A Project Report
on

IMAGE CAPTION AND SPEECH GENERATION USING LSTM AND GTTS API

submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SICENCE AND ENGINEERING

by

17WH1A0549	**Ms. S.POOJA SAI SREE**
17WH1A0524	**Ms. D.MANISHA**
17WH1A0529	**Ms. S.SUSMITHA**

under the esteemed guidance of

Mr. K.BHARGAV RAM

Assistant Professor

Department of Computer Science and Engineering

BVRIT HYDERABAD

College of Engineering for Women
(NBA Accredited – EEE, ECE, CSE and IT)
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)

Bachupally, Hyderabad – 500090

April, 2021

DECLARATION

We hereby declare that the work presented in this project entitled **"IMAGE CAPTION AND SPEECH GENERATION USING LSTM AND GTTS API"** submitted towards completion of Project Work in IV year of B.Tech., CSE at 'BVRIT HYDERABAD College of Engineering For Women', Hyderabad is an authentic record of our original work carried out under the guidance of Mr. K.Bhargav ram, Assistant Professor, Department of CSE.

Sign. with date:

Ms. S.POOJA SAI SREE

(17WH1A0549)

Sign. with date:

Ms. D.MANISHA

(17WH1A0524)

Sign. with date:

Ms. S.SUSMITHA

(17WH1A0529)

BVRIT HYDERABAD
College of Engineering for Women
(NBA Accredited – EEE, ECE, CSE and IT)
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)

Bachupally, Hyderabad – 500090

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Project Work report on "**IMAGE CAPTION AND SPEECH GENERATION USING LSTM AND GTTS API**" is a bonafide work carried out by Ms. S.POOJA SAI SREE (17WH1A0549) ; Ms. D.MANISHA (17WH1A0524) ; Ms. S.SUSMITHA (17WH1A0529) in the partial fulfillment for the award of B.Tech. degree in **Computer Science and** Engineering**, BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad**, affiliated to Jawaharlal Nehru Technological University Hyderabad, Hyderabad under my guidance and supervision.

The results embodied in the project work have not been submitted to any other University or Institute for the award of any degree or diploma.

Head of the Department

Dr. K. Srinivasa Reddy

Professor and HoD,

Department of CSE

Guide

Mr. K.Bhargav ram

Assistant Professor

External Examiner

ACKNOWLEDGEMENT

We would like to express our sincere thanks to **Dr. K V N Sunitha, Principal**, **BVRIT HYDERABAD College of Engineering for Women**, for providing the working facilities in the college.

Our sincere thanks and gratitude to our **Dr. K. Srinivasa Reddy, Professor**, Department of CSE, **BVRIT HYDERABAD College of Engineering for Women** for all the timely support and valuable suggestions during the period of our project.

We are extremely thankful and indebted to our internal guide, **Mr. K.Bhargav ram, Assistant Professor**, Department of CSE**, BVRIT HYDERABAD College of Engineering for Women** for his constant guidance, encouragement and moral support throughout the project.

Finally, we would also like to thank our Project Coordinator, all the faculty and staff of **CSE** Department who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

Ms. S.POOJA SAI SREE

(17WH1A0549)

Ms. D.MANISHA

(17WH1A0524)

Ms. S.SUSMITHA

(17WH1A0529)

Contents

ABSTRACT

This project aims to develop Image caption and speech generator, a tool which generates captions or descriptions for an image according to the content observed. Along with description also generates audio/speech for the description obtained. Image caption and speech generator is trained on the Flickr 8k dataset. The dataset consists of images, where each of them is paired with five different captions which provide clear descriptions of the salient entities and events.

The training is done using machine learning models such as VGG16 model, Long Short-Term Memory model. The task is to generate relatable caption/description and audio for a given image.

When an image is provided as input, the trained model generates a description for it. Later using GTTS(Google Text To Speech) API audio/speech is obtained.

LIST OF FIGURES

1. INTRODUCTION

Caption generation is one of the challenges in rapid developments of machine learning. It includes generating descriptions from the content observed in the image. The traditional machine learning algorithms were not successful in doing this task. But the deep learning models showed greater impact compared to machine learning models. This is due to their property of capturing the connection present on the relevant image and their ability to generalize is much better than traditional methods.

1.1 Objectives

The main objective of this project is to create image caption and speech generator. When a user provides an image, the model processes the image and generates the description for the given image. Later audio/speech is also generated for the obtained speech. The model will be trained on Flickr 8k dataset using deep learning models such as VGG16, Long Short Term Memory for generating description. GTTS API is used for speech generation.

1.2 Methodology

To train the model both image and text data is required. Therefore the Flickr 8k dataset is used. The following section provides more details related to dataset, models used in the project.

1.2.1 Dataset

The dataset for the experiment is downloaded from Kaggle which contains collection of 8091 images in which the image data is segregated into 6091 Flickr8kTrainImages, 1000 Flickr8kDevImages and 1000 Flickr8kTestImages and the text data contains the five descriptions list for every image in the text format.

The Flickr8kTrainImages photo data is used to train the defined model, Flickr8kDevImages is used to check the performance of the trained model and Flickr8kTestImages photo data is used for the evaluation of the model.

1.2.2 Models:

1.2.2.1 VGG16:

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous model submitted to ILSVRC-2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another.

Features:

- It is also called the OxfordNet model, named after the Visual Geometry Group from Oxford.
- Number 16 refers that it has a total of **16 layers** that has some **weights**.
- It Only has Conv and pooling layers in it.
- always use a 3 x 3 Kernel for convolution.
- 2x2 size of the max pool.
- has a total of about 138 million parameters.
- Trained on ImageNet data
- It has an **accuracy** of 92.7%.
- it has one more version of it Vgg 19, a total of 19 layers with weights.

Architecture:

The input to the network is image of dimensions *(224, 224, 3)*. The first two layers have *64* channels of *3*3* filter size and same padding. Then after a max pool layer of stride *(2, 2)*, two layers which have convolution layers of 256 filter size and filter size *(3, 3)*. This followed by a max pooling layer of stride *(2, 2)* which is same as previous layer. Then there are *2* convolution layers of filter size *(3, 3)* and *256* filter. After that there are *2* sets of *3* convolution layer and a max pool layer. Each have *512* filters of *(3, 3)* size with same padding. This image is then passed to the stack of two convolution layers.

In these convolution and max pooling layers, the filters we use is of the size *3*3* instead of *11*11* in AlexNet and *7*7* in ZF-Net. In some of the layers, it also uses *1*1* pixel which is used to manipulate the number of input channels. There is a padding of *1-pixel* (same padding) done after each convolution layer to prevent the spatial feature of the image.

After the stack of convolution and max-pooling layer, we got a *(7, 7, 512)* feature map. We flatten this output to make it a *(1, 25088)* feature vector.

After this there are *3 fully* connected layer, the first layer takes input from the last feature vector and outputs a *(1, 4096)* vector, second layer also outputs a vector of size *(1, 4096)* but the third layer output a *1000* channels for *1000* classes of ILSVRC challenge, then after the output of 3rd fully connected layer is passed to softmax layer in order to normalize the classification vector.

After the output of classification vector top-5 categories for evaluation. All the hidden layers use ReLU as its activation function. ReLU is more computationally efficient because it results in faster learning and it also decreases the likelihood of vanishing gradient problem.

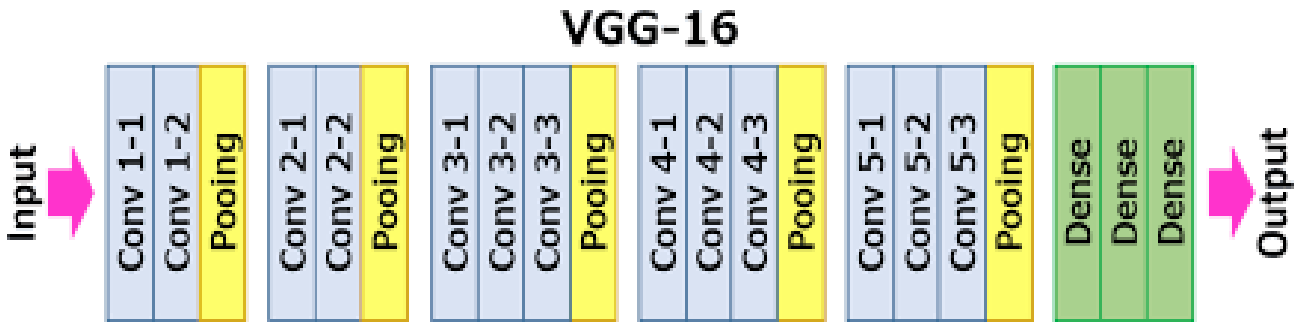

Fig 1.2.2.1: VGG-16 Architecture

No	Convolution	Output Dimension	Pooling	Output Dimension
layer 1&2	convolution layer of 64 channel of 3x3 kernel with padding 1, stride 1	224x224x64	Max pool stride=2, size 2x2	112x112x64
layer3&4	convolution layer of 128 channel of 3x3 kernel	112x112x128	Max pool stride=2, size 2x2	56x56x128
layer5,6,7	convolution layer of 256 channel of 3x3 kernel	56x56x256	Max pool stride=2, size 2x2	28x28x256
layer8,9,10	Convolution layer of 512 channel of 3x3 kernel	28x28x512	Max pool stride=2, size 2x2	14x14x512
layer11,12,13	Convolution layer of 512 channel of 3x3 kernel	14x14x512	Max pool stride=2, size 2x2	7x7x512

Fig 1.2.2.1.1: VGG-16 Layers Table

1.2.2.2 LSTM:

Long Short Term Memory Network is an advanced RNN, a sequential network, that allows information to persist. It is capable of handling the vanishing gradient problem faced by RNN. A recurrent neural network also known as RNN is used for persistent memory. RNNs work, they remember the previous information and use it for processing the current input. The shortcoming of RNN is, they can not remember

Long term dependencies due to vanishing gradient. LSTMs are explicitly designed to avoid long-term dependency problems.

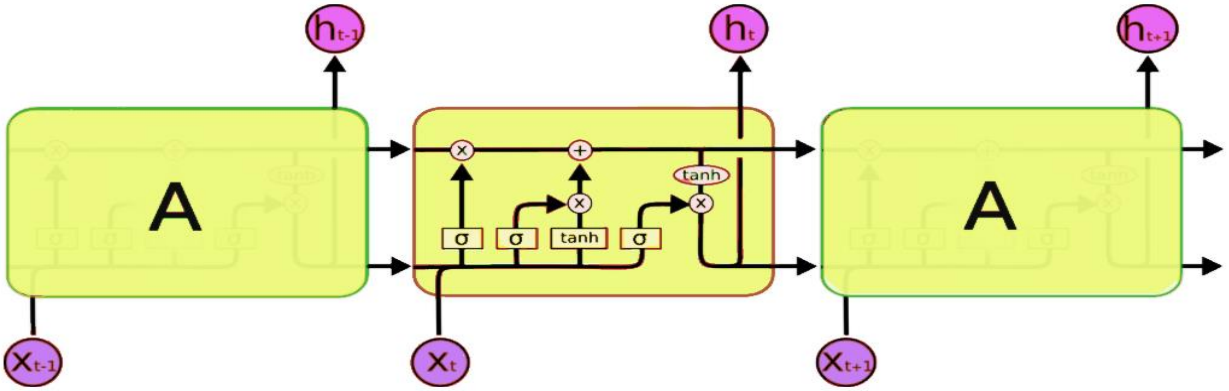

Fig 1.2.2.2: LSTM ARCHITECTURE

At a high-level LSTM works very much like an RNN cell. Here is the internal functioning of the LSTM network. A typical LSTM network is composed of different memory blocks called cells**.** There are two states that are being transferred to the next cell; the cell state and the hidden state. The memory blocks are responsible for remembering things and manipulations to this memory is done through three major mechanisms, called gates**.** The LSTM consists of three parts, as shown in the image below and each part performs an individual function. The first part is called Forget gate, the second part is known as the Input gate and the last one is the Output gate. The first part chooses whether the information coming from the previous timestamp is to be remembered or is irrelevant and can be forgotten. In the second part, the cell tries to learn new information from the input to this cell. At last, in the third part, the cell passes the updated information from the current timestamp to the next timestamp.

1.2.3 Required APIs

GTTS API:

Google Text-to-Speech is a Python library and CLI tool to interface with Google Translate text-to-speech API**.** Generally text-to-speech is the process of converting words into a vocal audio form. The program, tool, or software takes an input text from the user, and using methods of natural language processing understands the linguistics of the language being used, and performs logical inference on the text. This processed text is passed into the next block where digital signal processing is performed on the processed text. Using many algorithms and transformations this processed text is finally converted into a speech format. This entire process involves the synthesizing of speech. This seems like quite a complicated process, but using python and the gTTS module, this process can be simplified to just a few lines of code.The tool which converts the text entered, into audio which can be saved as a mp3 file. There are 3 ways to run this file.

1. Directly running the saved file from the folder.
2. Using the OS module
3. Using the playsound module

The gTTS module can be used extensively on languages such as English, French, German etc., as well. The speech can be delivered in any one of the two available audio speeds, fast or slow.This is extremely useful when there is a communication barrier and the user is unable to convey his messages to people.

2. THEORETICAL ANALYSIS OF THE PROPOSED PROJECT

2.1 Requirements Gathering

2.1.1 Software Requirements

Programming Language : Python 3.6

Graphical User Interface: TKinter

Dataset : Flickr 8k

Packages : Tensorflow, Keras, Numpy, NLTK, Pickle, GTTS

Framework : PyCharm

Tool : Google Colab

2.1.2 Hardware Requirements

Operating System: Windows 10

Processor : Intel Core i5-2450M

CPU Speed : 2.50 GHz

Memory : 3.7 GiB (RAM)

2.2 Technologies Description

Python

Python is an interpreted high-level general-purpose programming language, has design philosophy emphasizes code readability with it's notable use of significant indentation, the recommended indent size is four spaces. Developed by Guido van Rossum in the late 1980's and first released in 1991.

Python strives for a simpler, less-cluttered syntax and grammar. Python is used for developing web applications, data science and for rapid application development. Python is portable, extensible, embeddable and scalable.

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage

collector for memory management. It supports multiple programming paradigms, including structured, procedural, object-oriented and functional programming, and has a large and comprehensive standard library.

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

The simple syntax rules of python makes it easier to keep the application maintainable. It is compatible with major platforms and systems, simplifies complex software development and adopts test driven development. It supports GUI applications that can be created and ported to many system calls. It has a mature and supportive python community.

Tensorflow

TensorFlow is a free and open-source software library for machine learning, based on dataflow and differentiable programming across a range of tasks. It is a symbolic math library. It is also used for machine learning applications such as neural networks. It is used for both research and production at Google. TensorFlow accepts data in the form of multi-dimensional arrays of higher dimensions called tensors.

It was developed by the Google Brain team for internal Google use and was released under the Apache License 2.0 in 2015. Google released Colaboratory, which is a TensorFlow Jupyter notebook environment that requires no setup to use.

Keras:

Keras is an open-source software library that provides a Python interface for artificial neural networks. It is a deep learning API. It runs on top of the machine learning platform Tensorflow. It is user-friendly, modular, and extensible. It acts as an interface for the TensorFlow library.

The core data structures of Keras are layers and models. It contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation_functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code.

Numpy

Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows Numpy to seamlessly and speedily integrate with a wide variety of databases.

NLTK:

The Natural Language Toolkit is a suite of libraries and programs for symbolic and statistical natural language processing. It was developed by Steven Bird and Edward Loper in the Department of Computer and Information Science at the University of Pennsylvania. NLTK includes graphical demonstrations and sample data.

NLTK is a powerful Python package that provides a set of diverse natural language algorithms. It is free, opensource, easy to use, large community, and well documented.

NLTK is intended to support research and teaching in NLP or closely related areas, including empirical linguistics, cognitive science, artificial intelligence, information retrieval, and machine learning. NLTK has been used successfully as a teaching tool, as an individual study tool, and as a platform for prototyping and building research systems.

NLTK consists of the most common algorithms such as tokenizing, part-of-speech tagging, stemming, sentiment analysis, topic segmentation, and named entity recognition.

Pickle Module:

Python pickle module is used for serializing and de-serializing a python object structure.

It's the process of converting a Python object into a byte stream to store it in a file/database, maintain program state across sessions, or transport data over the network. The pickled byte stream can be used to re-create the original object hierarchy by unpickling the stream.

Advantages of using Pickle Module:

1. **Recursive objects (objects containing references to themselves):** Pickle keeps track of the objects it has already serialized, so later references to the same object won't be serialized again.
2. **Object sharing (references to the same object in different places):** This is similar to self-referencing objects; pickle stores the object once, and ensures that all other references point to the master copy. Shared objects remain shared, which can be very important for mutable objects.
3. **User-defined classes and their instances:** Marshal does not support these at all, but pickle can save and restore class instances transparently. The class definition must be importable and live in the same module as when the object was stored.

Pickle is very useful when working with machine learning algorithms, to save them to be able to make new predictions at a later time, without having to rewrite everything or train the model all over again.

It can also be used to send data over a Transmission Control Protocol (TCP) or socket connection, or to store python objects in a database.

What can be pickled?

The Objects are those which can be pickled with the following datatypes:

- Booleans
- Integers
- Floats
- Complex numbers
- Strings
- Tuples
- Lists
- Sets
- Dictionaries

gTTS:

There are several APIs available to convert text to speech in Python. One of such APIs is the Google Text to Speech API which is known as the gTTS API. It is a Python library and CLI tool to interface with Google Translate's text-to-speech API.

gTTS is a very easy to use tool. It converts the text entered into audio which can be saved as a mp3 file, writes spoken mp3 data to a file, a file-like object for further audio manipulation, or stdout. It features flexible pre-processing and tokenizing.

Pycharm:

PyCharm is an integrated development environment used especially for python programming language. It is developed by the Czech company JetBrains. It provides code analysis, a graphical debugger, an integrated unit tester.

PyCharm is cross-platform, with Windows, macOS and Linux versions and it's community edition is released under the Apache License.

Features:

- Coding assistance and analysis, with code completion, syntax and error highlighting, linter integration, and quick fixes.
- **Project and code navigation:** specialized project views, file structure views and quick jumping between files, classes, methods and usages.
- **Python refactoring:** includes rename, extract method, introduce variable, introduce constant, pull up, push down and others.
- **Support for web frameworks:** Django, web2py and Flask for professional edition only.
- Integrated Python debugger.
- Integrated unit testing, with line-by-line code coverage.
- Google App Engine Python development for professional edition only.
- Version control integration: unified user interface for Mercurial, Git, Subversion, Perforce and CVS with change lists and merge.
- Support for scientific tools like matplotlib, numpy and scipy for professional edition only.

Advantages and Disadvantages of using pycharm:

Advantages:

- A plethora of productive shortcuts
- Ability to view the entire Python source code with a single click
- Availability of an array of plugins
- Easy-to-use
- Excellent community support
- Facilitates faster code development
- More powerful, commercial version available
- Straightforward installation process

Disadvantages:

- Costly paid version.
- May pose issues when trying to fixing tools like venv.
- Not suitable for Python beginners.
- Resource-intensive application, i.e., requires plenty of memory and storage space.

Google Colaboratory:

Colaboratory, or “Colab” for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education.

Colab is a free Jupyter notebook environment that runs entirely in the cloud.

Google Colab allows to:

- Write and execute code in Python
- Document your code that supports mathematical equations
- Create/Upload/Share notebooks
- Import/Save notebooks from/to Google Drive
- Import/Publish notebooks from GitHub
- Import external datasets e.g. from Kaggle
- Integrate PyTorch, TensorFlow, Keras, OpenCV
- Free Cloud service with free GPU

Colab works with most major browsers, and is most thoroughly tested with the latest versions of Chrome, Firefox and Safari.

Tkinter:

Tkinter is the de facto way in Python to create Graphical User interfaces and is included in all standard Python Distributions.This Python framework provides an interface to the Tk toolkit and works as a thin object-oriented layer on top of Tk.The Tk toolkit is a cross-platform collection of ‘graphical control elements’, aka widgets, for building application interfaces.

This framework provides Python users with a simple way to create GUI elements using the widgets found in the Tk toolkit. Tk widgets can be used to construct buttons, menus, data fields, etc. in a Python application. Once created, these graphical elements can be associated with or interact with features, functionality, methods, data or even other widgets.

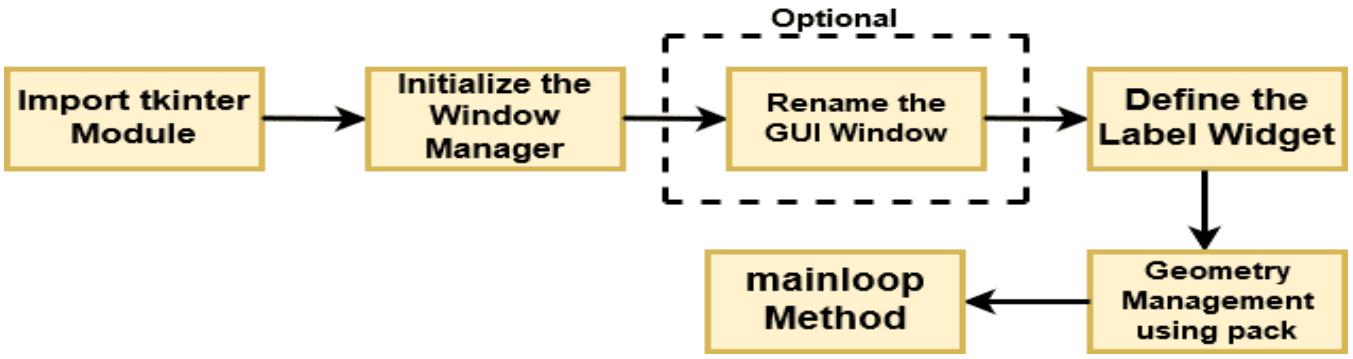

Fig 2.2: Flow diagram for a Basic Tkinter GUI

There are two main methods used which the user needs to remember while creating the Python application with GUI:

1. **Tk(screenName, baseName, className, useTk):** To create a main window, tkinter offers a method 'Tk(screenName, baseName, className, useTk)'. To change the name of the window, the className can be changed to the desired one.

   Code to create the main window of the application:

   ```
   m=tkinter.Tk()
   ```

2. **mainloop():** There is a method known by the name mainloop() is used when the application is ready to run. mainloop() is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

   Code:

   ```
   m.mainloop()
   ```

tkinter also offers access to the geometric configuration of the widgets which can organize the widgets in the parent windows. There are mainly three geometry manager classes class.

1) **pack() method**: It organizes the widgets in blocks before placing them in the parent widget.
2) **grid() method**: It organizes the widgets in grid before placing in the parent widget.
3) **place() method**: It organizes the widgets by placing them on specific positions directed by the programmer.

Working with widgets:

Widgets are the bread and butter of the Python GUI framework Tkinter. They are the elements through which users interact with the program. Each widget in Tkinter is defined by a class.

The following are some of the available widgets:

Widget class	Description
Label	A widget used to display text on the screen
Button	A button that can contain text and can perform an action when clicked
Entry	A text entry widget that allows only a single line of text
Text	A text entry widget that allows multiline text entry
Frame	A rectangular region used to group related widgets or provide padding between widgets

3. DESIGN

3.1 Architecture Diagram

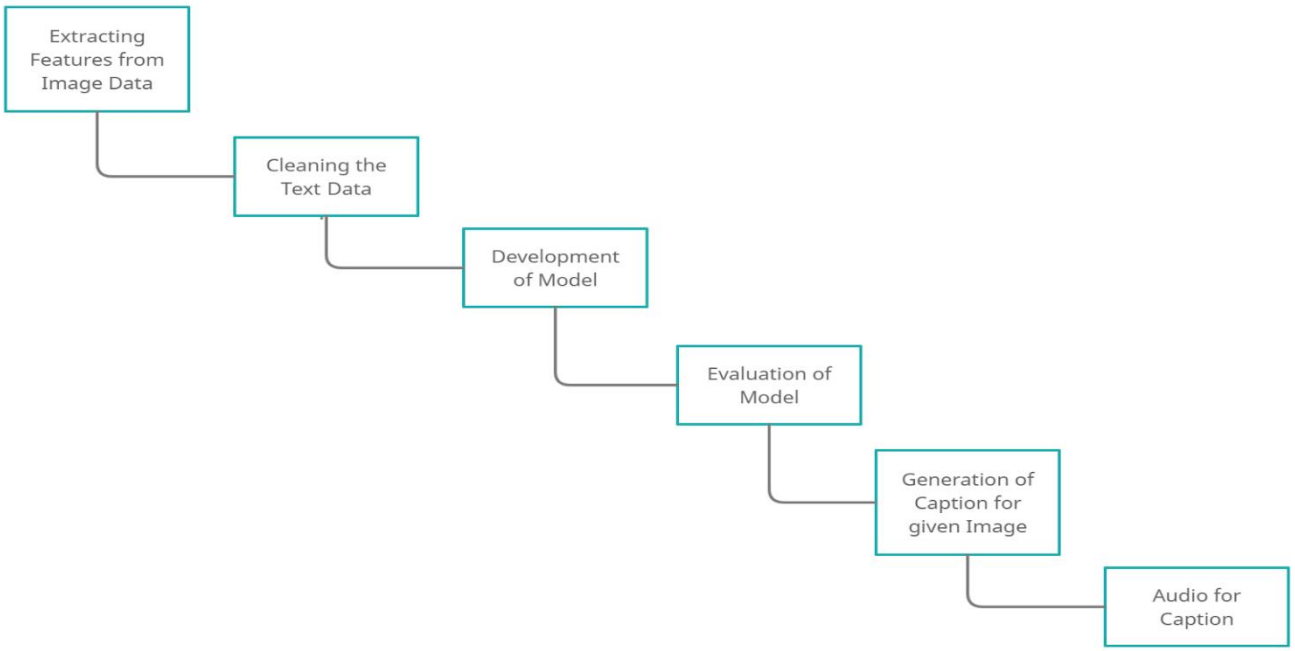

Fig 3.1: Architecture Diagram

Project Architecture contains the flow in which the project is developed. In this project there are six main steps as shown in above figure. Before the first step, the data should be loaded from the dataset.

Extracting Features from Photo Data:

The first step is extracting features from images. This is done using VGG16 model which is based on Convolutional Neural Network. The VGG model is loaded from Keras using the VGG class. The last layer from the loaded model will be removed, as we are interested in the internal representation of the photo right before a classification is made. These are the "features" that the model has extracted from the photo. Keras also provides tools for reshaping the loaded photo into the preferred size for the model (e.g. 3 channel 224 x 224 pixel image). Each photo in the dataset will be loaded, prepare it for VGG, and collect the predicted features from the VGG model. The image features are a 1-dimensional 4,096 element vector.

Cleaning the text data:

The dataset contains multiple descriptions for each photograph and the text of the descriptions requires some minimal cleaning. The descriptions are tokenized which will be easy to work with. The text will be cleaned in the following ways in order to reduce the size of the vocabulary of words:

- Convert all words to lowercase.
- Remove all punctuation.
- Remove all words that are one character or less in length (e.g. 'a').
- Remove all words with numbers in them.

Development of model:

The photo features and descriptions obtained from the above two steps are loaded. The descriptions are

Encoded using Tokenizer where each description is broke into several words and encoded them with unique integer. The model generated consists of three parts

- **Photo Feature Extractor**. It is a 16-layer VGG model pre-trained on the ImageNet dataset. The are pre-processed with the VGG model (without the output layer) and the extracted features predicted by this model is ued as input.
- **Sequence Processor**. It is a word embedding layer for handling the text input, followed by a Long Short-Term Memory (LSTM) recurrent neural network layer.
- **Decoder:** Both the feature extractor and sequence processor output a fixed-length vector. These are merged together and processed by a Dense layer to make a final prediction.

The Photo Feature Extractor model expects input photo features to be a vector of 4,096 elements. These are processed by a Dense layer to produce a 256 element representation of the photo. The Sequence Processor model expects input sequences with a pre-defined length (34 words) which are fed into an Embedding layer that uses a mask to ignore padded values. This is followed by an LSTM layer with 256 memory units. Both the input models produce a 256 element vector. Further, both input models use regularization in the form of 50% dropout. This is to reduce overfitting the training dataset.

The Decoder model merges the vectors from both input models using an addition operation. This is then fed to a Dense 256 neuron layer and then to a final output Dense layer that makes a softmax prediction over the entire output vocabulary for the next word in the sequence. Activation function used in hidden layers of the model is ReLU(Rectified Linear Unit). This is choosen based on architecture of hidden layers. ReLU do not have vanishing gradient problem while some other activation functions like sigmoid,tanh have. This is the reason ReLU is chosen over other functions. Hidden and Output layers have different activation functions which makes model to learn complex patterns. Output layer activation function is softmax, which is used in case of multiclass classification. Loss function used is categorical cross entropy, which is default for multiclass classification. This loss function works well with softmax activation function. After the development of model, it is trained using the training dataset.

The model is run over 20 epochs. Each epoch creates a model. The model which has low loss value is considered and sent for evaluation.

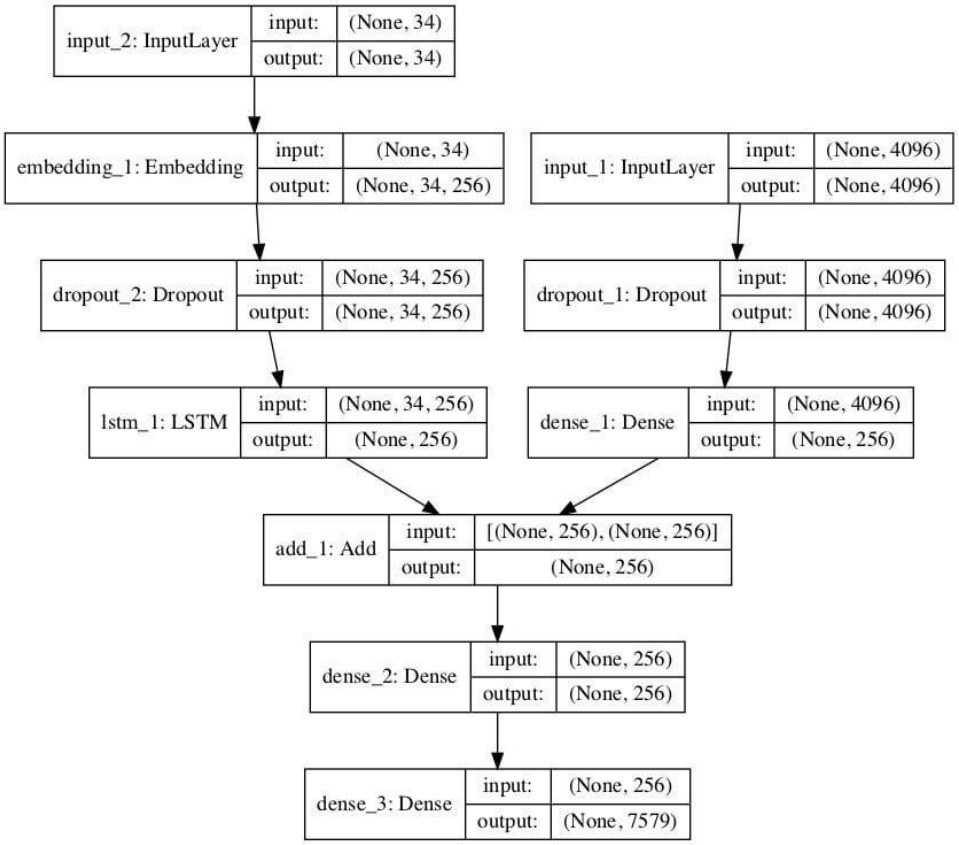

Fig 3.1.1: Plot of the Image caption generation model

Evaluation of model:

The trained model is saved and loaded to be ready for the evaluation. The test dataset is used to extract the required photo features from image dataset and vocabulary from the text data that consists of list of descriptions for every image in the test data. These are passed as input parameters for the evaluation where the sequences of text descriptions are generated for all the test data images, these predicted and actual descriptions are collected and evaluated collectively using the corpus BLEU score that summarizes how close the generated text is to the expected text.

BLEU Score:

BLEU for short Bilingual Evaluation Understudy is a measurement of the differences between an automatic translation and one or more human-created reference translations of the same source sentence.

The BLEU algorithm compares consecutive phrases of the automatic translation with the consecutive phrases it finds in the reference translation, and counts the number of matches, in a weighted fashion. These matches are position independent. A higher match degree indicates a higher degree of similarity with the reference translation, and higher score.

BLEU Score	Interpretation
< 10	Almost useless
10 - 19	Hard to get the gist
20 - 29	The gist is clear, but has significant grammatical errors
30 - 40	Understandable to good translations
40 - 50	High quality translations
50 - 60	Very high quality, adequate, and fluent translations
> 60	Quality often better than human

Fig 3.1.2: Bleu Score interpretation table

The bleu score achieved after implementing the function corpus_bleu() that compares the candidate and reference translations on given weights is 52.7 which ranges between 50 – 60 i.e, the generated sentences

by the trained model are very high quality, adequate, and fluent translations. Now the model can be used to implement prediction of the image caption for the given image.

Generation of caption for given Image:

In this step, an image is given as input by the user. So, when the user gives the input, VGG16 extracts features from the given image. Along with features of given image, the tokenized descriptions are passed into the model for prediction. The model gives the description for the given image.

Audio for caption:

The generated caption is converted into speech using gTTS API in english language. It can be also stored in mp3 file.

3.2 UML Diagrams

3.2.1 Use Case Diagram

A use case diagram is a way to summarize details of a system and the users within that system. It is generally shown as a graphic depiction of interactions among different elements in a system. Use case diagrams will specify the events in a system and how those events flow, however, use case diagram does not describe how those events are implemented. In the below use case diagram, firstly, modules are imported. Dataset is loaded. Next step is feature extraction using VGG16. Then the model build using different layers such as embedding, dense, LSTM. The next step is training the model using training dataset. After the training the model, the model is evaluated using Bleu score. Then the model is sent for prediction of description for the image given by user. Then speech is generated for the description using GTTS API.

.

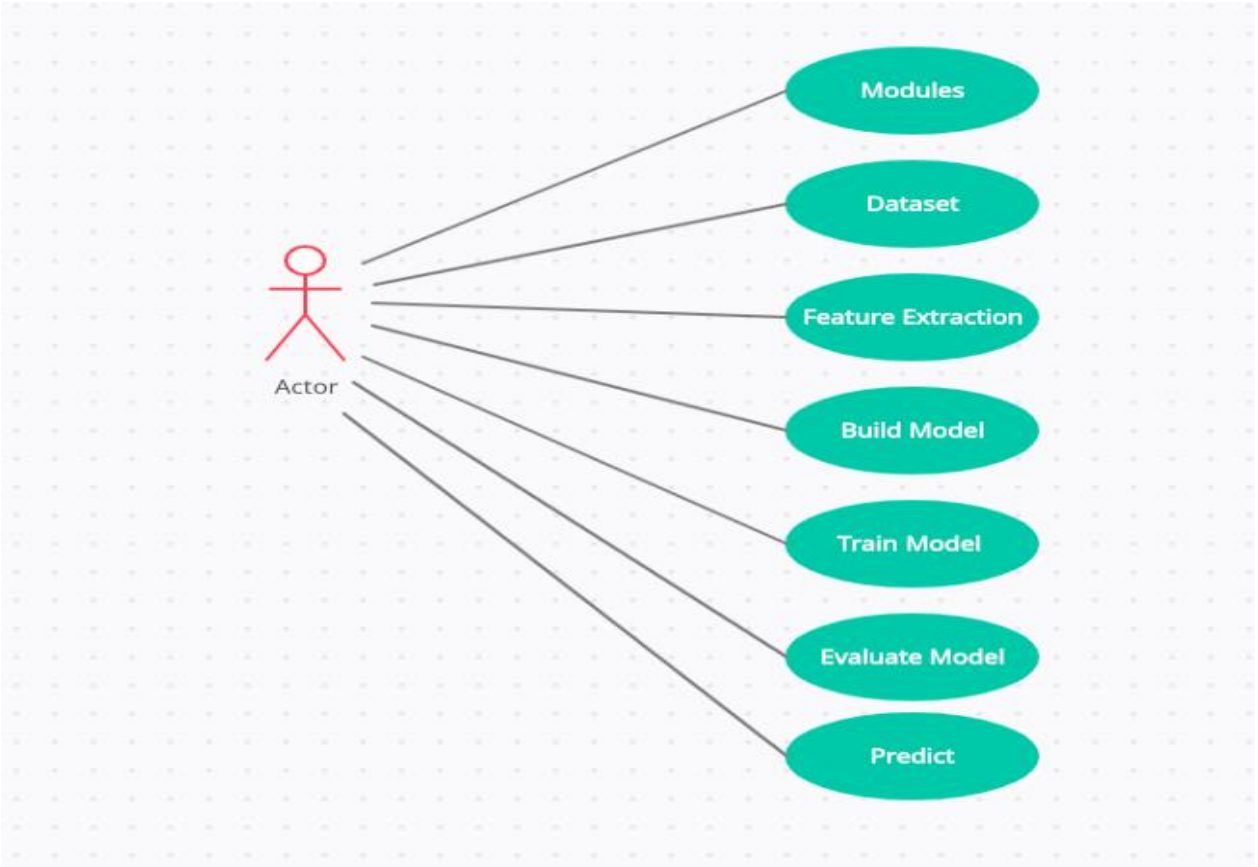

Fig 3.2.1: Use Case Diagram

3.2.2 Sequence Diagram

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

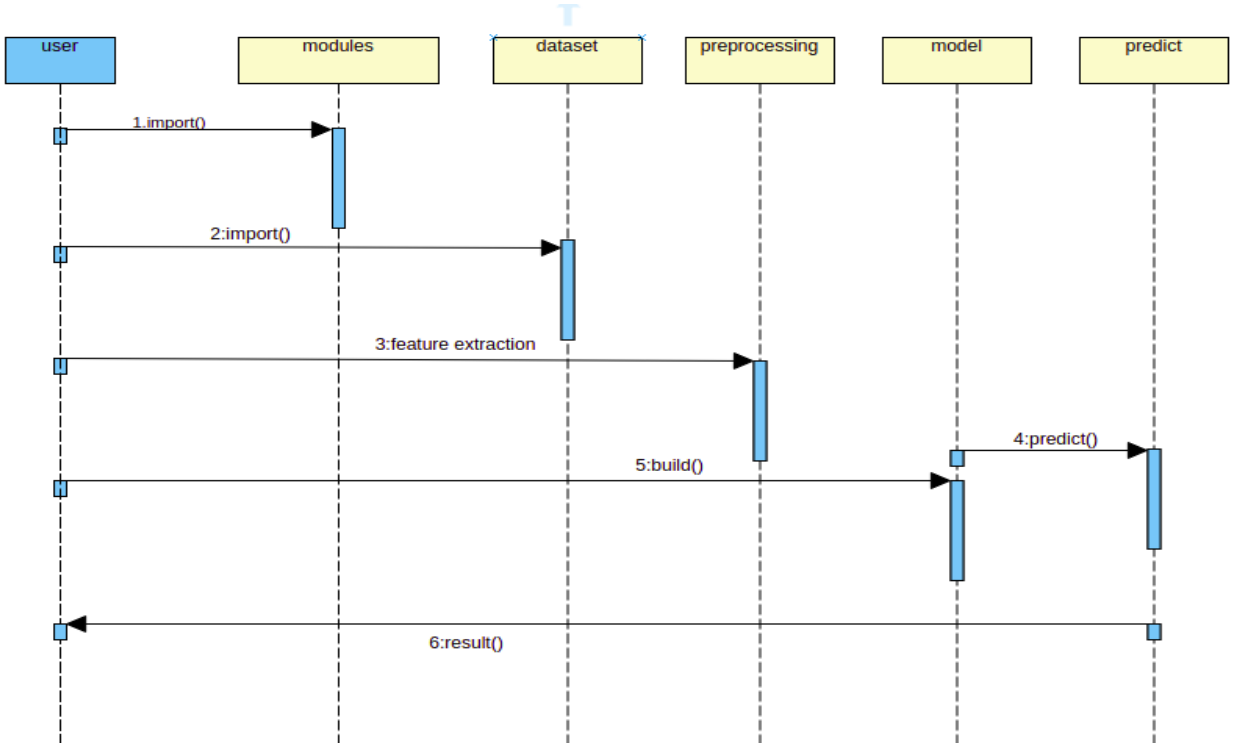

Fig 3.2.2: Sequence Diagram

3.2.3 Activity Diagram

An activity diagram visually presents a series of actions or flow of control in a system similar to a flowchart or a data flow diagram. Activity diagrams are often used in business process modeling. They can also describe the steps in a use case diagram. Activities modeled can be sequential and concurrent. In both cases an activity diagram will have a beginning (an initial state) and an end (a final state).

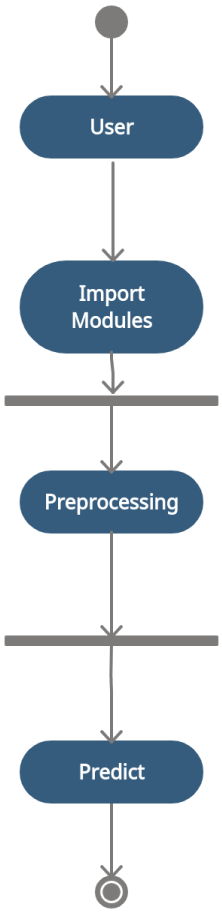

Fig 3.2.3: Activity Diagram

3.2.4 Class Diagram

The class diagram depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them. A class consists of its objects, and also it may inherit from other classes. A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.

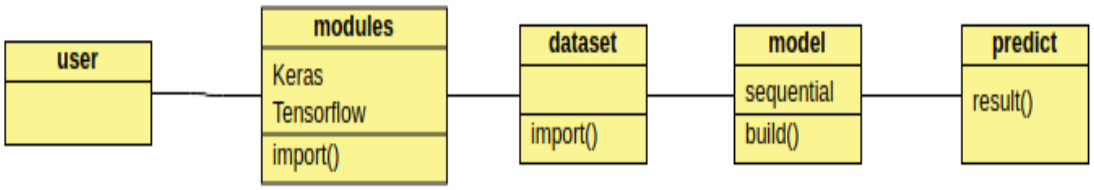

Fig 3.2.4: Class Diagram

4. IMPLEMENTATION

4.1 Coding

Image_caption_generator.ipynb

```
from keras.applications.vgg16 import VGG16
from keras.models import Model
from os import listdir
from pickle import dump
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input

#Preparing the Photo data:

photo_data = '/content/drive/MyDrive/Major project/Flicker8k_Dataset'
vgg_model = VGG16()
vgg_model = Model(inputs=vgg_model.inputs, outputs=vgg_model.layers[-2].output) image_features = dict()
count = 0
for pic_name in listdir(photo_data):
    file_path = photo_data + '/' + pic_name
    picture = load_img(file_path, target_size=(224, 224))
    picture = img_to_array(picture)
    count = count + 1
    print(count)
    picture = picture.reshape((1, picture.shape[0], picture.shape[1], picture.shape[2]))
    picture = preprocess_input(picture)
```

```
    features = vgg_model.predict(picture, verbose=0)
    pictureId = pic_name.split('.')[0]
    print('>%s' % pic_name)
    image_features[pictureId] = features
dump(features, open('/content/drive/MyDrive/Major project/text/features.pkl', 'wb'))

# Reading text from file:

text_file = '/content/drive/MyDrive/Major_project_dataset/Flickr8k_text/ Flickr8k.token.txt'
text_data= open(text_file, 'r')
text = text_data.read()
text_data.close()

# Mapping the descriptions to each Image:

descriptions = dict()
for line in text.split('\n'):
  words = line.split()
  if len(line) < 2:
    continue
  pictureId, pictureDes = words[0], words[1:]
  pictureId = pictureId.split('.')[0]
  pictureDes = ' '.join(pictureDes)
  if pictureId not in descriptions:
    descriptions[pictureId] = list()
  descriptions[pictureId].append(pictureDes)

# Cleaning the text data:
```

```
import string
for pic_id, pic_des in descriptions.items():
  for i in range(len(pic_des)):
    des = pic_des[i]
    des = des.split()
    #turning each description into lower case
    des = [word.lower() for word in des]
    #removing punctuations in each description
    s = " ".join(des)
    table = s.maketrans("",string.punctuation)
    des = [w.translate(table) for w in des]
    #removing words whose length is less than 1
    des = [word for word in des if len(word)>1]
    #removing alphanumeric words
    des = [word for word in des if word.isalpha()]
    pic_des[i] =  ' '.join(des)

# Saving the cleaned text data into a new text file:

all_desc = list()
for pic_id, pic_des in descriptions.items():
  for des in pic_des:
    all_desc.append(pic_id + ' ' + des)
cleaned_data = '\n'.join(all_desc)
fptr = open('/content/drive/MyDrive/project files/descriptions.txt', 'w')
fptr.write(cleaned_data)
fptr.close()

# Loading Train Dataset:
```

```
from numpy import array
from pickle import load
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.utils import plot_model
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Embedding
from keras.layers import Dropout
from keras.layers.merge import add
from keras.callbacks import ModelCheckpoint

train_set = '/content/drive/MyDrive/Major_project_dataset/Flickr8k_text/Flickr_8k.trainImages.txt'
fptr = open(train_set, 'r')
id_data = fptr.read()
fptr.close()
id_list = list()
for line in id_data.split('\n'):
if len(line) < 1:
continue
photo_id = line.split('.')[0]
id_list.append(photo_id)
id_list_train = set(id_list)

# Extracting and Storing Descriptions from train Images:
```

```
desc_path = '/content/drive/MyDrive/project files/descriptions.txt'
f = open(desc_path, 'r')
desc_data = f.read()
f.close()
train_desc = dict()
for line in desc_data.split('\n'):
tokens = line.split()
id, desc = tokens[0], tokens[1:]
if id in id_list_train:
if id not in train_desc:
train_desc[id] = list()
final_desc = 'startseq ' + ' '.join(desc) + ' endseq'
train_desc[id].append(final_desc)

# Extracting and Storing photo features from train images:

photo_path = '/content/drive/MyDrive/project files/features.pkl'
all_features = load(open(photo_path, 'rb'))
train_features = {k: all_features[k] for k in id_list_train}

# Preparing tokenizer:

desc_list = list()
for key in train_desc.keys():
[desc_list.append(d) for d in train_desc[key]]
tokenizer = Tokenizer()
tokenizer.fit_on_texts(desc_list)
dump(tokenizer, open('tokenizer.pkl', 'wb'))
```

```
vocabulary_size = len(tokenizer.word_index) + 1

# Finding maximum sequence length of descriptions:

max_length = max(len(d.split()) for d in desc_list)

# Defining the Model:

feature_input = Input(shape=(4096,))
feature_layer1 = Dropout(0.5)(feature_input)
feature_layer2 = Dense(256, activation='relu')(feature_layer1)
desc_input = Input(shape=(max_length,))
desc_layer1 = Embedding(vocabulary_size, 256, mask_zero=True)(desc_input)
desc_layer2 = Dropout(0.5)(desc_layer1)
desc_layer3 = LSTM(256)(desc_layer2)
decoder_layer1 = add([feature_layer2, desc_layer3])
decoder_layer2 = Dense(256, activation='relu')(decoder_layer1)
output_layer = Dense(vocabulary_size, activation='softmax')(decoder_layer2)
model = Model(inputs=[feature_input, desc_input], outputs=output_layer)
model.compile(loss='categorical_crossentropy', optimizer='adam')
model.summary()
plot_model(model, to_file='/content/drive/MyDrive/My_models/model.png', show_shapes=True)

# Development of the Model:
def sequence_generator(tokenizer, max_length, desc_list, photo, vocabulary_size):
X1, X2, y = list(), list(), list()
for desc in desc_list:
sequences = tokenizer.texts_to_sequences([desc])[0]
for i in range(1, len(sequences)):
```

```
input, output = sequences[:i], sequences[i]
input = pad_sequences([input], maxlen=max_length)[0]
output = to_categorical([output], num_classes=vocabulary_size)[0]
X1.append(photo)
X2.append(input)
y.append(output)
return array(X1), array(X2), array(y)

def data_generator(train_desc, train_features, tokenizer, max_length, vocab_size):
while 1:
for key, desc_list in train_desc.items():
photo = train_features[key][0]
image, desc_seq, output = sequence_generator(tokenizer, max_length, desc_list, photo, vocab_size)
yield [image, desc_seq], output

epochs = 20
steps = len(train_desc)
for i in range(epochs):
sequence = data_generator(train_desc, train_features, tokenizer, max_length, vocabalury_size)
model.fit_generator(sequence, epochs=1, steps_per_epoch=steps, verbose=1)
model.save('/content/drive/MyDrive/Image/new_models' + 'model_' + str(i) + '.h5')



# Loading Test Dataset:

from numpy import argmax
from pickle import load
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```

```
from keras.models import load_model
from nltk.translate.bleu_score import corpus_bleu

test_set = '/content/drive/MyDrive/Major_project_dataset/Flickr8k_text/Flickr_8k.testImages.txt'
f = open(test_set, 'r')
id_data = f.read()
f.close()
id_list = list()
for line in id_data.split('\n'):
if len(line) < 1:
continue
photo_id = line.split('.')[0]
id_list.append(photo_id)
id_list_test = set(id_list)

# Extracting and Storing Descriptions from Test images:

desc_path = '/content/drive/MyDrive/project files/descriptions.txt'
id_list = id_list_test
f = open(desc_path, 'r')
id_data = f.read()
f.close()
descriptions = dict()
 for line in id_data.split('\n'):
tokens = line.split()
img_id, img_desc = tokens[0], tokens[1:]
if img_id in id_list:
if img_id not in descriptions:
descriptions[img_id] = list()
```

```
desc = 'startseq ' + ' '.join(img_desc) + ' endseq'
descriptions[img_id].append(desc)
test_descriptions = descriptions

# Extracting and storing Photo features from Test images:

photo_path = '/content/drive/MyDrive/project files/features.pkl'
id_list = id_list_test
all_features = load(open(photo_path, 'rb'))
features = {k: all_features[k] for k in id_list}
test_features = features

# Evaluation of the Model:

model_path = '/content/drive/MyDrive/My_models/model_6.h5'
model = load_model(model_path)
def word_for_imgid(integer, tokenizer):
for word, index in tokenizer.word_index.items():
if index == integer:
return word
return None

def description_generator(model, tokenizer, photo, max_length):
in_text = 'startseq'
for i in range(max_length):
seq = tokenizer.texts_to_sequences([in_text])[0]
seq = pad_sequences([seq], maxlen=max_length)
y = model.predict([photo,seq], verbose=0)
y = argmax(y)
```

```
word = word_for_imgid(y, tokenizer)
if word is None:
break
in_text += ' ' + word
if word == 'endseq':
break
return in_text


reference_translation, candidate_translation = list(), list()
for img_id, desc_list in test_descriptions.items():
y = description_generator(model, tokenizer, test_features[img_id], max_length)
references = [d.split() for d in desc_list]
reference_translation.append(references)
candidate_translation.append(y.split())
print('BLEU-Score: %f' % (corpus_bleu(reference_translation, candidate_translation, weights=(1.0, 0, 0, 0))*100))
```

Image_caption_prediction.py

```
#Prediction:


from tkinter import *
from tkinter import filedialog
import os
import tkinter as tk
from PIL import Image, ImageTk


from gtts import gTTS


root = Tk()
```

```
root.title("Image Caption Generator")
root.geometry("700x700")
root.configure(bg='azure')

global res

l = Label(root)
def showimage():
fln = filedialog.askopenfilename(initialdir=os.getcwd(), title="select Image File", filetypes=(("JPG File", "*.jpg"), ("PNG file", "*.png"), ("All Files", "*.*")))
showimage.img_path = fln
img = Image.open(fln)
img.thumbnail((550,550))
img = ImageTk.PhotoImage(img)
lbl.configure(image=img, pady=35, bd="25", bg = "azure",relief="solid")
lbl.image = img
photo = extract_features(showimage.img_path)
description = generate_desc(model, tokenizer, photo, max_length)
sublist = ["startseq", "endseq"]
for sub in sublist:
description = description.replace(sub,' ')
res = " ".join(description.split())
l.config(text=res,font=("Georgia", 14, "bold"),padx=10,pady=25,bg="azure")
showimage.f = res
l.pack()


def play():
    tts = gTTS(text=showimage.f, lang='en')
```

```
    tts.save('myfile.mp3')
    os.system("mpg321 myfile.mp3")


frm = Frame(root)
frm.pack(side=BOTTOM, padx=15, pady=15)
frm.config(bg="azure")


lbl = Label(root)
lbl.pack()

def extract_features(filename):

model = VGG16()
model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
image = load_img(filename, target_size=(224, 224))
image = img_to_array(image)
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
image = preprocess_input(image)
feature = model.predict(image, verbose=0)
return feature

def word_for_id(integer, tokenizer):
for word, index in tokenizer.word_index.items():
if index == integer:
return word
return None
```

```
def generate_desc(model, tokenizer, photo, max_length):
in_text = 'startseq'
for i in range(max_length):
sequence = tokenizer.texts_to_sequences([in_text])[0]
sequence = pad_sequences([sequence], maxlen=max_length)
yhat = model.predict([photo,sequence], verbose=0)
yhat = argmax(yhat)
word = word_for_id(yhat, tokenizer)
if word is None:
break
in_text += ' ' + word
if word == 'endseq':
break
return in_text

tokenizer = load(open('/home/user/Downloads/tokenizer.pkl', 'rb'))

max_length = 34

model = load_model('/home/user/Downloads/model_6.h5')
btn = Button(frm, text="Browse Image", font="bold", command=showimage, bg="white")
btn.pack(side=tk.LEFT)

btn3 = Button(frm, text="play", font="bold", bg="white", command=play)
btn3.pack(side=tk.LEFT, padx=20)

btn2 = Button(frm, text="Exit", font="bold", bg="white", command=lambda: exit())
btn2.pack(side=tk.LEFT, padx=10)
 root.mainloop()
```

4.2 TEST CASES

A test case is a set of actions executed to verify a particular feature or functionality of your software application. A Test Case contains test steps, test data, precondition, postcondition developed for specific test scenario to verify any requirement. The test case includes specific variables or conditions, using which a testing engineer can compare expected and actual results to determine whether a software product is functioning as per the requirements of the customer.

TestCaseID	TestScenario	Expected	Actual	Pass
TC01	Check whether window is opening	Window should be opened	As Expected	Pass
TC02	Check whether browse button is working	Browse button should be worked	As Expected	Pass
TC03	Check whether image is getting browsed from images	Image should be browsed	As Expected	Pass
TC04	Check whether image is uploaded from the images	Image should be uploaded	As Expected	Pass
TC05	Check whether image is being displayed onto the window	Image should be displayed	As Expected	Pass
TC06	Check whether the description is displayed	Description should be displayed	As Expected	Pass
TC07	Check whether play button is working	Play button should be worked	As Expected	Pass
TC08	Check whether description is converted to speech	Description should be converted to speech	As Expected	Pass
TC09	Check whether getting the correct result	Should get correct result	As Expected	Pass
TC10	Check whether exit button is working	Exit button should be worked	As Expected	Pass

Fig 4.2: Test cases

4.3 MODEL TRAINING AND EVALUATION SCREENSHOTS

```
Dataset: 6000
Descriptions: train=6000
Photos: train=6000
Vocabulary Size: 7579
Description Length: 34
Model: "model_1"
```

Layer (type)	Output Shape	Param #	Connected to
input_4 (InputLayer)	[(None, 34)]	0	
input_3 (InputLayer)	[(None, 4096)]	0	
embedding_1 (Embedding)	(None, 34, 256)	1940224	input_4[0][0]
dropout_2 (Dropout)	(None, 4096)	0	input_3[0][0]
dropout_3 (Dropout)	(None, 34, 256)	0	embedding_1[0][0]
dense_3 (Dense)	(None, 256)	1048832	dropout_2[0][0]
lstm_1 (LSTM)	(None, 256)	525312	dropout_3[0][0]
add_1 (Add)	(None, 256)	0	dense_3[0][0] lstm_1[0][0]
dense_4 (Dense)	(None, 256)	65792	add_1[0][0]
dense_5 (Dense)	(None, 7579)	1947803	dense_4[0][0]

```
Total params: 5,527,963
Trainable params: 5,527,963
Non-trainable params: 0
_____
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1844: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
6000/6000 [==============================] - 1942s 323ms/step - loss: 5.1216
6000/6000 [==============================] - 1948s 325ms/step - loss: 3.8646
6000/6000 [==============================] - 1967s 328ms/step - loss: 3.6116
6000/6000 [==============================] - 1973s 329ms/step - loss: 3.4508
6000/6000 [==============================] - 1999s 333ms/step - loss: 3.3597
6000/6000 [==============================] - 1906s 318ms/step - loss: 3.2793
6000/6000 [==============================] - 2000s 333ms/step - loss: 3.2318
6000/6000 [==============================] - 1978s 330ms/step - loss: 3.1860
6000/6000 [==============================] - 1770s 295ms/step - loss: 3.1500
6000/6000 [==============================] - 1784s 297ms/step - loss: 3.1224
6000/6000 [==============================] - 1783s 297ms/step - loss: 3.0997
6000/6000 [==============================] - 1768s 295ms/step - loss: 3.0833
6000/6000 [==============================] - 1746s 291ms/step - loss: 3.0700
6000/6000 [==============================] - 1744s 291ms/step - loss: 3.0620
6000/6000 [==============================] - 1752s 292ms/step - loss: 3.0526
6000/6000 [==============================] - 1755s 292ms/step - loss: 3.0448
6000/6000 [==============================] - 1754s 292ms/step - loss: 3.0355
6000/6000 [==============================] - 1769s 295ms/step - loss: 3.0341
6000/6000 [==============================] - 1787s 298ms/step - loss: 3.0314
6000/6000 [==============================] - 1780s 297ms/step - loss: 3.0286
```

Fig 4.3: Training dataset

```
reference_translation, candidate_translation = list(), list()
for img_id, desc_list in test_descriptions.items():
  y = description_generator(model, tokenizer, test_features[img_id], max_length)
  references = [d.split() for d in desc_list]
  reference_translation.append(references)
  candidate_translation.append(y.split())
print('BLEU-Score: %f' % (corpus_bleu(reference_translation, candidate_translation, weights=(1.0, 0, 0, 0))*100))
```

```
BLEU-Score: 52.759237
```

Fig 4.3.1: Evaluating Model

4.4 INPUT AND OUTPUT SCREENSHOTS

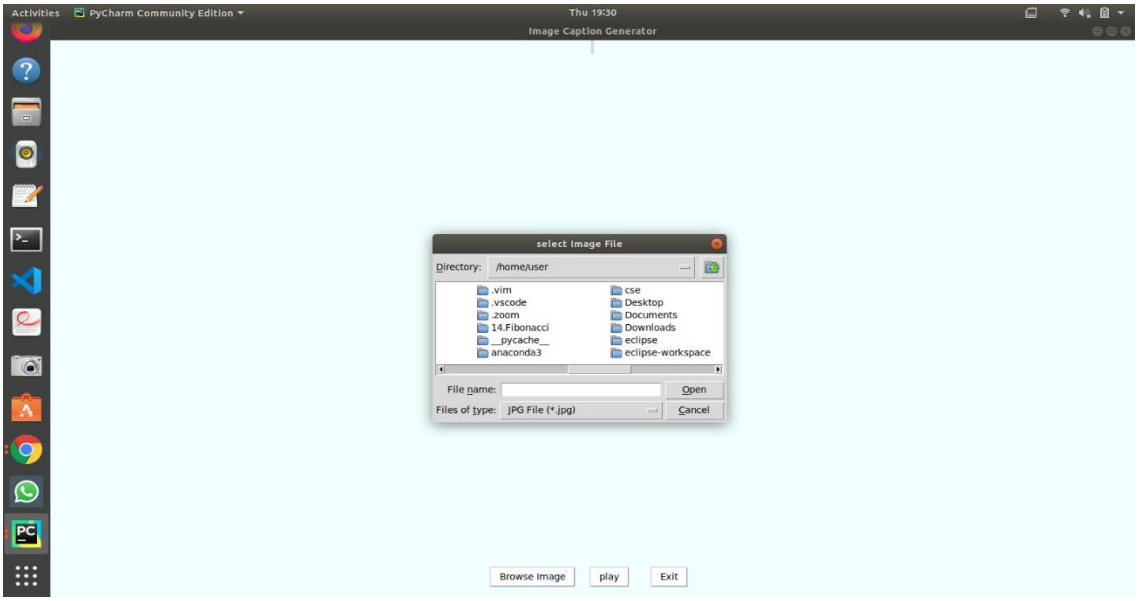

Fig 4.4: Image Browsing

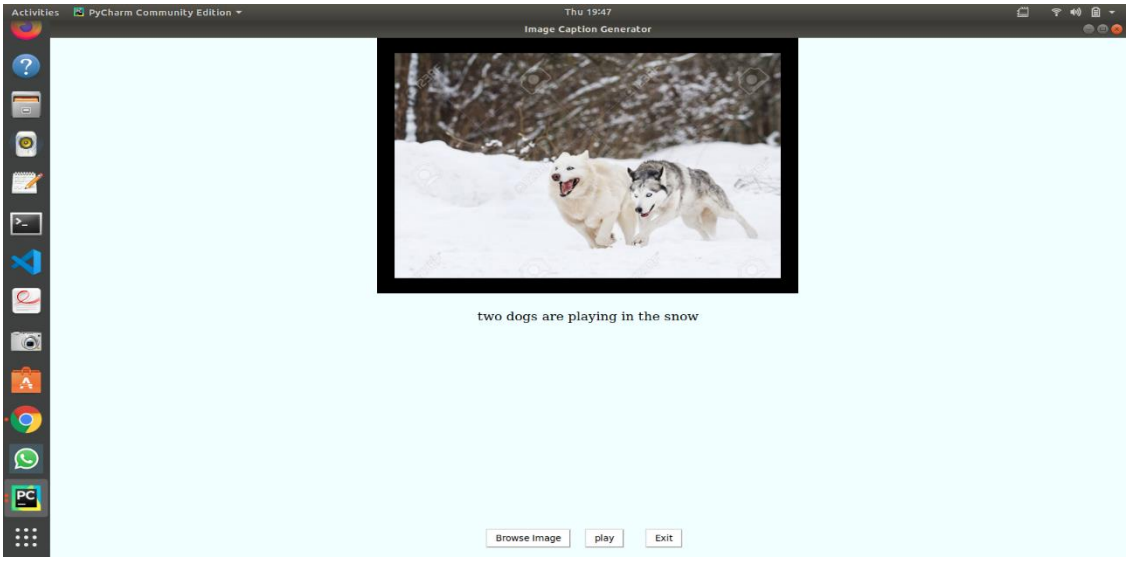

Fig 4.4.1: Description Generation

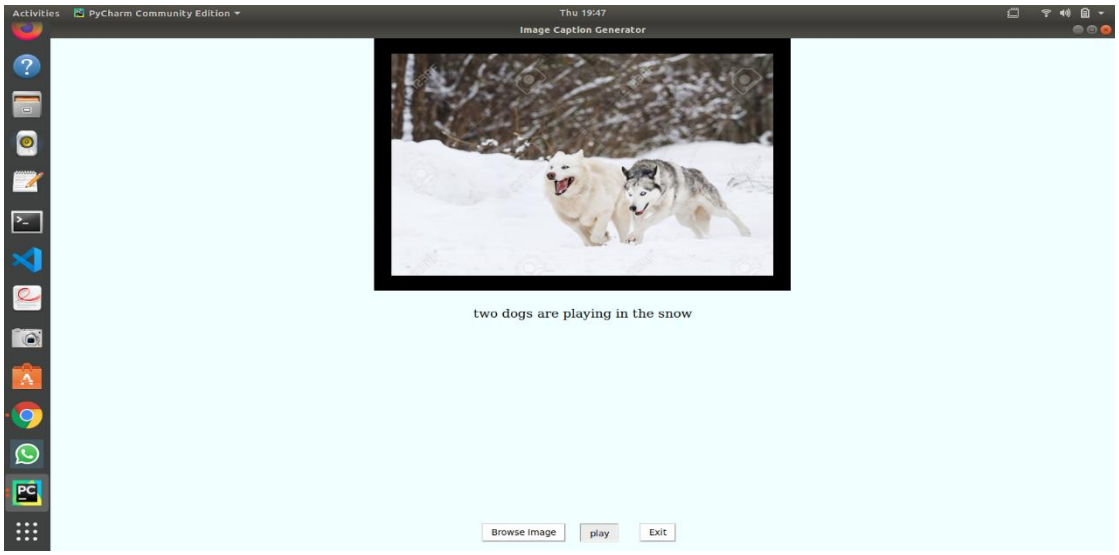

Fig 4.4.2: Speech Generation

5. CONCLUSION

A deep learning approach for the captioning of images and the GTTS API for the conversion of generated description into speech is implemented. The sequential API of keras was used with Tensorflow as a backend to implement the deep learning architecture to achieve an effective BLEU score of 0.52 for model. The Bilingual Evaluation Understudy Score, or BLEU for short, is a metric for evaluating a generated sentence to a reference sentence.

6. REFERENCES

[1] B.Krishnakumar , K.Kousalya , S.Gokul , R.Karthikeyan and D.Kaviyarasu, “IMAGE CAPTION GENERATOR USING DEEP LEARNING”, International Journal of Advanced Science and Technology, Vol. 29, No. 3s, (2020), pp. 975-980.

[2] Marc Tanti, Albert Gatt, Kenneth P. Camilleri, “What is the Role of Recurrent Neural Networks (RNNs) in an Image Caption Generator?”, arXiv:1708.02043v2 [cs.CL], 25 Aug 2017.

[3] Srikanth Tammina, “Transfer learning using VGG-16 with Deep Convolutional Neural Network for Classifying Images”, International Journal of Scientific and Research Publications, Volume 9, Issue 10, October 2019.