

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

# Image Captioning

## REVIEW

## CODE REVIEW 3

## HISTORY

### Meets Specifications

✨ Overall, great job. Congratulations.

You have learned a very powerful technique of using the encoder-decoder style of networks. These network have many applications specially when the encoder is also a RNN network. Such networks can learn a seq2seq transformation that is used in machine translation, speech recognition, building chat bot and many more potential applications. Here is [more information](#).

You also seem to have a good handle on things for this project. Everything was explained really well. You went beyond the project and tried beam search. This is always much appreciated.

Also, take a look at some [more information](#) on using CNN-LSTM for building image captioning.

The implementation in this project is roughly based on the [Show and Tell](#) paper. There is a newer version that also talk about the attention (an optional topic during the course lectures. [Check it out !](#)

### 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.

Thank you for submitting all required files.

## model.py

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


CNNEndoer is correctly implemented.

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

Your implementation for decoder is also correct.

## 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 parameters for `transform`, `mode`, `batch_size`, `vocab_threshold`, `vocab_from_file` are indeed correct. :thumbsup:

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

Thank you for providing a detailed description of your CNN-RNN network architecture. Your understanding of the whole architecture is impressive.

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.

Leaving transforms to default choice is correct. You also provided reasoning for not using additional transforms. Nicely 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.**

Indeed our encoder network is pre-trained and we only interested in modifying the last layer. In the decoder network, we need to train all parameters. It is also a good idea to get a good grasp on how many parameters are trainable in LSTM. Checkout this answer about [total number of trainable parameters](#) in LSTM.

**The submission describes how the optimizer was selected.**

Adam is a great choice for the optimizer. It is usually straightforward to tune and often time doesn't require a lot of experimentation to get it right. Here is [great summary](#) of why Adam is a powerful choice.

**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.**

Great job training the network from scratch. To take your model beyond the project, consider judging your model on the validation data as mentioned in the optional step in Training notebook. Additionally, you can try to visualize the shape of loss function. The [shape of the loss function](#) is often telling whether your hyper-parameters like learning rate etc are set properly or if they need tuning.

### 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.**

Using the default transforms is correct choice. However, you might want to avoid using transforms like random crop and flip for test data. The training and test data should be in a same format. The grayscale + normalization does make the train and test data in a same format. However, when you are doing data augmentation like random crop, flip, rotation etc, its only necessary to do on train set.

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

Sample is correctly implemented. Kudos for trying out beam search as well, looks good!

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

You correctly cleaned the 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 results looks great. The beam search produce much improved results. Thanks for trying it out and sharing with us. Great job!

 [DOWNLOAD PROJECT](#)

3

[CODE REVIEW COMMENTS](#)



[RETURN TO PATH](#)