university of groningen

faculty of science and engineering

Visual Question Answering With Enhanced Question-Answer Diversity

Master's Thesis

Artifical Intelligence

University of Groningen

Jelle Visser
s2238160

Primary Supervisor:	Prof. dr. L.R.B. Schomaker
Secondary Supervisor:	Dr. G. Maillette de Buy Wenniger

Artificial Intelligence, Faculty of Science and Engineering, University of Groningen

Abstract

Visual Question Answering (VQA) is a multi-modal Machine Learning task that consists of two inputs, an image and a natural language question about this image, and requires an answer. VQA models are often quite complex and are bad at answering previously unseen questions. Additionally, data collection requires intensive human labor, and it is hard to augment the natural language data. This research proposes a method of enhancing the question-answer (QA) input data, using the Visual Genome dataset to automatically generate new QA-pairs by using its extensive image annotations. We construct two baseline VQA-models, one that chooses an answer from a pre-defined list and one that generates an answer word-by-word with an LSTM, and train them on four datasets with different degrees of data augmentation. We compare the models using several metrics while testing on a holdout test set from the original dataset. Additionally, experiments are conducted where we measure how adding noise to the question embedding affects the performance of both baseline models, as an indication for robustness to uncertainty in the question input. We find that training on augmented datasets slightly decreases performance on the holdout test set for both baseline models. All models, however, show to be highly resistant to noise on the question embedding. Additionally, models trained on the augmented datasets appear to be more resistant to noise compared to models trained on the original dataset. This suggests that our method of data augmentation improves a VQA model's robustness to unseen data.

Contents

Chapter 1

Introduction

Visual question answering (VQA) is a relatively new artificial intelligence (AI) task spanning multiple disciplines [1, 2]. A typical VQA task consists of two inputs: an image and a natural language question about this image. Based on the image, the VQA system will then attempt to answer this question. This means that a VQA system needs to be able to process and understand visual input, while being capable of both interpreting and producing human language as well. The field of VQA therefore exists at the boundary between computer vision (CV) and natural language processing (NLP). Originally, VQA was proposed as challenge to simulate research and further develop both fields [3]. NLP is required to parse input questions and return a human-readable answer at the end of the pipeline. Due to the big variety in question types, VQA requires several different CV capabilities, such as image recognition, object detection, and activity recognition. More advanced questions even require aspects from knowledge representation, such as knowledge base reasoning and commonsense reasoning [3]. This multi-modal nature of VQA makes it a unique field of study, as most AI research focuses on emulating one aspect of human intelligence at a time. However, human intelligence is larger than the sum of its parts and requires complex coordination between multiple parts of the brain with different responsibilities. Indeed, the true challenge in VQA is not merely using the principles from both fields, but combining them as well, which is difficult since the disciplines of CV and NLP historically use different methods and models to solve their respective tasks. In other words: advancing multi-modal disciplines such as VQA brings us closer to a more general form of artificial intelligence [4].

Besides being a means of advancing AI in general, VQA has a wide range of possible practical applications. Broadly speaking, a VQA system allows humans to intuitively query visual content by using natural language. It allows for image retrieval based on semantic queries on image content or it could be employed as an aid to provide visual information to the visually impaired [5, 6]. Additionally, it could be used to implement more advanced human-machine communication in, for example, robotics [7].

Visual Question Answering is commonly posed as a supervised machine learning problem, meaning it requires of set of labeled data. In the case of VQA, each

data sample should contain an image, a question about this image, and a correct answer to this question. Most VQA methods use artifical neural networks (ANNs) and are based on the so-called joint embedding approach, a category of VQA pipeline archictectures that use a combination of Recurrent Neural Networks (RNNs) to represent the input question and Convolutional Neural Networks (CNNs) to extract image features [3, 6, 8]. These embeddings are then combined in order to produce an answer, either by posing it as *1-out-of-N* classification problem and choosing an answer from a list of pre-defined options, or by treating it as a sequence generation problem and using another RNN to generate an answer word-by-word. Such models are often augmented with an attention mechanism that learns to query specific parts of the input image based on the semantic contents of the input question, instead of querying the entire image [3, 6, 9–11].

State-of-the-art VQA models continuously seek to raise the bar for performance on benchmark VQA datasets [3, 6, 9]. These models are becoming increasingly complex, often containing a number of parameters in the order of millions or even tens of millions. This increases the risk of overfitting and many VQA models tend to suffer from the same problem at deployment: the system is unable to deal with questions that were not present in the data set it was trained on. The fact that these models are trained and tested in their "laboratory environment", without regard for their performance in the noisy real world, makes developing these models somewhat of a contrived problem. The set of possible questions that a user can ask is enormous and a single question can be formulated in many different ways. This vulnerability of VQA systems to the nuances of natural language is a major challenge in the field. Machine learning (ML) models in general are often made more robust by means of data augmentation, where additional training samples are generated in order to diversify the existing training data. Indeed, analysis of existing VQA algorithms has shown that they would greatly benefit from more training data [6] and creating a more diverse set of question-answer pairs for a VQA system to train on could possible greatly improve its robustness at deployment [12].

Augmenting language data is quite difficult, however, as algorithmically generating natural language is a field of study of its own. Using human annotators to generate additional data would be possible, but almost all VQA datasets are already highly dependent on human labor, as they are often constructed and annotated by crowdsourcing the task to human workers. Not only is this time-consuming and expensive, but one could also argue that this dependence on human labor is not truly intelligent. It would therefore be more desirable to use machine-generated samples to extend existing datasets. Several studies explore the concept of question generation (QG) as a means to augment and diversify question-answer (QA) data for visual question answering. One approach is to use a combination of image annotations and

so-called templates, which use annotations belonging to a particular image to fill in the blanks in a predefined set of natural language questions and answers, based on a predefined set of rules. [8, 13, 14]. Another approach is to use machine learning techniques to train a model to generate new QA-pairs based on either the image alone, or on a combination of the image and a set of annotations [12, 14, 15].

The most popular dataset for VQA is the VQA-real, or COCO-VQA, dataset developed by the authors of [1]. Upon its introduction, it contained more images, questions, and answers than any other dataset at the time. In addition to images and QA-pairs, the dataset contains annotations that list the objects present in each image as well. Krishna et al, however, argue that the key to success in cognitive tasks, such as VQA, lies in understanding the interactions and relationships *between* objects in an image [16]. They have developed another large-scale VQA dataset called Visual Genome (VG) that contains a dense collection of annotations for each image, describing the objects present in the image, their attributes, and their relationships to each other. Additionally, Visual Genome contains almost three times as many questions and over 40% as many unique answers as VQA-real. On paper, this means that the dataset allows for models that are better equipped to deal with the nuances of natural language compared to those trained on VQA-real. However, while Visual Genome is sometimes used as an auxiliary dataset to extend the data from VQA-real, very few models are trained purely on this dataset. Not many studies have explored using Visual Genome's rich annotations for augmenting its QA-data. Indeed, how the rich collection of annotations in the VG dataset can be used for designing and training VQA systems is still an open research question [3].

Ultimately, we want a VQA-system to behave intelligently and to perform well in the real world, where it will be confronted with unseen data and the versatility of natural language. We believe that the key to this lies in training a VQA system on a dataset with a large and diverse body of question-answer pairs. In this study we will therefore take the dataset with the largest and most diverse set of QA-data, Visual Genome, and extend the existing QA-pairs with new samples generated by a template-based question generation method. These templates will make use of the rich annotations that come with the VG dataset in order to generate questions that query not only the presence of objects, but also the relations between them. We construct three new sets of QA-pairs which each are supersets of the original Visual Genome question-answer set, with each set extending the original dataset with more newly-generated samples than the previous. The goal of this research is to look at the effect of our question-answer data augmentation method on a VQA system's performance and overall robustness to variations in the question input.

In order to measure whether this data augmentation method can improve a VQA system's performance, we construct two baseline models based on the strong but

conceptually simple architecture proposed in [11]. The core of the model is based on the joint embedding approach, using a CNN to represent the image and a RNN to represent the question. The model uses an attention mechanism based on the stacked attention network proposed in [10]. The first baseline model, from hereon referred to as *baseline model A*, passes the joint embedding through a set of fully connected layers to pick *1-out-of-N* answers from a pre-defined answer list. While this is the most common approach to VQA, a human is able to formulate an answer by using an arbitrary combination of words. We therefore construct a second baseline model, *baseline model B*, that uses an RNN to generate an answer word-by-word. Both baseline models are trained on the four different datasets, producing a set of trained models which will be compared and evaluated on several different performance metrics. We hypothesize that, for both baseline architectures, models trained on the augmented datasets perform better than those trained on the original dataset. We expect that the more generated QA-pairs are added to the original dataset, the better the performance of a model trained on this augmented dataset.

The next section, Chapter 2, provides the theoretical background of all techniques used in this study. Chapter 3 describes the Visual Genome dataset, the methods used to enhance this dataset, how the various VQA-models were implemented, and the experiments performed on the different trained models. Chapter 4 contains the results of these experiments, which are discussed and interpreted in Chapter 5. Suggestions for future research are discussed in Chapter 5 as well.

Chapter 2

Theoretical background

In this chapter, the theoretical background of the techniques used throughout this thesis will be described. First, an overview will be given of Artificial neural networks (ANNs) in general and the specific types of ANNs used in this study. This is followed by a brief description of relevant Natural language processing (NLP) methods. The final section is dedicated to techniques that are specific to Visual Question Answering (VQA).

2.1 Artificial neural networks

Artificial neural networks are machine learning models inspired by the (human) brain. The brain consists of neurons that, through the synaptic connections between them, can either have an excitatory or inhibitory effect on their neighboring neurons. ANNs consist of a set of processing units connected by weights, analogous to the brain's neurons and synapses. Each artificial neuron's state depends on its inputs, while its influence on others is determined by the combination of its state and the weights leading to other nodes.

All ANNs used in this study are trained within the *supervised learning* paradigm. In general, ANNs trained through supervised learning try to learn a function that maps given inputs to outputs, after being trained on example input-output pairs.

2.1.1 Perceptrons

One of the more basic forms of such ANNs is the *perceptron*, proposed by Rosenblatt in 1956 [17]. It forms the basis for many of the more complex network architectures we see today. Its most simple form is a binary classifier that requires a training set $D = \{(\mathbf{x_1}, t_1), ..., (\mathbf{x_n}, t_n)\}$. Each of the n training samples consists of an input vector of real values $\mathbf{x}$ and a corresponding binary target output t, the goal being to learn a function such that $f(\mathbf{x_i}) = t_i$. Here, $f(x)$ is one of the two binary labels, often given as $\{0, 1\}$ or $\{-1, 1\}$. Given the input vector $\mathbf{x}$, a weight vector $\mathbf{w}$ of equal dimensionality as $\mathbf{x}$, and a bias value b, the output $f(x)$ is computed using equation

2.1.

$$f(\mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{x} \cdot \mathbf{w}^{\mathbf{T}} - b \leqslant 0 \\ 1 & \text{if } \mathbf{x} \cdot \mathbf{w}^{\mathbf{T}} - b > 0 \end{cases} \tag{2.1}$$

The perceptron learns a linear hyperplane that acts as a decision boundary separating the input samples, as can be seen in Figure 2.1. Here, the bias b is a trainable parameter that does not depend on any input value, which shifts the decision boundary away from the origin. It is often implemented as an additional weight with a constant input value of 1. The action of calculating a neural network's output based on an input pattern and the network's weights and activations, is called a *forward pass* or *forward propagation*.

The perceptron is trained by presenting it with a sample input, comparing its output with the target output, and updating the weights. When using a linear activation, as is the case in the example described above, the weight update for weight j at node i is given by equation 2.2:

$$\Delta w_{ij} = \eta(t_j - y_j)x_i \tag{2.2}$$

Here, t_j and y_j the target and actual outputs at node j, and x_i the value of the input. η is the learning rate, a real valued number in the interval $[0, 1]$, which is used to control the size of each weight update. This value is a hyperparameter set before the training starts.

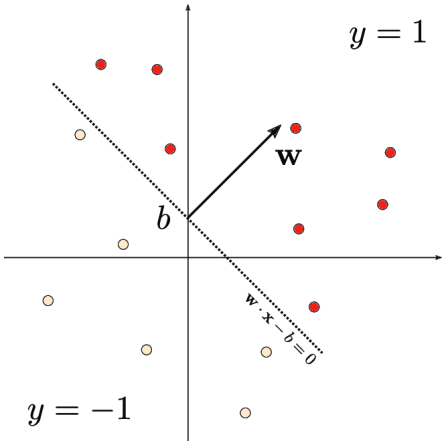


Figure 2.1: The perceptron as a hyperplane. Note that the bias b is necessary to move the boundary away from the origin, such that the two classes are separated. Here, the two classes are represented by $\{-1, 1\}$. Image retrieved from [18].

2.1.2 Multilayer perceptrons

While the perceptron is able to learn a linear decision boundary, it is unable to deal with problems that are not linearly separable. This hugely limits its usability for real world problems, which often require non-linear decision boundaries. In order to introduce non-linearity to the classifier, additional layers between the input and output layers can be added. However, adding hidden layers alone is not enough for a neural network to be able to learn a non-linear function: a forward pass would still consist of a series of dot products between vectors, i.e. a series of linear operations, which would never be able to learn a non-linear function. In order to solve this, we must also add a non-linear activation function to each of the network's units.

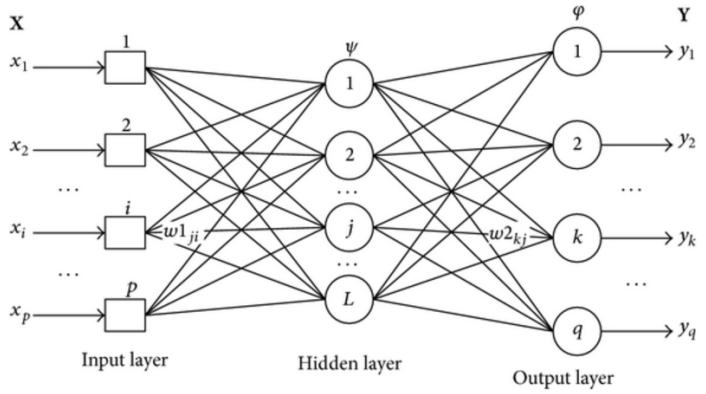


Figure 2.2: Diagram of a multilayer perceptron consisting of one hidden layer. Image retrieved from [19].

Such a neural network consisting of an input layer, one or more so-called hidden layers and an output layer comprised of non-linearly activated units, is called a multilayer perceptron (MLP) (Figure 2.2. The layers in an MLP are fully connected, meaning that each node i in layer n has a weight w_{ij} leading to each node j in the next layer $n + 1$. An MLP is a universal function approximator, meaning that with sufficient data, training time and hidden layers, it can approximate any measurable function to any degree of accuracy [20].

Each node in a layer is a perceptron by itself. The activations $\mathbf{a}(\mathbf{x})$ of an entire layer are calculated with equation 2.3.

$$\mathbf{a}(\mathbf{x}) \;=\; g(\mathbf{x}^T\mathbf{W} + \mathbf{b}) \tag{2.3}$$

Here, $g(\mathbf{x})$ is an activation function, $\mathbf{x}$ either the input vector or the activations of

the previous layer, $\mathbf{W}$ the weight matrix corresponding to the weights between the current and the previous layer, and $\mathbf{b}$ a vector consisting of a bias value for each unit.

2.1.3 Activation functions

Activation functions in artificial neural networks control the output value of each individual unit. In multilayer perceptrons, the activation function traditionally used for hidden units is the *logistic* or *sigmoid* function (equation 2.4).

$$\sigma(x) \;=\; \frac{1}{1 \;+\; e^{-x}} \tag{2.4}$$

The logistic function squashes its input into a value between 0 and 1. $\sigma(x)$ approaches 0 as x approaches $-\infty$, and 1 as x approaches $+\infty$. A downside of the sigmoid function is that its output is not zero-centered, as can be seen in Figure 2.3. This can lead to undesired properties during backpropagation, which will be described later. An alternative activation function that is similar to the logistic function, but produces a zero-centered output between -1 and 1, is the *hyperbolic tangent*, also seen in 2.3. The hyperbolic tangent, or *tanh*, is described in function 2.5.

$$tanh(x) \;=\; \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.5}$$

When sigmoid activations saturate either to 0 or 1, its gradients become close to zero. This, again, can lead to problems during backpropagation. The *rectified linear unit* (ReLU) solves by not letting its activations saturate. Instead, its activation is equal to its input for inputs larger than 0, and otherwise simply set to 0. A nice feature of this activation function is that it can be computed by simply thresholding its input at zero. It is therefore much easier to compute than either the sigmoid or tanh functions. The rectified linear unit is given by equation 2.6 and can be seen in Figure 2.3.

$$f(x) = \begin{cases} 0 & \text{for } x \leqslant 0 \\ x & \text{for } x > 0 \end{cases} \tag{2.6}$$

The activation function used at the output layer depends on the nature of the problem. For classification tasks, the *softmax* function is commonly used, which is also used in this study. The softmax function for the ith node in the output layer y is given by equation 2.7.

$$S(y_i) \;=\; \frac{e^{y_i}}{\sum_{j=1}^{K} e^{y_j}} \tag{2.7}$$

Here, K is the total number of classes, which is equal to the number of nodes in the output layer. It turns the activations of the output layer into a probability distribution of K probabilities, i.e. $\sum_{i=1}^{K} S(y_i) = 1$.

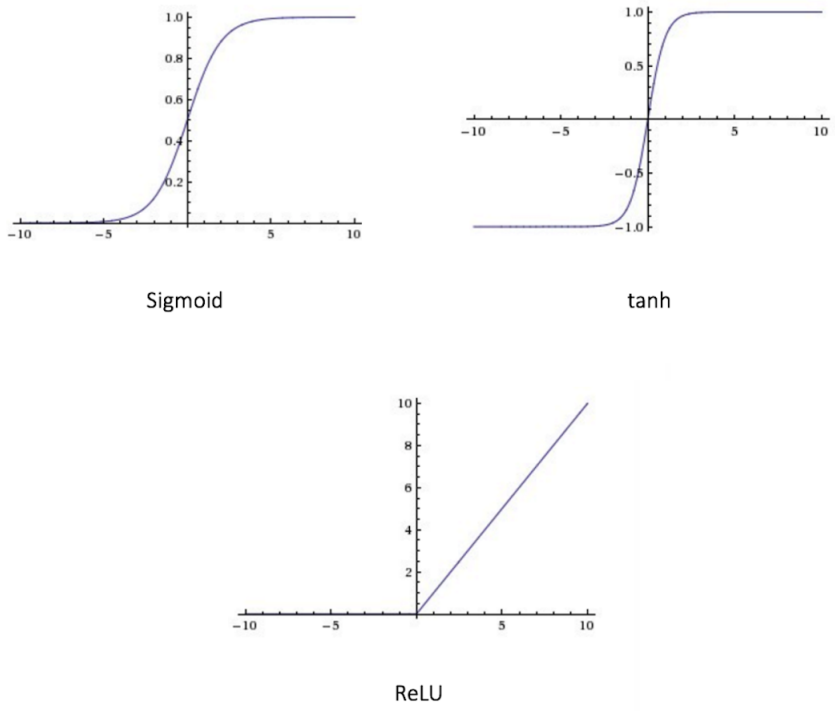


Figure 2.3: Comparison of the sigmoid, tanh and ReLU activation functions. Image retrieved from [21].

2.1.4 Loss functions

Supervised learning works by optimizing a model's internal parameters in order to maximize performance at some task, given a set of training inputs with desired outputs. In order for a neural network to learn, it is important to define some metric on which this optimization is based. Given a network's output based on some input, and the desired output corresponding to that input, we want to have a function that gives a useful measure of the difference between the target output and the actual output, i.e. the magnitude of the error. This function is called a *loss function*. A loss

function should be defined such that higher deviations of the actual output from the desired output produce higher loss values, meaning that decreasing the loss function should improve the model's performance at its particular task. The goal of the optimization task is then to update the neural network's internal parameters in such a way that the loss is minimized.

The loss function most commonly used for classification tasks, is the *cross entropy loss*. Cross entropy loss borrows the concept of entropy from information theory and quantifies the difference between two probability distributions. Given a target distribution P and an approximation of the distribution Q, the cross-entropy between P and Q is defined as the amount of bits required to represent some event with distribution Q when compared to P. Formally, it is given by equation 2.8.

$$CE(y_i, t_i) \; = \; -\sum_{j=1}^{C} t_i \log(y_i) \tag{2.8}$$

Here, y_i is the network output for input sample i, t_i the target output for sample i, C the number of classes and log the natural logarithm . Note that the cross entropy loss is commonly used with output that has passed through the softmax function (eq. 2.7), as it requires its inputs to be probability distributions.

2.1.5 Backpropagation and optimization

Given a neural network and a loss calculated for a particular sample, we need a method to update the neural network's parameters such that the value of the loss decreases. In general, this is done using optimization algorithms such as *stochastic gradient descent* (SGD) [22]. Algorithms based on gradient methods require the gradient of the loss function with respect to each weight in each of the network's layers to be computed. The backpropagation algorithm does this by using the chain rule to calculate the first order derivative of the loss function with respect to the weights at the output. This gradient is then propagated backwards through the network, where for each layer the partial derivative of the loss with respect to each weight is calculated, all the way until the first layer [23]. Essentially, we now know the contribution of each individual weight to the loss, either in the positive or negative direction. Note that backpropagation requires the activation functions used throughout the network to be differentiable.

Gradient-based optimization methods then use these partial derivatives to update each weight. For a weight update based on gradient descent, the update is a generalization of the perceptron weight update given in equation 2.2, such that it is suitable for neural networks consisting of multiple layers. The gradient descent

update for a weight w_j is defined in equation 2.9.

$$\Delta w_j = -\eta \frac{\delta E}{\delta w_j} \tag{2.9}$$

Here, $\frac{\delta E}{\delta w_j}$ is the partial derivative of loss E with respect to the weight and η the learning rate. $\frac{\delta E}{\delta w_j}$ gives the value that maximizes the change of w_j with respect to the loss. Since we want to minimize the loss, we apply this change in a negative direction. Neural networks are typically trained until their loss converges to a local minimum.

Batch gradient descent calculates the gradients based on all input patterns in the training set, which is computationally expensive. Stochastic gradient descent provides a method for determining the gradients based on one input sample at the time, which is more cost-effective and helps a model to converge for large datasets. SGD-based weight updates, however, can often be quite noisy. Therefore, in practice, a combination of batch gradient descent and SGD is used, where gradients are computed for a subset, or *mini-batch*, of input patterns.

In this study, a extension to the SGD algorithm called the Adaptive Moment Optimization (Adam) is used. It combines the principles of different optimization methods. It uses an exponentially decaying average of the previous gradients (first moment) and the square of previous gradients (second moment) to compute an adaptive learning rate for each parameter. This means that, in addition to a global learning rate η, Adam requires additional hyperparameters β_1 and β_2 that determine the decay rates of the first and second moments. Adam is a state-of-the-art method that is quite popular in deep learning, as it is able to achieve good results quickly [24].

2.1.6 Convolutional neural networks

Multilayer perceptrons require input in the form of a flattened vector, i.e. in a matrix of size $N \times 1$ with N the number of input values. When dealing with image input, this becomes problematic for multiple reasons. First of all, MLPs are ill-equipped to deal with the high-dimensional nature of image data. An image of height h and width w is typically represented by a $h \times w$ matrix for each color channel, meaning that even a small 100×100 image with 3 (RGB) color channels results in a 30000×1 MLP input vector. Combined with the fact that all layers in an MLP are fully connected, MLPs used in an image processing context are difficult to train and prone to overfitting. Additionally, important information on an image is not only retrieved from the presence of features, but also from how the features are located relative to

each other. Flattening an input image into a single vector removes this information from the data.

Convolutional neural networks (CNNs) are a class of neural networks that tackle these problems and are the de facto standard for image-related machine learning problems [25] [26]. They are even used for non-visual tasks, such as natural language processing [27] and time series analysis [28]. At the core of CNNs are convolutional layers that consist of a set of trainable filters, that learn to extract useful features from their input. A filter, or kernel, consists of a weight matrix that is smaller in dimension than the input image. This kernel is then convolved with the input, where at each step the element-wise product, or Hadamard product, of the kernel weights and the corresponding values of the input is computed. The resulting matrix is then summed, giving a scalar value for the corresponding element in the output. A visualization of this step can be found in Figure 2.4, The kernel is moved across the input in a step-wise manner, its movement specified by a *stride* value that determines the step size, and a *padding* method that tells the kernel how to deal with the input boundaries. The convolution operation of a kernel with its input results in a *feature map*, containing fine-grained features extracted from the layer's input. Since a single convolutional layers consists of multiple filters, a layer's output consists of a set of such feature maps. Convolutional layers in the early stages of a network learn to extract low-level features from an image, while layers further on learn to combine the low-level features into large scale patterns. Since the convolution of two matrices is a linear operation, a convolutional layer is followed by a non-linear activation function to induce non-linearity.

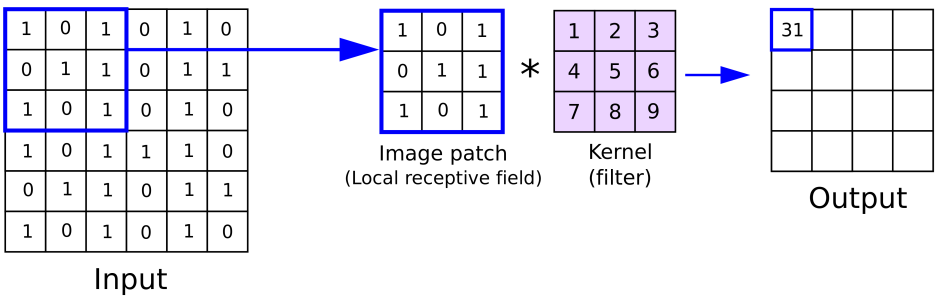


Figure 2.4: Example of a convolution operation between an input matrix and kernel with one channel. Image retrieved from [29].

In addition to convolutional layers, CNNs contain pooling layers that reduce the

dimensionality of the information flowing through the network. Similarly to a convolutional filter, a sliding window is moved across the input and a pooling operation is performed on each window. The most commonly used pooling operation is *max pooling*, where only the maximum value in the window is passed on to the output. In addition to decreasing the computional requirements of training a CNN, pooling layers can reduce noise and help extract dominant features that are positionally and rotationally invariant, at the cost of losing some of the finer-grained details in the input.

After a set of convolutional and pooling layers, CNNs typically end with one or more fully connected layers, functioning in the same way as an MLP. The fully connected layers flattens the input so that it can be used for higher level reasoning, such as classification.

There are several methods commonly used to increase the rate of convergence and reduce overfitting. Batch normalization normalizes a neuron's output based on the mean and variance of the inputs in a given batch [30]. Another technique is to randomly "ignore" certain neurons during training, such that their associated weights are set to 0 and they don't contribute to a network's output. This prevents co-dependency between individual neurons, which leads to a more robust network less prone to overfitting [31]. Dropout, as this is called, is often applied as a regularization method to fully connected layers, while batch normalization is used for convolutional layers.

2.1.7 Recurrent neural networks

Both the MLP and CNN architectures described above consider one input sample at a time, such that the output at a time step only depends on the current input, disregarding inputs at earlier time steps. For many tasks, this assumption of independence between inputs is non-problematic. However, this assumption does not hold when dealing with time series data or natural language, where the interpretation of a particular input depends on the context with regards to the other inputs. Recurrent neural networks (RNNs) are a class of neural networks that take temporal dependencies into account by not only considering the input at the current time step, but its internal state at previous time steps as well. For example, the Elman Network consists of an input layer, a fully connected hidden layer and an output layer. Given an input at time step t, the output of the hidden layer takes the product of the input and its weights as in 2.3, but adds the product of a separate weight matrix and the hidden state output at step $t-1$ [32]. While these simpler RNNs work for small time frames, they struggle with learning across many time steps. In large part, this is due to the *vanishing gradient problem*. Recall that gradients depend on the

first-order derivatives of activation function functions, which usually squash their outputs between $[-1, 1]$ or $[0, 1]$. This means that the absolute value of the gradient becomes a value between 0 and 1. Repeated multiplication of such values lead to the gradient becoming 0, leading to layers occurring near the start of the network being unable to learn. When dealing with gradients with an absolute value larger than 1, the opposite occurs as well: repeated multiplication of such values lead to the gradients to saturate, i.e. explode into large values that destroy a network's capability to learn.

A popular type of RNN that avoids the vanishing gradient problem and is able to successfully learn across many time steps, is the long short-term memory network (LSTM) [33]. In addition to a hidden state h, the LSTM as an additional memory state c that allows information to freely flow through time. It contains three *gates* that control the information in the memory cell, based on the current input and the hidden state of the previous time step. The *forget gate* determines which information in the memory cell is retained, the *input gate* determines which information is added and the *output gate* regulates how the network's state determines the output and the hidden state passed on to the next time step. The memory cell allows for the backpropagated error to remain constant over time, preventing gradients from both vanishing and exploding.

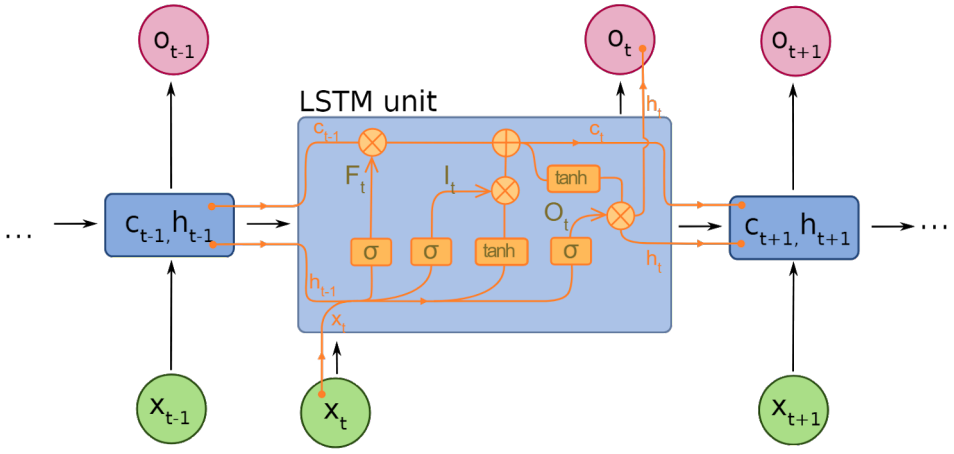


Figure 2.5: Diagram of an LSTM cell. Image retrieved from [34].

2.1.8 Sequence to sequence learning

In general, RNNs give one output for each time step or a single output after the last time step. VQA, on the other hand, takes a question of length n as input, but may require to output an answer of different length m. Indeed, a large number of tasks require many-to-many sequence prediction. Using RNNs for tasks that take a sequence as an input and require it to output a sequence of a different length, however, is non-trivial. In the case of natural language, for example, the output at a time step (a word or character) does not only depend on the history of the inputs, but also on the history of its previously generated outputs (the preceding part of the generated sequence). Here, so-called *sequence to sequence* or *seq2seq* models are often used. They require two separate networks: an *encoder* RNN and a *decoder* RNN. The encoder network processes the input at each time step, without yielding any output. The final state of the network now contains a representation of the input sequence, which is passed to the decoder network. This representation is sometimes referred to as the "thought vector". A $\langle start \rangle$ token is now passed to this decoder network and the output value at the first time step of the output sequence is predicted. This output value is then provided to the decoder network as input in the next time step. This cycle continues until either an $\langle end \rangle$ token is predicted or some maximum length is reached. Note that sequence to sequence models in NLP can either generate sentences on the word or character level. A diagram of a sequence to sequence model consisting of LSTMs can be seen in Figure 2.6.

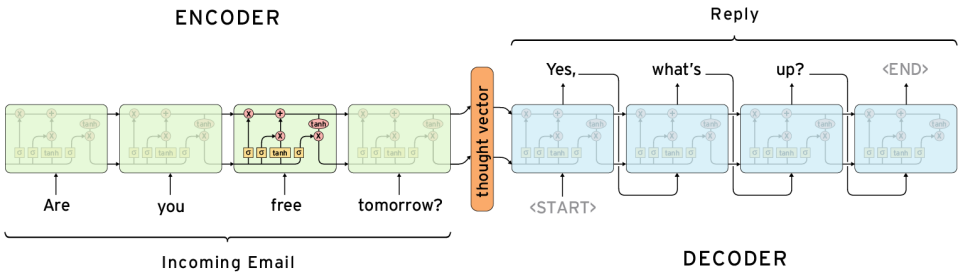


Figure 2.6: Diagram of an LSTM-based sequence to sequence model in a regular question-answering context. Image retrieved from [35].

It is interesting to note that the thought vector is somewhat Bayesian in nature, as it captures the state of the input augmented with the conditionals on which to base the output sequence. The conditionals can be factors such as the input's language or subject matter.

2.2 Natural language processing

Natural language processing seeks to bridge the gap between human-produced language and computers. It concerns tasks such as machine translation, natural language generation and speech recognition. Modern NLP solutions often use neural network-based methods, such as LSTMs and sequence to sequence models. Supervised learning, however, is but one of the many elements in the NLP-toolbox. In this chapter, we take a look at additional NLP techniques used in this study.

2.2.1 Word embeddings

The performance of neural networks on natural language tasks is largely dependent on how we choose to numerically represent the words we feed into a network. One of the simplest of these representations is the *one-hot encoding*. Given a vocabulary V of size $|V| = 4$ consisting of the words *man*, *woman*, *king*, and *queen*, we assign each word a unique integer label. The one-hot encoding vector of a word $v_i \in V$ with integer label i, is then given by a vector of length $|V|$ with the ith element set to one, and the rest to zero. The one-hot encoding for all four words is given in table 2.1.

	1	**2**	**3**	**4**
Man	1	0	0	0
Woman	0	1	0	0
King	0	0	1	0
Queen	0	0	0	1

Table 2.1: One-hot encodings for an example vocabulary consisting of four words. The leftmost column contains each word in natural language, the topmost row the corresponding integer labels. The 4-dimensional one-hot encoding vector for each word can be read from its corresponding row.

In practice, a vocabulary used in an NLP task can contain thousands of words. This means that the one-hot encoding for each word is a high-dimensional, extremely sparse vector. Additionally, this method of encoding contains no information about the similarity and semantic relations between words. While neural networks such as the LSTM described in paragraph 2.1.7 can be used to capture syntactic structures in natural language text, we also require them to learn the semantic meaning behind words and the relationships between words. Learning the semantic aspects of language is a challenge in itself and requires individual words

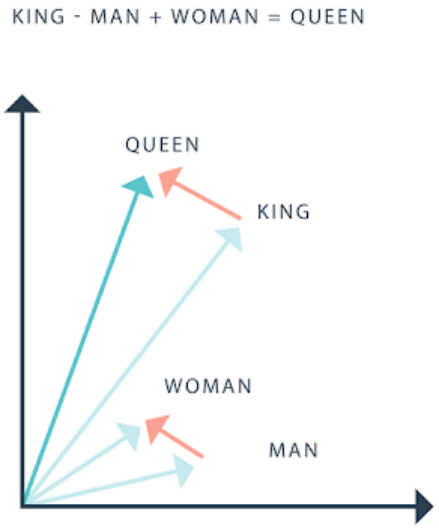


Figure 2.7: A hypothetical word embedding for a four-word vocabulary. Note that the difference between *queen* and *king* is equal to the difference between *woman* and *man*. Retrieved from [36].

to be represented in a way that captures semantic information. Ideally, each word would be mapped into a vector space such that similar words are close together, in a way that meaningful arithmetic can be performed on these representations with regards to their size and angle. For the four-word vocabulary example, it would look like Figure 2.7. Here, *king* and *queen* are close together, as are *man* and *woman*. Additionally, the difference between *queen* and *king* is numerically the same as the difference between *woman* and *man*.

This study uses a popular word embedding technique called Global Vectors for Word Representation, or GloVe, to represent individual words [37]. GloVe embeddings are low-dimensional vectors of 300 elements at most, that meet the requirement of capturing word similarity. It works by generating a co-occurrence matrix of words in a given *corpus*, a corpus being a structured set of texts often used for computational linguistics purposes. In a co-occurrence matrix X, each element X_{ij} stands for how frequently a word i occurs in the context of word j. Given their co-occurrence X_{ij}, a constraint is placed upon the word vectors $\mathbf{w_i}$ and $\mathbf{w_j}$ (equation 2.10).

$$\mathbf{w}_i^T \mathbf{w}_j + b_i + b_j = log(X_{ij}) \tag{2.10}$$

Here, b_i and b_j are scalar biases for both words. The problem is then posed as a weighted least-squares regression model by casting equation 2.10 as a regression problem and adding a weighting function $f(X_{ij})$. The weighting function ensures that the model not only learns from extremely common word pairs. The regression model J is given in equation 2.11.

$$J = \sum_{i,j=1}^{V} f(X_{ij})(\mathbf{w}_i^T \mathbf{w}_j + b_i + b_j - log(X_{ij}))^2 \tag{2.11}$$

The creators of GloVe have found the weighting function given in 2.12 to work well. They recommend taking $\alpha = 3/4$ [37].

$$f(X_{ij}) = \begin{cases} (X_{ij}/X_{max})^\alpha & \text{for } X_{ij} < X_{max} \\ 1 & \text{otherwise} \end{cases} \tag{2.12}$$

2.2.2 WordNet

In the age of *Good Old-Fashioned Artifical Intelligence*, models capturing the semantic relations between words were not learned, but constructed by hand instead. WordNet is such a hand-crafted lexicon, containing symbolic representations of words and the relations between them [38]. It includes the lexical categories nouns, verbs, adjectives and adverbs. Within a lexical category, words that are synonymous are grouped into *synsets*. Synsets are connected to each other through several semantic relationships. An important relation is the hypernym-hyponym relation, where X is a hypernym of Y if Y is a kind of X. Conversely, this means Y is a hyponym of X. Synsets can be hierarchically structured according to hypernym-hyponym relations, as can be seen in Figure 2.8. WordNet provides a method of exploring similarity and relations between words in a way that intuitive for humans, as opposed to the more abstract and numerical approach of word embeddings.

2.2.3 Sentence similarity scores

A common challenge in NLP is evaluating computer-generated sentences. In general the goal is, given one or more computer generated *candidate* sentence(s) and a target *reference* sentence, to calculate a score representing the similarity between the candidate(s) and the reference that corresponds to human judgement. The challenge lies in the fact that a single sentence can be constructed in many different ways that still have the same semantic content. In the context of VQA, a particular answer to a question can be phrased in multiple ways that are still correct.

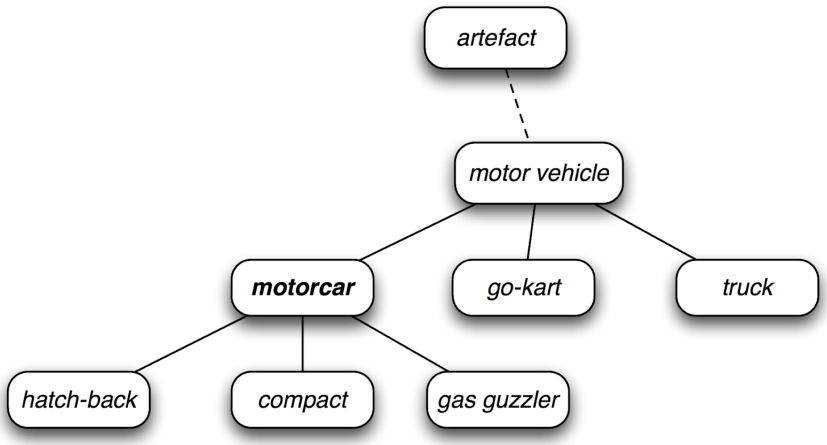


Figure 2.8: A subtree of WordNet representing hypernym-hyponym relations between several words. Retrieved from [39].

BLEU (bilingual evaluation understudy) is an often used metric for sentence similarity. It is one of the first of such metric to claim a high correlation with human judgement and is inexpensive to compute [40]. Given a candidate (machine-generate) sentence and a reference (target) sentence, BLEU computes a modified precision score for each n-gram (group of n words) in the candidate sentence, where precision is the proportion of n-grams that occur in both the candidate and reference sentence with respect to the total number of n-grams in the candidate sentence. For each n-gram m_w in a candidate sentence, its frequency in the reference sentence is counted and the maximum value m_{max} of m_w is set to this frequency. Without this modification to the precision score, the unigram BLEU-score for candidate sentence "the the the the the the the" and "the man is walking on the street" would return a perfect score, since each unigram in the candidate sentence occurs in the reference sentence. With the modification, the score becomes $\frac{2}{7}$ since the word "the" only occurs in the reference sentence twice.

METEOR is an extension on BLEU that takes both the precision and recall into account, with recall being the proportion of words that occur in both the candidate and reference sentence with respect to the total number of words in the reference sentence. A score F_{mean} combining precision P and recall R using the harmonic mean is computed with equation 2.13. Note that METEOR only uses unigrams to compute this score. METEOR requires an alignment to be made between the reference and candidate sentence, such that each word in the candidate translation

maps to either one or zero words in the reference, as illustrated in Figure 2.9. When multiple alignments are possible, the one with the fewest intersecting mappings is chosen.

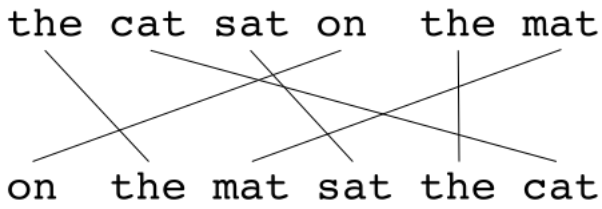


Figure 2.9: An alignment between two sentences. Note that multiple alignments are possible, but that the one with the fewest crossing lines is chosen. Retrieved from [41].

This alignment is used to calculate a penalty p. Both sentences are divided into chunks, where each chunk represents the largest group of words that are adjacent in both the reference and the candidate. The penalty p is then computed based on the amount of such chunks n_c divided by the total amount of mapped unigrams u_m, as in equation 2.14. The final METEOR score M is then computed with equation 2.15. Note that the value of p becomes 0.5 when computed for single-word sentences, meaning that the value of M gets capped at 0.5 as well.

$$F_{mean} = \frac{10PR}{9P + R} \tag{2.13}$$

$$p = 0.5 \left(\frac{n_c}{u_m} \right)^3 \tag{2.14}$$

$$\mathrm{M} = F_{mean}(1 - p) \tag{2.15}$$

Note that so far, only the presence and placement of words and/or n-grams in both sentences have been used to compute these metrics. In addition to this, METEOR has an option to use both stemming and synonomy when matching words. The former compares the stems of words, such that "walking" and "walks" are a positive match. The latter matches words when they are synonyms as well, such as in the case of "cat" and "feline". Both stemming and synonymy is done using WordNet [42].

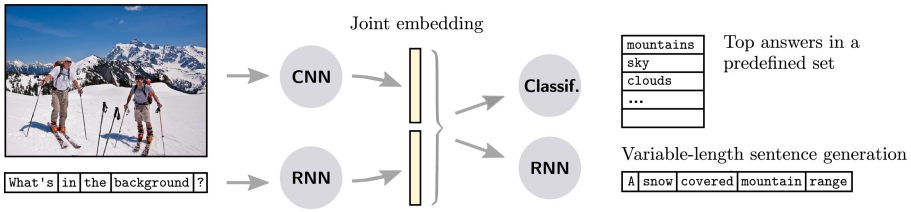


Figure 2.10: A diagram of the general structure of joint embedding VQA models. Image retrieved from [3].

2.3 Visual question answering

The main challenge of VQA lies in successfully combining both computer vision and natural language processing. This chapter describes how the techniques described in chapters 2.1 and 2.2 can be combined in order to build a complete visual question answering pipeline. Additional VQA-specific methods are explored as well.

2.3.1 Joint embedding models

The most common method used for VQA is based on transferring both image and text input to a common embedding. Inspired by developments in deep learning for both NLP and computer vision, these so-called joint embedding approaches allow for learning representations in a common feature space. This approach was first explored in the context of image captioning, such as in [43]. It has shown to be useful for performing inference over both the image and question input, which makes it quite well suited for the VQA task as well. The joint embedding approach in its most basic form can be divided into three stages. First, a useful representation of both the image and question features is obtained. Then, these representations need to be combined such that both the visual and language features are retained. In the last step, this combined representation is used to determine the answer output. In this section, we look at some common methods for implementing each of these stages. A diagram of the general structure of joint embedding models can be found in Figure 2.10.

Image and question features

Extracting features from the image input is in almost all cases done with a CNN. Often, models are used that were pre-trained on object recognition, usually with images from the ImageNet database [44]. Here, the last layer of the CNN is re-

moved, as this specifically corresponds to the classes of the dataset the network was originally trained on. This gives us a flattened $1 \times N$ feature vector representation of the input image. Commonly used architectures are VGGNet [45] (Figure 2.11), GoogLeNet [46] and ResNet [47].

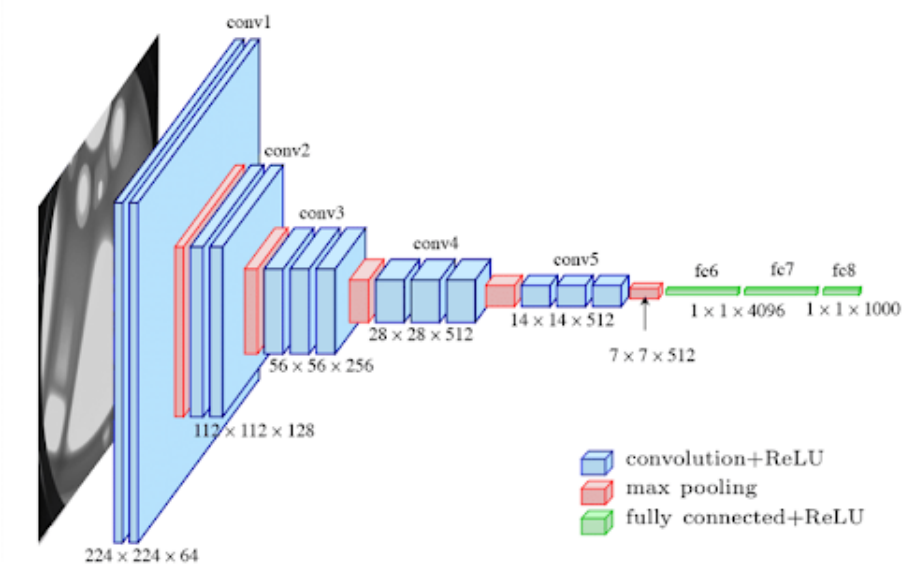


Figure 2.11: Diagram of the VGG16 convolutional neural network architecture. Image retrieved from [21].

Many joint embedding models use a combination of some word embedding and a recurrent neural network in order to featurize the input question. A popular method is to use pre-trained word embeddings such as Word2Vec [48] or GLoVe [37]. Each word in the input question is embedded and fed into the RNN, which is often an LSTM [33] or the similar *gated recurrent unit* (GRU) [49]. The RNN does not compute an output value at each time step. Instead, once each token in the input has been entered into the RNN, the network's internal state is used as a featurization of the entire question.

RNNs have been criticized for being difficult to fine-tune, taking a large amount of time to train and being complex due to a high amount of trainable parameters. In addition to this, some studies have found convolutional architectures to outperform RNNs in modelling sequences [50–52]. Some studies therefore use a CNN to encode the input question. For example, [53] propose a CNN-only method that uses CNNs

for embedding the image, the question and the combination of the two. They pass 3x3 convolutions over word embeddings of the input, meaning they look at each word and its immediate neighbours. They use a third multi-modal CNN to create a joint embedding of the image and question representations. [54] perform a more extended study on using CNNs for character and word-level question embeddings. They take a joint LSTM-CNN model as a baseline and replace the LSTM in that model with several different CNN architectures.

Combining the multimodal features

One of the main challenges of VQA is combining both the linguistic and the visual information in a useful manner. The first step is to represent both the input image and question in a common space, which is done using the methods described above. Given two feature vectors, one corresponding to the input image and the other to the input question, the next step is to combine them in some way. Here, basic linear operations are commonly used, such as:

- Element-wise addition of the vectors.
- Element-wise multiplication (Hadamard product) of the vectors.
- A combination of element-wise addition and multiplication. This is done, for example, in the *DualNet* architecture proposed by Saito et al. [55].
- Concatenation (adjoin) of the vectors.
- Taking the outer product of the vectors (bilinear pooling).

Note that the last method mentioned, bilinear pooling, encodes the full second order interactions between all elements in both vectors by taking their outer product. While this representation allows for modeling rich interactions between the two vectors, computing it is a challenge of its own: taking the outer product of both feature vectors is simply too computationally demanding and vastly increases training time. In general, bilinear pooling approaches therefore rely on simplification or approximation of this outer product. Fukui et al. propose Multimodal compact bilinear pooling (MCB), which projects the image and question representations into lower dimensional spaces [56]. A Fourier transform is applied to both projections, after which they are combined in the Fourier space by element-wise multiplication. Ben-younes et al. reduce model complexity by expressing the outer product tensor using Tucker decomposition, which decomposes a tensor into a set of matrices and a (small) core tensor [57].

Some approaches focus on learning an optimal representation. An example is the Multimodal Residual Learning architecture proposed in [58], where residual learning blocks are used to learn a joint representation of both visual and language inputs.

Output

Given a combined representation of the input features, the most straightforward way to determine an output answer is to treat the VQA task as a classification problem. Generally, a list of the top n answers in the dataset used for training the model define the classes. The classification itself is commonly done by an MLP, meaning that the combined features are passed into a series of fully connected layers where the size of the final layer equals n.

While treating VQA as a classification problem is relatively simple and makes inference computationally inexpensive, it is arguably an unrealistic representation of intelligent question answering. In principle, the set of possible sentences, and therefore possible answers to a question, is infinite. In this regard, it's counterintuitive to limit a question answering system to a maximum number of pre-defined answers. Related to this, a classifier-based VQA system is unable to capture the compositionality of human language: even if it is able to learn the meaning of individual concepts, it will not know how to produce an answer containing an arbitrary combination of these concepts. Given an image of a bag containing an apple and a pear and that both "an apple" and "a pear" are in the list of possible answers, a classifier VQA model can only answer with either "an apple" or "a pear", unless a variation of the phrase "an apple and a pear" happens to occur in the top n answers in the dataset. Additionally, a particular answer in such a system can only be phrased in a single way, while natural language would be able to convey it in many different manners.

A more intelligent approach would therefore be to treat VQA as a sequence generation problem instead of a classification problem. This is usually done by constructing the VQA system as a sequence to sequence model as described in section 2.1.8. Such a model is not bound by a pre-defined list of answers and is able to construct new sentences using arbitrary combinations of tokens, meaning it is better equipped to deal with the compositionality of natural language. Instead of several fully connected layers, the output is determined by a separate RNN which acts as a decoder. This is commonly done by treating the combined input image and question features as the thought vector to be passed on to the decoder network. At each time step, the decoder's output is passed through a softmax function which picks a word or character to be generated. Gao et al. use a variation on this method: instead of let-

ting the decoder RNN directly produce the output, a separate set of fully connected layers is used to combine the image features, the question features, the decoder RNNs internal state and the decoder RNNs output at each time step, before passing it through a softmax function [59]. While it is possible to do sequence to sequence VQA on both the word and character level, most studies on VQA only consider using whole words. Wang et al. found their model to perform better when using word-based representations compared to character based representations. They argue that this to be due to the brevity of the language data used in VQA: they believe that the input questions are simply too short for the model to learn that the space character is the delimiter for words [54].

2.3.2 Attention mechanisms

So far, the methods described have used the entire input image in their joint embeddings. This is somewhat counterintuitive with respect to how a human would typically approach the VQA task. Images can convey a lot of complex information, most of which is not useful when answering a specific question. Indeed, findings in cognitive psychology argue against the interpretation of human vision as a picture-like representation of the world in which everything is present simultaneously in equal detail [60]. Given a question and an image, a human is primed to look for certain objects in the image based on the semantic contents of the question. It therefore makes sense for a human to first identify the image regions that are relevant to this question and to give more importance to these regions, filtering out image regions that do not contribute to finding a solution.

This principle also seems to hold in machine learning: teaching a machine to "attend" relevant regions of the input space instead of using global features alone can greatly increase performance in various fields of machine learning [6]. These so-called *attention mechanisms* are used successfully in both NLP and computer vision applications, which suggests they might be interesting for VQA as well. Indeed, many studies have examined the use of attention for VQA in some capacity. Usually, the attention is incorporated in the image representation, such that the contents of the input question determine the salient regions in the image. Both question and image features are used to compute some weighted attention distribution over regions in the input image. This weighted distribution is then combined with the original image features, giving a set of image-weighted features. Generally, these image-weighted features are then combined with the question features and passed to the part of the model responsible for the output, just as in regular joint embedding models. A diagram of the general shape of joint embedding models with attention can be found in Figure 2.12.

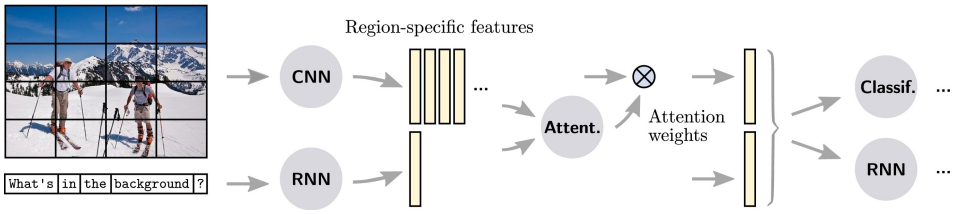


Figure 2.12: A diagram of the general structure of joint embedding models with attention [3].

An added bonus of attention is that it provides a level of explainability to a VQA model, as the attention distribution can be easily visualized. More specifically, one can generate a heat map based on the values of the attention weights. Overlaying this heat map on the original image then shows exactly which local regions of the image the network attended when computing its output, which makes the model itself more transparent.

Local feature representation

A non-trivial part of using attention in VQA is finding a representation of image features at the local level, such that local features from regions relevant to answering a question can be weighted based on their importance. A common way to do this, is dividing the input image into a grid of $m \times n$ regions, and using the image features in each rectangle in the grid as a local region. In most joint embedding models without attention, the image features from the final CNN-layer are used, which is a flattened vector in which all spatial information is lost. Instead, many attention methods take the features from the last layer before the pooling operations that flatten the features. This gives an $m \times n$ grid of feature vectors, uniformly spread across the image.

An alternative method is to propose a set of non-uniform local regions for each image and to encode each region with a CNN. For example, both [61] and [62] use the *Edge Boxes* from [63] to propose a set of bounding boxes for each input image. These bounding boxes are then treated as the local regions on which the attention weights are calculated.

Das et al. take yet another approach: they have created the VQA-HAT (Human ATtention) data set, where human participants were subjected to the VQA task and asked to annotate the parts of the image that contributed to their answer [64]. This dataset can be used to train attention mechanisms or act as a baseline with which other attention mechanisms can be compared.

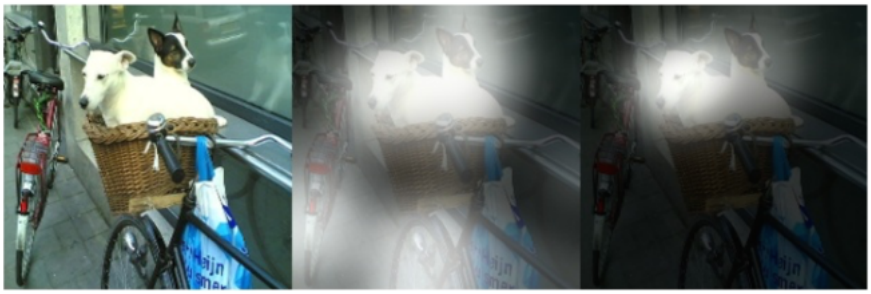


Figure 2.13: An example of stacked attention with two layers. The question asked is: "What are sitting in the basket on the bicycle?". Retrieved from [10].

Stacked attention networks

In this study, an attention architecture with grid-based spatial attention called *stacked attention networks* (SANs) is used [10]. Like the grid-based attention methods described above, SANs take a set of $m \times n$ feature vectors from the image processing CNN and use the semantic representation of the input question to query this image. The novelty of this method is that it queries the image multiple times: the authors of [10] argue that the VQA task often requires multi-step reasoning and therefore propose an architecture with multiple attention layers, also referred to as *glimpses*. The first layer queries all potentially relevant regions in the input image, while subsequent layers look at subsets of the original set of regions, eventually pinpointing the most relevant region or regions for answering the question. An example of the result of two-layer stacked attention can be seen in Figure 2.13, where the question "What are sitting in the basket on the bicycle?" is asked with respect to an image with two dogs sitting in a bicycle-mounted basket. The first attention layer attends both bicycles and the dogs, while the second layer only attends the dogs themselves.

For a matrix of image features $\mathbf{v}_I$ and vector of question features $\mathbf{v}_Q$, the authors of [10] pass both to one or more MLPs of one hidden layer, where the number of MLPs corresponds to the number of desired glimpses. For each of the k glimpses, the outputs of the hidden layer and the output layer of the corresponding MLP is formally represented by equations 2.16 and 2.17.

$$\mathbf{h}_A^k = \tanh(\mathbf{W}_{I,A}^k \mathbf{v}_I \oplus (\mathbf{W}_{Q,A}^k \mathbf{u}^{k-1} + b_A^k)) \tag{2.16}$$

$$\mathbf{p}_I^k = \text{softmax}(\mathbf{W}_P^k \mathbf{h}_A^k + b_P^k) \tag{2.17}$$

Here $v_I \in \mathbb{R}^{d \times m}$, with d the size of each image feature vector and m the number of image regions in the grid. $v_Q \in \mathbb{R}^d$ is a vector of size d. $W^k_{I,A}$ and $W^k_{Q,A}$ are the weights at glimpse k for the image features and the question features, respectively. W^k_P stands for the weights between the hidden layer and the output. b^k_A and b^k_P denote the biases at both layers. $\oplus$ stands for the addition of a matrix and a vector, meaning that the vector at the right side of the operator is combined with each column of the matrix at the left side by point-wise addition. u^k is the query vector at glimpse k, with u^0 set to be v_Q and subsequent values of u^k determined by equation 2.19. Now, $p^k_I \in \mathbb{R}^m$ is a vector of size m, where each element l stands for the attention probability of each of the m image regions. For each image region index l, we multiply the lth element of p with the lth feature vector in v_i and calculate the weighted sum across all image regions (equation 2.18). This results in an attention-weighted image feature vector $\tilde{v}^k_I$, which is used to update the query vector u^k as per equation 2.19.

$$\tilde{v}^k_I = \sum_l p^k_l v_l \tag{2.18}$$

$$u^k = \tilde{v}^k_I + u^{k-1} \tag{2.19}$$

The authors of [10] use the final value of u^k to determine the final answer output of the VQA model.

2.3.3 Other VQA algorithms

Compositional models

The downside of methods that rely on CNNs and/or RNNs for image and question feature extraction, is that these extraction models are somewhat monolithic in nature. Joint embedding methods use a "one size fits all" approach to VQA, where the computations done by the model are the same for every input, no matter the type or complexity of the question asked. Compare, for example, the complexity and compositionality of the questions "Is this a car?" and "Why is the man to the right of the third chair holding a hammer in his left hand?": the former represents a simple binary classification task, while the latter requires a combination of object recognition, spatial recognition, counting and in-depth reasoning across multiple objects. It would make sense to use an approach that is modular in nature, where its size and shape depend on the nature of the inputs. Here, two types of compositional models that take such a modular approach are briefly described.

Andreas et al. have proposed a method called Neural Module Networks (NMN) [65]. They define a set of modules, where each module is trained to perform one

specific task. Each module has a strictly defined input and output type. For example, the attention module outputs an attention distribution based on an image while the classification module outputs label based on an image plus an attention distribution. They use a parser to express the grammatical relations between parts of the input question, from which a symbolic representation is extracted, formatted as a tree. Based on a set of rules, each node in this tree is then replaced by one of the modules. They still use a pre-trained CNN to extract image features, but the architecture of the NMN network itself completely depends on the contents of the input question.

Dynamic Memory Networks (DMN) are another modular approach used for general question answering that use a memory component together with attention mechanisms [66]. Xiong et al. first applied this method to the VQA task [67]. Their model has a sentence input model that embeds the natural language input and passes it to a layer with a bi-directional GRU, which retrieves a set of "facts" from the input. The bi-directional aspect of the GRU allows for its internal state at a certain time step to be influenced by both past and future time steps, meaning it allows for information to flow between different inputs. Visual facts are retrieved from the input image in a similar fashion: a pre-trained CNN traverses the image in a snake-like manner, extracting features from patches in the image. These are passed to a bi-directional GRU, allowing for interactions between different image patches. An episodic memory module, consisting of an attention-based GRU, is then used to pass over the set of facts multiple times, selecting facts that are relevant for answering the question and constructing context vectors that model the interactions between these facts. The final state of this module is then used to output an answer.

Knowledge base approaches

Some questions in the VQA task require complex reasoning that is not directly related to the visual information in the input image, but instead rely on general knowledge about the world. Consider an image of the Eiffel Tower with the question "Which city is pictured here", which requires the knowledge that that particular structure is located in Paris. Regular joint embedding methods can only deal with information that is present in the training set, which limits its ability to answer such questions. Many approaches therefore augment their models with information from external knowledge bases. Wu et al., for example, extract individual attributes from an input image using a CNN and use the top 5 attributes to query DBpedia, an external knowledge database [2]. At the same time, they use an LSTM to generate a caption describing the image. The extracted attributes, the encoded response from the knowledge base and a vector representing the generated caption are then com-

bined and passed as input to an encoder LSTM, whose final internal state is then passed to a decoder LSTM to generate an answer sequence.

2.3.4 Question-answer generation

Several studies explore the concept of question generation (QG) as a means to augment and diversify question-answer data for visual question answering. The different methods can be roughly divided into two categories: template-based methods and generative methods.

Template-based approach commonly use a combination of image annotations provided by the dataset and so-called templates, which are natural language sentence with one or more words left blank, where the blanks are filled in based on the annotations for a given image. An early study that used this method was done in 2011, where templates were used to create questions based on an input text [13]. In [14], the authors use the object annotations provided by the VQA-real dataset to construct QA-pairs with regards to the presence of objects in an image. They derive binary "yes/no"-questions ("Is there a...?") and counting questions ("How many...?") directly from the list of objects in an image. Additionally, they look at the *supercategories* (supersets) of objects in the image to query what particular instance of an object category can be seen, e.g. asking "What type of vehicle can be seen?" when "car" can be found in the object list. If several supercategories relate to a particular environment, e.g. "outdoors" or "kitchen", they generate questions about the location where the picture was taken. If multiple supercategories relate to a particular sport, they generate QA-pairs with regards to the type of sports activity being displayed. [8] use the same templates for generating new QA-pairs, and extend these positional reasoning templates based on the location of the bounding boxes of different objects. Additionally, they create "absurd" QA-pairs that are unanswerable given the image. While the authors mainly use VQA-real for the above templates, they use Visual Genome to create additional positional reasoning QA-sets and QA-pairs that query the color of certain objects. In [12], the authors parse scene description annotations, which provide a description of a particular image using a full sentence, and construct a syntactic tree. They then employ several different algorithms that each perform a set of mutations on the generated syntax tree to generate question-answer pairs in several different categories.

Another approach is to use machine learning techniques to train a model to generate new QA-pairs based on either the image alone, or on a combination of the image and a set of annotations. In addition to the template-based approach described in the previous paragraph, [14] constructs an LSTM-based method that is trained on the VQA-real training data to generate both a question and answer word-by-word,

based on an input image. To ensure the generated question is relevant to the image, they generate a set of 30 candidate questions and pick the most frequent one. In order to ensure diversity, the generated question is rejected if it already exists somewhere in the dataset as a whole. In [15], a more complex method is used. First, they sample a set of N pre-defined question types that represent the various QA-types (e.g. "What", "Where" etc.) and encode each together with the input image. Each encoded image-question type vector is used as a condition for a conditional variational autoencoder, which generates several candidate QA-pairs. A separate network is trained to learn a specific question type distribution for an image and is used to select the final set of QA-pairs to be included in the dataset. This is based on the intuition that the probability of different question types occurring is different depending on the content of images.

Chapter 3

Methods

In this chapter, the methods used to answer the research question are described. Section 3.1 takes a look at the dataset used in this study, while section 3.2 explains how new question-answer pairs are generated within this dataset. Section 3.3 provides an overview of which models were used and how they were set up. Finally, section 3.4 describes the experiments done using these models and data.

3.1 Dataset used in this study

In this study, the Visual Genome (VG) data set is used [16]. It is a VQA dataset that, in addition to a collection of training images with human-generated question-answer pairs, contains extensive annotations for each image, descriptions of particular regions in each images, in addition to attributes of and relationships between elements in the image. While it has fewer images than the most popular dataset, VQA-real, it contains over 1.7 million question-answer pairs, meaning it has one of the highest amounts of question-answer diversity of any VQA dataset. Some recent studies have used VG to augment the training data from some other datasets, but how the rich collection of annotations in the VG data can be used for designing and training VQA systems is still an open research question [3]. The amount of images and QA-pairs in the dataset, plus the average length in words of both questions and answers, can be found in Table 3.1.

Total images	108,077
Total QA-pairs	1,773,258
Average QA-pairs per image	16.4
Average question length (words)	6.0 ± 1.9
Average answer length (words)	1.9 ± 1.3

Table 3.1: Statistics on the images and QA-pairs in the Visual Genome dataset.

Visual Genome distinguishes seven types of answers, based on the word with which they start [16]. The types are "what", "where", "when", "who", "why", "how", "which". The total amount and proportion of QA-pairs for each category can be found in Table 3.2. The dataset's creators have left out binary questions, i.e. questions that can be answered with either "yes" or "no". For example, 38.37% of the QA-pairs in the VQA-real dataset, consist of binary questions, meaning that a trivial model can achieve a reasonable performance by merely predicting "yes" or "no" as an answer [1]. Additionally, human bias appears to play a role when coming up with such questions: a human annotator is more likely to ask a question about something that is present in a particular image. In the first iteration of the VQA-real dataset, this led to questions starting with "Is there a..." being predominantly answered with "yes" [1]. This means that a model trained on this data is prone to merely learning the prior probability of an answer occurring.

Type	What	Where	When	Who	Why	How	Which
Number	874,318	50,883	245,072	78,978	38,725	157,343	188,068
Percentage	49.3%	2.9%	13.8%	4.5%	2.2%	8.9%	10.6%

Table 3.2: Distribution of the different question-answer types across the seven QA categories in the Visual Genome dataset.

In addition to question-answer pairs, each image in Visual Genome comes paired with rich annotations [16]. These annotations are:

- **Region descriptions**, which provide natural language descriptions of specific regions in an image, together with a bounding boxes detailing the locations of

the region described.

- **Objects**. A list of objects that occur in the image.
- **Attributes** that each describe the state of one of the objects in the image. If there is a red car in the image, this is split into the attribute "red" belonging to the object "car". A single object can have multiple attributes attached to it.
- **Relationships** that connect two objects. These are usually in the form of a verb or preposition, such as "man wears hat" or "tree between rocks".
- **Region graphs**. These are directed graphs where the nodes consist of the objects, attributed and relationships for a given region.
- **A scene graph** that is the union of all region graphs, resulting in a single graph describing the entire image.

In this study, we use the objects and relationships to generate new question-answer pairs. The total amount, the unique amount and the average amount per image for both annotation types can be found in Table 3.3.

Ann. type	Objects	Relationships
Total	1,366,673	1,531,448
Unique	75,729	40,480
Avg. per image	21.24	17.68

Table 3.3: The total amount, the unique amount and the average amount per image for the object and relationship annotation types in the Visual Genome dataset.

All noun phrases that occur in QA-pairs and region descriptions, objects, attributes, and relationships are canonicalized to WordNet synsets [16]. This allows for effectively and consistently querying the same concept, in addition to having the potential to help train models that can learn from contextual information across multiple images.

3.2 Question-answer generation

The main goal of this study is to enhance Visual Genome's question-answer data with new, automatically generated samples, and to measure the effect this has on

a VQA model's performance. As mentioned in 3.2, VQA question-answer generation is commonly done by either using a template-based method, or by a generative machine learning-based approach. Here, a template is a natural language sentence with one or more words left blank, where the blanks are filled in based on the annotations for a given image. Such an approach is more straightforward than a generative machine learning-approach, as it does not require training and makes good use of the available data annotations in Visual Genome. Additionally, Kafle and Kanan found that template-based QA-pair enhancement provided a bigger performance boost than data augmentation based on a generative approach with recurrent neural networks [14]. We will therefore use a template-based method for enhancing the existing QA-data in Visual Genome.

In this study, templates will be used for both questions and answers, such that the objects and relationships of an image can be used to fill in these blanks, thus generating new question-answer pairs. We use these templates to generate three types of QA-pairs: binary (yes/no) questions, questions that start with "what" and questions that start with "where". For each category, several different templates are used that are phrased differently, as to ensure variety in both questions and answers. Our template-based approach is more extensive than the one designed in [14]: in their work, 86.2% of template-generated questions were binary questions, and almost all questions were related to the presence of objects. In another study, Kafle and Kanan mention using Visual Genome to generate new QA-pairs with templates. However, they make little use of VG's extensive annotations and only generate questions about spatial relations or colors. They do not report the effect their dataset enhancement has on the performance of their models [8]. In this study, we make sure binary question do not become too prevalent. Additionally, we generate more diverse QA-pairs that are harder to answer in the form of "what"/"where" questions, as the QA-pairs generated in these categories require reasoning about the relationship between objects.

External libraries will be used to make the generated sentences grammatically correct, such as NLTK [68]. NLTK has a built-in lemmatizer based on WordNet, which is used to determine whether nouns were singular or plural in order to choose the correct article for generated nouns. Furthermore, based on the plurality of the subject of a generated sentence, the sentence's verb is conjugated using a machine learning-based verb conjugator. While this combination of techniques is able to generate grammatically correct sentences in almost all cases, it is unable to deal with mass nouns, i.e. uncountable objects such as "water" or "air".

Note that from here on, certain tags will be used as placeholder tokens when describing the templates. "$\langle ART \rangle$" is used for articles, "$\langle NOUN \rangle$" for nouns, "$\langle VERB \rangle$" for verbs, and "$\langle PREP \rangle$" for prepositions.

3.2.1 Binary questions

As mentioned in section 3.1, the creators of Visual Genome have left binary questions out of their dataset on purpose to prevent the inclusion of too many easy-to-answer questions and because these questions are prone to bias. However, binary questions are relatively easy to generate and may help train a model to determine the presence of queried elements in an image, which is required for virtually any visually-grounded question. We alleviate the first problem by making sure the total amount of binary questions in the dataset does not become too large. In VQA-real, for example, 38.37% of all QA-pairs are answered with either "yes" or "no" [1]. Furthermore, as we'll see later, binary questions are not included in the test set on which the performance of each model is benchmarked. We circumvent the problem of bias, referring to the problem where the answer "yes" occurs significantly more often than "no" for binary questions, by making sure that for each generated QA-pair with answer "yes", a QA-pair with the answer "no" is generated. If possible, the latter queries an object that is similar to the queried object in the former, but that is not present in the image. This way, the generated "no"-question queries an object that might have believably been present but is not actually in the image, making the question more challenging to answer. Additionally, binary questions are kept out of the test set used to measure all models' performances. The templates used for the binary questions can be found in Table 3.4.

Question templates	**Answer templates**
"Is there $\langle ART \rangle \langle NOUN \rangle$?"	
"Is there $\langle ART \rangle \langle NOUN \rangle$ present?"	
"Does the image contain $\langle ART \rangle \langle NOUN \rangle$?"	"Yes." / "No."
"Is there $\langle ART \rangle \langle NOUN \rangle$ in the picture?"	
"Can $\langle ART \rangle \langle NOUN \rangle$ be seen?"	

Table 3.4: The templates used to generate the binary questions. The answer is either "Yes" or "No", depending on whether the queried noun is present in the image.

Note that there is no placeholder for the verb and that the binary question templates are all in the singular form, as we only query the presence of a single instance of an object, even when multiple are present in an image.

In order to generate binary questions for an image, the following algorithm is

used:

1. Construct a list of synsets corresponding to the objects present in the image. Only objects taking up an image area larger than a threshold of 2000 pixels are considered.

2. Select a random subset of these objects, of a size corresponding to half of the number of QA-pairs we want to generate.

3. For each object in this subset:

   (a) Determine the correct article ("a" or "an") for the object noun.

   (b) Generate a question using a random binary question template, plugging in the article and the noun for the placeholders and setting the corresponding answer to "Yes".

   (c) Using WordNet, try to find an antonym (the opposite) of the object that is not present in the image. If this cannot be found, try to find another hyponym of the object's hypernym, i.e. another object from the same class/type, that is not in the image. If this cannot be found either, which rarely occurs, select a random object that is not already in the image from the list of all objects that occur in Visual Genome.

   (d) For the object retrieved in the previous step, generate a binary question and fill in the noun and its correct article. The answer is set to "No".

3.2.2 "What" and "where" questions

The second and third types of generated QA-pairs are questions that start with the word "what" or "where". Both question types were generated from the binary relationship annotations in Visual Genome. These annotations describe the relationship between two objects by either using a verb, describing an action, or a preposition, describing a spatial relationship. "What"-type questions can be generated for both types, as these questions can either query about an action or a location. Since "where"-type questions can only query about the location of an element, meaning they were only generated using preposition-based relationship annotations. The templates used to generate the "What"-QA-pairs, for both preposition- and verb-based relationships, can be found in Table 3.5. The templates used to generate QA-pairs starting with "Where" based on prepositional relationships can be found in Table 3.6.

Question templates	Answer templates
Preposition-based	
"What is/are $\langle PREP \rangle$" the $\langle NOUN2 \rangle$?	"$\langle ART \rangle \langle NOUN1 \rangle$ is/are $\langle PREP \rangle$ the $\langle NOUN2 \rangle$."
"What is/are located $\langle PREP \rangle$" the $\langle NOUN2 \rangle$?	"$\langle ART \rangle \langle NOUN1 \rangle$ is/are located $\langle PREP \rangle$ the $\langle NOUN2 \rangle$."
Verb-based	
"What does/do the $\langle NOUN1 \rangle \langle VERB \rangle$?"	"The $\langle NOUN1 \rangle \langle VERB \rangle$ $\langle ART \rangle \langle NOUN2 \rangle$."

Table 3.5: The templates used to generate the "What"-questions based on relationship annotations of the form "$\langle NOUN1 \rangle \langle PREP \rangle \langle NOUN2 \rangle$" for relationships that are prepositions, or "$\langle NOUN1 \rangle \langle VERB \rangle \langle NOUN2 \rangle$" for verbs.

Question templates	Answer template
"Where is/are the $\langle NOUN1 \rangle$?" "Where is/are the $\langle NOUN1 \rangle$ located?"	"$\langle PREP \rangle$ the $\langle NOUN2 \rangle$"

Table 3.6: The templates used to generate the "Where"-questions based on relationship annotations of the form "$\langle NOUN1 \rangle \langle PREP \rangle \langle NOUN2 \rangle$".

The format of relationship annotations is "$\langle NOUN1 \rangle \langle PREP \rangle \langle NOUN2 \rangle$" for relationships based on prepositions and "$\langle NOUN1 \rangle \langle VERB \rangle \langle NOUN2 \rangle$" for verbs. The algorithm used to generate both QA-types for a given image is simpler than the one used for binary questions:

1. For each relationship in the set of relationship annotations for the image:
   (a) Randomly select a question and answer template, corresponding to the QA-type and type of relationship (verb or preposition).
   (b) Determine the plurality of "$\langle NOUN1 \rangle$".

(c) Determine the correct article ("a" or "an") for "$\langle NOUN1 \rangle$".

(d) **If the relationship is verb-based:** conjugate the verb.

(e) Generate a question and answer by plugging the article, both nouns, preposition/verb into the template.

(f) **If the desired number of QA-pairs has been generated:** break.

Note that if we try to generate "Where"-type questions and the relationship for a given attribute is a verb, we simply skip to the next relationship attribute.

3.2.3 Generated datasets

With this template-based QA-pair generation methods, we create three new datasets, each with 2, 4 and 6 new QA-pairs per QA-category per image. That means that 6, 12 and 18 new QA-pairs are generated for each image in the dataset, respectively. From hereon, the original set of question-answer pairs as present in the Visual Genome dataset will be referred to as QA0. The three newly generated datasets will be referred to as QA2, QA4 and QA6. The total amount of question-answer pairs for each dataset can be found in Table 3.7.

Dataset	N QA-pairs
QA0	1,445,322
QA2	1,966,798
QA4	2,435,480
QA6	2,842,182

Table 3.7: Total number of question-answer pairs for the original dataset (QA0) and the augmented datasets (QA2, QA4, and QA6).

3.3 Models and hyperparameters

In order to measure the effect of enhancing the Visual Genome dataset with new QA-pairs, a baseline model is constructed based on the joint embedding approach with stacked attention. The model is based on the architecture proposed by Kazemi and Elqursh in [11], with a few modifications. Their proposed model is able to achieve good accuracy on the VQA-real dataset, while being architecturally simple,

which makes it a suitable baseline for comparing the different generated QA-sets. Two different versions of this baseline model are created: one where the output is treated as a *1-out-of-N* classification problem, and another where it is treated as a many-to-many sequence generation problem.

3.3.1 Image features

The input images are rescaled to a resolution of 448x448 pixels using Lanczos filtering, normalized and converted to the RGB-format. The image features are extracted using the VGG16 CNN architecture as seen in Figure 2.11. Since we want to use a grid-based attention mechanism, we extract the image features from the final layer before the last pooling layer, giving us a grid of 14x14 feature vectors of size 512. For one of the experiments, we replace the VGG16 architecture by ResNet-152, again extracting the features from the last layer before the final pooling layer. This gives us features of size 14x14x2048. The CNN's internal parameters are frozen, meaning that they do not update during training of the VQA-model at large. The feature vectors are regularized by using l2-normalization.

Note that we do not augment the image data by (randomly) applying operations such as shear, rotation, mirroring on the input images. While this is common practice in computer vision, using these augmentation methods is non-trivial for the VQA task. After all, many answers rely on the spatial relationships of elements in the image, which may not be maintained by the augmentation operations. Take, for example, the question "Where is the dog" with the answer label "To the left of the chair": mirroring this image would invalidate this QA-pair.

3.3.2 Question features

For each of the four datasets, a vocabulary is constructed consisting of all tokens in both the questions and answers in the dataset. Input questions are tokenized and each token is embedded using GloVe-vectors of size 300 that were pre-trained on the *Wikipedia 2014* and *Gigaword 5* corpora [37]. If a token in the input question does not occur in the set of pre-trained GloVe vectors, its vector elements are randomly sampled from a normal distribution with a standard deviation of 0.6. While it is possible to update the GloVe vectors during training, preliminary testing showed that this did not make a difference. Like the pre-trained CNN, the GloVe vector parameters are therefore frozen during training.

Each embedding is fed to an LSTM of one hidden layer with 1024 nodes. The final cell state of this LSTM is then used to represent the input question. The maximum token length of the input questions is capped at 25 words. The LSTM's weights

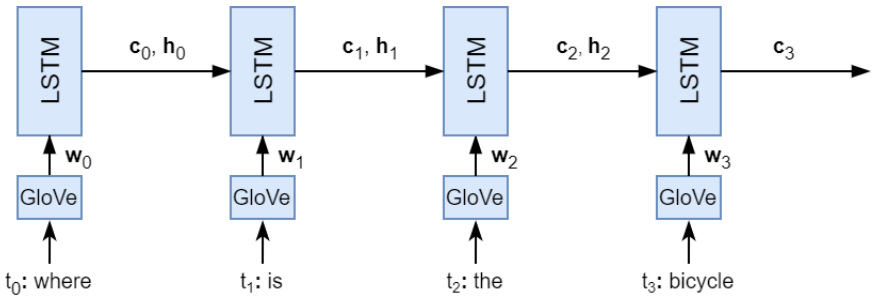


Figure 3.1: Diagram of the question feature extraction method used in this study. For each word at time step t_i, its GloVe embedding $\mathbf{w}_i$ is retrieved and passed as input to the LSTM, which outputs its cell and hidden states $\mathbf{c}_i$ and $\mathbf{h}_i$ to itself at the next time step. After the last input token, the final cell state is used to represent the question features.

are initialized using Glorot initialization [69]. A diagram of how the question feature extraction is set up in this study, can be found in Figure 3.1.

3.3.3 Stacked attention

Like in [11], a stacked attention network with two glimpses is implemented using a CNN, with one convolutional layer per glimpse. Two stacked attention maps are calculated using the 14x14 feature map grid extracted from the input image CNN and the state of the question LSTM. The CNN features are passed through a convolutional layer with kernel size 1 and 512 output channels. The LSTM state is put through a linear layer with output size 512 and then tiled such that the feature vector is repeated over a 14x14 grid. Both the newly computed CNN and LSTM features are combined into a single 14x14x512 grid of feature vectors by element-wise addition, which are passed through a ReLU function. Then, the combined features are passed through a convolutional layer with 2 output channels and kernel size 1, which produces the two attention maps. Finally, the attention maps are put through a softmax function and then multiplied with the original grid of feature maps produced by the image input CNN, which gives us a set of attention-weighted feature vectors.

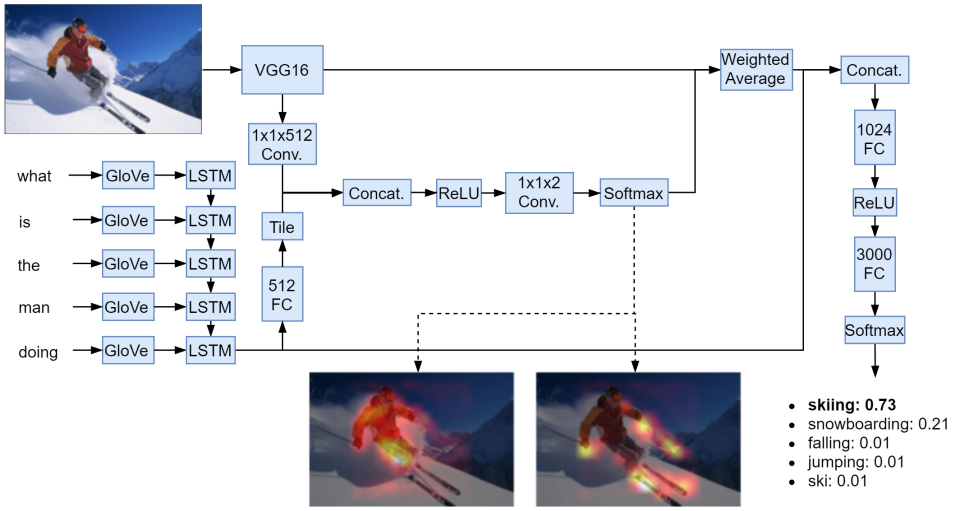


Figure 3.2: Diagram of the architecture of baseline model A as described in this chapter. It shows how the input image and question (left) are processed to compute the two attention maps (bottom) and predict an answer from a pre-defined list (right). The example image, question, answer, and attention maps are taken from [11].

3.3.4 Baseline model A: classifier

The first of the two variations on the baseline model treats VQA as a 1-out-of-N classification problem, taking the 3000 most frequently occurring answers in the dataset as our classes. First, the attention-weighted feature vectors and the question LSTM cell state are combined through concatenation and passed through a fully connected (FC) layer of size 1024 with ReLU activation. This is followed by a second fully connected layer of 3000 neurons, of which the ouput is passed through the softmax function and used to choose one of the 3000 possible answers. An overview of the architecture of baseline model A can be seen in Figure 3.2.

3.3.5 Baseline model B: LSTM-output

The second variation on the baseline model treats the task as a many-to-many sequence generation problem. A decoder LSTM is constructed, consisting of one hidden layer of size 1024, which generates the answer word-by-word. In addition to the question LSTM's cell state, we also concatenate the attention-weighted image feature vectors with the question LSTM's hidden state. Both vectors are then used to initialize the output LSTM's own cell and hidden state, respectively. From a

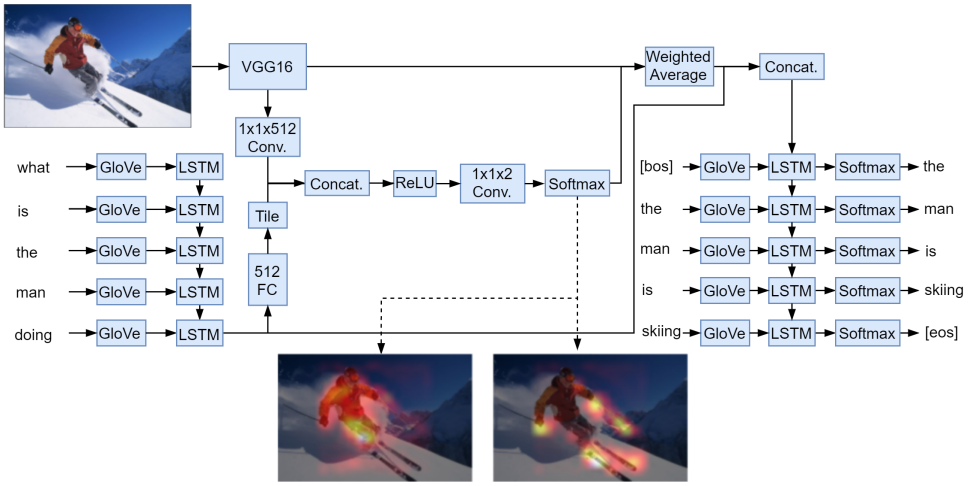


Figure 3.3: Diagram of the architecture of baseline model B as described in this chapter. It shows how the input image and question (left) are processed to compute the two attention maps (bottom) and predict an answer word-by-word (right). The example image, question, answer, and attention maps are taken from [11].

Bayesian perspective (see section 2.1.8), this thought vector's conditionals are the image features and the attention maps.

At the first time step, the "$\langle BOS \rangle$" (beginning of sentence) token is passed as input to the decoder LSTM. The LSTM's output is put through a softmax function and is used to generate a word from the vocabulary, which is then passed as input to the decoder LSTM at the second time step. This continues until the "$\langle EOS \rangle$" (end of sentence) token is generated. Like the question input LSTM, the decoder LSTM's weights are initialized with Glorot initialization [69]. The same vocabulary is used for both LSTMs. An overview of the architecture of baseline model B can be seen in Figure 3.3.

3.3.6 Training

The training samples are randomly split into a training and validation set with a ratio of 0.8/0.2, respectively. For the classifier output model, we calculate the cross entropy loss based on the predicted answer class and the target answer class. For the LSTM-output model, the cross entropy loss is based on the predicted token and the target token at each time step.

The choice of optimizer, optimizer parameters and dropout configuration was

kept the same as in [11]. The model is optimized using the Adam optimizer with a learning rate of 0.001, a β_1 of 0.9, and a β_2 of 0.999. Dropout with a rate of 0.5 is applied to the inputs to the question LSTM, attention layers, the fully connected layers for the classifier output, and the answer generation LSTM for the sequence to sequence output.

The amount of training epochs and the size of the mini-batches were determined by running some preliminary experiments. Each model is trained for 25 epochs, where each epoch consists of a full cycle of showing all training samples to the model, as the preliminary experiments showed that model convergence occurs within this period. The mini-batch size is set to 128. An overview of the hyperparameter settings used for both baseline models can be found in Table 3.8.

Parameter	**Baseline model A**	**Baseline model B**
Batch size	128	128
Optimizer	Adam	Adam
Learning rate	0.001	0.001
β_1	0.9	0.9
β_2	0.999	0.999
Loss function	Cross entropy	Cross entropy
Weight initialization	Glorot	Glorot
Dropout rate	0.5	0.5
Max. training epochs	25	25
Image size	448x448	448x448
Word embedding size	300	300
N hidden layers (input LSTM)	1	1
Hidden layer size (input LSTM)	1024	1024
Kernel size attention conv. layer 1	1x1	1x1
N output channels attention conv. layer 1	512	512
Kernel size attention conv. layer 2	1x1	1x1
N output channels attention conv. layer 2 (*glimpses)*	2	2
N hidden layers (output MLP)	1	-
Hidden layer size (output MLP)	1024	-
N answer classes	3000	-
N hidden layers (output LSTM)	-	1
Hidden layer size (output LSTM)	-	1024

Table 3.8: Hyperparameter settings used for both baseline model A and B.

3.4 Experiments

We now have four different datasets with varying amounts of generated QA-pairs, and two variations of a baseline VQA pipeline. Both baseline architectures are trained on all four datasets, and tested using several metrics that will be described in this section. Each model-dataset combination is trained three times, with random initialization of model weights and training-validation split within the training dataset. For each trained model, the model checkpoint that performed best on the validation set is used for testing. All testing is done using a holdout test set consisting of 10% of the QA-pairs in the original Visual Genome set (QA0), in order to provide a common ground. The experiments done to measure the performance of the classifier model and the LSTM-output model are described in sections 3.4.1 and 3.4.2, respectively. In addition to quantifying the performance of both baseline models, we also want to get a measure of the robustness of both models. These experiments are described in section 3.4.3. All significance tests are done using the chi-square test.

3.4.1 Baseline model A

The main metric used for measuring baseline model A's performance is classification accuracy, i.e. the proportion of QA-pairs in the test set that were answered correctly. During testing, QA-samples with answers that do not occur in the top-3000 answer list are skipped.

First, we train the model three times on each of the four datasets, and report the mean accuracy for each dataset-model combination. We repeat this for a variation of the classifier architecture where the VGG16 CNN extracting the input image features is replaced by ResNet-152, which is a much more complex architecture, in order to measure the effect it has on the overall VQA model's performance. Note that this is the only experiment with the ResNet-based architecture: all other experiments are done on models that use VGG16. For the models using the VGG16 architecture, we also report the accuracy on "what" and "where"-type QA-pairs only. Additionally, we report the BLEU and METEOR scores as well. In this case, they measure the similarity between the sentence chosen from the answer list and the target sentence. While it is uncommon to use these sentence similarity scores in such a manner, this is done in order to provide a means to directly compare the performance of baseline models A and B.

Additionally, we want to have a measure of how close the model's predictions are to the real answer, even when the selected output answer is wrong. The proportion of times the label answer of each test set QA-pair occurred in the top 10

predictions is reported as well.

Finally, in order to get some measure of how the models perform on unseen data, we report their performance on the test set of the VQA-real dataset, one of the most commonly used datasets for VQA [1]. It contains approximately twice the amount of images of Visual Genome, but has significantly fewer questions and (unique) answers. For this experiment, we used the validation set of "VQA 2.0", the second release of the VQA-real dataset. QA-pairs in this dataset are formatted differently from those in Visual Genome: each question has 10 human-annotated answers. The evaluation metric provided by VQA-real's creators is therefore used, which depends on how many human annotators agree on a particular model-predicted answer. Given the predicted answer a and the amount of annotator answers that agree on this answer n, the formula used to calculate the accuracy is given in Equation 3.1.

$$\text{Acc}(a) \;=\; \min\left(\frac{n}{3}, 1\right) \tag{3.1}$$

Note that this always gives an accuracy score between 0 and 1.

3.4.2 Baseline model B

Baseline model B is trained three times on each of the four datasets. BLEU and METEOR are used as performance metrics, as both are commonly reported in studies on natural language generation.

BLEU is calculated for all n-grams with $n \in \{1,\ 2,\ 3,\ 4\}$. The average of these values is reported as well. BLEU's default behavior is to return 0 when no n-gram overlaps are found, which is problematic as many answers in Visual Genome are approximately 1.9 words in length. This effect is mitigated by using a smoothing function that adds 0.1 to the precision when it has 0 counts. This same method of reporting is used for the BLEU/METEOR scores with regards to the classifier model.

3.4.3 Noise resistance

In the third set of experiments, the trained models' resistance to noise is measured. Noise resilience could be a useful metric for testing a VQA model's robustness to unseen question inputs: if a model's predicted answer to a particular question changes when noise is added to the question's embedding, it means that it's vulnerable to small changes in the question input. Given a particular image, a VQA system's answer should only depend on the semantic contents of the question and be independent of the exact phrasing of this question.

During testing, a random vector with the same size as the GloVe embedding vector is generated for each sample by drawing from a normal distribution with

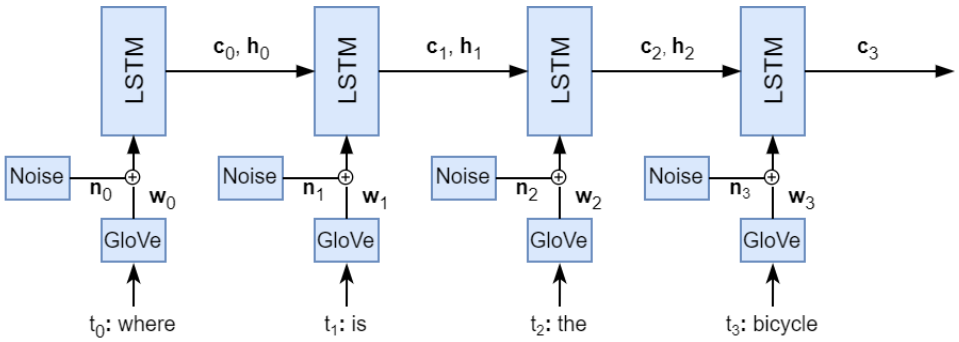


Figure 3.4: Modification of Figure 3.1 which shows how a noise vector $\mathbf{n}_i$ is added to word embedding $\mathbf{w}_i$ at each time step t_i.

mean 0 and variance 1. At each time step t_i, this noise vector $\mathbf{n}_i$ is multiplied by a scalar noise ratio $r \in \mathbb{R}$ and added to the question embedding $\mathbf{w}_i$. A new noise vector is generated at each time step. This process is shown in Figure 3.4. We take the trained models for both baseline architectures and repeatedly test them with the test set, each time increasing the value of r. For baseline model A, we look at the effect this has on the classification accuracy. For baseline model B, we measure the effect of increasing r on the METEOR score. The values used for r are in the range $[0.05, 0.1, 0.2, 0.3, ..., 1.0]$.

In order to more closely study the differences in noise robustness between the models trained on different datasets, we compare the different model-dataset combinations at three values of the noise ratio r: 0 (no noise), 0.5 (half of the noise vector), and 1 (the full noise vector). Again, this is done separately for baseline models A and B.

3.4.4 Attention maps

Finally, we take a look at the behavior of the stacked attention subnetwork. This is done by presenting a trained model from each dataset-model category with the same image and question, taken from the Visual Genome dataset, and visualizing the attention-weighted image produced by the model using a heat map. In the color scheme used, blue represents low importance, while red means the attention weight is high for a particular region. The heat maps are smoothed using Gaussian smoothing for a more coherent visualization.

3.5 Implementation and hardware

All code is written in Python. All VQA-architectures are implemented using the PyTorch machine learning library [70]. The library NLTK is used for auxiliary NLP tasks, such as ensuring correct grammar in generated QA-pairs, and calculating the BLEU and METEOR metrics [68].

All models were trained and tested on the Peregrine computer cluster, courtesy of the Center for Information Technology of the University of Groningen. Depending on availability, either Nvidia Tesla K40 or Nvidia Tesla V100 GPUs were used for training and testing the models.

Chapter 4

Results

In this chapter, the results from the experiments described in section 3.4 are reported. We take a look at a few generated question-answer-pairs (QA-pairs) in section 4.1. The results from the experiments measuring both baseline models' performances can be found in section 4.2.1 and 4.2.2. The effects of adding noise to the question embedding are described in section 4.4. Finally, we visualize some attention heat maps in section 4.5.

4.1 Generated question-answer pairs

It is hard to automatically determine the quality of generated question-answer-pairs (QA-pairs). This is therefore done qualitatively, by looking at samples from the set of QA-pairs generated by the algorithms in section 3.2. Figure 4.1 shows an image from the Visual Genome dataset which was used to generate new QA-pairs. Table 4.1 shows six examples of QA-pairs that were generated using this image: two binary questions, two "what"-type questions (one based on a preposition, one on a verb), and two "where"-type questions. The first column in the table shows which annotation was used to generate the QA-pair. Recall that object annotations are used for binary questions, and that relationship annotations are used for the other two categories.

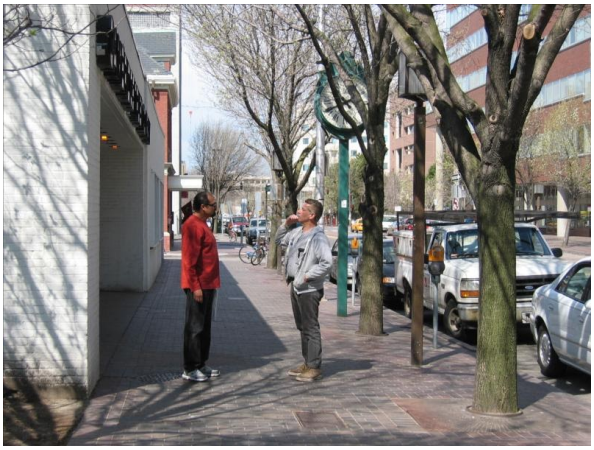


Figure 4.1: Example image from the Visual Genome dataset, used to generate the QA-pairs in Table 4.1.

Annotation	QA-type	Generated question	Generated answer
car	Binary	"Does the image contain a car?"	"Yes"
car	Binary	"Does the image contain a motorcycle?"	"No"
shade ON sidewalk	What (prep.)	"What is on the sidewalk?"	"A shade"
man WEARING shoes	What (verb)	"What does the man wear?"	"Shoes"
sidewalk NEXT TO street	Where	"Where is the sidewalk?"	"Next to the street"
bike PARKED ON sidewalk	Where	"Where is the bike located?"	"Parked on the sidewalk"

Table 4.1: Six examples of generated QA-pairs for the given image. The first column contains the annotations used to generate each pair, meaning it is either an object (for binary questions) or a relationship (for "what"/"where"-questions).

Note that, in Table 4.1, the negatively-answered binary question "Does the image contain a motorcycle?" is also based on the attribute car, even though this attribute does not occur in the QA-pair. In this case, the template algorithm used WordNet to find "motor vehicle" as the direct hypernym of "car", and then selected "motorcycle" as an alternative hyponym of "motor vehicle". "motorcycle" does not occur in the list of object annotations for the image, so it was used to generated the second binary question.

As mentioned in section 3.2, the template-based method is unable to correctly deal with mass nouns or nouns that only occur in the plural form, generating questions such as "Does the image contain a scissors"? Additionally, there appear to be a few errors in the synsets attached to some relationships annotations. Here, the relationships' synsets are marked as being verbs, while they are in fact prepositions. These mistakes are rare, but often occur in clusters for the same image, suggesting a mistake by the human annotator for a particular image. An example of such an image can be seen in Figure 4.2, and a few wrongly-generated QA-pairs corresponding to this image can be found in in Table 4.2. Note that the machine learning-based verb conjugation method from NLTK still attempts to conjugate the relations, e.g. "at end ofs" and "inises", even though they do not exist.

Figure 4.2: Example image from the Visual Genome dataset, used to generate the QA-pairs in Table 4.1.

Annotation	QA-type	Generated question	Generated answer
underpass AT END OF street	What (verb)	"What does the underpass at end of?'	"The underpass at end ofs a street."
tables ARE SURROUNDED BY chairs	What (verb)	"What does the tables are surrounded by?"	"The tables are surrounded by chairs."
reflection IN window	What (verb)	"What does the reflection in?"	"The reflection inises a window."

Table 4.2: Three examples of generated QA-pairs for the given image. These QA-pairs contain mistakes, as the synsets describing the relationships of the attributes they are based on are wrongly marked as verbs, instead of prepositions. The first column contains the relationship annotations used to generate each pair.

4.2 Effects of QA-enhancement

4.2.1 Baseline model A

First, we take a look at the overall classification accuracy of the classifier baseline model as a function of the dataset used for training. We do this for the default classifier model, which uses the VGG16, and the model that uses ResNet-152 for image feature extraction. The mean accuracy of three models plus the standard error for each model-dataset combination can be found in Table 4.3. For an overview of the four datasets QA0, QA2, QA4, and QA6, see section 3.2.3. Using ResNet-152 instead of VGG16 provides a significant improvement in the classification accuracy. For both variations of the classifier model, extending the Visual Genome set of question-answers has a small, albeit significant, negative effect on the accuracy. The reason that ResNet-152 is not used as the default CNN in both baseline models, is its complexity: using ResNet greatly increases the training time and memory requirements of the model, especially when using the larger datasets and baseline model B.

Baseline model A is tested on several additional conditions: the proportion of times the label answer of each test set QA-pair occurred in the top-10 predictions, the accuracy on "what"-type questions, the accuracy on "where"-type questions and the accuracy on the VQA-real rest set. These results can be seen in Table 4.4. A brief explanation of what the VQA-real dataset is, can be found in the final paragraph of section 3.4.1. We see that for each trained model, the correct answer is in the top-10 predictions at least 85% of the time. In other words: even when the predicted

Dataset	Accuracy (VGG)	Accuracy (ResNet)
QA0	43.29 ± 0.08	44.84 ± 0.03
QA2	42.97 ± 0.03	44.20 ± 0.10
QA4	42.46 ± 0.03	43.55 ± 0.09
QA6	42.13 ± 0.10	43.15 ± 0.07

Table 4.3: Classification accuracy on the test set for the classifier VQA model, for each of the four datasets. The first column represents the default classifier model with VGG16, and the second column the model that uses ResNet-152 instead. Each cell shows the mean accuracy of three trained models, plus the standard error.

Dataset	Top-10	What	Where	VQA-real
QA0	85.69 ± 0.03	40.12 ± 0.07	14.68 ± 0.09	37.82 ± 0.07
QA2	85.62 ± 0.02	38.65 ± 0.02	14.11 ± 0.14	35.96 ± 0.64
QA4	85.15 ± 0.10	37.76 ± 0.07	13.27 ± 0.01	36.14 ± 1.55
QA6	85.28 ± 0.11	37.06 ± 0.08	13.17 ± 0.22	34.40 ± 1.60

Table 4.4: Classification accuracy values across different testing conditions on the test set for the classifier VQA model, for each of the four datasets. Each cell shows the mean accuracy of three trained models, plus the standard error.

answer is wrong, it is usually not too far off of the correct one. The effect of the dataset used during training on the top-10 accuracy is non-significant.

The accuracy on "what"-type questions significantly decreases as the degree of dataset enhancement increases. The same holds for the "where"-type questions, in addition to the accuracy for these QA-pairs being quite low overall.

Finally, we see in the VQA-real column in Table 4.4 that the classification accuracy on the validation set of VQA-real significantly decreases as the degree of dataset enhancement increases.

The BLEU scores, their averages and the METEOR scores for baseline model A can be seen in Table 4.5. For all six measures, the score goes down as the degree of dataset enhancement increases. In every case, this change is significant. While these sentence similarity measures are commonly used for language generation tasks, and not for classification tasks, we include them to be able to directly compare both

Dataset	BLEU1	BLEU2	BLEU3	BLEU4	BLEU (AVG)	METEOR
QA0	48.84 ± 0.09	20.85 ± 0.08	14.43 ± 0.05	11.00 ± 0.03	29.92 ± 0.06	31.42 ± 0.08
QA2	48.66 ± 0.02	20.85 ± 0.02	14.42 ± 0.02	10.98 ± 0.02	29.82 ± 0.02	31.35 ± 0.04
QA4	48.04 ± 0.03	20.43 ± 0.05	14.13 ± 0.04	10.77 ± 0.03	29.41 ± 0.03	30.87 ± 0.06
QA6	47.68 ± 0.11	20.23 ± 0.05	14.00 ± 0.04	10.68 ± 0.02	29.18 ± 0.07	30.66 ± 0.08

Table 4.5: BLEU and METEOR scores on the test set for baseline model A, for each of the four datasets. Each cell shows the mean score of three trained models, plus the standard error.

Dataset	BLEU1	BLEU2	BLEU3	BLEU4	BLEU (AVG)	METEOR
QA0	40.62 ± 0.05	16.91 ± 0.04	11.27 ± 0.02	9.15 ± 0.02	24.89 ± 0.03	22.73 ± 0.04
QA2	39.80 ± 1.08	16.49 ± 0.48	11.02 ± 0.30	8.96 ± 0.24	24.38 ± 0.66	21.73 ± 0.57
QA4	36.74 ± 0.13	15.00 ± 0.07	10.09 ± 0.04	8.24 ± 0.03	22.49 ± 0.08	19.68 ± 0.09
QA6	37.30 ± 0.93	15.26 ± 0.42	10.26 ± 0.26	8.36 ± 0.20	22.83 ± 0.56	20.02 ± 0.53

Table 4.6: BLEU and METEOR scores on the test set for baseline model B, for each of the four datasets. Each cell shows the mean score of three trained models, plus the standard error.

baseline models.

4.2.2 Baseline model B

The BLEU scores, their averages and the METEOR scores for baseline model B can be seen in Table 4.6. With regards to both the BLEU and METEOR scores, the degree of dataset enhancement has a slight but significant negative effect on the performance of baseline model B. The performance of the models trained on QA6 is slightly better than those trained on QA4, but has a high standard error.

4.3 Justification of noise resistance experiments

The experiments described in sections 4.4 and 4.5 were not originally included in this study. However, as can be seen in sections 4.2.1 and 4.2.2, enhancing the Visual Genome dataset with additional QA-pairs using the template-based method

described in section 3.2 appears to slightly decrease the performance of both baseline models. On paper, however, diversifying and extending a dataset with useful additional samples should always have a positive effect on a machine learning model's performance. As seen in [14], a template-based QA-pair augmentation approach is able to increase a model's performance on the VQA-task.

We therefore perform some additional analyses on the different trained models to get a clearer image of the effect of our dataset augmentation method. We argue that dataset enhancement must have some positive effect on a VQA model's robustness, which inspired the noise resistance experiment as described in section 3.4.3. The results can be found in 4.4.

4.4 Noise resistance

4.4.1 Baseline model A

Figure 4.3 shows the effect of applying increasing amounts of random noise to the question embedding. The x-axis shows the proportion of the random noise vector as described in section 3.4.3, the y-axis shows the classification accuracy. Each line shows the average accuracy of three baseline A-models trained on the corresponding dataset.

We can see that all four model-dataset combinations are able to endure a reasonably large amount of noise before their accuracy collapses: up until a noise ratio of 0.1, the accuracy scores remain largely intact and they only start rapidly deteriorating after a noise ratio of 0.2. For noise ratio values of 0.3 up until 0.8, the ranking of the dataset with regards to their models' accuracy is the exact opposite of the one found in Table 4.3. This is the phase where the largest loss of accuracy occurs. At around a noise ratio of 1.0, the curves appears to flatten.

Indeed, the curve suggests that our QA-generation method improves baseline model A's resistance to noise. In order to examine this further, we compare the exact values of the curves at noise ratio 0, 0.5, and 1 (i.e. no noise, half of the noise vector, and the full noise vector). The results can be found in Figure 4.4. The exact loss of accuracy for noise ratios 0.5 and 1, compared to their original value, can be found in Table 4.7.

Note that the bars for a noise ratio of 0 correspond to the original average accuracies as seen in VGG-column in Table 3.4.3. We see that for a noise ratio of 0.5, the models trained on QA6 attain the highest average accuracy, followed by QA4, and then QA2. Here, QA0 performs the worst. When the noise ratio is 1, where the curves become flat, QA0 again performs slightly better than the others, but the average accuracies are otherwise quite similar.

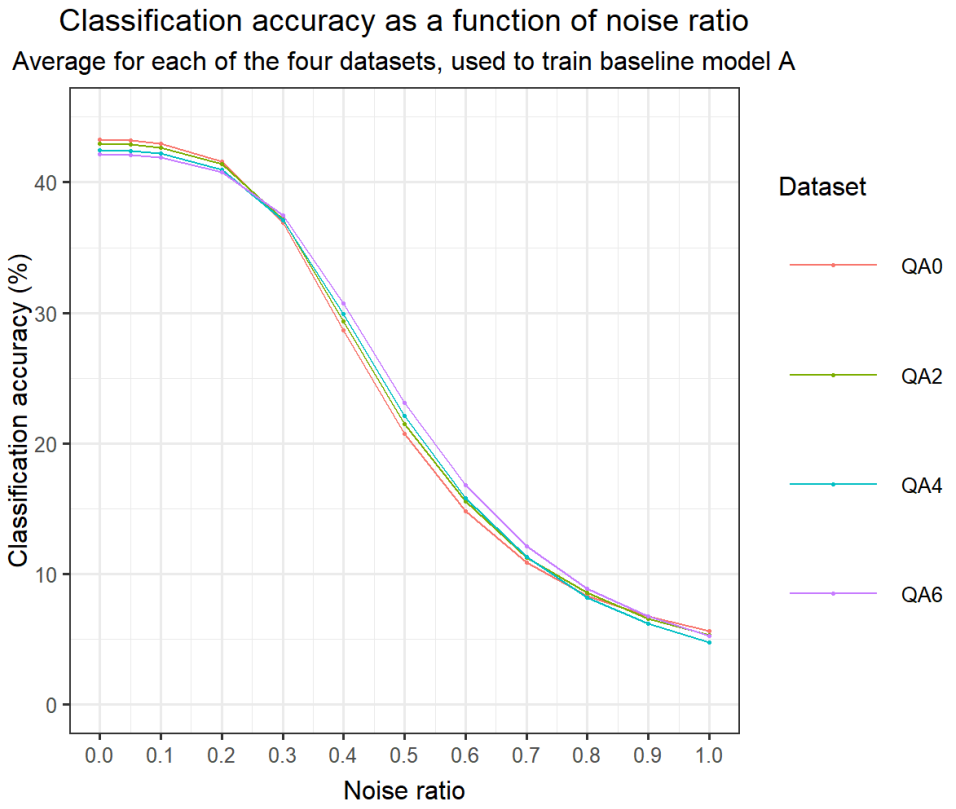


Figure 4.3: Mean classification accuracy as a function of the proportion of noise applied to the question embedding. Each line shows the average accuracy of three baseline A-models trained on the corresponding dataset.

Overall, however, it appears from Table 4.7 that the more new QA-pairs are generated, the lower the loss of accuracy due to noise compared to its original value at the 0.5 noise ratio point. At a noise ratio of 1, the loss in performance is approximately the same for datasets QA0, QA2, and QA4. The performance loss is somewhat less for QA6.

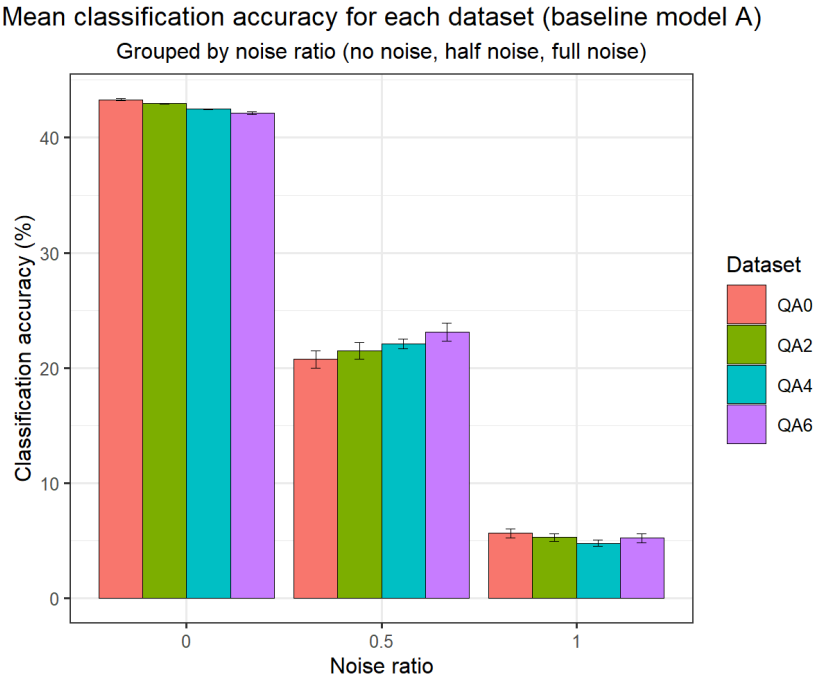


Figure 4.4: Mean classification accuracy per model-dataset combination as a function of the proportion of noise applied to the question embedding, for noise ratio values of 0, 0.5, and 1. The architecture used was baseline model A. The error bars show the standard error.

Dataset	Accuracy diff. (noise ratio 0.5)	Accuracy diff. (noise ratio 1)
QA0	-22.54 $\pm$ 0.81	-37.64 $\pm$ 0.47
QA2	-21.46 $\pm$ 0.69	-37.67 $\pm$ 0.31
QA4	-20.34 $\pm$ 0.42	-37.68 $\pm$ 0.29
QA6	-19.00 $\pm$ 0.68	-36.90 $\pm$ 0.29

Table 4.7: The mean difference in accuracy due to noise for each dataset used to train baseline model A, compared to the original mean accuracy, for noise ratios 0.5 and 1. The standard error is shown as well.

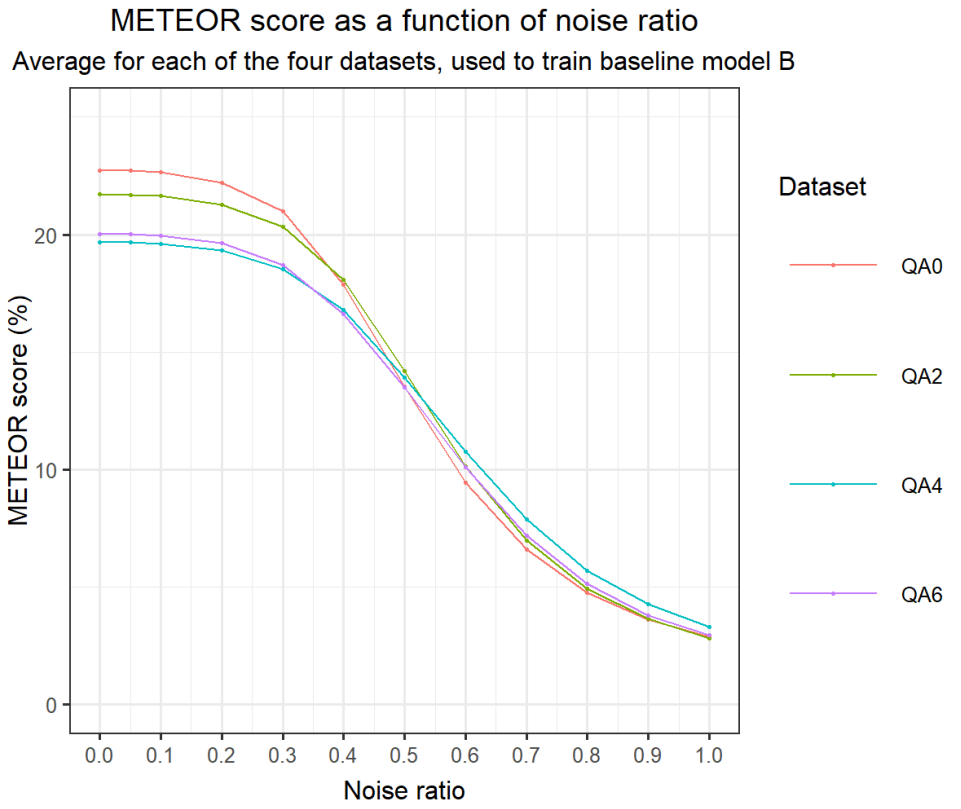


Figure 4.5: Mean METEOR score as a function of the proportion of noise applied to the question embedding. Each line shows the average accuracy of three baseline B-models trained on the corresponding dataset.

4.4.2 Baseline model B

We repeat the noise experiments of the previous section (4.4.1) for baseline model B. The effect of applying increasing amounts of random noise to the question embedding can be seen in 4.5. The x-axis shows the proportion of the random noise vector as described in section 3.4.3, the y-axis shows the METEOR score. Each line shows the average accuracy of three baseline B-models trained on the corresponding dataset. Here, the models trained on QA0 perform worse on average than the others for noise ratios higher than 0.5. Overall, the ranking of the model-dataset combinations fluctuates as the noise ratio increases. Again, in order to get a clearer image, we take a look at the performances for noise ratios 0, 0.5, and 1 in Figure

4.6. The precise loss in METEOR score for noise ratios 0.5 and 1, compared to their original value, can be found in Table 4.8.

At noise ratio 0.5, models trained on QA2 perform best on average, followed by QA4, and then a tie between QA0 and QA6. For all model-dataset combinations but QA4, the standard errors are quite high and overlap each other. For noise ratio 1, the METEOR scores for QA0, QA2, and QA6 are similar with overlapping error bars. The mean METEOR score at this noise ratio of models trained on QA4, outperforms the others with a low standard error.

In Table 4.8, we again see that the performance loss is less for the enhanced datasets, at both noise ratio 0.5 and 1. This is especially visible for QA4 and QA6.

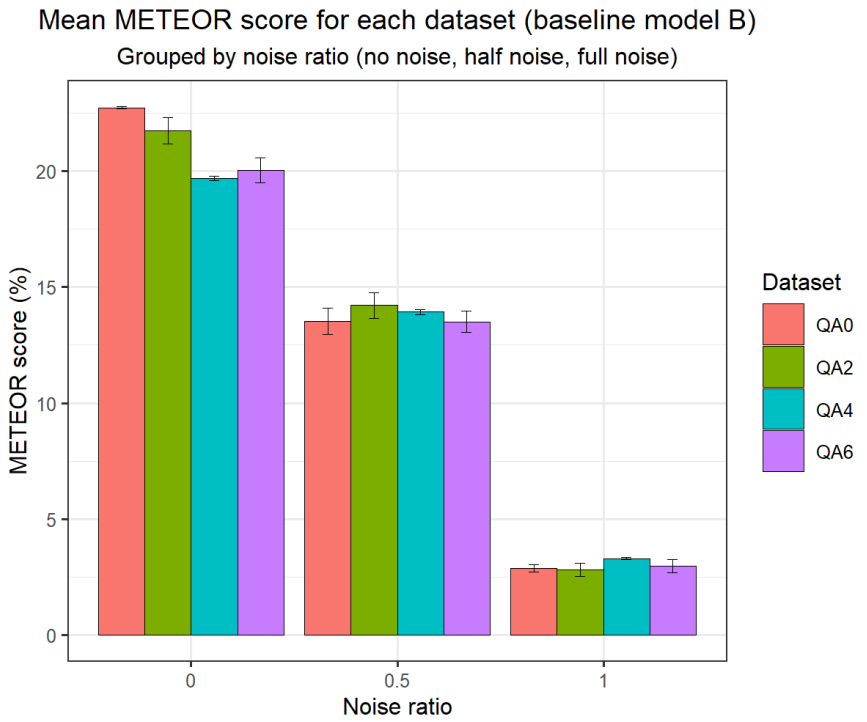


Figure 4.6: Mean METEOR score per model-dataset combination as a function of the proportion of noise applied to the question embedding, for noise ratio values of 0, 0.5, and 1. The architecture used was baseline model B. The error bars show the standard error.

Dataset	METEOR diff. (noise ratio 0.5)	METEOR diff. (noise ratio 1)
QA0	-9.21 $\pm$ 0.58	-19.85 $\pm$ 0.19
QA2	-7.52 $\pm$ 0.92	-18.91 $\pm$ 0.72
QA4	-5.76 $\pm$ 0.07	-16.38 $\pm$ 0.05
QA6	-6.52 $\pm$ 1.00	-17.05 $\pm$ 0.79

Table 4.8: The mean difference in METEOR score due to noise for each dataset used to train baseline model B, compared to the original mean METEOR score, for noise ratios 0.5 and 1. The standard error is shown as well.

4.5 Attention maps

A large-scale quantitative analysis of the stacked attention module is beyond the scope of this study. In this section, however, we will take a qualitative look at the attention maps produced by the different model-dataset combinations, for the same image and question. The attention maps are visualized as heat maps, with blue indicating low importance and red indicating high importance. These results can be found in 4.7. In each row, the left image shows the original visual input, the center image the attention map of the first glimpse, and the right image the attention map of the second glimpse. The text below each set of three images shows the input question and the answer produced by the model. The model and the dataset it was trained on is shown in the subcaptions.

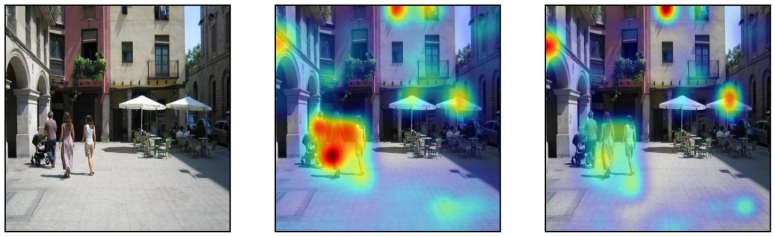
(a) Baseline model A, trained on QA0.

(b) Baseline model A, trained on QA2.

(c) Baseline model A, trained on QA4.

(d) Baseline model A, trained on QA6.

Figure 4.7: Example attention heat maps of both glimpses produced by baseline models A and B, trained on the four different QA-sets. The input question and the answer produced by the model are given for each image set. **Continued on next page.**



(e) Baseline model B, trained on QA0.

(f) Baseline model B, trained on QA2.

(g) Baseline model B, trained on QA4.

(h) Baseline model B, trained on QA6.

Figure 4.7: Example attention heat maps of both glimpses produced by baseline models A and B, trained on the four different QA-sets. The input question and the answer produced by the model are given for each image set. **Continued from previous page.**

Chapter 5

Discussion

This thesis proposes a template-based method for automatically extending question-answer pairs of the Visual Genome visual question answering dataset with new samples, using its rich annotations of objects and their relations in each image. The goal of this research is to look at the effect of this question-answer data augmentation method on a VQA system's performance and overall robustness to variations in the question input. The template-based approach was used to create three new datasets (referred to in this study as "QA2", "QA4" and "QA6") from the original Visual Genome set of question answers ("QA0"). The effects of this data augmentation approach were studied by training two separate architectures, named "baseline model A" and "baseline model B", on several datasets with different amounts of newly-generated QA-pairs. The former model treats the VQA-task as a 1-out-of-N classification problem, while the latter treats it as a sequence generation problem where an answer is constructed word-by-word. The findings of Chapter 4 are summarized and interpreted in section 5.1. Suggestions for future research are made in section 5.2.

5.1 Conclusions

5.1.1 Method of question generation

The template-based approach is able to generate well-formulated natural language sentences in the English language, as can be seen in Table 4.1. Qualitative exploration of the generated QA-pairs show that most are free from spelling and grammatical errors. Some errors do occur due to mistakes in the relationship annotations, as can be seen in Table 4.2. Another type of error occurs when generating QA-pairs about mass nouns or words that only occur in the plural form, such as "water" or "scissors". Interestingly enough, while the algorithmically generated QA-pairs are mostly free of spelling and grammatical errors, their human-annotated counterparts often do contain these types of mistakes. Additionally, the QA-pairs generated by humans are often formulated quite curtly, while the template-based generation

method produces fuller sentences. The template-based method is lightweight compared to a machine learning-based approach, as it requires no prior training.

Originally, counting-type questions were implemented as well. The templates started with (a variation on) "How many $\langle NOUN \rangle$ are in the image?" and used the list of objects for the relevant image to generate the correct answer. As opposed to the VQA-real dataset, however, Visual Genome does not guarantee that the object annotations are complete and/or free from duplicates. Therefore, not all objects present in an image are named and the same object may occur multiple times in the objects lists. This meant that the counting-type template was unreliable and therefore omitted from the question generation process.

5.1.2 Effects of question-answer data enhancement

We expected to see a positive effect of the amount of generated QA-pairs added to the Visual Genome dataset on the accuracy of both baseline models. This was not the case, unfortunately. Table 4.3 shows that the best classification accuracy is attained by the models trained on QA0, the original dataset, with the accuracy declining for each subsequent dataset (QA2, QA4, and QA6). For all model-dataset combinations, the performance improves when ResNet-152 instead of VGG16 is used for image feature extraction.

Table 4.4 shows that all model-dataset combination achieve a high performance of around 85%, meaning that the correct answer is in the top-10 predictions at least 85% of the time. There is no significant effect of the dataset used to train each model on this score, which makes sense as this performance measure is quite lenient: even if two models (slightly) differ on the top-1 accuracy, it's not surprising that they score equally well on the top-10 accuracy.

Table 4.4 shows that the accuracy on "what" and "where"-type questions significantly decreases as the degree of dataset enhancement increases. Additionally, the accuracy is much lower for "where"-questions than for "what" questions. A reason for this could be that "where"-questions are inherently more ambiguous, as there are always multiple ways to describe an objects location in an image. An object's location can always, for example, either be given in relation to some other object in the image (e.g. "to the left of the chair") or in relation to the image as a whole (e.g. "on the right"). Additionally, "where"-type QA-pairs are quite sparse to begin with, making up only 2.9% of Visual Genome's (original) QA-pairs [16].

Furthermore, we see in the VQA-real column in Table 4.4 that the classification accuracy on the validation set of VQA-real significantly decreases as the degree of dataset enhancement increases. All models attain a reasonable accuracy on the VQA-real dataset compared to their accuracy on the Visual Genome dataset, con-

sidering VQA-real consists of unseen data. A comparison to other studies based on the VQA-real accuracy is made in section 5.1.5.

The performance of baseline model B also declines as the degree of QA-pair augmentation increases in the dataset. This effect is visible in the BLEU scores for n-grams with $n \in \{1, 2, 3, 4\}$, their average and the METEOR score, as can be seen in Table 4.6. While the scores are lower than those of baseline model A, baseline model B is overall able to generate answers reasonably well. Note that, while they are among the *de facto* standard measure for the quality of machine-generated sentences, both BLEU and METEOR are not ideal. Both achieve a good correlation with human judgment at the corpus level (0.817 and 0.964, respectively), but even METEOR only achieves a human judgment correlation of 0.403 at the sentence level. Furthermore, BLEU for longer n-grams and METEOR heavily penalize one-word sentences. At the same time, a lot of answers in VQA datasets tend to be one word in length: even Visual Genome, which has longer answers on average, only has a mean answer length of 1.9 words [16].

Based on this testing method, it is hard to conclude whether our method of dataset augmentation improves the models' performance on unseen data. Originally, since performance in real-world situations is a central point of this study, we intended to employ human participants to test our different trained models. This idea was scrapped due to time constraints and all models were tested on holdout set from the original Visual Genome question-answer pairs instead. However, this method of testing is quite strict, as the test dataset contains none of the template-generated QA-pairs, meaning the different trained models were compared on a playing field that is in favor of models that were trained on this exact dataset.

5.1.3 Robustness to noise

Since there was no positive effect of our QA-data augmentation method on the performance of both baseline models with regards to the holdout test set, we conducted additional noise resilience experiments on baseline model A and B as described in section 3.4.3. Tables 4.7 and 4.8 suggest that for both baseline models, the loss in performance due to noise is lower for the models trained on the augmented datasets, compared to models trained on the original dataset. Overall, all trained models are quite robust to noise, as their performance remains largely intact for up to a noise ratio of 0.2. For baseline model A, during the interval of noise ratio values where the largest deterioration in performance occurs, the models trained on augmented datasets even lead to better performances than those trained on the original dataset, even if models trained the original dataset perform better when no noise is applied. The above suggests that our method of dataset enhancement increases the degree

of noise resistance for both baseline models, and are therefore more robust. This implies that models trained on datasets that have been augmented by our method of template-based enhancement, are better equipped to deal with the uncertainty of the real world.

5.1.4 Attention heat maps

For both baseline models, we looked at the attention map generated for an instance of each model-dataset combination, for the same image and QA-pair from the Visual Genome dataset. As can be seen in 4.7, all model-dataset combinations are able to answer the question correctly (Q: "What color shirt is the woman wearing?", A: "White."), except for model A trained on QA4 (4.7c), where the answer is "Black". The attention is generally focused around the two women in the left of the picture, around the part most relevant for answering the question (the robe of the woman of the left, the shirt of the woman to the right). Baseline model B trained on QA4 (4.7g) is the exception here, as the attention appears to be quite random. All attention maps appear to contain some anomalies, where non-relevant regions are deemed important by the attention mechanism as well. In none of these examples, the benefits of using stacked attention are exploited. Ideally, the first glimpse would focus on both women and the second glimpse on the region where the right woman's shirt is present.

5.1.5 Comparison to other work

For this study, a relatively simple but robust architecture was chosen based on the model described in [11], the only difference being that this study used the VGG for the image input as opposed to ResNet, bar for one experiment. We tested baseline model A on the validation set of the 2.0-version of the VQA-real dataset, allowing for a direct comparison with [11], who did the same with their architecture. Their VQA-model, both trained and tested on the VQA-real dataset, achieved an accuracy of 59.67%, which at the time (early 2017) was 0.5% higher than the previously best reported result. Our best dataset-model combination, baseline A trained on QA0, achieved an accuracy of 37.82% on the VQA-real validation set, which is much lower.

It appears that Visual Genome is used far less often than VQA-real. When it is used, it is commonly to extend the VQA-real dataset, as they have some images in common. It is hard to find studies that trained a model solely on VG, making it difficult to directly compare our models with regards to their performance on Visual Genome alone. A study from 2019 that does only use VG, is [71]. Their

best architecture achieves a validation accuracy of 47.60%, which is much closer to the classification accuracy achieved by the models in this study (see Table 4.3). They use a baseline model that is able to achieve a test accuracy of 70.34% on VQA-real. However, when the authors of [71], train the same model on Visual Genome, they achieve a test accuracy of 44.68%, which is similar to what happened when the model from [11] was adapted for this study. A reason for this apparent discrepancy could be that Visual Genome's (much) larger and more varied body of questions and answers make it more difficult for a model to learn the VQA-task from the data. If this is the case, it could be that attaining a good performance on the VG dataset is more indicative of a VQA-model's robustness and generalization capability than its performance on the VQA-real dataset.

5.1.6 Contributions to AI

The positive results of the noise experiments can be considered a useful addition to the field of VQA, where many studies are focused on achieving small performance gains compared to the current state-of-the-art without regard to actual performance in the real world. This seems to forgo the original goal of the introduction of VQA, which was to get closer to a form of general artificial intelligence. Such an intelligence should be able to combine multiple modalities of reasoning in order to function in a complex and chaotic world. The results of the noise experiments suggest that our method of dataset augmentation is able to increase a VQA model's ability to deal with the complexity and nuances of human language. In general, performing experiments on the robustness of trained models might be a good practice to adopt when developing VQA systems, as it provides an indication of real-world performance and penalizes the risk of overfitting due to training and testing in a contrived environment.

5.2 Future research

This research originally planned to test the trained models on human participants. A study where humans are asked to query the trained models with their own images and questions is, of course, the most direct way of testing how VQA model behaves in a real-life situation. We argue that, while the state-of-the-art focuses on marginally increasing model performance on curated datasets, it would be much more interesting to evaluate VQA pipelines on how well they perform in the real world. We therefore recommend conducting a VQA-study that incorporates testing on human participants.

As mentioned in section 5.1.1, it appears that it is more difficult to attain a good performance on the Visual Genome dataset on VQA-real. It could be the case that performance on Visual Genome is more indicative of real-world performance than VQA-real. However, one must also account for the possibility that the data in Visual Genome is, for some reason, of lesser quality and/or less suitable for learning the VQA-task than the data in VQA-real. We therefore suggest further examining the possible reasons for this discrepancy, especially since studies that train VQA-models on the VG dataset only are quite rare. The noise resilience experiments as decribed in section 3.4.3 could be quite useful in such a study. Additionally, these experiments could be used to study whether models trained on VG are more robust to variations in the question input than those trained on VQA-real, as suggested in Chapter 1.

On the other hand, VQA-real has some clear advantages over VG: besides the fact that VQA-real contains twice as many images, its image object annotations are guaranteed to be complete and free from duplicates [8]. In Visual Genome, the same object may be annotated twice, or some objects present on an image may remain unannotated, which makes data augmentation based on image annotations much more difficult and prone to mistakes. It might therefore be worthwhile to construct a new VQA dataset that combines the large body of images and annotation quality of VQA-real with the large variety of question-answer pairs and annotation quantity of Visual Genome.

Even though the object annotations in Visual Genome are not guaranteed to be complete and free from duplicates, there are still additional templates that could be constructed. Attribute annotations, which are of the form "$\langle NOUN \rangle$ is $\langle ADJ \rangle$" (with "$\langle ADJ \rangle$" referring to an adjective), could be used to construct questions that query the state and properties of objects in the image. "Where"-type questions based on the pixel coordinates of objects in the images could be constructed as well, which would specifcally query the spatial location of objects with regards to other objects or the overall image.

Section 3.3.1 mentions that, while computer vision-related tasks often use augmentation on the training images, doing so for VQA is non-trivial as operations such as rotation and mirroring may change the correct answer for a particular question. However, it might be interesting to study the effect of image data augmentation based on transformations that preserve the spatial relationships between elements in an image. An example of such a method is the random elastic morphing used in [72], which is analogous to projecting the images on a rubber sheet and applying random non-uniform local distortions. Performing such augmentations on the image input might further improve a VQA system's performance and robustness.

The de facto standard in VQA is to extract image features using convolutional neural networks that have been pretrained on the ImageNet dataset, meaning that

its internal parameters are frozen and remain constant while training the VQA system [3,9]. While doing so greatly reduces a VQA pipeline's training time, it may still be worthwhile for future studies to see whether training the CNN feature extractor from scratch benefits the performance of a particular VQA approach, as opposed to blindly using a pretrained version.

As mentioned in section 2.3.1, there are several advantages to using a convolutional neural network instead of a recurrent neural network for question feature extraction, while relatively few studies on VQA have explored this method so far. This study originally intended to construct an additional baseline model that would use a CNN for the question inputs, but this did not happen due to time constraints. Still, we suggest further exploring using CNN not only for parsing the question input, but for generating answers token-by-token as well.

Finally, we recommend exploring additional methods for embedding language data in the VQA task. Input questions are often represented by either (pretrained) Word2Vec or GloVe word vectors [3]. There are, however, some interesting alternatives that could be utilized. A powerful language representation model that has rapidly gained in popularity in the last year is the *Bidirectional Encoder Representations from Transformers*, or BERT [73]. BERT promises a conceptually simple but powerful language representation that can be easily adapted to a task such as (visual) question answering, which makes it an interesting technology for VQA. Another promising word embedding approach is the *pyramidal histogram of characters* (PHOC), proposed in [74]. Instead of extracting semantic representations of words from a training set, PHOC encodes words by constructing histograms of characters occurring in that word at several levels, where each level consists of substrings of the string at the level above. This allows for useful encoding of previously unencountered words. Additionally, it increases robustness to spelling errors in the input, as the PHOC-encoding of a misspelled word is by definition close to the encoding of its correct form. Another encoding method that allows for embedding out-of-vocabulary words is Byte-Pair Encoding (BPE), which finds a middle ground between encoding tokens at the character and the word level [75]. BPE uses language corpora to learn which combinations of characters frequently occur together and saves these frequent character combinations as "subwords". New words are then encoded as a series of these learned subwords.

Bibliography

[1] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh, "VQA: Visual question answering," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015 Inter, pp. 2425–2433, 2015.

[2] Q. Wu, P. Wang, C. Shen, A. Dick, and A. van den Hengel, "Ask me anything: Free-form visual question answering based on knowledge from external sources," 2015. [Online]. Available: http://arxiv.org/abs/1511.06973

[3] Q. Wu, D. Teney, P. Wang, C. Shen, A. Dick, and A. van den Hengel, "Visual question answering: A survey of methods and datasets," *Computer Vision and Image Understanding*, vol. 163, pp. 21–40, 2017.

[4] I. Ilievski and J. Feng, "Multimodal learning and reasoning for visual question answering," *Nips'17*, no. Nips, pp. 551–562, 2017. [Online]. Available: https://papers.nips.cc/paper/6658-multimodal-learning-and-reasoning-for-visual-question-answering

[5] J. P. Bigham, C. Jayant, H. Ji, G. Little, A. Miller, R. C. Miller, R. Miller, A. Tatarowicz, B. White, S. White, and T. Yeh, "VizWiz: Nearly real-time answers to visual questions," in *In Proceedings of the 23nd annual ACM symposium on User interface software and technology*, 2010, pp. 333–342. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1866029.1866080

[6] K. Kafle and C. Kanan, "Visual question answering: Datasets, algorithms, and future challenges," *Computer Vision and Image Understanding*, 10 2016.

[7] B. He, M. Xia, X. Yu, P. Jian, H. Meng, and Z. Chen, "An educational robot system of visual question answering for preschoolers," in *2017 2nd Inter-*

national Conference on Robotics and Automation Engineering, ICRAE 2017, vol. 2017-Decem, 2018, pp. 441–445.

[8] K. Kafle and C. Kanan, "An analysis of visual question answering algorithms," *CoRR*, vol. abs/1703.09684, 2017. [Online]. Available: http://arxiv.org/abs/1703.09684

[9] S. Manmadhan and B. C. Kovoor, *Visual question answering: a state-of-the-art review*. Springer Netherlands, 2020, no. 0123456789. [Online]. Available: https://doi.org/10.1007/s10462-020-09832-7

[10] Z. Yang, X. He, J. Gao, L. Deng, and A. J. Smola, "Stacked attention networks for image question answering," *CoRR*, vol. abs/1511.02274, 2015. [Online]. Available: http://arxiv.org/abs/1511.02274

[11] V. Kazemi and A. Elqursh, "Show, ask, attend, and answer: A strong baseline for visual question answering," 2017.

[12] M. Ren, R. Kiros, and R. S. Zemel, "Image question answering: A visual semantic embedding model and a new dataset," *CoRR*, vol. abs/1505.02074, 2015. [Online]. Available: http://arxiv.org/abs/1505.02074

[13] M. Heilman, "Automatic factual question generation from text," Ph.D. dissertation, USA, 2011.

[14] K. Kafle, M. Yousefhussien, and C. Kanan, "Data augmentation for visual question answering," in *Proceedings of the 10th International Conference on Natural Language Generation*. Santiago de Compostella, Spain: Association for Computational Linguistics, Sep. 2017, pp. 198–202. [Online]. Available: https://www.aclweb.org/anthology/W17-3529

[15] Z. Fan, Z. Wei, P. Li, Y. Lan, and X. Huang, "A question type driven framework to diversify visual question generation," *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2018-July, pp. 4048–4054, 2018.

[16] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L. J. Li, D. A. Shamma, M. S. Bernstein, and L. Fei-Fei, "Visual genome: Connecting language and vision using crowdsourced dense image annotations," *International Journal of Computer Vision*, vol. 123, no. 1, pp. 32–73, 2017.

[17] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[18] Y. Ioannou, "Structural priors in deep neural networks," Ph.D. dissertation, 09 2017.

[19] W. Guo and H. Xue, "Crop yield forecasting using artificial neural networks: A comparison between spatial and temporal models," *Mathematical Problems in Engineering*, vol. 2014, pp. 1–7, 01 2014.

[20] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359 – 366, 1989. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0893608089900208

[21] M. T. Tarabay, *Comparison of the sigmoid, tanh and ReLU activation functions*, May 2019. [Online]. Available: http://rafietarabay.blogspot.com/2019/05/pytorch-and-deep-learning.html

[22] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, no. 3, pp. 400–407, 09 1951. [Online]. Available: https://doi.org/10.1214/aoms/1177729586

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Internal Representations by Error Propagation*. Cambridge, MA, USA: MIT Press, 1986, p. 318–362.

[24] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[25] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," 05 2010, pp. 253–256.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012.

[27] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1746–1751. [Online]. Available: https://www.aclweb.org/anthology/D14-1181

[28] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. Muller, "Deep learning for time series classification: a review," *CoRR*, vol. abs/1809.04356, 2018. [Online]. Available: http://arxiv.org/abs/1809.04356

[29] A. H. Reynolds, "Convolutional neural networks (CNNs)," Oct 2017. [Online]. Available: https://anhreynolds.com/blogs/cnn.html

[30] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[32] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179 – 211, 1990. [Online]. Available: http://www.sciencedirect.com/science/article/pii/036402139090002E

[33] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[34] "File:long short-term memory.svg." [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=60149410

[35] S. Abeywardana, "Sequence to sequence tutorial," Jun 2018. [Online]. Available: https://towardsdatascience.com/sequence-to-sequence-tutorial-4fde3ee798d8

[36] B. Shogry, "Making sense of messy bank data," Nov 2018. [Online]. Available: https://blog.plaid.com/making-sense-of-messy-data/

[37] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. [Online]. Available: https://www.aclweb.org/anthology/D14-1162

[38] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller, "Introduction to WordNet: An On-line Lexical Database*," *International Journal of Lexicography*, vol. 3, no. 4, pp. 235–244, 12 1990. [Online]. Available: https://doi.org/10.1093/ijl/3.4.235

[39] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, 1st ed. O'Reilly Media, Inc., 2009.

[40] K. Papineni, S. Roukos, T. Ward, and W. J. Zhu, "BLEU: a method for automatic evaluation of machine translation," 10 2002.

[41] F. Tyers, "File:meteor-alignment-a.png," Nov 2006. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=1356114

[42] S. Banerjee and A. Lavie, "METEOR: An automatic metric for MT evaluation with improved correlation with human judgments," in *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. Ann Arbor, Michigan: Association for Computational Linguistics, June 2005, pp. 65–72. [Online]. Available: https://www.aclweb.org/anthology/W05-0909

[43] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," *CoRR*, vol. abs/1411.4555, 2014. [Online]. Available: http://arxiv.org/abs/1411.4555

[44] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[45] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," pp. 1–14, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[46] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[47] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[48] T. Mikolov, G. Corrado, K. Chen, and J. Dean, "Efficient estimation of word representations in vector space," 01 2013, pp. 1–12.

[49] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[50] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," no. Nips, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[51] M. Elbayad, L. Besacier, and J. Verbeek, "Pervasive attention: 2D convolutional neural networks for sequence-to-sequence prediction," 2018. [Online]. Available: http://arxiv.org/abs/1808.03867

[52] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," 2018. [Online]. Available: http://arxiv.org/abs/1803.01271

[53] L. Ma, Z. Lu, and H. Li, "Learning to answer questions from image using convolutional neural network," 2015. [Online]. Available: http://arxiv.org/abs/1506.00333

[54] Z. Wang and S. Ji, "Learning convolutional text representations for visual question answering," 2017. [Online]. Available: http://arxiv.org/abs/1705.06824

[55] K. Saito, A. Shin, Y. Ushiku, and T. Harada, "Dualnet: Domain-invariant network for visual question answering," *CoRR*, vol. abs/1606.06108, 2016. [Online]. Available: http://arxiv.org/abs/1606.06108

[56] A. Fukui, D. H. Park, D. Yang, A. Rohrbach, T. Darrell, and M. Rohrbach, "Multimodal compact bilinear pooling for visual question answering and visual grounding," 2016. [Online]. Available: http://arxiv.org/abs/1606.01847

[57] H. Ben-younes, R. Cadène, M. Cord, and N. Thome, "MUTAN: multimodal tucker fusion for visual question answering," *CoRR*, vol. abs/1705.06676, 2017. [Online]. Available: http://arxiv.org/abs/1705.06676

[58] J. Kim, S. Lee, D. Kwak, M. Heo, J. Kim, J. Ha, and B. Zhang, "Multimodal residual learning for visual QA," *CoRR*, vol. abs/1606.01455, 2016. [Online]. Available: http://arxiv.org/abs/1606.01455

[59] H. Gao, J. Mao, J. Zhou, Z. Huang, L. Wang, and W. Xu, "Are you talking to a machine? dataset and methods for multilingual image question answering," *CoRR*, vol. abs/1505.05612, 2015. [Online]. Available: http://arxiv.org/abs/1505.05612

[60] R. Rensink, "The dynamic representation of scenes," *Visual Cognition*, vol. 7, pp. 17–42, 01 2000.

[61] K. J. Shih, S. Singh, and D. Hoiem, "Where to look: Focus regions for visual question answering," *CoRR*, vol. abs/1511.07394, 2015. [Online]. Available: http://arxiv.org/abs/1511.07394

[62] I. Ilievski, S. Yan, and J. Feng, "A focused dynamic attention model for visual question answering," *CoRR*, vol. abs/1604.01485, 2016. [Online]. Available: http://arxiv.org/abs/1604.01485

[63] C. Zitnick and P. Dollar, "Edge boxes : Locating object proposals from edges," vol. 8693, 09 2014.

[64] A. Das, H. Agrawal, C. L. Zitnick, D. Parikh, and D. Batra, "Human attention in visual question answering: Do humans and deep networks look at the same regions?" in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

[65] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein, "Deep compositional question answering with neural module networks," *CoRR*, vol. abs/1511.02799, 2015. [Online]. Available: http://arxiv.org/abs/1511.02799

[66] A. Kumar, O. Irsoy, J. Su, J. Bradbury, R. English, B. Pierce, P. Ondruska, I. Gulrajani, and R. Socher, "Ask me anything: Dynamic memory networks for natural language processing," *CoRR*, vol. abs/1506.07285, 2015. [Online]. Available: http://arxiv.org/abs/1506.07285

[67] C. Xiong, S. Merity, and R. Socher, "Dynamic memory networks for visual and textual question answering," *CoRR*, vol. abs/1603.01417, 2016. [Online]. Available: http://arxiv.org/abs/1603.01417

[68] E. Loper and S. Bird, "NLTK: The natural language toolkit," in *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*, 2002.

[69] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 249–256, 01 2010.

[70] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information*

Processing Systems 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. dÁlché Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[71] H. Kim and M. Bansal, "Improving visual question answering by referring to generated paragraph captions," *CoRR*, vol. abs/1906.06216, 2019. [Online]. Available: http://arxiv.org/abs/1906.06216

[72] M. Bulacu, A. Brink, T. v. d. Zant, and L. Schomaker, "Recognition of handwritten numerical fields in a large single-writer historical collection," in *2009 10th International Conference on Document Analysis and Recognition*, 2009, pp. 808–812.

[73] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[74] J. Almazán, A. Gordo, A. Fornés, and E. Valveny, "Word spotting and recognition with embedded attributes," in *TPAMI*, 2014.

[75] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725. [Online]. Available: https://www.aclweb.org/anthology/P16-1162