td>
</tr>
<tr>
<td>max pool</td>
<td>$3 \times 3$</td>
<td>2</td>
<td>$7 \times 7 \times 832$</td>
<td>–</td>
</tr>
<tr>
<td>inception</td>
<td>–</td>
<td>–</td>
<td>$7 \times 7 \times 832$</td>
<td>1072K</td>
</tr>
<tr>
<td>inception</td>
<td>–</td>
<td>–</td>
<td>$7 \times 7 \times 1024$</td>
<td>1388K</td>
</tr>
<tr>
<td>avg pool</td>
<td>$7 \times 7$</td>
<td>1</td>
<td>$1 \times 1 \times 1024$</td>
<td>–</td>
</tr>
<tr>
<td>dropout (0.4)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>linear</td>
<td>–</td>
<td>–</td>
<td>$1 \times 1 \times 1024$</td>
<td>1024K</td>
</tr>
<tr>
<td>dropout (0.4)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>linear</td>
<td>–</td>
<td>–</td>
<td>$1 \times 1 \times 80$</td>
<td>82K</td>
</tr>
<tr>
<td>softmax</td>
<td>–</td>
<td>–</td>
<td>$1 \times 1 \times 80$</td>
<td>–</td>
</tr>
</tbody>
</table>

Table 4: GoogLeNet architecture used for detecting concepts in images [22].

In total, this network contains less than 7 million parameters, which is not much considering the amount of layers. According to Szegedy et al. [22], reducing the amount of parameters severely reduces overfitting and decreases training time and memory usage. A trained GoogLeNet only takes up 28MB of space when saved and is extremely fast at testing time, enabling its use on mobile devices and embedded systems.

The network is trained on the segmentations using gradient descent with Nesterov momentum [21] for 25 epochs. For the first 10 epochs, the weights of the convolutional layers are locked to prevent the noisy gradients from the randomly initialized fully-connected layers from changing the pretrained weights too much. Cross-entropy loss is used and Top 1 accuracy is used for validation. The segmentation masks are stretched to use the entire $224 \times 224$ image space (aspect ratio is retained). This removes scale variance and reduces overfitting. Furthermore, segmentations that have a surface smaller than 500 pixels are removed from training as they provide no meaningful
information. The biases in the first convolutional layer are set to 0 to ensure the black background causes no activations. Finally, since the class balance is so skewed – the most prevalent class occurs 185,316 times, while the rarest class occurs only 135 times – the amount of data per epoch is limited to 5000 per class. Note that if a class has more than 5000 samples, each epoch different data is shown to the network. Effectively, this means samples in underrepresented classes will be shown to the network more than samples in large classes.

4.2.2 Testing methodology

During the testing phase, two different methods are employed. In the first case, the concept detector is applied on the object proposals that are generated by Edgebox and Deepbox. The resulting softmax vectors are summed per image for each of the object proposals. The resulting vector is used as input for the question answering model.

In the second method, which reaches a higher performance, the concept detector is applied on the full image, thereby removing the dependency on the quality of the segmentations. The softmax layer is removed and the raw activations of the 80-dimensional layer are used as input for the question answering model. In Chapter 6, the results for both methods are shown.

4.3 Regression

Another method that was tried was using regression instead of classification. The idea here was that the relation would be somewhat linear between on one hand, the image features of the segmentations and the full image and on the other hand, the numerical output. Regression could – in theory – be better at modeling this relationship. The softmax layer was removed and instead of k units – one for every class – a single unit was used as regression output. The loss for determining the gradient was changed from cross-entropy to mean squared error – since we try to obtain a posterior mean instead of a probability distribution. The regression model was trained only on questions that started with how many... and that had numerical answers. The results (Table 9) were not as expected, so regression was not used in the final model.

4.4 Postprocessing repair

For some questions, such as how many... questions, we know that the answer should be numerical. Often, the network will predict other answers as well, such as the string equivalents of the numeric digits, e.g. one instead of 1. For questions that start with are there..., does this..., and so on, we expect as an answer yes, no or a word that exists in the question. For example, the question does this image contain a cat? always has to be answered by either yes or no, while the question is there a cat or a dog in this image? should be answered with either cat, dog, yes or no. Using a simple rule-based program, questions that start with how many always get the numerical answer that generates the highest softmax response in the network. For questions that expect a closed answer, the answer is selected with the highest softmax response that is either yes, no or any word that exists in the question. Performing this postprocessing repair boosts performance slightly, as shown in Chapter 6.
EXPERIMENTS

5.1 DATASET Splits

The VQA dataset is split into six different parts:

- **train2014**: The training set. Answers are available.
- **val2014**: The validation set. Answers are available.
- **test-dev2015**: The development testing set. Answers are not available but evaluation can be done as often as desired on the online leaderboard on the VQA website.
- **test-standard**: Testing set. Answers are not available. Evaluation can only be done five times through the online leaderboard on the VQA website. This is the dataset the VQA organizers request researchers to show.
- **test-challenge**: Testing set used for determining the winner of the VQA challenge. Results from this set are not made public until the challenge has ended.
- **test-reserve**: Testing set. Results are never made public. This test set only exists to ensure participants are not overfitting on other test sets.

Most of the results reported in this thesis are evaluated on **test-dev2015**. Some results, such as the ground truth annotation network are evaluated on the validation set, since it requires extra information while testing. The final result is reported on **test-standard**, as requested by the VQA organizers. Performance between **test-dev2015** and **test-standard** is shown to be comparable.

5.2 EXPERIMENTS

Several experiments have been conducted to test the performance of the aforementioned subsystems. First, the performance of using ground truth annotations as an extra input to the question answering network is shown. This network is trained on the training set and tested on the validation set. This system cannot be tested on the test set, since the ground truth annotations are not available for it. Note that the ground truth annotations system was tested to see what the maximum attainable performance would be when segmenting objects in the image. It cannot be used in practice for new images due to the lack of segmentations for these images. Afterwards, a comparison is given between this system and a system that generates the segmentations itself, i.e. by using the edgebox and deepbox systems. This system is tested both on the validation set and the test-dev split.

Next, performance is shown for the system that uses the concept detectors. Results for this system are reported on test-dev only. Results for post-processing repair are shown only on the final model (which uses the concept detector features). The scores of the final model are reported on both test-dev and test-standard.

All reported scores are on the open-ended metric.
5.3 IMPLEMENTATION

For the question answering model, an existing implementation\textsuperscript{1} was adapted, written in Torch \textsuperscript{6}. All other code was written in Python and Theano. The concept detector network uses Lasagne\textsuperscript{2} as its neural network library. All models were trained and tested on a computing cluster with Tesla K20 GPU’s. The final model was trained on a Tesla M40 GPU to accommodate for the extra requirement in memory.

\textsuperscript{1} https://github.com/HyeonwooNoh/DPPnet
\textsuperscript{2} https://github.com/Lasagne/Lasagne
6 RESULTS

6.1 PRIOR RESULTS

Results from prior works are repeated here for convenience.

<table>
<thead>
<tr>
<th>Method</th>
<th>All</th>
<th>Yes/No</th>
<th>Number</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM Question + Image</td>
<td>53.74</td>
<td>78.94</td>
<td>35.24</td>
<td>36.42</td>
</tr>
<tr>
<td>DPPnet</td>
<td>57.22</td>
<td>80.71</td>
<td>37.24</td>
<td>41.69</td>
</tr>
<tr>
<td>DPPnet (fixed CNN)</td>
<td>56.74</td>
<td>80.48</td>
<td>37.20</td>
<td>40.90</td>
</tr>
<tr>
<td>DPPnet (fixed CNN, downsized)</td>
<td>56.11</td>
<td>79.88</td>
<td>36.81</td>
<td>40.18</td>
</tr>
</tbody>
</table>

Table 5: Results for prior works [2] and [15]. Tested on test-dev2015.

6.2 SEGMENTATIONS

In Figure 9 some example questions and their answers are shown for various segmentation strategies. In the first and second images, we see that automatic segmentation techniques such as Edgeboxes and Deepboxes have trouble segmenting the objects, while in the third image, this seems to be no problem. This makes sense when realizing that Edgeboxes and Deepboxes create rectangular object proposals, which suits the third picture very well, but the first and second less so. With ground truth object proposals (shown as the colored overlay in the images) all answers in the examples are given correctly.

![Figure 9: Ground truth annotations and questions for some images in the MSCOCO dataset. Correct answers given in bold.](image)

In the table below, numerical results are shown for ground truth annotations, Edgeboxes and Deepboxes.
6.3 Concept Detectors and Post-processing Repair

The concept detector network reaches 87% Top 1 accuracy on the validation set. In Figure 10 we can see some example questions and images with the answers given by the regular DPPnet and the DPPnet with concept detector information. In the first image, the bias of DPPnet towards more often occurring answers can be seen (see Figure 3). Using concept detectors, the answer is closer to the truth, but still not correct (note that the soft rejection metric in Equation 1 does not score in terms of numerical proximity). In the second and third images, we can see the disadvantage of building a scale invariant system. The concept detector activations for both images are almost equal, caused by the larger objects in the third image compared to the second image.

<table>
<thead>
<tr>
<th>Method</th>
<th>All</th>
<th>Yes/No</th>
<th>Number</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPPnet (fixed CNN, downsized)</td>
<td>51.94</td>
<td>78.34</td>
<td>33.66</td>
<td>36.77</td>
</tr>
<tr>
<td>Ground truth annotations</td>
<td>52.29</td>
<td>78.34</td>
<td>36.46</td>
<td>36.77</td>
</tr>
<tr>
<td>Edgebox</td>
<td>52.08</td>
<td>78.34</td>
<td>34.97</td>
<td>36.77</td>
</tr>
<tr>
<td>Deepbox</td>
<td>52.16</td>
<td>78.34</td>
<td>35.36</td>
<td>36.77</td>
</tr>
</tbody>
</table>

Table 6: Results for using the ground truth annotations and for using the edgebox and deepbox systems. Tested on val2014; scores not comparable with test-dev results.

In the table below, numerical results are shown for the regular DPPnet and the DPPnet with concept detector activation input, both on the full image and on the deepbox segmentations separately. It also shows the results for the finetuned, non-downsized DPPnet.

Figure 10: Ground truth annotations and questions for some images in the MSCOCO dataset. Correct answers given in bold.
### Table 7: Results for using the concept detectors with/without post-processing compared to DPPnet. Tested on test-dev 2015.

<table>
<thead>
<tr>
<th>Method</th>
<th>All</th>
<th>Yes/No</th>
<th>Number</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPPnet (fixed CNN, downsized)</td>
<td>56.11</td>
<td>79.88</td>
<td>36.81</td>
<td>40.18</td>
</tr>
<tr>
<td>Concept detectors on deepbox segm. (fixed CNN, downsized)</td>
<td>56.13</td>
<td>79.99</td>
<td>36.87</td>
<td>40.16</td>
</tr>
<tr>
<td>Concept detectors on full image (fixed CNN, downsized)</td>
<td>56.34</td>
<td>79.99</td>
<td>37.31</td>
<td>40.45</td>
</tr>
<tr>
<td>Concept detectors on full image (fixed CNN, downsized, +pp repair)</td>
<td>56.45</td>
<td>80.03</td>
<td>37.46</td>
<td>40.66</td>
</tr>
<tr>
<td>DPPnet (finetuned, no downsizing)</td>
<td>57.22</td>
<td>80.71</td>
<td>37.24</td>
<td>41.69</td>
</tr>
<tr>
<td>Concept detectors on full image (finetuned, no downsizing, +pp repair)</td>
<td>58.01</td>
<td>80.89</td>
<td>38.03</td>
<td>42.44</td>
</tr>
</tbody>
</table>

### Table 8: Results for using the concept detectors with post-processing repair compared to DPPnet. Tested on test-standard.

<table>
<thead>
<tr>
<th>Method</th>
<th>All</th>
<th>Yes/No</th>
<th>Number</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPPnet (finetuned, no downsizing)</td>
<td>57.36</td>
<td>80.28</td>
<td>36.92</td>
<td>42.24</td>
</tr>
<tr>
<td>Concept detectors on full image (finetuned, no downsizing)</td>
<td>58.13</td>
<td>80.83</td>
<td>38.09</td>
<td>42.67</td>
</tr>
</tbody>
</table>

### 6.4 Regression

<table>
<thead>
<tr>
<th>Method</th>
<th>All</th>
<th>Yes/No</th>
<th>Number</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification</td>
<td>52.29</td>
<td>78.34</td>
<td>36.46</td>
<td>36.77</td>
</tr>
<tr>
<td>Regression</td>
<td>52.23</td>
<td>78.40</td>
<td>34.84</td>
<td>37.01</td>
</tr>
</tbody>
</table>

### Table 9: Results for using regression instead of classification for questions with numerical answers. Tested on val2014 with ground truth object proposals; scores not comparable with test-dev results.

### 6.5 In Challenge Context

In the context of the VQA challenge, using concept detectors as an additional input to the classification network scores 5th place out of 30 on numerical answers, tested on the test-standard dataset split. The #1 achieved a score of 39.18, which is a difference of 1.09 with our method. Places #1 through #4 are primarily focused on attention-based systems, although at the time of writing only the titles of the methods are available and not the papers.
Table 6 shows the results for using Edgebox and Deepbox segmentations as additional input into the classification network. We can see that using the ground truth annotations helps in improving the performance, especially for numerical questions (33.66 vs. 36.46). Using Edgeboxes or Deepboxes instead of the ground truth annotations still increases performance compared to the baseline. Obviously they do not perform as well as the ground truth annotations. The differences in the mean scores for the All and Number categories are all significant (smallest difference: Wilcoxon signed-rank, $z = -1.72$, $p = 0.043$). We can conclude from this that Deepboxes work significantly better for our use case than Edgeboxes. This makes sense considering the amount of segmentations that are used is limited and Deepbox reorders the segmentations, causing our system to select ‘better’ segmentations.

As can be seen from Table 7, post-processing increases overall performance on a fixed, downsized CNN by 0.11 (Wilcoxon signed-rank, $z = -1.93$, $p = 0.027$). We can also see that the concept detectors on the full image perform significantly better (Wilcoxon signed-rank, $z = -2.8$, $p < 0.001$) than on segmentations, with a score of 56.34 versus 56.13. This difference is even bigger for Number questions. In Table 8 we can see the results for the concept detectors versus the baseline DPPnet on test-standard. The difference in scores for All questions is 57.36 versus 58.13. The improvement for Number questions only is even larger: 36.92 to 38.09. The overall improvement on All questions is significant with a $z = -4.6$ and a p-value $p < 0.001$. From this we can conclude that using concept detectors as an additional input into the network improves performance. Using concept detectors on the full image is better than using it on the segmentations. Furthermore, post-processing repair improves performance.

By using activations of concepts as an additional input, we give the network a measure of ‘how much’ of a certain concept is present in the image. Of course, some questions concern concepts that are not in the MSCOCO segmentation dataset. The concept detectors will have no effect on these type of questions. Furthermore, the concept detector activations have trouble distinguishing between one large concept and two smaller concepts. This is due to the scale invariance that is introduced by stretching the concepts to the full canvas. Still, stretching gives a better performance than using the original concept size, since there will be too much background in the image and the concept will be very small after downsizing the full mask. Quantitatively, stretching gives the concept detector network a Top 1 accuracy of 87% while not stretching reaches only an accuracy of 64%. Still, ideally the network would have some method of differentiating between one large concept and two smaller concepts. If the quality of automatic object segmentations would be better, this could be used to solve this.

Even after all improvements mentioned, the performance of automatic visual question answering is not nearly as high as human performance. Especially on counting questions, machine performance is lacking (38.09) while human performance is very accurate (86.32). One of the possible reasons for this is because humans are better at utilizing the context of an image when segmenting it into separate objects. The segmentation systems used, such as Edgeboxes and Deepboxes are rather contextless in the sense that they have no understanding of what it is they are looking at. They use edge detection to find features important for segmentation, but this is limited and as such...
these systems have a hard time segmenting more complex objects. Furthermore, convolutional neural networks struggle with recognizing objects that are very small. In the concept detector, smaller objects are removed from training since they severely decrease performance. However, the network sees very few smaller objects because of this. Humans have less of a problem with smaller objects. Nevertheless, postprocessing repair and concept detectors do improve performance on counting questions significantly and if research into the topic continues reaching and exceeding human performance might be possible one day.
FURTHER WORK

As can be seen from the results in Table 7, concept detectors work a lot better when run on the full image as opposed to running them on segmentations. This can be explained by the bad quality of the segmentations generated by Edgebox and Deepbox. These segmentations often have a large overlap which skews counting results. If the segmentations would be better, this would remove the problem of scale invariance caused by running the concept detectors on the full image. Newer, more sophisticated segmentation techniques such as U-net [17], YOLO [16] and fully convolutional networks [14] often gain a much higher segmentation performance by not just proposing bounding boxes but by actually segmenting the image itself. It would be interesting to see what results could be achieved when running the concept detectors on the output of these segmentation networks.

Another interesting topic of research would be how well the system would perform in a real world application such as the mobile device usecase as discussed in the introduction. The created system should be fast and small enough to be run on a mobile device. Further research could be conducted to look into this.

8.1 ACKNOWLEDGEMENTS

Thanks to Maaike de Boer and Tom Heskes for their advice during the research and thesis writing process. Thanks to Pim Schellart for providing access to the astrophysics GPU cluster, enabling faster prototyping.


