Automatic Image Caption Generator using CNN and LSTM using PyTorch

Prannoy Noel Bhumana
prannoy.bhumana@student-cs.fr

Shahmir Kazi
shahmir.kazi@student-cs.fr

Abstract

In this project, we systematically analyze a deep neural networks based image caption generation method. With an image as the input, the method can output an English sentence describing the content in the image. Current research[1] is considered, and our research builds upon those topics and proposes a methodology while analyzing three components of the method: convolutional neural network (CNN), recurrent neural network (RNN) and sentence generation. This topic allows for applications in a variety of fields including visual impairment, e- Commerce, social media and content extraction.

1. Introduction

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.

Image Captioning refers to the process of generating textual description from an image – based on the objects and actions in the image. It is a task that involves computer vision and natural language processing concepts to recognize the context of an image and describe them in a natural language like English.[2]

This is because image captions are used in a variety of applications. For example:

1. It can be used to describe images to people who are blind or have low vision and who rely on sounds and texts to describe a scene.
2. In web development, it's good practice to provide a description for any image that appears on the page so that an image can be read or heard as opposed to just seen. This makes web content accessible.
3. It can be used to describe video in real time.
4. It may be used in niche subject areas such as medicine, where it may be used as a guide to human judgement

Instagram partially uses this feature to detect objects in image. Here is a live example

2. Problem definition

Given an image, we want to obtain a sentence that describes what the image consists of. 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.

Hence the problem may be described as:

1. Given labeled images and test images

2. Use Deep Learning and Natural Language Processing techniques (CNN, RNN, Sentence Generation

3. To achieve highly accurate automatic image captioning.

3. Related work

Automatically describing the content of an image is a fundamental problem in artificial intelligence that connects computer vision and natural language processing. Earlier methods first generate annotations (i.e., nouns and adjectives) from images **(Sermanet et al., 2013; Russakovsky et al., 2015)** [7], then generate a sentence from the annotations **(Gupta and Mannem,)** [7].

Donahue et al. **(Donahue et al.,)** [8]developed a recurrent convolutional architecture suitable for large-scale visual learning, and demonstrated the value of the models on three different tasks: video recognition, image description and video description. In these models, long-term dependencies are incorporated into the network state updates and are end-to-end trainable. The limitation is the difficulty of understanding the intermediate result. The LRCN method is further developed to text generation from videos **(Venugopalan et al.,)** [8].

Instead of one architecture for three tasks in LRCN, Vinyals et al. **(Vinyals et al.,)** [6] proposed a neural image caption (NIC) model only for the image caption generation. Combining the GoogLeNet and single layer of LSTM, this model is trained to maximize the likelihood of the target description sentence given the training images. The performance of the model is evaluated qualitatively and quantitatively. This method was ranked first in the **MS COCO Captioning Challenge (2015)** [6] in which the result was judged by humans.

Comparing LRCN with NIC, we find three differences that may indicate the performance differences. First, NIC uses GoogLeNet while LRCN uses VGGNet. Second, NIC inputs visual feature only into the first unit of LSTM while LRCN inputs the visual feature into every LSTM unit. Third, NIC has simpler RNN architecture (single layer LSTM) than LRCN (two factored LSTM layers).

We verified that the mathematical models of LRCN and NIC are exactly the same for image captioning. The performance difference lies in the implementation and LRCN has to trade off between simplicity and generality, as it is designed for three different tasks. Instead of end-to-end learning, Fang et al. **(Fang et al.,)** [7] presented a visual concepts based method.

First, they used multiple instance learning to train visual detectors of words that commonly occur in captions such as nouns, verbs, and adjectives. Then, they trained a language model with a set of over 400,000 image descriptions to capture the statistics of word usage. Finally, they re-ranked caption candidates using sentence-level features and a deep multi-modal similarity model. Their captions have equal or better quality 34% of the time than those written by human beings.

The limitation of the method is that it has more human controlled parameters which make the system less reproducible. We believe the web application **"captionbot" (Microsoft)** [7] is based on this method.

4. Dataset & features

We used the COCO (Common Objects in Context) dataset for training the model. COCO is a commonly used dataset for such tasks since one of the target family for COCO is captions. Every image comes with 5 different captions produced by different humans, hence every caption is slightly (sometimes greatly) different from the other captions for the same image. Here is an example of a data point from the COCO dataset:

The final version of COCO-Stuff, that is presented on this page. It includes all 164,000 images from COCO 2017:

- Train 118,000
- Validation 5,000,
- Test-Dev 20,000,
- Test-Challenge 20,000.

It covers 172 classes: 80 thing classes, 91 stuff classes and 1 class 'unlabeled'. This dataset will form the basis of all upcoming challenges.

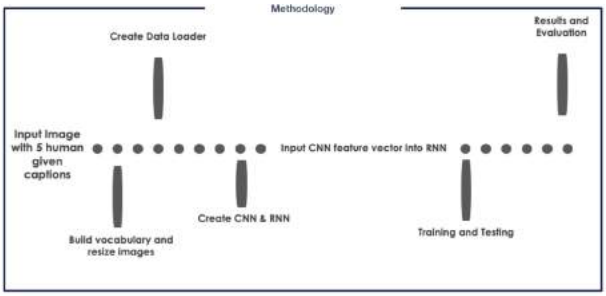
5. Methodology and Solution

The task of image captioning can be divided into two modules logically – one is an image based model – which extracts the features and nuances out of our image, and the other is a language based model – which translates the features and objects given by our image based model to a natural sentence.

For image based model (viz encoder) – we usually relied on a Convolutional Neural Network model. And for our language based model (viz decoder) – we rely on a Recurrent Neural Network [3]. Usually, a pretrained CNN extracts the features from our input image. The feature vector is linearly transformed to have the same dimension as the input dimension of the RNN/LSTM network. This network is trained as a language model on our feature vector.

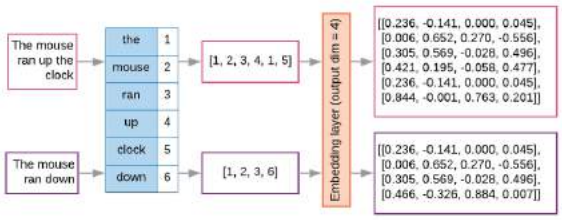
For training our LSTM model, we pre-define our label and target text.

The below figure summarises the method, details of which are explained below:



5.1. Building the vocabulary

First, we iterate through all of the training captions and create a dictionary that maps all unique words to a numerical index. So, every word we come across will have a corresponding integer value that can find in this dictionary. The words in this dictionary are referred to as our vocabulary. The vocabulary typically also includes a few special tokens.



t

We save the vocabulary locally and also print out its length in notebook:

1. Total vocabulary size: 11560
2. Saved the vocabulary wrapper as a local pickle file

5.2. Resizing the images

While loading the dataset, it is important to have uniformity over the dataset, to prepare the data for input into our model. Simultaneously, size of the images must also be taken into account, Hence the images are resized through the function `resize_images`.

Through this we resize the images in `image_dir` and save into `output_dir` by following `image_size` = [256,256] .

5.3. Creating the Data Loader

We created a custom data loader after performing preprocessing to the dataset. We tokenized the captions and stored all of the words in a dictinoary with unique ids and also added ¡start¿ and ¡end¿ tags for the whole caption. We also collated the data which creates mini-batch tensors from the list of tuples (image, caption). This is used in data loader which has a batch size of 128 and shuffles the images. [5]

5.4. Creating CNN & RNN

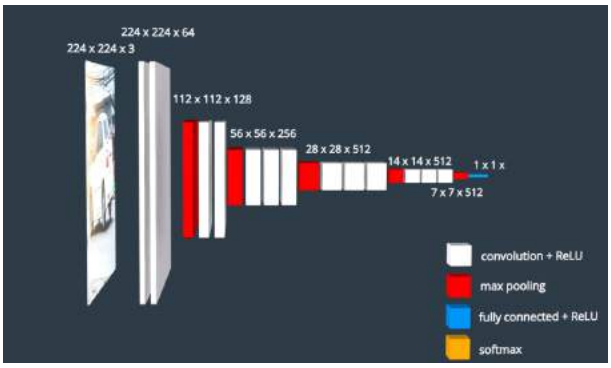
A convolutional neural network can be used to create a dense feature vector. This dense vector, also called an embedding, can be used as feature input into other algorithms or networks. This embedding becomes a dense representation of the image and will be used as the initial state of the LSTM.

CNN features have the potential to describe the image. To leverage this potential to natural language, a usual method is to extract sequential information and convert them into language. In most recent image captioning works, they extract feature map from top layers of CNN, pass them to some form of RNN and then use a softmax to get the score of the words at every step [2].

Now our goal is, in addition to captioning, also recognize the objects in the image to which every word refers to. In other word, we want **position information**. Thus we need to extract feature from a lower level of CNN, encode them into a vector which is dominated by the feature vector corresponding to the object the word wants to describe, and pass them into RNN [2].

We used pre-trained rest-net 152 architecture in the encoder. We have an image and we pass it through a slightly modified densely connected neural network to

obtain a 1024-dimensional output vector. It is 8x deeper than VGG nets but still having lower complexity. The architecture and layers of restnet are shown below.

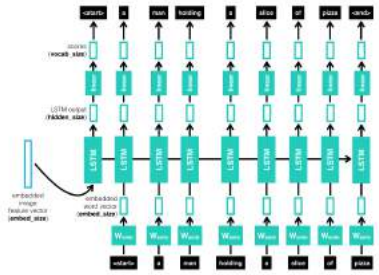


An embedding layer is also added so that it transforms each word in a caption into a vector of a desired consistent shape. At this point, we cannot directly feed words into an LSTM and expect it to be able to train or produce the correct output.

These words first must be turned into a numerical representation so that a network can use normal loss functions and optimizers to calculate how "close" a predicted word and ground truth word (from a known, training caption) are? So, we typically turn a sequence of words into a sequence of numerical values; a vector of numbers where each number maps to a specific word in our vocabulary.

Decoder part of the architecture is where thing can get messy and difficult to debug and understand, but let's give it a shot. We could use Gated Recurrent Unit or Long Short-Term Memory unit, in our particular case, we have used the latter. It is crucial to mention that there is still a lot of debate going on which recurrent cell is better GRU or LSTM. Both have shown to be working very well in numerous applications with mixed performance gains [3]. An LSTM cell has Long and Short term Memory as the name implies. A high level anatomy of the LSTM cell is as follows:

1. LSTM cell has an input for a data point (at time t=n)
2. LSTM cell has an input for a cell state (previous cell state)
3. LSTM cell has an input for a hidden state (previous hidden state)
4. LSTM cell has an output for a cell state (current cell state)
5. LSTM cell has an output for a hidden state (current hidden state)
6. LSTM cell's data output is the LSTM cell's hidden state output



For every step in the sequence we use exactly same LSTM (or GRU) cell, so the goal of the cell under optimization is to find the right set of weights to accommodate the whole dictionary of words (characters in char-to-char models). This means that for every word in our sentence (which is a sequence) we are going to feed the word as input and get some output which is typically a probability distribution over the whole dictionary of words. This way we can obtain the word that the model thinks fits the most given the previous word.

5.5. Training and Testing

The Decoder will be made of LSTM cells which is good for remembering the lengthy sequences of words. Each LSTM cell is expecting to see the same shape of the input vector at each time-step. The very first cell is connected to the output feature vector of the CNN encoder. The input to the RNN for all future time steps will be the individual words of the training caption. So, at the start of training, we have some input from our CNN, and LSTM cell with initial state. Now the RNN has two responsibilities:

1. To Remember spatial information from the input feature vector.
2. To Predict the next word.

We know that the very first word it produces should always be the **start** token and the next word should be those in the training caption. At every time step, we look at the current caption word as input and combine it with the hidden state of the LSTM cell to produce an output.

This output is then passed to the fully connected layer that produces a distribution that represents the most likely next word. We feed the next word in the caption to the

network and so on until we reach the **end** token. The hidden state of an LSTM is a function of the input token to the LSTM and the previous state also referred to as the recurrence function.

The recurrence function is defined by weights and during the training process, this model uses back-propagation to update these weights until the LSTM cells learn to produce the correct next word in the caption given the current input word.

As with most models, we can also take advantage of batching the training data. The model updates its weights after each training batch with the batch size is the number of image caption pairs sent through the network during a single training step. Once the model is trained, it will have learned from many image caption pairs and should be able to generate captions for new unseen image data.

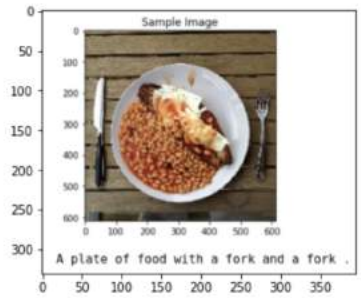
5.6. Evaluation and Conclusion

For evaluating the model, we tried to use BLEU score. But we were not able to implement it due to time factor. But we tried our best to to put a score on the predicted output. Most of the times, the model performed very well, but sometimes, it is not able to capture all items in the image. This is because we trained on 25% of the dataset. We even performed the training on google cloud platform [4].

We uploaded our resized images and json files which has captions as labels. For the model to train on whole dataset, takes a lot of computation power and time to look after. But our training was on almost 5GB of images. We think, that's sufficiently good enough to predict some sentences. The results are shown below:

```
test_image('/home/bpanoel12/images/test_images/7.png')
```

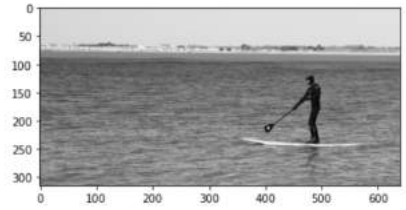
```
<start> a close up of a plate of food with a fork . <end>
```



Here we can see that model performed very well and was able to capture all the items in images without a mistake. We can also see the label at the bottom which is almost similar to the predicted sentence.

```
test_image('/home/bpanoel12/images/test_images/000000001490.jpg')
```

```
<start> a man riding a wave on a surfboard . <end>
```

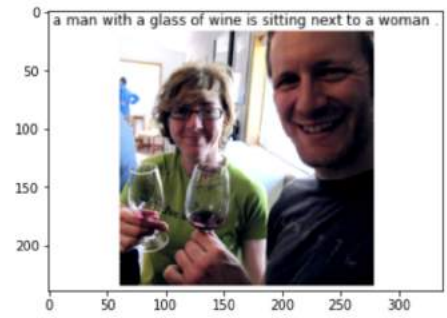


```
test_image('/home/bpanoel12/images/test_images/000000000885.jpg')
```

```
<start> a man is playing tennis on a tennis court . <end>
```


```
test_image('/home/bpanoel12/images/test_images/1.png')
```

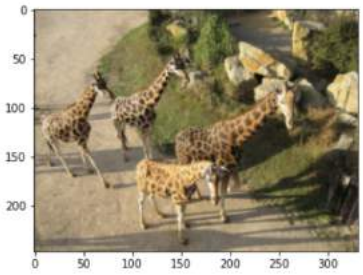
```
<start> a man is holding a hot dog on a table . <end>
```



This prediction is the man holding a hot-dog on table. But apparently, he's holding a glass of wine with a woman besides him. Here, we can see that model was able to capture that man is holding a object. But, it predicted a wrong object. We also have another example below. It almost holds the same case. We can improve this by training on our whole dataset.

```
test_image('/home/bpanoel12/example.png')
```

```
<start> a giraffe is standing in the grass with a tree . <end>
```



6. References

[1] Lakshminarasimhan Srinivasan , Dinesh Sreekanthan , Amutha A.L. International Journal of Applied Engineering Research. Link

[2] R. Subash, Dr.R.Jebakumar, Yash Kamdar, Nishit Bhatt International Conference on Physics and Photonics Processes in Nano Sciences. Link

[3] Jia-Yu Pan , Hyung-Jeong Yang, Pinar Duygulu and Christos Faloutsos. Department of Computer Engineering, Bilkent University, Ankara, Turkey. Link

[4] Shuang Bai and Shan An School of Electronic and Information Engineering, Beijing Jiaotong University, No.3 Shang Yuan Cun, Hai Dian District, Beijing, China. Link

[5] Annika Lindh, Robert J. Ross, Abhijit Mahalunkar ADAPT Centre, Dublin, Ireland Generating Diverse and Meaningful Captions. Link

[6] Andrej Karpathy, Fei-Fei Li. Automated Image Captioning with ConvNets and Recurrent Nets. Link

[7] Jianhui Chen, Wenqiang Dong, Minchen Li. Image Caption Generator Based On Deep Neural Networks. Link

[8] Ye Tian, Tianlun Li, Stanford Deep learning projects Link

Acknowledgement

We would like to thank Yunjey Choi for useful information on the ideas behind this project, and the write up.