UMONS
Université de Mons

Faculté Polytechnique

Improving Image Captioning with dense annotations

Master thesis
In fulfillment of the requirements for the degree of
Master in Electrical Engineering
Specialist Focus in Signals, Systems and BioEngineering

Robin JACOBS

POLYTECH MONS

Under the supervision of :
Prof. Thierry DUTOIT (proponent)
Ir. Jean-Benoit DELBROUCK (co-proponent)
Ir. Stephane DUPONT (co-proponent)
August 2019

pôle hainuyer

Acknowledgments

I would like to express my thanks of gratitude to my proponents, especially Jean-Benoit Delbrouck for their guidance and help in this project completion.

I also would like to express my deepest appreciation to Sorin Mogosan for his help for the realisation of this report.

In addition, I would like to thank my family and my friends for their support during all my studies which this thesis represents the final step.

Contents

List of Figures

Introduction

Everybody has already seen in a movie or read in a book a story with an artificial intelligence able to communicate smoothly with the user. Who never dreamed about having an intelligent home able to help our domestic chores or executing computer tasks just by asking to our computer to do so? In these days, we often heard about "artificial intelligence" and great progress are done every day in this research area. Thus, we can see interactive robot from Japan talking with customers in a store, or any other impressive result.

But of course, to obtain such amazing achievements, a lot of work and different technology are required.
One of the most basic and essential function needed by these robots is the ability to perceive and understand their environment. Indeed, to efficiently interact with its surrounding, it is necessary to be able to analyze it on the most accurate way. This basic skill is called image captioning. It consists in a model to analyze an image and to give a complete description of the scene which is seen.

To this end, there are a lot of research which propose new methods to always achieve better results, by improving the models of deep learning, using new technology, or improving the databases used to train these models and teach them to recognize and comprehend their environment. This last option will be used in this thesis to improve the existing image captioning systems.

The idea is to provide a new database composed of a set of pictures with their description, and the position on these images of the objects mentioned in these captions. This way, the model will be able to learn how to localize the important areas in a picture and to focus on them.

We will begin this report by explaining the theoretical notions relevant for the good understanding of this work and the image captioning model on which we based this thesis. Then, we will explain in detail the manipulations which will be executed to generate the new database. We will at first describe the databases used as a basis for this work, then, we will describe

how we will manipulate them. Finally, we will modify an existing model of image captioning to adapt it to our purpose. This way, it will be able to learn how to localize information on a picture. Finally, we will show some results generated with this new model.

Chapter 1

Related work and theoretical notions

In this chapter, we will explain some theoretical notions which we find necessary to understand the work presented in this document.

1.1 Image captioning

There is now a lot of studies about automatically generating a description of an image [10]. This process is called **Image captioning**, also known as Automatic image annotation, automatic image tagging or linguistic indexing [11]. It is a very important and necessary task to generate a rich and natural description of an image, in order for a robot to communicate effectively with a human user[10]. To execute this process, a robot requires to identify the different objects represented in the picture, but also their attributes and relationships. After understanding the scenery, it will need to generate a correct sentence explaining it. A typical image captioning program will use a convolutional neural network (CNN) for the image encoder and a recurrent neural network (RNN) based sentence decoder [10] (these notion are discussed further in this chapter).

1.2 Reinforcement learning

A method to do machine learning, called **reinforcement learning**, is to consider an agent who is taking actions in an environment [12]. This agent (for example a game character) will evolve in a discrete time system. At each time, it will have a set of all possible moves it can execute, its actions. Each action will provide a reward, a feedback by which the success of the agent can be measured. The goal of this method is to maximize his reward during the trajectory which is the sequence of actions and states of the agent [13].

1.3 Encoder-Decoder architecture

An encoder is a network that will take the input for example an image, and give a feature map/vector/tensor as an output. The information of the inputs will be contained in these features which represent the input. For example, a Convolutional Neural Network (CNN) as described later can be used as an encoder.
The encoder are trained with the decoder and the difference between the actual and reconstructed input is used in the loss-function.
Of course, in actual application, we will not try to retrieve the actual input. For example, for image captioning, the input will be the image but the output will be a sequence decoded word by word by the decoder that describe it. [14]

1.4 Convolutional Neural Network (CNN)

The **Convolutional Neural Network** (CNN) is a deep learning algorithm which will allow to segment an input image and to assign an importance to each one of these parts (using weights and bias). The goal is to differentiate the different objects on the picture. The advantage of a CNN (or ConvNet) is that the pre-processing required is very low. Furthermore, CNN can learn filters abilities [1].
The configuration of a CNN is inspired by biological process where individual neurons respond only on stimuli on restricted area. These regions called receptive fields overlap in order to cover the entire visual field [15].
When the input is for example a 128x128 RGB pixels images, it will be computed by a 128x128x3 matrix in a program (128x128 for the spatial information and 3 for the color). We can easily see that if we work with high definition pictures, the matrix will quickly become enormous and the computing will be very intensive. The CNN will allow to reduce the format of the image in a form which will smooth the process while keeping a maximum of the information. Several layers will be used for this purpose and finally, the output will be a vector which contains the class score to identify the object analyzed.

1.5 Recurrent Neural Network (RNN)

The **Recurrent Neural Network** (RNN) is a class of neural network which can study temporal dynamic behavior [16]. Indeed, RNN are networks using loops to allow the information to persist. In the following diagram, the input x_t will give an output h_t and a loop will pass information from one step to the next one [2].

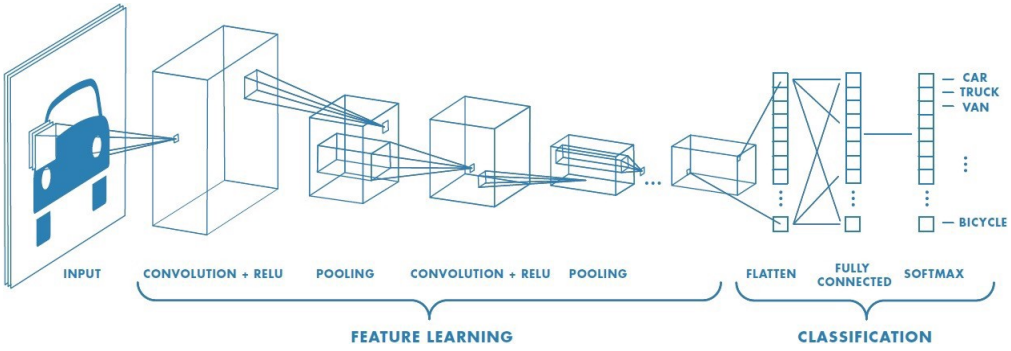

Figure 1.1: Illustration of a CNN[1]

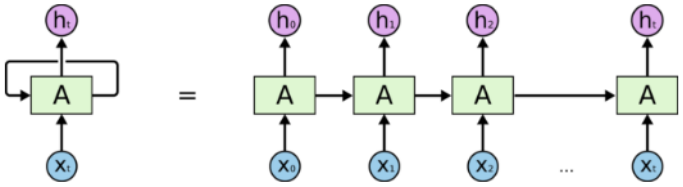

Figure 1.2: An unrolled recurrent neural network[2]

With this kind of configuration, we can easily understand that RNN are commonly used in sequences and lists applications such as speech recognition, language modeling,translation and, like in our case, image captioning.

1.6 Long Short-Term Memory (LSTM)

As we have seen, the RNN are networks able to use past information. unfortunately, sometimes, the information can be far away in the past compared to when it is needed. If in theory the RNNs are able to handle such "long term dependencies", in practice they are not suitable for such tasks [17][18]. In these cases, we use a special kind of RNN called **Long Short-Term Memory** networks.

1.7 Markov Decision Process (MDP)

A **Markov decision process** (MDP) is a discrete time stochastic control process [3]. In this model, an agent take action with random results. The process is a fourtuple (S, A, P_a, R_a, γ) with [19]:

- S, a set of possible state
- A, a set of possible actions
- R_a a real value reward function
- $P_a(s,\ s')$, the probability that the action a in the state s will lead to the state s'
- γ, the discount factor with $\gamma \in [0, 1]$. it will decrease the importance of the reward in the future

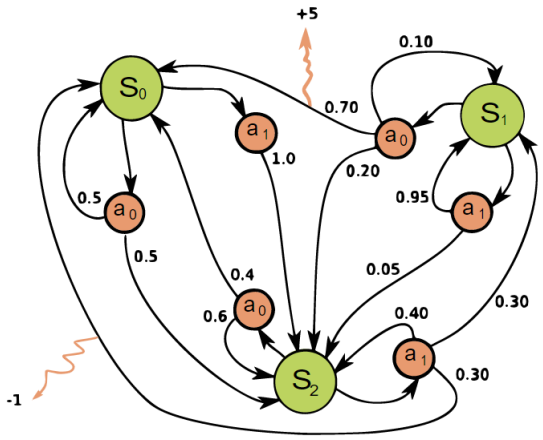

Figure 1.3: Example of a simple MDP with three states (green circles) and two actions (orange circles), with two rewards (orange arrows)[3]

With these elements, the purpose of a Markov Decision Process is to maximize the return G_t [20].

$$G_T = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

As we can see, the discount factor γ will indicate how much the system will take into account the future information. If $\gamma = 0$, the system will take in consideration only the first reward, it will be "short-sighted". On the contrary, if $\gamma = 1$, it will take all the future rewards.

1.8 Beam search

The beam search is a heuristic search algorithm which will explore a tree by not considering only a limited number of nodes. It is an optimization of

best-first search that will look at each node and then reduce the memory requirements.[21]
Beam search is used in the decoder to not just be lazy and take the words with the best score, but to find the most optimal sequence. This is why this method is often used for language modeling [6].

1.9 Visual attention

The visual attention of a model represents the localization of where on a picture the program will look at to retrieve the information it needs.

When we use such tool, the input of the model is not the image I anymore but weighted image features instead. Therefore, a low weight imply an area with poor information and consequently, these area will be "darkened".

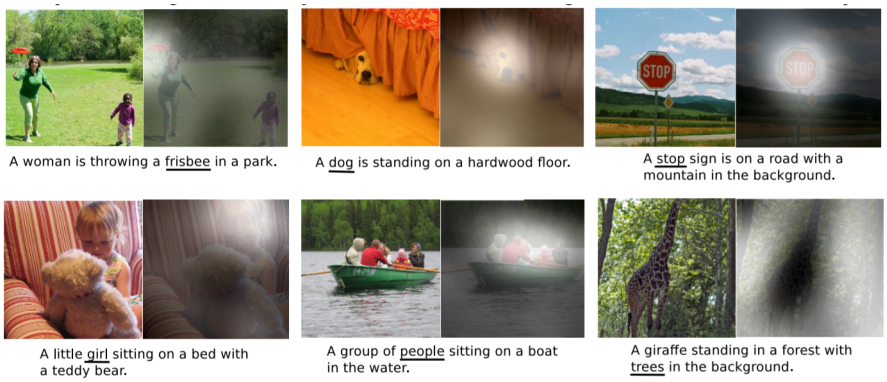

Figure 1.4: Exemple of attending to the correct object [4]

The figure 1.4 shows the attention of the model directed toward a specific object from its caption. The white indicates the the attended regions and the rest of the picture is darkened because the information we are looking for doesn't come from these area.

As we can guess, Visual attention can be very helpful for image captioning as illustrated in the figure 1.5 since the model will focus on the different objects represented on the picture [4]. Identify the importance of these object and improving the accuracy of this method is the purpose of this thesis.

The weighted features for a LSTM can be computed with the following method [5]:

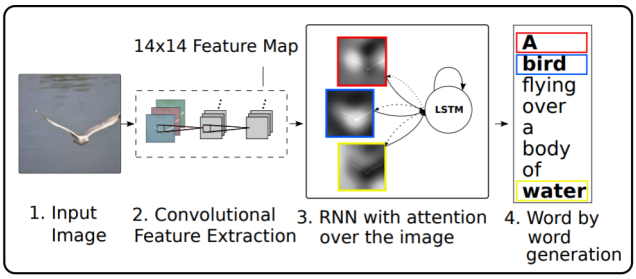

Figure 1.5: Image captioning with model of attention[4]

Be $x_1, x_2, x_3, ..., x_n$, different sub-section of the image I. We can compute the score s_i to measure the attention of x_i using:

$$s_i = \tanh\left(W_c C + W_x X_i\right) = \tanh\left(W_c h_{t-1} + W_x x_i\right)$$

With h_i, the hidden state of the LSTM and $C = h_{t-1}$, the context.
Then, s_i are passed to a softmax function to compute the weights α_i.

$$\alpha_i = \text{softmax}(s_1, s_2, ..., s_i, ...)$$

Finally, we can obtain the final input of the LSTM by taking the weighted average

$$Z = \sum_i \alpha_i x_i$$

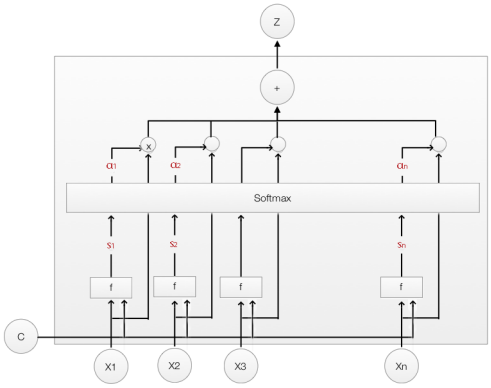

Figure 1.6: computation of the weighted features [5]

You can find this tool applied to image captioning in the following picture 1.7. You can see in a brighter colour the area of the image the model is paying attention at. For each word, we can see the localization of the attention will change.

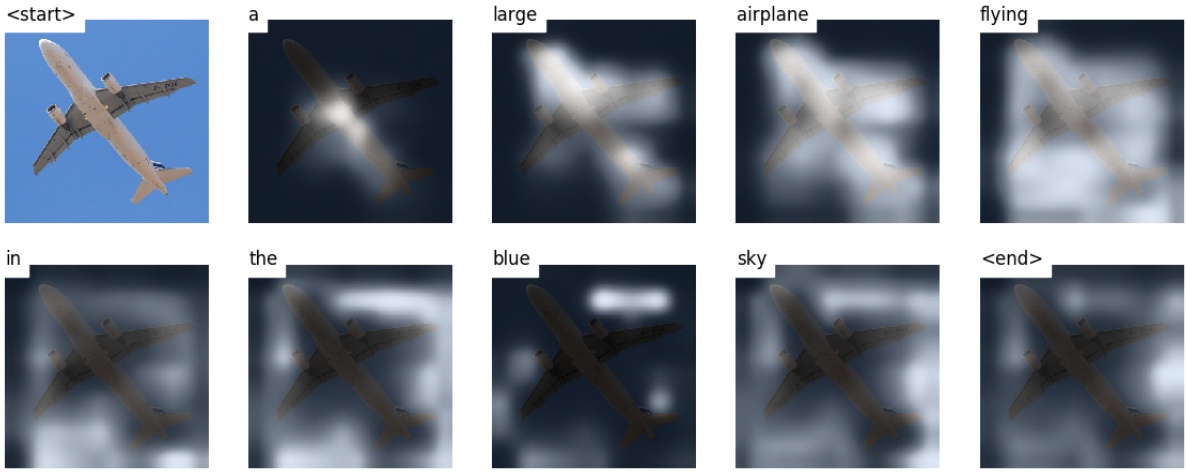

Figure 1.7: visual attention for image captioning [6]

1.10 Model Overview

The model developed for this project is mainly based on the following work: https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning

In this section, we will briefly present the main parts of this model such as the encoder, decoder, etc...

1.10.1 Encoder

The inputs of the model are images and are encoded using Convolutional Neural Networks (CNN). The goal is that the encoded picture will contains all the information summarized in a smaller form.

Since there already exist pre-trained CNNs, it is not needed to train a new one from scratch.
The model chosen for this program is the **101 layered Residual Network trained on the ImageNet classification task** available in pyTorch.

With this encoder, we will obtain from a picture with 3 colors, a height H and a width W, a new encoded image with 2048 learned channels and a size of 14x14. In other word, we will transform a picture of dimension (3, H, W) into a new one of dimension (2048, 14, 14).

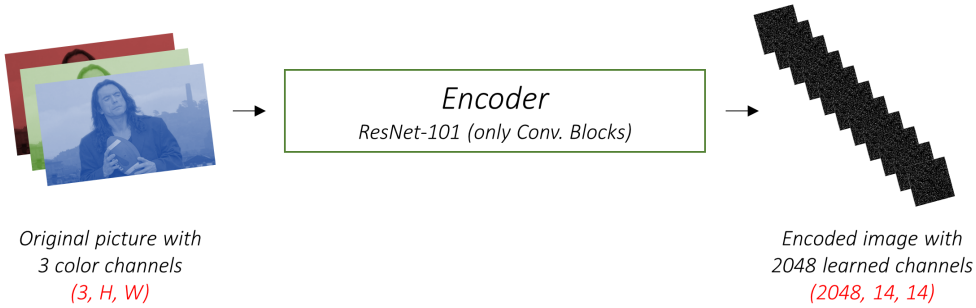

Figure 1.8: Encoder representation [6]

1.10.2 Decoder

The decoder will generate a description of the picture using the encoded picture from the encoder. From this input, it will give word by word the desired caption.

Since a sentence is generated, a Recurrent Neural Network will incarnate this decoder and will use a LSTM.

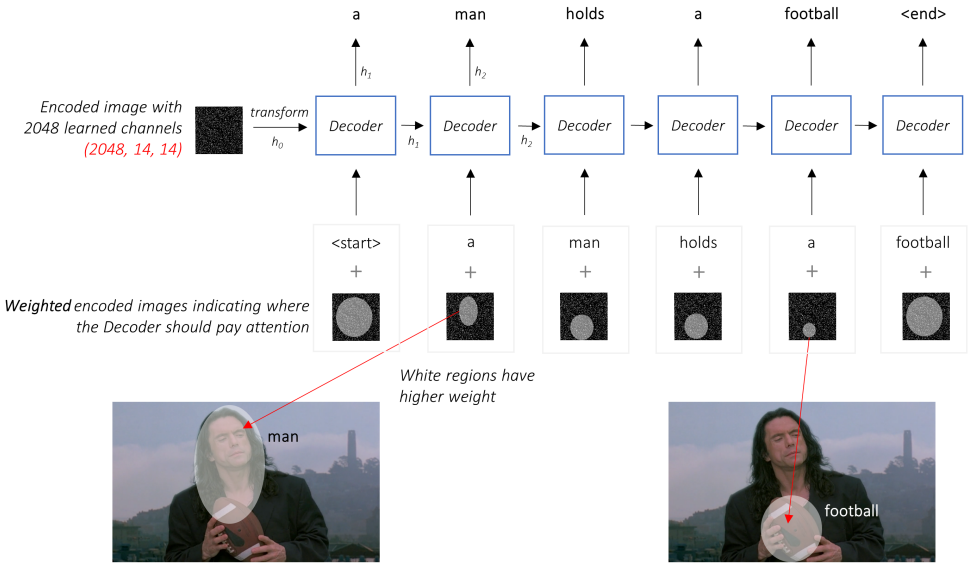

Figure 1.9: Decoder with attention [6]

We work on a model using attention, this is why the decoder must be

able to change its attention on the picture for each word of the caption. For example, on the figure 1.9, we can see that the model will focus on the man, then on the football.

Since we use visual attention, the pixels from the image will be weighted depending of their importance for each step of the sequence. This weighted representation of the image can be concatenated with the previously generated word at each step to generate the next word.

To briefly describe how the attention is generated, the model looks at the sentence generated so far and try to guess what needs to be described next. For example, when *"a man holds"* is generated, the model will looks at what the man is holding: the ball.

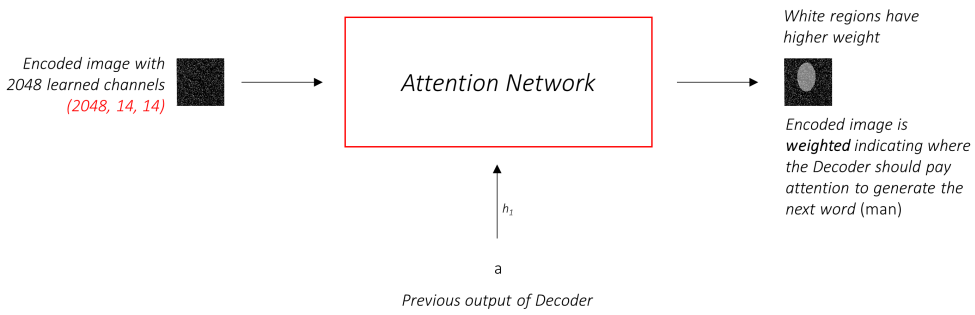

Figure 1.10: Attention network representation [6]

We use **soft attention**. Thus, the sum of the weights of the pixels is equal to 1. If there are P pixels in the encoded image, ate each step t, we have:

$$Z = \sum_{p}^{P} \alpha_{p,t} = 1$$

You can say that α is the probability that the pixel is relevant to generate the next word of the caption.

1.10.3 Model representation

We are now able to represent the entire model on the figure 1.11.

In short:

- An Image of dimension (3, H, W) is taken as an input.
- This image will be transformed by the encoder in an encoded image of dimension (2048, 14, 14).
- The initial hidden stage h_0 is created from this encoded image for the LSTM Decoder.
- At each step of the caption generation in the decoder, the encoded image and the previous hidden state is used to generate the new attention weights α. Furthermore a word is generated using the previous word and the weighted image.

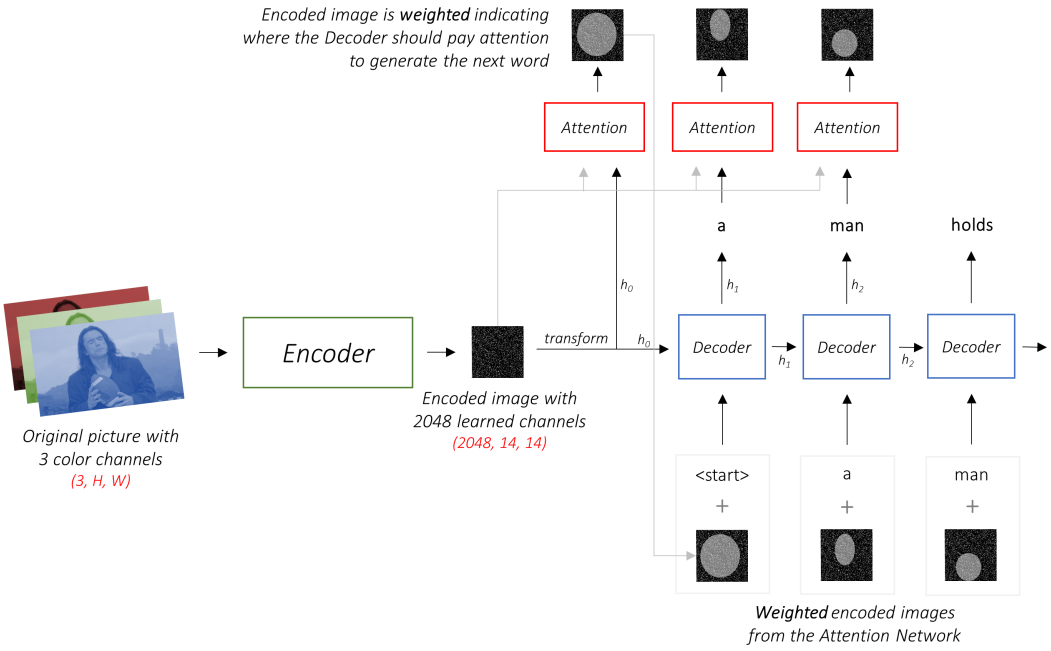

Figure 1.11: model [6]

1.10.4 Beam Search

The Decoder give as an output a score for each word available in the vocabulary of the model.
The easiest way would be to take the word with the best score at each step but this is not the best way. Indeed, if the first word is a wrong guess, each word which follows will be a sub-optimal solution. This kind of mistake can take place at any moment during the captioning. It often happens that

the solution will be better if you take the second or third highest score for the first, and/or second word of the caption. This is why the purpose of the beam search is to chose the sequence with the best overall score from several possible sequences and not immediately chose each word.

The beam search will process as follow:

- For the first word, we will considerate the best k candidates.
- We generate k seconds words for each candidate.
- We select the top k [first word + second word] combinations by adding their scores.
- for these k second words, select k third words and again, chose the top k [first word + second word + third word] combinations.
- repeat at each decode step.
- when k sequences are finished, choose the sentence with the highest score.

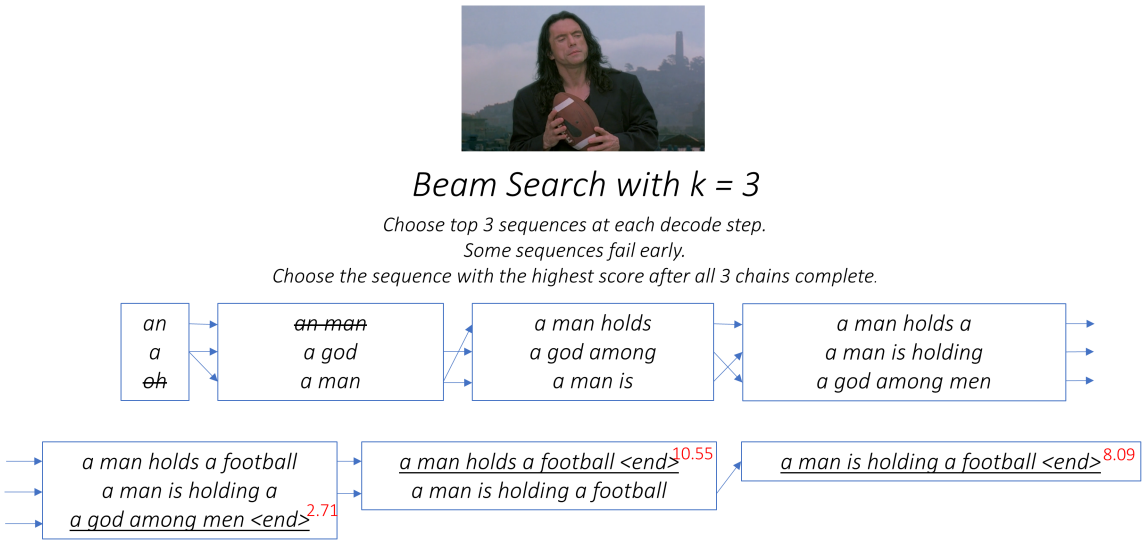

Figure 1.12: beam search illustration [6]

As it is shown in the example represented on the figure 1.12, some sequenced may be disposed early since their score is too low compared with other sequences. Once a sequence generated the <end> item, the sentence

is validated. When three sentences are finished, we choose the one with the highest score between the k sentence.

Chapter 2

Databases Manipulation

In this chapter, we will begin by describing the datasets used for this thesis realization. In a second time, we will explain how we used them to create a new dataset, ready to be utilized in the training of a neural network for image captioning.

2.1 Presentation of the databases

As we have already spoken about, obtaining a complete and exact description of an image is essential in a large number of applications.
To obtain these descriptions, a neural system need to be trained efficiently. A lot of datasets exists, containing numerous pictures with for example object attributes, scene attributes, keypoints, 3D scene information, etc [7].
Each one of these databases try to provide the best material to train a neural network in the best way possible, allowing to give the most exact description of a scenery.

We will now briefly present the two datasets used in this thesis to create a new one.

2.1.1 MS COCO

Created in 2015, **MS COCO** is a largely used database for object detection and recognition. The goal of its creators was to correct three problems they identified in scene understanding: the non-iconic views detection, the contextual reasoning between objects and the precise two dimensional localization of objects. [7]
Indeed, if a system can easily find and recognize an object in the center of the image, unobstructed, in the first plan etc, on the other hand it will encounter more difficulties if this object is in the background, partially occluded, amid clutter, ... like we would see them in a everyday scene. The

MS COCO database tried to take a maximum of "natural images", which contain several objects. Then, identification of the objects can also use the context and the environment of the object to recognize its nature in many case [7]. You can see an illustration of this concept on the figure 2.1.

Figure 2.1: Example of (a) iconic object images, (b) iconic scene images, (c) non-iconic images[7]

Image annotation:

To annotate the images, the dataset categorize the objects present between the 91 categories. In total, the dataset has 2,500,00 labeled instances in 328,000 images [7].

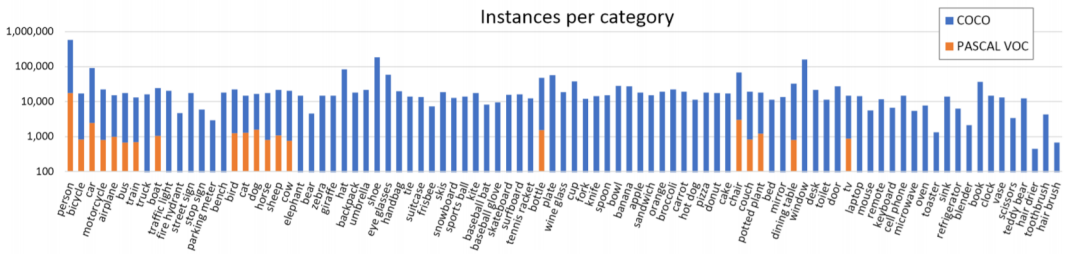

Figure 2.2: Number of annotated instances per category for MS COCO and PASCAL VOC [7]

The data can be downloaded as a json file, as well as the images used. You will find information about each pictures as its size, id, url to find it, name and the caption describing it. Notice that each image possess several captions.

You can find a complete description of the MS COCO database by following the link:
https://arxiv.org/pdf/1405.0312.pdf

2.1.2 Visual Genome

The **Visual Genome** database was created in 2016 with the purpose of increasing the cognitive capacity of computers. Indeed, in order to not just recognize and identify but to really reason about visual world, a model needs to understand not only the present objects, but also the relationships between them, their attributes etc. For example, to efficiently answer the question "What vehicle is the man riding?", the program will need to identify the objects on the picture (man, horse, carriage), but also the relationships *riding(man,carriage)* and *pulling(horse, carriage)*. It will then be able to give he proper answer: *"the person is riding a horse-drawn carriage."* [8].

Data representation:

Each image of the dataset is selected to have a high density of information. The 108,000 images have an average of 35 objects, 26 attributes and 21 pairwise relationships between objects.
Thereby, we will find for each image information as [8]:

- **Regions and their description**: the image is divided in smaller parts with a description of it.
- **Objects and their bounding box**: each objects present on the image are identified and their position is given by a bounding box (i.e. four coordinate points to frame the object)
- **A set of attributes**: the object can have some attributes as their color, their state (for example *standing*)
- **A set of relationship**: two objects can be linked by a relationship (for example *jumping over*)
- **A set of region graph**: a graph which represent each region will be generated by combining the objects, attributes and relationship
- **One scene graph**: unlike the region graph, this time it will represent the entire picture
- **A set of question/answer**: different QA pairs are provided, based on regions or on the entire image

On the figure 2.3, you can see a picture with three region graphs. The first one identify four objects *(man, woman, bench and river)* . There are also three relationships set: *in front of(man, river)*, *sits on(man and woman, river)*.
The other two regions show the attributes of the objects. The *bench* has the attributes *worn*, *wooden*, *grey* and *weathered*, while the *man* has the attribute *bald*.

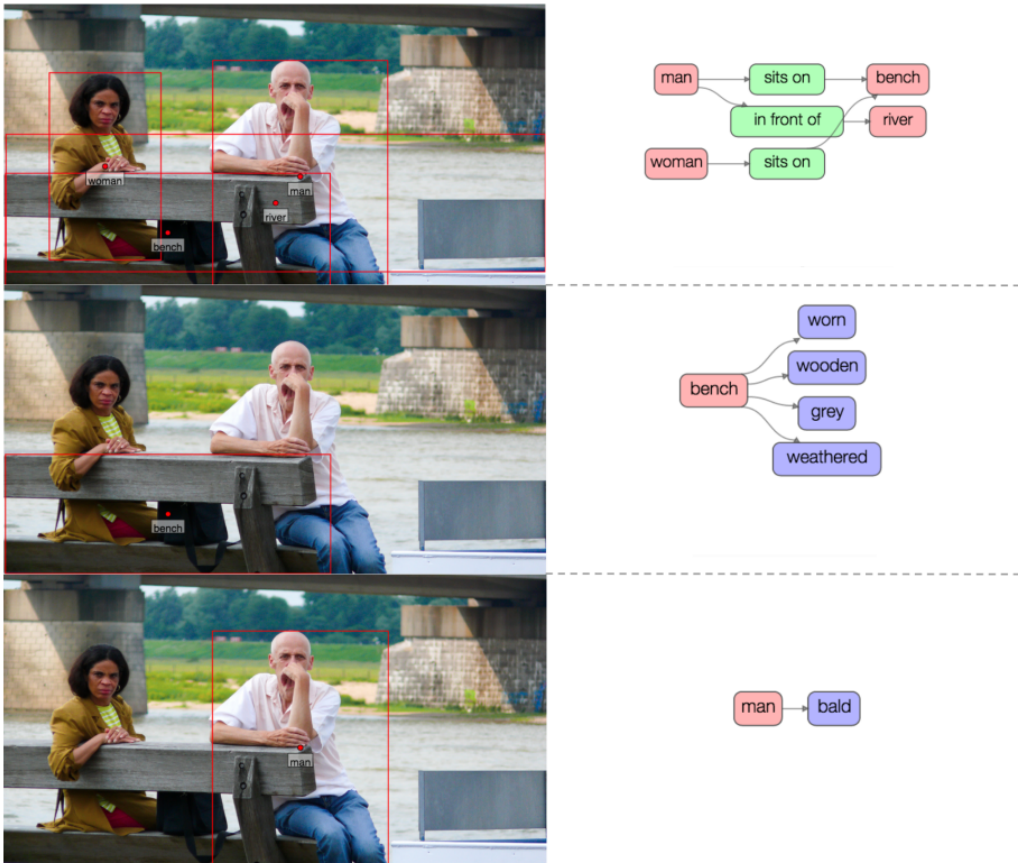

Figure 2.3: An example image from the Visual Genome dataset with 3 region graphs [8]

Finally, on the figure 2.4 you can see the entire scene graph which combine all the different region graph. Thus, it will contain all the information needed: the identified objects, their relationships and their attributes.
You can see that this "simple" image possess 19 objects, 26 attributes and 16 relationships.
Notice that you can retrieve elements of the region graphs previously shown.

Figure 2.4: Scene graph of the image in the figure 2.3 [8]

You can find a complete description of the Visual Genome database by following the link:
https://arxiv.org/abs/1602.07332

2.2 Manipulation algorithm

Today, there are more and more projects where robots must have an efficient description of its surrounding. Be it for the robot to interact with its environment, or to collaborate with a human user, the more this description will be complete, the more efficient its actions will be and the more complex will be the tasks which it will be able to process.

The aim of this master thesis is to improve the results of the image captioning.

For this purpose, we will work on the databases used to train the model which process the image captioning itself. To be more specific, we will try to take advantages from the two databases **MS COCO** and **Visual Genome**. To this end, we will based our new database on the captions from the MS COCO dataset and improve it by adding bounding box from Visual Genome. By doing it, the aim is that the model will localize more easily the most

important elements on the picture.
To explain it in an simple way, we will look at each word in the MS COCO caption and try to find the matching bounding box in the Visual Genome data. This action is possible because the Visual Genome pictures database is greatly based on the same images as MS COCO [8].

2.2.1 Databases format

The datasets from MS COCO and Visual Genome can be both downloaded on their website. We worked for this thesis with the following versions of the data:

For **MS COCO** from the 2017 version:

- captions_train2017.json
- captions_val2017.json

For **Visual Genome**, from the version 1.4 of July 12, 2017:

- image_data.json
- objects.json

As you can see, all these files are in json format thus, we will work in python and use dictionaries. As a reminder, dictionaries are a data type, which can contain unordered, changeable and indexed data [22]. They are implemented like shown in this example:

```
this_dictionary = {
    "brand": "Frod",
    "model": "Mustang",
    "year": 1964
}
```

Where the dictionary *this_dictionary* contains two strings elements *brand* and *model* with respectively the value "Frod" and "Mustang" and a int elements *year* equal to 1964.
In our case, the two files from MS COCO, *captions_train2017.json* and *captions_val2017.json* are in the same form. They only treat different images separated for the purpose of training and evaluate the model. This is why the *captions_train2017.json* file is much bigger. To get all the information for our new database, the two sets must be processed. They are in the following form:

```
captions_val2017 = {
    "info": {
        "description": "COCO 2017 Dataset",
        "url": "http://cocodataset.org",
        "version": "1.0",
        "year": 2017,
        "contributor": "COCO Consortium",
        "date_created": "2017/09/01"
    }
    "licenses": [{
        "url": "http://creativecommons.org/licenses/by-nc-sa
    /2.0/",
        "id": 1,
        "name": "Attribution-NonCommercial-ShareAlike License"
    },...]
    "images": [{
        "license": 4,
        "file_name": "000000397133.jpg",
        "coco_url": "http://images.cocodataset.org/val2017
    /000000397133.jpg",
        "height": 427,
        "width": 640,
        "date_captured": "2013-11-14 17:02:52",
        "flickr_url": "http://farm7.staticflickr.com
    /6116/6255196340_da26cf2c9e_z.jpg",
        "id": 397133
    },...]
    "annotations": [{
        "image_id": 179765,
        "id": 38,
        "caption": "A black Honda motorcycle parked in front of
    a garage."
    },...]
}
```

As you can guess, there is a lot of elements in this dictionary. It even contains several lists of dictionaries as data *(licenses, images and annotations)*. The useful information for our project are:

- the **image_id** in annotations which indicates with which picture we are working.
- the **caption** in annotations which is the sentence that describe the picture.

For the Visual Genome files, in the first time, we will use *image_data.json* which has the form:

```
image_data = [{
    "width": 375,
```

```
3       "url": "https://cs.stanford.edu/people/rak248/VG_100K
        /2338537.jpg",
4       "height": 500,
5       "image_id": 2338537,
6       "coco_id": 341778,
7       "flickr_id": 3309377405
8   },...]
```

where we can once again find a list of dictionaries. What we will use this time is the image_id and the coco_id. They will be used to find the matching between the two databases. Indeed, if they used common images, the two datasets don't have the same identification system. Fortunately, since MS COCO's pictures are used to build Visual Genome, it kept the identification used in MS COCO in memory.

Finally, the last file used from Visual Genome is *objects.json* and has the form:

```
1   objects = [{
2       "image_id": 1,
3       "objects": [{
4           "synsets": ["clock.n.01"],
5           "h": 339,
6           "object_id": 1058498,
7           "names": ["clock"],
8           "w": 79,
9           "y": 91,
10          "x": 421
11      }, ...]
12  },...]
```

As you ca see, this time we have a list of objects with their information for each picture. This is why we have a list of dictionaries inside a list of dictionaries.
For this last file, the important data are the image_id, to identify which image we are working with, the names which indicates what is the object and finally, the coordinates x, y, w and h which represent the four coordinates of the bounding box we are searching for (the position in a two dimensional plan, the width and height).
Notice that these last information are also in the file *attributes.json* allowing us to work with it instead of *objects.json*.

2.2.2 Algorithm description

We will now describe in a simple way the algorithm we used to create the new database using both MS COCO and Visual Genome.

1. The first step is to find out if the images from Visual Genome are common with MS COCO to see with which picture we will work.This information can easily be found in the Visual Genome file *image_data.json*. Indeed, the pictures coming from MS COCO possess a value for their attribute *"coco_id"*, the others have the value set as *none*. With this identification number, we can find the corresponding data from MS COCO. However, since we don't know if the image with a *coco_id* is from *captions_train2017.json* or from captions_val2017.json, we have to search for it.

2. If the image exists in both MS COCO and Visual Genome, it means that we can use it. In this case, we will need the caption from MS COCO. This caption is located in the *"annotations"* part under the form of a string. For a reminder, this is the sentence describing the picture. The right caption can be localized thanks to the *image_id* corresponding to the previous *coco_id*.

3. For every image, we will take each word of its caption separately.

4. We will look if this word correspond to an object in the Visual Genome database. For this purpose, at first, we have to find the right group of objects in the file *objects.json* by verifying the *image_id*. In a second time, we will browse the objects present in the picture and see if its name match with the word in the caption. Unfortunately, they are some compound words in the objects list, but we improve the code so that it will verify the same number of words as the object name in the *objects.json* file. In the same way, it will be able to register more than once a bounding box if several objects with the same name are represented on the picture.

5. If an object is the same as the word from the caption, we will retrieve its bounding box in *objects.json* and save it.

6. If no match is found between the word and the objects from the Visual Genome dataset, we put the coordinates at (x = -1, y = -1, w = 0 and h = 0). This means that the word was not found in the database.

7. Finally, we will save the results in a json format. This results will contains all the images present in the Visual Genome and the MS COCO database. It will provide the caption describing the picture, the identification numbers and the bounding box for each word of this caption.

You can find a summary of this algorithm in the following:

Database creation algorithm

1. in *image_data* from Visual Genome, we look if they possess a match in MS COCO, in other word, if they possess a *coco_id*
2. in the positive case, we take the caption of the picture from MS COCO
3. **for** each word in the caption:
4. | we are looking in *object.json* at the corresponding *image_id* if an object is the same as the word
5. | if we find the same word, we take the corresponding bounding box
6. | else we put the bounding box coordinates at (0,0,0,0)

   end
7. we save the results

2.2.3 Code description

In this section,we will describe several parts of the code used to process the algorithm described in the previous section.
As previously said, to be completed the algorithm must be executed with both *captions_train2017.json* and *captions_val2017.json* to be completed, but we will only present one of them for more convenience.

Obviously, the first step was the open the needed different *json* files. it can easily be done with the following lines:

```
import json
with open('captions_val2017.json') as json_data:
    data_coco = json.load(json_data);
with open('image_data.json') as json_data:
    data_image_visual = json.load(json_data);
with open('objects.json') as json_data:
    objects_visual = json.load(json_data);
coco_annotations = data_coco.get("annotations");
```

As we can see, we import the json package to be able to work with this kind of documents. Then we open the three following files: *captions_val2017.json*, *image_data.json* and *objects.json*. Finally, we extract the *"annotations"* part from *captions_val2017.json* since we will need the information it contains.

The next step is to find all the images present in both databases:

```
result = [];
size_max = 24999 # maximum size of the database (-1)
for element_visual in data_image_visual:
```

```
4       if count > size_max:
5           break
6       descrip = []
7       test_found = 0;
8       for element_coco in coco_annotations:
9           if element_visual.get("coco_id") == element_coco.get("
        image_id"):
10              test_found = 1;
11              descrip.append({"description":element_coco.get("
        caption"),"caption":None})
12              new_result = {
13                  "id_coco": element_coco.get("image_id"),
14                  "id_visual": element_visual.get("image_id"),
15                  "width": element_visual.get("width"),
16                  "height": element_visual.get("height"),
17                  "mycaption":None,
18                  "split": None #will be defined later
19              }
20      if test_found == 1:
21          result.append(new_result);
22          result[-1]['mycaption'] = descrip;
```

This part is browsing all the elements in the Visual Genome dataset to see if they can find them in MS COCO. When a match is find, we create a new element in the list *result* with as information: the MS COCO identification number *id_coco*, the Visual Genome identification number *id_visual* and the dimension of the picture. Several descriptions are provided, this is why we create a new list in *descrip* with all of them. For each one of them, we prepare the *caption* variable to receive the localization each word on the picture. Finally *descrip* is stored in *mycaption*. The *split* variable will be defined later to split the elements of the database in three group: for the training, the test and the validation. The variables with None as a value will be defined later.

This process is computed two time, for the training and validation datas from the MSCOCO dataset.

We will take the next parts of the code by small portion to understand it more easily:

```
1   for element in result:
2       new_caption = [];
```

We are browsing the elements presents in *result*. For each one of them, we will create an new list called *new_caption* which will contain the words of its *description* and the bounding bow for each word.

```
1       for objects_element in objects_visual:
2           if objects_element.get("image_id") == element.get("
        id_visual"):
3               objects_in_element = objects_element.get("objects");
```

In the Visual Genome's data from *objects.json*, we will search for the objects present in the same image than the element from *result* which we are treating. We will then take the *"objects"* dictionary from *objects.json*.

```
1               sentence = element.get("description").split();
2               position = 0;
3           for word in sentence:
```

We split the sentence describing the picture to get its words in a list form and set a counter named *position* to 'zero'. This counter will serve in the case where the object is a compound word. After that, we will browse this sentence.

```
1           mult_check = 0;
2                   for objects_in_image in objects_in_element:
3                       object_names = objects_in_image.get("names")
        ;
```

We set the *mult_check* variable to 'zero'. It is used in the case were multiple objects with a matching name are found and to indicate if we found a match between a word of the sentence and the name of an object. We are extracting the name of the objects since they are stored in a list.

```
1                       full_word = word;
2                       for name in object_names:
3                           nb_words = len(name.split());
```

This part is needed if the object is compound of several words. The word compared later is stored in *full_word* and will not be altered if the object's name is a simple word. This is why we will count the length of the name of the object, *name*.

```
1                       if (position+nb_words-1)<len(sentence):
2                           for i in range(position, (position+
        nb_words-1)):
```

```
3                                   full_word = full_word + " " +
    sentence[i+1];
```

If the position of the word in the sentence of the caption allow it, we are taking the same number of word as the object's name to make the comparison. Indeed, if for example we have an object with a two words name and we are working with the last word of the sentence, we can't take the next one and the *full_word* will be only the last one. The program knows were we are in the sentence thanks to the *position* variable.

```
1                           full_word = full_word.replace('.','');
```

This command will erase the dot at the end of the caption. Otherwise, it will be taken into account for the comparison between the last word and the objects.

```
1                           if full_word == name:
2                               mult_check = 1;
3                               new_caption.append({"word":full_word
    ,
4                                                   "x":
    objects_in_image.get("x"),
5                                                   "y":
    objects_in_image.get("y"),
6                                                   "w":
    objects_in_image.get("w"),
7                                                   "h":
    objects_in_image.get("h")
8                                                   });
9                           position = position +1;
```

If we find a word identical to an object's name, we are adding the word with its bounding box from Visual Genome in the *"caption"* part in *result*. We are setting the *mult_check* on '1' to indicate that we have found a match. The position in the sentence is being incremented too.

```
1                   if mult_check == 0:
2                       new_caption.append({"word": full_word,
3                                           "x": 0,
4                                           "y": 0,
5                                           "w": 0,
6                                           "h": 0
7                                           });
8       element["caption"] = new_caption
```

If we didn't find any object corresponding to the word, we set its bounding box to (0, 0, 0, 0). We can now save the results in *"caption"* located in the result list.

To explain it more easily, this part of the code was divided in small portions. You can find in the following box the code assembled.

```
for element in result:
    count2 = count2+1;

    if count2 < (count/20):
        element["split"] = "val"
    elif count2 < (count/10):
        element["split"] = "test"
    else:
        element["split"] = "train"
    for objects_element in objects_visual:
        if objects_element.get("image_id") == element.get("id_visual"):
            objects_in_element = objects_element.get("objects");
            for element_mycaption in element.get("mycaption"):
                new_caption = [];
                sentence = element_mycaption.get("description").split();
                position = 0;
                for word in sentence:
                    mult_check = 0;
                    for objects_in_image in objects_in_element:
                        object_names = objects_in_image.get("names");
                        full_word = word;
                        for name in object_names:
                            nb_words = len(name.split());
                            if (position+nb_words-1)<len(sentence):
                                for i in range(position, (position+nb_words-1)):
                                    full_word = full_word + " " + sentence[i+1];
                            full_word = full_word.replace('.','');# necessary because there is a dot with the last word of the sentences
                            if full_word == name:
                                if mult_check ==0:
                                    count3 = count3 +1;
                                    new_caption.append({"word": full_word,
                                                        "x": objects_in_image.get("x"),
```

```
33                                                                  "y":
      objects_in_image.get("y"),
34                                                                  "w":
      objects_in_image.get("w"),
35                                                                  "h":
      objects_in_image.get("h")
36                                                                  });
37                                      mult_check = 1;
38                                  position = position +1;
39                          if mult_check == 0:
40                              new_caption.append({"word": full_word,
41                                                  "x": 0,
42                                                  "y": 0,
43                                                  "w": 0,
44                                                  "h": 0
45                                                  });
46                      c=c+1
47                      if c==500:
48                          c=0
49                          print(count2,"/",count," elements and ",
      count3," results found")
50                      element_mycaption["caption"] = new_caption
```

Finally, the results can be saved in a new json file with the following commands.

```
1  with open('drive/My Drive/mydatabase.json', 'w') as f:
2      json.dump(result, f, indent=4)
```

2.3 Results

In the end, we obtain a list of dictionaries with the following form:

```
1  result = [{
2          "id_coco": ,
3          "id_visual": ,
4          "height": ,
5           "width": ,
6           "mycaption":[{
7                "description": ,
8                "captions":[{
9                    "word": ,
10                   "x": ,
11                   "y": ,
12                   "w": ,
13                   "h":
```

```
14                          }]
15              }],
16              "split":
17  }]
```

Finally, we obtain a new database of more than 50.000 elements. It contains as information: the identifications of the images, the size of this picture, several sentences describing it and for each word of these captions, the bounding box to localize the information on the picture.

We find more or less five objects by image with their bounding box. However, by analyzing the results, we can see that there are several captions with no objects found in the Visual Genome database. After an investigation, we have concluded that this is due to a vocabulary problem. Indeed, we saw several captions using words close from the names of the objects used in Visual Genome but not exactly the same. This simple fact prevent the algorithm to find a matching bounding box. A solution could be to use a database of synonyms when we are comparing the words of the captions with the objects.

The complete codes can be found by following the link:
https://github.com/Robin-Jacobs/TFE

Chapter 3

Image Captioning Model

3.1 Implementation

3.1.1 Dataset

The used dataset is of course, the new dataset presented in the previous chapter.

To use it, it is needed to run the program *DatabaseManipulation.py* to create the required json file. This new file named by default *mydatabase.json* contains the cations and the position of the objects mentioned in these captions. Each image possess five captions.

To run this code, you need to download the following files from the MSCOCO and Visual Genome datasets:

For **MS COCO** from the 2017 version:

- captions_train2017.json
- captions_val2017.json

For **Visual Genome**, from the version 1.4 of July 12, 2017:

- image_data.json
- objects.json

3.1.2 inputs of the model

Our model uses the four following inputs:

Images

To be compatible with the pre-trained encoder, the images needs to be processed before being used. All this pre-processing is detailed in the following link:

https://pytorch.org/docs/master/torchvision/models.html

The value of each pixel must be normalisde ot be set between 0 and 1. Then the images are all resized to 256x256 pixels.

Finally, the images are in the shape [N, 3, 256, 256] with N, the batch size.

Captions

If caption is the target of the model, it is also an input to train it.

Each word is generated using the previous one, this is why a word "zero", *<start>* is added at the beginning of the captions.
In the same way, we add *<end>* at the end of each caption to indicate the end of the sentence to the decoder.
Since the captions are stocked in fixed size tensors, we fill the missing words to reach the required size with some *<pad>* tokens.

In addition, Pytorch needs words encoded as indices to works, this is why a *word_map* is created to assign index number to each word of the vocabulary, including the *<start>, <end>* and *<pad>* tokens. A sentence become something like:

9876 1 5 120 1 5406 9878 9878 9878 ... 9878

Finally, the captions are an int tensor of shape [N, L] with L, the padded length.

Caption Lengths

Since all the captions are set to a fixed size, the length of each caption is saved to know how many token we must take without taking the *<pad>*. This length is the actual size of the sentence +2 for the two tokens *<start>* and *<end>*.

The caption length are stored in an int tensor of dimension N.

Bounding Boxes

The last element we need as an input is the localisation of the objects mentioned in the captions.
For each word, we get the four coordinates of the box which frame the object and the dimensions of the picture related.
The boxes are stored in the same order as the captions, in other word, we add some box of a size equal to 0 for the *<start>, <end>* and *<pad>* tokens. This way we can easily link a box with the word it is reporting to.

The shape is also a tensor of dimension [N, L]. each box being a list of 6 int [image width, image height, box x position, box width, box y position, box height].

3.1.3 Pre-Processing of the inputs

Before running the model, the inputs must be pre-processed as explained previously. To execute this operation, we use the files *create_input_files.py*, calling the function with the same name in *utils.py*.

This will read the data and save the result in a HDF5 file for the images and three json files for the encoded caption, their lengths and the boxes.

3.1.4 Encoder

The encoder is implemented in the file *models.py*.

As mentioned, pre-trained encoders are available in the Pytorch module. We use ResNet-101 to encode the images. If this encoder is able to classify the images too, it is not needed in this case so it is not performed.

In addition, all the images are resized to a 125x256 format for uniformity to store them in a tensor.

3.1.5 Attention

The attention is implemented in *models.py*.

It is composed of linear layers which will transform the encoded image and the hidden state (output) from the decoder in the same dimension. Then, they are added and the rectifier is activated. A final layer is added to reduce the dimension of the result to 1. Finally, a Softmax is applied to generate the weighs alphas.

3.1.6 Decoder

The decoder is implemented in *models.py* under the name of *DecoderWithAttention*.

The decoder take as an input the output of the decoder, meaning a tensor with a shape [N, 14, 14, 2048].

Of course, we do not need to process the <*pad*> tokens, this is why the captions of the N images are decreased in length. It is reminded that the size of the captions used to reduce the captions is stored in a tensor like the captions. By doing so, the captions and images are sorted.

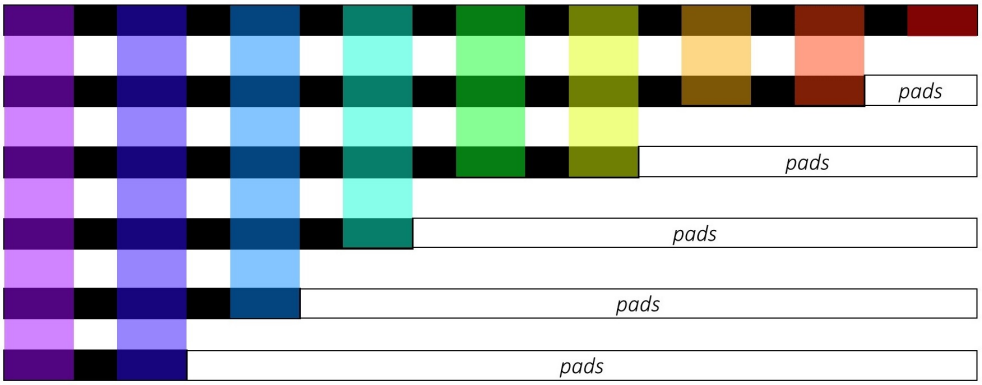

Figure 3.1: Padded sequences sorted by decreasing lengths[6]

At each time step, only the colored region is processed, therefore the batch size is changing, and we define the effective batch size N_t. The sorting is needed for this step. Indeed, it is necessary so only the top elements with more than N_t words in their caption will be processed. For example, at the third step, we process only the top 5 images, using the top 5 outputs from the previous step since these outputs are inputs for the decoding.

The attention is performed between each step of the decoder, this is why the iteration is performed manually. The weights and attention are computed at each step. The developers of the visual attention recommend to pass the attention weighted through a filter or a gate to gain more emphasis on the objects in the image.

This attention is concatenated with the embedding of the previous word to process the new hidden state. This new hidden state is transformed into

score for each word of the vocabulary thanks to a linear layer.

In addition, these weights are used with the localization of the boxes for each word to compute a new term for the loss function. This operation is performed for each word for each caption and will be detailed later in this report.

3.2 Utilization of the model

In this section, we will briefly describe how to use the model presented in this report.

3.2.1 Pre-processing

To use this image captioning model, we need to run a few step in order to prepare the data.

First, this model is based on a new dataset created in the beginning of this thesis. We need to create it. This is why we need to launch the file *DatabaseManipulation.py*. As mentioned before, some files are needed:

For **MS COCO** from the 2017 version:

- captions_train2017.json
- captions_val2017.json

For **Visual Genome**, from the version 1.4 of July 12, 2017:

- image_data.json
- objects.json

It will generate a new json file *mydatabase.json*.

In addition to this file, we will need the images from Visual Genome to run the next process: *create_input_files.py*.

This way, all the files needed for the training, validation and the word map will be generated.

3.2.2 Training

When the previously mentioned files are generated, the training can be launched with the file *train.py*.
The parameters are in the beginning of the file and can be modified to point

to the right file which contains the pre-processing files.

The training can also be resume from a checkpoint by pointing to it in the beginning of the code. Indeed, checkpoint are saved during the training and named *BEST_checkpoint_coco_5_cap_per_img_5_min_word_freq.pth.tar* by default.

Validation is performed after each epoch of the training and give scores of the model.

3.2.3 Captioning

To caption an image, we can use the command:

python caption.py –img='path/to/image.jpeg' model='path/to/BEST_checkpoint_coco_5_cap_per_img_5_min_word_freq.pth.tar' word$_m ap =' path/to/WORDMAP_coco_5_cap_per_img_5 _min_w ord_freq.json' beam_size = 5$

in this command, we specify the path to the image we want to caption, to the word map, to the trained model (the best checkpoint) and the size for the beam search.

We get as a result something like the figure 3.2. We can see the caption describing the image and the attention for each word of the sentence.

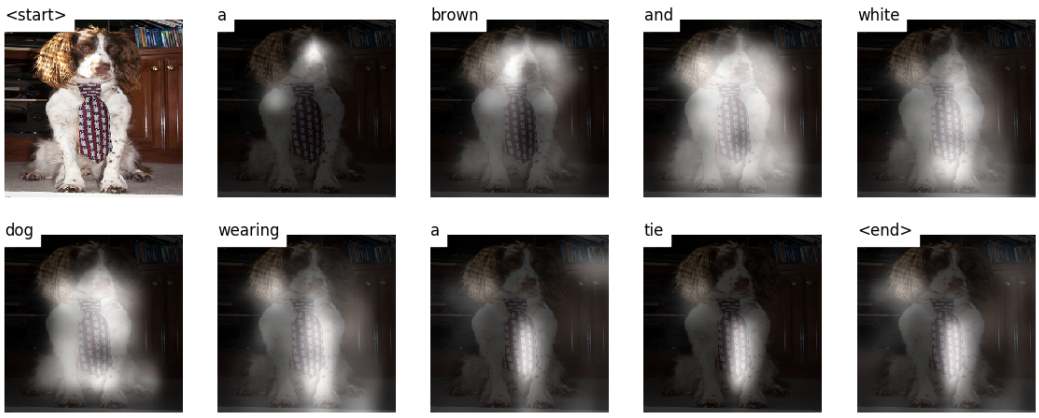

Figure 3.2: An example of result

3.3 Loss Function

The output of the model is a sentence, a sequence of words. This is why we use a cross entropy loss which is indicated for this purpose. We only need to pass the raw scores from the decoder and the function will process the softmax and log operations.

In addition, the developers of the visual attention method recommend to add a "doubly stochastic regularization" as a second loss. Indeed, thanks to a softmax function, the sum of the attention weights is equal to 1, but it is also encouraged to the weights of a single pixel p to sum to 1 across all timesteps T.

$$\sum_{t}^{T} \alpha_(p,t) \approx 1$$

This added loss can be interpreted as encouraging the model to pay equal attention to every part of the image across the time. It was noticed that it improved the scores of the captioning and lead to richer descriptions.
In pactice, we add to the loss function the term :

$$Loss+ = (1 - \sum \alpha)^2$$

Finally, we are adding a last term to our loss function. This term is calculate from the bounding box and will be explained in a more detailed way in the following subsection.

3.3.1 New Loss Contribution

This new term in the loss represents the ability of the model to localize efficiently the objects it needs to mention in the caption.

For this purpose, we will finally use the bounding boxes passed as an input in our model. At each time step of the decoder, we will compare the localisation of the object of the model thanks to the weights alphas with the real localization given by the bounding boxes. By doing so, it is hoped that the model will be improved and will localize the important information of the image more efficiently.

The first idea to compare these localization was to use the **Intersection over Union** or (**IoU**) method. This is a tool especially designed to evaluate the accuracy of an object detection on a particular dataset [9].

Intersection over Union can be computed by:

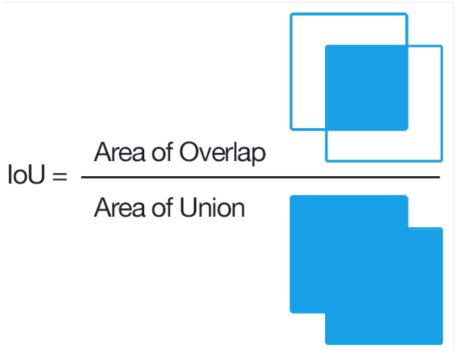

Figure 3.3: Computing the Intersection of Union is dividing the area of overlap between the bounding boxes by the area of union [9]

Unfortunately, we have quickly realised that this method will not be the best choice in our case:

Indeed, this method can easily be computed when the two positions are delimited by square figures. As a matter of fact, if it is quick and easy to calculate the intersection and union area in this case, in our model, the alphas will give irregular forms, which make it more difficult to compute these areas.
In addition, the alphas at a given time is a matrix representing the image, with for each pixel, a coefficient given its importance. In other word, the pixels of the entire image have a value and the area of the object given by the alphas is always the entire picture. The figure 3.4 show how the alphas can be represented on a picture with a shape [5x5].

0,001	0,001	0,001	0,001	0,001
0,002	0,005	0,005	0,01	0,007
0,003	0,005	0,09	0,12	0,08
0,003	0,008	0,1	0,2	0,1
0,002	0,005	0,06	0,1	0,09

Figure 3.4: Example of alphas matrix

As we can see, all pixels have a value and it is difficult to find a defined

form.

Since the IoU doesn't seem to be available in our case, the idea we have developed to measure the accuracy of the localisation is the following:

For each bounding box, we will create a new matrix of the size of the image. The terms of this matrix will get as a value 1 if it is inside the box and 0 if it is outside.
We will then multiply term by term this matrix with the alphas matrix. By taking the sum of the terms of this matrix product, we will get a score between 0 and 1. The figure 3.5 show the principle of the method.

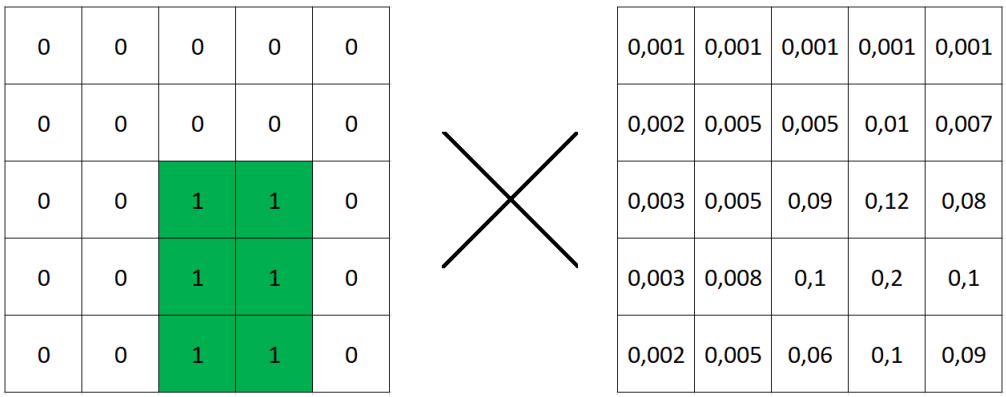

Figure 3.5: Illustration of the new loss determination

The first matrix of the figure 3.5 give the information about the localization of the bounding box, in green in the matrix; The second matrix is the matrix of he alphas. By doing a term to term multiplication, this product will give as a result the figure 3.6

It can be seen that the more the value inside the box are high, the more attention the model is supposed to give to the correct localization. This is why this method is accurate to measure the accuracy of the alphas.

We can now compute a score for the localisation :

$$score = \sum ResultMatrix = 0.09 + 0.1 + 0.06 + 0.12 + 0.2 + 0.1 = 0.67$$

Since we want to minimise the loss function, we will finally calculate the new loss by:

$$NewLoss = 1 - \sum ResultMatrix$$

0	0	0	0	0
0	0	0	0	0
0	0	0,09	0,12	0
0	0	0,1	0,2	0
0	0	0,06	0,1	0

Figure 3.6: Result of the new loss determination

3.4 Model Evaluation

The model performances are measured using the BLEU-4 metric. This method evaluates generated captions against reference captions. For each generated caption, we use all the corresponding captions from the databases as the reference captions.

The BLEU tool is available in the NLTK module.

3.5 Results

In this section, we will present some result we got from the new model developed for this thesis.

The complete codes of the model can be found by following the link: https://github.com/Robin-Jacobs/TFE

Unfortunately, we did not have the time to process the entire training of the system, meaning that the scores and results presented in this report are still not fully optimized. however, we can already give some results at this time.

During the training, we get results such as the batch time (the time needed for a batch to be processed), the Loss, the mean accuracy of the top-5 elements in the batch, and finally, at the end of each epoch, the BLEU-4 score.

Thanks to this information, we get an idea of the performances of our

model during its training. In the following table, we give the results of our first epochs:

EPOCH	LOSS	TOP-5 ACCURACY	BLEU-4
2	4.245	70.052	0.1937
3	4.163	71.342	0.2020
4	4.106	71.910	0.2063
5	4.071	72.239	0.2141

For reference, the BLEU-4 score peaked at 0.2325 at the 13^{th} epoch in the original model.
Since our model is increasing is score of about 0.007 by epoch, even is the speed of improvement is dropping, we can guess that we will exceed the result of the original model.

Even with this partially trained model, we can already generate captions with our model. Some captions generated with our model are displayed in the last section of this chapter.

3.6 Possible Improvements

Even if the first results given by this new model are encouraging, it can still be improved further. We will not mention of course that the training of the model should be finished as an improvement.
We will give in this section some leads to improve it.

At first, the database could be improved. indeed, there are several ways to improve it. We could increase the number of elements in the dataset for the training. Another solution could be to get better localization for the words of the captions. indeed, we could use a precise segmentation of the object instead of the boxes to be more accurate. In addition, and as mentioned in the database manipulation chapter, the box of some words are missed, this new database could then be redone manually to give to each word of each caption a precise localization.

The model can also be improved. By example, the method to measure the accuracy of the visual attention can probably be optimised. Indeed, for example, if the box is too large, there is a risk that the alphas will always be located inside the box, given a high score in every case. A solution to this problem could be to use the average of the alphas inside the box. An example can be seen on the figure 3.7. In this case, the box in green in way too big and the score would be equal to 0,989 even if the localization is not very good.

0,001	0,001	0,001	0,001	0,001
0,002	0,005	0,005	0,01	0,007
0,003	0,005	0,09	0,12	0,08
0,003	0,008	0,1	0,2	0,1
0,002	0,005	0,06	0,1	0,09

Figure 3.7: illustration of a wrong localization

3.7 Some Captioning

In this section are shown some results generated with our new model.

The figures 3.8 and 3.9 show the image captioning at the second and the 5^{th} epoch of the training. If the captions are the same, we can see a more precise localization of the objects in the second picture. It can be seen especially on the second word when the model will not look at the barrier but really concentrate on the man and the horse.

Figure 3.8: Example of captioning at the second epoch: A man riding a horse in a field

Figure 3.9: Example of captioning at the 4^{th} epoch: A man riding a horse in a field

The pictures 3.8 and 3.9 also show how the model really works. You can see that, if the first word is generally a determinant, the program will look at the most important objects after that: the person and the horse. At first, it considers the two objects, then will put them together with the sentence "man riding a horse". After this step, you can see that the model will pay attention the its surrounding and recognize the field. It will then generate: "A man riding a horse in a field."

You can also here see the comparison between a caption generated by the original model on the figure 3.13 and by our model on the figure 3.14. We can see that both models still have room for improvements, but let's say that a cat is closer to a cat than a sheep.

Figure 3.10: Example of captioning: A zebra grazing on grass in a field

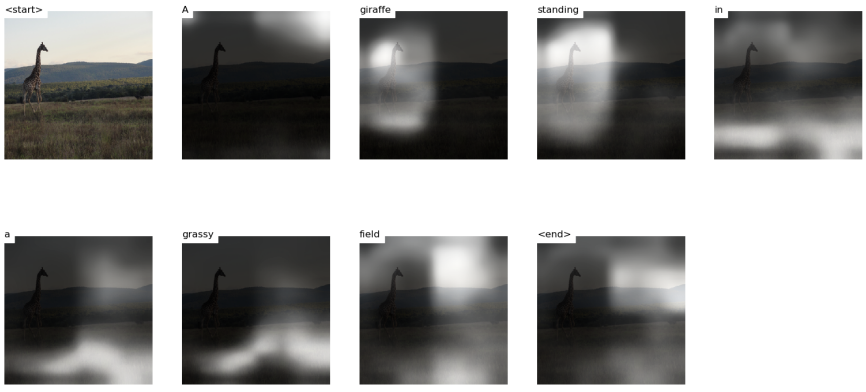

Figure 3.11: Example of captioning: A giraffe standing in a grassy field

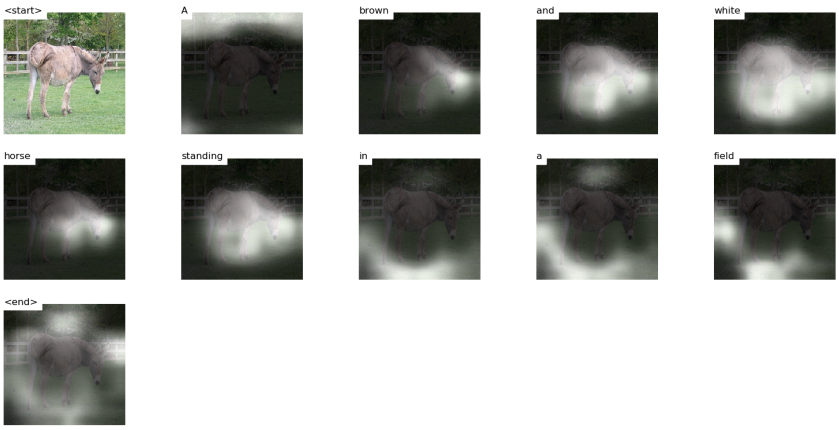

Figure 3.12: Example of captioning: A brown and white horse standing in a field

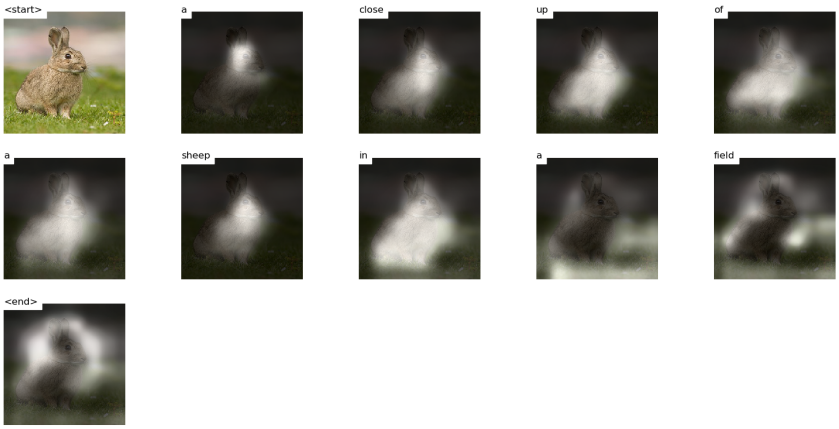

Figure 3.13: Captioning with the old model: A close up of a sheep in a field

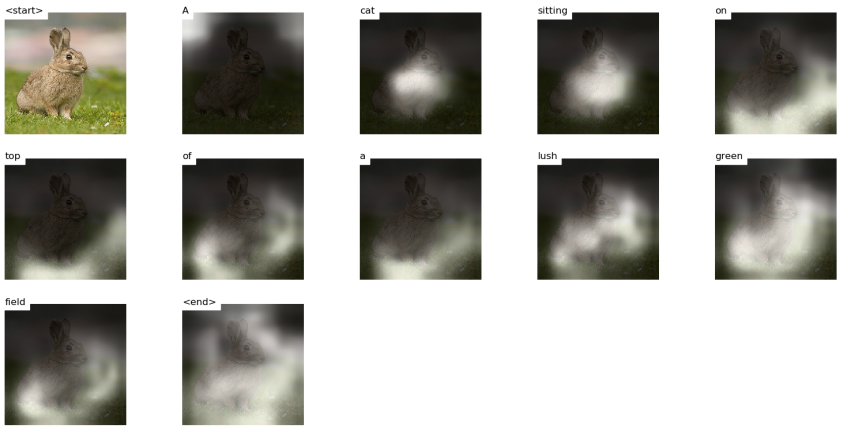

Figure 3.14: captioning with our new model: A cat sitting on top a lush green field

Conclusion

The domain of artificial intelligence is very innovative and progress in this area occurs almost every day. Therefore, A smooth interaction between the user and a robot or a computer doesn't seem to be a dream anymore.

It is obvious than such a complex technology requires a lot of effort to work properly.
Indeed, before being able to interact with it, a robot need to recognize and understand its environment. This is why the analysis of an image and the ability to describe it is so important. This skill called "image captioning" is essential for a robot able to understand its surrounding. It will be then allowed to take more and more complex actions.

The aim of this thesis aimed to increase the results get thanks an existing deep learning model of image captioning.

To accomplish this goal, the innovation was to combine two databases which use different methodology to generate a new one. The idea was to create a dataset meant to teach the model to localize the important information on the image.

Essential steps to do that were learning the essential knowledge and various notions which helped to understand this thesis and realize it. Then, studying several databases before merging two of them to create a new one. These new data are essential to be able to complete the task required. The next step was to study the model on which was based our thesis and to understand it. It was then possible to modify this model and adapt it to use the new database. By doing so, this new image captioning model was able to learn how to find the most significant area on a picture by using visual attention combined with our database. For this purpose, we have imagined a new solution to measure the accuracy of the localization of the information. Finally, we have shown some results generated by this new model.

Of course, this new model can still be improved, for example by getting more precise localization of the objects in the database or by improving the

measure of the accuracy of the alphas in the model.

We are hoping that those few steps helped to contribute to the enrichment of the scientific knowledge and more specifically in the artificial intelligence area.

Bibliography

[1] Sumit Saha. A comprehensive guide to convolutional neural networks, Dec 2018. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[2] Suvro Banerjee. An introduction to recurrent neural network, May 2018. https://medium.com/explore-artificial-intelligence/an-introduction-to-recurrent-neural-networks-72c97bf0912.

[3] wikipedia. Markov decision process, may 2019. https://en.wikipedia.org/wiki/Markov_decision_process.

[4] Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Courville Aaron, Ruslan Salakhudinov, Richard S. Zemal, and Bengio Yoshua. Show, attend and tell: Neural image caption generation with visual attention, Apr 2016. https://arxiv.org/pdf/1502.03044.pdf.

[5] Jonathan Hui. Soft & hard attention, Mar 2017. https://jhui.github.io/2017/03/15/Soft-and-hard-attention/.

[6] Sagar Vinodababu. a pytorch tutorial to image captioning, 2018. https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning/issues?page=2q=is%3Aissue+is%3Aopen.

[7] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piort Dollar. Microsoft coco: Common objects in context, Feb 2015. https://arxiv.org/pdf/1405.0312.pdf.

[8] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li Li-Jia, David A. Shamma, Michael S. Bernstein, and Fei-Fei Li. Visual genome: Connecting language and vision using crowdsourced dense image annotations, Feb 2016. https://arxiv.org/abs/1602.07332.

[9] Adrian Rosebrock. Intersection over union (iou) for object detection, Nov 2016. https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

[10] Li Zhang, Flood Sung, Feng Liu, Tao Xiang, Shaogang Gong, Yongxin Yang, and Timothy M. Hospedales. Actor-critic sequence training for image captioning, nov 2017. https://arxiv.org/abs/1706.09601.

[11] wikipedia. Automatic image annotation, apr 2019. https://en.wikipedia.org/wiki/Automatic_image_annotation.

[12] wikipedia. Reinforcement learning, may 2019. https://en.wikipedia.org/wiki/Reinforcement_learning.

[13] skymind. A beginner's guide to deep reinforcement learning. https://skymind.ai/wiki/deep-reinforcement-learning.

[14] Kashif Ali Siddiqui. What is an encoder/decoder in deep learning?, Mar 2018. https://www.quora.com/What-is-an-Encoder-Decoder-in-Deep-Learning.

[15] wikipedia. Convolutional neural network, may 2019. https://en.wikipedia.org/wiki/Convolutional_neural_network.

[16] wikipedia. Recurrent neural network, may 2019. https://en.wikipedia.org/wiki/Recurrent_neural_network.

[17] "Colah". understanding lstm networks, Aug 2015. https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[18] wikipedia. Long short-term memory, may 2019. https://en.wikipedia.org/wiki/Long_short-term_memory.

[19] Bob Givan and Ron Parr. An introduction to markov decision process. https://www.cs.rice.edu/ vardi/dag01/givan1.pdf.

[20] Mohammad Ashraf. Reinforcement learning demystified: Markov decision processes (part 1), Apr 2018. https://towardsdatascience.com/reinforcement-learning-demystified-markov-decision-processes-part-1-bf00dda41690.

[21] wikipedia. Beam search, jun 2019. https://en.wikipedia.org/wiki/Beam_search.

[22] "w3schools". Python dictionaries, 2016. https://www.w3schools.com/python/python_dictionaries.asp.