Captioning ImageNet

A report submitted for COMP8755

Supervisor: Prof. Stephen Gould

Australian National University

Songtuan Lin

U6162630
Image captioning plays an important role in various Computer Vision research areas, however, the available datasets which can be used to train image captioning model are limited. Motivated by this situation, this project aims to expand the available datasets by captioning images within ImageNet. Specifically, we used a state-of-the-art method to generate captions for each image and applied constrained beam search to augment them. By doing this, we created a new dataset which can be used to improve existing image captioning models. Furthermore, since we construct this new dataset in a semi-autonomous way, we cannot guarantee that each data in this dataset has good quality. Therefore, we also conduct some experiments to find the method which can filter the bad cases in our dataset.
Acknowledgments

Taking COMP8755 gives me a great opportunity to conduct a research project and learn from world-leading professors. Specifically, by doing this project, I gained a deeper understanding of various deep-learning technologies and learned the process of conducting a research project. Moreover, I sincerely appreciate my supervisor Professor Stephen Gould, who gives me a lot of guidance and help in this project. In the same time, I would like to thank people who spend time to assess this project, particularly Dr. Ramesh Sankaranarayana and Prof. Weifa Liang.
# Contents

1 Introduction ................................................................. 7
  1.1 Motivation ..................................................................... 7
  1.2 Contributions ............................................................... 7
  1.3 Report Outline .............................................................. 7

2 Background ........................................................................ 9
  2.1 Image Captioning ............................................................ 9
    2.1.1 Definition ................................................................. 9
    2.1.2 History .................................................................... 11
  2.2 ImageNet ........................................................................ 11
    2.2.1 WordNet .................................................................. 11
    2.2.2 ImageNet .................................................................. 12
  2.3 Project Motivation Revisit ................................................ 12

3 Model .................................................................................. 14
  3.1 Encoder .......................................................................... 14
    3.1.1 Convolution ............................................................... 15
    3.1.2 Convolution Neural Network ........................................ 16
    3.1.3 Region Proposal Network ............................................ 17
  3.2 Decoder ........................................................................... 19
    3.2.1 Long-short Term Memory .............................................. 19
    3.2.2 Word Embedding ....................................................... 21
    3.2.3 Math Behind LSTM .................................................... 22
  3.3 Attention ......................................................................... 22
    3.3.1 Naive Attention ......................................................... 22
    3.3.2 Button-Up-Top-Down Attention ................................... 23
  3.4 Complete Model ............................................................... 23

4 Decoding Algorithm ............................................................. 25
  4.1 Greedy ........................................................................... 25
  4.2 Beam Search .................................................................... 26
  4.3 Constrained Beam Search .................................................. 27
    4.3.1 Constrained Beam Search as Finite Automata ................ 28

5 Implementation .................................................................... 30
  5.1 Pipeline .......................................................................... 30
  5.2 Tag Pre-process ............................................................... 31
List of Figures

2.1 Sample structure of WordNet .................................................. 12
3.1 Convolution Neural Network .................................................... 17
3.2 Feature map in RPN ............................................................... 18
3.3 LSTM cell ................................................................. 19
3.4 LSTM ................................................................. 19
3.5 LSTM in image captioning ..................................................... 20
3.6 BUTD cell ................................................................. 23
3.7 BUTD ................................................................. 24
4.1 Beam Search ................................................................. 26
5.1 Pipeline of Project ............................................................. 30
5.2 The pattern of finite automata used in project ......................... 33
5.3 Sample captions for synset n01496331 .................................... 35
5.4 Sample captions for synset n01440764 .................................... 35
5.5 Sample captions for synset n01667778 .................................... 35
6.1 Example of captions generated with and without using constrained beam search . 38
6.2 Comparison of log-probabilities ............................................. 39
6.3 Examples that contradict the initial assumption ......................... 40
6.4 Result of second experiment .................................................. 40
7.1 NFA ................................................................. 43
7.2 DFA ................................................................. 43
List of Tables

4.1 Example of greedy algorithm .......................................................... 25
Chapter 1

Introduction

1.1 Motivation

Image captioning has been identified as an important component in many Computer Vision (CV) tasks, e.g. Visual Question Answering (VQA) and scene understanding. However, current resources which can be used to train and construct caption generator only include Microsoft COCO (Lin et al., 2015) and Flickr (Plummer et al., 2017) dataset. Motivated by this, we seek to extent these datasets in a semi-autonomous way and provide more annotations which may be useful for building quality caption generator. Specifically, we will caption ImageNet (Deng et al., 2009) which contains about 14 millions images and each of them has an associated word tag to describe the entity contained in it (but not a full description, i.e. caption). In order to fully use these tags and improve the quality of generated captions, our work will apply Constrained Beam Search algorithm (Anderson et al., 2017a), which can force specific words (image tags in this case) to appear somewhere in the generated captions.

1.2 Contributions

The main contributions of this project includes the following two aspects:

- We used state-of-the-art approach to caption the images within ImageNet and we hope this can extend the image captioning datasets used in our community. Moreover, we applied constrained beam search algorithm in our caption generation process and in many cases, this algorithm can improve the quality of captions.

- We performed some fundamental analysis over generated captions, which can lead us to find some clues to further filter the good and bad captions.

1.3 Report Outline

This report describes the approaches used in this project along with the project’s outcomes. Specifically, the content of each chapter is summarized as below:

- In chapter 2, we will provide some essential background knowledge, which includes the formal definition of image captioning task and the structure of ImageNet dataset.
Chapter 3 and 4 give the explanation of the approach we used to caption images. Specifically, Chapter 3 introduces a state-of-art deep-neural-network based approach which used to estimate the probability distribution of captions. Chapter 4 then presents the decoding algorithm which used to generate captions based on the estimated distribution.

In chapter 5, we describe the detail of our implementation in this project and present some example outcomes.

The analysis of our outcomes and the associate conclusions will be presented in Chapter 6.

Chapter 7 is the conclusion of this report, in which, we also mention some future work of this project.
Chapter 2

Background

2.1 Image Captioning

2.1.1 Definition

Image captioning refers to the task that generating description of images. These descriptions usually present the content of images and hence, image captioning also be treated as a bridge which connect Computer Vision (CV) and Natural Language Processing (NLP).

However, generating a real natural language description for an image is a complicate task. Therefore, what we actually do in image captioning is generating a natural-language-like description, i.e. generate the description by using a language similar to natural language. This language is called the language of image captioning and can be defined by using some concepts of formal language. Specifically, we first introduce the concept of alphabet, which is a finite and non-empty set of symbols. The symbol, which can be every thing, e.g. digital numbers or English word can both be symbol. Some common alphabet may be:

- A binary alphabet \( \Sigma \) which contains two symbols: 0 and 1, i.e. \( \Sigma = \{0, 1\} \).

- A vocabulary \( \mathcal{V} \) which contains some English words as symbols, e.g. \( \mathcal{V} = \{\text{cat, dog, \cdots}\} \).

By giving the concept of alphabet, we can then introduce the definition of string. A string is a sequence of symbols chosen from an alphabet. For example, a sequence of symbols chosen from binary alphabet form a string and may look like this: 0 1 0 0 1. Following this idea, we can also define the string used in the language of image captioning:

A string in the language of image captioning is a sequence of symbols chosen from a vocabulary \( \mathcal{V} \).

With this definition, it is straightforward to define the language of image captioning:

By giving a specific vocabulary \( \mathcal{V} \), the language of image captioning refers to a set of all possible strings. Each of these strings is a sequence of symbols chosen from \( \mathcal{V} \).

Following these definitions, the target of image captioning task is actually generating a string to describe the given image. Particularly, the vocabulary used in the task is a list of English words
plus two special symbols: **SOS** and **EOS**, which represent 'start of sentence' and 'end of sentence' respectively. One example of image captioning may look like this:

![SOS a close up of a small animal on a road EOS](image)

The most important question for the image captioning task is how to generate the string for a given image. Indeed, it is reasonable for us to think that by giving an **arbitrarily** image \(I\), the caption string \(S\) associate with this image always satisfies a specific probability distribution. More precisely, we can think each word in \(S\) as a random variable whose value can be any symbol in vocabulary \(V\), \(i.e.\) the \(i\)th word \(w_i\) in \(S\) is a random variable and \(w_i \in V\). Therefore, the entire string along with the image \(I\) follows a probability distribution given by:

\[
P(S, I) = P(w_1 w_2 \cdots w_n | I; \Theta)
\]  

(2.1)

In which, \(\Theta\) is a set of parameters and indicates the distribution is parameterized by it. From the prospective of frequency probability, there always exists a perfect value for each element of \(\Theta\) and we just do not know yet. However, once we have estimated the approximate value for them, we can then model the probability distribution of the string and pick suitable symbol from vocabulary \(V\) for each \(w_i\). As a result, the most challenge question becomes how to estimate \(\Theta\). The approach is the classical probability inference method, \(i.e.\) estimate the parameters from a large amount of input data. Being more specific, in the case of image captioning, we input a large number of images which have been captioned by human (called **training data**) and estimate \(\Theta\) from them. This process of estimating parameters \(\Theta\) is called **training** and the distribution \(P(S, I)\) is called the **model**. In this report, the detail about how training process work will be omitted since we will use a pre-trained model (Singh et al., 2020), \(i.e.\) a probability distribution \(P(S, I)\) with estimated parameters \(\Theta\), in this project. However, some detail about what the parameters \(\Theta\) are and what is the actual formula of the model \(P(S, I)\) will be introduced in Chapter 3.

By having the estimated model \(P(S, I)\), the string can be finalized by selecting symbol for each variable \(w_i\). Ideally, the best string \(S^*\) will be the one who has the maximum probability according to the model:

\[
S^* = \arg \max_{w_1 w_2 \cdots w_n} P(w_1 w_2 \cdots w_n | I)
\]

However, in practice, this is usually infeasible and hence, efficient algorithm must be applied to choses symbol for each \(w_i\) and to approximate \(\max P(S, I)\). This part will be formally introduced in Chapter 4.

Following the discussion above, we can formally define the image captioning task as follow:
By given a target image $\mathcal{I}$, find the value of $w_1 w_2 \cdots w_n$ which can maximize the probability $\mathcal{P}(w_1 w_2 \cdots w_n | \mathcal{I})$, or it’s equivalent $\log \mathcal{P}(w_1 w_2 \cdots w_n | \mathcal{I})$.

Mathematically, we can rewrite this definition to an equivalent form:

$$
\max_{w_1 w_2 \cdots w_n} \log \mathcal{P}(w_1 w_2 \cdots w_n | \mathcal{I}) \tag{2.2}
$$

Particularly, in order to distinguish the random variable $w_i$ from the symbol in vocabulary, we will call the symbol in vocabulary as *token* in the rest of this report. Therefore, when we say something like the $i$th word in caption in following chapters, it actually refer to the random variable $w_i$ but not the token in vocabulary.

### 2.1.2 History

The research of image captioning can back to 20th century. The early task was focus on generating natural language description from video materials. After then, researchers start to pay attention to images. The widely used approaches in that early stage include several image parse algorithms, e.g. And-or graph (AoG) (Yao et al., 2010), combine with template-based text generation algorithm. However, the drawback of this family of algorithm is obvious: They are limit to specific scene, i.e. they cannot be widely used.

The breakthrough happened around 2010, when researchers start applying deep learning model to image captioning task. Some significant works in this stage include Xu et al. (2015), Anderson et al. (2017b). Particularly, the *encoder-decoder* architecture (Sutskever et al., 2014) which initially target for NLP task was successfully been transplanted to image captioning and achieved outstanding result. More detail about these deep learning based approaches will be introduced in next chapter and our experiments will also build upon these models.

### 2.2 ImageNet

Since our target is to caption images contained within ImageNet, it is worth to spend some time to introduce the structure of this dataset. However, in order to better describe ImageNet, we first introduce another dataset, WordNet, which has a strong connection with ImageNet.

#### 2.2.1 WordNet

Generally, WordNet is a large collection of words. However, unlike normal dictionary, WordNet collect the synonyms together to form a set called *synset* (short for synonyms set) and assign a unique ID named *WordNet ID* to it. For example, a synset along with the corresponding WordNet ID take the format:

\[
\text{WordNet ID} \downarrow \text{n02084071} : \{\text{dog, domestic dog}\}
\]

Additionally, WordNet also maintain an ISA (is a) relation, or hyponymy relation, between associated synsets, i.e., WordNet links a synset to a more generic one. For example, synsets \{*pooch*, *doggie*\} (WordNet ID: n02084732) and \{*hunting dog*\} (Wordnet ID: n02087122) are both a subtype of synset \{*dog, domestic dog*\} (WordNet ID: n02084071) and hence form an ISA relation. As a result, WordNet links these three synsets as follow:
It is not hard to notice that by maintaining this ISA relation, synsets in WordNet form a directed graph, or more precisely, tree structure. As a result, with a given synset or WordNet ID, we can easily find its ancestor by using graph algorithm like deep first search. This idea will be very useful in our project since a given WordNet ID along with it’s ancestors describe the same entity and can play an important role in caption generation step. We will talk more details about this idea in later chapters.

2.2.2 ImageNet

After we have a basic understanding of the structure of WordNet, we can now formally introduce ImageNet. ImageNet is a large image dataset which contain about 14-million images. Particularly, ImageNet group the images into different sets and all images in the same set contain the same entity. For example, all images with dogs will be placed into the same set. In order to better describe each set and the entity contained by its images, ImageNet associates each set to a WordNet ID, i.e. ImageNet uses WordNet ID to indicate the entity contained in the images within each set. For convenience, we call the set in ImageNet as ImageNet set and it’s associated WordNet ID as ImageNet tag.

2.3 Project Motivation Revisit

Since the essential background has been presented, we can now give a deeper explanation for this project’s motivation. In this project, we aim to caption images in ImageNet and hence, create a new dataset. In order to do this, we first use a state-of-the-art model to generate captions for each image in ImageNet. However, the dataset created in this way is not good enough because it can hardly be used to improve an existing model. Particularly, we cannot use a dataset, which is created by an existing model, to improve the same model. Therefore, in order to make some difference, our next move is crucially: Modifying the captions generated in last step by using an algorithm called constrained beam search. More precisely, the target of using constrained beam search is to modify each caption generated in last step so that we can guarantee the modified version contains at least one phrase from the synset which is associated with each image. For example:

(a) Original: A close up of a rock in the dirt

(b) Modified: A close up of a snake on the ground
Therefore, by making this modification, it is expected that we can improve the quality of a large portion of captions which are generated in first step. As a result, it is reasonable to think that this modified dataset contain a lot of new features which cannot be produced by the original model and hence, the model can be further improved by taking this dataset as input.

Indeed, the second step can be regarded as a data augmentation process, in which we add extra features to original dataset and hence, give the dataset ability to produce better models.
Chapter 3

Model

In last chapter we mentioned that the key step in image captioning task is to estimate the probability distribution of the caption string (Equation 2.1). In this Chapter, we will continue to discuss how to construct this probability model. Inspired by how human describe an image: we first read some key information from image. The estimation of the probability model also involves a step similar to this, i.e. extract information, which we called feature in image captioning task, from input image. Indeed, this step also reflected in Equation 2.1, in which, the probability distribution is conditioned on input image $I$. The next stage is to model the distribution $P(S, I)$ with the extracted features. Technically, the first stage can be done through an encoder and the second step is accomplished by decoder. The full process is shown as follow:

Generally, both encoder and decoder are mathematical models which take one or more inputs and produce an output, e.g. encode take image as input and output extracted features. However, we will not go through heavy math in this chapter as it can be found in previous works: Ren et al. (2015), Hochreiter and Schmidhuber (1997), and is slightly beyond the scope of this project. Instead, we will focus on the intuitive explanation about how these components work and provide some basic mathematical explanation.

3.1 Encoder

We start from the structure of encoder, which work as a feature extractor in our image captioning model. Since computer parses image as an multi-dimension tensor, the features extracted from a given image is also a tensor which is obtained through some mapping from original image. The operation which can construct this mapping is called convolution.
3.1.1 Convolution

The target of convolution is to map a \( m \times n \) matrix \( M \) to a new one \( M' \). In order to do this, convolution first introduces a small \( k \times k \) matrix \( K \) called kernel. This kernel matrix will slide over \( M \) and takes element-wise multiplication with the sub-matrix it covers within \( M \) and sum them together to form the new element in \( M' \). The following example demonstrate a simple convolution over a \( 3 \times 3 \) matrix \( M \) with a \( 2 \times 2 \) kernel. The kernel \( K \) and matrix \( M \) are:

\[
K = \begin{bmatrix} k_1 & k_2 \\ k_3 & k_4 \end{bmatrix} \quad M = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,1} & m_{3,2} & m_{3,3} \end{bmatrix}
\]

Kernel \( K \) will slide over \( M \) and produce the mapped matrix, e.g.:

\[
o_1 = m_{1,1}k_1 + m_{1,2}k_2 + m_{2,1}k_3 + m_{2,2}k_4
\]

The above example demonstrates how convolution operates on a two-dimension matrix. Indeed, we can easily extend it to a multi-dimension tensor, e.g. three-dimension image tensor. Particularly, for a \( c \times m \times n \) tensor, we can expand our kernel to size \( c \times k \times k \) and repeat the same procedure. Additionally, by adding the number of kernels, we can also increase the depth of our outcome tensor. Mathematically, we can write the convolution as:

\[
f(T, K_1, K_2, \cdots, K_c) = T'
\]

where \( T \) is the input tensor, \( K_i \) is kernel and the output tensor \( T' \) must has depth equal to \( c \). Since an image is parsed as a tensor by computer, by applying convolution operation to a given image, we can successfully map the original image to a new tensor and accomplish a partial feature extraction step. The ‘partial’ here indicates one single convolution operation is usually not enough to fetch satisfied feature from an image, therefore, a more complex architecture, named Convolution Neural Network, which build upon convolution operation should be used as a complete feature extractor. We will introduce this architecture with more details in next section but now, we focus on the other question raise from here: Once we apply convolution operation to feature extraction, what should be the value of each kernel \( K_i \). Indeed, the value of \( K_i \) is part of the parameters that we need to establish in our model \( P(S, T) \). Recalling Equation 2.1, in which \( \Theta \) is a set of parameters, therefore, \( K_i \in \Theta \).

Example

We then look at an example which demonstrate how convolution operation works. Consider a \( 3 \times 3 \) black-white image \( I \) and two \( 2 \times 2 \) kernels \( K_1, K_2 \). The image \( I \) will be processed as a \( 2 \times 3 \times 3 \) tensor by computer:

\[
I = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}
\]
In here, the left matrix represents the first channel of the image tensor and the right matrix is the second channel. Since the image tensor has depth(channel) equals to 2, the depth of $K_1$ and $K_2$ should also be 2, i.e. the actual shape of $K_1$ and $K_2$ should be $2 \times 2 \times 2$. We then assume $K_1$ takes the following values:

$$K_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Similarly, we can assume the value of $K_2$ is:

$$K_2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

By following the procedure we discussed above, the convolution result produced by $K_1$ will be:

$$O_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Particularly, each element is obtained through convolution operation shown as:

$$\begin{cases} 1 = 0 \times 1 + 1 \times 0 + 1 \times 0 + 0 \times 1 + 1 \times 0 + 0 \times 1 + 1 \times 1 + 1 \times 0 \\
3 = 1 \times 1 + 1 \times 0 + 0 \times 0 + 0 \times 1 + 0 \times 0 + 1 \times 1 + 1 \times 1 + 0 \times 0 \\
2 = 1 \times 1 + 0 \times 0 + 0 \times 0 + 0 \times 1 + 1 \times 0 + 1 \times 0 + 1 \times 1 + 1 \times 0 \\
1 = 0 \times 1 + 0 \times 0 + 0 \times 0 + 0 \times 1 + 1 \times 0 + 0 \times 1 + 1 \times 1 + 0 \times 0 \end{cases}$$

Specifically, we label the first channel(the left part matrix) of $K_1$ with red colour and the second channel with green colour to better illustrate the calculation. Similarly, the kernel $K_2$ will produce the output:

$$O_2 = \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix}$$

Therefore, the final output tensor $T'$ of this convolution operation is the concatenation of $O_1$, $O_2$ and has size $2 \times 2 \times 2$:

$$T' = \begin{bmatrix} 1 & 3 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 4 & 1 \\ 2 & 1 \end{bmatrix}$$

### 3.1.2 Convolution Neural Network

Based on the idea of convolution, one architecture build upon it and worked well as feature extractor is called Convolution Neural Network (CNN) (Krizhevsky et al., 2012). CNN is initially proposed to tackle image classification problem, which also requires a feature extractor to fetch features from images. By considering the outstanding performance of using CNN in image classification, it is reasonable to transfer this architecture into image captioning task.

Indeed, CNN is a stack of convolution operations. Recalling our introduction before, one convolution can map a tensor to a new one with different size. We can keep repeating this process for each new tensor until we produce one with desired size, e.g. a vector, as shown in the following figure:
Each single convolution operation in CNN is called convolution layer or simply layer. Since CNN is consist of several convolution operations (layers), the kernels in each layer will become parameters when CNN is applied to image captioning task and used to model the distribution $P(S, I)$.

**Example**

In order to better demonstrate the architecture of CNN, a 2-layer CNN will be taken as an example here. Consider the same input $3 \times 3$ black-white image $I$ as in last example and a 2-layer CNN whose first layer is a convolution operation with two $2 \times 2$ kernels $K^{(1)}_1$ and $K^{(1)}_2$. In this layer, we assume the value of $K^{(1)}_1$, $K^{(1)}_2$ are identical to $K_1$ and $K_2$ in our previous example. Therefore, the outcome of this first convolution layer is:

$$T_1 = \begin{bmatrix} 1 & 3 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 4 & 1 \\ 2 & 1 \end{bmatrix}$$

We now assume the second layer is a convolution operation with three $2 \times 2$ kernels $K^{(2)}_1$, $K^{(2)}_2$ and $K^{(2)}_3$:

$$K^{(2)}_1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \quad K^{(2)}_2 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad K^{(2)}_3 = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

By following the same convolution procedure, we then get the final output of this CNN architecture:

$$o = [12 \ 5 \ 9]$$

In this example, the final output is a tensor with size $3 \times 1 \times 1$, in practice, we simply denote it as a vector: $o = [12 \ 5 \ 9]^T$. Indeed, converting an input image to a vector is a typical choice for many computer vision task like image classification. Even in earlier age of image captioning, this is also a golden choice. However, by carefully choosing the size of kernels in different layers, we can produce the output tensor with arbitrary size. This property is applied to a further modified architecture called Region Proposal Network, which is used to build a more powerful feature extractor and will be introduced in next section.

### 3.1.3 Region Proposal Network

Despite the vanilla CNN we introduced before can do a great job on feature extraction, compressing the entire image into one single vector would still lose some important 'local' features. Specifically, for a given image, the most important features it presents could be some entities, e.g. people, animals or objects. Therefore, in order to generate better captions, it is more reasonable to fetch
the separate features from those key entities. The new architecture which accomplish this task is called \textit{Region Proposal Network} (RPN) (Ren et al., 2015) and it is built upon traditional CNN. Specifically, the first component of RPN is a normal CNN. However, unlike our example in last section, this CNN does not produce a single feature vector, instead it map the input image tensor to a small size tensor called \textit{feature map} of the original image. Intuitively, we can think each portion of feature map represent the feature of some part of original image, as shown in the Figure 3.2, the portion labeled with orange color in feature map compresses the information of the blue part of original image:

![Figure 3.2: Feature map in RPN](image)

The next thing RPN do is using this feature map to detect key entities in original image. Particularly, for each portion of the feature map, RPN defines several rectangles called \textit{bounding boxes} and detect whether the corresponding area in original image contains entities which can fill in these boxes. The bounding boxes which contain key entities are called \textit{proposals} and will be selected for next move.

Finally, RPN collects those proposals and fetch their associated features from feature map. These proposals’ feature can then be used to tackle further task, which is estimate $P(S, I)$ in our case. The detail implementation of RPN and the parameters of this architecture which need to be estimated when it is applied to model $P(S, I)$ are extremely complex. By considering the scope of this project, which does not intend to replicate this architecture hand by hand, we will omit the further explanation here. However, we will go through an intuitive example to better demonstrate the basic procedure of RPN. More detail about this architecture can be referred to original paper (Ren et al., 2015).

**Example**

We can use Figure 3.2 as an example. Particularly, the image in this example can be considered as a 4 x 4 gray-scale image, which will be parsed as a two-dimensional matrix by computer. Furthermore, recalling that by carefully choosing the size of kernels in CNN component within RPN, we can produce a feature map with arbitrary size. In this example, we assume the feature map has size 1 x 2 x 2, as shown in Figure 3.2. It is clearly that the ration between the original image’s height(width) and the feature map’s height(width) is 2 : 1. As a result, each element of feature map compresses the information of a 2 x 2 sub-image in original input image, just like Figure 3.2. In here, the size of sub-image is calculated based on the ratio between original image and feature map, i.e. $2 = 4 \times \frac{1}{2}$, where $\frac{1}{2}$ is the ratio.

For each sub-image mapped by the feature map, as mentioned above, RPN will define several bounding boxes and detect whether there exist some boxes which contain key entities in this sub-image. For example, the blue part in Figure 3.2 may contain a key entity dog and can fit into one
bounding box. After that, the feature associate with the selected boxes, e.g. the box contain the
dog, will be extracted and used for further caption generation task.

3.2 Decoder

We have already mentioned how to extract features from a given image, in this section, we will
introduce how these features can be used to estimate probability distribution 2.1. We will start
with the introduction of a architecture which been widely used to generate plain string $S$, i.e.
model the probability distribution:

$$P(S) = P(w_1 w_2 \cdots w_n; \Theta)$$ (3.1)

which is the joint probability of a string without a given image $I$ and $\Theta$ is the set of parameters
that need to be estimated in this model.

This architecture is called *Long-short Term Memory* (LSTM) and is first proposed by Hochreiter
and Schmidhuber (1997).

3.2.1 Long-short Term Memory

The atomic unit of LSTM is LSTM cell, which takes two input vectors: $x$ and $h$ and produces one
output vector $\hat{h}$:

The complete LSTM is built by stacking each LSTM cell together. Particularly, if we denote the
$i$th cell as $L_i$, then, the output of $L_i$ will also be used as a part of the input for $L_{i+1}$, as shown in
the following figure:

For convenience, in complete LSTM, we denote the input of cell $L_i$ as $x_{i-1}$, $h_{i-1}$ and output as
$h_{i+1}$. Intuitively, we can think the $i$th input $h_i$ for cell $L_{i+1}$ compresses the information or state
contained by the previous cells $L_1 L_2 \cdots L_i$ and hence, it is usually called state variable. The other input variable $x_i$ is a specific input at $(i + 1)$th step and is called time-step variable.

When LSTM is used to model distribution 3.1, the $i$th output $h_i$ usually has the other usage. Specifically, we usually process and normalize it to be a probability distribution $p_i$. This is usually accomplished by a mathematical function called softmax. The principle of softmax is really easy. Particularly, for the vector:

$$h_i = [h_1 \ h_2 \ \cdots \ h_{|V|}]^T$$

In which, the number of elements in $h_i$ should always equal to the size of vocabulary $V$. Then, the result of softmax over $h_i$ will be:

$$\text{softmax}(h_i) = \left[ \frac{\exp(h_1)}{\sum_j \exp(h_j)} \ \frac{\exp(h_2)}{\sum_j \exp(h_j)} \ \cdots \ \frac{\exp(h_{|V|})}{\sum_j \exp(h_j)} \right]^T$$

It is not hard to verify that each element in $\text{softmax}(h_i)$ is in the range $[0, 1]$ and the sum over them is equal to 1. Therefore, it can be proved that the outcome of softmax function is a probability distribution (denote as $p_i$) and it is reasonable to think that this distribution is the probability distribution of the $i$th word in caption. More precisely, $p_i$ is usually considered as the conditional probability distribution of $i$th word $w_i$ by given the previous word sequence $w_1 w_2 \cdots w_{i-1}$:

$$p_i = P(w_i | w_1 w_2 \cdots w_{i-1})$$

With this distribution, the final output token of $w_i$ can be drawn from the associated distribution $p_i$. This step is called decoding and many decoding algorithm which specifies how to draw the final output token from the distribution has been proposed. The detail of these decoding algorithms will be introduced in next chapter.

One last question here is that for each cell $L_i$, what should be the value of time-step input $x_{i-1}$. Indeed, the input $x_{i-1}$ for the $i$th cell $L_i$ is the final output token selected by decoding algorithm from the distribution $p_{i-1}$ produced by the previous cell $L_{i-1}$. Therefore, the LSTM used for string generation will look like this:

![Diagram of LSTM in image captioning]

Moreover, for the first cell $L_1$ in LSTM, it’s time-step input $x_0$ is always the token SOS and it’s state input $h_0$ is initialized to a specific value, e.g. zero vector. The string generation is completed when the token EOS is selected. Indeed, this is also the reason why the tokens SOS and EOS should be included in the vocabulary.

Following the discussion above, it can be seen that each LSTM cell $L_i$ model the distribution $P(w_i | w_1 w_2 \cdots w_{i-1})$ and pick a token for $w_i$ according to some decoding algorithms. After then,
the picked token will be used as input $x_i$ for next cell $L_{i+1}$ and produce $P(w_{i+1} | w_1 w_2 \cdots w_i)$. We can assume that the decoding algorithm selected each word as:

$$w_1 = v_1, w_2 = v_2, \cdots w_n = v_n$$

Where $v_1, v_2, \cdots v_n$ are some tokens in vocabulary $V$. Thus, by applying chain rule, we can get:

$$\log P(v_1 v_2 \cdots v_n) = \sum_{i=1}^{n} \log P(v_i | v_1 v_2 \cdots v_{i-1})$$

Therefore, it can be seen from here that LSTM does not actually model distribution $P(w_1 w_2 \cdots w_n)$, instead, it calculate the probability value for a specific string, e.g. $v_1 v_2 \cdots v_n$. The role of decoding algorithm playing here is to select best token for each variable $w_i$ which can maximize the final probability of the string, i.e.

$$\max_{w_1 w_2 \cdots w_n} \log P(w_1 w_2 \cdots w_n)$$

Moreover, it is straightforward to apply LSTM to image captioning task. Particularly, we can use the extracted features of the given image as the first state-input of LSTM, i.e. the extracted features will be the state-input $h_0$ of first cell $L_1$ in LSTM. However, the question here is that $h_0$ requires a one-dimensional vector and the extracted feature produced by RPN, as mentioned in previous sections, is a two-dimensional matrix. One solution for this is to sum the feature matrix over row-dimension, e.g. assume the feature matrix $F$ is:

$$F = \begin{bmatrix} f_1 & f_2 & \cdots & f_n \end{bmatrix}^T$$

We can then produce a new vector $f'$ as: $f' = \sum_{i=1}^{n} f_i$, which can be used as state-input to $L_1$. By counting this vector $f'$ in, the probability distribution produced by each cell will then becomes:

$$p_i = P(w_i | w_1 w_2 \cdots w_{i-1}, I)$$

Therefore, it is easy to prove that under this situation, the LSTM will model the probability distribution 2.1, which is the probability of generated string by given a input image.

### 3.2.2 Word Embedding

We have seen that LSTM will use token in vocabulary $V$ as input vector. However, this may raise one question: how computer parse a English token as a numerical vector. The most fundamental method is to represent each token as an one-hot vector. One-hot vector is a special vector with all of it’s element set to 0 except one whose value will be set to 1. As a result, if a one-hot vector has length equal to the size of vocabulary, then, the element been set to 1 in the vector indicates which token in vocabulary this vector represent. For example, consider a vocabulary has three token: \{ people, cat, dog \}. Then, the one-hot vector: $[0 \ 1 \ 0]^T$ represents the token cat and the vector: $[0 \ 0 \ 1]^T$ represents token dog.

Even though the one-hot vector do represent a English token as numerical vector and can be used as input to LSTM, the drawback of using one-hot vector is obvious: it loses the meaning, or feature, of each token. For example, with the one-hot representation, there is no significant
difference between token like **people**, **cat** and **dog**. However, the tokens **dog** and **cat** should share more similarity and in the mean time, token **people** should have more different meaning with them.

In order to address this problem, one more advanced technology, which used to convert token into numerical vector, is called *word embedding* (Mikolov et al., 2013). By applying this technology, the tokens which have similar meaning will share more similarity after them are converted to vector. The similarity mentioned here, usually indicate mathematical similarity like *cosine similarity*. For example, consider two vectors: $x_1$ and $x_2$, their cosine similarity is defined as:

$$\cos(x_1, x_2) = \frac{x_1 \cdot x_2}{|x_1| |x_2|}$$

Thus, the vectors convert from tokens **dog** and **cat** in previous example will have high cosine similarity. In contrast, the vector of **people** will share a low cosine similarity with both **dog** and **cat**.

In our project, we will not re-implement word-embedding to produce word vectors. Instead, a set of pre-trained word vectors, called Glove (Pennington et al., 2014), which contains the vectors of almost all common English tokens used in our daily life, will be used in this project. Therefore, the detail of how word embedding is implemented is beyond the scope of this project and will be omitted in this report.

### 3.2.3 Math Behind LSTM

We have mentioned that each LSTM cell $L_i$ takes a time-step input $x_{i-1}$ and a state input $h_{i-1}$. In this section, we are going to introduce the math behind each LSTM cell and how these two inputs are used to produce the output $h_i$. Specifically, for each LSTM cell, e.g. $L_i$, it can be mathematically defined as follow:

\[
\begin{align*}
i &= \sigma(W_{ih}h_{i-1} + W_{ix}x_{i-1} + b_i) \\
f &= \sigma(W_{fh}h_{i-1} + W_{fx}x_{i-1} + b_f) \\
o &= \sigma(W_{oh}h_{i-1} + W_{ox}x_{i-1} + b_o) \\
g &= \tanh(W_{gh}h_{i-1} + W_{gx}x_{i-1} + b_g) \\
c &= f \circ h_{i-1} + i \circ g \\
h_i &= o \circ \tanh(c)
\end{align*}
\]

in which, $\circ$ denotes the element-wise multiplication of two matrices/vectors and is called *hadamard product*. The matrices $W_{ih}, W_{ix}, W_{fh}, W_{fx}, W_{oh}, W_{ox}, W_{gh}, W_{gx}$ along with the vectors $b_i, b_f, b_o, b_g$ are the parameters need to be estimated, i.e.

$$\{W_{ih}, W_{ix}, W_{fh}, W_{fx}, W_{oh}, W_{ox}, W_{gh}, W_{gx}, b_i, b_f, b_o, b_g\} \subseteq \Theta.$$  

### 3.3 Attention

#### 3.3.1 Naive Attention

Despite a simple CNN-LSTM architecture can accomplish the image captioning task, it is usually not enough to generate a quality caption. Inspired by how human read texts and images, an
attention mechanism (Bahdanau et al., 2014) was proposed and can be equipped with encoder-decoder architecture. Specifically, when human read text, we will pay different weight of attention to different part of text. Inspired by this, we can use a similar manner to deal with a sequence of inputs. Particularly, consider a sequence of inputs which can be assembled as a matrix $X = [x_1, x_2, \ldots, x_n]^T$, where each $x_i$ represents an input vector. The attention block take this matrix as input and produce a weight distribution over it, i.e. a distribution $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_n]$ with each $\alpha_i \geq 0$ and $\sum_{i=1}^{n} \alpha_i = 1$. The component $\alpha_i$ indicate how much attention should be paid to input vector $x_i$. Usually, we will construct a new input vector $\hat{x}$, which is the weighted sum over $X$:

$$\hat{x} = \sum_{i=1}^{n} \alpha_i x_i$$

In the case of image captioning, recalling that our RPN encoder will produce a set of feature vectors, which can be assembled as a matrix $X$. Thus, by input this matrix to attention module and make the weighted sum over it, we can get a new feature vector $\hat{x}$.

### 3.3.2 Button-Up-Top-Down Attention

A slightly modification can be applied to the original attention block and target specifically to image captioning task. This new version of attention module is called Button-Up-Top-Down (BUTD) Attention, which take two inputs:

- A matrix $X$ which represents a set of feature vectors extracted from image.
- A vector $q$ which is called query vector.

The output of BUTD attention, like the original attention module, is the weight distribution over input features. Intuitively, we can think BUTD attention produces the attention weight according to a specific query (this is why $q$ called query vector).

### 3.4 Complete Model

This section will introduce the complete model used in this project. The model, which is called Button-Up-Top-Down Model (BUTD) (Anderson et al., 2017b). Generally, BUTD follows the classical encoder-decoder architecture with some modifications over original LSTM decoder. Particularly, the atomic unit of BUTD decoder is called BUTD cell, which is built upon naive LSTM cell, as shown in following figure:

![BUTD cell diagram](image.png)

Figure 3.6: BUTD cell
It can be seen from the figure that BUTD cell is consisted of a 2-layer LSTM cell with a BUTD attention module. The output \( \mathbf{h}^{(1)} \) of LSTM cell 1 has two internal usages:

- Used as a query vector and input to BUTD attention module.
- Used as a partial input to LSTM cell 2.

The inputs of BUTD attention module are image feature matrix \( \mathbf{X} \) and query vector \( \mathbf{h}^{(1)} \). It will produces the weight distribution and we can then calculate the weighted sum \( \hat{\mathbf{x}} \) of \( \mathbf{X} \) based on this distribution. Finally, \( \hat{\mathbf{x}} \) and \( \mathbf{h}^{(1)} \) will be concatenated together, i.e. \( [\hat{\mathbf{x}}, \mathbf{h}^{(1)}] \), to use as a input vector of LSTM cell 2, which will also produce a output, i.e. \( \mathbf{h}^{(2)} \). The final output of BUTD cell is consisted of two component: \( \mathbf{h}^{(1)} \) and \( \mathbf{h}^{(2)} \), which are the outputs of each LSTM cell respectively.

Similar to native LSTM, the BUTD decoder is constructed by connect each BUTD cell together. For convenience, we also denote the \( i \)th BUTD cell as \( L_i \) and denote it’s outputs as \( \mathbf{h}^{(1)}_i \) and \( \mathbf{h}^{(2)}_i \) respectively. Particularly, one of \( L_i \)’s output \( \mathbf{h}^{(2)}_i \) will be used to estimate the probability distribution \( \mathcal{P}(w_i | w_1 w_2 \cdots w_{i-1}) \) by applying softmax function.

The entire BUTD architecture is shown in Figure 3.7. For better illustrate how this architecture work and also summarizes the content of this chapter, we will look at a simple example. Consider an input image with arbitrarily size \( H \times W \). This image will be parsed by the computer as a \( 3 \times H \times W \) tensor denoted as \( \mathcal{I} \). Tensor \( \mathcal{I} \) will then be used as input to RPN encoder, which will produce a set of feature vectors and format them as a \( N \times D \) matrix \( \mathbf{X} \), in which \( N \) is the number of vectors and \( D \) is the dimension(size) of each vector. This matrix \( \mathbf{X} \) will then be used as input to each BUTD cell in BUTD decoder. More precisely, \( \mathbf{X} \) is the input of BUTD attention module in each cell. For the caption generation, which is performed by BUTD decoder, start with cell \( L_1 \), the time step input \( x_0 \) of this cell is the word vector for token \( \text{SOS} \) and the two state input \( \mathbf{h}^{(1)}_0, \mathbf{h}^{(2)}_0 \) will both be initialized to zero-vector \( \mathbf{0} \). Then, for each cell \( L_i \) with \( i > 1 \), the time-step input \( x_{i-1} \) is the word vector for the selected token of \( w_{i-1} \) and the state inputs \( \mathbf{h}^{(1)}_{i-1}, \mathbf{h}^{(2)}_{i-1} \) are the outputs of last BUTD cell \( L_{i-1} \). Particularly, \( \mathbf{h}^{(2)}_{i} \) will also be used to estimate the distribution \( \mathcal{P}(w_i | w_1 w_2 \cdots w_{i-1}) \) and this distribution will be further used by decoding algorithm to choose the best token for \( w_i \). The entire caption generation process will stop when token \( \text{EOS} \) is selected for a word \( w_j \).

![Figure 3.7: BUTD](image-url)
Chapter 4

Decoding Algorithm

Recalling what we discussed in last chapter, the \(i\)th cell in our BUTD decoder will model the probability distribution of the \(i\)th word \(w_i\) in caption, i.e. \(P(w_i|w_1,w_2,\ldots,w_{i-1},I)\). In order to generate the complete caption, efficient decoding algorithm must be used to determine the final value of \(w_i\) based on it’s probability distribution. This section will introduce some common algorithms, includes the one used in this project.

4.1 Greedy

The most common choice for decoding algorithm is greedy algorithm. Particularly, for each \(w_i\), it’s final value will be the token which can maximize the probability \(P(w_i|w_1w_2\cdots w_{i-1},I)\), i.e.

\[
  w_i = \operatorname{arg\,max}_{w_i} P(w_i|w_1w_2\cdots w_{i-1},I)
\]

However, the drawback of this greedy algorithm is obvious. Specifically, consider the objective function of image captioning task and the inequality:

\[
  \max \log P(w_1w_2\cdots w_n|I) \geq \sum_{i=1}^{n} \max \log P(w_i|w_1w_2\cdots w_{i-1},I) \tag{4.1}
\]

The inequality indicates that in most case, greedy cannot solve the objective function. An example may be helpful to explain this: consider a caption which has two words: \(w_1w_2\), and a vocabulary contains two tokens \(\{v_1, v_2\}\). Assume \(w_1\) and \(w_2\) follow the probability distribution:

<table>
<thead>
<tr>
<th></th>
<th>(w_1 = v_1)</th>
<th>(w_1 = v_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P(w_1</td>
<td>I))</td>
<td>0.45</td>
</tr>
<tr>
<td>(P(w_2</td>
<td>w_1 = v_1, I))</td>
<td>0.1</td>
</tr>
<tr>
<td>(P(w_2</td>
<td>w_1 = v_2, I))</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Table 4.1: Example of greedy algorithm
Even though this is just an assumption probability distribution, we can still identify the fact that:

\[
\max \log \mathcal{P}(w_1, w_2 | I) = \log \mathcal{P}(w_1 = v_1, w_2 = v_2 | I) \\
> \max \log \mathcal{P}(w_1 | I) + \max \log \mathcal{P}(w_2 | w_1, I) \\
= \log \mathcal{P}(w_1 = v_1 | I) + \log \mathcal{P}(w_2 = v_2 | w_1 = v_1, I)
\]

This inequality indicates that the greedy algorithm cannot perfectly maximize the objective function. Therefore, one modified algorithm which targets for tackling this problem is called beam search (Wiseman and Rush, 2016) and will be introduced in next section.

### 4.2 Beam Search

Ideally, the objective function can be perfectly solved by brute force searching each possible combination of \(w_1 w_2 \cdots w_n\), i.e. check all possibility for each \(w_i\). However, this is infeasible in practical as the time complexity of this brute force algorithm is \(O(n^{|V|})\), where \(n\) is the number of words in caption and \(|V|\) is the size of vocabulary (We can assume \(|V| = N\)). Therefore, in order to make the decoding algorithm feasible and better approximate \(\max \log \mathcal{P}(w_1, w_2, \cdots, w_n | I)\), we can extend the original greedy algorithm. Particularly, for each \(w_i\), instead of choosing only one token which maximize \(\mathcal{P}(w_i | w_1 w_2 \cdots w_n, I)\), we can choose \(k\) candidate tokens which result in top \(k\) joint probability values. This extended greedy algorithm is called beam search and may be better illustrated as the following figure:

![Beam Search Diagram](image)

The above figure illustrates how beam search works by choosing \(k = 3\), i.e. choose 3 candidates for each \(w_i\). In this figure, \(v_i^{(1)}\), \(v_j^{(1)}\) and \(v_k^{(1)}\) are the three candidate tokens for \(w_1\). Our BUTD decoder
will then calculate the probability distribution of $w_2$ based on each of these three candidates, i.e. $P(w_2|w_1 = v_i^{(1)}, \mathcal{I}), P(w_2|w_1 = v_j^{(1)}, \mathcal{I})$ and $P(w_2|w_1 = v_k^{(1)}, \mathcal{I})$. By using these three distribution, we can easily model the following three joint probabilities:

\[
\begin{align*}
P(w_1 = v_i^{(1)}, w_2|\mathcal{I}) &= P(w_2|w_1 = v_i^{(1)}, \mathcal{I})P(w_1 = v_i^{(1)}|\mathcal{I}) \\
P(w_1 = v_j^{(1)}, w_2|\mathcal{I}) &= P(w_2|w_1 = v_j^{(1)}, \mathcal{I})P(w_1 = v_j^{(1)}|\mathcal{I}) \\
P(w_1 = v_k^{(1)}, w_2|\mathcal{I}) &= P(w_2|w_1 = v_k^{(1)}, \mathcal{I})P(w_1 = v_k^{(1)}|\mathcal{I})
\end{align*}
\]

We then choose another 3 candidate tokens for $w_2$ which have the highest value among these three joint probability: $v_p^{(2)}, v_q^{(2)}$ and $v_r^{(2)}$. In this example, We assume that $v_p^{(2)}$ and $v_q^{(2)}$ are both drawn from distribution $P(w_1 = v_i^{(1)}, w_2|\mathcal{I})$ and $v_r^{(2)}$ is drawn from $P(w_1 = v_k^{(1)}, w_2|\mathcal{I})$. The thing need to be noticed here is that, for each chosen $w_2$, we need to memorize it’s associated previous sequence. For example, since $v_p^{(2)}$ is chosen from distribution $P(w_2|w_1 = v_i^{(1)}, \mathcal{I})$, we must memorize the entire sequence $w_1w_2 = v_i^{(1)}v_p^{(2)}$. Therefore, it is not hard to see from this that beam search is actually holding three sequences which have the highest probability. We then repeat this process until all three candidate sequences reach the EOS token. The final output will be the sequence which has the highest joint probability among the three candidates.

Intuitively, we can think that the three candidate sequences are stored in a beam and this is the reason why this decoding algorithm is called beam search. Based on the original beam search, many modified version has been proposed to accomplished some specific tasks. We will use one of them called constrained beam search in our project.

### 4.3 Constrained Beam Search

The original beam search will update cached sequences by appending the new tokens which can maximize the joint probability. This process can be seen as an unconstrained update process, i.e. the sequences we generate do not follow any rules. However, in practical, we usually hope that our caption can meet some specific criteria, e.g. must contain a specific token. In order to satisfy this kind of criteria, a modified beam search called constrained beam search is proposed by Anderson et al. (2017a). Instead of maintaining only one beam, constrained beam search forces the generated caption to satisfy some specific constrains by extending the number of beams and assign different update rules to them. In order to make this more concrete, consider an example that the generated captions must contain token $v_s$. In order to do this, we will need two beams: $B_1$ and $B_2$. Similar to the original beam search, each of these two beams maintains $k$ candidate sequences (result in $2k$ candidates in total) and when constrained beam search start to choose token for each $w_i$, it will first produce $2k$ joint probability distribution, $k$ for $B_1$ and the rest for $B_2$:

\[
B_1 \begin{cases} 
\mathcal{P}(S_1^{(1)}, w_i|\mathcal{I}) \\
\vdots \\
\mathcal{P}(S_k^{(1)}, w_i|\mathcal{I})
\end{cases} \quad B_2 \begin{cases} 
\mathcal{P}(S_1^{(2)}, w_i|\mathcal{I}) \\
\vdots \\
\mathcal{P}(S_k^{(2)}, w_i|\mathcal{I})
\end{cases}
\] (4.2)

Where $S_i^{(1)}$ and $S_i^{(2)}$ ($1 \leq i \leq k$) are the candidate sequences stored in $B_1$ and $B_2$ respectively. Each beam will then be updated by choosing $k$ candidate $w_i$ from these $2k$ distributions which have top probabilities. These steps are almost the same as the original beam search. The key distinguish
Therefore, in previous example, the rule that can denote the rule which specifies beam \( B \) actually specify what tokens it accept from any other beams \( B \)

1. if \( w_i \) is selected from the distributions \( \{ P(S^{(1)}_j, w_i | \mathcal{I}) | 1 \leq j \leq k \} \), i.e. the sequence generated before \( w_i \) is from \( B \) itself, the specific token \( v_s \) must not be selected.

2. \( w_i \) cannot be selected from distributions \( \{ P(S^{(2)}_j, w_i | \mathcal{I}) | 1 \leq j \leq k \} \), i.e. the sequence before \( w_i \) cannot come from \( B_2 \).

In the mean time, \( B_2 \) should follow the rules:

1. if \( w_i \) is selected from the distributions \( \{ P(S^{(1)}_j, w_i | \mathcal{I}) | 1 \leq j \leq k \} \), i.e. the sequence before \( w_i \) comes from \( B_1 \), \( w_i \) can only be \( v_s \).

2. if \( w_i \) is selected from distributions \( \{ P(S^{(2)}_j, w_i | \mathcal{I}) | 1 \leq j \leq k \} \), there is no restriction.

For example, at time step \( t \), constrained beam search need to pick tokens for \( w_t \). At this time step, both \( B_1 \) and \( B_2 \) maintain \( k \) candidate sequences: \( B_1 : \{ S^{(1)}_1, \ldots, S^{(1)}_k \} \), \( B_2 : \{ S^{(2)}_1, \ldots, S^{(2)}_k \} \) and will produce the probabilities similar to 4.2 with \( i = t \). Then, when constrained beam search start to choose \( w_t \) for \( B_1 \), according to rule 2, it will not consider any probability produced by \( B_2 \), i.e. \( \{ P(S^{(2)}_j, w_t) | 1 \leq j \leq k \} \). This indicates that for some tokens \( v_i \), where \( 1 \leq v_i \leq N \), \( P(S^{(2)}_j, w_t = v_i | \mathcal{I}) \) may be able to achieve a high probability, but it will still not be selected by \( B_1 \). Similarly, base on rule 1, the probabilities \( \{ P(S^{(1)}_k, w_t = v_s | \mathcal{I}) | 1 \leq j \leq k \} \) will be excluded no matter how high their values are. As a result, the candidate for \( w_t \) in \( B_1 \) are the tokens which produce the top \( k \) values among the set of probabilities \( \{ P(S^{(1)}_j, w_t \neq v_s | \mathcal{I}) \} \).

Regarding to \( B_2 \), according to rule 1, it will only consider the probabilities \( \{ P(S^{(1)}_j, w_t = v_s | \mathcal{I}) \} \) in \( B_1 \) and base on rule 2, the distributions hold by \( B_2 \) itself are all valid. As a result, \( B_2 \) will choose \( w_t \) from the set of distributions: \( \{ P(S^{(1)}_j, w_t = v_s | \mathcal{I}) | 1 \leq j \leq k \} \cup \{ P(S^{(2)}_j, w_t | \mathcal{I}) | 1 \leq j \leq k \} \).

One more important step in constrained beam search is initialization. Particularly, One beam will be selected as initial beam and contains a \( \text{SOS} \) token as startup sequence, this is similar to the original beam search. However, for the rest beams, them will be initialized as empty beams. In the above example, \( B_1 \) will be selected as initial beam and be initialized with a \( \text{SOS} \) token. On the other hand, \( B_2 \) will be initialized as empty.

Indeed, it can be seen that the key to use constrained beam search is to determine the number of beams along with their corresponding updating rules. This is usually a tricky task but in next section, an interesting perspective of constrained beam search will be introduced and there is a programmatic method to apply constrained beam search into image captioning task.

### 4.3.1 Constrained Beam Search as Finite Automata

A closer look at the previous example indicates that the rules assigned to a specific beam \( B_i \) actually specify what tokens it accept from any other beams \( B_j \), include itself. For convince, we can denote the rule which specifies beam \( B_i \) only accepts a set of tokens \( \mathcal{V}_s \) from beam \( B_j \) as:

\[
B_j \xrightarrow{\mathcal{V}_s} B_i
\]

Therefore, in previous example, the rule that \( B_2 \) only accept token \( v_s \) from \( B_1 \) can be denoted as \( B_1 \xrightarrow{\{v_s\}} B_2 \). Indeed, for any rule \( B_j \xrightarrow{\mathcal{V}_s} B_i \), it is reasonable to think that the candidate sequences
in $B_j$ can be transited to $B_i$ by generating a token $v \in V_s$. From this perspective, if we think each beam as a state, then, the rules associate with different beams actually define a state transition function $\delta(B_j, v) = B_i$, which implies state $B_j$ can transit to state $B_i$ by selecting $v$ as next token. Therefore, the constrained beam search can be abstracted as a finite automata $Q(B, V, \delta, b_0, B_s)$, in which, each beam corresponding to a state and form the set of states $B$, the input alphabet of this finite automata is the vocabulary $V$ and the rules define the transition function $\delta$. Moreover, the initial state $b_0$ of this finite automata is the initial beam in the constrained beam search algorithm and the set of accept states $B_s$ is the set of beams which contain the sequences satisfy the constrains.

To make this illustration more concrete, in the previous example, the constrained beam search can be abstracted as a finite automata $Q(B, V, \delta, b_0, B_s)$ with $B = \{B_1, B_2\}$, $b_0 = B_1$ and $B_s = \{B_1\}$. Particularly, the transition function $\delta$ is defined as:

$$\delta(B_i, v) = \begin{cases} B_1 & \text{if } B_i = B_1 \text{ and } v \neq v_s \\ B_2 & \text{if } (B_i = B_1 \text{ and } v = v_s) \text{ or } (B_i = B_2 \text{ and } v \in V) \end{cases}$$

This finite automata can also be illustrated as a figure shown below:

![Finite Automata Diagram](image)

Indeed, the conversion between constrained beam search and finite automata is bi-directional, which means, any finite automata can also be implemented as constrained beam search. By taking this into account, we can then develop a programmatic way to assign the rules in constrained beam search. Particularly, we can construct any finite automata and implement it by constrained beam search, which allow us to generate captions which accepted by the finite automata, i.e. with some specific constrains.

Moreover, by considering the fact that any regular expression can also be converted to a equivalent finite automata and hence, implemented by constrained beam search. An alternative approach to use constrained beam search in image captioning task is to design a regular expression which specifies the format of the captions and implement it as constrained beam search. This is usually a more convenient method since designing regular expression is usually a easy task and there exists a programmatic approach to convert regular expression to a corresponding finite automata.
Chapter 5

Implementation

This chapter will assemble the blocks mentioned in previous chapters together and describe more details about how this project is designed and implemented.

5.1 Pipeline

We have mentioned in Chapter 2 that the images within ImageNet are categorized into different ImageNet sets, and each set has an associate tag(synset) to describe the entity contained in each image. Our primary target in this project is to caption the images in each set and augment the generated captions by using constrained beam search to force one of the phrase in ImageNet tag appears in the captions. More precisely, the traditional method to generate captions is BUTD model combined with original beam search. In our project, we used constrained beam search to replace original beam search and hence, modify the generated captions and achieve the target of data augmentation. In order to accomplish this task, we designed the follow pipeline:

![Figure 5.1: Pipeline of Project](image-url)
It can be seen from this pipeline that the entire process start with picking a set in ImageNet. We then perform some pre-process over the synset(tag) of selected ImageNet set so that it can be better used by later stages. After that, we construct a finite automata associates with this selected set and use it to implement constrained beam search. The next step is to actually generate the captions for images in the selected set. After then, we check whether there exists unprocessed sets, i.e. set that have not been captioned, if the answer is yes, we repeat the process, otherwise, we finish the entire work.

In the rest of this chapter, we will describe more details about how each stages in above pipeline is implemented, we will particularly focus on tag pre-process and finite automata construction since the majority workload of this project are these two stages.

5.2 Tag Pre-process

After picking a set to caption, the first step is to pre-process the tag of this set. Specifically, we will collect the tag of selected ImageNet set along with it’s hyponymy. Recalling what we discussed in chapter 2, each ImageNet tag is a synset in WordNet and this tag along with it’s hyponymy form a tree structure. Therefore, at this stage, a simple depth-first-search is applied to this tree structure. The search algorithm will terminate when it reaches certain depth, i.e. the certain number of upper-levels, in this project, we set it to six. The outcome of this step is a list of phrases which contains the original tag and associated hypomymy. Each phrase may be consists of one or multiple tokens. For example, consider an ImageNet set with tag n02087122: \{hunting dog\}. The final collected phrases for this set after tag pre-process will be:

\{hunting dog, dog, domestic dog\}

There are two reasons why we should collect the hyponymy instead of using original tag only:

- Our predefined vocabulary \(\mathcal{V}\) may lack the synonyms associate with the original tag. Adding hyponymy can overcome this problem.

- Some synonyms associate with original tag may be rare and our captioning model cannot handle them well. By replacing them with some common hyponymy, we can enhance the robust of our model.

In the later caption generation stage, our goal is to ensure that one of the collected phrases can appear in our generated captions and hence better describe the content of the images. However, by considering the case that some images may contain the entity described by it’s tag multiple times, e.g. some images in n02087122 set may contain two or more dogs. Therefore, our collected phrases, which contain only singular, cannot fully describe the images. As a result, we still need to extend the list by pluralizing it’s items.

After we have finished this data pre-process step, our next move is to use the collected tags and hyponymys to construct a finite automata which can be used by constrained beam search algorithm.
5.3 Construct Finite Automata

5.3.1 Algorithm

The next stage is to construct a finite automata which can be used by constrained beam search. In order to do it in a systematic way, we follow the idea described in section 4.3: Constructing a regular expression and convert it to finite automata. Therefore, in this stage, we first build a regular expression which specifies the format (constrains) the generated captions should follow. In this case, the constrain is that the generated caption should contain one of the phrases we collected in tag pre-process stage. Specifically, for the given ImageNet set, we can generalize the format of collected phrase list $p$ as:

$$p = \{ v_1^{(1)} v_2^{(1)} \cdots v_1^{(r_1)} , v_1^{(2)} v_2^{(2)} \cdots v_1^{(r_2)} , \cdots , v_1^{(n)} v_2^{(n)} \cdots v_1^{(r_n)} \}$$

The above notation indicates that the $i$th phrase in $p$ is $v_1^{(i)} v_2^{(i)} \cdots v_1^{(r_i)}$, which contains $r_i$ tokens. Based on this, the regular expression $r$ which specify the desired constrain is:

$$r = * (v_1^{(1)} v_2^{(1)} \cdots v_1^{(r_1)}) | (v_1^{(2)} v_2^{(2)} \cdots v_1^{(r_2)}) | \cdots | (v_1^{(n)} v_2^{(n)} \cdots v_1^{(r_n)}) *$$

In which, the symbol * denote wild card, which can match any sequence of tokens. In order to make this more concrete, consider the example before, in which we assume the collected phrases are: \{ hunting dog, dog, domestic dog \}. Therefore, the corresponding regular expression constructed by the rule mentioned above is:

$$r = * (\text{hunting dog}) | (\text{dog}) | (\text{domestic dog}) *$$

The real phrase list collected in this project will be larger, but this small example may be able to illustrate how regular expression is constructed.

We then start to convert the regular expression to finite automata, which can then be implemented as constrained beam search (section 4.3.1). The method we used to accomplish this conversion is called McNaughton-Yamada-Thompson algorithm (Hopcroft et al., 2006), which start by constructing several small finite automatas for each phrase in list and stacking them together to form the final one.

Specifically, for each phrase, i.e. $v_1^{(i)} v_2^{(i)} \cdots v_1^{(r_i)}$, the small finite automata associate with it is initialized with 2 states: INIT and ACCEPT, which represent initial and accepting state respectively.

We then consider two cases based on the length of this phrase (the number of tokens it contains):

- For the phrase which contains only one token ($r_i = 1$), we claim that this token can trigger state transition between INIT and ACCEPT, i.e. the corresponding finite automata is:

```
start \rightarrow INIT \xrightarrow{v_1^{(i)}} ACCEPT
```

- For the phrase which contains multiple tokens, we add a middle state for each token and treat this token as the trigger of state transition, i.e.
We then combining each separate finite automata together by merging their INIT and ACCEPT states and stack the rest part, i.e.:

In the above figure, $S$ is the set of phrases which contain only one token. The last piece of our construction is two wildcard symbol at the beginning and the end of the regular expression. In order to embedded these two wildcard into our finite automata, we just add two self-loop edges to INIT and ACCEPT states which indicate these two states can transit to themselves by accepting any token. Therefore, the final finite automata will look like this:

Figure 5.2: The pattern of finite automata used in project

To better illustrate how this finite automata construction algorithm works, we again use the simple phrase list: \{hunting dog, domestic dog, dog\} in previous example as an demonstration.

5.3.2 Example

Consider the phrase list \{hunting dog, domestic dog, dog\} which contains three phrases. Our goal is to construct a finite automata which is equivalent to the regular expression $r$:

\[
r = *(\text{hunting dog})|(\text{dog})|(\text{domestic dog})*
\]

According to the construction algorithm, we first need to build three small finite automatas for each phrase:
- For the phrase `dog` which contains only one single token, it’s finite automata will look like this:

```
start  ----> INIT ----> dog ----> ACCEPT
```

- For the phrases `hunting dog` and `domestic dog`, they are similar cases since both of them contain two tokens. Therefore, their corresponding finite automatas will look like following:

```
start  ----> INIT ----> M^(1) ----> dog ----> ACCEPT
        (a)

start  ----> INIT ----> M^(2) ----> dog ----> ACCEPT
        (b)
```

We then combine these small finite automata together and handle the case of wildcard by adding self-loop edge to `INIT` and `ACCEPT` states. As a result, the final finite automata is:

```
start  ----> INIT ----> dog ----> ACCEPT
        (a)

start  ----> INIT ----> M^(1) ----> dog ----> ACCEPT
        (b)
```

5.4 Caption Generation

With the finite automata ready, we can then step into next stage: caption the images within the selected synset. In this stage, we will use the BUTD architecture mentioned in Chapter 3 to model the distribution 2.2 and use constrained beam search to generate strings which match the regular expression described in Section 5.3. The principle of how BUTD and constrained beam search work have already been discussed in Chapter 3, 4 and hence will be omitted here.

The BUTD model used in our project is built by using a framework called Pythia (Singh et al., 2020) developed by facebook research. Moreover, Pythia also provides a pre-trained BUTD model
which is used in our project. The terminology pre-trained here, as mentioned in Chapter 2, indicates a model with estimated parameters.

5.5 Result

We have already explained the implementation details of this project in previous sections. In this section, we aim to demonstrate some example outcomes. Additionally, to make a better demonstration of the performance of data augmentation, i.e. the performance of constrained beam search, we list both captions generated by using original beam search and constrained beam search. In the following examples, the captions labelled with original are those generated using original beam search and the captions labelled with constrained are generated by constrained beam search.

Figure 5.3: Sample captions for synset n01496331

Original: A close up of a black toaster on the ground.
Constrained: A picture of a fish in the water.

Original: A bird that is swimming in the water.
Constrained: A fish that is swimming in the water.

Figure 5.4: Sample captions for synset n01440764

Original: A man holding a banana in his hands.
Constrained: A man holding a fish in his hand

Original: A man holding a banana in his hands.
Constrained: A man holding a yellow fish in his hand

Figure 5.5: Sample captions for synset n01667778

Original: A bird sitting on top of a dirt ground.
Constrained: A turtle sitting on top of a dirt ground.

Original: There is a painting on the side of a cake.
Constrained: There is a turtle that is sitting on the ground.
In the above examples, the tokens with red color are the phrases from collected phrase list. For convenience, we omit the SOS and EOS tokens in each caption.

The last thing need to be mentioned here is the format of our output files. In this project, we use separate files to store the caption result of each set in ImageNet. Specifically, for each set, we use a json file to store the id of each image along with it’s caption. A sample output json file looks like this:

```json
{
    "n01580077_21219.JPEG" : "a blue bird sitting on top of a red pot",
    "n01580077_12854.JPEG" : "a close up of a bird on a beach",
    "n01580077_2106.JPEG" : "a black and white bird sitting on a branch"
}
```
Chapter 6

Analysis

Despite we have shown some successfully augment examples in previous chapter, by considering the complexity of image captioning task, a large part of our generated captions cannot work as good as these examples. However, if we are able to filter those bad captions, i.e. the cases that constrained beam search fails to improve the quality of captions, our captioned ImageNet dataset may be more valuable and can have a better performance when it is used to re-train image captioning model. Therefore, we performed some experiments over our captioning results and trying to find some clues which may be useful to filter bad captions. Specifically, we are willing to compare the joint log-probability of captions generated by constrained beam search with those generated by original beam search.

6.1 Experiment Setting Up

We first picked 100 images from 10 ImageNet set, i.e. 10 images from each set. We then examined the generated captions of these images and manually categorized them into three sets:

- The set $S_1$ contains the images whose captions generated by constrained beam search have better quality than those generated by original beam search.

- The set $S_2$ contains the images whose captions generated by constrained beam search are identical to those generated by original beam search.

- The set $S_3$ contains the images whose captions generated by constrained are not better than those generated by original beam search.

Apparently, we are more interested in the images contained by the first and third sets since these two sets may provide some information about how to filter the bad captions generated by constrained beam search. Therefore, our experiments will focus on conducting comparison over these two sets. However, this does not mean the images in the second set are not important, in contrast, for the case that constrained beam search and original beam search generate the same caption for a given image, it usually indicates that this generated caption can perfectly describe the content of this image.
6.2 Experiment Method

The basic idea of this experiment is simple: for each image, we have two captions, one is generated by constrained beam search and it has an associated log-probability $p_c$, the other is generated by original beam search and has log-probability $p_o$. We then compare the value of $p_c$ and $p_o$ and it may lead us to find some useful information. For example, Figure 6.1a shows the case that constrained beam search improve the quality of caption. The notation $p_{o1}$ and $p_{c1}$ in this figure denote the log-probabilities of captions generated by original beam search and constrained beam search respectively. On contrast, Figure 6.1b shows the case in which constrained beam search fail to produce a better caption.

(a) **Original**: An elephant is laying on a rock in a zoo. ($p_{o1} = -10.84$)

**Constrained**: A lizard laying on top of a pile of rocks. ($p_{c1} = -13.45$)

(b) **Original**: A bird that is sitting in the water. ($p_{o2} = -5.43$)

**Constrained**: A bird sitting on a snake in the water. ($p_{c2} = -16.76$)

Figure 6.1: Example of captions generated with and without using constrained beam search.

By comparing Figure 6.1a and Figure 6.1b, we can find out the difference between log-probabilities in Figure 6.1a is smaller than that in Figure 6.1b, i.e.:

$$|p_{o1} - p_{c1}| < |p_{o2} - p_{c2}|$$

Based on this observation, we give an initial assumption:

For a given image, if $|p_o - p_c|$ is small, then constrained beam search improves the quality of caption.

In order to verify this assumption, we perform the same comparison for each image in $S_1$ and $S_3$ and draw the bar charts as shown in Figure 6.2 to demonstrate the results: In Figure 6.2, the blue bars show the log-probabilities associate with the captions generated by constrained beam search and red bars indicate the log-probabilities of captions generated by original beam search. Figure 6.2a shows the comparison of images in $S_1$, which is the case that constrained beam search can improve the quality of captions. On contrast, Figure 6.2b shows the comparison of the images in $S_3$. By examining these two figures, we can find out:

- The captions produced by constrained beam search have lower log-probabilities compare with those generated by original beam search. It is because that constrained beam search force some extra tokens to appear in the caption and by adding these constrains, the cost is the reduction of the value of log-probability.
Constrained beam search improves the captions

(a) Constrained beam search improves the captions

Constrained beam search does not improve the captions

(b) Constrained beam search does not improve the captions

Figure 6.2: Comparison of log-probabilities

- In most cases, the difference between the log-probabilities in Figure 6.2a is smaller than that in Figure 6.2b. To make it more concrete, the following figures list some examples to demonstrate this:

<table>
<thead>
<tr>
<th>Original</th>
<th>Constrained</th>
</tr>
</thead>
<tbody>
<tr>
<td>A young boy standing on top of a dirt field. ($p_o = -10.41$)</td>
<td>A little lizard standing on top of a dirt field. ($p_c = -14.61$)</td>
</tr>
<tr>
<td>A person holding a skateboard on the ground. ($p_o = -7.37$)</td>
<td>A lizard with a skateboard in the air. ($p_c = -20.54$)</td>
</tr>
<tr>
<td>A close up of a rock in the dirt. ($p_o = -9.97$)</td>
<td>A close up of a snake on the ground. ($p_c = -12.58$)</td>
</tr>
<tr>
<td>A chocolate cake sitting on top of a wooden table. ($p_o = -5.138$)</td>
<td>A chocolate snake sitting on top of a wooden table. ($p_c = -15.852$)</td>
</tr>
</tbody>
</table>

in which, the figures listed in left column are drawn from $S_1$ and those in right column belong to $S_3$. Moreover, the average difference of log-probabilities shown in Figure 6.2a is 3.8821, which is smaller than the value 5.9515 in Figure 6.2b.
There exist some cases which contradict our assumption. For example, consider Figure 6.3e and Figure 6.3f which drawn from $S_1$ and $S_3$ respectively:

(e) **Original**: A small bird sitting on a tree branch. ($p_o = -4.27$)
**Constrained**: A small lizard sitting on a tree branch. ($p_c = -12.80$)

(f) **Original**: A close up of a cat laying on a bed. ($p_o = -8.96$)
**Constrained**: A close up of a snake laying on a bed. ($p_c = -11.42$)

Figure 6.3: Examples that contradict the initial assumption.

Apparently, Figure 6.3e has a higher log-probability different than Figure 6.3f and hence, violate our assumption.

By considering these observations, it can be seen that our initial assumption is incorrect. Therefore, we proposed our new assumption, which is slightly different than the original one:

For a given image, if $|p_o - p_c|$ is small, then it is more likely that constrained beam search modifies the caption in a correct way.

In order to verify this new assumption, we repeat the same experiment process over another 100 images. We again omit the cases that constrained beam search generates the same captions as original beam search and focus on the studying of the difference between log-probabilities. The experiment results are shown in Figure 6.4.

![Case that constrained beam search improve the outcome](image1)

![Case that constrained beam search does not improve the outcome](image2)

(a) Constrained beam search improves the captions
(b) Constrained beam search does not improve the captions

Figure 6.4: Result of second experiment

In this experiment, the average difference of log-probabilities in Figure 6.4a is 6.4097, which is slightly smaller than the value 7.1695 in Figure 6.4b. However, it definitely not correct to say that the smaller difference means the better caption quality. Therefore, by considering this, it is
feasible to set up a threshold value to filter the captions generated by constrained beam search, e.g. when the difference of log-probability exceed this certain value, we may be able to consider this caption as a bad case.
Chapter 7

Conclusion

7.1 Summary
This project accomplished the work that captioned the images within ImageNet dataset. Specifically, we used Bottom-Up-Top-Down architecture, which is the state-of-art encoder-decoder architecture, to estimate the probability distribution of caption strings and then applied constrained beam search to generate actually caption for each image. We also perform some essential analysis over our captioned data and draw the conclusion: If the caption generated by constrained beam search has similar log-probability value with the one generated by original beam search, it is more likely constrained beam search produces a good example. Based on this conclusion, we also proposed a possible method to filter the good and bad captions over our generated captions.

7.2 Future Work

7.2.1 Comparison between NFA and DFA
We have mentioned in previous chapters that constrained beam search is implemented based on a given finite automata, which is converted from a regular expression. However, for a specific regular expression, it’s conversion to finite automata is not unique. For example, consider a sample phrase list: \{oscine, oscine bird\}, which is extracted from a real phrase-list in our project. In order to make the generated caption contains one of the phrase in this list, we need to construct a regular expression as follow:

\[ r = *(oscine)|(oscine\ bird)* \]

We can then convert it to two different finite automata, one is constructed by using the algorithm we described previous:
This finite automata is also called none-deterministic finite automata (NFA). The characteristic about NFA is that for a specific state, e.g., INIT state in this example, it may transit to multiple possible states when it receives one token, e.g., INIT may transit to ACCEPT or $M_1^{(1)}$ when it receive input token oscine.

The other different finite automata can be constructed from the same regular expression $r$ is shown in Figure 7.2. It can be seen that in this finite automata, each state can only transit to one other state when it receives a specific token. This type of finite automata is called deterministic finite automata (DFA). The advantage of DFA compares with NFA is that DFA is easy to implement. However, the cost of this is that DFA usually requires more states, as shown in Figure 7.1 and 7.2. Inspired by this, one interesting work is to compare the memory cost of NFA and DFA when they are implemented as constrained beam search.

### 7.2.2 Re-training

In Chapter six, we proposed a method to distinguish the good and bad captions generated in our project. We can use this method to filter the bad captions and use the remaining dataset to perform a re-training work. Specifically, we can use our dataset as training dataset and input it to BUTD model and inspect whether we can construct a better model by using this dataset. However, by considering the size of our dataset, this work requires heavy workload and computing resource. As a result, we did not count it as a part of this project and hope we may be able to accomplish it in future work.
Bibliography


Appendices
Appendix A

Study Contract
# INDEPENDENT STUDY CONTRACT PROJECTS

*Note: Enrolment is subject to approval by the course convenor*

## SECTION A (Students and Supervisors)

<table>
<thead>
<tr>
<th>UniID:</th>
<th><em><strong><strong>u6162630</strong></strong></em>_____</th>
</tr>
</thead>
<tbody>
<tr>
<td>SURNAME:</td>
<td>Lin__________</td>
</tr>
<tr>
<td>FIRST NAMES:</td>
<td>Songtuan__________</td>
</tr>
<tr>
<td>PROJECT SUPERVISOR <em>(may be external)</em>:</td>
<td>Stephen Gould____________</td>
</tr>
<tr>
<td>FORMAL SUPERVISOR <em>(if different, must be an RSSCS academic)</em>:</td>
<td>Stephen Gould____________</td>
</tr>
<tr>
<td>COURSE CODE, TITLE AND UNITS:</td>
<td>COMP8755, Individual Computing Project, 12 units</td>
</tr>
</tbody>
</table>

### COMMENCING SEMESTER

- [ ] S1
- [x] S2

**YEAR:** _______ **Two-semester project (12u courses only):** [x]

## PROJECT TITLE:

Captioning ImageNet

## LEARNING OBJECTIVES:

1. Experience in training and testing deep learning models.
2. Understanding of state-of-the-art captioning models.
3. Understanding cutting-edge research project.

## PROJECT DESCRIPTION:

This project aims to caption the 14 million images within the ImageNet dataset. Specifically, in order to accomplish this task, this project will use constrained beam search algorithms proposed by Anderson et al. (2016) in *Guided Open Vocabulary Image Captioning with Constrained Beam Search*. By completing the project, it is expected that we can find some useful annotations which can contribute to the improvement of image captioning model. Moreover, this project will be divided into 3 phases:

1. Constructing the model to caption the images within ImageNet by using constrained beam search. This phase is expected to take 12 weeks to finish.

---

Research School of Computer Science

*Form updated Jun 2018*
2. Analysing the outcome of phase 1, which will take about 2 weeks.

3. Improving the captioning model base on the result of phase 2 (8 weeks).

After the end phase of the project, an intern-report is required.

ASSessment (as per the project course’s rules web page, with any differences noted below).

<table>
<thead>
<tr>
<th>Assessed project components</th>
<th>% of mark</th>
<th>Due date</th>
<th>Evaluated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report: style: Project report (e.g. research report, software description...)</td>
<td>(min 45, def 60) 60%</td>
<td>end of semester 1, 2020.</td>
<td>(examiner)</td>
</tr>
<tr>
<td>Artefact: kind: Software (e.g. software, user interface, robot...)</td>
<td>(max 45, def 30) 30%</td>
<td>end of semester 1, 2020.</td>
<td>(supervisor) Stephen Gould</td>
</tr>
<tr>
<td>Presentation:</td>
<td>10%</td>
<td></td>
<td>(course convenor)</td>
</tr>
</tbody>
</table>

MEETING DATES (IF KNOWN):

STUDENT DECLARATION: I agree to fulfil the above defined contract:

......Songtuan Lin......
Signature

...23/7/2019...................
Date

SECTION B (Supervisor):

I am willing to supervise and support this project. I have checked the student's academic record and believe this student can complete the project. I nominate the following examiner, and have obtained their consent to review the report (via signature below or attached email)

..............................................
Signature

24 July 2019
Date

Examiner: Ramesh Sankaranarayana
Name: ..............................................

(Nominated examiners may be subject to change on request by the supervisor or course convenor)

REQUIRED DEPARTMENT RESOURCES:
Research School of Computer Science

Form updated Jun 2018
1. Access to GPU server (paloalto).
2. Weekly meetings: Thursday 11:00 am – 11:30 am.

SECTION C (Course convenor approval)

Signature

Date 25/7/14
Appendix B

Artefacts

B.1 File Structure

The files included in our artefact are organized as below:

```
Captioning-ImageNet-Pythia
  dataset
    __imagenet-dataset.py
    __customized-dataset.py
  model
    __butd.py
  modules
    __captioner.py
    __constrained-beam-search.py
    __rcnn-encoder.py
    __vocabulary.py
  utils
    __finite-automata.py
    __table-tensor.py
    README.md
    __caption-imagenet.py
    __data-reader.py
```

B.2 Contributions

As mentioned in project report, we used a framework called Pythiy provided by facebook research to build our own BUTD model. As a result, a small part of code in files `captioner.py`, `rcnn-decoder.py` and `butd.py`, which implement the BUTD model in our project, have consulted the source code of Pythia. Except these, all work in this project are completed individually.

B.3 Usage and Features

The usage and features of this code implementation can be found in the README file.
Appendix C

README
Captioning ImageNet

This project captions the images within ImageNet dataset in a semi-autonomous way. Specifically, we used state-of-the-art caption generator plus constrained beam search algorithm to accomplish this task.

Motivation

This project is motivated by the situation that the available datasets used to train image captioning model are limited to Microsoft COCO and Flickr. Therefore, we want to caption the images in ImageNet and extend the available datasets.

Framework Used

The caption generator in this project is built upon Pythia, which is developed by facebook research group and provides a pre-trained BUTD caption generator. Moreover, since Pythia rely on Pytorch, this project also requires Pytorch installed.

Features

The basic feature of this project is that it can generate caption for images in ImageNet by using the method described in the project report. However, the most exciting part about this project is that it can accept almost any regular expression which specify the format of caption and implements this regular expression as constrained beam search. More precisely, this project can be used to caption arbitrary image with it's generated caption follow some constrains specified by user-defined regular expression.

Installation

In order to use this project, Pythia should first be installed:

```
git clone https://github.com/Songtuan-Lin/pythia.git
cd pythia/
git reset --hard 3325b89023472f9307b4e665e6429dbcb391d77
sed -i '/torch/d' requirements.txt
pip install -e .
cd vqa-maskrcnn-benchmark/
python setup.py build develop
```

If the installation failed, check whether all dependencies are installed:

```
pip install ninja yacs cython matlablib demjson
```

Then, clone this repo and make a directory called `model_data`. This `model_data` directory is used to hold pre-trained model data:

```
cd Captioning-ImageNet-Pythia/
mkdir model_data/
```

Finally, download pre-trained model data:

```
wget -O model_data/butd.yaml https://dl.fbaipublicfiles.com/pythia/pretrained_models/coco_captions/butd.yaml
wget -O /model_data/detectron_model.yaml https://dl.fbaipublicfiles.com/pythia/detectron_model/detectron_model.yaml
wget model_data/detectron_weights.tar.gz https://dl.fbaipublicfiles.com/pythia/data/detectron_weights.tar.gz
tar xf model_data/detectron_weights.tar.gz
```

Now, we are ready to go!

Usage

To caption the images in ImageNet, simply execute `caption_imagenet.py` file by giving it three command line arguments: the root directory of ImageNet dataset, the target directory to hold the caption results and the upper levels of ImageNet tag to trace(mentioned in project report)
To support input regular expression, we also provide following classes:

1. `utils.finite_automata.FiniteAutomata`: Construct finite automata by giving a regular expression as input.
2. `utils.table_tensor.TableTensor`: Transfer transition tables of a finite automata to Pytorch tensor.
3. `dataset.customized_dataset.CustomizedDataset`: Load arbitrary dataset and transition tables which are represented as Pytorch tensor.

Additionally, regular expression should be consist of following symbols:

1. `.`: Match any single character.
2. `?`: Match zero or more occurrences of the preceding element.
3. `(`: Worked as delimiter, do not match any symbol.
4. `)`: the same as `(`.
5. `a-zA-Z`: Alphabet
6. `space`: used to seperate token, does not match any symbol.

Particularly, wildcard matching can be replaced as: `(.?)`. Moreover, it is strongly suggest that using `(` and `)` to seperate each component in the regular expression. For example, if we want to input regular expression 'dog|cat', we strongly recommand rewrite it as `(dog|cat)`.

The example which demonstrate how to construct and use regular expression will be presented in Demo.

**Demo**

The following code snippet demonstrate how to construct finite automata and visualize it by giving a regular expression:

```python
from utils.finite_automata import FiniteAutomata
from utils.table_tensor import TableTensor

reg = '.?(animal|bird).?'
nfa = FiniteAutomata(reg)
nfa.visualize()
```

The complete demo, which shows how to use our code to caption ImageNet and how to caption image with arbitrary regular expression, can be find here:

Open in Colab

**API Reference**

class `utils.finite_automata.FiniteAutomata` (reg): This class take an input regular expression and produce the corresponding NFA. The main class methods include:

1. `transitions()`: This method returns the transition table which is corresponding to input regular expression.
2. `visualize()`: This method visualize the finite automata.

class `utils.table_tensor.TableTensor`(vocab, table): This class takes two arguments: a pre-defined vocabulary and a transition table produced by class FiniteAutomata. The main class methods include:

1. `to_tensors()`: Convert the transition table to Pytorch tensor.

class `dataset.customized_dataset.CustomizedDataset`(root_dir, transitions): This class takes a file directory and a transition table which produced by TableTensor as arguments and construct a Pytorch dataset.

**Implementation Note**

The core of our implementation is how we represent the transition table of a finite automata as Pytorch tensor. In our code, we represent the transition table as a list of tensors. This list contains k tensors, where k equals to the number of states in finite automata. The ith tensor in the list indicates which tokens in the vocabulary can trigger the state transition from another states to state i. More precisely, if we denote the ith tensor in the list as Ti, then, Ti has size (num_states, vocab_size) and the transition table is interpreted as:

1. If Ti[j, k] = 0, then the kth token in the vocabulary can trigger the state transition from state j to state i.
2. If Ti[j, k] = 1, otherwise.

By representing transition table as list of tensor, we can then implement constrained beam search. This part of code has been well commented and hence will not be explained here.

**Script to Download Captioned Data**

Since our captioned images are stored in AWS S3 storage, which can only be accessed under certain permission, we provide a script to download captioned data:

Open in Colab