IMAGE CAPTIONING USING DEEP LEARNING MODELS

A Project report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR'S OF TECHNOLOGY

in

COMPUTER SCIENCE ENGINEERING and

ELECTRONICS AND COMMUNICATIONS

ENGINEERING

by

K.Karthik Kumar(111915061)

Mohammed Thousif Ahmed(111916029)

K.Rohit Reddy(111915059)

B.Karthik Reddy(111916063)

Semester: IV

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING and ELECTRONICS AND COMMUNICATIONS ENGINEERING

Indian Institute of Information And Technology Pune

Near Bopdev Ghat, Kondhwa Annexe, Yewalewadi, Pune, Maharashtra 411048

FEBRUARY 2021

BONAFIDE CERTIFICATE

This is to certify that the project report entitled **" Image Captioning using Deep Learning models "** submitted by **K.Karthik kumar,Mohammed Thousif Ahmed, K.Rohit Reddy,B.Karthik Reddy** bearing the **MIS No: 111915061, 111916029, 111915059, 111916063** respectively, in completion of our project work under the guidance of **Dr.Sanjeev Sharma** is accepted for the project report submission in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science Engineering and Electronics and Communications Engineering in the Department of Computer Science and Engineering, Indian Institute of Information Technology,Pune, during the academic year 2020-21.

Dr.Sanjeev Sharma
Project Guide
Assistant Professor
Department of CSE
IIIT Pune

Dr.Tanmoy Hazra
Head of the Department
Assistant Professor
Department of CSE
IIIT Pune

Project Viva-voce held on 15-03-2021

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

This project would not have been possible without the help and cooperation of many. I would like to thank the people who helped me directly and indirectly in the completion of this project work.

First and foremost, I would like to express my gratitude to our beloved director, **Dr.Anupam Shukla**, for providing his kind support in various aspects.

I would like to express my gratitude to my project guide **Dr.Sanjeev Sharma**, Assistant Professor, Department of CSE, for providing excellent guidance, encouragement, inspiration, constant and timely support throughout this B.Tech project.

I would like to express my gratitude to the head of department **Dr.Tanmoy Hazra**, Assistant Professor, Department of CSE, for providing his kind support in various aspects.

I would also like to thank all the faculty members in the Dept. of CSE and my classmates for their steadfast and strong support and engagement with this project.

Abstract

In recent years, with the rapid development of artificial intelligence, image caption has gradually attracted the attention of many researchers in the field of artificial intelligence and has become an interesting and arduous task. Image caption, automatically generating natural language descriptions according to the content observed in an image, is an important part of scene understanding, which combines the knowledge of computer vision and natural language processing. The application of image caption is extensive and significant, for example, the realization of human-computer interaction. This paper summarizes the related methods and focuses on the attention mechanism, which plays an important role in computer vision and is recently widely used in image caption generation tasks. Furthermore, the advantages and the shortcomings of these methods are discussed, providing the commonly used datasets and evaluation criteria in this field.

Keywords : Image Captioning,Artificial Intelligence,Natural Language.

TABLE OF CONTENTS

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Chapter Introduction

1.1 Deep Learning

To know about Deep learning we need to first understand about artificial intelligence. Artificial intelligence (AI) is a very vast branch in Computer Science and it is used for constructing smart machines. The smart machines are capable of executing a task that requires human intelligence. Deep learning and machine learning are learning techniques in Artificial Intelligence. We have achieved many advancements in machine learning and deep learning. Now , as we are discussing Deep Learning. Deep learning is an Artificial Intelligence (AI) function that works almost similar to human brain in terms of performing tasks such as processing data and creating patterns in terms of decision making.Deep learning is a part of machine learning in artificial intelligence which consist networks such as deep neural networks which are capable of learning unsupervised data.Consider a human brain: it consists of millions of neurons. A deep neural network (DNN) has several layers in between input and output layers. Each layer consists of several nodes and their similar to neurones of the human brain. To solve a task the system has to process layers of data between input and output. Stimulus must be passed at the input layer in order to execute a task.

1.2 Thesis Outline

The thesis is organized as follows:

Chapter 1 provides a general introduction to the thesis.

Chapter 2 says the necessary of this type of projects in the present world.

Chapter 3 talks about various technologies used in this project.

Chapter 4 gives the design and analysis of the entire control system.

Chapter 5 explains our proposed scheme.

Chapter 6 provides the experimental results of the proposed scheme and it analyses the activity of this control system in different conditions.

Chapter 7 concludes our work and gives the future work which can be done to improve this scheme.

Chapter 2

Objectives

Introduction of chapter 2.

2.1 Objectives

The aim of image captioning is to automatically describe an image with one or more natural language sentences. its main challenges arise from the necessity of translating between two distinct, but usually paired, modalities.

To learn the applications of deep learning.

To create the application for helping the blind people.

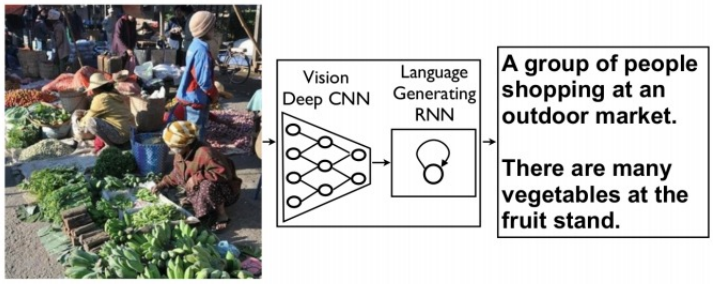
Chapter 3

Literature Review

3.1 Paper 1

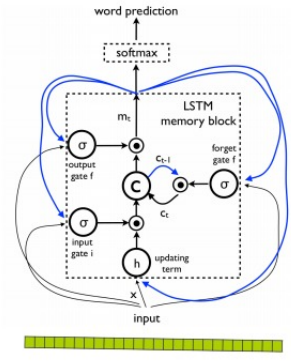
Previous Approaches - 1

This is a work by Vinyals et al(2015), Show and Tell: A Neural Image Caption Generator



Previous Approaches - 1

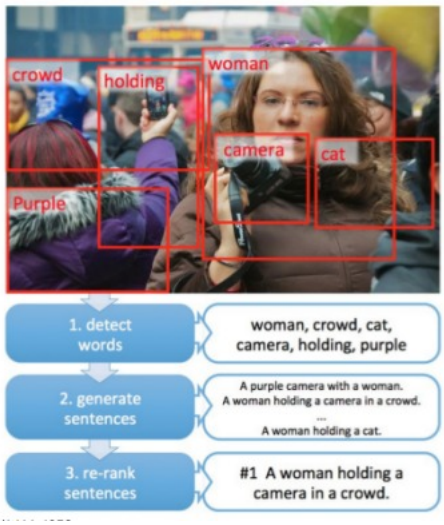
- Generate a fixed length vector representation of the image, say V_i. This representation is taken by extracting the features from a hidden layer of a pretrained CNN
- Convert the caption to a vector representation, say V_c
- V_i is fed to an Long Short Term Memory RNN(LSTM) as auxiliary input
- Now V_c is fed, element by element to the RNN to train it



3.2 Paper 2

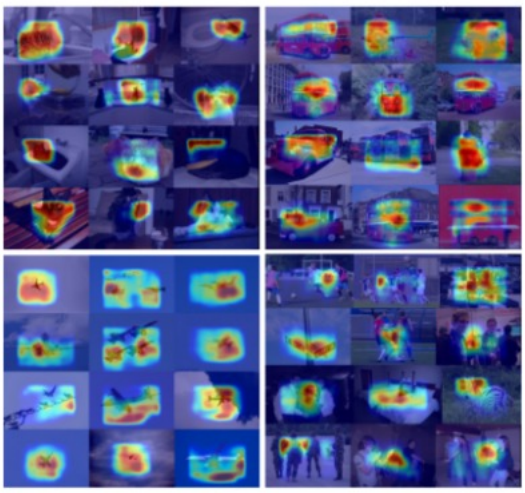
Previous Approaches - 2

This is a work by Fang et al(2015), From Captions to Visual Concepts and Back



Previous Approaches - 2

Word Detection Stage
They filter out the most common words in the training set and then use noisy-OR Multiple Instance Learning - taking a sliding window on the image as the bags - to determine the regia for each word for each image. The need for bounding boxes is eliminated by taking the representation generated by a CNN trained on ImageNet for the base of the MIL



3.3 Table

S.NO	NAME	WORK DESCRIPTION	PUBLICATION YEAR
1	VINYALS et al	A Neural image caption generator using vision deep CNN and language generating RNN.	2015
2	FANG et al	Analyzing visual concepts and generating captions, This process contains 3 stages, ● Word detection ● Sentences generation ● Ranking generated sentences	2015

Chapter 4

Methodology

The below figure shows the models used in our project and their evolution.

figure reference fig: 4.1 in the page : 7.

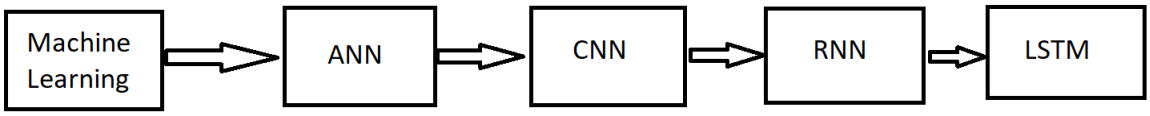


Figure 4.1: Flowchart

4.1 CNN(Convolution Nueral Networks)

As the name says this model uses Convolution in at least one of it's layers. Here in the first part, we use convolution operation for feature extraction. In this feature extraction we extract important points or features from the image known as Filters.

4.2 RNN(Recurrent Nuereal Networks)

The main purpose to use RNN is that it is helpful in modelling sequential data. In a RNN model there are multiple ANN's in which we not only pass the data from the hidden layer to it's output but also pass the data to the hidden layer of the next ANN. This gives a benefit that there can be a clear outlook of the whole output produced.

4.3 LSTM(Long Short Term Memory)

RNN's cannot understand the context of the the given input. They cannot recall the data which was said long before. RNN's remember the data for small duration of time but when lot of words are fed, the information gets lost. Here is where Long Short Term Memory is used.

4.4 Natural Language Processing

Research on Natural Language Processing has been started about 50 years ago and now came into work as a technological evolution in terms of computers is huge. Natural Language processing (NLP) is a branch of artificial intelligence. It deals with interaction between computers and human language. It uses computer as a medium to read and understand the language irrespective of spoken or written. This also includes translation of one language into another language. Natural Language Processing helps computer to communicate with humans in human Language. It is because of NLP the computers are able to read text, hear speech,interpret the language.

4.5 Libraries

4.5.1 Numpy

Numpy is also a Python programming language library . Numpy stands for numerical python. Numpy gives us support while accessing large dimensional array size and matrices. It also has a large collection of high level mathematical functions such as algebra, fourier transform ,trigonometry , statistics. The main difference between Pandas and numpy is : Pandas deals with tabular data whereas Numpy deals with numerical data.

4.5.2 Pandas

Python has many PIP libraries in which Pandas is also one of those libraries. It mainly deals with data structures and data operations for manipulating numerical tables and time series.

4.5.3 Keras

It provides us with a Python interface for artificial neural networks. Keras acts as a wrapper for tensorflow libraries . It also acts as a wrapper for Microsoft cognitive Toolkit , Theano and Plaid ML.

4.5.4 Tensor Flow

Research and engineers developer tensorflow working with Google's machine intelligence organisation. It is used in deep neural networks and machine learning research that can be used along with Python C++ and many other API. It mainly concentrates on data flow and differentiable programming

Chapter 5

Our Work

5.1 Libraries Imported

Here we imported these libraries in our program. They are: 1)Tensor Flow 2)keras

5.1.1 Tensor Flow

Research and engineers developer tensorflow working with Google's machine intelligence organisation.It is used in deep neural networks and machine learning research that can be used along with Python C++ and many other API.It mainly concentrates on data flow and differentiable programming

5.1.2 Keras

Keras is an open source neural-network library written in python for developing and evaluating deep learning models. It wraps the efficient numerical computation libraries Theano and TensorFlow and allows you to define and train neural network models in just a few lines of code

5.2 Algorithm used

CNN is an efficient recognition algorithm which is widely used in pattern recognition and image processing. ... Generally, the structure of CNN includes two layers one is feature extraction layer, the input of each neuron is connected to the local receptive fields of the previous layer, and extracts the local feature.

The main purpose to use RNN is that it is helpful in modelling sequential data. In

a RNN model there are multiple ANN's in which we not only pass the data from the hidden layer to it's output but also pass the data to the hidden layer of the next ANN..

5.3 PROGRAM

```
# -*- coding: utf-8 -*-
"""Untitled5.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1L77xfsro-YNlyEWbXiiSxNGX-McQ4hMr
"""

pip install q keras==1.2.2

pip install tensorflow==0.12.1

import keras
import os
keras.__version__

# Commented out IPython magic to ensure Python compatibility.
import glob
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
# %matplotlib inline
import pickle
from tqdm import tqdm
import pandas as pd
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import LSTM, Embedding, TimeDistributed, Dense,Merge,
    RepeatVector, Activation, Flatten
from keras.optimizers import Adam, RMSprop
from keras.layers.wrappers import Bidirectional
from keras.applications.inception_v3 import InceptionV3
from keras.preprocessing import image
import nltk

from google.colab import drive
drive.mount('/content/drive')

token = '/content/drive/My Drive/ML/Flickr8k.token.txt'

captions = open(token, 'r').read().strip().split('\n')

d = {}
for i, row in enumerate(captions):
    row = row.split('\t')
    row[0] = row[0][:len(row[0])-2]
    if row[0] in d:
        d[row[0]].append(row[1])
```

```
    else:
        d[row[0]] = [row[1]]

d['1000268201_693b08cb0e.jpg']

images = '/content/drive/My Drive/ML/Flicker8k_Dataset'

# Contains all the images
img = [fil for fil in os.listdir(images) if fil.endswith(".jpg")]
len(img)

train_images_file = '/content/drive/My Drive/ML/Flickr_8k.trainImages.
    txt'

train_images = set(open(train_images_file, 'r').read().strip().split('\
    n'))
len(train_images)

def split_data(l):
    temp = []
    for i in img:
        if i in l:
            temp.append(i)
    return temp

# Getting the training images from all the images
train_img = split_data(train_images)
len(train_img)

val_images_file = '/content/drive/My Drive/ML/Flickr_8k.devImages.txt'
val_images = set(open(val_images_file, 'r').read().strip().split('\n'))

# Getting the validation images from all the images
val_img = split_data(val_images)
len(val_img)

test_images_file = '/content/drive/My Drive/ML/Flickr_8k.testImages.txt
    '
test_images = set(open(test_images_file, 'r').read().strip().split('\n'
    ))

# Getting the testing images from all the images
test_img = split_data(test_images)
len(test_img)

def preprocess_input(x):
    x /= 255.
    x -= 0.5
    x *= 2.
    return x

def preprocess(image_path):
    img = image.load_img(image_path, target_size=(299, 299))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)

    x = preprocess_input(x)
    return x
```

```
103
104 plt.imshow(np.squeeze(preprocess(images + '/'+ train_img[0])))
105
106 model = InceptionV3(weights='imagenet')
107
108 from keras.models import Model
109
110 new_input = model.input
111 hidden_layer = model.layers[-2].output
112
113 model_new = Model(new_input, hidden_layer)
114
115 tryi = model_new.predict(preprocess(images+'/'+train_img[0]))
116
117 tryi.shape
118
119 def encode(image):
120     image = preprocess(image)
121     temp_enc = model_new.predict(image)
122     temp_enc = np.reshape(temp_enc, temp_enc.shape[1])
123     return temp_enc
124
125 encoding_train = {}
126  for img in tqdm(train_img):
127      encoding_train[img] = encode(images+'/'+img)
128
129 with open("/content/drive/My Drive/ML/encoded_images_inceptionV3.p", "
    wb") as encoded_pickle:
130       pickle.dump(encoding_train, encoded_pickle)
131
132 encoding_train = pickle.load(open('/content/drive/My Drive/ML/
    encoded_images_inceptionV3.p', 'rb'))
133
134 encoding_train['3556792157_d09d42bef7.jpg'].shape
135
136 encoding_test = {}
137  for img in tqdm(test_img):
138      encoding_test[img] = encode(images+'/'+img)
139
140 with open("/content/drive/My Drive/ML/encoded_images_test_inceptionV3.p
    ", "wb") as encoded_pickle:
141       pickle.dump(encoding_test, encoded_pickle)
142
143 encoding_test = pickle.load(open('/content/drive/My Drive/ML/
    encoded_images_test_inceptionV3.p', 'rb'))
144
145 print(test_img[0])
146 encoding_test[test_img[0]].shape
147
148 train_d = {}
149 for i in train_img:
150     if i in d:
151         train_d[i] = d[i]
152
153 train_d
154
155 train_d['3556792157_d09d42bef7.jpg']
156
```

```
val_d = {}
for i in val_img:
    if i in d:
        val_d[i] = d[i]

len(val_d)

test_d = {}
for i in test_img:
    if i in d:
        test_d[i] = d[i]

len(test_d)

caps = []
for key, val in train_d.items():
    for i in val:
        caps.append('<start> ' + i + ' <end>')

words = [i.split() for i in caps]

unique = []
for i in words:
    unique.extend(i)

unique = list(set(unique))

with open("/content/drive/My Drive/ML/unique.p", "wb") as pickle_d:
    pickle.dump(unique, pickle_d)

unique = pickle.load(open('/content/drive/My Drive/ML/unique.p', 'rb'))

len(unique)

word2idx = {val:index for index, val in enumerate(unique)}

word2idx['<start>']

idx2word = {index:val for index, val in enumerate(unique)}

idx2word[3102]

max_len = 0
for c in caps:
    c = c.split()
    if len(c) > max_len:
        max_len = len(c)
max_len

len(unique), max_len

vocab_size = len(unique)

vocab_size

train_d.items()

f = open('/content/drive/My Drive/ML/flickr8k_training_dataset.txt', 'w
```

```
    ')
f.write("image_id\tcaptions\n")

for key, val in train_d.items():
    for i in val:
        f.write(key + "\t" + "<start> " + i +" <end>" + "\n")

f.close()

df = pd.read_csv('/content/drive/My Drive/ML/flickr8k_training_dataset.txt', delimiter='\t')

df

c = [i for i in df['captions']]
len(c)

imgs = [i for i in df['image_id']]

a = c[-1]
a, imgs[-1]

for i in a.split():
    print (i, "=>", word2idx[i])

samples_per_epoch = 0
for ca in caps:
    samples_per_epoch += len(ca.split())-1

samples_per_epoch

def data_generator(batch_size = 32):
        partial_caps = []
        next_words = []
        images = []

        df = pd.read_csv('/content/drive/My Drive/ML/flickr8k_training_dataset.txt', delimiter='\t')
        df = df.sample(frac=1)
        iter = df.iterrows()
        c = []
        imgs = []
        for i in range(df.shape[0]):
            x = next(iter)
            c.append(x[1][1])
            imgs.append(x[1][0])



        count = 0
        while True:
            for j, text in enumerate(c):
                current_image = encoding_train[imgs[j]]
                for i in range(len(text.split())-1):
                    count+=1

                    partial = [word2idx[txt] for txt in text.split()[:i+1]]
```

```
                    partial_caps.append(partial)

                    # Initializing with zeros to create a one-hot
    encoding matrix
                    # This is what we have to predict
                    # Hence initializing it with vocab_size length
                    n = np.zeros(vocab_size)
                    # Setting the next word to 1 in the one-hot encoded
     matrix
                    n[word2idx[text.split()[i+1]]] = 1
                    next_words.append(n)

                    images.append(current_image)

                    if count>=batch_size:
                        next_words = np.asarray(next_words)
                        images = np.asarray(images)
                        partial_caps = sequence.pad_sequences(
    partial_caps, maxlen=max_len, padding='post')
                        yield [[images, partial_caps], next_words]
                        partial_caps = []
                        next_words = []
                        images = []
                        count = 0

data_generator(batch_size=128)

embedding_size = 300

image_model = Sequential([
        Dense(embedding_size, input_shape=(2048,), activation='relu'),
        RepeatVector(max_len)
    ])
image_model.summary()

caption_model = Sequential([
        Embedding(vocab_size, embedding_size, input_length=max_len),
        LSTM(256, return_sequences=True),
        TimeDistributed(Dense(300))
    ])
caption_model.summary()

final_model = Sequential([
        Merge([image_model, caption_model], mode='concat', concat_axis
    =1),
        Bidirectional(LSTM(256, return_sequences=False)),
        Dense(vocab_size),
        Activation('softmax')
    ])

final_model.compile(loss='categorical_crossentropy', optimizer=RMSprop
    (), metrics=['accuracy'])

final_model.summary()

final_model.fit_generator(data_generator(batch_size=128),
    samples_per_epoch=samples_per_epoch//128, nb_epoch=1,
                              verbose=2)
```

```
321
322 final_model.fit_generator(data_generator(batch_size=128),
       samples_per_epoch=samples_per_epoch//128, nb_epoch=1,
323                            verbose=2)
324
325 final_model.fit_generator(data_generator(batch_size=128),
       samples_per_epoch=samples_per_epoch//128, nb_epoch=1,
326                            verbose=2)
327
328 final_model.fit_generator(data_generator(batch_size=128),
       samples_per_epoch=samples_per_epoch//128, nb_epoch=1,
329                            verbose=2)
330
331 final_model.fit_generator(data_generator(batch_size=128),
       samples_per_epoch=samples_per_epoch//128, nb_epoch=1,
332                            verbose=2)
333
334 final_model.fit_generator(data_generator(batch_size=128),
       samples_per_epoch=samples_per_epoch//128, nb_epoch=1,
335                            verbose=2)
336
337 final_model.fit_generator(data_generator(batch_size=128),
       samples_per_epoch=samples_per_epoch//128, nb_epoch=1,
338                            verbose=2)
339
340 final_model.optimizer.lr = 1e-4
341 final_model.fit_generator(data_generator(batch_size=128),
       samples_per_epoch=samples_per_epoch//128, nb_epoch=1,
342                            verbose=2)
343
344 final_model.fit_generator(data_generator(batch_size=128),
       samples_per_epoch=samples_per_epoch//128, nb_epoch=1,
345                            verbose=2)
346
347 final_model.save_weights('/content/drive/My Drive/ML/
       time_inceptionV3_7_loss_3.2604.h5')
348
349 final_model.load_weights('/content/drive/My Drive/ML/
       time_inceptionV3_7_loss_3.2604.h5')
350
351 final_model.fit_generator(data_generator(batch_size=128),
       samples_per_epoch=samples_per_epoch//128, nb_epoch=1,
352                            verbose=2)
353
354 final_model.fit_generator(data_generator(batch_size=128),
       samples_per_epoch=samples_per_epoch//128, nb_epoch=1,
355                            verbose=2)
356
357 final_model.save_weights('/content/drive/My Drive/ML/time_inceptionV3_3
       .21_loss.h5')
358
359 final_model.fit_generator(data_generator(batch_size=128),
       samples_per_epoch=samples_per_epoch//128, nb_epoch=1,
360                            verbose=2)
361
362 final_model.fit_generator(data_generator(batch_size=128),
       samples_per_epoch=samples_per_epoch//128, nb_epoch=1,
363                            verbose=2)
```

```
364
365 final_model.save_weights('/content/drive/My Drive/ML/time_inceptionV3.
       h5')
366
367 final_model.fit_generator(data_generator(batch_size=128),
       samples_per_epoch=samples_per_epoch//128, nb_epoch=1,
368                           verbose=2)
```

Listing 5.1: Program

```
1  # -*- coding: utf-8 -*-
2  """new_Predict_Caption.ipynb
3
4  Automatically generated by Colaboratory.
5
6  Original file is located at
7      https://colab.research.google.com/drive/1LGfLa9V36ssn06E-
       Qe87dZBJUcfkq8Kw
8  """
9
10
11
12 # Commented out IPython magic to ensure Python compatibility.
13 import glob
14 from PIL import Image
15 import numpy as np
16 import matplotlib.pyplot as plt
17 # %matplotlib inline
18 import pickle
19 from tqdm import tqdm
20 import pandas as pd
21 from keras.preprocessing import sequence
22 from keras.models import Sequential
23 from keras.layers import LSTM, Embedding, TimeDistributed, Dense,Merge,
       RepeatVector, Activation, Flatten
24 from keras.optimizers import Adam, RMSprop
25 from keras.layers.wrappers import Bidirectional
26 from keras.applications.inception_v3 import InceptionV3
27 from keras.preprocessing import image
28 import nltk
29
30 import keras
31 import tensorflow as tf
32
33
34
35
36
37 from google.colab import drive
38 drive.mount('/content/drive')
39 path = '/content/drive/My Drive/ML/'
40
41 model = InceptionV3(weights='imagenet')
42
43 from keras.models import Model
44
45
46 new_input = model.input
47 hidden_layer = model.layers[-2].output
```

```
48
49 model_new = Model(new_input, hidden_layer)
50 model_new._make_predict_function()
51 graph = tf.get_default_graph()
52
53 def preprocess_input(x):
54     x /= 255.
55     x -= 0.5
56     x *= 2.
57     return x
58
59 def preprocess(image_path):
60     img = image.load_img(image_path, target_size=(299, 299))
61     x = image.img_to_array(img)
62     x = np.expand_dims(x, axis=0)
63
64     x = preprocess_input(x)
65     return x
66
67 def encode(image):
68     image = preprocess(image)
69     global graph
70     with graph.as_default():
71         temp_enc = model_new.predict(image)
72     temp_enc = np.reshape(temp_enc, temp_enc.shape[1])
73     return temp_enc
74
75 unique = pickle.load(open('/content/drive/My Drive/ML/unique.p', 'rb'))
76
77 word2idx = {val:index for index, val in enumerate(unique)}
78
79 idx2word = {index:val for index, val in enumerate(unique)}
80
81 def data_generator(batch_size = 32):
82         partial_caps = []
83         next_words = []
84         images = []
85
86         df = pd.read_csv('/content/drive/My Drive/ML/
    flickr8k_training_dataset.txt', delimiter='\t')
87         df = df.sample(frac=1)
88         iter = df.iterrows()
89         c = []
90         imgs = []
91         for i in range(df.shape[0]):
92             x = next(iter)
93             c.append(x[1][1])
94             imgs.append(x[1][0])
95
96
97
98         count = 0
99         while True:
100            for j, text in enumerate(c):
101                current_image = encoding_train[imgs[j]]
102                for i in range(len(text.split())-1):
103                    count+=1
104
```

```
                    partial = [word2idx[txt] for txt in text.split()[:i+1]]
                    partial_caps.append(partial)

                    # Initializing with zeros to create a one-hot encoding matrix
                    # This is what we have to predict
                    # Hence initializing it with vocab_size length
                    n = np.zeros(vocab_size)
                    # Setting the next word to 1 in the one-hot encoded matrix
                    n[word2idx[text.split()[i+1]]] = 1
                    next_words.append(n)

                    images.append(current_image)

                    if count>=batch_size:
                        next_words = np.asarray(next_words)
                        images = np.asarray(images)
                        partial_caps = sequence.pad_sequences(partial_caps, maxlen=max_len, padding='post')
                        yield [[images, partial_caps], next_words]
                        partial_caps = []
                        next_words = []
                        images = []
                        count = 0

embedding_size = 300
max_len = 40
vocab_size = 8256

image_model = Sequential([
        Dense(embedding_size, input_shape=(2048,), activation='relu'),
        RepeatVector(max_len)
    ])

caption_model = Sequential([
        Embedding(vocab_size, embedding_size, input_length=max_len),
        LSTM(256, return_sequences=True),
        TimeDistributed(Dense(300))
    ])

final_model = Sequential([
        Merge([image_model, caption_model], mode='concat', concat_axis=1),
        Bidirectional(LSTM(256, return_sequences=False)),
        Dense(vocab_size),
        Activation('softmax')
    ])

# final_model.compile(loss='categorical_crossentropy', optimizer=RMSprop(), metrics=['accuracy'])

final_model.load_weights(path + 'tiime_inceptionV3_1.5987_loss.h5')
final_model._make_predict_function()
graph1 = tf.get_default_graph()

def predict_caption(image):
```

```
    start_word = ["<start>"]
    while True:
        par_caps = [word2idx[i] for i in start_word]
        par_caps = sequence.pad_sequences([par_caps], maxlen=max_len,
    padding='post')
        e = encode(image)
        global graph1
        with graph1.as_default():
            preds = final_model.predict([np.array([e]), np.array(
    par_caps)])
        word_pred = idx2word[np.argmax(preds[0])]
        start_word.append(word_pred)

        if word_pred == "<end>" or len(start_word) > max_len:
            break

    return ' '.join(start_word[1:-1])

def beam_search_prediction(image, beam_index = 3):
    start = [word2idx["<start>"]]

    start_word = [[start, 0.0]]

    while len(start_word[0][0]) < max_len:
        temp = []
        for s in start_word:
            par_caps = sequence.pad_sequences([s[0]], maxlen=max_len,
    padding='post')
            e = encode(image)
            preds = final_model.predict([np.array([e]), np.array(
    par_caps)])

            word_preds = np.argsort(preds[0])[-beam_index:]

            # Getting the top <beam_index>(n) predictions and creating
    a
            # new list so as to put them via the model again
            for w in word_preds:
                next_cap, prob = s[0][:], s[1]
                next_cap.append(w)
                prob += preds[0][w]
                temp.append([next_cap, prob])

        start_word = temp
        # Sorting according to the probabilities
        start_word = sorted(start_word, reverse=False, key=lambda l: l
    [1])
        # Getting the top words
        start_word = start_word[-beam_index:]

    start_word = start_word[-1][0]
    intermediate_caption = [idx2word[i] for i in start_word]

    final_caption = []

    for i in intermediate_caption:
        if i != '<end>':
            final_caption.append(i)
```

```
        else:
            break

    final_caption = ' '.join(final_caption[1:])
    return final_caption
```

Listing 5.2: Program

```
# -*- coding: utf-8 -*-
"""app1.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1YymXhfOdlSIE9L711xrWtOVjCmcQDBs8
"""

!pip install q keras==1.2.2

pip install tensorflow==1.13.1

!pip install flask-ngrok

from flask_ngrok import run_with_ngrok

import flask
from flask import Flask,render_template,redirect,request

# Commented out IPython magic to ensure Python compatibility.
# %mkdir static -p

# Commented out IPython magic to ensure Python compatibility.
# %mkdir templates -p

# Commented out IPython magic to ensure Python compatibility.
# %%writefile templates/index.html
# <!DOCTYPE html>
# <html>
# <head>
#   <title>Image Captioning</title>
# </head>
# <body>
#
#   <h1> Image Captioning</h1>
#
#   <form method='POST' action='/' enctype='multipart/form-data'>
#     <input type='file' name='userfile' placeholder='Your Image'>
#     <input type='submit'>
#   </form>
#
#
#   {% if your_result %}
#
#
#     <h1><i> {{ your_result['caption']}} </i> </h1>
#     <img src = "{{url_for('static', filename=your_result['image'])}}">
#     <br>
```

```
#
#
#    {% endif %}
#
#
# </body>
# </html>
#

from google.colab import drive
drive.mount('/content/drive')

import sys
sys.path.insert(0,'/content/drive/My Drive/ML')

import new_predict_caption

app = flask.Flask(__name__,static_folder='/content/static')
run_with_ngrok(app)

@app.route('/')
def hoe():
    return render_template('index.html')

@app.route('/',methods=['GET','POST'])
def pred():
    if request.method == 'POST':
        f = request.files['userfile']
        name = f.filename
        path='/content/static/{}'.format(name)
        f.save(path)


        caption = new_predict_caption.predict_caption(path)
        result_dic = {
            'image' : name,
            'caption' : caption
        }
        print(caption)

    return render_template('index.html', your_result=result_dic)

app.run()
```

Listing 5.3: Program

5.4 Program Explanation

5.4.1 .

First, we install all the necessary packages such as keras, tensorflow and then we import them.

5.4.2 Understanding the Data

Along with the images we also get some text files related to the images. One of the files is "Flickr 8k.token.text" which contains the name of each image along with its 5 captions. Now, we create a dictionary which contains the name of the image as keys and a list of the 5 captions for the corresponding image as values.

5.4.3 Data Preprocessing — Images

Images are nothing but input to our model. Any input to a model must be given in the form of a vector.We need to convert every image into a fixed sized vector which can then be fed as input to the neural network. For this purpose, we opt for transfer learning by using the InceptionV3 model (Convolutional Neural Network) created by Google Research. This model was trained on Imagenet dataset to perform image classification on 1000 different classes of images. However, our purpose here is not to classify the image but just get a fixed-length informative vector for each image. This process is called automatic feature engineering . Hence, we just remove the last softmax layer from the model and extract a 2048 length vector. Now, we pass every image to this model to get the corresponding 2048 length feature vector. We save all the features in a Python dictionary and save it on the disk using Pickle file.

5.4.4 Data Preprocessing — Captions

Captions are something that we want to predict. So during the training period, captions will be the target variables that the model is learning to predict. But the prediction of the entire caption, given the image does not happen at once. We will predict the caption word by word . Thus, we need to encode each word into a fixed size vector. We will create two Python Dictionaries namely "wordtoix" (pronounced — word to index) and "ixtoword" (pronounced — index to word). We will represent every unique word in the vocabulary by an integer (index). We have 8763 unique words across all the 40000 image captions. However, if we think about it, many of these words will occur very few times, say 1, 2 or 3 times. Since we are creating a predictive model, we would not like to have

all the words present in our vocabulary but the words which are more likely to occur or which are common. Hence we consider only those words which occur at least 10 times in the entire corpus. We have 1652 unique words in the corpus and thus each word will be represented by an integer index between 1 to 1652.

We will add two tokens in every caption as follows

'startseq' -¿ This is a start sequence token which will be added at the start of every caption.

'endseq' -¿ This is an end sequence token which will be added at the end of every caption.

5.5 Data Preparation using Generator Function

We have to train our model on 6000 images and each image will contain a 2048 length feature vector and the caption is also represented as numbers. This amount of data for 6000 images is not possible to hold into memory so we will be using a generator method that will yield batches. To train a model on a particular dataset, we use some version of Stochastic Gradient Descent (SGD). With SGD, we do not calculate the loss on the entire data set to update the gradients. Rather in every iteration, we calculate the loss on a batch of data points (typically 64, 128, 256, etc.) to update the gradients. This means that we do not require to store the entire dataset in the memory at once. Even if we have the current batch of points in the memory, it is sufficient for our purpose. A generator function in Python is used exactly for this purpose. It's like an iterator which resumes the functionality from the point it left the last time it was called.

5.6 Training the model

To train the model, we will be using the 6000 training images by generating the input and output sequences in batches and fitting them to the model using model.fit generator() method. We also save the model to our models folder. This will take some time depending on your system capability.

Chapter 6

Results and Discussion

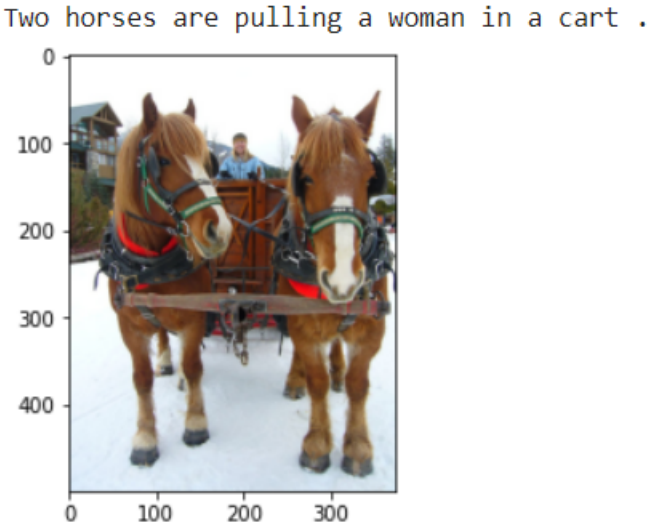


Figure 6.1: Result1

Figure 6.2: Result2

Figure 6.3: Result3

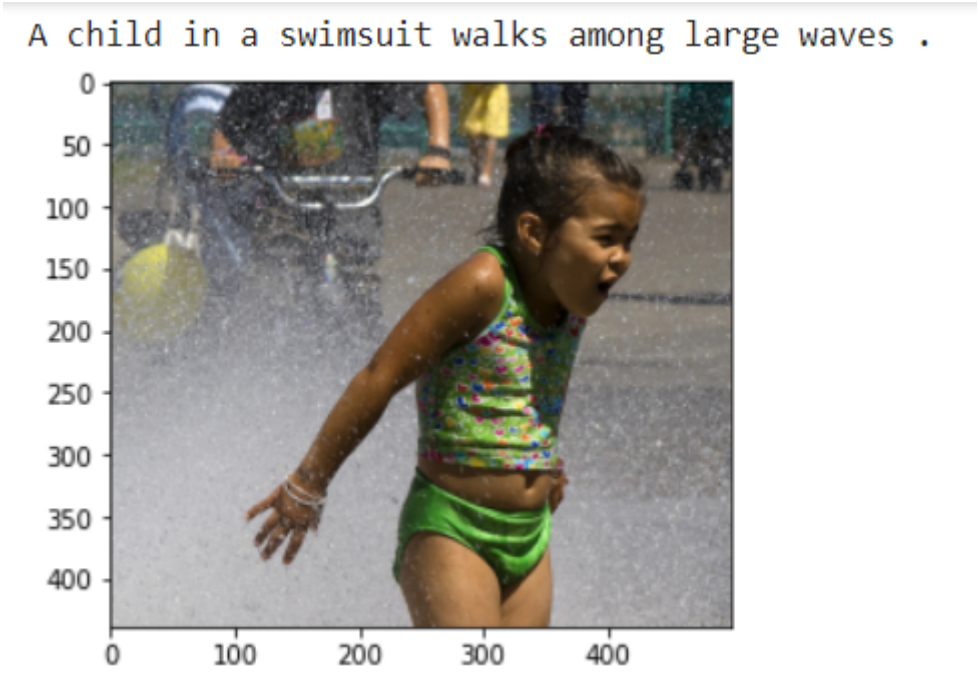


Figure 6.4: Result4

Chapter 7

Conclusion and Future Work

We have introduced a model that generates natural language descriptions of image regions based on obtained labels in form of a dataset of images and their respective sentence description, and with few assumptions. experimental results show that this task still has better performance systems and improvement. It mainly faces the following three challenges: first, how to generate complete natural language sentences like a human being; second, how to make the generated sentence grammatically correct; and third, how to make the caption semantics as clear as possible and consistent with the given image content

Some key points to note are that our model depends on the data, so it cannot predict the words that are out of its vocabulary. We used a small dataset consisting of 8000 images. For production-level models, we need to train on datasets larger than 100,000 images which can produce better accuracy models. A lot of modifications can be made to improve this solution like using a larger dataset, changing the model architecture, e.g. include an attention module, doing more hyper parameter tuning (learning rate, batch size, number of layers, number of units, dropout rate, batch

Chapter 8

References

8.1 The research by Lowe, D. G. on Distinctive Image Features from ScaleInvariant Key points.

8.2 Vikram Mullachery, Vishal Motwani(Image Captioning) (https://paperswithcode.com/paper/image-captioning)

8.3 Hashall Lamba (https://towardsdatascience.com/image-captioning-with-keras-teaching-computers-to-describe-pictures-c88a46a311b8)