Individual project for natural language processing

Image Captioning

XinZhe Jin

Abstract

The objective of the following report is to explain how to create a model to generate a description with a photo. Training a deep learning model that combines two different problems in the fields of Artificial Intelligence (AI), computer vision and natural language processing.

1. Introduction

Giving a caption of an image is a task that every human can do, but it has been an important problem in the AI field because of the complexity of the task, because it includes two different problems that a simple model couldn´t resolved.

The first problem was the image classification, a very well-studied topic in the field of the computer vision. Where you can find a lot of models that can resolve this task.

The second one belongs to the field of natural language processing, where, we need to create a well-written sentence that describe a photo.

Both problems can be easily resolved individually, using the investigations in their relatives' fields, but the combination of them is a task hard to perform.

In spite of these premises, in recent years, researchers has achieved models combining architectures of both problems that perform quite well in the task of image captioning.

They discover that if you combine a convolutional neural network (CNN), that performs well in the task of image classification adding to the output a recurrent neural network (RNN), that had achieved good results in the text generation task, it is possible to create a mix architecture that resolve the problem.

In the following report, it will be exposed an example of an architecture and how to process the dataset for training the model, including the format of the dataset and how to adapt it as an input for the model.

2. The dataset

The dataset that I have chosen for the demonstration, it's a small dataset called Flickr 8k, which contains a total of 8000 images and 5 different captions for each one of them. The dataset is divided into 6000 images for training, 1000 for dev-test and the rest for test-set.

- 1000268201_693b08cb0e.jpg#0 A child in a pink dress is climbing up a set of stairs in an entry way .
- 1000268201_693b08cb0e.jpg#1 A girl going into a wooden building .
- 1000268201_693b08cb0e.jpg#2 A little girl climbing into a wooden playhouse .
- 1000268201_693b08cb0e.jpg#3 A little girl climbing the stairs to her playhouse .
- 1000268201_693b08cb0e.jpg#4 A little girl in a pink dress going into a wooden cabin .

Figura 1. An image of the dataset with his captions.

In spite of the availability of many other bigger datasets like MS COCO (containing more than 120 thousand images with their captions) or Flickr 30k dataset that has more photos and captions than the dataset I have picked. The reason why I have selected this one for the demonstration, is because of the limitation of time and resources that I have for training the model.

However, using Google Colaboratory platform to develop the model, I have avoided all those problems and create a model, using Keras library, that perform quite well with the few data that I have provided.

Creating the model

First, you can find all the steps in the IPython notebook attached with the report called TrainingModel.

Starting with the process, we need to process all the dataset in order to prepare all the images and captions for the training phase. So, in first place, we need to associate every caption with his relative image, with the information of “Flickr8k.token.txt”, I have created a dictionary which contains the name of the pictures as keywords and a list of captions related to each one.

After getting all the dataset in the dictionary, the next step is clearing the data like lower-casing all the words, eliminating words with numbers and special tokens (‘%’, ‘.’, …etc.).

Once We have cleared the data, we will create a vocabulary of the most commons words that will help the model to make better predictions. This vocabulary has 1651 unique words, but we will add one more index for the zero padding that I will be explained later, so we will have a total of 1652 words.

Now, we will load all the training images from “Flickr_8k.trainImages.txt” into Colaboratory. Simultaneously, we will add all the captions for the images into a new dictionary, but this time we will add two tokens, '*startseq*' and '*endseq'* to delimit the start and end of the description.

However, we can´t use directly the images as input for our model, so we have to convert the image into a vector first, using PIL and NumPy, we can transform all the images into vectors of 299x299x3. These vectors will be used for transfer learning using the CNN model Inception V3 create by Google.

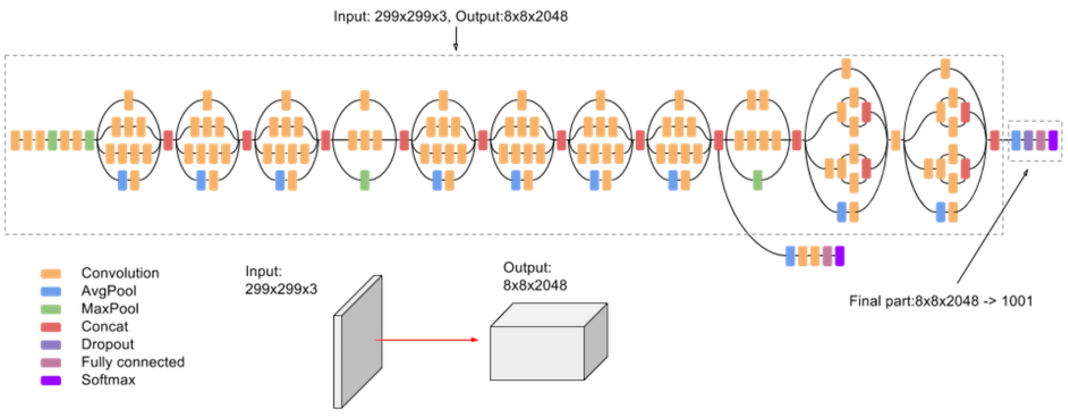


Figure 2. Architecture of Inception V3.

The chosen model had already been trained with a large dataset of images for doing the image classification task, but we don´t want to classify the image. Instead of that, we will take only the part of the architecture that extract the features of the images into a 2048 length vector. Connecting his output as the input for a new network that will take all those features and generate a description.

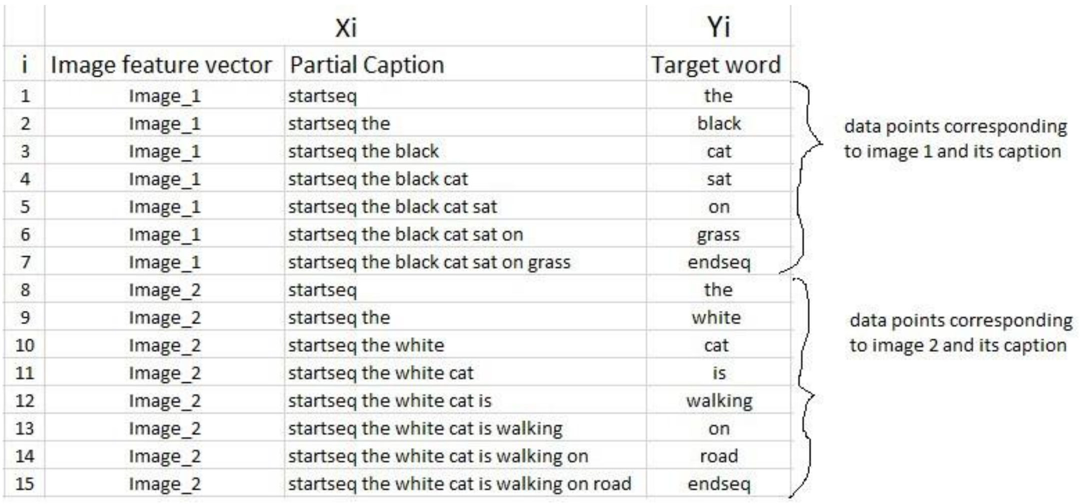
Hence, we need to apply transfer learning to the CNN, because the previous trained images don´t provide captions for those images, therefore we have to re-trained the model with the Flickr 8K dataset and using the sentences for the generation network.

Keras has already implemented functions that let us download Inception´s weights and select the architecture without the classification part, so all we need to do is call the function.

Once we have downloaded the model, we will introduce all the image vectors into the network, getting a 2048 length vector from the output that we will use after.

Now, we will prepare the captions for training. So on, we will create two dictionaries, the first one will contain the words of the vocabulary as keys and each one will be associated to a value from 1 to 1652 that will be the index of the word in the vocabulary. The second dictionary will have as key the index and the word as value. In addition, we will calculate also the maximum length of a caption that we will be an important parameter.

In order to train the model, we will pass him the captions and the features vectors, however, we won´t be passing all the caption at once, instead of that, we will do it in iterations, i.e., we will be starting with the first word of the caption and the model will try to predict the next word, and for the following iterations, we will add one more word to the partial caption, in order to predict the incoming word. You can see an example of the data matrix in the next figure.



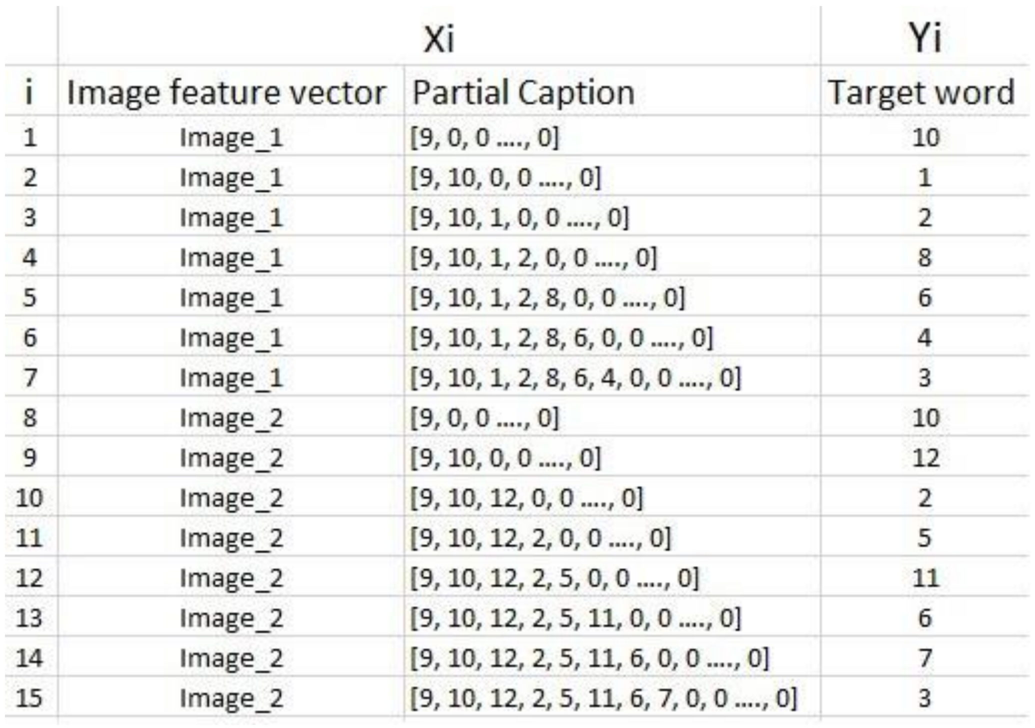
	Xi		Yi
i	Image feature vector	Partial Caption	Target word
1	Image_1	startseq	the
2	Image_1	startseq the	black
3	Image_1	startseq the black	cat
4	Image_1	startseq the black cat	sat
5	Image_1	startseq the black cat sat	on
6	Image_1	startseq the black cat sat on	grass
7	Image_1	startseq the black cat sat on grass	endseq
8	Image_2	startseq	the
9	Image_2	startseq the	white
10	Image_2	startseq the white	cat
11	Image_2	startseq the white cat	is
12	Image_2	startseq the white cat is	walking
13	Image_2	startseq the white cat is walking	on
14	Image_2	startseq the white cat is walking on	road
15	Image_2	startseq the white cat is walking on road	endseq

data points corresponding to image 1 and its caption
data points corresponding to image 2 and its caption

Figure 3. Example of the data matrix.

We will use a Recurrent Neural Network for processing all the captions, because of the structure of the network, we can store the information from previous iterations.

Nevertheless, we won´t be passing directly the captions in English, rather we will use the dictionaries that we have created before to convert each word into an index. Besides, in order to the limitation of memory we have in Colaboratory, we will do batch processing, separating the dataset in different groups. Therefore, we have to fill the captions with 0´s for getting the same length in every caption. That´s the reason, we add 0 as a word in our vocabulary and calculate the maximum length.

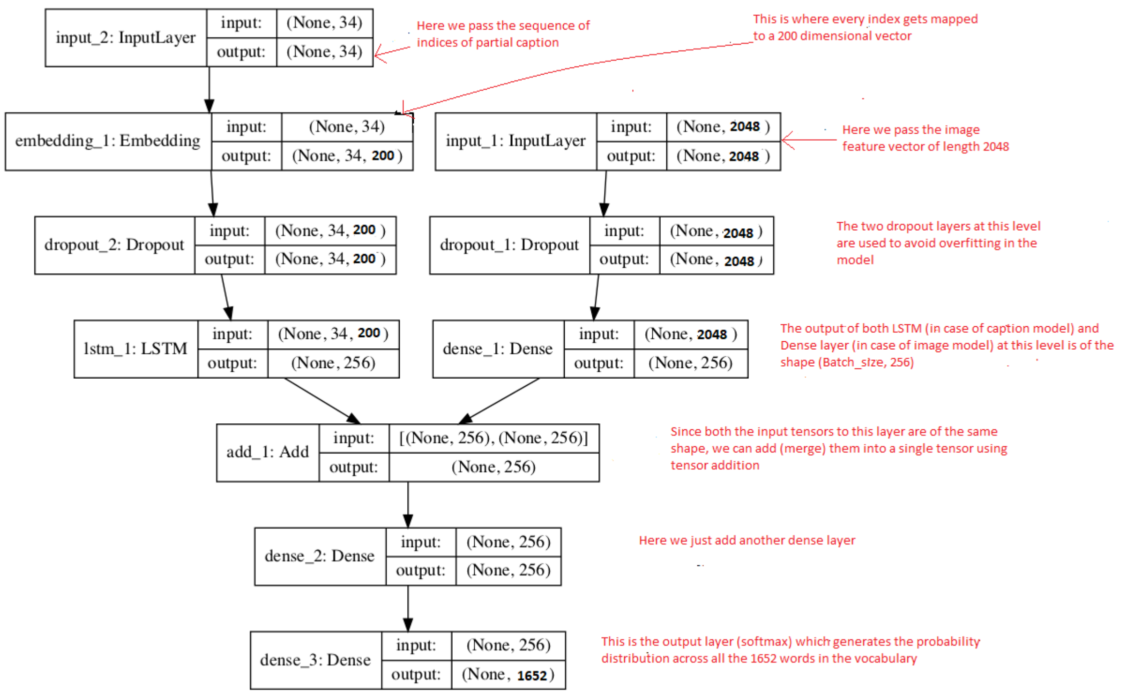


	Xi		Yi
i	Image feature vector	Partial Caption	Target word
1	Image_1	[9, 0, 0 …., 0]	10
2	Image_1	[9, 10, 0, 0 …., 0]	1
3	Image_1	[9, 10, 1, 0, 0 …., 0]	2
4	Image_1	[9, 10, 1, 2, 0, 0 …., 0]	8
5	Image_1	[9, 10, 1, 2, 8, 0, 0 …., 0]	6
6	Image_1	[9, 10, 1, 2, 8, 6, 0, 0 …., 0]	4
7	Image_1	[9, 10, 1, 2, 8, 6, 4, 0, 0 …., 0]	3
8	Image_2	[9, 0, 0 …., 0]	10
9	Image_2	[9, 10, 0, 0 …., 0]	12
10	Image_2	[9, 10, 12, 0, 0 …., 0]	2
11	Image_2	[9, 10, 12, 2, 0, 0 …., 0]	5
12	Image_2	[9, 10, 12, 2, 5, 0, 0 …., 0]	11
13	Image_2	[9, 10, 12, 2, 5, 11, 0, 0 …., 0]	6
14	Image_2	[9, 10, 12, 2, 5, 11, 6, 0, 0 …., 0]	7
15	Image_2	[9, 10, 12, 2, 5, 11, 6, 7, 0, 0 …., 0]	3

Figure 4. Data matrix after the transformation.

Once, we have finished with the process of the dataset, we will continue with the training of the model.

Since, the model receive two inputs, an image vector and a partial caption. We will use a merge model. You can see the architecture of the model in the figure 5.



input_2: InputLayer | input: (None, 34) | output: (None, 34)
Here we pass the sequence of indices of partial caption
This is where every index gets mapped to a 200 dimensional vector
embedding_1: Embedding | input: (None, 34) | output: (None, 34, 200)
input_1: InputLayer | input: (None, 2048) | output: (None, 2048)
Here we pass the image feature vector of length 2048
dropout_2: Dropout | input: (None, 34, 200) | output: (None, 34, 200)
dropout_1: Dropout | input: (None, 2048) | output: (None, 2048)
The two dropout layers at this level are used to avoid overfitting in the model
lstm_1: LSTM | input: (None, 34, 200) | output: (None, 256)
dense_1: Dense | input: (None, 2048) | output: (None, 256)
The output of both LSTM (in case of caption model) and Dense layer (in case of image model) at this level is of the shape (Batch_size, 256)
add_1: Add | input: [(None, 256), (None, 256)] | output: (None, 256)
Since both the input tensors to this layer are of the same shape, we can add (merge) them into a single tensor using tensor addition
dense_2: Dense | input: (None, 256) | output: (None, 256)
Here we just add another dense layer
dense_3: Dense | input: (None, 256) | output: (None, 1652)
This is the output layer (softmax) which generates the probability distribution across all the 1652 words in the vocabulary

Figure 5. Architecture of the merge model.

The left side of the architecture is the decoder that will process the captions, meanwhile the right side will get the output from the encoder, Inception V3, and extract the features from the vector. And the given output will be the probabilities of the next word to occur in the input caption.

Finally, the model was trained for 30 epochs, using a batch of 3 images in the first 20 epochs and for the last 10 epochs a batch of 6 images.

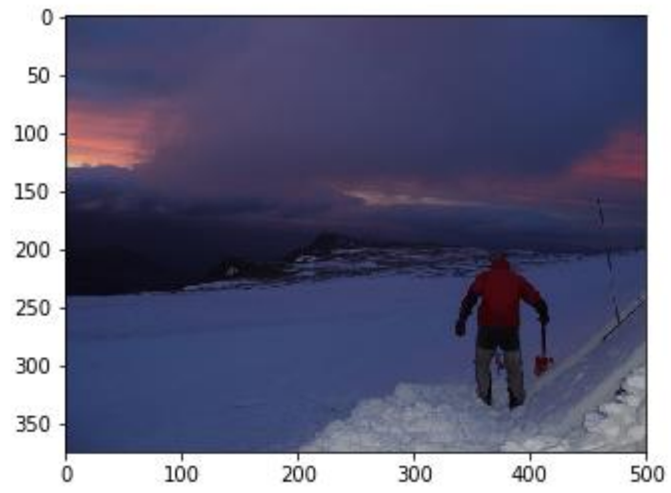
All this process took about 3 hours in the Google Colaboratory platform using a GPU provided by Google.

Testing the model

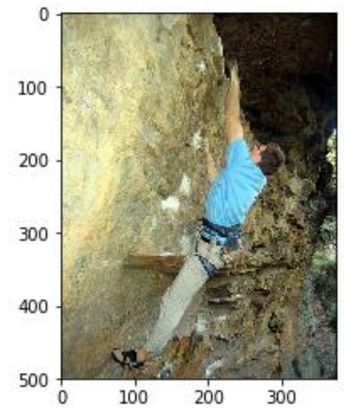
For our last step, we will test how our model works with unseen images. Following the same steps, we have done for the training process. We will introduce our testing images into the Inception V3 model in order to get our features vectors.

With the feature vector, we will provide it into our trained model with “startseq” as a partial caption. Then, we will choose from the output the word with the highest probability, this method is also known as Maximum likelihood Estimation.

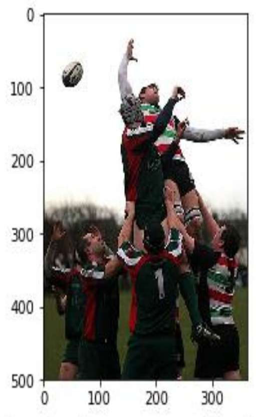
We will be repeating this step, adding to the partial caption the new word, until we reach the maximum length or finding the word “endseq”. Reporting the final caption without our two delimiters.



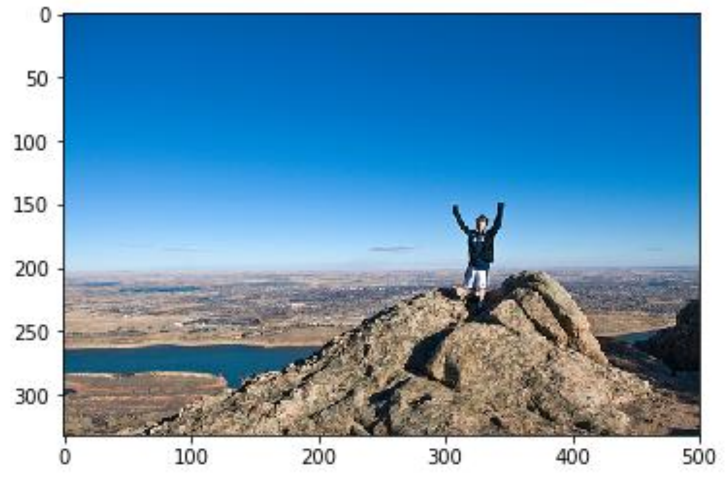
```
Greedy: man is standing on top of mountain looking at the mountains
```



```
Greedy: man in harness climbs cliff
```



```
Greedy: two men are playing football type football
```



```
Greedy: man in red shirt is standing on top of cliff overlooking the ocean
```

As you can see, some of the predictions are pretty accurate and could perfectly be a description of the image. However, there are also bad descriptions or reporting a sentence that doesn't have a correct structure.

In conclusion, despite the accuracy of our model, we are able to describe some the images quite well, using only a small dataset. So, if we try to train the model with bigger datasets, we will probably get better captions from new images. However, note that it will also require more training time and memory.

Furthermore, there are more changes that we could introduce to our model to improve the caption generation. For example, we could change the Inception V3 encoder for another CNN structure so we could get better features from the images. Or we can use another search algorithm for choosing the word after the prediction like Beam Search algorithm.

Concluding with my report, in case you want to test the notebooks, you will need to download the Flickr 8K dataset and change all the paths in the notebooks to your own paths.

Bibliography

1. https://towardsdatascience.com/image-captioning-in-deep-learning-9cd23fb4d8d2

2. https://towardsdatascience.com/image-captioning-with-keras-teaching-computers-to-describe-pictures-c88a46a311b8

3. A Comprehensive Survey of Deep Learning for Image Captioning. Source: https://arxiv.org/pdf/1810.04020.pdf

4. Show and Tell: A Neural Image Caption Generator. Source: https://arxiv.org/pdf/1411.4555.pdf