CS577 s20-Final Project

## Department of Computer Science

## Illinois Institute of Technology

Yuhan Li A20431466

Jia Tan A20433208

Apr 25, 2020

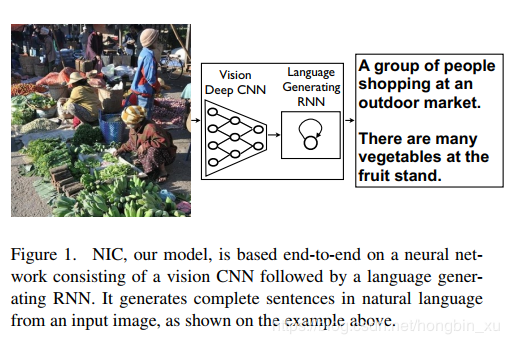
**Abstract**

Automatically describing the content of an image is a fundamental problem in artificial intelligence that connects computer vision and natural language processing. In this project, we implement the idea of the paper “Show and Tell: A Neural Image Caption Generator” . We use inceptionV3 as pre-trained CNN to extract features from images and LSTM as RNN to generate image descriptions.

# Problem statement

Our problem is to automatically describe the content of an image. This problem is encountered when we want automatic image annotation or image annotation. It can be used for applications such as image search and helping visually impaired people to view pictures, it is very valuable to have images with text descriptions.It is easy for humans, but very difficult for machines. Because it involves both understanding the content of images and translating the understood content into natural language. We need to let the computer understand the content of the image, and also need to express the meaning in human words, and the words must be concatenated in the correct way to be understood. The paper “Show and Tell: A Neural Image Caption Generator” did a great job in this problem. It presents a generative model based on a deep recurrent architecture that combines advances in computer vision and machine translation. We will implement this paper’s idea.

# Proposed solution

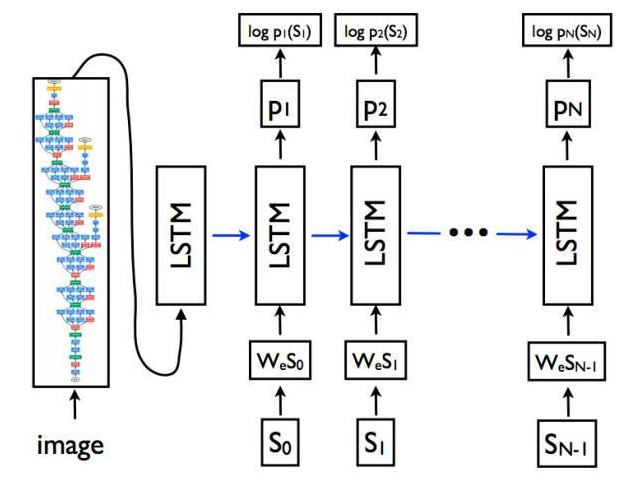


* + NIC Model

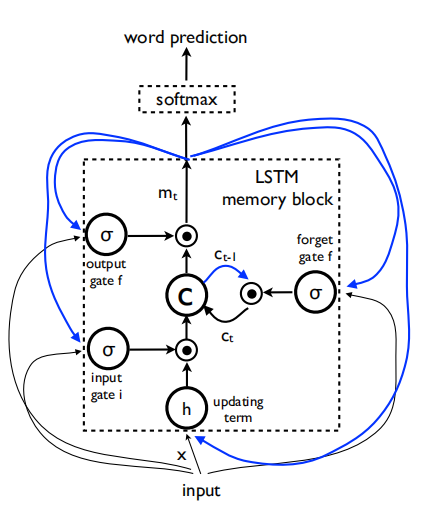
The model in this paper is inspired by machine translation using RNN, where the task is to transform a sentence *S* written in a source language, into its translation *T* in the target language, by maximizing *p*(*T|S*). An “encoder” RNN *reads* the source sentence and transforms it into a rich fixed-length vector representation, which in turn in used as the initial hidden state of a “decoder” RNN that *generates* the target sentence.

And in our problem, we replace the encoder RNN by a deep convolution neural network (CNN). CNNs can produce a rich representation of the input image by embedding it to a fixed-length vector. Hence, it is natural to use a CNN as an image“encoder”, by first pre-training it for an image classification task and using the last hidden layer as an input to the RNN decoder that generates sentences (see Figure. 1). We call this model the Neural Image Caption, or NIC.

In order to build an image caption generator model, we will merge these two architectures into a CNN-RNN model. CNN is used to extract features from images. We will use the pre-trained inceptionV3 model. LSTM will use information from CNN to help generate image descriptions.



* + Convolutional Neural networks (CNN)  
    CNN are specialized deep neural networks that can process data with input shapes such as 2D matrices. Used for image classification and image recognition. Given an image, significant features can be extracted through CNN, usually represented by a fixed-length vector. The extracted features are internal representations of the image, not something that humans can directly understand.
  + Long short term memory (LSTM)  
    LSTM are a type of RNN (Recurrent Neural Network), which is very suitable for sequence prediction problems. Based on the previous text, we can predict what the next word will be. It overcomes the limitations of RNN with short-term memory, thus proving its effectiveness from traditional RNN. LSTM can execute relevant information during the entire input processing, and through the "forget gate", it can discard irrelevant information.



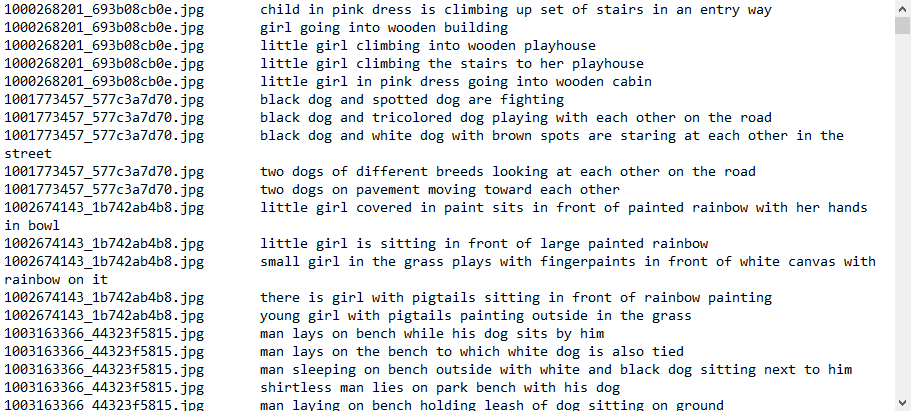
# Implementation details

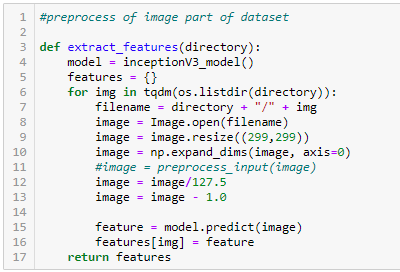
* + 1) Dataset Preparation

DataSet: we use the Flicker8k dataset. After download and unzip the dataset, there are two directories.

Flicker8k\_Dataset：contains 8092 jpg formed images.

Flickr8k\_text：contains a lot of txt descriptions.Flickr\_8k.trainImages.txt, Flickr\_8k.testImages.txt, Flickr\_8k.devImages.txtcontains image’s filename for training, testing,developing. Flickr8k\_token.txt: contains image descriptions text.

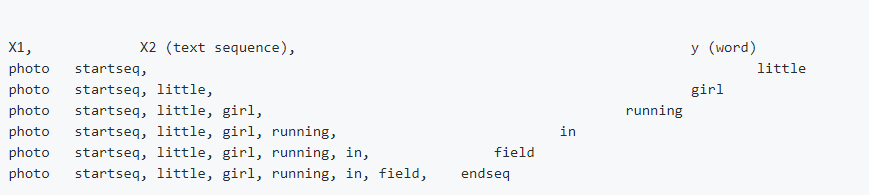
* + 2) Load text and perform data cleansing  
    The data set contains multiple descriptions for each photo, and the description text needs some minimal cleaning. Each photo has a unique identifier. This identifier is used in the text file of the photo file name and description.  
    Next, we will step through the photo description list. The following defines a function load\_descriptions (). Given the text of the loaded document, this function returns the dictionary of photo identifiers to the description. Each photo identifier is mapped to a list of one or more text descriptions. Next, we need to clean up the description text.  
    Ideally, we want a word that is both expressive and as small as possible. The smaller the vocabulary, the faster the model training speed. We will clean up the text in the following ways to reduce the vocabulary of the words we need to process: 1. Convert all words to lower case. 2. Delete all punctuation mark. 3. Delete all words that are no longer than one character (for example "a"). 4. Delete all words with numbers.  
    Finally, write a clean description into 'descriptions.txt'.  
    Here is part of 'descriptions.txt'.
  + 
  + 3) Extracting the feature vector   
    We use inceptionV3 as the pre-trained model that has been trained on large data sets, and extract features from these models and use them for our tasks. We plan to implement the model without the last classification layer, so we have the 2048 feature vector of output. These features would be the fixed-length “encoding”. We put the features for the training images and we will map image names with their respective feature array into “features.p” file.



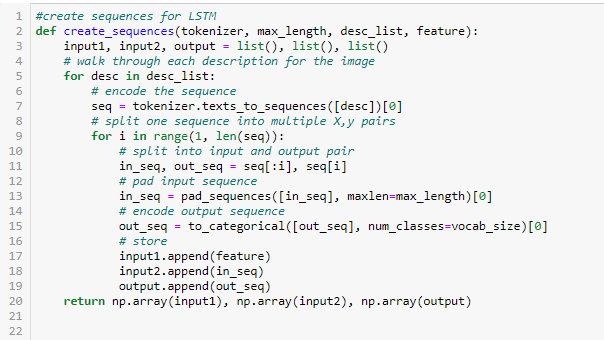
* + 4) Loading dataset for Training the model  
    We load the Flickr\_8k.trainImages.txt file, which contains a list of 6000 image names used for training.  
    Also create a dictionary with a clean description, which contains the title of each photo in the photo list. We also attached “startseq” and “endseq” notations to each subtitle. We need to do this so that our LSTM model can recognize the beginning and end of subtitles.
  + 5) Tokenizing the vocabulary   
    We will use “Tokenizer” function from Keras library to create a tokenizer of the training descriptions and save it to the "tokenizer.p" file.

Later the tokenizer will be used to extract the words and map each word in the tokenizer to a unique index .

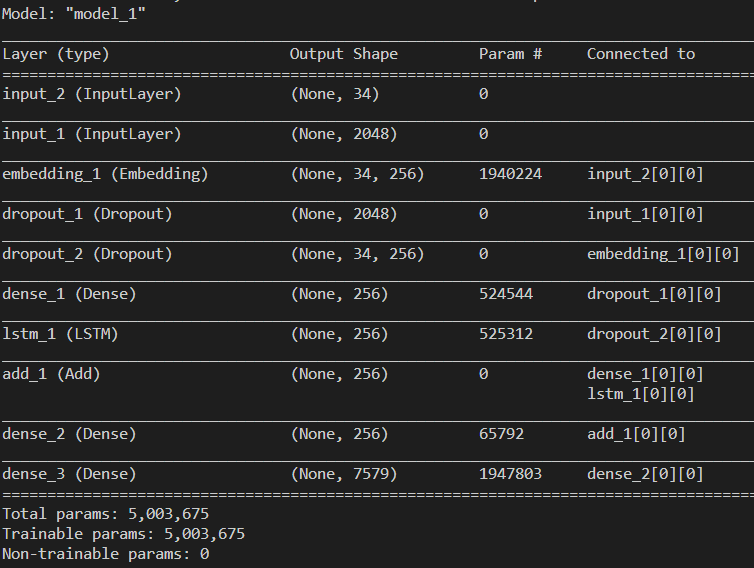
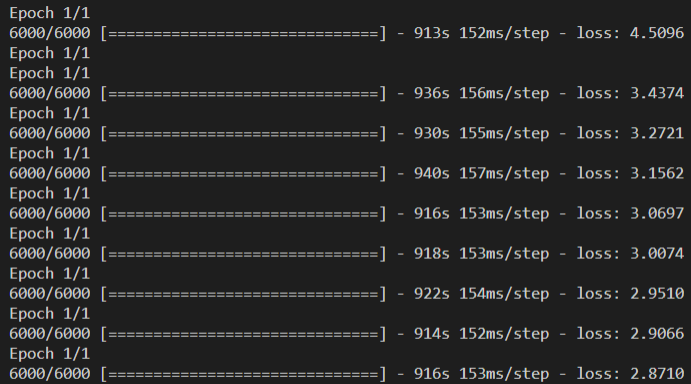
* + 6) Create sequences for LSTM and data generator

The LSTM is very suitable for sequence prediction problems. Based on the previous sequenced text, we can predict what the next word will be. For example, sequenced words are “a little girl running in field”, so it will be 6 input-output pairs to training the model:

So we need to create sequences for LSTM. There are two inputs, one for sequenced words, one for images’ features. The output would be the next word. And this predicted word would be added to the sequenced words to do the next prediction.



We also use a generator method to process batches, which can avoid holding too much data into memory.

* + 7) Defining the model  
    To define the structure of the model, we will be using the Keras Model from Functional API. It will consist of three major parts:  
    A. Feature Extractor – The feature extracted from the image has a size of 2048, with dropout layer and a dense layer, we will reduce the dimensions to 256 nodes.  
    B. Sequence Processor – An embedding layer will handle the textual input, and then a dropout layer followed. The LSTM will get the input from feature extractor and sequenced text as inputs and the output will be the input to the later dense layer.
  +   
    For our model, we have totally 10 layers and 5003675 parameters.
  + 8) Training the model  
    In the training step,we will use the 6000 training images by generating the input and output sequences in batches and fitting them to the model. We also save the model for each epoch to check whether the model becomes better and the loss is going down. We set 10 epochs and each epoch spent about 900 second. The final model loss is 2.8710.  
    
  + 9) Testing the model  
    We write a testing\_inceptionV3.py. It will load the “features.p” and “tokenizer.p” to process the testing images and descriptions. And the pre-trained model will be loaded to predict the result.

As the paper mentioned, there are two prediction algorithms. The first one is **sampling** where we just sample the first word according to *p*1, then provide the corresponding embedding as input and sample *p*2, continuing like this until we sample the special end-of-sentence token or some maximum length. The second one is **BeamSearch**: iteratively consider the set of the *k* best sentences up to time *t* as candidates to generate sentences of size *t* + 1, and keep only the resulting best *k* of them. This better approximates *S* = arg max*S0 p*(*S0|I*). [1]

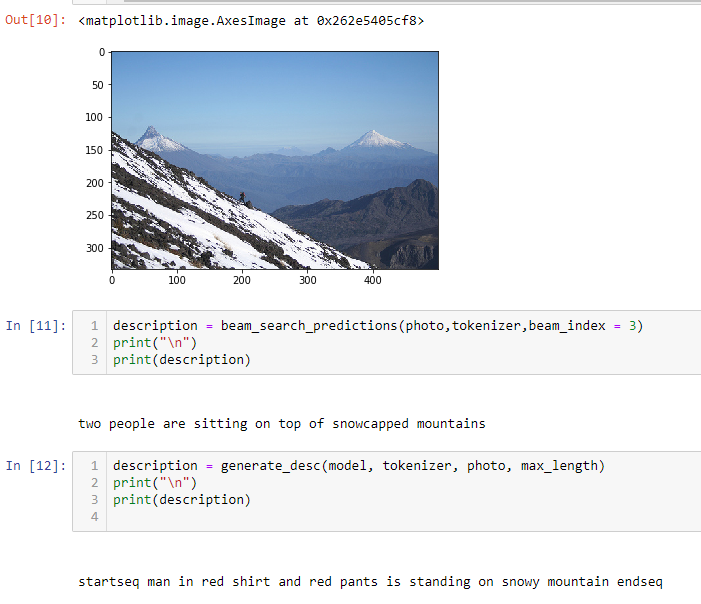
We implement both of them in the testing\_inceptionV3.py.

# Results and discussion

Testing the model:

We try to use some test images to predict the descriptions. The results are as below:



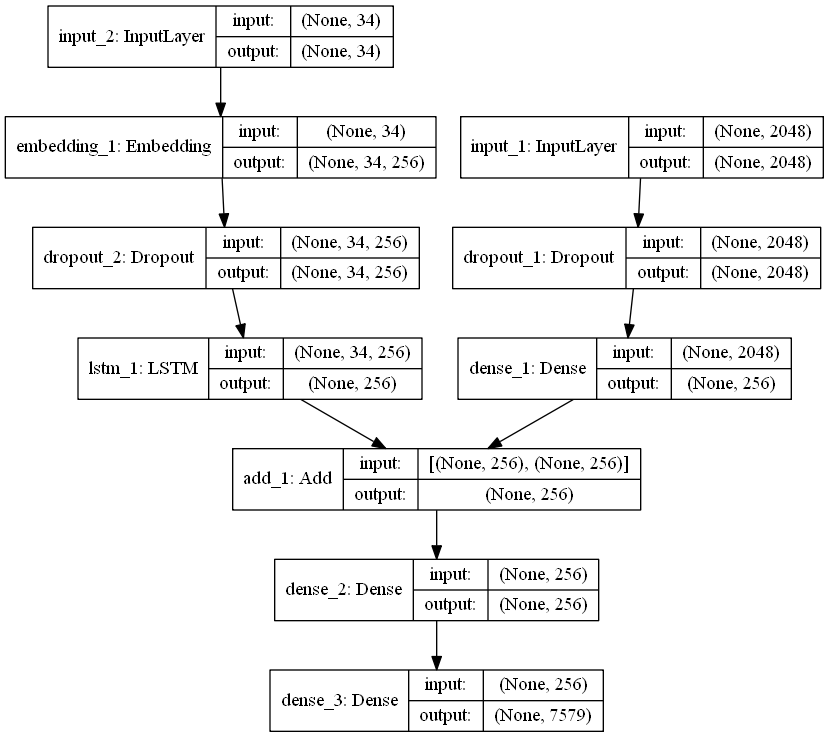




The results are not perfect, but meet our expectations.

Our model is as below:

The left part is the text input to LSTM while the right part is the process of the image features’ input. As we use the InceptionV3 as transfer learning to image features, it saves us a lot of time. This model runs well as one epoch about 15mins for the Flicker8k\_Dataset. NIC model did a good job for this problem.



Problems we met in the project:

We first used “<start>” and “<end>” as our notation for starting and ending a description. And do the tokenizer for the notationed descriptions. But we didn’t pay attention to the Tokenizer function in Keras would ignore all the punctuations, so “<start>” becomes “start” and “<end>” becomes “end”. That makes no difference between the notation and normal words and thus the model it trained is not right. Later we found the problem and decided to change the notation to “startseq” and “endseq” which is not a normal word but also without punctuations. And changed our codes to get an right model.

1. **Model improvement**

We implemented the image title generator through the CNN-RNN model. But our model has certain limitations, it cannot predict words outside of the training\_txt data. The occurrence of some items is too small, which leads to a decrease in the prediction accuracy of some infrequent objects. Some small movements cannot be distinguished, such as standing and running.

So for widely used we need larger dataset and different types of objects and movements to get a high-precision model.

1. **References**
   * (1)Vinyals, Oriol, et al. "Show and tell: A neural image caption generator." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015
   * (2) Flicker8k\_Dataset :<http://academictorrents.com/details/9dea07ba660a722ae1008c4c8afdd303b6f6e53b>
   * (3) Cho, Kyunghyun, et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." arXiv preprint arXiv:1406.1078 (2014).
   * (4) .Kulkarni, Girish, et al. "Babytalk: Understanding and generating simple image descriptions." IEEE Transactions on Pattern Analysis and Machine Intelligence 35.12 (2013): 2891-2903.