A PROJECT REPORT

on

ENHANCING VISUAL UNDERSTANDING THROUGH VERBAL IMAGE CAPTION GENERATOR USING ADVANCED DNN TECHNIQUES

Submitted in partial fulfillment of the requirements for the award of

BACHELOR OF ENGINEERING

IN

INFORMATION TECHNOLOGY

Submitted by

TIRUVEEDHULA AMRUTHA LAKSHMI – 20BQ1A12G9

RAYALA MEGHANA – 20BQ1A12E2

TUMMALAPUDI SOWMYA – 20BQ1A12H1

KOPPULA BHAVYA SREE – 21BQ5A1222

Under the esteemed guidance of

Mr. B. AVINASH M. Tech (Ph. D);

Assistant Professor

DEPARTMENT OF INFORMATION TECHNOLOGY

VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY

NAMBUR (V), PEDAKAKANI (M), GUNTUR-522 508, TEL no: 0873 2118036,

www.vvitguntur.com, approved by AICTE, permanently affiliated to JNTUK

Accredited by NAAC with "A" grade, Accredited by NBA for 3 years

VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY:: NAMBUR

BONAFIDE CERTIFICATE

Certified that this project report **"ENHANCING VISUAL UNDERSTANDING THROUGH VERBAL IMAGE CAPTION GENERATOR USING ADVANCED DNN TECHNIQUES"** is the bonafide work of by "**T. AMRUTHA LAKSHIMI (20BQ1A12G9), R.MEGHANA(20BQ1A12E2),T.SOWMYA(20BQ1A12H1),K.BHAVYA SREE(21BQ5A1222)**", who carried out the project under my guidance during the year 2020 towards partial fulfillment of the requirements of the Degree of Bachelor of Technology in Information Technology from Jawaharlal Nehru Technological University, Kakinada. The results embodied in this report have not been submitted to any other University for the award of any degree.

Signature of the Supervisor

B. AVINASH
Assistant Professor, IT.

Signature of the Head of the Department

Dr. KALAVATHI ALLA
Professor, IT.

Submitted for Viva voce Examination held on _____

EXTERNAL EXAMINER

VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY:: NAMBUR

CERTIFICATE OF AUTHENTICATION

We solemnly declare that this project **"ENHANCING VISUAL UNDERSTANDING THROUGH VERBAL IMAGE CAPTION GENERATOR USING ADVANCED DNN TECHNIQUES"** is the Bonafide work done purely by us, carried out under the supervision of Mr. B. AVINASH, towards partial fulfillment of the requirements of the Degree of Bachelor of Technology in Information Technology from Jawaharlal Nehru Technological University, Kakinada during the year 2023-24.

It is further certified that this work has not been submitted, either in part of in full, to any other department of the Jawaharlal Nehru Technological University, or any other University, institution or elsewhere, or for publication in any form.

Signature of the Student

T. AMRUTHA LAKSHMI – 20BQ1A12G9_____.

R. MEGHANA – 20BQ1A12E2_____.

T. SOWMYA – 20BQ1A12H1_____.

K. BHAVYA SREE – 21BQ5A1222_____.

ACKNOWLEDGEMENT

We take this opportunity to express our deepest gratitude and appreciation to all those people who made this project work easier with words of encouragement, motivation, discipline, and faith by offering different places to look to expand my ideas and help me towards the successful completion of this project work.

First and foremost, we express our deep gratitude to **Mr. Vasireddy Vidyasagar**, Chairman, Vasireddy Venkatadri Institute of Technology for providing necessary facilities throughout the Information Technology program.

We express our sincere thanks to **Dr. Y. Mallikarjuna Reddy**, Principal, Vasireddy Venkatadri Institute of Technology for his constant support and cooperation throughout the Information Technology program.

We express our sincere gratitude to **Dr.A. Kalavathi**, Professor & HOD, Information Technology, Vasireddy Venkatadri Institute of Technology for her constant encouragement, motivation and faith by offering different places to look to expand my ideas. We would like to express our sincere gratitude to our guide **Mr. Avinash** for his insightful advice, motivating suggestions, invaluable guidance, help and support in successful completion of this project.

We would like to take this opportunity to express our thanks to the **teaching and non-teaching** staff in the Department of Information Technology, VVIT for their invaluable help and support.

T. Amrutha Lakshmi
R. Meghana
T. Sowmya
K. Bhavya Sree

ABSTRACT

In recent years, image caption generators have made significant strides in utilizing Deep Learning techniques to automatically generate descriptive captions for images. The use of Deep Learning techniques by image caption generators to automatically provide meaningful descriptions for images has advanced significantly in recent years. This technology improves accessibility for visually impaired individuals and enriches multimedia platforms. By combining image captioning and voice synthesis, this project promotes a more inclusive and comprehensive understanding of the digital visual world. However, these systems are limited in their reliance on textual outputs, which can be challenging for users with visual impairments or in situations where reading is impractical. To address this limitation, we propose an innovative extension to the existing image caption generator. Our project integrates a verbal description mechanism to read out the generated captions, enhancing accessibility and user experience. This concise integration empowers users to interact with visual content audibly, making the system more inclusive and user-friendly in various contexts. This extension opens-up new possibilities for applications in assistive technologies, multimedia presentations, and content consumption.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 INTRODUCTION

In today's digitally driven world, the fusion of advanced Deep Neural Network (DNN) techniques and natural language processing has opened up exciting possibilities for enhancing visual understanding. Visual content, such as images and videos, has become increasingly prevalent in our daily lives, making it essential to bridge the gap between visual data and human comprehension. One remarkable application of this synergy is the development of a Verbal Image Caption Generator.

The project, "Enhancing Visual Understanding through Verbal Image Caption Generator using Advanced DNN Techniques," is an innovative endeavor designed to address the challenge of making visual content more accessible and meaningful to individuals by automatically generating descriptive and contextually relevant captions for images. This project combines the power of cutting-edge DNN techniques with natural language processing to create a sophisticated system capable of transforming images into human-understandable text.

The need for such a system is driven by the ever-growing volume of visual content on the internet, in sectors like e-commerce, social media, and content creation, where images and videos abound. While visual content can be incredibly engaging, it can also pose accessibility challenges for people with visual impairments or those who require textual descriptions to comprehend the content fully. Furthermore, this technology has vast potential in various industries, including healthcare (medical image descriptions), autonomous vehicles (visual perception), and education (improving accessibility for educational materials).

In this project, we will leverage advanced DNN techniques, such as convolutional neural networks (CNNs) for image processing and recurrent neural networks (RNNs) for natural language generation, to build a robust and accurate Verbal Image Caption Generator. The system will analyze the visual features of an image and translate them into informative textual descriptions. It will also incorporate context and semantics to ensure that the generated captions are not just a collection of keywords but provide a deeper understanding of the visual content.

1.2 NEED OF THE PROJECT

The project "Enhancing Visual Understanding through Verbal Image Caption Generator using Advanced DNN Techniques" serves a multitude of important needs in the current digital landscape. One of its most significant contributions is in the realm of accessibility and inclusivity. In today's digital world, visual content is pervasive, and many websites, social media platforms, and educational resources predominantly rely on images and videos. This leaves individuals with visual impairments at a distinct disadvantage, as they may struggle to access and comprehend such content. This project addresses this need by automatically generating descriptive and contextually relevant verbal descriptions of images, thereby ensuring a more inclusive digital environment for all users.

Content moderation is another pressing issue in the age of user-generated content and social media platforms. Ensuring that harmful or inappropriate content is swiftly identified and moderated is paramount. The project's ability to generate textual captions for images can greatly aid in this effort. By providing a textual understanding of the image content, it becomes easier to detect and categorize problematic visuals, contributing to safer and more responsible content sharing.

The project's applications extend to emerging technologies as well. In the context of autonomous vehicles, this technology can aid in visual perception and understanding the surroundings. It can describe objects, road conditions, and potential hazards, contributing to safer and more efficient autonomous driving systems. Furthermore, for content creators and marketers, this technology can streamline the process of producing descriptive content for their images.

Additionally, the project enhances the understanding of artificial intelligence and machine learning systems. AI systems can greatly benefit from having richer contextual information about images, enabling them to make more informed decisions and provide superior user experiences in applications ranging from virtual assistants to recommendation systems. Finally, the project contributes to improved user experience. By providing descriptive captions for images, user interfaces become more intuitive and user-friendly, enriching the way users interact with various digital platforms and applications.

CHAPTER II

SYSTEM ANALYSIS

2.1 REQUIREMENT ANALYSIS

2.1.1 Non-Functional Requirements

Performance Requirements*:*

The System should generate captions and read them aloud within a reasonable response time for each Image. Also, System should be able to handle a growing dataset and user load without significant degradation in Performance.

Reliability:

The system should be available and reliable, with a high uptime percentage. It should have mechanisms in place to recover gracefully from failures, such as data errors.

Resource Utilization:

The system must be optimized for resource utilization, including CPU, memory, and Storage to minimize Operating Costs.

Backup and Recovery:

The System should have Robust backup and recovery mechanisms in place to prevent data loss and enable quick system restoration in case of failures.

2.1.2 Functional Requirements

High Accuracy:

The system's ability ensure that generated Captions are Accurate and contextually relevant to Images.

Real-time Processing:

System must Provide the capability to generate captions and read them aloud in real-time for Images

Voice Output:

Provide an Option to read aloud the generated captions using text-to-speech (TTS) engine.

Scalability:

The ability of System to Handle a large number of Images.

2.2 Hardware Requirements:

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by the software engineers as the starting point for the system design. In hardware requirement we require all those components which will provide the platform for the development of project. The minimum hardware required for the development of this project is as follows –

- LAPTOP
- HARD DISK : 500 GB - 1 TB
- RAM : 8GB
- PROCESSOR : AMD Ryzen 5 5500U

2.3 Software Requirements:

The software requirements are the software specifications of the system. It should include both a definition and specification of requirements. It is a set of what the system should do rather than how it should do it. It is useful in estimating cost, planning team activities, performing tasks and tracking the team's progress throughout the development activity.

- OPERATING SYSTEM : WINDOWS 7 AND ABOVE
- TECHNOLOGY STACK : PYTHON 3.6, TENSORFLOW 1.14.0
- TOOLS : JUPYTER NOTEBOOK 6.0.3,

2.4 EXISTING SYSTEM

The existing system involves a software application or algorithm that takes an input image as its primary data source. This image is then processed using advanced computer vision techniques and deep learning models. These models analyze the visual features of the image and extract relevant information about objects, scenes, and context within the image. Once the image features are extracted, they are fed into a natural language processing (NLP) model. This NLP model generates a coherent and contextually relevant caption that describes the content of the image. The model has

been trained on a large dataset of images paired with corresponding captions to learn how to generate accurate and informative descriptions. Users interact with the existing system through a graphical user interface (GUI) or an application programming interface (API). They upload or input an image into the system, and the system processes the image and generates a caption. Users typically view the generated caption on the same interface.

2.5 PROPOSED SYSTEM

•

The Proposed methodology for generating captions is the detection and recognition of objects using deep learning with voice synthesis. It consists of object detection, feature extraction, Convolution Neural Network (CNN) for feature extraction and for scene classification. So, to make our image caption generator model, we will be merging these architectures. It is also called a CNN-RNN model. We will use the pre-trained model Exception. And LSTM will use the information from CNN to help generate a description of the image. Once the caption is generated, it will play it. Our proposed system enhances image captioning by integrating Text-to-Speech (TTS) technology, generating voice narrations for image captions, improving accessibility for visually impaired users and those who prefer auditory content. The system comprises four primary components: image captioning, TTS, user interface, and accessibility features, allowing users to customize voice settings. Benefits include enhanced accessibility, improved engagement, and the creation of multimodal content. This system is applicable in educational platforms, multimedia presentations, and online content accessibility, catering to a broader audience. It aligns with digital accessibility principles, promoting a more inclusive and engaging digital experience for all users. This innovative approach bridges the gap between visual and auditory information, ensuring a richer and more inclusive interaction with visual content.

CHAPTER III

SYSTEM DESIGN

3.1 SYSTEM ARCHITECTURE

The system architecture for an image caption generator project begins with the Image Processing Component, where the user submits an image as input. This component is responsible for receiving the image, performing preprocessing tasks such as resizing, cropping, and normalization, and then forwarding the processed image to the subsequent components.

The core of the system lies in the Image Caption Generator Component, which takes the processed image as input. This component employs deep learning techniques, such as a pre-trained Convolutional Neural Network (CNN) to extract relevant features from the image, followed by a Recurrent Neural Network (RNN) to generate textual captions describing the image content. This caption generation process results in a text-based description that conveys the content and context of the image.

The next crucial step is the Text-to-Speech (TTS) Component, which takes the generated text caption as input and transforms it into audio. Utilizing a Text-to-Speech engine, such as Google Text-to-Speech or Amazon Polly, or open-source libraries like gTTS, the system converts the text into an audio representation, making it accessible to users who prefer auditory information.

The Audio Output Component is responsible for playing the generated audio to the user. Depending on the chosen development platform and technology stack, this component can employ platform-specific audio playback APIs to ensure that the audio caption is presented to the user.

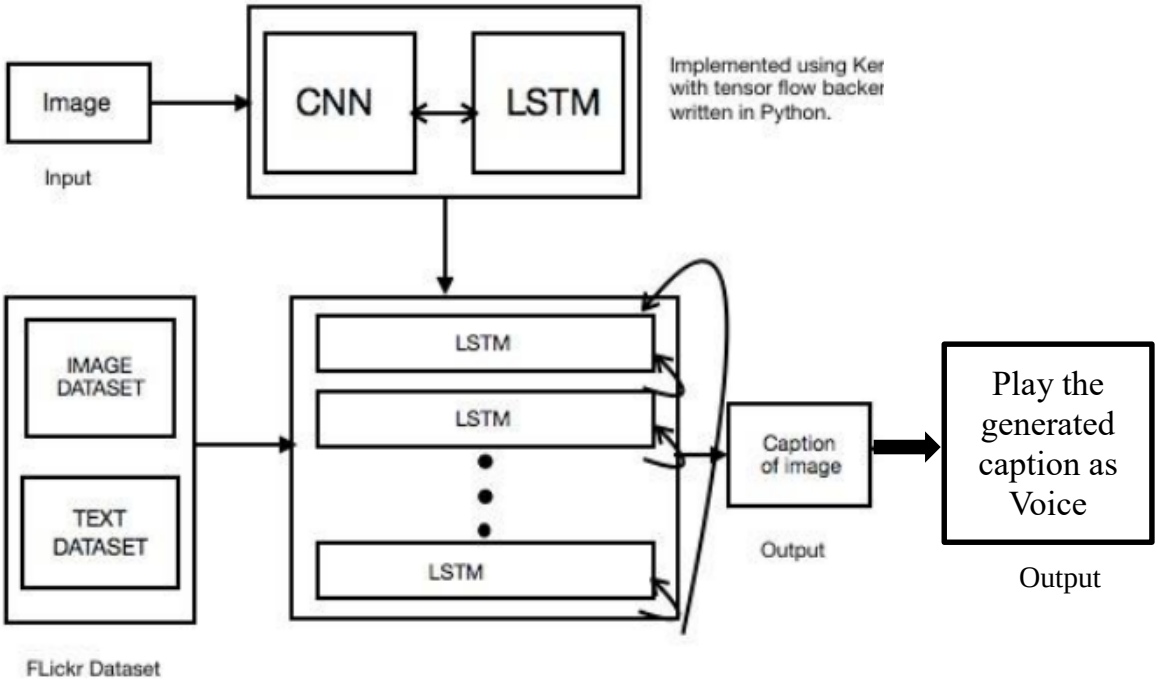

Fig 3.1 System Architecture of Image Caption Generator

3.2 UML DIAGRAMS

UML diagrams are crucial as they provide a visual blueprint, ensuring clarity in the design of complex interactions between image processing, CNN, LSTM, and text-to-speech components. They facilitate effective communication among team members, enabling a shared understanding of system architecture. Additionally, UML diagrams aid in documentation, validation, and project management, ultimately streamlining development and ensuring project success. Beyond their role in communication, UML diagrams serve as living documentation, capturing vital design decisions and facilitating knowledge transfer among team members. They act as early warning systems, enabling the detection of potential flaws or inconsistencies before they manifest as issues during implementation.

3.2.1 USE CASE DIAGRAM

The use-case diagram for the Image Caption Generator project depicts key actors like the

“User” and “Text-to-Speech Engine”. It outlines three primary use cases: “Upload Image,” “Generate Caption,” and “Read Caption Aloud”. This concise diagram offers a high-level view of system functionality and actor interactions, aiding in requirements analysis and system communication.

Use-case diagram for Image Caption Generator:

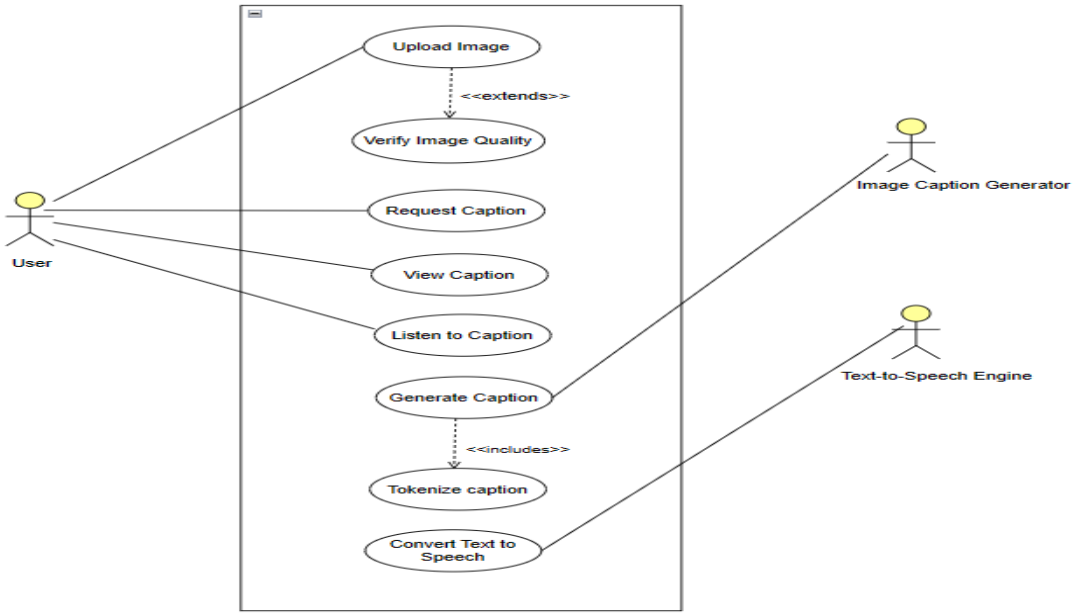

Fig 3.2 Image Caption Generator Use-case diagram

3.2.2 CLASS DIAGRAM

A Class diagram would provide a structured overview of the project's object-oriented design. It would include key classes and their relationships within the system. Some essential classes might include “Image-Processor” for handling image-related tasks, “Caption-Generator” responsible for generating captions using CNN and LSTM models, “Text-To-Speech” for converting text to speech, and “User” to represent user interactions.

Relationships between these classes, such as associations and dependencies, would be depicted to illustrate how they collaborate to achieve the system's functionalities. This class diagram would serve as a blueprint for the project’s architecture, guiding developers in implementing the different components and ensuring a cohesive and organized system for image captioning and auditory feedback.

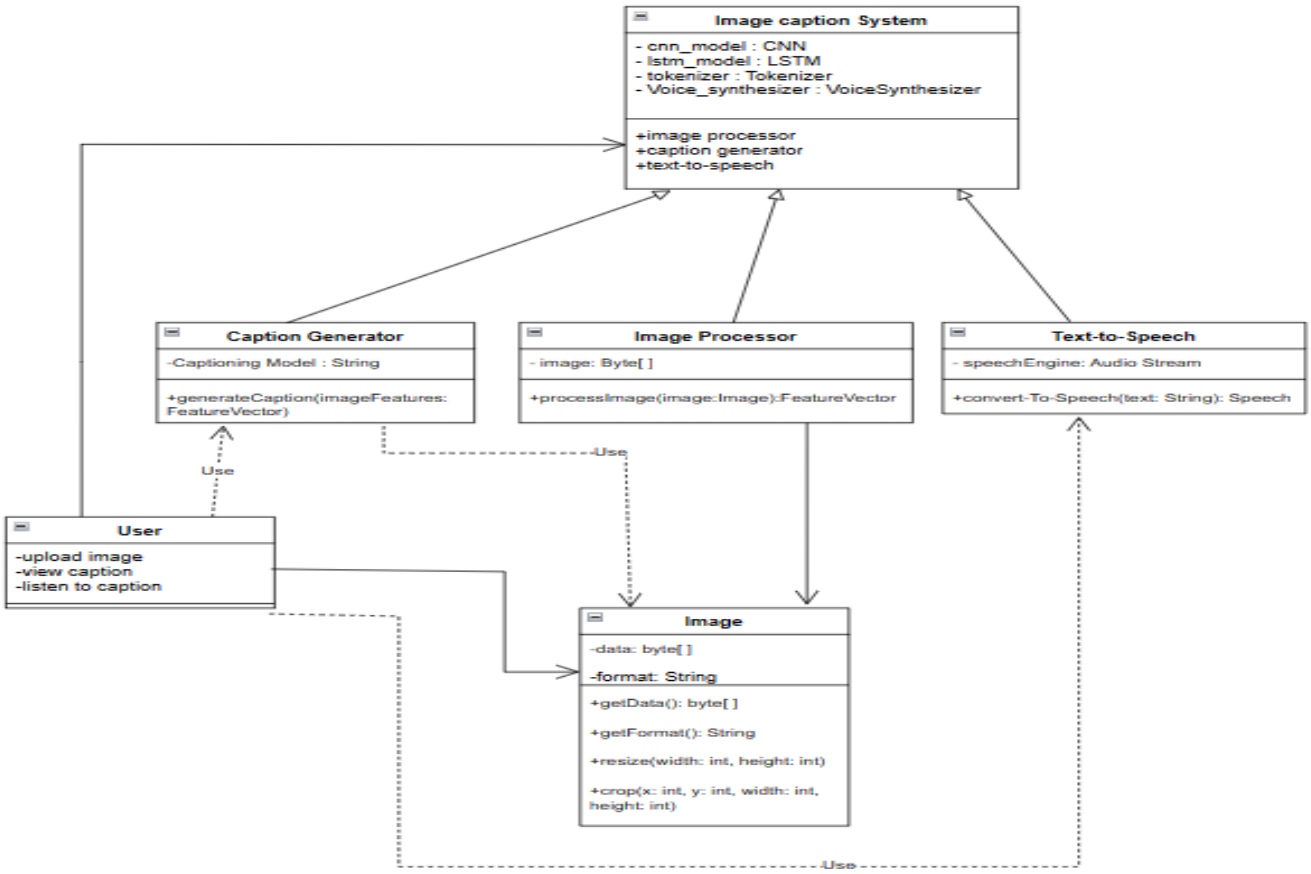

Fig 3.3 Class Diagram

3.2.3 ACTIVITY DIAGRAM

The activity diagram would provide a sequential representation of the system's processes. It would begin with the user uploading an image, followed by the image processing phase, which involves resizing and normalization. Subsequently, the diagram would depict the generation of captions using deep neural networks (CNN and LSTM). Once captions are generated, the activity diagram would show the text-to-speech conversion, where the system reads the caption aloud. This visual representation clarifies the workflow, ensuring a step-by-step understanding of how the system processes images and generates spoken captions, facilitating both design and implementation phases of the project.

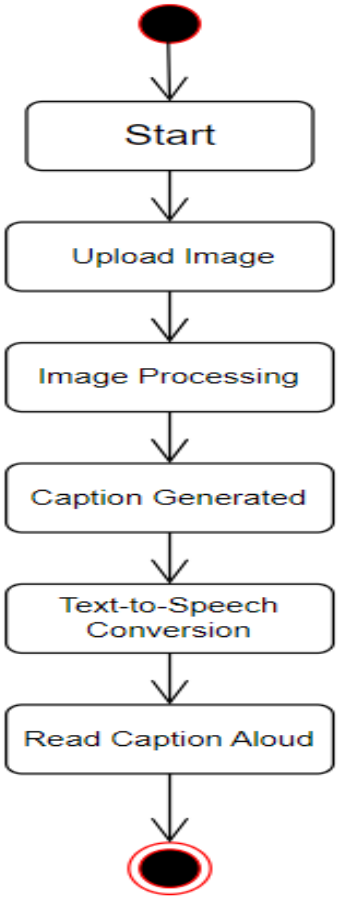

Fig 3.4 Activity Diagram

3.2.4 SEQUENCE DIAGRAM

The sequence diagram illustrates the chronological flow of interactions between system components and actors. The “User” initiates the process by uploading an image, triggering a sequence of events that includes image processing, caption generation, and finally, the reading of the generated caption aloud by the “Text-to-Speech Engine”. This diagram provides a visual representation of the step-by-step execution of the system, facilitating understanding and analysis of its operational logic.

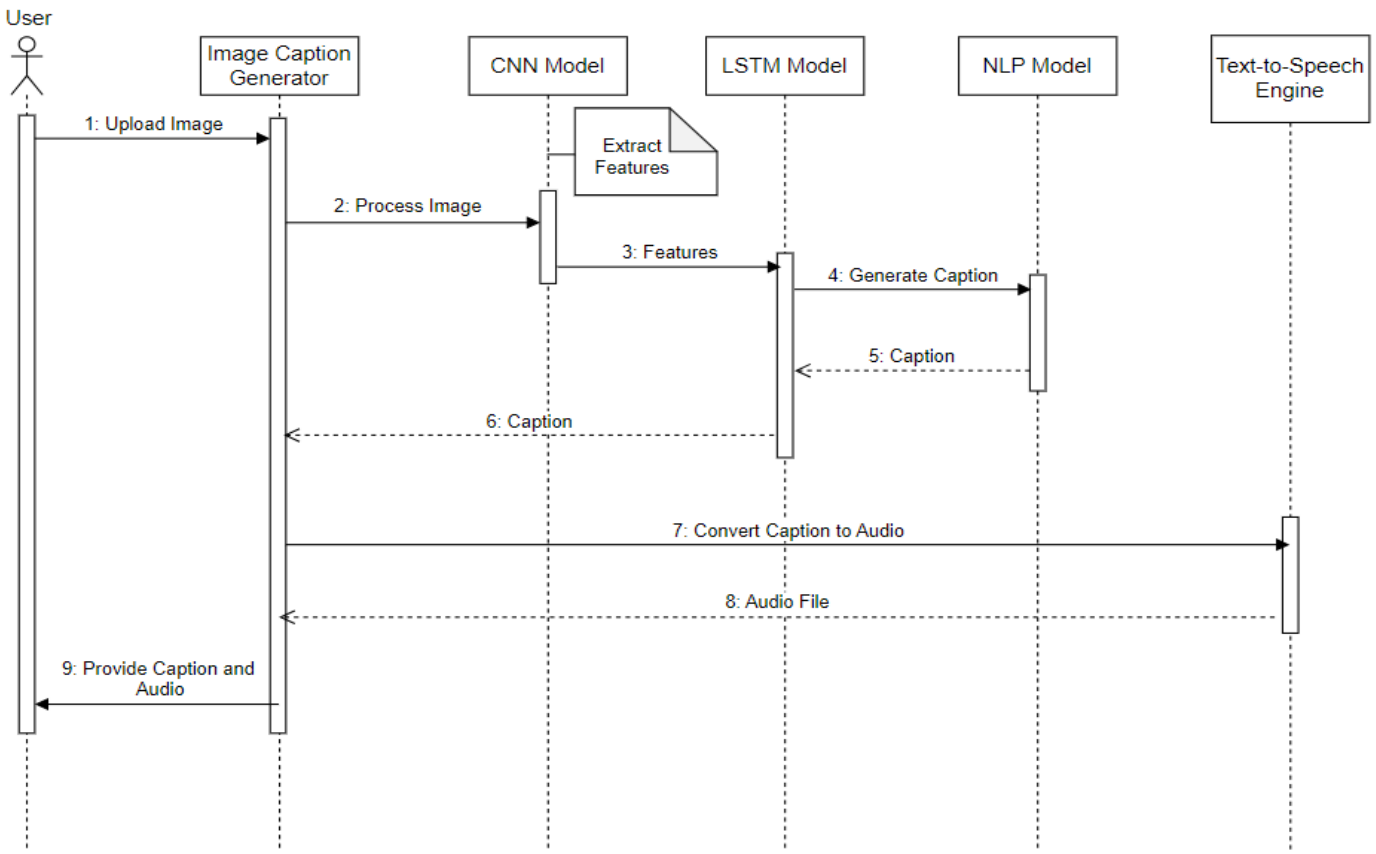

Fig 3.5 Sequence Diagram

CHAPTER IV
SYSTEM IMPLEMENTATION

4.1 TECHNOLOGIES USED:

4.1.1 PYTHON

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released in 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3.

The Python 2 language, i.e. Python 2.7.x, was officially discontinued on 1 January 2020 (first planned for 2015) after which security patches and other improvements will not be released for it. With Python 2's end-of-life, only Python 3.5.x and later are supported.

Python interpreters are available for many operating systems. A global community of programmers develops and maintains Python, an open source reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)) Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing and a combination of reference counting and a cycle- detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is sometimes termed the **off-side rule**, which some other languages share, but in most languages, indentation doesn't have any semantic meaning.

4.1.2 JUPYTER NOTEBOOK

Project Jupyter is a project and community whose goal is to "develop open source software, open-standards, and services for interactive computing across dozens of programming languages". It was spun off from IPython in 2014 by Fernando Pérez.

The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people at project jupyter.

IPython Notebook project itself. The name, Jupyter, comes from the core supported programming languages that it supports: Julia, Python, and R. Jupyter ships with the IPython kernel, which allows you to write your programs in Python, but there are currently over 100 other kernels that you can also use.

The Jupyter Notebook is not included with Python, so if you want to try it out, you will need to install Jupyter.

Fig 4.1 Jupyter Notebook

There are many distributions of the Python language. This article will focus on just two of them for the purposes of installing Jupyter Notebook. The most popular is CPython, which is the reference version of Python that you can get from their website. It is also assumed that you are using python 3.

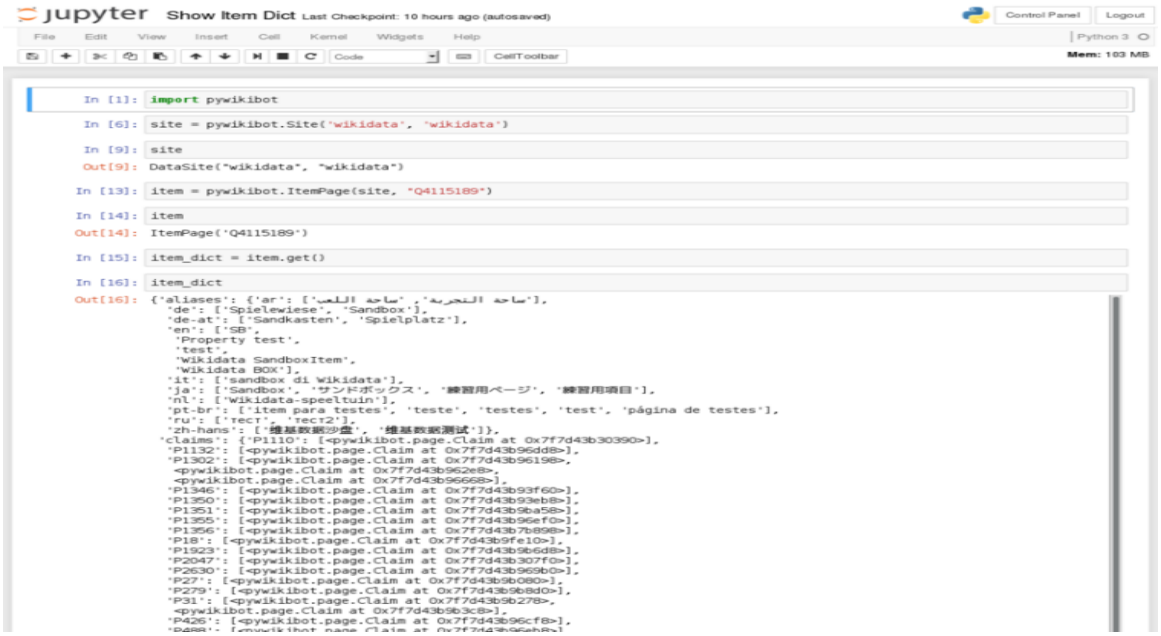

Fig 4.2 Jupyter Notebook Interface

A Notebook's cell defaults to using code whenever you first create one, and that cell uses the kernel that you chose when you started your Notebook. In this case, you started yours with

Python 3 as your kernel, so that means you can write Python code in your code cells. Since your initial Notebook has only one empty cell in it, the Notebook can't really do anything.
Thus, to verify that everything is working as it should, you can add some Python code to the cell and try running its contents. Jupyter Notebook supports adding rich content to its cells. In this section, you will get an overview of just some of the things you can do with your cells using Markup and Code.

Jupyter Notebook provides a browser-based REPL built upon a number of popular open-source libraries:

- IPython
- ØMQ (ZeroMQ)
- Tornado (web server)
- jQuery
- Bootstrap (front-end framework)
- MathJax

4.1.3 TENSORFLOW

TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. TensorFlow is a symbolic math library based on dataflow and differentiable programming. It is used for both research and production at Google. TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache License 2.0 in 2015.

TensorFlow is Google Brain's second-generation system. Version 1.0.0 was released on February 11, 2017. While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS.

TensorFlow is an end-to-end open-source platform for machine learning. TensorFlow is a rich system for managing all aspects of a machine learning system; however, this class focuses

on using a particular TensorFlow API to develop and train machine learning models. See the TensorFlow documentation for complete details on the broader TensorFlow system.

TensorFlow APIs are arranged hierarchically, with the high-level APIs built on the low-level APIs. Machine learning researchers use the low-level APIs to create and explore new machine learning algorithms. In this class, you will use a high-level API named tf.keras to define and train machine learning models and to make predictions. tf.keras is the TensorFlow variant of the open-source Keras API.

TensorFlow allows developers to create dataflow graphs—structures that describe how data moves through a graph, or a series of processing nodes. Each node in the graph represents a mathematical operation, and each connection or edge between nodes is a multidimensional data array, or tensor.TensorFlow provides all of this for the programmer by way of the Python language. Python is easy to learn and work with, and provides convenient ways to express how high-level abstractions can be coupled together. Nodes and tensors in TensorFlow are Python objects, and TensorFlow applications are themselves Python applications.

The actual math operations, however, are not performed in Python. The libraries of transformations that are available through TensorFlow are written as high-performance C++ binaries. Python just directs traffic between the pieces, and provides high-level programming abstractions to hook them together.

TensorFlow applications can be run on most any target that’s convenient: a local machine, a cluster in the cloud, iOS and Android devices, CPUs or GPUs. If you use Google’s own cloud, you can run TensorFlow on Google’s custom TensorFlow Processing Unit (TPU) silicon for further acceleration. The resulting models created by TensorFlow, though, can be deployed on most any device where they will be used to serve predictions.

4.2 ALGORITHMS

4.2.1 Convolutional Neural Network (CNN):

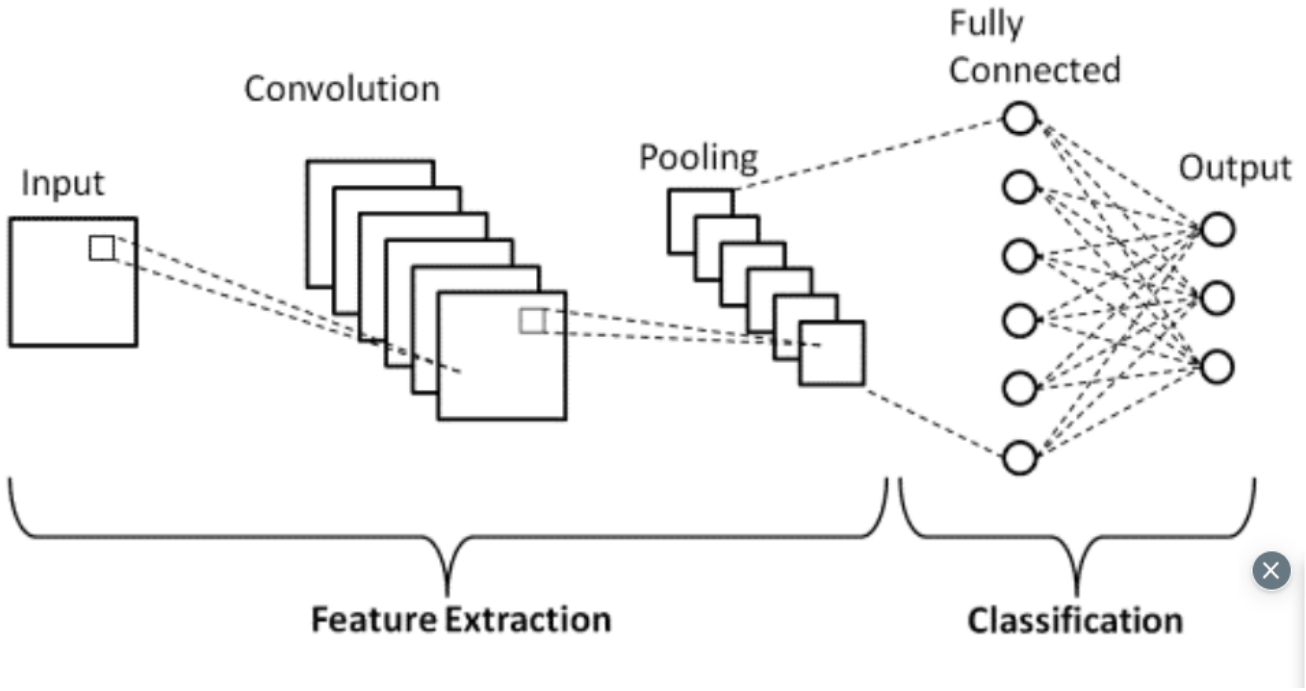

Fig 4.3 CONVOLUTIONAL NEURAL NETWORK

Input Image:

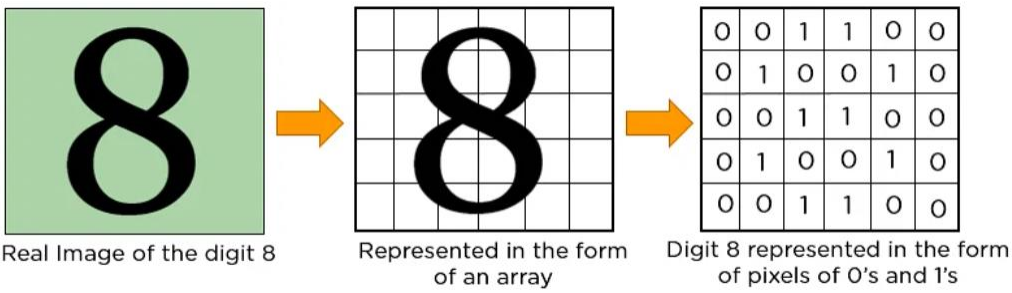

Fig 4.4 Input Image

Layers in Convolutional Neural Network:

1. Convolution Layer
2. ReLu Layer
3. Pooling Layer
4. Fully connected Layer

1. Convolution Layer:

This layer is used to extract the various features from the input images by performing convolution between the input image and a filter of a particular size M x M. The output is termed as the Feature map consists information about the image such as the corners and edges.

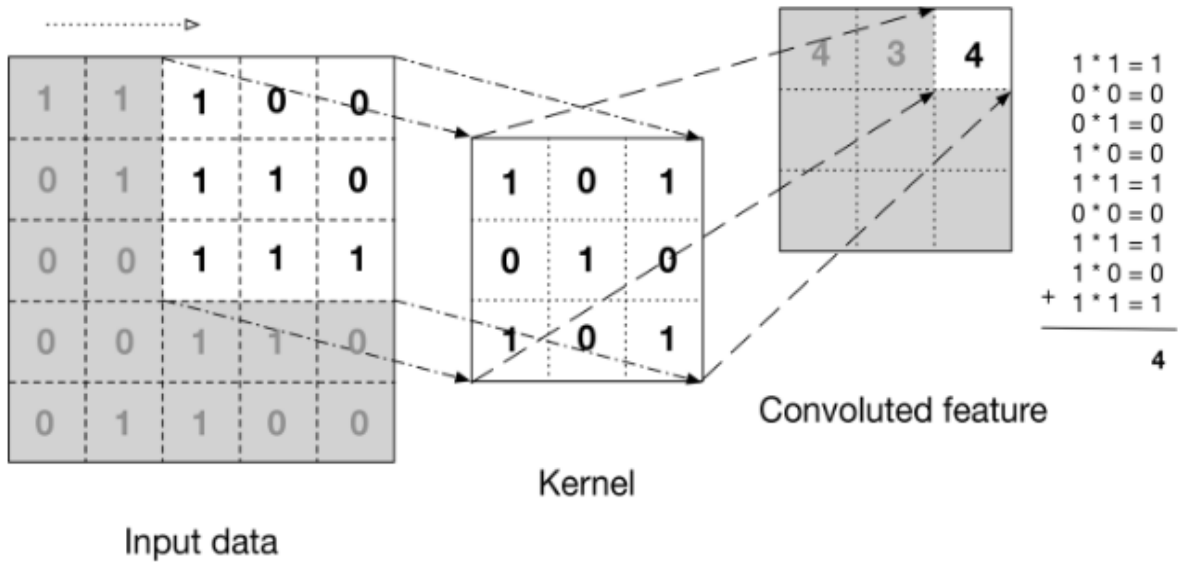

Fig 4.4.1 In Convolutional layer

2. ReLu Layer:

ReLU stands for the rectified linear unit. Once the feature maps are extracted, the next step is to move them to a ReLU layer.It performs an element-wise operation and sets all the negative pixels to 0.

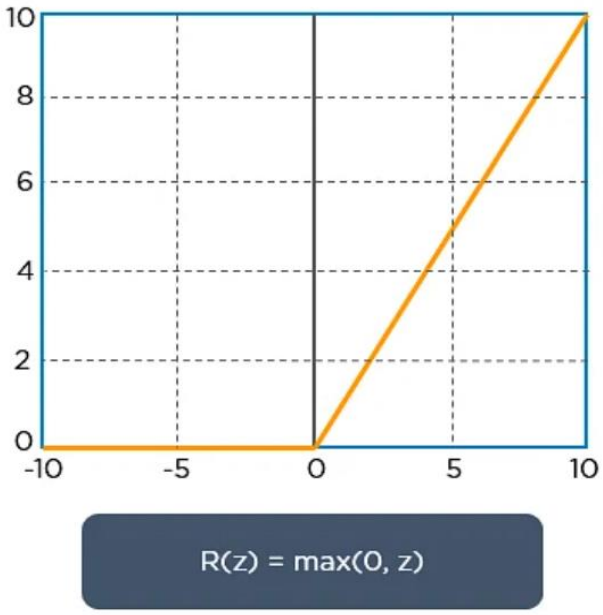

Fig 4.4.2 In ReLu Layer

3. Pooling Layer:

Pooling process which means it reduces the dimensionality of our matrix. The main importance of pooling is to reduce overfitting and computation in our dataset. Also it detect the edges, corners and facial features like nose, eyes by using multiple filters.

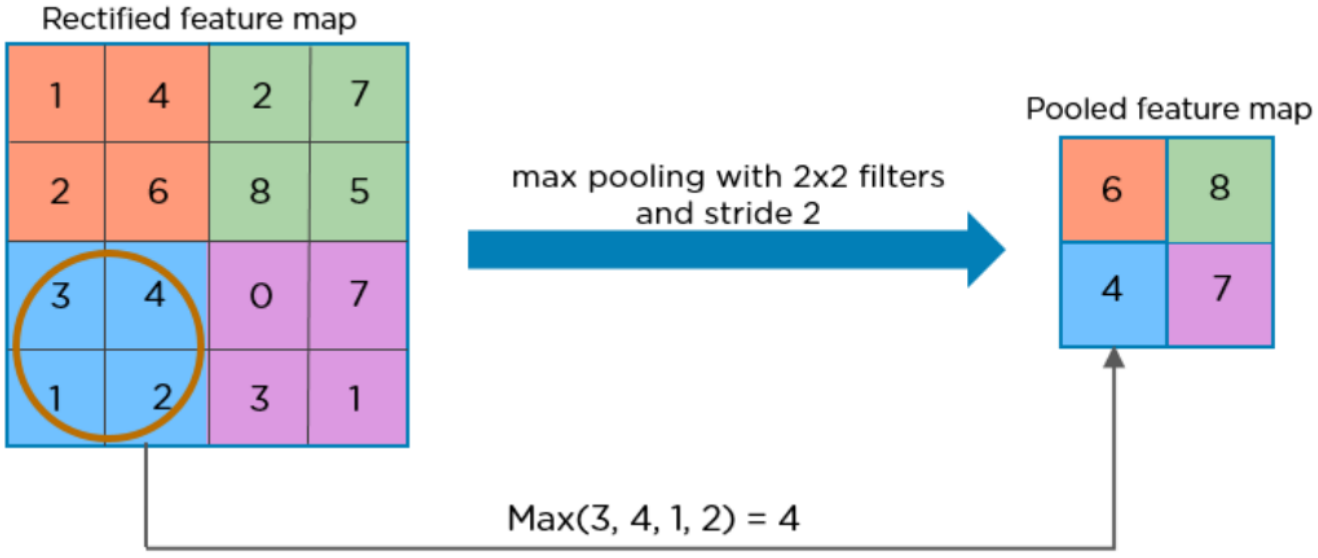

Fig 4.4.3 In Pooling Layer

4. Fully connected layer:

In this, the input image from the previous layers are flattened and fed to the FC layer. After the FC layer, the output is a vector of class scores. An activation function like soft-max is applied to obtain class probabilities for classification.The highest probability score indicates the predicted class or category for the input data.

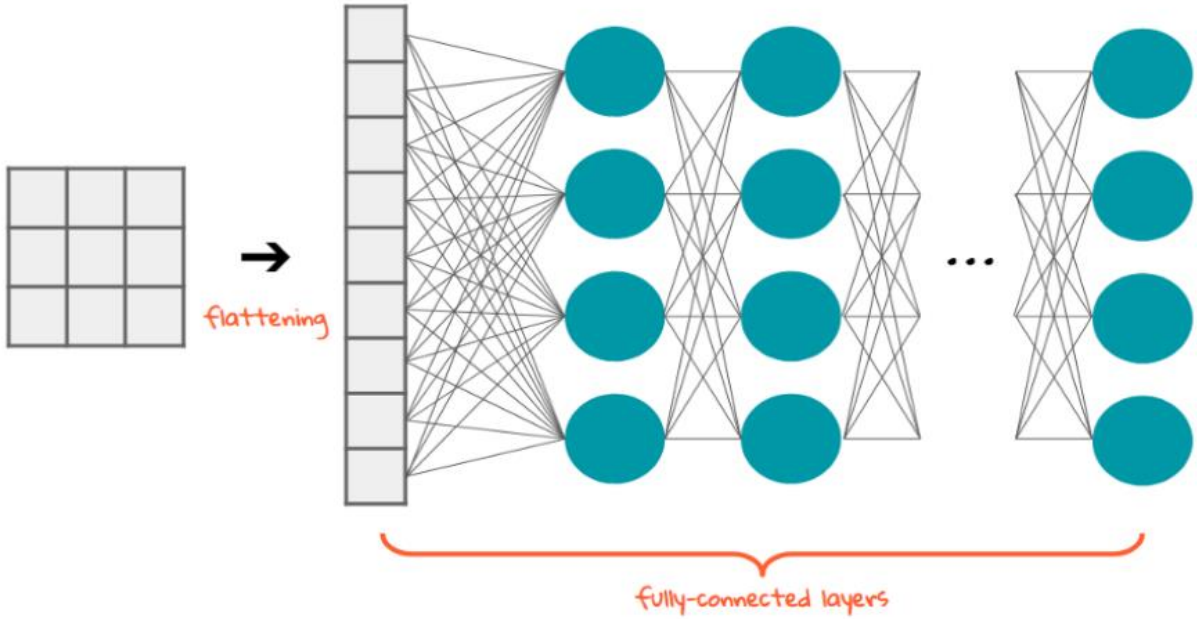

Fig 4.4.4 Fully Connected Layer

Summary of CNN:

1. Provide the data/dataset (any format of data) into the convolution layer.
2. Take convolution with featured kernel/filters.
3. Next apply a layer, Pooling for reducing the dimensions.
4. If you need a deeper understanding add these layers repeatedly.
5. Next step is to apply to flatten for a single column.
6. Feed into a fully connected layer.
7. Now lastly train the model with appropriate metrics and optimizer.

4.2.2 LSTM ARCHITECTURE:

LSTMs Long Short-Term Memory is a type of RNNs Recurrent Neural Network. LSTMs are able to process and analyse sequential data, such as time series, text and speech. They use a memory cell and gates to control the flow of information, allowing them to selectively retain or discard information as needed. LSTMs are widely used in various applications such as natural language processing, speech recognition, and time series forecasting.

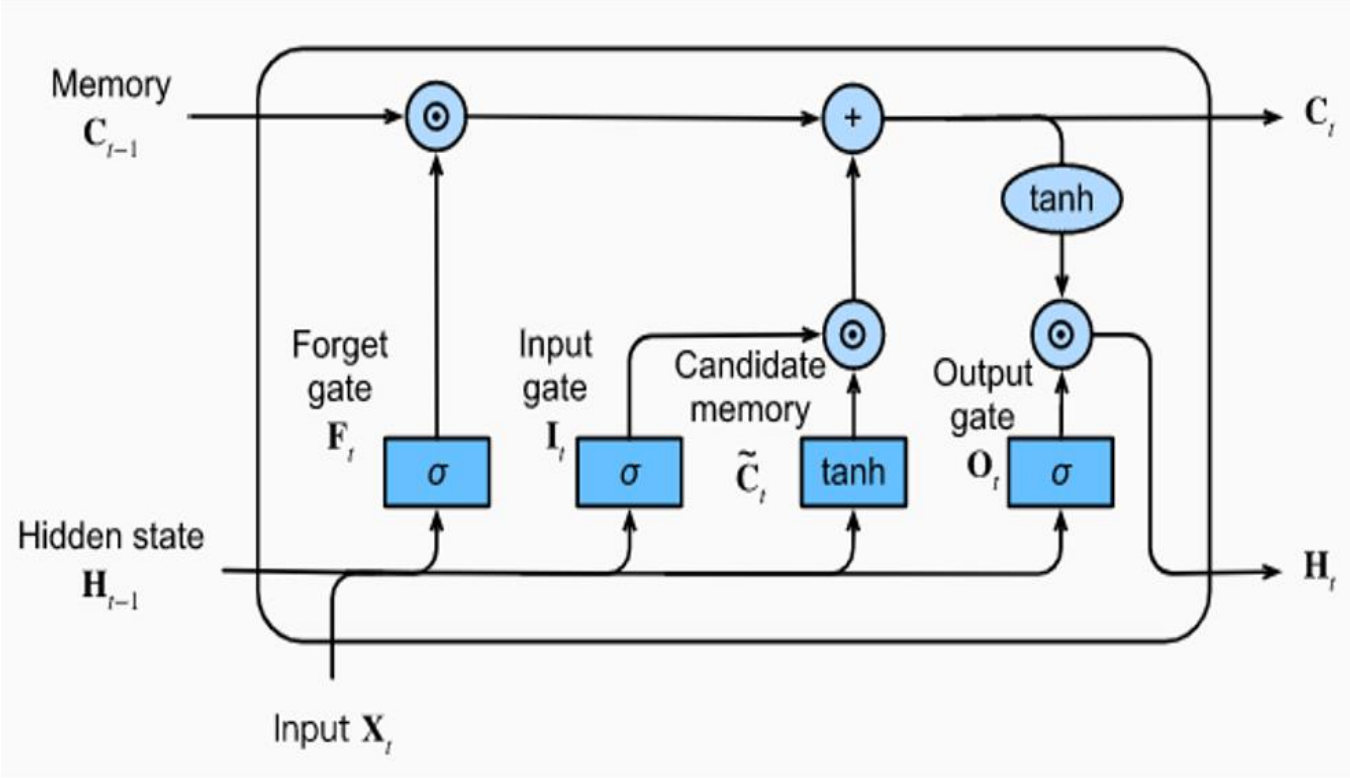

Fig 4.5 LSTM Architecture

BREAKING DOWN THE ARCHITECTURE OF LSTM:

This is the internal functioning of the LSTM network. The LSTM network architecture consists of three parts, as shown in the image below, and each part performs an individual function.

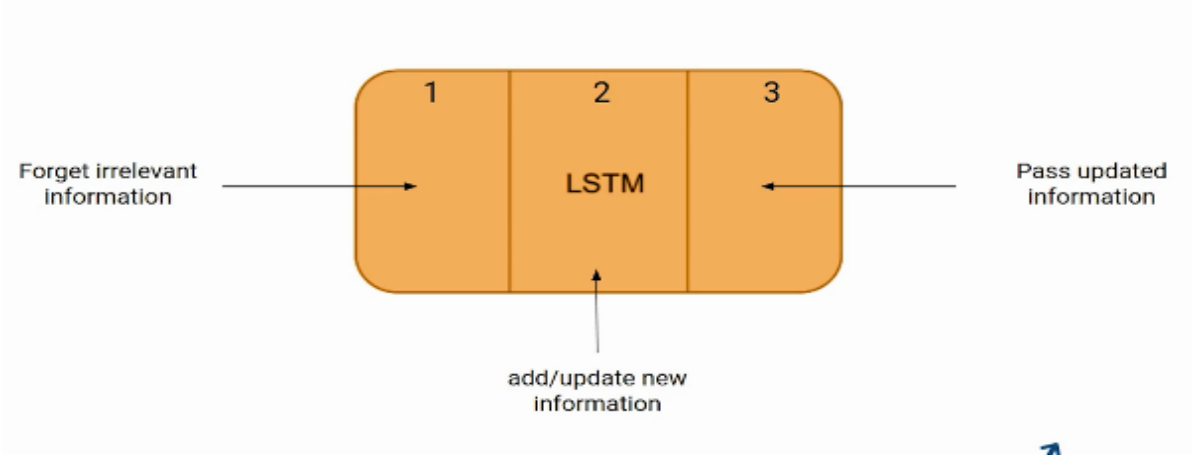

Fig 4.5.1 Breaking Down of LSTM Architecture

THE LOGIC BEHIND LSTM:

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. This one cycle of LSTM is considered a single-time step.

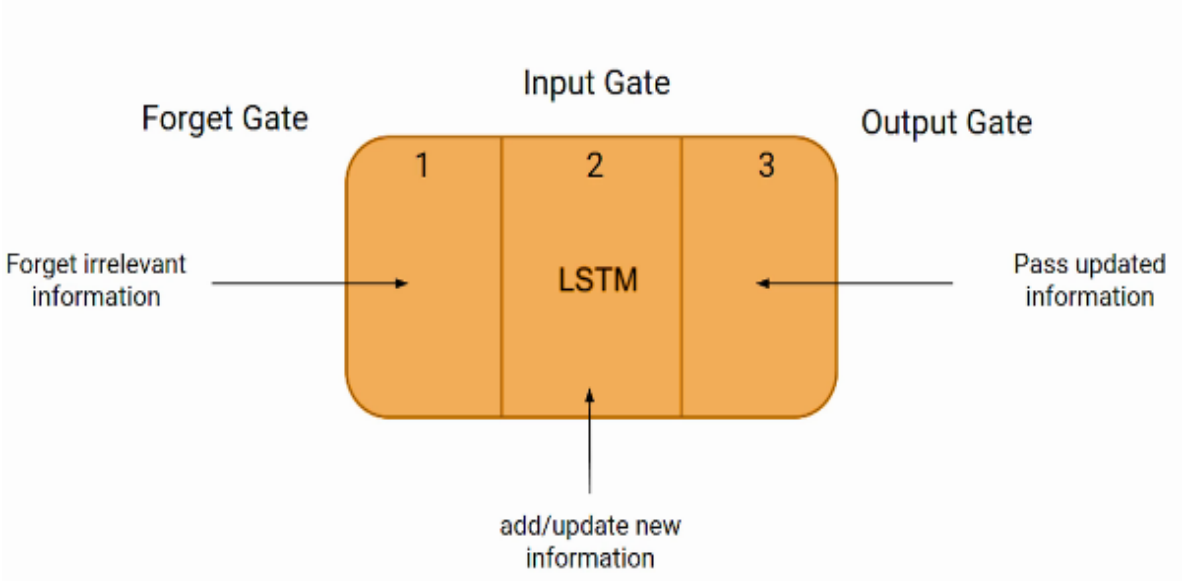

Fig 4.5.2 LSTM Gates

These three parts of an LSTM unit are known as gates. They control the flow of information in and out of the memory cell or LSTM cell.The first gate is called **Forget gate, the** second gate is known as **the Input gate,** and the last one is **the Output gate**.

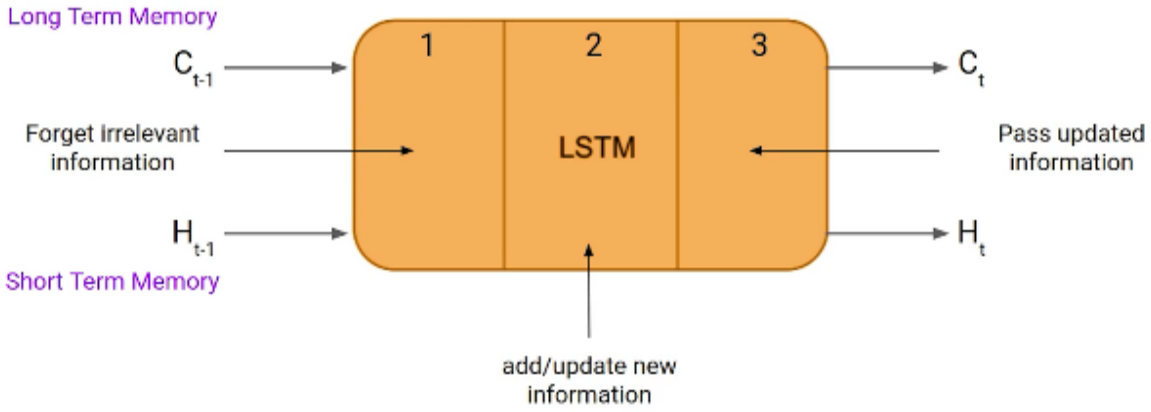

Fig 4.5.3 Detailed LSTM Gates

Just like a simple RNN, an LSTM also has a hidden state where H(t-1) represents the hidden state of the previous timestamp and H(t) is the hidden state of the current timestamp.In addition to that, LSTM also has a cell state represented by C(t-1) and C(t) for the previous and current timestamps, respectively.Here the hidden state is known as Short term memory, and the cell state is known as Long term memory.

1. **INPUT GATE:**

The input gate decides which information to store in the memory cell. It is trained to open when the input is important and close otherwise.

2. **FORGET GATE :**

The forget gate decides which information to discard from the memory cell. It is trained to open when the information is no longer important and close when it is.

3. **OUTPUT GATE :**

The output gate is responsible for deciding which information to use for the output of the LSTM. It is trained to open when the information is important and close when it is not. The gates in an LSTM are trained to open and close based on the input and the previous hidden state. This allows the LSTM to selectively retain or discard information, making it more effective at capturing long-term dependencies.

4.3 SAMPLE CODE

```
# First, we import all the necessary packages
import string
import numpy as np
from PIL import Image
import os
from pickle import dump, load
from keras.applications.xception import Xception, preprocess_input
from keras.preprocessing.image import load_img, img_to_array
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.layers import add
from keras.models import Model, load_model
from keras.layers import Input, Dense, LSTM, Embedding, Dropout
# small library for seeing the progress of loops.
from tqdm.notebook import tqdm
tqdm().pandas()

    0it [00:00, ?it/s]

# Getting and performing data cleaning

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

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

#Data cleaning- lower casing, removing puntuations and words containing numbers
def cleaning_text(captions):
    table = str.maketrans("",string.punctuation)
    for img,caps in captions.items():
```

```
    for i,img_caption in enumerate(caps):

            img_caption.replace("-"," ")
            desc = img_caption.split()

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

        img_caption = ' '.join(desc)
        captions[img][i]= img_caption
  return captions

  def text_vocabulary(descriptions):
       # build vocabulary of all unique words
       vocab = set()
       for key in descriptions.keys():
           [vocab.update(d.split()) for d in descriptions[key]]
       return vocab

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


 # Set these path according to project folder in you system
dataset_text = "D:\dataflair projects\Project - Image Caption Generator\Flickr_8k_text"
dataset_images = "D:\dataflair projects\Project - Image Caption Generator\Flicker8k_Dataset"

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

```
#cleaning the descriptions
clean_descriptions = cleaning_text(descriptions)

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

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


Length of descriptions = 8092
Length of vocabulary = 8763


# Extracting the feature vector from all images
from pickle import load
file_path="C:\\Users\\Pavan\\Downloads\\python-project-image-caption-
generator\\features.p"
features = load(open(file_path,"rb"))


# Loading dataset for Training the model
#load the data
def load_photos(filename):
    file = load_doc(filename)
    photos = file.split("\n")[:-1]
    return photos
def load_clean_descriptions(filename, photos):
    #loading clean_descriptions
    file = load_doc(filename)
    descriptions = {}
    for line in file.split("\n"):
        words = line.split()
        if len(words)<1 :
            continue
        image, image_caption = words[0], words[1:]
        if image in photos:
            if image not in descriptions:
```

```
                descriptions[image] = []
            desc = '<start> ' + " ".join(image_caption) + ' <end>'
            descriptions[image].append(desc)
    return descriptions
def load_features(photos):
    #loading all features
    file_path="C:\\Users\\Pavan\\Downloads\\python-project-image-caption-
generator\\features.p"
    all_features = load(open(file_path,"rb"))
    #selecting only needed features
    features = {k:all_features[k] for k in photos}
    return features
filename = dataset_text + "/" + "Flickr_8k.trainImages.txt"
#train = loading_data(filename)
train_imgs = load_photos(filename)
train_descriptions = load_clean_descriptions("descriptions.txt", train_imgs)
train_features = load_features(train_imgs)

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

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

from keras.preprocessing.text import Tokenizer
def create_tokenizer(descriptions):
    desc_list = dict_to_list(descriptions)
```

```
        tokenizer = Tokenizer()
        tokenizer.fit_on_texts(desc_list)
        return tokenizer

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

7577

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

32

```
#Create Data Generator
#create input-output sequence pairs from the image description.
    #data generator, used by model.fit_generator()
    def data_generator(descriptions, features, tokenizer, max_length):
        while 1:
            for key, description_list in descriptions.items():
                #retrieve photo features
                feature = features[key][0]
                input_image, input_sequence, output_word = create_sequences(tokenizer, max_length, description_list, feature)
                yield [[input_image, input_sequence], output_word]

    def create_sequences(tokenizer, max_length, desc_list, feature):
        X1, X2, y = list(), list(), list()
```

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

#You can check the shape of the input and output for your model
[a,b],c = next(data_generator(train_descriptions, features, tokenizer, max_length))
a.shape, b.shape, c.shape
    ((47, 2048), (47, 32), (47, 7577))

#Defining the CNN-RNN Model:
from keras.utils import plot_model
# define the captioning model
def define_model(vocab_size, max_length):
    # features from the CNN model squeezed from 2048 to 256 nodes
    inputs1 = Input(shape=(2048,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)

    # LSTM sequence model
```

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

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

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

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

 #Training  the model
 # train our model
 print('Dataset: ', len(train_imgs))
 print('Descriptions: train=', len(train_descriptions))
 print('Photos: train=', len(train_features))
 print('Vocabulary Size:', vocab_size)
 print('Description Length: ', max_length)

 model = define_model(vocab_size, max_length)
 epochs = 5
 steps = len(train_descriptions)
 # making a directory models to save our models
 os.mkdir("models5")
 for i in range(epochs):
      generator=    data_generator(train_descriptions,    train_features,    tokenizer,
```

```
max_length)
        model.fit(generator, epochs=1, steps_per_epoch= steps, verbose=1)
        model.save("models5/model_" + str(i) + ".h5")
```

Dataset: 6000
Descriptions: train= 6000
Photos: train= 6000
Vocabulary Size: 7577
Description Length: 32
Model: "model_1"

Layer (type)	Output Shape	Param #	Connected to
input_19 (InputLayer)	[(None, 32)]	0	[]
input_18 (InputLayer)	[(None, 2048)]	0	[]
embedding_1 (Embedding)	(None, 32, 256)	1939712	['input_19[0][0]']
dropout_2 (Dropout)	(None, 2048)	0	['input_18[0][0]']
dropout_3 (Dropout)	(None, 32, 256)	0	['embedding_1[0][0]']
dense_3 (Dense)	(None, 256)	524544	['dropout_2[0][0]']
lstm_1 (LSTM)	(None, 256)	525312	['dropout_3[0][0]']
add_181 (Add)	(None, 256)	0	['dense_3[0][0]', 'lstm_1[0][0]']
dense_4 (Dense)	(None, 256)	65792	['add_181[0][0]']
dense_5 (Dense)	(None, 7577)	1947289	['dense_4[0][0]']

```
==========================================================
 Total params: 5002649 (19.08 MB)
 Trainable params: 5002649 (19.08 MB)
 Non-trainable params: 0 (0.00 Byte)
 _________________________________________________________________

 6000/6000 [==============================] - 1915s 318ms/step - loss:
 4.4984
 6000/6000 [==============================] - 2014s 336ms/step - loss:
 3.6527
 6000/6000 [==============================] - 1574s 262ms/step - loss:
 3.3662
 6000/6000 [==============================] - 1565s 261ms/step - loss:
 3.1909
 6000/6000 [==============================] - 1450s 242ms/step - loss:
 3.0741

#Testing the model
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.applications.xception import Xception, preprocess_input
import pickle

# Load the tokenizer
with open("tokenizer.p", "rb") as tokenizer_file:
    tokenizer = pickle.load(tokenizer_file)
```

```
# Load the model
Model = ad_model("C:\\Users\\Pavan\\Downloads\\python-project-image-caption-
    generator\\models\\model_9.h5")

# Load the Xception model
xception_model = Xception(include_top=False, pooling="avg")

# Function to extract features from an image
def extract_features(filename, model):
    try:
        image = Image.open(filename)
    except:
        print("ERROR: Couldn't open image! Make sure the image path and extension are
    correct")
        return None

    image = image.resize((299, 299))
    image = np.array(image)

    # For images that have 4 channels, convert them into 3 channels
    if image.shape[2] == 4:
        image = image[..., :3]

    image = np.expand_dims(image, axis=0)
    image = preprocess_input(image)
    features = model.predict(image)
    return features

# Function to generate a description for the image
def generate_desc(model, tokenizer, photo, max_length):
    in_text = 'start'
    for i in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
```

```
        sequence = pad_sequences([sequence], maxlen=max_length)
        pred = model.predict([photo, sequence], verbose=0)
        pred = np.argmax(pred)
        word = word_for_id(pred, tokenizer)
        if word is None:
            break
        in_text += ' ' + word
        if word == 'end':
            break
    return in_text

# Function to map an integer to a word
def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None

# Specify the image path here
       img_path="C:\\Users\\Pavan\\Downloads\\python-project-image-caption-
  generator\\Flickr8k_Dataset\\Flicker8k_Dataset\\49553964_cee950f3ba.jpg"
        # Specify the maximum caption length
        max_length = 32

        # Extract features from the image
        photo = extract_features(img_path, xception_model)

        if photo is not None:
# Generate a description for the image
description = generate_desc(model, tokenizer, photo, max_length)
print("\nGenerated Caption:")
print(description)
```

```
# Display the image
img = Image.open(img_path)
plt.imshow(img)
plt.axis('off')
plt.show()
```

```
1/1 [==============================] - 1s 1s/step
Generated Caption:
start man is kayaking in the water end
```

```
#Generate Voice from description
from gtts import gTTS
from playsound import playsound
mytext = description
language = 'en'
filename1 = "voice0003.mp3"
myobj = gTTS(text=mytext, lang=language, slow=False)
myobj.save(filename1)
playsound(filename1,block=False)
```

4.4 OUTPUT SCREENS

Fig 4.6 Caption Generated For Input Image

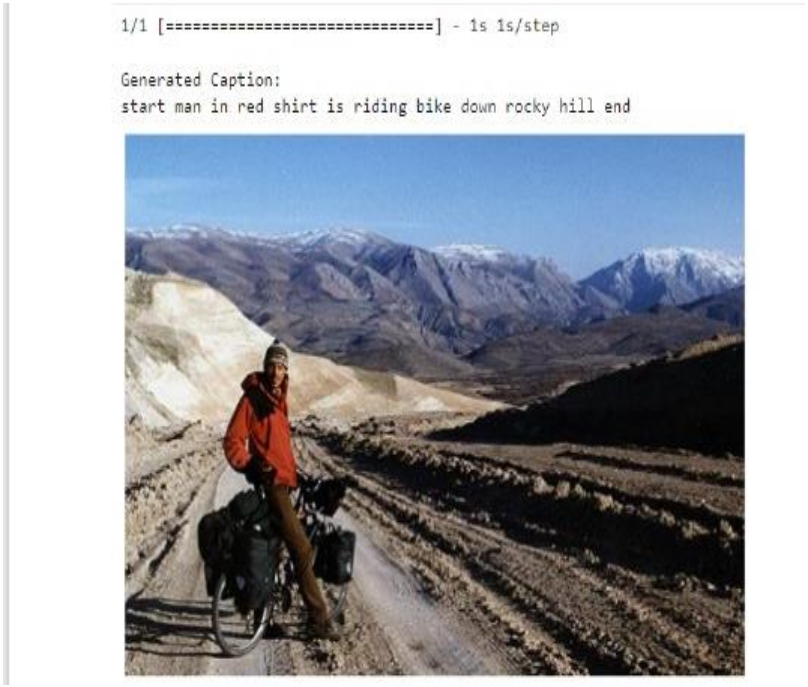

Fig 4.7 Caption Generated for Another Input Image

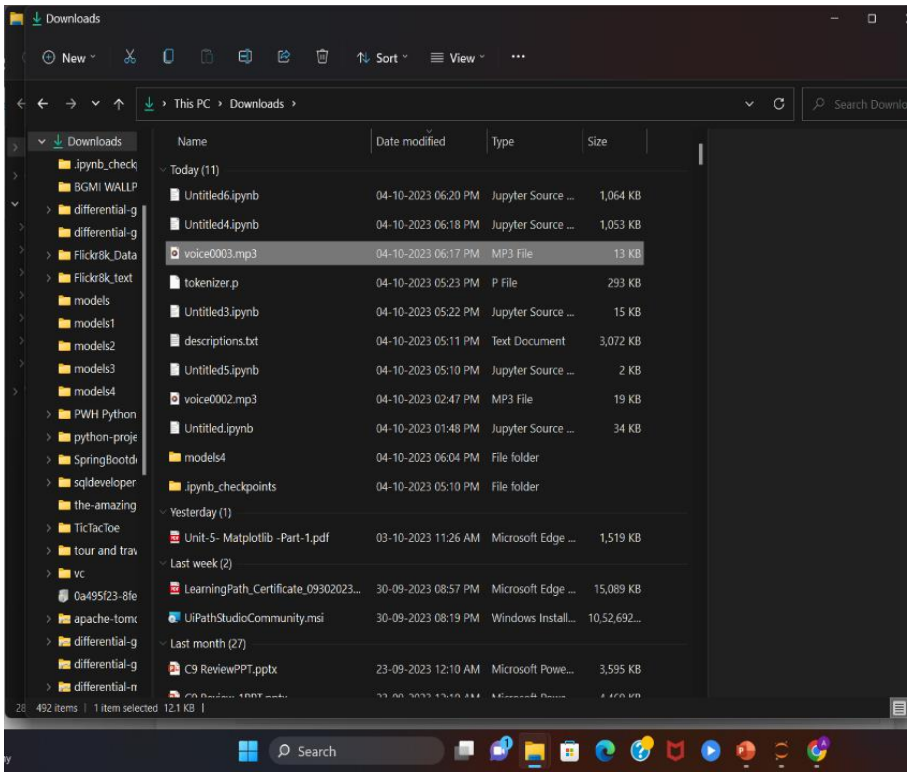

Fig 4.8 Audio file generated

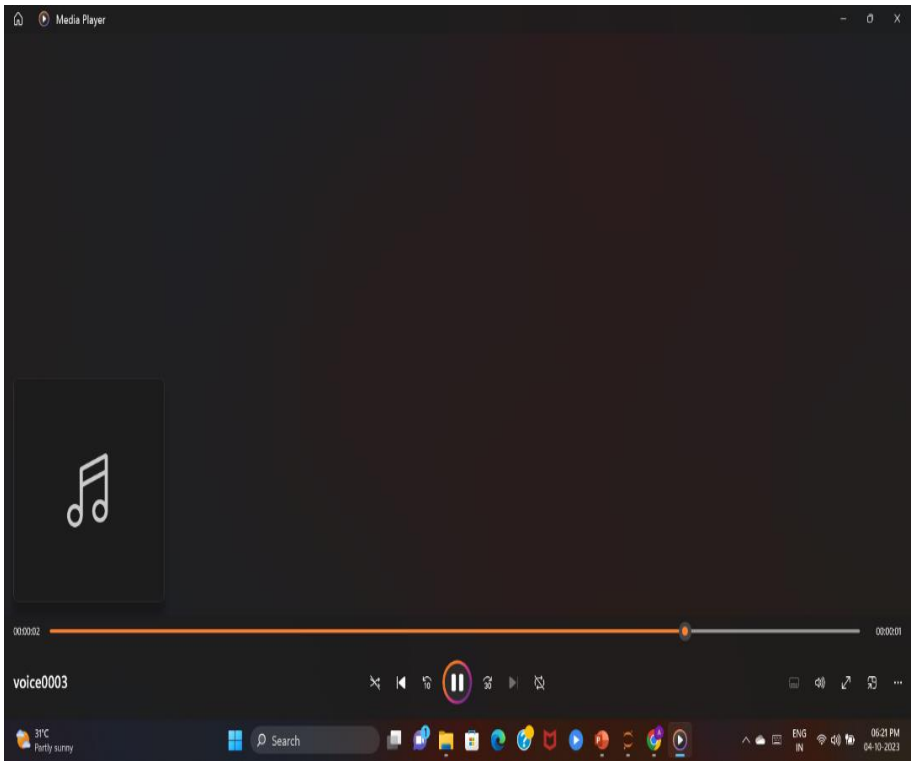

Fig 4.9 Playing the Generated Audio form of Caption

Chapter V

System Testing

5.1 PURPOSE OF TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests; each test type addresses a specific testing requirement. Testing and test design are parts of quality assurance that should also focus on bug prevention. A prevented bug is better than a detected and corrected bug. Testing consumes at least half of the time and work required to produce a functional program. History reveals that even well written programs still have 1-3 bugs per hundred statements. Testing is the process of executing a program with the aim of finding errors. To make our software perform well it should be error-free. If testing is done successfully, it will remove all the errors from the software.

5.2 TESTING STRATEGIES

In order to uncover the errors, present in different phases we have the concept of levels of testing. The Software testing has a prescribed order with the following list of software testing categories arranged in chronological order for our project testing and to generate test cases for output. These are the steps taken to fully test new software in preparation for marketing it.

Types of testing :

- ***Unit testing*** performed on each module or block of code during development. Unit Testing is normally done by the programmer who writes the code.
- ***System testing*** done by a professional testing agent on the completed software product before it is introduced to the market.

5.2.1 UNIT TESTING

Unit testing focuses verification effort on the smallest unit of software i.e., the module. Using the detailed design and the process specifications testing is done to uncover errors within the boundary of the module. All modules must be successful in the unit test before the start of the integration testing begins. It has been seen that each activity class runs after its development using unit testing.

Unit testing is performed at the first stage of testing as it is performed first of all other testing processes. Unit testing is also known as white box testing. So it's generally performed by developers. In our Project Unit testing for a captioning model involves verifying its components, including image input processing, feature extraction, caption generation, and vocabulary handling.

It also tests loss functions, hyperparameter tuning, and low-level components such as RNN layers. The goal is to ensure the model's internal logic and functionality. .

```
In [10]: # train our model
         print('Dataset: ', len(train_imgs))
         print('Descriptions: train=', len(train_descriptions))
         print('Photos: train=', len(train_features))
         print('Vocabulary Size:', vocab_size)
         print('Description Length: ', max_length)

         model = define_model(vocab_size, max_length)
         epochs = 2
         steps = len(train_descriptions)
         # making a directory models to save our models
         os.mkdir("models4")
         for i in range(epochs):
             generator = data_generator(train_descriptions, train_features, tokenizer, max_length)
             model.fit(generator, epochs=1, steps_per_epoch= steps, verbose=1)
             model.save("models4/model_" + str(i) + ".h5")

         Dataset:  6000
         Descriptions: train= 6000
         Photos: train= 6000
         Vocabulary Size: 7577
         Description Length:  32
         Model: "model_1"
         _________________________________________________________________________________________
```

Fig 5.2 Unit Testing

5.2.2 SYSTEM TESTING

System Testing is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications. Usually, the software is only one element of a larger computer-based system. Ultimately, the software is interfaced with other software/hardware systems. System Testing is actually a series of different tests whose sole purpose is to exercise the full computer-based system.

System Testing for our Project involves end-to-end evaluation, usability and performance checks, and ensuring security, compatibility and scalability. It also includes user .It also includes user acceptance testing, accessibility validation, and adherence to regulations, while ensuring data integrity and error handling. The goal is to verify the system's functionality, usability, and reliability in a real-world context.

User acceptance testing is conducted to validate the system against user expectations while regression testing ensures new changes don’t introduce issues. Deployment and rollback procedures are also tested to ensure smooth updates and potential issue recovery.

CHAPTER VI

CONCLUSION AND FUTURE SCOPE

6.1 CONCLUSION

In conclusion, this project successfully enhances visual understanding and accessibility through an accurate image caption generator and clear voice synthesis. User testing confirmed inclusivity and user-friendliness. Future work involves expanding datasets, real-time captioning, multilingual support, advanced accessibility features, and model improvements, ensuring the project remains at the forefront of image captioning and accessibility technology. This project bridges the gap between visual content and understanding, making multimedia accessible to a global audience while contributing to accessible technology advancement.

6.2 FUTURE SCOPE

In this project's future work, we plan to expand the image dataset to enhance caption accuracy and diversity. Real-time image captioning will enable users to upload or capture images on the spot, promoting dynamic caption generation. Multilingual support is on the horizon, making captions accessible in various languages. Advanced accessibility features will be explored to meet diverse user needs, and continuous model improvement will ensure our technology remains up-to-date with the latest advancements, bolstering caption quality and accuracy. These initiatives aim to keep our project at the forefront of accessible image captioning technology.

APPENDIX

REFERENCES

[1] Panicker, M. J., Upadhayay, V., Sethi, G., & Mathur, V. (2021). Image caption generator. International Journal of Innovative Technology and Exploring Engineering (IJITEE), 10(3).

[2] Raypurkar, M., Supe, A., Bhumkar, P., Borse, P., & Sayyad, S. (2021). Deep learning based image caption generator. International Research Journal of Engineering and Technology (IRJET), 8(03).

[3] Wang, H., Zhang, Y., & Yu, X. (2020). An overview of image caption generation methods. Computational intelligence and neuroscience, 2020.

[4] Krishnakumar, B., Kousalya, K., Gokul, S., Karthikeyan, R., & Kaviyarasu, D. (2020). Image caption generator using deep learning. International Journal of Advanced Science and Technology, 29(3s), 975-980.

[5] Han, S. H., & Choi, H. J. (2020, February). Domain-specific image caption generator with semantic ontology. In 2020 IEEE International Conference on Big Data and Smart Computing (BigComp) (pp. 526-530). IEEE.

[6] Kamal, A. H., Jishan, M. A., & Mansoor, N. (2020, November). Textmage: The automated bangla caption generator based on deep learning. In 2020 International Conference on Decision Aid Sciences and Application (DASA) (pp. 822-826). IEEE.

[7] Shinde, V. D., Dave, M. P., Singh, A. M., & Dubey, A. C. (2020). Image caption generator using big data and machine learning. International Research Journal of Engineering and Technology (IRJET), 7(04).

[8] Kumar, N. K., Vigneswari, D., Mohan, A., Laxman, K., & Yuvaraj, J. (2019, March). Detection and recognition of objects in image caption generator system: A deep learning approach. In 2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS) (pp. 107-109). IEEE.

[9] Sharma, G., Kalena, P., Malde, N., Nair, A., & Parkar, S. (2019, April). Visual image caption generator using deep learning. In 2nd international conference on advances in Science & Technology (ICAST).

[10] Kesavan, V., Muley, V., & Kolhekar, M. (2019, October). Deep learning based automatic image caption generation. In 2019 Global Conference for Advancement in Technology (GCAT) (pp. 1-6). IEEE.

[11] https://ieeexplore.ieee.org/document/8276124

[12]https://www.analyticsvidhya.com/blog/2018/04/solving-an-image-captioning-task-using-deep-learning/