

IMAGE CAPTION GENERATOR

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering

by

C.PAVAN (Reg.No. - 40111571)
K.NARASIMHA BABU (Reg.No. - 40111583)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Category – I University by UGC

Accredited with Grade "A++" by NAAC | 12B Status by UGC | Approved by AICTE

**JEPPIAAR NAGAR, RAJIV GANDHISALAI,
CHENNAI - 600119**

APRIL-2024





SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited "A++" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

It is to certify that this Project Report is the bonafide work of **C. PAVAN** (0111571) and **K. NARASIMHA BABU** (40111583) who carried out the Project titled "IMAGE CAPTION GENERATOR" under my supervision from Nov 2023 to April 2024.

Internal Guide

Dr. SONIA JENIFER RAYEN M.Tech., Ph.D.,

HEAD OF THE DEPARTMENT
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
Dr. LAKSHMANAN M.P.E., Ph.D.,
Jeppiaar Nagar, Raj: Gandhi Salai,
Chennai - 600 119.

Submitted for Viva voce Examination held on 16-04-24

Wair 16/04/2024

Internal Examiner

16/04/24

External Examiner

DECLARATION

I, PAVAN C (40111571), hereby declare that the Project Report entitled "IMAGE CAPTION GENERATOR" done by me under the guidance of Dr. Sonia Jenifer Rayen, M.Tech., Ph.D., is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE: 16-04-24

PLACE: Chennai

C. Pavan

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph.D., Dean**, School of Computing, **Dr. L. Lakshmanan M.E., Ph.D.**, Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. Sonia Jenifer Rayen M.Tech., Ph.D.**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

An advanced Image Caption Generator is needed in an era of increasing visual content on the internet, the ability to automatically generate descriptive captions for images has become crucial for enhancing accessibility and user experience. It leverages state-of-the-art deep neural networks, specifically Convolutional Neural Networks (CNNs) for image feature extraction and Recurrent Neural Networks (RNNs) for generating coherent and contextually relevant captions.

The system takes an image as input and employs a pre-trained CNN to extract high-level features, creating a rich representation of the visual content. Subsequently, an RNN-based sequence-to-sequence model processes these features to generate natural language captions. To improve the quality and fluency of captions, the model incorporates attention mechanisms, allowing it to focus on different parts of the image while generating each word.

It has broad applications in fields such as image indexing, content retrieval, and accessibility, making digital visual content more understandable and engaging for a wide range of users. It contributes to the advancement of deep learning techniques in computer vision and natural language processing, pushing the boundaries of AI capabilities in understanding and describing visual information.

TABLE OF CONTENTS

Chapter No.	TITLE	Page No.
	ABSTRACT	iv
	LIST OF FIGURES	vi
1	INTRODUCTION	1
2	LITERATURE SURVEY	4
	2.1 Inferences from Literature Survey	4
	2.2 Existing Systems and Methods	6
	2.3 Open problems in Existing System	8
3	REQUIREMENTS ANALYSIS	11
	3.1 Feasibility Studies/Risk Analysis of the Project	11
	3.2 Software and Hardware Requirements	13
	3.3 External Interface Requirements	13
4	DESCRIPTION OF PROPOSED SYSTEM	15
	4.1 Selected Methodology or process model	15
	4.2 Architecture / Overall Design of Proposed System	15
5	IMPLEMENTATION	18
	5.1 Pre-requisites	18
	5.2 Project Structure	18
	5.3 Building the Python Project	19
6	RESULTS AND DISCUSSIONS	27
	6.1 Results	27
	6.2 Discussions	27
7	CONCLUSION, LIMITATIONS AND FUTURE SCOPE	30
	7.1 Conclusion	30
	7.2 Limitations	31
	7.3 Future Scope	32
	REFERENCES	33
	APPENDIX	35

A	Source Code	35
B	Screen Shots	41
C	Research Paper	43

LIST OF FIGURES

Figure No.	FIGURE NAME	Page No.
2.1	Novel Caption Generator	6
2.2	A block diagram of multimodal space-based image captioning	8
4.1	Model, Image Caption Generator	16
4.2	Forget Gate, Input Gate, Output Gate	17
5.1	Flicker DataSet text format	19
5.2	Flicker Dataset Python File	20
5.4	Final Model Structure	25
5.5	Output Caption of Given Image	26

CHAPTER 1

INTRODUCTION

In today's digital age, the ubiquity of visual content is undeniable. From social media platforms to e-commerce websites and educational resources, images have become a fundamental means of communication and information dissemination. Yet, one critical challenge remains: how can we make these images accessible and comprehensible to everyone, including individuals with visual impairments or those seeking enhanced search and retrieval capabilities?

The answer lies in the development of an advanced Image Caption Generator—a cutting-edge application of artificial intelligence (AI) and computer vision. This project embarks on a journey to bridge the gap between the visual and linguistic realms by harnessing the power of deep learning techniques and neural networks. The objective is to create a system that can automatically generate descriptive and contextually relevant captions for a wide array of images, making them not only visually appealing but also informative and accessible.

The rationale behind this endeavor is twofold. First and foremost, it addresses the imperative of accessibility. For individuals with visual impairments, providing meaningful descriptions of images through generated captions enables them to engage with and understand digital content that was previously beyond their reach. This inclusivity aligns with the principles of universal design and ensures equitable access to information.

1.2 Objectives:

- Create a deep learning-based image captioning model that can take images as input and generate descriptive and contextually relevant captions.
- Gather a diverse dataset of images and corresponding human-generated captions, ensuring a wide range of image categories and themes. Preprocess the data to prepare it for training.
- Implement and fine-tune pre-trained CNN models (e.g., VGG, ResNet) to extract high-level features from input images, creating a robust representation of visual content.
- Integrate attention mechanisms into the model to allow it to focus on different regions of the image while generating each word in the caption, improving the relevance of descriptions.
- Implement and use evaluation metrics such as BLEU, METEOR, and CIDEr to quantitatively assess the quality and relevance of generated captions in comparison to ground truth annotations.
- Test the image caption generator across various domains, including social media images, product images, educational materials, and more, to assess its adaptability and versatility.

1.3 IMAGE CAPTIONING:

Process :-

Image Captioning is the process of generating textual description of an image. It uses both Natural Language Processing and Computer Vision to generate the captions. Image captioning is a popular research area of Artificial Intelligence (AI) that deals with image understanding and a language description for that image. Image understanding needs to detect and recognize objects. It also needs to understand scene type or location, object properties and their interactions. Generating well-formed sentences requires both syntactic and semantic understanding of the language. Understanding an image largely depends on obtaining image features² For example, they can be used for

automatic image indexing. Image indexing is important for Content-Based Image Retrieval (CBIR) and therefore, it can be applied to many areas, including biomedicine, commerce, the military, education, digital libraries, and web searching. Social media platforms such as Facebook and Twitter can directly generate descriptions from images. The descriptions can include where we are (e.g., beach, cafe), what we wear and importantly what we are doing there.

Techniques :-

The techniques used for this purpose can be broadly divided into two categories: (1) Traditional machine learning based techniques and (2) Deep machine learning based techniques. In traditional machine learning, hand crafted features such as Local Binary Patterns (LBP), Scale-Invariant Feature Transform (SIFT), the Histogram of Oriented Gradients (HOG) and a combination of such features are widely used. In these techniques, features are extracted from input data. They are then passed to a classifier such as Support Vector Machines (SVM) in order to classify an object. Since hand crafted features are task specific, extracting features from a large and diverse set of data is not feasible. Moreover, real world data such as images and video are complex and have different semantic interpretations. On the other hand, in deep machine learning based techniques, features are learned automatically from training data and they can handle a large and diverse set of images and videos. For example, Convolutional Neural Networks (CNN) are widely used for feature learning, and a classifier such as Softmax is used for classification. CNN is generally followed by Recurrent Neural Networks (RNN) or Long Short-Term Memory Networks (LSTM) in order to generate 10 captions. Deep learning algorithms can handle complexities and challenges of image captioning quite well.

CHAPTER 2

LITERATURE SURVEY

2.1 INFERENCES FROM LITERATURE SURVEY

Systematic Literature Review on Image Captioning, Raimonda Staniūtė

Image captioning is a very exciting exercise and raises tough competition among researchers. There are more and more scientists who are deciding to explore this study field, so the amount of information is constantly increasing. It was noticed that the results are usually compared with quite old articles, although there are dozens of new ones, with even higher results and new ideas for improvements. The comparison with older articles gives a misunderstanding of the real view of result increase—usually there have been much higher results already achieved, however not included in the paper. New ideas can also very easily become lost if they are not looked for carefully. In order to prevent good ideas been lost and to increase fair competition among the new models created, this systematic literature review summarizes all the newest articles and their results in one place. Moreover, it is still not clear if MS COCO and Flickr30k datasets are enough for model evaluation and if they serve sufficiently well when having in mind diverse environments.

Show and Tell: A Neural Image Caption Generator (2015), Oriol Vinyals, Alexander Toshev

This seminal work introduced a neural network architecture for image captioning. It used a convolutional neural network (CNN) for image feature extraction and a long short-term memory (LSTM) network for generating captions.

Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering (2018), Peter Anderson, Xiaodong He

This work proposed a novel approach by combining bottom-up and top-down attention mechanisms. The bottom-up attention involved identifying salient image regions, and the top-down attention guided the generation process based on linguistic context.

Image Captioning with Semantic Attention (2017), Bryan A. Plummer, Liwei Wang, Chris M

Introducing semantic attention, this paper aimed to improve image captioning by aligning image regions with semantically relevant words. This approach helped generate more accurate and contextually meaningful captions.

Knowing When to Look: Adaptive Attention via A Visual Sentinel for Image Captioning (2017), Jiasen Lu, Caiming Xiong

This work proposed an adaptive attention mechanism that decides when to attend to different parts of the image. It introduced a visual sentinel to help the model determine whether to rely on visual attention or focus on the language model.

Up-Down: A Residual Recurrent Network for Image Captioning (2018), Qiuyuan Huang, Haoran Wang

The Up-Down model used a residual recurrent network and integrated bottom-up attention. It achieved state-of-the-art performance on multiple benchmarks and demonstrated the effectiveness of residual connections in image captioning.

Contextual Attention for Image Captioning (2018), Subhashini Venugopalan, Linjie Yang

This paper proposed contextual attention, which takes into account the global image context while attending to local regions. The model demonstrated improved performance in generating contextually relevant captions.

Meshed-Memory Transformer for Image Captioning (2019), Hengyuan Hu, Zihui Li, Gang Yu

Introducing a meshed-memory transformer, this paper applied a self-attention mechanism to capture long-range dependencies in image features. The model showed competitive performance on image captioning tasks.

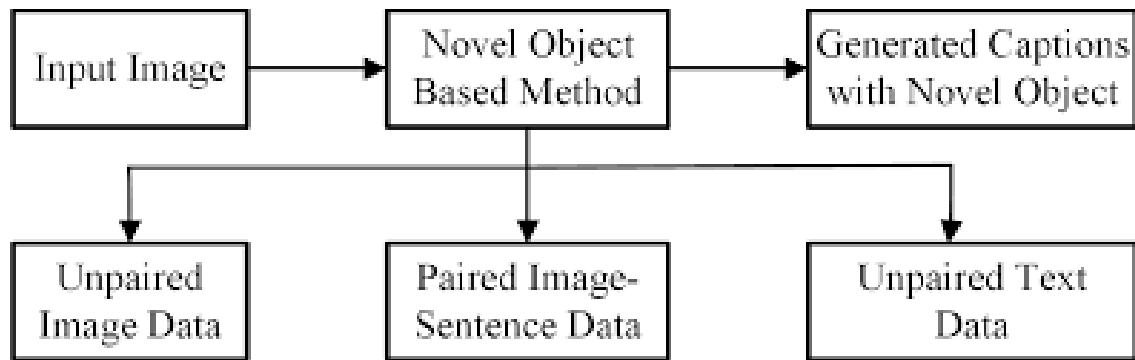


Fig: 2.1 NOVEL CAPTION GENERATION.

2.2 Existing Systems and Methods

DEEP LEARNING BASED IMAGE CAPTIONING METHODS:

We draw an overall taxonomy in Figure 1 for deep learning-based image captioning methods. We discuss their similarities and dissimilarities by grouping them into visual space vs. multimodal space, dense captioning vs. captions for the whole scene, Supervised learning vs. Other deep learning, Encoder-Decoder architecture vs. Compositional architecture, and one „Others“ group that contains Attention-Based, Semantic Concept-Based, Stylized captions, and Novel Object-Based

captioning. We also create a category named LSTM vs. Others. A brief overview of the deep learning-based image captioning methods is shown in table. It contains the name of the image captioning methods, the type of deep neural networks used to encode image information, and the language models used in describing the information. In the final column, we give a category label to each captioning technique based on the taxonomy in Figure 1.

2.2.1 VISUAL SPACE VS. MULTIMODAL SPACE :

Deep learning-based image captioning methods can generate captions from both visual space and multimodal space. Understandably image captioning datasets have the corresponding captions as 14 text. In the visual space-based methods, the image features and the corresponding captions are independently passed to the language decoder. In contrast, in a multimodal space case, a shared multimodal space is learned from the images and the corresponding caption-text. This multimodal representation is then passed to the language decoder.

VISUAL SPACE :

Bulk of the image captioning methods use visual space for generating captions. In the visual space-based methods, the image features and the corresponding captions are independently passed to the language decoder.

MULTIMODAL SPACE:

The architecture of a typical multimodal space-based method contains a language Encoder part, a vision part, a multimodal space part, and a language decoder part. A general diagram of multimodal space-based image captioning methods is shown in Figure 2. The vision part uses a deep convolutional neural network as a feature extractor to extract the image features. The language encoder part extracts the word features and learns a dense feature embedding for each word. It then forwards the semantic temporal context to the recurrent layers. The multimodal space part maps the image features into a common space with the word features.

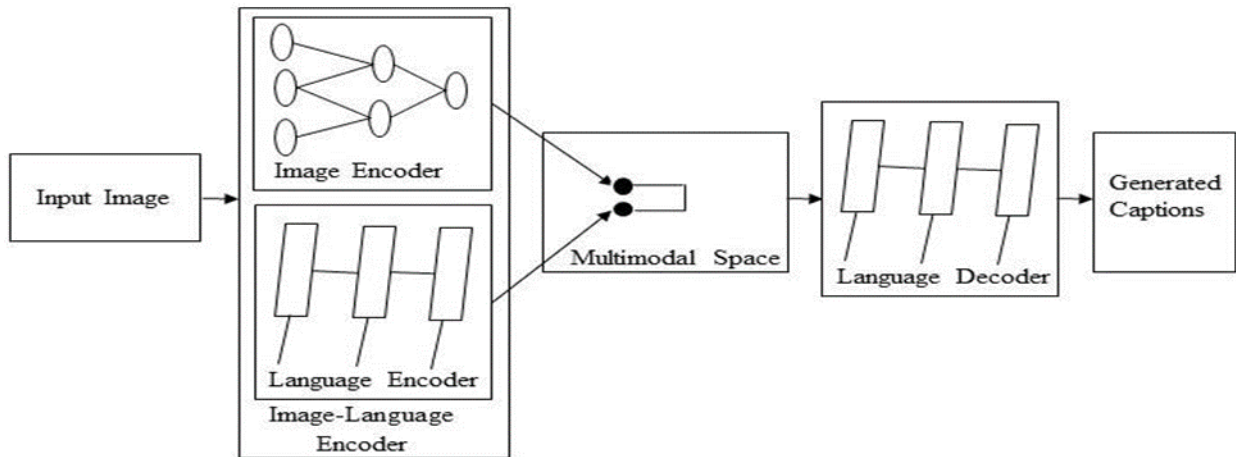


Fig: 2.2. A block diagram of multimodal space-based image captioning.

2.3 Open problems in Existing System

SUPERVISED LEARNING VS. OTHER DEEP LEARNING:

In supervised learning, training data come with desired output called label. Unsupervised learning, on the other hand, deals with unlabeled techniques. Reinforcement learning is another type of machine learning approach where the aim of an agent is to discover data and/or labels through exploration and a reward signal. A number of image captioning methods use reinforcement learning and GAN-based approaches. These methods sit in the category of "Other Deep Learning". Generative Adversarial Networks (GANs) are a type of unsupervised learning.

2.3.1 SUPERVISED LEARNING-BASED IMAGE CAPTIONING:

Supervised learning-based networks have successfully been used for many years in image classification , object detection and attribute learning . This progress makes researchers interested in using them in automatic image captioning .In this paper, we have identified a large number of supervised learning-based image captioning methods. We classify them into different categories: (i) Encoder-Decoder Architecture, (ii) Compositional Architecture, (iii) Attentionbased, (iv) Semantic concept-based, (v) Stylized captions, (vi) Novel object-based, and (vii) Dense image captioning.

2.3.2 OTHER DEEP LEARNING-BASED IMAGE CAPTIONING:

In our day to day life, data are increasing with unlabeled data because it is often impractical to accurately annotate data. Therefore, recently, researchers are focusing more on reinforcement learning and unsupervised learning-based techniques for image captioning.

2.3.3 DENSE CAPTIONING VS. CAPTIONS FOR THE WHOLE SCENE:

In dense captioning, captions are generated for each region of the scene. Other methods generate captions for the whole scene.

2.3.4 DENSE CAPTIONING:

The previous image captioning methods can generate only one caption for the whole image. They use different regions of the image to obtain information of various objects. However, these methods do not generate region wise captions. Johnson et al. [62] proposed an image captioning method called DenseCap. This method localizes all the salient regions of an image and then it generates descriptions for those regions. A typical method of this category has the following steps: (1) Region proposals are generated for the different regions of the given image. (2) CNN is used to obtain the region-based image features. (3) The outputs of Step 2 are used by a language model to generate captions for every region. A block diagram of a typical dense captioning method is given in Figure 4.

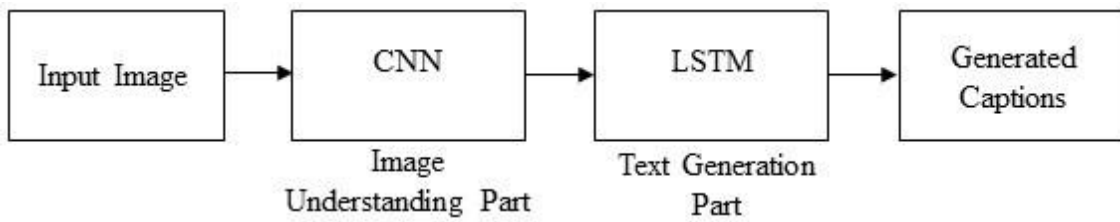


Fig:2.3: A block diagram of simple Encoder-Decode architecture

2.3.5 Fine-Grained Object Recognition and Description

Many image captioning models struggle with providing detailed descriptions of fine-grained objects or subtle visual details. Improving the recognition and description of specific objects within an image remains a challenge.

2.3.6 Handling Ambiguity and Generating Diverse Captions

Addressing the issue of ambiguity in images, where multiple valid interpretations exist, is an ongoing challenge. Generating diverse and creative captions for a single image is also a desirable but challenging task.

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 Feasibility Analysis of the Requirements:

In our process, the feasibility analysis is, the tool which we are used as Spyder IDE from Anaconda Navigator. The programming language which we are used as Python.

1. Technical Feasibility:

Availability of Technology: Evaluate the availability of necessary technologies such as deep learning frameworks (e.g., TensorFlow, PyTorch) and pre-trained models for image recognition and natural language processing.

Skill and Expertise: Assess the technical skills and expertise required to implement and train the image captioning model, including proficiency in programming languages (e.g., Python), machine learning concepts, and neural network architectures.

Hardware Requirements: Determine the hardware requirements for training and inference, considering factors such as GPU availability and computational resources.

2. Operational Feasibility:

Data Acquisition: Evaluate the feasibility of acquiring a sufficient amount of high-quality image-caption pairs for training the model. Consider the availability of annotated datasets like MSCOCO or Flickr30k and the effort required for data preprocessing.

Model Training: Assess the feasibility of training the model within a reasonable time frame, considering computational resources, training data size, and optimization techniques.

Integration: Evaluate the feasibility of integrating the image captioning model into the desired application or platform, including compatibility with existing systems and APIs.

3. Economic Feasibility:

Cost Analysis: Estimate the costs associated with data acquisition, hardware infrastructure (e.g., GPU servers), software licenses, and personnel (e.g., data annotators, machine learning engineers).

Return on Investment (ROI): Assess the potential benefits and returns from deploying the image captioning system, considering factors such as improved user experience, increased efficiency, and potential revenue streams.

Sustainability: Consider the long-term sustainability of the project in terms of ongoing maintenance, support, and updates, and evaluate whether the projected benefits outweigh the costs.

Conclusion:

Summarize the findings of the feasibility analysis, highlighting any potential risks, challenges, or constraints.

Provide recommendations or mitigation strategies for addressing identified feasibility issues, such as leveraging cloud-based resources for scalability or outsourcing data annotation tasks.

Determine whether the project is deemed feasible based on the analysis and whether it aligns with the organization's goals and resources.

3.2 SOFTWARE AND HARDWARE REQUIREMENTS

Based on the reference of IEEE Std. 830 – 1998 [19] Recommended Practice for Software Requirements, The Hardware and Software Requirements are described

Software Requirements :

Python 3.6 or higher.

Anaconda Software

OS : Windows 7 or above versions

Hardware Requirements :

System : Processor of frequency 1.5GHz or above

HardDisk : 200GB

Higher RAM, of about 4GB or above.

3.3 EXTERNAL INTERFACE REQUIRIMENTS

3.3.1 User Interfaces:

A user interface specification (UI specification) is a document that captures the details of the software user interface into a written document. The specification covers all possible actions that an end user may perform and all visual, auditory and other interaction elements. A user interface (UI) is the space where interactions between humans and machines occur.

In our process, the user choose the input by using tkinter and load or read the input by using panda's package.

3.3.2 Hardware Interfaces:

A hardware interface specifies the plugs, sockets, cables and electrical signals that pass through each line between the CPU and a peripheral device or

communications network.

In our process, there is no connection between software and hardware. Here we are used the hardware devices such as mouse, keyboard.

3.3.3 Software Interfaces:

Software interfaces (programming interfaces) are the languages, codes and messages that programs use to communicate with each other and to the hardware. Examples are the Windows, Mac and Linux operating systems, SMTP email, IP network protocols and the software drivers that activate the peripheral devices.

In our process, we are using PYTHON language to detect the different types of botnet attacks. Here we are running our process in SPYDER IDE from Anaconda Navigator.

3.3.4 Communication Interfaces:

Communications interfaces: wireless or wired technologies are used to connect devices to one another, the Internet, remote servers, etc. From: Internet of Things. In our process, the input dataset as DDOS dataset. For each of the IoT devices we trained and optimized on 2/3 of its benign data (i.e., the training set of each device). This was done to capture normal network traffic patterns. The test data of each device comprised of the remaining 1/3 of benign data plus all the malicious data. On each test set we applied the respective trained Search Similarity based for prediction as an anomaly detector. The detection of anomalies.

CHAPTER 4

DESCRIPTION OF PROPOSED SYSTEM

We use two techniques mainly CNN and LSTM for image classification. So, to make our image caption generator model, we will be merging these architectures.

- CNN is used for extracting features from the image. We will use the pre-trained model Xception.
- LSTM will use the information from CNN to help generate a description of the image.

4.1 CONVOLUTIONAL NEURAL NETWORK

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms.

Convolutional Neural networks are specialized deep neural networks which can process the data that has input shape like a 2D matrix. Images are easily represented as a 2D matrix and CNN is very useful in working with images.

It scans images from left to right and top to bottom to pull out important features from the image and combines the feature to classify images. It can handle the images that have been translated, rotated, scaled and changes in perspective.

4.2 ARCHITECTURE

LSTM stands for Long short term memory, they are a type of RNN (recurrent neural network) which is well suited for sequence prediction problems. Based on the previous text, we can predict what the next word will be. It has proven itself effective from the traditional RNN by overcoming the limitations of RNN which had short term memory.

LSTMs are designed to overcome the vanishing gradient problem and allow them to retain information for longer periods compared to traditional RNNs. LSTMs can maintain a constant error, which allows them to continue learning over numerous time-steps and backpropagate through time and layers.

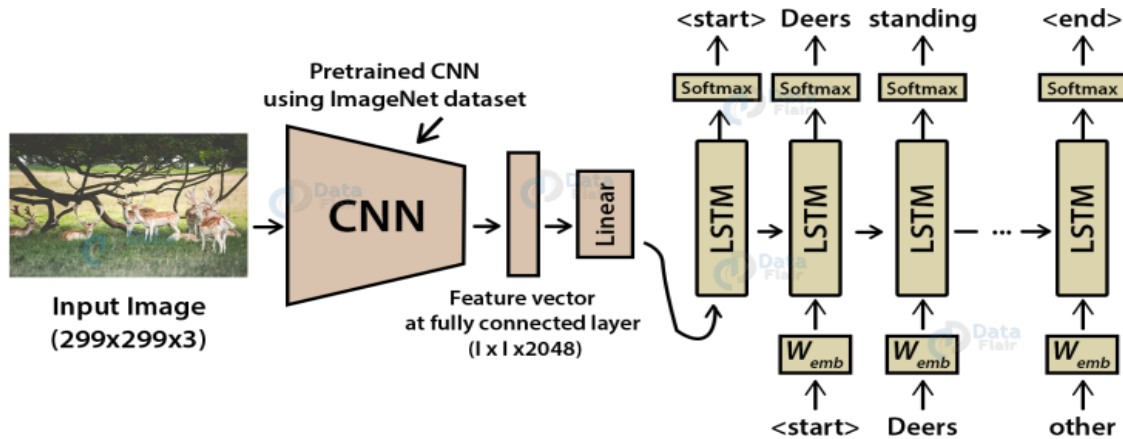


Fig: 4.1. Model, Image Caption Generator

LSTMs use gated cells to store information outside the regular flow of the RNN. With these cells, the network can manipulate the information in many ways, including storing information in the cells and reading from them. The cells are individually capable of making decisions regarding the information and can execute these decisions by opening or closing the gates. The ability to retain information for a long period of time gives LSTM the edge over traditional RNNs in these tasks.

The chain-like architecture of LSTM allows it to contain information for longer time periods, solving challenging tasks that traditional RNNs struggle to or simply cannot solve.

The three major parts of the LSTM include:

Forget gate—removes information that is no longer necessary for the completion of the task. This step is essential to optimizing the performance of the network.

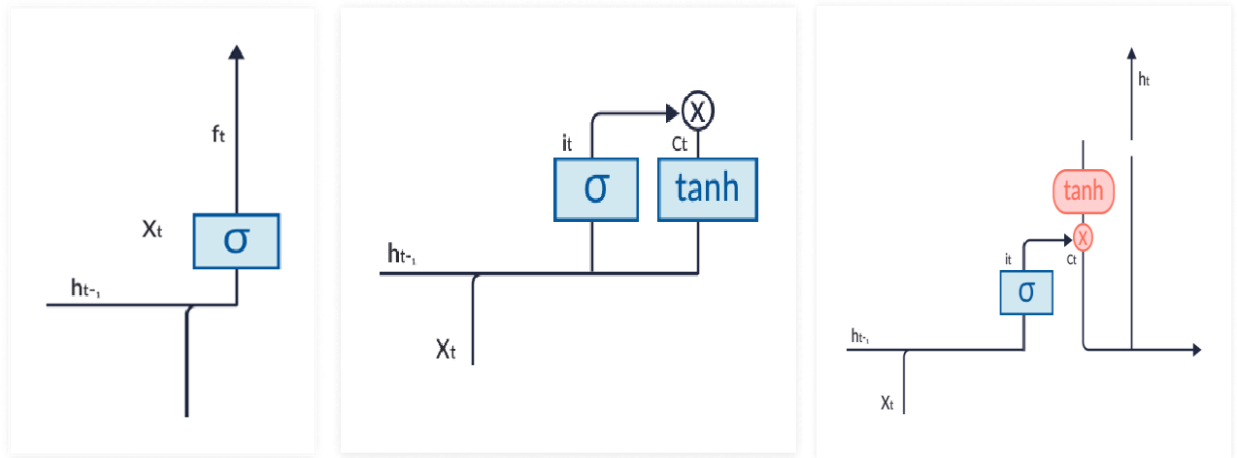


Fig: 4.2 Forget Gate, Input Gate, Output Gate

Input gate—responsible for adding information to the cells

Output gate—selects and outputs necessary information

The CNN LSTM architecture involves using Convolutional Neural Network (CNN) layers for feature extraction on input data combined with LSTMs to support sequence prediction. This architecture was originally referred to as a Long-term Recurrent Convolutional Network or LRCN model, although we will use the more generic name “CNN LSTM” to refer to LSTMs that use a CNN as a front end in this lesson.

This architecture is used for the task of generating textual descriptions of images. Key is the use of a CNN that is pre-trained on a challenging image classification task that is re-purposed as a feature extractor for the caption generating problem.

CHAPTER 5

IMPLEMENTATION

5.1 PRE-REQUISITES

This project requires good knowledge of Deep learning, Python, working on Jupyter 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
- jupyterlab

5.2 PROJECT FILE STRUCTURE

Downloaded from dataset:

Flicker8k_Dataset – Dataset folder which contains 8091 images.

Flickr_8k_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 Xception 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.

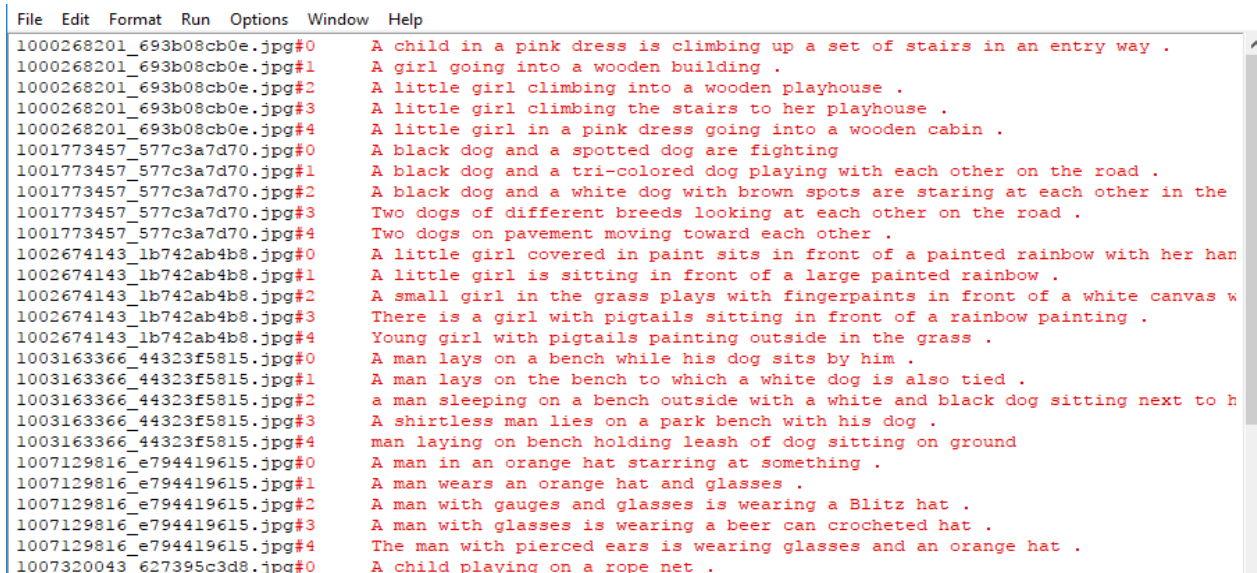
Training_caption_generator.ipynb – Jupyter notebook in which we train and build our image caption generator.

5.3 BUILDING THE PYTHON BASED PROJECT

Let's start by initializing the jupyter notebook server by typing jupyter lab 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.

5.3.1 GETTING AND PERFORMING DATA CLEANING

The main text file which contains all image captions is **Flickr8k.token** in our **Flickr_8k_text** folder.



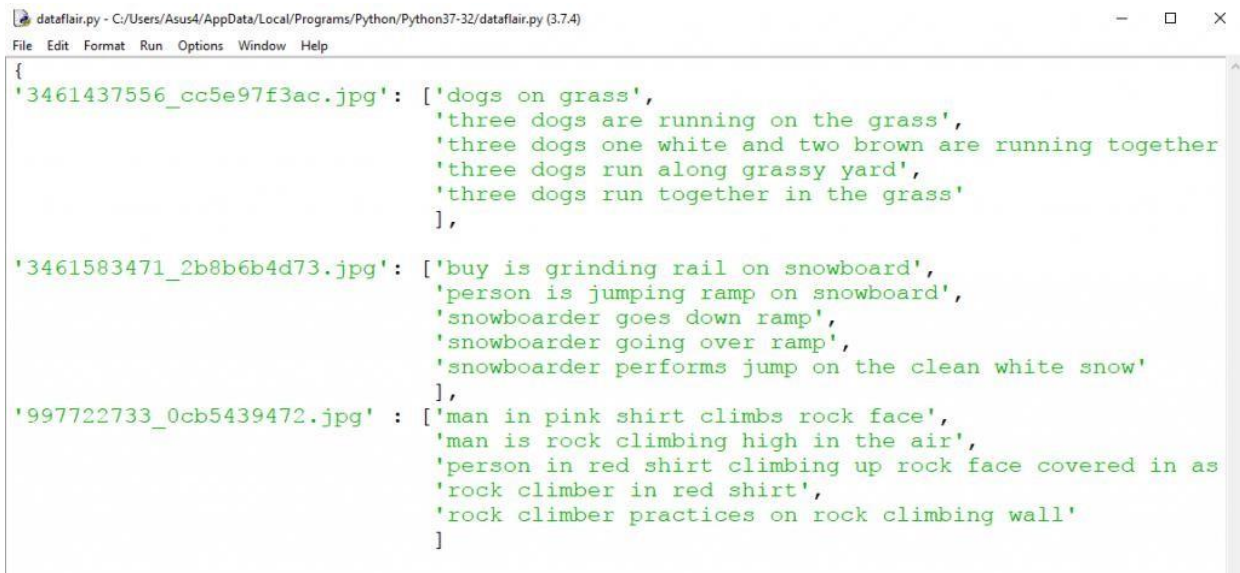
```
File Edit Format Run Options Window Help
1000268201_693b08cb0e.jpg#0 A child in a pink dress is climbing up a set of stairs in an entry way .
1000268201_693b08cb0e.jpg#1 A girl going into a wooden building .
1000268201_693b08cb0e.jpg#2 A little girl climbing into a wooden playhouse .
1000268201_693b08cb0e.jpg#3 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: 5.1. Flickr DataSet text format

The format of our file is image and caption separated by a new line (“\n”).

Each image has 5 captions and we can see that #(0 to 5)number is assigned for each caption. We will define 5 functions:

- 5.3.1.1 **load_doc(filename)** – For loading the document file and reading the contents inside the file into a string.
- 5.3.1.2 **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.



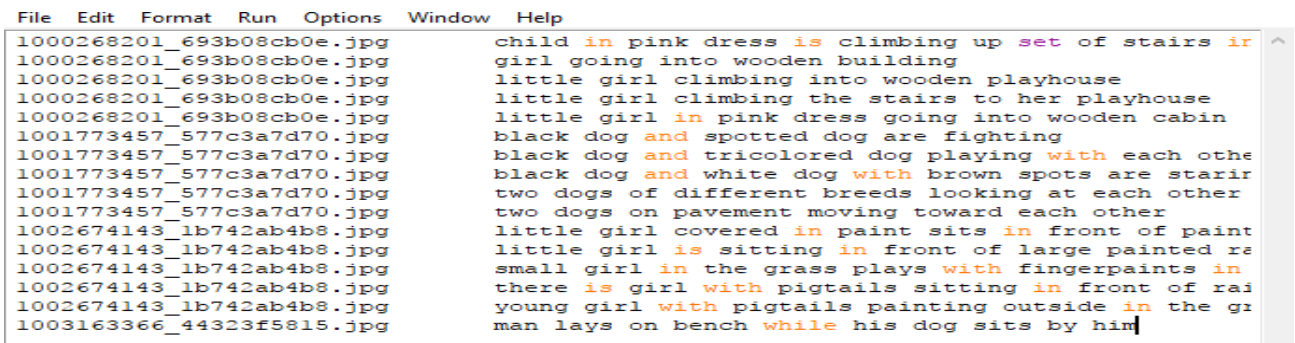
```
{
'3461437556_cc5e97f3ac.jpg': ['dogs on grass',
                              'three dogs are running on the grass',
                              'three dogs one white and two brown are running together',
                              'three dogs run along grassy yard',
                              '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: 5.2. Flickr Dataset Python File

- 5.3.1.3 **cleaning_text(descriptions)** – This function takes all descriptions and performs data cleaning. This is an important step when we work with textual data, according to our goal, we decide what type of cleaning we want to perform on the text. In our case, 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 wheelchair”

5.3.1.4 **text_vocabulary(descriptions)** – This is a simple function that will separate all the unique words and create the vocabulary from all the descriptions.

5.3.1.5 **save_descriptions(descriptions, filename)** – This function will create a list of all the descriptions that have been preprocessed and store them into a file. We will create a descriptions.txt file to store all the captions. It will look something like this:



The screenshot shows a text editor window with a menu bar (File, Edit, Format, Run, Options, Window, Help) and a list of image descriptions. Each line consists of a filename followed by a description. Words in the descriptions are highlighted in various colors (pink, blue, green, orange) to represent a word-level analysis. The descriptions include:

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

Fig: 5.3. Description of Images

5.3.2 EXTRACTING THE FEATURE VECTOR FROM ALL IMAGES

This technique is also called transfer learning, we don't have to do everything on our own, we use 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. We can directly import this model from the keras.applications . Make sure you are connected to the internet as the weights get automatically downloaded. Since the Xception model was originally built for imagenet, we will do little changes for integrating with our model. One thing to notice is that the Xception model takes 299*299*3 image size as input. We will remove the last classification layer and get the 2048 feature vector.

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

The function **extract_features()** will extract features for all images and we will map image names with their respective feature array. Then we will dump the features dictionary into a “features.p” pickle file.

This process can take a lot of time depending on your system. I am using an Nvidia 1050 GPU for training purpose so it took me around 7 minutes for performing this task. However, if you are using CPU then this process might take 1-2 hours. You can comment out the code and directly load the features from our pickle file.

5.3.3 LOADING DATASET FOR TRAINING THE MODEL

In our **Flickr_8k_test** folder, we have **Flickr_8k.trainImages.txt** file that contains a list of 6000 image names that we will use for training.

For loading the training dataset, we need more functions:

5.3.3.1 **load_photos(filename)** – This will load the text file in a string and will return the list of image names.

5.3.3.2 **load_clean_descriptions(filename, photos)** – This function will create a dictionary that contains captions for each photo from the list of photos. We also append the <start> and <end> identifier for each caption. We need this so that our LSTM model can identify the starting and ending of the caption.

5.3.3.3 **load_features(photos)** – This function will give us the dictionary for image names and their feature vector which we have previously extracted from the Xception model.

5.3.4 TOKENIZING THE VOCABULARY

Computers don’t understand English words, for computers, we will have to represent them with numbers. So, we will map each word of the vocabulary with a unique index value. Keras library provides us with the tokenizer

function that we will use to create tokens from our vocabulary and save them to a “**tokenizer.p**” pickle file.

Our vocabulary contains 7577 words. We calculate the maximum length of the descriptions. This is important for deciding the model structure parameters. Max_length of description is 32.

5.3.5 Create Data generator

Let us first see how the input and output of our model will look like. To make this task into a supervised learning task, we have to provide input and output to the model for training. We have to train our model on 6000 images and each image will contain 2048 length feature vector and caption is also represented as numbers. This amount of data for 6000 images is not possible to hold into memory so we will be using a generator.

This generator will yield the following input and output sequence.

For example:

The input to our model is [x1, x2] and the output will be y, where x1 is the 2048 feature vector of that image, x2 is the input text sequence and y is the output text sequence that the model has to predict.

Table 5.1 : Word Prediction Generation Step by Step

x1(feature vector)	x2(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

5.3.6 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:

5.3.5.1 **Feature Extractor** – The feature extracted from the image has a size of 2048, with a dense layer, we will reduce the dimensions to 256 nodes.

5.3.5.2 **Sequence Processor** – An embedding layer will handle the textual input, followed by the LSTM layer.

5.3.5.3 **Decoder** – By merging the output from the above two layers, we will process by the dense layer to make the final prediction.

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

5.3.7 Training the model

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

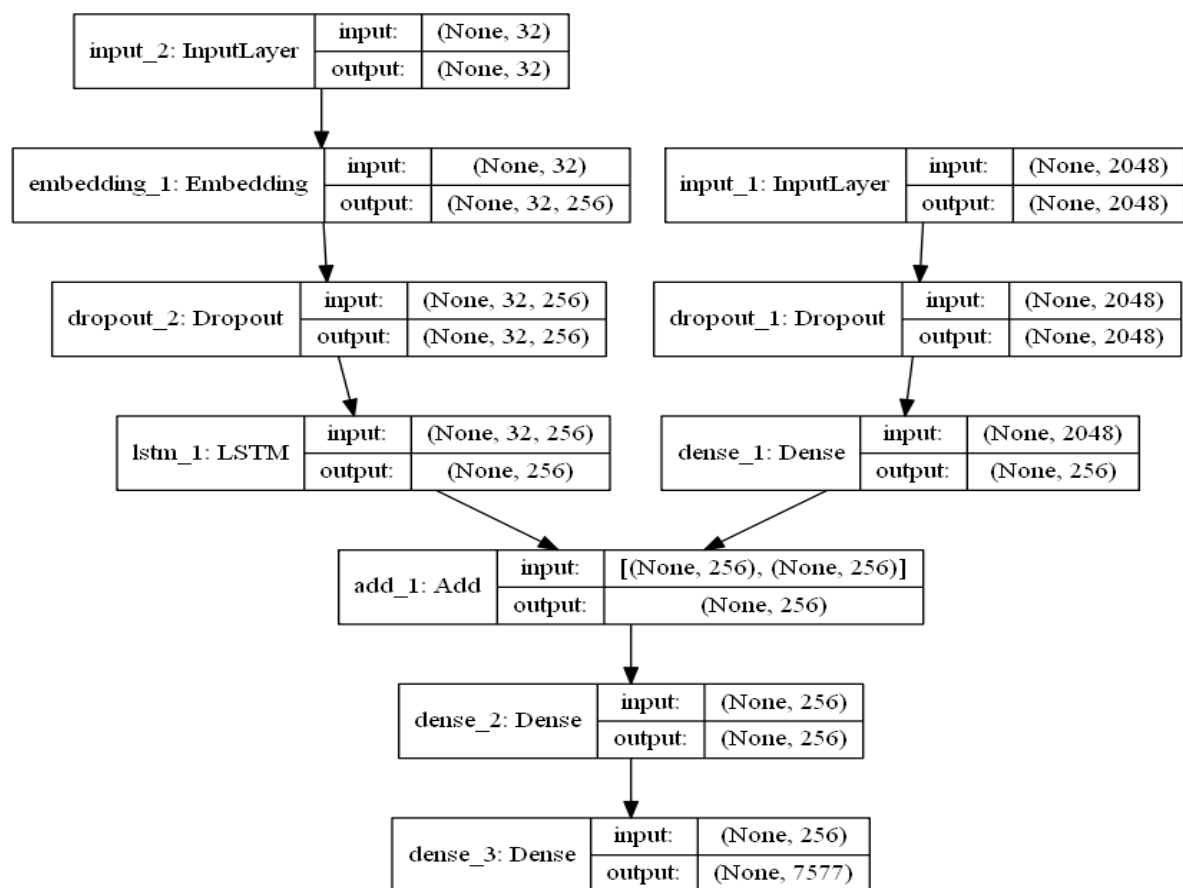


Fig: 5.4. Final Model Structure

5.3.8 Testing the model

The model has been trained, now, we will make a separate file `testing_caption_generator.py` which will load the model and generate predictions. The predictions contain the max length of index values so we will use the same `tokenizer.p` pickle file to get the words from their index values.

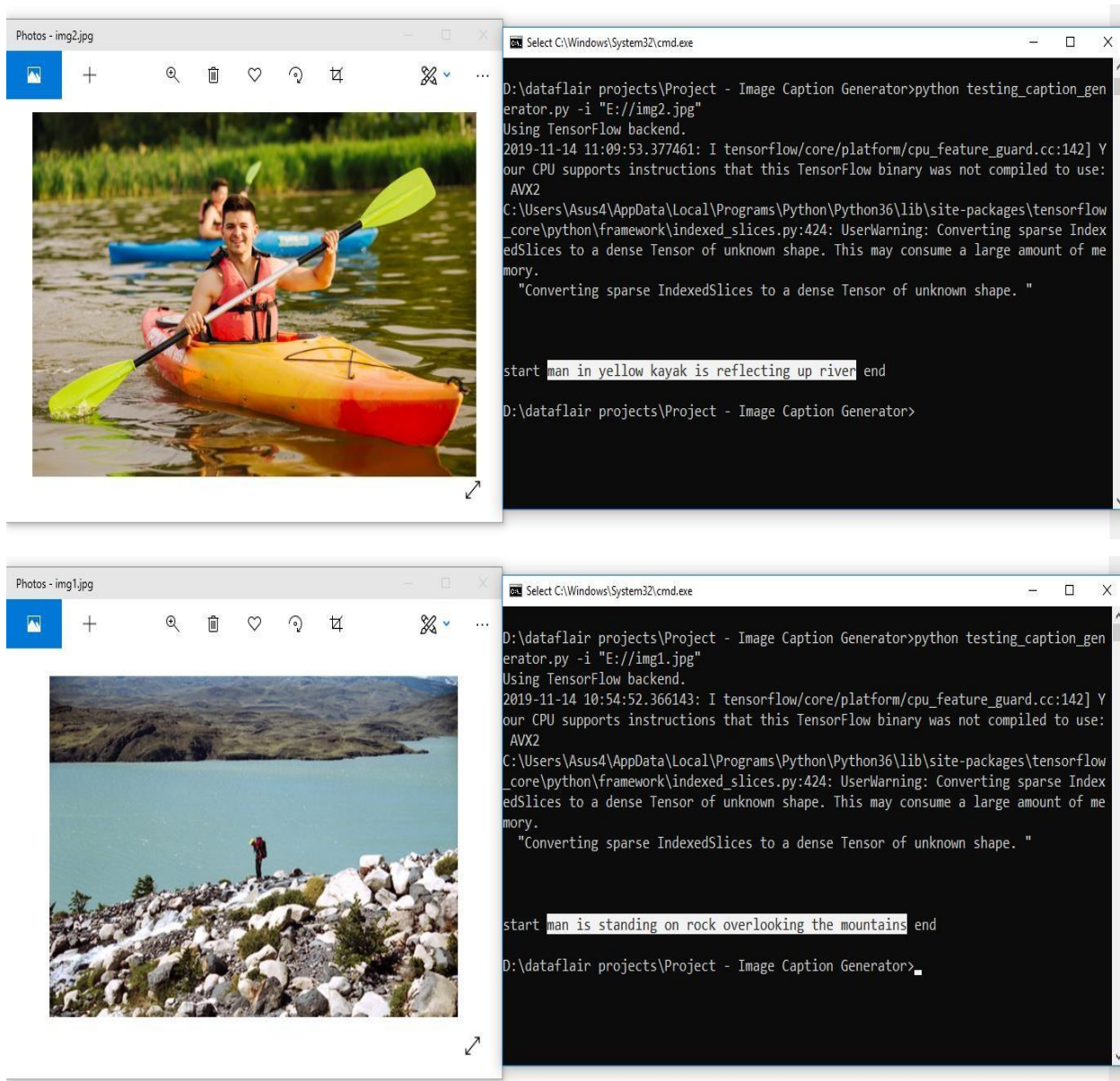


Fig: 5.5. Output Caption of Given Image

CHAPTER 6

RESULTS AND DISCUSSIONS

6.1 RESULTS

Quantitative Evaluation:

Performance Metrics: Report quantitative evaluation metrics such as BLEU, METEOR, CIDEr, and ROUGE scores to assess the quality of generated captions.

Comparison: Compare the performance of the proposed model with baseline models or state-of-the-art approaches if applicable.

Tabular Presentation: Present results in a tabular format for clarity and easy comparison.

Qualitative Evaluation:

Example Captions: Provide qualitative assessment by showcasing example images along with their generated captions.

Visual Inspection: Analyze the generated captions qualitatively to assess their relevance, coherence, and fluency.

6.2 DISCUSSIONS

Performance Analysis:

Interpretation of Metrics: Interpret the quantitative metrics to understand the strengths and weaknesses of the model.

BLEU Score Analysis: Discuss how well the generated captions match the reference captions and analyze potential discrepancies.

METEOR and CIDEr Scores: Provide insights into how well the model captures

linguistic nuances and consensus among human annotators.

Strengths of the Model:

Semantic Understanding: Highlight the model's ability to capture meaningful relationships between objects, actions, and scenes in images.

Creativity and Diversity: Discuss instances where the model generates diverse and creative captions, showcasing its capability beyond simple descriptions.

Scalability: Evaluate the model's scalability to different datasets and types of images.

Limitations and Challenges:

Ambiguity Handling: Address challenges related to ambiguous scenes or objects where the model may struggle to generate accurate captions.

Hallucination and Inaccuracies: Discuss instances of hallucinated details or inaccuracies in generated captions and potential reasons behind them.

Data Bias: Acknowledge any biases present in the training data and their impact on the model's performance.

Future Directions:

Improvement Strategies: Propose potential improvements to address the identified limitations, such as incorporating better attention mechanisms or exploring advanced language models like transformers.

Fine-tuning and Transfer Learning: Discuss the potential benefits of fine-tuning the model on specific domains or transferring knowledge from related tasks like image classification or object detection.

Ethical Considerations: Highlight the importance of ethical considerations, such as fairness, privacy, and inclusivity, in the development and deployment of image captioning systems.

Applications and Implications:

Real-World Applications: Discuss potential applications of image captioning in domains such as accessibility, content indexing, and human-computer interaction.

Social and Cultural Impact: Consider the broader societal implications of deploying image captioning technology, including its impact on language use, representation, and accessibility.

CHAPTER 7

CONCLUSION, LIMITATION AND FUTURE SCOPE

In this chapter we have thrown some light on the conclusion of our project. We have also underlined the limitation of our methodology. There is a huge possibility in this field, as we have discussed in the future scope section of this chapter.

7.1 CONCLUSION

We have reviewed deep learning-based image captioning methods. We have given a taxonomy of image captioning techniques, shown a generic block diagram of the major groups and highlighted their pros and cons. We discussed different evaluation metrics and datasets with their strengths and weaknesses. A brief summary of experimental results is also given. We briefly outlined potential research directions in this area. Although deep learning-based image captioning methods have achieved a remarkable progress in recent years, a robust image captioning method that is able to generate high quality captions for nearly all images is yet to be achieved. With the advent of novel deep learning network architectures, automatic image captioning will remain an active research area for some time. We have used Flickr_8k dataset which includes nearly 8000 images, and the corresponding captions are also stored in the text file. Although deep learning-based image captioning methods have achieved a remarkable progress in recent years, a robust image captioning method that is able to generate high quality captions for nearly all images is yet to be achieved. With the advent of novel deep learning network architectures, automatic image captioning will remain an active research area for sometime. The scope of image-captioning is very vast in the future as the users are increasing day by day on social media and most of them would post photos.

7.2 LIMITATIONS

The neural image caption generator gives a useful framework for learning to map from images to human-level image captions. By training on large numbers of image-caption pairs, the model learns to capture relevant semantic information from visual features.

However, with a static image, embedding our caption generator will focus on features of our images useful for image classification and not necessarily features useful for caption generation. To improve the amount of task-relevant information contained in each feature, we can train the image embedding model (the VGG-16 network used to encode features) as a piece of the caption generation model, allowing us to fine-tune the image encoder to better fit the role of generating captions.

Also, if we actually look closely at the captions generated, we notice that they are rather mundane and commonplace. Take this possible image-caption pair for instance:

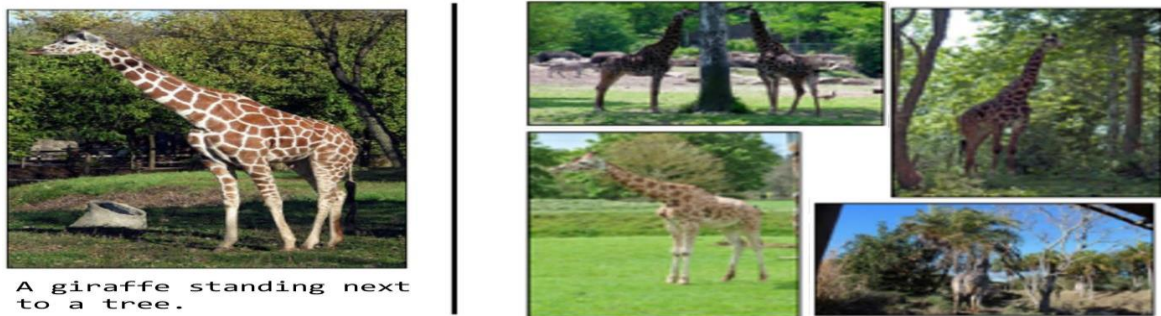


Fig: 7.1. The above picture depicts clear limitation the Model because it relies most on the training dataset.

This is most certainly a “giraffe standing next to a tree.” However, if we look at other pictures, we will likely notice that it generates a caption of “a giraffe next to a tree” for any picture with a giraffe because giraffes in the training set often appear near trees.

7.3 FUTURE SCOPE

Future work Image captioning has become an important problem in recent days due to the exponential growth of images in social media and the internet. This report discusses the various research in image retrieval used in the past and it also highlights the various techniques and methodology used in the research. As feature extraction and similarity calculation in images are challenging in this domain, there is a tremendous scope of possible research in the future. Current image retrieval systems use similarity calculation by making use of features such as color, tags, IMAGE RETRIEVAL USING IMAGE CAPTIONING 54 histogram, etc. There cannot be completely accurate results as these methodologies do not depend on the context of the image. Hence, a complete research in image retrieval making use of context of the images such as image captioning will facilitate to solve this problem in the future. This project can be further enhanced in future to improve the identification of classes which has a lower precision by training it with more image captioning datasets. This methodology can also be combined with previous image retrieval methods such as histogram, shapes, etc. and can be checked if the image retrieval results get better.

REFERENCES

1. Abhaya Agarwal and Alon Lavie. 2008. Meteor, m-bleu and m-ter: Evaluation metrics for high-correlation with human rankings of machine output. In *Proceedings of the Third Workshop on Statistical Machine Translation*. Association for Computational Linguistics, 115–118.
2. Ahmet Aker and Robert Gaizauskas. 2010. Generating image descriptions using dependency relational patterns. In *Proceedings of the 48th annual meeting of the association for computational linguistics*. Association for Computational Linguistics, 1250–1258.
3. Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. 2016. Spice: Semantic propositional imagecaption evaluation. In *European Conference on Computer Vision*. Springer, 382–398.
4. Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. 2017. Bottom-up and top-down attention for image captioning and vqa. *arXiv preprint arXiv:1707.07998* (2017).
5. Jyoti Aneja, Aditya Deshpande, and Alexander G Schwing. 2018. Convolutional image captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5561–5570.
6. Lisa Anne Hendricks, Subhashini Venugopalan, Marcus Rohrbach, Raymond Mooney, Kate Saenko, Trevor Darrell,
7. Junhua Mao, Jonathan Huang, Alexander Toshev, Oana Camburu, et al. 2016. Deepcompositional captioning.
8. Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*.
9. Shuang Bai and Shan An. 2018. A Survey on Automatic Image Caption Generation. *Neurocomputing*. *ACM Computing Surveys*, Vol. 0, No. 0, Article 0. Acceptance Date: October 2018.0:30 Hossain et al.
10. Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, Vol. 29. 65–72.
11. Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*. 1171–1179.

12. Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research* 3, Feb, 1137–1155.
13. Adam L Berger, Vincent J Della Pietra, and Stephen A Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational linguistics* 22, 1 (1996), 39–71.
14. Raffaella Bernardi, Ruket Cakici, Desmond Elliott, Aykut Erdem, Erkut Erdem, Nazli Ikizler-Cinbis, Frank Keller, Adrian Muscat, Barbara Plank, et al. 2016. Automatic Description Generation from Images: A Survey of Models, Datasets, and Evaluation Measures. *Journal of Artificial Intelligence Research (JAIR)* 55, 409–442.
15. David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research* 3, Jan (2003), 993–1022.
16. Cristian Bodnar. 2018. Text to Image Synthesis Using Generative Adversarial Networks. *arXiv preprint arXiv:1805.00676*.
17. Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 1247–1250.
18. Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. 1992. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 144–152.
19. Andreas Bulling, Jamie A Ward, Hans Gellersen, and Gerhard Troster. Eye movement analysis for activity recognition using electrooculography. *IEEE*

APPENDIX

A. Source Code:

```
# 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 punctuations 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\Flickr8k_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"

class RNN_Decoder(tf.keras.Model):
    def __init__(self, embedding_dim, units, vocab_size):
        super(RNN_Decoder, self).__init__()
        self.units = units

```

```

# input_dim = size of the vocabulary
# Define the embedding layer to transform the input caption sequence
self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)

# Define the Long Short Term Memory layer to predict the next words in the
sequence
self.lstm = tf.keras.layers.LSTM(self.units, return_sequences=True,
return_state=True)

# Define a dense layer to transform the LSTM output into prediction of the
best word
self.fc = tf.keras.layers.Dense(vocab_size) #, activation='softmax')

# A function that transforms the input embeddings and passes them to the
LSTM layer
def call(self, captions, features, omit_features = False, initial_state = None,
verbose = False):
    if verbose:
        print("Before embedding")
        print(captions.shape)

    embed = self.embedding(captions) #(batch_size, 1, embedding_dim)

    if verbose:
        print("Embed")
        print(embed.shape)

    features = tf.expand_dims(features, 1)

    if verbose:
        print("Features")
        print(features.shape)

    # Concatenating the image and caption embeddings before providing them to
LSTM
    # shape == (batch_size, 1, embedding_dim + hidden_size)
    lstm_input = tf.concat([features, embed], axis=-2) if (omit_features == False)
    else embed

    if verbose:
        print("LSTM input")
        print(lstm_input.shape)

    # Passing the concatenated vector to the LSTM
    output, memory_state, carry_state = self.lstm(lstm_input,
initial_state=initial_state)

    if verbose:
        print("LSTM output")

```

```

        print(output.shape)

    # Transform LSTM output units to vocab_size
    output = self.fc(output)

    return output, memory_state, carry_state

def reset_state(self, batch_size):
    return tf.zeros((batch_size, self.units))

units = embedding_dim = 512 # As in the paper
vocab_size = min(top_k + 1, len(tokenizer.word_index.keys()))

# Initialize encoder and decoder
encoder = CNN_Encoder(embedding_dim)
decoder = RNN_Decoder(embedding_dim, units, vocab_size)

# Initialize optimizer
optimizer = tf.keras.optimizers.Adam()

# As the label is not one-hot encoded but indices. Logits as they are not
probabilities.
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True,
reduction='none')

# Computes the loss using SCCE and calculates the average of singular losses in
the tensor
def loss_function(real, pred, verbose=False):
    loss_ = loss_object(real, pred)

    if verbose:
        print("Loss")
        print(loss_)

    loss_ = tf.reduce_mean(loss_, axis = 1)

    if verbose:
        print("After Mean Axis 1")
        print(loss_)

    return loss_

# Key Point: Any Python side-effects (appending to a list, printing with print, etc)
will only happen once, when func is traced.
# To have side-effects executed into your tf.function they need to be written as TF
ops:
@tf.function
def train_step(img_tensor, target, verbose=False):
    if verbose:
        print("Image tensor")

```

```

print(img_tensor.shape)

print("Target")
print(target.shape)

# The input would be each set of words without the last one (<end>), to leave
space for the first one that
# would be the image embedding
dec_input = tf.convert_to_tensor(target[:, :-1])

# Source: https://www.tensorflow.org/api_docs/python/tf/GradientTape
with tf.GradientTape() as tape:
    features = encoder(img_tensor)

    if verbose:
        print("Features CNN")
        print(features)

    predictions, _, _ = decoder(dec_input, features, verbose=verbose)

    if verbose:
        print("Predictions RNN")
        print(predictions)

    caption_loss = loss_function(target, predictions) # (batch_size, )

    # After tape
    total_batch_loss = tf.reduce_sum(caption_loss) # Sum (batch_size, ) => K
    mean_batch_loss = tf.reduce_mean(caption_loss) # Mean(batch_size, ) => K

    # Updated the variables
    trainable_variables = encoder.trainable_variables + decoder.trainable_variables
    gradients = tape.gradient(caption_loss, trainable_variables)
    optimizer.apply_gradients(zip(gradients, trainable_variables))

    return total_batch_loss, mean_batch_loss

EPOCHS = 5
start_epoch = 0

for epoch in range(start_epoch, EPOCHS):
    real_epoch = len(loss_plot) + 1
    start = time()
    total_loss = 0

    for (batch, (img_tensor, target)) in enumerate(dataset):
        total_batch_loss, mean_batch_loss = train_step(img_tensor, target,
        verbose=False)
        total_loss += total_batch_loss

```

```

        if batch % 100 == 0:
            print ('Epoch {} Batch {} Batch Loss {:.4f}'.format(real_epoch, batch,
mean_batch_loss.numpy()))

        print ('Total Loss {:.6f}'.format(total_loss))
        epoch_loss = total_loss / NUM_STEPS

        # storing the epoch end loss value to plot later
        loss_plot.append(epoch_loss)

        if epoch % 5 == 0:
            ckpt_manager.save()

        print ('Epoch {} Epoch Loss {:.6f}'.format(real_epoch, epoch_loss))
        print ('Time taken for 1 epoch {} sec\n'.format(time() - start))

```

B. SCREENSHOTS

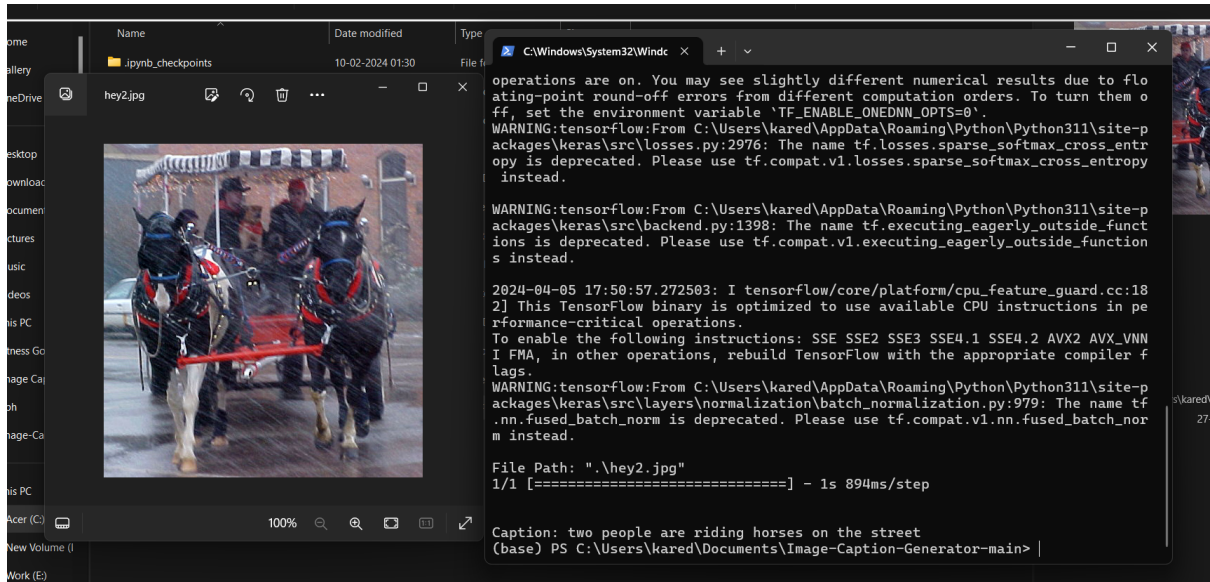


Fig: B.1

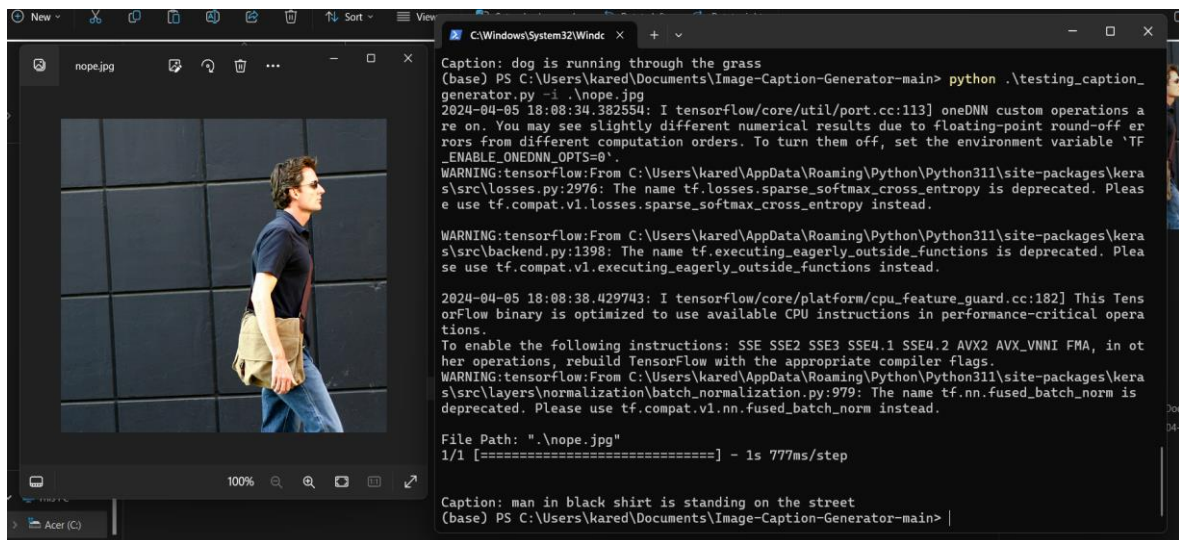


Fig: B.2

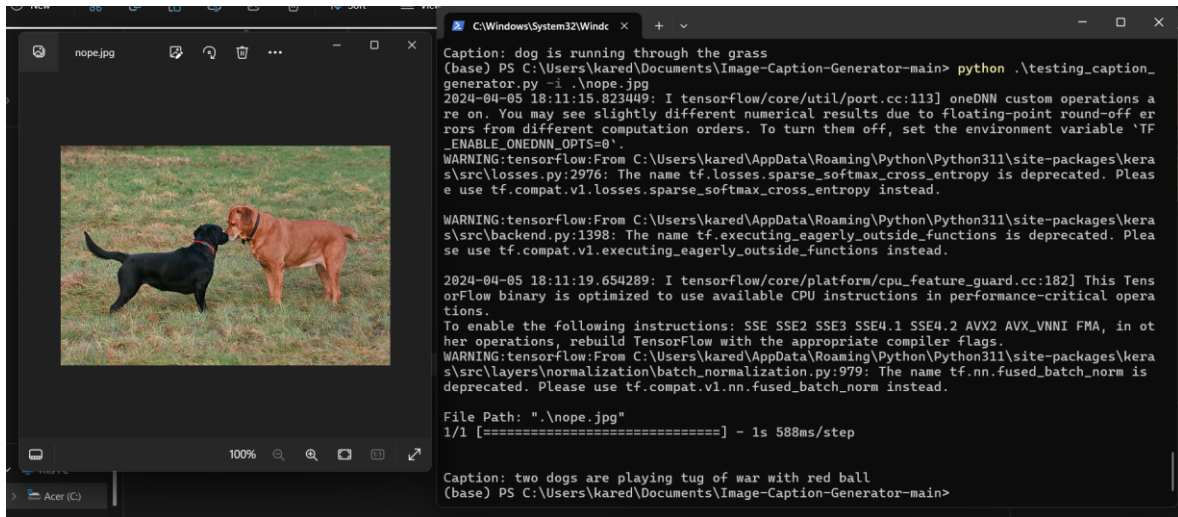


Fig: B.3

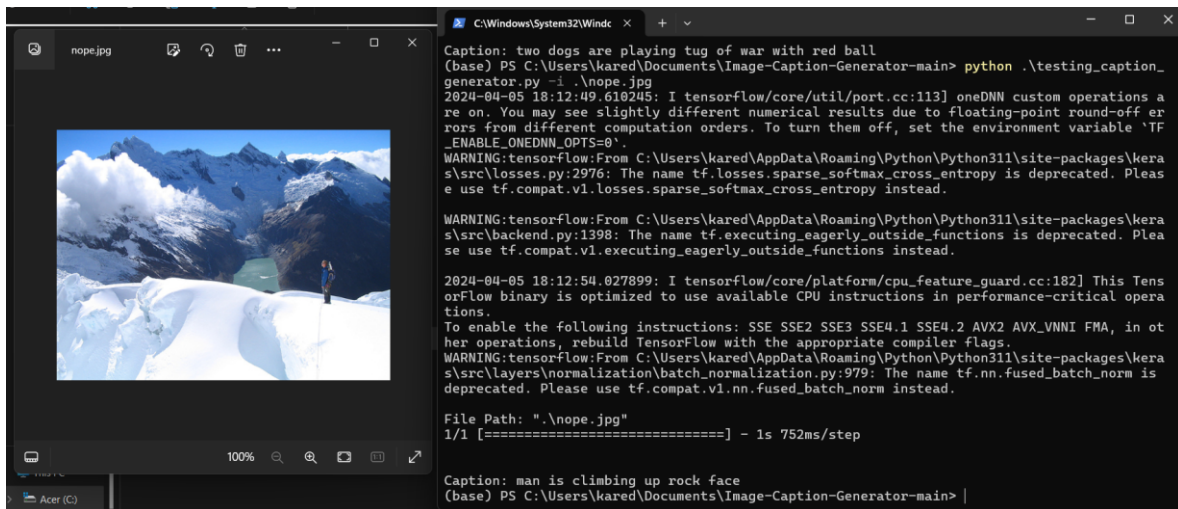


Fig: B.4