

# **CAPTION GENERATOR USING IMAGE ANALYTICS**

*A Report submitted*

*in partial fulfillment for the Course*

**DATAWAREHOUSING AND MINING**

*by*

E SAI SRIKAR 190330064

pursued in

**Department of Computer Science and Engineering**



**Aziznagar, Moinabad Road, Near TS Police Academy,  
Hyderabad, Telangana-500075, India.**

**NOVEMBER, 2021**

# CERTIFICATE

This is to certify that the project report entitled **REAL TIME CAPTION GENERATOR USING IMAGE ANALYTICS** submitted by E Sai Srikar to the KL University Hyderabad Campus, in partial fulfillment for the Course DATAWAREHOUSING AND MINING, of the degree of **B. Tech in Computer Science and Engineering** is a bonafide record of project work carried out by us under our supervision. The contents of this report, in full or in parts, have not been submitted to any other Institution or University for the award of any degree or diploma.

Dr KVSN Rama Rao

Dr Anoop Tiwari

Dr G Kiran Kumar

**Supervisors**

Dr KVSN Rama Rao

**Course-Coordinator**

Counter signature of HOD with seal

# DECLARATION

We declare that this project report titled **REAL TIME CAPTION GENERATOR USING IMAGE ANALYTICS** submitted in partial fulfillment of the Course **DATAWAREHOUSING AND MINING** of **B. Tech in Computer Science and Engineering**, is a record of original work carried out by us under the supervision of Dr KVS N Rama Rao, Dr Anoop Tiwari, Dr G Kiran Kumar and has not formed the basis for the award of any other degree or diploma, in this or any other Institution or University. In keeping with the ethical practice in reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

E Sai Srikar 190330064

10<sup>th</sup> November 2021

## ACKNOWLEDGMENTS

Before getting into the thickest of things, we would like to thank the personalities who were part of my project in numerous ways, those who gave me outstanding support from birth of the project.

We are extremely thankful to our beloved management providing necessary infrastructure and resources for the accomplishment of our project at **KLH, Hyderabad.**

We are highly indebted to Dr. L. Koteswara Rao, Principal of **KLH, Hyderabad**, for his support during the tenure of the project.

We are very much obliged to our beloved **Dr. E. Anupriya, Head of the Department of Computer Science & Engineering**, KLH, for providing the opportunity to undertake this project and encouragement in completion of this project.

We hereby wish to express our deep sense of gratitude to **Dr KVSN Rama Rao, Dr Anoop Tiwari, Dr G Kiran Kumar**, Department of Computer Science and Engineering, KLH, for the esteemed guidance, moral support and invaluable advice provided by them for the success of the project.

We are also thankful to all the staff members of Computer Science and Engineering department who have cooperated in making our project a success. We would like to thank all our parents and friends who extended their help, encouragement, and moral support to complete our project work.

Thanks for your valuable guidance and kind support.

E Sai Srikar 190330064

# REAL TIME CAPTION GENERATOR USING IMAGE ANALYTICS

## Abstract

In this project, we systematically analyze a deep neural networks based image caption generation method. With an image as the input, the method can output an English sentence describing the content in the image. We analyze three components of the method: convolutional neural network (CNN), recurrent neural network (RNN) and sentence generation. By replacing the CNN part with three state-of-the-art architectures, we also propose a simplified version the Gated Recurrent Units (GRU) as a new recurrent layer, implementing both Python and Keras library in Jupyter Notebook. The simplified GRU achieves comparable result when it is compared with the long short-term memory (LSTM) method. But it has few parameters which saves memory and is faster in training. Finally, we generate multiple sentences using Beam Search. The experiments show that the modified method can generate captions comparable to the-state-of-the-art methods with less training memory.

## CONTENTS

CERTIFICATE	i
DECLARATION	ii
ACKNOWLEDGEMENT	iii
ABSTRACT	iv
CONTENTS	v
LIST OF FIURES	vi
1.INTRODUCTION	1
2.LITERATURE SURVEY	3
3.ANALYSIS	7
4.IMPLEMENTATION	14
5.RESULT AND DISCUSSION	15
6.CONCLUSION	24
7.REFERENCES	26

## LIST OF FIGURES

FIG.NO. NO.	TITLE	PAGE
1	Image caption generation pipeline	2
2	The visual-semantic alignment method	6
3	Three variations of the LRCN image captioning architecture	12
4	Our Data Science Project Life Cycle	24

# CHAPTER 1

## INTRODUCTION

Automatically describing the content of images using natural languages is a fundamental and challenging task. It has great potential impact. For example, it could help visually impaired people better understand the content of images on the web. Also, it could provide more accurate and compact information of images/videos in scenarios such as image sharing in social network or video surveillance systems. This project accomplishes this task using deep neural networks. By learning knowledge from image and caption pairs, the method can generate image captions that are usually semantically descriptive and grammatically correct.

Human beings usually describe a scene using natural languages which are concise and compact. However, machine vision systems describes the scene by taking an image which is a two dimension arrays. From this perspective, Vinyal et al. (Vinyals et al., ) models the image captioning problem as a language translation problem in their Neural Image Caption (NIC) generator system. The idea is mapping the image and captions to the same space and learning a mapping from the image to the sentences. Donahue et al. (Donahue et al., ) proposed a more general Long-term Recurrent Convolutional Network (LRCN) method. The LRCN method not only models the one-to-many (words) image captioning, but also models many-to-one action generation and many-to-many video description. They also provides publicly available implementation based on Caffe framework (Jia et al., 2014), which further boosts the research on image captioning. This work is based on the LRCN method.

Although all the mappings are learned in an end to-end framework, we believe the benefits of better understanding of the system by analyzing different components separately. Fig. 1 shows the pipeline. The model has three components. The first component is a CNN which is used to understand the content of the image. Image understanding answers the typical questions in computer vision such as “What are the objects?”, “Where are the objects?” and “How are the objects interactive?”. For example, the CNN has to recognize the “teddy bear”, “table” and their relative locations in the image. The second component is a RNN which is used to generate a sentence given the visual feature. For example, the RNN has to generate a sequence of probabilities of words given two words “teddy bear, table”. The third component is used to generate a sentence by exploring the combination of the probabilities. This component is less studied in the reference paper(Donahue et al., ). This project aims at understanding the impact of different components of the LRCN method (Donahue et al., ).We have following contributions:

- understand the LRCN method at the implementation level.
- analyze the influence of the CNN component by replacing three CNN architectures (two from author’s and one from our implementation).
- analyze the influence of the RNN component by replacing two RNN architectures. (one from author’s and one from our implementation).



- analyze the influence of sentence generation method by comparing two methods (one from author's and one from our implementation).

1

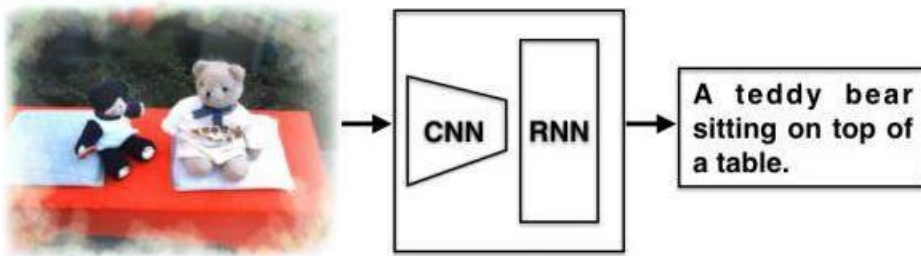


Figure 1: Image caption generation pipeline. The framework consists of a convolutional neural network (CNN) followed by a recurrent neural network (RNN). It generates an English sentence from an input image.

## Chapter 2

### Literature Survey

Automatically describing the content of an image is a fundamental problem in artificial intelligence that connects computer vision and natural language processing. Earlier methods first generate annotations (i.e., nouns and adjectives) from images (Sermanet et al., 2013; Russakovsky et al., 2015), then generate a sentence from the annotations (Gupta and Mannem, ). Donahue *et al.* (Donahue et al., ) developed a recurrent convolutional architecture suitable for large-scale visual learning, and demonstrated the value of the models on three different tasks: video recognition, image description and video description. In these models, long-term dependencies are incorporated into the network state updates and are end-to-end trainable. The limitation is the difficulty of understanding the intermediate result. The LRCN method is further developed to text generation from videos (Venugopalan et al., ).

Instead of one architecture for three tasks in LRCN, Vinyals *et al.* (Vinyals et al., ) proposed a neural image caption (NIC) model only for the image caption generation. Combining the GoogLeNet and single layer of LSTM, this model is trained to maximize the likelihood of the target description sentence given the training images. The performance of the model is evaluated qualitatively and quantitatively. This method was ranked first in the MS COCO Captioning Challenge (2015) in which the result was judged by humans. Comparing LRCN with NIC, we find three differences that may indicate the performance differences. First, NIC uses GoogLeNet while LRCN uses VGGNet. Second, NIC inputs visual feature only into the first unit of LSTM while LRCN inputs the visual feature into every LSTM unit. Third, NIC has simpler RNN architecture (single layer LSTM) than LRCN (two factored LSTM layers). We verified that the mathematical models of LRCN and NIC are exactly the same for image captioning. The performance difference lies in the implementation and LRCN has to trade off between simplicity and generality, as it is designed for three different tasks.



Instead of end-to-end learning, Fang *et al.* (Fang et al., ) presented a visual concepts based method. First, they used multiple instance learning to train visual detectors of words that commonly occur in captions such as nouns, verbs, and adjectives. Then, they trained a language model with a set of over 400,000 image descriptions to capture the statistics of word usage. Finally, they re-ranked caption candidates using sentence-level features and a deep multi-modal similarity model. Their captions have equal or better quality 34% of the time than those written by human beings. The limitation of the method is that it has more human controlled parameters which make the system less re-producible. We believe the web application **captionbot** (Microsoft, ) is based on this method.

Karpathy *et al.* (Karpathy and Fei-Fei, ) proposed a visual-semantic alignment (VSA) method. The method generates descriptions of different regions of an image in the form of words or sentences (see Fig. 2). Technically, the method replaces the CNN with Region-based convolutional Networks (RCNN) so that the extracted visual features are aligned to particular regions of the image. The experiment shows that the generated descriptions significantly outperform retrieval baselines on both full images and on a new dataset of region-level annotations. This method generates more diverse and accurate descriptions than the whole image method such as LRCN and NIC. The limitation is that the method consists of two separate models. This method is further developed to dense captioning (Johnson et al., 2016) and image based question and answering system (Zhu et al., 2016).

5



Figure 2: The visual-semantic alignment method can generate descriptions of

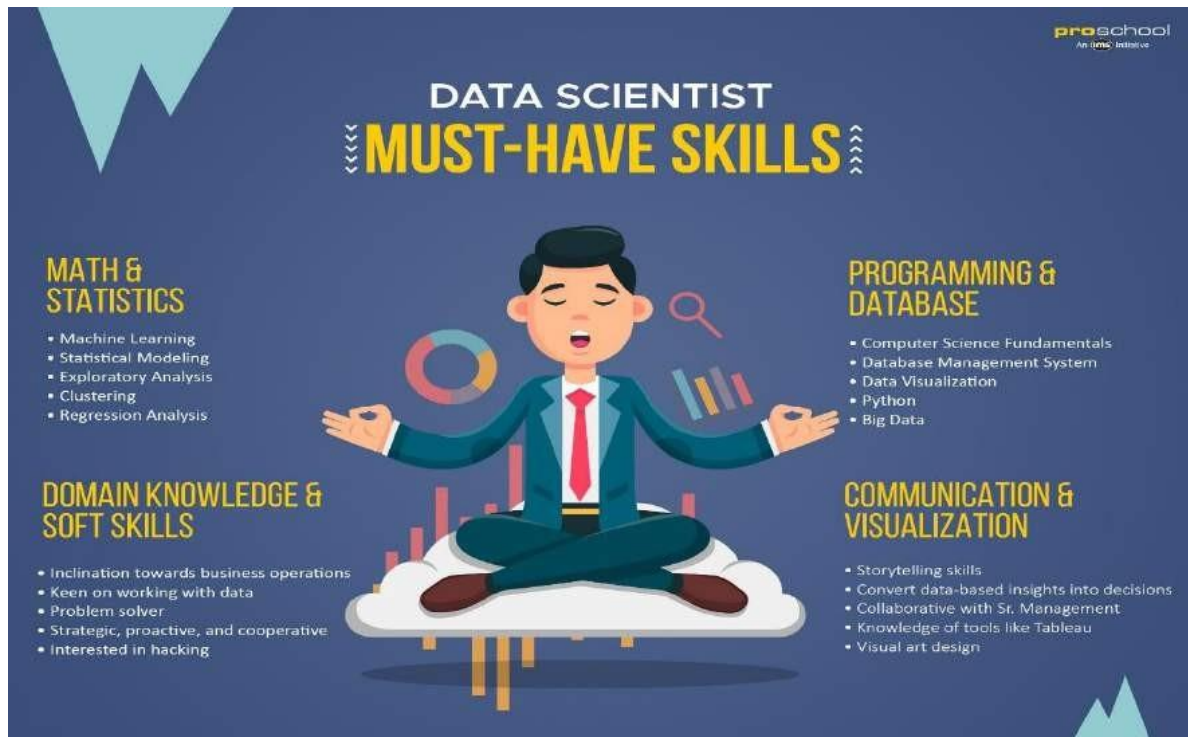
## **Chapter 3**

### **ANALYSIS**

#### **1.Our Problem Statement**

- In today's world, technology plays a vital role, without technology it is hard to imagine our day to day life. Everyday an innovation is taking place in some part of the world. Image recognition and computer vision are such developments that are taking the world by storm.
- You saw an image and your brain can easily tell what the image is about, but can a computer tell what the image is representing? Computer vision researchers worked on this a lot and they considered it impossible until now!
- With the advancement in Deep learning techniques, availability of huge datasets, and computing power of data science, we can build models that can tell what an image is, what objects are seen.





7

## 2. Description Of Our Problem

**Task** In this project, we want to build a system that can generate an English sentence that describes objects, actions or events in an RGB image:

$$S = f(I) \quad (1)$$

where  $I$  is an RGB image and  $S$  is a sentence,  $f$  is the function that we want to learn.

**Corpus** We use the MS COCO Caption (Chen et al., 2015) as the corpus. The captions are gath-

ered from human beings using Amazon’s Mechanical Turk (AMT). We manually checked some examples by side-by-side comparing the image and corresponding sentences. We found the captions are very expressive and diverse. The COCO Caption is the largest image caption corpus at the time of writing. There are 413,915 captions for 82,783 images in training, 202,520 captions for 40,504 images in validation and 379,249 captions for 40,775 images in testing. Each image has at least 5 captions. The captions for training and validation are publicly available while the captions for testing is reserved by the authors. In the experiment, we use all the training data in the training process and 1,000 randomly selected validation data in the testing process.

### **3.Our Proposed Solution**



For image caption generation, LRCN maximizes the probability of the description giving the image:

$$\theta^* = \arg \max_{\theta} \sum_{(I,S)} \log p(S|I; \theta) \quad (2)$$

where  $\theta$  are the parameters of the model,  $I$  is an image, and  $S$  is a sample sentence. Let the length of the sentence be  $N$ , the method applies the chain rule to model the joint probability over  $S_0, \dots, S_N$ :

$$\log p(S|I) = \sum_{t=0}^N \log p(S_t|I, S_0, \dots, S_{t-1}) \quad (3)$$

where the  $\theta$  is dropped for convenience,  $S_t$  is the word at step  $t$ .

The model has two parts. The first part is a CNN which maps the image to a fixed-length visual feature. The visual feature is embedded to as the input  $v$  to the RNN.

$$v = W_v(\text{CNN}(I)) \quad (4)$$

where  $W_v$  is the visual feature embedding. The visual feature is fixed for each step of the RNN.

$$v = W_v(\text{CNN}(I)) \quad (4)$$

where  $W_v$  is the visual feature embedding. The visual feature is fixed for each step of the RNN.

In the RNN, each word is represented a one-hot vector  $S_t$  of dimension equal to the size of the dictionary.  $S_0$  and  $S_N$  are for special start and stop words. The word embedding parameter is  $W_s$ :

$$x_t = W_t S_t, \quad t \in \{0 \cdots N - 1\} \quad (5)$$

In this way, the image and words are mapped to the same space. After the internal processing of the RNN, the features  $v$ ,  $x_t$  and internal hidden parameter  $h_t$  are decoded into a probability to predict the word at current time:

$$p_{t+1} = LSTM(v, x_t, h_t), \quad t \in \{0 \cdots N - 1\} \quad (6)$$

Because a sentence with higher probability does not necessary mean this sentence is more accurate than other candidate sentences, post-processing method such as **Beam Search** is used to generate more sentences and pick top-K sentences.

### 3.1 Convolutional Neural Networks

In this project, a convolutional neural network (CNN) maps an RGB image to a visual feature vector. The CNN has three most-used layers: convolution, pooling and fully-connected layers. Also, Rectified Linear Units (ReLU)  $f(x) = \max(0, x)$  is used as the non-linear active function. The ReLU is faster than the traditional  $f(x) = \tanh(x)$  or  $f(x) = (1 + e^{-x})^{-1}$ . Dropout layer is used to prevent overfitting. The dropout sets the output of each hidden neuron to zero with a probability (i.e., 0.5). The “dropped out” neurons do not contribute to the forward pass and do not participate in back-propagation.

The AlexNet (Krizhevsky et al., 2012), VGGNet (Simonyan and Zisserman, 2014) and GoogLeNet (Szegedy et al., 2015) are three widely used deep convolutional neural network architecture. They share the convolution  $\rightarrow$  pooling  $\rightarrow$  fully-connection  $\rightarrow$  loss function pipeline but with different shapes and connections of layers, especially the convolution layer. AlexNet is the first deep convolutional neural network used in large scale image classification. VGGNet and GoogLeNet achieves the-start-of-the-art performance in ImageNet recognition challenge 2014 and 2015.



When the CNN combines the RNN, there are specific considerations of convergence since both of them has millions parameters. For example, Vinyals *et al.* (Vinyals et al., ) found that it is better to fix the parameters of the convolutional layer as the parameters trained from the ImageNet. As a result, only the non-convolution layer parameters in CNN and the RNN parameters are actually learned from caption examples.

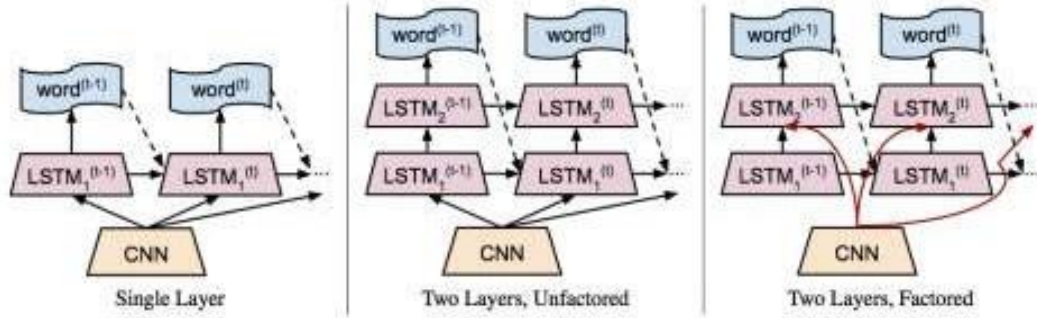


Figure 3 - Three variations of the LRCN image captioning architecture. The most right two-layers factored LSTM is used in the method. Figure from (Donahue et al., ).

### 3.2 Recurrent Neural Network (RNN)

To prevent the gradients vanishing problem, the long short-term memory (LSTM) method is used as the RNN component. A simplified LSTM updates for time step  $t$  given inputs  $x_t$ ,  $h_{t-1}$ , and  $c_{t-1}$  are:

$$\begin{aligned}
 i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\
 f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\
 o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\
 g_t &= \phi(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
 c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
 h_t &= o_t \odot \phi(c_t)
 \end{aligned} \tag{7}$$

where  $\sigma(x) = (1 + e^{-x})^{-1}$  and  $\phi(x) = 2\sigma(2x) - 1$ . In addition to a hidden unit  $h_t \in R^N$ , the LSTM includes an input gate  $i_t \in R^N$ , forget gate  $f_t \in R^N$ , output gate  $o_t \in R^N$ , input modulation gate  $g_t \in R^N$ , and memory cell  $c_t \in R^N$ . These additional cells enable the LSTM to learn extremely complex and long-term temporal dynamics. Additional depth can be added to LSTMs by stacking them on top of each other. Fig. 3 shows three version of LSTMs. The two-layers factored LSTM achieves the best performance and is used in the method.

In this project, we proposed a simplified version of GRU in section 5.1 which also avoids the vanishing gradient problem and can be easily implemented in Caffe based on the current Caffe LSTM framework. We also provide the MATLAB program in the Appendices verifying our derivation of BPTT on the original GRU model.

### 3.3 Sentence Generation

The output of LSTM is the probability of each word in the vocabulary. **Beam search** is used to generate sentences. Beam search is a heuristic search algorithm that explores a graph by expanding the most promising node in a limited set. In addition to beam search, we also use k-best search to generate sentences. It is very similar to the time synchronous Viterbi search. The method iteratively selects the  $k$  best sentences from all the candidate sentences up to time  $t$ , and keeps only the resulting best  $k$  of them.



## Chapter 4

### Implementation

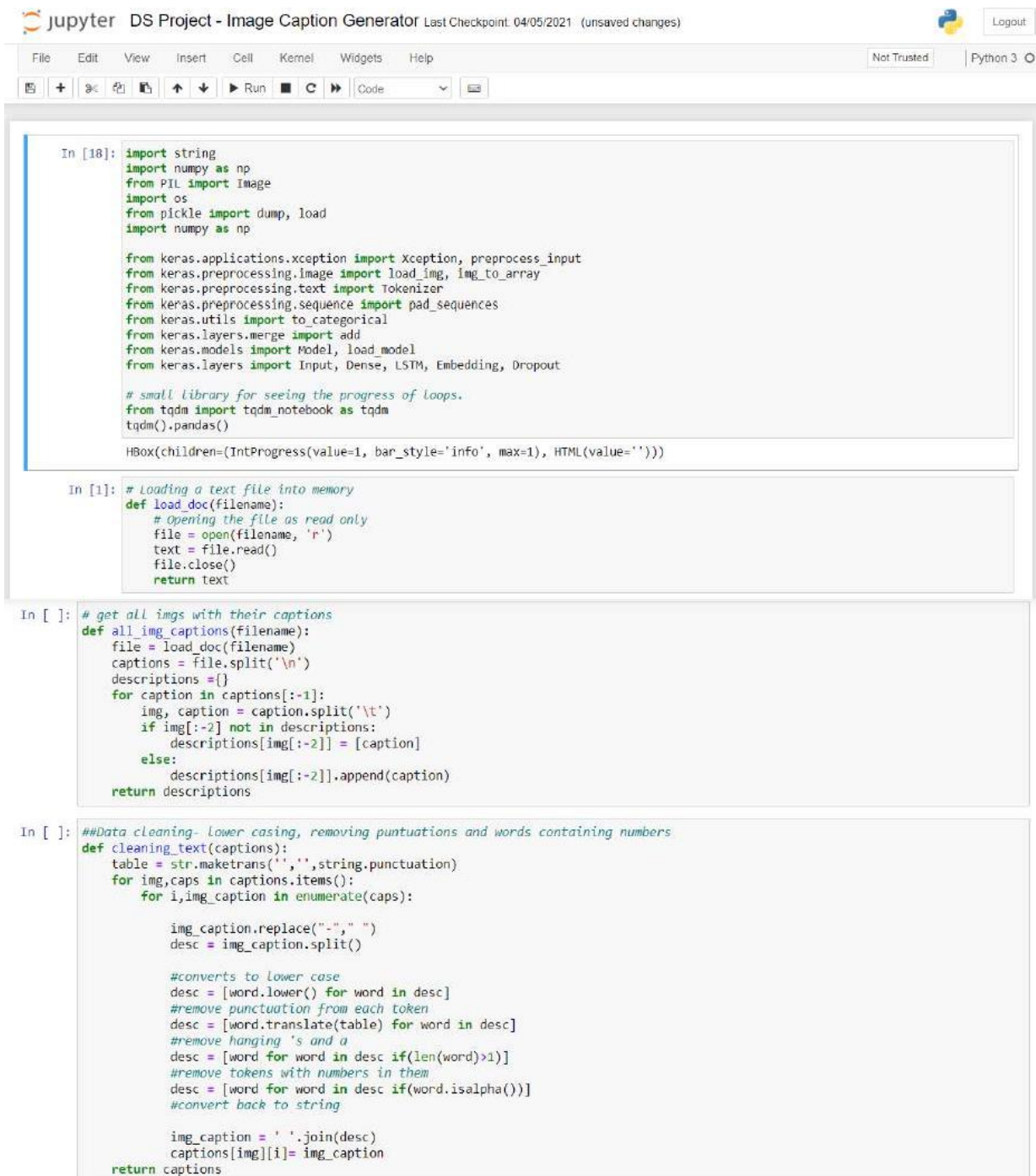
**Preprocessing** Because we want to keep the architecture of the CNN, the input image are randomly cropped to the size of  $224 \times 224$ . As a result, only part of the images are used in training at particular iteration. Because one image will be cropped multiple times in the training, the CNN can probably see the whole image in the training (once for part of the image). However, the method only sees part of the image in the testing except the dense cropping is also used (our project does not use dense crop). For the sentences, the method first creates a vocabulary only from the training captions and removes lower frequency words (less than 5). Then, words are represented by one-hot vectors.

### Training Method

The neural network is trained using the mini-patch stochastic gradient descent (SGD) method. The base learning rate is 0.01. The learning rate drops 50% in every 20,000 iterations. Because the number of training samples is much smaller than the number of parameters of the neural network, overfitting is our big concern. Besides the dropout layer, we fixed the parameters of the convolutional layers as suggested by (Vinyals et al., ). All the network are trained in a Linux machine with a Tesla K40c graphic card with 12Gb memory.

# Chapter 5

## RESULT AND DISCUSSION



The image shows a Jupyter Notebook interface for a project titled "DS Project - Image Caption Generator". The interface includes a top bar with the Jupyter logo, the project name, a last checkpoint timestamp of "04/05/2021", and a "Logout" button. Below this is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. On the right side of the menu bar, it says "Not Trusted" and "Python 3". A toolbar with various icons for file operations and execution is located below the menu bar. The main area contains three code cells. The first cell, labeled "In [18]:", contains a series of import statements for modules like string, numpy, PIL, os, pickle, keras, and tqdm. The second cell, labeled "In [1]:", defines a function named "load\_doc" that takes a filename and returns the text content of the file. The third cell, labeled "In [ ]:", defines a function named "all\_img\_captions" that takes a filename and returns a list of image captions and descriptions. The code is written in Python and uses various libraries for image processing and deep learning.

```
In [18]: import string
import numpy as np
from PIL import Image
import os
from pickle import dump, load
import numpy as np

from keras.applications.xception import Xception, preprocess_input
from keras.preprocessing.image import load_img, img_to_array
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.layers.merge import add
from keras.models import Model, load_model
from keras.layers import Input, Dense, LSTM, Embedding, Dropout

# small library for seeing the progress of loops.
from tqdm import tqdm_notebook as tqdm
tqdm().pandas()

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

In [1]: # Loading a text file into memory
def load_doc(filename):
    # Opening the file as read only
    file = open(filename, 'r')
    text = file.read()
    file.close()
    return text

In [ ]: # get all imgs with their captions
def all_img_captions(filename):
    file = load_doc(filename)
    captions = file.split('\n')
    descriptions = {}
    for caption in captions[:-1]:
        img, caption = caption.split('\t')
        if img[:-2] not in descriptions:
            descriptions[img[:-2]] = [caption]
        else:
            descriptions[img[:-2]].append(caption)
    return descriptions

In [ ]: ##Data cleaning- lower casing, removing punctuations and words containing numbers
def cleaning_text(captions):
    table = str.maketrans('', '', string.punctuation)
    for img,caps in captions.items():
        for i,img_caption in enumerate(caps):
            img_caption.replace("-", " ")
            desc = img_caption.split()

            #converts to lower case
            desc = [word.lower() for word in desc]
            #remove punctuation from each token
            desc = [word.translate(table) for word in desc]
            #remove hanging 's and a
            desc = [word for word in desc if(len(word)>1)]
            #remove tokens with numbers in them
            desc = [word for word in desc if(word.isalpha())]
            #convert back to string
            img_caption = ' '.join(desc)
            captions[img][i] = img_caption
    return captions
```

```

In [21]: def text_vocabulary(descriptions):
# build vocabulary of all unique words
vocab = set()

for key in descriptions.keys():
    [vocab.update(d.split()) for d in descriptions[key]]

return vocab

In [22]: #All descriptions in one file
def save_descriptions(descriptions, filename):
    lines = list()
    for key, desc_list in descriptions.items():
        for desc in desc_list:
            lines.append(key + '\t' + desc )
    data = "\n".join(lines)
    file = open(filename, "w")
    file.write(data)
    file.close()

In [24]: dataset_text = "D:\dataflair projects\Project - Image Caption Generator\Flickr 8k text"
dataset_images = "D:\dataflair projects\Project - Image Caption Generator\Flickr8k_Dataset"

In [26]: #we prepare our text data
filename = dataset_text + "/" + "Flickr8k.token.txt"
#loading the file that contains all data
#mapping them into descriptions dictionary img to 5 captions
descriptions = all_img_captions(filename)
print("Length of descriptions =" ,len(descriptions))

#cleaning the descriptions
clean_descriptions = cleaning_text(descriptions)

#building vocabulary
vocabulary = text_vocabulary(clean_descriptions)
print("Length of vocabulary = " , len(vocabulary))

#saving each description to file
save_descriptions(clean_descriptions, "descriptions.txt")

Length of descriptions = 8092
Length of vocabulary = 8763

In [28]: def extract_features(directory):
    model = xception( include_top=False, pooling='avg' )
    features = {}
    for img in tqdm(os.listdir(directory)):
        filename = directory + "/" + img
        image = Image.open(filename)
        image = image.resize((299,299))
        image = np.expand_dims(image, axis=0)
        #image = preprocess_input(image)
        image = image/127.5
        image = image - 1.0

        feature = model.predict(image)
        features[img] = feature
    return features

In [29]: #2048 feature vector
features = extract_features(dataset_images)
dump(features, open("features.p", "wb"))

In [19]: features = load(open("features.p", "rb"))

```



```
In [20]: #Load the data
def load_photos(filename):
    file = load_doc(filename)
    photos = file.split("\n")[:-1]
    return photos

def load_clean_descriptions(filename, photos):
    #loading clean descriptions
    file = load_doc(filename)
    descriptions = {}
    for line in file.split("\n"):

        words = line.split()
        if len(words)<1:
            continue

        image, image_caption = words[0], words[1:]

        if image in photos:
            if image not in descriptions:
                descriptions[image] = []
            desc = '<start> ' + " ".join(image_caption) + ' <end>'
            descriptions[image].append(desc)

    return descriptions

def load_features(photos):
    #loading all features
    all_features = load(open("features.p", "rb"))
    #selecting only needed features
    features = {k:all_features[k] for k in photos}
    return features
```

```
In [21]: filename = dataset_text + "/" + "Flickr_8k.trainImages.txt"

#train = loading_data(filename)
train_imgs = load_photos(filename)
train_descriptions = load_clean_descriptions("descriptions.txt", train_imgs)
train_features = load_features(train_imgs)
```

```
In [22]: #converting dictionary to clean list of descriptions
def dict_to_list(descriptions):
    all_desc = []
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc

#creating tokenizer class
#this will vectorise text corpus
#each integer will represent token in dictionary

from keras.preprocessing.text import Tokenizer

def create_tokenizer(descriptions):
    desc_list = dict_to_list(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(desc_list)
    return tokenizer
```

```
In [23]: # give each word a index, and store that into tokenizer.p pickle file
tokenizer = create_tokenizer(train_descriptions)
dump(tokenizer, open('tokenizer.p', 'wb'))
vocab_size = len(tokenizer.word_index) + 1
vocab_size
```

Out[23]: 7577

```
In [24]: #calculate maximum length of descriptions
def max_length(descriptions):
    desc_list = dict_to_list(descriptions)
    return max(len(d.split()) for d in desc_list)

max_length = max_length(descriptions)
max_length
```

Out[24]: 32

```
In [25]: features['1000268201_693b08cb0e.jpg'][0]
```

```
Out[25]: array([0.36452794, 0.12713662, 0.0013574 , ..., 0.221817 , 0.01178991,
0.24176797], dtype=float32)
```

```
In [26]: # Define the model

#1 Photo feature extractor - we extracted features from pretrained model Xception.
#2 Sequence processor - word embedding layer that handles text, followed by LSTM
#3 Decoder - Both 1 and 2 model produce fixed length vector. They are merged together and processed by dense layer to make final
```

```
In [27]: #create input-output sequence pairs from the image description.

#data generator, used by model.fit_generator()
def data_generator(descriptions, features, tokenizer, max_length):
    while 1:
        for key, description_list in descriptions.items():
            #retrieve photo features
            feature = features[key][0]
            input_image, input_sequence, output_word = create_sequences(tokenizer, max_length, description_list, feature)
            yield [[input_image, input_sequence], output_word]

def create_sequences(tokenizer, max_length, desc_list, feature):
    X1, X2, y = list(), list(), list()
    # walk through each description for the image
    for desc in desc_list:
        # encode the sequence
        seq = tokenizer.texts_to_sequences([desc])[0]
        # split one sequence into multiple X,y pairs
        for i in range(1, len(seq)):
            # split into input and output pair
            in_seq, out_seq = seq[:i], seq[i]
            # pad input sequence
            in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
            # encode output sequence
            out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
            # store
            X1.append(feature)
            X2.append(in_seq)
            y.append(out_seq)
    return np.array(X1), np.array(X2), np.array(y)
```

```
In [28]: [a,b],c = next(data_generator(train_descriptions, features, tokenizer, max_length))
a.shape, b.shape, c.shape
```

```
Out[28]: ((47, 2048), (47, 32), (47, 7577))
```

```
In [29]: from keras.utils import plot_model

# define the captioning model
def define_model(vocab_size, max_length):

    # features from the CNN model squeezed from 2048 to 256 nodes
    inputs1 = Input(shape=(2048,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)

    # LSTM sequence model
    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(256)(se2)

    # Merging both models
    decoder1 = add([fe2, se3])
    decoder2 = Dense(256, activation='relu')(decoder1)
    outputs = Dense(vocab_size, activation='softmax')(decoder2)

    # tie it together [image, seq] [word]
    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer='adam')

    # summarize model
    print(model.summary())
    plot_model(model, to_file='model.png', show_shapes=True)

    return model
```

```
In [30]: # train our model
print('Dataset: ', len(train_imgs))
print('Descriptions: train=', len(train_descriptions))
print('Photos: train=', len(train_features))
print('Vocabulary Size:', vocab_size)
print('Description Length: ', max_length)

model = define_model(vocab_size, max_length)
epochs = 10
steps = len(train_descriptions)
# making a directory models to save our models
os.mkdir("models")
for i in range(epochs):
    generator = data_generator(train_descriptions, train_features, tokenizer, max_length)
    model.fit_generator(generator, epochs=1, steps_per_epoch= steps, verbose=1)
    model.save("models/model_" + str(i) + ".h5")
```

```
Dataset: 6000
Descriptions: train= 6000
Photos: train= 6000
Vocabulary Size: 7577
Description Length: 32
```

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 32)	0	
input_1 (InputLayer)	(None, 2048)	0	
embedding_1 (Embedding)	(None, 32, 256)	1939712	input_2[0][0]
dropout_1 (Dropout)	(None, 2048)	0	input_1[0][0]
dropout_2 (Dropout)	(None, 32, 256)	0	embedding_1[0][0]
dense_1 (Dense)	(None, 256)	524544	dropout_1[0][0]

## Working of the model: Example 1



This is the image we are feeding to the model as input

```
Command Prompt
start young boy in red shirt is running through the grass end
C:\Users\saisr\Downloads\python-project-image-caption-generator>python testing_caption_generator.py -i "C:\Users\saisr\Downloads\Flicker8K-Dataset\Flicker8K-Dataset\168321294_4a5e68535f.jpg"
2021-04-13 16:49:04.010078: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'cudart64_110.dll'; dlerror: cudart64_110.dll not found
2021-04-13 16:49:04.011074: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
2021-04-13 16:49:05.950489: I tensorflow/compiler/xrt/xrt_device.cc:41] Not creating XRT devices, tf_xrt_enable_xrt_devices not set
2021-04-13 16:49:05.951332: I tensorflow/stream_executor/platform/default/dso_loader.cc:60] Successfully opened dynamic library nvidia_cuda.dll
2021-04-13 16:49:05.988510: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1720] Found device 0 with properties:
pciBusID: 0000:01:00.0 name: GeForce RTX 2080 computeCapability: 7.5
coreClock: 1.35GHz coreCount: 30 deviceMemorySize: 6.00GiB deviceMemoryBandwidth: 241.85GiB/s
2021-04-13 16:49:05.985158: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'cudart64_110.dll'; dlerror: cudart64_110.dll not found
2021-04-13 16:49:05.985717: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'cublas64_11.dll'; dlerror: cublas64_11.dll not found
2021-04-13 16:49:05.986721: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'cublasLt64_11.dll'; dlerror: cublasLt64_11.dll not found
2021-04-13 16:49:05.986909: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'cuffrt64_10.dll'; dlerror: cuffrt64_10.dll not found
2021-04-13 16:49:05.987512: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'curand64_10.dll'; dlerror: curand64_10.dll not found
2021-04-13 16:49:05.988193: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'cusolver64_10.dll'; dlerror: cusolver64_10.dll not found
2021-04-13 16:49:05.989297: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'cusparse64_11.dll'; dlerror: cusparse64_11.dll not found
2021-04-13 16:49:05.990258: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1757] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would l
like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.
Skipping registering GPU devices...
2021-04-13 16:49:05.991099: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with Intel Deep Neural Network Library (oneDNN) to use the following CPU instructions in perf
ormance-critical operations: AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2021-04-13 16:49:05.992272: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1201] Device interconnect StreamExecutor with strength 1 edge matrix:
2021-04-13 16:49:05.992401: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1267]
2021-04-13 16:49:05.992788: I tensorflow/compiler/xrt/xrt_device.cc:99] Not creating XRT devices, tf_xrt_enable_xrt_devices not set
2021-04-13 16:49:07.300280: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR optimization passes are enabled (registered 2)

start motorcycle racer is riding on the track end
C:\Users\saisr\Downloads\python-project-image-caption-generator>
```

**“start motorcycle racer is riding on the track end”**

Above statement is generated by the model that we have generated and executed.

\*start indicates the beginning of the caption and end indicates the closing of caption



## Another Example: Example 2



The image we are giving to the caption generating model as input

```
Command Prompt
pciBusID: 0000:01:00.0 name: GeForce RTX 2060 computeCapability: 7.5
coreClock: 1.35GHz coreCount: 30 deviceMemorySize: 6.00GiB deviceMemoryBandwidth: 241.85GiB/s
2021-04-13 16:55:01.171579: M tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'cudart64_110.dll'; dlerror: cudart64_110.dll not found
2021-04-13 16:55:01.172540: M tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'cublas64_11.dll'; dlerror: cublas64_11.dll not found
2021-04-13 16:55:01.173186: M tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'cublaslt64_11.dll'; dlerror: cublaslt64_11.dll not found
2021-04-13 16:55:01.176382: M tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'cufft64_10.dll'; dlerror: cufft64_10.dll not found
2021-04-13 16:55:01.176984: M tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'curand64_10.dll'; dlerror: curand64_10.dll not found
2021-04-13 16:55:01.177493: M tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'cusolver64_10.dll'; dlerror: cusolver64_10.dll not found
2021-04-13 16:55:01.178157: M tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'cuspars64_11.dll'; dlerror: cuspars64_11.dll not found
2021-04-13 16:55:01.178629: M tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found
2021-04-13 16:55:01.179287: M tensorflow/core/common_runtime/gpu/gpu_device.cc:1757] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.
Skipping registering GPU devices...
2021-04-13 16:55:01.180620: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2021-04-13 16:55:01.181684: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1203] Device Interconnect StreamExecutor with strength 1 edge matrix:
2021-04-13 16:55:01.181739: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1267]
2021-04-13 16:55:01.182039: I tensorflow/compiler/xrt/xla_gpu_device.cc:99] Not creating XLA devices, tf_xla_enable_xla_devices not set
2021-04-13 16:55:02.493179: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR optimization passes are enabled (registered 2)

start black dog is running through the water end
C:\Users\salir\Downloads\python-project-image-caption-generator>
```

**“start black dog is running through the water end”**

-the image caption that our caption generator has generated

We have listed many such examples below:



A tennis player in a tennis court with a racket.



A man in a baseball shirt is playing a game of baseball.



A train is traveling down a train track.



A group of zebras standing next to each other.



A large kite flying over a cloudy day.



A street sign on a street next to a building.



A large elephant with a large baby elephant in a grassy field.



A red and green bus parked on a street.



A cat is sitting on a couch.



A living room with a large window.



A man riding a snowboard on a snow covered slope.



A man riding a surfboard in the ocean.





A man in a suit and tie is holding a tie.



A horse with a horse on top of a building.



A group of people standing in a room with a picture of people.



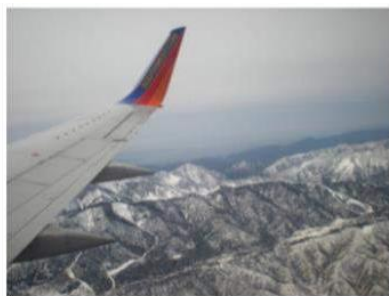
A man in a blue shirt is holding a frisbee.



A large bird flying over a large building.



A group of people standing next to a tree.



A person flying a kite in the air.



A black bear laying on top of a dirt field.



A man with a hat and a tie.



A black and white cat laying on a bed.



A toilet sitting on a desk next to a laptop.



A motorcycle with a motorcycle on the back of a motorcycle.

## Evaluation

- The project is successful. We have finished all the goals before the deadline. The system can generate sentences that are semantically correct according to the image. We also proposed a simplified version of GRU that has less parameters and achieves comparable result with the LSTM method.
- The strength of the method is on its end-to-end learning framework. The weakness is that it requires large number of human labeled data which is very expensive in practice. Also, the current method still has considerable errors in both object detection and sentence generation.

## Chapter 6

# CONCLUSION AND FUTURE WORK

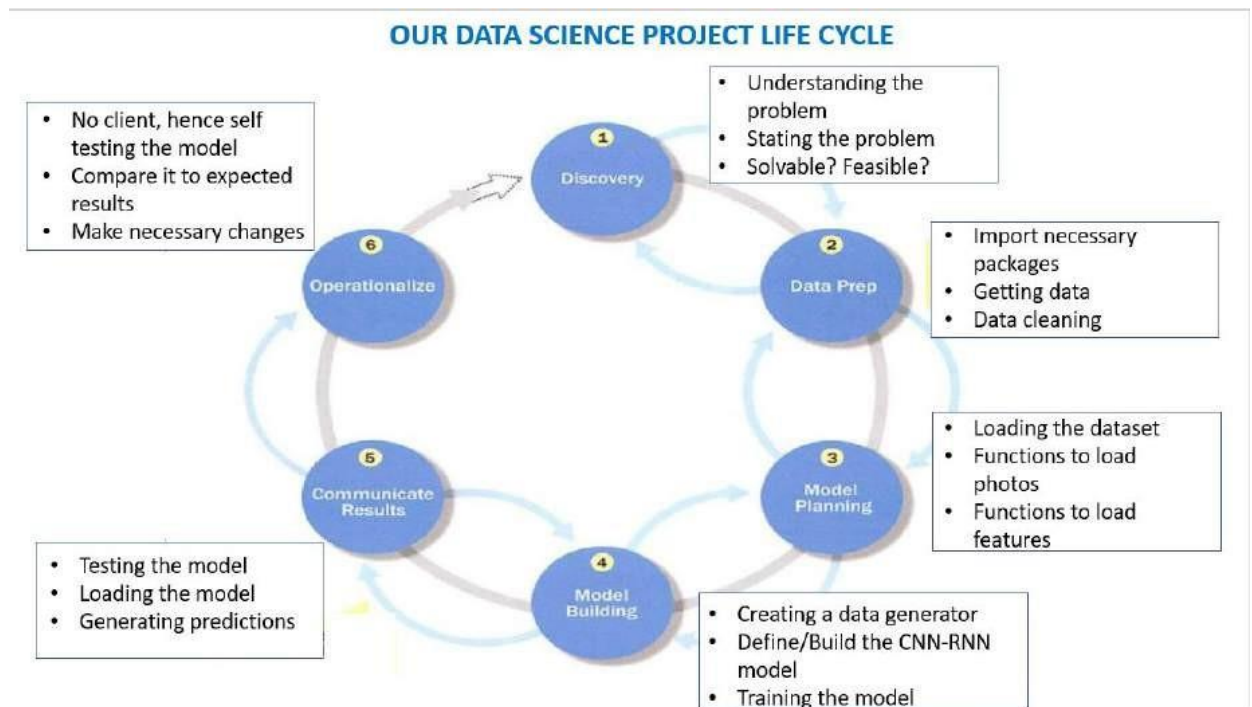


Figure 4 – My Data Science Project Life Cycle



## CONCLUSION

We analyzed and modified an image captioning method LRCN. To understand the method deeply, we decomposed the method to CNN, RNN, and sentence generation. For each part, we modified or replaced the component to see the influence on the final result. The modified method is evaluated on the COCO caption corpus. Experiment results show that: first the VGGNet outperforms the AlexNet and GoogLeNet in BLEU score measurement; second, the simplified GRU model achieves comparable results with more complicated LSTM model; third, increasing the beam size increase the BLEU score in general but does not necessarily increase the quality of the description which is judged by humans.

**Future work** In the future, we would like to explore methods to generate multiple sentences with different content. One possible way is to combine interesting region detection and image captioning. The VSA method (Karpathy and Fei-Fei, ) gives a direction of our future work. Taking Fig. 2 as an example, we hope the output will be a short paragraph: "Jack has a wonderful breakfast in a Sunday morning. He is sitting at a table with a bouquet of red flowers. With his new iPad air on the left, he enjoys a plate of fruits and a cup of coffee." The short paragraph naturally describes the image content in a story-telling fashion which is more attractive to the human beings.

## Chapter 6

### REFERENCES

- [Chen et al.2015] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollar, and C Lawrence Zitnick. 2015. Microsoft coco captions: Data collection and evaluation server. *arXiv preprint arXiv:1504.00325*.
- [Cho et al.2014] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.
- [Donahue et al.] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *IEEE CVPR*.
- [Fang et al.] Hao Fang, Saurabh Gupta, Forrest Iandola, Rupesh K Srivastava, Li Deng, Piotr Dollar, Jianfeng Gao, Xiaodong He, Margaret Mitchell, John C Platt, et al. From captions to visual concepts and back. In *IEEE CVPR*.
- [Gupta and Mannem] Ankush Gupta and Prashanth Mannem. From image annotation to image description. In *Neural information processing*.