CS512 - FINAL PROJECT REPORT

“Caption generation”

A20432671 Thomas Ehling
&
A20432063 Benjamin Scialom

Table of Contents :

I. Project Statement

The goal of the project was to generate captions for a given picture. It requires to understand the content of the picture and use natural language processing to produce the caption. This topic is constantly examined by scientist in computer vision and deep learning.

Neural Image Caption (NIC) uses a **convolution neural network** that encodes an image into a compact representation, followed by a recurrent neural network (RNN : language generating neural network) that generates a corresponding sentence. The model is trained and evaluate to maximize the likelihood of the sentence given the image.

According to our publication the encoder RNN is replacing by a deep convolutional neural network (CNN). Actually, it is very common to use a CNN as an image "encoder", by first pre-training it for an image classification task and using the last hidden layer as an input to the RNN decoder that generates sentences.

However, our project focused on the computer vision part. So we just adapted a code from internet for the natural language processing part.

<u>There are 4 main steps to our project :</u>

- Prepare the data-set of Flickr8k : extract features from the images and clean the captions
- Define and train the caption generator model
- Evaluate the performance of the model
- Use the model to generate captions from an image

Every sections above Section IV "Proposed Solution" has been redacted by both Benjamin and Thomas, and therefore will figure in both of the reports. Sections IV, and all sections after it are specific to each of the student, and will be different in each of the reports.

II. Organization and responsibilities

First of all, we have worked on the general aspect/idea of the code. That is to say what was needed to implement all that was required to make the generation of caption work.
Then, we have split the project in different step/code to work on different aspects at the same time.

On one hand, the preparation of the data-set and the training have been implemented on Python notebooks, using colab. That was necessary since a large computing time is necessary to prepare the data-sets and to train the models on it.

On the other hand, the evaluation and the final test have been developed in python scripts, that can be run locally.
Doing that allow the user to generate captions from an image without using colab. He just has to run the **"test.py"** python file, giving the image and the model files.

<u>What was our responsibilities ?</u>

- We **both work on the construction of the extraction-feature model** which is used for the preparation of the data-set and the training. It is also used in the final test file.

- **Thomas** took care of the preparation of the data-set and the evaluation of the model. (prepare_dataset.ipynb, eval.py)

- **Benjamin** took care of the model training and the final test file which is used to predict caption according to a picture. (train.ipynb, test.py)

III. Implementation

We made 4 program files, one for each part of the project :

- 2 Python Notebook files :
  - prepare_dataset.ipynb : extract features from the images and clean the captions
  - train.ipynb : define and train the caption generator model

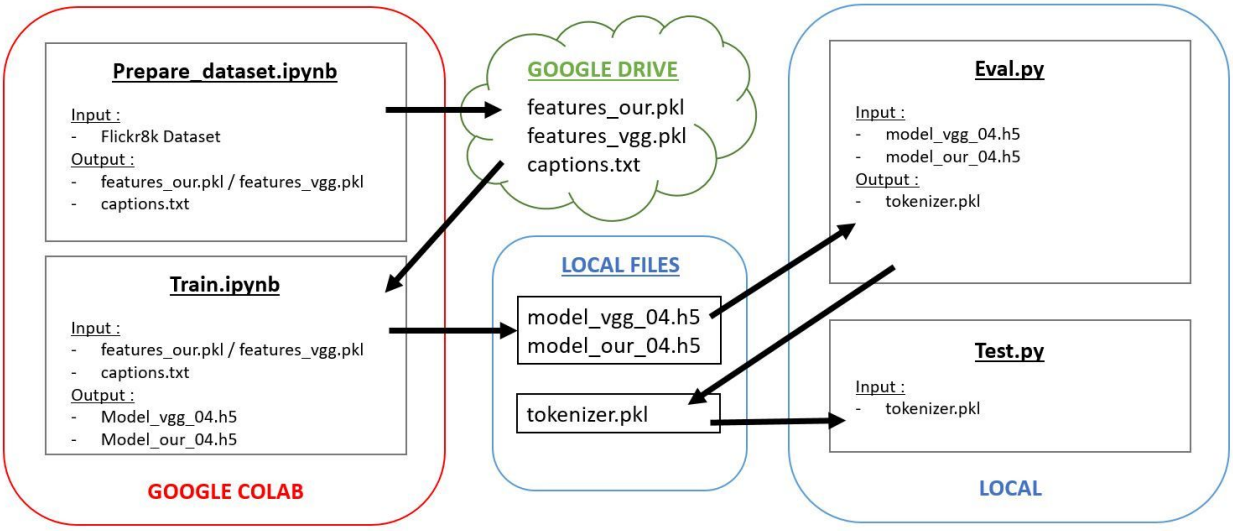
-> Make sure to create a folder CV_project in your My_drive to put and run all the .ipynb
-> prepare_dataset.ipynb has to be run first

- 2 Python files :
  - eval.py : evaluate the performance of the model
  - test.py : generate caption for the given image

The two file in the Notebook format are computing the training of the several models (cf Proposed Solutions), and have been made to be run on the Google Colab Plateform.

We used Python 3.5.

The relations between the files are represented in the following graph :



As each files generate data that are used by the other ones, there is a precise order to run them.

1. prepare_datadet.ipynb : (COMPUTATION TIME : ~1H)
   a. Create a folder “CV_Project” on your drive account

b. Open the file with the Google Colab platform
c. Click on “run all”
d. Select the Google account with the drive containing the “CV_project” folder
e. Wait for the computation to finish

2. traint.ipynb : (COMPUTATION TIME : ~2h: 17 min per epoch (4 per model))
    a. Open the file with the Google Colab platform
    b. Click on “run all”
    c. Select the Google account with the drive containing the “CV_project” folder
    d. Wait for the computation to finish
    e. Download the desired .h5 files generated in the “CV_project” folder, for the next part

3. eval.py : (COMPUTATION TIME : ~9 MINUTES)
    a. Execute the file from the command line, with the following arguments : (If no argument are given, default files will be loaded from the /data folder)
        i. **-m** next argument should be the **path of the file.h5** you want to use
        ii. **-f** next argument should be the path of the **feature.pkl** you want to use

-> for these two files used by the algorithm you have the choice between “our” or “VGG16”. If you are using “our” make sure it is precise in the name of the file (it is by default).

4. test.py :
    a. Execute the file from the command line, with the following arguments :
        i. -p next argument should be the picture you want to predict (mandatory)
        ii. -t next argument should be the token file you want to use. This should not be used.
        iii. **-m** next argument should be the **path of the file.h5** you want to use. It depends which model is used “our” or “VGG”

Because of the long computation time, all our resulting files are given in the /data folder. Doing so, any part can be executed separately, given the right files (cf graph).

IMPORTANT NOTE :

All along the project, we used two models to generate the feature : the predefined model VGG16 and our own model.

This choice have been made in order to test the functionalities very soon with the predefined model, and have a better idea of our model performance.

Therefore, whenever a model or a file is created, it is always specified if it has been made with our model or with the vgg model within the name of the file.

If you have any questions about the implementation part (or the rest of the report), please feel free to reach out to any of us to the following email :

- bscialom@hawk.iit.edu

- tehling@hawk.iit.edu

Or ask for an appointment, so we can show you a live demonstration.

IV. Proposed Solution

From now on, all the remaining of the report have been redacted by, and describe the work of I, Benjamin SCIALOM.

1. **Model training : train.ipynb**

To see the result you just have to execute the train.ipynb file on colab.
You will have to give access to your drive and create a folder CV_project in My_drive.
Moreover, the prepare_dataset.ipynb has to be run first.
Make sure to set the file in python 3 and activate the use of GPU.

How to do it ?

Modify -> set parameters : python 3 and GPU.

Inputs :

- **features_vgg.pkl :** pickle file containing the features generated from the images of the Flickr8k dataset, with the VGG16 model.
- **features_our.pkl :** pickle file containing the features generated from the images of the Flickr8k dataset, with our own model.
- **captions.txt :** text files with all the cleaned captions, computed with the captions from the Flickr8k dataset.

Outputs :

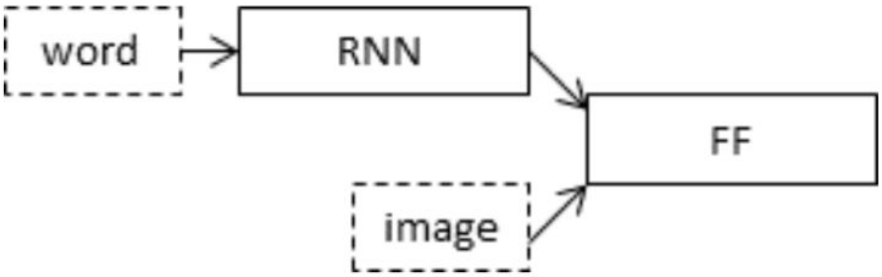
- **model_vgg_04.h5 :** file with all the weights of the caption generation model, trained on the features generated by the vgg model.
  model_our_04.h5 : file with all the weights of the caption generation model, trained on the features generated by our model.

In the Flicker8k dataset, there is a file named “trainImages.txt” that contains the list of all the images for the training and testing dataset.

After loading these files in memory, the ids are extracted and compared to the “captions.txt” created during the preparation of the data, in order to create dictionaries with all the cleaned captions for the training.

Although this project is fascinating and we chose it, there is still an a part unrelated to Computer Vision, and only concerning the Natural language processing part in order to generate fully understandable parts.
This aspect of the project is implemented in this section of the code.
It his commented with capital letters on the code : the functions "*create_sequences()*" and "*data_generator()*" have been made from online sample code (cf. references).

The caption generation model respect the format describe in the publication :



For the training, we experienced a memory issue here, as google colab cannot handle large data in memory (no more than 12 Gb).
So, the solution is to process epochs by epochs with the help of the creation of a generator and the function *model.fit_generator()*, from the keras library. As a result, we train one epoch at the time, and we do not need a big amount of RAM.The only drawback of doing that is that we can't have a nice graph on Tensorflow Board since a .h5 file is created for each epoch.

```
model = define_model(vocab_size, max_length)
epochs = 4
steps = len(train_captions)
for i in range(epochs):
      generator = data_generator(train_captions, train_features_our,
tokenizer, max_length)
      model.fit_generator(generator, epochs=1,
steps_per_epoch=steps,callbacks=[tbCallBack], verbose=1)
      model.save(PATH_DRIVE+'model_our_' + str(i) + '.h5')
```

The code just upper was for ou model as you can see on the last line but we have also implemented the one for the VGG16 model in order to compare.

However, even of colab, it takes around 45 min by epoch since the data-set is quite big and the extraction of feature for all pictures is an heavy process.

So, we trained two models, each of them with either the features extracted with our model or the feature extracted with the vgg16 model. The weights of these two models are saved to ".h5" files on the drive, under the names of "model_vgg_04.h5" and "model_our_04.h5". The 04 is here to indicate the number of epochs, as we are saving one file for reach epoch.

2. **Captioning a random picture : test.py**

Inputs :

- **model_vgg_04.h5 :** file with all the weights of the caption generation model, trained on the features generated by the vgg model.
  model_our_04.h5 : file with all the weights of the caption generation model, trained on the features generated by our model.

-> You have to choose between those two file. By default, model_our_04.h5 is selected.

- **tokenizer.pkl :** This file has been produced during the evaluation part made by Thomas. This file contains the correspondences between words and ids'.

-> This file choosen by default so you don't have to care about it.

- **Picture.jpg :** You need to enter the picture you want to captioned. You have to put the path of the picture.

So, if you are using the default configuration, it goes like this :

```
dhcp217:src benjamin.s$ python3 test.py -p ../data/boy.jpg
```

Then you will have to make a choice about the model to use. Make sure if you want to use the VGG16 model to add the right model, like this :

```
dhcp217:src benjamin.s$ python3 test.py -p ../data/boy.jpg -m ../data/model_4_VGG.h5
```

Outputs :

- The caption of the image will be displayed in the terminal.

Here, the idea is to extract features from the image and then predict the caption to display. To extract the features, we have to use a model. Let say we are using our model :

The Architecture and parameters of the model we used has been chosen by **Thomas and I together.**

```
#create the model
def create_model():
    model = Sequential()
    model.add(ZeroPadding2D((1,1),input_shape=(224,224,3)))
    model.add(Conv2D(receptive_fields[0], (3, 3), activation=my_activation))
    model.add(ZeroPadding2D((1,1)))
    model.add(Conv2D(receptive_fields[0], (3, 3), activation=my_activation))
    model.add(MaxPooling2D(pool_size=my_pool_size, strides=my_stride))

    model.add(ZeroPadding2D((1,1)))
    model.add(Conv2D(receptive_fields[1], (3, 3), activation=my_activation))
    model.add(ZeroPadding2D((1,1)))
    model.add(Conv2D(receptive_fields[1], (3, 3), activation=my_activation))
    model.add(MaxPooling2D(pool_size=my_pool_size, strides=my_stride))

    model.add(ZeroPadding2D((1,1)))
    model.add(Conv2D(receptive_fields[2], (3, 3), activation=my_activation))
    model.add(ZeroPadding2D((1,1)))
    model.add(Conv2D(receptive_fields[2], (3, 3), activation=my_activation))
    model.add(ZeroPadding2D((1,1)))
    model.add(Conv2D(receptive_fields[2], (3, 3), activation=my_activation))
    model.add(MaxPooling2D(pool_size=my_pool_size, strides=my_stride))

    model.add(ZeroPadding2D((1,1)))
    model.add(Conv2D(receptive_fields[3], (3, 3), activation=my_activation))
    model.add(ZeroPadding2D((1,1)))
    model.add(Conv2D(receptive_fields[3], (3, 3), activation=my_activation))
    model.add(ZeroPadding2D((1,1)))
    model.add(Conv2D(receptive_fields[3], (3, 3), activation=my_activation))
    model.add(MaxPooling2D(pool_size=my_pool_size, strides=my_stride))

    model.add(ZeroPadding2D((1,1)))
    model.add(Conv2D(receptive_fields[4], (3, 3), activation=my_activation))
    model.add(ZeroPadding2D((1,1)))
    model.add(Conv2D(receptive_fields[4], (3, 3), activation=my_activation))
```

```
    model.add(ZeroPadding2D((1,1)))
    model.add(Conv2D(receptive_fields[4], (3, 3), activation=my_activation))
    model.add(MaxPooling2D(pool_size=my_pool_size, strides=my_stride))

    model.add(Flatten())
    model.add(Dense(dense_values[0], activation=my_activation))
    model.add(Dropout(dropout))
    model.add(Dense(dense_values[1], activation=my_activation))

    return model
```

So, the model is loaded and used to extract the feature from the image.

These features are used to predict the caption thanks to the weights (.h5) saved in the training part. This model is different from the one just upper and includes the natural language processing part.
However, the prediction given at the exit of the model.predict function is only an integer which correspond to a word. So, we have to found the corresponding word in the token file generated in the evaluation part.

The result is display in the terminal. I used markers for the beginning and the end. This is necessary because it helps the program to know when to stop looking for each caption.
The markers are ‘debut’ and ‘fin’ which are the french version of ‘start’ and ‘end’. By doing that, we expect no misunderstanding in the reading in the caption since there is a distinction between french and english.

<u>Problem encountered :</u>

- I have version’s problem since on my computer can only use keras and tensorflow on python 3.6 and this environment is not compatible with cv2. I had to find another way to load my picture and to read it.
- We noticed rapidly with Thomas that we have to use the same markers but the first attempt was a failure.
- Retrieve the word in the token file needs to break sometimes because it can not be found and I didn’t think about that at the beginning.
- Understanding the natural language processing part to make our program work with it.

V. Results

1. Model training : train.ipynb

As we trained our model with keras, Tensorboard seemed to be the best solution to have a nice representation of the loss function, as we studied it in class and in the assignment 4.

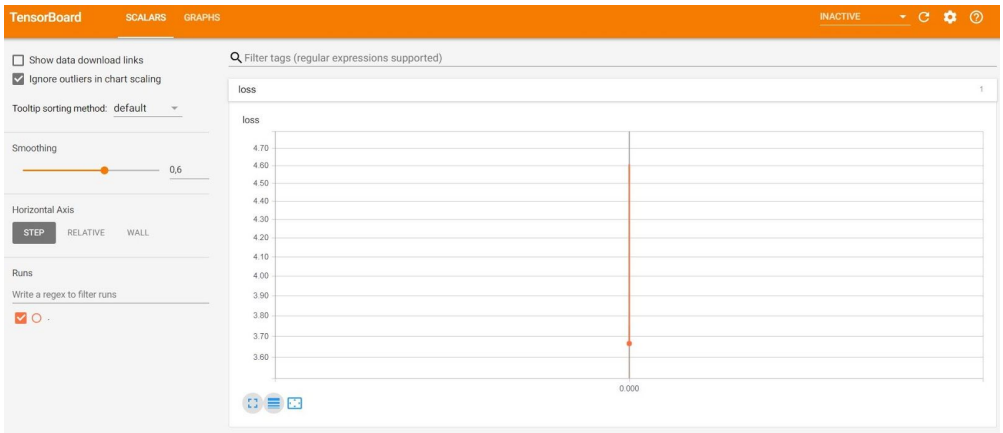
Tensorboard has still been implemented successfully even though it is not compatible natively with Google Colab, by analyzing the http traffic with the ngrock package as I have done that in the assignment 4.

```
# Ngrock Package to see results in Tensorboard WHEN USING GOOGLE COLAB

!wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
!unzip -o ngrok-stable-linux-amd64.zip

LOG_DIR = './Graph'
get_ipython().system_raw(
    'yes | tensorboard --logdir {} --host 0.0.0.0 --port 6006 &'
    .format(LOG_DIR)
)
```

Unfortunately, the curve obtained cannot be used, as we had to implement the progressive loading and run one epoch at the time. So we can only see the curve for the last epoch :



At least we can notice that the curve is decreasing.

How can you access the tensor-board via colab ?

To get the tensor board under tou just have to click on the link generated at the end of the file.

```
[ ]  # Link to use to see Tensorboard WHEN USING GOOGLE COLAB

     get_ipython().system_raw('./ngrok http 6006 &')
     !curl -s http://localhost:4040/api/tunnels | python3 -c \
         "import sys, json; print(json.load(sys.stdin)['tunnels'][0]['public_url'])"
```

https://d903668b.ngrok.io

As It is still necessary to track the value of the loss function, we printed the loss value at each epochs :

For the model trained with the our features :

```
[27]  model = define_model(vocab_size, max_length)
      epochs = 4
      steps = len(train_captions)
      for i in range(epochs):
        generator = data_generator(train_captions, train_features_our, tokenizer, max_length)
        model.fit_generator(generator, epochs=1, steps_per_epoch=steps,callbacks=[tbCallBack], verbose=1)
        model.save(PATH_DRIVE+'model_our_' + str(i) + '.h5')
```

```
Epoch 1/1
6000/6000 [==============================] - 2309s 385ms/step - loss: 4.5917
Epoch 1/1
6000/6000 [==============================] - 2319s 386ms/step - loss: 3.8486
Epoch 1/1
6000/6000 [==============================] - 2310s 385ms/step - loss: 3.5872
Epoch 1/1
6000/6000 [==============================] - 2304s 384ms/step - loss: 3.4180
```

For the model trained with the vgg features :

```
model = define_model(vocab_size, max_length)
epochs = 4
steps = len(train_captions)
for i in range(epochs):
  generator = data_generator(train_captions, train_features_vgg, tokenizer, max_length)
  model.fit_generator(generator, epochs=1, steps_per_epoch=steps,callbacks=[tbCallBack], verbose=1)
  model.save(PATH_DRIVE+'model_vgg_' + str(i) + '.h5')
```

```
Epoch 1/1
6000/6000 [==============================] - 2340s 390ms/step - loss: 4.6581
Epoch 1/1
6000/6000 [==============================] - 2367s 395ms/step - loss: 3.8782
Epoch 1/1
6000/6000 [==============================] - 2354s 392ms/step - loss: 3.6232
Epoch 1/1
3100/6000 [==============>..............] - ETA: 19:00 - loss: 3.4780
```

For both of the model, we can clearly see that the loss function is decreasing at each epoch, while staying over 3 (limit to not overfit) according to the literature.
So the model are fitting our data correctly.

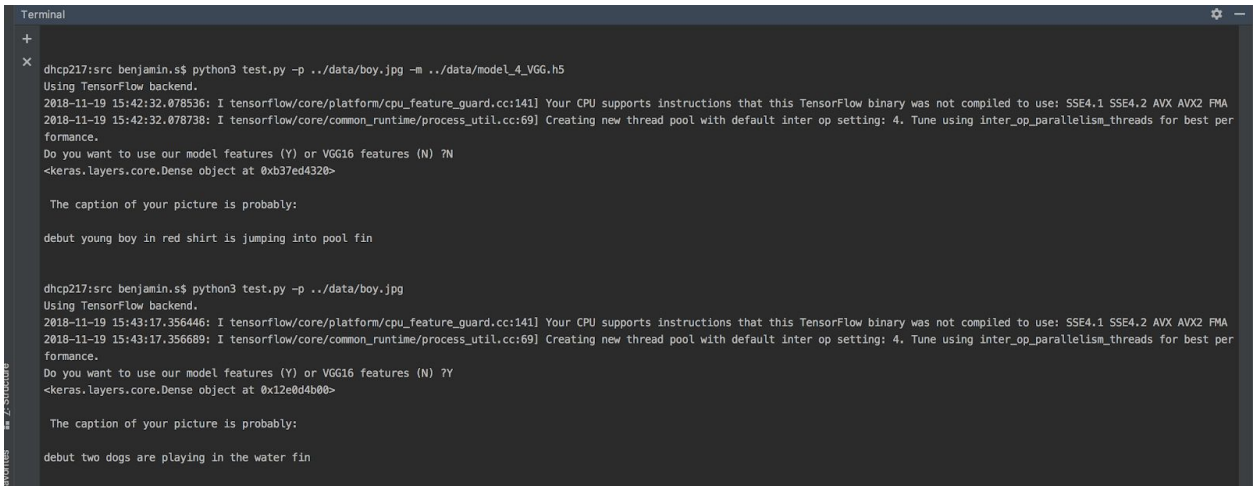
Here is the loss evolution in function of the epochs :

2. Captioning a random picture : test.py

First, you have to be aware that the Flickr8 data set is only constituted of 6000 thousand image for training and and 1000 for evaluating. That is not a big number and can explain that some caption are wrong or inaccurate.
Moreover, there is a big distinction between the human interpretation and the computer interpretation.

The test has been used on this picture :

These are the results given by the test program first with the VGG model and then with our model.



Our caption : “two dogs playing in the water”
VGG16 caption : “young boy is jumping into pool”

As you can see our caption is LESS accurate than the one of the VGG16 model and that is totally understandable since our model should be more optimized.

References

Flickr8k Dataset :

The request to use the dataset has been made through the url :
https://forms.illinois.edu/sec/1713398
Our Webtools ID is 1713398.

We are hereby required to cite M. Hodosh, P. Young and J. Hockenmaier (2013) "Framing Image Description as a Ranking Task: Data, Models and Evaluation Metrics", Journal of Artificial Intelligence Research, Volume 47, pages 853-899 http://www.jair.org/papers/paper3994.html

Publication :

- https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Vinyals_Show_and_Tell_2015_CVPR_paper.pdf (our publication)
- M. Hodosh, P. Young, and J. Hockenmaier. Framing image description as a ranking task: Data, models and evaluation metrics. JAIR, 47, 2013.
- W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. In arXiv:1409.2329, 2014.
- K. Papineni, S. Roukos, T. Ward, and W. J. Zhu. BLEU: A method for automatic evaluation of machine translation. In ACL, 2002.

Codes :

- https://www.pythonforbeginners.com/basics/string-manipulation-in-python
- https://docs.python.org/2/library/tokenize.html
- https://github.com/tensorflow/tensorflow/blob/master/tensorflow/contrib/eager/python/examples/generative_examples/image_captioning_with_attention.ipynb
- https://github.com/vsmolyakov/cv/tree/master/captions
- https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python/
- https://colab.research.google.com/notebooks/io.ipynb
- https://www.quora.com/What-is-the-VGG-neural-network
- https://stackoverflow.com/

- Tensorflow documentation : https://www.tensorflow.org/api_docs/
- Keras documentation : https://keras.io/
- OpenCV documentation : https://docs.opencv.org/2.4/index.html