UDACITY

‹ Return to Classroom

Image Captioning

REVIEW

CODE REVIEW 3

HISTORY

Meets Specifications

Congratulations on finishing this project, you did great. In your answers, I see that you do not have any potential confusion and in fact, you seem to have a clear understanding of the project. Well done! 😊

If you are interested in knowing more about this task like where it can be useful, you can go through these resources:

- Visual Question Answering
- Rich Image Captioning in the Wild
- Image Captioning and Visual Question Answering Based on Attributes and External Knowledge
- Intention Oriented Image Captions with Guiding Objects
- Object Counts! Bringing Explicit Detections Back into Image Captioning
- A Multi-task Learning Approach for Image Captioning
- Counter Factual Visual Explanations

Since you successfully solved this task and already referred to the Show, Attend and Tell paper, now is the right time to make an attempt at writing code for this cooler, attention-based captioning problem. You can seek inspiration from these works on GitHub - One and Two.

Good luck with the rest of the nanodegree. 😊👍

Files Submitted

The submission includes model.py and the following Jupyter notebooks, where all questions have been answered:

2_Training.ipynb, and
3_Inference.ipynb.

All the required files have been submitted and all the questions have been answered. 😊

model.py

The chosen CNN architecture in the `CNNEncoder` class in model.py makes sense as an encoder for the image captioning task.

A wise move to go along with the `ResNet50` model as it is proven to have given very good results for this kind of problem.

If interested, you can try replacing it with Inception model as explained in Google's blog post on this task.

The chosen RNN architecture in the `RNNDecoder` class in model.py makes sense as a decoder for the image captioning task.

Your `RNNDecoder` looks great. Well done! 😊

If interested, check out this nice paper to know more about learning CNN-LSTM architectures for image caption generating use cases.

2_Training.ipynb

When using the `get_loader` function in data_loader.py to train the model, most arguments are left at their default values, as outlined in Step 1 of 1_Preliminaries.ipynb. In particular, the submission only (optionally) changes the values of the following arguments: `transform`, `mode`, `batch_size`, `vocab_threshold`, `vocab_from_file`.

The arguments have been set reasonably well. Thank you for explaining how certain values have been chosen. 😊

The submission describes the chosen CNN-RNN architecture and details how the hyperparameters were selected.

Your reasoning in the answers is on point. You have nicely given references to resources referred to as well. It is quite clear to me that you made well-informed decisions in coming up with the decoder architecture and also in selecting the hyperparameters, it is also clear that you are comfortable with the overall pipeline of the project.

Additional comments for real-time usage of these trained models:
It is nice to see that you followed good resources before starting. Most of the students often follow the suggested papers and it is probably the right thing to do when you are dealing with a new problem provided you are not sure where to begin. But what is also important is to understand the extreme cases like how small a network can I use to make the model predict decent quality captions. If you have the time and a good GPU to work this (I understand this is a big "if"), you should always be able to answer (to yourself at the very least) some questions like:

- Why `ResNet50` ?
- Why not `ResNet18` or other smaller but powerful network architectures like `MobileNet` , `ShuffleNet` and `EffNet` ?
- Can this problem be solved with smaller `embed_size` and `hidden_size` , maybe as small as `128` and even `64` ? Where is the breaking point?
- What other data transforms can help with the task at hand?
- How important is it to tune the `vocab_threshold` value?
- How can I quantitatively evaluate my models? (read about BLEU score here)

Experimenting with different configurations such as the ones I mentioned can be extremely boring and would require a GPU of your own but the insights can really help us when we're dealing with real-time memory/computational constraints (often faced when deploying trained models on edge devices). Anyways, you did the right thing no doubt but please try and make sure you can answer these kinds of questions to yourself whenever you are on to new problems. Overall, a great job! 😊

The transform is congruent with the choice of CNN architecture. If the transform has been modified, the submission describes how the transform used to pre-process the training images was selected.

The transform is in accordance with the choice of CNN architecture. The answer clearly suggests that you correctly understand what is going on here and why. 😊

Ever wonder if `RandomVerticalFlip` would help this task generalize or make the learning harder?

The submission describes how the trainable parameters were selected and has made a well-informed choice when deciding which parameters in the model should be trainable.

Your understanding of what parameters have been updated during the training (and why) is absolutely correct. 😊

The submission describes how the optimizer was selected.

Yes, `Adam` is often the safest choice to go with while solving these kinds of problems. Here is another interesting paper that summarizes all the existing optimizers in one place. Also, `Adam` actually adjusts the learning rate by itself while training but that is not possible when we train the model for just 1-3 epochs, this means (from my experience with this project) any other learning rate but `0.001` would have yielded different (potentially bad) results.

The code cell in Step 2 details all code used to train the model from scratch. The output of the code cell shows exactly what is printed when running the code cell. If the submission has amended the code used for training the model, it is well-organized and includes comments.

The code in Step 2 is well written with comments. 👍

You could try validating your models which would help you a great deal in avoiding overfitting and also in tuning your hyperparameters appropriately. And its always nice to see your loss value go down, graphically. 😉

3_Inference.ipynb

The transform used to pre-process the test images is congruent with the choice of CNN architecture. It is also consistent with the transform specified in `transform_train` in 2_Training.ipynb.

The chosen transform is efficient and it is nice to see that you did not include the data augmentation operation (`RandomHorizontalFlip`) in the test transform, something many students blindly include without putting much thought into it. 😅

The implementation of the `sample` method in the `RNNDecoder` class correctly leverages the RNN to generate predicted token indices.

Perfect, the entries from the output of the `sample` method do leverage the LSTM architecture to generate valid token indices. Each entry in the output corresponds to an integer that indicates a token in the vocabulary. RNN has worked correctly. 😊

The `clean_sentence` function passes the test in Step 4. The sentence is reasonably clean, where any `<start>` and `<end>` tokens have been removed.

The assertion holds and the sentences are reasonably clean, and any `<start>` and `<end>` tokens have been removed. Nice way of cleaning the sentences.

List comprehensions are always both computationally and aesthetically better choices over looping. Here's a more "pythonic" way of cleaning the sentences:

```
def clean_sentence(output):
    sentence = ''
    filtered_output = [index for index in output if (index != 0 and index != 1)]
    tokens = [data_loader.dataset.vocab.idx2word[index]
                           for index in filtered_output]
    ## Or you could do it all with one line of code
    tokens = [data_loader.dataset.vocab.idx2word[x] for x in output if x not in [0,
1]]

    sentence = ' '.join(tokens)
    return sentence
```

The submission shows two image-caption pairs where the model performed well, and two image-caption pairs where the model did not perform well.

The predictions look very cool. You pass this rubric for correctly providing two pairs of examples, one where the model performed well and another where the model did not perform well. Good job!

DOWNLOAD PROJECT

3 CODE REVIEW COMMENTS ›

RETURN TO PATH