Udacity Machine Learning Engineer Nanodegree

Capstone Project

Image Captioning

Sana Shah

April 24th, 2020

Definition

Project Overview

Generating a description of an image is called image captioning. Image captioning requires to recognize the important objects, their attributes and their relationships in an image. It also needs to generate syntactically and semantically correct sentences. Deep learning-based techniques and machine learning are capable of handling the complexities and challenges of image captioning.

Everyday, we encounter a large number of images from various sources such as the internet, news articles, document diagrams and advertisements. These sources contain images that viewers would have to interpret themselves. Most images do not have a description, but the human eye can easily understand them without their detailed captions. However, machine needs to interpret some form of image captions if humans need automatic image captions from it, which is what this project is supposed to do.

Image captioning can be used in a variety of applications, here are a few examples:

- In web development, it is a good practice to label the images, but it can be a tedious task for a human. Automatically generated captions can be very helpful.
- Captions can also be used to describe a video in real time, and provide subtitles.

- Would be greatly helpful for visually impaired people. Can be used in the applications that are aimed to help visually impaired people.

There are a number of scenarios in which we can find great use of automatic image captioning. By designing an algorithm which can automatically generate these captions using computer vision, we can take great advantage. This project is inspired by a project that I did in another Udacity's nanodegree.

Problem Statement

Two problems will be tackled in this project, first will be recognizing an imagine that is considered a computer vision problem. Secondly, after recognizing an image, we would be required to generate some text to caption that image, which is a natural language procession problem.
So, in conclusion we need to design a machine learning model that can firstly detect what is happening in the image, and then provide the appropriate captions. It is likely to use two algorithms in this model, i.e a CCN (Convolutional Neural Network) to classify the image and an RNN (Recurrent Neural Network) to generate the text.

Metrics

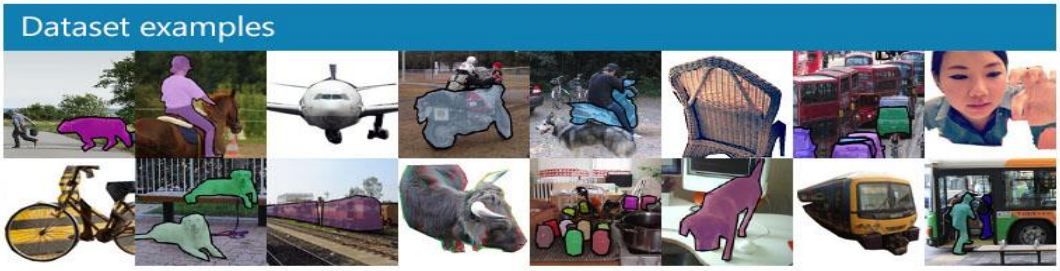
In order to calculate the accuracy, I will write a script or will be using one that can be online to calculate the BLEU score of my model.
BLEU, or the Bilingual Evaluation Understudy, is a **score** for comparing a candidate translation of text to one or more reference translations. Although developed for translation, it can be used to evaluate text generated for a suite of natural language processing tasks. BLEU uses a modified form of precision to compare a candidate translation against multiple reference translations. The metric modifies simple precision since machine translation systems have been known to generate more words than are in a reference text. I have not mentioned this in my proposal as I found this method after researching various papers.

Analysis

Data Visualization and Exploration

The dataset that I have used for this project is Microsoft's COCO (Common objects in context) dataset. COCO was an initiative to collect natural images, the images that reflect everyday scene and provides contextual information. In everyday scene, multiple objects can be found in the same image and each should be labeled as a different object and segmented properly. COCO dataset provides the labeling and segmentation of the objects in the images. A machine learning practitioner can take advantage of the labeled and segmented images to create a better performing object detection model. Each image in the dataset has a total of five related captions. Here is an example,

The dataset has 2,500,000 labeled instances in 328,000 images.
The COCO dataset is an excellent choice as it comes with 80 classes, 80,000 training images and 40,000 validation images.



I have used the basic python libraries in order to plot the image in the notebook provided with the submission. Below is the example:

Image Plotting

Now, in the cell below I have plotted an image, with its corresponding lables!

```
import numpy as np
import skimage.io as io
import matplotlib.pyplot as plt
%matplotlib inline

# pick a random image and obtain the corresponding URL
ann_id = np.random.choice(ids)
img_id = coco.anns[ann_id]['image_id']
img = coco.loadImgs(img_id)[0]
url = img['coco_url']

# print URL and visualize corresponding image
print(url)
I = io.imread(url)
plt.axis('off')
plt.imshow(I)
plt.show()

# load and display captions
annIds = coco_caps.getAnnIds(imgIds=img['id']);
anns = coco_caps.loadAnns(annIds)
coco_caps.showAnns(anns)
```

```
http://images.cocodataset.org/val2014/COCO_val2014_000000533532.jpg
```

```
there are many bags and luggages on the floor
A woman standing around with several pieces of luggage on the floor
Two female travelers wait among piles of luggage.
Two women are in the waiting area with many bags of luggage.
People standing near luggage placed on the floor.
```

As displayed in the output above, each image from the dataset comes with 5 labels that define what is happening in that image, exactly what we need to caption the image! These images are great to be used as a dataset for training.

Data Loading

The data is loaded by using a data loaded function, which has the following parameters:

1. **transform** - an image transform specifying how to pre-process the images and convert them to PyTorch tensors before using them as input to the CNN encoder.

2. **mode** - one of 'train' (loads the training data in batches) or 'test' (for the test data).
3. **batch_size** - determines the batch size. When training the model, this is number of image-caption pairs used to amend the model weights in each training step.
4. **vocab_threshold** - the total number of times that a word must appear in the in the training captions before it is used as part of the vocabulary. Words that have fewer than vocab_threshold occurrences in the training captions are considered unknown words.
5. **vocab_from_file** - a Boolean that decides whether to load the vocabulary from file.

Algorithm and Techniques

The Algorithms

The way I will make my model do image captioning will have two main components, the CNN (Convolutional Neural Network) and RNN (Recurrent Neural Network). It will also have an encoder and a decoder.

Convolutional Neural Network

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. In neural networks, Convolutional neural network is one of the main categories to do images recognition, images classifications. Objects detections, recognition faces etc., are some of the areas where CNNs are widely used.

Recurrent Neural Network

Recurrent Neural Network (RNN) are a type of Neural Network where the output from previous step are fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are

required and hence there is a need to remember the previous words. In our case, that would be to generate a sentence. In this project, I will be using LSTM's (Long Short Term Memory) which are a type of RNN. LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs, hidden Markov models and other sequence learning methods in numerous applications.

Captioning is all about combining these two to use their most powerful attributes in order to produce captions with high accuracy. I will establish this also with the help of encoders and decoders, which is explained later in this report.

Benchmark

I have read the contents of this research paper, which gave me an idea about my initial aim and benchmark, as I have decided to calculate the BLEU score of my model, I will aim for a score of slightly higher than 0.1. I have chosen this based on limited resources at my house and as I trained the model on udacity's workspace, I did not have much GPU limit left. I compared this value against other model such as GAN in the research paper.

Methodology

Data Pre-processing

Image Pre-Processing

Image pre-processing is relatively straightforward (from the __getitem__ method in the CoCoDataset class):

```
# Convert image to tensor and pre-process using transform
image = Image.open(os.path.join(self.img_folder, path)).convert('RGB')
image = self.transform(image)
```

After loading the image in the training folder with name path, the image is pre-processed using the same transform (transform_train) that was supplied when instantiating the data loader.

Caption Pre-Processing

The captions also need to be pre-processed and prepped for training. In this example, for generating captions, we are aiming to create a model that predicts the next token of a sentence from previous tokens, so we turn the caption associated with any image into a list of tokenized words, before casting it to a PyTorch tensor that we can use to train the network.

To understand in more detail how COCO captions are pre-processed, we'll first need to take a look at the vocab instance variable of the CoCoDataset class. The code snippet below is pulled from the __init__ method of the CoCoDataset class:

```
    def __init__(self, transform, mode, batch_size, vocab_threshold, v
    ocab_file, start_word,

            end_word, unk_word, annotations_file, vocab_from_file, img
    _folder):

            ...

            self.vocab = Vocabulary(vocab_threshold, vocab_file, start
    _word,

  end_word, unk_word, annotations_file, vocab_from_file)

        ...
```

From the code snippet above, we can see that data_loader.dataset.vocab is an instance of the Vocabulary class from vocabulary.py. We use this instance to pre-process the COCO captions (from the __getitem__ method in the CoCoDataset class):

```
    # Convert caption to tensor of word ids.

    tokens = nltk.tokenize.word_tokenize(str(caption).lower())# line 1

    caption = []                                              # line 2

    caption.append(self.vocab(self.vocab.start_word))         # line 3
```

```
    caption.extend([self.vocab(token) for token in tokens])    # line 4

    caption.append(self.vocab(self.vocab.end_word))            # line 5

    caption = torch.Tensor(caption).long()                     # line 6

sample_caption = 'A person doing a trick on a rail while riding a skat
eboard.'
```

In **line 1** of the code snippet, every letter in the caption is converted to lowerca se, and the nltk.tokenize.word_tokenize function is used to obtain a list of strin g-valued tokens. Run the next code cell to visualize the effect on sample_capti on.

```
import nltk

sample_tokens = nltk.tokenize.word_tokenize(str(sample_caption).lower())
print(sample_tokens)
```

```
['a', 'person', 'doing', 'a', 'trick', 'on', 'a', 'rail', 'while', 'riding', 'a', 'skateboard', '.']
```

In **line 2** and **line 3** we initialize an empty list and append an integer to mark th e start of a caption. The paper that you are encouraged to implement uses a spe cial start word (and a special end word, which we'll examine below) to mark th e beginning (and end) of a caption.

This special start word ("<start>") is decided when instantiating the data loader and is passed as a parameter (start_word). You are **required** to keep this para meter at its default value (start_word="<start>").

As you will see below, the integer 0 is always used to mark the start of a

caption.

```
sample_caption = []

start_word = data_loader.dataset.vocab.start_word
print('Special start word:', start_word)
sample_caption.append(data_loader.dataset.vocab(start_word))
print(sample_caption)
```

```
Special start word: <start>
[0]
```

In **line 4**, we continue the list by adding integers that correspond to each of the tokens in the caption.

```
sample_caption.extend([data_loader.dataset.vocab(token) for token in sample_tokens])
print(sample_caption)
```

```
[0, 3, 98, 754, 3, 396, 39, 3, 1009, 207, 139, 3, 753, 18]
```

In **line 5**, we append a final integer to mark the end of the caption.

Identical to the case of the special start word (above), the special end word ("<end>") is decided when instantiating the data loader and is passed as a parameter (end_word).

```
end_word = data_loader.dataset.vocab.end_word
print('Special end word:', end_word)

sample_caption.append(data_loader.dataset.vocab(end_word))
print(sample_caption)
```

```
Special end word: <end>
[0, 3, 98, 754, 3, 396, 39, 3, 1009, 207, 139, 3, 753, 18, 1]
```

Finally, in **line 6**, we convert the list of integers to a PyTorch tensor and cast it to long type. You can read more about the different types of PyTorch tensors on the website.

```
import torch

sample_caption = torch.Tensor(sample_caption).long()
print(sample_caption)
```

```
tensor([   0,    3,   98,  754,    3,  396,   39,    3, 1009,
         207,  139,    3,  753,   18,    1])
```

And that's it! In summary, any caption is converted to a list of tokens, with *special* start and end tokens marking the beginning and end of the sentence:

```
[<start>, 'a', 'person', 'doing', 'a', 'trick', 'while', 'riding', 'a'
, 'skateboard', '.', <end>]
```

This list of tokens is then turned into a list of integers, where every distinct word in the vocabulary has an associated integer value:

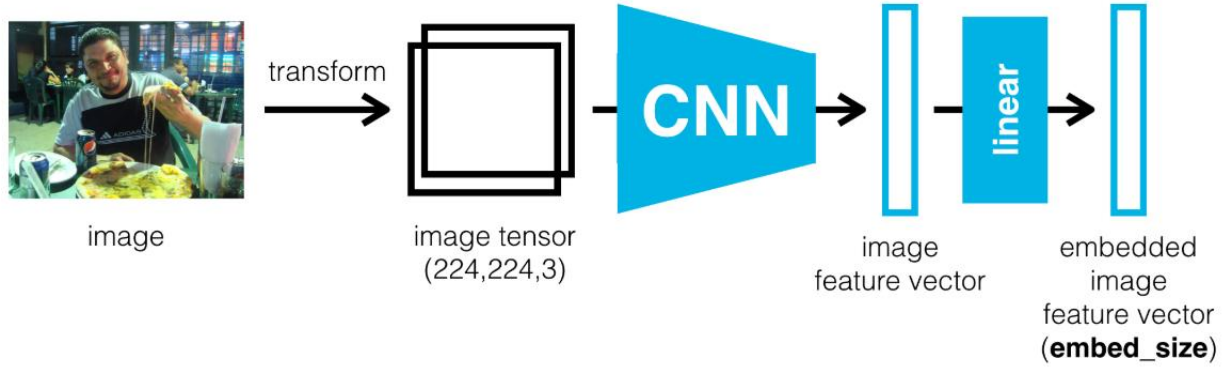
```
[0, 3, 98, 754, 3, 396, 207, 139, 3, 753, 18, 1]
```

Implementation

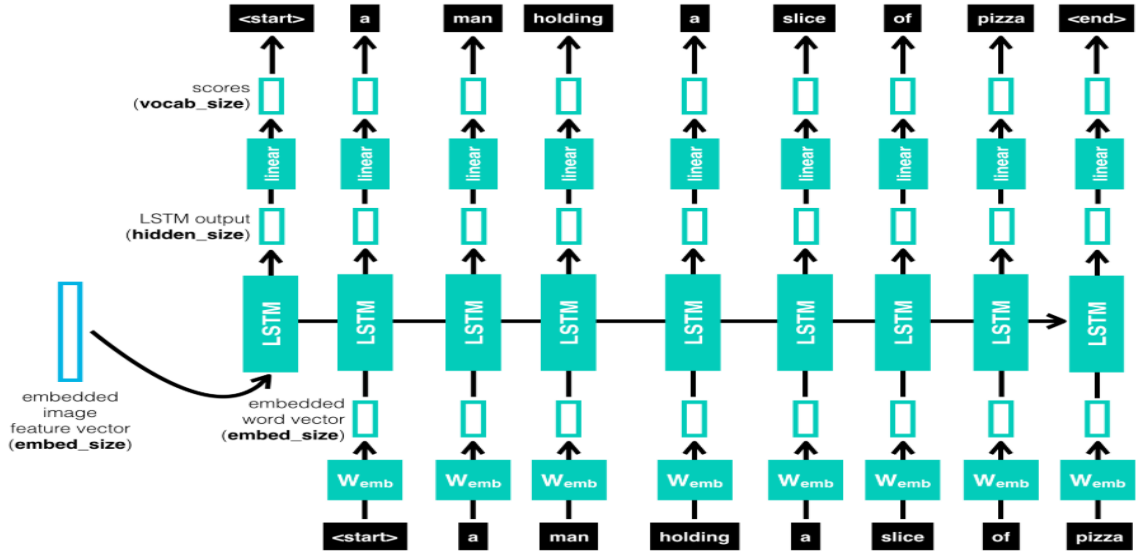
After the preprocessing step, the data is provided to an encoder and a decoder.

Encoder and the Decoder

The encoder that we provide us a way that uses the pre-trained ResNet-50 architecture (with the final fully-connected layer removed) to extract features from a batch of pre-processed images. The output is then flattened to a vector, before being passed through a Linear layer to transform the feature vector to have the same size as the word embedding.



The decoder RNN consists of an embedding layer, an LSTM with one hidden layer and a finaly fully connected layer that maps the hidden state output to the vocab size.



The Training Setup

My training model has the following parameters.

- batch_size - the batch size of each training batch. It is the number of image-caption pairs used to amend the model weights in each training step.
- vocab_threshold - the minimum word count threshold. Note that a larger threshold will result in a smaller vocabulary, whereas a smaller threshold will include rarer words and result in a larger vocabulary.
- vocab_from_file - a Boolean that decides whether to load the vocabulary from file.
- embed_size - the dimensionality of the image and word embeddings.
- hidden_size - the number of features in the hidden state of the RNN decoder.
- num_epochs - the number of epochs to train the model.
- save_every - determines how often to save the model weights.
- print_every - determines how often to print the batch loss to the Jupyter notebook while training.
- log_file - the name of the text file containing - for every step - how the loss and perplexity evolved during training.

Which I have provided the following values,

```
batch_size = 128         # batch size
vocab_threshold = 5        # minimum word count threshold
vocab_from_file = True   # if True, load existing vocab file
embed_size = 512           # dimensionality of image and word embeddings
hidden_size = 512          # number of features in hidden state of the RNN decoder
num_epochs = 5             # number of training epochs
save_every = 1             # determines frequency of saving model weights
print_every = 100          # determines window for printing average loss
log_file = 'training_log.txt'       # name of file with saved training loss and perplexity
```

Refinement

For the refinement, I tested various different parameters for training. To select the embed and hidden size (=512) I used this paper. Regarding the batch size (=128) was a based-on suggestion of Udacity materials for ideal choice to start with. The weight initialization was select assuming a gaussian error distributio

n as well a input normalized data. I did not want to tweak the learning rate so I decided to start with an ADAM optimizer which is regarded as a good first choice because through adaptive learning rates it avoids being stuck in local minima.

As this is a very advance topic and I'm currently a beginner, I could not do mu ch refinements other than tweaking and testing the parameters. The number of epochs affected my result. At first, I trained the model on around 3 epochs and got a BLEU score of 0.08 which was lower than what I needed. After running the training job for about 9 hours with 5 epochs, I finally got a BLEU score of 0.16.

Results

Model Evaluation and Validation

For the validation, I have decided to use the BLEU score. The score was devel oped for evaluating the predictions made by automatic machine translation syst ems. It is not perfect, but does offer 5 compelling benefits:

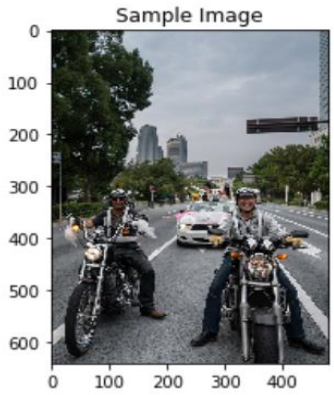
- It is quick and inexpensive to calculate.
- It is easy to understand.
- It is language independent.
- It correlates highly with human evaluation.
- It has been widely adopted.

Validating the model involves creating a json file such as this one containing the model's predicted captions for the validation images. Then, I will use a script that can be found online to calculate the BLEU score of my model.

The model was tested on the test images from the COCO dataset. The images are first loaded with the help of the data loader. Then they are pre-processed and passed through the encoder and decoder. Finally, in the end the images are passed through the model and the following results were obtained. Below are the examples of some of the output.

Sample Output:

a bathroom with a toilet and a sink.

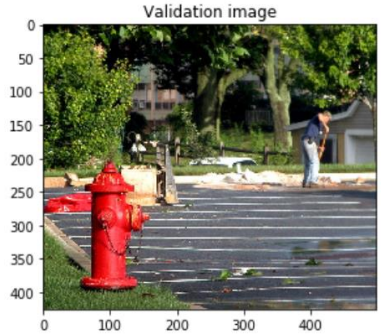


a man riding a motorcycle down a street.

Justification

As you can observe from the output, the result is satisfactory and it goes long well with the benchmark value which I set which was to be higher than 0.1

It can be seen from the image below that the output provided by this algorithm satisfies the condition.




```
Original caption:  A red fire hydrant stands in a parking lot while a man holding the handle of a tool is in the distance.
Generated caption:  a red fire hydrant sitting on the side of a road.
BLEU_1: Cumulative 1-gram: 0.166667
```