

Towards Open-Ended VQA Models Using Transformers

BY

ALBERTO MARIO BELLINI

B.S., Politecnico di Milano, Milan, Italy, July 2017

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2020

Chicago, Illinois

Defense Committee:

Natalie Parde, Chair and Advisor

Barbara Di Eugenio

Wei Tang

Pier Luca Lanzi, Politecnico di Milano

...

ACKNOWLEDGMENTS

First and foremost, I would like to sincerely thank my advisors Prof. Natalie Parde, Prof. Mark James Carman, and Prof. Matteo Matteucci. Their contribution to this work is priceless, and, without their help, I would still be stuck at organizing ideas. Not only have they supported me throughout this long journey with their expertise and knowledge, but they even managed to share their passion and love to work in this area of research.

I would genuinely love to thank my whole family, who provided me with all the means to work on this project by continually supporting me and my ideas. I feel privileged to have had the opportunity to focus on my thesis without worrying about anything, having always had them backing me up.

A special “thank you” goes to all my friends that always believed in me and in my objective, in particular to my great friend and colleague, Gianpaolo, who helped me on countless occasions by listening to my doubts and suggesting new ideas.

Finally, I would like to thank all the special people that I met in my life, who contributed to make me the man who I am today; even though we parted ways, I will always be thankful to you all.

AMB

TABLE OF CONTENTS

<u>CHAPTER</u>		<u>PAGE</u>
1	INTRODUCTION AND MOTIVATION	1
1.1	VQA for Social Good	3
1.2	A different approach	4
1.3	Outline	4
2	BACKGROUND	5
2.1	Visual Question Answering	5
2.2	Natural Language Processing	7
2.2.1	NLP in VQA Systems	9
2.2.2	Preprocessing Pipeline	9
2.2.2.1	Tokenization	10
2.2.2.2	Encoding	14
2.2.2.2.1	Dictionary Lookup encoding	14
2.2.2.2.2	One Hot encoding	15
2.2.2.2.3	Word Embedding	17
2.3	Computer Vision	20
2.3.1	Computer Vision in VQA Systems	22
2.4	Conclusion	22
3	COMPUTATIONAL MODELS	24
3.1	Computational models for text	24
3.1.1	Recurrent Neural Networks	25
3.1.1.1	Limitations	29
3.1.1.2	Long-Short Term Memory Networks	29
3.1.2	The Transformer	34
3.1.2.1	Model Architecture	36
3.1.2.2	Attention	38
3.1.2.3	Multi-Head Attention	40
3.1.2.4	Position-Wise Feed-Forward Networks	42
3.1.2.5	Positional Encoding	43
3.1.3	Generative Pre-trained Transformer (GPT-2)	43
3.1.4	Bidirectional Encoder Representation from Transformers (BERT)	45
3.1.5	Discussion	46
3.2	Computational models for images	47
3.2.1	Convolutional Neural Networks	47
3.2.1.1	Convolutions	49
3.2.1.2	Non-Linear activations	51

TABLE OF CONTENTS (continued)

<u>CHAPTER</u>		<u>PAGE</u>
	3.2.1.3 Pooling	52
	3.2.1.4 The classifier	52
	3.2.2 VGGNet	52
	3.3 Conclusion	53
4	RELATED WORKS	54
	4.1 Neural Module Networks	56
	4.2 Hierarchical Co-attention	60
	4.3 Multimodal Compact Bilinear Pooling	61
	4.4 Conclusion	63
5	DATASET AND IMPLEMENTATION	64
	5.1 Dataset	64
	5.1.1 VQAv2 Dataset	66
	5.1.2 Preprocessing	69
	5.2 Baselines	73
	5.2.1 Captioning Baseline	73
	5.2.1.1 Model architecture	73
	5.2.1.2 Dataset	75
	5.2.1.3 Training	77
	5.2.2 GPT-2 Answering Baseline	78
	5.2.2.1 Architecture	78
	5.2.2.2 Dataset	79
	5.2.2.3 Training	80
	5.2.3 BERT Answering Baseline	82
	5.2.3.1 Architecture	82
	5.2.3.2 Dataset	83
	5.2.3.3 Training	85
	5.2.4 VQA Baseline	85
	5.2.4.1 Architecture	86
	5.2.5 Dataset	88
	5.3 Proposed architecture - VGGPT-2	89
	5.3.1 Architecture	89
	5.3.2 Image Encoder	91
	5.3.3 Language Model	92
	5.3.4 Attention	93
	5.3.5 Answer Generator	97
	5.3.6 Dataset	100
	5.3.7 Training	101
6	EVALUATION	103
	6.1 Answer Generation	103

TABLE OF CONTENTS (continued)

<u>CHAPTER</u>		<u>PAGE</u>
6.2	Quantitative evaluation	104
6.2.1	Accuracy	105
6.2.2	Bilingual Evaluation	109
6.2.3	Word Mover’s Distance	116
6.2.4	Remarks on BERT	120
6.3	Qualitative evaluation	120
6.3.1	Classification performances	122
6.3.2	Reasoning performances	125
6.3.3	Generation performances	128
6.3.4	Conclusions	128
7	CONCLUSIONS	131
7.1	Future work	131
	CITED LITERATURE	133
	APPENDICES	138
	Appendix A	139
	Appendix B	140
	VITA	141

LIST OF TABLES

<u>TABLE</u>		<u>PAGE</u>
I	SAMPLE DICTIONARY LOOKUP ENCODING TABLE	15
II	EXAMPLE OF A ONE-HOT VECTOR ENCODING TABLE	16
III	EXAMPLE OF AN EMBEDDING TABLE	20
IV	VQA DATASET STRUCTURE	67
V	CAPTIONING BASELINE DATASET STRUCTURE.	77
VI	GPT-2 BASELINE DATASET STRUCTURE.	81
VII	BERT BASELINE DATASET STRUCTURE.	84
VIII	STRUCTURE OF VGGNET-11 WITHOUT FINAL CLASSIFIER. .	92
IX	BASELINE DATASETS VS VGGPT-2 DATASET.	101
X	VARIATION OF N-GRAM WEIGHTS IN BLEU METRIC.	111
XI	BLEU SCORES USING ADD-1 SMOOTHING FUNCTION.	113
XIII	QUESTION : “ WHAT DO YOU SEE? ”.	123
XV	QUESTION : “ WHAT COLOR IS THE TRAIN? ”.	124
XVII	QUESTION : “ WHAT ARE THE ANIMALS DOING? ”.	126
XIX	QUESTION : “ WHAT IS IN THE PLATE? ”.	127
XXI	EXAMPLE IN WHICH VGGPT-2 GENERATES BOTH A QUES- TION AND A CORRECT ANSWER.	129
XXIII	EXAMPLE IN WHICH VGGPT-2 GENERATES A CAPTION FOR THE IMAGE.	130

LIST OF FIGURES

<u>FIGURE</u>		<u>PAGE</u>
1	Common structure of a VQA System	6
2	Embeddings for three tokens in \mathbb{R}^3	19
3	Results of an object detection system in action.	21
4	Recursion of hidden states in a RNN.	26
5	Base structure of a RNN.	27
6	Recursion of hidden states in a Bidirectional RNN.	28
7	Standard RNN activation.	30
8	An LSTM cell.	31
9	LSTM activation functions.	32
10	Transformer Encoder-Decoder stack.	37
11	Single head attention example.	40
12	Multiple head attention example.	42
13	General structure of a CNN.	48
14	Examples of convolutions	50
15	Neural Module Network structure.	58
16	Hierarchical Co-attention.	61
17	VQAv2 complementary image pair.	68
18	VQAv2 distribution example.	69
19	Captioning model structure.	74
20	VQA-Baseline Model structure.	86
21	VGGPT-2 Model structure.	90
22	VGGPT-2 Attention Mechanism.	95

LIST OF FIGURES (continued)

<u>FIGURE</u>		<u>PAGE</u>
23	VGGPT-2 Answer Generation.	99
24	VGGPT-2 Train Loss.	102
25	Evaluation Accuracy.	105
26	Accuracy comparison.	107
27	Best model accuracy.	108
28	Evaluation BLEU.	112
29	Evaluation Lengths.	114
30	Number of comparable results in WMD.	118
31	Evaluation Word Mover’s Distance.	119
32	Web Interface.	122
33	VGGPT-2 softmaps associated with Table XIII.	123
34	VGGPT-2 softmaps associated with Table XV.	124
35	VGGPT-2 softmaps associated with Table XVII.	126
36	VGGPT-2 softmaps associated with Table XIX.	127
37	VGGPT-2 softmaps associated with Table XXI.	129
38	VGGPT-2 softmaps associated with Table XXIII.	130

LIST OF ABBREVIATIONS

VQA	Visual Question Answering
NLP	Natural Language Processing
NLU	Natural Language Understanding
NLG	Natural Language Generation
OOV	Out Of Vocabulary
BPE	Byte Pair Encoding
NN	Neural Network
FFNN	Feed Forward Neural Network
DNN	Deep Neural Network
MLP	Multi-Layer Perceptron
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
NMN	Neural Module Network
LSTM	Long-Short Term Memory
MT	Machine Translation
CPU	Central Processing Unit
GPU	Graphics Processing Unit

LIST OF ABBREVIATIONS (continued)

TPU	Tensor Processing Unit
MCB	Multimodal Compact Bilinear Pooling
BLEU	Bilingual Evaluation Understudy
WMD	Word Mover’s Distance

SUMMARY

In this work, we introduce a new architecture to address the Visual Question Answering problem, an open field of research in the NLP and Vision community.

In the last few years, with the advent of Deep Learning and the exponential growth of computing power, researches came up with brilliant solutions to tackle the problem.

However, most of the related work share a standard limitation: the number of possible answers is usually restricted to a limited set of candidates, limiting the power of such models.

In this work, we describe a new architecture that employs new state-of-the-art language models, such as the Transformer, to generate open-ended answers. In the end, our contribution to the scientific community lies in a new approach that allows VQA systems to generate unconstrained answers.

First, we introduce the necessary background as well as the most critical computational models to deal with text and images. Ultimately, we show that our architecture compares well with other VQA models, setting a new baseline for future work.

CHAPTER 1

INTRODUCTION AND MOTIVATION

In the last couple of years, we have observed outstanding results on a wide variety of tasks in the Artificial Intelligence and Machine Learning fields. Such achievements have led to incredible advancements in several different areas of research. They are slowly becoming part of our everyday life by being used in devices or systems that we access regularly.

Greatly inspired by the passion for these topics, the results and the huge steps that technology took in the past years, allowing us to process enormous amounts of data, we decided to step into the well known Visual Question Answering research area with the goal of improving the current state-of-the-art from certain points of view, such as overcoming the short and concise answers usually generated by most VQA systems.

Visual Question Answering is an open, multidisciplinary field of research that combines Vision, Natural Language Understanding, and Deep Learning to answer open-domain questions about images.

This task is challenging since it involves both the understanding of what is being asked (i.e., the question, asked in natural language) and the reasoning on the associated image to seek relevant information. Assuming the previous two steps provide enough context, a VQA system should then be capable of generating meaningful answers in natural language, preserving semantic and syntactic correctness.

Even though the task might seem to be trivial for a human being, this is not always the case. A question asked in the wrong way or an image shot with the wrong angle (or even both things together) might confuse even a human being, resulting in poor answers.

This highlights the intrinsic complexity of this problem and the challenges associated with it.

Current state-of-the-art architectures achieve excellent results and can generalize reasonably well on different question-image pairs. These models exhibit significant accuracies on the datasets they were trained on. Still, the majority of them share a standard limitation: the generated answers tend to be short and concise, with almost no context.

While a typical human being’s behavior tends to enrich his answers, providing longer sentences with punctuation and arguments, these systems seem to lack this ability and tend to output single labels. These shortcomings are usually correlated with how these models were trained and on the architectures that have been implemented. For instance, whenever the problem is tackled as a classification task, the answers are forced to lie within a limited set of labels, thus reducing performances and generalization capabilities.

Motivated by the great benefits that Visual Question Answering can bring to the lives of many people, we try to address the discussed limitations with a different approach. Our main goal is to generate open domain answers in natural language whose lengths are not limited to any size but depend on the context.

As highlighted before, traditional models tend to give short answers like “yes”, “red”, “5”. It is clear that even though these systems seem to have learned how to reason on the inputs,

and perform well within a set of candidate question-answer pairs, they appear to be still limited and not ready for real-world scenarios. We want to overcome this "laziness" in favor of longer and more meaningful answers, where specific words are justified by particular captions. We do believe that aside from the ground truth answer, whether it be a single label or a small sequence of words, it would be great to have some extra context that might describe, for instance, why that label was chosen as the answer.

We hope that this piece of work will help future research in focusing more and more on the richness and quality of the answers instead of bare ground truth matching. We believe that the whole point of VQA is not just answering with single words but coming up with systems that, eventually, will act like real human beings.

1.1 VQA for Social Good

An active driver for this research project has been the will to help the research community in coming up with a system that someday might be integrated into tools for blind people. Even though most of us can use their own eyes to perceive the surrounding environment better, many people are forced to use alternative means. Visual Question Answering systems, if tuned to a point where they can generalize well and provide rich answers, could result in being a real game-changer for the blind community.

With the help of such tools, these people might eventually gain back more independence, without the constant need of somebody else's eyes when all the means they have fail to help them.

This is probably one of the most exciting applications for VQA systems since it is one of those scenarios where Deep Learning meets with Healthcare.

1.2 A different approach

In 2017 the Google Brain team came up with a brand new architecture: the Transformer [1]. This is a new, attention-based, model that represents the new state-of-the-art in several different Natural Language Processing tasks. Amazed by the stunning capabilities of this architecture, especially when it comes to language generation, we were extremely motivated to adapt it for the VQA task, confident that its flexibility and power would have benefit the open domain nature of the answers we were trying to generate.

1.3 Outline

Going next, chapter 2 will provide all the background required to understand the related work and the implementations themselves.

Chapter 3 will go over some of the most significant computational models that will be at the core of our proposed baselines and implementation, such as common architectures used to deal with text and images.

All the related work will be introduced in chapter 4, where we will present what the scientific community has come up with to address the VQA problem, with all the associated limitations.

In chapter 5 we will discuss the corpus that we use, how we process it, and all the baselines. Finally, we will introduce our architecture and, in chapter 6, we will go through an in-depth evaluation process, both under a qualitative and quantitative point of view.

CHAPTER 2

BACKGROUND

In this chapter, we introduce the background required to proceed further in the reading. Concepts such as what is Natural Language Processing, Computer Vision, and what are they used for are going to be at the heart of this chapter.

2.1 Visual Question Answering

As already mentioned, VQA is a multidisciplinary task that sees two vast areas of research, namely Natural Language Processing and Computer Vision, working together and sharing knowledge to pursue a common objective. It is not the only case in which these two disciplines are put side by side; Image Captioning systems, for instance, try to achieve a very similar result and the only difference with respect to what is fed to a VQA system is the lack of a question.

If we had to tear apart the main components that make up a VQA system, we could schematize them as depicted in Figure 1:

- The **Inputs**: This is what goes into the model, in Figure 1 we can see a picture of an elephant and the question “What animal is it?”
- **Encoders**: The encoders are high-level components that are capable of extracting features from both the image and the question. They usually output vectors that project the information contained in the inputs in a higher-dimensional space.

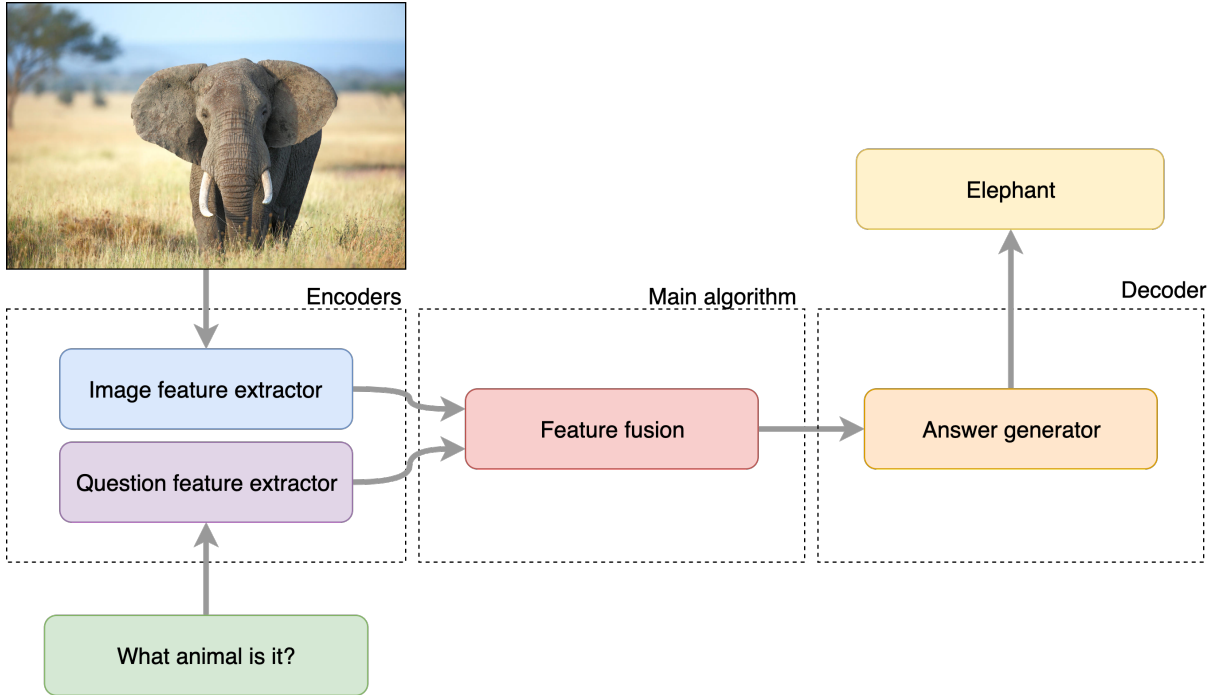


Figure 1: Common structure of a VQA system, image taken from [2]

- The **Main algorithm** accounts for merging the extracted features and projecting them in a common space where the information coming from the textual modality is combined with the visual one in a single block. Usually, an attention mechanism is employed here to use the question as a driver to focus on specific parts of the image.
- A **Decoder** finally takes care of generating the answer exploiting the information that comes from the previous processing steps.

- The **output** is a sequence of words that represents the answer, in this case, “elephant”.

Depending on the model in use, the output size may vary, but most models tend to keep it quite restricted, thus highlighting the need for open-domain approaches.

How these components are integrated and implemented profoundly depends on the model we are considering. Related works will show various topologies that try to maximize the performance of VQA systems, and even though they pursue the same objective, they differ a lot in their structure.

Before getting into details, we will now introduce some concepts of NLP and Computer Vision that will help to understand what these systems are really about.

2.2 Natural Language Processing

Natural Language Processing, commonly referred to as NLP, is a field of research that originated in the 1950s. It combines different methodologies from linguistics, artificial intelligence, and computer science to study the interaction between computers and humans, focusing its attention mainly on how to process and analyze large amounts of data written in natural language. In the past years, this sector has taken huge steps forward due to both the advancements made in technology, that allowed machines to process larger and larger chunks of data, and those ones in the Deep Learning field, where many of the tasks that were previously performed by statistical models experienced outstanding results using neural networks.

NLP is a vast area of research that consists of many sub-fields, such as Natural Language Understanding, Natural Language Generation, and down-stream tasks from Speech Recognition. They all deal with natural language but exhibit some differences in the ultimate objective.

Speech Recognition systems, for instance, are becoming popular in these years. The vocal assistants integrated into many of the most recent smartphones can process our voice and translate it to text just because there are speech recognition models under the hood. This is a very complicated task: the system has to convert an electric signal coming from a microphone to a sequence of words that ultimately represents what we said. From a high-level point of view, these systems sample the spectrum of our voice at a specific rate and extract features using techniques such as the Mel-Frequency Cepstral Coefficients [3] extractor. These features are then converted to words that will form the final recognized sequence.

One might argue that the latter systems perform Natural Language Understanding operations as well, and that is indeed partially true. Translating a vocal signal into a sequence of words underlines that the model in question understood what was initially recorded with the microphone. However, the concept of understanding is definitely broader and, most of the times, computers are just executing algorithms that are mapping something to something else using a specific methodology and without really caring about what is coming in input or what is leaving in output.

Even though there are contrasting opinions about whether or not a machine will ever be able to understand what it is fed with, recent work in the community has provided astonishing results in almost every task. Google Translate [4] is probably one of the most evident examples of how well an algorithm is capable of “understanding” and translating one language to another one; it performs so well that it has more than 200 million daily users and supports 104 different languages.

2.2.1 NLP in VQA Systems

A sound Visual Question Answering system should exhibit high capabilities in both understanding what the question is all about and, of course, generating a meaningful and correct answer. For this reason, NLU and NLG are two critical aspects of such models, and, in order to achieve good results, they should both be taken into account. First off, the model should be able to process the inputs and generate outputs. Of course, a machine does not speak the same language of a human, and, for this reason, there is an initial procedure that converts the inputs to the language domain of the computer and vice versa. This is common not only to VQA models but to almost every existing NLP system. In this case, the input is represented by a sequence (a question) and the image. Since, in this section, we are talking about NLP, we will focus our attention only on the question and will talk about the visual modality later on.

As mentioned above, we first need a way to convert this sequence to something understandable by a machine. This because the brain of the computer, the Central Processing Unit or CPU, doesn't know what a sequence of words is. It is just a complex circuit that performs billions of operations every second, like sums, multiplications, and many more. Fortunately, the community has come up with really smart ways of translating these sequences into something that can be processed by a machine through what is called a preprocessing pipeline.

2.2.2 Preprocessing Pipeline

A preprocessing pipeline is a sequential set of operations that are gradually applied to a sequence of words. Its goal is to translate a sentence in natural language to a vector of numbers.

Ultimately this vector will be the input to the considered model, which in turn will be able to process and make computations using the information contained in it.

The traditional pipeline consists of many steps, such as word tokenization, stemming, lemmatization, punctuation removal, normalization, augmentation, and finally encoding. However, in the next sections, we will address only the most relevant ones for this project.

2.2.2.1 Tokenization

The first and foremost step is the tokenization of the input sequence and consists of chopping it into pieces, called tokens. This is necessary in order to proceed in the preprocessing where every token can be analyzed and processed separately.

The tokenization can occur at diverse levels, and we will first focus our attention on the most common and simple way to turn a sequence of words into a list of tokens, namely, word-level tokenization. The idea is simple: given a sentence in natural language, an algorithm scans it sequentially and converts every word it encounters into a token. For instance, given the sentence:

I like playing football.

It will be tokenized as a list of tokens, where each token is a word:

[“I”, “like”, “playing”, “football”, “.”]

Depending on the tokenizer in use and on the task, punctuation might be thrown away. In this case, the dot becomes a unique token, even though this is not always true. After this procedure, it is possible to create a vocabulary of words. The words contained within this vocabulary are those that the model will be trained on. Depending on the size of the

vocabulary, the complexity of the system can increase or decrease; it rarely exceeds 50K-60K words, for computational limitations. In order to account for the Out Of Vocabulary words (OOV) special tokens are introduced, such as the `<UNK>` token. The latter is used whenever a word outside the pre-computed vocabulary is found in order to allow the model to generalize and deal with sequences of any kind. Of course, the smaller the vocabularies, the less powerful the final model will be since the majority of the input tokens would be treated as unknowns.

However, it is clear that this tokenization technique doesn't scale well and can lead to extremely large vocabularies, making the problem tough to tackle. Fortunately, there is a much more clever technique called Byte Pair Encoding (BPE) [5] that exploits a different technique of tokenization and leads to smaller vocabularies using a compression mechanism. To get familiar with the concept, consider the following words: great, greatest, smart, smartest.

The traditional word-level tokenization mechanism would have led to 4 different tokens since all the words are different. BPE instead would have come up with only three tokens, namely: great, smart, est

It is evident that in the end the size of the vocabulary will be smaller. BPE is a powerful compression technique that will be at the core of the tokenization step in the GPT-2 [6] architecture that we will discuss in the next chapters, and for this reason we will now explain how it works.

- First, we count how many times each word appears in the corpus we have, and we create a structure where we append to each word a special stop token `</w>` with the relative

word frequency. Then we split each word into characters plus the additional $\langle /w \rangle$ token.

For example, the result could look like this:

```

g r e a t  $\langle /w \rangle$ : 5
g r e a t e s t  $\langle /w \rangle$ : 6
s m a r t  $\langle /w \rangle$ : 2
s m a r t e s t  $\langle /w \rangle$ : 3

```

This means that in our corpus we found 5 times the word “great”, 6 times “greatest” and so on.

- Iteratively, we count the frequency of every consecutive byte pair and find the most frequent one. Afterward, we merge the two bytes together to form a single, new, token. For the above example, at the first iteration, it results that the bytes “e” and “s” form the most frequent pair “es”, with relative frequency $6+3=9$. At this point, we save the new “es” token to the vocabulary and proceed by replacing the pairs in the words as follows:

```

g r e a t  $\langle /w \rangle$ : 5
g r e a t e s t  $\langle /w \rangle$ : 6
s m a r t  $\langle /w \rangle$ : 2
s m a r t e s t  $\langle /w \rangle$ : 3

```

Note that these tokens are not yet definitive and might be removed by the following iterations.

- At the second iteration, “est” will be the next most frequent pair composed by “es” and “t” with relative frequency once more equal to 9. The merging and replacing operations are performed once again, and the procedure goes on until we reach a specific threshold, such as a maximum vocabulary size.
- Once the threshold is reached, our vocabulary will contain all the computed tokens, and each sequence of words will be tokenized accordingly. For instance, here you can see how the following sentence could be tokenized using BPE on a fake corpus:

Original sentence: I like playing football.

Intermediate tokenization: [“I”, “like”, “playing”, “football”, “.”]

BPE tokenization: [“I”, “like”, “play”, “ing”, “foot”, “ball”, “.”]

- The stop token is important since it helps during the tokenization procedure. Given the token “est</w>” we will know that it won’t belong to the word, say, “estimation” and we will avoid tokenizing it as [“est”, “imation”]. It will indeed help us with words that end with the same pattern, such as “smartest”, which will be tokenized as [“smart”, “est”].

The resulting tokenized string might be longer, but the vocabulary is usually smaller. Furthermore, in this way, we can help the model learn some language patterns. Take, for instance, the adjective “great”. If we do not use BPE the model might be able to understand its meaning and even those of “greater” and “greatest” just because it could have been trained on a corpus with these tokens. However, it might be unable to generalize well on unseen adjectives such as “taller” because, during the training procedure, he saw only the token “tall”.

BPE can lead a system to the understanding of the relationships between an adjective and its comparative pairs. With this technique, it is very likely that tokens such as “er” and “est” will be present in the vocabulary; for this reason, a model that exploits BPE will usually generalize better, allowing it to understand what “taller” means even though he has only seen “tall”. This doesn’t apply only to adjectives but to every grammatical structure.

2.2.2.2 Encoding

Following tokenization, we find the encoding step: a key part of the preprocessing pipeline. Here each token gets converted into something that a model can understand: a number. There are several different ways of encoding a token, and here we will describe some of the most used in the NLP field.

2.2.2.2.1 Dictionary Lookup encoding

Dictionary Lookup encoding is one of the most simple and most used forms of encoding. It is even the most intuitive form to convert a token into a number because it simply assigns to each word a unique value. For instance, whenever we see the word “playing” we assign it the value 5, whereas “football” will be converted to 6. From a high-level point of view, the common approach is to iteratively build a dictionary (or map) while tokenizing the available corpus, assigning to each newly encountered word, not present in the dictionary, a different and unique number.

Technically speaking, the dictionary is a widely used structure in Data Science named Hash-Table, where the keys are represented by words, and the values are numbers. An example is reported in Table I.

TABLE I: SAMPLE DICTIONARY LOOKUP ENCODING TABLE

Word	Value
playing	5
football	6
.	7
I	8
like	9

With the latter lookup table (Table I), the following sentence will be encoded as follows, assuming we are not using BPE tokenization.

- Original sentence: I like playing football.
- Tokenized sentence: ["I", "like", "playing", "football", "."]
- Encoded sentence: [8, 9, 5, 6, 7]

The latter vector is ready to be fed to the considered model and carries all the information needed. Note that if we had used BPE the lookup table would have contained all the tokens returned by the algorithm, and the encoding would have taken place in the same manner.

2.2.2.2.2 One Hot encoding

One Hot encoding is another mechanism used to encode a sequence of words. Even though we are not using it in this project, for reasons that will become obvious while reading through, it was worth mentioning its existence as a comparative approach.

The idea behind One Hot encoding is to convert each word to a vector of all zeros except for the position which represents the considered word, where a one is set. Once more, there is a lookup table, but this time instead of a single value we find a sparse vector. Table II helps to visualize this concept:

TABLE II: EXAMPLE OF A ONE-HOT VECTOR ENCODING TABLE

Word	One Hot vector
playing	[1, 0, 0, 0, 0]
football	[0, 1, 0, 0, 0]
.	[0, 0, 1, 0, 0]
I	[0, 0, 0, 1, 0]
like	[0, 0, 0, 0, 1]

The latter lookup table is built on a vocabulary with five words in total, and you can see how each one maps to its one-hot encoded representation. If we had to encode the sentence from our usual considered example, it would look like this:

- Original sentence: I like playing football.
- Tokenized sentence: ["I", "like", "playing", "football", "."]
- Encoded sentence: [[0, 0, 0, 1, 0], [0, 0, 0, 0, 1], [1, 0, 0, 0, 0], [0, 1, 0, 0, 0], [0, 0, 1, 0, 0]]

This time every word is not represented by a single number, but by a vector. The final sequence will thus be represented by a vector of vectors, namely, a matrix.

Even though this method reduces the ordering bias present in the Dictionary Lookup encoding, it doesn't scale well since we come up with huge sparse matrices. For each word we append to our vocabulary, we observe every vector increasing its size by one. This means that if we have a vocabulary of, say, 50K words, we would have a sparse matrix of size 50000x50000. Assuming we are using 4 bytes to represent each number (int32) we would need 10GB of RAM just to hold in memory the lookup table.

2.2.2.2.3 Word Embedding

Word Embedding is one of the most sophisticated techniques to encode a token. Not only this method allows for less sparse inputs, but it even gives the ability to the model to reason on the semantics of the tokens.

Conceptually it consists of mapping each token to a real-valued vector of a predefined size that brings the token into a different multidimensional space in which every vector belongs to \mathbb{R}^n , where n is equal to the embedding size. In this space, similar words, in terms of semantics, such as “king” and “queen” are close one to the other one. Contrarily, words such as “snow” and “cow” will be very distant because their semantics differs almost from every point of view.

To understand if two tokens are similar, we look at the angle α that is formed between the two relative embeddings and, more specifically, we examine how close $\cos(\alpha)$ is to 1. This metrics is called cosine similarity. The closer the two vectors are to each other, the closer to one the cosine between the two will be. In trigonometry, the cosine is measured in radians.

The formula for the cosine similarity between two vectors a and b is the following:

$$\text{Cosine Similarity} = \cos(a, b) = \frac{ab}{\|a\| \|b\|} = \frac{\sum_{i=1}^n a_i b_i}{\sqrt{\sum_{i=1}^n (a_i)^2} \sqrt{\sum_{i=1}^n (b_i)^2}} \quad (2.1)$$

To better understand the concept, Figure 1 shows how three embeddings for the words “king”, “queen” and “snow” are displaced in \mathbb{R}^3 . As highlighted by the plot, the angle between the embedding of “king” and “queen” represented by ϕ is small, thus indicating a high cosine similarity. The two tokens are indeed semantically similar. Contrarily the embedding for “snow” lies far away from the other two and, for instance, the angle θ formed with “queen” is vast, resulting in a small cosine similarity.

An exciting feature that comes with word embeddings is the possibility of performing mathematical operations between these vectors. Briefly, if we take the embedding that represents “king”, we subtract the embedding of “man” and then add back again the one for “woman” we will get a vector that is almost identical to the embedding of “queen”.

Word embeddings are usually learned using methods such as Neural Networks, probabilistic models, or dimensionality reduction on the co-occurrence matrix. Word2Vec [7] and GloVe [8] are probably the two most commonly used forms of word embeddings, and they both exploit neural networks trained on large corpora. Once these models have been trained, the embeddings are obtained by tearing apart the network and dumping all the hidden hyperparameters for every word in the vocabulary. Afterward, these embeddings are released to the community as a pre-trained set that everyone can integrate into their models.

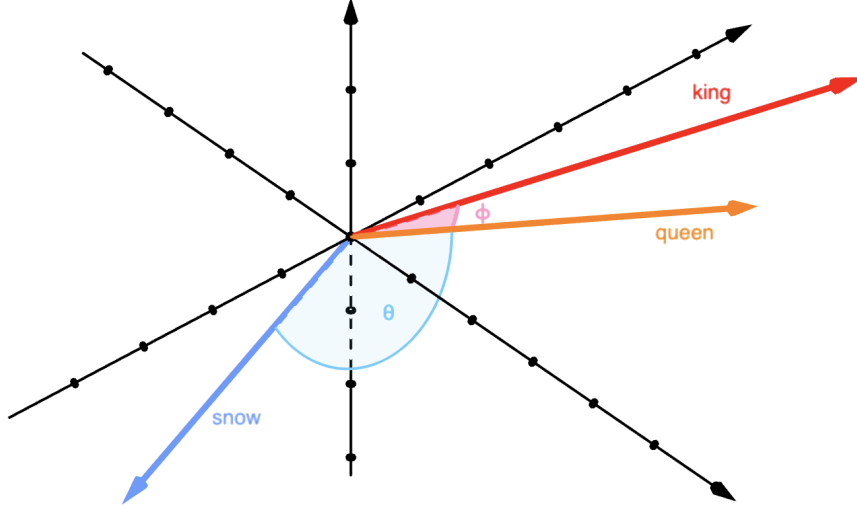


Figure 2: Embeddings for three tokens in \mathbb{R}^3

However, it is worth noticing that word embeddings might be learned end-to-end without taking them from pre-trained repositories. Of course, the convergence of the system will be much slower, and in most cases, will lead to lower performances.

For completeness, we report, in Table III, a word embedding encoding for our use case example. The size of the embeddings is set to 3 (\mathbb{R}^3), and the values have been randomly chosen, approximated to the fourth digit.

This is the resulting encoded sentence:

TABLE III: EXAMPLE OF AN EMBEDDING TABLE

Word	Embedding vector
playing	[0.1497, 0.3667, 0.0204]
football	[0.7123, 0.9967, 0.6225]
.	[0.5156, 0.8489, 0.4621]
I	[0.8605, 0.6216, 0.3802]
like	[0.5738, 0.822, 0.9366]

- Original sentence: I like playing football.
- Tokenized sentence: [“I”, “like”, “playing”, “football”, “.”]
- Encoded sentence: [[0.8605, 0.6216, 0.3802], [0.5738, 0.822, 0.9366], [0.1497, 0.3667, 0.0204], [0.7123, 0.9967, 0.6225], [0.5156, 0.8489, 0.4621]]

2.3 Computer Vision

Computer Vision is another vast area of research that tries to replicate all those tasks that a human visual system can achieve. While NLP tries to come up with methodologies to deal with natural language, the Computer Vision community is continuously seeking new techniques to acquire, process, and analyze images and videos to gain a high-level understanding of them. These high-level features usually contain spatial or semantic information about the input image or video, and are used by state of the art models to make an inference, such as telling us whether in a photo is depicted a zebra or a lion.



Figure 3: Results of an object detection system in action. Image created from a picture taken at [2]

Among the most critical tasks in Computer Vision, we find classification, object detection, segmentation, event detection, image restoration, and many more. Depending on the task, these systems are implemented adopting several different strategies, but Deep Learning is leading the way in many of them and has come up with a lot of with state of the art models. The power of nowadays Neural Networks to treat images is a breakthrough, and is widely adopted not only in research but even in our everyday life through sensors and smartphones, just to name a few.

2.3.1 Computer Vision in VQA Systems

Visual Question Answering systems exploit Computer Vision models to extract information from the provided image. We have talked a lot about sequences of words, like questions and answers, but the visual modality is a crucial aspect of this problem. Every question comes with an image, and vice versa. In this field, it makes no sense to process a single modality without accounting for the other one because the information required to produce an answer lies in both of them. Once the model receives a question, it is supposed to look at the provided image in order to seek for relevant information, or features, that it can exploit to generate an answer.

In the past years, the community came up with architectures that are extremely powerful at getting the semantics of an image, namely, Convolutional Neural Networks (CNNs). These models will be described later, but, for the moment, it is just worth noticing that most VQA systems integrate such architectures in their structure to process input images.

Once again these images need some processing before being fed to the model, but the way in which this operation is performed strictly depends on the adopted CNN; usually the image is resized to a specific size, split into its three channels (red, green and blue) and all the pixels intensities are normalized with ad-hoc methods. For this reason, the input usually results in being a 3-dimensional matrix (channels, height, width).

2.4 Conclusion

In this chapter we have introduced many concepts, from elementary ones, such as what a token is, to more difficult problems, such as how to measure the cosine similarity between two embeddings. All the background discussed will be at the core of all this project while we dive

into the details and will provide a good knowledge base to feel confident with the reading. In the following chapter we will start describing some computational models for text sequences and images, in order to later introduce the related work in VQA.

CHAPTER 3

COMPUTATIONAL MODELS

In this chapter we will present some computational models that are widely used in the Deep Learning community and can be found in several different VQA systems. The majority of these models are implemented using Neural Networks and serve different purposes depending on their structure. From a high-level point of view, these models represent sub-blocks that are usually combined together to build the final system.

Assuming the reader is familiar with the basic concept of what a Neural Network is, we will skip describing its main components such as the Perceptron, and we dive immediately into the most relevant architectures.

3.1 Computational models for text

Dealing with sequences of text requires taking into account the notion of temporality. In other words, the order in which the words appear in a sequence represents an essential piece of information that a model should consider. Standard Neural Networks, such as the Multi-Layer Perceptron, do not consider this notion and process inputs sequentially one after the other. Even when the execution is batched (most of the cases) and the computation occurs in parallel, each sample is processed independently, without considering what the network has already seen. If we were to provide an example, given the sentence:

“Richard is hosting a party tonight for his birthday.”

It is quite important to catch the relationship between “his” and “Richard” in order to understand what the sentence is about. In order to address the latter objective, the community came up with a technique that recurs the output with the next input of the network. In this way, we do not lose all the information that was previously processed because we are feeding back to the model something related to the past of the considered sequence. This type of architecture is typically described as a Recurrent Neural Network (RNN) and is one of the most adopted for dealing with textual sequences.

3.1.1 Recurrent Neural Networks

As mentioned above, Recurrent Neural Networks were introduced to deal with inputs whose temporality matters. These kinds of inputs are usually called Time Series and, even though they may vary from being sequences of words to stock price changes in a certain markets, they all carry pieces of information where each time step is somehow related to the previous and following ones. As highlighted with the stock price change example, Time Series are not related only to text sequences and can be helpful in a wide range of tasks. However, in this work, we will talk about their usage to deal with Natural Language inputs.

A key aspect of RNNs is the presence of an internal state that keeps track of what the network has seen while processing the series. This state, usually referred to as “Hidden State”, is passed along every time a new input is fed to the network, providing context for the current computation. In other words, the hidden state of a RNN represents an internal memory unit that the system can exploit when generating outputs.

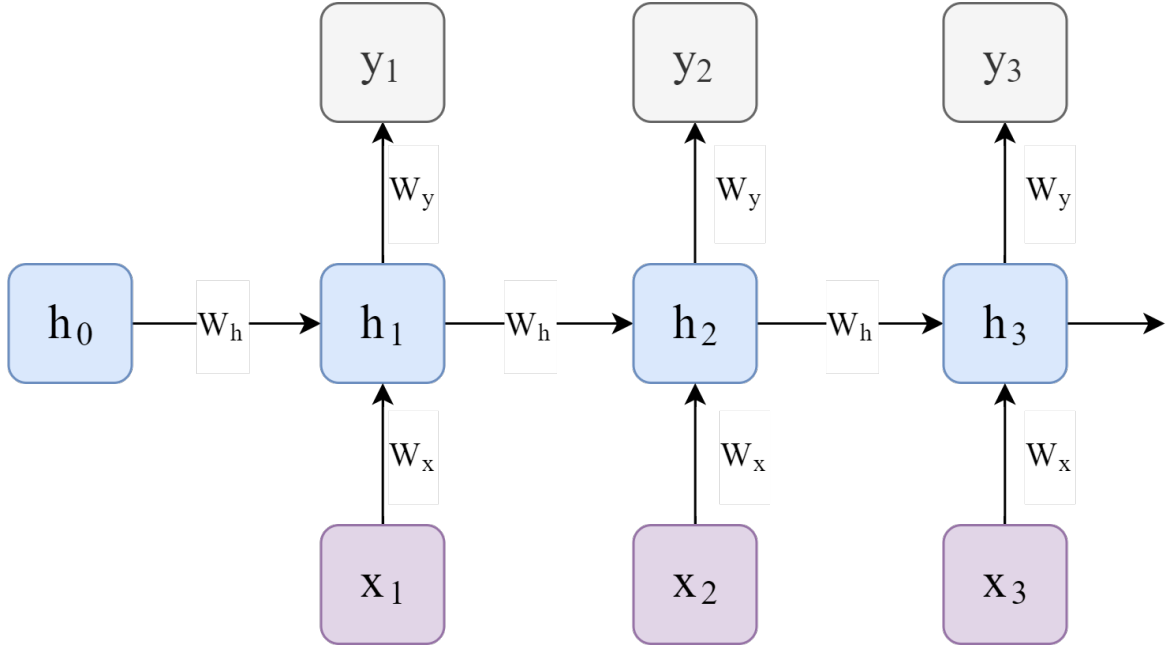


Figure 4: Recursion of hidden states in a RNN.

From a high-level point of view, we can split the information that lies in a RNN into three components:

- \vec{x} : The input vector which represents the input time series. Sticking with textual sequences, each element x_i represents a single encoded token.
- h_i : The hidden state of the network at time step i . It is computed using the hidden state at time step $i - 1$ and the input x_i at time step i .
- \vec{y} : The output vector which results from the computation of the current input x_i and the current hidden state h_i .

Figure 4 provides an overview of how the input interacts with the hidden state of the network to compute the output. At each time step i , the hidden states changes in order to keep track of what was previously fed as input. It is worth highlighting that even though Figure 4 shows how a RNN works unrolled in time (in this case over three time steps), in reality, the structure doesn't change depending on the size of the input but is fixed. Figure 5 exposes this concept visually, showing how the hidden states recurs over time.

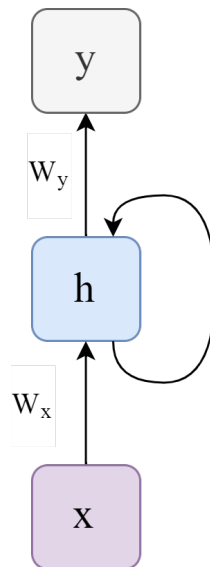


Figure 5: Given an input, the output of a RNN is computed using it and the current hidden state. The current hidden state takes into account the previous hidden state.

Sometimes, such as in Natural Language Understanding tasks, it is crucial to look not only back but even forward in the input. Bidirectional RNNs serve this purpose, and their structure is similar to standard RNNs except for a new hidden state that takes into consideration information even from future time steps. Figure 5 shows an example of a Bidirectional RNN unrolled over three time steps.

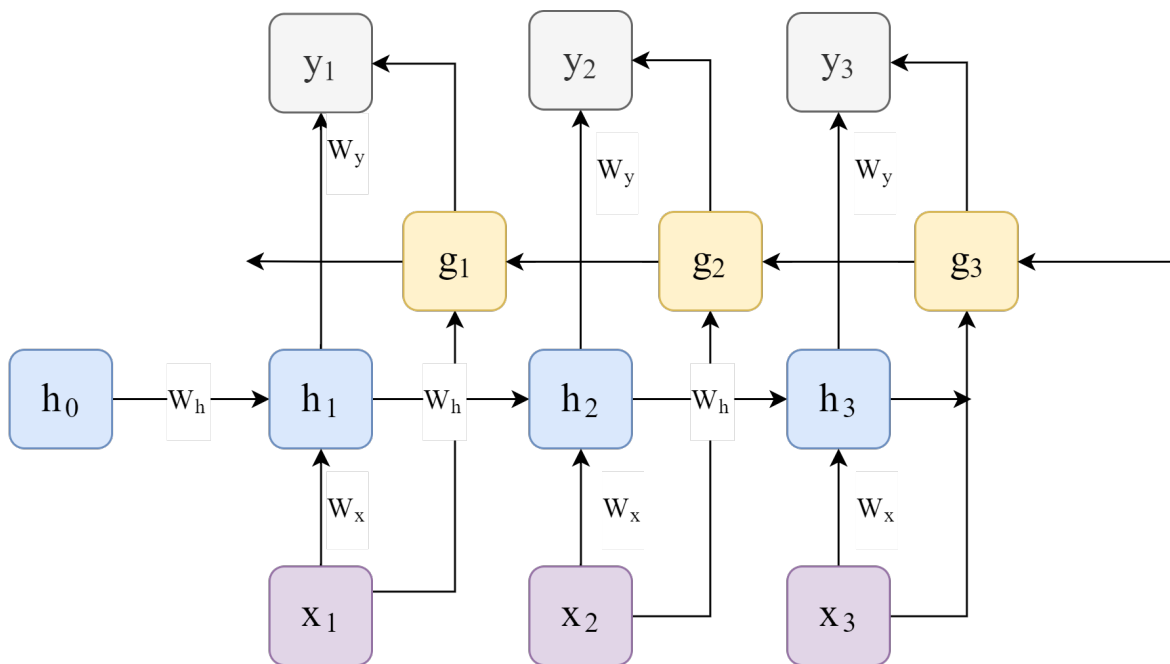


Figure 6: Recursion of hidden states in a Bidirectional RNN.

In order to improve the capability of understanding temporal relationships, we can make RNNs deeper by increasing the number of hidden states and concatenating them together. This leads to more powerful networks but comes at a cost both in terms of computational power and well-known issues.

3.1.1.1 Limitations

One of the worst issues with Deep Neural Networks (DNNs) and Deep RNNs is the so-called “vanishing gradient”. As highlighted above, in order to strengthen the understanding capabilities of such networks, and make them capable of dealing with long-term dependencies, the community tried to make them deeper. However, this strongly affected the computation of the gradients during the back-propagation procedure performed while training the network. More specifically, the deeper the system, the more the cases where small numbers get multiplied together, leading to near-zero gradients. This is a horrible case because if the gradients are equal to zero (vanished), the network will not be able to perform the update of its weights, resulting in the model not learning anything at all.

To overcome this issue, in 1997, two great minds came up with a new kind of RNN that was capable of dealing with this issue extremely well, namely, the Long-Short Term Memory (LSTM) Networks [9].

3.1.1.2 Long-Short Term Memory Networks

Traditional RNNs, while suited for many tasks that require short-term understanding, usually fail to catch long-term ones because of how they handle the information contained within

their hidden states at each time step. Furthermore, as described previously, making these networks deeper is not the right solution.

In order to catch and store long-term dependencies, LSTMs [9] were introduced. From a high-level point of view, these networks exhibit once again a recurrent architecture, but profoundly change how the hidden internal state, or memory, is stored and updated.

In traditional RNNs the input and current hidden state are passed through an activation function (see Figure 7). This lets through all the information coming from previous hidden states, and there is no filtering on what pieces of information from the past are significant. In other words, the hidden internal state is being always updated, regardless of the current input.

In Figure 7 we see a standard RNN in action: once again, \vec{X} represents the input time series and h_i represents the hidden state at time step i .

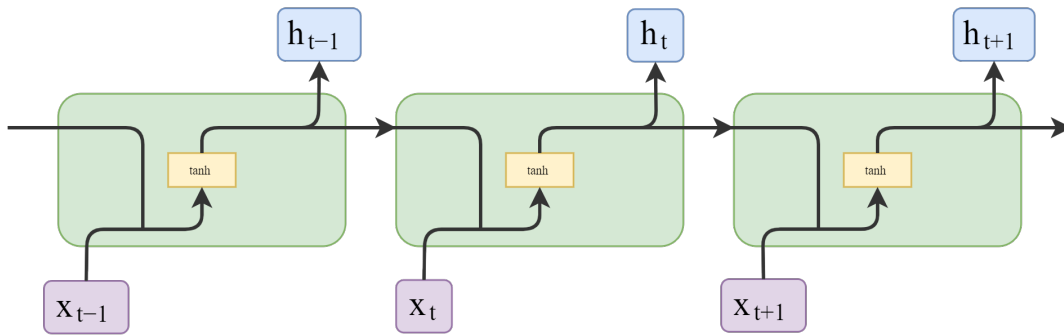


Figure 7: Standard RNN with a \tanh activation.

LSTMs provide a much more sophisticated gated mechanism to update, keep or reset the hidden internal state that, in turn, leads to more powerful architectures. An overview of such unit, usually referred to as *cell*, is provided in figure Figure 8.

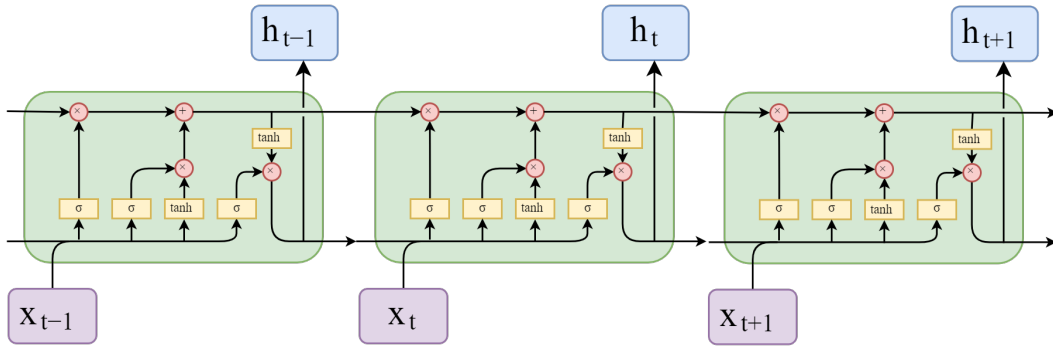


Figure 8: An LSTM cell unrolled over three time steps.

An LSTM exploits a total of 3 *sigmoid* and 2 *tanh* activation functions to update its internal state and generate outputs. We can subdivide these activation functions into three key components, or gates:

- **Forget Gate:** This gate is responsible for deciding which pieces of information we will throw away from the current cell state, or memory.
- **Input Gate:** This block is responsible for deciding which pieces of information contained in the input x_i can flow into the current cell state h_i .

- **Output Gate:** This is where the output of the current cell is computed.

In order to understand what is going on under the hood, let's consider a single time step and define some elements mathematically.

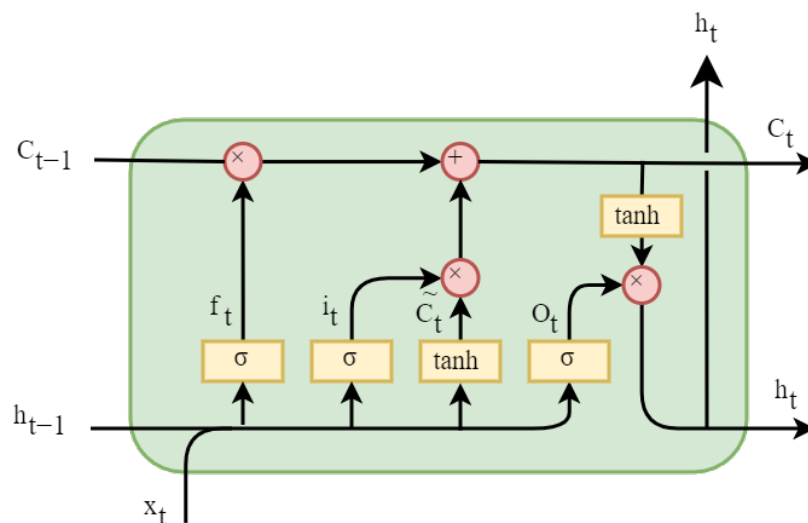


Figure 9: LSTM activation functions in detail.

First off, we will call C the internal state of the LSTM cell. This piece of information might be updated depending on the resulting activations at each time step. As usual x_t represents the input at time step t and h_t is the output of the cell at time step t .

The first step of a LSTM cell is to decide what information will be thrown away from the current cell state C_{t-1} . Given the current input x_t and the previous cell output h_{t-1} a value f_t between 0 and 1 is computed using a *sigmoid* activation function as follows:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (3.1)$$

where W_f and b_f are two parameters, learned during training. The resulting value f_t is then multiplied with each value in the cell state C_{t-1} (zero means completely throw away while one means to keep everything).

Afterwards, using once again the input x_t and the previous cell output h_{t-1} , we compute a new vector \tilde{C}_t of candidates ready to be inserted into the cell memory using a *tanh* activation. However the LSTM filters out some of the candidates using another *sigmoid* function i_t that is applied to the candidates \tilde{C}_t before adding them into the new memory state C_t . The input gate i_t and the candidates \tilde{C}_t are computed as:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (3.2)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad (3.3)$$

where W_i , W_C , b_i and b_C are parameters learned during training.

In order to update the internal cell state C (i.e. going from C_{t-1} to C_t) the forget gate f_t deletes unwanted past values and the input gate i_t selects the candidates that will be added to the cell state as follows:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_{t-1} \quad (3.4)$$

Finally, the output gate decides what will be the output of the cell at that specific time step. This value will be based on the updated hidden state C_t , but will be a filtered version of it. In fact one last *sigmoid* activation will compute the output gate filtering value o_t which, in turn, will be multiplied with the *tanh* activated hidden state C_t to compute the new output h_t as follows:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (3.5)$$

$$h_t = o_t * \tanh(C_t) \quad (3.6)$$

where W_o and b_o are parameters learned during training.

3.1.2 The Transformer

Back in chapter 1 we introduced a new powerful architecture called Transformer [1] without getting into any detail. Since this type of model will be at the core of our implementation, in this section we will dive into how it is structured and what is its purpose.

We have seen that most state-of-the-art models exploit recurrent architectures such as RNNs to capture dependencies between sequences. At each time step, these model generate or update a hidden state h_i as a function of previous hidden states h_{i-1} . The nature of such models is inherently sequential and assumes that the input is processed one element at a time. This

negatively affects parallelization within training examples and, as the input sequences become longer, gets more critical due to memory constraints.

The Transformer is a new architecture that relies entirely on attention mechanisms to understand long and short term dependencies within the input sequence. This allows for greater parallelization and more extended input sequence processing, achieving new state-of-the-art results in many different tasks.

One of the most significant applications of such a model is in the Machine Translation (MT) area, where we usually find high-level components (Encoders and a Decoders) working together to translate the sentence. The Encoder usually takes care of projecting the input, which might be a sentence in German, in another space that can be understood by the Decoder. In turn, the Decoder uses this encoded representation to generate the proper translation in, say, English. Until the introduction of this architecture, state-of-the-art MT models were built using LSTMs, already achieving great results.

The original implementation of the Transformer, described in [1], comes in an Encoder-Decoder fashion too. For this reason, we will first describe this model as a machine translator, and then we will address two specific variations of the system for other tasks. However, it is essential to immediately highlight that a Transformer is not always made up with an Encoder and a Decoder; GPT-2 [6], for instance, is an Encoder-only generative language model that will be described later on.

3.1.2.1 Model Architecture

As mentioned above, at its core, the canonical Transformer for MT is composed of an Encoder and a Decoder. In this case, the Encoder receives in input a sentence in a specific language and provides as output an encoded representation of it in a different subspace. This encoded representation holds several different pieces of information, such as the semantic of the question, language relationships, and much more. This encoded representation is then fed to the Decoder, which takes care of generating the output in a different language. The translation occurs while this sequence of operations takes place.

Formally, given in input a sequence $\vec{x} = (x_1, \dots, x_n)$, the Encoder maps it to a continuous representation $\vec{z} = (z_1, \dots, z_n)$. At this point the Decoder, using \vec{z} , generates the output sequence $\vec{y} = (y_1, \dots, y_n)$.

The Transformer is an auto-regressive model because while generating the output consumes the symbols generated at previous time steps as additional inputs. Both the Encoder and the Decoder consist of a stack of identical layers of size N (6 in the original paper). Even though the layers in both stacks are similar, there is a small, but paramount, difference:

- **Encoder:** Two sublayers compose each layer in the Encoder. The first one is a multi-headed self-attention layer, while the second one consists of a standard position-wise Feed-Forward Neural Network. A residual connection [10] is employed around them, followed by layer-normalization [11]. In other words, the output of each sublayer is computed as:

$$Output = LayerNorm(x + Sublayer(x))$$

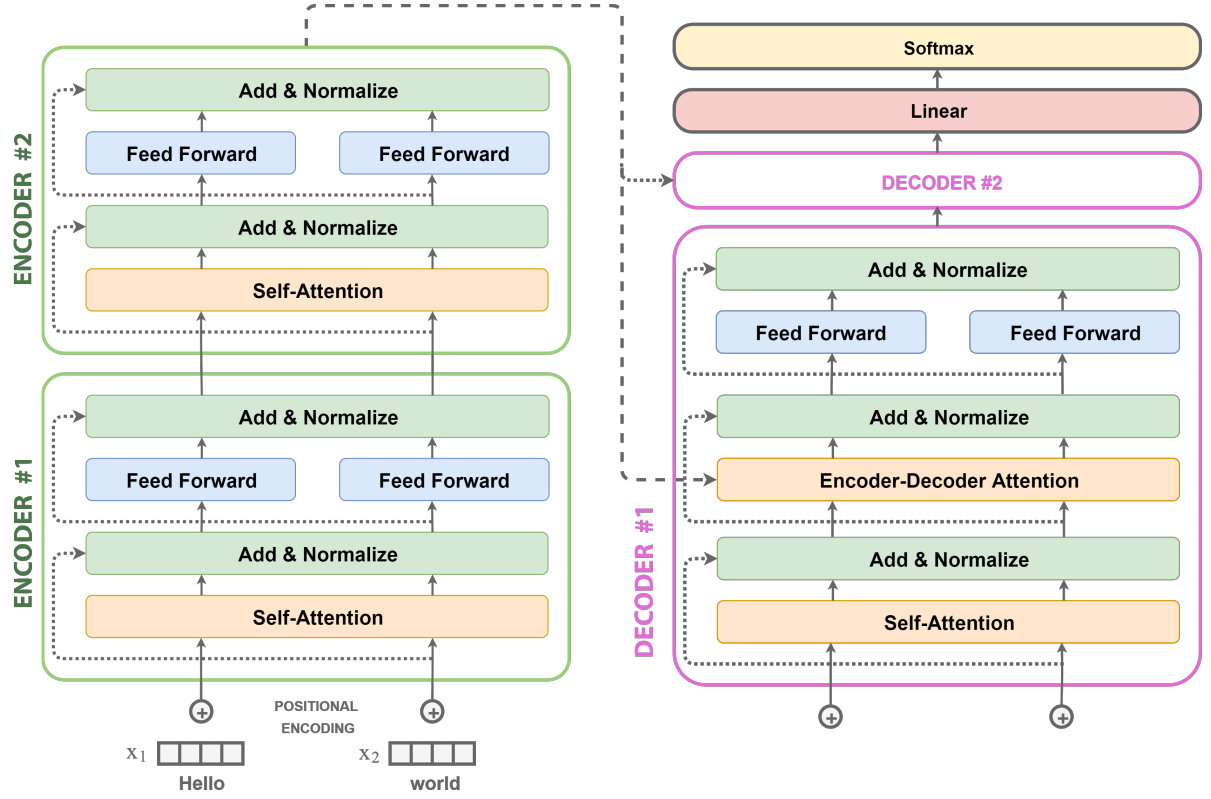


Figure 10: Transformer Encoder-Decoder stack example.

where Sublayer (x) is the function implemented by the Sublayer.

- **Decoder:** Each Decoder layer is composed of the same two sublayers present in the Encoder plus an intermediate sublayer, namely, an Encoder-Decoder self-attention layer which performs multi-head Attention on the output of the Encoder. Even in this case, residual connections are employed and are followed by a normalization step. Another difference in the Decoder is the fact that, in each self-attention layer, every position can attend (or look at) previous positions only. In other words, while generating the $i - th$

token, it can look at inputs up to i only. This prevents looking in the future while generating the output and is extremely important during the training phase.

At the very beginning of each stack there is an Embedding layer that accounts for projecting the input sequence into the right space, ready to be processed by both the Encoder and the Decoder.

Figure 10 shows the structure of one of the many layers in the stack, both for the Encoder and the Decoder. The overall architecture consists of a pile of these layers. The original paper uses six layers for both parts, while other models, such as GPT-2 Large (774M parameters), exploits a stack of 36 encoders.

3.1.2.2 Attention

The attention mechanism is what the Transformer is all about. It accounts for understanding relationships in the input and replaces the hidden state structure that was present in the LSTMs. It consists of mapping a Query and a set of Key-Value pairs to an output. These elements are all vectors of size d_k that are generated starting from each input embedding using matrices learned during the training step. Conceptually, exploiting mathematical operations, we are trying to say how much a specific element in the input relates to all the other features and, in turn, generate an output that accounts for these interactions. But before getting into details, it is worth defining these entities more formally.

Given an input sequence of n words $\vec{s} = (s_1, \dots, s_n) \in \mathbb{R}^{n \times 1}$, its embedded matrix representation $X = (\vec{x}_1, \dots, \vec{x}_n) \in \mathbb{R}^{n \times d_{model}}$ and three matrices, learned during training, $W_q \in \mathbb{R}^{d_{model} \times d_k}$,

$W_k \in \mathbb{R}^{d_{model} \times d_k}$ and $W_v \in \mathbb{R}^{d_{model} \times d_k}$ we can compute the Queries $Q \in \mathbb{R}^{n \times d_k}$, Keys $K \in \mathbb{R}^{n \times d_k}$ and Values $V \in \mathbb{R}^{n \times d_k}$ as follows:

$$Q = X * W_q$$

$$K = X * W_k$$

$$V = X * W_v$$

Q , K and V are three matrices and carry information about the input sequence that are used to calculate the attention value, computed as follows:

$$Attention(Q, K, V) = Softmax\left(\frac{Q * K^T}{\sqrt{d_k}}\right) * V \quad (3.7)$$

Each self-attention Sublayer in the Encoder and Decoder stacks performs this operation. Then, as we have seen, this result is summed with the residuals and normalized. Afterward, it is fed to the position-wise FFNN sublayer. This set of operations is repeated in each layer of the stack, where the input to each Encoder/Decoder is the output of the previous one, except for the first, which deals with the Embeddings.

To better understand how the attention mechanism works, consider the input sentence:

The animal didn't cross the street because it was too tired.

Given the sequence, the attention mechanism computes, for each word in the sentence, an attention vector. Figure 11 shows the attention values for the word "it": brighter shades of orange correspond to words which are given more Attention, whereas lighter ones indicate less

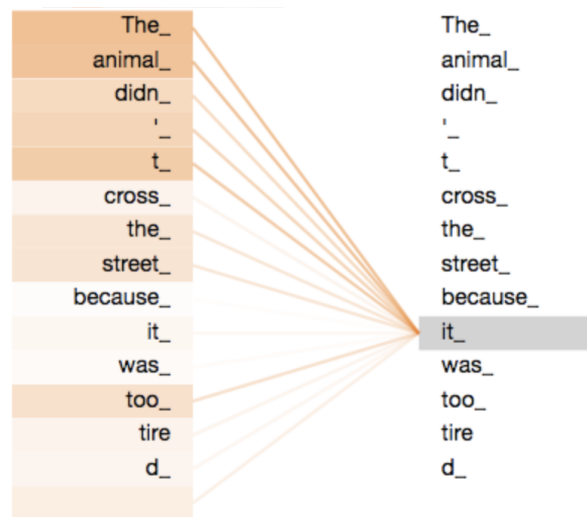


Figure 11: Single head attention example. Image adopted from [12]

important ones. For instance, it can be seen that a lot of Attention is put on the word "*The*" and "*animal*", and effectively in this case "*it*" refers precisely to that subject.

As it is evident from Equation 3.7 the Attention is being computed for every element in the input sequence in parallel exploiting matrix computations. This, in the end, leads to a model that trains way faster with respect to other models, such as LSTMs, due to the highly parallel architecture of modern GPUs and TPUs. At the same time, the input window can be much broader and can account for longer sequences because less memory is being used.

3.1.2.3 Multi-Head Attention

Even though in the previous example we described the attention layer as if it was outputting a single vector for each input token, in practice, the original architecture computes multiple

attention vectors simultaneously. These various vectors are then concatenated together and multiplied with a weight matrix, performing what is called a Multi-Head Attention operation (In the original paper, $h = 8$ different heads are used).

As reported in [1], “Multi-Head Attention allows the model to jointly attend to information from different representation sub-spaces at different positions”; this ultimately lead to an increase of the model performances.

The computation of each head remains unchanged, and the final Multi-Head Attention is computed as:

$$MultiHeadAttention(Q, K, V) = Concat(head_1, \dots, head_h) * W_o \quad (3.8)$$

where $W_o \in \mathbb{R}^{d_{model} \times d_k}$ is a weight matrix learned during the training phase and:

$$head_i = Attention(Q * W_q^i, K * W_k^i, V * W_v^i) \quad (3.9)$$

with $W_q^i \in \mathbb{R}^{d_{model} \times d_k}$, $W_k^i \in \mathbb{R}^{d_{model} \times d_k}$ and $W_v^i \in \mathbb{R}^{d_{model} \times d_k}$.

As highlighted by Equation 3.9 the computation of each head is identical to the single head version (see Equation 3.7) and the only difference lies in the multiple head concatenation and multiplication with W_o .

Figure 12 shows how a multiple head attention mechanism put Attention on the words of our toy example. The image shows how each head displaces attention when considering the token

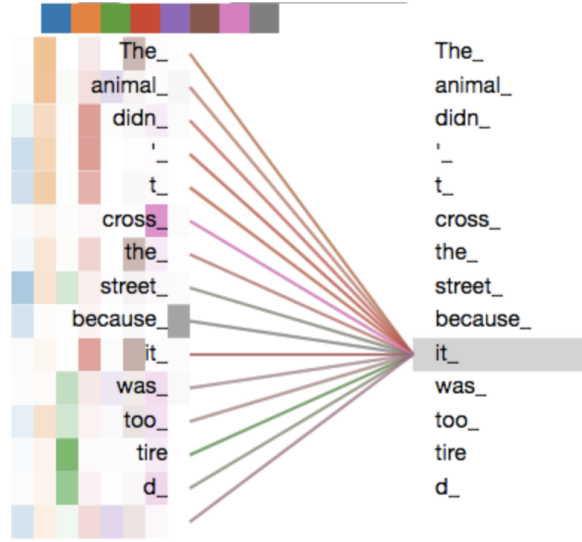


Figure 12: Multiple head attention example ($h = 8$).Image adopted from [12]

"it". Each head is represented by a different color and the intensity indicates the attention level.

3.1.2.4 Position-Wise Feed-Forward Networks

Once Attention has been computed in each Sublayer, it is fed with a Position-Wise FFNN, which is applied to each position identically. This Sublayer consists of two Linear transformations with a ReLU in between as:

$$FFN(x) = \max(0, x * W_1 + b_1) * W_2 + b_2 \quad (3.10)$$

where W_1 , W_2 , b_1 and b_2 are the weights of the considered Sublayer, and are learned within the training phase.

3.1.2.5 Positional Encoding

Since the model is not recurrent and does not exploit convolutions (we will describe them in the next sections), without a proper encoding of the input sequence, it will be unable to use the ordering information present in it. For this reason, the authors of [1], decided to inject information about the position of the words in the sequence using cosine functions of different frequencies. These positional encodings are computed starting from the input embeddings and are later on added to them, resulting in new embeddings that carry not only the semantics of the words but even positional information.

3.1.3 Generative Pre-trained Transformer (GPT-2)

OpenAI’s Generative Pre-trained Transformer (GPT-2) [6] is one of the most exciting and powerful applications of transformers. This model is a large-scale unsupervised language model that is capable of performing several different tasks all in one such as paragraph generation, reading comprehension, and machine translation. Trained on over 8 million web pages, it is so powerful that the biggest architecture that has been trained, with 1542M parameters, was not released to the public until recently due to the fear of being used as a fake news generator.

In this work we decided to take and tune its smallest public version (117M parameters) for our VQA model as a replacement of the commonly used LSTMs. As mentioned many times, we hoped to exploit such power to generate better, open-domain answers.

This model works differently from the Transformer presented in the original paper in that it lacks the Encoder-Decoder structure described previously. This because GPT-2 hasn't been trained to perform Machine Translation tasks only, but rather to generate tons of text either in a conditioned or unconditioned manner. This is why it consists of a massive stack of encoders, without any decoder.

The following extract is taken from a CNN article [13]:

“In the Persian Gulf, a US aircraft carrier, the USS Abraham Lincoln, lurks off the Iranian coast, sending out a message of aggression. Meanwhile, in the Levant, the administration of US President Donald Trump starts to roll out his version of a widely anticipated peace plan for the Israelis and the Palestinians. ”

To provide an example of how the conditioned GPT-2 model works, we report a small snippet of the model output when fed with the previous context:

“Mindful of this, the leaders here reached agreement after weeks of diplomatic belaboring on a project spanning 11 conferences, with six Palestinian initiatives and 10 Arab initiatives sitting beside one another. Going all in, however, could have exposed Mr. Trump to a game of chicken...” [continues]

The model is capable of generating sequences that are correlated with the input exploiting the knowledge that it learned. In this case, the discussion of a US carrier in the Persian Gulf led it to talk about Palestinians and Arab initiatives. Furthermore, the model made references even to the US president. Even though sentences are not always meaningful, they are, most of

the times, syntactically correct. This has been our driver to experiment with GPT-2 and led to exciting results.

3.1.4 Bidirectional Encoder Representation from Transformers (BERT)

Google’s Bidirectional Encoder Representation from Transformers [14] is another powerful pre-trained architecture based on the Transformer. It is conceptually similar to GPT-2 [6] but exhibits a key difference in directionality by which textual sequences are processed.

While GPT-2 is unidirectional and processed sequences from left to right, BERT endorses a bidirectional approach and, when generating the token at position i , is allowed to look backward and forward in the input sequence.

BERT’s primary goal is to provide the community with a pre-trained language model that can be used to quickly come up with state-of-the-art architectures for a multitude of NLP tasks just by fine-tuning the last classification layer, usually referred to as “head”.

While training BERT, in order to encourage the bi-directional behavior at fine-tuning time, the authors trains it on two different language tasks:

1. **Masked Language Model:** The first task on which BERT is trained consists in predicting masked tokens in the input, which are randomly replaced 15% of the times. If the i -th token is chosen for masking, 80% of the times it is replaced by a special token [MASK], 10% of the times it is replaced by a random token and 10% of the times it remains unchanged. The reason for which the i -th token is not always replaced with the [MASK] token lies in the fact that, at fine-tuning time, we will never feed the model a

[MASK] token, but only un-masked sentences. The goal of this training task is solely to teach the model to be bi-directional.

2. **Next Sentence Prediction:** Since the majority of the downstream tasks relate to understanding the relationship between sequences, BERT authors decided to train it even to make next sentence prediction. The way in which they achieve the goal consists in feeding the model a pair of sentences A and B and letting it classify whether B follows A or not. During training, they feed pairs such that 50% of the times B follows A and 50% not.

Another difference between BERT and GPT-2 is that the former employs WordPiece tokenization [15] while the latter uses BPE [5].

While developing the baselines for this work, we experimented with BERT and found out that, even though its bidirectional nature is beneficial for most NLP tasks, it seriously hurts VQA systems because it allows the model to see the answer during the training phase.

3.1.5 Discussion

As we have seen, RNNs, LSTMs, and Transformers are robust architectures that are widely adopted in current state-of-the-art NLP models. They allow machines to exploit the short and long-term dependencies of the sequences when generating output, providing some understanding of the surrounding context. LSTMs are present in almost every related work that we will introduce in the next chapter, and the Transformer will be at the core of our implementation, so it was worth adding some mathematical notation to understand key concepts better.

Now that we have addressed the essential computational models for text sequences, we will move the focus to models for image processing.

3.2 Computational models for images

The scientific community, trying to emulate how the human eye works, came up with models that can understand the information contained within images and videos brilliantly. Nowadays, we are able to perform, to name a few, image and video recognition, media recreation, image analysis, and transfer learning thanks to the architectures developed in the Computer Vision area of research.

When talking about Deep Learning and images, the first architecture that comes to the mind of every Data Scientist is the so-called Convolutional Neural Network or simply CNN. Over time scientists understood that this architecture was much more powerful with respect to standard Feed-Forward Networks thanks to the way in which the image features were progressively extracted and to the input invariance introduced by convolutions.

In this section, we will solely address this type of model since it is the only used within this work, and afterward we will discuss one task-specific architecture implemented using CNNs.

3.2.1 Convolutional Neural Networks

A CNN is a type of Neural Network that is able to take in input an image and assign to the various objects present in it different weights and biases in order to understand their semantics and distinguish one from the other ones.

Even though we commonly refer to CNNs as architectures made only for images (or 2D inputs), as in this case, it is worth mentioning that they are used even for processing text sequences (1D inputs) or volumetric data (3D inputs).

These types of networks perform well at processing the information present within an image with a sequence of operations that progressively extracts higher-level features.

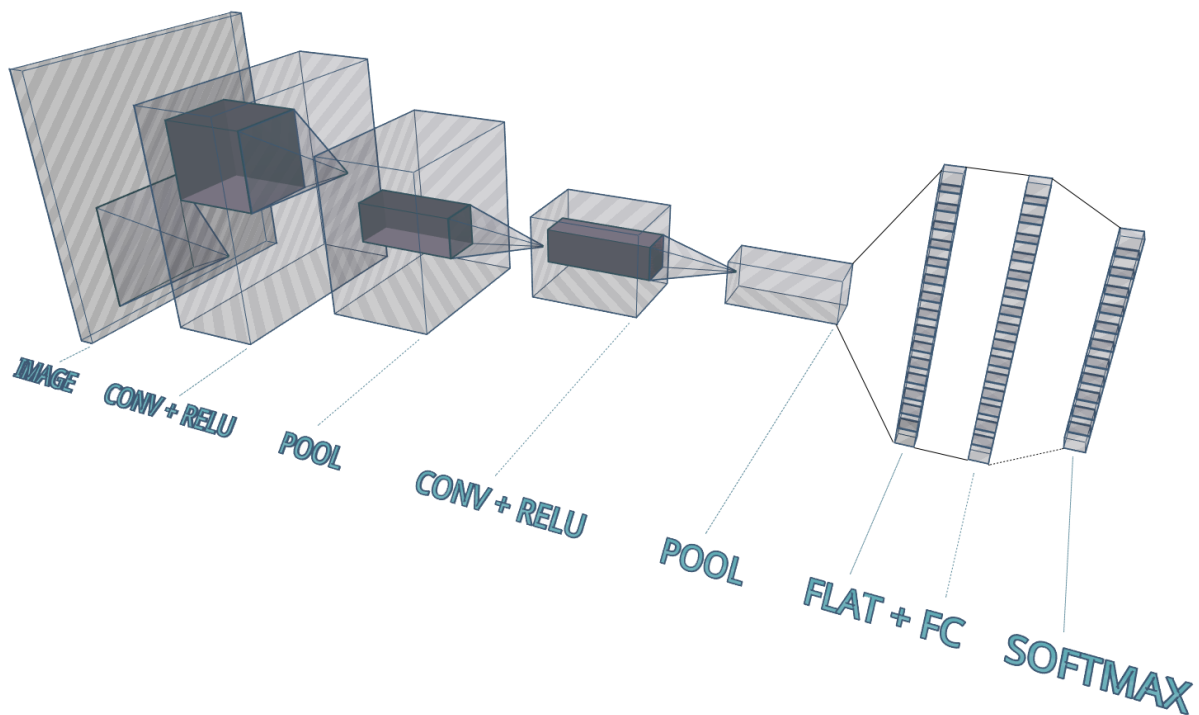


Figure 13: General structure of a CNN.

The general structure of a CNN (see Figure 13) is made up of a sequence of three operations that might be repeated many times, followed, usually, by a classifier. Even though CNNs are primarily used to classify what they are looking at, this is not always the case. For instance, in this work, we will integrate CNNs as features extractors without any classifier.

The main components of a CNN are the following ones:

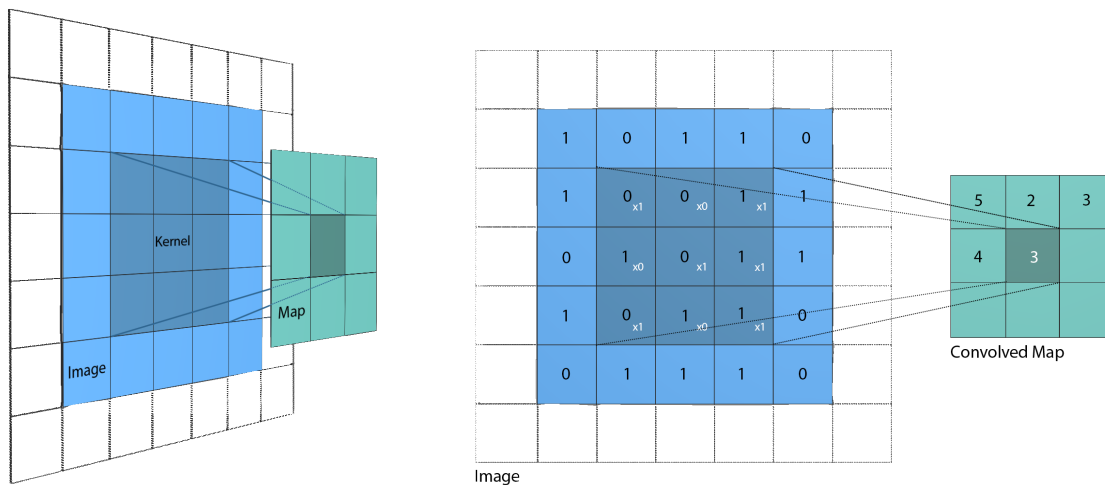
- **Convolution layers**
- **Non-Linear activations**
- **Pooling layers**
- **Final classifier**

Before describing each component into details, we assume every image to be represented in an RGB color space with 8 bits of depth. A value between 0 and 1 represents every pixel, and every image has three channels, namely, Red, Green, and Blue. Formally, given an image of size $width \times height$ it will be represented by a vector $I \in \mathbb{R}^{width \times height \times 3}$.

3.2.1.1 Convolutions

Convolutions are basic functions for image processing in Computer Vision. From a mathematical point of view, it consists of sliding a Filter (also known as Kernel) over the entire image to produce an output map. The filter is a square matrix of fixed size, and the amount of pixels by which it is moved over the image at each step is called Stride. This operation condenses the information present in the convolved pixels into a single value. Depending on the values

contained within the filter, the convolution might capture information such as image edges or other patterns.



(a) Filter of size 3x3 (dark blue) going over a portion of the 5x5 input image (blue) to generate the output map (green).

(b) Convolution between a 5x5 image (blue) and a 3x3 filter (dark blue). Stride = 1. The resulting map (green) has size 3x3.

Figure 14: Examples of convolutions

The input image might be padded with 0 values on the border to obtain different sized output maps.

Generally speaking, given:

- **An Image** $I \in \mathbb{R}^{w \times h \times c}$, where w = image width, h = image height and c = image channels (usually 3).

- **A Kernel** $K \in \mathbb{R}^{w_k \times h_k}$, where w_k = kernel width and h_k = kernel height.

a convolution $C_{i,j}$ of the image I with the Kernel K to produce the element at position i, j in the output map is defined as:

$$C_{i,j} = \sum_{k=0}^w \sum_{p=0}^h \sum_{m=0}^c K^{w_k-k, h_k-p} * I^{k+i, p+j, m} \quad (3.11)$$

where $w_k < w$ and $h_k < h$, that means the kernel is smaller than the image.

In Figure 14 is shown an example of a convolution between a 5x5 image with a 3x3 kernel. The Input image is convolved with the Kernel, and each value (i, j) in the output map, depicted in green, is computed according to Equation 3.11.

Convolutions are compelling operators that are used not only within CNNs but even in several different standalone software. In the latter case, most of their applications include generating ad-hoc filters, such as sharpening or blurring an image. The exciting aspect of a CNN is that it learns by itself what features each filter should extract depending on the task and the data.

3.2.1.2 Non-Linear activations

Following each convolution, there is a non-linear activation function to introduce non-linearities in the network. Furthermore, since these architectures are usually deep, a ReLU activation is usually employed to help the gradient flow, since its derivative is always equal to one.

3.2.1.3 Pooling

After the non-linear activation, we find a pooling layer. Similar to what happens with the convolutions, this operation accounts for reducing the dimensionality of the output maps to a smaller space. The dimensionality reduction operation helps to reduce the computational power needed to process the entire image. Furthermore, the most critical aspect of the pooling procedure is its capability to extract features that are invariant both to rotational and positional shifts.

Given a portion of the convolution output map, pooling is implemented by taking either the maximum value in the considered region (Max-Pool) or its average value (Avg-Pool).

3.2.1.4 The classifier

The final classifier accounts for flattening the output of the last pooling layer and passing it through a Feed-Forward Network, on top of which we usually find a softmax over the possible classes.

3.2.2 VGGNet

VGGNet [16] is one of the many state-of-the-art CNNs for large-scale image recognition. It came up in 2014 to compete on the Large Scale Visual Recognition Challenge (ILSVRC) [17], setting new state-of-the-art results. In this work, we made heavy use of this architecture by taking several different pre-trained version of this network, like VGGNet-11, and integrating them into our baselines and implementations.

One exciting aspect of this network is that it uses the smallest possible receptive fields (i.e., kernels) to capture spatial dependencies, that is 3x3. ReLU activations are employed, and the last pooling layer, in the case of VGGNet-11, produces 512 different maps.

The classifier is composed of three fully connected layers, two of size 4096 and the last one of size 1000, equivalent to the possible classes in the ILSVRC challenge. VGGNet comes in 6 different configurations: the smallest one has 11 layers of convolutions, non-linear activations, and pooling, whereas the biggest one has 19 layers. No matter which architecture we choose, VGGNet remains a mighty Deep Convolutional Neural Network. For computational limits, in this work, we stick with the 11 layers configuration.

3.3 Conclusion

In this chapter we discussed how some of the most important computational models for this project work, describing which are the state-of-the-art techniques to deal with textual and visual inputs. In the next chapter, we will talk about the current state-of-the-art in the VQA field and will see how the community tried developed approaches that combine both LSTMs and CNNs to achieve the final goal.

CHAPTER 4

RELATED WORKS

After having provided the reader with the necessary background in Chapter 2 and the knowledge of the most used models for NLP and Computer Vision in Chapter 3, we will now discuss related works in the Visual Question Answering field of research.

The most common architecture for a VQA system is to put side by side textual and visual feature extractors and feed their output to an answer generator. Most of the time, these components are just encoders and decoders cleverly combined and followed by a final classifier.

The related works in VQA show that the most traditional way of dealing with textual sequences is to use an LSTM. As we have seen, these architectures can catch long-term dependencies in the input sequence and usually represent the best approach. Image features are instead commonly extracted using pre-trained CNNs: the majority of the related works that we will now introduce employ networks such as VGGNet [16] or ResNet [18]. These networks are usually pre-trained on ImageNet [17] and achieve high accuracy on the classes in the mentioned dataset.

There are two main advantages of using pre-trained CNNs: the first one is that we can exploit networks that are already capable of extracting visual features very well. The second one is that the size of the final model will be smaller since we have fewer parameters to train.

The usual approach is to integrate such CNNs in the VQA models and fine-tune only some of its layers, keeping intact the knowledge that comes with the pre-trained network. Most of

the time, the final classifier of these CNNs is removed in favor of other layers that bring the output into a space where it can be combined with the output of the language model. In turn, this new layer, as well as the language model, are trained together to learn how to work side by side.

As we will discuss soon, one of the most delicate aspects of VQA is how the two modalities (question and image) are brought to a shared space where they can jointly provide a valid representation to generate correct answers. Conventional approaches, such as in [19], include mathematical operations between vectors, like point-wise multiplications or sums. Other approaches, like the ones in [20], prefer projecting the two vectors in a higher dimensional space using an outer product followed by convolutions in a Fast Fourier Transform space.

When building a VQA system, it is crucial to understand whether or not the built system is exploiting both modalities or not. For this reason, it is common practice to create first some baselines and then compare their output with the final model. Such baselines are usually built incrementally, first considering one modality at a time and then combining them together.

In [19] and [21], the authors first implement some models that perform random guessing, then they proceed by always choosing the most common answer depending on the type of question. Later on, their most sophisticated baseline uses K-Nearest Neighbor [22] to find the closest question-image pair and chooses its ground truth as the answer.

This incremental approach is fundamental, and, in this work, we follow the same pattern to build our system.

The final implementation in [21], called LSTM Q+I, is a powerful model that exploits both modalities to generate the answer. It uses a pre-trained VGGNet network, that provides an encoded representation of the image, side by side with an LSTM, to extract the question features.

The model combines the two resulting feature vectors using a simple point-wise multiplication that is ultimately fed into a final, fully-connected, layer. On top of the last layer, a softmax function is employed to distribute probabilities over the possible answers that span across 1000 different candidates.

The latter softmax highlights how the problem is addressed in a classification manner. This approach does not leave space for open domain and open-ended answers since the last classifier forces the output to lie within a fixed set of choices. This system achieves roughly 57% overall accuracy on the VQA Dataset [21] that they introduce, but is far from being a system that can be used in a real-world scenario due to the way in which they treat the problem.

However, it is essential to note that the approach discussed in [21] brought a vast contribution to this work since they introduce a new Dataset for Visual Question Answering that we adopt to train and test our models. Furthermore, we took the idea of using point-wise multiplication to combine features while developing our attention mechanism.

4.1 Neural Module Networks

Recent literature [23] [24] decided to tackle the VQA problem by trying to exploit the intrinsic compositional nature of questions to create ad-hoc networks on the fly. These networks, called Neural Module Networks (NMN), change their structure depending on the question by

combining a set of modules. These modules are, in turn, pre-trained neural networks that achieve small tasks, like saying whether in an image there is a dog or not.

Every module is made up such that it can be concatenated with every other module available, giving almost unconstrained limitations to the layout of the final network.

In these architectures [23] [24], there are usually a set of modules that focus the interest on specific parts of the image, classify what they see and generate a label.

Consider the following question:

Where is the dog?

This sequence, when fed to a NMN, induces the creation of an ad-hoc layout where several different modules are combined together. Conceptually a good layout is one where a module accounts for focusing the attention on a portion of the image where the dog lies, leaving to another module the task of answering the question. Figure 15 provides a visual example: given a picture and a question, a network is dynamically created combining modules. Attention modules are in green and labeling modules in blue. In this case, two modules are used: one to locate the dog (dog module) and the other to generate the answer (where module).

Modules are combined sequentially, and the output of each one is the input to the following one. The last module is then connected with an LSTM (such as in [23]) via a point-wise multiplication, followed by a classifier.

In [23] there are a total of 5 different modules that can be concatenated to attend, re-attend, combine, measure, and classify every input image. Each one of these modules generates a mask over the image depending on how the question is parsed, and, in the end, these maps,

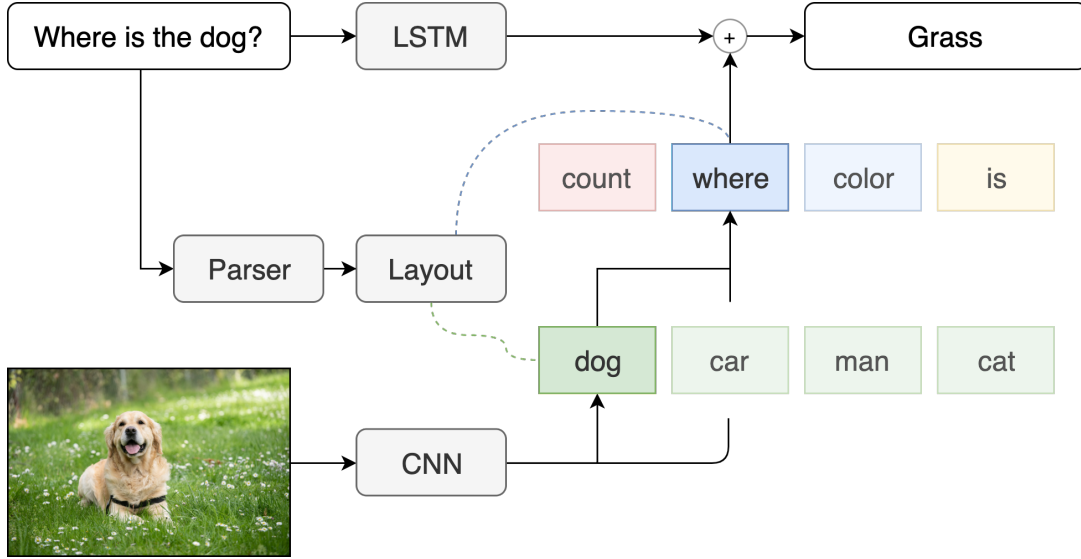


Figure 15: Example of a Neural Module Network described in [23]. Dog image adopted from [2]

concatenated together, provide a spot of interest that is ultimately used to generate the final answer.

To dynamically structure the layout of the network, the authors of [23] decided to parse the input question using the Stanford Parser [25]. Then the parsed question is used to obtain a universal dependency representation of the input [26], which is responsible for combining the modules of the network together.

Another approach that uses Neural Module Networks is shown in [24]. Even though the idea of creating a dynamic structure depending on the question remains unchanged, the way in which the authors deal with the input question differs.

In this case, a Reinforcement Learning technique is employed, where they exploit a layout policy instead of a fixed parser. The policy outputs a layout probability distribution $p(layout|question)$ from which the most probable layout is sampled. The latter policy is not restricted to a small set of candidates but accounts for all possible layouts of the network.

This idea is a step forward with respect to the approach discussed in [23] since common language parsers usually introduce severe limitations since they are not designed for language and vision tasks and must be modified using custom rules. In turn, this often leads to invalid layouts.

The model exploits a multi-layer LSTM that acts as the encoder for the input question; the encoded representation is then fed to another, policy-based, LSTM that accounts for the creation of the layout.

During the training of this architecture, a joint loss, that accounts for both the layout policy and the parameters in each neural module, is minimized, allowing the model to learn how to parse the question into specific linguistic structures and how to use the latter ones to build a valid layout.

It is worth noticing that these approaches treat the problem as a classification task, too, where single tokens compose each answer. Nevertheless, this kind of architecture inspired many other works, including the one that lead to the creation of the Natural Language for Visual Reasoning dataset [27] [28], where NMNs have been employed to answer questions on real images.

4.2 Hierarchical Co-attention

The work described so far is mainly interested to understand which are the relevant parts of the image to look at. However, as the authors of [29] highlight, it is important to look at the most relevant parts of the question too. Metaphorically speaking, aside from “where to look”, another critical aspect of answering the question is “what words to listen to” within the input.

A recently proposed approach [29] tries to jointly reason on the question and the image in a hierarchical way. This model, while generating the answer, first reasons at a word-level, then at a sentence-level, and finally considers the entire question. For each considered level, an attention map is generated on the image, which in turn prioritizes portions of the question, throwing away useless words. These levels are subsequently combined recursively to produce a distribution over a set of candidate answers.

This model introduces a new attention mechanism called Co-Attention, which, unlike previous work, jointly reasons on the image and on question attentions. Not only is the question representation used to drive the attention on the image, in this case the image drives the attention on the question too, leading to better results.

An embedding matrix is used to extract semantic features at a word level, and a 1D convolution takes care of encoding the meaning of each sentence. Finally, at the question level, a standard RNNs is used to catch longer-term dependencies and semantics.

The authors experimented with two different CNNs to extract visual features, namely, VGGNet and ResNet [18], with the latter one providing slightly higher accuracies.

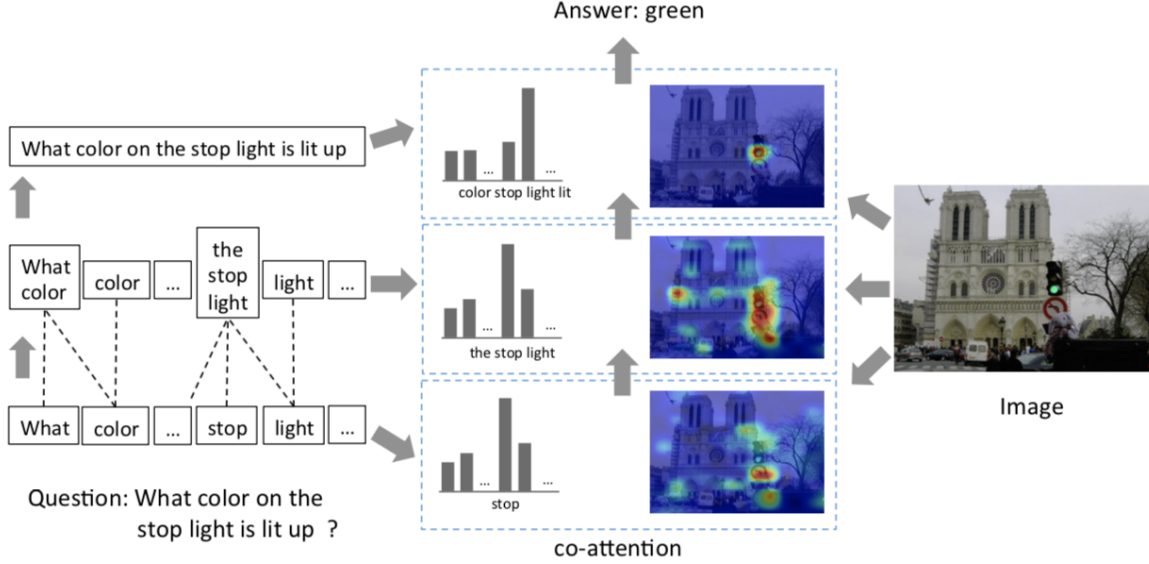


Figure 16: Example of the Hierarchical Co-attention mechanism in action at word, sentence and question level with corresponding softmaps. Reprinted with permission (reported in Appendix B)

The features from both modalities are then merged together with a set of mathematical operations that make up the attention mechanism and are finally fed to the classifier.

Even though this approach still treats the problem as a classification task, it inspired our work, especially with the introduction of the Co-Attention mechanism. As we will see in the implementation chapter, our model employs a similar technique where both the image and the question are used to compute attention over the whole input.

4.3 Multimodal Compact Bilinear Pooling

All the approaches described in the previous three sections rely on combining visual and textual vector representations using element-wise or sum operations, as well as concatenation.

Multimodal Compact Bilinear Pooling (MCB) [20] is a robust architecture that exploits an outer product to combine the two modalities to maximize the multiplicative interaction between their two vector representations. The authors of the paper believe that the traditional way of combining the two modalities is not enough to project them in the right space and thus employ this different mechanism.

In MCB the vector representation for text and visual inputs are computed, respectively, by a 2-layer LSTM with 1024 units each and a 152-layer Residual Network [30] (a particular type of CNN), recalling what is usually done in other related works.

Once the multimodal representations are ready, in order to deal with the high dimensional vectors resulting from traditional bilinear pooling, the system exploits a different technique [31] to efficiently compress the output for every modality: the Multimodal Compact Bilinear Pooling operation is first approximated by randomly projecting the image and question into a higher dimensional space, using a method called Count Sketch [32]; afterward, the two representations are convolved using element-wise product in the Fast Fourier Transform (FFT) space.

MCB is used twice in the best performing model:

- **First MCB:** given the outputs of the CNN and the outputs of the LSTM, it is used to compute the attention over the input image.
- **Second MCB:** given the attention output computed by the first MCB and the outputs of the LSTM, it is used to generate the final answer.

While writing this work, MCB is still the state of the art model on the VQA Dataset [21], achieving an overall accuracy of 66.7%. Once again, this approach does not generate open domain answers and distributes the output over a fixed set of possible choices.

4.4 Conclusion

Even though the scientific community in the VQA area of research developed different architectures other than those described above, in this chapter, we reported the most interesting and compelling ones for this work.

As we have seen, every model is always composed of an image encoder, which usually is a CNN, and a text encoder, where we usually see one LSTM in action. The difference among the described approaches lies in the way in which these two encoded representations are brought to a common subspace. Some systems employ only point-wise multiplications, whereas others integrate attention mechanisms to understand the input. However, the majority of the architectures present for VQA tasks address the problem as if they were dealing with a classification problem, resulting in short answers.

As we have stated in the motivation for this work, we try to overcome this limitation with a new kind of architecture that is conceptually similar to the ones previously described, but differs in how the output is generated. As we will see in the next chapter, we eliminate the final classifier and look at this problem from a different perspective.

CHAPTER 5

DATASET AND IMPLEMENTATION

In this chapter, we discuss the materials and methods used to develop our work. We first introduce the adopted dataset, describing which are its features and limitations. Afterward, we dive into the data preprocessing pipeline followed to prepare the inputs to our models. After this, we introduce some baselines and discuss their implementation. Finally, we present our new architecture, namely, VGGPT-2, and compare it against the baselines, qualitatively and quantitatively.

5.1 Dataset

While trying to choose the most proper dataset for this work, we had to keep in mind that we needed a large corpus with a specific structure. More specifically, we were interested in samples that were composed of a question on a specific image with its associated answer.

Finding such a large corpus is not an easy task since the ground truth usually has to be annotated by real human beings, and this operation requires time. To deal with this issue, in the past, the community adopted synthetic datasets generated by algorithms, such as with NLVRv1 [27]. The latter approach allows for nearly unconstrained corpora since questions, images, and answers are generated by machines. However, this approach lacks the usage of real-world images, which are the most interesting to deal with.

Thankfully, with the help of great web-based tools, such as Amazon Mechanical Turk [33], the scientific community has collected several different annotations (in terms of question/answer pairs) on real-world images and came up with datasets that are extremely useful for training VQA systems.

While choosing the dataset for this work, we considered the following corpora:

- **NLVRv1** [27]: The Natural Language for Visual Reasoning v1 is a synthetically generated corpus whose task is to say whether a question about an image is true or not. We discarded this dataset due to the lack of real-world images and binary types of answers (true/false).
- **NLVRv2** [28]: The Natural Language for Visual Reasoning v2 is the evolution of NLVRv1 and consists of real-world images. The data was collected via crowdsourcing, and the task is the same one of NLVRv1. We discarded this dataset because of the binary nature of its answers.
- **VCR** [34]: The Visual Commonsense Reasoning dataset comes with questions about images where a rationale supports each answer. Its goal is to train models that provide answers with supporting facts. We discarded this dataset due to its complexity: all the images come with spatial annotation that tell where the objects are and every ground truth has multiple supporting facts that we didn't intend to exploit.
- **CLEVRv1** [35]: The Compositional Language and Elementary Visual Reasoning dataset consists of a set of questions on synthetically generated images about different shapes of different colors. Even though CLEVR is a great dataset, widely used within the scientific community, we discarded it due to its synthetic nature.

- **MSCOCO** [36]: The Microsoft Common Objects in Context dataset is a large scale corpus whose primary goal is to train object-detection systems. Unfortunately, even though well structured, it lacks the question modality and is better suited for captioning tasks. The latter issue forced us to discard this dataset.
- **SQuAD** [37]: The Stanford Question Answering dataset is a reading comprehension corpus, consisting of questions posed by crowd-workers on several different articles. Extremely useful to train language models to Question Answering tasks, we dropped it for the lack of visual modality.
- **VQAv1** [19]: The Visual Question Answering dataset is composed of a set of open-domain questions on real-world images taken from MSCOCO. Each question is associated with a specific image and comes with several different annotations (i.e., answers). We discarded this dataset in favor of the latest version of it.
- **VQAv2** [21]: It is the evolution of VQAv1 and the corpus that we use in this work. Structured as VQAv1, it consists of a more balanced version of VQAv1 and has almost double the number of annotations both for the training and testing set.

5.1.1 VQAv2 Dataset

As previously mentioned, the Visual Question Answering v2 dataset is the corpus that we use for our work. Based on real-world images, it was created to help the community developing VQA systems that are able to reason on question/image pairs.

Each pair comes with several different annotations (i.e., answers) that act as ground truth; as we will discuss later, we combine the latter ones with the questions, and finally, we train our systems. Even though the annotations are, in some cases, a bit short-ended, we thought that this would be the best corpora to use due to its structure and simplicity to use.

The dataset is divided into three splits, and its structure is reported in Table IV. For any question, there are 10 different annotations (i.e., ground truths), and for this reason, the number of answers shown in Table IV is always ten times the number of questions.

For computational reasons and memory limitations, multiple questions might refer to a single image. Yet the dataset remains well balanced.

TABLE IV: VQA DATASET STRUCTURE

Split	Number of questions	Number of answers	Number of images
Training	443,758	4,437,580	82,783
Validation	214,354	2,143,540	40,504
Testing	447,793	0	81,434

In this work, we make use only of the training and validation splits since they are sufficient to achieve our goal.

A key aspect of VQAv2 is that it is properly balanced. For instance, given a question that begins with “what color”, the answers that span across the most common colors, such

as “red”, “green”, “blue” are well balanced. This is important because if the answers to this kind of question referred only to, say, “orange”, it would have meant that an intrinsic bias was present in the data. If this was the case, any system trained on such a corpus wouldn’t be able to generalize well and, when asked a question that begins with “what color”, it could always output the same biased answer (in this case “orange”).



Figure 17: Example of a complementary image pair. Each question has two similar images with different answers to the question to reduce biases in the data. Images taken from [2]

Furthermore, the authors [21] of the corpus provide similar images with the same questions but different answers to help reduce these biases even more; an example of such concept is depicted in Figure 17.

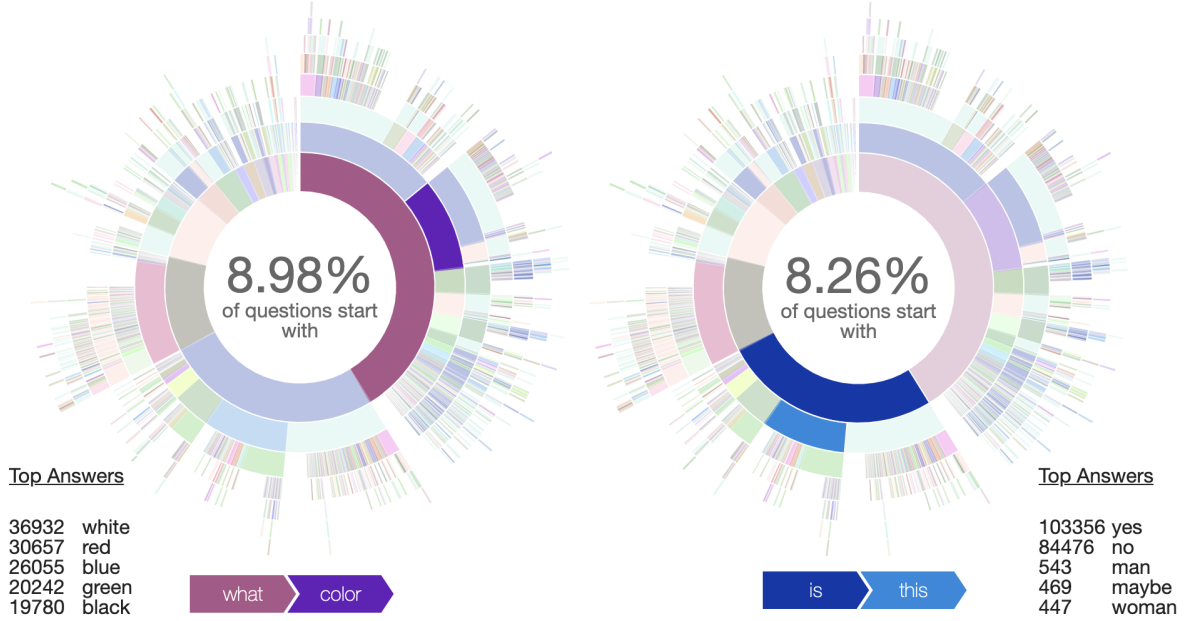


Figure 18: Example of the percentage of questions that begin with "what color" and "is this" in the VQAv2 dataset. As the image highlights, the top answers in both cases are really well-balanced.

The dataset was balanced to account for several different types of questions that are very likely to be subject to biases, such as those whose answers are "yes" and "no". In Figure 18, we report the top answers for two specific types of questions, and, as depicted in the image, not only the question types are balanced between themselves, but their relative answers are properly distributed too.

5.1.2 Preprocessing

Since this dataset is huge and the computational power at our disposal is limited, we had to select a subset of the corpus to train our models in a reasonable amount of time.

The data is already properly formatted, with garbage characters neither in the questions nor in the annotations. For this reason, we didn't have to perform any sanitization. Furthermore, in our language models, with the only exception for one baseline, we decided to keep the punctuation, hoping that they could understand these patterns and exploit them to generate open-ended questions.

To select our candidates, these are the main steps that we followed:

1. **Grayscale removal:** First and foremost, we dropped all samples which were associated with non-RGB images. Since we had plenty of data, we decided to keep only questions on colored pictures.
2. **Longer answer first:** Since we were interested in creating models that generate open-ended answers (i.e. longer answers), we decided to prioritize those samples whose annotations were longer and gave less importance to those which had concise answers.
3. **Limited annotations:** The idea of training our models on all the questions, exploiting all the 10 different ground truths per question, was unfeasible. For this reason, we randomly select up to 4 annotations for each question. This ensures that at most 4 identical question/image pairs will be fed to the model, with 4 different answers.
4. **Optimization:** Since, as we will discuss later, there are issues related to padding and performances, we remove sequences (question + answer) whose length-frequency is under a specific threshold (1000). In the end, this allows us to feed the model bigger batches of shorter sequences, boosting training times. To provide the reader with an example, it might help thinking about having a single sequence of 100 tokens in our data. If this was

the case, every other sequence would have been padded to match this size, resulting in extremely big and sparse tensors being fed to the model.

These steps are performed before everything else, and account for the creation of two pre-processed corpora (Training and Testing) that contain the most interesting samples for our needs; from now on, we will refer to these two sets as **TR-Base** and **TS-Base**.

As we will discuss soon, these two sets are far from being ready to be used with our models for these reasons:

- **Multi-Modality:** We experiment with models that accept multi-modal inputs: Question-only, Image-Only and Question+Image.
- **Sequence processing:** The models we consider encode the textual sequences with different algorithms such as NLTK word tokenizer [38], WordPiece encoding [15] and Byte-Pair encoding [5].
- **Multiple Inputs:** Depending on the model, we usually have to provide additional information. For instance, BERT, aside from the sequence, expects even token type ids and input mask.

It is evident that we couldn't have a single and unified dataset to work with; this issue forced us to create dynamic sub-datasets starting from **TR-Base** and **TS-Base**, depending on the considered system.

Later in this chapter, while going through each model, we will discuss how we compute each sub-dataset.

However, before getting into details, we highlight another key difference between **TR-Base** and **TS-Base**: while each sample in **TR-Base** has an associated answer, **TS-Base** has no answers at all. This distinction is explained as follows:

1. While training, we perform Professor Forcing [39], and for this reason, the answer should always be contained in the input. Briefly, Professor Forcing consists in training the system to predict the input sequence shifted ahead in time by one.
2. While evaluating, we perform Beam Search, and we do not want our answers to be in the input. Thus, we remove all the answers from **TS-Base** and create a separate file **TS-Map** that maps every sample in **TS-Base** with all its ground truths. Later, this map will be used to evaluate the model output.

To better understand how **TR-Base**, **TS-Base** and **TS-Map** are made, we report an example here:

- **TR-Base Sample**: [...,[Sample ID, Question, Image, Answer], ...]
- **TS-Base Sample**: [..., [Sample ID, Question, Image], ...]
- **TS-Map Sample**: {..., Sample ID: [Answer1, ..., Answer10], ...}

each answer in **TR-Base** is chosen randomly across the 10 different annotations.

Note that the sub-sets (**TS-Model**) that will be built starting from **TS-Base** will consist of the same exact samples, tokenized in different ways and with different modalities. This allows us to evaluate our systems on the same samples.

On the contrary, the sub-sets (**TR-Model**) built from **TR-Base** slightly differ since after tokenizing, to optimize training times, we drop sequences whose relative length frequency is below a threshold.

Now that we have discussed how we preprocess the data and what **TR-Base**, **TS-Base** and **TS-Map** are, we can introduce our baselines.

5.2 Baselines

To better evaluate our model, we decided to consider three different baselines that exploit only a limited set of modalities, plus a pre-trained VQA system. To generate answers, we propose one captioning model that makes use only of the visual input, two other systems that exploit only the question, without being allowed to look at the image, and a full VQA baseline that uses both the question and the image.

5.2.1 Captioning Baseline

Our captioning baseline was created to understand how much the textual input (i.e., the question) is essential to create a good VQA model. Conceptually, this is a VA (Video-Answer) system since it lacks the possibility to see the question.

This model has been built following the approach discussed in [40], and its implementation is based on a GitHub repository [41].

5.2.1.1 Model architecture

The architecture has been initially developed to create captioning systems and, on the latter task, it performs well. However, since our dataset is for VQA models, we didn't expect this baseline to achieve good performances.

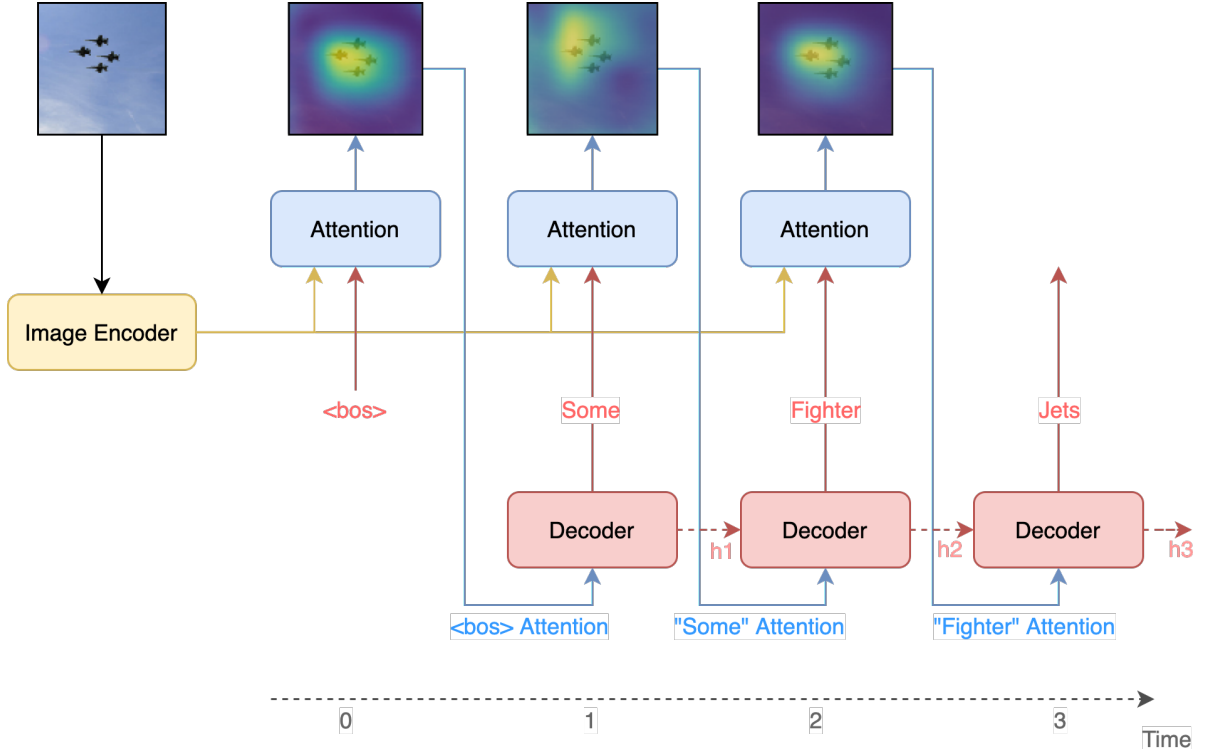


Figure 19: Captioning model structure. The picture shows how the three main components (Encoder, Decoder and Attention) work together to generate the output sequence.

The model is composed of three main blocks:

1. **Image Encoder:** This block encodes the input image $I \in \mathbb{R}^{3 \times 256 \times 256}$ into an encoded representation of multiple feature maps $M \in \mathbb{R}^{512 \times 7 \times 7}$. The encoder, made up with VGGNet-11 [16], gradually reduces the size of the image (from 256×256 to 7×7) while expanding the number of channels (from 3 to 512). The latter feature maps M are then fed to an attention layer side to side with the outputs of a decoder to perform attention on the image.

2. **Decoder:** This block is responsible for generating the output sequence (i.e. the answer).

It consists of a single LSTM cell of 256 neurons with an additional embedding layer of size 256, learned end-to-end within the training phase. At each time step i , the decoder, using the output of the attention layer A_{i-1} generated at the previous time step, takes care of producing the next token T_i .

3. **Attention:** This layer takes as input the encoded feature maps $M \in \mathbb{R}^{512 \times 7 \times 7}$ and, for each token T_i generated by the Decoder at time step i , distributes attention over the image in order to focus on the most relevant spots. Even though we will describe this layer more in detail later, being very similar to the attention mechanism employed in our model, it is worth noticing that this block takes care of bringing the two modalities to a common sub-space where they are merged together into a compact representation A_i . In turn, as described previously, the latter one is used as input to the decoder to generate the next token T_{i+1} .

A visual representation on this structure is reported in Figure 19. It is important to highlight that, at each time step i , the hidden state h_i of the Decoder is updated with the hidden state h_{i-1} of the previous time step (as seen in chapter 3). This allows the system to account for all the tokens generated at previous time steps and the associated attention maps.

5.2.1.2 Dataset

The dataset used to train this model is built starting from **TR-Base** and **TS-Base**. To generate the Training **TR-Cap** and Testing **TS-Cap** sub-sets, we first drop the question modality, since this system is meant for captioning. Afterward, we encode the answers in **TR-Cap** using

the NLTK [38] word tokenizer (we employ BPE tokenization only for transformer-based architectures), dropping all the punctuation to speed up the training times. **TS-Cap**, as explained before, does not contain any answer and consists only of the input images.

We introduce three unique tokens to account for the beginning, end of the sentence, and out-of-vocabulary (OOV) words, which are, respectively, $\langle \textit{bos} \rangle$, $\langle \textit{eos} \rangle$ and $\langle \textit{unk} \rangle$.

Furthermore, in order not to end up with an enormous vocabulary, every word in the corpus whose frequency is less than a threshold (30) is replaced with the $\langle \textit{unk} \rangle$ token. Finally, using Dictionary Lookup Encoding (2.2.2.2.1), we convert the tokenized sequences to vectors.

The images are resized to a fixed size of 256×256 pixels each in RGB mode and then are normalized to meet the VGGNet input requirements. This last step is mandatory since we use a pre-trained version of VGGNet-11, which has been trained with normalized input images. The resize operation takes place by randomly cropping portions of the image.

An example for the samples in **TR-Cap** and **TS-Cap** is hereby reported:

- **TR-Cap Sample:** [...,[Sample ID, Image, Answer], ...]
- **TS-Cap Sample:** [..., [Sample ID, Image], ...]

at evaluation time, we decode the model output and compare it against the references in **TS-Map**.

The final corpus size remains unchanged, both for the training and testing steps, and is reported in Table V. After the processing, the vocabulary consists of 3520 tokens. In Table V we report some statistics: **Size** = number of elements in each split, **Seq Len** = Sequence

TABLE V: CAPTIONING BASELINE DATASET STRUCTURE.

	TR-Cap	TS-Cap
Size	10^6	10^5
Seq Len	10	-
Min QL	-	-
Max QL	-	-
Avg QL	-	-
Min AL	3	-
Max AL	10	-
Avg AL	3.57	-
Avg $\langle pad \rangle$	6.42	-
Uniq Img	71.440	32.622

length, **QL** = Question Length, **AL** = Answer Length, **Avg $\langle pad \rangle$** = Average number of $\langle pad \rangle$ tokens in each sequence, **Uniq Img** = Number of Unique Images in the split.

5.2.1.3 Training

We train this model using a technique called Professor Forcing [39] for 10 epochs with batches of size 192. As optimizer, we employ an adaptive momentum technique called Adam [42] with an initial learning rate set to $4e-4$. Dropout is set to 0.4, and the loss is computed with a doubly stochastic attention regularization that updates the gradient using *CrossEntropy* on the output sequence with a regularization factor computed starting from the attention maps.

The training takes approximately 10 hours on a machine with an RTX 2070. We will evaluate this baseline together with the other models in chapter 6.

5.2.2 GPT-2 Answering Baseline

The second baseline that we built is made up of a pre-trained GPT-2 [6] Transformer. In this case, we get rid of the visual modality and see how this system is able to answer questions just exploiting its language biases.

GPT-2 was trained on an enormous corpus, and, for this reason, can perform several different downstream NLP tasks, as described in section 3.1.3. Generally speaking, this model can already answer a multitude of questions correctly, but, as we will discuss while evaluating this baseline, the visual modality is vital to answer image-specific questions.

Since our final model is built on top of GPT-2, this baseline is handy to say whether or not it uses the visual modality. Without following this approach, we couldn't know if the answer is generated by intrinsic information that comes in the pre-trained model or not.

5.2.2.1 Architecture

The architecture follows a traditional Transformer structure except for the lack of a Decoder. As discussed in the paper, this model can perform several different tasks without an Encoder-Decoder structure, since it is possible to condition the task directly in the input sequence.

Conceptually this means that if we intend to use GPT-2 as a translator from, say, English to French, we have to condition the input like:

$$\textit{“English sentence } A = \textit{French sentence } A, \textit{ English sentence } B = \textit{”}$$

and the model will output the translation for sentence B.

We exploit this conditioning capability during the training phase by giving the model batches of sequences where each sequence is composed of the concatenation of Question and Answer.

With this approach, the model understands that it has to perform a QA task and, as we will discuss later, produces answers which are based only on the input question, since there is no input image.

We use the smallest pre-trained architecture with $L = 12$ encoder layers and $h = 12$ attention heads. Each token is embedded with 768 dimensional vectors, which corresponds even to the hidden size of the Transformer. The model, trained solely on English corpora, totals roughly 117M parameters.

5.2.2.2 Dataset

We create our Training set **TR-GPT2** from **TR-Base** by dropping the visual modality and keeping only question and answer pairs; we subsequently concatenate each pair together to form a sequence of words.

The Testing set **TS-GPT2** is computed from **TS-Base** by dropping the images and keeping the questions only, since they are the ones we are going to use in the evaluation phase.

Later, we process the sequences in **TR-GPT2** and **TS-GPT2** using a BPE tokenizer specifically designed for GPT-2.

We introduce four new tokens: $\langle \textit{bos} \rangle$, $\langle \textit{eos} \rangle$, $\langle \textit{sep} \rangle$, and $\langle \textit{pad} \rangle$ to indicate the beginning and end of the sequence, the separation between question and answer and finally the padding token.

In **TR-GPT2**, we make use of all the four tokens while encoding the input sequence; Contrarily, in **TS-GPT2**, we use $\langle \textit{bos} \rangle$ and $\langle \textit{sep} \rangle$ since we don't have an answer.

For example, given a question **Q** and an answer **A**, a sequence **S** in **TR-GPT2**, before being encoded, is created as follows:

- **Q**: “*What is it?*”
- **A**: “*A dog*”

S: [`<bos>`, 'What', 'is', 'it', '?', `<sep>`, 'A', 'dog', `<eos>`, `<pad>`, `<pad>`]

the latter example assumes a padding size of 11.

In **TS-GPT2**, we perform the same operation but without the answer, thus effectively inserting only `<bos>` and `<sep>`.

Afterward, the sequences in **TR-GPT2** get padded to a fixed size in order to be batched together.

The final vocabulary contains 50,254 different BPE-encoded words: 50,250 are taken from the pre-trained architecture, and 4 (the new tokens) are manually inserted. Table VI presents some statistics about the dataset: **Size** = number of elements in each split, **Seq Len** = Sequence length, **QL** = Question Length, **AL** = Answer Length, **Avg <pad>** = Average number of `<pad>` tokens in each sequence, **Uniq Img** = Number of Unique Images in the split. Note that in the testing set we do no pad sequences.

5.2.2.3 Training

We fine-tune this baseline for a total of 3 epochs using batches of 64 elements each. Once again, we employ Adam [42] as optimizer and initialize the learning rate to $5e - 5$, as sug-

TABLE VI: GPT-2 BASELINE DATASET STRUCTURE.

	TR-GPT2	TS-GPT2
Size	10^6	10^5
Seq Len	24	22
Min QL	5	6
Max QL	21	22
Avg QL	8.45	9.48
Min AL	3	-
Max AL	18	-
Avg AL	3.89	-
Avg $\langle pad \rangle$	11.65	-
Uniq Img	-	-

gested in the paper. Even in this case, we compute the loss through Professor Forcing with *CrossEntropy*.

We do not train the model for too long since we don't want to damage its linguistical capability with our small dataset and are solely interested in teaching it how to use the newly introduced tokens. Furthermore, as we will soon discuss, three epochs are more than enough to fine tune this architecture.

On an RTX 2070 the training takes approximately 12 hours.

As we will discuss later, it is crucial to train the system to generate the $\langle eos \rangle$ token when it has finished producing the output, especially while Beam Searching the answers.

5.2.3 BERT Answering Baseline

Curious about how BERT would have compared to GPT-2 in this task, we decided to implement a second answering baseline (QA only). More specifically, we were interested in knowing whether or not the bi-directionality that comes with BERT was capable of boosting performances in this kind of task.

However, as we will see in the evaluation section, this is not the case, and we suspect that training a bi-directional model with sequences that contain both the question and the answer cannot work. The latter statement comes from the fact that if the model is allowed to see what tokens are on the right of the one currently being generated, it will never understand how to predict them.

Conceptually speaking, if during the training phase the system is allowed to see the target sequence, it will never learn how to generate it. As we will discuss, BERT is learning only to shift in time the input sequence by 1, and during the evaluation phase, where we feed only the question, it doesn't know what to do.

5.2.3.1 Architecture

Without going into the details of how the bi-directionality is implemented in this Transformer, well described in the paper [14], we report only that we take a pre-trained version of BERT for Masked Language Model (see section 3.1.4) with the most similar configuration to the other answering baseline. Once again, we have $L = 12$ layers with $h = 12$ heads each, both the embedded vectors and the hidden size is equal to 768 and, in total, the model consists of 110M parameters.

5.2.3.2 Dataset

The dataset (**TR-BERT** & **TS-BERT**) is built similarly to the one introduced for the GPT-2 baseline (see section 5.2.2.2). The real difference here is that we use a different tool for tokenizing and encoding the sequences: a WordPiece tokenizer [15] specifically built for BERT.

Moreover, we do not introduce any new token and exploit the ones that come with BERT, following the advice of the paper [14]. These tokens are $\langle \textit{cls} \rangle$ and $\langle \textit{sep} \rangle$ and can be used to achieve a multitude of NLP tasks.

To better understand the difference between BERT and GPT-2 inputs, considering the question **Q** and answer **A** of the example introduced in section 5.2.2.2, each sequence is processed as follows:

S: [$\langle \textit{cls} \rangle$, 'What', 'is', 'it', '?', $\langle \textit{sep} \rangle$, 'A', 'dog', $\langle \textit{sep} \rangle$, $\langle \textit{pad} \rangle$, $\langle \textit{pad} \rangle$]

In section 5.1.2 we said that some models required additional inputs to work correctly. BERT represents one of these exceptions and requires us to generate, for each sequence, two additional vectors: the Token Type Ids and the Attention Mask. These two additional pieces of information are essential for training BERT correctly because they tell the model which parts of the sequence make up the question and the answer, and which tokens can be ignored while computing the Multi-Head attention, like $\langle \textit{pad} \rangle$.

Sticking with our example, the two additional vectors are computed as follows:

S: [$\langle \textit{cls} \rangle$, 'What', 'is', 'it', '?', $\langle \textit{sep} \rangle$, 'A', 'dog', $\langle \textit{sep} \rangle$, $\langle \textit{pad} \rangle$, $\langle \textit{pad} \rangle$]

Token Type Ids: [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1]

Attention mask: [1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0]

On the one hand, in the token type ids vector, each 0 corresponds to a token in the question, whereas 1 means that the token belongs to the answer. This helps the model understanding the difference between question and answer. The attention mask vector, on the other hand, tells BERT on which tokens it should compute the attention. All the tokens with a corresponding value of 0 in this vector are ignored, like the $\langle pad \rangle$. Table VII reports some metrics about this dataset: **Size** = number of elements in each split, **Seq Len** = Sequence length, **QL** = Question Length, **AL** = Answer Length, **Avg $\langle pad \rangle$** = Average number of $\langle pad \rangle$ tokens in each sequence, **Uniq Img** = Number of Unique Images in the split. Note that in the testing set we do no pad sequences.

TABLE VII: BERT BASELINE DATASET STRUCTURE.

	TR-BERT	TS-BERT
Size	10^6	10^5
Seq Len	24	23
Min QL	5	6
Max QL	21	23
Avg QL	8.48	9.52
Min AL	3	-
Max AL	18	-
Avg AL	3.71	-
Avg $\langle pad \rangle$	11.79	-
Uniq Img	-	-

For what concerns the vocabulary, we keep the one that comes with BERT, with 30.522 total WordPiece-encoded words.

5.2.3.3 Training

We train this model for a total of 10 epochs using batches of 20 elements on a RTX 2070 for roughly 14 hours. The usual optimizer is employed (Adam), and the learning rate is set to $5e - 5$. The loss is computed in the same way as in the GPT-2 baseline. As for the other systems, we will discuss the results in the next chapter.

5.2.4 VQA Baseline

So far, we considered baselines that make use only of one single modality at a time. However, we thought that it would have been a good approach to compare our final architecture with another system that exploits both visual and textual inputs.

For this very reason, we implemented a fourth baseline that uses questions and images to generate answers. This model is the implementation of what is discussed in [43], and we were able to reproduce results using the code contained within Cyanogenoid’s repository [44]. It is important to highlight that this is a **strong baseline** (as reported in the title of the paper), and achieves very high performances.

We do not re-train or fine-tune this architecture on our corpus since we took a pre-trained version run on the VQAv2 dataset. In other words, this model was ready to use out-of-the-box without any need for fine-tuning.

As we have already discussed, the usual approach in the VQA community is to address the problem as if it was a classification task, and this system makes no exception. The model is

indeed trained to distribute probabilities over a predefined set of possible answers. For this reason, as we will discuss in the next chapter, the accuracy of the system on exact matches is higher with respect to our architecture, but it is definitely a much more limited architecture since it can generate only a predefined set of answers.

5.2.4.1 Architecture

The architecture consists of four main components:

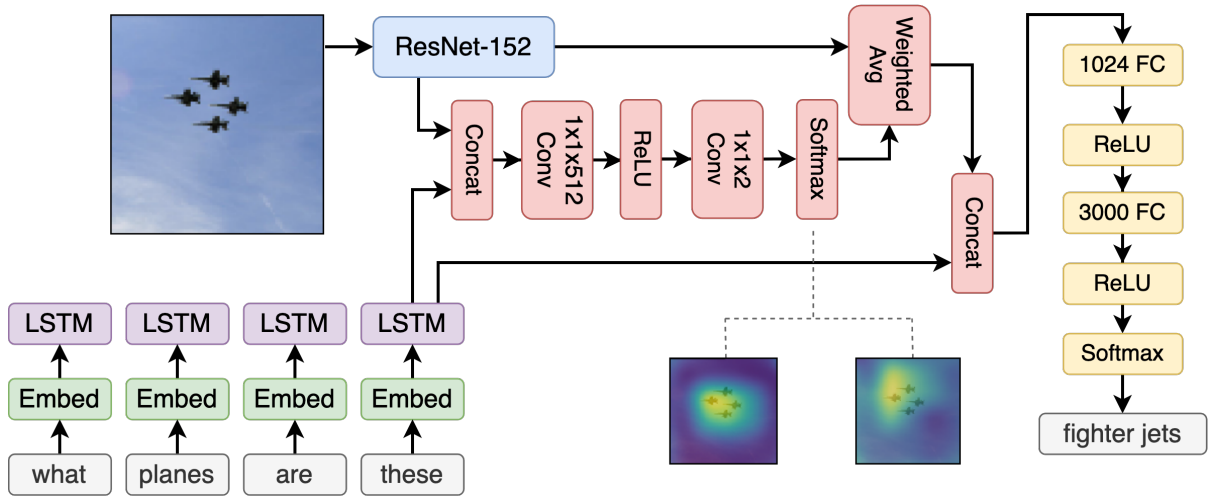


Figure 20: The picture shows the architecture of the VQA Baseline introduced in [43].

1. **Image Encoder:** As usual, this block extracts visual features using a CNN. In this case, the authors of [43] exploit a pre-trained version of ResNet [18] where the last pooling layer returns 2048 maps of size 14×14 . In Figure 20 we highlight this block in blue.

2. **Question Encoder:** In order to get a compact representation of what the question is about, this system uses an LSTM unit with 1024 neurons. The LSTM is fed with the input sequence where each token has been embedded into a 300-dimensional vector, learned end-to-end during training. In Figure 20 the LSTM is unrolled over time with a purple color.
3. **Stacked attention:** This architecture makes use of an attention layer to merge the two modalities together. The authors concatenate the features extracted from the CNN and the LSTM together, and then they pass the resulting vector through a series of convolutions and non-linear activations followed by a *Softmax* function. The output of ResNet is then averaged with the values computed by the *Softmax* and is then concatenated back with the features extracted by the LSTM. In Figure 20 we highlight attention operations with red.
4. **Classifier:** The output of the attention layer is passed through two separate fully-connected layers (yellow blocks in Figure 20), with non-linear activations in-between, which are then followed by a *Softmax* that distributes probabilities over a fixed set of candidate answers (3000 possible choices).

This architecture, even though not very complicated, serves as a good baseline for our experiments, producing meaningful results.

5.2.5 Dataset

Following Cyanogenoid’s implementation [44], we build two different vocabularies (**Vocab-Q** and **Vocab-A**) that contain, respectively, all the possible words in the questions and all the possible answers in the annotations.

Afterwards, we create **TS-VQA-Baseline** processing every question in **TS-Base** by word-tokenizing and encoding it with the corresponding word-id in **Vocab-Q** (lookup encoding). At this point, every question is a vector of numbers. Finally, every question is padded to the longest possible sequence with a *<pad>* token.

At the end of this process, mandatory to use the pre-trained checkpoint downloaded from [44], **Vocab-Q** totals 15192 words and **Vocab-A** is made up by 3000 possible answers. Once again, to ensure consistent results, **TS-VQA-Baseline** is built starting from **TS-Base** and contains 100K samples.

5.3 Proposed architecture - VGGPT-2

After having discussed the corpus and the baselines in use, in this section, we present the architecture that we introduce to address the VQA problem, namely, VGGPT-2.

As we will shortly discuss, the key difference of this architecture from the ones discussed in the related works is that it is capable of generating open-ended answers. In other words, we do not distribute probabilities over a predefined set of answers but exploit the power of a language model (GPT-2) to generate arbitrarily long sequences.

Furthermore, we contribute to the scientific community by creating a previously-unseen architecture that combines Transformers with CNNs in a way that, at the time of writing this work, has never been tried before.

This model intensely relies on attention mechanisms at multiple levels and is capable of combining the information contained in the question to look at specific portions of the image. At the same time, the information contained in the interesting spots of the image are used to drive the generation of the answer, conditioned even by the question itself.

5.3.1 Architecture

This model is composed of four high-level components that work together to process both the visual and textual modalities and generate an open-ended answer, which can briefly be described as:

1. **Image Encoder:** Extracts image features and feeds them to the attention mechanism.
2. **Language Model:** Extracts features from the question and shares them with the attention mechanism.

3. **Attention Mechanism:** Using the features extracted from the input, generates an attention vector.
4. **Answer Generator:** Concatenates the language model output with the attention output to generate the answer.

in Figure 21 we show how these components are connected together.

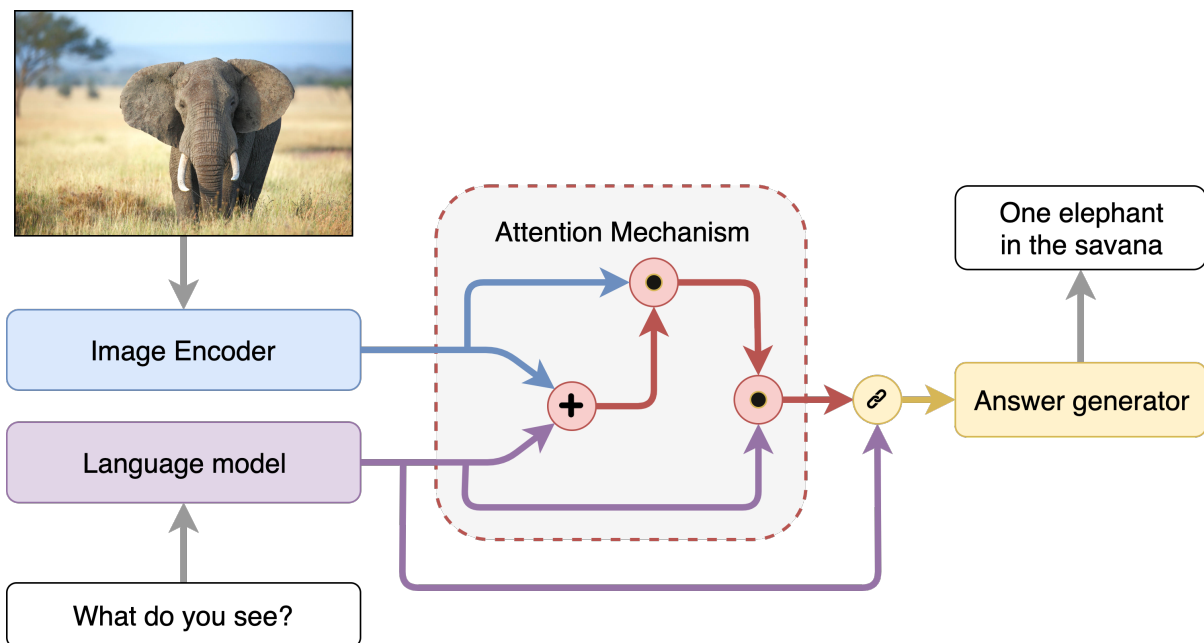


Figure 21: VGGPT-2 Model structure. The picture shows, from an extremely high point of view, how the four main components (Image Encoder, Language Model, Attention Mechanism, and Answer Generator) are combined together to generate the answer. Note that the operations inside the circles are representative, and the vectors are not combined directly, as depicted above.

We now describe each component in detail and how it interacts with the others.

5.3.2 Image Encoder

Since we expect the questions to be about a specific image, we first need a way to extract information about the visual input. For this reason, we decide to embed in our architecture an Image Encoder, which is made starting from VGGNet-11.

It has been proved that very deep CNNs perform particularly well in extracting high-level features from images, and, for this reason, we decide to take a pre-trained architecture specifically designed to achieve such a task.

As we have already discussed in section 3.2.2, VGGNet-11 is a CNN with 11 subsequent weight layers: the first 8 consist of convolutions and pooling operations, whereas the last 3 are a sequence of fully-connected layers that act as a classifier.

In the classifier, the first two layers, of size 4096, flatten the output of the previous 8 layers, and the last distributes probability over the 1000 different classes present in ImageNet [17], the dataset on which VGGNet was trained on.

Since, in our context, we do not need VGGNet to act as a classifier, we follow the best-practices and drop the last three fully connected layers. In this way, given an image, the encoder produces a set of maps that contain high-level extracted features.

We now introduce some notation to better understand how our Image Encoder works. The input consists of an Image $\mathbf{I} \in \mathbb{R}^{c_i \times w_i \times h_i}$, with $c_i = 3$ different channels (RGB) of width $w_i = 224$ and height $h_i = 224$. The output of the network, after going through the 8 layers described in Table VIII, consist of a set of maps $\mathbf{M} \in \mathbb{R}^{c_o \times w_o \times h_o}$, with $c_o = 512$ different channels of width

$w_o = 7$ and height $h_o = 7$. \mathbf{M} will be one of the two inputs to our Attention Mechanism. As the image depicts, the network performs a series of convolutions and max-pooling operations to gradually extract the visual features.

TABLE VIII: STRUCTURE OF VGGNET-11 WITHOUT FINAL CLASSIFIER.

Layer number	Operations
1	3D-Convolution, 64 channels Max-Pooling
2	3D-Convolution, 128 channels Max-Pooling
3	3D-Convolution, 256 channels
4	3D-Convolution, 256 channels Max-Pooling
5	3D-Convolution, 512 channels
6	3D-Convolution, 512 channels Max-Pooling
7	3D-Convolution, 512 channels
8	3D-Convolution, 512 channels Max-Pooling

5.3.3 Language Model

As anticipated, we take as Language Model the OpenAI’s GPT-2 (small, 117M) transformer. Once again, we take this pre-trained architecture, and we refine it in order to integrate the latter in our system.

GPT-2 consists of two core blocks: the first one is the Transformer, which is responsible for computing the multi-head attention discussed previously, and the second one is the final classifier which, for each token processed by the Transformer, produces a distribution of probabilities over the vocabulary of words (recall that this is a language model that exploits attention, which means that the probability $p(w_i|t_i, t_{i-1}, \dots, t_{i-k})$ of generating the word w_i at time step i is conditioned on the input tokens $t_i, t_{i-1}, \dots, t_{i-k}$ up to time step i).

Since our goal is to have the Language Model interacting with the Image Encoder, we split apart these two blocks and use them separately in our model. Given a sequence of n input tokens $\mathbf{S} \in \mathbb{R}^{n \times 1}$, the output of the transformer $\mathbf{T} \in \mathbb{R}^{n \times \text{hidden}}$ consists of a sequence n hidden vectors of size $\text{hidden} = 768$. We use \mathbf{T} as the second input to the Attention Mechanism, while the final classifier \mathbf{C} (also known as **head**) will be discussed later in the Answer Generation block.

5.3.4 Attention

This block is the most important part of our full implementation. It is capable of combining features coming from two, totally different sub-spaces into a single representation. In this part of the architecture, we effectively see in action a Co-Attention mechanism since we use the question to drive the focus on the image and vice versa.

In order to understand how this block works, let's recall what its inputs are:

- **Image Encoder Output:** a vector $\mathbf{M} \in \mathbb{R}^{512 \times 7 \times 7}$
- **Language Model Output:** a vector $\mathbf{T} \in \mathbb{R}^{n \times 768}$.

The first operation consists in flattening \mathbf{M} into a more convenient representation $\mathbf{M}_{\text{flat}} \in \mathbb{R}^{512 \times 49}$.

Then, both \mathbf{M}_{flat} and \mathbf{T} are fed to two separate Linear layers ($\mathbf{Lin}_{\mathbf{M}}$ and $\mathbf{Lin}_{\mathbf{T}}$) that bring the two vectors into a common subspace of size $AttDim = 512$:

- $\mathbf{Lin}_{\mathbf{M}}$ (# of params = $512 \times AttDim$): brings $\mathbf{M}_{\text{flat}} \in \mathbb{R}^{512 \times 49}$ into $\mathbf{M}_{\text{att}} \in \mathbb{R}^{AttDim \times 49}$
- $\mathbf{Lin}_{\mathbf{T}}$ (# of params = $768 \times AttDim$): brings $\mathbf{T} \in \mathbb{R}^{n \times 768}$ into $\mathbf{T}_{\text{att}} \in \mathbb{R}^{n \times AttDim}$

At this point, for every element $\mathbf{T}_{\text{att}}^i \in \mathbb{R}^{AttDim}$ in \mathbf{T}_{att} , with $i \in \mathbb{N} \wedge i \in [0, n)$, we repeat the following set of operations (it might help to check Figure 22 at each step):

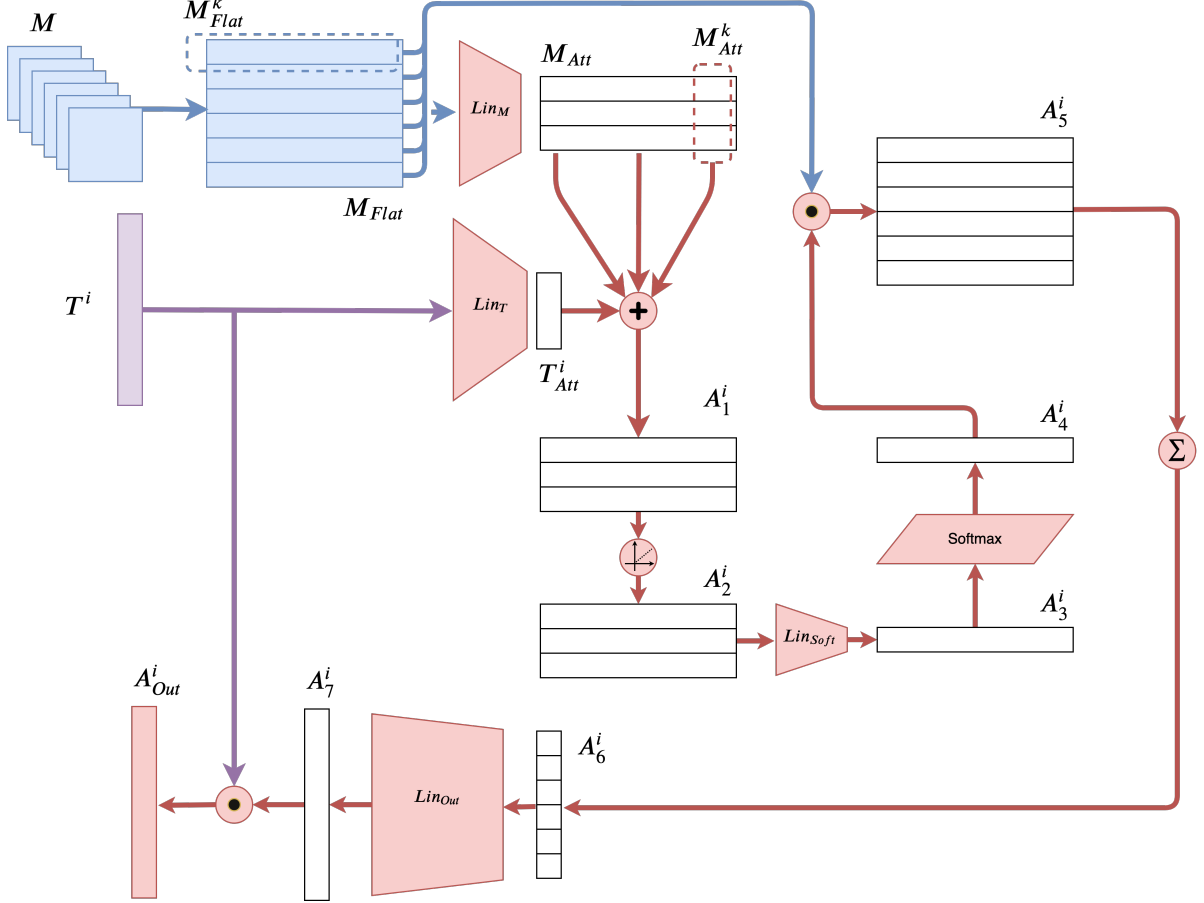


Figure 22: VGGPT-2 Attention Mechanism in detail. On the top left, in blue, there are the maps \mathbf{M} coming out from VGGNet-11. In purple we highlight \mathbf{T}^i , one of the n elements in \mathbf{T} . In practice the attention mechanism is always fed with \mathbf{T} but in this diagram, for the sake of simplicity, we consider just a single element. Finally, red is used to represent the flow of the attention operations.

1. We sum $\mathbf{T}_{\text{att}}^i$ to each element $\mathbf{M}_{\text{att}}^k \in \mathbb{R}^{AttDim}$ of \mathbf{M}_{att} , with $k \in \mathbb{N} \wedge k \in [0, 49)$, obtaining the first multi-modal attention representation $\mathbf{A}_1^i \in \mathbb{R}^{AttDim \times 49}$ for the i -th token in the input sequence.
2. We rectify \mathbf{A}_1^i using a **ReLU** unit and obtain an intermediate attention representation $\mathbf{A}_2^i \in \mathbb{R}^{AttDim \times 49}$.
3. Using a Linear layer (**Lin_{soft}**) we compress $AttDim$ down to 1. In other words, **Lin_{soft}** brings $\mathbf{A}_2^i \in \mathbb{R}^{AttDim \times 49}$ into $\mathbf{A}_3^i \in \mathbb{R}^{1 \times 49}$.
4. In order to obtain a softmax that highlights relevant spots of the image given a specific word at position i , we pass \mathbf{A}_3^i through a *Softmax* function and obtain $\mathbf{A}_4^i \in \mathbb{R}^{1 \times 49}$, where $\mathbf{A}_4^{i, h} \in (0, 1)$ with $h \in \mathbb{N} \wedge h \in [0, 49)$. Even though the mathematical notation might be confusing, at this point every pixel in \mathbf{A}_4^i has an associated value between 0 and 1, which indicates how much important is that pixel to answer the question.
5. Afterwards we perform a **point-wise multiplication** between every element $\mathbf{M}_{\text{flat}}^k \in \mathbb{R}^{1 \times 49}$ in the original flattened maps \mathbf{M}_{flat} with \mathbf{A}_4^i , obtaining a new vector $\mathbf{A}_5^i \in \mathbb{R}^{512 \times 49}$. In this step we perform the **Question-to-Image attention**, since we exploit the softmaxes computed at the previous step to mask out irrelevant portions of the image.
6. Later we sum, for each channel in \mathbf{A}_5^i , all the masked pixels together, reducing the dimension from 49 to 1. In other words, we go from $\mathbf{A}_5^i \in \mathbb{R}^{512 \times 49}$ to $\mathbf{A}_6^i \in \mathbb{R}^{512 \times 1}$ by summing all the elements together on the pixel axis.

7. We then bring \mathbf{A}_6^i back to the transformer space through a final Linear layer ($\mathbf{Lin}_{\text{out}}$), which expands $\mathbf{A}_6^i \in \mathbb{R}^{512 \times 1}$ to $\mathbf{A}_7^i \in \mathbb{R}^{768}$.
8. The last step computes the **Image-to-Question attention** through a final **point-wise multiplication** between \mathbf{A}_7^i and every element $\mathbf{T}^i \in \mathbb{R}^{768}$ of \mathbf{T} , which is the original output of the transformer before this layer. The final output of the Attention Mechanism for every embedded token \mathbf{T}^i and the maps \mathbf{M} coming from the Image Encoder is the vector $\mathbf{A}_{\text{out}}^i \in \mathbb{R}^{768}$.

We described, for the sake of simplicity, how the attention layer works considering one single embedded token \mathbf{T}^i at a time. However, now that we have discussed how the mechanism works for a single element, it is easy to scale it up with a sequence of n elements. The vectors in the attention layer just have an additional dimension that accounts for the token under consideration, resulting in: $\mathbf{A}_1 \in \mathbb{R}^{n \times \text{AttDim} \times 49}$, $\mathbf{A}_2 \in \mathbb{R}^{n \times \text{AttDim} \times 49}$, $\mathbf{A}_3 \in \mathbb{R}^{n \times 1 \times 49}$, $\mathbf{A}_4 \in \mathbb{R}^{n \times 1 \times 49}$, $\mathbf{A}_5 \in \mathbb{R}^{n \times 512 \times 49}$, $\mathbf{A}_6 \in \mathbb{R}^{n \times 512 \times 1}$, $\mathbf{A}_7 \in \mathbb{R}^{n \times 768}$ and $\mathbf{A}_{\text{out}} \in \mathbb{R}^{n \times 768}$.

The output \mathbf{A}_{out} of the Attention Mechanism, as we will discuss in the next section, is then combined back with the output of GPT-2 to generate the answer. This layer totals roughly 8M parameters.

The Attention Mechanism used in this model is very similar to the one used in our Captioning baseline, introduced in [40] and implemented in Sgrvinod's repository [41].

5.3.5 Answer Generator

We generate the final answer exploiting once again our language model. While most related works would have fed the output of the Attention Mechanism to a classifier, in order to distribute

probabilities over a predefined set of answers, we address the problem from another point of view.

We basically try to condition the generation of words in GPT-2 using the output of the Attention layer in a particular way; instead of feeding the output \mathbf{T} of the transformer directly to its **head**, we first perform a key operation: we take the Attention output \mathbf{A}_{out} and concatenate it with the transformer output \mathbf{T} into a new vector $\mathbf{O} \in \mathbb{R}^{n \times 1536}$. Then, with the help of a new Linear layer ($\mathbf{Lin}_{\text{Classifier}}$), we distribute probabilities over the original vocabulary of words of GPT-2. In order not to lose the valuable information contained in the weights of the pre-trained classifier \mathbf{C} (or **head**), we do the following:

1. We initialize the first 768×50254 weights of $\mathbf{Lin}_{\text{Classifier}}$ with the ones present in the original **head** of GPT-2.
2. We initialize the last 768×50254 weights, that account for the Attention layer output, to 0.

This allows the model to start training from a status where it is already able to generate sequences of words correctly since, in the beginning, all the weights relative to the Attention Mechanism are set to 0, and the only ones that influence the output are coming from \mathbf{T} . This last Linear layer consists of roughly 77.1M of parameters alone (1536×50254), so it is imperative to perform such initialization to train the system in a reasonable amount of time (≈ 50 hours on an RTX 2070 with 8GB of GDDR6).

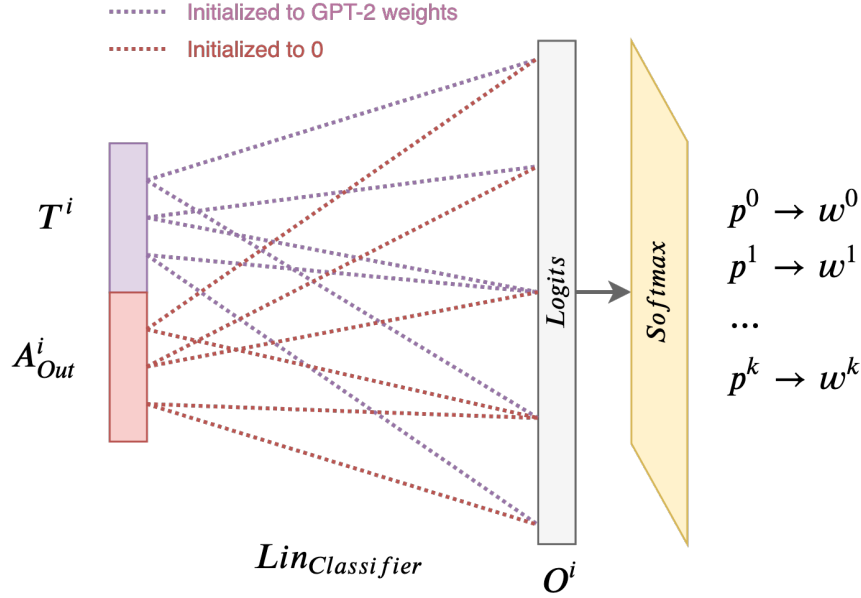


Figure 23: The image shows how each output \mathbf{T}^i of the Transformer is concatenated to its relative Attention layer output \mathbf{A}_{out}^i . We use different colors to indicate how we initialize the weights in LinClassifier .

Figure 23 shows how we concatenate the two vectors together: once again we consider a single element \mathbf{T}^i in \mathbf{T} with its corresponding attention output \mathbf{A}_{out}^i .

Afterward, we feed \mathbf{O} , also known as the **Logits** vector, to a softmax that distributes probabilities over our vocabulary. To generate the output word w^i corresponding to the input token at position i , we Beam Search the probability space with different beam sizes, as we will discuss in the next chapter.

What is essential to understand is that, instead of using a compact representation of the input question and the image to distribute probabilities over a fixed set of answers, we dynamically generate each word constantly conditioning it on the previously generated words.

At evaluation time, this allows the system to generate arbitrarily long answer since:

1. Given a Question of n words and an Image, we first let the system generate the initial word in the answer, conditioned only on the Question and the Image.
2. After we have generated the first word, we append it to the original question, which becomes a sequence of $n + 1$ words. This time, the second word in the answer gets generated conditioned on the new sequence.
3. The procedure goes on until the system outputs the $\langle eos \rangle$ token.

5.3.6 Dataset

The corpus used to train this architecture is the exact same one of our GPT-2 baseline (see section 5.2.2.2), with the only difference being the addition of the visual modality. We report in Table IX all the datasets side to side. The size of the sequences, question and answers varies depending on the tokenizer in use and the preprocessing thresholds: **Size** = number of elements in each split, **Seq Len** = Sequence length, **QL** = Question Length, **AL** = Answer Length, **Avg $\langle pad \rangle$** = Average number of $\langle pad \rangle$ tokens in each sequence, **Uniq Img** = Number of Unique Images in the split.

TABLE IX: BASELINE DATASETS VS VGGPT-2 DATASET.

	Captioning		GPT-2		BERT		VGGPT-2	
	Train	Test	Train	Test	Train	Test	Train	Test
Size	10^6	10^5	10^6	10^5	10^6	10^5	10^6	10^5
Seq Len	10	-	24	22	24	23	24	22
Min QL	-	-	5	6	5	6	5	6
Max QL	-	-	21	22	21	23	21	22
Avg QL	-	-	8.45	9.48	8.48	9.52	8.45	9.48
Min AL	3	-	3	-	3	-	3	-
Max AL	10	-	18	-	18	-	18	-
Avg AL	3.57	-	3.89	-	3.71	-	3.89	-
Avg $\langle pad \rangle$	6.42	0	11.65	0	11.79	0	11.65	0
Uniq Img	71440	32822	-	-	-	-	71450	32822

5.3.7 Training

We train the system for **20 epochs** using **batches of 20** elements. The learning rate is initialized to **5e-5** and **Adam** is employed as optimizer. Even in this case we exploit **Professor Forcing** with a **Cross-Entropy** loss function that ignores $\langle pad \rangle$ tokens. In Figure 24 we report the smoothed training loss, with a brief description.

The overall architecture, disabling VGGNet-11 weight updates, consists of 202M parameters since we fine-tune GPT-2 (117M) side to side with the Attention Mechanism (8M) and the final classifier (77M). Training took ≈ 50 hours on an NVIDIA RTX 2070 with 8GB of GDDR6 memory.

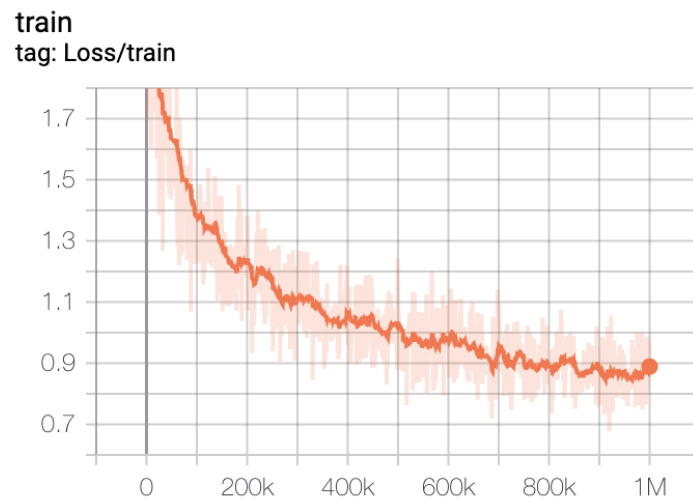


Figure 24: VGGPT-2 Train Loss: we plot the loss for each iteration (one batch) over the whole training duration. Each epoch consists of 50K iterations, for a total of 1M iterations ($50K \times 20$). We report the smoothed loss which descends to a value of ≈ 0.9

CHAPTER 6

EVALUATION

In the previous chapter we have introduced several different architectures, and we are now going to evaluate them under both a quantitative and qualitative point of view. Since we propose a different way of generating answers, numerical results are not enough to fully evaluate our results.

6.1 Answer Generation

Before getting into the evaluation process, we need to address one last detail that highlights how we obtain answers from our systems. Depending on the considered model, we follow two different approaches:

- To get answers from our Captioning, **BERT** and **GPT-2** Baseline as well as **VGGPT-2** we perform **Beam Search**:
 - Initially we feed the system a question \mathbf{Q} and let it generate a probability distribution over all the words in the vocabulary. At this point we keep the \mathbf{K} most probable words in this distribution w^1, \dots, w^k .
 - At the second time step we create \mathbf{K} new sequences $\mathbf{S}^{\mathbf{K}}$, where each sequence \mathbf{S}^i is created by concatenating \mathbf{Q} with w^i .
 - At every new time step we keep on pooling the \mathbf{K} most probable words out of the $\mathbf{K} \times \text{VocSize}$ ones generated by the $\mathbf{S}^{\mathbf{K}}$ sequences. For each new word, we keep track of

the source sequence and update $\mathbf{S}^{\mathbf{K}}$ by keeping only those sequences which generated the last most probable words.

- The process goes on until either a threshold is reached (in our case 20 words) or a stop word is generated (like the $\langle eos \rangle$ token).
- We experimented with different beam sizes (from 1 to 50) on a small sub-set of the validation corpus and discovered that increasing the beam size always leads to worst results. For this reason, we fix the beam size to 1 across all of our experiments. In other terms, we always pool the most probable words out of the ones generated by our systems, effectively performing a greedy search. All the results that we report are computed with beam size equal to 1. Future work will provide a comparison with bigger beam sizes, but for the time being, we stick to 1; this even because the evaluation process is prolonged due to the small batch size (=1) that we must use.
- On the contrary, for the **VQA-Baseline**, we always take the most probable output from the ones generated by the system. The latter architecture does not generate a sequence of words incrementally; it outputs a single sentence (the answer) directly. This procedure is straight-forward and can be parallelized very well (we use batches of 192 elements at a time).

6.2 Quantitative evaluation

In this section we report some numerical results that help visualize the performance differences among the proposed architectures. We evaluate our models using the following three

metrics: Accuracy, BLEU [45], and Word Mover’s Distance [46]. As we will discuss soon, these metrics are very different but help understanding how each system behaves.

6.2.1 Accuracy

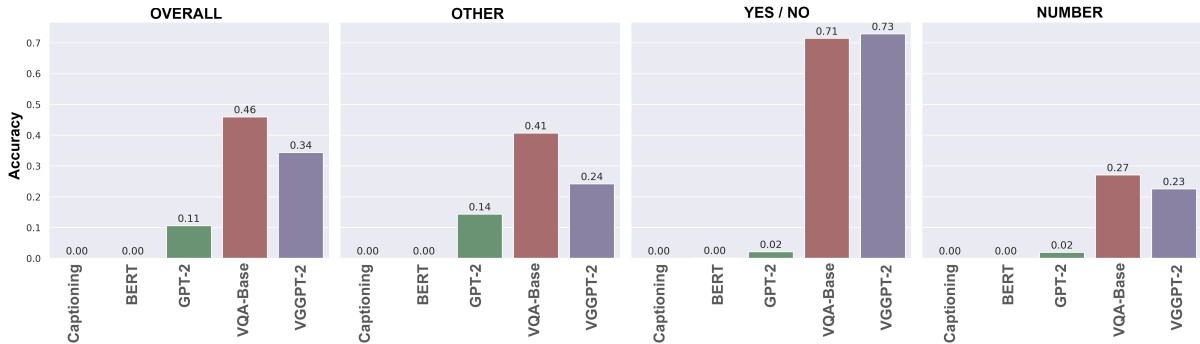


Figure 25: Evaluation Accuracy for the four main type of answers. **Overall** refers to the overall accuracy, **yes/no** to that of the binary answers, **number** to those which requires the system to count and finally **other** refers to the accuracy of the other type of answers.

We split the accuracy evaluation process into two parts, following what is usually done in VQA papers on the VQAv2 corpus. First, in Figure 25, we report results for the 4 most essential types of accuracy and then, as shown in Figure 27, we show which are the questions for which our models achieve the highest scores.

In Figure 25 we introduce the following 4 accuracy metrics:

1. **Overall accuracy:** this is the overall accuracy computed for all the answers in the validation set.

2. **Yes/No accuracy:** this is the accuracy on binary answers like “yes” and “no”.
3. **Numerical accuracy:** accuracy computed solely on answers which require the model to count objects or elements in the picture.
4. **Other accuracy:** accuracy computed on all the answers but Yes/No and Numerical ones.

As Figure 25 highlights, both the Captioning and BERT baseline consistently have an accuracy of 0, whereas VGGPT-2, that reaches an overall accuracy of **0.34**, compares well with the VQA-Baseline, which achieves an overall score of **0.46**. We were expecting these results for the Captioning baseline but not for BERT; further investigation will reveal that BERT is indeed not meant for language generation and thus fails in generating any answer.

Just by looking at these accuracies, it is evident that VGGPT-2 consistently beats GPT-2, confirming that the visual modality is indeed of great help to our architecture. Furthermore, VGGPT-2 is better than the VQA-Baseline when it comes to counting objects in the image, suggesting that it’s very deep architecture allows for better reasoning.

The big jump in performances in the “yes/no” type of answers between VGGPT-2 and GPT-2 is exciting, suggesting that GPT-2 is performing random guessing with these kinds of answers while VGGPT-2 has learned to reason quite well.

The motivation behind the low scores of both the Captioning and BERT systems might hide in the fact that they tend to generate very repetitive and incorrect answers. For instance, the Captioning system never understands when to stop generating words, consistently producing sequences of 20 tokens.

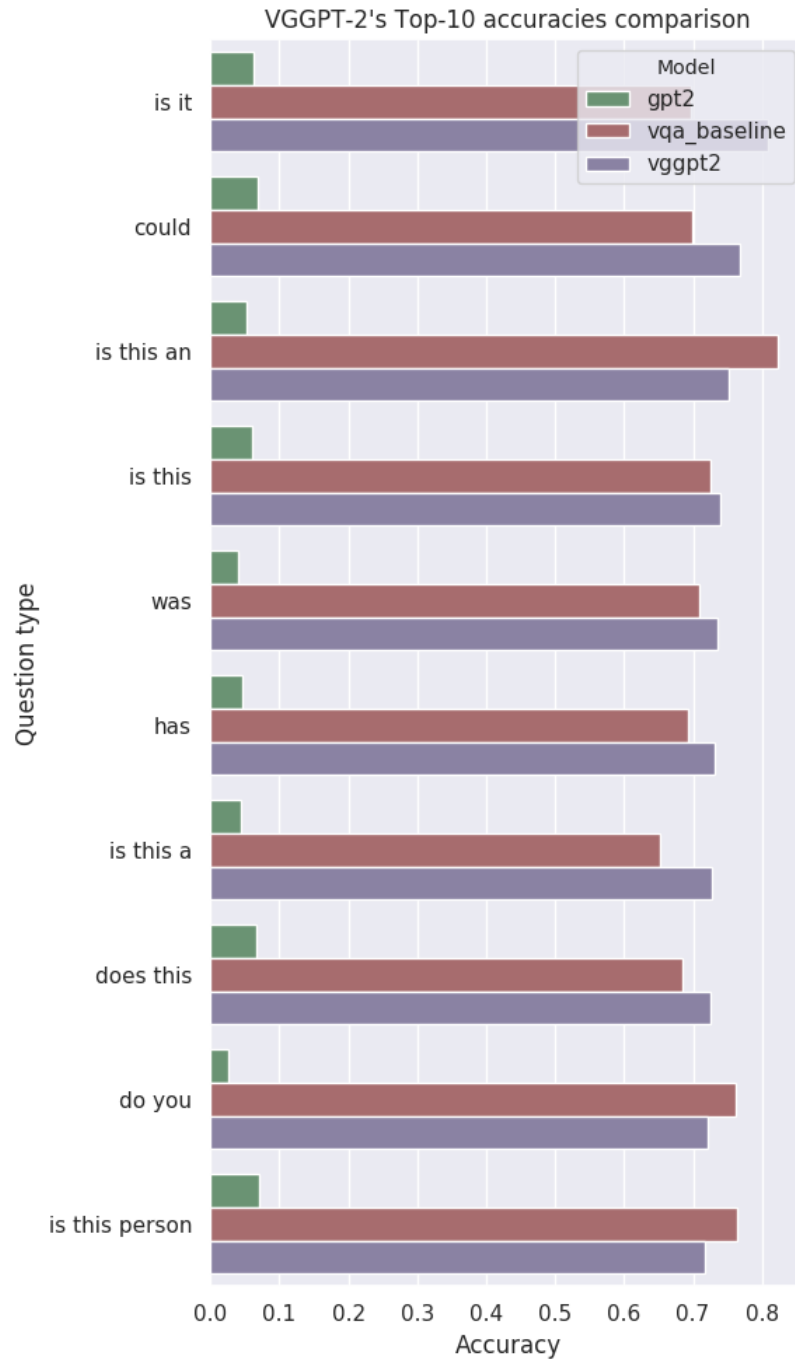


Figure 26: This picture shows how the best 10 Per-Question accuracies of VGGPT-2 compare with GPT-2 and the VQA-Baseline. We omit the Captioning system and BERT due to their poor performances. As the graph shows, VGGPT-2 compares well with our VQA-Baseline.

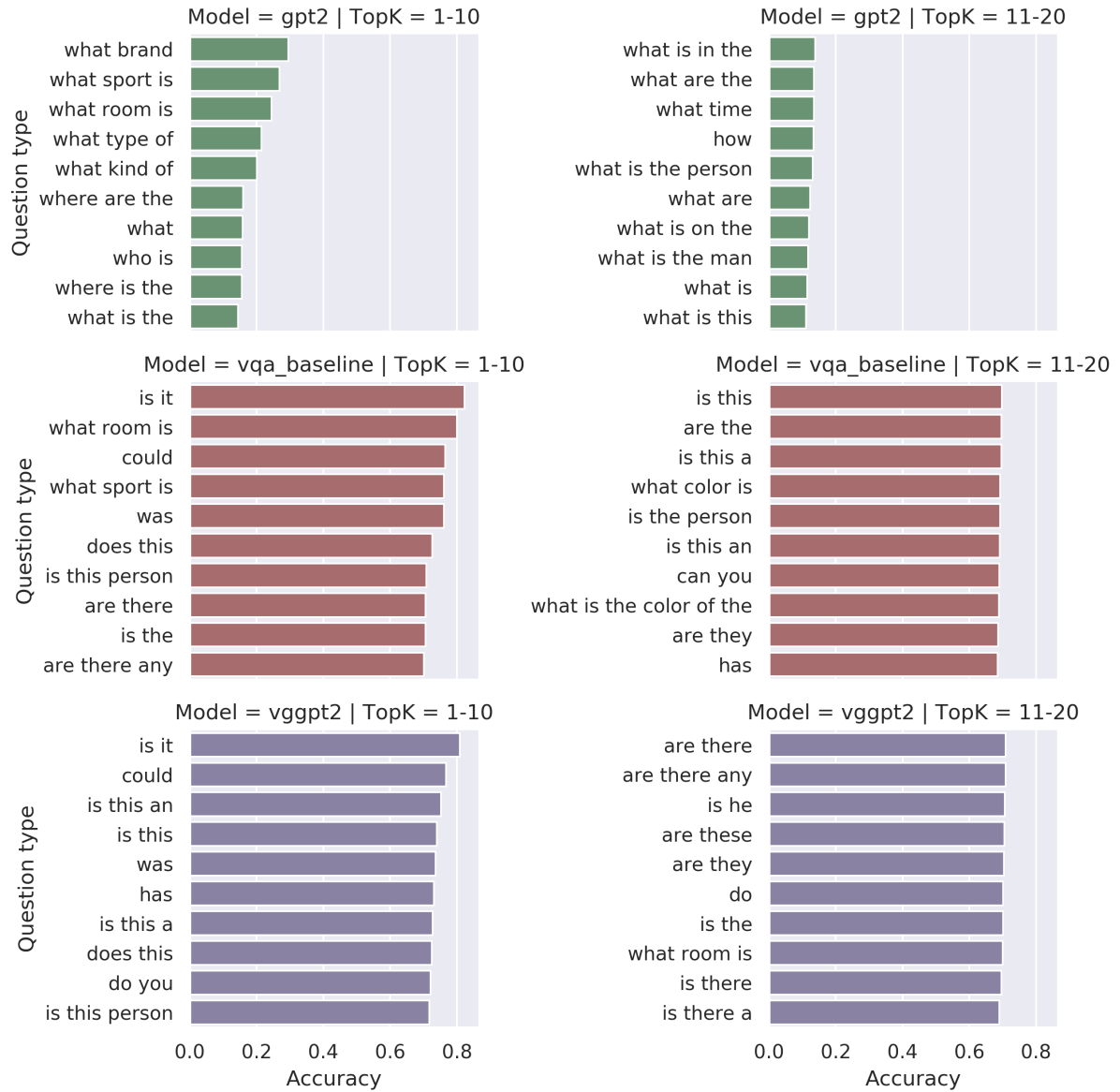


Figure 27: In this picture we report, individually, the 20 best Per-Question accuracies for GPT-2, VQA-Baseline, and VGGPT-2. On the left column we report the first best 10 accuracies, and in the right column the following 10.

Figure 26 shows how the top ten VGGPT-2’s per-question accuracies compare with GPT-2 and the VQA-Baseline and, in Figure 27, we report, for each model, what are the question types that lead to the best 20 accuracies.

As shown in Figure 27, GPT-2 can generate the answers solely exploiting its language biases: it achieves high accuracies whenever the answer can be predicted from the question. In contrast, the two VQA systems make use of the visual modality to generate outputs that ultimately lead to higher accuracies.

Both the VQA-Baseline and VGGPT-2 score high in the type of question that start with “Is it..”; this is consistent with the fact that their accuracy related to binary answer is high.

Even though the accuracy metric suggests that VGGPT-2 is not as powerful as the VQA-Baseline, we must recall that our system is meant to generate open-ended answers. This strongly affects the accuracy metric, significantly penalizing VGGPT-2 outputs. Overall we can say that our architecture is at least comparable with the VQA-Baseline, suggesting that we followed the right approach.

6.2.2 Bilingual Evaluation

Not satisfied by the results provided by the Accuracy metric, we decided to proceed and further investigate our results using the BLEU [45] metric.

Even though BLEU was originally developed for translation systems, it can be used for a series of NLP downstream tasks, such as evaluating the quality of the answers that our system generates. In fact, we use BLEU to check how similar our answers are to the ground truths.

When considering BLEU scores, a perfect match gives a value of 1, whereas total miss-match results in a 0. It is worth noticing that not even humans can achieve a consistent score of 1 since the translations (or, as in our case, the answers) might slightly differ from the references.

The metric works by counting **n-gram** overlaps between the generated output and the ground truths. The comparison is made regardless of word order in the sentences. The metric keeps track of the occurrences of reasonable words in the sentences. It automatically penalizes translations which consist of repetitions (this is referred to in the paper as “modified n-gram precision”).

To account for 0 n-gram counts, which can cause the BLEU score to be equal to zero, we experiment with 4 different smoothing functions present in the NLTK framework:

- **Add-epsilon:** the first smoothing technique consists in adding ϵ counts to precisions with 0 counts. The NLTK framework defaults to $\epsilon = 0.1$ and we keep this value without modifying it. In other words, if the counts for the unigram “dog” is 0, it becomes 0.1.
- **Add-one:** this methods consists in adding 1 to both numerator and denominator, regardless of the counts. This technique is discussed in [47].
- **NIST-Geometric:** in this case, as reported on the NLTK documentation [38], “smoothing is computed by taking $\frac{1}{2^k}$, instead of 0, for each precision score whose matching n-gram count is null”. k is set to 1 for the first n that have a 0 n-gram count. Once k is set to 1, each subsequent $(n+1)$ -gram will have $k = k + 1$. This means that if we have, for instance, no trigrams, k will be 0 for unigrams and bigrams, 1 for trigrams, and 2 for 4-grams.

- **Chen and Cherry:** the method described in [48] suggests to follow a different approach since, as reported on the NLTK documentation [38], “shorter translations may have inflated precision values due to having smaller denominators”. Instead of replacing null n-gram counts as in the NIST-Geometric approach, the paper suggests to divide by $\frac{1}{\ln(\ln(T))}$ where $\ln(T)$ is the Translation length.

To evaluate our results using this metric, we compute a corpus-level BLEU score by comparing each output with its relative 10 ground truths. In Figure 28 we report four different types of BLEU that consider modified 1-gram, 2-gram, 3-gram and 4-gram precisions (BLEU-1, BLEU-2, BLEU-3 and BLEU-4). In each BLEU, the weights associated to each n-gram vary. If we call α , β , γ and δ the weights associated to unigrams, bigrams, trigrams and 4-grams, we can synthesize their variation in Table X

TABLE X: VARIATION OF N-GRAM WEIGHTS IN BLEU METRIC.

	α	β	γ	δ
BLEU-1	1	0	0	0
BLEU-2	0.5	0.5	0	0
BLEU-3	$0.\bar{3}$	$0.\bar{3}$	$0.\bar{3}$	0
BLEU-4	0.25	0.25	0.25	0.25



Figure 28: Modified 1-gram, 2-gram, 3-gram and 4-gram precision BLEU computed using 4 different smoothing methods for our architectures.

We report in Table XI the BLEU score values related to the “add-1” smoothing function only, since the results with other smoothing functions are very similar. It is interesting to see that VGGPT-2 beats our VQA-Baseline when it comes to longer n-gram precision, highlighting the capability of the system to generate better open-ended answers.

TABLE XI: BLEU SCORES USING **ADD-1** SMOOTHING FUNCTION.

	Captioning	BERT	GPT-2	VQA-Baseline	VGGPT-2
BLEU-1	0.010	0.003	0.293	0.594	0.463
BLEU-2	0.000	0.000	0.149	0.187	0.223
BLEU-3	0.010	0.000	0.074	0.066	0.101
BLEU-4	0.000	0.000	0.027	0.011	0.036

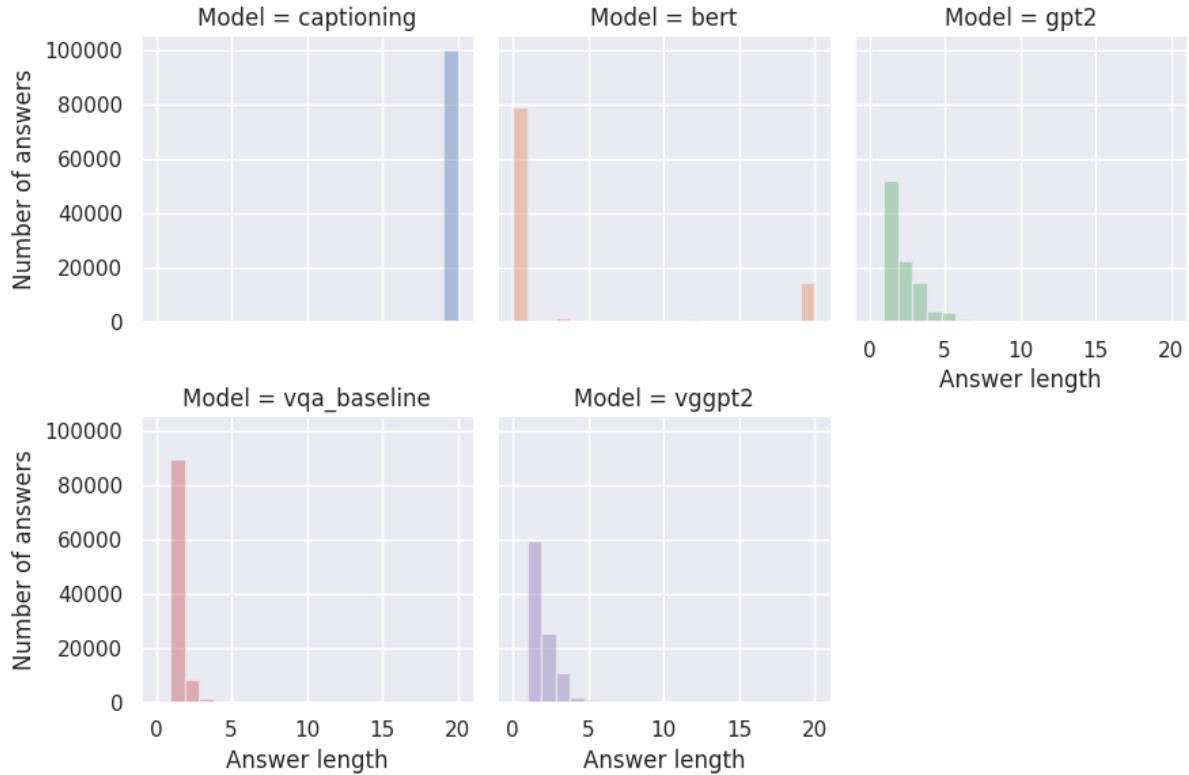


Figure 29: Distribution of answer lengths across our models. On the x-axis of each plot there is the answer length and on y-axis there is the corresponding number of answers. To compute the answer length in the VQA-Baseline we word-tokenize the predicted answer. As highlighted by the graph, VGGPT-2 produces slightly more short answers with respect to GPT-2, indicating that training the system on our short-ended dataset is affecting its capability of generating longer sequences. Yet the answers generated by VGGPT-2 are generally longer than those of our VQA-Baseline.

In Figure 28, we report the BLEU scores of our systems on the validation set and in Figure 29 we show what the answer length distribution in each model is. Interestingly, even though we employ different smoothing functions, the scores are very similar and consistent. From the previous bar-plots, we can make the following considerations:

- It is evident that, as we consider bigger n-gram overlaps (BLEU-2, -3 and -4), VGGPT-2 performs almost every time better with respect to our VQA-Baseline. We were expecting this result since the latter model cannot generate longer answers, and, for this reason, when we compute the BLEU on longer sequences, it doesn't score as good as VGGPT-2. Even though the difference is marginal, it is consistent and remains constant as we increase the n-gram size.
- The two VQA models outperform all the other systems, indicating that the multimodal input is of great benefit to the quality of the answers.
- GPT-2, being a model pre-trained on large textual corpora, achieves stunning performances when we consider its BLEU scores. This is not surprising considering that the model is very deep in nature and its knowledge base is wide: it can answer many questions without looking at any image; however, when questions are image-related, it can only perform random guessing, exhibiting a lower BLEU score with respect to the VQA systems.
- The Captioning and BERT baselines achieve inferior scores, once again because of the bad quality answers they generate. After further investigating, we discovered that both architectures fail to learn what the task is: the Captioning system always generates

sequences of “yes” and “no”, whereas BERT outputs either garbage words or nothing at all. Later we will address more precisely why these models, and especially BERT, which is apparently similar to GPT-2, perform so bad.

Even though BLEU helps to understand some critical aspects of how our systems perform, it has some severe shortcomings: it is not capable of dealing with the semantic contained in the answer. In a real-world scenario, if we ask a human something, he might reply in several different (and correct) ways. BLEU, however, fail to give high scores to correct answers that use different words.

Consider the question “What color is the water?” and assume the ground truth is “Blue”; an answer like “Same color as the sky” would result in a BLEU score of 0, even though it might be a perfect answer (provided there are no clouds in the sky). The latter issue convinced us to experiment with one further metric, namely, Word Mover’s Distance [46].

6.2.3 Word Mover’s Distance

To check the semantic similarity between the answers generated by our models and the ground truths, we employed a powerful metric based on pre-trained word-embeddings, which is the Word Mover’s Distance (WMD) [46].

Before getting into details, consider the following two sentences, taken from [49]:

Obama speaks to the media in Illinois

The President greets the press in Chicago

it is evident that they do not have any word in common, and, for this reason, their BLEU score would be 0, even though they carry the same piece of information.

Word Mover’s Distance is capable of saying how much two sentences are “semantically distant” by projecting the tokens of each sentence in a higher-dimensional space (word-embeddings) and computing which is the minimum cumulative distance that words from sentence A need to travel to match precisely those in sentence B. A distance of 0 indicates perfect semantic match. In contrast, the maximum distance is arbitrary and might be infinite (no relation at all).

In our evaluation process, we employ WMD with 100-dimensional pre-trained GloVe embeddings [8] and discover that VGGPT-2 is indeed generating quality answers, following the trend highlighted by both Accuracy and BLEU metrics.

Since we use pre-trained embeddings and their vocabularies, it might happen that some words in our answers or ground truths do not have a corresponding embedding. This usually results in an infinite distance between the two sentences and forces us to drop the latter when plotting the results (see Figure 30).

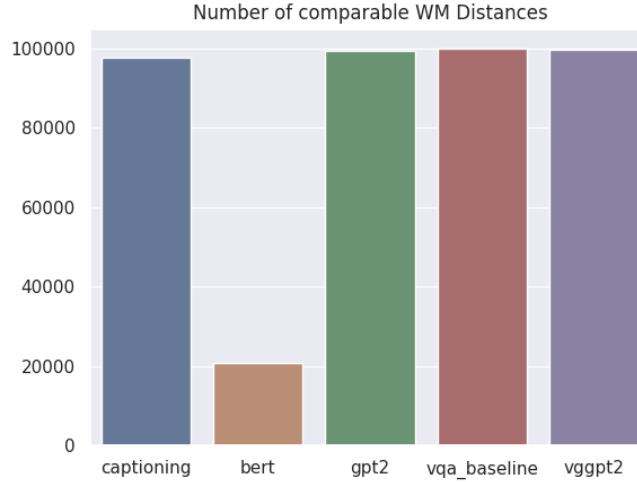


Figure 30: Number of answers in each model for which we can compute the WMD to the ground truths. As we discuss, depending on the output, the WM distance to the ground truths might be infinite. When this is the case, we are unable to say how distant the answer is from the ground truths, and we discard the value. BERT, producing almost every time empty or garbage sequences, has $\approx 20K$ answers whose WMD is not infinite.

In Figure 31 we show the WMD scores for each model. The graph highlights that the VQA architectures generate a significantly larger amount of answers whose WM distance is 0, indicating that the latter models are making good use of both the visual and textual modality to generate the outputs. If we compare the VQA-Baseline with VGGPT-2, we can see a strong

correlation between the two models since they tend to produce answers which are really close (semantically) to the ground truths.

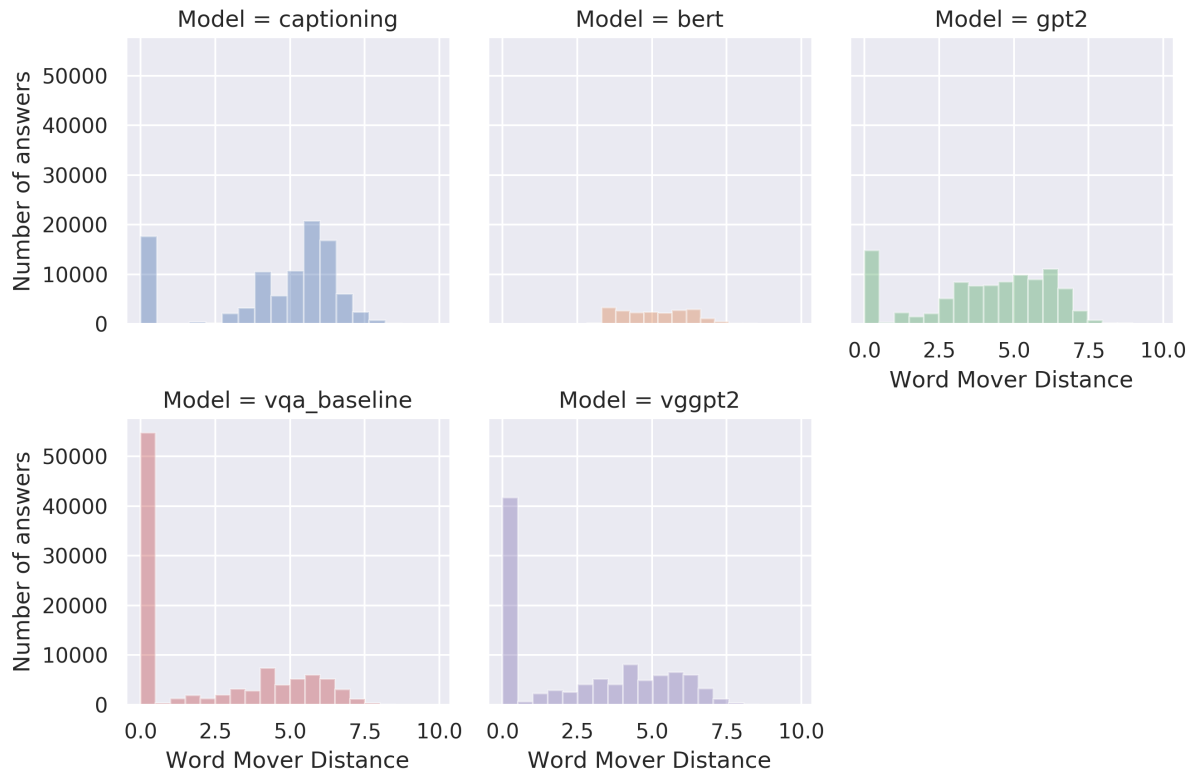


Figure 31: Word Mover’s Distance computed for all of our models on the testing set. It is evident that the VQA-Baseline and VGGPT-2 generate better answers, since the distribution of the distances is closer to 0.

Even though WMD is not a definitive metric of evaluation for VQA architectures, it confirms the general trend and supports the fact that our architecture is indeed able to address the VQA task quite well. Further qualitative evaluation will provide some other exciting results.

6.2.4 Remarks on BERT

The BERT Baseline performs so bad because we are using its architecture as if it was meant for language generation. Although extremely powerful for many NLP downstream tasks, its bi-directionality does not allow the system to generate the next word given the probability distribution of the previous context. In other words, this architecture is not able to generate reasonable answers (or textual sequences) conditioning them on the previous context because the model is always allowed to look at both directions in the input. These architectures are complicated, and we didn't notice this issue until we evaluated all our systems. However, we decided to compare the results even with BERT to show its shortcomings when it comes to VQA and language generation.

6.3 Qualitative evaluation

In this section, we report some real-world results of our proposed architecture, VGGPT-2, side to side with those of the VQA-Baseline.

Aside from the quantitative results, which are very important to say how a system compares with other architectures, we wanted to test our model with arbitrary questions to check its ability to reason on multimodal inputs.

We decided to perform a qualitative evaluation because we believe the metrics mentioned in the previous section were not enough to provide a sound overview of our architecture’s performances.

For this reason, we built a Web-Interface (see Figure 32) connected to a Python Backend that allows anybody to interact directly with the system. The platform initially allows users to upload their own images or select them from a random sample taken from the validation set. Afterward, the users are redirected to an interactive page where they can ask multiple questions on the selected (or uploaded) image. The backend is connected to both VGGPT-2 and the VQA-Baseline.

We experimented with several different types of questions and checked the ability of our VQA systems to classify, count, and detect different objects in the images.

We now report some results considering different question/image pairs and briefly discuss the differences between the two architectures. For VGGPT-2, we even report the softmaps computed within the attention mechanism, which show how the focus on the image changes depending on the considered word in the question and the answer. Softmaps for the three tokens *<bos>*, *<sep>*, and *<eos>* are not reported. Note that the tokens that appear within the softmaps are BPE-Encoded.

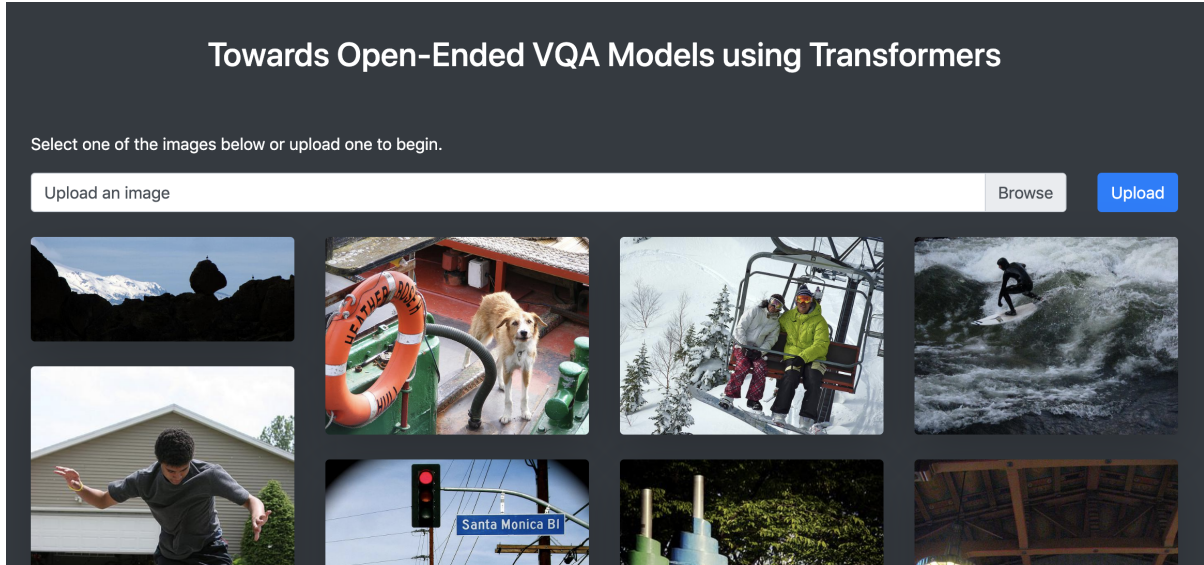
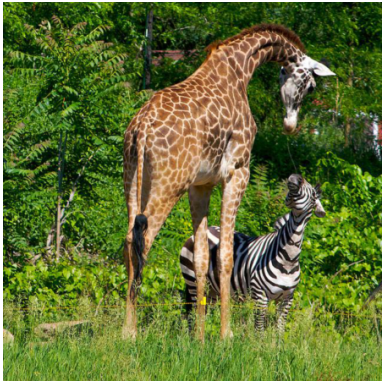


Figure 32: Screenshot of our web interface layout. At the top there is the possibility to upload an image and, below, there are some sample images that can be chosen for an immediate interaction.

6.3.1 Classification performances

First, we report some outputs to **classification-like** questions to check the ability of our model to describe what it is seeing.

TABLE XIII: QUESTION : “WHAT DO YOU SEE?”.



Model	Answer
VQA-Baseline	giraffes
VGGPT-2	zebra and giraffe

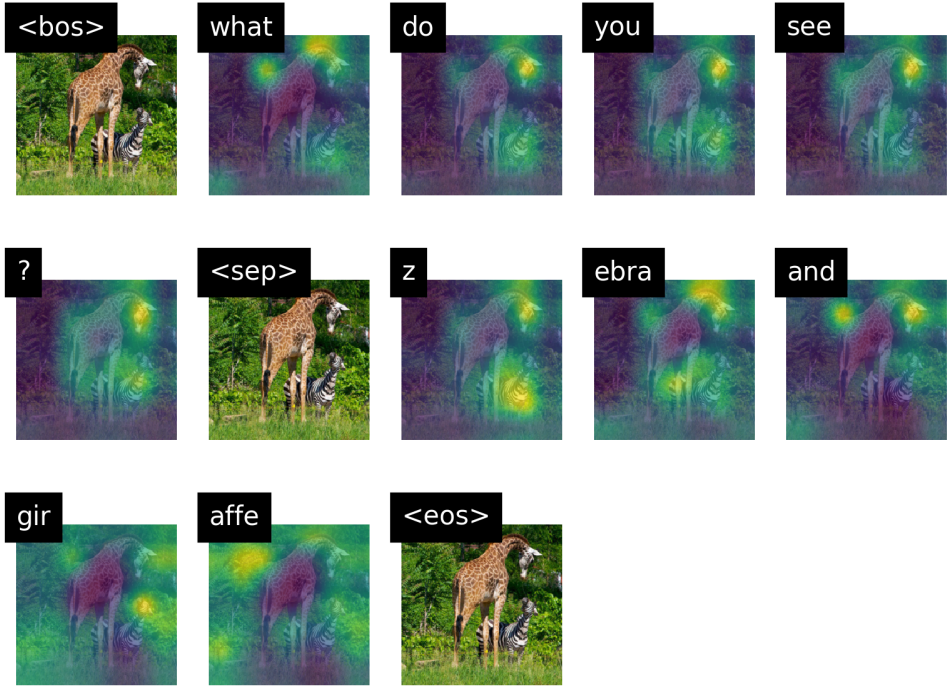


Figure 33: VGGPT-2 softmaps associated with Table XIII.

TABLE XV: QUESTION : “WHAT COLOR IS THE TRAIN?”.



Model	Answer
VQA-Baseline	red
VGGPT-2	red and white

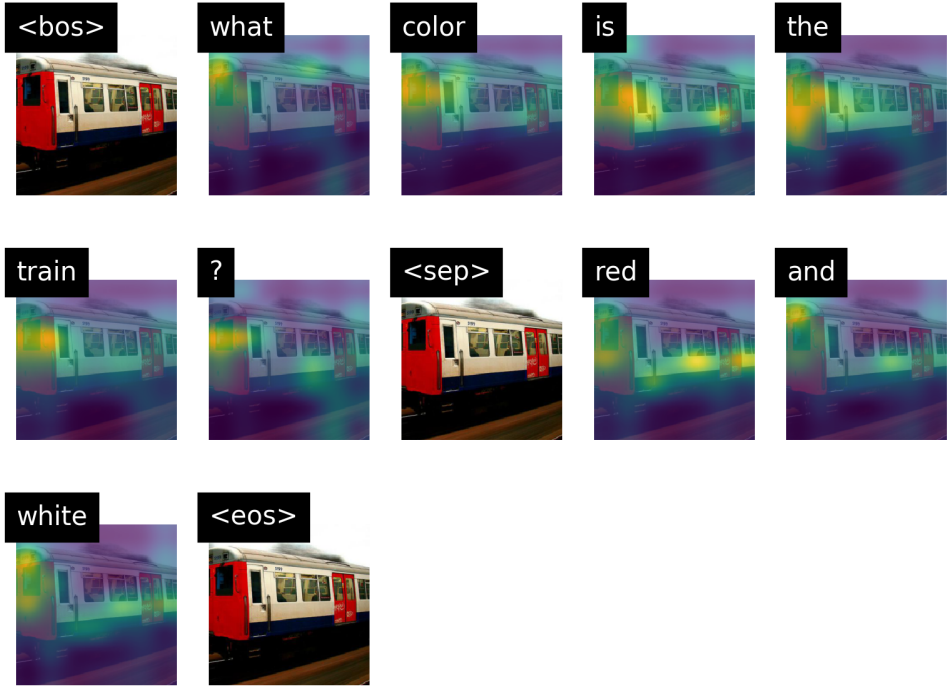


Figure 34: VGGPT-2 softmaps associated with Table XV.

The outputs associated with the example reported in Table XIII show the limitations of the VQA-Baseline: the picture contains multiple animals, but the baseline can only predict one of the two since it has a fixed set of possible answers. On the other hand, VGGPT-2 successfully generates the correct answer using its language model. The same issue limits the baseline output reported in Table XV, where it can only predict one color at a time. Our architecture is indeed able to addend the image and answer correctly.

6.3.2 Reasoning performances

Afterward, we start asking our models more complex answers and discover that VGGPT-2 is indeed performing even better than our baseline, answering most of the times with richer sentences in correct English. From our point of view, VGGPT-2 is a better model with respect to our VQA-Baseline when it comes to real-world VQA.

If we consider the example reported in Table XVII, we can clearly see the limitations of treating the problem as a classification task. A system that distributes probabilities over a fixed set of answers, such as our VQA-Baseline, will never be able to produce an answer like “walking in water” if it is not one of the possible candidates. On the contrary, VGGPT-2 is flexible and scales well with more complex questions.

Once again, in Table XIX, we clearly see how the language model empowers our architecture, allowing it to generate a better description of what is on the plate.

TABLE XVII: QUESTION : “WHAT ARE THE ANIMALS DOING?”.



Model	Answer
VQA-Baseline	drinking
VGGPT-2	walking in water

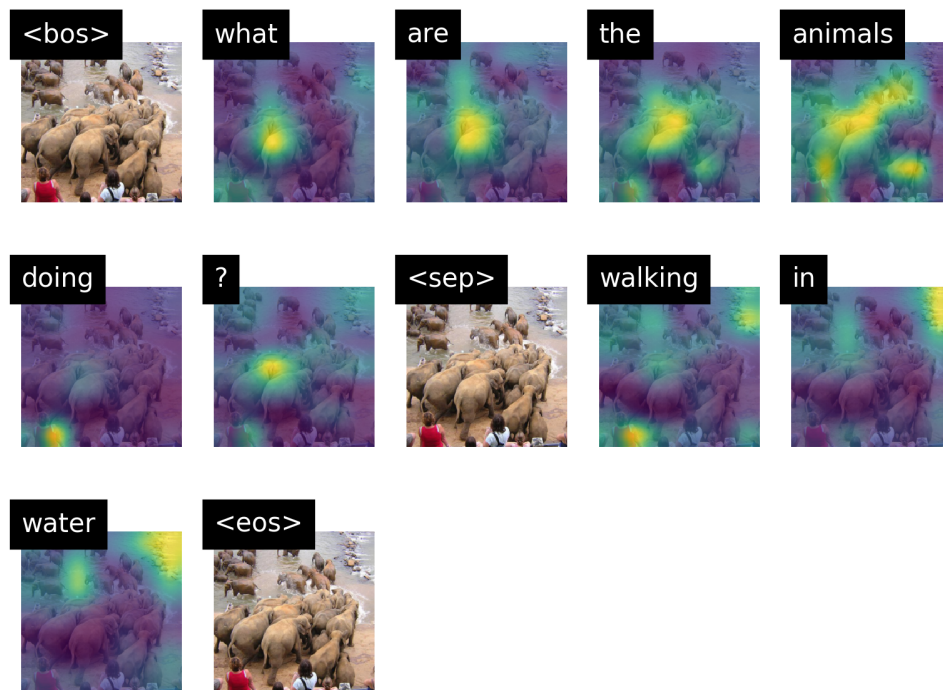


Figure 35: VGGPT-2 softmaps associated with Table XVII.

TABLE XIX: QUESTION : “WHAT IS IN THE PLATE?”.



Model	Answer
VQA-Baseline	salad
VGGPT-2	toast, orange, and beans

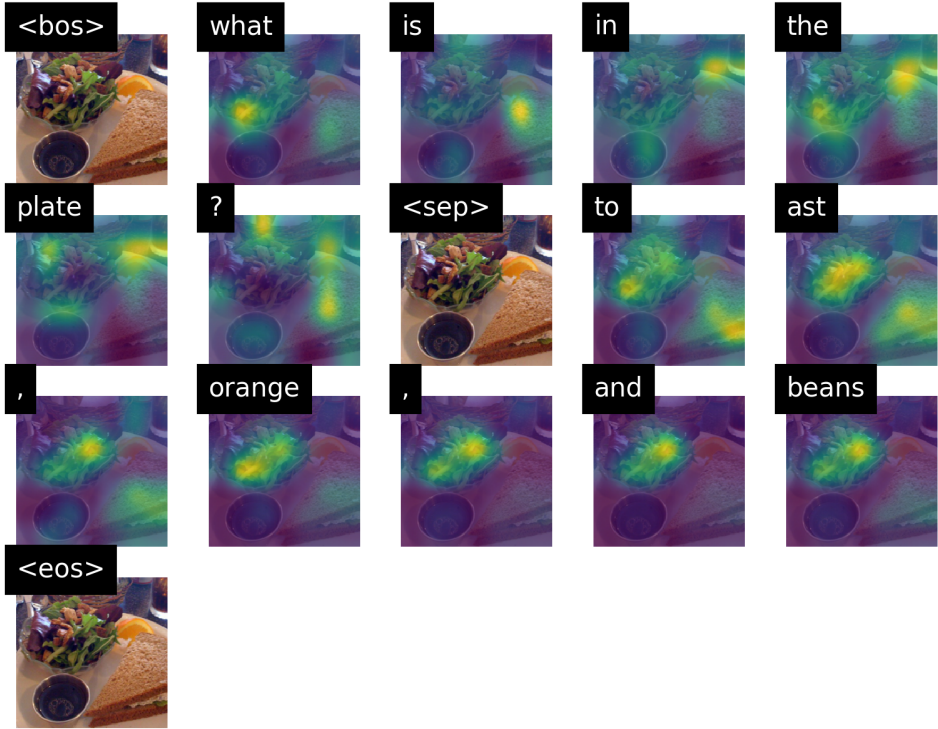


Figure 36: VGGPT-2 softmaps associated with Table XIX.

6.3.3 Generation performances

We discover that our system can generate a question autonomously with an associated answer or a description of what it sees without any input at all. This is fascinating because if we do not feed any question, the VQA-Baseline always returns “yes”, whereas VGGPT-2 generates something which is always correlated to the image. In other words, the VQA-Baseline is not able to generate anything if not conditioned on a question. On the contrary, VGGPT-2 can exploit only the visual features to generate sequences that makeup questions, answers, or captions, all on its own. We report two examples in Table XXI and Table XXIII.

6.3.4 Conclusions

As we highlighted in this section, a qualitative evaluation has revealed that our model performs indeed well, and definitely better with respect to what the quantitative evaluation told us. We are satisfied with these results, which have definitely gone beyond our expectations.

We showed that it is indeed possible to use a Transformer as a new language model to address the VQA task, and we are motivated to keep on refining our architecture to improve performances.

TABLE XXI: EXAMPLE IN WHICH VGGPT-2 GENERATES BOTH A QUESTION AND A CORRECT ANSWER.



Model	Answer
VQA-Baseline	yes
VGGPT-2	How many buses are there? 1

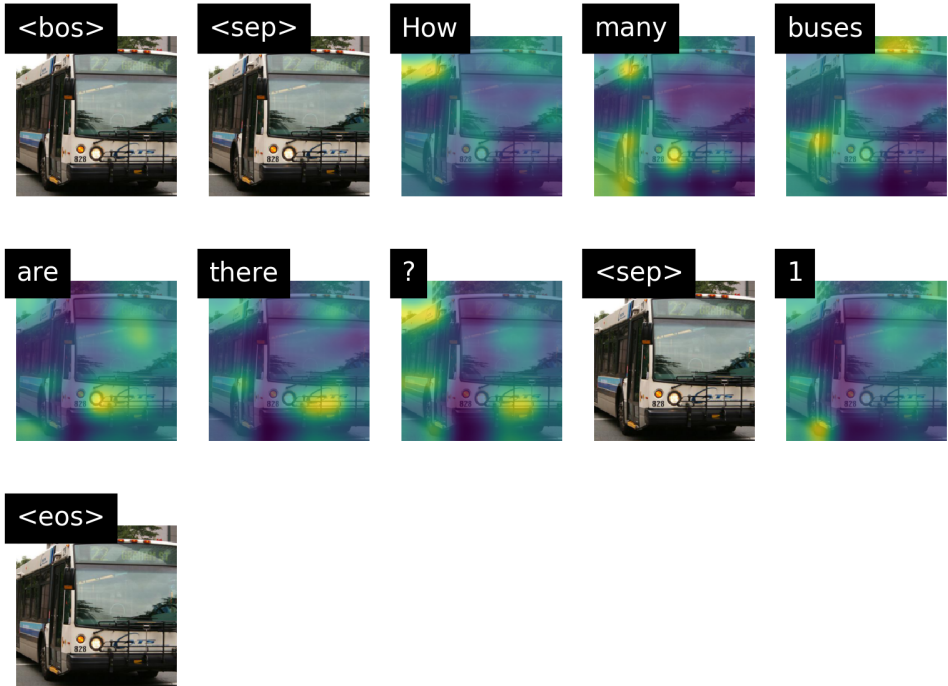


Figure 37: VGGPT-2 softmaps associated with Table XXI.

TABLE XXIII: EXAMPLE IN WHICH VGGPT-2 GENERATES A CAPTION FOR THE IMAGE.



Model	Answer
VQA-Baseline	yes
VGGPT-2	3 zebras

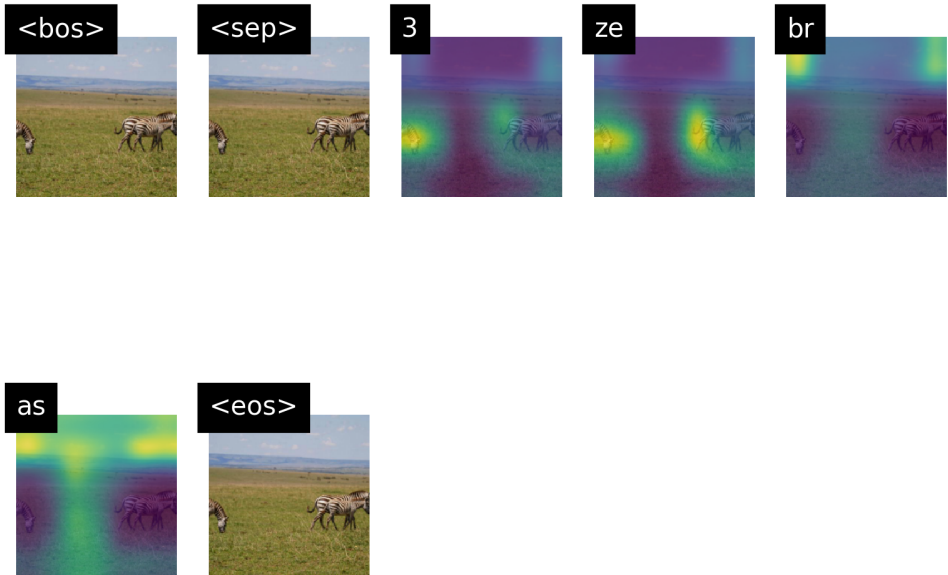


Figure 38: VGGPT-2 softmaps associated with Table XXIII.

CHAPTER 7

CONCLUSIONS

In this work, we discussed a new approach to Visual Question Answering, firmly believing in the need to address this problem in an open-ended fashion. We have introduced a lot of concepts, such as what a Transformer is, and we proposed a new architecture that tries to exploit its power to deal with the problem differently.

At the beginning of this work, we were unsure about whether or not we would have ever been able to build a system fit for the task using a new type of architecture. We had to face countless challenges to reach a point where our implementation was actually able to generate answers, and, after fine-tuning the structure of our system for months, the results have gone, in the end, beyond our expectations.

Our primary objective was not to beat the current state of the art, but rather addressing the problem from a different point of view, providing the community with a new baseline that intends to move the attention towards an open-ended VQA approaches.

We are thrilled to have had the opportunity to work in this field and hope that this work will eventually be taken into consideration by the scientific community for further improvements.

7.1 Future work

Even though this is the end of this document, we do not intend to stop our research for better VQA models. Future work will try to build a different and more efficient architecture

while maintaining the power of the system, and possibly improving it. Briefly, this is our roadmap towards improving our model:

- **Lighter model:** We would like to get rid of the attention mechanism adopted from [40] and gradually build a lighter and more powerful layer avoiding, when possible, unnecessary operations. We believe that we can get rid of our massive final linear transformation and intend to change how we use the attention vector.
- **Multiple attention heads:** As discussed in [1], multi-headed attention systems usually perform better. For this very reason, instead of sticking with a single attention vector for each token in the input sequence, we would like to concatenate the outputs of multiple attention heads together and see if we manage to boost performances.
- **More powerful language model:** Even though our architecture is already profound (202M parameters), we wonder how the system would behave using more powerful versions of GPT-2, such as the ones with, respectively, 345M, 774M or 1558M parameters. OpenAI shows that the performances of these systems scale well with the number of parameters; for this reason, we are confident that combining more powerful language models in our system would result in better answers. However, at the moment, we are limited by the computing power at our disposal.
- **Different image encoders:** As reported in the implementation, for what concerns the visual modality, we stuck with VGGNet-11 [16]. We want to experiment with different image encoders, such as ResNet [18], which exhibits great performances in our VQA-Baseline.

CITED LITERATURE

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I.: Attention is all you need. CoRR, abs/1706.03762, 2017.
2. The internet’s source of freely useable images. <https://unsplash.com>. [Online; Accessed 2019-10-30].
3. Huang, X., Acero, A., and Hon, H.-W.: Spoken Language Processing: A Guide to Theory, Algorithm, and System Development. Upper Saddle River, NJ, USA, Prentice Hall PTR, 1st edition, 2001.
4. Google translate. <https://translate.google.com/>. [Online; Accessed 2019-10-15].
5. Shibata, Y., Kida, T., Fukamachi, S., Takeda, M., Shinohara, A., and Shinohara, T.: Byte pair encoding: A text compression scheme that accelerates pattern matching. 09 1999.
6. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I.: Language models are unsupervised multitask learners. OpenAI Blog, 1(8), 2019.
7. Mikolov, T., Chen, K., Corrado, G., and Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013.
8. Pennington, J., Socher, R., and Manning, C.: Glove: Global vectors for word representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.
9. Hochreiter, S. and Schmidhuber, J.: Long short-term memory. Neural computation, 9:1735–80, 12 1997.
10. He, K., Zhang, X., Ren, S., and Sun, J.: Deep residual learning for image recognition. CoRR, abs/1512.03385, 2015.
11. Ba, J. L., Kiros, J. R., and Hinton, G. E.: Layer normalization. arXiv preprint arXiv:1607.06450, 2016.

CITED LITERATURE (continued)

12. The illustrated transformer. <http://jalammar.github.io/illustrated-transformer/>. [Online; Accessed 2019-10-28].
13. Trump dances with danger in middle east. <https://edition.cnn.com/2019/05/26/middleeast/trump-middle-east-iran-analysis-trump-intl/index.html>. [Online; Accessed 2019-10-28].
14. Devlin, J., Chang, M., Lee, K., and Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. CoRR, abs/1810.04805, 2018.
15. Schuster, M. and Nakajima, K.: Japanese and korean voice search. In 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 5149–5152. IEEE, 2012.
16. Simonyan, K. and Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.
17. Large scale visual recognition challenge (ilsvrc). <http://www.image-net.org/challenges/LSVRC/>. [Online; Accessed 2019-10-29].
18. He, K., Zhang, X., Ren, S., and Sun, J.: Deep residual learning for image recognition. CoRR, abs/1512.03385, 2015.
19. Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Lawrence Zitnick, C., and Parikh, D.: Vqa: Visual question answering. In The IEEE International Conference on Computer Vision (ICCV), December 2015.
20. Fukui, A., Park, D. H., Yang, D., Rohrbach, A., Darrell, T., and Rohrbach, M.: Multimodal compact bilinear pooling for visual question answering and visual grounding. arXiv preprint arXiv:1606.01847, 2016.
21. Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Zitnick, C. L., and Parikh, D.: VQA: visual question answering. CoRR, abs/1505.00468, 2015.
22. Altman, N. S.: An introduction to kernel and nearest-neighbor nonparametric regression. The American Statistician, 46(3):175–185, 1992.
23. Andreas, J., Rohrbach, M., Darrell, T., and Klein, D.: Deep compositional question answering with neural module networks. ArXiv, abs/1511.02799, 2015.

CITED LITERATURE (continued)

24. Hu, R., Andreas, J., Rohrbach, M., Darrell, T., and Saenko, K.: Learning to reason: End-to-end module networks for visual question answering. CoRR, abs/1704.05526, 2017.
25. Klein, D. and Manning, C. D.: Accurate unlexicalized parsing. In Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1, pages 423–430. Association for Computational Linguistics, 2003.
26. De Marneffe, M.-C. and Manning, C. D.: The stanford typed dependencies representation. In Coling 2008: proceedings of the workshop on cross-framework and cross-domain parser evaluation, pages 1–8. Association for Computational Linguistics, 2008.
27. Suhr, A., Lewis, M., Yeh, J., and Artzi, Y.: A corpus of natural language for visual reasoning. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 217–223, 2017.
28. Suhr, A., Zhou, S., Zhang, I., Bai, H., and Artzi, Y.: A corpus for reasoning about natural language grounded in photographs. CoRR, abs/1811.00491, 2018.
29. Lu, J., Yang, J., Batra, D., and Parikh, D.: Hierarchical question-image co-attention for visual question answering. In Advances In Neural Information Processing Systems, pages 289–297, 2016.
30. He, K., Zhang, X., Ren, S., and Sun, J.: Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.
31. Gao, Y., Beijbom, O., Zhang, N., and Darrell, T.: Compact bilinear pooling. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 317–326, 2016.
32. Charikar, M., Chen, K., and Farach-Colton, M.: Finding frequent items in data streams. In International Colloquium on Automata, Languages, and Programming, pages 693–703. Springer, 2002.
33. Amazon mechanical turk. <https://www.mturk.com/>. [Online; Accessed 2019-11-3].
34. Zellers, R., Bisk, Y., Farhadi, A., and Choi, Y.: From recognition to cognition: Visual commonsense reasoning. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2019.

CITED LITERATURE (continued)

35. Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., and Girshick, R. B.: CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. CoRR, abs/1612.06890, 2016.
36. Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L.: Microsoft COCO: common objects in context. CoRR, abs/1405.0312, 2014.
37. Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P.: Squad: 100, 000+ questions for machine comprehension of text. CoRR, abs/1606.05250, 2016.
38. Loper, E. and Bird, S.: Nltk: The natural language toolkit. In Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1, ETMTNLP '02, pages 63–70, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.
39. Lamb, A. M., Goyal, A. G. A. P., Zhang, Y., Zhang, S., Courville, A. C., and Bengio, Y.: Professor forcing: A new algorithm for training recurrent networks. In Advances In Neural Information Processing Systems, pages 4601–4609, 2016.
40. Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A. C., Salakhutdinov, R., Zemel, R. S., and Bengio, Y.: Show, attend and tell: Neural image caption generation with visual attention. CoRR, abs/1502.03044, 2015.
41. Show, attend, and tell — a pytorch tutorial to image captioning. <https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning>. [Online; Accessed 2019-11-5].
42. Kingma, D. and Ba, J.: Adam: A method for stochastic optimization. arxiv 2014. arXiv preprint arXiv:1412.6980, 2014.
43. Kazemi, V. and Elqursh, A.: Show, ask, attend, and answer: A strong baseline for visual question answering. CoRR, abs/1704.03162, 2017.
44. Strong baseline for visual question answering. <https://github.com/Cyanogenoid/pytorch-vqa>. [Online; Accessed 2019-11-5].
45. Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J.: Bleu: a method for automatic evaluation of machine translation. In Proceedings of the 40th annual meeting on

CITED LITERATURE (continued)

- association for computational linguistics, pages 311–318. Association for Computational Linguistics, 2002.
46. Kusner, M., Sun, Y., Kolkin, N., and Weinberger, K.: From word embeddings to document distances. In International conference on machine learning, pages 957–966, 2015.
 47. Lin, C.-Y. and Och, F. J.: Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics, page 605. Association for Computational Linguistics, 2004.
 48. Chen, B. and Cherry, C.: A systematic comparison of smoothing techniques for sentence-level bleu. In Proceedings of the Ninth Workshop on Statistical Machine Translation, pages 362–367, 2014.
 49. Word mover’s distance for text similarity. <https://towardsdatascience.com/word-movers-distance-for-text-similarity-7492aeca71b0>. [Online; Accessed 2019-10-18].
 50. Malinowski, M. and Fritz, M.: A multi-world approach to question answering about real-world scenes based on uncertain input. In Advances in Neural Information Processing Systems 27, eds. Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, pages 1682–1690. Curran Associates, Inc., 2014.
 51. Coco - common objects in context. <http://cocodataset.org/>. [Online; Accessed 2019-5-18].

APPENDICES

Appendix A


COPYRIGHT PERMISSION 1

All images reported in section 6.3 were built starting from those contained in the VQAv2 dataset [21], with major modifications: every image is randomly cropped and put side to side with softmaps that are generated by our models. Furthermore, the images contained in VQAv2 were obtained by the authors at the time of generating the dataset with a Creative Commons license from the Flickr website. You can find the license of each image in the annotation files at [51].

Appendix B

COPYRIGHT PERMISSION 2

The permission to use the image of Figure 16 of section 4.2, taken from [29], is reported here.

Alberto Mario Bellini
30 ottobre 2019 16:22


Permission to use images contained in one of your papers

A: jiasenlu@gatech.edu, jw2yang@gatech.edu, dbatra@gatech.edu, parikh@gatech.edu

Hello,

My name is Alberto Mario Bellini and I am a M.Sc. Computer Science student at the University of Illinois at Chicago (UIN 660190113).

I am contacting you to kindly request a written permission to integrate some images contained in your paper "Hierarchical Question-Image Co-Attention for Visual Question Answering" into my thesis.

To be more specific, I would like to insert the attention maps generated at word, sentence and question level into the related work section of my thesis, citing you as the source.

My university (UIC) requires a written agreement from you for me to use such images and I cannot proceed until you give me an official permission to use your media.

I am working on VQA and would like to compare my approach, which uses OpenAI's GPT-2 and Google's BERT transformers to generate open domain answer, with yours.


My model generates soft maps too and I would love to show how they differ from the ones shown in your work.

I would really appreciate your help.

Thank you very much for you time.

Best regards,

Alberto Mario Bellini.
abelli6@uic.edu

Parikh, Devi
30 ottobre 2019 21:39


Re: Permission to use images contained in one of your papers

A: Alberto Mario Bellini, Cc: Lu, Jiasen, Yang, Jianwei, Batra, Dhruv

Yes, please feel free to use the images with appropriate citation.

[Sent via a mobile device]

VITA

NAME	Alberto Mario Bellini
EDUCATION	
Master of Science in Computer Science	
University of Illinois at Chicago, USA, Fall 2019	
Master Degree in Computer Science and Engineering	
Politecnico di Milano, Italy, expected Spring 2020	
Bachelor Degree in Computer Science and Engineering	
Politecnico di Milano, Italy, Spring 2017	
LANGUAGE SKILLS	
Italian	Native speaker
English	Full working proficiency
	A.Y. Fall 2018 - One Semester of study abroad in Chicago, Illinois
	A.Y. Summer 2012 - Full time international student at Rangitoto College, New Zealand
	2017 - IELTS examination
TECHNICAL SKILLS	
Programming Languages	C, Java, Python, Swift, Objective-C, Javascript
	PHP, HTML, Swift, Objective-C, Javascript, SQL
Frameworks	PyTorch, Bootstrap, NodeJS.
Operating Systems	Unix-like (macOS, Debian, Kali etc..) and Microsoft Windows
Softwares	JetBrain's suite, Adobe's suite, Cad softwares (3DS Max, Cinema 4d), Game engines (Unity3D and UnrealEngine)

VITA (continued)

PUBLICATIONS

- Fall 2018 Federico Amedeo Izzo, Lorenzo Aspesi, Alberto Bellini, Chiara Pacchiarotti, Federico Caimi, Gianluigi Persano, Niccolò Izzo, Pietro Tordini, Luca Mottola, Massimo Bianchini, and Stefano Maffei. 2018. 64Key: A Mesh-based Collaborative Platform. In Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems (SenSys '18), Gowri Sankar Ramachandran and Bhaskar Krishnamachari (Eds.). ACM, New York, NY, USA, 422-423. DOI: <https://doi.org/10.1145/3274783.3275214>
-