

COMP5623 Coursework on Image Caption Generation

Submission

Submit work electronically via Minerva. This coursework is worth 25% of the credit on the module.

Motivation


Through this coursework, you will:

- Understand the principles of text pre-processing and vocabulary building
- Gain experience working with an image to text model
- Use and compare two different text similarity metrics for evaluating an image to text model

Setup and resources

Please implement this coursework using Python and PyTorch. A GPU is not required to complete the coursework, but you may use one if you choose.

You are provided with starter code for both questions which will be described below, as well as a report template. This coursework will use the Flickr8k image caption dataset [1] for image caption generation. The dataset consists of 8000 images, each of which has five different descriptions of the salient entities and activities.

| Sample image and corresponding 5 reference 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> |

Dataset download

Please download the following:

1. *Flickr8k_Dataset.zip* from <https://github.com/jbrownlee/Datasets/releases/tag/Flickr8k>
2. *Flickr8k_train.token.txt* and *Flickr8k_test.token.txt* from <https://github.com/ysbecca/flickr8k-custom/tree/main/captions>

We recommend this method of downloading the data using a Colab or Jupyter notebook, or from a command line on your machine:

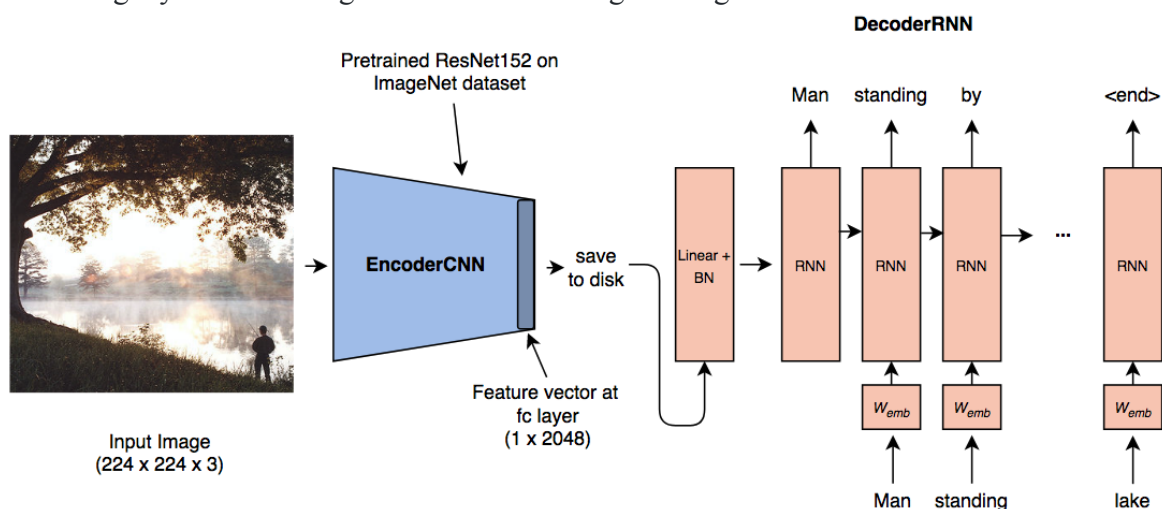
```
! wget https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flickr8k_Dataset.zip
```

```
! unzip Flickr8k_Dataset.zip
```

```
! git clone https://github.com/ysbecca/flickr8k-custom
```

QUESTION 1

The basic principle of our image-to-text model is as pictured in the diagram below, where an Encoder network encodes the input image as a feature vector by providing the outputs of the last fully-connected layer of a pre-trained CNN. This pretrained network has been trained on the complete ImageNet dataset and is thus able to recognise common objects. These features are then fed into a Decoder network along with the reference captions. As the image feature dimensions are large and sparse, the Decoder network includes a linear layer which downsizes them, followed by a batch normalisation layer to speed up training. Those resulting features, as well as the reference text captions, are passed into a recurrent network (we will use an RNN). The reference captions used to compute loss are represented as numerical vectors via an embedding layer whose weights are learned during training.



The Encoder-Decoder network could be coupled and trained end-to-end, without saving features to disk; however, this requires iterating through the entire image training set during training. We can make the training more efficient by decoupling the networks. We will first extract the feature representations of the images from the Encoder and save them (*extract_features.py*). During training of the Decoder (*decoder.py*), we only need to iterate over the image feature data and the reference captions.

1.1 Text preparation

Open the token file *Flickr8k_train.token.txt* to see the reference caption format. Then look at *extract_features.py* and locate where the token file is read into `lines`. This function is provided for you. The first step is to parse the reference captions from the training set, which is in *utils.py* in function `parse_lines()`. To fill in the function, parse the `lines` variable in the starter code, splitting the image ID from the caption text. Save the image IDs in a list `image_ids`. For each caption:

1. Clean text by removing any punctuation (periods, commas, etc.), numbers, and symbols.
2. Convert all remaining words into lowercase.

| |
|--|
| Printing <code>cleaned_captions[:2]</code> should give the following: |
| <code>['a child in a pink dress is climbing up a set of stairs in an entry way', 'a girl going into a wooden building']</code> |

Next, we need to build the vocabulary. The vocabulary consists of all the possible words which can be used - both as input into the model, and as output predictions, and we will build it using the cleaned words found in the reference captions from the training set. In the vocabulary each unique word is mapped to a unique integer. Read through and understand *vocabulary.py* and then complete the function `build_vocab(cleaned_captions)` in *utils.py* by following these steps:

- a. Collect words from the cleaned captions, ignoring any words which appear 3 times or less; this should leave you with roughly 3200 words (plus or minus is fine). As the vocabulary size affects the embedding layer dimensions, it is better not to add the very infrequently used words to the vocabulary.
- b. Add the collected words to an instance of the `Vocabulary()` object provided.

1.2 Extracting image features

Read through the `EncoderCNN` class in *models.py* and complete the *forward()* function.

Next, read through *extract_features.py* and write the code to iterate through the entire training set once, passing the images to the encoder model and saving all the features. Note that as this is a

forward pass only, no gradients are needed. Save the features to a file to avoid repeating this step, as it is the only step involving the entire training set of images.

1.3 Training DecoderRNN

Read through the `DecoderRNN` in *models.py*. First, complete the decoder by adding 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`. Then, read through *decoder.py* and write the code to train the decoder by passing the features, reference captions, and targets to the decoder, then computing loss based on the outputs and the targets. Train for exactly five epochs. You should see the loss go down to a low 2.-value. No need to use a validation set.

QUESTION II

2.1 Generating predictions on test data

To generate a prediction from the model, the test images must be passed into the encoder to extract the image features. Remember to first process the images with the `data_transform`. Then pass the features from the encoder into the decoder `sample()` function.

The decoder `sample()` function returns a one-dimensional vector of word IDs of 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. Use this output and the vocab to complete `decode_caption()` in *utils.py*, converting the word IDs into words and stringing the caption together.

Present three sample test images containing different objects, along with your model's generated captions and the 5 reference captions. (*report 2.1.2*)

2.2 Caption evaluation via text similarity

Quantitative evaluation of image to text models is not a trivial task. There are different methods for measuring the performance of such models. We will evaluate our model by measuring the text similarity between the generated caption and the reference captions, using two commonly used methods: Bilingual Evaluation Understudy (BLEU) and cosine similarity. For comparing the generated captions, please add code to the eval section of *decoder.py*, creating new functions in *utils.py* as needed.

(1) BLEU for evaluation

One common way of comparing a generated text to a reference text is using BLEU. 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. You can import it like this:

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

You may need to install the `nltk` package using `pip` or `conda` first.

Evaluate the trained model on the test set using the BLEU score and report the overall average score (*report 2.2.1*). Display one sample image with a high BLEU score and one image with a low BLEU score along with its generated caption and the reference captions (*report 2.2.2*).

(2) Cosine similarity

Another common way to measure the similarity between two texts is using cosine similarity. The cosine similarity measures the cosine of the angle between two vectors in n-dimensional space. The smaller the angle, the greater the similarity.

To use the cosine similarity to measure the similarity between the generated caption and the reference captions:

- a. Find the embedding vector of each word in the caption
- b. Compute the average vector for each caption
- c. Compute the cosine similarity score between the average vector of the generated caption and average vector of each reference caption
- d. Compute the average of these scores

The Python `scikit-learn` package provides a function to compute the cosine similarity between two vectors, you can import the function using the following command:

```
from sklearn.metrics.pairwise import cosine_similarity
```

Evaluate the trained model on the test set using the cosine similarity score and report the overall average score (*report 2.2.3*). Display one sample image with a high cosine similarity score and one image with low cosine similarity score along with its generated caption and the reference captions (*report 2.2.4*).

2.3 Comparing text similarity methods

Compare the model's performance on the test set evaluated using BLEU and cosine similarity and identify some weaknesses and strengths of each method (*report 2.3.1*). Please note, to

compare the average test scores, you need to rescale the Cosine similarity scores [-1 to 1] to match the range of BLEU method [0.0 - 1.0]. Find one example where both methods give similar scores and another example where they do not and discuss (*report 2.3.2*).

Submission details

Please submit the following:

1. Your filled-in CW2 Report Template file, named *[my_student_username]_cw1.pdf* as a **PDF file**. Please do not remove any of the existing template file.
2. All code written for both questions, as Python files, except for *config.py*. If you worked in a Notebook, please export to one or more .py file before submitting. Please zip the Python files together.

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>)