

QUOTE IT

IMAGE CAPTION GENERATOR

A Minor Project Report

in partial fulfilment for the award of the degree

of

BACHELOR's OF TECHNOLOGY

In

COMPUTER SCIENCE ENGINEERING

by

MAYANK SINGH, PRITIKA RANA and PRIYA CHAWLA

Guided by

Dr SUNITA TIWARI

Assistant. Professor



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

G.B. PANT GOVT. ENGINEERING COLLEGE, NEW DELHI.

(AFFILIATED TO GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY, DELHI)

DECEMBER, 2020

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the Minor Project entitled “**Image Caption Generator**” in partial fulfilment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering / Information Technology affiliated to **Guru Gobind Singh Indraprastha University, New Delhi** and submitted to the Department of Computer Science and Engineering G.B.Pant Govt. Engineering College is an authentic record of my work carried out during a period from August 2020 to Dec 2020. The matter represented in this report has not been submitted by me for the award of any other degree of this or any other institute/university.

Date: - 19-12-2020

Name and Roll No:

Mayank Singh(00420907218)

Pritika Rana(02420902717)

Priya Chawla(41420902717)

This is to certify that the above statement made by the candidate is correct to the best of our knowledge.

Signature of Supervisor

Date:-

Name & Designation

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	LIST OF FIGURES	i
	ACKNOWLEDGMENT	iii
	ABSTRACT	iv
CHAPTER 1.	INTRODUCTION	1
1.1	OVERVIEW	1
1.2	MOTIVATION	1
1.3	PROBLEM STATEMENT	1
1.4	SCOPE OF PROJECT	1
1.5	ACCESS TO USERS	2
CHAPTER 2.	RELATED WORKS	3
2.1	INTRODUCTION	3
2.2	EXISTING SYSTEM	3
2.3	PROPOSED SYSTEM	4
2.3.1	IMAGE FEATURE EXTRACTION	4
2.3.2	THE LANGUAGE MODEL	5
2.3.3	FITTING THE MODEL	6
2.3.4	CAPTION GENERATION	6
2.3.5	SUGGESTION API	6
CHAPTER 3.	PROBLEM DESCRIPTION & SPECIFICATION	7
3.1	PROBLEM OVERVIEW	7

3.2	SPECIFICATION	7
3.3	REQUIREMENTS	8
3.3.1	SOFTWARE REQUIREMENTS	8
3.3.2	HARDWARE REQUIREMENTS	8
3.3.3	OTHER REQUIREMENTS	9
CHAPTER 4.	SYSTEM DESIGN	10
4.1	INTRODUCTION	10
4.1.1	DATA SOURCES	10
4.1.2	CONVOLUTION NEURAL NETWORK	10
4.1.3	RECURRENT NEURAL NETWORK	11
4.2	ARCHITECTURE DIAGRAM	11
4.3	UML DIAGRAM	12
4.4	DATA FLOW DIAGRAM	14
4.5	FLOW CHART DIAGRAM	14
CHAPTER 5.	IMPLEMENTATION AND RESULT	16
5.1	INTRODUCTION	16
5.2	MODEL SELECTION	16
5.3	IMAGE PREPROCESSING	17
5.4	TEXT PREPROCESSING	19
5.5	CREATING VOCABULARY	21
5.6	PREPARING THE MODEL	23
5.7	CONFIGURING THE MODEL	25

5.8	TRAINING AND SAVING THE MODEL	28
5.9	PREDICTING THE CAPTIONS	29
5.10	WEB PAGE DEVELOPMENT	31
5.11	API INTEGRATION	32
5.12	PROVIDING USER WITH CAPTIONS	32
CHAPTER 6.	CONCLUSION AND FUTURE SCOPE	34
6.1	CONCLUSION	34
6.2	FUTURE SCOPE	34
CHAPTER 7.	REFERENCES	36
APPENDICES		39
APPENDIX 1:	DETAILED TEST STRATEGY AND TEST CASES	39
APPENDIX 2:	USER GUIDE	40

List of Figures

Figure No.	Figure Title	Page No.
1	Flow Diagram of the project	8
2	Architecture of the image captioning model	11
3	Illustration image of functioning of the captioning model	12
4	Use Case Diagram for the captioning system	12
5	DFD Level 0	13
6	DFD Level 1	13
7	DFD Level 2	14
8	State Chart Diagram of the system	14
9	Flow chart for the caption generation process	15
10	Code Snippet showing the selection and adjustment of the model	16
11	Output representing different layers of the selected model	17
12	Code snippet showing preprocessing of images	17
13	A sample picture fetched from the dataset	18
14	Code snippet - Extracting feature map from passed images	18
15	Output representing the process of feature map creations	19
16	Code snippet showing text preprocessing	20
17	Sample captions fetched from Flickr8k.token.txt	20
18	Code snippet showing the process of caption dictionary creation	21
19	Sample view of caption dictionary	21
20	Code snippet showing Vocabulary Creation Process	22

21	Sample output showing the count of different words	22
22	Sample output showing the word dictionary	23
23	Sample output showing the integer form captions	23
24	Code snippet showing the max length of the caption present	24
25	Output showing the length of the largest caption in our collection	24
26	Code snippet showing the input preparation for the model	24
27	Code snippet showing the conversion to NumPy array form	25
28	Sample output showing the shapes of input	25
29	Configuring process of the image and language model	26
30	Output showing the image sequential model	26
31	Output showing the language sequential model	27
32	Output showing the combined mode	27
33	Detailed layout of the model along with shapes of inputs	28
34	Code snippet showing the training process	29
35	Sample output of the training process	29
36	Sample output of the inverse dictionary	29
37	Code snippet showing the image preprocessing for testing	30
38	Code snippet showing the prediction process and results	30
39	Some sample results generated by the model	31
40	Layout of the Home page of the Web App	33
41	Sample final results on the Web Platform	33
A1	Sample image showing loss and accuracy obtained by the model	38

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my supervisors **Dr Sunita Tiwari & Dr Anu Saini** for providing their invaluable guidance, comments and suggestions throughout the project. I would specially thank Dr Sunita Tiwari for constantly motivating me to work harder and suggesting me different ways to tackle the problems encountered.

Also, I would like to thank my team members for their assistance in the project. I would like to express my gratitude for the Department of Computer Science Engineering, Gobind Ballabh Pant Government Engineering college to provide me with the opportunity to work on this project and to all the teachers and mentors guiding along the way.

- Mayank Singh (00420907218)
- Pritika Rana (02420902717)
- Priya Chawla (41420902717)

ABSTRACT

The project image caption generator provides a user option to obtain captions of different categories of their choice. It also provides users with the option of uploading an image from their device and obtaining the caption predicted by our image captioning model. The user interacts with the project with the help of a Web Platform which provides the user with these options.

The image caption generation process uses Convolution Neural Network to extract the image features i.e, basically recognize the objects in the images and then convert it into a Numpy array form what we call a feature map. This feature map is saved and can be used for offline use. We use the Long Short-Term Memory technique of Recurrent Neural Network for processing the language part of the captioning model. We are using the ResNet50 model provided by the Tensorflow library for the purpose of image feature generation. In this project, we are providing local training to the pre-trained ResNet50 model(on ImageNet Dataset) on Flickr8k dataset.

This auto caption generation process enables the user to automate the task of captioning or categorizing the model by using the captions predicted by the model. Whenever the user provides the image to the Web App it gives the model to the backend containing the model, which uses the ResNet50 model to generate the image features and then our trained and saved model is used for predicting the next word as it moves across the feature map, connecting these words finally provides us with the final caption that is been displayed to the user on the Web Platform. Now the user can copy the caption provided and use it at the place of his/her choice.

Currently, the project emphasises on providing caption to the single images provided by the user. But this model has immense future scope like text-to-speech conversion which can benefit visually impaired persons, video frame summary generation which can also be used for surveillance purposes. By the help of this project, we put forward the idea of automating the task of caption generation but, this project carries much more opportunities which also we would like to pursue in future.

CHAPTER-1

INTRODUCTION

1.1 OVERVIEW

The Image Caption Generator provides captions for the images provided by the user. The images provided by the user through the Web App are being supplied to the backend deep learning model. This model uses ResNet50 model provided by the Tensorflow library to extract the features of the images. We use Long Short-Term Memory technique of Recurrent neural network to predict the next word in the caption sequence by feeding the image features and the dictionary of words. The model predicts the word having the highest probability, to follow in the caption chain and finally provides the user with the caption result. The flask library is used for the creation of the Web App. In the later chapter, we will walk through the journey of the construction of the project.

1.2 MOTIVATION

The Image Caption Generator provides captions for the images provided by the user. These captions can be used for automating the task of image captions and classifications based on different categories, which was earlier done by humans, by specifically visiting through all the images, captioning and then performing any processing on it. It can also be used to generate voice signals for assisting visually impaired persons. The captions can also contain information about the location, conditions and mood of the person, which again can be used for social media tagging, for location-related features and for creating suggestions for shopping, detecting and suggesting items or activities based on mood. The image caption generator can be used with video frames to provide help to security agencies about and mischievous activities with their parameter of inspection. The captions generated could also be used for improving search engines.

1.3 PROBLEM STATEMENT

The objective of the project is to generate a caption for the image provided by the user by using Convolution Neural Network and Long Short-Term Memory technique and interacting with the user with the help of our web interface.

1.4 SCOPE OF PROJECT

This project automates the captioning process which was to be performed by an individual. The project helps in various ways such as saving the time and effort of providing caption to the images by a person. The user interacts to the project using the web interface. A web interface is a handy tool which provides users not only with captions but also suggestion for different similar captions and quotes which can be used by users. The project also plans to enhance the model by training it on the images given every time by the users by saving them too for the later use and learning from it.

1.5 ACCESS TO USERS

The project model is hosted on a web platform. The user can avail the service from any internet-enabled device, they just have to visit the specified web-page and provide the image. The model will perform the prediction and provide the user with a suitable caption along with some other suggestions of captions that can be copied by the user from there and then the user can use them on any platform of his/her choice.

CHAPTER - 2

RELATED WORK

2.1 INTRODUCTION

In this section, we will look into some of the earlier works presented in the direction of this project. There has been an extensive amount of work that has already happened in the field of image recognition and natural language processing. While preparing this project we have gathered ideas from different research materials and models. Here, we are going to enlist them.

2.2 EXISTING SYSTEM

Lately a lot of research has been done on automatic image captioning. The research can be briefly categorized into three different categories including the template-based approaches, retrieval-based approaches, and novel image caption generation approaches.

The template-based approach is aimed at generating captions by using fixed templates with a number of blank slots, in which way different objects, attributes, and actions are detected first and then the blank spaces in the templates are filled. For example, Farhadi use a triplet of scene elements to fill the template slots for generating image captions. Li et al extract the phrases related to detected objects, attributes, and their relationships for this purpose. Kulkarni et al. adopt a conditional random field (CRF) method to infer the objects, attributes, and prepositions before filling in the gaps. Template-based methods can generate grammatically correct captions. However, templates are predefined and length of captions cannot be variable.

The retrieval-based approach tries to generate description for an image by selecting the most semantically similar sentences from sentence pool or directly copying sentences from other visually similar images. For example, Gong et al. utilize stacked auxiliary embedding method to generate image descriptions from millions of weakly annotated images. Ordonez find similar images in the Flickr database and return the descriptions of these retrieved images to query based on millions of images and their corresponding descriptions. Sun use semantic similarity and visual similarity scores to cluster similar terms and images together first and then retrieve the caption of target image from captions of similar images in the same cluster. Hodosh establish a ranking-based framework to treat sentence-based image description as the task of ranking a set of captions for each test image. These methods generate general and

syntactically correct captions. However, it is difficult for them to generate image-specific and semantically correct captions.

Different from the mentioned two categories, novel caption generation approaches mainly use deep learning and machine learning to generate the new captions. A general implementation of this method is to analyze the visual content of the image first and then generate image captions from the visual content using a language model. For instance, Vinyals et al. use CNN as an encoder for image classification and LSTM as a decoder to generate sentence for the description. The main drawbacks of the work are the quick model overfitting, so they use the heavy and expensive GoogLeNet with 22 hidden layers and the absence of attention layer that significantly improved the description accuracy. Karpathy investigate the possibility of generating an image description in natural language. Their approach uses image datasets and their description in natural language and seeks an intermodal correspondence between words from the description and visual data. The first model aligns the fragments of sentences to the visual areas, then forms a single description by multimodal embedding. This description is treated as learning data for a second model of a recurrent neural network that learned to a generate caption. Xu et al. use a convolutional neural network to extract feature maps and LSTM to describe the input image, by processing already extracted feature maps. The limitation of this work is the using of obsolete and expensive Oxford VGGnet, where the quality of image classification is low in the modern CNN . Some researchers have put their attention on classification as Yu. who propose a SVM classification-based two-side cross-domain algorithm by inferring intrinsic user and item features (CTSIF-SVMs), a two-side cross-domain algorithm with expanding user and item features via the latent factor space of auxiliary domains (TSEUIF).

2.3 PROPOSED SYSTEM

Our proposed model, for automatic image captioning, is based on ResNet50 and LSTM. The ultimate purpose of this model is to generate the proper description for the given images. To do so, the model is designed with an encoder-decoder architecture based on CNN and RNN. In particular, to extract visual features, we use the ResNet50 network as the encoder to generate a one-dimensional vector representation of the input images. After that, to generate the description sentences, we adopt the LSTM as the language model for the decoder to decode the vector into a sentence.

2.3.1. Image Feature Extraction

For image feature extraction we used CNN, ResNet50, which is a deep network that has 50 layers. The image feature extractor needs an image of 224x224x3 size. The model uses ResNet50 pretrained on ImageNet dataset where the features of the image are extracted just

before the last layer of classification. Another dense layer is added and converted to get a vector of length 2048.

The model takes an image and produces a caption, encoded as a sequence of 1-K coded words.

$$y = \{y_1, y_2, \dots, y_c\}, y_i \in R^K,$$

where K is the size of the dictionary and c is the caption length. We use CNN in particular, ResNet50, to obtain set annotation vectors like the feature vectors. The extractor produces L-vectors, all of which is a D-dimensional representation of the corresponding part of an image.

2.3.2. The Language Model

In our design, we adopt LSTM as our language model to generate proper caption based on the input vector from the ResNet50 output.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f),$$

where the output vector of the previous cell h_{t-1} with the new element of the sequence x_t is concatenated and passed as one vector through the layer with the sigmoid activation function.

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t.$$

Two created vectors are used to update the state from C_{t-1} to C_t . To do this, we multiply the past state by f_t to “forget” the data recognized as unnecessary in the previous step, then add $i_t * \widetilde{C}_t$.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i),$$

$$\widetilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c).$$

The input gate must determine what values will be updated, and the tanh layer creates a vector of new candidates for \widetilde{C}_t , and values can be added to the cell state.

$$h_t = o_t * \tanh(C_t)$$

The obtained values of C_t and h_t are transmitted to the neural network input at a time $t+1$.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o),$$

$$p_t = \text{soft max}(h_t).$$

The multiplicative filters allow to effectively train LSTM, as they are good to prevent the exploding and vanishing gradients. Nonlinearity is provided by the sigmoid (σ) and the

hyperbolic tangent $\tanh()$. In the last equation, h_t is fed to the softmax function to calculate the probability distribution p_t over all the words. This function is calculated and optimized on the entire training dataset. The word with maximum probability is selected at each time step and fed into the next time step input to generate a full sentence.

2.3.3. Fitting the Model

After building the model, the model is fit using the training dataset. The model is made to run for 50 epochs and the best model is chosen among the 50 epochs by computing loss function on Flickr8k development dataset. The model with the lowest loss function is chosen for generating captions.

2.3.4. Caption Generation

To test our trained model, we input an image to the model. Next the image is fed into the feature extractor to recognize what all objects and scenes are depicted in the image, after resizing it. The process of caption generation is done using the RNN trained model. Then for that image, sequentially, word-by-word the caption is generated by selecting the word with maximum weight for the image at that particular iteration. The indexed word is converted to word and then appended into the final caption. When a tag is detected or the size of the caption reaches 40, the final caption is generated and printed along with the input image.

2.3.5. Suggestion API

Finally, in the web app, the description string is then passed to API which will suggest a few caption options to the user for selecting their preferred one.

CHAPTER - 3

PROBLEM DESCRIPTION AND SPECIFICATION

3.1 PROBLEM DESCRIPTION

The problem introduces a captioning task, which requires a computer vision system to both localize and describe salient regions in images in natural language. The image captioning task generalizes object detection when the descriptions consist of a single word.

3.2 SPECIFICATION

Image caption generator is a task that involves computer vision and natural language processing concepts to recognize the context of an image and describe them in a natural language like English.

The aim of image captioning is to automatically describe an image with one or more natural language sentences. This is a problem that integrates computer vision and natural language processing, so its main challenges arise from the need of translating between two distinct, but usually paired modalities. First, it is necessary to detect objects on the scene and determine the relationships between them and then, express the image content correctly with properly formed sentences. The generated description is still much different from the way people describe images because people rely on common sense and experience, point out important details and ignore objects and relationships that they imply. Moreover, they often use imagination to make descriptions vivid and interesting. The process flow diagram below in Figure 1 outlines the working of the project.

Process Flow

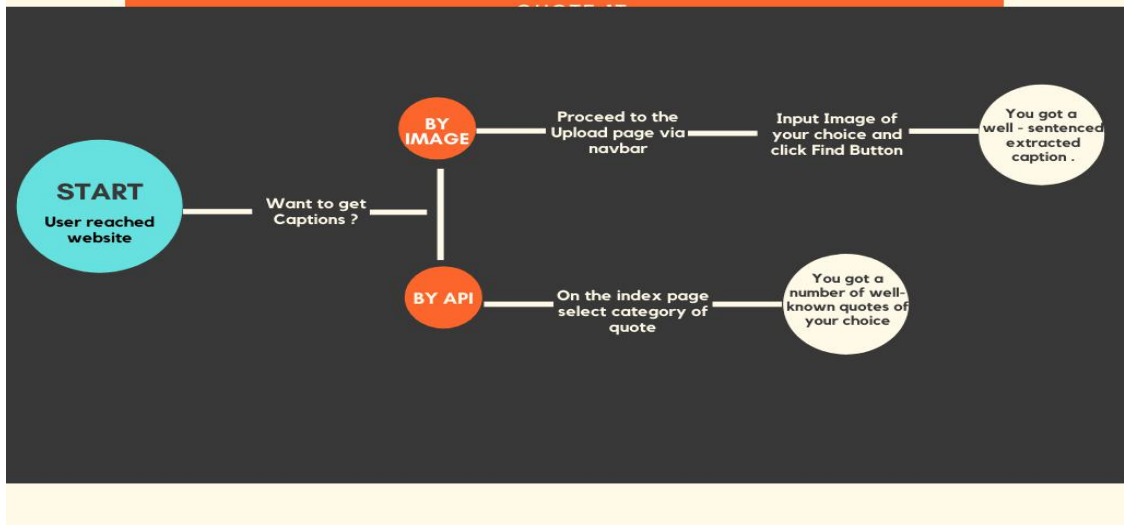


Figure 1: Process Flow

3.3 REQUIREMENTS

3.3.1 SOFTWARE REQUIREMENTS

- Python Version 3.6+
- IDE: JupyterLab or PyCharm and Sublime Text(for Web Development)
- Frameworks:-
 - Flask
 - Bootstrap
- Python Libraries:-
 - Tensorflow: 2.3.1
 - Keras: 2.4.3
 - Numpy: 1.18.5
 - Pandas: 1.1.14

3.3.2 HARDWARE REQUIREMENTS

- Processor:- Intel i5+ or AMD A8+
- RAM:- 4GB/+
- Hard Disk Space:- 20GB
- GPU:- 2GB (Anything more is a plus)

3.3.3 OTHER REQUIREMENTS

- Flickr 8k Dataset containing images and corresponding caption data^[11]
- For training the model - Kaggle Notebook or Google Colab should be preferred (Recommended)
- For API- GoodQuotes API

CHAPTER - 4

SYSTEM DESIGN

4.1 INTRODUCTION

We have written data preprocessing scripts to process raw input data (both images and captions) into proper format; A pre-trained Convolutional Neural Network architecture as an encoder to extract and encode image features into a higher dimensional vector space; An LSTM-based Recurrent Neural Network as a decoder to convert encoded features to natural language descriptions; Attention mechanism which allows the decoder to see features from a specifically highlighted region of the input image to improve the overall performance

4.1.1. Data Sources

We have identified the three most commonly used image caption training datasets in the Computer Vision research domain - COCO dataset [6], Flickr8k [3] and Flickr30k [8]. These datasets contain 123,000, 31,000 and 8,000 captions annotated images respectively and each image is labelled with 5 different descriptions. Currently, Flickr8k dataset which contains the least number of images is used as our primary data source over the other two due to our limited storage and computational power.

4.1.2. Convolutional Neural Network (Encoder)

The encoder needs to extract image features of various sizes and encode them into vector space which can be fed to RNN in a later stage. VGG-16 and ResNet are commonly recommended as image encoders. We chose to modify the pre-trained ResNet50 model provided by Tensorflow library. In this task, CNN is used to encode features instead of classifying images. As a result, we removed the fully connected layers and the max pool layers at the end of the network. Under this new construction, the input image matrix has dimension $N \times 3 \times 256 \times 256$, and the output has dimension $N \times 14 \times 14 \times 512$. Furthermore, in order to support input images with various sizes, we added an adaptive 2d layer to our CNN architecture.

4.1.3. Recurrent Neural Network (Decoder)

The decoder needs to generate image captions word by word using a Recurrent Neural Network - LSTMs which is able to sequentially generate words. The input for the decoder is the encoded image feature vectors from CNN and the encoded image captions produced in data preprocessing stage.

4.2 ARCHITECTURE DIAGRAM

The Backend System architecture Diagram is as follows:

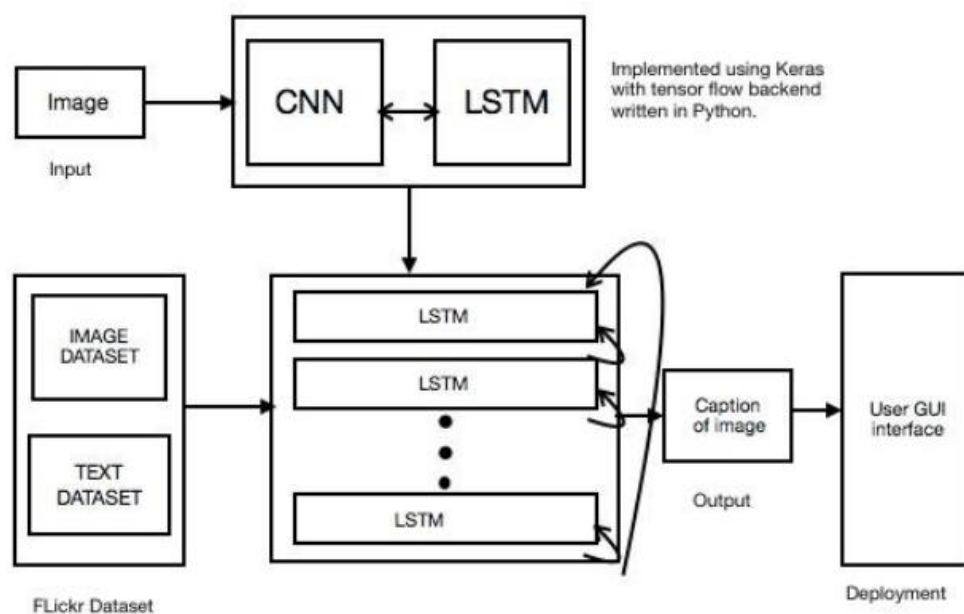


Figure 2: Architecture of the image captioning model

The model uses ResNet50 as the CNN model and generated Region features of the passed images which are feed to RNN(LSTM) network which is fed to the model for predicting the next word in the captions on the basis of highest probability obtained. All these next words are added to generate the captions till the final caption is generated. Figure 2 shows a brief architecture for the project. The image is passed and fed to CNN and LSTM model, which internally is composed of various layers and fed with regional image and text dataset. At the user interacts with the project through the GUI deployed for them. Figure 3 explains the process used by CNN i.e, object detection and feature map preparation. These features along with text dataset are given to the vector of LSTM model with predicts the objects in the

image and return the word in the sequence which has the highest probability which is used for generating the caption.

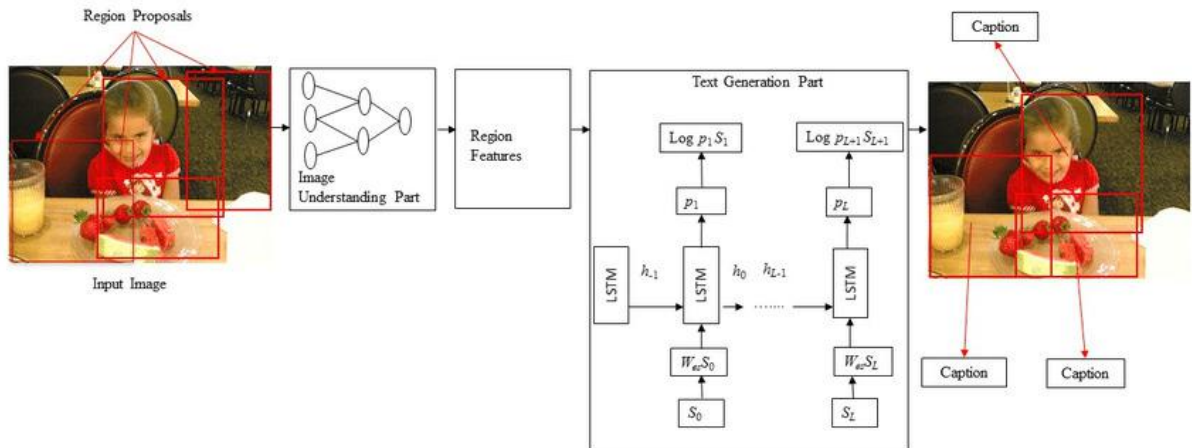


Figure 3: Illustration image of functioning of the captioning model

4.3 UNIFIED MODELING LANGUAGE(UML)

4.3.1 USE CASE DIAGRAM

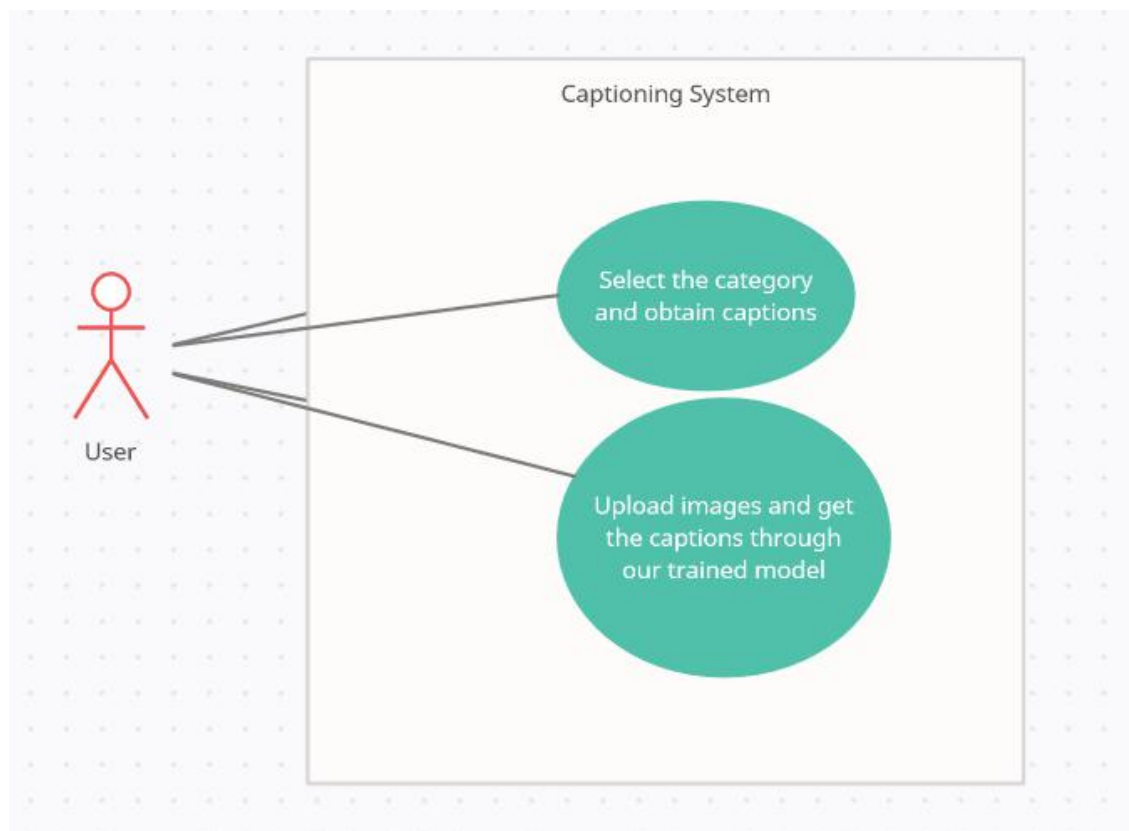


Figure 4: Use Case Diagram for the captioning system

Figure 4 shows that there are basically two use-cases available for the user. When the user visits the web site they can choose:

- 1.) Any category available on the home page to obtain the collection of images of that category and then select and copy the caption of their choice.
- 2.) User can navigate to the Upload section and then upload the image from their device and then by the help of the image caption generator model running in the background he/she can obtain the caption for the provided image.

4.3.2 DATA FLOW DIAGRAM

4.3.2.1 Data Flow Diagram Level 0

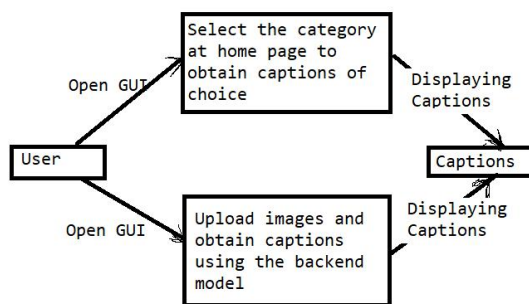


Figure 5: DFD level 0

In Figure 5 the level 0 DFD shows the top-level view of the data flow. It shows that initially when the user interacts with the GUI, they have two options. The first one is to choose a caption of their favourite category and the second option is to provide an image to the model for generation caption.

4.3.2.2 Data Flow Diagram Level 1

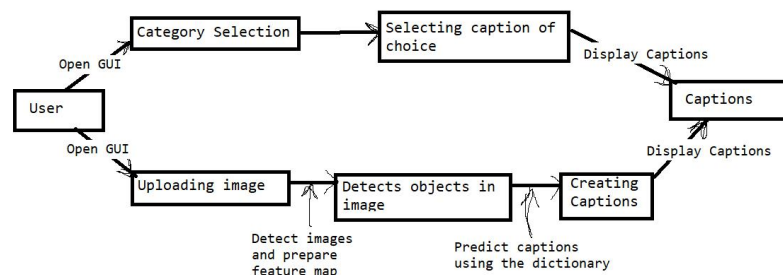


Figure 6: DFD level 1

In figure 6 we see the medium level view of data flow. It shows how the input passed goes through a channel of different segments. In our case when a user uploads the image it passes through the CNN model and the features of the image generated are then passed to Language model which gives the captions to the GUI for display.

4.3.2.3 Data Flow Diagram Level 2

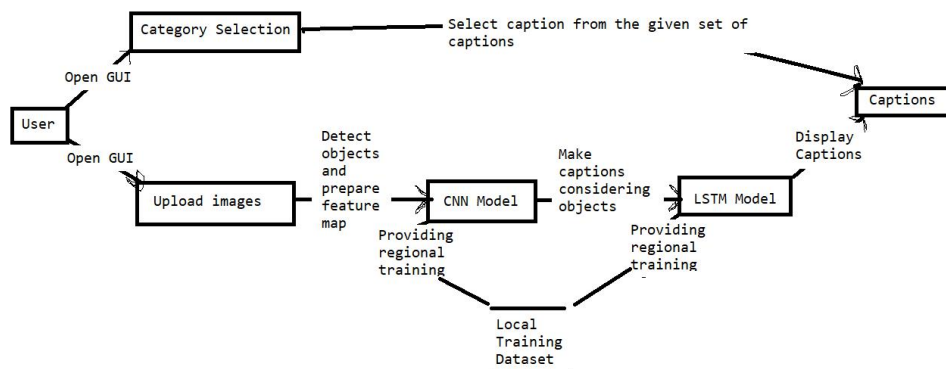


Figure 7: DFD level 2

In figure 7 explains the functioning of the model in great depth. The first option remains the same, but the second option shows the usage of a regional dataset to train the model. This improves the functioning of the model by training the model on our custom dataset.

4.4 STATE CHART DIAGRAM

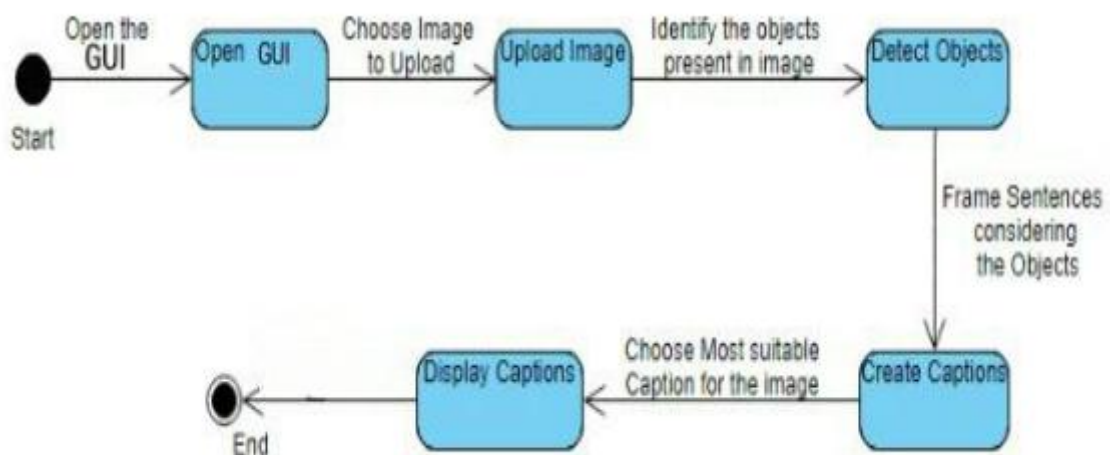


Figure 8: State Chart Diagram of the system

Figure 8 shows the state of the program at every deciding state and its output which are fed to the next state as input. The program basically contains five states as stated in the figure. The GUI and uploading of the image are handled by the flask library. The object detection phase uses the CNN model to detect the objects in the figure and then the create caption state uses the LSTM model to generate the captions.

4.5 FLOW CHART FOR CAPTION GENERATION PROCESS

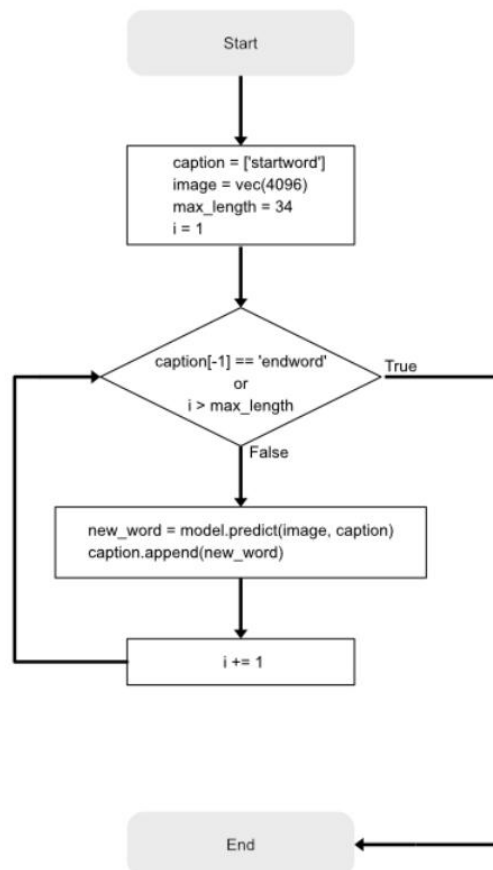


Figure 9: Flow chart for the caption generation process

Figure 9 tells us about the functioning of the caption prediction task. Once the image feature of the passed test image is extracted. The trained model tries to predict the next word in the caption by using the Dense layer and agrmax function. The inverse dictionary is used for obtaining the word corresponding to the integer provided. When the “endword” is passed to the model, it stops and the final caption is given in output.

CHAPTER - 5

IMPLEMENTATION AND RESULT

5.1 OVERVIEW

In this section, we are going to cover all the steps we have taken while implementing the program. The whole program consists of nine different sections which we are going to cover in later sections.

5.2 MODEL SELECTION

We have used the ResNet50 model of the TensorFlow in our project for image processing as shown in Figure 10. We first import ResNet50 from the Tensorflow module, which can be found in tensorflow.keras.applications module. Once we import the model, it will download the model from the TensorFlow website and save it into the 'incept_model' variable. Then we remove the last two layers of the ResNet50 model as the last to later are not of our use. We also save the model in our local directory using the 'save' method; so that it can be used later in offline mode.

```
# 1.Selecting the model*****
from tensorflow.keras.applications import ResNet50
incept_model = ResNet50(include_top=True)

# 1A. Adjusting the model
from tensorflow.keras.models import Model
last = incept_model.layers[-2].output
modele = Model(inputs = incept_model.input, outputs = last)
modele.summary()

# 1B. Saving the model for offline use
modele.save('ResNet50.h5')
print('ResNet50 model saved')
```

Figure 10: Code Snippet showing the selection and adjustment of the model

```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels.h5
102973440/102967424 [=====] - 1s 0us/step

```

```
modele.summary()
```

```

conv5_block3_2_relu (Activation (None, 7, 7, 512) 0 conv5_block3_2_bn[0][0]
-----
conv5_block3_3_conv (Conv2D) (None, 7, 7, 2048) 1050624 conv5_block3_2_relu[0][0]
-----
conv5_block3_3_bn (BatchNormali (None, 7, 7, 2048) 8192 conv5_block3_3_conv[0][0]
-----
conv5_block3_add (Add) (None, 7, 7, 2048) 0 conv5_block2_out[0][0]
conv5_block3_3_bn[0][0]
-----
conv5_block3_out (Activation) (None, 7, 7, 2048) 0 conv5_block3_add[0][0]
-----
avg_pool (GlobalAveragePooling2 (None, 2048) 0 conv5_block3_out[0][0]
=====
Total params: 23,587,712
Trainable params: 23,534,592
Non-trainable params: 53,120
=====

```

Figure 11: Output representing different layers of the selected model

5.3 IMAGE PREPROCESSING

We use ‘Flickr8k Dataset’ which we have downloaded and saved in the local directory as shown in Figure 12. By using the ResNet50 model we are extracting image features and saving it for later use in a pickle file named ‘features.p’.

```

# 2.Image Preprocess*****
# 2A. Providing image path

#images_path = 'input/Flicker8k_Dataset/' #for online testing
images_path = "C:\\Users\\msn21\\Desktop\\Minor Project\\pratika\\Flicker8k_Dataset\\"
images = glob(images_path+'*.jpg')
print('Length of images:', len(images))
print(images[0].split('/')[ -1])
print(images[0].split('\\')[ -1])

print('Sample first 5 images')
print(images[:5])

|
# 2B. Testing by fetching some images

import matplotlib.pyplot as plt

for i in range(5):
    plt.figure()
    img = cv2.imread(images[i])
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.imshow(img)

```

Figure 12: Code snippet showing preprocessing of images

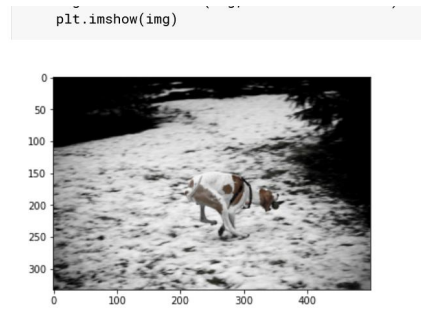


Figure 13: A sample picture fetched from the dataset

Figure 14 shows the process of creating a feature map of a specified limit of images from the regional image dataset. Here we have selected 1500 images for the training purpose because of hardware limitations. The image in the images array is fetched and converted to a coloured and resized image of 224 x 224 x 3 dimension. Then our defined ResNet 50 model as “modele” extracts and store the image feature in the image feature variable in the NumPy array form. The process of extraction is shown in figure 15 and figure 16 shows a snap of image features of some sample images.

```
# 2C. Preparing Feature Map of Images

print('Preparing Feature Map..')
images_features = {}
count = 0
for i in images:
    img = cv2.imread(i)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (224,224))

    img = img.reshape(1,224,224,3)
    pred = modele.predict(img).reshape(2048,)

    im_name = i.split('\\')[-1]
    #img_name=im_name[18:43]
    img_name = im_name

    images_features[img_name] = pred

    count += 1

    if count > 1499:
        break

    elif count % 100 == 0:
        print(count)
```

Figure 14: Extracting feature map from the passed images

```

Processed: 350
Processed: 400
Processed: 450
Processed: 500
Processed: 550
Processed: 600
Processed: 650
Processed: 700
Processed: 750
Processed: 800
Processed: 850
Processed: 900
Processed: 950
Processed: 1000
Processed: 1050
Processed: 1100
Processed: 1150
Processed: 1200
Processed: 1250
Processed: 1300
Processed: 1350
Processed: 1400
Processed: 1450
Extracted image feature of 1500 images

```

```

images_features
{
  '3226254560_2f8ac147ea.jpg': array([6.2825638e-01, 7.8491366e-01, 5.6670177e-01, ..., 8.9139181e-01,
    4.5924771e-05, 0.0000000e+00], dtype=float32),
  '214543992_ce6c0d9f9b.jpg': array([0.46220586, 0.29161927, 0.06892882, ..., 0.573068 , 0.
    0.28785384], dtype=float32),
  '2366643786_9c9a830db8.jpg': array([0.8897092 , 0.16498396, 0.03830731, ..., 0.2704721 , 0.678553
    04, 0.15241733], dtype=float32),
  '3368819708_0bfa0808f8.jpg': array([0.4930518 , 0.648831 , 0.03944141, ..., 0.49528658, 0.048742
    61, 0.
    ], dtype=float32),
  '2190227737_6e0bde2623.jpg': array([0.82795006, 0.
    , 0.98436755, ..., 0.12665229, 0.509681
    17, 2.5331454 ], dtype=float32),
  '2752809449_632cd991b3.jpg': array([0.24190295, 0.35491422, 0.03130041, ..., 0.
    , 0.128231
    11

```

Figure 15: Output representing the process of feature map creations and sample view of the feature map of some images

5.4 TEXT PREPROCESSING

We are using ‘Flickr8k.token.txt’ which contains five captions for all the images in the Flickr8k Image dataset. We split them into different lines and convert them in string form,

ready for our operation as shown in figure 16 and figure 17. Then we prepare a dictionary consisting of images as the keys and a list of 5 captions as the values for all the images in the feature map that we saved earlier. Then we modify the captions by adding ‘startofseq’ word at the front of every caption denoting the start of the caption and ‘endofseq’ at the end of every caption denoting the end of that particular caption. Then we again attach these modified captions with the respective images as shown in figure 18 and the corresponding results in figure 19. We did this for preparing the caption vector which will later be used by our model for prediction.

```
# 3. Text Preprocess*****

# 3A. Providing Path of captions

#caption_path = 'input/Flickr_8k_text/Flickr8k.token.txt' # for online testing
caption_path = "C:\\Users\\msn21\\Desktop\\Minor Project\\pratika\\Flickr8k_Text\\Flickr8k.token.txt"

# 3B. Splitting captions line-wise and converting to right format

captions = open(caption_path, 'rb').read().decode('utf-8').split('\n')

# 3C. Testing captions

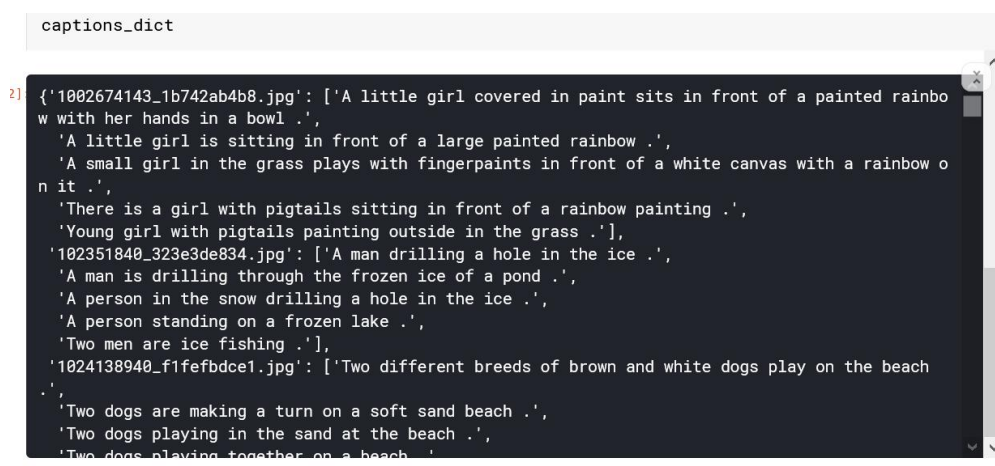
print('Captions..')
print(captions)
print()

print('Length of captions....')
print(len(captions))
print()

print('Sample Image')
print(captions[0].split('\t')[0][:-2])
print()

print('Sample Description')
print(captions[0].split('\t')[1])
print()
```

Figure 16: Code snippet showing text preprocessing



```
captions_dict
2) {'1002674143_1b742ab4b8.jpg': ['A little girl covered in paint sits in front of a painted rainbow with her hands in a bowl .',
  'A little girl is sitting in front of a large painted rainbow .',
  'A small girl in the grass plays with fingerpaints in front of a white canvas with a rainbow on it .',
  'There is a girl with pigtails sitting in front of a rainbow painting .',
  'Young girl with pigtails painting outside in the grass .'],
 '102351840_323e3de834.jpg': ['A man drilling a hole in the ice .',
  'A man is drilling through the frozen ice of a pond .',
  'A person in the snow drilling a hole in the ice .',
  'A person standing on a frozen lake .',
  'Two men are ice fishing .'],
 '1024138940_f1fefbdce1.jpg': ['Two different breeds of brown and white dogs play on the beach .',
  'Two dogs are making a turn on a soft sand beach .',
  'Two dogs playing in the sand at the beach .',
  'Two dogs playing together on a beach .']}
```

Figure 17: Sample captions fetched from Flickr8k.token.txt


```

# 3D. Creating a Dictionary with image number as key and all five captions in list form as values
captions_dict = {}
for i in captions:
    try:
        img_name = i.split('\t')[0][:2]
        caption = i.split('\t')[1]
        if img_name in images_features:
            if img_name not in captions_dict:
                captions_dict[img_name] = [caption]
            else:
                captions_dict[img_name].append(caption)
    except:
        pass

# 3E. Testing caption dictionary
print("Caption Dictionary")
print(captions_dict)
print()

print('Length of caption_dict..')
print(len(captions_dict))

# 3F. Adding 'startofseq' and 'endofseq' into all captions to represent the start and end of the captions
def preprocessed(txt):
    modified = txt.lower()
    modified = 'startofseq ' + modified + ' endofseq'
    return modified

# 3G. Attaching modified captions with there respective images
for k,v in captions_dict.items():
    for vv in v:
        captions_dict[k][v.index(vv)] = preprocessed(vv)

```

Figure 18: Code snippet representing the modification process of the captions and caption dictionary creation

```

captions_dict
{
  '1002674143_1b742ab4b8.jpg': ['startofseq a little girl covered in paint sits in front of a paint
ed rainbow with her hands in a bowl . endofseq',
  'startofseq a little girl is sitting in front of a large painted rainbow . endofseq',
  'startofseq a small girl in the grass plays with fingerpaints in front of a white canvas with a
rainbow on it . endofseq',
  'startofseq there is a girl with pigtails sitting in front of a rainbow painting . endofseq',
  'startofseq young girl with pigtails painting outside in the grass . endofseq'],
  '102351840_323e3de834.jpg': ['startofseq a man drilling a hole in the ice . endofseq',
  'startofseq a man is drilling through the frozen ice of a pond . endofseq',
  'startofseq a person in the snow drilling a hole in the ice . endofseq',
  'startofseq a person standing on a frozen lake . endofseq',
  'startofseq two men are ice fishing . endofseq'],
  '1024138940_f1fefbdce1.jpg': ['startofseq two different breeds of brown and white dogs play on th
e beach . endofseq',
  'startofseq two dogs are making a turn on a soft sand beach . endofseq',

```

Figure 19: Sample view of caption dictionary

5.5 CREATING VOCABULARY

Next, ahead we count the number of times any particular word appears in any of the captions among all the captions present as shown in figure 20 code snippet. Then we provide an integer number to every word forming a dictionary of different words present in the

captions as shown in results in figure 22. Then we replace every word in every caption with the integer key corresponding to it as the value in the caption dictionary which earlier had images and the list of its corresponding five captions. The new caption dictionary contains images and corresponding captions but in integer form as shown in figure 23.

```
# 4. Create Vocabulary*****
# 4A. Counting the number of times any particular word present in any caption has appeared among
# all the captions and creating its dictionary

count_words = {}
for k,vv in captions_dict.items():
    for v in vv:
        for word in v.split():
            if word not in count_words:
                count_words[word] = 0
            else:
                count_words[word] += 1

print('Length of count words..')
print(len(count_words))
print()

print('Count_Words..')
print(count_words)

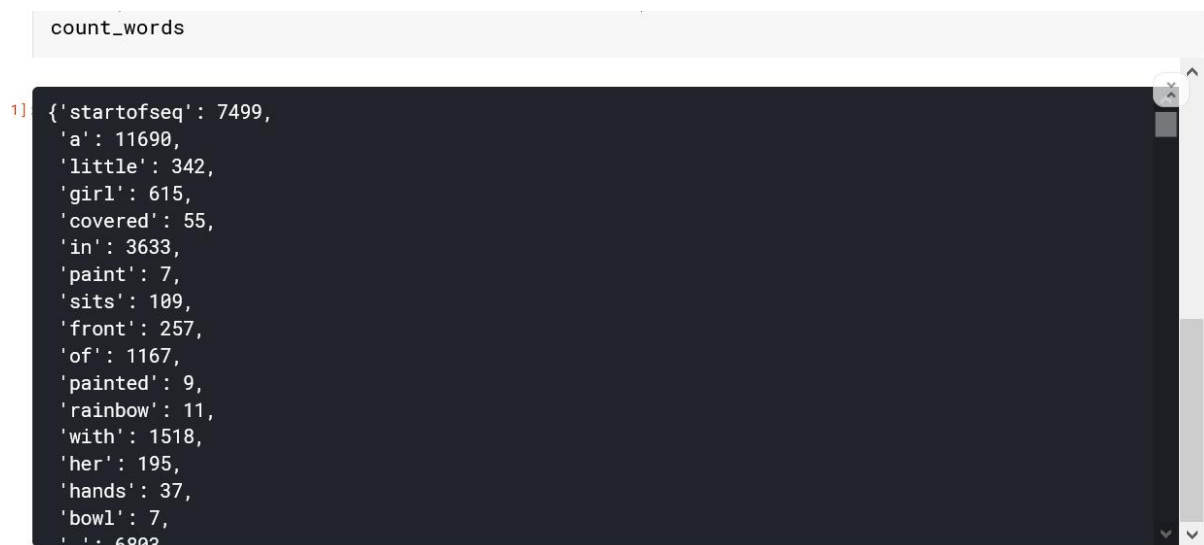
# 4B. Converting dictionary to integer form - as neural network can process integers not strings
# ie allotting every string a integer for recognition

THRESH = -1
count = 1
new_dict = {}
for k,v in count_words.items():
    if count_words[k] > THRESH:
        new_dict[k] = count
        count += 1
    |

# printing the integer form of the dictionary
print('New Dictionary')
print(new_dict)

# length of integer form of dictionary - should be equal to count words
print('Length of New Dictionary')
print(len(new_dict))
```

Figure 20: Code snippet showing Vocabulary Creation Process



```
count_words
1) {'startofseq': 7499,
  'a': 11690,
  'little': 342,
  'girl': 615,
  'covered': 55,
  'in': 3633,
  'paint': 7,
  'sits': 109,
  'front': 257,
  'of': 1167,
  'painted': 9,
  'rainbow': 11,
  'with': 1518,
  'her': 195,
  'hands': 37,
  'bowl': 7,
  '!': 6803}
```

Figure 21: Sample output showing the count of different words


```
new_dict
1 {'startofseq': 1,
  'a': 2,
  'little': 3,
  'girl': 4,
  'covered': 5,
  'in': 6,
  'paint': 7,
  'sits': 8,
  'front': 9,
  'of': 10,
  'painted': 11,
  'rainbow': 12,
  'with': 13,
  'her': 14,
  'hands': 15,
  'bowl': 16,
  '': 17}
```

Figure 22: Sample output showing the word dictionary

```
captions_dict
1 {'103205630_682ca7285b.jpg': [[1, 2, 68, 19, 69, 70, 71, 29, 23, 39, 17, 18],
  [1, 47, 48, 49, 72, 73, 74, 75, 39, 50, 68, 29, 2, 44, 5, 46, 17, 18],
  [1, 47, 48, 76, 71, 2, 77, 39, 50, 78, 29, 75, 79, 80, 46, 18],
  [1, 47, 48, 81, 45, 29, 75, 39, 81, 82, 83, 84, 5, 13, 2, 77, 85, 17, 18],
  [1, 47, 86, 45, 35, 2, 77, 68, 87, 29, 2, 88, 89, 17, 18]]}
```

Figure 23: Sample output showing the integer form captions attached with there respective images

5.6 PREPARING MODEL

5.6.1 Preparing Input for the model

Now we need to obtain the maximum length of the caption that is present in our captions, this will be one of the inputs of the model which will help it decide the length of the caption to be predicted. Vocab size is the number of words we have in the dictionary, this would also be one of the inputs to the model. The process of obtaining the max length caption in our collection is shown in figure 24 with results in figure 25. We use a generator function to prepare the input for training the model. The outputs of the generator function are three NumPy arrays which contain the image and the first words and the next predicted word (the word having the highest probability). The generator method is displayed in figure 26 and a few the variables are created in figure 27 code snippet along with some sample values. Figure 28 shows the length and shape of input which are fed to the model for training.

```

# 5.Build Generator Function*****
from keras.utils import to_categorical
from keras.preprocessing.sequence import pad_sequences

# 5A. Obtaining maximum length of caption present for any caption for any image
MAX_LEN = 0
for k, vv in captions_dict.items():
    for v in vv:
        if len(v) > MAX_LEN:
            MAX_LEN = len(v)
            print(v)

# max length of caption
print()
print('max length:', MAX_LEN)
print()

# batch size is no of images to be processed at a time
# vocab size is the size of dictionary formed using words from captions
Batch_size = 5000
VOCAB_SIZE = len(new_dict)

```

Figure 24: Code snippet showing the max length of the caption present

```

MAX_LEN

Maximum length caption in our set of captions

36

```

Figure 25: Output showing the length of the largest caption in our collection

```

# 5B. Function for generating input and output sequence for the model
def generator(photo, caption):
    n_samples = 0

    X = []
    y_in = []
    y_out = []

    for k, vv in caption.items():
        for v in vv:
            for i in range(1, len(v)):
                # appending image features to variable X
                X.append(photo[k])

                # from passing the input sequence already known - like from 0 upto 1 then from 0 upto 2 then 0 upto 3 and so on..
                in_seq = v[:i]
                # next word to come in sequence
                out_seq = v[i]

                # padding the sequences at end with zero to form every captions of max length
                in_seq = pad_sequences([in_seq], maxlen=MAX_LEN, padding='post', truncating='post')[0]
                # providing probability number for every output word in dictionary that can arrive next in the sequence
                # the out_seq word will have the highest probability ie 1 other will have zero or 0<=P(E)<1
                # this is provided as output layer in our model is Dense layer using softmax - which provides probability to every word
                out_seq = to_categorical([out_seq], num_classes=VOCAB_SIZE)[0]

                y_in.append(in_seq)
                y_out.append(out_seq)

    return X, y_in, y_out

# 5C. Collecting input and output to be supplied for training
X, y_in, y_out = generator(images_features, captions_dict)

```

Figure 26: Code snippet showing the input form preparation which will be fed to the model

```

# 5D. Checking length of inputs to be passed
print('Xlen', 'y_inlen', 'y_outlen')
print(len(X), len(y_in), len(y_out))

# 5E. Converting to numpy array - form required for model.fit()
X = np.array(X)
y_in = np.array(y_in, dtype='float64')
y_out = np.array(y_out, dtype='float64')

# 5F. Checking the shape of input
print('Xshape', X.shape)
print('y_inshape', y_in.shape)
print('y_outshape', y_out.shape)

# sample input value for a given array value in numpy array
print('X[710]')
print(X[710])

print('y_in[2]')
print(y_in[2])

```

Figure 27: Code snippet showing the testing of shapes of inputs and conversion to NumPy array form

```

Length of different inputs that are to be given to the model for training
X length: 96528
y_in length: 96528
y_out length: 96528

```

Shape of inputs

```

((96528, 2048), (96528, 36), (96528, 4074))

```

Figure 28: Sample output showing the shapes of input

5.7 CONFIGURING THE MODEL

5.6.2.1 Configuring Image Model

We use the ‘Sequential’ model present in tensorflow.keras.model library to sequentially add different layers which we want to use for processing the image. We have a ‘Dense layer’ and ‘RepeatVector layer’ and ‘relu’ activation function. The code snippet in Figure 29 highlights the process of configuration of image and language model. Figure 30 and Figure 31 shows the different layers present in the image and language model selected and different parameters they work on.

```

# 6.Configuring Model*****
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.utils import plot_model
from keras.layers.merge import add
#from keras.callbacks import ModelCheckpoint
from tensorflow.keras.layers import Dense, Input, Convolution2D, Dropout, LSTM, TimeDistributed, Embedding,
from tensorflow.keras.models import Sequential, Model

print()
print('Preparing model...')
embedding_size = 128
max_len = MAX_LEN
vocab_size = len(new_dict)

# 6A. Configuring Image model
print('Image Sequential model')
image_model = Sequential()

image_model.add(Dense(embedding_size, input_shape=(2048,), activation='relu'))
image_model.add(RepeatVector(max_len))

image_model.summary()

# 6B. Configuring Language model
print('Language Sequential model')
language_model = Sequential()

language_model.add(Embedding(input_dim=vocab_size, output_dim=embedding_size, input_length=max_len))
language_model.add(LSTM(256, return_sequences=True))
language_model.add(TimeDistributed(Dense(embedding_size)))

language_model.summary()

# 6C. Combining both the models
print('Combining both image and language Sequential model')
conca = Concatenate()([image_model.output, language_model.output])
x = LSTM(128, return_sequences=True)(conca)
x = LSTM(512, return_sequences=False)(x)
x = Dense(vocab_size)(x)
out = Activation('softmax')(x)
model = Model(inputs=[image_model.input, language_model.input], outputs = out)

```

Figure 29: Code snippet showing the configuring process of the image and language model

Configuring Image Model		
Model: "sequential"		
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	262272
repeat_vector (RepeatVector)	(None, 36, 128)	0
Total params: 262,272		
Trainable params: 262,272		
Non-trainable params: 0		

Figure 30: Output showing the image sequential model

```
Configuring Language Model
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 36, 128)	521472
lstm (LSTM)	(None, 36, 256)	394240
time_distributed (TimeDistri	(None, 36, 128)	32896

```
Total params: 948,608
Trainable params: 948,608
Non-trainable params: 0
```

Figure 31: Output showing the language sequential model

Both of the models i.e. image and language model are combined using the concatenate layer and the resulting model layers are shown in figure 32. Figure 33 is the descriptive model generated through Keras plot model method. This shows different layers along with the shapes and when they are added to the model and the output generated by them.

```
Combining the model
Model: "functional_3"
```

Layer (type)	Output Shape	Param #	Connected to
embedding_input (InputLayer)	[(None, 36)]	0	
dense_input (InputLayer)	[(None, 2048)]	0	
embedding (Embedding)	(None, 36, 128)	521472	embedding_input[0][0]
dense (Dense)	(None, 128)	262272	dense_input[0][0]
lstm (LSTM)	(None, 36, 256)	394240	embedding[0][0]
repeat_vector (RepeatVector)	(None, 36, 128)	0	dense[0][0]
time_distributed (TimeDistribut	(None, 36, 128)	32896	lstm[0][0]
concatenate (Concatenate)	(None, 36, 256)	0	repeat_vector[0][0] time_distributed[0][0]
lstm_1 (LSTM)	(None, 36, 128)	197120	concatenate[0][0]

Figure 32: Output showing the combined model

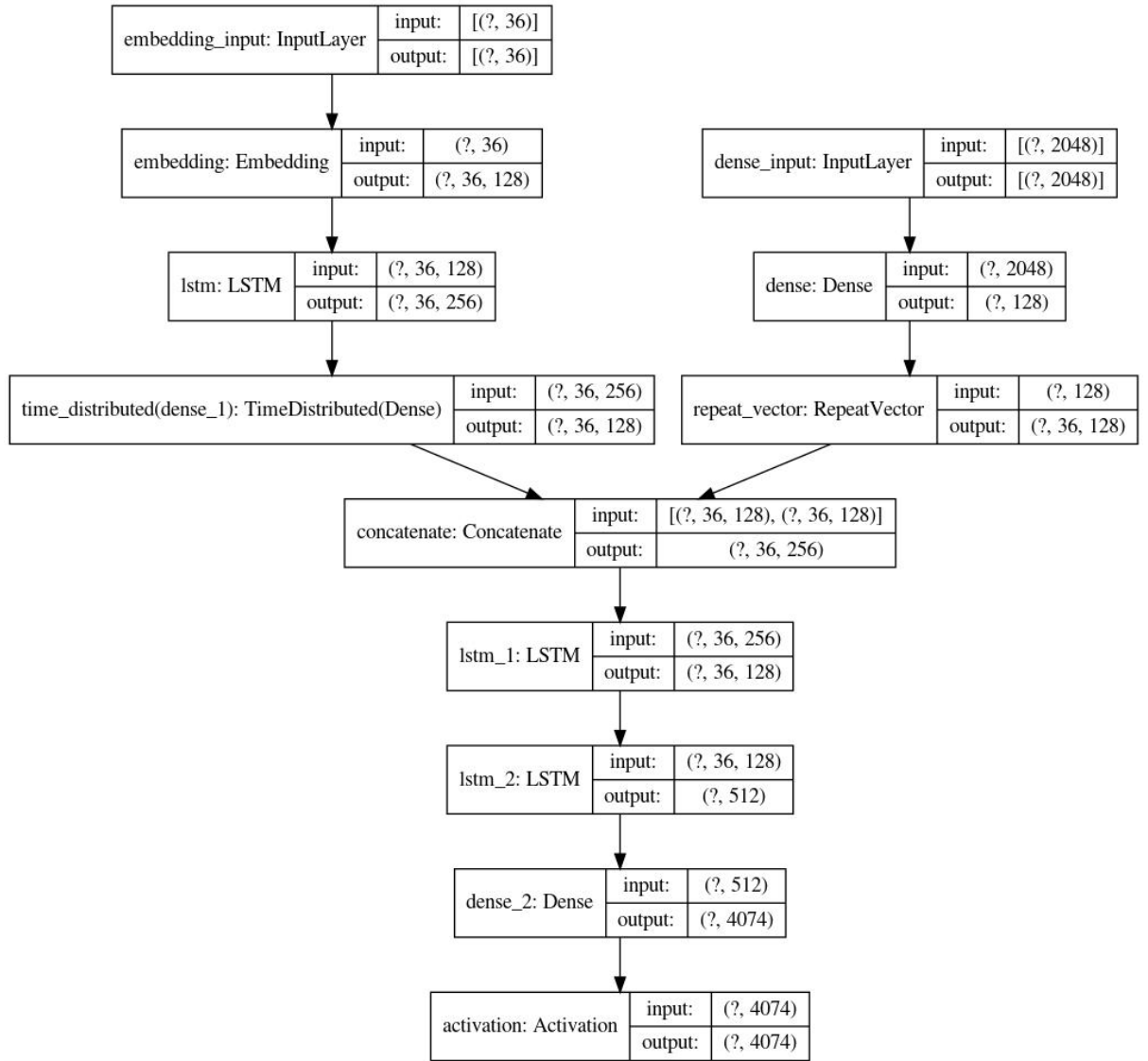


Figure 33: Detailed layout of the model along with shapes of inputs

5.8 TRAINING AND SAVING THE MODEL

We put the model on training with the inputs we have generated through the generator function in the above section using TensorFlow model.fit() function for 50 epochs as shown in figure 34. Once the training is complete we save the trained model for later use and operating in offline mode. An output snippet showing the training process is attached in figure 35. Then we prepare an inverse dictionary where every number of the word dictionary is now the key and every word of the dictionary is value. The inverse dictionary sample output is displayed in figure 36. This will be used in prediction when our model will give the number out of the dictionary having the highest probability we will check the corresponding value and return that word.


```

# 7. Training the model*****
print()
print('Training the model')
model.fit([X, y_in], y_out, batch_size=512, epochs=10)
print()

# saving the model
print('Saving the model..')
model.save('trainedmodel.h5')
print('Saved Trained model')

# saving model weights
model.save_weights('mine_model_weights.h5')

# preparing inverted dictionary - here value will be key and key will be value
print('Preparing inverted dictionary')
inv_dict = {v:k for k, v in new_dict.items()}

# saving numpy vocab dictionary
np.save('vocab.npy', new_dict)

```

Figure 34: Code snippet showing the training process and creation of an inverse dictionary

```

189/189 [=====] - 13s 67ms/step - loss: 1.3277 - accuracy: 0.6549
Epoch 44/50
189/189 [=====] - 12s 66ms/step - loss: 1.2880 - accuracy: 0.6626
Epoch 45/50
189/189 [=====] - 13s 67ms/step - loss: 1.2446 - accuracy: 0.6742
Epoch 46/50
189/189 [=====] - 13s 68ms/step - loss: 1.2067 - accuracy: 0.6811
Epoch 47/50
189/189 [=====] - 12s 66ms/step - loss: 1.1667 - accuracy: 0.6880
Epoch 48/50
189/189 [=====] - 13s 67ms/step - loss: 1.1290 - accuracy: 0.6982
Epoch 49/50
189/189 [=====] - 13s 67ms/step - loss: 1.0929 - accuracy: 0.7056
Epoch 50/50
189/189 [=====] - 13s 67ms/step - loss: 1.0606 - accuracy: 0.7134
Training Done

```

Figure 35: Sample output of the training process

```

inv_dict
{1: 'startofseq',
 2: 'a',
 3: 'little',
 4: 'girl',
 5: 'covered',
 6: 'in',
 7: 'paint',
 8: 'sits',
 9: 'front',
10: 'of',
11: 'painted',
12: 'rainbow',
13: 'with',
14: 'her',
15: 'hands',

```

Figure 36: Sample output of the inverse dictionary

5.9 PREDICTING THE CAPTIONS

Now we have stored the ResNet50 model, trained model, dictionary of words and inverted dictionary. We can use these stored files to run or model on images to predict the captions. We have prepared a function which can fetch the images from our test folder and provide it one by one for predicting the captions. Figure 37 shows the process of preparation of test image provided by the user.

```
# 8. Predictions*****
print('Testing Phase...')
# path for providing the images
test_img_path = 'C:\Users\msn21\Desktop\Minor Project\new\DESCIT\DESCIT\Predict\Test Images\'
test_images = glob(test_img_path+'*.jpg')

# 8A. Function for preparing test set for testing
def getImage(x):
    test_img_path = test_images[x]

    test_img = cv2.imread(test_img_path)
    test_img = cv2.cvtColor(test_img, cv2.COLOR_BGR2RGB)

    test_img = cv2.resize(test_img, (224,224))

    test_img = np.reshape(test_img, (1,224,224,3))

    return test_img

# sample test feature of random image (image no. 2500)
test_feature = modele.predict(getImage(2500)).reshape(1,2048)

print('Printing test features')
print(test_feature)
```

Figure 37: Code snippet showing the image preprocessing for the testing purpose

```
# 8B. Selecting random images from dataset for prediction - and displaying them
for i in range(5):

    no = np.random.randint(0000,1500, (1,1)) [0,0]
    test_feature = modele.predict(getImage(no)).reshape(1,2048)

    test_img_path = test_images[no]
    test_img = cv2.imread(test_img_path)
    test_img = cv2.cvtColor(test_img, cv2.COLOR_BGR2RGB)
    # giving the first caption word for model to start
    text_inp = ['startofseq']

    count = 0
    caption = ''
    max_cap_length = 36
    while count < max_cap_length:
        count += 1

        encoded = []
        for i in text_inp:
            encoded.append(new_dict[i])

        encoded = [encoded]

        encoded = pad_sequences(encoded, padding='post', truncating='post', maxlen=MAX_LEN)
        # model predicts the number in dictionary with the highest probability
        prediction = np.argmax(model.predict([test_feature, encoded]))
        # we are using the inverse dictionary to find the corresponding word
        sampled_word = inv_dict[prediction]

        if sampled_word == 'endofseq':
            break

        caption = caption + ' ' + sampled_word

        text_inp.append(sampled_word)

    plt.figure()
    plt.imshow(test_img)
    plt.xlabel(caption)

print("Done !!")
print('Find captions below:')
```

Figure 38: Code snippet showing the prediction process and results



Figure 39: Some sample results generated by the model

Figure 38 code snippet shows the batch mode operation of the model which selects some random images from the test images folder and provides captions for them. Few of the results are attached in Figure 39.

5.10 WEB PAGE DEVELOPMENT

For the purpose of providing interfaces to users, we needed to develop a platform. Thus we chose web development for this. In this project, we created a web app using Python and it's web framework flask, which reduces development time and allows us to build faster and smarter.

For the Frontend, we have created 5 web pages which include:

1. **Index.html** - Main welcome page of our web app, allows navigation to other pages and displays the categories of available quotes.
2. **Services.html** - It can be considered as the second option in Navigation. It is an important page because a user can actually upload the image here and gets the extracted caption.
3. **About.html** - About Page is for users knowledge about website, developers etc

4. **Contact.html** - In case of any query, enquiry, report, appreciation users can communicate to the development team
5. **Single.html** - Last but not least, this page is not directly accessible. On the index page when a user wants the quotes of a particular category and thus clicks its card, he/she is directed to this page, where they get a number of related quotes of that single category.

5.11 API INTEGRATION

In order to provide a vast number of quotes and categories, we connected our app to API. In this project, we used GoodQuotes API. API Integration is done using Javascript.

5.12 CONNECTING MODEL WITH FRONTEND AND PYTHON CODE

The backend and frontend both work together to serve a single goal. It's pretty helpful to keep it in mind at all times. They are made, so a user can access them.

WORKING OF WEBSITE

The user points their browser to one of your website's URLs and waits for the browser to render the page. The user sees a useful and usable page. The user interacts with the page.

Thus till now, our website was working as static, it was just connected to API. In order to connect it to our working python code, we used flask i.e, micro-framework of python. In this, we created several routes using route decorator and thus it helped us in hosting it to a local server - `http://localhost:7000/`

5.13 PROVIDING USER WITH CAPTIONS ON THE PLATFORM

On reaching Upload Page when a user inputs an image, The `services()` function is decorated with `@app.route` so that it is invoked when the browser sends a POST request. Using the request module of Flask it creates an object of the file and saves it in local storage. At the same time, it sends that image to python code by calling its function which in response returns the output string, which is then displayed as a caption.

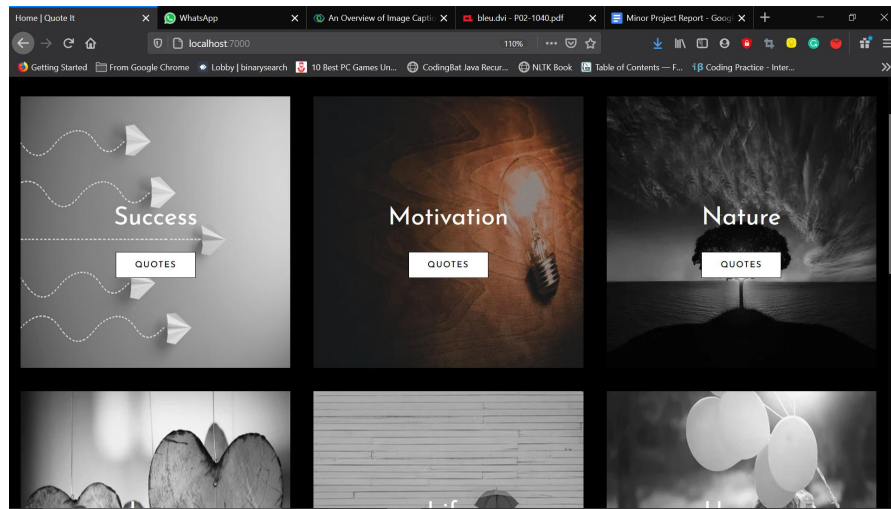


Figure 40: Layout of the Home page of the Web App

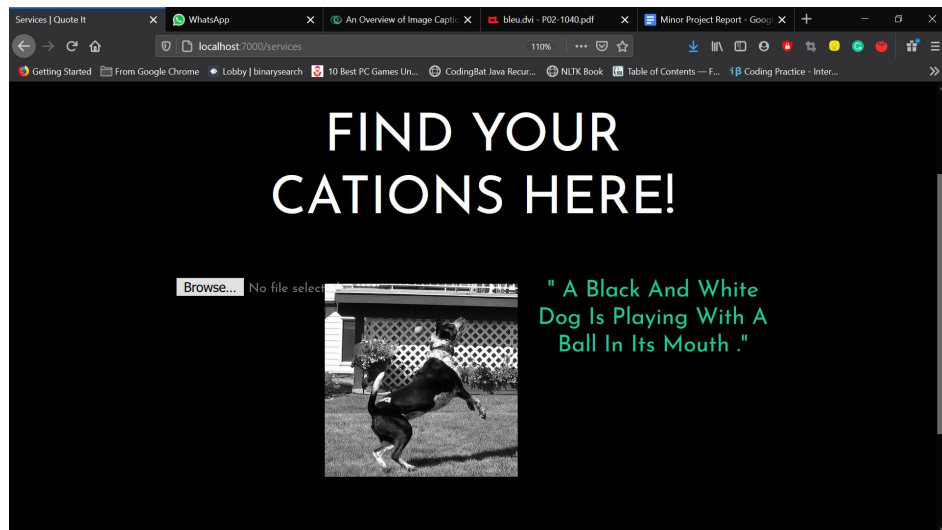


Figure 41: Sample results displayed at the Web Platform when using the model for prediction

Figure 40 shows the front page of the Web App as visible to the user. Here he/she can select the category of their choice to get the captions or they can select the Upload tab in the left top parallel bars. When the user will upload the image and press on the button. The model functions in the background and provide the caption which is shown on the Web App as shown in figure 41. Flask library is used to fetch the caption predicted by the model and display it on the Web Page.

CHAPTER - 6

CONCLUSIONS AND FUTURE SCOPE

6.1 CONCLUSIONS

Through the project, we were able to achieve an accuracy of 74.34 and a loss of 1.06 when training for 50 epochs. The model is able to predict the caption to a very close extent. Through this project, the user can obtain the caption of images from anywhere by just uploading the image within a few seconds. A batch of images can also be passed to obtain the captions for all the images, which can be used by the user for automating the task of manual captioning the images or classification of images by the use of captions of the images.

6.2 FUTURE SCOPE

There is a huge feature approach for this model some of which are enlisted below:

- This model is currently trained for a set of 1500 images from the desired set of 6000 images from the Flickr8k dataset, due to limitations of the hardware. Even then it has shown quite promising results for a large set of images. If this model can be made to train on 6000 images the model's accuracy will enhance a lot and the loss will also decrease. Thus we will be able to provide users with better results.
- Option for single and batch mode operation. This feature will provide the user with the option to choose the single and batch mode right from the platform only.
- Bunch caption suggestion - this feature will provide the user with a bunch of captions that can be used by the user.
- Text-to-speech conversion option - this feature will provide an option to generate the audio clip for the caption generated which can be used to help the visually impaired peoples.
- Video Frames Recognition - this feature will provide the user with the option to provide videos as input and obtain the important features like captions, or summary from it.

- An android app using this model can be a handy tool, which can be used by a huge number of people.
- Plugin feature - this feature will provide it with the functionality to be added with different image apps in the device and directly suggest them captions.

CHAPTER - 7

REFERENCES

- [1] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun - Deep Residual Learning for Image Recognition - Dec 10, 2015 - (arXiv:1512.03385v1)
- [2] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich - Going deeper with convolutions - Sep 17, 2014 - (arXiv:1409.4842v1)
- [3] Francois Chollet - Xception: Deep Learning with Depthwise Separable Convolutions - Apr 4, 2017 - (arXiv:1610.02357v3)
- [4] Priyanka Jadhav, Sayali Joag, Rohini Chaure, Sarika Koli - Automatic Caption Generation for News Images - ISSN:0975-9646
- [5] Vishwash Batra, Yulan He, George Vogiatzis - Neural Caption Generation for News Images
- [6] Kishor Prajapati, Shardul Wadekar, Bhushan Bobhate and Amruta Mhatre - Auto-Caption Generation for News Images - ISSN: 2278-0181
- [7] Mohmedhanif Nashipudi - WEB 2.0 AND FOLKSONOMY - ISSN: 2250-1142 - March2012
- [8] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun - Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification - Feb 6, 2015 - (arXiv:1502.01852v1) {Parametric Rectified Linear Unit (PReLU)}
- [9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun - Identity Mappings in Deep Residual Networks - Jul 25, 2016 - (arXiv:1603:05027v3) {2.3.1 and 2.3.2}
- [10] Diederik P. Kingma and Jimmy Lei Ba - ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION - Jun 30, 2017 - (arXiv:1412.6980v9)
- [11] Andrej Karpathy and Li Fei-Fei - Deep Visual-Semantic Alignments for Generating Image Descriptions

- [12] Yansong Feng - Automatic Caption Generation for News Images - 2011
- [13] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho and Yoshua Bengio - Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling - Dec 11, 2014 - (arXiv:1412.3555v1) {LSTM - GRU}
- [14] Marc Tanti, Albert Gatt and Kenneth P. Camilleri - What is the Role of Recurrent Neural Networks (RNNs) in an Image Caption Generator? - Aug 25, 2017 - (arXiv:1708.02043v2)
- [15] Jing Yang and Jun Wang - Tag clustering algorithm LMMSK: improved K-means algorithm based on latent semantic analysis - April 2017
- [16] Marc Tanti, Albert Gatt and Kenneth P. Camilleri - Where to put the Image in an Image Caption Generator - 12-Mar-2018 - (arXiv:1703.09137v2)
- [17] A. Farhadi, M. Hejrati, M. A. Sadeghi et al., “Every picture tells a story: generating sentences from images,” in *Computer Vision – ECCV 2010*, K. Daniilidis, P. Maragos, and N. Paragios, Eds., pp. 15–29, Springer, 2010.
- [18] S. M. Li, G. Kulkarni, T. L. Berg, A. C. Berg, and Y. J. Choi, “Composing simple image descriptions using web-scale n-grams,” in *Proceedings of the Fifteenth Conference on Computational Natural Language Learning. Association for Computational Linguistics*, pp. 220–228, Portland, Oregon, USA, 2011.
- [19] G. Kulkarni, V. Premraj, S. Dhar et al., “Baby talk: understanding and generating image descriptions,” in *CVPR means IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2891–2903, 2011.
- [20] Y. Gong, L. Wang, M. Hodosh, J. Hockenmaier, and S. Lazebnik, “Improving image sentence embeddings using large weakly annotated photo collections,” in *European Conference on Computer Vision*, pp. 529–545, Springer, 2014.
- [21] V. Ordonez, G. Kulkarni, and T. L. Berg, “Im2Text: Describing images using 1 million captioned photographs,” *Advances in Neural Information Processing Systems*, pp. 1143–1151, 2011.
- [22] C. Sun, C. Gan, and R. Nevatia, “Automatic concept discovery from parallel text and visual corpora,” in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 2596–2604, Santiago, Chile, 2015.

- [23] M. Hodosh, P. Young, and J. Hockenmaier, “Framing image description as a ranking task: Data, models and evaluation metrics,” *Journal of Artificial Intelligence Research*, vol. 47, pp. 853–899, 2013.
- [24] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, “Show and tell: a neural image caption generator,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3156–3164, Boston, MA, USA, 2015.
- [25] A. Karpathy and L. Fei-Fei, *Deep visual-semantic alignments for generating image descriptions*, Stanford University, 2017.
- [26] K. Xu, J. Ba, R. Kiros et al., “Show, attend and tell: neural image caption generation with visual attention,” in *International conference on machine learning*, pp. 2048–2057, Lille, France, 2015.
- [27] R. Kiros, R. Salakhutdinov, and R. S. Zemel, “Unifying visual-semantic embeddings with multimodal neural language models,” *Workshop on Neural Information Processing Systems (NIPS)*, 2014.
- [28] X. Yu, Y. Chu, F. Jiang, Y. Guo, and D. Gong, “SVMs Classification based two-side cross domain Collaborative Filtering by inferring intrinsic user and item features,” *Knowledge- Based Systems*, vol. 141, pp. 80–91, 2018.
- [29] X. Yu, F. Jiang, J. Du, and D. Gong, “A cross-domain collaborative filtering algorithm with expanding user and item features via the latent factor space of auxiliary domains,” *Pattern Recognition*, vol. 94, pp. 96–109, 2019.
- [30] <https://www.hindawi.com/journals/wcmc/2020/8909458/>

APPENDICES

APPENDIX 1

1.1 DETAILED TEST STRATEGY AND TEST CASES

For the testing process, we have selected the last 1000 images from the Flickr8k Dataset and by using the ‘random function’ of Numpy library we have fetched images from the Test Data Set and checked the captioning accuracy.

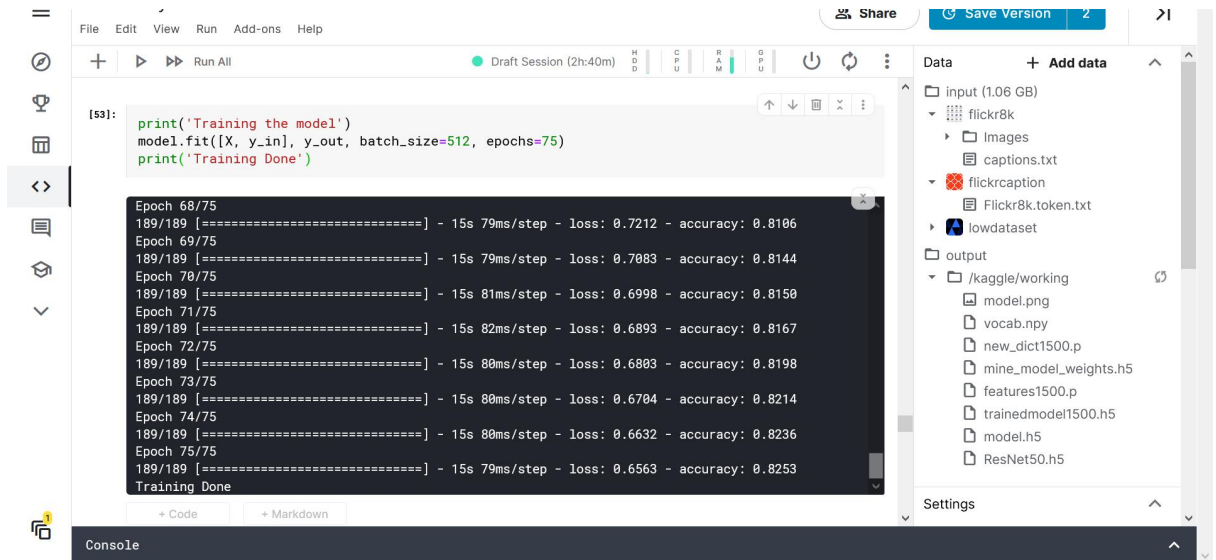


Figure A1: Sample image showing loss and accuracy obtained by the model

The last measured accuracy of the model at 50 epochs was 74.34 and the loss was 1.06. The model has the best accuracy on training it for 75 epochs is 82.53 and the loss of 0.65. We have used the inbuilt function of the model.train() to obtain the accuracy of the model. Figure A1 shows that model.fit() function’s inbuilt mechanism for detecting the loss and predicting the accuracy show us the loss and accuracy of the model.

APPENDIX 2

2.1 USER GUIDE

The user needs to follow the below-mentioned procedure for obtaining the captions for the images they provide.

Step 1: Visit to the webpage using _____

Step 2: Browse to upload section of the website

Step 3: Click on the browse button to select the image present in the system

Step 4: After selecting the image, the image will appear on the web page. Then press the Predict Caption button.

Step 5: Wait for a while, the model will provide you with the caption within a short interval

Step 6: Done. Now the user can copy the caption and use it at the place of their choice.

****EOF****