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. Stéphane DUPONT (co-proponent)
June 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. 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 explain the model

of image captioning on which we based this study and how it can be used to improve the results for image captioning by involving visual attention.

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 [9]. 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[9]. 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 [9] (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 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 [14].
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.

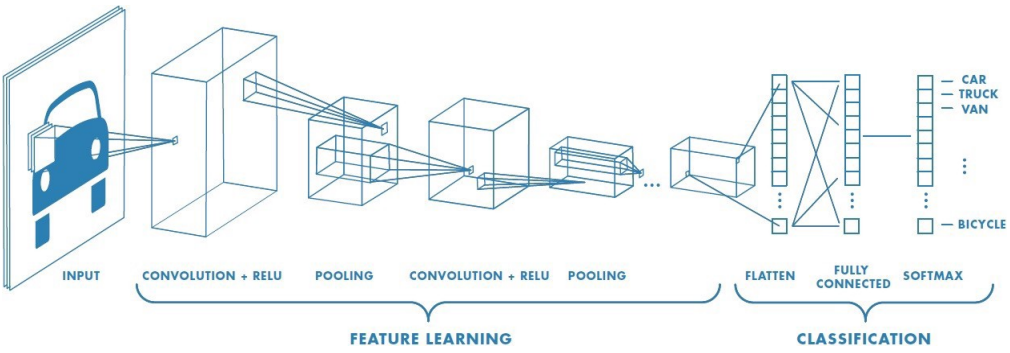

Figure 1.1: Illustration of a CNN[1]

1.4 Recurrent Neural Network (RNN)

The **Recurrent Neural Network** (RNN) is a class of neural network which can study temporal dynamic behavior [15]. 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].

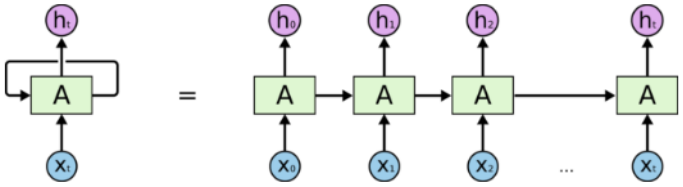

Figure 1.2: An unrolled recurrent neural network[2]

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.

1.5 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 [16][17]. In these cases, we use a special kind of RNN called **Long Short-Term Memory** networks.

1.6 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 [18]:

- 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

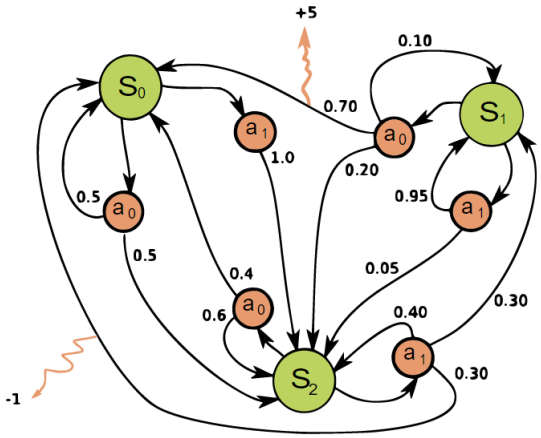

Figure 1.3: Example of a simple MDP with three states (green circles) and two actions (orange circles), with two rewards (orange arrows)[3]

With these elements, the purpose of a Markov Decision Process is to maximize the return G_t [19].

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

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.

1.7 Actor Critic

There exist two main types of deep reinforcement learning methods:

- **Value based** which will try to determine the optimal value function. This value function will map each state action to a value. The highest value will then be the best action. This method will be good in the cases where we have a finite set of actions.[4][20]

- **Policy based** which will directly find the optimal policy without using a value function. This method is adequate when the action space is continuous or stochastic and has a faster convergence but the main issue is to determine an accurate score function which will correctly

evaluate our policy.[20]

The Actor Critic algorithm will be the fusion of the two methods. The purpose is to take the advantages of both while eliminate their drawbacks.

To do that, we will split the model in two separate networks:[4]

- An Actor which decides which action to take
- A Critic which will evaluate the actions of the actor and tell how to adjust

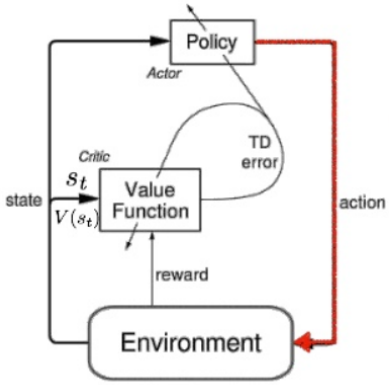

Figure 1.4: Actor Critic representation[4]

A good analogy to easily explain the actor critic is a young boy and his mother [4].
During his childhood, the boy (the actor) will explore his environment and try a lot of different actions (trying to walk, putting things in his mouth, playing, ...). During this time, his mother (the critic) will constantly watch him and will punish or reward him accordingly to his actions. The child will then adjust his behavior following his mother remarks. As the time will pass, the kid will learn which action is nice or bad and how to behave in his life. This is exactly the idea behind the actor critic.

The Actor Critic algorithm works using two neural networks which will be updated at each step (TD Learning). We can evaluate both: [20]

- Actor: a policy function which control how the agent acts

$$\pi(s, a, \theta)$$

- Critic: a value function which measure how good are these actions

$$\hat{q}(s, a, w)$$

Each model run in parallel. As previously said, the Actor is trained and will update the policy using the feedback from the Critic and in the same time, the Critic will also update his own way to provide feedback. This is why we have two set of weights (θ for our Action and w for our Critic) that must be optimized separately.[20]

Policy Update: $\Delta\theta = \alpha \nabla_\theta(log\ \pi_\theta(s, a))\hat{q}_w(s, a)$

q learning function approximation (estimate action value)

Value update: $\Delta w = \beta\ (R(s, a) + \gamma\hat{q}_w(s_{t+1}, a_{t+1}) - \hat{q}_w(s_t, a_t))\nabla_w\hat{q}_w(s_t, a_t)$

Policy and value have different learning rates

TD error

Gradient of our value function

1.8 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".

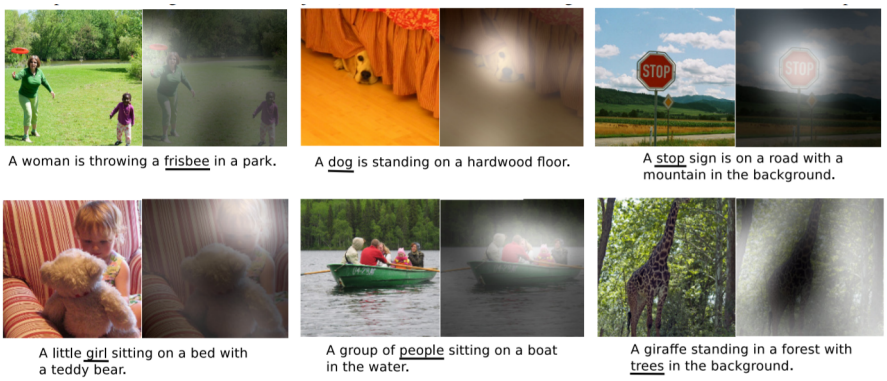

Figure 1.5: Exemple of attending to the correct object [5]

The figure 1.5 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 1.6 since the model will focus on the different objects represented on the picture [5]. Identify the importance of these object and improving the accuracy of this method is the purpose of this thesis.

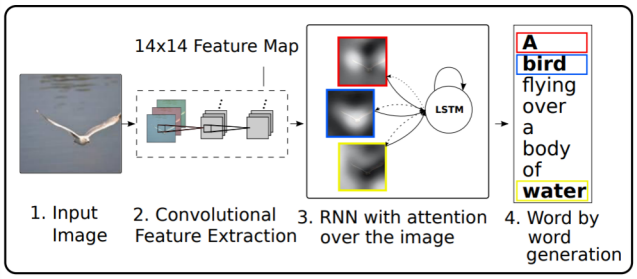

Figure 1.6: Image captioning with model of attention[5]

The weighted features for a LSTM can be computed with the following method [6]:

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

$$Z = \sum_i \alpha_i x_i$$

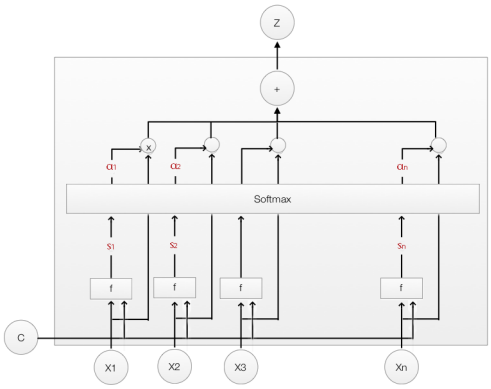

Figure 1.7: computation of the weighted features [6]

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].

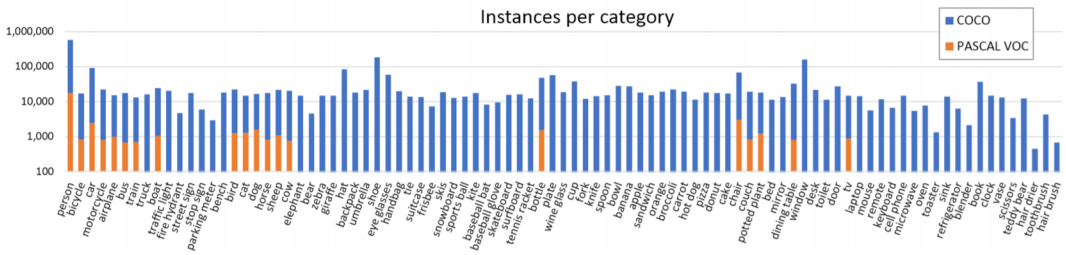

Figure 2.2: Number of annotated instances per category for MS COCO and PASCAL VOC [7]

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*.

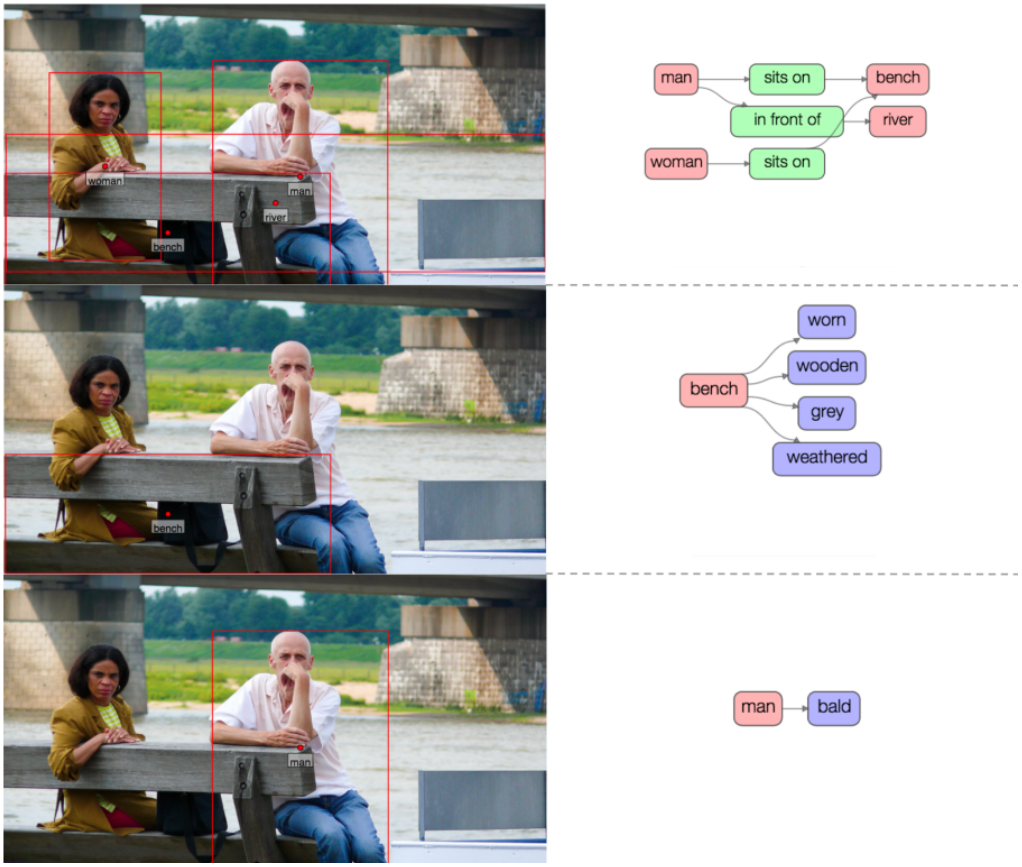

Figure 2.3: An example image from the Visual Genome dataset with 3 region graphs [8]

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 [21]. 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 (-1,-1,0,0)

   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 = [];
for element_visual in data_image_visual:
    for element_coco in coco_annotations:
```

```
4           if element_visual.get("coco_id") == element_coco.get("
    image_id"):
5               new_result = {
6                   "id_coco": element_coco.get("image_id"),
7                   "id_visual": element_visual.get("image_id"),
8                   "description": element_coco.get("caption"),
9                   "caption": None #will be defined later
10              }
11              result.append(new_result);
```

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*, the description of the picture, in other word its caption *description* and finally, we create a last parameter called *caption* with *None* as value. This parameter will be used later to store each word with its bounding box. Each new element is added at the end of the list *result*.

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.

```
                            if full_word == name:
                                mult_check = 1;
                                new_caption.append({"word":full_word
    ,
                                                    "x":
    objects_in_image.get("x"),
                                                    "y":
    objects_in_image.get("y"),
                                                    "w":
    objects_in_image.get("w"),
                                                    "h":
    objects_in_image.get("h")
                                                    });
                            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.

```
                    if mult_check == 0:
                        new_caption.append({"word": full_word,
                                            "x": -1,
                                            "y": -1,
                                            "w": 0,
                                            "h": 0
                                            });
    element["caption"] = new_caption
```

If we didn't find any object corresponding to the word, we set its bounding box to (-1, -1, 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:
    new_caption = [];
    for objects_element in objects_visual:
        if objects_element.get("image_id") == element.get("
    id_visual"):
```

```
5               objects_in_element = objects_element.get("objects");
6               sentence = element.get("description").split();
7               position = 0;
8               for word in sentence:
9                   mult_check = 0;
10                  for objects_in_image in objects_in_element:
11                      object_names = objects_in_image.get("names")
    ;
12                      full_word = word;
13                      for name in object_names:
14                          nb_words = len(name.split());
15                          if (position+nb_words-1)<len(sentence):
16                              for i in range(position, (position+
    nb_words-1)):
17                                  full_word = full_word + " " +
    sentence[i+1];
18                          full_word = full_word.replace('.','');
19                          if full_word == name:
20                              mult_check = 1;
21                              new_caption.append({"word":full_word
    ,
22                                                  "x":
    objects_in_image.get("x"),
23                                                  "y":
    objects_in_image.get("y"),
24                                                  "w":
    objects_in_image.get("w"),
25                                                  "h":
    objects_in_image.get("h")
26                                                  });
27                          position = position +1;
28                  if mult_check == 0:
29                      new_caption.append({"word": full_word,
30                                          "x": -1,
31                                          "y": -1,
32                                          "w": 0,
33                                          "h": 0
34                                          });
35      element["caption"] = new_caption
```

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

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

2.3 Results

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

```
result = [{
    "id_coco": ,
        "id_visual": ,
        "description": ,
        "caption":[{
        "word": ,
                "x": ,
                "y": ,
                "w": ,
                "h":
    }]
}]
```

Finally, 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

In this chapter, we will begin by presenting the model used as a basis for this thesis. After this description, we will explain how this model must be modified to be used with the new database created in the previous chapter. Finally, we will describe how to obtain the new loss function which should be used in the new model.

3.1 Presentation of the image captioning model

The thesis presented in this document is mainly based on the work explained in the scientific paper "Actor-Critic Sequence Training for Image Captioning" [9].
You can find this paper by following the link:
https://arxiv.org/pdf/1706.09601.pdf.

We will present this work in the following pages.

Most existing studies about image captioning using deep learning-based image encoder encounter two main problems:

- They are trained by a method called "Teacher-Forcing". Thereby, the network is trained by maximizing the like hood of each ground-truth word using the previous one. The problem is that during the testing phase, the model will use its own generated word to make its predictions, therefore the model will accumulate its mistakes. Thus, this creates a mismatch between training and testing.

- Cross entropy loss is often used to train the sequence model but the actual NLP quality metrics of interest (with which we evaluate them at test time) are non-differentiable metrics. Ideally sequence models

for image captioning should be trained to avoid exposure bias and directly optimize metrics for the task at hand.

The authors of the paper propose to to fix these two issues using reinforcement learning to train the captioning model. By sampling from the model during the training phase, they intend avoid the training-testing mismatch, and they propose to directly optimize the relevant rest-time metrics by considerating them as reward in a reinforcement learning context.
To be more specific, an actor-critic model is used with the public benchmark MS-COCO. The model presented by this study achieved high performance, being ranked third on the MS-COCO testing server leaderboard when submitted.

3.1.1 Methodology

The goal of this program is to generate, with an image I as input, a caption sequence $Y = \{y_1, ..., y_t\}$ describing this image. Of course, two set of images I and captions Y are set for the training and testing. The model can be evaluated thanks to the task specific score $R(\hat{Y}, Y)$, with $\hat{Y}$, the predicted caption sequence.

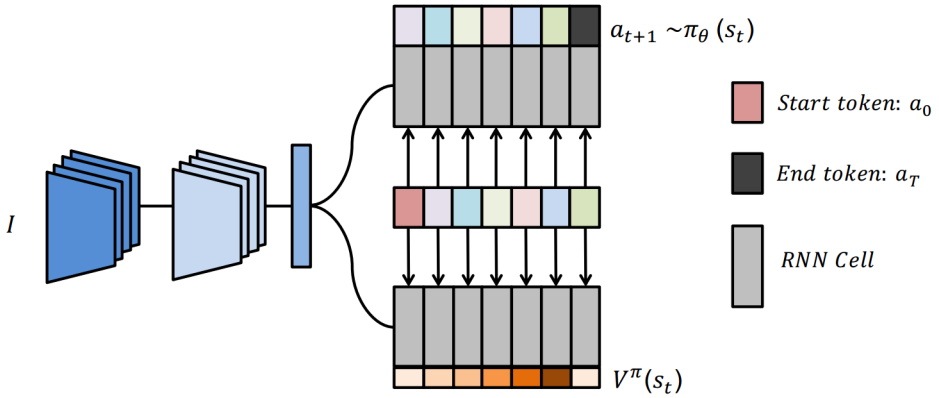

Figure 3.1: Schematic illustration of the actor-critic based captioning model (with word embedding layer omitted)[9]

The architecture used can be seen on the figure 3.1. A Convolutional Neural Network (CNN) is used as encoder and a Recurrent Neural Network (RNN) as the decoder. The image captioning problematic, can be seen as a Markov decision process (MDP) *{S, A, P, R, γ}*. In this way, the image captioning can be treated as a reinforcement learning problem. In this case, we will consider the image parts i_e (provided by the CNN from the picture

I) and the tokens/actions $\{a_0, a_1, ..., a_t\}$ as the state S. The state transition function P is $s_{t+1} = \{s_t, a_{t+1}\}$. $\gamma \in [0, 1]$ is the discount factor.[9]

3.1.2 Model

As previously said, actor-critic reinforcement learning is used to train the model. Actor-Critic is separated in two subnetworks: the policy network (**actor**) and the value network (**critic**).

Value Network

The value is the expected return as a function of the state s_t, given the policy π, the sampled actions a_t and rewards r_t. It can be written as an approximate state value function:

$$V^{\pi}(s_t) = \mathbb{E}[\sum_{l=0}^{T-t-1} \gamma^l r_{t+l+1} | a_{t+1}, ..., a_T \sim \pi, I] \tag{1}$$

The rewards are down-weighted by the discount factor γ in order to reduce the variance. However, this action introduce a bias. The easiest way to estimate the value function is to solve a nonlinear regression problem:

$$\min_{\phi} ||Q^{\pi}(s_t, a_{t+1}) - V_{\phi}^{\pi}(s_t)||^2 \tag{2}$$

with $Q^{\pi}(s_t, a_{t+1}) = (1 - \lambda) \sum_{n=1}^{\infty} G_t^n$

Policy Network

The policy network π uses a set of weights θ. At the each time t, the network will get as input the state s_t to generate the distribution over the set of possible actions. We can write: $a_{t+1} \sim \pi_{\theta}(s_t)$. The input image I is treated by the CNN to obtain I_e which is used with the start action a_0 to become the initial state s_0. In other words, we have:

$$s_0 = \{I_e, a_0\} \tag{3}$$

and therefore, we will have:

$$s_t = \{I_e, a_0, a_1, ..., a_t\} \tag{4}$$

A Long Short-Term Memory (LSTM) is used with the state s_t as input to obtain the hidden state h_{t+1}. A stochastic output layer f has to be added to generates the outputs a_{t+1}.

$$h_{t+1} = LSTM(s_t) \tag{5}$$

$$a_{t+1} \sim f(h_{t+1}) \tag{6}$$

As a result, we obtain a probability distribution of the actions a_{t+1} in function of the current state s_t.
Then, the **supervised learning** phase would consist to minimize the cross entropy loss:

$$\mathcal{L}_{XE}(\theta) = -\sum_{t=1}^{T} \log(\pi_\theta(y-t|y_0, ..., y_{t-1})) \tag{7}$$

with $\{y_0, y_1, ..., y_T\}$, the target ground-truth sequence.

By contrast, the **policy gradient** approach will maximize the expected accumulated reward. To do so, the gradient g will repeatedly be estimated. If we have the reward r_t provided in accordance with the action's efficiency, we have $g = \nabla_\theta \mathbb{E}[\sum_{t=1}^{T} r_t]$. In this configuration, the correct behavior is not shown, the program will learn by himself what are the best actions.
Using this approach, we usually train an expression of the form:

$$g = \mathbb{E}[\sum_{t=0}^{T-1} A^\pi(s_t, a_{t+1}) \nabla_\theta \log \pi_\theta(a_{t+1}|s_t)] \tag{8}$$

The advantage function $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$ tell how much an action is better (or worse) than the policy default behavior. Thereby if the advantage function is positive, the gradient term $A^\pi(s_t, a_{t+1}) \nabla_\theta \log \pi_\theta(a_{t+1}|s_t)$ will raise $\pi_\theta(a_\theta|s_t)$.
In the previous formula, we have:

$$Q^\pi(s_t, a_{t+1}) = (1-\lambda) \sum_{n=1}^{\infty} G_t^n \tag{9}$$

with G_t^n, the n-step expected return:

$$G_t^n = r_{t+1} + \gamma r_{t+2} + ... + \gamma^{n-1} r_{t+n} + \gamma^n V^\pi(s_t + n) \tag{10}$$

And we can write:

$$A^n(s_t, a_{t+1}) = Q^\pi(s_t, a_{t+1}) - V^\pi(s_t) = (1-\lambda) \sum_{n=1}^{\infty} G_t^n - V^\pi(s_t) \tag{11}$$

Furthermore, the gradient of policy network become:

$$g = \mathbb{E}[\sum_{t=0}^{T-1} ((1-\lambda) \sum_{n=1}^{\infty} G_t^n - V^\pi(s_t)) \nabla_\theta \log \pi_\theta(a_{t+1}|s_t)] \tag{12}$$

The λ parameter is set to 1 for the image captioning. This way, the estimator for the advantage and the value function is unbiased.

The reward is defined as followed:

$$r_t = \begin{cases} 0 & t < T \\ \text{score} & t = T \end{cases} \tag{13}$$

Indeed, the caption can only be evaluated when the sentence is finished. With this reward, we have:

$$Q^{\pi}(s_t, a_{t+1}) = \gamma^{T-t-1} r_T \tag{14}$$

And finally:

$$g = \mathbb{E}[\sum_{t=0}^{T-1} (\gamma^{T-t-1} r_T - V(s_t)) \nabla_\theta \log \pi_\theta(a_{t+1}|s_t)] \tag{15}$$

3.1.3 Experiments

The algorithms used are described in the following lines:

Algorithm 1: Actor-Critic Training for Image Captioning

1. **Require:** Actor $\pi(a_{t+1}|s_t)$ and Critic $V(s_t)$ with weights θ and ϕ respectively;
2. **for** *episode = 1 to max episode* **do**
3. Receive a random example *(I, Y)* and sample sequence of actions $\{a_1, ..., a_t\}$ according to current policy π_θ;
4. Compute temporal-difference target $Q^{\pi}(s_t, a_{t+1}) = \gamma^{T-t-1} r_T$ for $V(s_t)$;
5. Update Critic weights ϕ by minimizing Eq. 2
6. Update Actor weights θ using the gradient Eq. 15
7. **end**

Algorithm 2: Complete Actor-Critic Algorithm for Image Captioning

1. Initialize Actor $\pi(a_{t+1}|s_t)$ and Critic $V(s_t)$ with random weights θ and ϕ respectively;
2. Pre-train the actor to predict ground truth y_t given $\{y_1, ...y_t\}$ by minimizing Eq. 7;
3. Pre-train the critic to estimate $V(s_t)$ by running Algorithm 1 with fixed actor;
4. Run Algorithm 1;

The Actor and the Critic must be pre-trained as indicated in the algorithm 2. If they are trained directly, they would not provide adequate training response for one another. For example, in the actor's training, the tokens would receive only low rewards and be sampled randomly. Therefore, the training signal for the critic would not be relevant. In the same way, a

random critic would not be pertinent to train the actor. This is why the above algorithm is necessary.

Dataset and setting

The algorithms are used and evaluated using the **MSCOCO** dataset. It contains 82,783 training images and 40,504 validation images, each one of them annotated with about 5 captions.
To measure and compare their results, the BLEU, CIDEr, METEOR and ROUGE scores were used.

3.1.4 Results

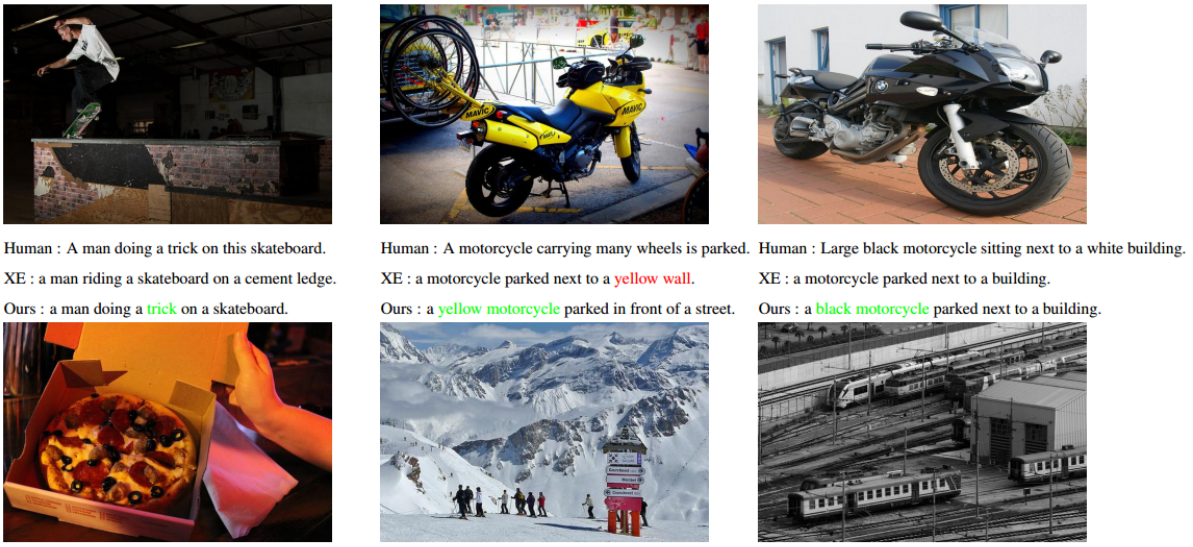

Figure 3.2: Qualitative results of image captioning on the MS COCO dataset[9]

The method proposed in the paper give very good results. It was ranked third on the MSCOCO image captioning challenge leaderboard when submitted (28 Nov 2017).
In addition, the computational cost was compared with several alternative methods and it appears that the cost was lower. This is probably due to attention cells absence.

You can see on the figure 3.2 the results compared with another algorithm using a similar architecture and with a human captioning.

3.2 Visual attention comparison

The final goal of the project initiated by this master thesis is to train the model which we have presented in the last section by comparing the bounding box from our new database with the visual attention generated by the model.

Unfortunately, the present model use as input for its system the entire image I which is treated by the CNN to obtain I_e. Then, I_e is used with the initial action a_0 to form the initial state of the system s_0.

This form is not adequate for our purpose. Indeed, the model will need to be transformed into a classical attention model in order to be able to localized the visual attention generated by the network. As seen in the first chapter, it is possible to change the input format to give to the program a weighted image features instead.

Then, we will be able to compare this visual attention with the bounding box from the database and base a model on this comparison. The system would then learn to efficiently locate the most important area on a picture.

This step of the project should be realized in a future thesis.

3.3 Establishment of the new loss function

Since the program will now compare the bounding box of the objects with their positions given by the visual attention of the model, it will need a new loss function to complete its purpose.

This loss function could be evaluated with the help of the **Intersection over Union** (IoU) method. It allows to evaluate the accuracy of an object detection on a particular dataset [10].

Intersection over Union can be computed by:

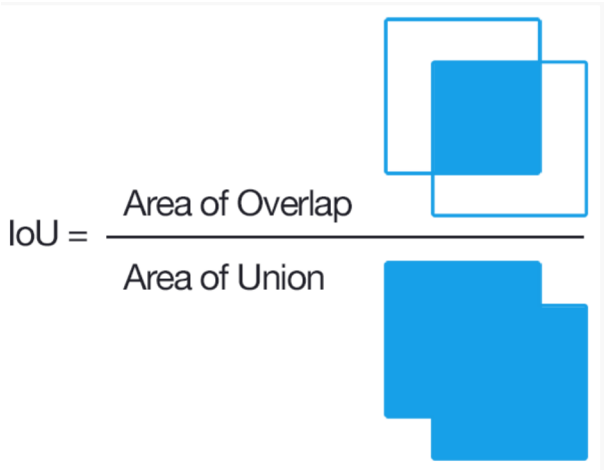

Figure 3.3: Computing the Intersection of Union is dividing the area of overlap between the bounding boxes by the area of union [10]

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. Finally, we explained how, in a future thesis, this model will be modified to learn how to find the most significant area on a picture by using visual attention combined with our database.

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] Sergios Karagiannakos. The idea behind actor-critics and how a2c and a3c improve them, nov 2019. https://sergioskar.github.io/Actor_critics/.

[5] 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.

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

[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] 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.

[10] 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/.

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

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

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

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

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

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

[20] Thomas Simonini. An intro to advantage actor critic methods: let's play sonic the hedgehog!, jul 2018. https://sergioskar.github.io/Actor_critics/.

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