UDACITY

Return to "Computer Vision Nanodegree" in the classroom

Image Captioning

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Hey!
Congratulations on completing the Image Captioning Project. The project meets specifications and you have done a fantastic job!
Suggestions:

1. Have a look at this link to read more on CNN-LSTM architecture used for image captioning.
2. To know more on how RNN and LSTM are used in image captioning, refer this.
3. Read this post to know the power of a CNN-RNN architecture.
4. Have a look at this tutorial as a reference for choosing hyperparameter values, transform, CNN-RNN architecture, optimizer and loss function.
   Good luck for the future projects!!

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.

Great! All the required files have been submitted!

model.py

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

The CNN architecture for the CNN encoder looks good, wise move to use the resnet pretrained model.

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

Good job on the RNN decoder architecture, the embeddings layer is necessary for the captions.

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 correctly set.

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

The CNN-RNN architecture chosen is correct. You might also want to try batch normalization in the CNN architecture and uniform weight initializations in the RNN or decoder architecture.

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, well done!

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.

You have made good decisions on selecting which parameters are to be kept as trainable and which are not to be kept trainable. It's true since we are using pretrained weights in the encoder model there are no trainable parameters except the fully connected layers in the Encoder but DecoderRNN has trainable

parameters.

The submission describes how the optimizer was selected.

Yeah, Adam is a safe option and performs well generally!

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 organized and well commented but I encourage you to validate your model in step 3 to check for overfitting and hyperparameter tuning. You can visualize the training, its always great to see your loss decreasing visually :)

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 same transform has been used for testing and training which is the right choice.

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

The Sample Method is implemented correctly. Each entry in the output corresponds to an integer that indicates a token in the vocabulary. RNN has worked perfectly.

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 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 are absolutely nailed ! Good job on providing two image caption pairs where performed well and not so well!

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review