Automatically Generate Description for Image with Deep Learning

Guang Yang

Dept. Electrical and Computer Engineering

University of Florida

Gainesville, Florida

Guangyang031@ufl.edu

Juiche Tsai

Dept. Electrical and Computer Engineering

University of Florida

Gainesville, Florida

juichetsai@ufl.edu

Shipeng Guo

Dept. Electrical and Computer Engineering

University of Florida

Gainesville, Florida

Shipeng.guo@ufl.edu

*Generating textual description for images looks like interesting and full of fun, however, this task of generating textual description for images is easy for human but hard for machine to do it automatically. As the rapid growing of deep learning, big improvements have been made using deep learning in many areas. We explored the ability of deep neural network to generate caption for images, and it turns out the results are pretty good. The whole network is composed of two modules, a CNN connected with a RNN. CNN is fed with images and output information in the images as a vector, then the vector is fed into the RNN module and the textual description would be output by the RNN. We use a pretrained CNN model, which is VGG16, and LSTM as unit of RNN to incorporate memory mechanism. MSCOCO dataset is used to train the model.*

Keywords— image caption, convolutional neural network, recurrent neural network, LSTM, MSCOCO

# Introduction

When it comes to image caption, many of us would take it as a novel and exciting thing. For humans ourselves, generating some corresponding textual descriptions to photographs is intuitive and natural. However, this task can be very challenging for machines, because it’s hard for them to understand the content of an image and then generate the content in the image into a logical and complete sentence.

Classification is the most popular part of computer vision applying deep learning technique, like assigning images into different categories using various artificial neural networks, like what we came across in ImageNet(1000 classifiers) Obviously, this kind of technique has been mature enough, and now we would like to do something more than that, inspired with machine translation as well, so what we are trying to do is to combine classification with natural language process to achieve automatically generating textual description for an image according to its content, which is more exciting and novel.

# General system architecture

To achieve the goal of generating desirable textual descriptions for images we must make good plans, so we divide our project into two parts: the first one is Feature Extraction from the image, in this part we will use CNN to let the machine get the content in the image, and the second is Language Model, and we will use RNN with the input data from the previous CNN.

The most challenging part of this project could be the realization of simultaneously process the two components of this model: Convolutional Neural Network and Recurrent Neural Network. Besides, when there are some objects which belong to the same wide category in different regions in an image, how can we get them related one by one?

Maybe one of the method that we can refer to is to leverage these large images sentence datasets by treating the sentences as weak labels in which contiguous segments of words correspond to some particular, but unknown location in the image provided. Our approach is to infer these alignments and use them to learn a generative model of descriptions. Consequently, our contributions are twofold:

(a)we are able to develop a deep neural network that could infer or score the latent alignment between regions and segments of the photos they present which is feature extraction we mentioned in the beginning of this section.

The feature extraction model is a neural network which is known as Convolutional Neural Network. When given a image as input of this network, it could generate some deep or, let’s say, salient features of this image, which often in form of a fixed-length vector. In another word, what extracted are internal characteristics of the image, instead of something directly intelligible.

Fortunately, this deep learning network, or CNN, is able to be trained directly on the images in the dataset.

Figure 1 is the first part. We use CNN to let the machine know the content of images

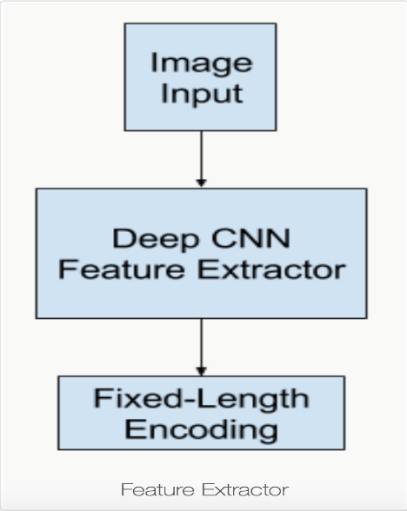


Figure 1

(b) we introduce a multimodal Recurrent Neural Network architecture which take an input, that is, Language Model.

Generally, a language holds the ability to predict the probability of the next word in the sequence given the words already present in the sequence.

For our topic – image captioning, the language model is a neural network that given the extracted features from the CNN above, the network is capable of predicting the sequence of all the words extracted and build up a whole and logical sentence at last.

In this project, we will use Long Short – Term Memory Network, or LSTM, instead of RNN, as the language model. LSTM is a enhanced version of RNN. It will be further explained in the later section.

Each word that is generated is then encoded using word embedding (such as word2vec) and passed as input to the decoder for generating the subsequent word. And there are some improvements in this part as well. In the future detail task, we may get into this and choose suitable one for our project.

The language model can be either trained alone using the output of upper network (CNN) or trained jointly with the feature extractor.

Figure 2 is the second part. We put the data generated by CNN to LSTM and let the machine generate a sentence to describe the image.

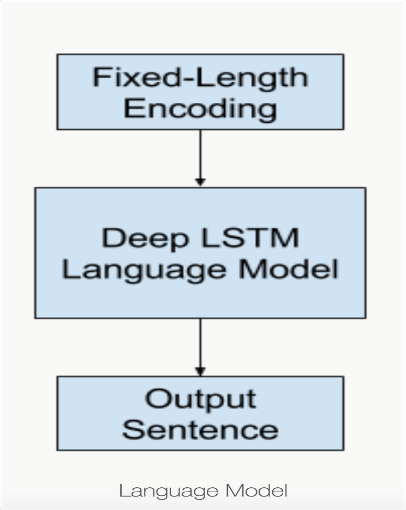


Figure 2

After that we can combine them together. This kind of architecture for the whole system is what we called Encoder-Decoder Architecture which is developed from machine translation, where an input sequence, say in Chinese, is encoded as a fixed-length vector by an RNN encoder network. A separate RNN decoder network then reads the encoding and generates an output sequence in another language, say English. Here, we replace the encoder with a deep CNN and the structure of decoder is LSTM.

# Details on system architecture

In this project, the basic architecture contains CNN and LSTM. The following paragraphs will explain the two models and describe how they connect.

1. Convolutional neural network

The first model for image processing is convolutional neural network (CNN). Figure 3 is the basic model of CNN.

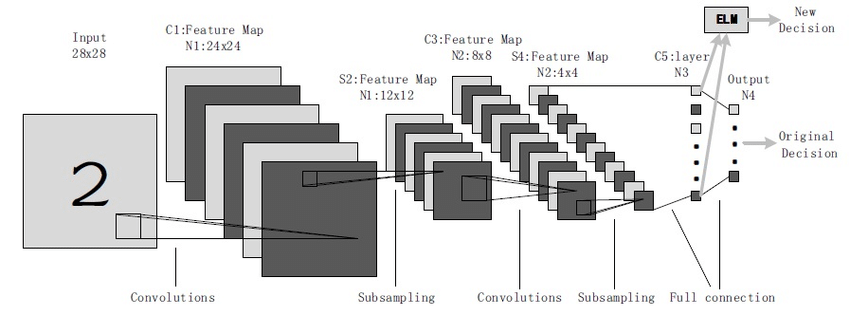


Figure 3

There are basically three kinds of layers: input layers, hidden layers and output layers. Input layer is the layer that we give input to the model. The number of neurons in input layer is the same as the number of features in input data.

Hidden layer is the layers in the middle. The data from input layer will feed into this layer. The number of hidden layers depends on the data size. Each hidden layer may have different neurons. The output of each layer is calculated by matrix multiplication of the previous layer. The matrix contains a bunch of learnable weights and bias.

The output of hidden layer will then be fed into a logistic function (we use softmax) here which translate the output into the probability score of each class of image.

The data is fed into input layer, processing in hidden layers and finally output in the output layer. This process is call forward propagation. After forward propagation, we can use the error function to calculate the error. Then, we backpropagate into the model by computing the derivatives. The step is backpropagation. We use backpropagation to minimize the loss of the model, so we can achieve the best performance.

Imagine taking a patch of a cuboid and running a small window on that area with m output and calculate them vertically. Then, we slide the window across the whole image, and we will get an image with different width, height and depth. Figure 4 below will show what I said above.

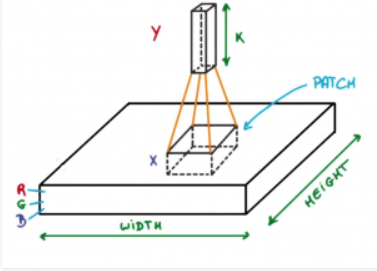


Figure 4

The mathematics involved in the CNN process

1. Convolutional layers have a set of learnable filters, and every filter may have different width and height and the same depth as the depth of input.
2. If we have to run CNN on an image with dimension, for example, 34x34x3, the possible size of filter can be WxWx3. W can be 3, 5, 7, and it will be relatively small compare with image dimension.
3. During the forward propagation, we slide the filter across the whole input step by step. Each step is called stride, and the stride can be 2, 3 or even 4. CNN computes the dot product between the patch from input volume and the weights of filters.
4. We will get a 2-D output for each filter when sliding the filters. Then, we will get output volume with a depth the same as the number of filters.

Different kinds of layer in convolutional neural network (Take the image of dimension 32x32x3)

1. Input layer: This layer holds the input of image with dimension 32x32x3
2. Convolution layer: This layer calculates the output volume by computing the dot product of image patches and all filters. For example, if we use 12 filters for this layer, we will get and output volume with dimension 32x32x12.
3. Activation function layer: This layer will apply element wise activation function to the output. There are some common activation functions such as ReLu, max pooling, sigmoid, tanh and leaky Relu. The dimension of output of this layer will not change.
4. Pool layer: The main function of this layer is to reduce the size of volume so that the computation can be faster and prevent some overfitting. There are two common type of pool layer, which are max pooling and average pooling. For example, if we use max pool with 2 x 2 with stride 2, the dimension of the result will be 16 x 16 x 12. The idea is show in figure 5

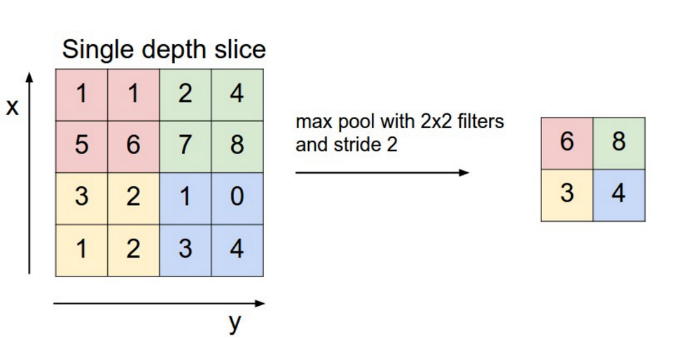


Figure 5 (max pool)

1. Fully-connected layer: This layer is basic neural network layer. The input is from the previous layer and extend the volume to a 1-D array of the size of the same number of classes as the previous input.

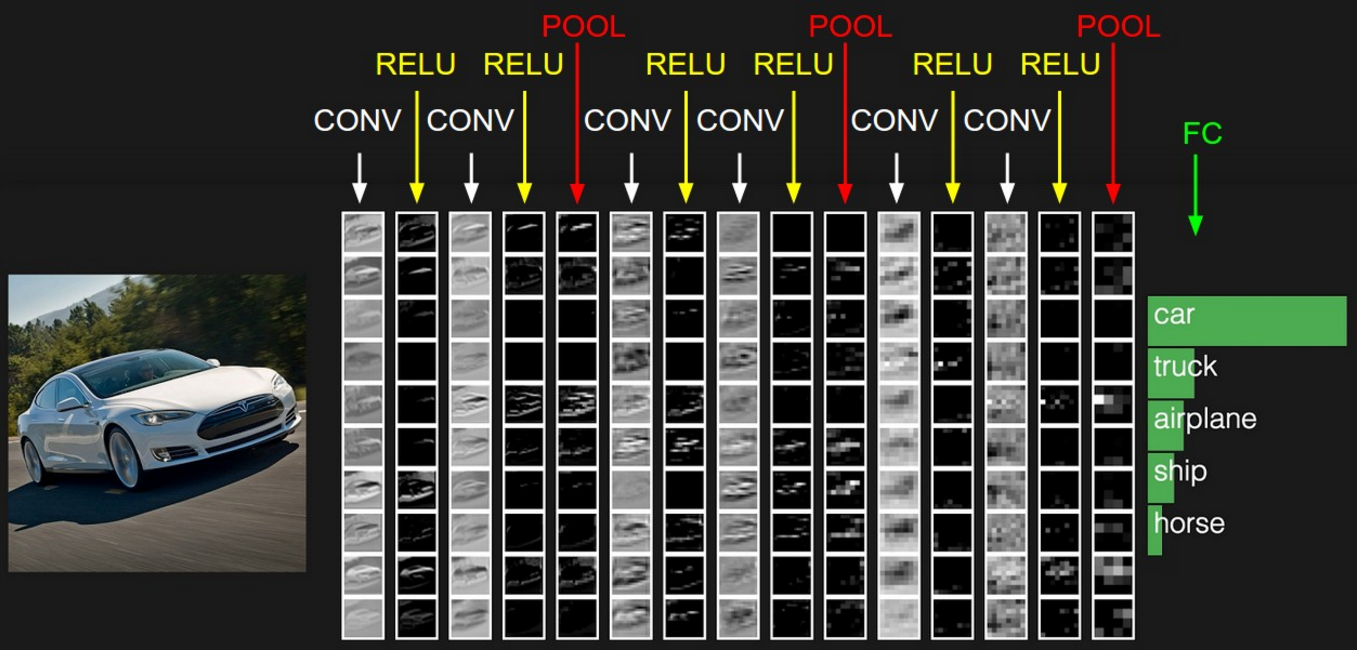


Figure 6 (fully-connected layer)

1. LSTM

We will use LSTM in the second part for text generating. In this section, we will discuss LSTM. Figure 7 is the basic architecture of LSTM.

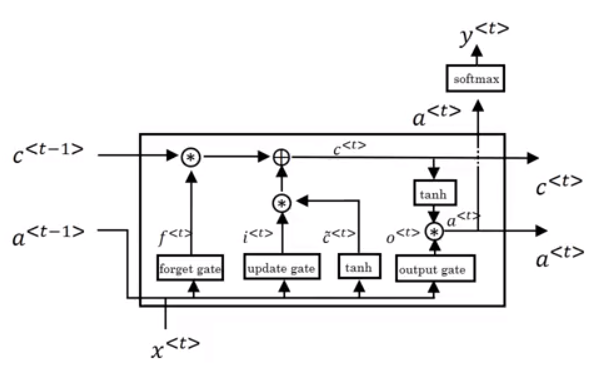


Figure 7

LSTM (long short-term memory) is generated from RNN (recurrent neural network). RNN is a very useful tool when it comes to language generating. However, there are still some weaknesses.

RNN is able to connect previous information from the previous unit. Nevertheless, sometimes we only have to look at the recent information. Take a language model trying to predict the next word based on the previous word for example. If the sentence is “the clouds are in the ***sky***”. We can predict the word “sky” without any further context. But in the sentence “The ***dog***, which has already eaten ………, **was** cute” and “The ***dogs***, which has already eaten ………, **were** cute”. In the sentence, we need to decide whether it needs to be was or were from the very previous word of the sentence and the gap between the relevant information can be very large. RNN is not able to connect the information. Theoretically, RNNs are capable of handling this kind of long-term dependencies, but in fact, it cannot. That is why we need LSTM to solve this problem.

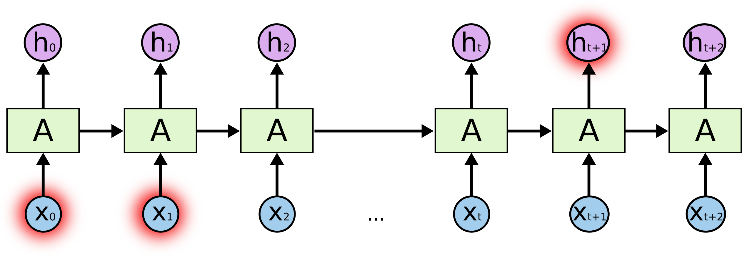


Figure 8 (RNN architecture)

Long short-term memory is a special kind of RNN. It is capable of learning long-term dependencies, and it is explicitly designed to solve the problem of long-term dependency that RNN cannot handle.

LSTM also has a chain like structure as RNN, but the structure of repeating module is different.

LSTM working process (explaining with figures):

1. The first step of LSTM is to decide what information we will throw away from the cell state. The decision is made by forget gate layer which is a sigmoid layer. It will look at ht−1 and xt and then output a number between 0 and 1 in Ct−1. 1 means keep this information while 0 means get rid of this piece of information.

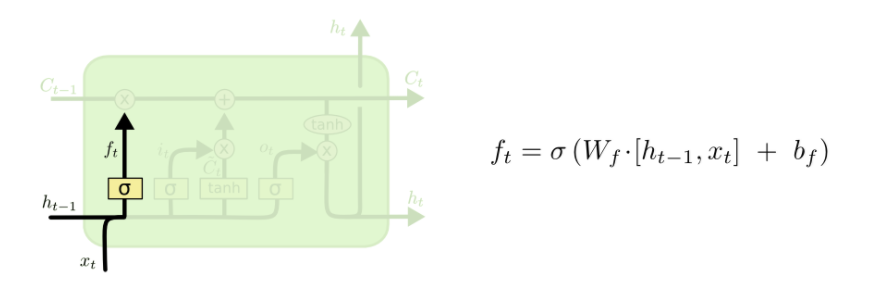


Figure 9 (first step of LSTM)

1. The second step is deciding what new information we need to store in the cell state. First of all, the input gate layer which is a sigmoid layer will decide which value will be updated. Next, a vector of new candidate values, Ct, will be created by a tanh layer. Then, we will combine the two values mentioned above to create an update to the state.

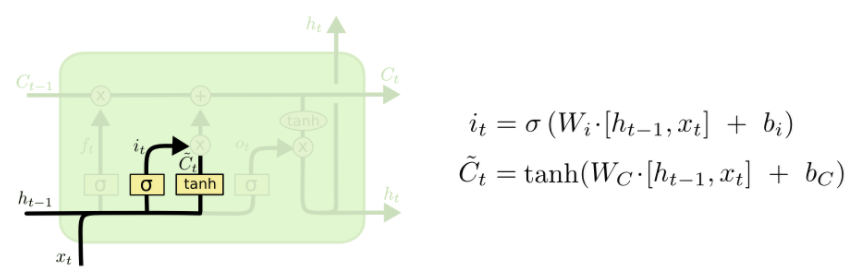


Figure 10 (second step)

1. In the third step, we will update the old cell state, Ct-1, into the new cell state Ct. We multiply the old state by ft and forget the information we decided to forget in the earlier step, and we get the new candidate value.

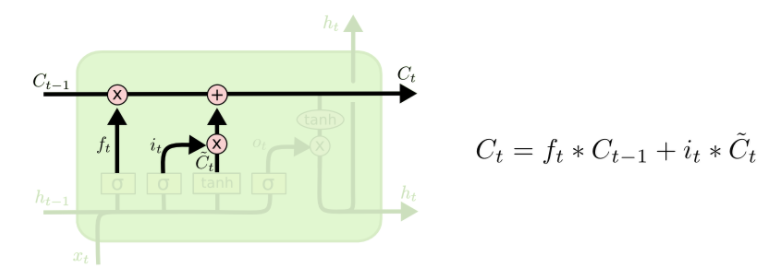


Figure 11 (third step)

1. Finally, we determine what we will output. The output will be based on cell state. First, we run a sigmoid layer to decide what parts of the cell state to output. Then, all the cell states go through tanh to let the value be between -1 and 1 and multiply it by the output of sigmoid gate. Therefore, we only have the output that we need.

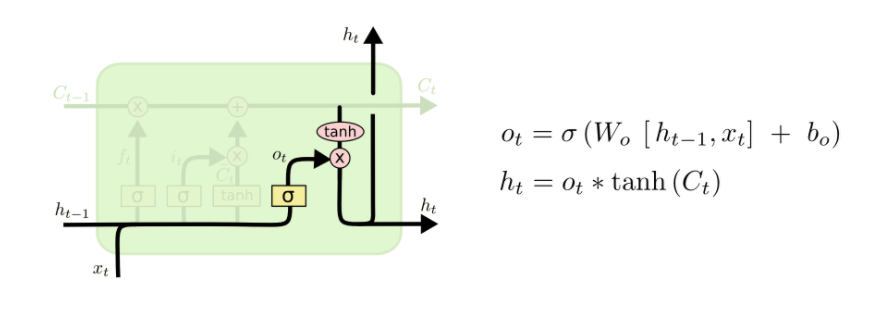


Figure 12 (final step)

1. Combine the two networks together.

We will combine the CNN and LSTM together to do this project. Like I mentioned before, CNN will let the machine know what object is in the image as well as how do they relate to each other and RNN can use the information from CNN to generate a sentence to describe the image. In detail, we use a pre-trained the CNN model VGG16, which was trained for image classification task, and eliminate the last layer (soft-max function here), and directly use the last hidden layer as input to the LSTM. After it has seen all potential features about images, LSTM can be trained through MSCOCO datasets to predict the probability of each word. That is, if we denote the input image by *I* and a true sentence by *S=(S0,…,SN),* the whole process can be generally shown as below:[6]

*∈*

*∈*

Here *St* stands for a one-hot vector whose length is the same as dictionary. And the loss of final LSTM can be inferred as below:

(d) Different approaches to generating sentences from an image.[6]

The first one is Sampling where we just take the top 1 word according to p1 and then take this as the input to next unit and sample p2.

Another one here is Beam Search which iteratively consider the top n best sentences to time t as candidates to generate sentences of size t+1, and keep only the resulting best n of them. For example, let’s assume the beam size to be 3, then at t=1, the 3 words with highest three probability are kept, let’s denote them as a1,a2,a3. Then at t=2, the 3 words combinations with highest 3 probabilities are kept, let’s say a1b2, a1b3, a3b2, where the universe of word combinations are a1b1, a1b2, a1b3, a2b1, a2b2, a2b3, a3b1, a3b2, a3b3. If the beam size is too large, it would be computationally expensive, on the other hand, if the beam size is too small, the robustness would be bad, because some choice of words with high probability may not be a good one in long term, if more words are kept, then this problem would possibly be fixed in the next time step.

We use beam search with beam size of 3, which we think is a good choice that can balance the computation workload and robustness searching mechanism.

# Evaluations of system

1. Choose evaluation metrics

How to evaluate the accuracy about our system? Unlike what the image classification uses, where image classification only has two possible results: True or False, evaluating the accuracy of generated caption is not that straightforward, because it is not easy to determine whether the generated description is either right or wrong, just because the generated sentences are not the same as the human-labeled sentences does not mean the generated ones are not right, they just need to sufficiently describe the information in the image, which in other words, means they only have to be of similar meaning of the human labeled sentences. We now consider using some metrics that can solve this problem.

There are several metrics available to measure the “distance” in terms of meaning between two sentences, for example, Bleu, METEOR, CIDER and ROUGH.

As inferred in [6], images in MSCOCO have contained corresponding sentences which labeled through Mechanical Turk experiment and we take these as reference sentences. The most common metric in the field so far is BLEU score, which is a measurement between automated caption and reference sentences inferred before. We calculate the frequency of n-gram of candidates in reference sentences (called precision)[7]. Even though there exists some shortcomings, it proves to correlate well with human evaluation.

Besides BLEU, the other metric we take into consideration is perplexity of our model. The perplexity is the geometric mean of the inverse probability for each predictable word. The lower the perplexity is, the more accurate this model is.

Recently, there comes another much more novel metric called CIDER[8]. Here, it measures the consistency between n-gram occurrences in generated and reference captions, in which the consistency is weighted by rarity and saliency.

Refer to [6], we can add more and various metrics to make our results much more convincing, like METEOR[9] and ROUGE[10].

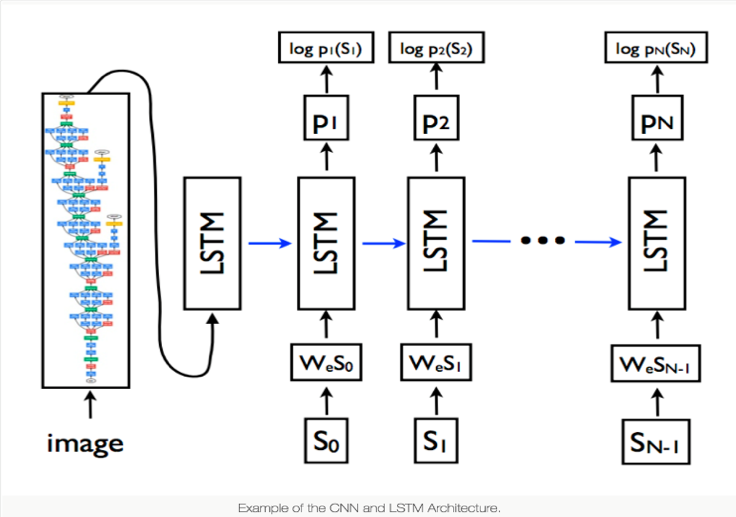


Figure 13 (basic structure of the project)

1. Datasets

As we said before, we decide to use Flickr8k as our datasets and the difference among different dataset is showed below.

|  |  |  |  |
| --- | --- | --- | --- |
| Dataset name | Size | | |
| train | valid | test |
| Pascal VOC 2008 [2] | - | - | 1000 |
| Flickr8k [42] | 6000 | 1000 | 1000 |
| Flickr30k [43] | 28000 | 1000 | 1000 |
| MSCOCO | 82783 | 40504 | 40775 |
| SBU [18] | 1M |  |  |

Except SBU, each image in those datasets has been labeled by 5 key words. SBU consists of descriptions given by image owners when uploaded them to Flickr. As such they are not guaranteed to be visual or unbiased and thus this dataset has more noise[6].

Obviously, MSCOCO has the most instances which can ease the overfitting problem and from several experiments, the model performance with MSCOCO is usually better than others. However, our laptops are not able to afford such a big dataset. Therefore, we decided to choose Flickr8k.

# experiment

Basically, we just following the neural network structure as we described above, which comprises a convolutional neural network as well as a recurrent neural network. With the convolutional neural network, the image are input and a vector that encode with the information in the image is output, which then is fed into the recurrent neural network(LSTM as units), in time step one, the first word vector is generated, and the ground truth word vector representing the information of the word in the human labeled sentences is compared with the generated one and their difference would be minimized and by back-propagating, the weights in RNN are trained. Several details should be mentioned as follows.

1. data preparation

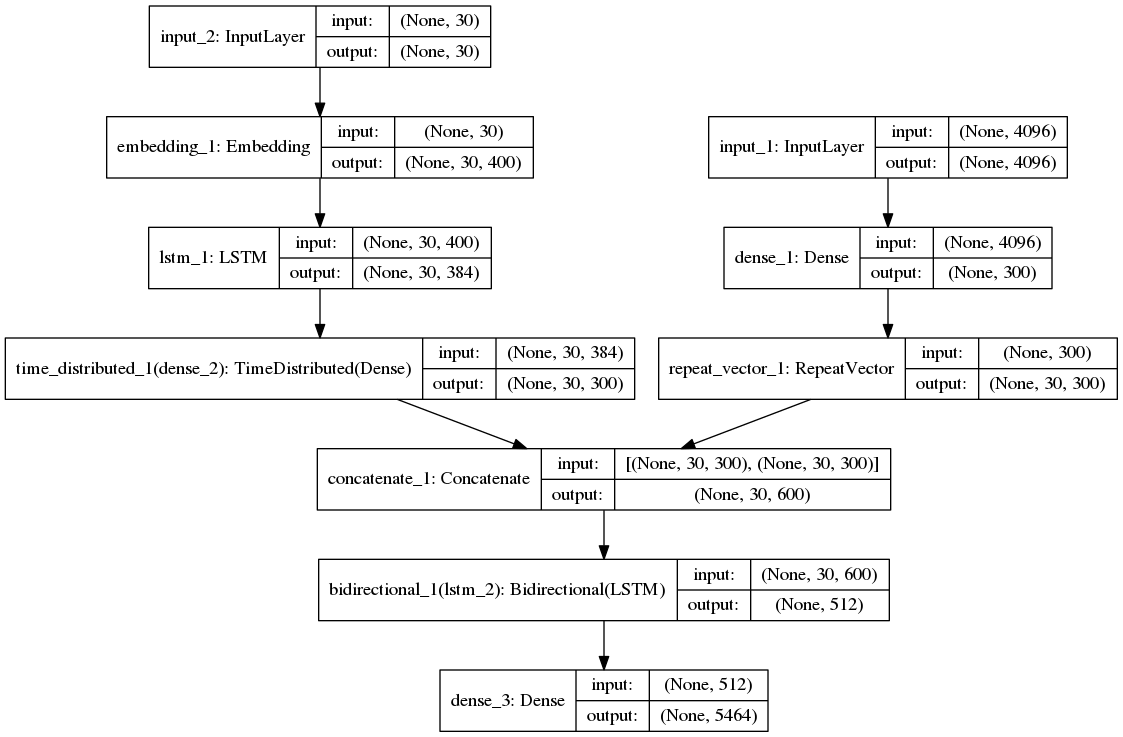
First, preparing text. The dataset is composed of around 8000 images with multiple descriptions for each image. Each image has a unique identifier, which is in the filename of images as well as in the text file of descriptions, so we can step through the list of image descriptions and save the first description. And the texts require minimal cleaning. First, the file containing all the image description is loaded, they would be cleaned so that the size of vocabulary can be reduced. By the way, the description is already tokenized, which means the sentences are broken up into words in this case. So the cleaning process including converting the words into lowercase, removing punctuation characters, removing the words which are composed of one character or less, e.g ‘a’. After reducing, we save the dictionary image identifiers and descriptions in the format of ‘one image identifier and description per line’.

Second, prepare the images. As mentioned before, in our project, what we apply is a combination of a pre-trained VGG16 network (we also tried InceptionV3) and a non-trained bidirectional LSTM. Then, we pre-compute these ‘photo features’ via this pre-trained model and save them in a file for future use. That is, after completing this process, what we should do is only to load the features from file and input corresponding texts for these images to train our LSTM model. Obviously, in this way, we don’t have to running each model through the whole network every time we want to test a new language model, which bring about much convenience and is able to save much memory for computer.

Third, further description pre-processing, the description text should be encoded to numbers so that we can input to the model and train.

What encoding the data here means is to create a consistent mapping from words to unique integer values. Some built in functions can be used which can learn this mapping from the loaded description data. Each description can be separated into single words.

The final model is shown below. In the left side, the model will import the features of training data. In the right side, the model will import the description. Then, they will be concatenated together. After they are concatenated, I use a bidirectional LSTM layer connecting with a dense layer to generate sentences. In the beginning, the last two layers are both dense layers. It could be trained faster, but the result was very bad. It will generate some nonsense sentences. Then, I change the dense layer before the final layer to a bidirectional LSTM layer. The training time increases a lot, the loss is also similar to the last model, but the results improve a lot.



1. Parameters of Networks

The model contains three parts:

First is feature extractor module achieved by a CNN, VGG16 and InceptionV3 model pretrained on ImageNet dataset is used, we have already fed all the images into the VGG model until the fully connected layer and we can directly use the feature vectors, which can significantly improve the training efficiency.

Second is sequence processor which is basically a word embedding layer for handling the text input, which is followed by LSTM layer.

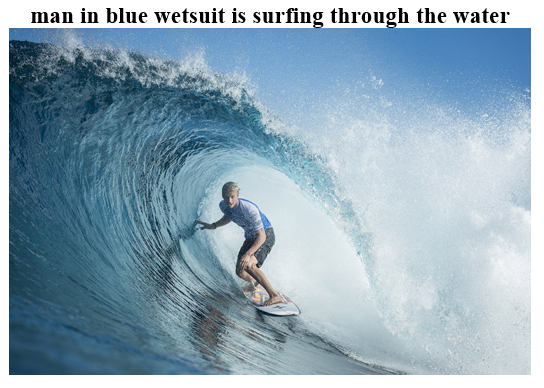
Third is a kind of interpreter achieved by an RNN with LSTM as units. The output of both the feature extractor and sequence processor are fixed length vector with the length of a maximum sequence, then they are concatenated and processed by LSTM.

In the beginning of our training process, learning rate will be higher so the model can be trained faster. After a while, we decrease our learning rate and try to hit the global minimum. We have tried different optimizers such as Adam, SGD and RMSprop, and we found SGD is too slow, RMSprop’s performance is a little bit worse than Adam’s performance, so we finally used the optimizer of Adam. Some regularizations are also used in order to prevent overfitting such as l2 regularizer or dropout (I tried ls regularizer, but the outcome is not good). There are 8000 images in my dataset, I use split them into 80% and 20%. 80% of the images are for training, and 20% of the images are for validation (testing). We did not train the model all at once. Instead, we separated into many stages. In the beginning, we used big learning rate to train, so the model can be trained faster. When the loss stops decreasing or even start to increase, we lower the learning rate. Then, when the model starts to be overfitting, we add some regularization into the model, and in this stage, we need to keep trying different values to find the best ones, because either the values are too big or too small will let the loss increases. During training, we also found an interesting phenomenon. When the loss stops decreasing, we can increase the batch size to reach a lower loss. When all the things that we could change in the model are done, we started to increase the batch size. After finishing training, we can use the trained model to generate texts for images.

1. Result

The results are shown below, and they are pretty good overall. The sentences are generated in the terminal, so I copy them and combine with the images. Most of the sentences can correctly describe the images. Surprisingly, some images may be blurred or have bad resolution (like the first result below), but the model still can generate right sentences to describe them.











# discussion

The system can automatically generate textual description for images, and the overall results are fairly good, but sometimes the model will generate some sentences not really describing the images. For example, it may generate wrong color of a person’s cloth or wrong something not related to the image. I think this is because the training set did not include the object or incidence in the testing image. For example, if the background is a bunch of rocks, but there are no training images with rock background, then the model will generate wrong description of the background, like woods or trees instead of rocks. The good thing is most of the sentences are reasonable, and although some sentences may have grammar mistakes, it is still easy to

understand for human beings. 

A brainstorm: We only trained RNN model and CNN is fixed, if there is additional training process which is training two modules jointly after RNN is trained solely, the results would be better, because the VGG16 was trained for classification, it must have bias toward some information in the image that is more important for classification and other information which is not that important for classification might be important for image caption task. Even for some model with attention module incorporated are just focus on the last one or two conv layers of pretrained CNN, which could also be biased, so what I was thinking is that actually using more conv layers all the way back to the first couple of conv layers, in this way the feature of various level can be utilized so that almost all the information of the original image can be obtained by the RNN. I was also thinking a discriminative network(part of GAN) can be used to discriminate between the sentences generated by the RNN and the true caption of the data, in this way, the RNN was learning how to generate captions with same style of human labeled caption instead of learning how to generate the exactly the same caption as labeled caption, and the results could be more naturally like human language.

# Reference

[1] Jason Brownlee.:’ How to Automatically Generate Textual Descriptions for Photographs with Deep Learning’.

<https://machinelearningmastery.com/how-to-caption-photos-with-deep-learning/> , 2017(13).

[2] Andrej Karpathy, Li Fei-Fei, “Deep Visual-Semantic Alignments for Generating Image Descriptions ,” Computer Vision and Pattern Recognition, 2015(4).

[3] Pranoy Radhakrishnan.:’ Image Captioning in Deep Learning.

<https://towardsdatascience.com/image-captioning-in-deep-learning-9cd23fb4d8d2>, 2017(9).

[4] Introduction to Convolutional Neural Networks for Vision Tasks.

<https://pythonmachinelearning.pro/introduction-to-convolutional-neural-networks-for-vision-tasks/>

[5] Understanding LSTM Networks

<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

[6] Oriol Vinyals, Alexander Toshev, “Show and Tell: Lessons learned from the 2015 MSCOCO Image Captioning Challenge,” Computer Vision and Pattern Recognition, 2016(9).

[7] K. Papineni, S. Roukos, T. Ward, and W. J. Zhu, “BLEU: A method for automatic evaluation of machine translation,” in ACL, 2002.

[8] R. Vedantam, C. L. Zitnick, and D> Parikh, “CIDEr: Consensus-based image description evaluation,” in arXIiv:1411.5726, 2015.

[9] S. Banerjee and A. Lavie, “Meteor: An automatic metric for mt evaluation with improved correlation with human judgements,” in Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization, vol. 29, 2005, pp. 65-72.

[10] C.-Y. Lin, “Rouge: A package for automatic evalution of summaries,” in Text summarization branches out: Proceedings of the ACL-04 workshop, vol. 8, 2004

[11] How to Develop a Deep Learning Photo Caption Generator from Scratch, Jason Brownlee

<https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python>