

# COMP5623 Coursework on Image Caption Generation

## ***Deadline***

10am 28<sup>th</sup> April 2020.

## ***Submission***

Submit work electronically via Minerva. This coursework is worth 20% of the credit on the module.

## ***Time allocation***

An indicative duration for this coursework is a total of around 17 hours.


## ***Motivation***

Through this coursework, you will:

- Understand the principles of text pre-processing and text embeddings
- Gain experience working with an image to text model
- Compare the performance and usage of RNNs versus LSTMs as sequence generators

## ***Setup and resources***

Please implement this coursework using Python and PyTorch. This coursework will use the Flickr8k image caption dataset [1].

Sample image and ground truth captions	
	<p>A child in a pink dress is climbing up a set of stairs in an entry way</p> <p>A girl going into a wooden building</p> <p>A little girl climbing into a wooden playhouse</p> <p>A little girl climbing the stairs to her playhouse</p> <p>A little girl in a pink dress going into a wooden cabin</p>

You are provided with a Jupyter Notebook file “COMP5623 CW2 Starter” with starter code.

## **Dataset download**

The zip files containing the image data and text data can be downloaded here:

<https://github.com/jbrownlee/Datasets/releases/tag/Flickr8k>

This dataset is larger than the ones we have used previously, so if you would like to work on Colab, it is recommended to download the zip files, unzip them, and then upload all the files to your Google Drive. This initial upload may take a while, but from then on you will only need to mount your Drive every time you start a new Colab session and the data will be immediately accessible. Mounting only takes a few seconds.

## Text preparation

The first step is to build a vocabulary. The vocabulary consists of all the possible words which can be used - both as input into the model, and as an output prediction. To build the vocabulary:

- a. Parse the `lines` variable in the starter code, splitting the image ID from the caption text.
- b. Split the caption text into words and trim any trailing whitespaces.
- c. Convert all words into lowercase by calling `word.lower()`.
- d. Remove any punctuation (periods, commas, etc.).
- e. Since the vocabulary size affects the embedding layer dimensions, it is better to remove the very infrequently used words from the vocabulary. Remove any words which appear 3 times or less; this should leave you with a vocabulary size of roughly 3440 (plus or minus some is fine).
- f. Add all the remaining words to an instance of the `Vocabulary()` object provided. Note that `vocab.add_word()` deals with duplicates already.

## Complete the encoder network

Carefully read through the `__init__()` function of the `EncoderCNN` and complete the class by writing the `forward()` function. Remember to put the part involving the `ResNet` in a `with torch.no_grad():` block.

## BLEU for evaluation

One common way of comparing a generated text to a reference text is using BLEU, or the Bilingual Evaluation Understudy. This article gives a good intuition to how the BLEU score is computed: <https://machinelearningmastery.com/calculate-bleu-score-for-text-python/>

The Python `nltk` package for natural language processing provides a function `sentence_bleu()` which computes this score given one or more reference texts and a hypothesis. For the following two sections, generate this score by using all five reference captions compared against each generated caption. You can import it on Colab like this:

```
from nltk.translate.bleu_score import sentence_bleu
```

If working on your local machine you may need to install the `nltk` package using `pip` or `conda` first.

## Train using an RNN for decoder

Add an RNN layer to the decoder where indicated, using the API as reference:

<https://pytorch.org/docs/stable/nn.html#rnn>

Keep all the default parameters except for `batch_first`, which you may set to `True`.

Add code to the training loop provided which generates a sample caption from two test set images before any training, and after each epoch. Display both the sample images, the reference captions, generated caption, and its BLEU score after every epoch. Train for 5 epochs.

### Displaying an image

To generate a prediction from your trained model, you will need to first pass the image tensor into the encoder to get the image features. Remember to first process the image with the `data_transform`. Because the `EncoderCNN` has a batch norm layer, and you are feeding in single images (plus we do not want to shift the mean and std deviation), run this line before passing in your images:

```
encoder.eval()
```

Pass the features from the encoder into the decoder `sample()` function.

The decoder output is a one-dimensional vector of word IDs, and length `max_seq_length`. This means that if the caption is shorter than `max_seq_length` you can ignore any padding after the `<end>` token.

Finally, use the vocab to convert the IDs into words and string the caption together.

Train for exactly five epochs. No need to use a validation set.

Using the fully trained model, generate captions for at least five additional test images and display along with BLEU scores and reference captions.

## Train using an LSTM for decoder

Now replace the RNN layer with an LSTM (leave the RNN layer commented out):

<https://pytorch.org/docs/stable/nn.html#lstm>

Again, keep all the default parameters except for `batch_first`, which you may set to `True`.

As with the RNN, display the same two sample images, generated captions and BLEU scores before training and after each epoch. Train for five epochs. Using the fully trained model, generate captions for at least five additional test images and display along with BLEU scores and reference captions.

## **Deliverable and Assessment**

You should submit a report of **not more than 8 pages** (excluding Python code):

1. A common practice in natural language processing is to lemmatize words before creating tokens. While we did not do this for our vocabulary, take a look at *Flickr8k.lemma.token.txt* and briefly explain what the advantages/disadvantages might be of using lemmatized vs regular tokens. (max. 5 marks)
2. Present the sample images and generated caption for each epoch of training for both the RNN and LSTM version of the decoder, including the BLEU scores. (max. 30 marks)
3. Compare training using an RNN vs LSTM for the decoder network (loss, BLEU scores over test set, quality of generated captions, performance on long captions vs. short captions, etc.). (max. 30 marks)
4. Among the text annotations files downloaded with the Flickr8k dataset are two files we did not use: *ExpertAnnotations.txt* and *CrowdFlowerAnnotations.txt*. Read the *readme.txt* to understand their contents, then consider and discuss how these might be incorporated into evaluating your models. (max. 10 marks)
5. Python code for all your work (max. 25 marks)

Your submission should include one PDF file for your report, and one Python or .pynb file, zipped together into a .zip file.

## **References**

[1] 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>)