IMAGE CAPTION GENERATOR BY DEEP LEARNING

TADIKONDA PRUDHVI SIVA SAI KRISHNA (1001774416),

SATISH RELLA (1001677995), ADITYA SAGAM (1001660179)

ABSTRACT

In recent decades, with the quick improvement of artificial intelligence, image captioning has step by step pulled in the consideration of numerous scientists in the field of man-made brainpower i.e. AI and has become an intriguing and difficult errand. Picture subtitle, consequently producing characteristic language portrayals as indicated by the substance saw in a picture, is a significant piece of scene understanding, which joins the information on computer vision and Natural Language Processing (NLP). The utilization of image subtitle is broad and noteworthy, for instance, the acknowledgment of human-computer interaction (HCI). This paper sums up the related techniques and spotlights on the consideration component, which assumes a significant role in computer vision and is as of late generally utilized in image caption generation tasks.

INTRODUCTION

Caption generation is a challenging artificial intelligence problem where a textual description must be generated for a given photograph. It requires both methods from computer vision to understand the content of the image and a language model from the field of natural language processing to turn the understanding of the image into words in the right order. Recently, deep learning methods have achieved state-of-the-art results on examples of this problem. Deep learning methods have demonstrated state-of-the-art results on caption generation problems. What is most impressive about these methods is a single end-to-end model can be defined to predict a caption, given a photo, instead of requiring sophisticated data preparation or a pipeline of specifically designed models.

DATASET

The Flickr 8k dataset a great dataset for training and use in a real time machine learning project since it is smaller and compatible for use on a personal desktop. It comprises of a variety of images in the real-life scenarios and hence is quite suitable for our requirement.

It consists of two parts:

- Flickr8k_Dataset: Contains approximately 8k pictures (.jpg)
- Flickr8k_text: .txt files containing 5 captions associated with each image in the dataset above. The dataset has a pre-defined training dataset (6,000 images), development dataset (1,000 images), and test dataset (1,000 images).

PROJECT DESCRIPTION

In this project, we systematically analyze a deep neural network-based image caption generation method. With an image as the input, the method can output an English sentence describing the content in the image. We analyze three components of the method: convolutional neural network (CNN), recurrent neural network (RNN) and sentence generation. Automatically describing the content of images using natural languages is a fundamental and challenging task. It has great potential impact. For example, it could help visually impaired people better understand the content of images on the web. Also, it could provide more accurate and compact information of images/videos in scenarios such as image sharing in social network or video surveillance systems. This project accomplishes this task using deep neural networks. By learning knowledge from image and caption pairs, the method can generate image captions that are usually semantically descriptive and grammatically correct.

Loading Data

To begin with, we should stack the readied photograph and content information so we can utilize it to fit the model.

We are going to train the data on the entirety of the photographs and captions in the training dataset. While training, we are going to screen the presentation of the model on the advancement dataset and utilize that exhibition to choose when to save models to record. The train and development dataset have been predefined in the *Flickr_8k.trainImages.txt* and *Flickr_8k.devImages.txt* files respectively, that both contain lists of photo file names. From these file names, we can extract the photo identifiers and use these identifiers to filter photos and descriptions for each set.
Presently, we can stack the photographs and portrayals utilizing the pre-characterized set of train or improvement identifiers.

The model we will create will produce an inscription given a photograph, and the subtitle will be created each word in turn. The arrangement of recently created words will be given as info. In this manner, we will require a 'first word' to commence the age procedure and a 'final word' to flag the finish of the subtitle.

The description text will need to be encoded to numbers before it can be presented to the model as in input or compared to the model's predictions. The first step in encoding the data is to create a consistent mapping from words to unique integer values. Keras provides the *Tokenizer* class that can learn this mapping from the loaded description data. We write a function to convert the dictionary of descriptions into a list of strings and the and one function that will fit a Tokenizer given the loaded photo description text.

Running this kind of a model first loads the 6,000 photograph identifiers in the test dataset. These highlights are then used to channel and burden the cleaned depiction content and the pre-registered photograph highlights.

We would now be able to encode the content. Every portrayal will be part into words. The model will be given single word and the photograph and produce the following word. At that point the initial two expressions of the depiction will be furnished to the model as contribution with the picture to produce the following word. This is the means by which the model will be prepared. For

instance, the information arrangement "young lady running in field" would be part into 6 info yield sets to prepare the model as below:

X1,	X2 (text sequence),	y (word)
photo	startseq,	little
photo	startseq, little,	girl
photo	startseq, little, girl,	running
photo	startseq, little, girl, running,	in
photo	startseq, little, girl, running, in,	field
photo	startseq, little, girl, running, in, field, endseq	

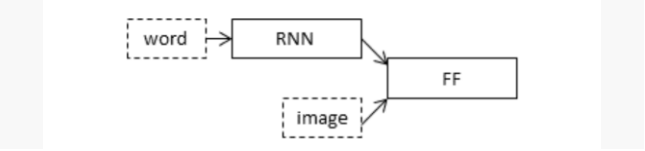
When the model is utilized to produce portrayals, the created words will be linked and recursively gave as contribution to produce an inscription for a picture. The capacity beneath named create_sequences(), given the tokenizer, a most extreme succession length, and the word reference everything being equal and photographs, will change the information into input-yield sets of information for preparing the model. There are two information exhibits to the model: one for photograph highlights and one for the encoded content. There is one yield for the model which is the encoded next word in the content arrangement.

The information content is encoded as numbers, which will be taken care of to a word inserting layer. The photograph highlights will be taken care of straightforwardly to another piece of the model. The model will yield a forecast, which will be a likelihood dispersion over all words in the jargon. The yield information will in this way be a one-hot encoded rendition of each word, speaking to a glorified likelihood dispersion with 0 qualities at all word positions aside from the genuine word position, which has an estimation of 1.

We now have enough to load the data for the training and development datasets and transform the loaded data into input-output pairs for fitting a deep learning model.

Defining the Model

We will define a deep learning based on the "*merge-model*" described by Marc Tanti, et al.



Schematic of the Merge Model For Image Captioning

We will describe the model in three parts:

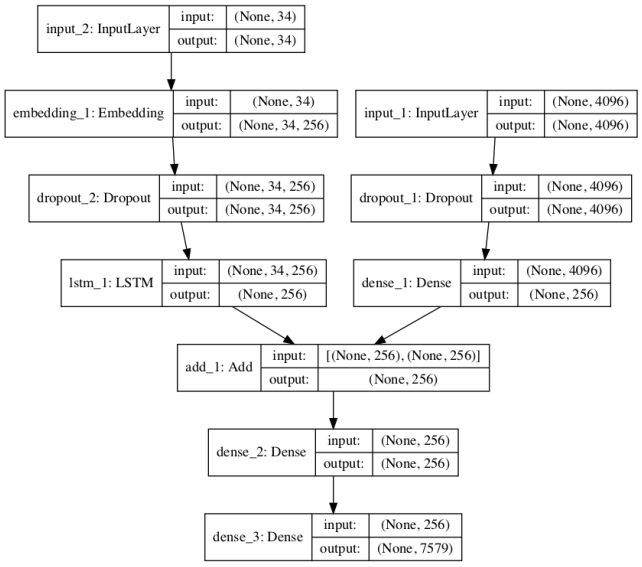
- **Photo Feature Extractor**. This is a 16-layer VGG model pre-trained on the ImageNet dataset. We have pre-processed the photos with the VGG model (without the output layer) and will use the extracted features predicted by this model as input.
- **Sequence Processor**. This is a word embedding layer for handling the text input, followed by a Long Short-Term Memory (LSTM) recurrent neural network layer.
- **Decoder** (for lack of a better name). Both the feature extractor and sequence processor output a fixed-length vector. These are merged together and processed by a Dense layer to make a final prediction.

The Photo Feature Extractor model expects input photo features to be a vector of 4,096 elements. These are processed by a Dense layer to produce a 256-element representation of the photo.

The Sequence Processor model expects input sequences with a pre-defined length (34 words) which are fed into an Embedding layer that uses a mask to ignore padded values. This is followed by an LSTM layer with 256 memory units.

Both the input models produce a 256 element vector. Further, both input models use regularization in the form of 50% dropout. This is to reduce overfitting the training dataset, as this model configuration learns very fast.

The Decoder model merges the vectors from both input models using an addition operation. This is then fed to a Dense 256 neuron layer and then to a final output Dense layer that makes a softmax prediction over the entire output vocabulary for the next word in the sequence.



Fitting the Model with Progressive Loading: If you need to utilize dynamic stacking, to prepare this model, this area will give you how. The initial step is we should characterize a capacity that we can use as the information generator. We will keep things exceptionally straightforward and have the information generator yield one photograph of information for every group. This will be the entirety of the groupings created for a photograph and its arrangement of portrayals. The capacity underneath data_generator() will be the information generator and will take the stacked literary portrayals, photograph highlights, tokenizer and max length. Here, I expect that you can fit this preparation information in memory, which I trust 8GB of RAM ought to be more than fit. How accomplishes this work? Peruse the post I just referenced over that presents information generators.

Note, this is a very basic data generator. The big memory saving it offers is to not have the unrolled sequences of train and test data in memory prior to fitting the model, that these samples are created as needed per photo.

Main references used for the project

1. https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python/ - We have developed the model based on the LSTM and RNN from this website.

Difference in APPROACH/METHOD between this project and the main project of the references

We have implemented, trained and tested the functionality for RSNET50 model instead of the VGG model. The result of this is a faster superior model that has lower process times.

Difference in ACCURACY/PERFORMANCE between the project and the main projects of the references

We have analyzed the project by using a different training model than the one used by our reference. We used RSNET50 as mentioned in the previous question and found that RSNET50 is faster than the VGG model.

Scores for VGG:

```
print('Photos: test=%d' % len(test_features))

# load the model
filename = '/content/gdrive/My Drive/Colab Notebooks/model_10.h5'
model = load_model(filename)
# evaluate model
evaluate_model(model, test_descriptions, test_features, tokenizer, max_length)
```

```
Using TensorFlow backend.
Dataset: 6000
Descriptions: train=6000
Vocabulary Size: 7579
Description Length: 34
Dataset: 1000
Descriptions: test=1000
Photos: test=1000
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slice
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
BLEU-1: 0.524012
BLEU-2: 0.274272
BLEU-3: 0.176716
BLEU-4: 0.072820
```

Scores for RSNET50:

```
# photo features
test_features = load_photo_features('/content/gdrive/My Drive/Colab Notebooks/features2.pkl', test)
print('Photos: test=%d' % len(test_features))

# load the model
filename = '/content/gdrive/My Drive/Colab Notebooks/model_10.h5'
model = load_model(filename)
# evaluate model
evaluate_model(model, test_descriptions, test_features, tokenizer, max_length)
```

```
Dataset: 6000
Descriptions: train=6000
Vocabulary Size: 7579
Description Length: 34
Dataset: 1000
Descriptions: test=1000
Photos: test=1000
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slices.py:434: UserWarning: Convert
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
BLEU-1: 0.589046
BLEU-2: 0.254838
BLEU-3: 0.163587
BLEU-4: 0.064970
```

List of contributions in the project

1. **SATISH RELLA:** Data Preprocessing prepare photo data, prepare text data,
2. **TADIKONDA PRUDHVI SIVA SAI KRISHNA:** Evaluate model, Develop deep learning model
3. **ADITYA SAGAM:** Train the deep learning model with progressive learning.

Analysis

What did we do well?

Data Preprocessing and Data Cleaning was efficiently completed. Development and training of the model was successfully implemented.

What could be done better?

We could have used a smaller and refined dataset for processing as the current dataset is huge so in order to provide more efficiency for the model, we need a more accurate dataset.

What is left for future work?

We could use a smaller vocabulary for faster processing times.

Conclusion

We implemented photo caption generator with better results by using the RSNET50 model.

References

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

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. IJCV, 115(3):211–252.
Ankush Gupta and Prashanth Man- nem. From image annotation to image description. In Neural information processing.
Justin Johnson, Andrej Karpathy, and Li Fei-Fei. 2016. Densecap: Fully convolutional localization networks for dense captioning. In IEEE CVPR.

https://www.hindawi.com/journals/cin/2020/3062706/