

VISUAL AUDIO FOR VISIONLESS USING DEEP LEARNING

A PROJECT REPORT

Submitted by

PREETHI.R

RENUKA.G

VEERAPAREDDY DIVYA

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING



PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

APRIL 2023

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report **“VISUAL AUDIO FOR VISIONLESS USING DEEP LEARNING”** is the bonafide work of **“PREETHI.R (211419104202), RENUKA.G (211419104220), VEERAPAREDDY DIVYA (211419104298)”** who carried out the project work under my supervision.

SIGNATURE

**Dr.L.JABASHEELA, M.E., Ph. D.,
HEAD OF THE DEPARTMENT**

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NASARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

SIGNATURE

**Mrs.D.JENNIFER, M.E., (Ph.D.),
SUPERVISOR
ASSISTANT PROFESSOR**

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NASARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

Certified that the above candidates were examined in the End Semester Project Viva-Voce Examination held on 11/04/2023

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION BY THE STUDENT

We PREETHI.R (211419104202), RENUKA.G (211419104220), VEERAPAREDDY DIVYA (211419104298) hereby declare that this project report titled “**VISUAL AUDIO FOR VISIONLESS USING DEEP LEARNING**”, under the guidance of Mrs.D.JENNIFER, M.E., (Ph.D.) is the original work done by us and we have not plagiarized or submitted to any other degree in any university by us.

1. PREETHI.R

2. RENUKA.G

3. VEERAPAREDDY DIVYA

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our respected Secretary and Correspondent **Dr.P.CHINNADURAI, M.A., Ph.D.**, for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We express our sincere thanks to our beloved Directors **Tmt.C.VIJAYARAJESWARI, Dr.C.SAKTHI KUMAR, M.E., Ph.D.** and **Dr.SARANYASREE SAKTHI KUMAR B.E., M.B.A., Ph.D.**, for providing us with the necessary facilities to undertake this project.

We also express our gratitude to our Principal **Dr.K.MANI, M.E., Ph.D.**, who facilitated us in completing the project.

We thank the Head of the CSE Department, **Dr.L.JABASHEELA , M.E., Ph.D.**, for the support extended throughout the project.

We would like to thank my Project Guide **Mrs.D.JENNIFER, M.E., (Ph.D.)** and all the faculty members of the Department of CSE for their advice and encouragement for the successful completion of the project.

PREETHI.R

RENUKA.G

VEERAPAREDDY DIVYA

ABSTRACT

The sense of sight is incredibly important for human beings, and its absence can significantly impact an individual's quality of life. Blindness can limit an individual's ability to perform everyday tasks independently, including navigating unfamiliar environments, identifying objects, and reading. According to the World Health Organization (WHO), approximately 285 million people worldwide are visually impaired, with 39 million of them being completely blind. Visual audio for Visionless combines the concept of image captioning and TTS that involves converting an image into textual and audio description using deep learning. This task is challenging as it requires the system to understand the content of an image and generate natural language captions that accurately describes the image content in both spoken and written form. In the proposed system, CNN (Inception V3) serves as the encoder and LSTM with attention serves as the decoder and generates the caption word by word, conditioning the prediction on the previous generated. Inception V3 and LSTM for image captioning can achieve good results due to the complementary strengths of these two models. Ultimately, captions and audio description for images in real-time are generated as soon as the system captures the scenario as an image.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	i
	LIST OF TABLES	iv
	LIST OF FIGURES	v
	LIST OF ABBREVIATIONS	vi
1.	INTRODUCTION	
	1.1 Overview	1
	1.2 Problem Definition	2
2.	LITERATURE SURVEY	3
3.	SYSTEM ANALYSIS	
	3.1 Existing System	5
	3.2 Proposed system	6
	3.3 Feasibility Study	7
	3.4 Hardware Environment	9
	3.5 Software Environment	9
4.	SYSTEM DESIGN	
	4.1 ER diagram	10
	4.2 Data dictionary	10
	4.3 Data Flow Diagram	11
	4.4 UML Diagrams	12

CHAPTER NO.	TITLE	PAGE NO.
5.	SYSTEM ARCHITECTURE	
	5.1 Module Design Specification	17
	5.2 Algorithms	26
6.	SYSTEM IMPLEMENTATION	
	6.1 Client-side coding	33
	6.2 Server-side coding	40
7.	PERFORMANCE ANALYSIS	
	7.1 Results & Discussion	53
	7.2 Test Cases & Reports	54
8.	CONCLUSION	
	8.1 Conclusion and Future Enhancements	56
	APPENDICES	
	A.1 Sample Screens	57
	REFERENCES	58

LIST OF TABLES

TABLE NO.	TABLE TITLE	PAGE NO.
4.1	Dataset Details	10
5.1	Data points corresponding to one image and its caption.	18
5.2	Word Prediction Generation Step by Step	24

LIST OF FIGURES

FIGURE NO.	FIGURE TITLE	PAGE NO.
4.1	ER Diagram	10
4.2	Data Flow Diagram Level 0	10
4.3	Data Flow Diagram Level 2	11
4.4	Use Case Diagram	13
4.5	Class Diagram	14
4.6	Activity Diagram	14
4.7	Sequence Diagram	15
4.8	Collaboration Diagram	15
5.1	Architecture Diagram	16
5.2	Flickr Data Set text format	20
5.3	Flickr Dataset Python File	21
5.4	Description of Images	22
5.5	Final Model Structure	25
5.6	Convolutional Neural Network	27
5.7	Working of Inception v3	29
5.8	Max Pooling Layer	30
5.9	A fully connected layer in a deep network	31
5.10	A multilayer deep fully connected network	32
7.1	Epoch Vs Accuracy	53

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
CNN	Convolutional Neural Network
R-CNN	Region-based Convolutional Neural Network
RNN	Recurrent Neural Network
GPU	Graphics processing unit
TPU	Tensor Processing Units
RODe	Region Based Object Detection
LSTM	Long short-term memory
VGG16	Visual Geometry Group
COCO	Common Object in Context
MSCOCO	Microsoft Common Objects in Context
UML	Unified Modeling Language
DFD	Data Flow Diagram
TTS	Text-To-Speech

CHAPTER 1

INTRODUCTION

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

Image captioning is an interdisciplinary field that combines computer vision and natural language processing. The goal of this project is to generate a human-like descriptive and audible sentence for an image, which requires both understanding the content of the image and generating coherent and grammatical language that accurately describes the content. Image captioning has numerous applications, including improving accessibility for the visually impaired, accessing information and improving the retrieval of images from large databases.

Real-time image captioning is a variant of image captioning where the goal is to generate captions as soon as the image is input into the system. To achieve real-time performance, the image captioning system must be optimized for speed, with efficient algorithms, efficient hardware, and fast data access. This can be achieved by reducing the size of the model, using efficient algorithms, such as quantization and pruning, and deploying the model on fast hardware, such as GPUs or TPUs.

In conclusion, Image captioning is a challenging problem that requires understanding the content of images and generating natural language that accurately describes the content. Combining CNNs and RNNs, or using Attention-based Transformers, is a common approach to image captioning. Real-time image captioning requires additional optimization for speed to generate captions as soon as the image is input into the system. With continued advancements in deep learning, computer vision, and natural language processing, image captioning is combined with audio generation to provide great support for visually impaired people.

1.2 PROBLEM DEFINITION

Image captioning is the process of generating a natural language description of an image. It has numerous applications, including helping visually impaired people understand images and assisting in automated image analysis. However, conventional image captioning systems rely on textual outputs, which are not suitable for visually impaired individuals. Therefore, there is a need to develop an image captioning system with an audio output to enhance accessibility.

The proposed system should be able to analyse an image and generate a descriptive caption that accurately represents the content of the image. The system should then convert the caption into an audio output, making it accessible to visually impaired individuals. The audio output should be clear and easy to understand, conveying the same information as the textual output.

The system should be able to handle a wide range of image types, including photographs, drawings, and paintings, and generate accurate and descriptive captions for each image. It should also be capable of processing images in real-time, enabling users to receive near-instantaneous audio feedback.

Overall, the aim of this project is to develop a system that can generate accurate and descriptive captions for images and convert them into an audio output, enhancing accessibility for visually impaired individuals and providing an efficient and reliable solution for automated image analysis.

CHAPTER 2

LITERATURE

SURVEY

CHAPTER 2

LITERATURE SURVEY

A. K. Poddar et al, [2] in their research, proposed a region based approach for captioning images. The Image Captioning is implemented by the objects detected using Region Based Object Detection (RODe). In the region-based approach, the image is divided into regions for Object Detection. The specific technique used by authors is Region Based CNN or R-CNN. The research work includes other techniques such as Feature Extraction and Scene Classification that make use of CNN. The RNN technique is used for the sentence generation with the help of objects and scene attributes and image features.

A. Salaberria et al, [3] focuses on better sentence learning using Deep Convolutional Network. A deep Fisher Kernel method is used for image feature extraction. The extracted activations are aggregated to form a Fisher Vector. Instead of LSTM, this approach uses gLSTM. Since the information of the input image is only provided in the first step, the input image information gets diluted as it proceeds. gLSTM provides a piece of additional information called 'guide', that pertains to the input image information throughout the process.

G. Geetha et al, [5] proposed a model that makes use of LSTM instead of simple RNN. The research implemented Feature Extraction using the pre-trained VGG16 model. Since RNN cannot remember the previously predicted words, LSTM can be used as LSTM can remember the words for a longer period of time. LSTM is a cell that stores the words till the caption is generated properly. According to this, the results observed in categories such as descriptions with errors, without errors, related and unrelated captions. These categories are due to considerations of the neighborhood of words.

J. Wang et al, [7] proposed an approach which enables the use of LSTM with Read-Only LSTM. LSTM cell is a storage mechanism that provides the ability to store the intermediate words while the read-only LSTM cell provides image features. The image content is provided only in the first step thus, the predicted word may or may not be related to the image. to relate both previous and current word, an additional unit is required in order to predict current word related to image content. The additional unit produced is called LSTM with Read-Only Unit that results in better accuracy.

I. Puthige et al, [6] proposed a Word Level Attention model for Image Captioning. The model has a word-level attention layer that is produced to process image features with two models for word prediction accurately. The two modes are Line Level Bidirectional spatial embedding used for feature maps and the Attention mode to extract word-level attention. It also used Bidirectional LSTM networks that process words in both the directions i.e., forward and backward. The Word Level Attention Extraction used to extract visual information to predict the next word using a softmax activation function.

M. A. Al-Malla, [8] proposed an image captioning model that uses attention and object features to mimic human image understanding. The model incorporates object detection, caption generation modules and leverages attention mechanisms to selectively focus on important image regions during captioning. The proposed model achieves state-of-the-art performance on the popular COCO dataset, demonstrating its effectiveness in generating accurate captions. The study contributes to the growing body of research on image captioning, and highlights the potential of incorporating attention and object features in such models to improve their performance.

CHAPTER 3

SYSTEM

ANALYSIS

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Image captioning with transformers involves using a transformer-based architecture to generate captions for images. The transformer model first encodes the input image using a convolutional neural network (CNN) to extract relevant visual features, and then feeds these features into a transformer encoder to produce a sequence of hidden states. The decoder then generates a sequence of words, using an attention mechanism to focus on different parts of the input image while generating the caption.

Image captioning can also be implemented using Inception v2 which is a convolutional neural network (CNN) architecture that was introduced in 2016 as an improvement over the original Inception architecture. To perform image captioning using Inception v2, you can use a combination of the CNN and a language model. The CNN extracts feature from the image, while the language model generates the text.

Disadvantage

However, there are also some disadvantages to using transformers for image captioning. One major disadvantage is that transformers can be computationally expensive, especially when dealing with large image datasets. Another potential disadvantage of using transformers for image captioning is that they require a large amount of training data to achieve good performance. This is because the transformer model must learn to capture complex relationships between the visual and textual information in the image, which can be difficult without enough training data.

3.2 PROPOSED SYSTEM

Image captioning systems using Inception v3, CNN is used to extract visual features from the input image, which are then fed into a language model to generate the caption. The process of image captioning with Inception v3 involves several steps. First, the input image is pre-processed to a fixed size and normalized to zero mean and unit variance. This ensures that the image has a consistent format and makes it easier for the CNN to extract relevant visual features.

Next, the pre-processed image is fed into the Inception v3 CNN, which extracts visual features from the image. The CNN uses a combination of convolutional and pooling layers to capture different levels of abstraction of the image. The output of the CNN is a set of feature maps that represent the visual information in the image. The feature maps are then flattened and fed into a language model, which generates the caption. The language model takes the visual features as input and generates a sequence of words that describe the image. The sequence of words can be optimized using a loss function that compares the generated captions to ground truth captions.

One advantage of using Inception v3 for image captioning is its ability to extract high-level features from images. The architecture is designed to capture a wide range of visual features, from low-level details to high-level features such as object categories and scene contexts. This makes it well-suited for image captioning tasks, which require a comprehensive understanding of the visual information in the image. Another advantage of using Inception v3 is its computational efficiency. The architecture is relatively lightweight compared to other CNN architectures, such as ResNet or VGG, which makes it faster and more memory efficient. This can be particularly advantageous when dealing with large datasets or deploying the model on resource-constrained devices.

3.3 FEASIBILITY STUDY

The objective of feasibility study is not only to solve the problem but also to acquire a sense of its scope. During the study, the problem definition was crystallized and aspects of the problem to be included in the system are determined. Consequently, benefits are estimated with greater accuracy at this stage. The key considerations are:

Economic feasibility

Technical feasibility

Social feasibility

Economic Feasibility

Economic feasibility studies not only the cost of hardware, software is included but also the benefits in the form of reduced costs are considered here. This project, if installed will certainly be beneficial since there will be reduction in manual work and increase in the speed of work.

Total number of lines of code (LOC) = 424

KLOC = $424/1000 = 0.424$

Effort = $2.4 * (0.424)^{1.05} = 0.975$ person-months

Development time = $2.5 * (0.975)^{0.38} = 2.47$ months

Average staff size = $0.975/2.47 = 0.412$ persons

Productivity = $0.424/0.975 = 0.434$ KLOC/person-months

P = 434 LOC/person-months

Technical Feasibility

Technical feasibility evaluates the hardware requirements, software technology, available personnel etc., as per the requirements it provides sufficient memory to hold and process.

- 1) Deep Learning algorithm - CNN
- 2) Artificial Intelligence - LSTM
- 3) Google Drive
- 4) IDE: Google Colab

Social Feasibility

Social feasibility is a detailed study on how one interacts with others within a system or an organization. Social impact analysis is an exercise aimed at identifying and analysing such impacts in order to understand the scale and reach of the project's social impacts. Social impact analysis greatly reduces the overall risks of the project, as it helps to reduce resistance, strengthens general support, and allows for a more comprehensive understanding of the costs and benefits of the project. The social application of the project is that it helps the visually impaired people in the following ways:

- 1) Assistance for Visually Impaired
- 2) Accessing information
- 3) Mobility and safety
- 4) Social interaction
- 5) Employment
- 6) Education

3.4 HARDWARE REQUIREMENT

Processor: Intel Core i5

RAM: 512 MB and above

Hard Disk: 40 GB and above

3.5 SOFTWARE REQUIREMENT

Backend: PYTHON

Frontend: TKINTER

Technology: Deep Learning

Operating System: Windows 7

Tools: Anaconda Navigator /TensorFlow/Jupyter/Google laboratory

CHAPTER 4

SYSTEM

DESIGN

CHAPTER 4

SYSTEM DESIGN

4.1. ER DIAGRAM

The following figure shows the Entity-Relationship diagram of the project.

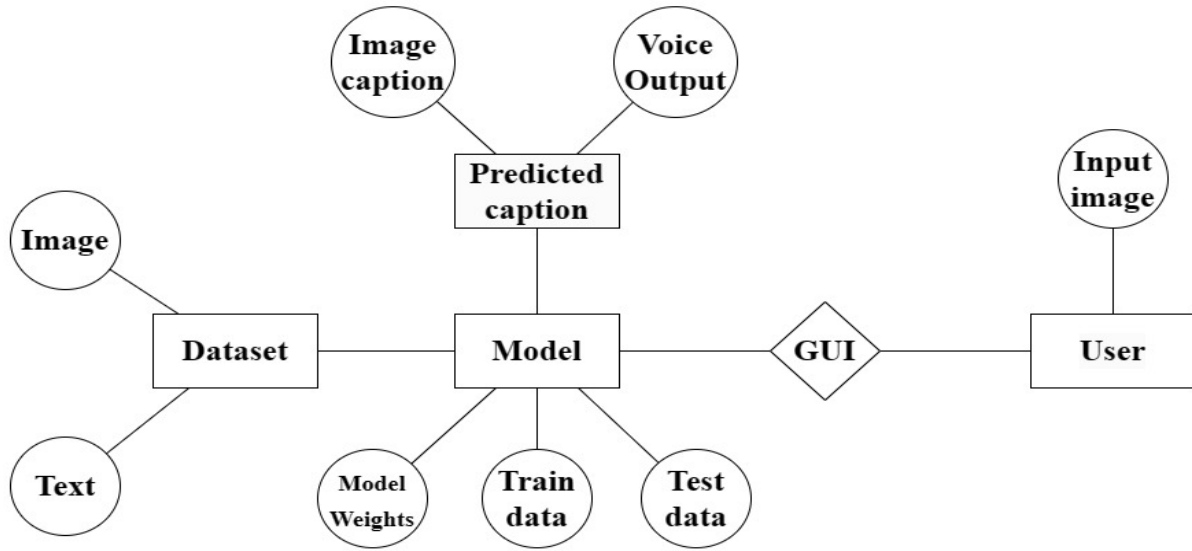


Fig 4.1. ER Diagram

4.2 DATA DICTIONARY

The Flickr30k dataset is a collection of 31,783 images and associated captions that are commonly used for training and evaluating image captioning models. The images were selected from the larger Flickr dataset, which is a collection of images shared on the Flickr photo-sharing website. Each image in the Flickr30k dataset is accompanied by five different captions, which were collected by asking human annotators to describe the contents of the image.

The dataset was released in 2014 and has been used extensively in research on image captioning, natural language processing, and computer vision. Overall, the Flickr30k dataset is a valuable resource for researchers and

practitioners interested in developing and evaluating image captioning models, as well as for those interested in exploring the intersection of computer vision and natural language processing.

Features of the dataset making it suitable for this project are:

- Multiple captions mapped for a single image makes the model generic and avoids overfitting of the model.
- Diverse categories of training images can make the image captioning model to work for multiple categories of images and hence can make the model more robust.

Table 4.1. Dataset Details

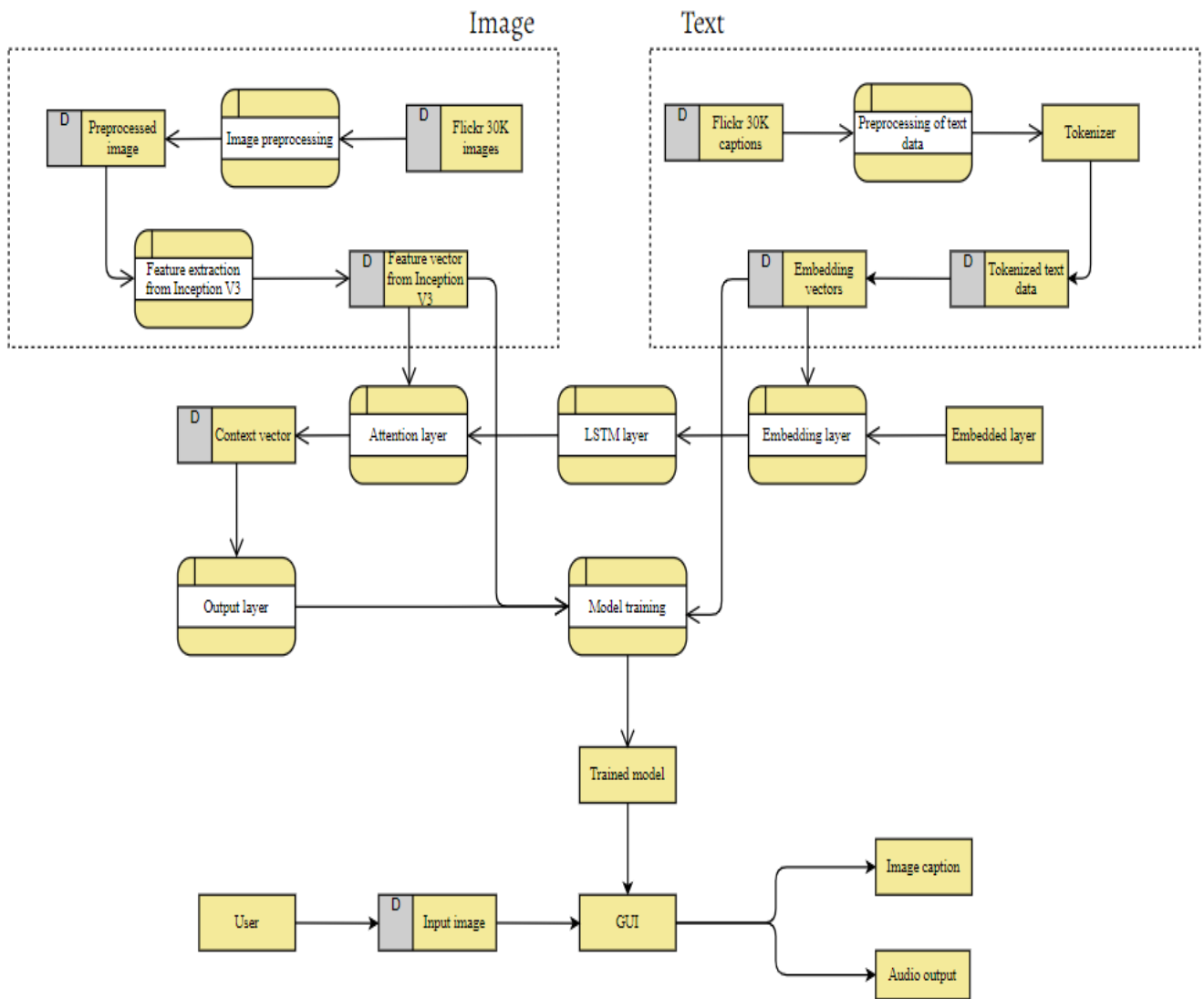
Dataset Name	Size		
	Train	Valid	Test
Flickr8K[19]	6000	1000	1000
Flickr30K[32]	28000	1000	1000
MSCOCO[33]	82783	40504	40775

4.3 DATA FLOW DIAGRAM

The Data Flow Diagram of the system is shown in **Fig 4.3**. This project requires a dataset which have both images and their caption. The dataset should be able to train the image captioning model.



Fig 4.2. Data Flow Diagram Level 0



Similarly, Level 1 DFD diagram includes the training and caption generation process.

Fig. 4.3. Data Flow Diagram Level 2

4.4 UML DIAGRAMS

Unified Modeling Language (UML) is a modeling language used in software engineering to visually represent systems, processes, and architectures. UML diagrams are graphical representations that use standardized symbols and notations to communicate complex systems and concepts. The UML diagrams of the system are as follows.

USE CASE DIAGRAM:

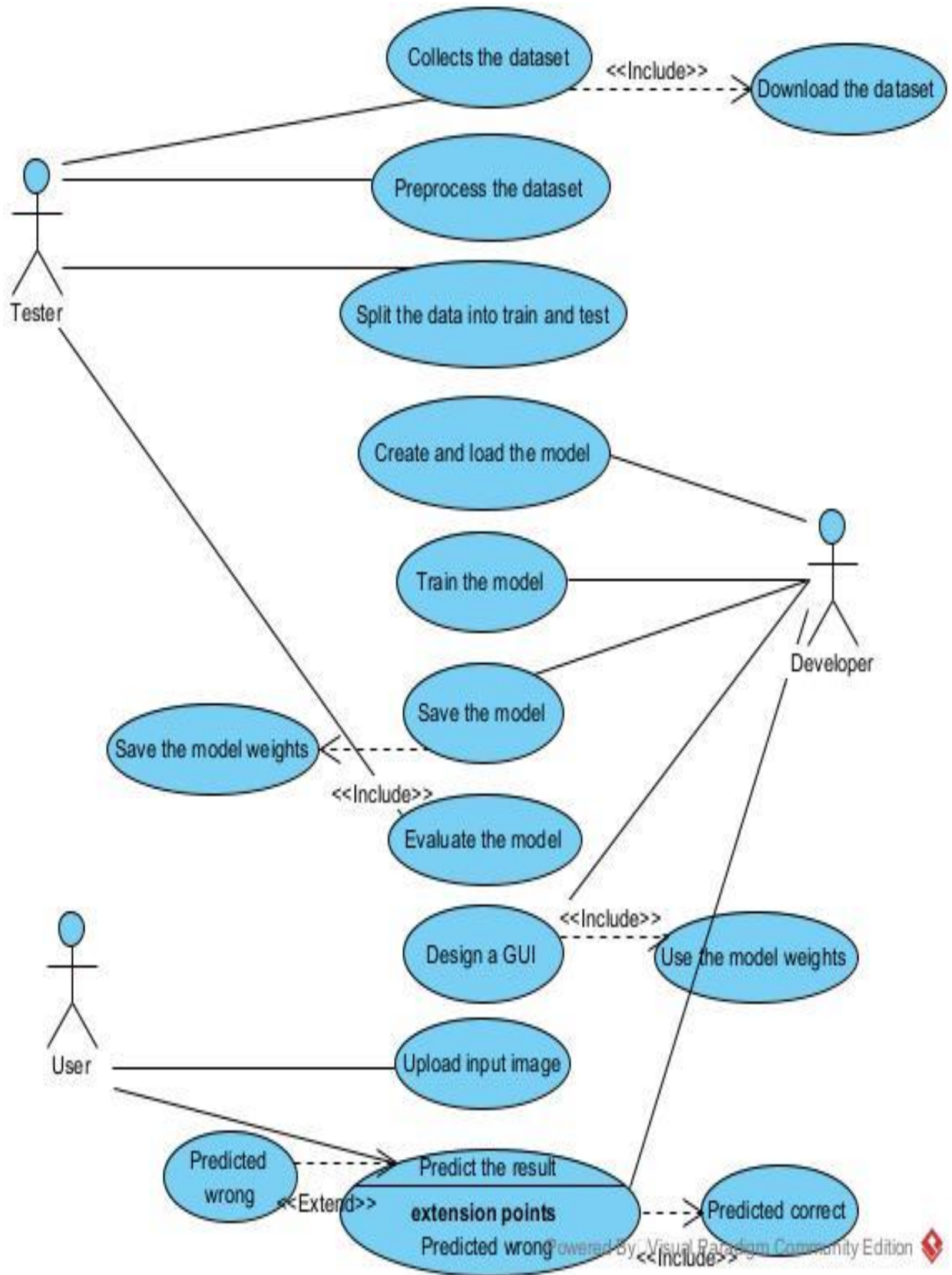


Fig. 4.4. Use Case Diagram

CLASS DIAGRAM:

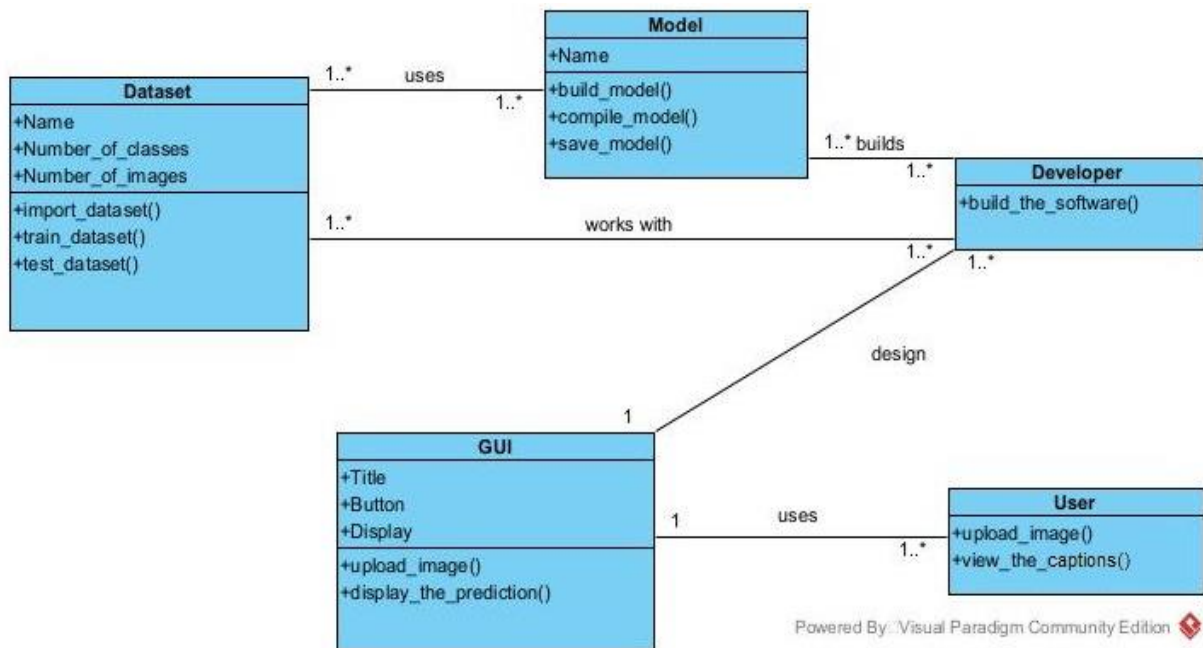


Fig. 4.5. Class Diagram

ACTIVITY DIAGRAM:

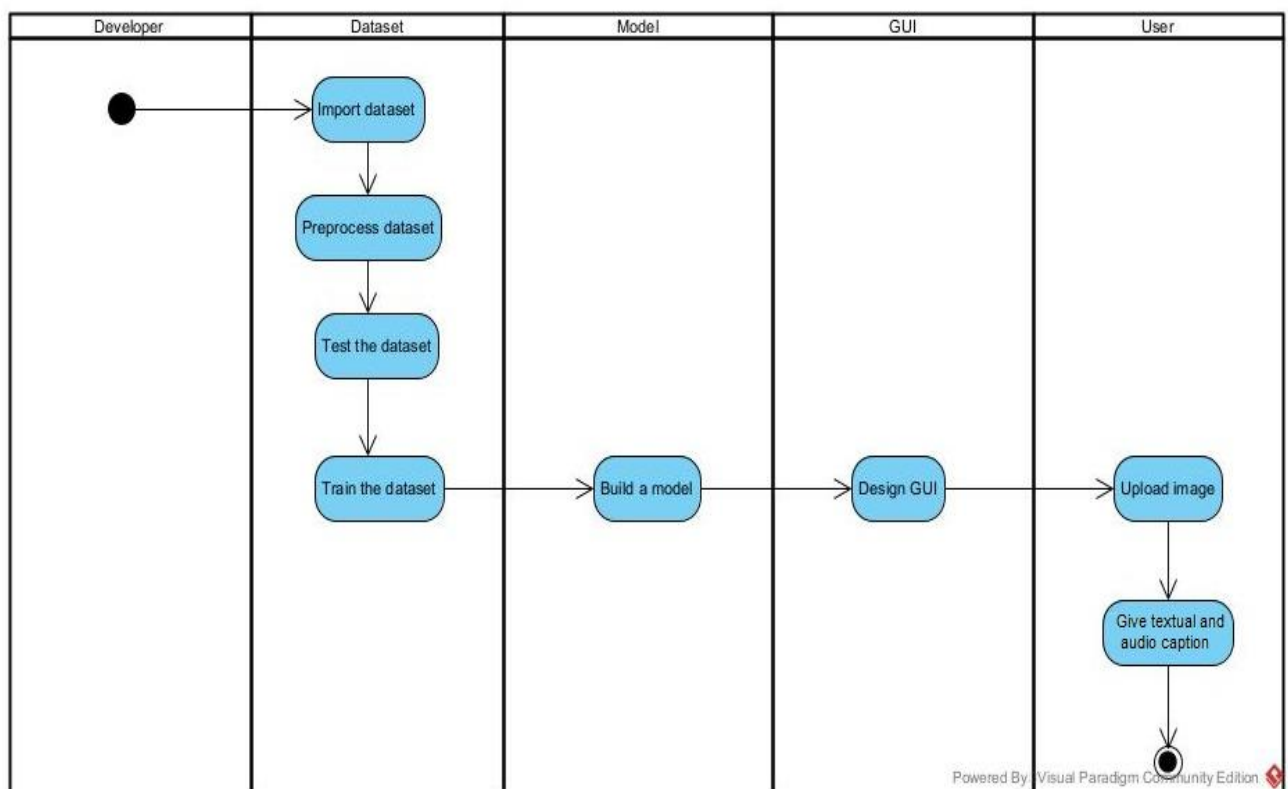


Fig. 4.6. Activity Diagram

SEQUENCE DIAGRAM:

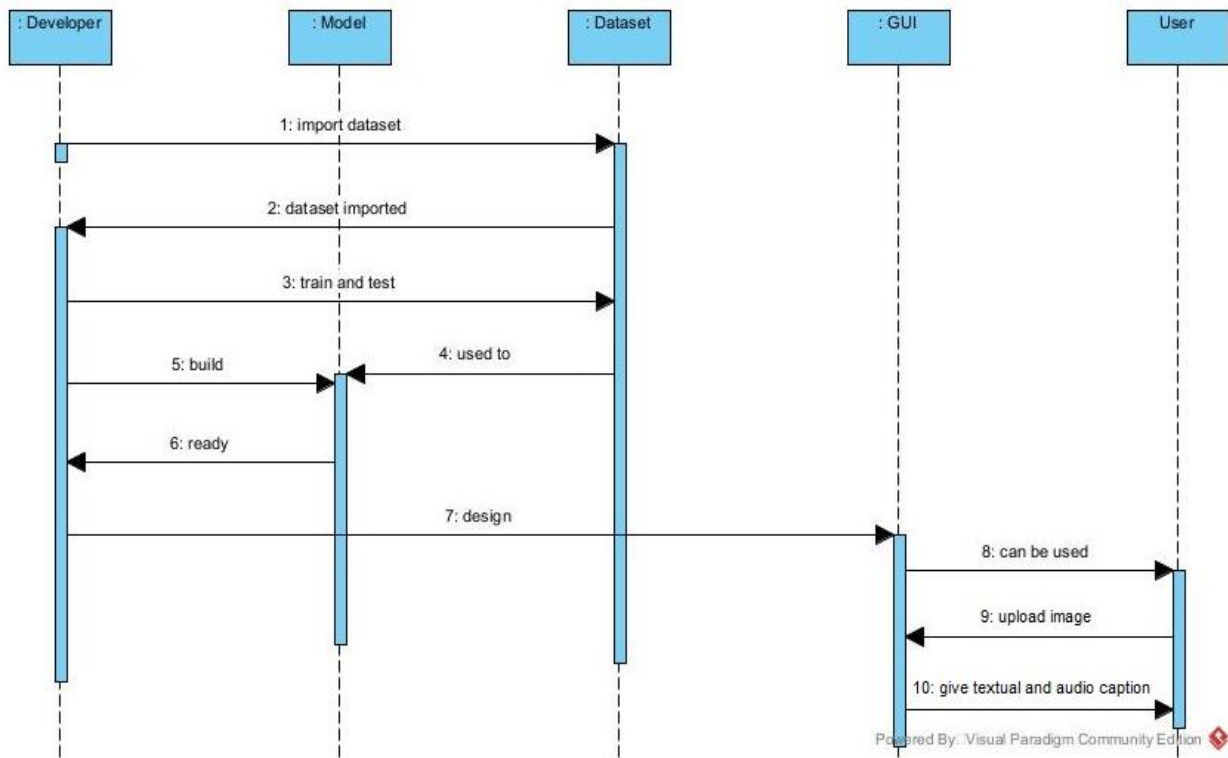


Fig. 4.7. Sequence Diagram

COLLABORATION DIAGRAM:

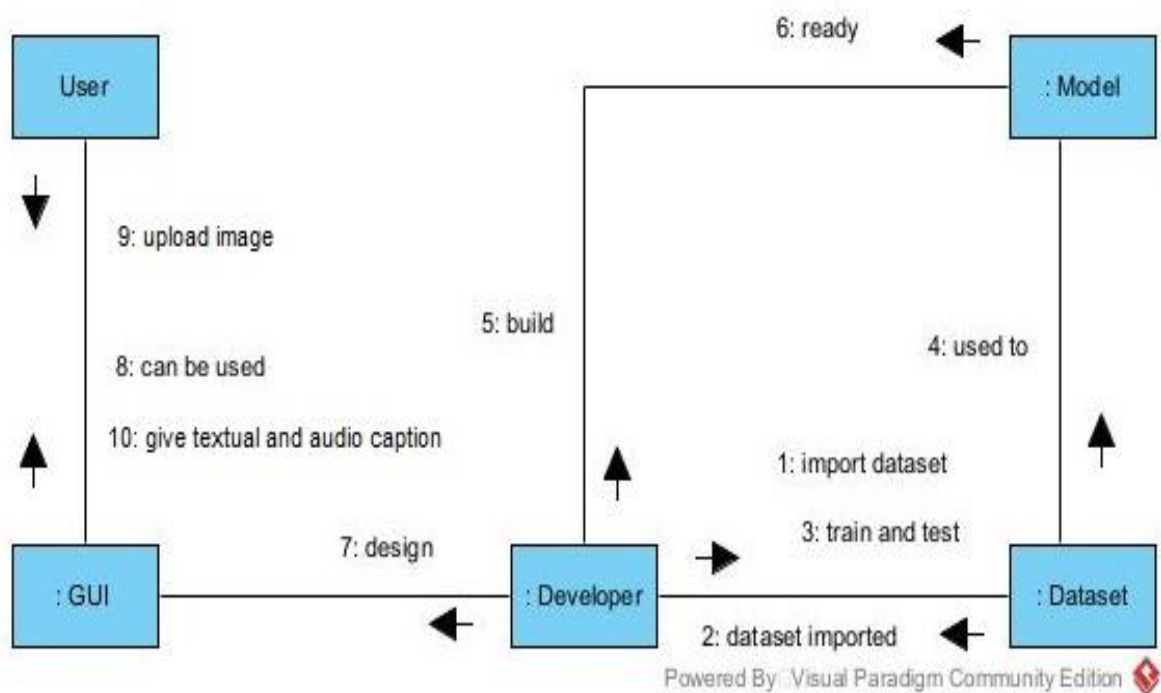


Fig. 4.8. Collaboration Diagram

CHAPTER 5

SYSTEM

ARCHITECTURE

CHAPTER 5

SYSTEM ARCHITECTURE

ARCHITECTURE OVERVIEW

This displays the whole design of our working model including the components and the states occurring during the execution of the process. It displays the very initial process of image feeding followed by the parsing and breaking down of the image into vectors, where all the data regarding the image is stored and fed to the model by using Flickr30k dataset. The CNN is used in the encoding and LSTM is used in the decoding the descriptive data which play the image again and again develop the caption with the help of language processing and data(trained) stored, thus provide generated caption as output which is later converted into audio form using python libraries.

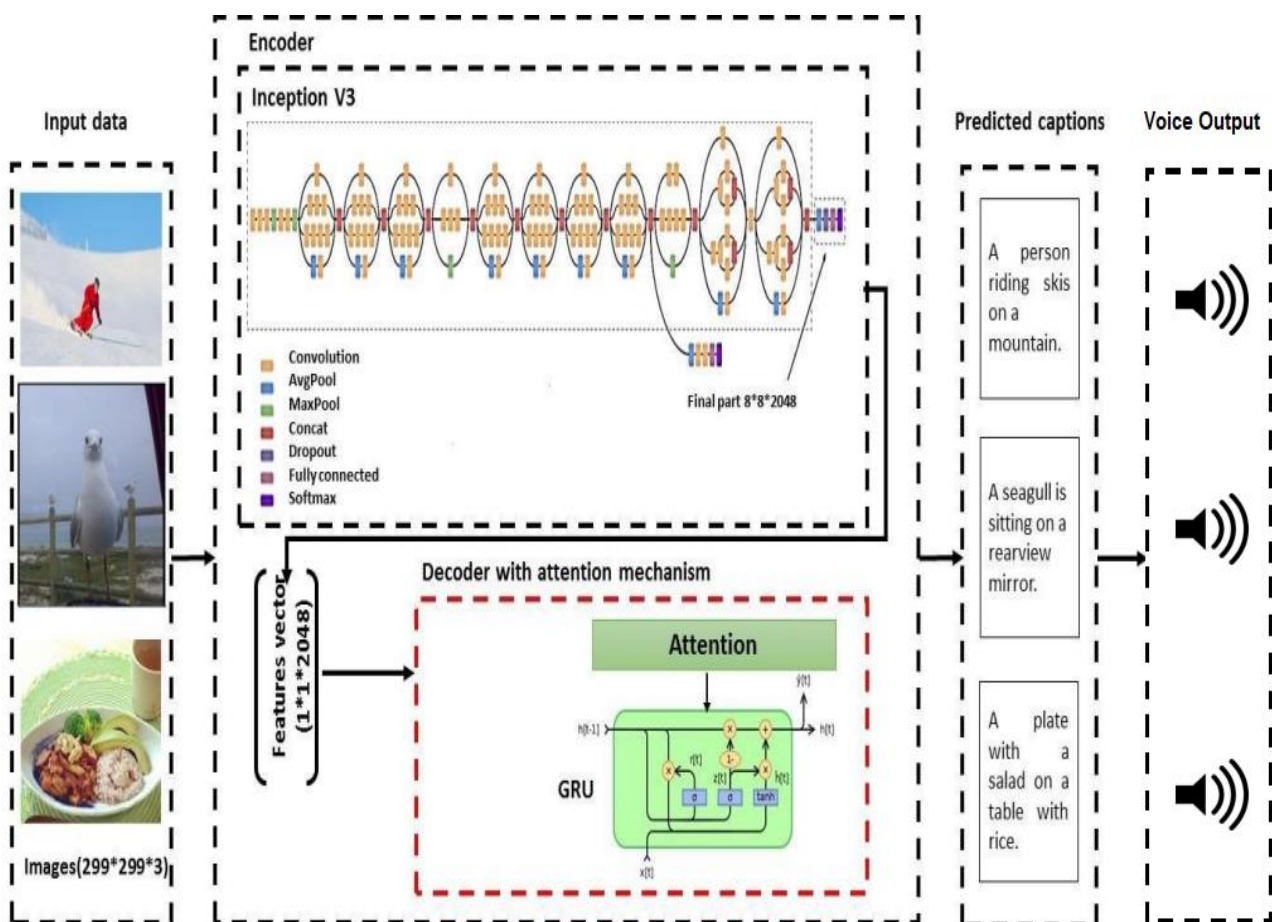


Fig 5.1. Architecture Diagram

5.1 MODULE DESIGN SPECIFICATION

Data Preprocessing — Images

Images are nothing but input (X) to our model. As you may already know that any input to a model must be given in the form of a vector.

We need to convert every image into a fixed sized vector which can then be fed as input to the neural network. For this purpose, we opt for **transfer learning** by using the InceptionV3 model (Convolutional Neural Network) created by Google Research.

This model was trained on Visual Genome Corpus dataset to perform image classification on 1000 different classes of images. However, our purpose here is not to classify the image but just get fixed- length informative vector for each image. This process is called automatic feature engineering.

Data Preprocessing — Captions

Here the task is to predict the captions. So, during the training period, captions will be the target variables (Y) that the model is learning to predict. But the prediction of the entire caption, given the image does not happen at once. Caption is predicted word **by word**. Thus, we need to encode each word into a fixed sized vector.

Data Preparation using Generator Function

Let's take the first image vector **Image_1** and its corresponding caption "**startseq the black cat sat on grass endseq**". Recall that, Image vector is the input and the caption is what we need to predict. But the caption is predicted as follows:

For the first time, the image vector and the first word is given as input and try to predict the second word, i.e.:

Input = Image_1 + 'startseq'; Output = 'the'

Then image vector and the first two words is provided as input and try to predict the third word, i.e.: Input = Image_1 + 'startseq the'; Output = 'cat'

And so on...

Thus, the data matrix for one image and its corresponding caption as follows:

Table 5.1 Data points corresponding to one image and its caption.

	Xi		Yi
i	Image feature vector	Partial Caption	Target word
1	Image_1	startseq	the
2	Image_1	startseq the	black
3	Image_1	startseq the black	cat
4	Image_1	startseq the black cat	sat
5	Image_1	startseq the black cat sat	on
6	Image_1	startseq the black cat sat on	grass
7	Image_1	startseq the black cat sat on grass	endseq

Pre-Requisites

This project requires good knowledge of Deep learning, Python, working on Colab notebooks, Keras library, Numpy, and **Natural language processing**. Make sure you have installed all the following necessary libraries:

- pip install tensorflow
- keras
- pillow
- numpy
- tqdm
- Colab

Project File Structure

Downloaded from dataset:

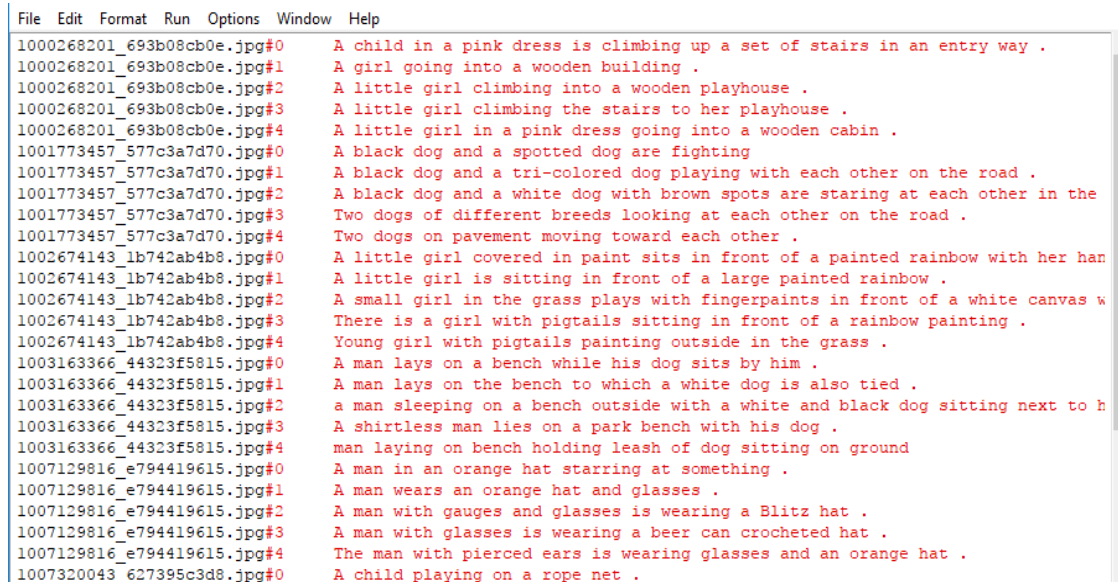
- **Flicker30k_Dataset** – Dataset folder which contains 31,783 images.
- **Flickr_30k_text** – Dataset folder which contains text files and captions of images. The below files will be created by us while making the project.
- **Models** – It will contain our trained models.
- **Descriptions.txt** – This text file contains all image names and their captions after preprocessing.
- **Features.p** – Pickle object that contains an image and their feature vector extracted from the Exception pre-trained CNN model.
- **Tokenizer.p** – Contains tokens mapped with an index value.
- **Model.png** – Visual representation of dimensions of our project.
- **Testing_caption_generator.py** – Python file for generating a caption of any image.
- **Training_caption_generator.ipynb** – Colab notebook in which we train and build our image caption generator.

Building The Python Based Project

Let's start by initializing the Colab notebook server by typing Colab in the console of your project folder. It will open up the interactive Python notebook where you can run your code. Create a Python3 notebook and name it training_caption_generator.ipynb.

Getting And Performing Data Cleaning

The main text file which contains all image captions is **Flickr30k.token** in our **Flickr_30k_text** folder.



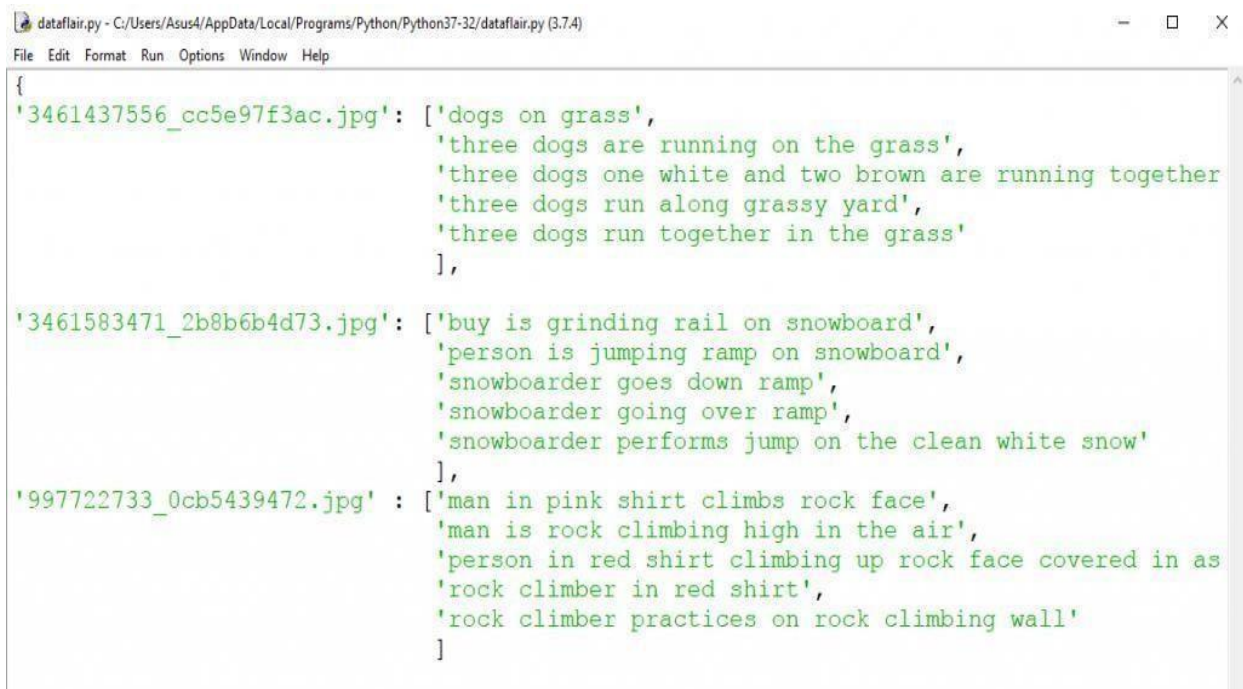
```
File Edit Format Run Options Window Help
1000268201_693b08cb0e.jpg#0 A child in a pink dress is climbing up a set of stairs in an entry way .
1000268201_693b08cb0e.jpg#1 A girl going into a wooden building .
1000268201_693b08cb0e.jpg#2 A little girl climbing into a wooden playhouse .
1000268201_693b08cb0e.jpg#3 A little girl climbing the stairs to her playhouse .
1000268201_693b08cb0e.jpg#4 A little girl in a pink dress going into a wooden cabin .
1001773457_577c3a7d70.jpg#0 A black dog and a spotted dog are fighting
1001773457_577c3a7d70.jpg#1 A black dog and a tri-colored dog playing with each other on the road .
1001773457_577c3a7d70.jpg#2 A black dog and a white dog with brown spots are staring at each other in the
1001773457_577c3a7d70.jpg#3 Two dogs of different breeds looking at each other on the road .
1001773457_577c3a7d70.jpg#4 Two dogs on pavement moving toward each other .
1002674143_1b742ab4b8.jpg#0 A little girl covered in paint sits in front of a painted rainbow with her han
1002674143_1b742ab4b8.jpg#1 A little girl is sitting in front of a large painted rainbow .
1002674143_1b742ab4b8.jpg#2 A small girl in the grass plays with fingerpaints in front of a white canvas w
1002674143_1b742ab4b8.jpg#3 There is a girl with pigtails sitting in front of a rainbow painting .
1002674143_1b742ab4b8.jpg#4 Young girl with pigtails painting outside in the grass .
1003163366_44323f5815.jpg#0 A man lays on a bench while his dog sits by him .
1003163366_44323f5815.jpg#1 A man lays on the bench to which a white dog is also tied .
1003163366_44323f5815.jpg#2 a man sleeping on a bench outside with a white and black dog sitting next to h
1003163366_44323f5815.jpg#3 A shirtless man lies on a park bench with his dog .
1003163366_44323f5815.jpg#4 man laying on bench holding leash of dog sitting on ground
1007129816_e794419615.jpg#0 A man in an orange hat starring at something .
1007129816_e794419615.jpg#1 A man wears an orange hat and glasses .
1007129816_e794419615.jpg#2 A man with gauges and glasses is wearing a Blitz hat .
1007129816_e794419615.jpg#3 A man with glasses is wearing a beer can crocheted hat .
1007129816_e794419615.jpg#4 The man with pierced ears is wearing glasses and an orange hat .
1007320043_627395c3d8.jpg#0 A child playing on a rope net .
```

Fig.5.2. Flickr Data Set text format

The format of our file is image and caption separated by a new line (“\n”). Each image has 5 captions and # (0 to 5) number is assigned for each caption.

Important 5 functions:

- **load_doc(filename)** – For loading the document file and reading the contents inside the file into a string.
- **all_img_captions(filename)** – This function will create a descriptions dictionary that maps images with a list of 5 captions. The descriptions dictionary will look something like the Figure.



```
dataflair.py - C:/Users/Asus4/AppData/Local/Programs/Python/Python37-32/dataflair.py (3.7.4)
File Edit Format Run Options Window Help

{
'3461437556_cc5e97f3ac.jpg': ['dogs on grass',
                              'three dogs are running on the grass',
                              'three dogs one white and two brown are running together',
                              'three dogs run along grassy yard',
                              'three dogs run together in the grass'
                              ],
'3461583471_2b8b6b4d73.jpg': ['buy is grinding rail on snowboard',
                              'person is jumping ramp on snowboard',
                              'snowboarder goes down ramp',
                              'snowboarder going over ramp',
                              'snowboarder performs jump on the clean white snow'
                              ],
'997722733_0cb5439472.jpg' : ['man in pink shirt climbs rock face',
                              'man is rock climbing high in the air',
                              'person in red shirt climbing up rock face covered in as',
                              'rock climber in red shirt',
                              'rock climber practices on rock climbing wall'
                              ]
}
```

Fig .5.3. Flickr Dataset Python File

- **cleaning_text(descriptions)** – This function takes all descriptions and performs data cleaning. This is an important step to work with textual data and decide what type of cleaning to perform on the text. Here, we will be removing punctuations, converting all text to lowercase and removing words that contain numbers. So, a caption like “A man riding on a three-wheeled wheelchair” will be transformed into “man riding on three wheeled wheelchairs”
- **text_vocabulary(descriptions)** – This is a simple function that will separate all the unique words and create the vocabulary from all the descriptions.
- **save_descriptions(descriptions, filename)** – This function will create a list of all the descriptions that have been preprocessed and store them into a file. Then descriptions.txt file is created to store all the captions. It will look something like this:

File	Edit	Format	Run	Options	Window	Help
1000268201_693b08cb0e.jpg						child in pink dress is climbing up set of stairs in
1000268201_693b08cb0e.jpg						girl going into wooden building
1000268201_693b08cb0e.jpg						little girl climbing into wooden playhouse
1000268201_693b08cb0e.jpg						little girl climbing the stairs to her playhouse
1000268201_693b08cb0e.jpg						little girl in pink dress going into wooden cabin
1001773457_577c3a7d70.jpg						black dog and spotted dog are fighting
1001773457_577c3a7d70.jpg						black dog and tricolored dog playing with each other
1001773457_577c3a7d70.jpg						black dog and white dog with brown spots are staring
1001773457_577c3a7d70.jpg						two dogs of different breeds looking at each other
1001773457_577c3a7d70.jpg						two dogs on pavement moving toward each other
1002674143_1b742ab4b8.jpg						little girl covered in paint sits in front of paint
1002674143_1b742ab4b8.jpg						little girl is sitting in front of large painted re
1002674143_1b742ab4b8.jpg						small girl in the grass plays with fingerpaints in
1002674143_1b742ab4b8.jpg						there is girl with pigtails sitting in front of rai
1002674143_1b742ab4b8.jpg						young girl with pigtails painting outside in the gr
1003163366_44323f5815.jpg						man lays on bench while his dog sits by him

Fig.5.4. Description of Images

Extracting The Feature Vector from All Images

This technique is also called transfer learning, this is done by using the pre-trained model that have been already trained on large datasets and extract the features from these models and use them for our tasks. We are using the Xception model which has been trained on imagenet dataset that had 1000 different classes to classify. This model can be directly imported from keras.applications . It is important to be connected with the internet as the weights get automatically downloaded. Since the Xception model was originally built for imagenet. One thing to notice is that the Xception model takes 299*299*3 image size as input. Removing the last classification layer, 2048 feature vector are retrieved.

```
model = Xception( include_top=False, pooling="avg" )
```

The function **extract_features()** will extract features for all images and maps image names with their respective feature array. Then the features dictionary is dump into a “features.p” pickle file. This process can take a lot of time depending on your system. If Nvidia 1050 GPU is used for training purpose, it will take around 7 minutes for performing this task. However, we are using an

CPU thus this process takes 4-5 hours. Code can be commented and directly load the features from pickle file.

Loading Dataset for Training The Model

In **Flickr_30k_test** folder, **Flickr_30k.trainImages.txt** file that contains a list of 28k image names is used for training. For loading the training dataset, more functions are required:

- **load_photos(filename)** – This will load the text file in a string and will return the list of image names.
- **load_clean_descriptions(filename, photos)** – This function will create a dictionary that contains captions for each photo from the list of photos. <start> and <end> identifier is appended for each caption. This is done so that LSTM model can identify the starting and ending of the caption.
- **load_features(photos)** – This function will give the dictionary for image names and their feature vector which was previously extracted from the Xception model.

Tokenizing The Vocabulary

Computers don't understand English words, thus it is represented in numbers. So, each word of the vocabulary is mapped with a unique index value. Keras library provides us with the tokenizer function is used to create tokens from our vocabulary and save them to a “**tokenizer.p**” pickle file. Maximum length of the descriptions is calculated. This is important for deciding the model structure parameters. Max_length of description is 32.

Create Data Generator

To see the input and output of the model will require a supervised learning model for training. Model should be trained on 28k images and each image will

contain 2048 length feature vector and caption is also represented as numbers. This amount of data for 28k images is not possible to hold into memory so generator method is used that will yield batches.

The generator will yield the input and output sequence. For example:

The input to our model is $[x_1, x_2]$ and the output will be y , where x_1 is the 2048 feature vector of that image, x_2 is the input text sequence and y is the output text sequence that the model has to predict.

Table 5.2. Word Prediction Generation Step By Step

x_1 (feature vector)	x_2 (Text sequence)	y (word to predict)
feature	start,	two
feature	start, two	dogs
feature	start, two, dogs	drink
feature	start, two, dogs, drink	water
feature	start, two, dogs, drink, water	end

Defining the CNN-RNN model

To define the structure of the model, we will be using the Keras Model from Functional API. It will consist of three major parts:

- **Feature Extractor** – The feature extracted from the image has a size of 2048, with a dense layer, next the dimensions are reduced to 256 nodes.
- **Sequence Processor** – An embedding layer will handle the textual input, followed by the LSTM layer.
- **Decoder** – By merging the output from the above two layers, a dense layer is

formed to make the final prediction. The final layer will contain the number of nodes equal to our vocabulary size.

Training the model

To train the model, 28k training images for generating the input and output sequences is retrieved in batches and fitting them to the model using `model.fit_generator()` method. Then save this model to a folder, this task might take time to save.

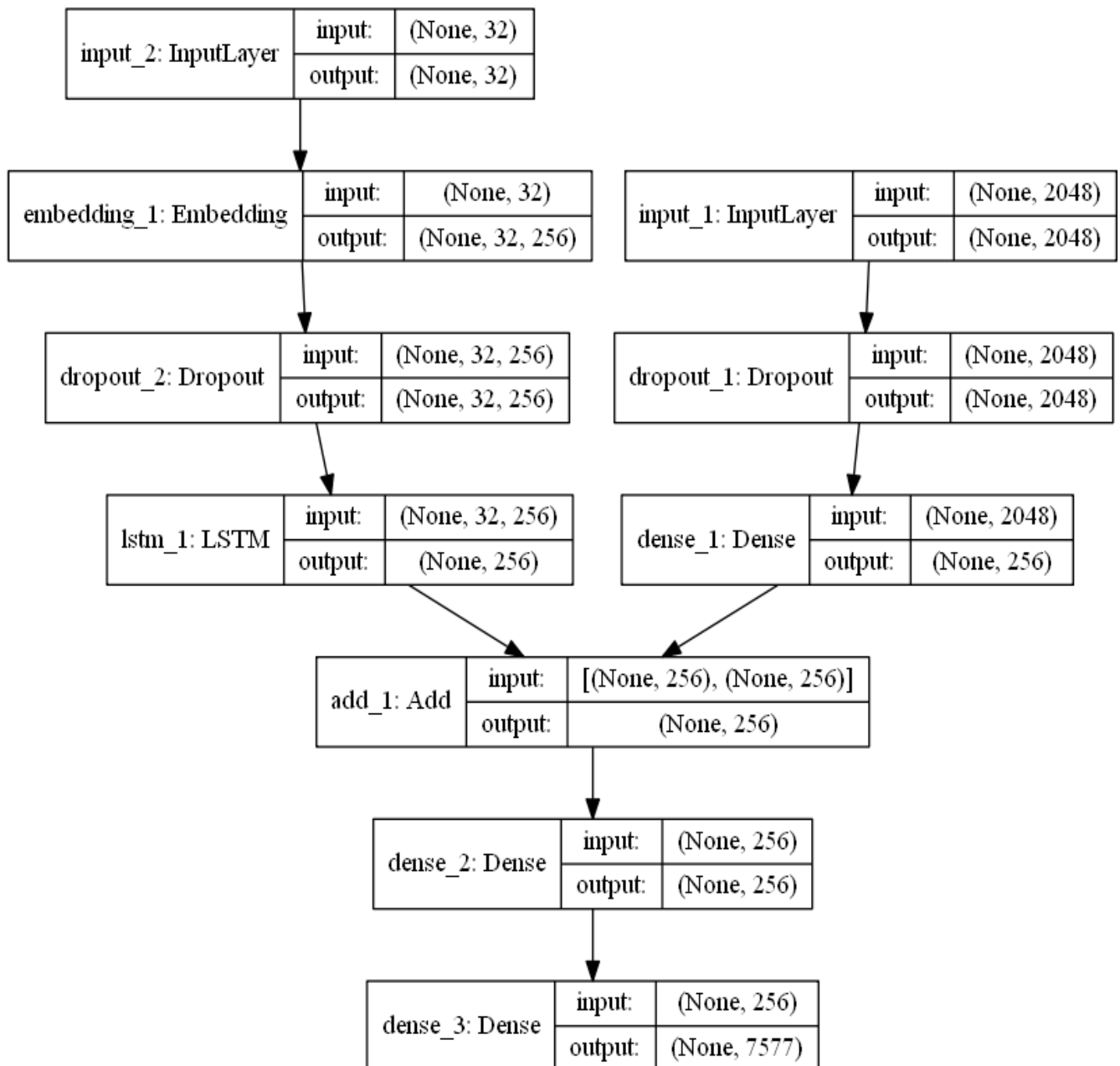


Fig.5.5. Final Model Structure

5.2 ALGORITHMS

Convolutional Neural Network

Artificial Neural Networks are used in various classification tasks like image, audio, words. Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use Recurrent Neural Networks more precisely an LSTM, similarly for image classification Convolution Neural networks is used.

The convolutional neural network, or CNN for short, is a specialized type of neural network model designed for working with two-dimensional image data, although they can be used with one-dimensional and three-dimensional data. Central to the convolutional neural network is the convolutional layer that gives the network its name. This layer performs an operation called a “*convolution*”.

- A convolutional neural network, or CNN, is a deep learning neural network sketched for processing structured arrays of data such as portrayals.
- CNN are very satisfactory at picking up on design in the input image, such as lines, gradients, circles, or even eyes and faces.
- This characteristic that makes convolutional neural network so robust for computer vision.
- CNN can run directly on a underdone image and do not need any preprocessing.
- A convolutional neural network is a feed forward neural network, seldom with up to 20.
- The strength of a convolutional neural network comes from a particular kind of layer called the convolutional layer.
- CNN contains many convolutional layers assembled on top of each other, each

one competent at recognizing more sophisticated shapes.

- With three or four convolutional layers it is viable to recognize handwritten digits and with 25 layers it is possible to differentiate human faces.
- The agenda for this sphere is to activate machines to view the world as humans do, perceive it in a similar fashion and even use the knowledge for a multitude of duties such as image and video recognition, image inspection and classification, media recreation, recommendation systems, natural language processing, etc.

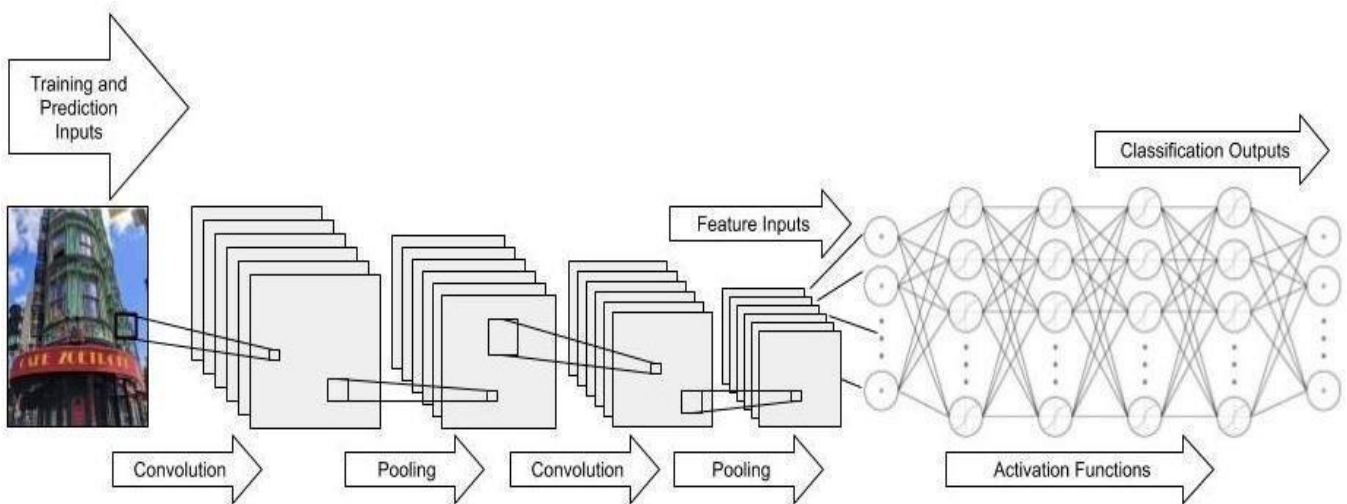


Fig.5.6 Convolutional Neural Network

The output from multiplying the filter with the input array one time is a single value. As the filter is applied multiple times to the input array, the result is a two-dimensional array of output values that represent a filtering of the input. As such, the two-dimensional output array from this operation is called a “*feature map*”.

Once a feature map is created, each value is passed in the feature map through a nonlinearity, such as a ReLU, much like we do for the outputs of a fully connected layer.

INCEPTION V3

Inception v3 is a convolutional neural network architecture that was developed by Google researchers in 2015. It was designed for image recognition and classification tasks with a focus on improving accuracy while reducing computational complexity and memory usage. One of the key innovations in Inception v3 is the use of factorized convolutions, which split the standard convolution operation into two separate operations: a $1 \times N$ convolution followed by an $N \times 1$ convolution. This reduces the number of parameters and computations required for each convolution, while also allowing the network to capture information at different scales and orientations.

Inception v3 also uses Inception modules, which are a series of convolutional layers that are combined in various ways to extract features at different levels of abstraction. Each module includes multiple branches, each of which performs a different type of convolution, such as 1×1 , 3×3 , and 5×5 . The outputs of these branches are concatenated and passed on to the next module, allowing the network to capture a wide range of features. Another key feature of Inception v3 is batch normalization, which normalizes the output of each layer to have zero mean and unit variance. This helps to improve training stability and reduce overfitting, as well as speed up the training process. Inception v3 also incorporates auxiliary classifiers, which are small classifiers that are inserted into the network at intermediate layers. These classifiers are trained to predict the final output labels and help to regularize the network and prevent overfitting. Overall, Inception v3 is a powerful deep learning architecture that has achieved state-of-the-art performance on a variety of image recognition benchmarks. It has been used in many applications, including facial recognition, object recognition, and image classification in medical imaging.

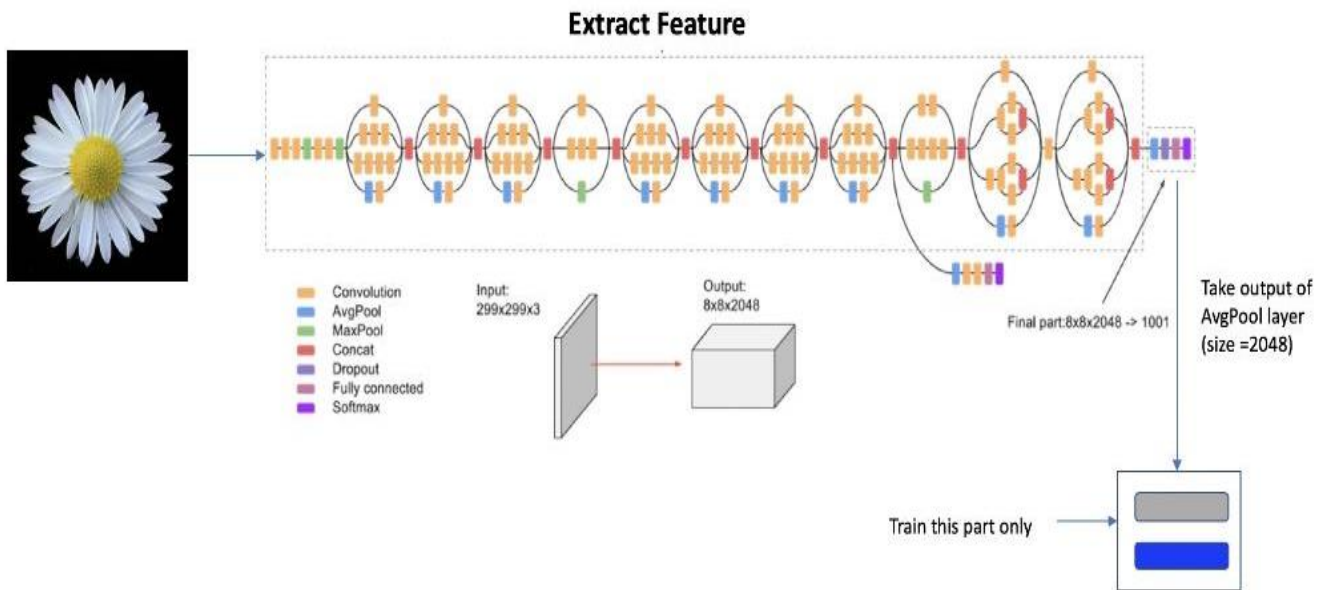


Fig.5.7 Working of Inception v3

POOLING LAYER

A pooling layer is a new layer added after the convolutional layer. Specifically, after a nonlinearity (e.g., ReLU) has been applied to the feature maps output by a convolutional layer.

Max Pooling Layer

Maximum pooling, or max pooling, is a pooling operation that calculates the maximum, or largest, value in each patch of each feature map. The results are down sampled or pooled feature maps that highlight the most present feature in the patch, not the average presence of the feature in the case of average pooling. This has been found to work better in practice than average pooling for computer vision tasks like image classification.

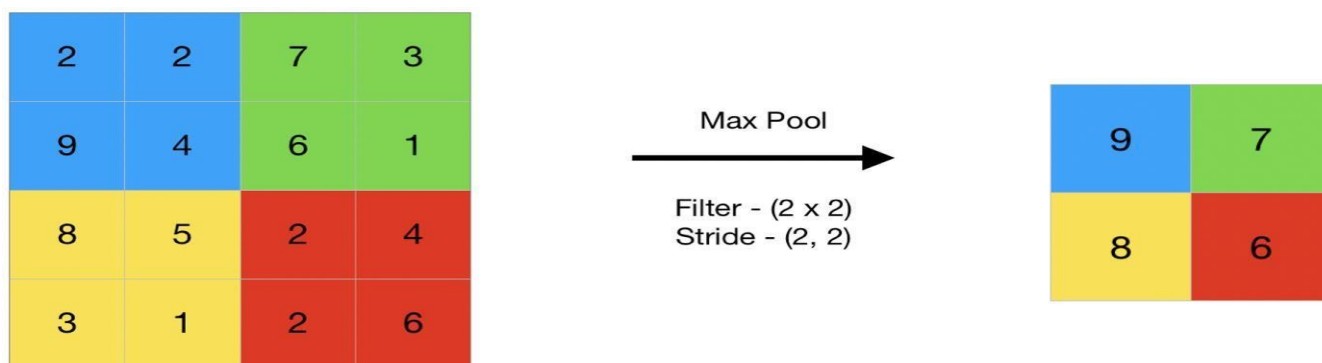


Fig.5.8 Max Pooling Layer

FULLY CONNECTED LAYER

Fully connected networks are the workhorses of deep learning, used for thousands of applications. The major advantage of fully connected networks is that they are “structure agnostic.” That is, no special assumptions need to be made about the input. In particular, the concept that fully connected architectures are “universal approximators” capable of learning any function. This concept provides an explanation of the generality of fully connected architectures, but comes with many caveats. While being structure agnostic makes fully connected networks very broadly applicable, such networks do tend to have weaker performance than special-purpose networks tuned to the structure of a problem space.

A fully connected neural network consists of a series of fully connected layers. A fully connected layer is a function from \mathbb{R}^m to \mathbb{R}^n . Each output dimension depends on each input dimension.

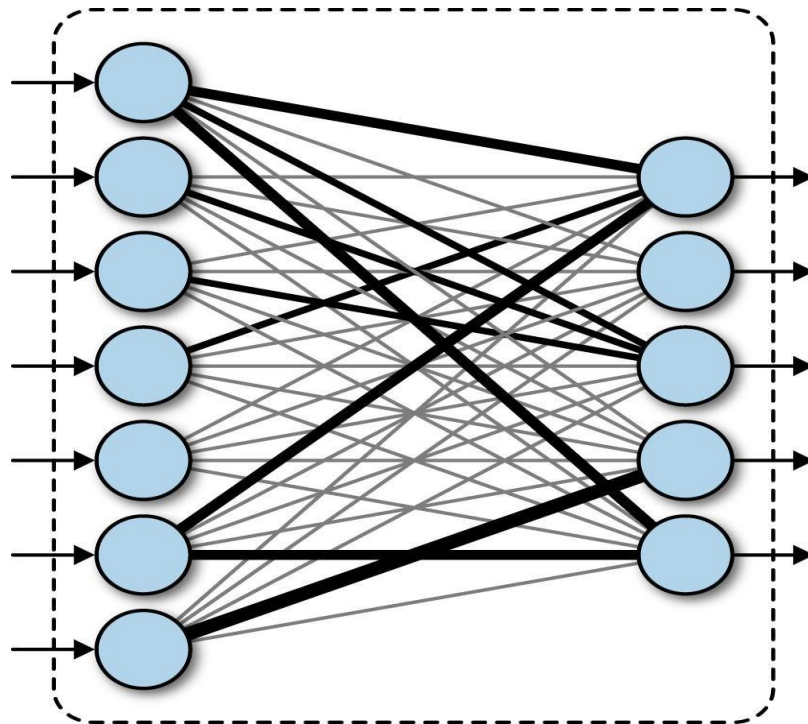


Fig.5.9 A fully connected layer in a deep network.

Let's dig a little deeper into what the mathematical form of a fully connected network is. Let $x \in \mathbb{R}^m$ represent the input to a fully connected layer. Let $y_i \in \mathbb{R}$ be the i -th output from the fully connected layer. Then $y_i \in \mathbb{R}$ is computed as follows:

$$y_i = \sigma(w_{i,1}x_1 + \dots + w_{i,m}x_m)$$

Here, σ is a nonlinear function (for now, think of σ as the sigmoid function introduced in the previous chapter), and the w_i are learnable parameters in the network. The full output y is then

$$y = \sigma(w_{1,1}x_1 + \dots + w_{1,m}x_m) : \sigma(w_{n,1}x_1 + \dots + w_{n,m}x_m)$$

Note that it's directly possible to stack fully connected networks. A network with multiple fully connected networks is often called a “deep” network as depicted

below.

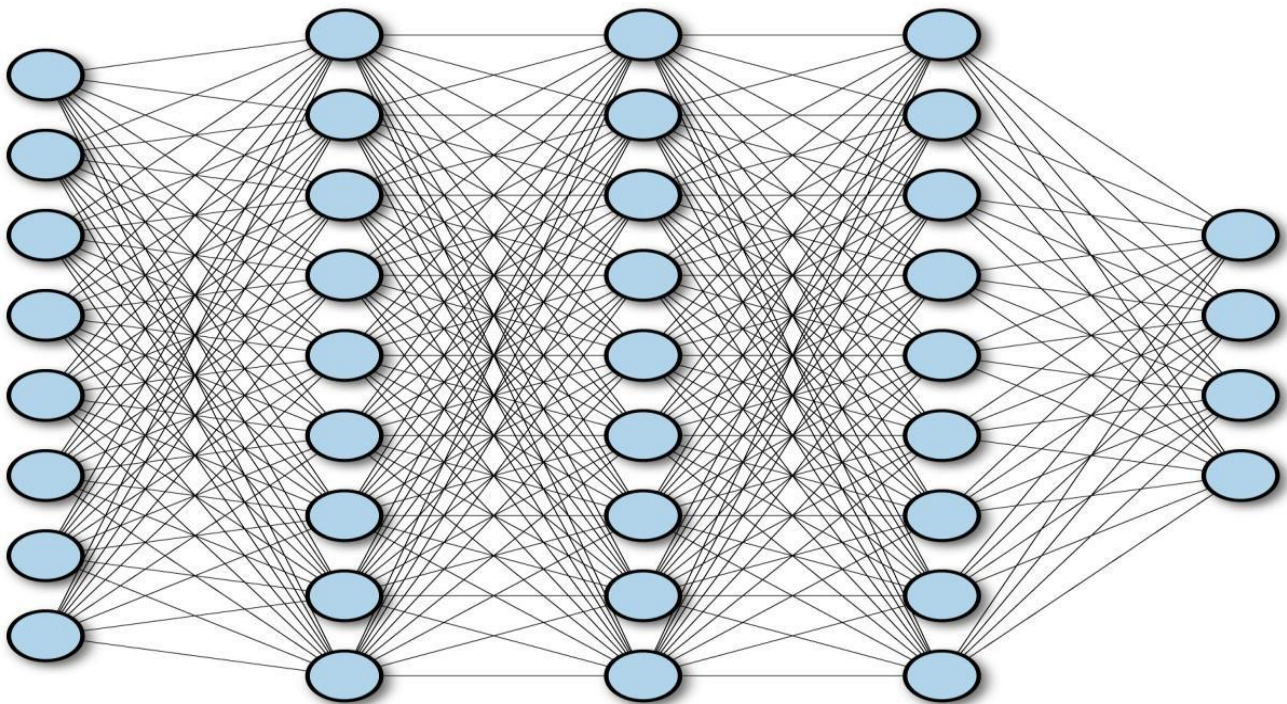


Fig.5.10 A multilayer deep fully connected network.

LSTM:

Traditional RNN suffers from vanishing and exploding gradient problem, which means that it cannot predict words in long-range dependencies. As the network gets deeper, the complexity increases and therefore the learning rate for the model becomes very slow, and the gradients of the cell decays as it is back propagated. If the activation function and the weights of the cells become less than 1, then the gradient vanishes. If it's more than 1, exploding gradient might happen. For that reason, LSTM, an improved version of RNN is used. This type of Neural Network has special units in addition to the standard units of RNN that uses a memory cell which maintains information in the memory for long periods of time and decides what to keep and what to forget.

CHAPTER 6

SYSTEM

IMPLEMENTATION

CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 CLIENT-SIDE CODING

Gui.py:

```
import tkinter as tk

from tkinter import filedialog

from tkinter import *

from PIL import ImageTk, Image

import numpy as np

import cv2

from tensorflow.keras.models import load_model

from tensorflow.keras.preprocessing.sequence import pad_sequences

from tensorflow.keras.models import Model

from tensorflow.keras.applications.inception_v3 import InceptionV3,
preprocess_input

from pickle import dump, load

from tensorflow.keras.preprocessing.image import load_img, img_to_array

from win32com.client import Dispatch

def speak(str1):

    speak=Dispatch ("SAPI.SpVoice")
```

```

speak.Speak(str1)

base_model = InceptionV3(weights =
r'inception_v3_weights_tf_dim_ordering_tf_kernels.h5')

vgg_model = Model(base_model.input, base_model.layers[-2].output)

def preprocess_img(img_path):

    #inception v3 expects img in 299*299

    img = load_img(img_path, target_size = (299, 299))

    x = img_to_array(img)

    # Add one more dimension

    x = np.expand_dims(x, axis = 0)

    x = preprocess_input(x)

    return x

def encode(image):

    image = preprocess_img(image)

    vec = vgg_model.predict(image)

    vec = np.reshape(vec, (vec.shape[1]))

    return vec

pickle_in = open("wordtoix.pkl", "rb")

wordtoix = load(pickle_in)

pickle_in = open("ixtoward.pkl", "rb")

```

```

ixtoword = load(pickle_in)

max_length = 74

def greedy_search(pic):

    start = 'startseq'

    for i in range(max_length):

        seq = [wordtoix[word] for word in start.split() if word in wordtoix]

        seq = pad_sequences([seq], maxlen = max_length)

        yhat = model.predict([pic, seq])

        yhat = np.argmax(yhat)

        word = ixtoword[yhat]

        start += ' ' + word

        if word == 'endseq':

            break

    final = start.split()

    final = final[1:-1]

    final = ' '.join(final)

    return final

def beam_search(image, beam_index = 3):

    start = [wordtoix["startseq"]]

    # start_word[0][0] = index of the starting word

```

```

# start_word[0][1] = probability of the word predicted

start_word = [[start, 0.0]]

while len(start_word[0][0]) < max_length:

    temp = []

    for s in start_word:

        par_caps = pad_sequences([s[0]], maxlen=max_length)

        e = image

        preds = model.predict([e, np.array(par_caps)])

        # Getting the top <beam_index>(n) predictions

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

        # creating a new list so as to put them via the model again

        for w in word_preds:

            next_cap, prob = s[0][:], s[1]

            next_cap.append(w)

            prob += preds[0][w]

            temp.append([next_cap, prob])

    start_word = temp

    # Sorting according to the probabilities

    start_word = sorted(start_word, reverse=False, key=lambda l: l[1])

    # Getting the top words

```

```

    start_word = start_word[-beam_index:]

start_word = start_word[-1][0]

intermediate_caption = [ixtoword[i] for i in start_word]

final_caption = []

for i in intermediate_caption:

    if i != 'endseq':

        final_caption.append(i)

    else: break

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

return final_caption

model = load_model('new-model-1.h5')

#initialise GUI

top=tk.Tk()

top.geometry('1236x600')

top.title('Caption Generator')

background_image = tk.PhotoImage(file=r'C:\Users\DELL\Documents\image-
caption-generator-using-deep-learning-master\eye.png')

background_label = tk.Label(top, image=background_image)

background_label.place(x=0, y=0, relwidth=1, relheight=1)

background_label.config(width=top.winfo_width(), height=top.winfo_height())

```

```

top.wm_attributes('-transparentcolor', '#ab23ff')

##top.configure(background='#CDCDCD')

label2=Label(top, font=('arial',15))

label1=Label(top, font=('arial',15))

label=Label(top, font=('arial',15))

sign_image = Label(top)

def classify(file_path):

    global label_packed

    enc = encode(file_path)

    image = enc.reshape(1, 2048)

    beam_3 = beam_search(image)

    print(beam_3)

    speak(beam_3)

    label1.configure(foreground='#011638', text = 'Statement: ' + beam_3)

    label1.pack(side = BOTTOM, expand = True)

def show_classify_button(file_path):

    classify_b=Button(top,text="Generate",command=lambda:
classify(file_path),padx=10,pady=5)

    classify_b.configure( foreground='black',font=('arial',10,'bold'))

    classify_b.place(relx=0.79,rely=0.46)

```

```

def upload_image():

    try:

        file_path=filedialog.askopenfilename()

        uploaded=Image.open(file_path)

        uploaded.thumbnail(((top.winfo_width()/2.25),(top.winfo_height()/2.25)))

        im=ImageTk.PhotoImage(uploaded)

        sign_image.configure(image=im)

        sign_image.image=im

        label.configure(text="")

        label1.configure(text="")

        label2.configure(text="")

        show_classify_button(file_path)

    except:

        pass

upload=Button(top,text="Upload an
image",command=upload_image,padx=10,pady=5)

upload.configure( foreground='black',font=('arial',10,'bold'))

upload.pack(side=BOTTOM,pady=50)

sign_image.pack(side=BOTTOM,expand=True)

top.mainloop()

```

6.2 SERVER-SIDE CODING

Image_Captioning.ipynb:

```
# This Python 3 environment comes with many helpful analytics libraries
installed

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import os

import tensorflow as tf

from tensorflow.keras.preprocessing.sequence import pad_sequences

from keras.preprocessing.text import Tokenizer

from keras.models import Model

from keras.layers import Flatten, Dense, LSTM, Dropout, Embedding,
Activation

from keras.layers import concatenate, BatchNormalization, Input

from tensorflow.keras.layers import concatenate

from tensorflow.keras.utils import to_categorical

from keras.applications.inception_v3 import InceptionV3, preprocess_input

from tensorflow.keras.utils import plot_model

import matplotlib.pyplot as plt

import cv2
```



```

import string

import time

print("Running.....")

from google.colab import drive

drive.mount('/content/drive')

token_path = '/content/drive/MyDrive/Colab Notebooks/Flickr30k.token.txt'

text = open(token_path, 'r', encoding = 'utf-8').read()

print(text[:100])

def load_description(text):

    mapping = dict()

    for line in text.split("\n"):

        token = line.split("\t")

        if len(line) < 2:

            continue

        img_id = token[0].split('.')[0]

        img_des = token[1]

        if img_id not in mapping:

            mapping[img_id] = list()

        mapping[img_id].append(img_des)

    return mapping

```

```

descriptions = load_description(text)

print("Number of items: " + str(len(descriptions)))

descriptions['1000268201_693b08cb0e']

def clean_description(desc):

    for key, des_list in desc.items():

        for i in range(len(des_list)):

            caption = des_list[i]

            caption = [ch for ch in caption if ch not in string.punctuation]

            caption = ".join(caption)

            caption = caption.split(' ')

            caption = [word.lower() for word in caption if len(word)>1 and
word.isalpha()]

            caption = '.join(caption)

            des_list[i] = caption

clean_description(descriptions)

descriptions['1000268201_693b08cb0e']

def to_vocab(desc):

    words = set()

    for key in desc.keys():

        for line in desc[key]:

```

```

        words.update(line.split())

    return words

vocab = to_vocab(descriptions)

len(vocab)

import glob

images = '/content/drive/MyDrive/Images/'

# Create a list of all image names in the directory

img = glob.glob(images + '*.jpg')

len(img)

train_path = '/content/drive/MyDrive/Colab
Notebooks/Flickr_30k.trainImages.txt'

train_images = open(train_path, 'r', encoding = 'utf-8').read().split("\n")

train_img = []

for im in img:

    if(im[len(images):] in train_images):

        train_img.append(im)

test_path = '/content/drive/MyDrive/Colab Notebooks/Flickr_30k.testImages.txt'

test_images = open(test_path, 'r', encoding = 'utf-8').read().split("\n")

test_img = []

for im in img:

```

```

    if(im[len(images): ] in test_images):

        test_img.append(im)

len(test_img)

#load descriptions of train and test set separately

def load_clean_descriptions(des, dataset):

    dataset_des = dict()

    for key, des_list in des.items():

        if key+'.jpg' in dataset:

            if key not in dataset_des:

                dataset_des[key] = list()

            for line in des_list:

                desc = 'startseq ' + line + ' endseq'

                dataset_des[key].append(desc)

    return dataset_des

train_descriptions = load_clean_descriptions(descriptions, train_images)

print('Descriptions: train=%d' % len(train_descriptions))

train_descriptions['1000268201_693b08cb0e']

from keras.preprocessing.image import load_img, img_to_array

def preprocess_img(img_path):

    #inception v3 expects img in 299*299

```

```

img = load_img(img_path, target_size = (299, 299))

x = img_to_array(img)

# Add one more dimension

x = np.expand_dims(x, axis = 0)

x = preprocess_input(x)

return x

base_model = InceptionV3(weights = 'imagenet')

base_model.summary()

model = Model(base_model.input, base_model.layers[-2].output)

#function to encode an image into a vector using inception v3

def encode(image):

    image = preprocess_img(image)

    vec = model.predict(image)

    vec = np.reshape(vec, (vec.shape[1]))

    return vec

#run the encode function on all train images

start = time.time()

encoding_train = { }

for img in train_img:

    encoding_train[img[len(images):]] = encode(img)

```

```

print("Time Taken is: " + str(time.time() - start))

#Encode all the test images

start = time.time()

encoding_test = { }

for img in test_img:

    encoding_test[img[len(images):]] = encode(img)

print("Time taken is: " + str(time.time() - start))

train_features = encoding_train

test_features = encoding_test

print("Train image encodings: " + str(len(train_features)))

print("Test image encodings: " + str(len(test_features)))

train_features['1000268201_693b08cb0e.jpg'].shape

#list of all training captions

all_train_captions = []

for key, val in train_descriptions.items():

    for caption in val:

        all_train_captions.append(caption)

len(all_train_captions)

#onsider only words which occur atleast 10 times

vocabulary = vocab

```

```

threshold = 10

word_counts = { }

for cap in all_train_captions:

    for word in cap.split(' '):

        word_counts[word] = word_counts.get(word, 0) + 1

vocab = [word for word in word_counts if word_counts[word] >= threshold]

print("Unique words: " + str(len(word_counts)))

print("our Vocabulary: " + str(len(vocab)))

#word mapping to integers

ixtoword = { }

wordtoix = { }

ix = 1

for word in vocab:

    wordtoix[word] = ix

    ixtoword[ix] = word

    ix += 1

vocab_size = len(ixtoword) + 1 #1 for appended zeros

vocab_size

#find the maximum length of a description in a dataset

max_length = max(len(des.split()) for des in all_train_captions)

```

```

max_length

#since there are almost 30000 descriptions to process we will use datagenerator

X1, X2, y = list(), list(), list()

for key, des_list in train_descriptions.items():

    pic = train_features[key + '.jpg']

    for cap in des_list:

        seq = [wordtoix[word] for word in cap.split(' ') if word in wordtoix]

        for i in range(1, len(seq)):

            in_seq, out_seq = seq[:i], seq[i]

            in_seq = pad_sequences([in_seq], maxlen = max_length)[0]

            out_seq = to_categorical([out_seq], num_classes = vocab_size)[0]

            #store

            X1.append(pic)

            X2.append(in_seq)

            y.append(out_seq)

X2 = np.array(X2)

X1 = np.array(X1)

y = np.array(y)

print(X1.shape)

```



```

#load glove vectors for embedding layer

embeddings_index = {}

glove = open('/content/drive/MyDrive/glove.6B.200d (1).txt', 'r', encoding = 'utf-8').read()

for line in glove.split("\n"):

    values = line.split(" ")

    word = values[0]

    indices = np.asarray(values[1: ], dtype = 'float32')

    embeddings_index[word] = indices

print('Total word vectors: ' + str(len(embeddings_index)))

emb_dim = 200

emb_matrix = np.zeros((vocab_size, emb_dim))

for word, i in wordtoix.items():

    emb_vec = embeddings_index.get(word)

    if emb_vec is not None:

        emb_matrix[i] = emb_vec

emb_matrix.shape

# define the model

ip1 = Input(shape = (2048, ))

fe1 = Dropout(0.2)(ip1)

```

```

fe2 = Dense(256, activation = 'relu')(fe1)

ip2 = Input(shape = (max_length, ))

se1 = Embedding(vocab_size, emb_dim, mask_zero = True)(ip2)

se2 = Dropout(0.2)(se1)

se3 = LSTM(256)(se2)

decoder1 = add([fe2, se3])

decoder2 = Dense(256, activation = 'relu')(decoder1)

outputs = Dense(vocab_size, activation = 'softmax')(decoder2)

model = Model(inputs = [ip1, ip2], outputs = outputs)

model.summary()

model.layers[2]

model.layers[2].set_weights([emb_matrix])

model.layers[2].trainable = False

model.compile(loss = 'categorical_crossentropy', optimizer = 'adam')

plot_model(model, to_file = 'model.png', show_shapes = True,
show_layer_names = True)

for i in range(30):

    model.fit([X1, X2], y, epochs = 1, batch_size = 256)

    if(i%2 == 0):

        model.save_weights("image-caption-weights" + str(i) + ".h5")

```

```

def greedy_search(pic):

    start = 'startseq'

    for i in range(max_length):

        seq = [wordtoix[word] for word in start.split() if word in wordtoix]

        seq = pad_sequences([seq], maxlen = max_length)

        yhat = model.predict([pic, seq])

        yhat = np.argmax(yhat)

        word = ixtoword[yhat]

        start += ' ' + word

        if word == 'endseq':

            break

    final = start.split()

    final = final[1:-1]

    final = ' '.join(final)

    return final

pic = list(encoding_test.keys())[250]

img = encoding_test[pic].reshape(1, 2048)

x = plt.imread(images + pic)

plt.imshow(x)

plt.show()

```

```
print(greedy_search(img))

model.save("my-cap.h5")

# assume X1 is the actual label of the data and X2 is the predicted label

acc = accuracy_score(X1, X2)

print("Accuracy: {:.2f}%".format(acc * 100))
```

CHAPTER 7

PERFORMANCE

ANALYSIS

CHAPTER 7

PERFORMANCE ANALYSIS

7.1 RESULTS & DISCUSSION

According to the trained model, it is observed that the percentage of the accuracy is gradually increasing with respect to the number of epochs. According to **Fig.7.1**, continuous improvement is found till the epoch 50, after which the accuracy has achieved to a certain level and no other improvements are found for further epochs. **Fig.7.1**. Epoch Vs Accuracy. The output of the model is generated in both description and audio form. It generates 3 different descriptions for each image and displays the accurate results.

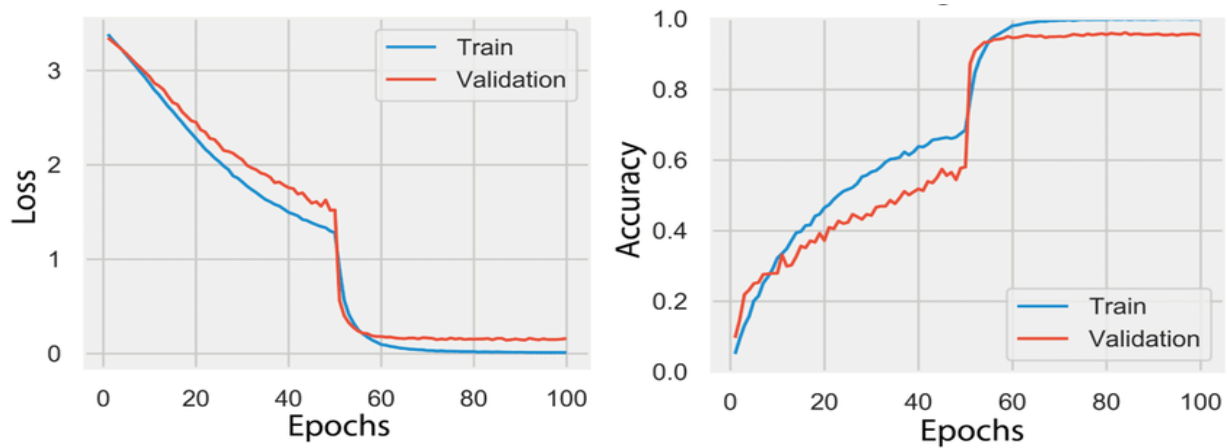


Fig 7.1. Epoch Vs Accuracy

7.2 TEST CASES & REPORTS

TEST CASE ID	INPUT	EXPECTED OUTPUT	OBTAINED OUTPUT	PASS/FAIL	REMARKS
TC01	Dataset	Successful import	Successful import	Pass	Imported successfully
TC02	Dataset	Pre-process successful	Pre-process successful	Pass	Pre-processed successfully
TC03	Dataset	Model creation	Model created	Pass	Model created successfully
TC04	Model	Successful compilation	Successful compilation	Pass	Model compiled successfully
TC05	Model	Successful loading of model	Successful loading of model	Pass	Model loaded successfully

TC06	Image	Successful upload of image	Successful upload of image	Pass	Uploaded successfully
TC07	Image	Recognize the image successfully	Recognize the image successfully	Pass	Classification successful
TC08	Image	Generate the caption	Generated the caption	Pass	Generation successful
TC09	Image	Generate audible caption	Generated audible caption	Pass	Generation successful

CHAPTER 8

CONCLUSION

CHAPTER 8

CONCLUSION

8.1 CONCLUSION AND FUTURE ENHANCEMENTS

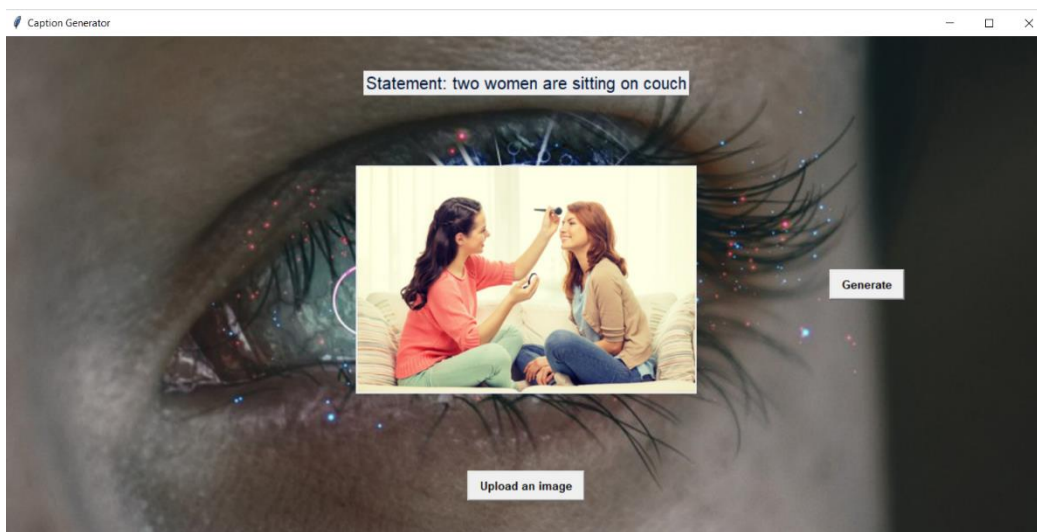
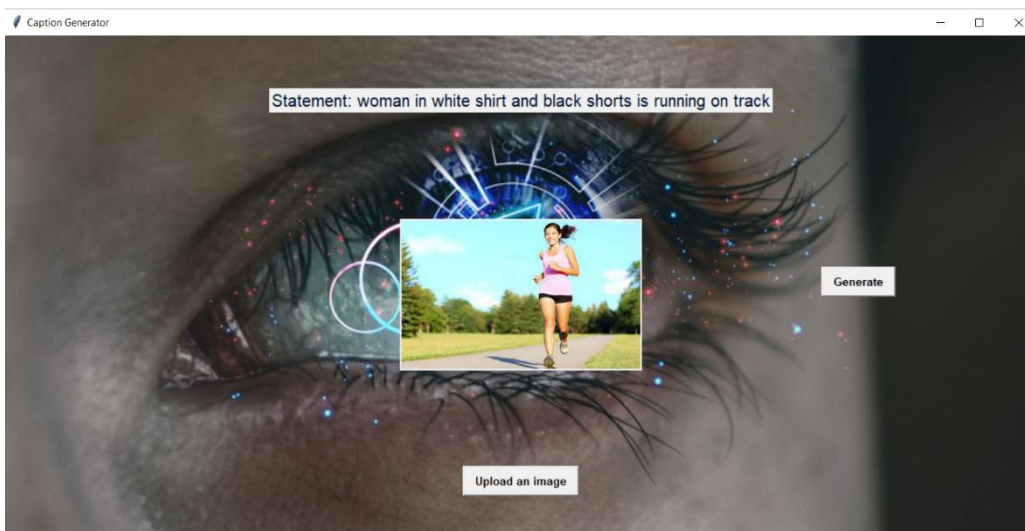
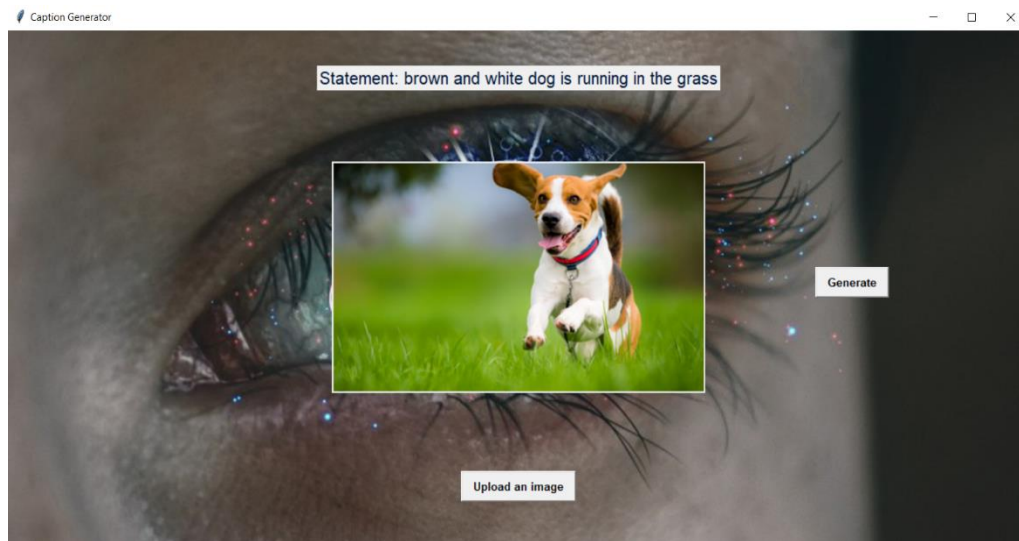
In this project, we present a combination of image captioning using Inception V3 and TTS (Text-To-Speech) model which successfully generate the textual and audible captions for images with high accuracy. The use of Inception V3 for feature extraction and an LSTM network with attention mechanism for sequence generation has proven to be effective in generating coherent and meaningful captions for a wide range of images. The model is trained on the Flickr 30k dataset for increasing the performance.

In future, the model could be trained on a larger and more diverse dataset (like MSCOCO) to further improve its performance and adaptability to various images. Overall, Visual Audio for Visionless has laid the foundation for further research and development in the field of computer vision and natural language processing, with potential applications in areas such as visual storytelling and assistive technology for visually impaired individuals.

APPENDICES

APPENDICES

A.1 Sample Screens



REFERENCES

REFERENCES

- [1] A. Attai and A. Elnagar, "A survey on Arabic Image Captioning Systems Using Deep Learning Models," *Displays*, vol. 114, 2022, doi: 10.1109/IIT50501.2020.9299027.
- [2] A. K. Poddar and R. Rani, "Hybrid Architecture using CNN and LSTM for Image Captioning in Hindi Language," in *Procedia Computer Science*, vol. 218, 2023, pp. 589-595, doi: 10.1016/j.procs.2023.01.049.
- [3] A. Salaberria, G. Azkune, O. Lopez de Lacalle, A. Soroa and E. Agirre, "Image captioning for effective use of language models in knowledge-based visual question answering," in *Expert Systems with Applications*, vol. 212, 2023, article no. 118669, doi: 10.1016/j.eswa.2022.118669.
- [4] A. Singh, S. M. Singh, L. S. Meetei, R. Das, T. D. Singh and S. Bandyopadhyay, "VATEX2020: pLSTM framework for video captioning," in *Procedia Computer Science*, vol. 218, 2023, pp. 981-990, doi: 10.1016/j.procs.2023.01.101.
- [5] G. Geetha, T. Kirthigadevi, G. Godwin Ponsam, T. Karthik and M. Safa, "Image Captioning Using Deep Convolutional Neural Networks (CNNs)," in *Journal of Physics*, vol. 1712, 2023, article no. 012015, doi: 10.1088/1742-6596/1712/1/012015.
- [6] I. Puthige, T. Hussain, S. Gupta and M. Agarwal, "Attention Over Attention: An Enhanced Supervised Video Summarization Approach," in *Procedia Computer Science*, vol. 218, 2023, pp. 203-212, doi: 10.1016/j.procs.2023.01.211.
- [7] J. Wang, S. Wang and Y. Zhang, "Artificial intelligence for visually impaired," in *Displays*, vol. 77, 2023, article no. 102391, doi: 10.1016/j.displa.2023.102391.

- [8] M. A. Al-Malla, A. Jafar and N. Ghneim, "Image captioning model using attention and object features to mimic human image understanding," in Springer, vol. Article 20, 2022, pp. 1-17, doi: 10.1186/s40537-022-00571-w.
- [9] M. Alfaro-Contreras, J. J. Valero-Mas, J. M. Iñesta, and J. Calvo-Zaragoza, "Late multimodal fusion for image and audio music transcription," *Expert Systems with Applications*, vol. 216, 119491, 2023, doi: 10.1016/j.eswa.2022.119491.
- [10] Md. S. Zaoad, M. M. R. Mannan, A. B. Mandol, M. Rahman, M. A. Islam, and M. M. Rahman, "An attention-based hybrid deep learning approach for Bengali video captioning," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 35, 2023, doi: 10.1016/j.jksuci.2022.11.015.
- [11] N. Wang, J. Xie, H. Luo, Q. Cheng, J. Wu, M. Jia, and L. Li, "Efficient Image Captioning for Edge Devices," *arXiv preprint arXiv:2212.08985*, 2022, doi: 10.48550/arXiv.2212.08985.
- [12] R. Castro, I. Pineda, W. Lim, and M. E. Morocho-Cayamcela, "Deep Learning Approaches Based on Transformer Architectures for Image Captioning Tasks," *IEEE Access*, vol. 10, pp. 48313-48323, 2022, doi: 10.1109/ACCESS.2022.3161428.
- [13] R. Ramos and B. Martins, "Using Neural Encoder-Decoder Models with Continuous Outputs for Remote Sensing Image Captioning," *IEEE Access*, vol. 10, pp. 40667-40678, 2022, doi: 10.1109/ACCESS.2022.3151874.
- [14] S. Nazir, D. M. Dickson and M. U. Akram, "Survey of explainable artificial intelligence techniques for biomedical imaging with deep neural networks," in *Computers in Biology and Medicine*, vol. 156, 2023, article no. 106668, doi: 10.1016/j.compbimed.2023.106668.