MSCOCO image captioning

Minheng Wu

The idea of this project is to closely replicate the 'show and tell' model developed by google. The idea was to train a model to give caption to a picture.

Dataset

State Farm Distracted Driver Detection
https://www.kaggle.com/c/state-farm-distracted-driver-detection/data
There are over 25K picture and 10 labels labeling the action that a driver is taking. Each picture has one label.

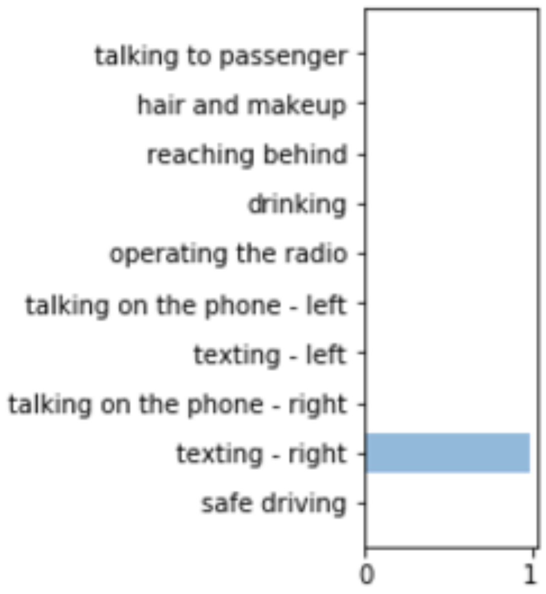
MSCOCO
http://mscoco.org/
A dataset provided by Microsoft, COCO stands for common objects in context. The data that I downloaded contains 80k+ images and their corresponding captions.

Preparation

The google paper uses its pre-trained CNN as a fixed feature extractor for images. So before I dig into this larger project of image captioning, I want to have a sub project to test how good are this well-known pre-trained models. The Data set I used to test is the State Farm data set

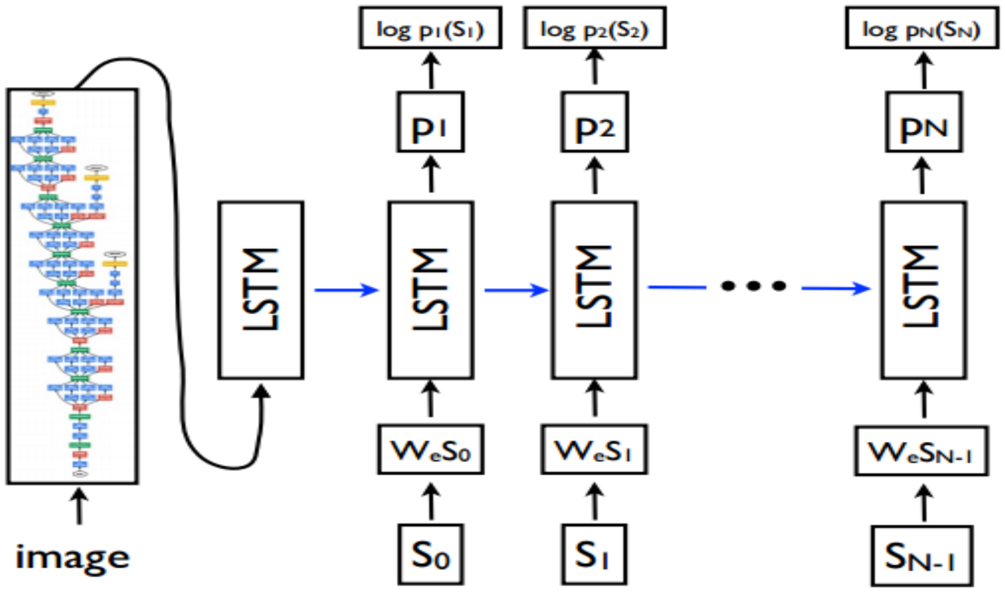
Training

First convert images to vectors. Each picture has dimension 480X640X3, then one hot encodes the label so each label has dimension 1X10. Before putting them through a CNN, I use KNN as a base model to test. So each image is flattened to be a 1XN vector, calculate the Euclidean distance of the testing data and each of the training data, with 5-neighbor training I was able to get an accuracy of 65%. The downside of this model is that testing time is long and is proportional to the training data set size.
For the CNN, I am using the Resnet developed by Microsoft. It was intended the train the Imagenet data set and won the competition. Let's say that one convolution, one batch normalization, one activation as one layer, the structure of Resnet consists multiple of these layers. The One I use in Keras is called Resnet50 so I am assuming there are 50 of such layers. In order to make the model compatible with my data, I replace the last few fully connected layers to one and have the output dimension to 10. Fix all the parameters from the Resnet and train only the parameters in the last fully connected layer. Essentially, I am treating Resnet as a black box feature extractor, train a simple network that maps feature to action. With only 2 epochs, I was able to get more than 99% validation accuracy. I also tested this model on various testing images, the outputted probability is very high towards the correct label.Without any need to hyper parameter search, this promising result shows that this kind of pre-train model is a very good feature extractor for images. We will now implement our image captioning model.



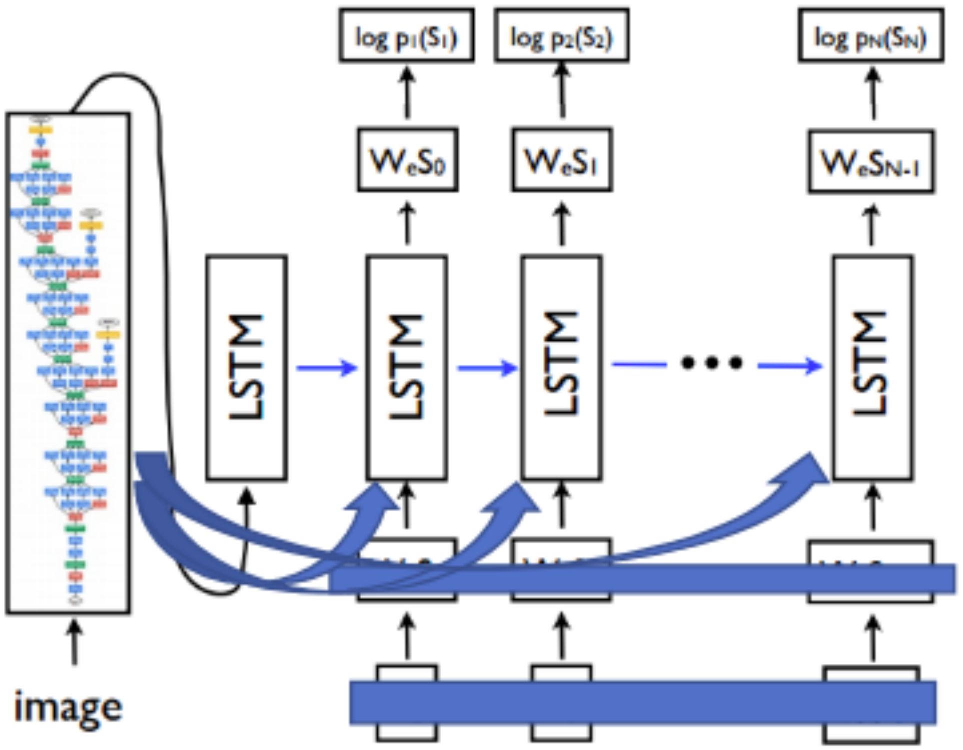
Img2txt

My model is based on the google show and tell paper.* The idea of the google show and tell is as follows:

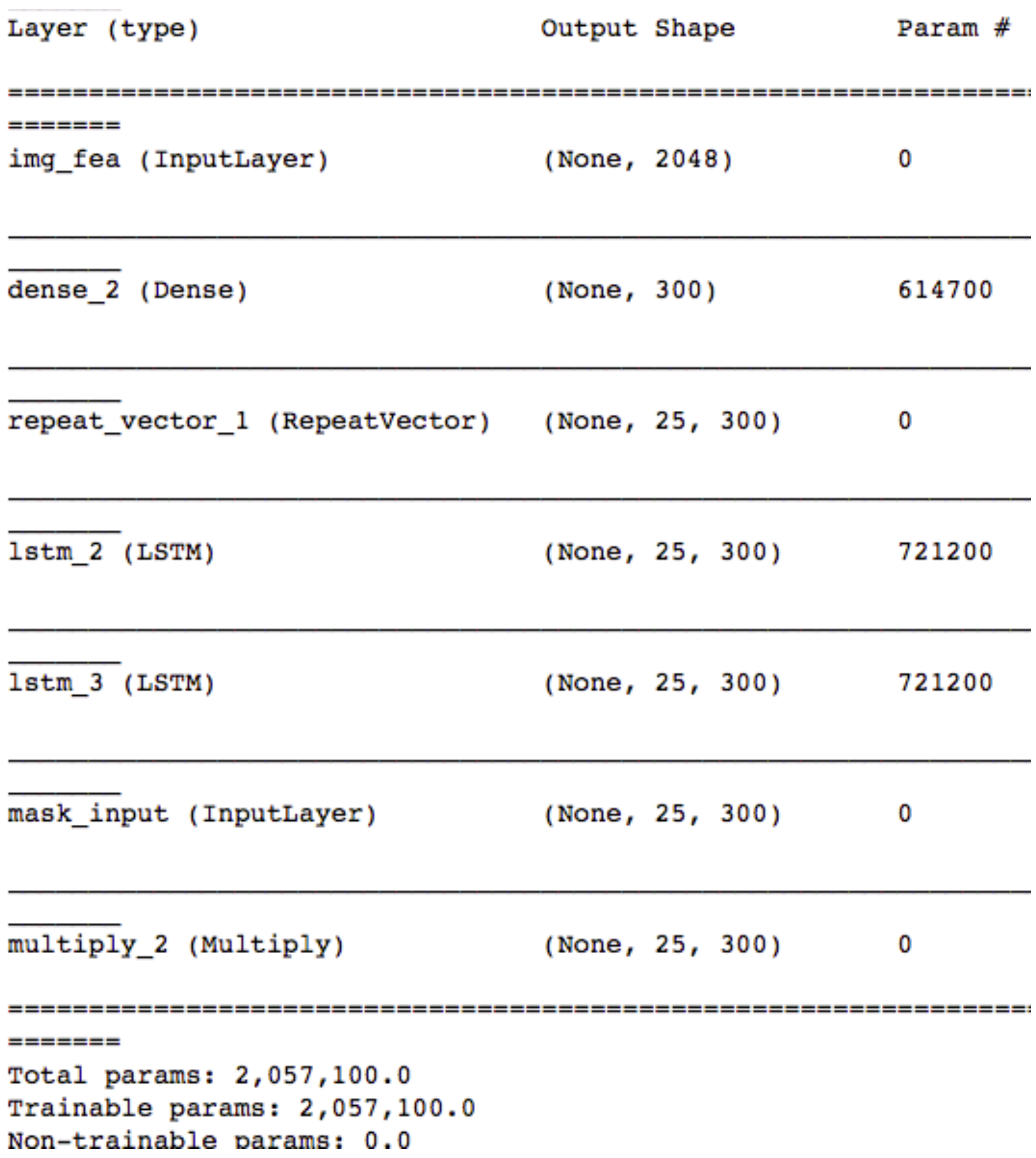


Where each image will go through a pre-train CNN(google uses its own Inception V3 which has more layer than the Resnet and yields better result in Imagenet competition), This will output a feature vector. For captions, the first thing they have done in google's paper is to index them. Combine all the captions to form a vocabulary list, use only the first 5000 words that occurs that most frequent, index them, and replace each word with its corresponding index in every captions, and then pad it the fix length of 25(add zero at the end for shorter captions and cut out for longer captions). Then they cut each caption into training_seq, target_seq, and mask. For example, a caption of [1, 2,3,4,0] will be divide into traing_seq of [1,2,3,0] . target_seq of [2,3,4,0] and mask of [1,1,1,0]. During training, the image vector will feed into an LSTM model to calculate the initial state, then the training_seq go through an embedding layer andl feed into the same LSTM network to predict target_seq, mask is use so that the padded token will not be considered toward loss calculation. At testing, where they no longer has the caption, they will feed the feature vector into the LSTM at time t0, have an output, feed that output back to the LSTM at time t1, and so forth.

Since Keras does not support such model building, I change the structure a little bit:



where at each time step the same feature vector will be fed to the LSTM. Other than that, the image feature extraction and caption encoding/embedding is same as the google paper. Here is my model structure

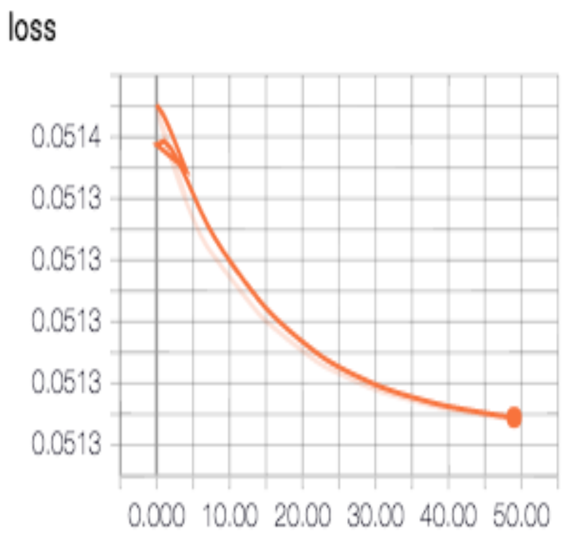


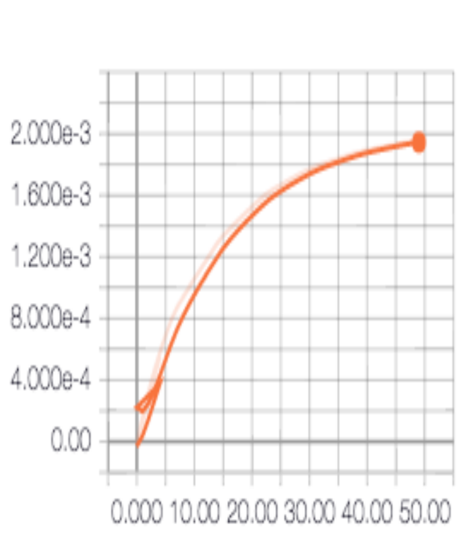
```
Layer (type)                     Output Shape          Param #
=================================================================
=======
img_fea (InputLayer)             (None, 2048)          0

_________________________________________________________________
_______
dense_2 (Dense)                  (None, 300)           614700

_________________________________________________________________
_______
repeat_vector_1 (RepeatVector)   (None, 25, 300)       0

_________________________________________________________________
_______
lstm_2 (LSTM)                    (None, 25, 300)       721200

_________________________________________________________________
_______
lstm_3 (LSTM)                    (None, 25, 300)       721200

_________________________________________________________________
_______
mask_input (InputLayer)          (None, 25, 300)       0

_________________________________________________________________
_______
multiply_2 (Multiply)            (None, 25, 300)       0
=================================================================
=======
Total params: 2,057,100.0
Trainable params: 2,057,100.0
Non-trainable params: 0.0
```

Hyper Parameter Choice:
I stick with the suggest of the google paper with learning rate initially set at 2 and exponentially decay with 0.5 after every 8 epochs. For loss function, since my output is the embedding of words, I use mean square error as my loss function instead of cross entropy used by google. With the same reason, I use r-square as my metric instead of accuracy.

Result

I have trained 50 epochs for this model. Loss went from 0.0514 to 0.0513. The loss curve gradually flattens out in the last few epoch. This is also the case for r-square, where it is slightly above 0 and barely increase. The output that I am getting is all 1s.





Take Away

The take away of this project is that pre-trained CNN works really well at extracting features. Keras is too high level and is not good for flexible model building. Tensorflow is a better choice at this situation. And model needs to train longer and with some better hyper parameter.