Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Visual Question Answering 2.0

Degree's Thesis
Audiovisual Systems Engineering

Author: Francisco Roldán Sánchez
Advisors: Xavier Giró-i-Nieto, Santiago Pascual de la Puente, Issey Masuda Mora

Universitat Politècnica de Catalunya (UPC)
2016 - 2017

Abstract

This bachelor's thesis explores different deep learning techniques to solve the Visual Question-Answering (VQA) task, whose aim is to answer questions about images. We study different Convolutional Neural Networks (CNN) to extract the visual representation from images: Kernelized-CNN (KCNN), VGG-16 and Residual Networks (ResNet). We also analyze the impact of using pre-computed word embeddings trained in large datasets (GloVe embeddings). Moreover, we examine different techniques of joining representations from different modalities. This work has been submitted to the second edition Visual Question Answering Challenge, and obtained a 43.48% of accuracy.

Resum

Aquest treball de fi de grau explora diferents tècniques d'aprenentatge profund (deep learning) per a solucionar la tasca de Respostes a Preguntes Visual (Visual Question-Answering), que té com a finalitat respondre preguntes sobre imatges. Estudiem differents xarxes convolucionals (CNN - *Convolutional Neural Networks*) per extreure la representació visual de les images: Kernelized-CNN (KCNN), VGG-16 i Residual Networks (ResNet). També analitzem l'impacte d'utilitzar *embeddings* pre-calculats que han estat entrenats amb bases de dades més grans (GloVe *embeddings*). També examinem diferents tècniques per a combinar vectors de dades de diferents modalitats. Aquesta feina ha estat presentada a la segona edició del Visual Question Answering Challenge i ha obtingut un 43.48% d'exactitud.

Resumen

Esta tesis explora diferentes técnicas de aprendizaje profundo (deep learning) para solucionar la tarea de Respuestas a Preguntas Visuales , que tiene como finalidad responder preguntas sobre imágenes.

Estudiamos diferentes redes convolucionales (CNN - *Convolutional Neural Networks*) para extraer la representación visual de las imágenes: Kernelized-CNN (KCNN), VGG-16 y Residual Networks (ResNet). También analizamos el impacto de utilizar *embeddings* precomputados que han sido entrenados en bases de datos más grandes (GloVe *embeddings*). Asimismo, examinamos diferentes técnicas para combinar vectores de datos de diferentes modalidades. Este trabajo ha sido presentado a la segunda edición del Visual Question Answering Challenge y ha obtenido un 43.48% de exactitud.

Acknowledgements

In the first place, I want to thank my tutor, Xavier Giro i Nieto, for all his help and efforts during the whole project, as well as for introducing me in deep learning field. Likewise, I also want to thank Santiago Pascual de la Puente and Issey Masuda Mora for all the time they spent in giving me advices to improve my work and helping me with technical issues.

I can't forget of my partners from the X-Thesis group, who gave me plenty of advices in our weekly meetings. and shared with me their gained knowledge during their research.

I would also want to thank Kevin McGuiness, from the Insight Centre for Data Analytics, for his suggestions in some of the experiments we made.

I would like to mention as well Albert Gil for his assistance in the usage of the clusters of servers of the GPI.

Revision history and approval record

Revision	Date	Purpose
0	26/06/2017	Document creation
1	28/06/2017	Document revision
2	29/06/2017	Document revision
3	30/06/2017	Document approbation

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Francisco Roldán Sánchez	frol14993@alu-etsetb.upc.edu
Xavier Giró i Nieto	xavier.giro@upc.edu
Santiago Pascual de la Puente	santiago.pascual@tsc.upc.edu
Issey Masuda Mora	

Written by:		Reviewed and approved by:		Reviewed and approved by:	
Date	26/06/2017	**Date**	30/06/2017	**Date**	30/06/2017
Name	Francisco Roldán	**Name**	Xavier Giró i Nieto	**Name**	Santiago Pascual
Position	Project Author	**Position**	Project Supervisor	**Position**	Project Supervisor

Contents

List of Figures

List of Tables

Chapter 1

Introduction

.

1.1 Statement of purpose

Human communication is based on the exchange of information between an emitter and a receiver. Through questions and answers, we humans can exchange and increase our knowledge. For this reason, the Artificial Intelligence (AI) community has focused a great attention on the creation of automatic systems capable of communicating through questions and answers.

Over the last five years, the number of research projects related to Deep Learning has increased exponentially, both in the academic and industrial worlds. Such growth has been boosted by the success of deep learning models in tasks which were considered especially challenging, such as computer Vision or natural language processing. Therefore, solutions have been found to many tasks obtaining good performances, leading to more complex tasks derived from these ones.

Since 2016, Virginia Tech organizes an annual competition combining the visual and textual modalities: the Visual Question Answering (VQA) challenge. This benchmark deals with the creation of an AI system able to answer natural language questions related to an image. Participating models require a step forward in terms of reasoning and comprehension of the visual scene in order to relate the content it to its natural language representation [3].

The most common solution to solve this task is using Convolutional Neural Network (CNNs) to obtain the visual representation, and word embeddings from Recurrent Neural networks (RNNs) (for instance Long Short-Term Memory Networks -LSTM- [4]) to obtain the textual representation. These two type of representations are combined to predict the answer [5] (Figure 1.1). The Image Processing Group (GPI) at the Universitat Politecnica participated in the 2016 edition as a result of the bachelor thesis by Issey Masuda [6], a first experience at our university in the fusion of textual and visual information with deep learning.

Based on this context, the objectives of this project were the following:

- Participate in the VQA Challenge 2017 organized by VirginaTech (deadline: 30 June 2017).
- Explore different ideas with the latest deep learning designs, considering the previous experience at UPC [6] as the baseline.
- Improve the model presented by the UPC team in Edition 2016 [6], which provides a 40.28% of accuracy in the new VQA 2.0 Dataset [7] used this year in the challenge (Chapter 4 for more details).
- Develop a reusable software project using programming good practices.

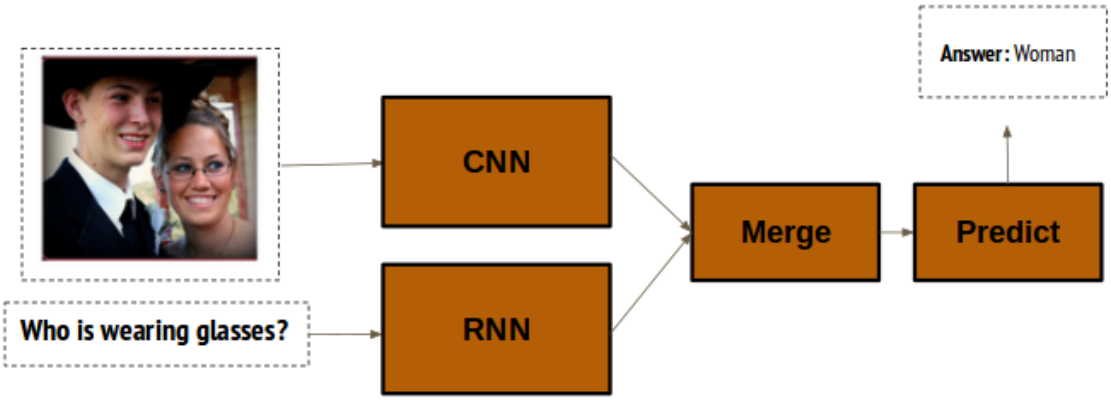

Figure 1.1: Typical VQA solution consisting on CNN and RNN

1.2 Requirements and specifications

The main goal for this project is to create and evaluate different Visual Question Answering models and submit the results to the 2nd Edition of the VQA challenge. Regarding this, the requirements are the following:

- Test different VQA models and try to increase the accuracy of the model submitted last year by the team representing the UPC.
- Prepare the UPC submission to the second edition of the CVPR17 VQA Challenge [1].
- Develop a software than can be used for future submissions to the VQA Challenge and future research in the field of visual reasoning [2].

The specifications are the following

- Use Python as a programming language to develop the project.
- Develop the project using the Keras[2] software framework for deep learning, a wrapper of lower level deep learning libraries such as Theano[3] or TensorFlow[4] backends.

1.3 Methods and procedures

This thesis considers as a baseline the model developed by the UPC team [6] for the VQA 2016 Challenge. This baseline project was developed using Keras, a deep learning framework built to enable easy and fast experimentation. This fast prototyping is accomplished by building a wrapper over Theano or TensorFlow frameworks, which provides a high-level neural networks

[1] http://www.visualqa.org/challenge.html
[2] https://keras.io/
[3] http://deeplearning.net/software/theano/
[4] https://www.tensorflow.org/

API with which you are able to build both convolutional and recurrent networks, as well as a combination of those. On that basis, Keras was a perfect perfectly to our needs. Regarding to the backend, we have chosen TensorFlow. This is largely due to the fact that TensorFlow compiles faster and, contrary to Theano, it provides a very useful visualization tool (TensorBoard). Apart from Keras we have used DeepFramework[5], an API that facilitates beginners to organize their projects and helps with tasks which are not purely related to deep learning.

The baseline project [6] used pre-computed visual features provided by the Computer Vision group at Universitat de Barcelona, obtained using Kernelized Convolutional Neural Networks [8]. This need of using pre-computed visual features is explained with more detail in Section"1.5, as the problem has persisted during the development of this thesis. This is why all the models considered in this thesis have pre-trained weights from models trained in a large and popular image classification dataset named ImageNet [9].

1.4 Work Plan

This project has been held between the GPI and the TALP research groups at Universitat Politècnica de Catalunya, having a regular weekly meeting to discuss decisions to be made. This meting has been complemented with a weekly seminar with other students developing their bachelor, master or Phd thesis at GPI to present our research and share our knowledge.

The work plan is described in the following work packages and Gantt diagram, as well as the modifications introduced since the first version.

1.4.1 Work Packages

- WP 1: Project proposal and work plan
- WP 2: Introduction to visual question-answering tasks and Keras
- WP 3: Dealing with VQA models
- WP 4: Critical Review of the project
- WP 5: Participate in the VQA Challenge
- WP 6: Final report of the project
- WP 7: Presentation and oral defense

[5] https://github.com/issey173/DeepFramework

1.4.2 Gantt Diagram

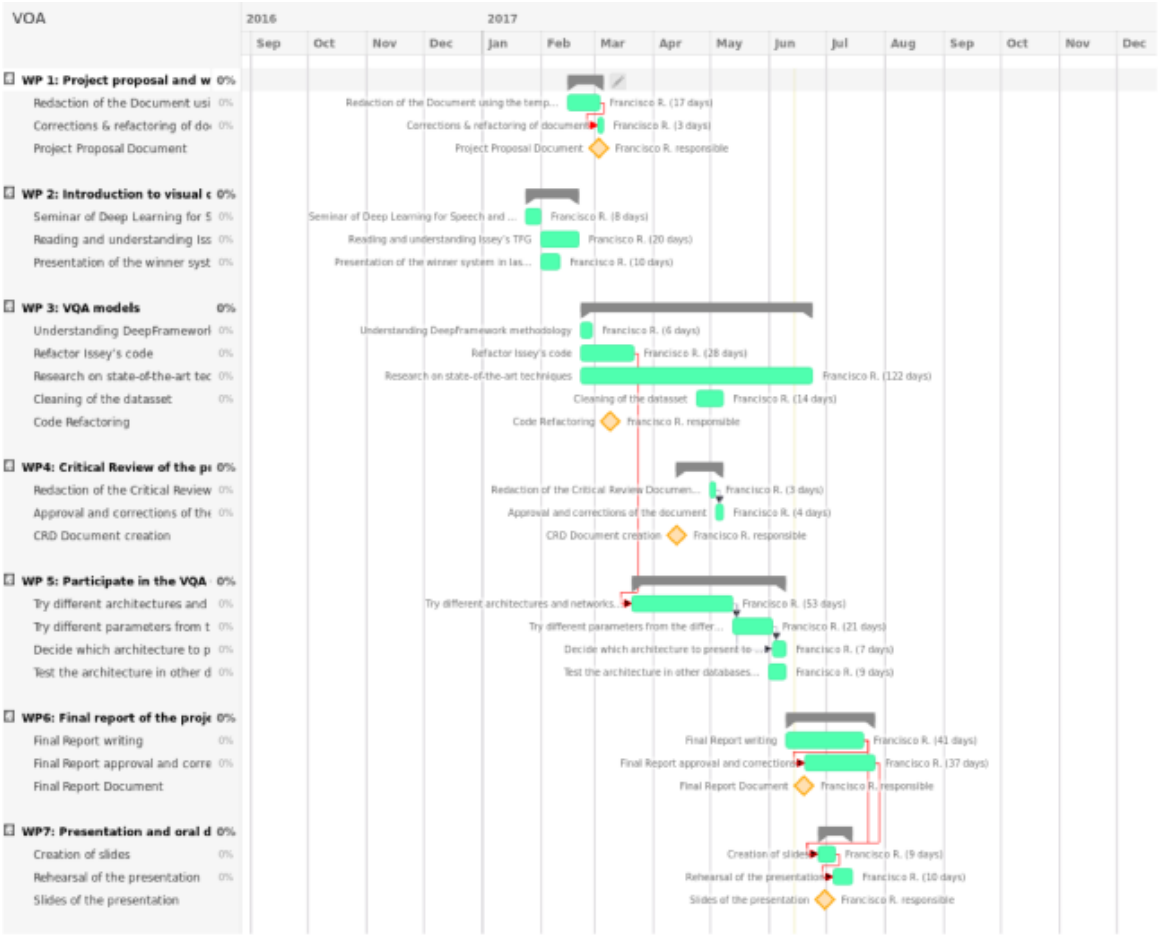

Figure 1.2: Gantt Diagram of the Degree Thesis

1.5 Incidents and Modification

The project has experienced some modifications since the beginning of it, mainly due to time constraints. Moreover, as long as some release dates were reached, some WP dates and milestones have been moved.

At the very beginning of the project we wanted to run completely over Keras and Deep-Framework, but we finally decided to replace some specific parts of this last API with our own implementation. This modification required much more time to reuse the code developed last year and, therefore, to begin to build new VQA architectures and test them.

Secondly, the limited computational resources to run our experiments have been an important bottleneck due to the high demand in the GPU cluster from the GPI research group. The VQA Dataset contains a large amount of samples to train our models. The VQA 1.0 from for the 2016 challenge contained 2,483,490 samples, while the VQA 2.0 dataset used in 2017 almost double its size, as it contains 4,437,570 samples). We need about 40 epochs to obtain an acceptable result for the train and validation curves. Notice that an epoch is defined as a single pass of

all the examples in the training set through the model under training. As a Bachelor student at GPI, we only have access to a single NVidia Titan X GPU at the cluster of servers of the Image Processing Group at UPC, equipped with 12 GB of RAM 4.1. With this resources, fine-tuning the visual representation of a basic VQA model takes about 11 hours per epoch. In other words, to train a model we would need to have our process running during 18 days to obtain an acceptable result. For this reason, it was not easy to train the visual feature extractor for the VQA task, forcing us to use pre-trained models from other datasets.

Chapter 2

State of the art

In recent years, multimodal problems that combine multiple data types (images, audio, video, natural language...) have been a rising trend in Artificial Intelligence research. This tasks are typically addressed by breaking down the whole problem into separate sub-tasks.

Visual Question-Answering is an example of these kind of multidisciplinary problems, in which the model needs to understand the scene and objects represented by an image together with the relations between them in order to answer natural language questions about them. Section 2.1 reviews the state of the art in VQA, focusing in Multimodal Compact Bilinear (MCB) pooling *et. al.* [1], the state of the art after the VQA Challenge 2016.

However, during the first half of 2017, it has been proved that these models have poor performances for complex questions which require understanding of many object attributes and relations between them, leading the community to look for models that perform more reasoning steps, emerging a new concept: Visual Reasoning (Section 2.2).

2.1 Visual Question Answering

Since the launch of the VQA Dataset, Visual Question Answering has received a lot of attention from the community. However, performances obtained until the moment are far from resembling the human level, so efforts to solve this task are growing exponentially.

As stated in Section 1.1, the most common solution to solve the VQA task is to merge the visual representation obtained from a convolutional neural network and the text representation obtained from an embedding recurrent network. However, humans do not answer questions by analyzing the image and the question independently, but we focus on parts of the image depending on the question. A typical approach to model this behavior is to merge the embedded question with the image representation, to predict attention weights over convolutional layers and later project the resulting image representation to the textual vector domain. This approach, called Soft Attention (SA) mechanism, was proposed by Xu *et al.* for image captioning [10] and by Yang *et al.* [11] and Xu and Saenko [12] for VQA.

Taking the SA as a basis, Fukui *et. al.* [1] made a study on how to merge the visual and the textual representations. Typically, the way to merge this vectors is through an element-wise sum or product or through a concatenation of both of them. However, they claimed that the outer-product is a more efficient way to fully capture the associations between both modalities, and, because it is unfeasible due of the high dimensionality of the parameters to learn, they propose a novel method called Multimodal Compact Bilinear pooling (MCB pooling).

Basically, they project both representations to a lower dimensional space using the Count Sketch algorithm [13], reducing the amount of parameters to be learned to obtain the multimodal representation. To avoid computing explicitly the outer product, and as the count sketch of the outer product of two vectors can be expressed as convolution of both count sketches, they

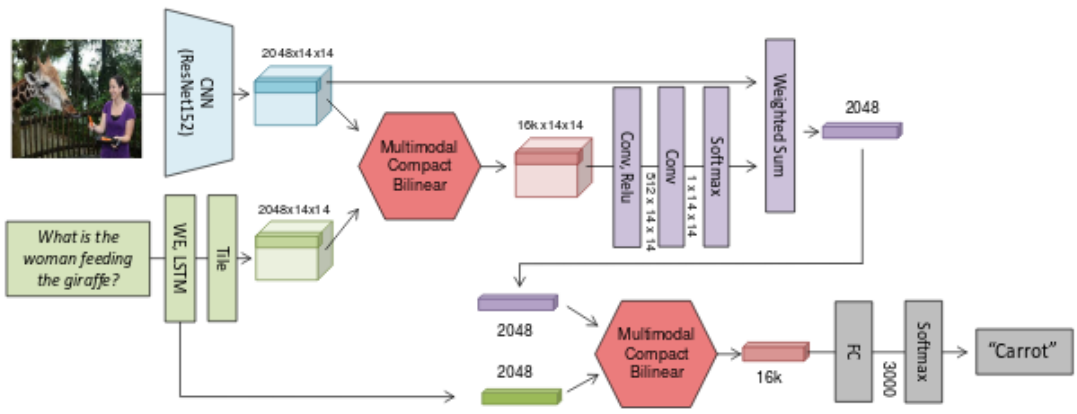

Figure 2.1: Multimodal Compact Bilinear Pooling with Attention architecture [1]

compute the element-wise product of both vectors in the frequency domain.

This method set a new state-of-the-art and won the 2016 Open-Ended VQA Challenge, obtaining a 66.9 % of accuracy. In the VQA 2.0 used in the 2017 challenge, it obtained a 62.27%.

This year at ICLR conference Jin-Hwa Kim *et al.* presented the Hadamard product for low-rank bilinear pooling [14], which outperforms the MCB pooling in the Multiple Choice category of the VQA 1.0 Dataset, obtaining a 70.29% in front of the 70.10% obtained by the MCB method. Moreover,on the Open-Ended category they perform with a 66.89% of accuracy.

2.2 Visual Reasoning

The idea of Visual Reasoning comes from the human skill to manipulate one's mental representation of an object or a scene in order to extract some conclusions about it. When humans are asked about a visual scene, we first process the question to understand what the emitter is talking about and, once it is understood, we perform a series of actions to analyze the image and be able to answer correctly to the question (*e.g.* for the question *"What color is the object next to the cube?"* we would have to first localize the cube, look next to it and then detect the color of the object in that position). This reasoning pipeline is not applied in conventional methods that solve VQA tasks, as the actions that the system should learn to perform are not determined.

One of the first approaches to deal with this pipeline was done by Jacob Andreas *et al.* who proposed the Neural Module Networks (NMN) [15]. Their research is based in exploiting the compositional language structure using a hand-engineered syntactic parser, which determines which actions to perform. Each of this actions represent a *module*, and each module is composed by a set of neural layers. Therefore, for each question the system decides which actions to perform between the possible set of actions and builds its neural network concatenating these modules. Then, it joins this visual vector representation to a textual embedded vector to obtain the most probable answer. Although this approach obtained worse results than the MCB pooling (58.7% of accuracy in the test-dev split of the VQA dataset), it represents a big step forward towards the modeling of humans capacity to reason about the world. Nevertheless, the syntactic parser fails when it has to deal with more complex questions that may involve many relationships between the objects of the scene (comparisons, counting, identifying attributes...).

Recently Justin Johnson *et al.* [2] have proposed a program-induction method inspired in the work previously mentioned, dividing the system into two different parts: the program generator and the execution engine. The program generator refers to the system that predicts the actions (named functions) to perform for an input question. This set of functions are named program. As well as in the NMN, each function represents a set of neural layers, so each program will be represented by its own neural network. The execution engine is the part of the system in charge of mapping each function to its neural module, building the network and predict the answer.

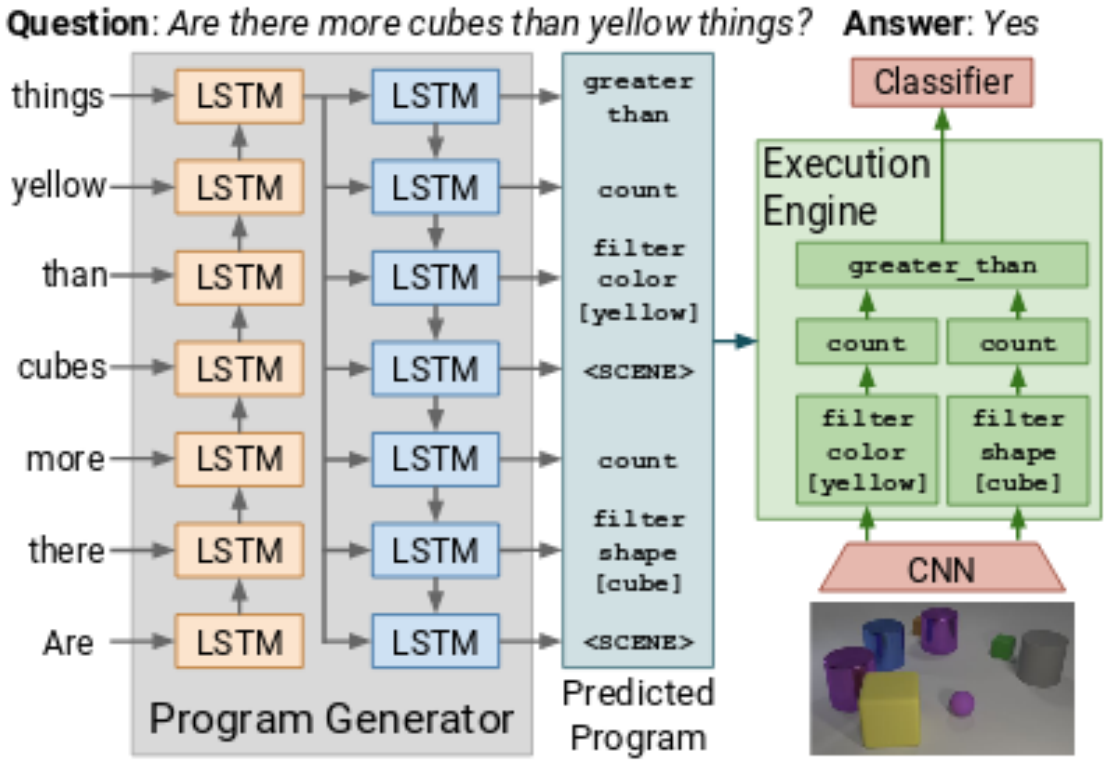

Figure 2.2: Visual Reasoning program-induction overview [2]

The main differences between the NMN and this novel method are:

- The syntactic parser is no longer hand-engineered. As questions are sequences of words, they replace the syntactic parser with a standard LSTM sequence-to-sequence model [16]. There is a catalogue of possible functions to predict which has a fixed arity (*e.g.* a comparison module would need two inputs while to detect the color of an object you would just need a vector of image features). This allows them to establish a syntactic hierarchy when deciding the order of the functions in the program generator.
- Modules have a generic architecture. Contrary to NMN, where modules had its own layers architecture, this approach keeps the same set of layers for each function and lets the network to learn to perform them. This becomes an advantage because fixing modules architectures ensure that for every valid predicted program the execution engine will generate a valid neural network.

Both components of the system can be trained independently if we have a program ground truth in our dataset. However, labeling a whole dataset with suitable programs is very expensive

and most of the datasets would have none or few programs as ground truth. The desired solution would be to train end-to-end the whole system using triplets of image-question-answer, but it is impossible to backpropagate the error through an argmax operator (the predictor layer of the program generator takes the most probable function at each timestep), so they replace this operator with sampling and make use of reinforcement learning using as a reward the zero-one loss of the execution engine. Doing so, they are now able to backpropagate the error. Notwithstanding, training from scratch the model using this method is also very expensive as the network must learn without understanding what functions mean. They finally conclude determining that a semi-supervised learning approach is the most suitable for the task context, using a small set of ground truth programs to train the program generator, to later fine-tune with reinforcement learning.

This method was tested through many experiments, showing that there is still a work to do to identify unknown attributes by the network or to process correctly long complex questions that include many descriptive aspects of the objects present in the scene. Furthermore, there is still the need to add functions that perform ternary operations (if/else/then) and loop operations (for/do) to be able to answer questions such as *"What color is the object of the unique shape?"*, in which we need to loop over every object detecting its shape and filter the color just if there is just one object of this shape.

Chapter 3

Methodology

3.1 VQA 2.0 Dataset

With the second edition of the VQA Challenge in 2017, the organizers have provided a new dataset. Many simple architectures performed surprisingly well in the 2016 challenge edition by learning language biases, resulting that these models ignore the visual information to answer the question. For instance, people tend to ask questions such as *"Is there any tree in the image?"* if there is actually a tree in the image, so many models would automatically answer *"yes"* without considering at all the image features. On the other hand, in order to avoid this problem, this year the dataset has been balanced by collecting pairs of complementary images which are similar between them, but have different answer for the same question (Fig. 3.1).

Figure 3.1: Examples of complementary images from the VQA 2.0 Dataset

This dataset uses the images from Microsoft Common Objects in Context (MSCOCO) Dataset [1], designed for image recognition, segmentation, and captioning, and containing 82,783 images in the train split, 40,504 in the validation split and 81,434 images in the test split. Each image has associated 5.36 questions on average, which results in a total of 443,757 questions on the training split, 214,354 on the validation one and 447,793 in the test. Each question has as a ground truth a set of 10 plausible answers, making the amount of samples up to 4,437,570, 2,143,540 and 4,477,930 for the train, validation and test splits respectively. Comparing to last year dataset we can notice that the number of samples has increased by a factor of 2 in each dataset division.

[1] http://mscoco.org/home/

All the questions and answers have been made by different annotators, and they are divided depending on the type of answer that is given (*yes/no* questions, *number* questions and *other*).

The Challenge itself is divided into four different categories, depending on the content of the image and the kind of answers to predict. It firstly differentiates between real and abstract (artificial) scenes. Also, questions can be Open-Ended, in which the most probable answer over a whole dictionary must be predicted, and Multiple-Choice, in an answer over four possible options must be chosen. In our case we have decided to work with the Real and Open-Ended category, the most challenging one.

3.2 Preprocessing Data

3.2.1 Image Data

When working with CNNs we normally use raw images as input data, but some preprocessing must be done before. CNNs learn by continually adding gradient error vectors (multiplied by a learning rate) obtained from a backpropagation through many matrices batch by batch. It is in our interest to ensure that every image has a similar range of values in order to avoid that the gradients run out of control, and the way to do it is by subtracting the mean of the whole image dataset to each sample. This way, the gradients act uniformly for each channel. Otherwise, the learning rate would cause corrections in each dimension that would differ, and compensating a correction in one weight dimension might imply undercompensating in another, provoking difficulties for the loss to stabilize.

3.2.2 Natural Language Data

Analogously to humans, an AI system chooses the answer to a question from a known vocabulary. This vocabulary, also called dictionary, contains a number of indexed unique words, usually between 10,000 or 20,000, which can be predefined and customized. In our case we customized our dictionary by parsing all the questions and answers, and selecting the top 10,000 most common words, a common figure in other similar works.

In order to feed our future network with natural language data, we first have to split the sentence (question) and assign each word to a number by mapping it to its corresponding index in the dictionary. This process is referred as *tokenizing* the question. As we need to have a fixed input shape to feed our network, we have set the maximum question length to 22 and, if the question under consideration is shorter, the vector is padded with zeros (which is not a valid token for a word).

3.3 Baseline

3.3.1 Architecture

The baseline of this project is the model presented by the UPC team in the CVPR16 VQA Challenge 3.2. This model uses precomputed image features obtained from our partners of the Computer Vision group at Universitat de Barcelona, who used a Kernelized CNN (KCNN) [8] to extract them. This kind of CNN aims to provide better image vectorization than vanilla CNN's, as those have shortcomings when trying to analyze complex images.

On the other hand, to obtain the textual vector, the tokenized question is fed into an embedding layer, which projects the question to a different space dimension which captures semantic and syntactic relationships between words and the context in which the word appears, to later use a one-layer LSTM to obtain the text representation. The use of LSTM in the question branch is due to the advantage of having memory over time, using a memory cell or state which is updated in each iteration and its output is related to this state.

This model obtains a 53,62% of accuracy in the real test of the VQA 1.0 dataset, while in the new dataset it obtains a 40.25%. More details are presented in chapter 4.

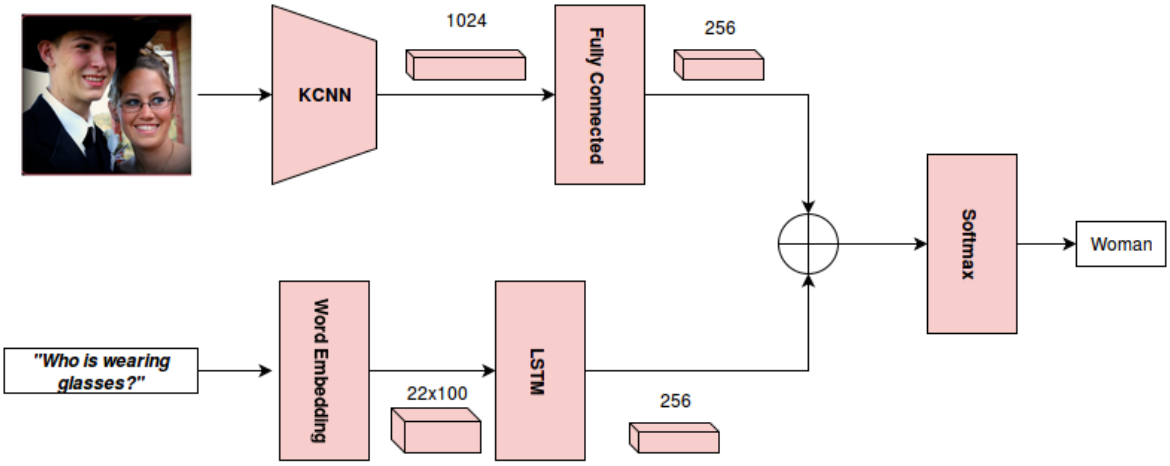

Figure 3.2: Baseline model diagram

3.3.2 Hyperparameters

When training a neural network, we do not only have to choose which layers to use and how to connect them, but also a large number of hyperparameters that influence the learning process.

Firstly, the precomputed image features have a dimension of 1024. On the text side, by contrast, we have an embedding layer of dimension 100 (resulting in a tensor of shape 22x100, being 22 the maximum length of the question 3.2.2), which is fed into an LSTM of 256 depth - that means, we use a LSTM with 256 hidden units. As we have to join both representations and they have different dimensions, we project the image features into the text representation dimension through a Fully Connected layer. Then, having both representations of the same dimension we can merge them by making an element-wise sum. We predict the answer by using

a softmax layer of the size of our dictionary, and associating each activation to its index in the dictionary. For instance, if the maximum value of the softmax layer is obtained the 10th neuron, we would select the word with index 10 of our vocabulary as the predicted answer.

During the training process, the parameters are updated in order to find a minimum in the loss function. This is done using a learning rate, which sets how quickly a network abandons old beliefs for new ones. In general, one wants to find a learning rate that is low enough for a model to converge to a useful configuration, but high enough to avoid a very long training time. The starting learning rate is set to 0.0001 and governed by the Adam optimizer[17].

In order to avoid having high variance in the weight updates, we feed the network with a fixed number of samples at each iteration and update them using the loss of all of them. We have set this number of samples, called *batch size*, to the maximum one that fits in the GPU RAM. This decision must also take into account the space for all the feature maps generated in each forward pass, as well as its weights. This way, this model uses a batch size of 128. The total training time of this model is approximately of 2 days (about 1.2 hours per epoch).

3.4 Towards Improving the Baseline Model

3.4.1 Language-only model

One of our biggest concerns about our model is using of the visual features and to avoid learning just biases from the language and the dataset. In order to be able to determine whether our models are using image features or not, we have built a model which uses just the questions as input, omitting completely the image features.

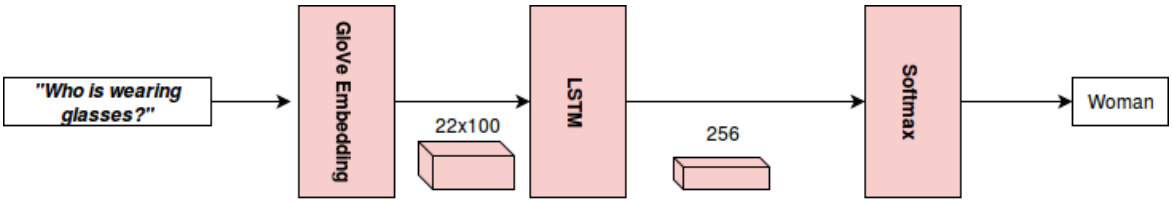

Figure 3.3: Language-only model to capture how many language biases the network learns

This model uses GloVe word embeddings [18], which we proved that work better for our VQA model than learnable word embeddings (see Section 3.4.2.3 and Chapter 4 for more details). The architecture follows the same proposed single LSTM layer to predict the most probable answer over the 10,000 words in the vocabulary.

3.4.2 VGG-16 based model

This approach consists on replacing the KCNN used in [6] with another more common CNN with high success in Visual Recognition tasks. We adopted the VGG-16 [19] model, which obtained the first and the second places in the localization and classification tasks respectively of the ImageNet ILSVRC-2014 Challenge [20].

3.4.2.1 Model Architecture

This model uses a truncated VGG-16 to obtain image features, meaning that we have removed the last fully connected layers and we just keep the convolutional layers. In particular, we remove all the VGG layers from the fc-4096 to the softmax layer. This causes that the image features extracted will not be a single vector as in the baseline model, but instead a 7x7x512 tensor. For this reason, we add a Flatten layer to reshape this features into a one dimensional vector (1x25088) that will be later projected to the dimension of the textual representation.

With this method we have also changed the way to obtain the multimodal representation, switching from an element-wise sum to an element-wise product, as this method obtains better performance according to [1].

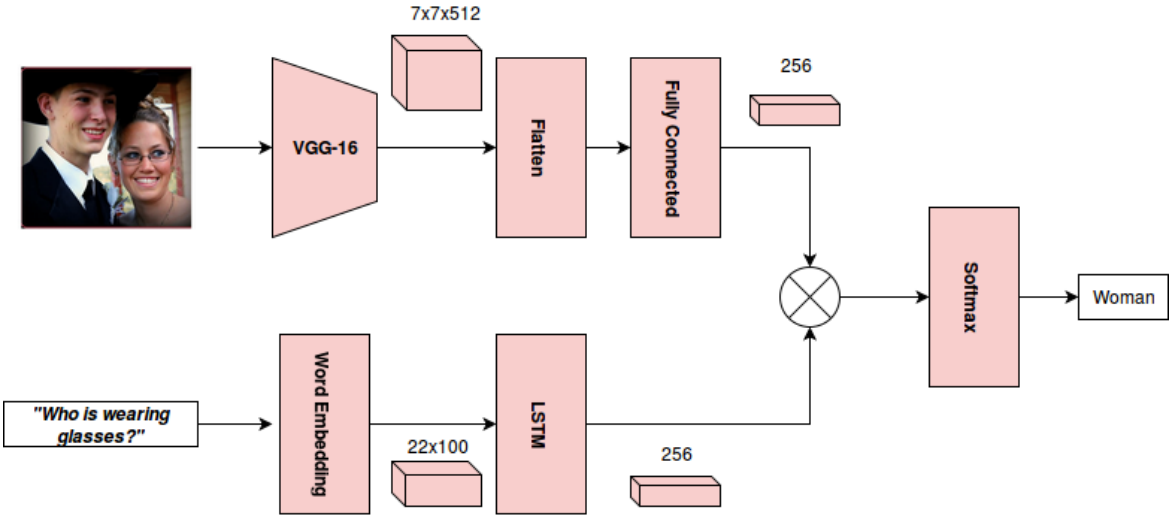

Figure 3.4: VGG based model diagram joining visual and textual representations through element-wise product

3.4.2.2 Attempting to Fine-Tune VGG-16

One of the obvious ways to exploit image information is to fine-tune the visual branch with the dataset we are using. That is, updating the weights of the CNN to adapt them to the different images in the VQA 2.0 dataset. When fine-tuning a network, one can update all parameters or freeze some layers, updating just weights of certain parts. However, fine-tuning the image branch has huge effects on the resources needed, making the training time to increase disproportionately, as the learning process takes 17 hours per epoch, resulting in 28 days of training to obtain acceptable weights. This make unfeasible to train this branch in the time span of this bachelor's thesis.

Nevertheless, one common cause of poor performance is misusing GPUs, or essentially "starving" them of data by not setting up an efficient data pipeline. In order to avoid this problem, we reorganized the data pipeline, saving all of it in the hard disk instead of the RAM. Doing so, however, we move the bottleneck to the CPU, as it will require more time to load the data while generating every batch. This is why we used a multi-threading system, in which each worker generates a batch, and a queue with a double size of the batch to store the data, optimizing the

memory handling. This allowed us to increase the training speed a 35.29 %, from 17 hours per epoch to 11 hours.

Even with this gain in performance, the training time was still too long, as it would require 18 days to fine-tune the VGG model. Therefore, no fine-tuning was considered in this thesis, taking all the CNN's weights from pre-trained models on ImageNet dataset[9]. Doing so, we decrease the training time to approximately 3 hours per epoch (5 days of training).

3.4.2.3 Moving to GloVe embeddings

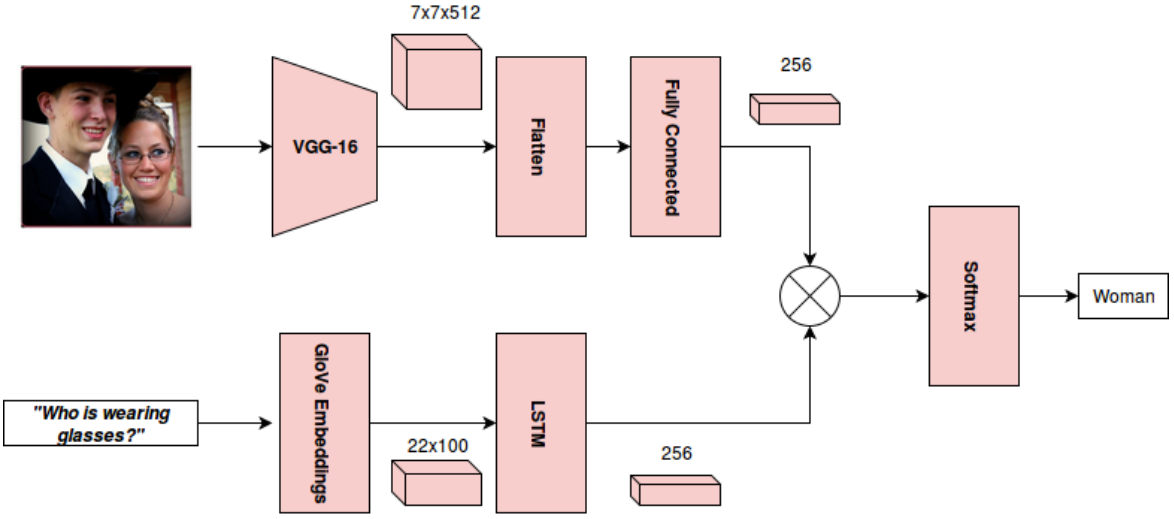

Figure 3.5: VGG based model using GloVe embeddings

Another improvement with respect to [6] was to using better word embeddings than the ones that can be learned using the backpropagation algorithm. We adopted the popular Global Vectors Word Representations (GloVe), proposed by Jeffrey Pennington *et al.* [18], which produces a matrix space with semantic meaning and has been proved to have good performances in word analogy tasks.

This embedding matrix contains 400,000 different words and their representation in 4 different dimensions (50, 100, 200 and 300), though we have used the 100-dimensional ones. In order to add those embeddings into our model, we have kept just the 10,000 words we are using and sorted the matrix fixing the same word order used in the dictionary. We also needed to add a zero vector at the first position due to the zero-padding in the question tokenization.

3.4.2.4 Batch Normalization and Average Pooling

As stated in 3.2.1, a normalization step is required as a preprocessing step to make data comparable across features. However, as data flows through a deep network, the weights and parameters adjust their values, sometimes making the data too big or too small again, a problem commonly named *internal covariate shift*. This difference in the range of values can provoke our training process to have difficulties to converge, as some features would provide more information to the network to others.

Under the assumption of not being exploding image features, we use Batch Normalization [21] to adapt their values, adding this layer before flattening the VGG image features extracted. As we are normalizing the visual features, we also want to do it with the textual representation in order to have similar range values for both vectors. Nonetheless, normalizing both branches or just one of them caused the network to overfit (Chapter 4), making us to consider other options. We hypothesized that the fully connected layer was causing this, as we were projecting a vector from 25,088 dimension to a 256, being possible this transformation to be too specific to generalize to new images. In order to have a reduced image feature vector, we tried to make an Average Pooling in two different ways: one first attempt calculating the mean of each row in the feature map, moving from a tensor of shape 7x7x512 to 1x7x512, and another making the same operation taking filters of 7x7, which results in a tensor of shape 1x1x512 (Fig. 3.6). However, the model still overfitted.

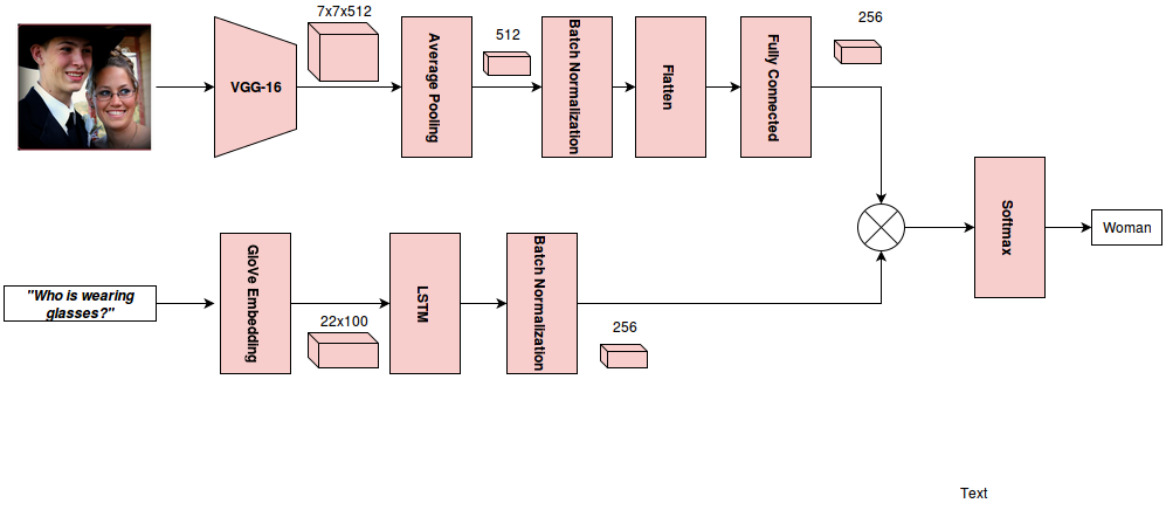

Figure 3.6: VGG based model using Batch Normalization and Average Pooling

In order to avoid this overfitting, we attempted several methods without success. Firstly, we tried adding dropouts [22] into the recurrent layers to constraint the learning in the textual part. Next, we tested with another kind of regularization, named "L2", which penalizes high variance features - adding this regularization increased the training time to 4.8 hours per epoch -, and tried reducing the learning rate from 10^{-4} to 10^{-5}

3.4.3 ResNet based Model

As we were having problems with overfitting using VGG features, and the training time was not negligible, we decided to move to another CNN which performs better for image classification tasks, as it was the winning entry in the ILSVRC 2015 ImageNet Challenge[20]. Deep Residual Networks (ResNet-50) [23]. Apart from performing better in those tasks, the visual features extracted have a much lower dimension - they are a vector of 2048 length for each image -, which makes us reduce the training time substantially (about 1 hour and 15 minutes per epoch, resulting in 2 days of training), as less parameters must be learned in the projection of the image representation to the semantic space.

3.4.3.1 Element-wise product merging

ResNet-50 authors *"explicitly reformulated the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions"*. This is done through *identity blocks*, which have three convolutional layers and at the end combine the resulting features with the input tensor through an element-wise sum. Each of these blocks uses Batch Normalization to adjust the range of values of the features generated, causing the features extracted for this model to have similar range of values (the mean of all the features is 0.5 and the variance 0.59), reason why no Batch Normalization have been added to the image branch. On the other hand, we have kept this layer into the textual branch after the recurrent layer to have both representations in the same range of values.

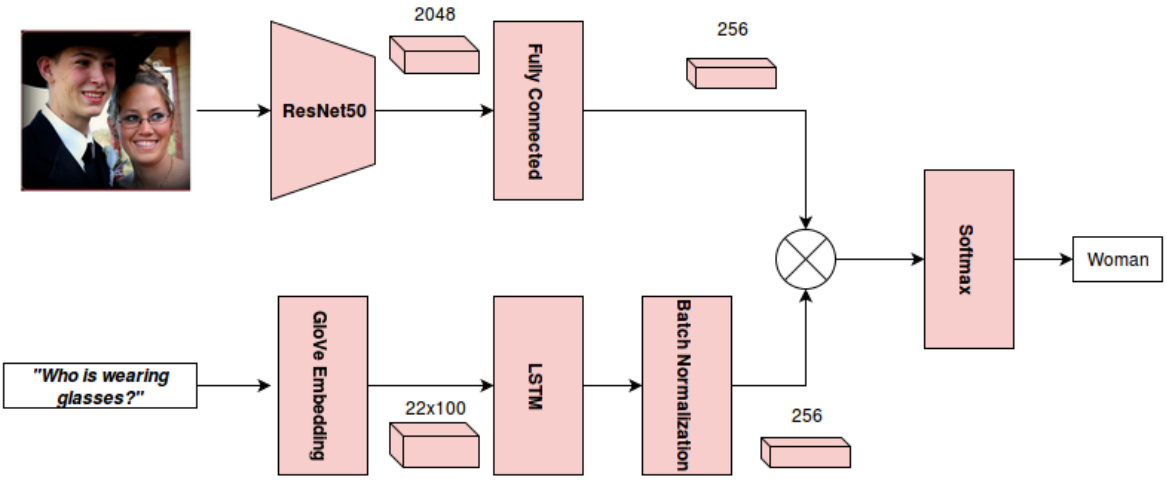

Figure 3.7: ResNet-50 based Model merging visual and textual representations using an element-wise product

As the network was overfitting with the Batch Normalization in both branches, we made two different tests: (1) adding a dropout to constraint the learning in the projection of the visual features to the semantic space and (2) adding a dropout after merging the textual and the visual representation.

3.4.3.2 Element-wise sum merging

The main reason why we moved from merging both representations using an element-wise sum to an element-wise product was because of the results shown by [1]. However, the best way of combining the representation may depend on the network and the hyperparameters used, being possible for a network to perform better obtaining the multimodal vector using a different technique from the element-wise product(Figure 3.8).

Therefore, we moved back to the element-wise sum, which seemed to be more intuitive mathematically (Figure 3.8), to compare between both methods of merging representations for our model. We kept the dropout layer after merging both representations as it was the most successful test of the previous section.

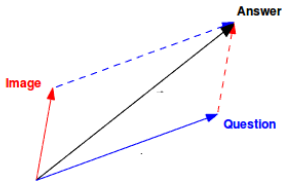

Figure 3.8: Vectorized representation of the merge operation using element-wise sum

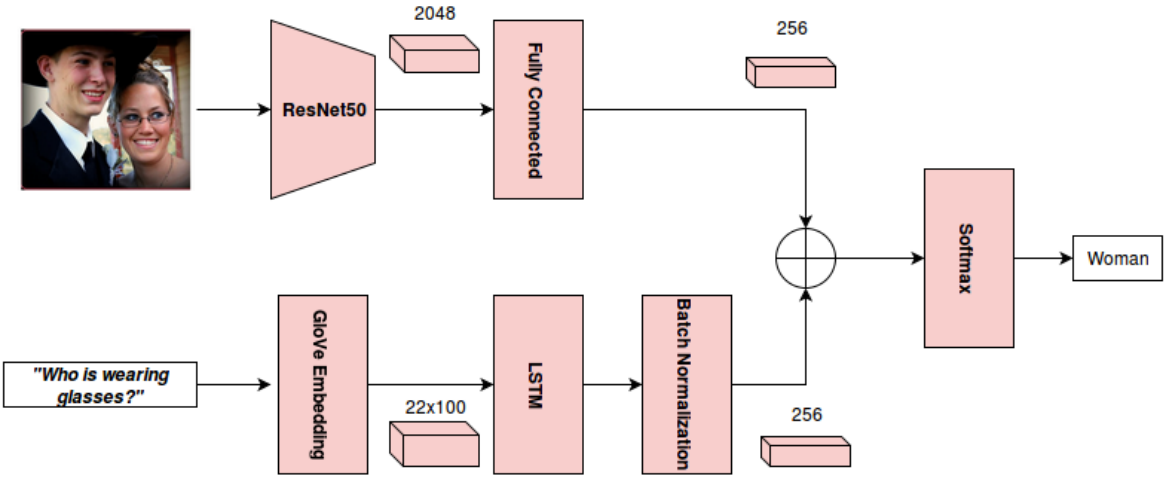

Figure 3.9: ResNet-50 based Model merging visual and textual representations using element-wise sum

3.4.3.3 Concatenation merging

As explained in Section 3.4.2.4, one of our concerns was the difference of dimensionality between the textual and the visual representations. All our experiments had taken the image features and projected them to the semantic space, reducing the dimensionality of them. In the case of the ResNet based model, moving from a 2048 dimensional vector to a 256 one, to later project again the merged vector to a 10,000 dimensional to predict the answer.

In order to avoid this abrupt changes in dimensionality, we can expand the dimension of the textual representation, for example using a stack of two LSTM's and concatenating their outputs like in [1] or simply by adding more hidden units, but this would also lead to an increase in training time. Instead, we omitted the projection of the image features to the semantic space and concatenate both representations (of size 2048 and 256 for image and textual features respectively). This way, the merged vector will have a dimensionality of 2304. The learning of the relations between the features of this multimodal vector were been tested in two different ways: (1) feeding directly the concatenated vector to the predictor layer and (2) adding an intermediate Fully Connected layer to work with a learnable multimodal vector.

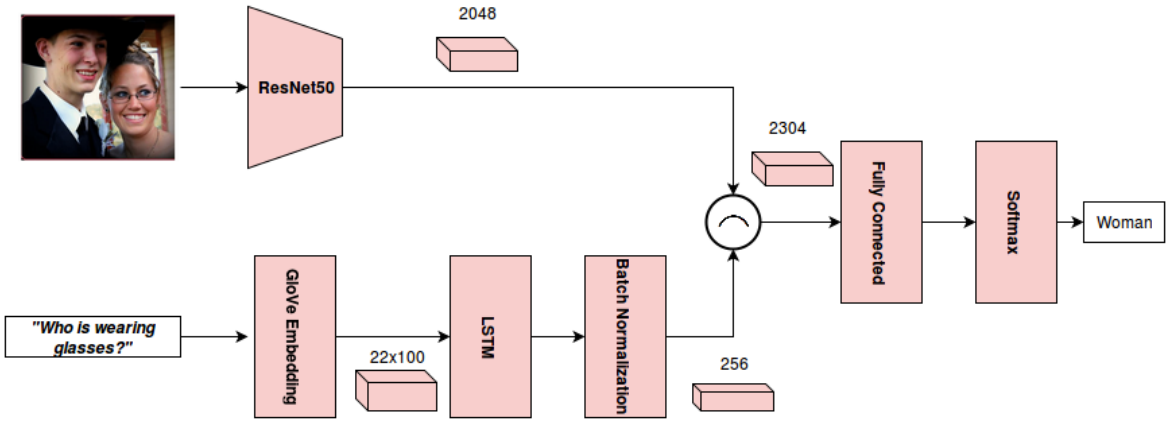

Figure 3.10: ResNet-50 based Model merging visual and textual representations using concatenation + FC

Chapter 4

Results

This chapter presents the results obtained with the different models explained in Chapter 3 to try to improve the baseline system described in Section 3.3.

4.1 Computational requirements

Experiments have been run with the computational resources available at the Image Processing Group of the Universitat Politecnica de Catalunya. When dealing with deep learning projects, one of the main concerns is the computation resources you have, as you are often working with hundreds of thousands or millions of data. This is why GPI has a cluster of servers which is shared between all the research group and in which we can run very long experiments. For each experiment we must ask for the resources needed (number of GPU's/CPU's, reserve RAM) and is sent to a queue of processes. As soon as there is enough resources as you demanded, your process start running. Notice that as TFG student it is just possible to reserve a NVidia Titan X GPU with 12GB of RAM [1] and run just one single process using it.

4.2 Dataset

Before describing how a VQA model is evaluated and the results obtained, it is worthwhile to summarize the explained in Section 3.1 about VQA 2.0 Dataset:

- Training split: 82,783 images, 443,757 questions and 4,437,570 answers
- Validation split: 40,504 images, 214,354 questions and 2,143,540 answers.
- Test split: 81,434 images and 447,793 questions.

There is also a smaller dataset split called *test-dev*, which has 107,394 questions associated to any of the images from the test split. This smaller set is used to test the different models, as there is a limit of 5 submissions for the whole test dataset.

4.3 Evaluation metric and EvalAI

All the different models have been tested using the evaluation metric proposed by the organizers of the challenge which, according to them, *"is robust to inter-human variability in phrasing the answers"*. This metric responds to the following equation:

[1] http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan-x/specifications

$$Acc(ans) = min(\frac{\#humans\ said\ ans}{3}, 1) \tag{4.1}$$

The overall accuracy is calculated by an average of the accuracies of all the answers to the test questions. Equation 4.1 means that an answer is given the maximum accuracy if at least three human annotators gave that answer for that question, while if the answer has no matches with the ground truth annotations it is given an accuracy of 0%. From this, each match implies an increase of 33% in accuracy. Notice that there are a total of 10 ground truth annotations for each question.

In order to calculate the overall accuracy you must submit a results JSON file which must follow the format established by the organizers. Before evaluating your submission file, their script makes some processing to the predicted answers like replacing upper case characters to lower case, adding apostrophes in contractions that are missing, removing determinants...

Furthermore, they have made use of a new platform to hold the challenge, named EvalAI. EvalAI is an open-source web platform that aims to help researchers, students and data scientists create, collaborate and participate in AI challenges.

They have divided the submissions into three different categories: Real test-dev2017, Real test2017 and Real Challenge test2017. The Real test-dev2017 is to test your different models with the test-dev split and has no limits in submissions, and the submissions done in this category are not considered to be participating in the VQA Challenge. On the other hand, we have the Real test2017 and Real Challenge test2017, whose submissions imply participate to the Challenge as you are using the test split. The main difference between these two categories is that for the first one your results will be shown to the community while for the second one your results remain invisible until the challenge deadline.

4.4 Experimentation

When working with Deep Learning projects we analyze the results using training and, more important, validation loss, computing those values at each epoch. If everything goes as expected, both the training and the validation losses should decrease epoch per epoch. Once the validation loss starts increasing or converges, we can stop training our model. In Section 4.4.1 we show and analyze the different curves obtained for some experiments. Notice that due to the singularity of the evaluation metric this curves just gives us an intuition of how is the network doing, but does not provide accurate information about the future results in the testing. Due to time constraints, we have forced the training to stop when there is no validation loss improvement in six consecutive epochs.

In Section 4.4.2 we show the results obtained in the *test-dev* split for the selected models and in 4.4.3 we expose some qualitative examples obtained from our best model.

We have trained all our models using triplets of image-question-answer except from the Language-Only model, in which we have used pairs of question-answer.

4.4.1 Experiment Analysis

In this section we will revise the training and validation loss curves for some different experiments we made:

- Decide which word embedding to use:

  One of the first experiments we decided to do was to choose between using a learnable word embedding layer or GloVe embeddings (Section 3.4.2.3). In order to do it we trained twice the model presented in Figure 3.5, one time using pre-computed GloVe and one other time training from scratch this layer. We can see in Figure 4.1 how using GloVe embeddings seem to work better, as the validation loss in lower.

- Test how much language biases is the network learning:

  Making use of the Language-Only model (Figure 3.3) we pretended to know how many biases are learned just from the language model. We present the losses obtained in Figure 4.2 , and see how the validation loss ends up higher for that model than for the tested in the previous experiment, indicating that the network is actually using image features to make predictions.

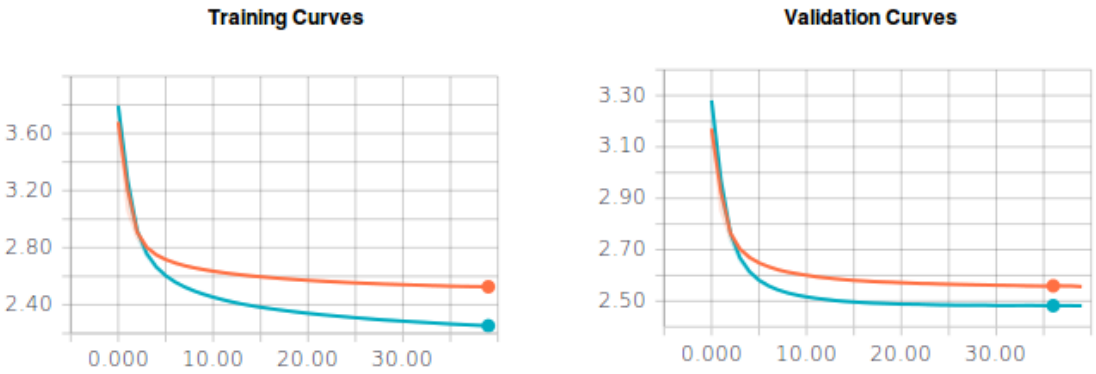

Figure 4.1: Training and validation losses using GloVe (blue) and a learnable embedding layer (orange).

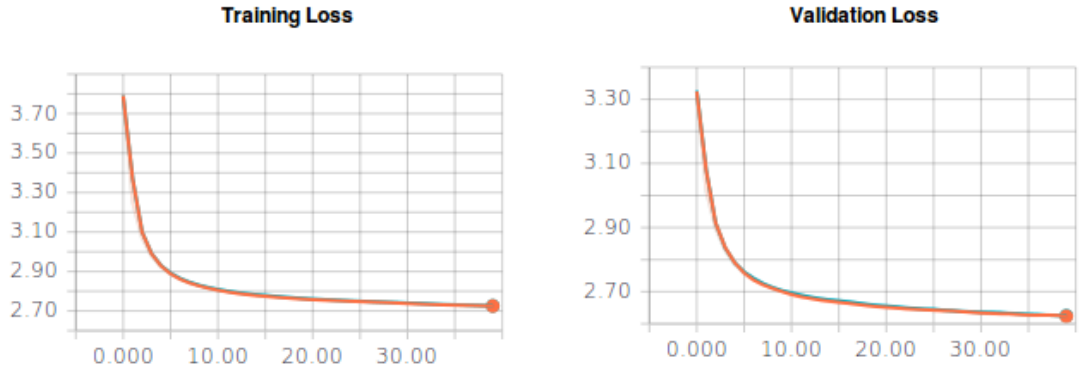

Figure 4.2: Language-Only model freezing Glove(orange) and fine-tuning it (blue).

- Usage of Average Pooling in image branch

  One of our hypothesis was that projecting a big vector to a much lower dimensional space could lead to bad generalization. In order to know if that statement was right, we tested the VGG based model using an Average Pooling of 7x7 (Figure 3.6 without the Batch Normalization layer) and another one without using it (Figure 3.5). The validation curves showed us that using this pooling does not help with the training, probably because we are getting rid of spatial information. This must be also the reason why when using ResNet features, which are averaged pooled, we obtain higher loss than with VGG features (compare validation loss values from Figure 4.3 and 4.4).

- Selection the merge operand

  One of the main issues we have to deal with when working with a VQA system is how to combine visual and textual information. We have tested the ResNet based model using three different merge operation methods and four experiments: element-wise product, element-wise sum and concatenation with an intermediate Fully Connected (FC) layer and without it (Figures 3.7, 3.9 , 3.10 and 3.10 without the FC after the merge operation respectively). In Figure 4.4 we can see the curves, and how the element-wise sum is the one which obtains lower validation loss.

  As we obtained better curves using the element-wise sum, we tested this join operation method also with the VGG based model using GloVe word embeddings (Figure 3.5) to compare the results in that network, although because of technical issues it was not possible to finish the training. However, in Figure 4.5 we see how using the sum operand was working worse.

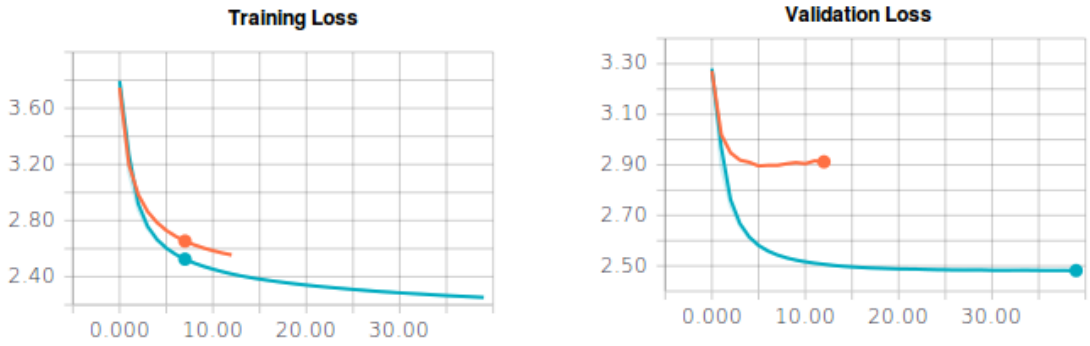

Figure 4.3: VGG based model from Figure 3.4 (blue) and from Figure 3.6 without the Batch Normalization layers (orange) to know the effect of Average Pooling.

- Addition of decay in learning rate:

  As we can see in Figure 4.4, the validation loss for the element-wise sum (purple curve) converge really quickly while the training loss remains decreasing. This can be a sign that we are doing "too big steps" when looking for the validation loss minimum. A way to deal with this behavior is to add a decay in the learning rate when it starts being in this situation, so that the learning rate value decreases and the "steps" are smaller. As we see in Figure 4.6, at the 20th epoch approximately this decay happened and the network was able to find a lower minimum.

- Fine-Tuning GloVe word embeddings:

  As we obtained better results using GloVe embeddings, we decided to try also initializing the word embedding layer with them but fine-tuning it, under the assumption that training

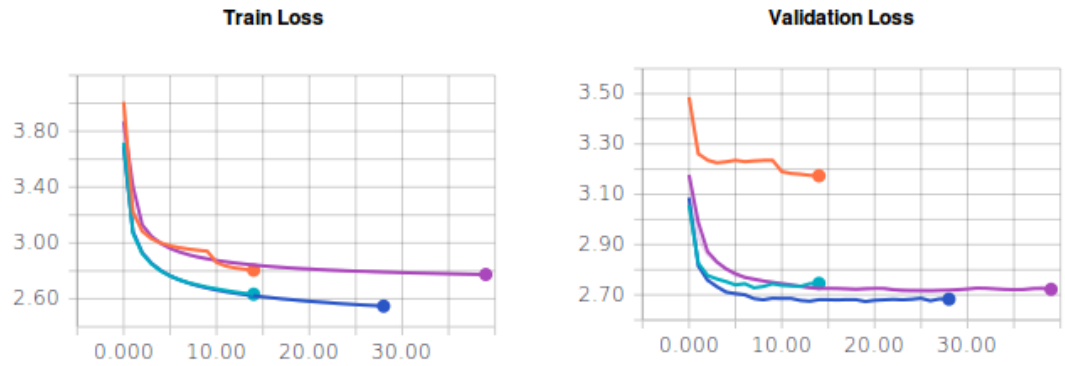

Figure 4.4: Training and validation losses for the ResNet model using element-wise product (purple), element-wise sum (dark blue), concatenation + FC (light blue) and concatenation (orange) methods.

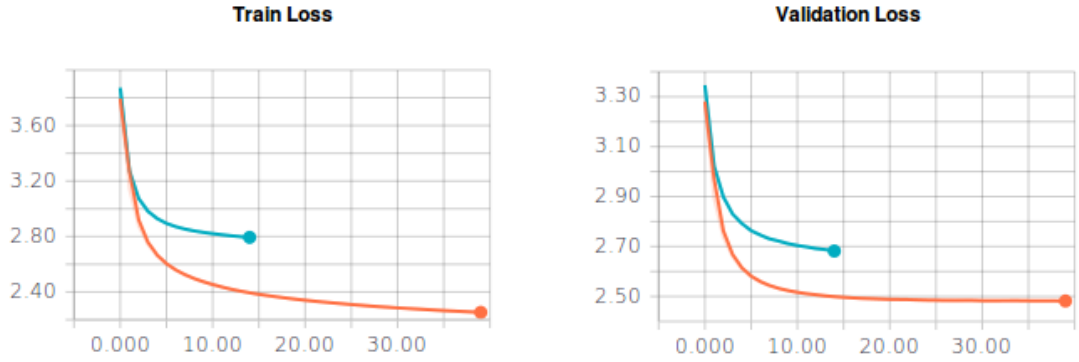

Figure 4.5: VGG based model with GloVe embeddings using element-wise product (orange) and element-wise sum (blue) merge operands

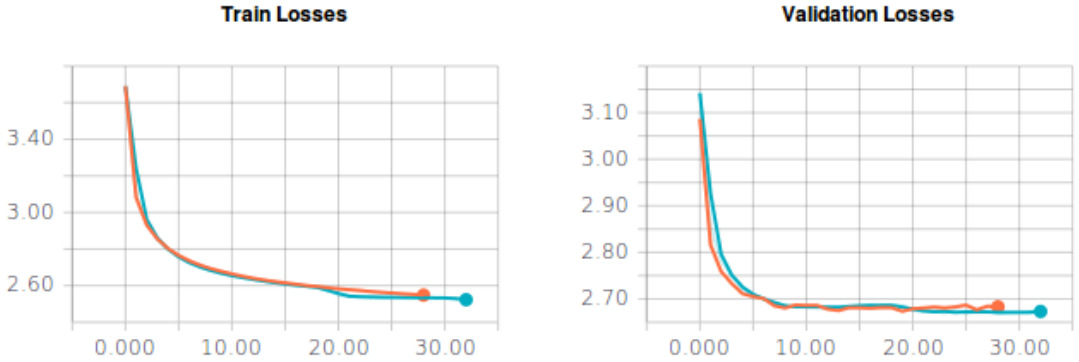

Figure 4.6: ResNet based model using element-wise sum with learning rate decay (blue) and without (orange)

the embedding layer could detect useful biases from the language (*e.g.*, every time a question starts with *"How many...?"* the model could detect that the answer must be a number). This can't be done freezing the GloVe embeddings as they have not been trained using a question-answering dataset. However, this did not suppose any improvement in performance.

- Adding Batch Normalization after joining visual and textual representations:

  The usage of Batch Normalization in all our models had provoked them to overfit. At last moment, we decided to make one more test using the ResNet model with element-wise sum and learning rate decay, but adding a Batch Normalization layer between the merge operation and the prediction block to ensure the whole batch is in the same range of values. Besides of obtaining better results for the validation loss (Figure 4.7), the training time decreases to less than an hour per epoch, confirming what it is stated in [21]. We had no time to compute its accuracy nor to test it with the VGG based model, but we expect it to improve the performance of our system.

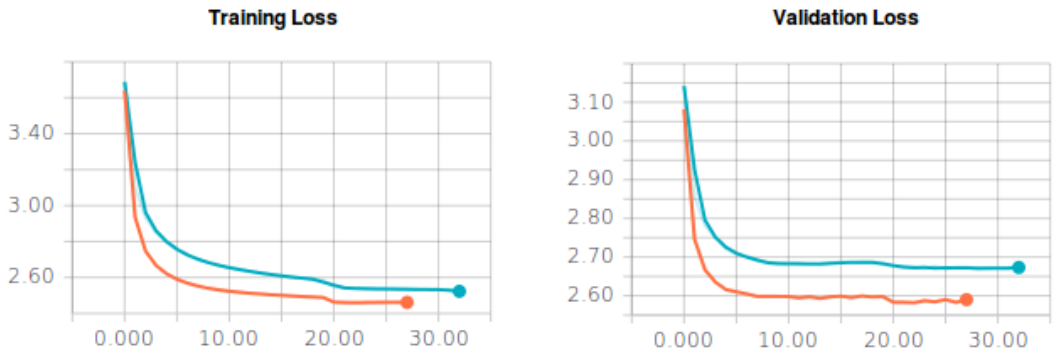

Figure 4.7: ResNet based model using element-wise sum with learning rate decay (blue) and adding a Batch Normalization after joining visual and textual representations (orange)

4.4.2 Quantitative Results

After having an intuitive idea of how our models are working using the validation curves, we have generated the predictions for the *test-dev* split dataset and evaluated them in EvalAI. The results obtained are shown in Table 4.1.

Model	Yes/No	Number	Other	Overall
Baseline (KCNN) (Figure 3.2)	66.05	29.77	20.35	40.25
Language-Only (Figure 3.3)	66.15	31.17	24.87	42.59
Language-Only + Fine-Tune GloVe (Figure 3.3)	66.14	30.42	24.47	42.31
VGG + elem-wise product (Figure 3.4)	66.59	31.01	**25.83**	43.21
VGG + GloVe + elem-wise product (Figure 3.5)	**67.10**	**31.54**	25.46	**43.30**
ResNet + elem-wise product (Experiment 1 Figure 3.7)	64.97	30.22	22.31	40.78
ResNet + elem-wise product (Experiment 2 Figure 3.7)	65.51	29.99	22.02	40.84
ResNet + elem-wise sum (Figure 3.9)	65.57	30.20	22.80	41.26
ResNet + elem-wise sum + LR decay (Figure 3.9	65.90	30.00	22.80	41.37
ResNet + concat + FC (Figure 3.10)	64.66	29.49	21.27	40.08

Table 4.1: Results for different models in the test-dev split

The most remarkable conclusion obtained from the results is that the models tested learn mostly from language biases, as the Language-Only model obtains a 42.59 % of accuracy, while the VGG-based with GloVe and element-wise product, which has been the most successful method, obtains a 43.30%.

Another point to highlight is the bad performance of omitting the spatial information, as the features obtained from both KCNN and ResNet have sizes of 1024 and 2048 respectively and got the worst accuracies -even worse than the language-only model-, while the obtained from VGG, our best model, have a size of 7x7x512. Looking at the results, it is clear that using average pooling before joining image and textual representation does not work for our task. That does not mean that VGG is the optimum CNN to use, probably we would have got better results extracting features from a previous layer of ResNet with spatial information.

Furthermore, we can see how using GloVe or a learnable word embedding layer has a tiny impact, performing a 0.09% better the first ones. Looking at the validation curves we may think that this difference is too low, but as the singularities of the evaluation metric can provoke "wrong" answers to obtain higher accuracies than 0% (look at Figure 4.9 for some examples). However, we see how fine-tuning this layer does not suppose any benefit.

The same could be said about setting a decay for the learning rate (mentioned as LR at table 4.1), which in the case of the ResNet based model performed a 0.11% better. No decay was applied to the VGG based model because the validation loss decreased epoch by epoch.

The method in Figure 3.5 was the one presented to the VQA Challenge and obtained a 43'48% of overall accuracy in the test split, a 31.38% in the *number* category, a 25.81% in the *other* and a 66.97% in the *yes/no*.

4.4.3 Qualitative Results

In this section we will present some qualitative results for our best model, obtained from the validation split as we don't have available the test questions' ground truth answers. We have also calculated the associated accuracy for all these examples, according to Equation 4.1.

As we can see in the different samples from Figure 4.9, the conceptual range of the questions in VQA 2.0. Dataset is huge, forcing the model to perform tasks such us object recognition, sentiment analysis, activity detection, attribute identification, etc.

From these examples we can try to extract some conclusions about the dataset and the models we have tested. For instance, in the upper left example of Figure 4.9 we see how the model answers the object it recognizes. We could think that the joining operation of the question and the image representations is not working correctly, at least for this sample, as it seems to be answering what would predict the ResNet model for an object recognition task. Nevertheless, when giving as input the question *"What are the people in the background doing?"* for the same image, it answers correctly that they are *"watching"*. The closeness in semantic meaning between words like *looking* and *watching* can also make us think that are the language biases which are being exploded, assuming that when someone makes a question related to this action (look, watch or similar) they expect the answer to be the main character of the scene. And the same could be happening in the middle bottom example of Figure 4.9, as animals lay down most frequently when they are tired and they want to rest.

In order to analyze this with more detail we have compared some complementary samples, which are images with the same question associated but with different answer for it (Section 3.1). From the examples shown in Figure 4.8 we see how our model struggles with those questions which require more comprehension from the image, like the upper left example, in which we must

relate the object we detect with an attribute (being acidic or not).

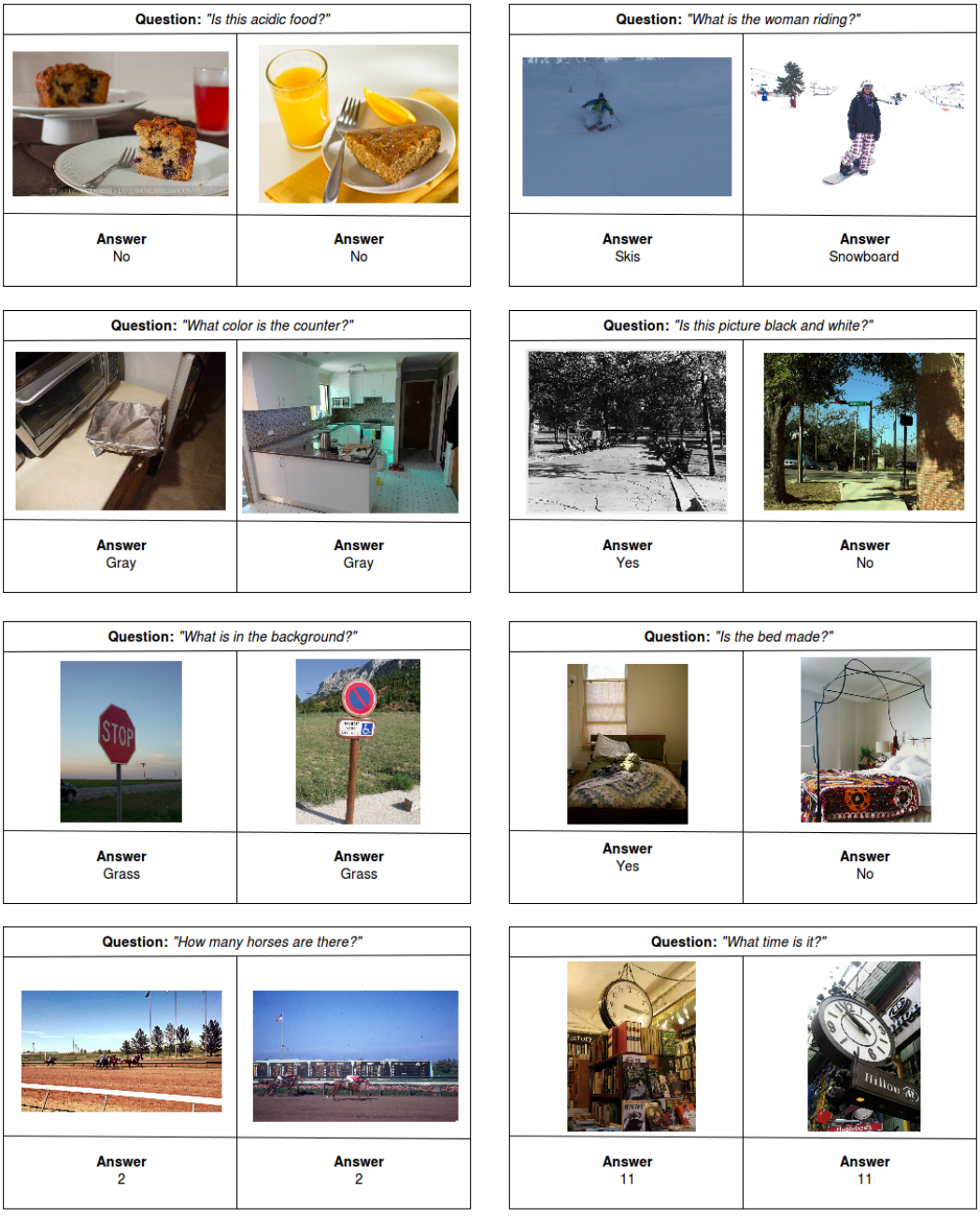

Figure 4.8: Predictions from random complementary samples

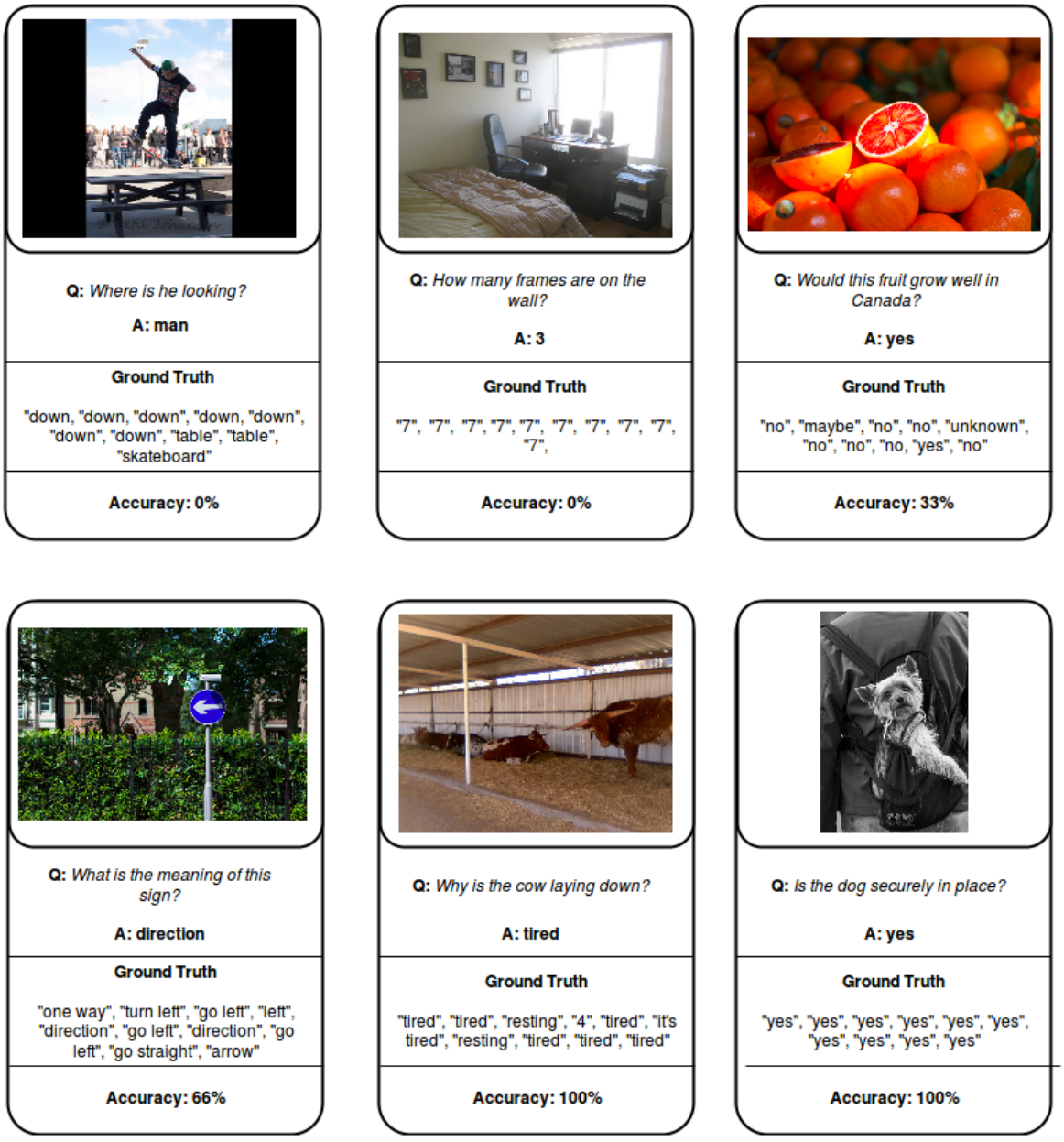

Figure 4.9: Results obtained from random samples using Equation 4.1

Chapter 5

Budget

This thesis has been developed without any aim to create any kind of product to be sold, so there will not be any analysis on this matters. As we have used the resources available at the Image Processing Group (GPI) at UPC there has been no real cost for the hardware needed. However, we can make an approximation about the cost we would have had if those resources were not provided by the research group looking at the prices offered by Amazon Web Services (AWS) at their cloud computing service.

The resources used for this project are a GPU with 12GB of RAM and about 40GB of regular RAM (Section 4.1). Taking this under consideration, the most similar EC2 instance on AWS to the resources used is the *p2.xlarge*, which gives us a GPU with 12 GB of RAM and a CPU with 61GB of RAM, having a cost of 0'9 $ per hour, which is equivalent to 21.60 $ per day. As we have used those resources about 100 days approximately, the cost ascends to 2160.00 $, which is equivalent to approximately 1940€ (currency exchange rate at 20/06/2017, when 1€= 1'11 $).

Apart from this, the only other cost that may be considered is the wage of the engineers working on the project, as all the software used is open-source and don't suppose any cost. The salary costs, considering that the length of the project has been of 24 weeks (Section 1.4.2), amounts to 18000€ (Table 5.1).

Therefore, we can estimate that the total cost of this project would be **19940€**.

	Wage/hour	**Work Time**	**Total**
Junior engineer	10 €/h	30 h/week	7200 €
Senior engineer	50 €/h	3 h/week	3600 €
Senior engineer	50 €/h	3 h/week	3600 €
Senior engineer	50 €/h	3 h/week	3600 €
		Total	18000 €

Table 5.1: Wages of the project participants

Chapter 6

Conclusions

When starting this project our main goal was to explore different deep learning ideas in order to improve UPC's last year submission to the VQA Challenge. Derived from this, we wanted also to present our results to the second edition of that challenge. Moreover, we wanted to build a software using best practices in order to make it reusable for future work.

Regarding the first objective we can say we have accomplished it, as we have improved last year model's performance in the current dataset by 3.05%. However, this improve is not huge, fact that remarks the need of working with bigger visual feature maps and/or fine-tuning the image branch, which was impossible for us due to computation constraints. We presented our model to the VQA Challenge and obtained a 43.48 % of accuracy, although at writing time the challenge is still active.

After looking at the results obtained by our model and comparing with last year winers of the challenge, model that obtains a 62.27% of accuracy in the current dataset (Section 2.1), it also seems obvious the need of adding attention to the model. This task remains to be done in the near future. Besides, we are also planning to add a better language model, as LSTM's are good just learning sequences. Instead, combining CNN's and LSTM's in the question branch we could create a more semantic-syntactic aware textual feature extractor.

Nonetheless, the latest research that has recently emerged about Visual Reasoning has attracted a lot of our attention. At the very beginning of the thesis, Facebook AI Research [1], together with Stanford University, announced a new Visual Question-Answering dataset, named CLEVR [24], which tests a range of visual reasoning abilities. We pretended to test our model in that dataset, but they did not release it until the end of the thesis.

The usage of modular networks, like in [15] or [2], implements much more accurately the reasoning steps that humans do when we are asked about an image. This is why, with the aim to create models able to reason, we want to try this new way of designing VQA systems and start working with CLEVR dataset.

[1] https://research.fb.com/category/facebook-ai-research-fair/

Bibliography

[1] Daylen Yang Anna Rohrbach Trevor Darrell Akira Fukui, Dong Huk Park and Marcus Rohrbach. Multimodal compact bilinear pooling for visual question answering and visual grounding. In *Conference on Empirical Methods in Natural Language Processing*, 2016.

[2] Laurens van der Maaten Judy Hoffman Li Fei-Fei C. Lawrence Zitnick Justin Johnson, Bharath Hariharan and Ross Girshick. Inferring and executing programs for visual reasoning. *arXiv preprint arXiv:1705.03633*, 2017.

[3] Jiasen Lu Margaret Mitchell Dhruv Batra C.Lawrence Zitnick Stanislaw Antol, Aishwarya Agrawal and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2425–2433, 2015.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. In *Neural computation*, 1997.

[5] Sainbayar Sukhbaatar Arthur Szlam Bolei Zhou, Yuandong Tian and Rob Fergus. Simple baseline for visual question answering. *arXiv preprint arXiv:1512.02167*, 2015.

[6] Issey Masuda, Santiago Pascual, and Xavier Giro. Open-ended visual question-answering. *arXiv preprint arXiv:1610.02692*, 2016.

[7] Douglas Summers-Stay Dhruv Batra Yash Goyal, Tejas Khot and Devi Parikh. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. In *Conference Computer Vision and Pattern Recognition*, 2017.

[8] Zhen Liu. Kernelized deep convolutional neural network for describing complex images. *arXiv preprint arXiv:1509.04581*, 2015.

[9] Richard Socher-Li-Jia Li Kai Li Jia Deng, Wei Dong and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Conference Computer Vision and Pattern Recognition*, 2009.

[10] Ryan Kiros-Aaron Courville Ruslan Salakhutdinov Richard Zemel Kelvin Xu, Jimmy Ba and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the International Conference on Machine Learning*, 2015.

[11] Jianfeng Gao-Li Deng Zichao Yang, Xiaodong He and Alex Smola. Stacked attention networks for image question answering. *arXiv preprint arXiv:1511.02274*, 2015.

[12] Huijuan Xu and Kate Saenko. Ask, attend and answer: Exploring question-guided spatial attention for visual question answering. In *Proceedings of the European Conference on Computer Vision*, 2016.

[13] Kevin Chen Moses Charikar and Martin Farach-Colton. Finding frequent items in data stream. In *Automata, languages and programming*, pages 693–703, 2002.

[14] Woosang Lim Jeonghee Kim Jung-Woo Ha Jin-Hwa Kim, Kyoung-Woon On and Byoung-Tak Zhang. Hadamard product for low-rank bilinear pooling. In *International Conference on Learning Representations*, 2017.

[15] Trevor Darrell Jacob Andreas, Marcus Rohrbach and Dan Klein. Neural module networks. In *Conference Computer Vision and Pattern Recognition*, 2016.

[16] Quoc V. Le Ilya Sutskever, Oriol Vinyals. Sequence to sequence learning with neural networks. In *Conference Neural Information Processing Systems*, 2014.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[18] Richard Socher Jeffrey Pennington and Christopher D. Manning. Glove: Global vectors for word representation. In *Conference on Empirical Methods in Natural Language Processing*, 2014.

[19] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[20] Jia Deng Hao Su Jonathan Krause Sanjeev Satheesh Sean Ma Zhiheng Huang et al. Russakovsky, Olga. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 2015.

[21] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, volume 37, pages 448–456, 2015.

[22] Alex Krizhevsky Ilya Sutskever Nitish Srivastava, Geoffrey Hinton and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15:1929–1958, 2014.

[23] Shaoqing Ren Kaiming He, Xiangyu Zhang and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*, 2016.

[24] Laurens van der Maaten Li Fei-Fei C. Lawrence Zitnick Ross Girshick Justin Johnson, Bharath Hariharan. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. *arXiv preprint arXiv:1612.06890*, 2016.