Report

# Goal

In this task my goal is to provide a solution that contains a pipeline which takes an input image and generates a description of what is shown on the image.

## Business Understanding

During solving this problem we are trying to provide a program that will help with a certain business task: to optimize a website with images for a search engine. Captions that our program will generate should fill HTML tags that describe images. Our solution will help the user find exactly what he is looking for.

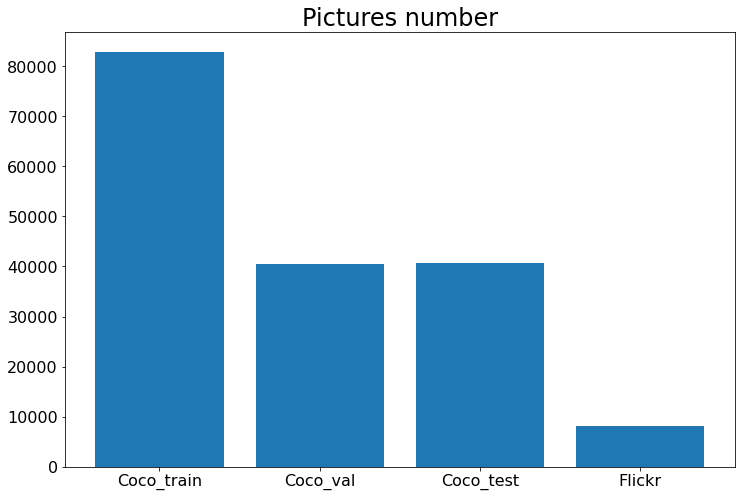
## Data

During this task I have worked with 2 datasets: Common Objects in Context (hereinafter COCO) and Flickr 8k. While COCO has a big amount of pictures, I used Flickr too to provide a diversity of datasets supported by my program.

The difference between them is that while COCO has captions stored in multiple JSONs and do not provide captions for the test data, Flickr just has one .txt file for all of the annotations.

# EDA

After data downloading it turned out that the COCO dataset is splitted not that well: it has nearly a 2:1:1 train/validation/test split which is unusual due to relatively too little data in the train set but who am I to change a dataset split ratio in such cases?



When I took a look on the annotations it would appear that all of them in a COCO dataset were too large to correctly show them in any editor available editor on computer so I had to apply commands like this to get a pretty-formatted jsons:

*$ cat original\_file.json | python3 -m json.tool > new\_file.json*

Both ‘captions\_train2014.json’ and ‘captions\_val2014.json’ have the following structure:

{

“info”: {

information about the COCO dataset

}

“images”: [

{

“license”: int (from 1 to 8)

“file\_name”: str (for example, "COCO\_train2014\_000000057870.jpg")

“coco\_url”: str

“height”: int

“width”: int

“date\_captured”: datetime (yyyy-mm-dd hh:mm:ss format)

“flickr\_url”: str

“id”: int, latest non-zero numbers in the “file\_name” (for example, 57870)

}

…

],

"licenses": [

information about different license types

],

"annotations": [

{

“image\_id”: int, equals to the “id” field in the “images” object

“id”: int, relations with the rest of json are not found

“caption”: str, description of what is on the image

}

…

]

}

Flickr captions and image names are just wrote in the .txt file line-by-line and are splitted by a comma and look like this:

image,caption

1000268201\_693b08cb0e.jpg,A child in a pink dress is climbing up a set of stairs in an entry way .

1000268201\_693b08cb0e.jpg,A girl going into a wooden building .

1000268201\_693b08cb0e.jpg,A little girl climbing into a wooden playhouse .

1000268201\_693b08cb0e.jpg,A little girl climbing the stairs to her playhouse .

1000268201\_693b08cb0e.jpg,A little girl in a pink dress going into a wooden cabin .

1001773457\_577c3a7d70.jpg,A black dog and a spotted dog are fighting

1001773457\_577c3a7d70.jpg,A black dog and a tri-colored dog playing with each other on the road .

1001773457\_577c3a7d70.jpg,A black dog and a white dog with brown spots are staring at each other in the street .

1001773457\_577c3a7d70.jpg,Two dogs of different breeds looking at each other on the road .

1001773457\_577c3a7d70.jpg,Two dogs on pavement moving toward each other .

1002674143\_1b742ab4b8.jpg,A little girl covered in paint sits in front of a painted rainbow with her hands in a bowl .

Both COCO and Flickr dataset have exactly 5 captions for each picture.

## Data Preparation

During the data preparation process the vocabulary is created. It consists of all the words that are present in the whole dataset.

Then we tokenize each word in the vocabulary and pass the model a tensor with a sequence of integers as a label.

Model processes it and also returns a sequence of integers which we convert into the words and write to the output file.

## COCO API

One of the datasets (COCO) has his own convenient API to work with.

It can be found [here](https://github.com/cocodataset/cocoapi/tree/master/PythonAPI) and consists of many useful attributes and methods, some of them I will further use in my code:

* anns (get all captions from the annotations file)
* loadImgs (get a picture by its id)
* loadAnns (get a caption by its id)
* getAnnIds (get all captions ids for a specific image)

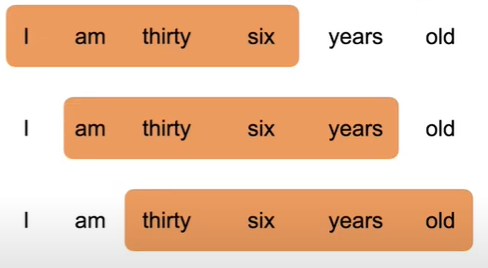
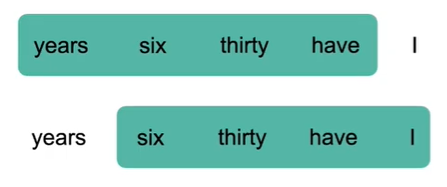
# Metrics

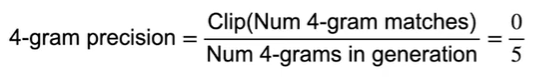
It is not that easy to evaluate the outputs of a text generating model but there are some metrics that can give a numerical value to the sentence.

In my work I used two of them: system-level BLEU (BiLingual Evaluation Understudy) and ROUGE (Recall-Oriented Understudy for Gisting Evaluation).

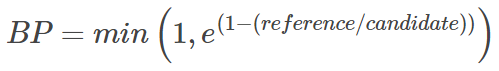
What BLEU does?

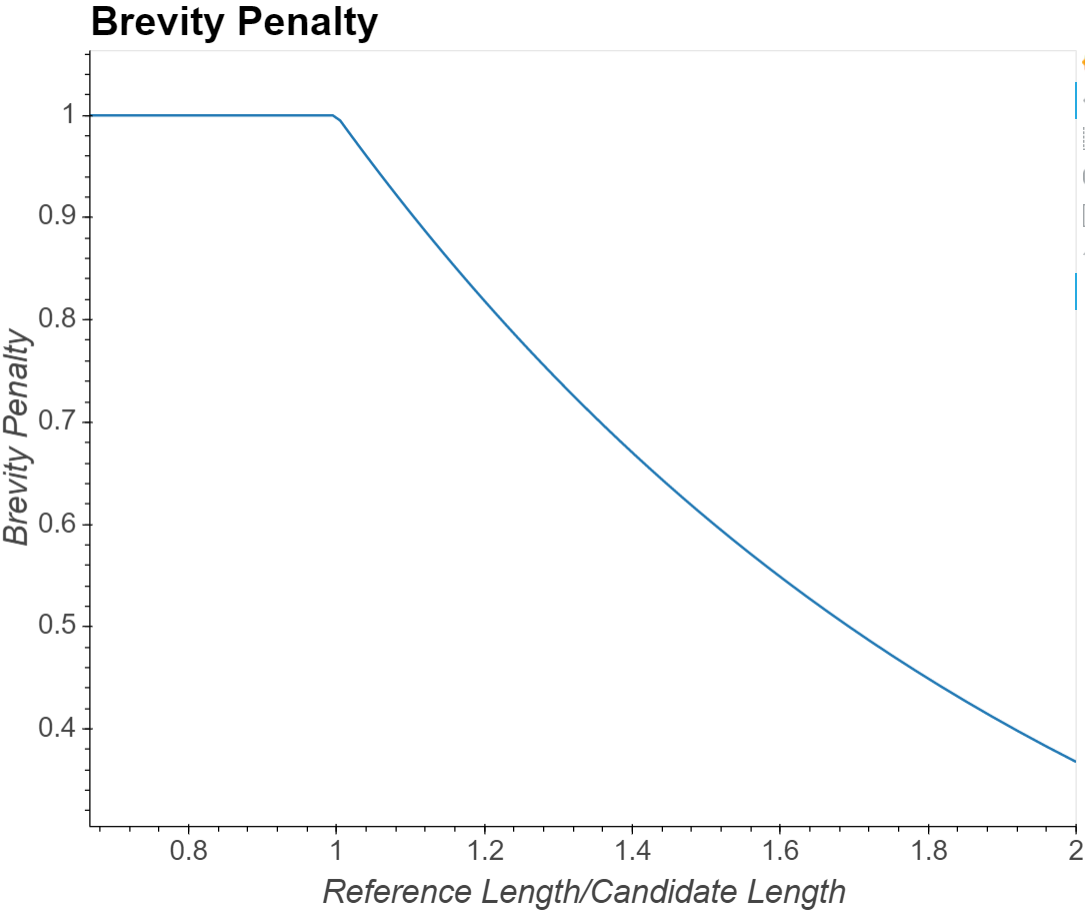
It takes all N-grams (fancy way to say ‘chunk of N words arranged in a row’) in the source and output and counts the precision just dividing the number of equal n-grams by the total number of N-grams, like here:





Then the N-gram is multiplied by a Brevity Penalty which penalizes too short output samples (that would have precision equals to 1) and calculates by the following formula:



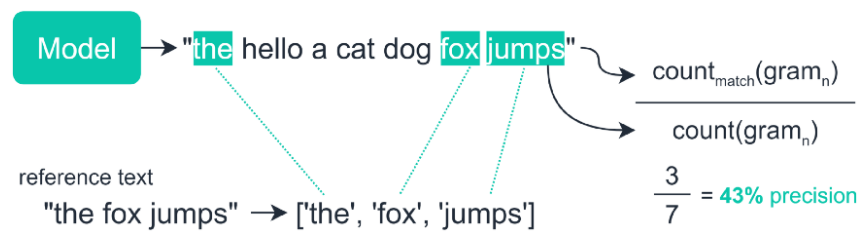


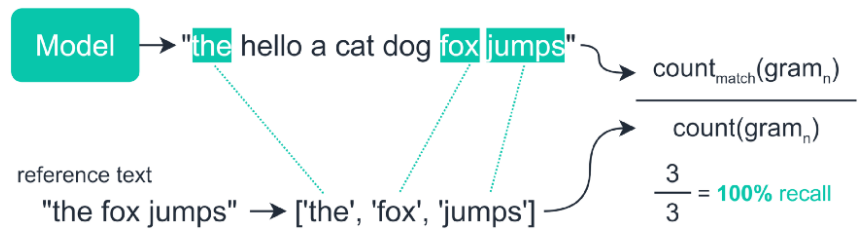
To be more precise, in this work the Corpus BLEU metric with smoothing was used. The difference between numbered BLEU and Corpus BLEU is that the last one takes BLEU-1, BLEU-2, BLEU-3 and BLEU-4 scores (which count precision for 1-, 2-, 3- and 4-grams respectively) and returns us the average.

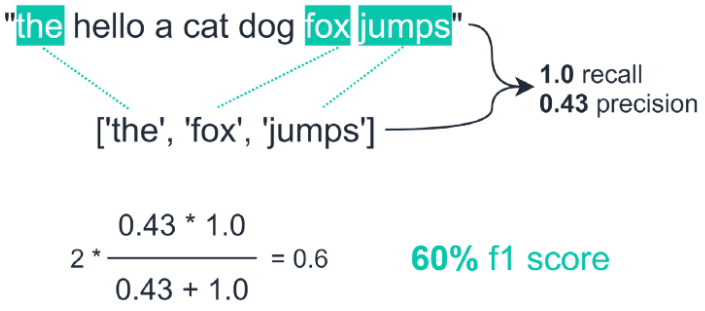
As for the ROUGE score, I used ROUGE-1 and ROUGE-L with a stemmer which is used to strip word suffixes to improve matching.

ROUGE-N score calculates precision, recall and F1 score for N-grams in the given source and output.

For example, ROUGE-1 gets scores in such a way:







# Modeling

Due to the task statement and different scientific works, for the image captioning tasks Encoder-Decoder neural network architecture is widely used.

The encoder, which consists of the Convolutional Neural Network (CNN), takes the image as the input and produces a vector of features.

We process it using an embedding layer which, in fact, gives a large ‘characteristic’ of each word by some number of parameters (hidden size).

Then in the decoder we use the Recurrent Neural Network (RNN) to process the source caption and generate the output one.

As to our particular case, I used a pretrained ResNet-101 model in the encoder.

Firstly I used a standard PyTorch Embedding learnable layer but then replaced it with pretrained GloVe embeddings. It consists of 300d vectors so the hidden size was set to 300.

Simple RNN would not be much effective without LSTM so I used the standard PyTorch module.

The RNN produces a vector of probabilities for each word with the length of the vocabulary size.

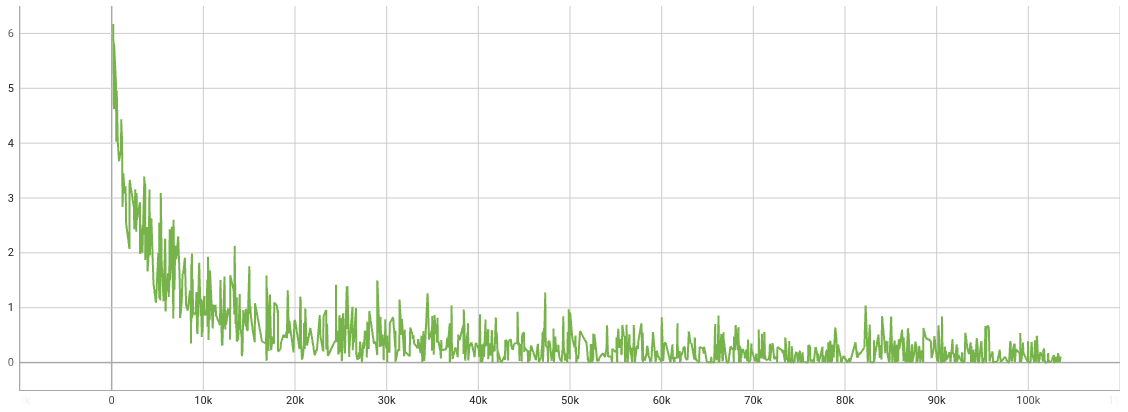
Then we take each word, get the maximum probability and add to the list of tokens which will be processed into words further.

# Output results

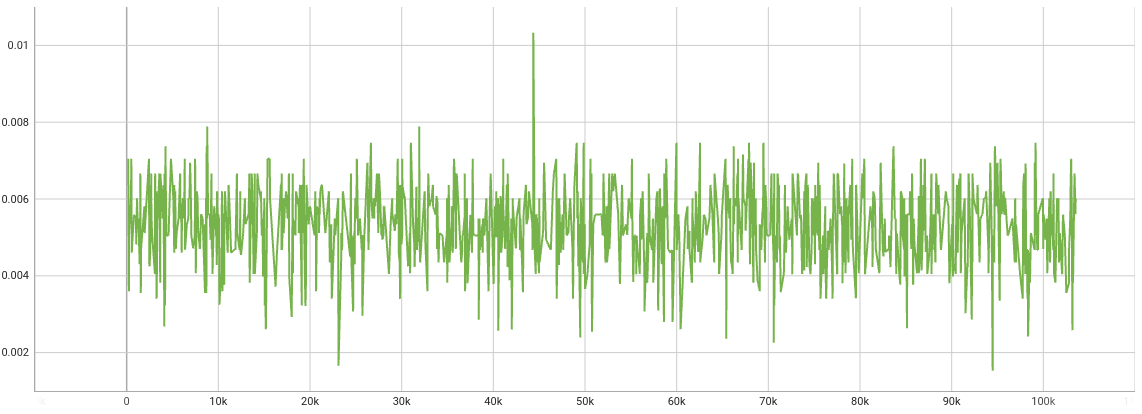
## Loss and metric plots

Different model runs did not provide me much different results so loss and metric plots was nearly the same and looked like this:

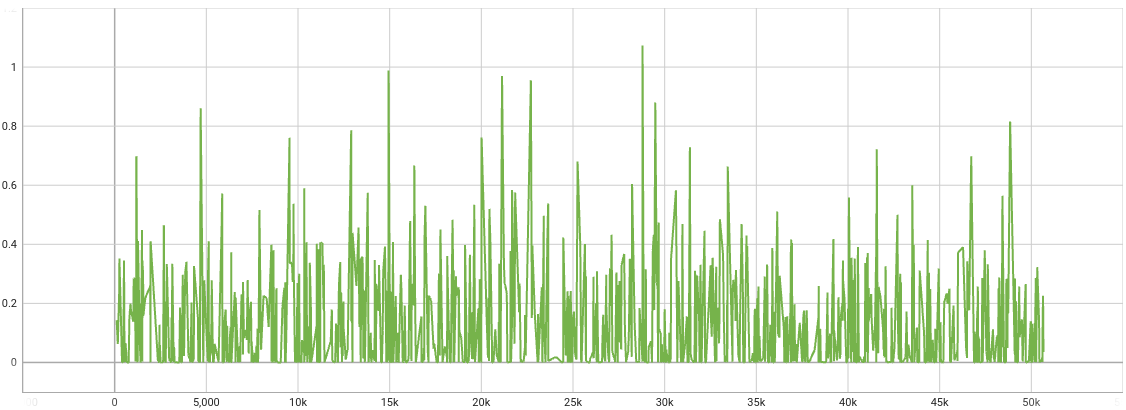
**Training loss, COCO:**



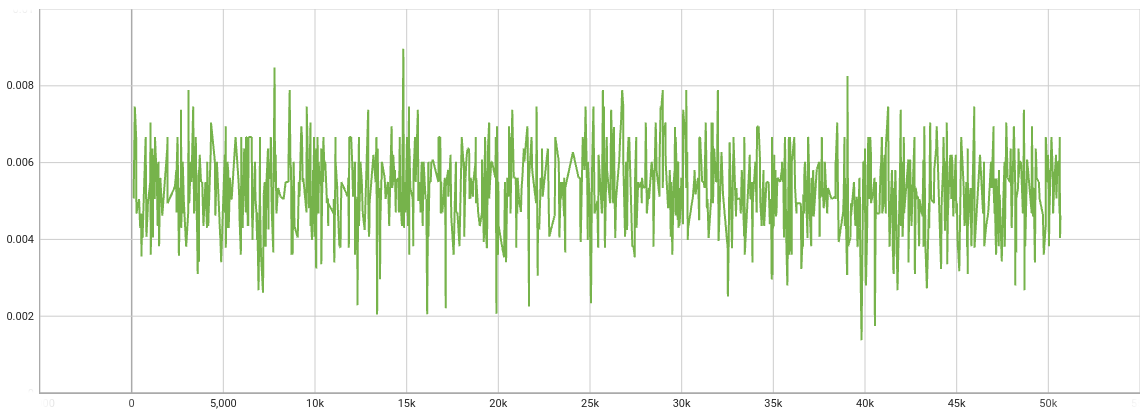
**BLEU training score, COCO:**



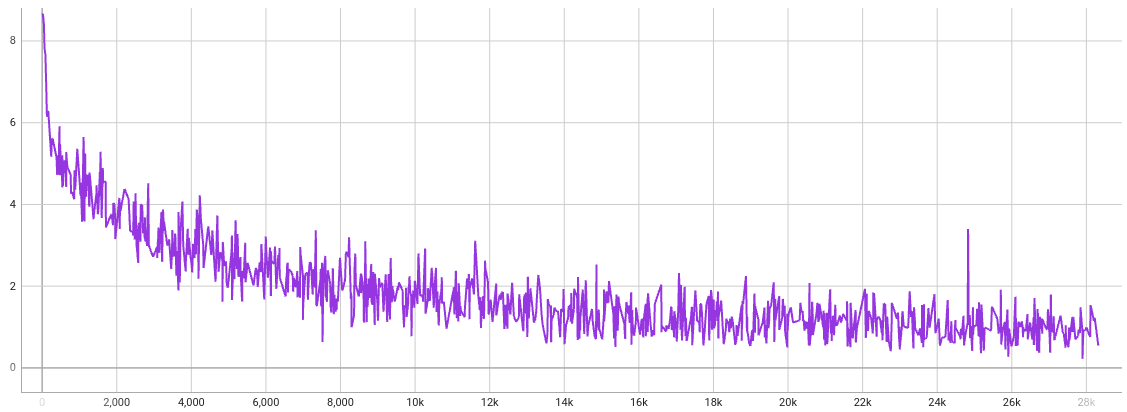
**Validation loss, COCO:**



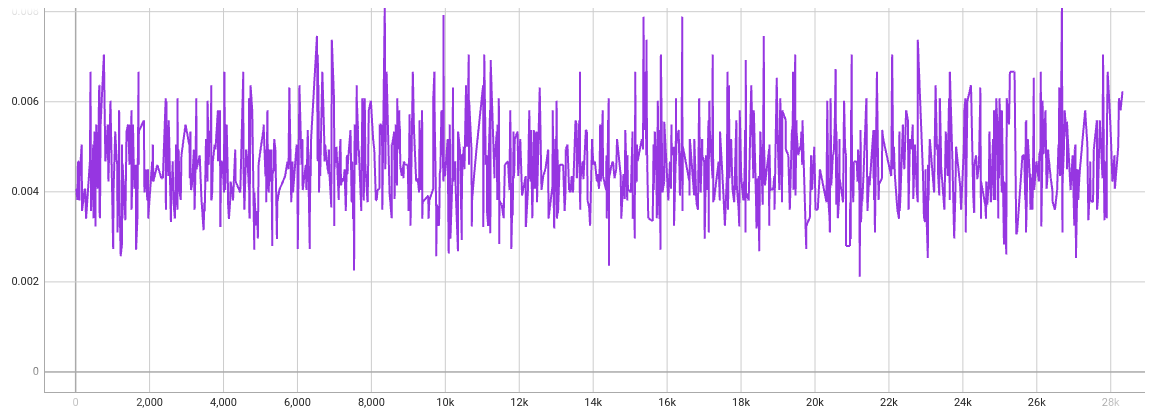
**BLEU validation score, COCO:**



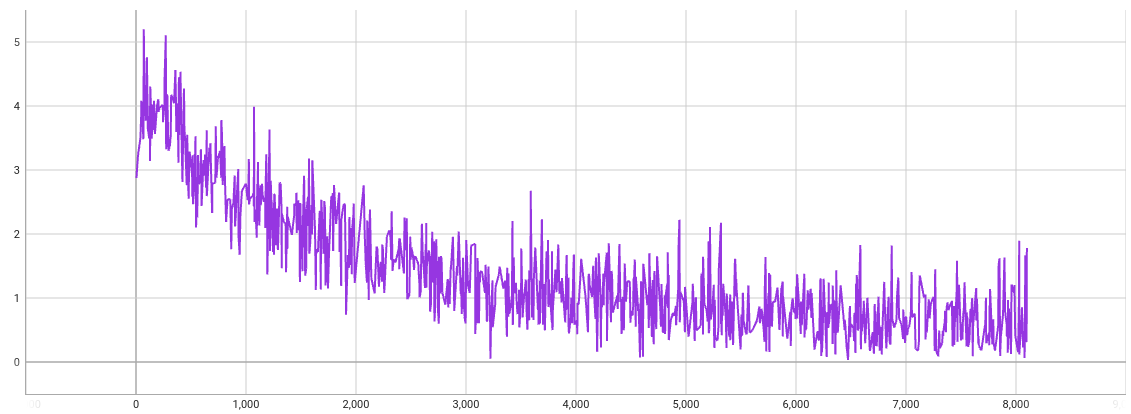
**Training loss, Flickr:**

****

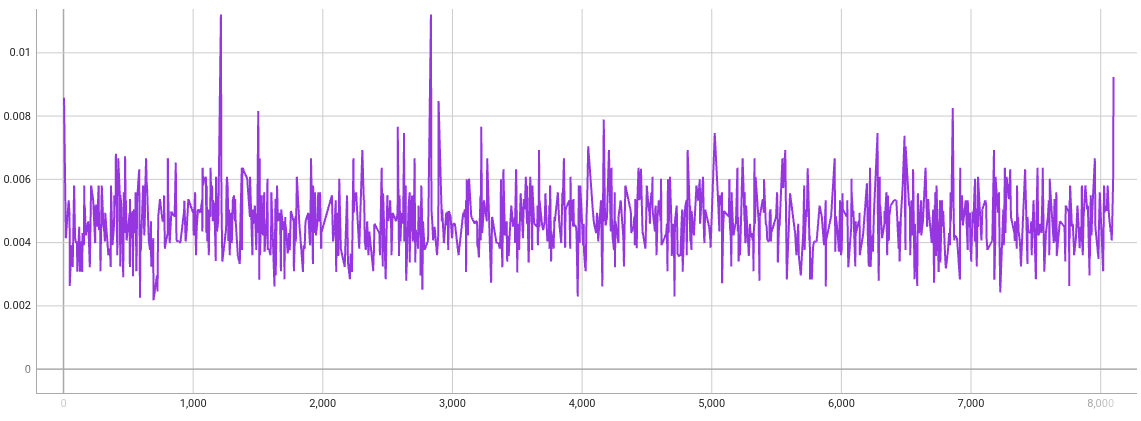
**BLEU training score, Flickr:**

****

**Validation loss, Flickr:**

****

**BLEU validation score, Flickr:**

****

Results obtained from different datasets do not differ much as from metrics, as from manual reading so I put results from the both datasets into the same tables.

## Training



| **Image** | **True caption** | **Predicted caption** |
| --- | --- | --- |
|  | A cat is sitting on top of a small stack of papers | a cat is sitting on top of a small stack of papers |
|  | this is a lady laying on a bed with a christmas stocking | this is a lady laying on a bed with a christmas identical |
|  | Purple and gold bed and flowers against a red wall | purple and gold bed and flowers against a red wall |
|  | A group of people standing near a table with wine | a group of people standing near a table with wine |
|  | A mother sheep in a field with two baby sheep | a mother sheep in a field with two baby sheep |
|  | a dog plays in the snow | a dog plays in the snow covered |
|  | a large crowd of people are sitting watching a dog jump high up to catch a frisbee with a man standing under the dog looking up at it | a large crowd of people are sitting watching a dog jump high up to catch a frisbee with a man standing under the dog looking up at it over |
|  | a surfer surfs | a surfer surfs course |

## Inference



| **Image** | **Caption** |
| --- | --- |
|  | 'urge', 'playroom', 'breasts', 'breasts', 'steamboat', 'flea', 'upper', 'upper', 'upper', 'upper', 'upper', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while', 'while' |
|  | 'flurry', 'ref', 'lace', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african', 'african' |
|  | 'shoeless', 'tomatos', 'la', 'rippling', 'lockers', 'playroom', 'breasts', 'purchasing', 'lockers', 'playroom', 'breasts', 'target', 'lockers', 'playroom', 'breasts', 'terrier', 'unpaved', 'armor', 'vacation', 'velvet', 'dice', 'breaded', 'toes', 'footed', 'isolated', 'skin', 'looked', 'cowboys', 'atm', 'teal', 'gestures', 'adolescent', 'paying', 'broad', 'handlers', 'cleans', 'starring', 'dice', 'springs', 'international', 'find' |
|  | 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human' |
|  | 'ignoring', 'ignoring', 'unpainted', 'cylinder', 'spiked', 'unpainted', 'brilliant', 'spiked', 'unpainted', 'brilliant', 'spiked', 'brilliant', 'cylinder', 'armor', 'brilliant', 'brilliant', 'armor', 'brilliant', 'armor', 'brilliant', 'armor', 'brilliant', 'armor', 'brilliant', 'armor', 'brilliant', 'armor', 'armor', 'brilliant', 'armor', 'armor', 'brilliant', 'armor', 'armor', 'armor', 'brilliant', 'armor', 'armor', 'armor', 'brilliant', 'armor' |
|  | 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human', 'human' |
|  | 'urge', 'brilliant', 'surfboarder', 'obscured', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football', 'football' |
|  | 'urge', 'playroom', 'breasts', 'breasts', 'hatchback', 'mannequin', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized', 'vandalized' |

As can be seen from the results above, the model produces extremely good results on the training data but during the inference we have just one repeatable word or bunch of different words that are out of the context of the image.

This may be caused by not only overfitting but also using different approaches in the training and inference processes. Since they differ, we can apply a scheduled sampling.

It passes to the model not only its previously generated word but ground-truth too, with some probability (basically 50/50).

Also the beam search realization could also help us get better results on the inference.

# Conclusions

To finalize this work, I have been able to create a program that can process different dataset types using encoder-decoder model with LSTM and attention and generate captions to images.

The main issue after all development, I would call the model overfitting. While making predictions that are almost identical to the source ones, the model totally crashes on the inference, returning if not just a single word repeated multiple times but some nonsense.

### Further Improvements

As for the next steps for improvement we can consider solving the model overfitting problem to get more intelligible outputs.

The key to improving could also be a scheduled sampling and beam search.

Also we can tune model hyperparameters with applying learning rate scheduler, try the bigger dataset or use pre-defined vocabulary.