MASTER THESIS

Bc. Jakub Hajič

Visual Question Answering

Institute of Formal and Applied Linguistics

Supervisor of the master thesis: RNDr. Milan Straka, Ph.D.
Study programme: Informatics
Study branch: Artificial Intelligence

Prague 2017
I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In Prague, 21. 7. 2017  ..............................................
Title: Visual Question Answering

Author: Bc. Jakub Hajič

Institute: Institute of Formal and Applied Linguistics

Supervisor: RNDr. Milan Straka, Ph.D., Institute of Formal and Applied Linguistics

Abstract: Visual Question Answering (VQA) is a recently proposed multimodal task in the general area of machine learning. The input to this task consists of a single image and an associated natural language question, and the output is the answer to that question. In this thesis we propose two incremental modifications to an existing model which won the VQA Challenge in 2016 using multimodal compact bilinear pooling (MCB), a novel way of combining modalities. First, we added the language attention mechanism, and on top of that we introduce an image attention mechanism focusing on objects detected in the image ("region attention"). We also experiment with ways of combining these in a single end-to-end model. The thesis describes the MCB model and our extensions and their two different implementations, and evaluates them on the original VQA challenge dataset for direct comparison with the original work.

Keywords: deep learning, image processing, natural language processing, multimodality, question answering, visual question answering
I would like to thank my supervisor, RNDr. Milan Straka, Ph.D., for his help, expertise and patience during the course of this work. Furthermore, I wish to thank my family and friends for their never-ending support and encouragement. I would also like to thank Dr. Mario Fritz from the Max-Planck-Institut für Informatik for introducing me to this topic. And finally, I would like to thank the countless contributors to all the open-source tools used throughout this work.
## Contents

1 Introduction .................................................. 3
   1.1 Detailed task description ................................ 4
   1.2 Possible applications .................................... 5

2 Related work .................................................. 7
   2.1 Building blocks ........................................... 7
      2.1.1 Convolutional neural networks ....................... 7
      2.1.2 Recurrent neural networks ....................... 8
   2.2 VQA approaches ........................................... 11
      2.2.1 Original approaches .............................. 11
      2.2.2 Deep network approaches ....................... 12

3 Datasets ....................................................... 14
   3.1 VQA ..................................................... 14
   3.2 VQA v2 .................................................. 15
   3.3 Visual Genome .......................................... 16
   3.4 DAQUAR ................................................ 16

4 Multimodal compact bilinear model ......................... 19
   4.1 Image processing .......................................... 19
   4.2 Language processing ...................................... 19
      4.2.1 Questions .......................................... 19
      4.2.2 Answers .......................................... 19
   4.3 Multimodal compact bilinear pooling .................... 20
      4.3.1 Count sketch algorithm ............................ 20
   4.4 Spatial attention ......................................... 21

5 Our method and implementation ............................. 24
   5.1 Language attention ........................................ 24
      5.1.1 Fully connected language attention (L-FC) ....... 24
      5.1.2 Strict linear language attention (L-LFC) ........ 26
      5.1.3 MCB language attention (L-MCB) ............... 26
   5.2 Region attention & combinations ........................ 27
      5.2.1 Attention mechanisms for regions .................. 28
      5.2.2 Three-way cascaded MCB at the end (LR-MCB) .... 29
      5.2.3 Fully connected combination (LR-FC) ............ 30
      5.2.4 Separate question processing stack (LR-Sep) .... 30
   5.3 Motivation for reimplementation in Tensorflow .......... 34
      5.3.1 Data parallelism ..................................... 34
      5.3.2 Documentation ...................................... 34
      5.3.3 Modules ........................................... 35
      5.3.4 Problems with Tensorflow .......................... 35
   5.4 Tensorflow implementation details ....................... 36
      5.4.1 Model ............................................... 36
      5.4.2 Evaluation .......................................... 38
1. Introduction

Visual Question Answering is an example of recently emerging multi-modal tasks. The input consists of a color image and a natural language question (English in the datasets we use). The question is related to the image, asking about properties of objects, relationships between objects or activities happening in the image (for examples see Figure 1.1). The answer to be given is also human readable, in state-of-the-art systems mostly just one word or phrase but potentially a fully-formed sentence.

Most competitive approaches to this task utilize deep learning in some way, capitalizing on recent progress in both image processing and natural language processing or understanding. Many of the approaches include an existing deep architecture for extracting a description vector for the input image and question separately and one or several ways of combining these vectors. See Chapter 2.

Figure 1.1: Sample images and questions from the VQA dataset. Figure taken from [Antol et al., 2015].
for an overview of these approaches and Chapter 3 for an overview of available datasets.

In our work we build on these existing approaches, extend them further and experiment with various methods of combining modalities. As a starting point we take the approach of Fukui et al. [2016], which won the VQA Challenge in 2016 [Antol et al., 2015]. It utilizes a spatial attention over the image and a LSTM network over the question. We combine this approach with two novel ideas, the somewhat popular “language attention”, notably used in machine translation by Bahdanau et al. [2014] and also proposed in VQA by Lu et al. [2016], and the idea of a secondary spatial attention focusing on regions with automatically detected objects. For a full description of our work, see Chapter 5. In Chapter 6 we discuss the results of our experiments, and in Chapter 7 we give details on how to run them using the attached code. Lastly in Chapter 8 we present our conclusions and look at possible future work.

All work described in Chapter 5, Chapter 6 and Chapter 7 was performed by the thesis author, except when explicitly mentioned (e.g., use of publicly available code or pre-trained models).

### 1.1 Detailed task description

In this work, we concentrate on the task as given in the VQA challenge, i.e., on the VQA dataset (see Chapter 3 for details about the dataset). Images in this dataset are from the MS COCO dataset (Common Objects in COntext) [Lin et al., 2014], which means (generally speaking) that the images do not contain single objects, but rather complex scenes. The authors of MS COCO focused on finding objects in what they call “non-iconic” images, i.e., those that are not typical in single object images. For instance, when searching for images of a bicycle on the internet, one will find out that most of the results will contain a profile view of a bicycle, which leads to models overfitting on this particular angle. The authors argue that datasets with “non-iconic” images lead to models which generalize better. Moreover, they search for images of fairly common objects but in “scenes”, such as “library”, “church”, “park”, “market” etc., which gives the common objects context and further improves generalization. This sets the scene for the VQA task – for “iconic” images, the questions would be rather limited, and difficult to stray from properties of the particular main object. Whereas in these scene-like images the relationships between multiple objects and common objects in non-standard poses or viewpoints aid generalization.
The questions in the dataset of Antol et al. [2015] were collected with a goal in mind: to avoid as much as possible trivial questions or those which do not rely on the image at all (or only very little). The authors have experimented with various ways to elicit these questions from human subjects, settling on the following setup (direct quote, Antol et al., 2015):

“We have built a smart robot. It understands a lot about images. It can recognize and name all the objects, it knows where the objects are, it can recognize the scene (e.g., kitchen, beach), people’s expressions and poses, and properties of objects (e.g., color of objects, their texture). Your task is to stump this smart robot! Ask a question about this scene that this smart robot probably can not answer, but any human can easily answer while looking at the scene in the image.”

Furthermore, annotators were asked to create questions which require the image in order to be answered and which are dissimilar to questions already asked by other annotators (which were shown to them alongside the image).

The answers were also generated by humans - each question was answered by ten unique annotators, and an answer is considered completely correct if at least three of them provided it, and \( \frac{1}{3} \) or \( \frac{2}{3} \) correct respectively if one or two of them came up with it. This is also reflected in the evaluation code provided by the VQA team. While this may make the evaluation more complex, as it introduces more parameters, it also allows the evaluation a to take into account some of the inherent ambiguity of the task, such as synonyms or possible variations in human color perception. Providing multiple references for evaluation in such circumstances is a common practice also in other fields, such as machine translation [Papineni et al., 2002].

1.2 Possible applications

The VQA task has not been around for very long, the first original work we were able to find comes from 2014. Therefore the performance of the approaches found so far does not warrant widespread commercial application yet; however, there is some potential as described below.

The task is an extension of an object detection or classification, which can be used to aid for example visually impaired people. The extension with questions could help in more complex cases, where the user might need more complex information, or may not be able to focus their device on a single object. Question answering could help find routes in complex outdoor settings (“Which way to the gate?”), answer questions about safety without the user having to approach
or engage in a potentially hazardous situation (“Is the floor clean?”, “Are my drawers closed?” “Can I cross?”) or in everyday situations where knowledge what an object is does not suffice (“Is the kitchen knife facing away from me?”, “Is the computer on?”, etc.).

Another application is familiarization with and navigation in foreign environments, for example for people in countries whose languages they may not speak. Navigation may be difficult when the user does not understand any signs and is unable to communicate with the locals. Applications on the market already exist which translate street signs or text in images in general, but again the question portion could help put the translated text into local cultural or situational context (“How do I get to the embassy?”, “Can I go through this door?”).

If the approach can be successfully extended to video, a further application might be for searching videos. A lot can be extracted by video captioning, but unless the questions are known beforehand the captioning (and potential metadata extraction from natural language captions) can still fall short. For example the sports and betting industries employ a large amount of human annotators who record every situation or action during a game or race, but even such annotation records only limited information, and extracting more types of information would require prohibitively more manpower. A system that not only records everything automatically, but which can also answer questions unforeseen at the time of recording, could save a considerable amount of manpower in the industry and improve the services they provide.

From a research perspective, this task is very interesting (together with other multi-modal tasks, such as captioning) because inevitably, different modalities have to be combined, and there is so far no consensus as to which way is the most efficient to do so or which one provides the best performance. In that sense, this task is an experimental proving ground which may lead to significantly more complex systems with the potential to gather a large amount of diverse information and use it effectively.
2. Related work

2.1 Building blocks

To discuss recent approaches to Visual Question Answering, we must first introduce the most popular building blocks for the models used. These mostly include convolutional neural networks (CNNs) for image processing and various types of recurrent neural networks (RNNs) for the processing of the question.

2.1.1 Convolutional neural networks

The first traces of convolutional layers can be found in the 1980s, in the Neocognitron model [Fukushima, 1980]. These include convolutional layers and an \textit{S-layer} which resembles a max-pooling layer. The authors use it for digit or character recognition, but the network was trained without supervised data. Authors note that the network "self-organized" such that only one neuron activated on the last layer, a different one for each class.

Later on, [LeCun et al., 1998] successfully used convolutional and subsampling layers (with a few fully connected layers at the end) to identify handwritten digits in a supervised setting, trained with the backpropagation algorithm [Linnainmaa, 1970] and [LeCun, 1985]. The basic approach in the area of using convolutional networks has remained almost untouched today. A lot of successful applications include convolutional layers and subsampling in deep neural network architectures. They are trained using backpropagation on supervised data [Krizhevsky et al., 2012] and [Simonyan and Zisserman, 2014, He et al., 2015].

![Figure 2.1: Example of a convolutional network model. Model and figure taken from LeCun et al. 1998.](image)

One of the most successful additions to the basic models of LeCun et al.
were residual connections, first introduced in He et al. [2015]. These connections address issues of optimizing deeper networks (with more layers) by allowing the gradient to flow through the many layers as well as around them, and during forward passes, they allow to choose whether to pass the data through another layer or around it (see Figure 2.2).

Figure 2.2: Residual connections in very deep convolutional networks. Figure taken from He et al. [2015].

2.1.2 Recurrent neural networks

According to Graves [2012], various forms of recurrent neural networks (RNNs) have been proposed at least since the early 1990s [Lang et al. 1990, Elman 1990, Jordan 1990]. The simplest RNN is a single hidden layer connected to itself, which is initialized with some input, cycled several times and queried for an output. This can equivalently be described in “unrolled” form, where the single hidden layer is represented as several layers in a sequence, while the layers share weights. In cases with a finite number of cycles, this is essentially the same as a feed-forward network with shared weights between layers, therefore enabling backpropagation.

For inputs in the form of a sequence, elements of the sequence are provided as input in each cycle. Similarly, the output can be queried in each cycle to create an output sequence. This can be done simultaneously or the entire input sequence can be read first and then the output is generated. For a graphical overview see Figures 2.3 and 2.4.

However, these basic recurrent networks tend to have issues with propagating gradient through these long chains of layers due to the problem of exploding/vanishing gradient [Hochreiter 1991]. This problem was addressed by Long
Short-Term Memory units (LSTMs) [Hochreiter and Schmidhuber, 1997]. In short, they contain gates which handle how much information (both in forward passes and during training in backpropagation) should flow from the input at each step versus the internal memory (hidden layer), and similarly for the output. This was further improved with the addition of the forget gate in [Gers et al., 2000]. In the simplest topology, there is an input and output at every time step (more generally at every sequence element). This can be expressed as:

\[
\begin{align*}
  f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
  i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
  o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\
  c_t &= f_t \circ c_{t-1} + i_t \circ \tanh(W_c x_t + U_c h_{t-1} + b_c) \\
  h_t &= o_t \circ \tanh(c_t)
\end{align*}
\]

where:
- $x_t$ represents the input at time $t$,
- $h_t$ is the LSTM output,
• $c_t$ is the cell state,
• $W_s, U_s$ are parameter matrices,
• $b_s$ are bias terms,
• $f_t$ is the forget gate vector,
• $i_t$ is the input gate vector,
• $o_t$ is the output gate vector,
• $\sigma$ represents the sigmoid activation functions
• $\circ$ represents elementwise multiplication (Hadamard product).

The diagrams of a LSTM unit both with and without a forget gate can be seen in Figure 2.5.

![Figure 2.5: Comparison of LSTM units without and with a forget gate. Figures taken from Gers et al., 2000. For detailed descriptions of each operation, see the original work.](image)

A simplification of the LSTM, the Gated Recurrent Unit (GRU), was proposed by Cho et al. [2014]. The difference to LSTMs is the missing output gate and no explicit split of cell state and cell output. More precisely:

$$z_t = \sigma_s(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma_s(W_r x_t + U_r h_{t-1} + b_r)$$
$$h_t = z_t \circ h_{t-1} + i_t \circ \sigma_t(W_h x_t + U_h (r_t \circ h_{t-1}) + b_h)$$

The functions of both the cell state and cell output are carried out by $h_t$. Instead of input, output and forget gates, GRUs use reset ($r_t$) and update ($z_t$) gates. Again, $\sigma$ represents activation functions, in this case $\sigma_s$ is the sigmoid function and $\sigma_t$ is the hyperbolic tangent in the original work.

RNNs capable of carrying information over long distances, such as LSTMs or GRUs, have played a very important role in various NLP tasks, such as tagging...
2.2 VQA approaches

VQA is a relatively recently presented task. The first published papers we know of are from 2014, the VQA challenge [Antol et al., 2015] was first held in 2016. Unless stated otherwise, the approaches (as described below) consider the task to be a classification task – using all (or at least a few thousand most popular) answers in the dataset as class labels.

2.2.1 Original approaches

The oldest approach we know of comes from 2014 [Malinowski and Fritz, 2014a]. The authors attempt to model the probability $P(A = a|Q, W)$, i.e., the probability of the answer given the question $Q$ and image representation $W$ ($W$ stands for “World”). $W$ is a representation of objects in the image and their relations, as extracted by various segmentation and recognition methods. Furthermore, the authors break down the probability as

$$P(A = a|Q, W) = \sum_T P(A = a|T, W)P(T, Q),$$

where $T$ is a latent variable which corresponds to a semantic tree representing the question. The authors further extend the model to allow for multiple interpretations of the image as well:

$$P(A = a|Q, W) = \sum_W \sum_T P(A = a|T, W)P(W, S)P(T, Q),$$

where $S$ are the raw results of the segmentation and recognition. This allows the model to consider multiple possible interpretations of the image.

A simpler approach from 2016 [Kafle and Kanan, 2016] only models the question type (object, color, counting, location) as a latent variable:

$$P(A = a, T = t|Q, I) = \frac{P(I|A = a, T = t, Q)P(A = a|T = t, Q)}{P(I|Q)}$$

where $A$ is the answer, $T$ is the question type, $I$ is the image and $Q$ is the question (or rather a vector representation thereof).
2.2.2 Deep network approaches

Simpler deep approaches focus only on encoding the question and image separately and combining them. Some ignore the sequential structure of the question and simply concatenate the text features (word embeddings) with the image features (a late layer in a CNN) before running a softmax to classify the output \cite{Zhou2015}. A more complex approach utilizes a CNN over the image as well as the question with a softmax classification at the end \cite{Ma2015}. A further refinement includes a CNN for the image features and a LSTM or GRU to encode the question \cite{Malinowski2016}. This work also attempts to generate multi-word answers (applicable to the DAQUAR dataset, see below) using an RNN. Still more complexity was added in \cite{Noh2015} where authors take a representation of the image obtained from a CNN and feed it through a fully connected network – but one of the layers has its weights generated by a GRU over the question.

More complex approaches make use of the concept of attention. In general terms, attention in deep networks is a set of parameters which control the amount of influence individual parts of the input have on the output with respect to another part of the input. A well known example (and one of the earliest uses) is in a machine translation model in \cite{Bahdanau2014}. In this case, the model "pays attention" to each word of the input sequence with respect to the current hidden state of the RNN generating the output sentence.

The models for VQA most often use attention for the image, i.e., "pay attention" to different regions or subimages w.r.t. the input question. \cite{Shih2015} does exactly that over extracted image regions w.r.t. a sum of the word embeddings; the attention is normalized and the vectors representing the regions are summed applying the calculated attention as weights. A more complex approach of \cite{Zhu2015} adds a LSTM over the question instead of a simple sum of the embeddings, and calculates the attention with respect to the output of the LSTM at each timestep. The attention-weighted sum at a particular timestep is then added as an input of the LSTM at the next timestep, in addition to the corresponding word from the question. A similar approach \cite{Yang2015} also calculates the attention multiple times, but instead of word-by-word it reads the entire sentence repeatedly, adding the weighted sum of the image regions to the vector representing the question (in this case the output vector of the LSTM from the last timestep) and repeats the entire process. Furthermore, the attention mechanism can be applied to both the image and the question, as proposed \cite{Lu2016}.
Even more complex approaches include networks that vary their architecture based on the question, such as Neural Module Networks in [Andreas et al. 2015]. The authors first parse the question using a dependency parser and using these dependencies create a symbolic representation composed of only a handful of different types of concepts, such as Attention, Combination, Classification etc. From this representation (more precisely a set of proposal representations), a network is composed of hand-designed modules which correspond to concept types. This network (or set of networks, one for each proposal) then generates the answer.

In the VQA challenge of 2016 [Antol et al. 2015], an overwhelming majority of the top entries were using deep networks, with many of them utilizing some kind of attention mechanism. The winning entry, which our work builds upon, did use attention over the image but also introduced a novel way of combining multiple vector representations (such as attention-weighted image and question), which they named multimodal compact bilinear pooling. In short, instead of other ways like concatenation, element-wise summation or multiplication, MCB approximates outer product, which allows all elements of both vectors (coming form the image and the question) to interact. More details about MCB can be found in Chapter 4.
3. Datasets

3.1 VQA

The VQA dataset [Antol et al., 2015] is used in the VQA challenge, providing a widely accepted comparison among methods. It consists of two subsets, abstract scenes and real images. For our purposes we only use the real images subset, the contents of which are described in Table 3.1.

<table>
<thead>
<tr>
<th></th>
<th>train</th>
<th>validation</th>
<th>test</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>images</td>
<td>82 783</td>
<td>40 504</td>
<td>81 434</td>
<td>204 721</td>
</tr>
<tr>
<td>natural language questions</td>
<td>248 349</td>
<td>121 512</td>
<td>244 302</td>
<td>614 163</td>
</tr>
</tbody>
</table>

Table 3.1: Statistics of the real images subset of VQA dataset.

The dataset also contains mock answers for the purposes of training a multiple-choice rather than an open-ended system, but in this work we are considering the open-ended task only.

The questions in the dataset are pre-processed, i.e. (direct quote from VQA website):  
- Spelling correction (using Bing Speller) of question and answer strings  
- Question normalization (first char uppercase, last char ‘?’)  
- Answer normalization (all chars lowercase, no period except as decimal point, number words → digits, strip articles (a, an, the))  
- Adding apostrophe if a contraction is missing it (e.g., convert "dont" to "don’t")

For the purposes of better insight the dataset is sometimes considered divided by the type of the question – specifically into the following three categories:

- yes/no questions comprise 38.37% of the dataset, with a bias towards ‘yes’ (58.83% of yes/no questions are answered yes),  
- number questions comprise 12.31% of the dataset, with the most popular number answer being ’2’ for 26.04% of the number questions,  
- other questions comprise the remaining 49.32% of the dataset.

This split may suggest some trivial solutions focusing on question types. The authors of the dataset have provided several baselines, including a language-only (i.e., based on the question only) and image-only model. From their results it

is evident that more information can be extracted from the question alone than
the image – their best language-only model reaches an accuracy of 53.74% and
their image model achieves 28.13% accuracy. Overall, their best combined model
reaches an accuracy of 57.75%, so the increase over the language-only model is
relatively small. This issue was partially addressed in the second version of the
dataset.

\subsection{VQAv2}

After the first dataset was used in the summer of 2016 in the first VQA challenge,
the authors of the dataset have published a paper on “lessons learned” from the
challenge [Goyal et al., 2017]. The main issue they discovered was that successful
methods exploited mainly the language input (the questions) and focused less on
the images. In order to address this issue they have extended the dataset with
questions that are identical to those which already existed in the dataset, with
a (somewhat) similar image but which lead to a different answer. Direct quote
from Goyal et al. 2017:

\begin{quote}
Our key idea to counter this language bias is the following – for every (image,
question, answer) triplet \((I, Q, A)\) in the VQA dataset, our goal is to identify an
image \(I_0\) that is similar to \(I\), but results in the answer to the question \(Q\) to become
\(A_0\) (which is different from \(A\)).
\end{quote}

The new dataset consists of exactly the same images as the original VQA
dataset, but almost twice as many questions, as shown in Table 3.2.

<table>
<thead>
<tr>
<th></th>
<th>train</th>
<th>validation</th>
<th>test</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>images</td>
<td>82,783</td>
<td>40,504</td>
<td>81,434</td>
<td>204,721</td>
</tr>
<tr>
<td>natural language questions</td>
<td>443,757</td>
<td>214,354</td>
<td>447,793</td>
<td>1,105,904</td>
</tr>
</tbody>
</table>

\begin{table}
\caption{Statistics of the real images subset of VQAv2 dataset.}
\end{table}

Similar images \((I_0\) candidates) were chosen automatically and annotators se-
lected which one they believed satisfied the criteria (question was relevant but
answer was different). A different set of annotators answered the questions. This
process also led to a re-balancing in terms of answer frequency for typical ques-
tions. According to the authors, the overall entropy weighted by question type
frequencies increased by 56%. Details can be seen in Figure 3.1.

The full VQAv2 dataset was published in April of 2017.
3.3 Visual Genome

The Visual Genome dataset [Krishna et al., 2016] contains about 100,000 images with many different annotations, such as regions and their descriptions or objects. For purposes of training our Visual Question Answering model, we use its question-answer pairs, there are 1,773,258 QA pairs pertaining to 101,174 images (with no explicit splits of the dataset). For an example see Figure 3.2. We use this dataset purely as auxiliary training data, i.e., we perform no evaluation on this dataset.

3.4 DAQUAR

"DAtaset for QUestion Answering on Real-world images" (DAQUAR) [Malinowski and Fritz, 2014b] is a much smaller dataset with 1,449 images and 12,468 question-answer pairs. The images are taken from the NYU Depth Dataset [Nathan Silberman and Fergus, 2012], which includes depth information and is
originally intended for segmentation. However, DAQUAR does not explicitly contain any depth information. Interestingly, results on the DAQUAR dataset are significantly worse than on VQA – the best overall accuracy reached that we know of is 25.74% [Malinowski et al., 2016], less than half the accuracy on the VQA dataset (the model shown in Malinowski et al. [2016] is not far from the state-of-the-art at the time when tested on VQA). We are not aware of an explanation of this phenomenon. Some examples of the images and questions can be found in Figure 3.3[2]
Figure 3.3: Example DAQUAR images. From DAQUAR website.
4. Multimodal compact bilinear model

In this chapter, we discuss Fukui et al. [2016], the winner of the VQA Challenge 2016, in more detail, as our method builds on this model.

4.1 Image processing

Most of the image processing is done using a pre-trained image processing convolutional network, in this case Resnet-152 [He et al., 2015], trained on the ImageNet 2015 challenge dataset [Russakovsky et al., 2015].

The network is not fine-tuned in any way, however to be able to utilize spatial attention the authors use the output of the last 2-dimensional layer (pool5) before the final, fully connected 1000-way classifier. The input images are resized to double that of the original Resnet-152 dimension, i.e., 448×448×3 instead of 224×224×3. This leads to an output of dimension 14×14×2048 for each image, which is precomputed and the parameters of this network are never updated in training.

4.2 Language processing

4.2.1 Questions

To represent the questions, the authors use two embeddings. One of them is a part of the model and is trained from scratch on the actual data. The other is the pre-trained GloVe module [Pennington et al., 2014], which is not updated during training. These are concatenated and followed by a nonlinearity (tanh) and a two-layer 1024-dimensional LSTM. The outputs of both layers are concatenated, which leads to a T×2048 output (where T is the length of the question, limited by an upper bound of 15 in this case, and the beginning of the question is truncated if necessary).

4.2.2 Answers

Because the answers in the dataset are single words or phrases, the task lends itself to an interpretation as a multi-way classification task, rather than a natural
language answer generation task. Thus the authors choose the 3000 most frequent words from the training set and treat it as a 3000-way classification task.

4.3 Multimodal compact bilinear pooling

Several different ways of combining two or more vectors exist in deep neural networks, such as concatenation, elementwise multiplication or addition. In this work, the authors have attempted to increase the possible flow of information by utilizing outer product as a way of combining vectors. However, any fully connected layer between the outer product of $\sim 10^3$-dimensional vectors and a $\sim 10^3$ dimensional layer would contain roughly on the order of $10^9$ parameters ($n \cdot m \cdot o$, where $n, m$ are the dimensionalities of the input vectors and $o$ is the dimensionality of the output), which is impractical to train. Thus the authors opt for a technique to sample the outer product without the necessity of ever calculating it explicitly.

4.3.1 Count sketch algorithm

The authors use the notion of a count sketch [Charikar et al., 2002] known from information retrieval, which is essentially a data structure for storing frequency information about elements of a stream of data. Each item in the incoming stream $S$ is hashed to a set of locations in a count sketch 2D matrix (in each row of this matrix, the item is hashed to a single column), which are either decremented or incremented according to another hash function. The values in the count sketch table which correspond to a particular item from the stream approximate the frequency of that item. If the table has just one row and our stream contains at most $n$ distinct items, this is a projection from an $n$-dimensional space ($\mathbb{N}^n$, an integer count for each item) to an $m$-dimensional space, where $m$ is the number of columns in that one row.

**Theorem 1** (proven by Pham and Pagh [2013]). Let $x, x' \in \mathbb{R}^n$ be the true frequencies of items in two streams $S, S'$ consisting of the same $n$ distinct items, not necessarily with the same frequencies. Further let $h \in \{1..m\}^n$ and $s \in \{-1,1\}^n$. Define $\varphi(x, h, s)$ as the count sketch of stream $S$. Then $\varphi(x, h, s) \ast \varphi(x', h, s) = \varphi(x \otimes x', h, s)$, where $\otimes$ represents outer product and $\ast$ represents convolution.

In this theorem $h$ represents a hash function returning a location in the final count sketch table (single row) for each item in $x$, and $s$ represents a second
hash function determining if the value in that location should be increased or decreased.

This theorem enables us to calculate the count sketch of the outer product by convolving count sketches of the input vectors, which takes quasilinear time ($O(n \log n)$) in the sum of the vector dimensions rather than linear in their product (using Fast Fourier Transform [Cooley and Tukey, 1965]).

Furthermore, it allows us to adjust the dimension of the output – we can have an $m$-dimensional count sketch, and the required fully connected layer (which for the full outer product had $n \cdot m \cdot o$) now has simply $m \cdot o$ parameters. Moreover, it is applicable to vectors of different dimensions ($n_1, n_2$).

The above procedure is also followed in Fukui et al. [2016] by a signed square root operation and $L_2$ normalization. Unfortunately, these steps are given without explicit explanation. Through our experimentation we have not been able to find the purpose of the square root (though if applied it must be signed, otherwise it negates the $\pm 1$ effect of the count sketch). We believe the normalization may aid in training the following layers of the model (as these normalized values in their input will be in the same range all the time, which may not hold for the raw count sketch output which contains an element of (pseudo-)randomness in its hash functions).

As we have mentioned before, the entire purpose of multi-modal compact bilinear pooling (MCB) is to approximate the information one would get from calculating the full outer product. The MCB operation takes two vectors (of dimensions $n_1, n_2$, not necessarily equal) and approximates their outer product with a vector of dimension $m$. The authors experiment with various values of $m$ between 1024 and 32000. They find no improvement in overall accuracy between 16000 and 32000, so they settle on 16000 for input vectors of size 2048.

The simplest model thus consists of image preprocessing using a convolutional network, question preprocessing using an LSTM and a combination of the two using MCB. With this model the authors achieve an overall accuracy of 59.83%, which outperforms models which utilize concatenation or elementwise operations on the vectors (sum or product). A diagram of this model can be found in Figure 4.1.

### 4.4 Spatial attention

A further improvement on top of the above described approach is the concept of spatial attention. It is essentially the same as the concept of language attention.
introduced by Bahdanau et al. [2014], but in two dimensions (x, y in an image) instead of one dimension (t in a sequence of words – a sentence). For language, the elements to which the model “pays attention” are words, arranged in a sequence. In spatial attention, the elements are parts of an image, in this case a regular grid of 14x14 subimages. Therefore, we want a scalar “attention value” (also “associated energy” as it is called in Bahdanau et al. [2014]) for each of these subimages (grid squares).

However, having introduced the concept of MCBs, in this model the authors calculate what they call MCB attention. As expected, it is calculated with respect to the question, more specifically to the output of question preprocessing – in the case of the two-layered LSTMs they use, it is the concatenation of the last output of the two LSTMs. However, instead of a (shallow) neural network to determine the attention value, the authors use the MCB as follows:

Let Q be the vector representation of the question (last outputs concatenated), and let I_{(i,j)} be the vector representation of a particular image region. Recall that after pre-processing the image shape is 14 × 14 × 2048, i ∈ {1..14} and j ∈ {1..14}, and thus I_{(i,j)} ∈ ℝ^{2048}.

Then

\[ E'_{(i,j)} = MCB_{16000}(Q, I_{(i,j)}) \]
\[ E''_{(i,j)} = FC_{512}(E'_{(i,j)}) \]
\[ E_{(i,j)} = FC_{1}(E''_{(i,j)}) \]
\[ A = \text{softmax}(E) \]
where $MCB_{16000}$ represents the MCB operation described above with an output dimension of 16000, and $FC_x$ represents a fully connected layer with $x$ output units. There is an intermediate step ($E''$) which the authors do not explain in detail, but it adds about $8 \cdot 10^6$ parameters, which increases the complexity of the attention model and may give it an opportunity to learn a more complex relationship between the MCB output and the attention value. See Figure 4.2 for a diagram of this model.

![Figure 4.2: MCB model with spatial attention. Figure taken from Fukui et al., 2016.](attachment:image.png)
5. Our method and implementation

Our method extends the previously described MCB method of [Fukui et al. 2016]. As mentioned previously, it consists of two main ideas: language attention and region attention.

Otherwise, we keep the basic approach of the original work. We do not fine-tune the CNN responsible for processing the image, and we treat the task as a 3000-way classification (i.e., we do not attempt any language generation techniques). We also use the same GloVe vectors and train our own custom embeddings.

5.1 Language attention

The idea for this extension comes from the original language attention work by [Bahdanau et al. 2014]. Due to the inherent bias present in the dataset which seems to suggest more information is contained in the question than the image (see the baseline results in [Chapter 3]), we have pursued understanding the question further than [Fukui et al. 2016], who – by choice – concentrate on the image or rather on the novel way of combining the two modalities (MCB).

As with any attention mechanism, we need to specify the elements which we “pay attention to” as well as with respect to what the attention is calculated. In the case of [Bahdanau et al. 2014] this means attention over the input sentence with respect to the most recent generated word of the output. In our case it is attention over words of the question with respect to a vector characterizing the image. Because the image pre-processing phase ends with a 2048-dimensional vector for each of 14×14 regions, we simply take the mean of these vectors to represent the image. As for the words, we use the outputs of the second layer of the two-layer LSTM in the original model.

We have tried multiple variants of language attention, described in more detail in Section 5.1.1, Section 5.1.2 and Section 5.1.3.

5.1.1 Fully connected language attention (L-FC)

In this variant, we follow [Bahdanau et al. 2014] almost exactly. In our case, we embed the image vector and each word vector into a space of equal dimension and
use an element-wise sum of the outputs of those, and add a last fully connected layer over this sum with a single output. These outputs are then run through a softmax. The only thing missing here is a nonlinearity before the softmax layer. A schema of language attention can be found on Figure 5.1.

More precisely:

\[
Q'_k = W_q Q_k + b_q \\
\hat{v}' = W_i \hat{v} + b_i \\
S_k = Q'_k \oplus \hat{v}' \\
E_k = w_s S_k + b_c \\
A = \text{softmax}(E)
\]

where:

- \(Q_k\) is a vector representing each word (\(k\)-th output of the LSTM),
- \(i\) is a vector representing the image,
- \(W_s\) are weight matrices and \(b_s\) bias terms and \(w_s\) is a weight vector.

This implies that:

- \(S_k\) is a vector in the space where \(Q_s\) and \(I\) have been projected into,
- \(E_k\) is a scalar.
Finally, $E$ is the concatenation of $E_*$ into a vector, which is run through a softmax. The final output is then as in [Bahdanau et al., 2014]:

$$c = \sum_{i=1}^{T} A_i Q_i$$

where $T$ is the length of the sentence.

### 5.1.2 Strict linear language attention (L-LFC)

We have also tried a version of this attention with no nonlinearities at all, that means leaving out the softmax entirely ($A = E$). While it may seem contrary to [Bahdanau et al., 2014], the absence of softmax or any kind of normalization of attention values ($E_k$) can allow the model to find the question irrelevant – when the attention value $A_k$ is low for all words (doesn’t even sum to one), this may mean the question is not very relevant. On the other hand, if questions contain different amounts of relevant words, they can all have the same $A_k$; whereas with softmax, two relevant words within the same question must have $A_k \leq 0.5$, while a lone relevant word can have $A_k \leq 1.0$. The softmax-less model may thus allow the network to decide the attention value in a global context, rather than within a single question only.

### 5.1.3 MCB language attention (L-MCB)

The success of [Fukui et al., 2016] suggests the MCB could be a useful way of combining modalities, so we have also tested the MCB as a method for calculating attention, in a similar manner to how the original work uses it to calculate spatial attention over the image:

$$E_k' = MCB_{16000}(Q_k, I)$$
$$E_k'' = FC_{512}(E_k')$$
$$E_k = FC_{1}(E_k'')$$
$$A = \text{softmax}(E)$$

The details can also be seen in Figure 5.2. Note that the language processing layers (LSTMs, embeddings) and the final parts of the attention mechanism (softmax, scalar multiplication) stay the same; the only thing that changes is the way the modalities are combined.

As with the original MCB procedure, we keep the intermediate step. We also keep all of the MCB post-processing, i.e., signed square root and $L_2$ normalization.
5.2 Region attention & combinations

The concept of region attention is inspired by the fact that some objects may be split across multiple parts of the 14×14 grid of the original image. In those cases, the network used to extract the descriptor vector may struggle to produce a useful descriptor of regions which only cover parts of objects. It was pre-trained on an object classification challenge, and it would be followed by a fully connected layer which solves a simpler task (no question input) and has the entire layer of parameters to solve it. However, in our task we are trying to calculate the relevance of the region to the question, and we would intuitively expect a full characterization of the region, rather than an intermediate result.

This led us to create a model where the attention is not paid to grid squares but to the actual, whole objects found in the image. Due to recent progress in object detection, we do not necessarily have to solve the localization task – we can use existing work.

We have chosen the publicly available Faster-RCNN [Ren et al., 2015] model.
including a pre-trained network from Charles Shang (TFFRCNN). This lets us extract bounding boxes for 20 different classes of objects. While this is definitely not enough to locate every object in the image, many of the classes appear in the images quite often, such as “person”, “chair” or “dining table” (each of these classes appear in more than 10% of images from the original MS COCO image dataset; we do not have exact statistics for the VQA dataset). This leads us to believe that the objects detected may be relevant to the question fairly often.

This particular implementation gives us access to bounding boxes as well as their detection scores (between 0.0 and 1.0), and originally runs non-maxima suppression on each category individually. Because we are not interested in concrete categories but rather in all objects in the image, we combine all of the bounding boxes and run non-maxima suppression again, and then take the top 10 bounding boxes by score (in some images 10 such boxes do not reach a suggested threshold of 0.3, and in those cases we consider only those above the threshold).

Furthermore, we need to calculate a vector characteristic for each of these boxes, or “regions”. This is done using a pretrained CNN, in our case the publicly available Inception-ResNet-v2 model downloaded directly from Tensorflow Github, pretrained on the ILSVRC 2012 challenge. We cut it off at the last layer before the fully connected layer which is designed for classification (1001 outputs for 1000 classes and “no class”).

5.2.1 Attention mechanisms for regions

We experiment with two attention mechanisms; one is based on Bahdanau et al. and the other uses the MCB attention. Just like with language attention, this is only in one dimension – but in this case we do not consider any ordering to be significant and do not apply any sequential RNNs over the regions, but rather simply order them by score and let the model decide on how much attention it should pay to which region.

We provide a schema of MCB region attention in Figure 5.3. Note that the descriptor extraction is not part of the model we are training, but is provided for a deeper overview of the region attention mechanism.

The schema is essentially the same as the MCB language attention, just with

---

1. [https://github.com/CharlesShang/TFFRCNN](https://github.com/CharlesShang/TFFRCNN)
2. Non-maxima suppression in this context means that if bounding boxes of detected objects from the same class have high overlap, only the one with higher score is kept.
3. [https://github.com/tensorflow/models/tree/master/slim](https://github.com/tensorflow/models/tree/master/slim)
different inputs – the attention is paid to the regions with respect to the question.
The schema for FC region attention would look just like the schema for FC language attention, just with the inputs switched.

![Figure 5.3: Schema of MCB region attention.](image)

### 5.2.2 Three-way cascaded MCB at the end (LR-MCB)

Because the object classes in our regions do not contain all the possible objects in our images, we believe it is necessary to use the extracted regions as a supplement to rather than a replacement of the original vector representing the image. This creates a challenge beyond the original paper – in this case we are combining at least three different “modalities” or vectors: question vector, image vector and region vector (regardless of attention mechanisms).

Based on [Theorem 1](#) proving that the count sketch of an outer product is a convolution of the individual count sketches, the obvious extension is as follows:
Original:
\[ \varphi(x \otimes x', h, s) = \varphi(x, h, s) \ast \varphi(x', h, s) \]

Extended:
\[ \varphi(x \otimes x' \otimes x''', h, s) = \varphi(x \otimes x', h, s) \ast \varphi(x'', h, s) \]
\[ = (\varphi(x, h, s) \ast \varphi(x', h, s)) \ast \varphi(x'', h, s) \]

where \( \otimes \) represents outer product and \( \ast \) represents convolution.

In our model we use it as a “MCB cascade”: first, we combine two of our modalities and then the result with the last modality. Inspired by residual connections and to avoid interference between the modalities during learning, we have added a residual connection from the initial MCB to the final result. More precisely:
\[ M_1 = MCB(I, Q) \]
\[ M_2 = MCB(M_1, R) \]
\[ M = M_1 \oplus M_2 \]
\[ Ans = \text{ArgMax}(FC_{3000}(M)) \]

A schema of this can be found in Figure 5.4.

5.2.3 Fully connected combination (LR-FC)

We could also leave out MCB altogether and use an universal mechanism similar to the “-FC” construction in language attention, i.e., embed all attention-weighted modalities into the same space and then combine them (an elementwise product in our case). For a visual overview, see Figure 5.5.

5.2.4 Separate question processing stack (LR-Sep)

We have also decided to build a completely separate question processing part of the model for the purpose of extracting region attention. We suspected some interference between information coming from the grid squares and regions could cause difficulty in training. This way we could isolate the two different “modalities” having conflicting influences on parameters of the language part of the model. At the end we combine the attention weighted question in an MCB but separately from the original language attention & spatial attention MCB, and concatenate the two results. For a detailed schema see Figure 5.6.
Figure 5.4: Schema of the three-way MCB model.
Figure 5.5: Schema of the three-way FC integration.
Figure 5.6: Schema of the separate question stack model.
All three model schemas in Figure 5.4, Figure 5.5, Figure 5.6 contain only the parts of the model that we actually train, i.e., they do not include the descriptor extraction for grid squares and regions.

## 5.3 Motivation for reimplementation in Tensorflow

As our work mostly extends the MCB model of Fukui et al. [2016], we have started with their provided implementation in Python & Caffe [Jia et al., 2014]. We were also able to verify its performance directly. However, after several iterations this has proven somewhat impractical for rapid prototyping due to poor documentation of the Python interface for Caffe and a model design which did not allow for much parallelism (apart from running on a GPU). Therefore we have decided to reimplement the model in Tensorflow [Abadi et al., 2015]. In addition to reasons stated above, Tensorflow is also evolving faster and has implementations of some state-of-the-art approaches readily available (for example unrolled RNNs with dynamic length), which cuts down on development time.

### 5.3.1 Data parallelism

In the original work of Fukui et al. [2016], all data is loaded in the first layer of the network, however the layers cannot be executed in parallel. This means the GPU is idle while waiting for I/O operations. According to our measurements 30-40% of the computation time is spent by loading the data. Parallelising this process could thus lead to a significant speedup. While rewriting this model for Tensorflow, we have intentionally left the data loading outside of the network itself, because it is executed by the CPU and thus can run in parallel with the network execution. With a simple use of the Queue class from the multiprocessing module we were able to execute the computation and data loading in parallel and speed up training significantly (the GPU is no longer idle during data loading, that means we save the 30-40% of the original runtime that was devoted to I/O).

### 5.3.2 Documentation

While the general concepts remain the same across frameworks, implementation details do matter (as we discuss later). In this work we mention concepts such as

https://docs.python.org/2/library/multiprocessing.html#exchanging-objects-between-processes
as “Fully connected” layers, various RNNs and CNNs, but in reality these have many parameters and possibilities, such as the type of initialization of parameters (random uniform vs. random normal, range of values, ...), use of bias units, optimization parameters etc. It is much more challenging to adjust these when documentation is poor or nonexistent, and requires reading the code directly. While Caffe itself does have documentation, the Python interface used by the original work does not, and the authors of the original work ([Fukui et al., 2016]) have also added their own modules to Caffe which of course do not have documentation either, apart from scientific descriptions in the paper. As a result we have decided to rewrite the model from scratch in Tensorflow, where the situation with documentation is much better (even if not ideal). We have kept data loading and preprocessing routines (but moved them outside the model itself), and we also use the same evaluation routines, but these are provided by the VQA Challenge rather than by the authors of the original work.

5.3.3 Modules

In general, Caffe was designed for Convolutional Neural Networks. LSTMs and other RNNs have been implemented later, but the support for them is limited. The main issue we ran into is RNNs with variable lengths of input (required for different length of questions), which can be done using a workaround in Caffe but is thus difficult to modify and adapt, and is much easier in Tensorflow using the dynamic_rnn function.

5.3.4 Problems with Tensorflow

Despite all of these positives, the main problem was the inability to reproduce the results using the Tensorflow version of the model. Even though we were able to get fairly close, the experiments did not result in the same exact numbers. Eventually, in the interest of having numbers comparable to the ones provided in the original work, we implemented some of the experiments back in Caffe and continued our work in Caffe. Nevertheless we provide details of the Tensorflow implementation in Section 5.4 for the sake of those who wish to try and get beyond our results using Tensorflow.
5.4 Tensorflow implementation details

In this chapter we will look into more detail of our Tensorflow implementation. For details on how to use it see Chapter 7.

We will describe the model structure (Section 5.4.1) as captioned by our Tensorflow implementation, the evaluation used (Section 5.4.2) and also which pre-processing has been performed for both text and images (Section 5.4.3).

5.4.1 Model

Embeddings

As mentioned before, the model in the original work as well as our models use two different embeddings (Figure 5.1, the green boxes marked “WE” and “Glove”), one of which are the GloVe vectors that are pre-trained and not part of the model itself. The other embedding is randomly initialized and trained along with the rest of the model. In Tensorflow, we use the `embedding_lookup` from the `nn` module in Tensorflow. This is essentially the same as the `Embed` layer in Caffe. While embeddings could technically be implemented using simpler modules as a matrix-vector multiplication with a one-hot vector, both of these implementations use a more sensible approach of simply selecting a particular row of the embedding matrix. We have initialized the embedding matrix from a uniform distribution in $[-0.08, 0.08]$, the same as in the original work.

After the selection of an appropriate embedding vector, the original work of Fukui et al. [2016] suggests using a nonlinearity (tanh). We do not believe this is standard practice when using embeddings, but we ran some of our models with and without it and did not find any significant difference in the results.

Text input: LSTMs

One of the key differences between Caffe and Tensorflow is the existence of the `dynamic_rnn` method in Tensorflow. This allows us to process questions of varying lengths in batches without the need for padding. In theory this should not affect the results and it should also reduce computation time, as the useless multiplication by a padding constant can be omitted. For the sake of being as close to the original as possible, we have tried both approaches (padding and dynamic) but better results have been obtained using the `dynamic_rnn`. From a technical perspective, the questions do still come in padded because Tensorflow cannot in general handle a tensor with a varying dimension, but `dynamic_rnn` performs the
correct computation if given sequence lengths and ignores the padding.

Both frameworks contain an LSTM cell which we use for the two input layers (Figure 5.1, boxes marked “LSTM1” and “LSTM2”) that process the text (question). The internals of the LSTM were initialized from a uniform distribution in $[-0.08, 0.08]$, the same as the original work.

After the two LSTM layers, we add a dropout layer with a dropout ratio of 0.3. We use the dropout on the output side of the LSTM, not between the hidden states as they are passed through timesteps. Again, this follows the original work. We also use dropout in the language attention module, i.e., after every output, not just the last one which is the only one utilized in the original work. Dropout is not used at test time, as is the usual practice in DNNs.

**Image processing**

As we have mentioned before, no image processing networks are part of the model, i.e., they are not trained with the model. The computation is done in advance, thus our model takes the 2048-dimensional grid descriptors as input ($14 \times 14 \times 2048$). The only difference between the frameworks is in the order of channels, width, height and number of samples in the input – Tensorflow is NWHC and Caffe is NCWH by default, and these settings have been kept.

**Spatial attention**

MCB was implemented in Tensorflow by Ronghang Hu and the implementation is publicly available on the web. We use this implementation, but we have added convolutional layers from the layers module and rectified linear units from the nn module.

**Language attention**

For the FC attention model, we use the dense() unit from the layers module combined with basic arithmetic operations, together with tiling and other necessary data transformations. In the case of MCB attention, we use the same publicly available implementation as with spatial attention.

**Region attention**

The implementation of the region attention mechanism is very similar to the implementation of language attention. In both cases, there is a set of vectors. The

[https://github.com/ronghanghu/tensorflow_compact_bilinear_pooling](https://github.com/ronghanghu/tensorflow_compact_bilinear_pooling)
only difference is that in language attention, the order is set, whereas the region descriptors are without any ordering. However, for the purposes of attention, this is irrelevant, as the attention value (the “Att” layer in Figure 5.1) is calculated independently for each region as well as each word of a sentence.

**Combinations**

For combinations, we use the dense unit from the layers module and the publicly available implementation of MCB, we do not need any other module or function even for the three-way MCB due to the decomposition which we have shown in Section 5.2.2.

### 5.4.2 Evaluation

Evaluation is done using the code provided by the organizers of the VQA challenge. They provide all the necessary pre-processing of the answers (such as contractions removal, see Chapter 3 for exact details). It also calculates the overall accuracy, and it is the same as the code used in the actual evaluation of the challenge’s test sets on the evaluation servers.

### 5.4.3 Data and preprocessing

**Language data**

Most of the preprocessing is already done in the data provided by the VQA challenge (see Chapter 3 for exact details). Fukui et al. [2016] added the GloVe vectors, which are available as part of the spacy module for Python.

**Image data**

Image data preprocessing is provided in the publicly available code for the original implementation of the Fukui et al. [2016] paper. The code takes each image and runs it through a network pre-trained on Imagenet [Russakovsky et al., 2015], and stores the output. In this case it is the Resnet-152 model [He et al., 2015]. We have tried using a more recent network, the Inception-ResNet-v2 [Szegedy et al., 2016], but have gained no noticeable performance gains.

---

Region data

Region extraction itself is available publicly in implementations of Faster-RCNN [Ren et al., 2015]. We use this implementation and the pre-trained network from Charles Shang (TFFRCNN)[7] but we had to modify this implementation to obtain only 10 regions per image, as described in Section 5.2. Furthermore, we ran the extracted regions through a CNN to extract region descriptors. In this case it was the Inception-ResNet-v2 model [Szegedy et al., 2016].

7https://github.com/CharlesShang/TFFRCNN
6. Experimental setup and results

6.1 Setup

The experimental setup is very straightforward. We use the VQA training set and the Visual Genome dataset as training data. The experiments are run with a batch size of 16.

The following parameters are kept constant for all experiments:
- the dimension of the custom embedding is 300,
- the internal dimension of any LSTM is 1024,
- in language attention layers, the LSTM outputs and the image descriptor are projected into a 1024 dimensional space,
- if included, the dimension of region descriptor is 1536.

In the LR-FC experiment, the dimension of the space to which all three modalities (language, image, regions) are projected into is 2048.

The setup of the optimizer is also the same for all experiments. We use the Adam optimizer [Kingma and Ba, 2014] with the following parameters:
- base learning rate: 0.0007
- first moment momentum: 0.9
- second moment momentum: 0.999
- $\epsilon$: $10^{-8}$

We have been using the LRC/DLL GPU cluster run by the Institute of Formal and Applied Linguistics at MFF UK. All of the GPUs are Nvidia GTX 1080s with 8GB of memory.

Using the above hardware, each individual experiment described below ran for about one week before training converged. Using Tensorflow, we were able to cut the time down (see Section 5.3.1) to 4-5 days, but ultimately we had to switch back to Caffe (see Section 5.3.4).

We had no explicit automatic stopping conditions, but we basically ran manual early stopping, i.e., we stopped training once performance on the validation set stopped improving for about one day, the equivalent of about 70,000 batches or one full pass of the training set.
6.2 Evaluation

We evaluate the experiments mostly on the validation set to give us an overview of how the models might perform and to track progress during training. Evaluation of the models on the test set is severely limited by the competition organizers (to avoid overfitting), with the exception of the test-dev set, on which we have evaluated several of the most promising Caffe models.

Naming of the experiments is in general as follows:

- Original - the original setting from [Fukui et al., 2016]
- First part of experiment name:
  - L: language attention layer used
  - R: image region attention layer used
- Second part of experiment name:
  - FC: Attention mechanism based on [Bahdanau et al., 2014] with fully connected transformations into a shared space (see Figure 5.1). In L-experiments this signifies the language attention mechanism, in LR-experiments the final combination.
  - LFC: Same as FC but no softmax layer
  - MCB: multi-modal compact bilinear pooling. In L-experiments this signifies the language attention mechanism, in LR-experiments the final combination.
  - MCBC: MCB with concatenation
  - Sep: special model with separate stacks and concatenation

In the results tables, more details are given about which method have been used in which experiment. Not all combinations have been used. Only those giving promising results after a number of epochs have been kept and fully evaluated.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Accuracy</th>
<th>Language Att.</th>
<th>Region Att.</th>
<th>Combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>58.60</td>
<td>-</td>
<td>-</td>
<td>MCB</td>
</tr>
<tr>
<td>L-FC</td>
<td>58.49</td>
<td>FC (Bahdanau)</td>
<td>-</td>
<td>MCB</td>
</tr>
<tr>
<td>L-MCB</td>
<td>58.55</td>
<td>MCB</td>
<td>-</td>
<td>MCB</td>
</tr>
<tr>
<td>LR-MCB</td>
<td>56.98</td>
<td>FC</td>
<td>MCB</td>
<td>MCB-cascade</td>
</tr>
</tbody>
</table>

Table 6.1: Results on the validation set for methods implemented in Tensorflow.
<table>
<thead>
<tr>
<th>Experiment</th>
<th>Accuracy</th>
<th>Language</th>
<th>Region</th>
<th>Combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>62.18</td>
<td>-</td>
<td>-</td>
<td>MCB</td>
</tr>
<tr>
<td>L-FC</td>
<td>62.25</td>
<td>FC</td>
<td>-</td>
<td>MCB</td>
</tr>
<tr>
<td>L-LFC</td>
<td>62.56</td>
<td>FC</td>
<td>-</td>
<td>MCB</td>
</tr>
<tr>
<td>L-MCB</td>
<td>61.55</td>
<td>MCB</td>
<td>-</td>
<td>MCB</td>
</tr>
<tr>
<td>LR-MCB</td>
<td>60.17</td>
<td>FC</td>
<td>MCB</td>
<td>MCB-cascade</td>
</tr>
<tr>
<td>LR-MCBC</td>
<td>60.00</td>
<td>FC</td>
<td>MCB</td>
<td>MCB+concat</td>
</tr>
<tr>
<td>LR-FC</td>
<td>57.93</td>
<td>FC</td>
<td>MCB</td>
<td>FC</td>
</tr>
<tr>
<td>LR-Sep</td>
<td>60.68</td>
<td>FC</td>
<td>MCB</td>
<td>Separate stacks + concat</td>
</tr>
</tbody>
</table>

Table 6.2: Results on the validation set for methods implemented in Caffe.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Accuracy</th>
<th>Language</th>
<th>Region</th>
<th>Combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>63.60</td>
<td>-</td>
<td>-</td>
<td>MCB</td>
</tr>
<tr>
<td>L-FC</td>
<td>63.32</td>
<td>FC</td>
<td>-</td>
<td>MCB</td>
</tr>
<tr>
<td>L-LFC</td>
<td>63.22</td>
<td>FC</td>
<td>-</td>
<td>MCB</td>
</tr>
<tr>
<td>L-MCB</td>
<td>63.17</td>
<td>MCB</td>
<td>-</td>
<td>MCB</td>
</tr>
</tbody>
</table>

Table 6.3: Test-dev results of selected methods (all in Caffe).

## 6.3 Discussion of the results

Below we discuss the three most striking observations which we propose based on the results and what we have learned during model design, implementation and while training and testing.

### 6.3.1 Language attention

Based on the validation results, adding language attention seems to slightly improve the performance of the model. However, this was not confirmed on the test-dev set. This is true for the original attention model suggested by [Bahdanau et al. 2014](#), but not in the case of MCB-based attention.
In essence, the MCB method of \footnotesize{Fukui et al.} \small{2016} is an approximation of a very large layer, namely the outer product of two vectors. It may be the case that the outer product is unnecessarily complex (and therefore sparse from the training data point of view), and that the smaller element-wise product of \footnotesize{Bahdanau et al.} \small{2014} is sufficient to extract all the information possible. In the approximation some of the information may be inadvertently lost, as the approximation is essentially random and not fine-tuned during the learning process.

On the other hand, MCB has improved the overall performance when used in the spatial attention (we have kept this for all our models, as this approach has won the 2016 VQA challenge). This would imply that the information flow through the spatial attention mechanism is sufficient to saturate it, and it could possibly be beneficial to expand it even further. In part, this is what led us to the region attention mechanism.

6.3.2 Region attention

Contrary to the results of experiments which include the language attention module, adding region attention to the models resulted in inferior performance. In the simpler models where it is simply “tacked on” to the existing model (such as LR-MCB), we at first thought this may be due to the cascading MCB approach, as some of the information goes through the approximation twice and may get distorted, or get conflicting gradient information during backpropagation. This led us to the model which only concatenates the outputs and does not attempt to perform 3D-MCB, but this did not help, and neither did doing away with the MCB mechanism at the end completely and transferring to the \footnotesize{Bahdanau et al.} \small{2014} attention model.

As a last resort, we have also attempted to separate the region attention mechanism completely to avoid any gradient conflicts. This led to a slight improvement in the experiment called LR-Sep, albeit not quite up to the level of language attention alone. The cause of this is not clear to us at this time, we suspect this may be due to issues with counting objects if they end up in different regions. It could also be caused by the absence of a model confidence mechanism for each of the modalities leading to too much focus on objects of the twenty extracted classes in cases where they are not relevant.
6.3.3 Tensorflow implementation

The reimplementation into Tensorflow should have given us more flexibility. We have devoted a lot of effort to the reimplementation and to all of the details, and succeeded in the implementation itself and in running several experiments as described earlier in this chapter (details of the implementation can be found in Section 5.4). However, we have not been able to reach the same performance of the original Caffe implementation, despite reaching as exact match of all details and parameters as possible, and eventually had to return to Caffe as the framework of choice to complete the work. The reasons for this are not clear at this time. We suggest more investigation in Chapter 8 – it may prove necessary to investigate random seeds and their usage throughout the two frameworks (possibly removing all non-determinism), match the models exactly (starting from individual layers and working up to more complex models) and finally match the training process step by step to pinpoint the exact differences. We consider this to be outside the scope of this work.
7. User documentation and attachment description

7.1 Attachment structure

The attachment contains the following directories:

- **caffe**, which contains executable code of the L-MCB model in Caffe,
- **tf**, which contains executable code of the L-MCB model in Tensorflow,
- **caffe-other**, which contains other experiments in Caffe,
- **tf-other**, which contains other experiments in Tensorflow,
- **Faster-RCNN**, which contains code to adapt the existing Faster RCNN implementation to our data,
- **lists-of-files**, which contains lists of files by extension.

The **caffe-other** and **tf-other** directories contain other experiments in various stages of completion. Of interest are the files named `train_att_bc.py` in **caffe-other** and `model.py` in **tf-other**, which contain the models for all the experiments we have carried out, not just those mentioned in the results. These files can be run using a Python 2.7 interpreter and all the prerequisites listed below, and will train the models on the appropriate data. However they would require some adaptation to run in the demo code or on the test-dev set, as they were only designed as experiments. Each of them contains a small README.txt with a brief description of the experiment.

7.2 Running the Caffe implementation

7.2.1 Requirements

- Python 2.7
- OpenCV
- the VQA-MCB implementation from [https://github.com/akirafukui/vqa-mcb](https://github.com/akirafukui/vqa-mcb) and all its prerequisites, incl. all labeled as optional:
  - Compiled branch `feature/20160617_cb_softattention` from custom Caffe fork available at [https://github.com/akirafukui/caffe](https://github.com/akirafukui/caffe)
  - pre-trained ResNet-152 model from [https://github.com/KaimingHe/deep-residual-networks](https://github.com/KaimingHe/deep-residual-networks)
  - VQA tools available at [https://github.com/VT-vision-lab/VQA](https://github.com/VT-vision-lab/VQA)
(update the VQA tools path in caffe/config.py after installation)

- VQA v1 full dataset obtainable from http://visualqa.org/download.html
- Visual Genome dataset downloadable from https://visualgenome.org/ (After unzipping, the dataset is split into two directories, VG_100K and VG_100K_2. For simplicity, place all the images into one directory called VG_100K).

Some of these requirements have their own prerequisites. We recommend using virtualenv, and installing all the required parts at once using pip. We have stored a full list in caffe/caffe_requirements.txt, but please note that this does not include Caffe, for which a custom branch must be compiled, and OpenCV, which is not distributed through pip and also will most likely have to be compiled for your particular machine. To use our list, run the following command inside an activated virtualenv:

    pip install -r caffe_requirements.txt

Furthermore obtain the GloVe vectors using the SpaCy module:

    python -m spacy download en

Furthermore, pre-trained models are necessary to run the all parts of the code:

- The Resnet-152 used in image pre-processing. Please follow the instructions in the vqa-mcb README.md on how to obtain it. For the demo you will also need the proto_test_batchsize1.prototxt file, which we have provided in the caffe/preprocess directory. Please update the configuration at the beginning of caffe/demo.py with the location of the necessary files.

- Our example model, which can be obtained from http://ufal.mff.cuni.cz/~straka/hajick/caffe/result/ It consists of two files, _iter_310000.caffemodel and _iter_310000.solverstate. Please download them both and place them into the caffe/result/ sub-directory.

### 7.2.2 Preprocessing

All the data needs to be preprocessed. Use the instructions at https://github.com/akirafukui/vqa-mcb/blob/master/preprocess/README.md to preprocess the VQA dataset and the language part of Visual Genome.

But please note that that README.md does not contain information on how to preprocess the images from the Visual Genome dataset. We recommend changing configurations in vqa-mcb/preprocess/config.py as follows:
OUTPUT_PATH = "../../genome/"
OUTPUT_PREFIX = "resnet_res5c_bgrms_large/"

and add the line

GENOME_IMAGE_PATH = "../../genome/VG_100K/"

(set paths to the location of your Visual Genome dataset and desired output locations), then running preprocess/extract_resnet.py, but with a different __main__:

```python
if __name__ == '__main__':
    extract_features('', config.OUTPUT_PREFIX + '')
```

and with modified line 26:

```python
target_path = os.path.join(config.GENERE_IMAGE_PATH, target_data)
```

We have provided files with these modifications: config_genome.py and extract_resnet_genome.py. Once you obtain the vqa-mcb code, you may insert them into vqa-mcb/preprocess and they should run.

But any way of feeding the Visual Genome images to the pre-processing code will suffice.

After pre-processing, modify the locations of the preprocessed data and VQA tools in the caffe/config.py file.

Please note that the example model does not contain region attention. For pre-processing of regions, see Section 7.4. This will allow you to run some of the experiments in caffe-other/ which contain region attention.

Pre-processing is not necessary to run the demo code.

### 7.2.3 Running the code

To run training, simply run

```bash
python train.py
```

in the caffe/ directory. To resume training, run
To generate the answers on the test-dev dataset, run

```
python exec_test.py --restore-from '/PATH/TO/.solverstate/FILE'
```

The `--restore-from` argument is required.

### Running the demo

To run the demo, update the variables `VQA_MCB_PATH` and `MODEL_DIRECTORY` in `demo.py` to point to your installation of `vqa-mcb` and to your directory with the downloaded pre-trained models. Then run:

```
python demo.py --image IMAGE_PATH --question QUESTION
```

The last thing the code will print are the five most likely answers, along with scores. We have provided a sample image, `000456.jpg`.

The code is setup to run on GPUs. On a GTX 1080 GPUs training takes roughly one week. It is possible to run the demo up on CPU only by setting `caffe.set_mode_cpu()` (see line 63 in `demo.py`).

### 7.2.4 Contents of the caffe directory

The directory contains all code necessary to run the model on the test set, a custom image-question pair and to run training. Some of the code is our work and some has been reused from the original implementation.

#### Partially our work

- **network.py**: The model definition itself, within the `qlstm` function. Defines the network.
- **train.py**: The training code. It handles everything including printing progress, running validation and resuming training from existing models.
- **config.py**: Model configuration; contains dimensions of LSTMs, embeddings, input images as well as data locations etc.
- **exec_test.py**: Runs the model on the test-dev set and stores the result.
- **demo.py**: Demo code, for running the model with single image-question pair.

Adapted from a demo server provided by the authors of the original work.

#### Not our work

- **visualize_tools.py**: Module containing tools for visualization of parts of the models, and most importantly the `exec_validation` function for executing validation and testing using VQA tools.
• vqa_data_provider_layer.py: Module containing all necessary functions to group the preprocessed data correctly into batches and serve them as is or through a Caffe layer.
• demo_provider_layer.py: A slightly modified vqa_data_provider_layer.py which provides data for the demo.
• qlstm_solver.prototxt: The configuration of the solver and selected training parameters.
• 000456.jpg: A sample image.

The preprocess subdirectory
Contains files necessary for preprocessing.

The result subdirectory
Contains files generated during training or initialization.

• proto_train.prototxt: Caffe-readable definition of the training model. Transformed from the Python definitions in network.py to a prototxt format readable by Caffe itself.
• proto_test.prototxt: Caffe-readable definition of the model used for testing. Transformed from the Python definitions in network.py to a prototxt format readable by Caffe itself.
• adict.json: The answer dictionary – translates between class ids and words they represent. Used to transform the network output (id) to actual English words.
• vdict.json: The question dictionary – mapping between words and their numerical ids. Used to transform input sentences into numerical ids for embedding lookup.

7.3 Running code in Tensorflow

7.3.1 Requirements
The requirements for running the Tensorflow code are almost the same as with Caffe described above, but with a few key differences:

• There is no need to acquire the specific branch of Caffe, but an installation of Caffe is required for preprocessing.
• The tensorflow-gpu package version 1.1.0 or greater is required. The full list of requirements is in tf/requirements.txt, and can be used just
as with Caffe.

- The Tensorflow MCB implementation is included, but the sequential FFT module will most likely need to be recompiled for any new machine by running `tf/sequential_fft/compile.sh`.

Furthermore, pre-trained models are necessary to run the all parts of the code:

- The Resnet-152 used in image pre-processing. Please follow the instructions in the vqa-mcb README.md on how to obtain it. For the demo you will also need the proto_test_batchsize1.prototxt file, which we have provided in the caffe/preprocess directory. Please update the configuration at the beginning of caffe/demo.py with the location of the necessary files.

- Our example model, which can be obtained from [ufal.mff.cuni.cz/~straka/hajick/tf/ckpt/](ufal.mff.cuni.cz/~straka/hajick/tf/ckpt/). It consists of four files. Please download them all and place them into the tf/ckpt subdirectory.

### 7.3.2 Preprocessing

The data pre-processing steps are exactly the same as with the Caffe code. Due to the space requirements we recommend executing them only once and sharing the location between the Caffe and Tensorflow models. Preprocessing is not necessary to run demo code.

### 7.3.3 Running the code

To run training, simply run:

```
python train.py
```

in the tf/ directory. To resume training, run:

```
python train.py --restore_from 'YOUR_CHECKPOINT.ckpt'
```

By default this points to model.ckpt. You can use your own but you must place them in the ckpt/ subdirectory.

To run on the test-dev dataset, run

```
python exec_test.py --restore_from 'YOUR_CHECKPOINT.ckpt'
```

By default this points to model.ckpt. You can use your own but you must place them in the ckpt/ subdirectory.

### Running the demo

To run the demo on your own question-image pair, please update the variables VQA_MCB_PATH and MODEL_DIRECTORY in demo_prep.py and demo_exec.py to point
to your installation of vqa-mcb and to your directory with the downloaded pre-trained models. Then run:

```
./demo.sh IMAGE_PATH QUESTION
```

The last thing the code will print is most probable answer according to the model.

### 7.3.4 Contents of the tf folder

**Our work**

- **network.py**: The model definition itself, within a `Network` object(). Includes functions for forward pass (`predict()`), validation (`test()`) and training (forward-backward) pass (`train()`).
- **train.py**: The training code. Handles everything including printing progress, running validation and resuming from existing models. Parts of code which interface with the provided VQA validation code are identical to the Caffe version.
- **config.py**: Model configuration, contains dimensions of LSTMs, embeddings, input images as well as data locations etc.
- **exec_test.py**: Runs the model on the test-dev set and stores the result.
- **demo_prep.py, demo_exec.py and demo.sh**, scripts to run the demo.

**Not our work:**

- **visualize_tools.py**: Contains tools for visualization of parts of the models, and most importantly the `exec_validation` function for executing validation and testing using VQA tools.
- **vqa_data_provider_layer.py**: Contains all necessary functions to group the preprocessed data correctly into batches and serve them as is or through a Caffe layer.
- **vqa_data_provider_layer_right_padded.py**: Very similar to `vqa_data_provider_layer.py` but contains code which pads the questions on the right instead of left, which is more convenient for the `dynamic_rnn` unit.

---

1The shell script is necessary because the Caffe preprocessing and Tensorflow model cannot run in the same thread. This is a known issue. [https://github.com/tensorflow/tensorflow/issues/916](https://github.com/tensorflow/tensorflow/issues/916)
• demo_provider_layer.py: A slightly modified vqa_data_provider_layer.py which provides data for the demo. Also right padded.

• compact_bilinear_pooling.py: The MCB implementation by Ronghang Hu.

• compact_bilinear_pooling_test.py: Tests of the MCB implementation.

• adict.json: The answer dictionary – translates between class ids and words they represent. Used to transform the network output (id) to actual English words.

• vdict.json: The question dictionary – mapping between words and their numerical ids. Used to transform input sentences into numerical ids for embedding lookup.

• sequential_fft/ subdirectory: part of the the MCB implementation by Ronghang Hu.

7.4 Region extraction

The example models do not contain the region attention mechanism. To perform preprocessing necessary for regions, obtain the TFFRCNN code from https://github.com/CharlesShang/TFFRCNN and follow all installation instructions. Then do the following:

• create the models/ subdirectory in the TFFRCNN root and place any pretrained models you have downloaded in there.

• put the Faster-RCNN/run_bbox_detection.py file into the TFFRCNN/faster_rcnn/ directory

• Point the COCO_PREFIX and GENOME_PREFIX variables in run_bbox_detection.py to the correct location of your downloaded image datasets.

• put the Faster-RCNN/run.sh file into the TFFRCNN/ directory and modify the model path inside to fit your chosen model.

• Run the run.sh script in TFFRCNN/ to detect and save all bounding boxes. (This took a few days on a GTX 1080.)

• Point the data_dir and dest_dir variables in get_bbox_desc_vqa.py and get_bbox_desc_genome.py to the directories with the extracted bounding boxes.

• Point the ir2.ckptfilepath at the filepath of your Inception-ResNet-

- Run the `get_bbox_desc_genome.py` and `get_bbox_desc_vqa.py` scripts. (This also took a few days on a GTX 1080.)
8. Conclusions and future work

8.1 Conclusions

In this work we have taken an existing, successful approach to the VQA task, and attempted to improve it based on available methods. We have also experimented with various ways of combining them. Unfortunately we have not been able to achieve significant improvements to the results of the original model.

In the case of language attention we have observed that using MCB to combine the modalities performs worse than the less complex model based on language attention [Bahdanau et al., 2014]. We suggest this may be due to the fact that MCB inherently contains an approximation, albeit of a more complex operation, and thus it may hurt the performance by losing relevant information.

For region attention, we have observed that it is best to use two separate language processing models, one to combine with spatial attention (and potentially language attention), and one for region attention. Despite that the region attention mechanism led to inferior performance. We suggest this may be due to the lack of a balancing mechanism, irrelevance of the regions to the question or problems with counting when objects to be counted end up in different regions. It could also be due to an increase in model complexity and thus insufficient data (sparse data problem).

We have also found out that not all frameworks are created equal. Despite attempting to match the provided Caffe implementation of the original work as closely as possible in Tensorflow, we have not been able to match the results. We think there may be minor differences in implementation of some core parts. We have also, in this case not surprisingly, seen a dependency on initialization methods for parameters, and have achieved best results when reproducing them as exactly as possible. The MCB module is also highly dependent on random initialization, which may affect the outcome.

8.2 Future work

As the original authors of the VQA challenge suggest in [Goyal et al., 2017], the shift for any future work must be towards extracting more information from the image. We think the first step to that is fine-tuning the pre-processing CNN together with the model. However, doing this will be computationally expensive and thus difficult to rapidly prototype with multiple different CNNs to choose
On the other hand, we think that the number of images currently available across the datasets may not be enough to sustain such a heavy focus on image (pre)processing, in particular these models may start to overfit. We believe a way to counter this is some sort of content-aware data augmentation. Data augmentation methods have proven highly effective since the work of Krizhevsky et al. [2012]. The new VQA v2 dataset could be seen as a form of data augmentation for this task, but still requires a lot of human annotator input, limiting its efficiency.

Thirdly, we believe the difference in frameworks deserves more attention. Choice of framework is hardly ever mentioned in papers as significant and in general large frameworks like Caffe, Tensorflow or Torch are seemingly believed to be equal in performance. Our results suggest this is not the case, at least for models which use modules and methods across multiple fields (language, images).

Finally, if the method is ever to have any practical application, we believe it must be extended to video input (which is readily available e.g. on a vast majority of personal mobile devices), which should provide several orders of magnitude more information to work with to figure out the answer to the types of questions that we have worked with.
Bibliography


<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Sample images and questions from the VQA dataset. Figure taken from [Antol et al., 2015].</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>Example of a convolutional network model. Model and figure taken from [LeCun et al., 1998].</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>Residual connections in very deep convolutional networks. Figure taken from [He et al., 2015].</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>Basic and unrolled RNN. Figure taken from [Olah].</td>
<td>9</td>
</tr>
<tr>
<td>2.4</td>
<td>RNN usage for various input and output types. Figure taken from [Karpathy].</td>
<td>9</td>
</tr>
<tr>
<td>2.5</td>
<td>Comparison of LSTM units without and with a forget gate. Figures taken from [Gers et al., 2000]. For detailed descriptions of each operation, see the original work.</td>
<td>10</td>
</tr>
<tr>
<td>3.1</td>
<td>Answer frequencies in the VQA and VQAv2 datasets. Figure taken from [Goyal et al., 2017].</td>
<td>16</td>
</tr>
<tr>
<td>3.2</td>
<td>Example image with various annotations. From Visual Genome website.</td>
<td>17</td>
</tr>
<tr>
<td>3.3</td>
<td>Example DAQUAR images. From DAQUAR website.</td>
<td>18</td>
</tr>
<tr>
<td>4.1</td>
<td>Simple MCB model. Figure taken from [Fukui et al., 2016].</td>
<td>22</td>
</tr>
<tr>
<td>4.2</td>
<td>MCB model with spatial attention. Figure taken from [Fukui et al., 2016].</td>
<td>23</td>
</tr>
<tr>
<td>5.1</td>
<td>Schema of fully connected language attention.</td>
<td>25</td>
</tr>
<tr>
<td>5.2</td>
<td>Schema of MCB language attention.</td>
<td>27</td>
</tr>
<tr>
<td>5.3</td>
<td>Schema of MCB region attention.</td>
<td>29</td>
</tr>
<tr>
<td>5.4</td>
<td>Schema of the three-way MCB model.</td>
<td>31</td>
</tr>
<tr>
<td>5.5</td>
<td>Schema of the three-way FC integration.</td>
<td>32</td>
</tr>
<tr>
<td>5.6</td>
<td>Schema of the separate question stack model.</td>
<td>33</td>
</tr>
</tbody>
</table>
List of Tables

3.1 Statistics of the real images subset of VQA dataset. . . . . . . . . 14
3.2 Statistics of the real images subset of VQAv2 dataset. . . . . . . . 15

6.1 Results on the validation set for methods implemented in Tensorflow. . 41
6.2 Results on the validation set for methods implemented in Caffe. . 42
6.3 Test-dev results of selected methods (all in Caffe). . . . . . . . . . 42