Deep Network tuning for Vision and Language

How it all works

Thomas Chaton - Tran Thanh Toan

Submitted for the Semester project

Eurecom

February 2017

Contents

Chapter 1

Introduction

Lots of Deep Neural Networks have been recently introduced and shown impressive accuracy in vision and language modeling tasks, outperforming previous existing models. Deep Neural Networks build complex models from large quantities of data. As in the COCO (Common Objects in COntext) challenge: "captions challenge 2015", they have the stunning ability to describe the visual content of images and videos with words and sentences.

The goal of the project will be to use available resources on Internet as models, data, thesis papers, code in order to run a Deep Learning trained model on a standard and specialized data set of images. . We looked through all the available systems, and two models caught our attention. So we will present them in more details:

- NeuralTalk2, which is an open source software for Automatic Image Captioning with Text-Conditional Attention on GitHub developed in Lua under Torch/Cuda . The project was based on "Watch What You Just Said: Image Captioning with Text-Conditional Attention " written by Luowei Zhou1 , Chenliang Xu2 , Parker Koch3 , and Jason J. Corso3 in 24 Nov 2016

- DenseCap, which is an open source software for Automatic Dense Captioning on GitHub developed in Lua under Torch/Cuda. The project was based on "DenseCap: Fully Convolutional Localization Networks for Dense Captioning " written by Justin Johnson Andrej Karpathy Li Fei-Fei Department of Computer Science, Stanford University in 2015

The restricted data set of images will be data captured during festivals, such as the Cannes film festival, or the Avignon festival, within the research project GAFES.

Firstly, we will use some kind of bootstrapping method to finetunne Neuraltalk . To do so,

we will feed it by its own predictions. It will give us insight of its ability to be finetunned. Secondly, we will try to use DenseCap pre-trained model in order to finetunne Neuraltalk. Thirdly, we will try to use DenseCap predictions in order to evaluate NeuralTalk ones.

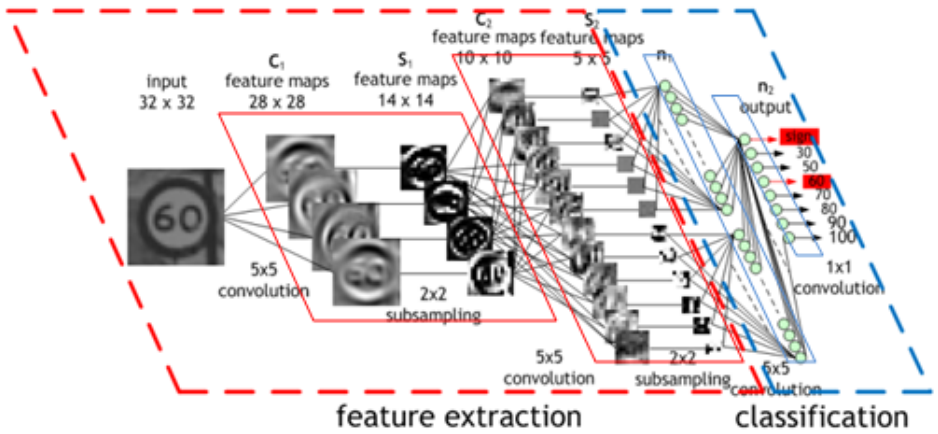
Chapter 2

Available Systems

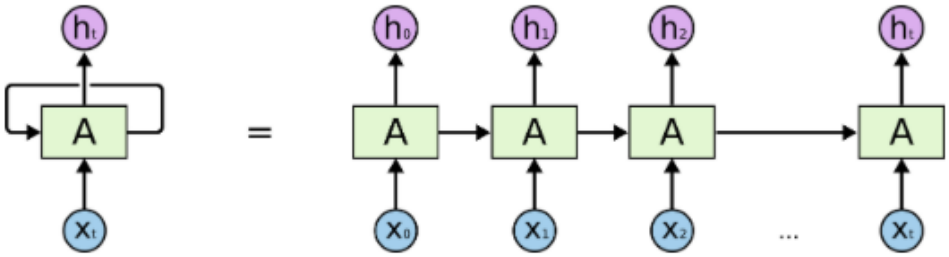
When we started to read about the available systems. We realize that there were often based on the same global components . Indeed , most of them based their computer vision related part on a CNN or a derivative of it and their natural language processing related part on a RNN or a derivative.

2.1 Common Architectures

- In machine learning, a convolutional neural network (CNN, or ConvNet) is a type of feedforward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex, whose individual neurons are arranged in such a way that they respond to overlapping regions tiling the visual field. Convolutional networks were inspired by biological processes and are variations of multilayer perceptrons designed to use minimal amounts of preprocessing.



- A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed cycle. This creates an internal state of the network which allows it to exhibit dynamic temporal behavior. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs. This makes them applicable to tasks such as unsegmented connected handwriting recognition or speech recognition.



2.2 Open Source Softwares

In most of papers we have read, benchmarking of the presented system against the others was presented. Here is an example from the paper at the origin of NeuralTalk (first version in Python) : Deep-Visual-Semantic Alignments for Generating Image Descriptions and its github: NeuralTalk written by Andrej Karpathy Li Fei-Fei Department of Computer Science, Stanford University .

	Image Annotation				Image Search			
Model	**R@1**	**R@5**	**R@10**	**Med *r***	**R@1**	**R@5**	**R@10**	**Med *r***
Flickr30K								
SDT-RNN (Socher et al. [49])	9.6	29.8	41.1	16	8.9	29.8	41.1	16
Kiros et al. [25]	14.8	39.2	50.9	10	11.8	34.0	46.3	13
Mao et al. [38]	18.4	40.2	50.9	10	12.6	31.2	41.5	16
Donahue et al. [8]	17.5	40.3	50.8	9	-	-	-	-
DeFrag (Karpathy et al. [24])	14.2	37.7	51.3	10	10.2	30.8	44.2	14
Our implementation of DeFrag [24]	19.2	44.5	58.0	6.0	12.9	35.4	47.5	10.8
Our model: DepTree edges	20.0	46.6	59.4	5.4	15.0	36.5	48.2	10.4
Our model: BRNN	**22.2**	**48.2**	**61.4**	**4.8**	**15.2**	**37.7**	**50.5**	**9.2**
Vinyals et al. [54] (more powerful CNN)	23	-	63	5	17	-	57	8
MSCOCO								
Our model: 1K test images	38.4	69.9	80.5	1.0	27.4	60.2	74.8	3.0
Our model: 5K test images	16.5	39.2	52.0	9.0	10.7	29.6	42.2	14.0

Table 1. Image-Sentence ranking experiment results. **R@K** is Recall@K (high is good). **Med *r*** is the median rank (low is good). In the results for our models, we take the top 5 validation set models, evaluate each independently on the test set and then report the average performance. The standard deviations on the recall values range from approximately 0.5 to 1.0.

	Flickr8K				Flickr30K				MSCOCO 2014					
Model	B-1	B-2	B-3	B-4	B-1	B-2	B-3	B-4	B-1	B-2	B-3	B-4	METEOR	CIDEr
Nearest Neighbor	—	—	—	—	—	—	—	—	48.0	28.1	16.6	10.0	15.7	38.3
Mao et al. [38]	58	28	23	—	55	24	20	—	—	—	—	—	—	—
Google NIC [54]	63	41	27	—	66.3	42.3	27.7	18.3	66.6	46.1	32.9	24.6	—	—
LRCN [8]	—	—	—	—	58.8	39.1	25.1	16.5	62.8	44.2	30.4	—	—	—
MS Research [12]	—	—	—	—	—	—	—	—	—	—	—	21.1	20.7	—
Chen and Zitnick [5]	—	—	—	14.1	—	—	—	12.6	—	—	—	19.0	20.4	—
Our model	57.9	38.3	24.5	16.0	57.3	36.9	24.0	15.7	62.5	45.0	32.1	23.0	19.5	66.0

Table 2. Evaluation of full image predictions on 1,000 test images. **B-n** is BLEU score that uses up to n-grams. High is good in all columns. For future comparisons, our METEOR/CIDEr Flickr8K scores are 16.7/31.8 and the Flickr30K scores are 15.3/24.7.

From those graphics , we can observe that NeuralTalk is the model the most tested against different datasets and its results are among the best . And more , it is important to note that the effective implementation in NeuralTalk2 is said to perform better than the original version. It also has a gain of speed of 100 140 times.

But we also found an other Github project that was inspired from NeuralTalk2 and from the research paper : Watch What You Just Said: Image Captioning with Text-Conditional Attention written by Luowei Zhou1 , Chenliang Xu2 , Parker Koch3 , and Jason J. Corso3 Robotics Institute, University of Michigan, Department of Computer Science, University of Rochester, Electrical and Computer Engineering, University of Michigan and its github e2e-gLSTM-sc .
In this paper, they show how they improve the LSTM of NeuralTalk2 by adding a CNN fine-tunning; which provides better textual-related image features for their proposed attention model. But they explained that, even if it performed better, it comes with a price. This system overfits and there is a several time complexity by the CNN addition.

Those models are evaluated by :

- BLEU (bilingual evaluation understudy) is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another. Quality is considered to be the correspondence between a machine's output and that of a human: "the closer a machine translation is to a professional human translation, the better it is" this is the central idea behind BLEU. BLEU was one of the rst metrics to achieve a high correlation with human judgements of quality, and remains one of the most popular automated and inexpensive metrics.

- METEOR (Metric for Evaluation of Translation with Explicit ORdering) is a metric for the evaluation of machine translation output. The metric is based on the harmonic mean of unigram precision and recall, with recall weighted higher than precision. It also has several features that are not found in other metrics, such as stemming and synonymy matching, along with the standard exact word matching. The metric was designed to x some of the problems found in the more popular BLEU metric, and also produce good correlation with human judgement at the sentence or segment level.

Chapter 3

Neuraltalk2

3.1 Architecture

NeuralTalk2 is composed of a RCNN for the visual part : a Region-based Convolution Neural Network. It is a state-of-the-art visual object detection system that combines bottom-up region proposals with rich features extraction computed by a convolutional network. And a BRNN for the natural language processing part : The principe of a Bidirectional Recurrent Neural Networks is to split the neurones of a regular RNN into two directions, one for positive time direction (forward state), and another for negative time direction (backward state) . Those two states output are not connected to the inputs of the opposite direction. By using two time directions, input information from the past and future of the current time frame can be used unlike standard RNN which requires delays for including future information.

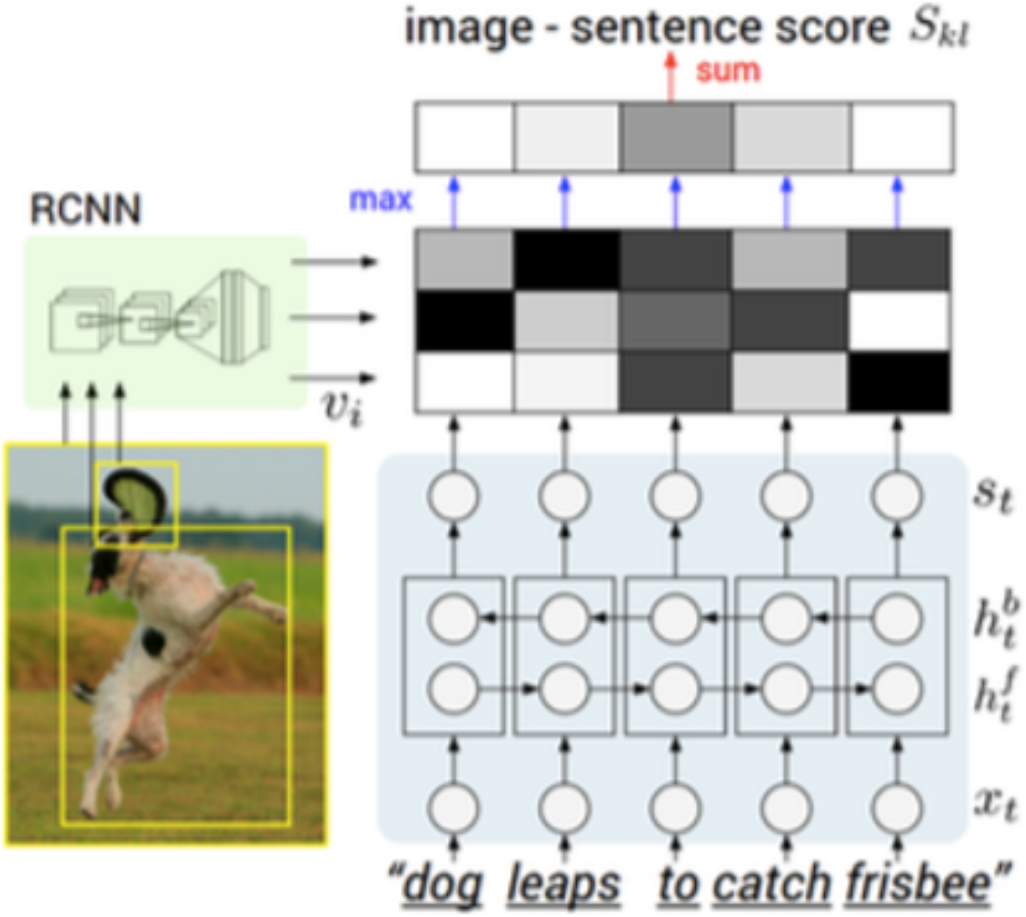


Diagram for evaluating the image − sentence score S_{kl}

Object regions are embedded with a CNN (left). Words (enriched by their context and projected into word2vect) are embedded in the same multimodal space with a BRNN (right). Pairwise similarities are computed with inner products (magnitudes shown in grayscale) and finally reduced to image-sentence score.

3.2 Setting up invironment

Here is the list of libraries we installed to make the project works:

- Python 3.4 Python 3.4 includes a range of improvements of the 3.x series, including hundreds of small improvements and bug fixes. Some parts of the neuraltalk2 required Python to run such as SimpleHTTPServer.
- Jupyter Notebook The Jupyter Notebook is a web application that allows you to create and share documents that contain live code, equations, visualizations and

explanatory text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, machine learning and much more. We used it to works on notebook in order to be more flexible during our work session.

- Tensorflow: TensorFlow is an open source software library (based on Python) for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) that flow between them. This flexible architecture lets you deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device without rewriting code. TensorFlow also includes TensorBoard, a data visualization toolkit.

3.3 Installation of NeuralTalk2 and its dependencies

NeuralTalk2 is an open source software that makes automatic captionning. It is an improvement from NeuralTalk (written in Python - Numpy). Compared to the first version, NeuralTalk2 is batched, uses Torch, can runs on both CPU and GPU and supports fine-tunning. With all those improvements, it can compile 140 faster than before. The code of NeuralTalk2 is written in Lua and requires Torch and a few libaries as folfowed:

- Torch is a scientific computing framework with wide support for machine learning algorithms that puts GPUs first. It is easy to use and efficient, thanks to an easy and fast scripting language, LuaJIT, and an underlying C/CUDA implementation. The goal of Torch is to have maximum flexibility and speed in building your scientific algorithms while making the process extremely simple. Torch comes with a large ecosystem of community-driven packages in machine learning, computer vision, signal processing, parallel processing, image, video, audio and networking among others, and builds on top of the Lua community.

Command line in Linux

```
$ curl -s https://raw.githubusercontent.com/torch/ezinstall/master/install-deps — bash
$ git clone https://github.com/torch/distro.git /torch –recursive
$ cd /torch;
$ ./install.sh # and enter "yes" at the end to modify your bashrc
$ source /.bashrc
```

- LuaRocks is the package manager for Lua modules, allows user to create and install Lua modules as self-contained packages called rocks. Using Luarocks, we also install 5 packages nn, nngraph, image, cutorch and cudann that go with it.

Command line in Linux

```
$ wget https://luarocks.org/releases/luarocks-2.4.1.tar.gz
$ tar zxpf luarocks-2.4.1.tar.gz
$ cd luarocks-2.4.1
$ ./configure; sudo make bootstrap
$ sudo luarocks install luasocket
$ luarocks install nn
$ luarocks install nngraph
$ luarocks install image
$ luarocks install cutorch
$ luarocks install cunn
```

- The Lua CJSON module provides JSON support for Lua. We're also going to need the cjson library so that we can load/save json files

Command line in Linux

```
$ make install
$ make
$ cp cjson.so $ LUA_MODULE_DIRECTORY
```

- loadcaffe to train your models user will need loadcaffe, since we are using the VGGNet.

Command line in Linux

```
$ luarocks install loadcaffe
```

- torch hdf5 allows users to read and write Torch data from and to HDF5 files. The format is fast, flexible, and supported by a wide range of other software - including MATLAB, Python, and R.

Command line in Linux

```
$ sudo apt-get install libhdf5-serial-dev hdf5-tools
$ git clone git@github.com:deepmind/torch-hdf5.git
$ cd torch-hdf5
$ luarocks make hdf5-0-0.rockspec LIBHDF5_LIBDIR="/usr/lib/x86_64-linux-gnu/"
```

- h5py package is a Pythonic interface to the HDF5 binary data format. It lets users store huge amounts of numerical data, and easily manipulate that data from NumPy.

Command line in Linux

```
$ conda install h5py # Anaconda/Miniconda
$ enpkg h5py # Canopy
```

3.4 Hands-on: How it works

Now that we have installed eveything to make NeuralTalk2 run. We need to download the pretrained checkpoint for GPU : GPU pre-trained weights and the one for CPU CPU pre-trained weights. They are both around 600MB, large because it contains the weights of the VGG-16, which are kind of heavy. They were trained on the MS COCO dataset. You can evaluate your images by running the following commands:

Command line in Linux

```
$ th eval.lua -model [/path/to/model] -image_folder [/path/to/image/directory]
num_images 1
```

The options of NeuralTalk2 are the following:

Option	Purpose	Default value
Input paths		
-model	Path to model to evaluate	
Basic options		
-batch_size	If > 0 then overrule, otherwise load from checkpoint	1
-num_images	How many images to use when periodically evaluating the loss? (-1 = all)	100
-language_eval	Evaluate language as well (1 = yes, 0 = no)? BLEU/CIDEr/ METEOR/ROUGE_L? requires coco-caption code from Github	0
-dump_images	Dump images into vis/imgs folder for vis? (1=yes,0=no)	1
-dump_json	Dump json with predictions into vis folder? (1=yes,0=no)	1
-dump_path	Write image paths along with predictions into vis json? (1=yes,0=no)	0
Sampling options		
-sample_max	1 = sample argmax words. 0 = sample from distributions	1
-beam_size	Used when sample_max = 1, indicates number of beams in beam search. Usually 2 or 3 works well. More is not better. Set this to 1 for faster runtime but a bit worse performance	2
-temperature	Temperature when sampling from distributions (i.e. when sample_max = 0). Lower = "safer" predictions	1.0
For evaluation on a folder of images		
-image_folder	If this is nonempty then will predict on the images in this folder path	
-image_root	In case the image paths have to be preprended with a root path to an image folder	
For evaluation on MSCOCO images from some split		
-input_h5	Path to the h5file containing the preprocessed dataset. empty = fetch from model checkpoint	
-input_json	Path to the json file containing additional info and vocabulary. empty = fetch from model checkpoint	
-split	If running on MSCOCO images, which split to use: val | test | train	test
-coco_json	If nonempty then use this file in DataLoaderRaw (see docs there). Used only in MSCOCO test evaluation, where we have a specific json file of only test set images	
Misc		
-backend	nn | cudnn	cudnn
-id	An id identifying this run/job. Used only if language_eval = 1 for appending to intermediate files	evalscript
-seed	Random number generator seed to use	123
-gpuid	Which gpu to use. -1 = use CPU	0

The option we are interested in are:

- model to indicate the model we are using

- num_images to specify that we want to evaluate all our images (set it to -1)
- image_folder to precise where our images are
- gpuid is set to -1 if you are running it on CPU eitherwise on GPU

In order to visualize the result, you can do:

Command line in Linux

```
$ cd vis
$ python m SimpleHTTPServer
```

It will launch a webServer you can reach on localhost:8000 in your brower and you will see your predicted captions along with their image.

3.5 Performing bootstrapping on Cannes images

3.5.1 Cleaning the dataset

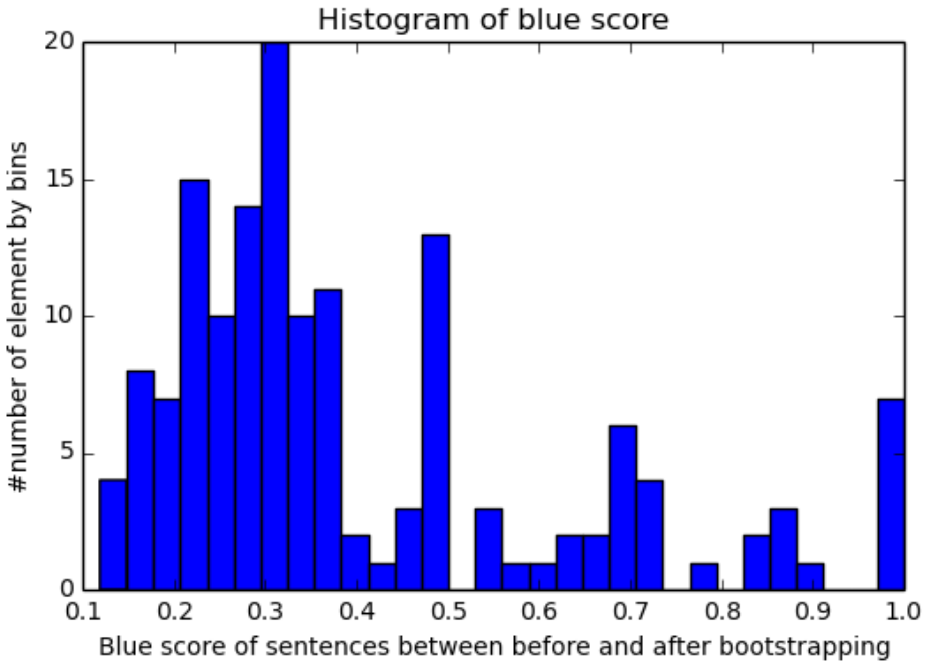
In order to clean the data, we used a method based on hexadecimal representation of an image. We used this website method for Detecting duplicate images using python. With it, we are able to reduce the dataset from 977 images to 644 . The method is the following:

Algorithm 3.1 Method to exclude duplicates based on hex strings

```
for each image in dataset do
    all_hex = []
    diff = []
    gray_img = convert to grayscale image and rescale it to a fixed size
    pixels = Get the list of pixel from gray_img
    for each row in gray_img do
        for each col in gray_img do
            left_pixel = pixels[col][row]
            right_pixel = pixels[col+1][row]
            diff.append(right_pixel superior than left_pixel)
        end for
    end for
    hex = []
    decimal = 0
    for each i,value in enumerate(diff) do
        if value then
        decimal += 2**(i   (mod 8))
        end if
        if i   (mod 8) = 7 then
            hex.append(hex(decimal))
            decimal = 0
        end if
    end for
    all_hex.append(hex)
end for
for each hex in all_hex do
    keep_only_one =
    if keep_only_one[hex] not exists then
        keep_only_one[hex] = get(img_name)
    end if
end for
```

3.5.2 Bootstrapping Method

We use the eval.lua code on the Cannes images which write down results to a json file format. Then, we split the image dataset in train and test and run the train.lua code on the train images and save the newly trained model. We finnaly evaluate the new trained model on the test image and display the bleu score of the first evaluation with the new to see if the network had learn something (measure of changes).
We plotted the distribution of those blue-score :

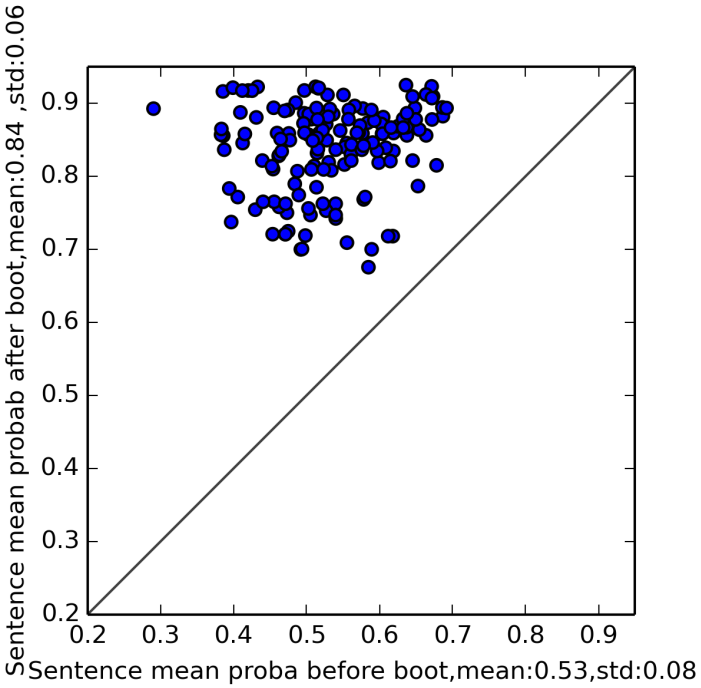


Histogram of blue-score between before and after bootstrapping on the test set

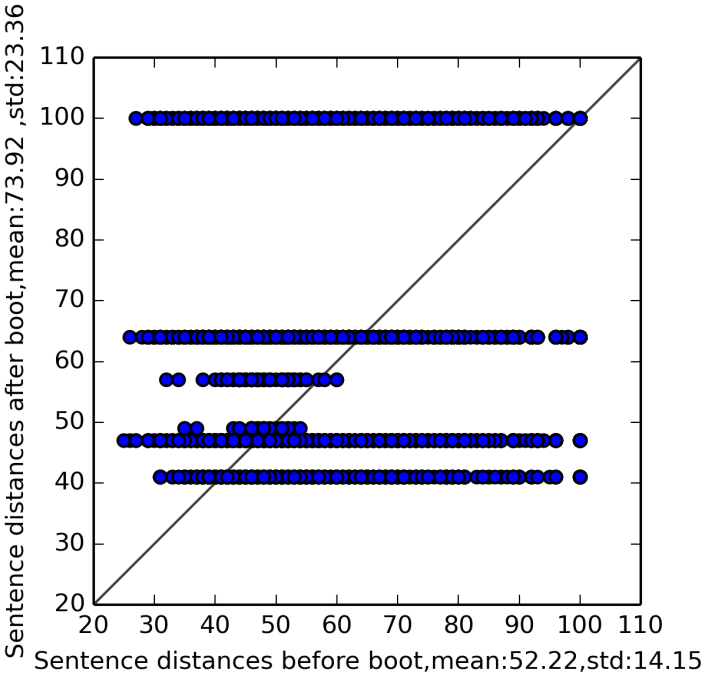
- We can see that most of predicted sentences have changed. Only few have remained the same.

- For bleu score less than 1, we can see that the mass of blue score is located between 0.3-0.4

SCORE	Reference	Predictions
1.0	a group of people standing in a room	a group of people standing in a room
0.674	a man in a suit and tie standing next to a building	a man in a suit and tie standing in a room
0.287	a man and a woman standing next to each other	a group of people standing in a room
0.162	a white and black cat sitting on a window sill	a group of people standing in a room



We can observe that sentence average probability are higher after being finetunned on the train part of a specialized dataset. It is normal, because NeuralTalk selects its dictionnary from the captions to learn. Here, we gave it only words related to this dataset. So it is like 1.5 times more confident about those images.



Graph of sentence distances before/after bootstrapping

But as we can see, even if it has a better confidence, it tends to more often predict the same things.

Chapter 4

DenseCap

4.1 Architecture

4.1.1 A end-to-end network : First in its kind

The ability to effortlessly point out and describe all aspects of an image relies on a strong semantic understanding of a visual scene and all of its elements. However, despite numerous potential applications, this ability remains a challenge for our state of the art visual recognition systems. In the last few years, significant progress have been done on both image captionning and object detection. The goal of DenseCap was to provide a end-to-end network that will unify those two tasks and give even greater results. It was trained on Visual Genome Database

4.1.2 Description of its components

- A Convolutional Neural Network is used to extract core features of the image.They used for this task the VGG-16 architecture for its state-of-the-art performance. It consists of 13 layers of 3*3 convolutions interspersed with 4 layers of 2*2 max pooling. They removed the final pooling layer, so that an image of size 3*W*H gives rise to a tensor of features of shape C*W′ ∗ H' $withC = 512, H' = \lfloor\frac{H}{16}$ and $W' = \lfloor\frac{W}{16}$

- A Fully Convolutional LocalizationLayer receives an input tensor of activations, identies spatial regions of interest and smoothly extracts a fixed-sized representation from each region

- A RNN Language Model which is an Long Short Term Memory Network. They feed him T + 2 word vectors with a x_{-1} = CNN(I) is the region code encoded with a linear layer and followed by a ReLU non-linearity, x_0 corresponds to a special START token.

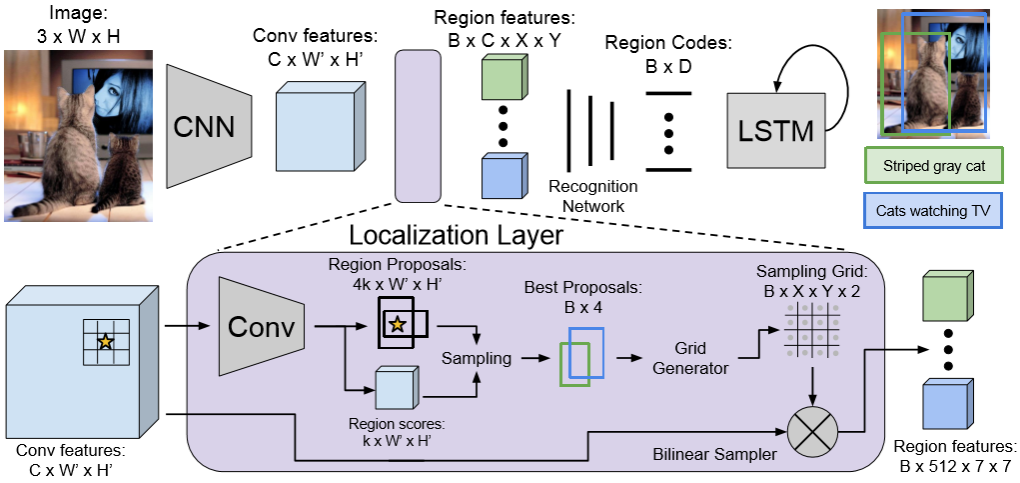
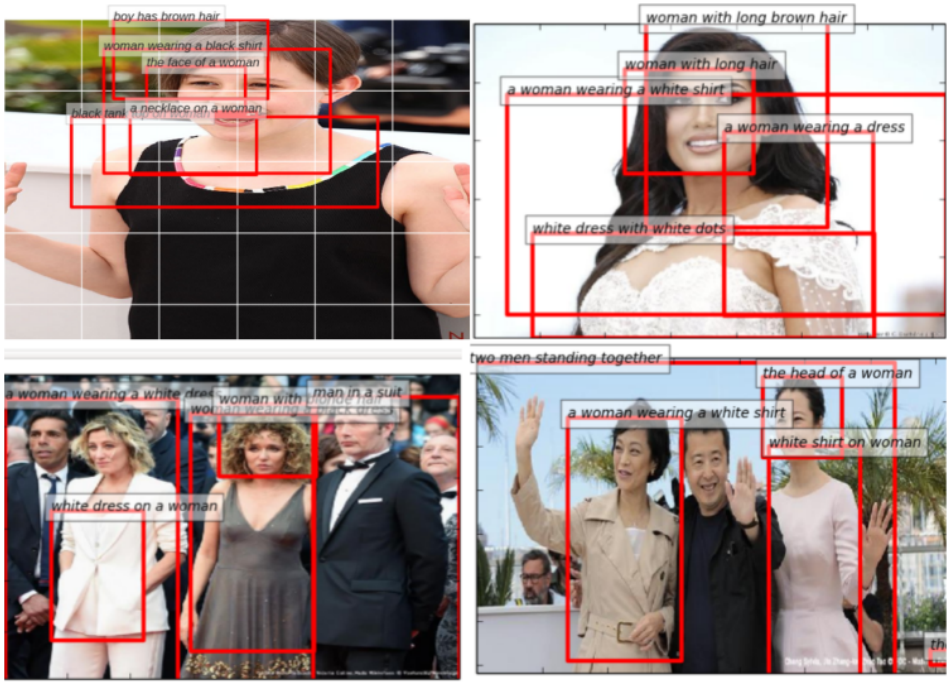


Image of the DenseCap Network Architecture



Examples of dense captionning on Cannes dataset

4.2 Setting up

DenseCap is an open source software that makes automatic dense captionning . It is an improvement of NeuralTalk2 because the neural network has an localizer that makes object detection and focus before predicting its caption. It is batched and uses Torch, can runs both on CPU and GPU and supports finetunning. However, the network was too heavy to hold on 2Go GPU . So training was not possible and evaluation took approximately 45-60min/images on CPU (it took a little bit less than 3 weeks to make run on all the dataset). In order to make it work, we needed to install those libraries (expect ones installed for NeuralTalk2):

- LuaRocks, the package manager for the Lua programming language. It allows you to install Lua modules as self-contained packages called "rocks", which also contain dependency information. LuaRocks supports both local and remote repositories, and multiple local rocks trees.

Luarocks Installation

```
$ wget https://luarocks.org/releases/luarocks-2.4.1.tar.gz
$ tar zxpf luarocks-2.4.1.tar.gz
$ cd luarocks-2.4.1
$ ./configure; sudo make bootstrap
$ sudo luarocks install luasocket
$ lua
$ Lua 5.3.3
```

- Dependencies

Command line to install / update these dependencies with luarocks

```
$ luarocks install torch
$ luarocks install nn
$ luarocks install image
$ luarocks install lua-cjson
$ luarocks install https://raw.githubusercontent.com/qassemoquab/
stnbhwd/master/stnbhwd-scm-1.rockspec
$ luarocks install https://raw.githubusercontent.com/jcjohnson/
torch-rnn/master/torch-rnn-scm-1.rockspec
```

- GPU Acceleration : We were using an NVIDIA GPU and CUDA, so we had to install torch/cutorch and torch/cunn.

Command line in Linux

```
$ luarocks install cutorch
$ luarocks install cunn
$ luarocks install cudnn
```

4.3 Use DenseCap regions to train NeuralTalk

Our assumption was we could use DenseCap predictions to train NeuralTalk. In fact, DenseCap generated 80 bounding boxes per images along with its 16 sentence long caption and its score. We thought it might be interesting, due this dense captionning, to retain the best regions and feed them to NeuralTalk2.
We needed a way to select those best regions . It occured to us that It might be possible, thanks to those 80*16 words and their scores, to find the most interesting sentences/words (the most recurrent combined by their relevance in the generation) .

4.3.1 Installation of some NLP librairies in python

- NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.
  We used principaly the lexical WordNet to compare the similarity between words

  Command line in Linux

  ```
  $ pip install nltk
  $ python
  $ import nltk
  $ nltk.download()
  ```

  Select wordnet and install it.

- FuzzyWuzzy is an open source library on github. It uses Levenshtein Distance to calculate the differences between sequences in a simple-to-use package. It helped us to find the similarity between sentences.

  Command line in Linux

  ```
  $ pip install fuzzywuzzy
  ```

  We used only fuzz.token_sort_ratio("fuzzy wuzzy was a bear", "wuzzy fuzzy was a bear") which give a score o 100 for this example.

4.3.2 Find best regions

First of all, we began to suppress all the stop words present in those generated captions due to the fact they don't bring any information but only makes sentence correct. And we came up with those two methods :

- Our first approach was to use some NLP libraries in python. We extracted the stop word from the total list of words and with the remainings, we associate each words with its scores and see how they correlated with all the others weighted by their relative scores. We will end up with a distribution of word-score and we might be able to extract the more relevant words. After that, it was possible to tell if each generated sentence was containing some of those "best words" and associate a new score to the bounding boxes by summing number of matched word score multiplied by the score of the regions .

- Our second approach was quite similar to the first one. We decided to use in addition the fact that some regions were overlapping. So in order to make the new scoring system continuous , we made a combinaison of all word pair and for each word of the pair we associated a new score which is the sum of exp[percentage of overlapping regions*α] * similarity between pair of words * DenseCap score of the sentence where the word was found .

- Our last approach was similar to the second one. But ther, we decided to use sentences instead of words. We used the same algorithm that in the second approach where word are replaced by sentence.

Algorithm 4.1 First Approach

Require: *dataset* = The 80 Generated regions along with its score and caption per images

for each *data* of an image in *dataset* **do**
  wordDict = 0
  splitData = Split each sentences of the captions of *data* into separated words and associate each one of them to their respective score (sentence score).
  cleanedData = Get rid of the stop words and the UNK Token in *splitData* {Token to mark the end to the sentence by DenseCap}
  allCombi = All the pair of combinaison of word-score in *cleanedData*
  for each *pair* in *allCombi* **do**
    $word_1$ = *pair*[0][0]
    $score_1$ = *pair*[0][1]
    $word_2$ = *pair*[1][0]
    $score_2$ = *pair*[1][1]
    wordDict[$word_1$]+=$score_1$*similarity($word_1$,$word_2$)
    wordDict[$word_2$]+=$score_2$*similarity($word_1$,$word_2$)
  end for
end for
for each *data* of an image in *dataset* **do**
  for each *obj* in *data* **do**
    captions = *obj*[0]
    score = *obj*[1]
    captionDict[*captions*] = matched_Word_Weighted_By_Scores(*captions*,*wordDict*)**score*
  end for
end for

Algorithm 4.2 Second Approach

Require: *dataset* = The 80 Generated regions along with its score and caption per images

captionDict (defaultdict in python)
for each *data* of an image in *dataset* **do**
wordDict (defaultdict in python)
$\alpha = 2$ {α is used to make a distorsion inside the exponential. It helps to seperate the best words from others. We tried several values of α from 1 to 3}
splitData = Split each sentences of the captions of *data* and associated each one of them to the respective score of the sentence and bounding box
cleanedData = Get rid of the stop words and the UNK Token in *splitData*
allCombi = All the pair of combinaison of word-score-boundingBox in *cleanedData*
for each *pair* in *allCombi* **do**
$word_1 = pair[0][0]$
$score_1 = pair[0][1]$
$boundingBox_1 = pair[0][2]$
$word_2 = pair[1][0]$
$score_2 = pair[1][1]$
$boundingBox_2 = pair[0][2]$

$wordDict[word_1] += score_1 *$
similarity($word_1$,$word_2$)*
exp [$OverLappingRegionsPercentage$(boundingBox$_1$,$boundingBox_2$)*$\alpha$]

$wordDict[word_2] += score_2 *$
similarity($word_1$,$word_2$)*
exp [$OverLappingRegionsPercentage$(boundingBox$_1$,$boundingBox_2$*$\alpha$])
end for
end for
for each *data* of an image in *dataset* **do**
for each *obj* in *data* **do**
$captions = obj[0]$
$score = obj[1]$
$captionDict[captions]$ = matched_Word_Weighted_By_Scores($captions$,$wordDict$)*$score$
end for
end for

Algorithm 4.3 Last Approach

Require: *dataset* = The 80 Generated regions along with its score and caption per images

for each *data* of an image in *dataset* **do**

sentencedDict = 0

$\alpha = 2$ {α is used to make a distorsion from the exponential higher. It helps to seperate the best words from other. We tried several values of α from 1 to 3}

splitData = Split each sentences of the captions of *data* and associated each one of them to the respective score of the sentence and bounding box

cleanedData = Get rid only of UNK Token in *splitData*

cleanedDataReformed = Join words to recreate a sentence.

allCombi = All the pair of combinaison of sentence-score-boundingBox in *cleanedDataReformed*

for each *pair* in *allCombi* **do**

$sentence_1 = pair[0][0]$

$score_1 = pair[0][1]$

$boundingBox_1 = pair[0][2]$

$sentence_2 = pair[1][0]$

$score_2 = pair[1][1]$

$boundingBox_2 = pair[0][2]$

$sentencedDict[sentence_1] += score_1 *$
similarity($sentence_1$,$sentence_2$)*
exp [$OverLappingRegionsPercentage$(boundingBox$_1$,$boundingBox_2$)*$\alpha$]

$sentencedDict[sentence_2] += score_2 *$
similarity($sentence_1$,$sentence_2$)*
exp [$OverLappingRegionsPercentage$(boundingBox$_1$,$boundingBox_2$*$\alpha$])

end for

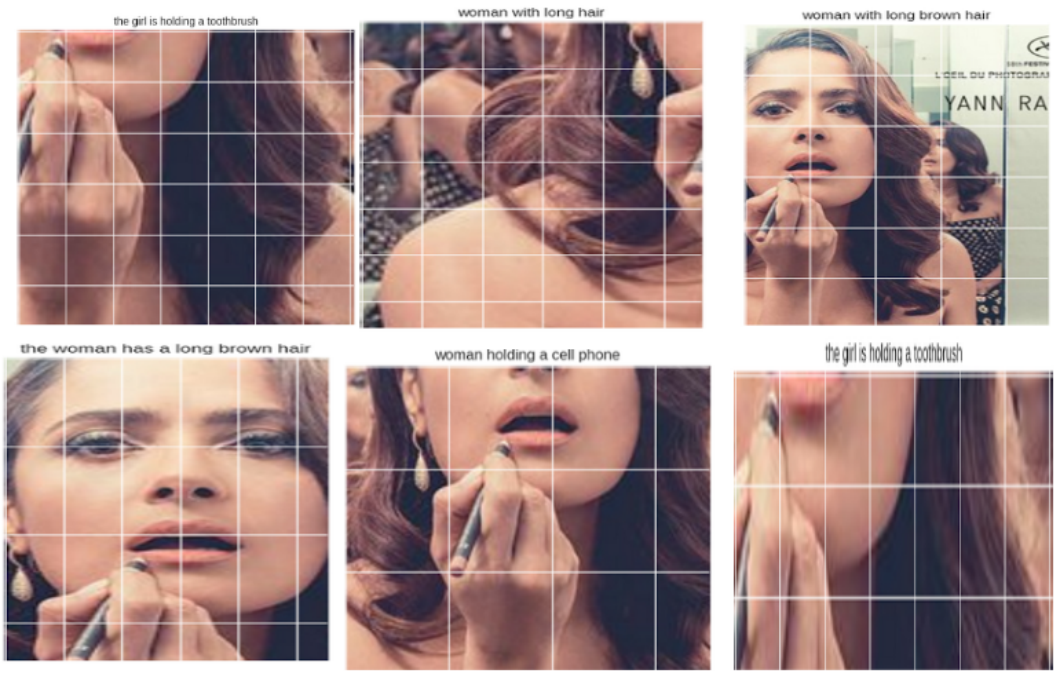
end for

4.3.3 Mini conclusion on algorithms performance

- The first approach gave medium results. By taking the 5 best regions/captions, errors came from some recurrent sentences/words in the initial training set of DenseCap as "cellphone', "blue sky with clouds",etc. And more,some word similarities weren't defined, bringing a zero contribution to the sum .

- The second approach gave better results. By taking the 5 best regions/captions, we were able to extract often the most relevant regions of the predicted regions/captions

- The first approach gave even better results because of taking into account the word context through sentence similarity. By taking the 5 best regions/captions, we were able to extract most of the times the most relevant regions of the predicted regions/captions.



Selection of 6 best captions

4.3.4 Train NeuralTalk with the selected regions

As we had the 5 best regions per image , it was now possible to train NeuralTalk2 with those part of images.

- Our first approach was to associated each regions to its caption. So we splitted them into a train and test dataset with a rate of 0.9 .and cropped all those images on the selected regions.

- The second approach was to associated the global images with the N (here 5) best selected captions. So we splitted them into a train and test dataset with a rate of 0.9.

We used some code to prepare the data to the right format for NeuralTalk2. We created a list of dictionnary that were containning in "file_path": path to the images and in "captions": a list of dictionnary with "caption" associated to the caption. After it , we used their prepro.py file and we created both an output.json and an output.hd5 files.

Command line in Linux

```
$ th train.lua -input_h5 .../output.hd5 -input_json .../output.json
```

We ran it with several parametrizations on 400-500 epoch. We tried both approaches with both –cnn_finetunning option activated or not. Unfortunately, the network wasn't able to learn anything and we got terrible results. Those terrible results might come from two different facts that :

- We had a too small dataset to train it in the two approaches (only 1000 images maximum) where it was trained on millions of images.

- But we also observed that the network was creating its vocabulary dictionnary from the set of word extracted from the sentences we gave him. From our 1000 images, it got only 100 different words instead of 10 000 from the pre-trained vocab. So we decided to modify the code in order for him to always charge the complete vocabulary dictionnary. In order to do that, I created an new opt "vocab" with the path to save the dictionnary. I added this line of code : "utils.write_json('./vis/vocab.json', vocab)". It saves the all vocab to a json format. And in the DataLoader, I modify the code to make it load this file.

We tried again to train the network with the two approaches, with all different options and by adding the complete vocab. And once again, it didn't work as expected.

4.4 Use best captions/words to evaluate NeuralTalk2 predictions

With the same assumption that the captions for local regions of images generated by Densecap is far better than global caption generated by Neuraltalk2. We come up with the idea of using the information provided by Densecap's captions to evaluate captions of Neuraltalk2.
If an image is not well discribed by the Neuraltalk2's generated caption, the information in this caption is also not close to the information of local captions generated by Densecap. Our solution to extract the information of the captions is still the same as the above.

- Removing stop words such as 'a', 'an', 'the', etc. As mentioned before, these stop words do not provide much discriminative information since they can appear in every caption. Due to the fact that there is no standard list for stopwords, we decided to use the first list appeared when searching "list of stop words" on Google. this is the link of the website http://xpo6.com/list-of-english-stop-words/. There not many differences of changing the stop words list since our auto generated captions are quite simple comparing to other kinds of documents.

- Stemming: reducing inflected (or sometimes derived) words to their word stem such as 'men' into man. This can be very important since the measure of similarities of words provided by NLP libraries in python does not work if a word is not in the list. Another region is that Densecap provided information based on local regions, which makes a tendency to detect individual instead of a group. For example, we hardly get the word 'people', 'men' or 'women' but usually get 'man' and 'woman' in Densecap but vice versa for Neuraltalk2. Stemming the words make the two dictionary using by Neuraltalk2 and Densecap closer to each other.

- Mapping densecap and neuraltalk2 result. Since Densecap and Neuraltalk2 does not follow the same format, neuraltalk2 do the captioning task by copying all the images and rename them while Densecap keeping the same name. We lost track of what global caption in Neuraltalk2 corresponds to the list of captions on Densecap. We have to do the mapping by using a hash function for mapping the images.

At this step, each image's local and global captions is summarized by a list of stemmed keyswords. We propose a measurement for the similarity of information in the caption by Neuraltalk and Densecap as follow. The similarity of a caption c by Neuraltalk2 with key

words n_1, n_2, n_l is the sum of the similarities of n_i to a key words d_j in the top k caption generated by Densecap. For the sake of running time, we choose k = 5, which give quite good result.

Since we do not have the label for the data, we can only evaluated by self evaluation. Each caption by Neuraltalk2 can be classified into 3 classes, true, partially true and false. Since the data only contains images of people and sign, we base on the gender and the number of people in the image. If a caption of people well describes gender and decides if it is one person or a group of people, it is true. If it can only get one out of 2 information, we grade it as partially true and else, it is false. For image of signs, if it recognizes that it is a sign in general, it it is true. if caption describe an images of both sign and people as sign, it is partially true. Otherwise, it is wrong.

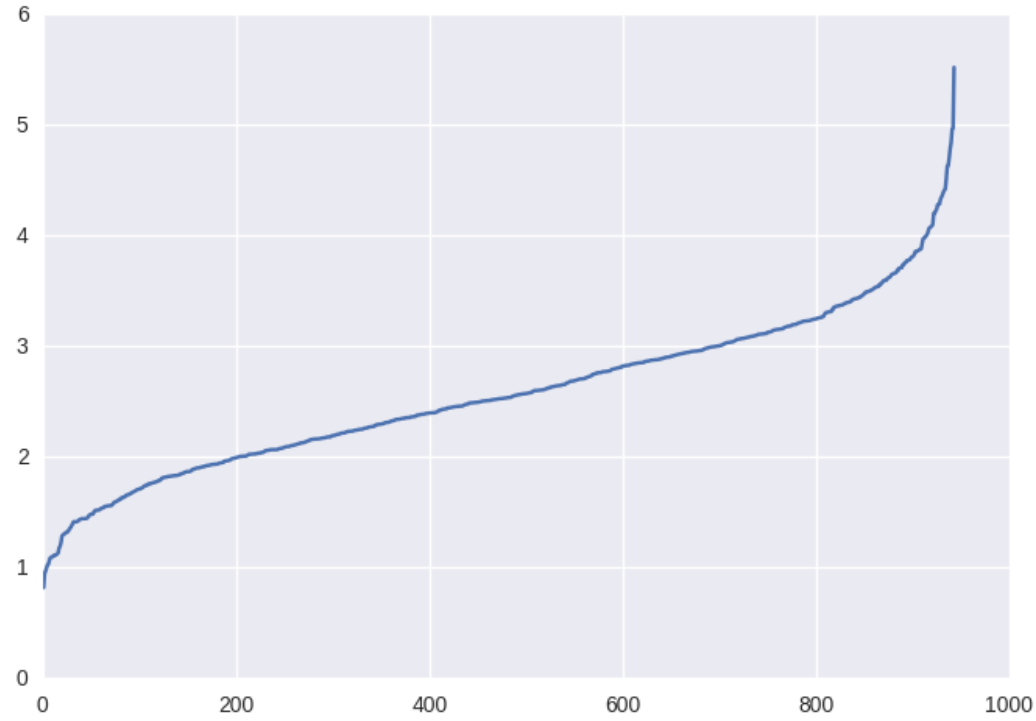
Algorithm 4.4 Proposed Evaluation of Neuraltalk2's caption and Densecap's caption

Require: *dataset* = a list of key words of caption generated by Neuraltalk and key words of 80 generated caption and corresponding score by Densecap of images in dataset.

```
for each caption_keywordsc in dataset do
    score[c] = 0
    for each key_word n in c do
        for each key_word d in key_words_densecap[0:5] do
            score[c] = similarity(n_i,d)*densecap_score
        end for
    end for
end for
```

4.4.1 Similarity by NLP library

The hypothesis for using Densecap as an evaluation for Neuraltalk2 is that higher score the more accurate the caption generated by Neuraltalk2 are.



Distribution of the scores

By looking at the distribution of scores, we can separate the curve into 3 parts. The middle part, where score ≤ 1.5 and ≥ 3.5, it is linear, while the other part are different. Therefore, we predict that the caption with score ≥ 3.5 are good, while caption with score ≤ 1.5 are the bad ones.

The result turn out to be better than the last part, the captions with score ≥ 3.5 gives us 12 true caption, 17 partially true and 0 false caption, while caption with score ≤ 1.5 gives us 4 true, 13 partially true and 34 false.

One of the interesting result is that all of the images with captions that have score ≥ 3.5 are about people, there are no signs and poster images in the well-captioned set.

4.4.2 Similarity by Word2Vec library

As suggestion from the professor, We conduct the an experiment with the same proposed evaluation except for the similarity function, which now use Word2Vec. Word2vec use the continuous bag-of-words and skip-gram architectures for computing vector representations of words, which help us to measure the similarities of any 2 words in the vocabulary. We just want to try different approaches for the similarity function.

The result turn out to be nearly the same, the set of "good" captions are a bit different.

Chapter 5

Conclusion

First, we focus our work on trying to use bootstrap method to finnetunne NeuralTalk2 on a standard and specialized data set . But as it wasn't labeled, we were able to estimate/measure only two things. How much the evaluations before and after the bootstrapping method were different and how much the network was confident about it. To overcome the issue of unlabeled data set, we have chosen to focus on an other open source software DenseCap which is able to create dense captionning. We had in mind, that it might be possible to use those informations either to train NeuralTalk2 or to evaluate it. So secondly, we used DenseCap evaluations and find a way to get the best regions (most significative) per images to get them to be feed by NeuralTalk2. But unfortunately for us, it didn't work as expected. We consider several ways to make it work without success . We arrived to the conclusion that we hadn't enought data and power (all training were done on CPU) to make it work correctly. Lastly, we focused our work on a way to evaluate NeuralTalk2 predictions from the tremendous amount of information given by DenseCap and we were able to isolate the best labeled captions by NeuralTalk2.

Bibliography

[1] Andrej Karpathy, Li Fei-Fei, Department of Computer Science, Stanford University. *Deep Visual-Semantic Alignments for Generating Image Descriptions*.

[2] Luowei Zhou, Chenliang Xu, Parker Koch, and Jason J. Corso. Robotics Institute, University of Michigan, Department of Computer Science, University of Rochester, Electrical and Computer Engineering, University of Michigan. *Watch What You Just Said: Image Captioning with Text-Conditional Attention*.

[3] Justin Johnson, Andrej Karpathy, Li Fei-Fei. Department of Computer Science, Stanford University. *DenseCap: Fully Convolutional Localization Networks for Dense Captioning*

[4] Oriol Vinyals, Alexander Toshev, Samy Bengio, Dumitru Erhan. *Show and Tell: A Neural Image Caption Generator*