

[Return to "Computer Vision Nanodegree" in the classroom](#)

Image Captioning

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congratulations on completing the project! Your model performs well on the images. All the best in 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! You have submitted all of the files and have answered the questions.

model.py

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



Excellent! The chosen CNN architecture in the `EncoderCNN` class in `model.py` makes sense.

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


 Awesome! The chosen RNN architecture in the `DecoderRNN` class in `model.py` makes sense.

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`.

 Good job! Most arguments for `get_loader` function were left at their default except for specific ones.

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

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

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.

 Good! The transform is congruent with the choice of CNN architecture.

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.

 Great! The submission describes how the trainable parameters were selected.

The submission describes how the optimizer was selected.

 Excellent! The submission describes how the optimizer was selected.


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

Shows exactly what is printed when running the code cell. If the submission has appended the code used for training the model, it is well-organized and includes comments.

 Awesome! The output of the code cell shows exactly what is printed when running the code cell.

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.

 Good! The transform used to pre-process the test images is congruent with the choice of CNN architecture.

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

 Excellent! The implementation correctly leverages the RNN to generate predicted token indices.

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

 Well done! The sentence is reasonably clean.

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 submission have the required examples.

 [DOWNLOAD PROJECT](#)

[RETURN TO PATH](#)

[Rate this review](#)

