TRIBHUVAN UNIVERSITY

INSTITUTE OF SCIENCE AND TECHNOLOGY

A PROJECT REPORT ON

IMAGE CAPTION GENERATOR

SUBMITTED TO

DEPARTMENT OF STATISTICS AND COMPUTER SCIENCE
PATAN MULTIPLE CAMPUS
Patan Dhoka, Lalitpur

In partial fulfillment of the requirements for the Bachelor's degree in Computer Science and Information Technology (B.Sc. CSIT)

Submitted by:

AASTHA OLI (20001/075) [5-2-22-380-2018]

PRAKRITI PUDASAINI (20063/075) [5-2-22-431-2018]

SONIKA ACHARYA (20093/075) [5-2-22-461-2018]

Under the Supervision of

Dadhi Ram Ghimire

May, 2023

TRIBHUVAN UNIVERSITY

INSTITUTE OF SCIENCE AND TECHNOLOGY

A PROJECT REPORT ON

IMAGE CAPTION GENERATOR

SUBMITTED TO

DEPARTMENT OF STATISTICS AND COMPUTER SCIENCE
PATAN MULTIPLE CAMPUS
Patan Dhoka, Lalitpur

In partial fulfillment of the requirements for the Bachelor's degree in Computer Science and Information Technology (B.Sc. CSIT)

Submitted by:

AASTHA OLI (20001/075) [5-2-22-380-2018]

PRAKRITI PUDASAINI (20063/075) [5-2-22-431-2018]

SONIKA ACHARYA (20093/075) [5-2-22-461-2018]

Under the Supervision of

Dadhi Ram Ghimire

May, 2023

ACKNOWLEDGEMENT

We would like to express our sincere gratitude towards the Institute of Science and Technology, Tribhuvan University for the inclusion of the major project in the course of Bachelors in Computer Science and Information Technology (BSc.CSIT).

We are thankful to the Department of Computer Science and Information Technology, Patan Multiple Campus for constant guidance and supervision, as well as for providing all the necessary resources for the successful completion of the project.

We would like to extend our sincere regard to our supervisor Mr. Dadhi Ram Ghimire for his crucial supervision, constructive comments and suggestions, which gave the final shape of this project. We would also like to thank the faculty members of campus, all of our friends, seniors and colleagues for their valuable suggestions, comments and support.

Aastha Oli

Prakriti Pudasaini

Sonika Acharya

ABSTRACT

Image captioning is becoming one of the most frequently used technologies in the modern day. Deep neural network models are used to generate descriptions of the images. Image captioning is the process of creating a description for an image.

Automatically describing the content of an image is a fundamental problem in artificial intelligence that connects computer vision and natural language processing. It requires both methods from computer vision to understand the content of the image and a language model from the field of natural language processing to turn the understanding of the image into words in the right order. It requires recognizing the important objects, their attributes, and the relationships among the objects in an image. It generates syntactically and semantically correct sentences.

In this project, we present a model that generates a description of an image. The important aspect in implementing the image caption generator project includes data processing using Flickr 30k dataset, then extracting the features using CNN and generating captions using LSTM.

Keywords: *Convolutional Neural Network (CNN), Long short-term memory (LSTM), Image Caption Generator, Computer Vision, Natural Language Processing.*

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

BLEU	Bilingual Evaluation Understudy
CNN	Convolutional Neural Network
CV	Computer Vision
GPU	Graphics Processing Unit
LSTM	Long Short-Term Memory
METEOR	Metric for Evaluation for Translation with Explicit Ordering
MSCOCO	Microsoft Common Objects in Context
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
RAM	Random Access Memory
ReLU	Rectified Linear Unit
ResNet	Residual Network
RNN	Recurrent Neural Network
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
SGD	Stochastic Gradient Descent
UI	User Interface

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Introduction

Image caption generator is a process of recognizing the context of an image and annotating it with relevant captions. Indeed, a description must capture not only the objects contained in an image, but it also must express how these objects relate to each other as well as their attributes and the activities they are involved in. The important aspects in caption generators for images involve CNN (Convolutional Neural Network) that extracts the features of the images and LSTM (Long short-term memory) which generates the caption from the extracted information of the image.

Initially, it was considered impossible that a computer could describe an image. With advancement of Deep Learning Techniques, and large volumes of data available, we can now build models that can generate captions describing an image.

The architecture of the model comprises two networks that are added together. Firstly, a CNN model, trained on an image classification task which is used to capture the details of the image. Moving forward, a Language generating model (RNN) is used to generate the appropriate caption given the image mappings. The task makes the CNN an image encoder and the fixed length vector mapping from the CNN is fed as the input to the RNN decoder. This decoder then generates the sentences.

In this project we are going to build one such annotation tool which is capable of generating very relevant captions for the image with the help of datasets.

1.2 Problem Statement

Describing the content of an image using properly formed English language is a challenging task for a machine. The task requires an algorithm to not only understand the content of the image, but also to generate language that connects to its interpretation. Mimicking the human ability of providing descriptions for images by a machine is itself a remarkable step along the line of Artificial Intelligence. Traditionally, computer systems have been using predefined templates for generating text descriptions for images. However, these approaches do not provide sufficient variety required for generating lexically rich text descriptions. Neural network models have been put forward to suppress the given issue. The proposed concept could have great impact, for instance by helping visually impaired people better understand the content of images on the web.

1.3 Objectives

- To generate accurate and relevant textual descriptions for images that are semantically meaningful and grammatically correct.
- To provide a better understanding of the content and context of images, which can help with tasks such as content-based image retrieval, visual search, and image recommendation.
- To assist visually impaired individuals in understanding the content of images, by providing them with accurate and detailed textual descriptions.
- To develop a deeper understanding of natural language processing and computer vision and improve skills in deep learning frameworks and techniques.

1.4 Scopes and Limitations

1.4.1 Scope

With the advancements in deep learning techniques, the accuracy of image caption generators has improved significantly, making them more useful for a wide range of applications. Image caption generators can provide semantically meaningful descriptions that can help improve image retrieval, visual search, and image recommendation systems. Image captioning can be used to generate captions for social media posts, blog posts, and other types of content. This can save time for content creators who would otherwise need to manually write captions for each image they use. Image captioning can help people with visual impairments to understand the content of images that they would not otherwise be able to see. By generating descriptive captions for images, machine learning models can make the content of these images accessible to a wider audience.

1.4.2 Limitation

- This project is limited by the biases in the training data, which can result in inaccurate or inappropriate descriptions for certain images.
- The project may not always be able to accurately understand the context of an image, which can result in misleading or irrelevant descriptions.
- The project may not always produce creative or unique descriptions, which can limit their usefulness for certain applications.
- The project can be computationally expensive, which can limit their scalability and accessibility.
- The project may not always produce grammatically correct or fluent descriptions, particularly for complex or ambiguous images.

1.5 Development Methodology

Data Collection

We have identified the three most commonly used image caption training datasets in the Computer Vision research domain - MS COCO dataset, Flickr8k and Flickr30k. These datasets contain 123,000, 31,000 and 8,000 caption annotated images respectively and each image is labelled with 5 different descriptions. Currently, the Flickr30k dataset which contains 31,783 images collected from Flickr with 5 captions per image, totaling 158,915 captions is used as our primary data source.

Data Preprocessing

Input data is composed of images and captions, and hence we need to pre-process both the images into proper format for CNN network and the text captions into proper format for RNN network. Since our image caption generator pipeline is leveraging pre-trained CNN networks we need to transform images into the correct format. Text preprocessing includes simple techniques of converting all the characters into lowercase and removing all the non-alphabetic characters including punctuations and special characters.

Image Encoding (Convolutional Neural Network)

The encoder needs to extract image features of various sizes and encodes them into vector space which can be fed to RNN in a later stage. With the progress of the project we have used ResNet50 architecture from tensorflow as pre-trained CNN architecture.

In this task, CNN is used to encode features instead of classifying images. As a result, we removed the fully connected layers and the max pool layers at the end of the network.

Text Decoding (Recurrent Neural Network)

In the context of image captioning, the features extracted by the convolutional layers are typically used to initialize a recurrent neural network (RNN) that generates a sequence of words representing the image's caption.

The decoder needs to generate image captions word by word using a Recurrent Neural Network - LSTMs which are able to sequentially generate words. The input to the RNN is a fixed-length vector of image features extracted using a CNN. The RNN then generates a sequence of words one at a time, with each word conditioned on the previous words in the sequence and the image features.

Model Training

The CNN and RNN models are then trained together on the image-caption dataset. The CNN is used to extract image features, which are then fed into the RNN to generate captions. The RNN is trained to minimize the difference between the predicted captions and the actual captions.

Evaluation

The nature of our RNN output is a series likelihood of words' occurrences, and in order to quantify the quality of the RNN output, we propose to use categorical cross entropy loss. We have used Adam optimizer to optimize the model. Adam (Adaptive Moment Estimation) is an optimization algorithm used in deep learning to update the weights of the neural network during training. It is a stochastic gradient descent based optimizer that can efficiently handle large datasets and high-dimensional parameter spaces.

We have used the BLEU score (sentence BLEU) as an evaluation metric for cross validation along with human evaluation.

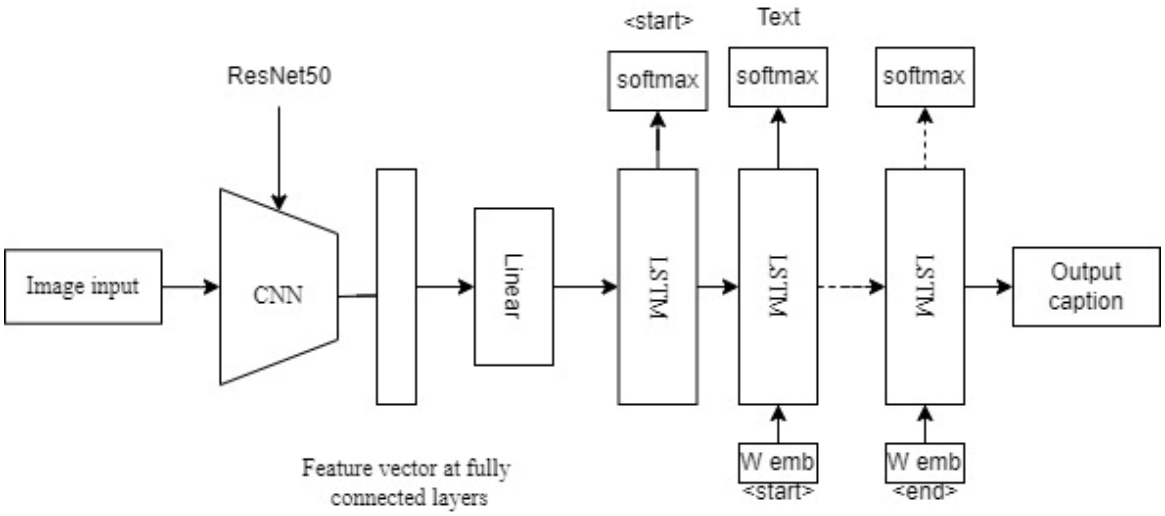

Figure 1.1: Development Methodology

1.6 Report Organization

This report is divided into 6 chapters. Each chapter discusses different issues related to this project. The outline of each chapter is stated below. A basic introduction about the image caption generator has been described in chapter 1. The project statement, objectives, and applications of the project have also been described in this chapter. Chapter 2 covers the important background information and history about the different techniques used regarding the image caption generating. Chapter 3 gives information about functional, non-

functional and system requirements of the application. Design of the system and details of the algorithm that we use throughout the development is discussed in chapter 4. Chapter 5 covers implementation, the overall overview of the project and system testing of the project. Finally, the conclusion drawn from this project by the team members is covered in chapter 6.

CHAPTER 2: BACKGROUND STUDY AND LITERATURE REVIEW

2.1 Background Study

In the last few years, the problem of generating descriptive sentences automatically for images has gained a rising interest in Natural language processing (NLP). Image captioning is a task where each image must be understood properly and be able to generate suitable captions with proper grammatical structure.

The image caption generator task involves extracting visual features from the image using deep learning techniques, and then using those features to generate a coherent and descriptive sentence that accurately describes the content of the image. The generated caption should be both grammatically correct and semantically meaningful.

The development of image caption generators has important applications in fields such as visual search, content-based image retrieval, and assistive technology for the visually impaired. The ability to automatically generate captions for images can help users to search for images more effectively, as well as improve accessibility for those with visual impairments.

Evaluation of image caption generators typically involves using metrics such as BLEU, METEOR, and ROUGE, which measure the similarity between the generated captions and human-written captions for the same image. In recent years, there has been a growing interest in developing more advanced evaluation metrics that take into account semantic similarity and other factors.

There are a series of relevant research papers attempting to accomplish this task in the last decades, but they face various problems such as grammar problems, cognitive absurdity and content irrelevance. However, with the unparalleled advancement in Neural Networks, some groups started exploring Convolutional Neural Network (CNN) and Recurrent Neural Network (CNN) to accomplish this task and observed very promising results.

2.2 Literature Review

There are a series of relevant research papers attempting to accomplish this task in the last decades, but they face various problems such as grammar problems, cognitive absurdity and content irrelevance.

With the unparalleled advancement in Neural Networks, some groups started exploring Convolutional Neural Networks and recurrent neural networks to accomplish the task of image caption generator and observed very promising results.

The work proposed in [1] uses a combination of a deep Convolutional Neural Network and a Recurrent Neural Network to achieve the task of image caption generator. In addition, the work in [2] is built upon the deep CNN and a RNN by adding attention mechanisms. Many of the methods for image caption generation are based on recurrent neural networks and inspired by the successful use of sequence to sequence training with neural networks for machine translation [3].

One major reason image caption generation is well suited to the encoder-decoder framework (of machine translation) is because it is analogous to "translating" an image to a sentence. The first approach to use neural networks for caption generation was [4], who proposed a multimodal log-bilinear model that was biased by features from the image. This work was later followed by [5] whose method was designed to explicitly allow a natural way of doing both ranking and generation. Unlike [4] whose models see the image at each time step of the output word sequence, [1] only shows the image to the RNN at the beginning. [6] proposed an extension of the long short term memory (LSTM) model, coined as gLSTM for short. In particular, semantic information was extracted from the image as extra input to each unit of the LSTM block, with the aim of guiding the model towards solutions that are more tightly coupled to the image content. Prior to the use of neural networks for generating captions, two main approaches were dominant. The first involved generating caption templates which were filled in based on the results of object detections and attribute discovery. The second approach was based on first retrieving similar captioned images from a large database then modifying these retrieved captions to fit the query.

The approach in [7] draws on the success of the top-down image generation models which uses a deep convolutional neural network to generate a vectorized representation of an image and then feed into a Long-Short-Term Memory (LSTM) network, which then generates captions.

In this project we have combined the CNN based model with LSTM architecture to generate the textual descriptions of the images. The closest work to the project [8] uses a ResNet50 and LSTM with soft attention to conduct the automatic image captioning.

The encoder adopts ResNet50 based on the convolutional neural network, which creates an extensive representation of the given image by embedding it into a fixed length vector.

The decoder is designed with LSTM, a recurrent neural network and a soft attention mechanism, to selectively focus the attention over certain parts of an image to predict the next sentence. In contrast to this, our project uses the deep convolutional network, ResNet50 [9] to extract the features of the images to a fixed length vector with Recurrent Neural Network, LSTM [10] for sequence modelling to create a single network that generates descriptions of images.

The work in [11] proposes a method of automatic machine translation evaluation that is quick, inexpensive, and language-independent, that correlates highly with human evaluation called a BLEU score. In this work, we evaluate the performance of the model in generating the caption using BLEU score.

In 2011, the introduction of the Flickr30k dataset, a dataset containing 31,783 images with five captions each, provided researchers with a larger and more diverse dataset for training image captioning models.

CHAPTER 3: SYSTEM ANALYSIS

3.1 System Analysis

3.1.1 Requirements Analysis

The image caption generation problem, which is time and cost-efficient to be solved with programming skills. Hence, the requirements for the project are basic hardware requirements and basic software tools.

It describes the hardware and software components of a system required to develop and use software efficiently.

Hardware Requirements

The computer must contain a memory RAM (at least 8 GB), it is recommended that the computer also contains a GPU, which will make the training, and processing of the model fast and efficient.

Software Requirements

The system was developed entirely on Python and its libraries. Python is a multi-pattern, general-purpose, interpreted, high-level programming language. We made the use of Google Colaboratory to work with the project as it provides free access to computing resources including GPU. Google Drive is used along with Google Colaboratory to save and load data required for the project. Kaggle also provides an online Jupyter Notebook environment.

The requirements may be functional and non-functional and both are essential to a successful software project. The following Functional and non-functional requirements were identified on the system.

Functional Requirements

The functional requirements define the function of the system where functions are the collective sets of input, their behaviors and their output. The application should be able to accurately generate the description for the given image.

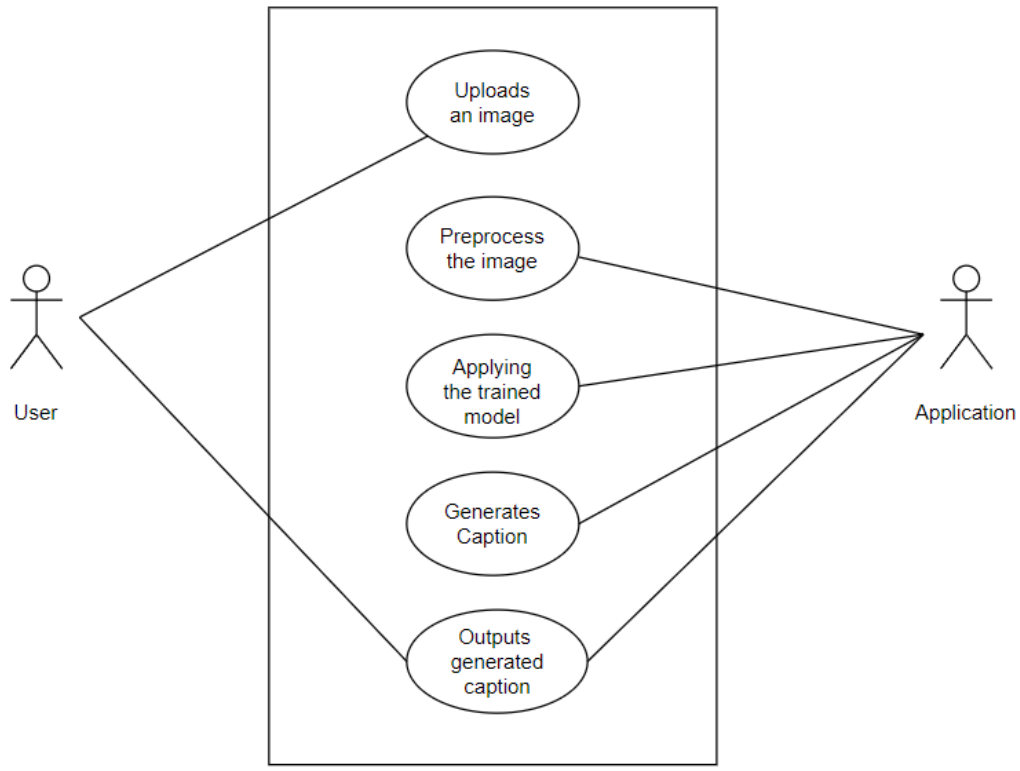

Figure 3.1: Use Case Diagram of the System

Non-functional Requirements

The non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behavior. The application should be consistent at generating captions whenever the same image is passed into the pipeline.

3.1.2 Feasibility Analysis

A feasibility study can help to identify if a given project should be undertaken by delineating costs and risks associated with various areas of the project's development.

Technical

As a hardware part, all we need is a working desktop or laptop and an internet connection. We will be working on Google Colaboratory notebook for generating the model. For the implementation we will be simply going through Streamlit library provided by Python to generate basic UI where the users can upload the images and can get the caption as a result.

Primary Coding Language: Python

Required Libraries: Numpy, OpenCV, Keras, Tensorflow, Streamlit, NLTK

Operational

This project is fully online based thus no additional hardware components are required to develop or operate it. The project will be implemented in a way that it will allow the functioning of image captioning generation smoothly. It will provide a user friendly user interface in a modular fashion.

Economic

This project is economically feasible as hardware components are already available for us and the software components used are free-to-use and open source. Once the project is ready and running, we will need additional costs for enriching the features which should not be a huge economic obstacle.

Schedule

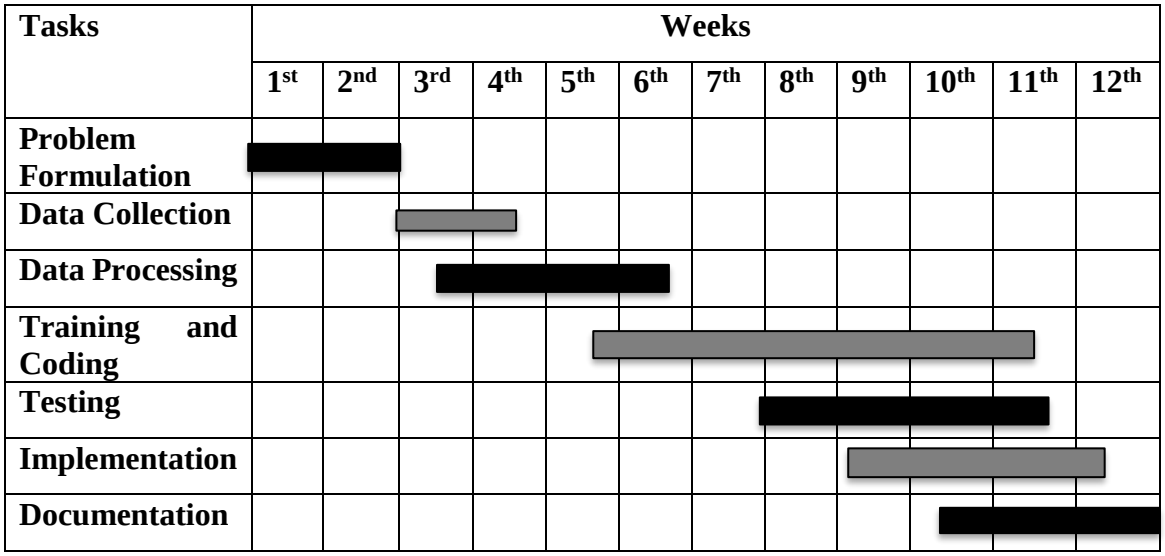

Figure 3.2: Gantt chart for Project Schedule

CHAPTER 4: SYSTEM DESIGN

4.1 Design

The system takes an image as an input from the user, preprocess the image by transforming the image into the format that is suitable for the pre-trained ResNet50 architecture. The model then generates the fixed length feature vectors of the image which is then sent to the model which is trained with Flickr30k dataset and saved in .h5 format. The model uses the softmax function that generates probability distribution across all the words. Greedy search algorithm was the applied to select the words with maximum probability.

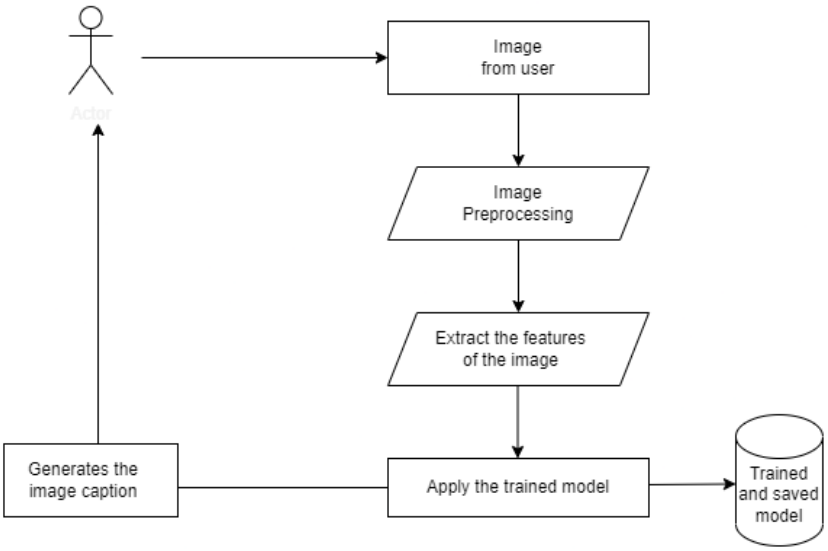

Figure 4.1: Basic System Design

4.2 Algorithm Details

Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) is a Deep Learning algorithm which takes in an input image and assigns importance (learnable weights and biases) to various aspects/objects in the image, which helps it differentiate one image from the other. The neural network consists of several convolutional layers mixed with nonlinear and pooling layers. When the image is passed through one convolution layer, the output of the first layer becomes the input for the second layer. This process continues for all subsequent layers.

After a series of convolutional, nonlinear and pooling layers, it is necessary to attach a fully connected layer. This layer takes the output information from convolutional networks. Attaching a fully connected layer to the end of the network results in an N dimensional vector, where N is the number of classes from which the model selects the desired class.

Convolutional layers: These layers apply a set of learnable filters (also known as kernels or weights) to the input image, producing a set of feature maps. Each filter is designed to detect a specific pattern or feature in the input image. The filters are learned during the training process and are optimized to maximize the accuracy of the network.

Pooling layers: These layers reduce the spatial dimensions of the feature maps by selecting the maximum value (max pooling) or the average value (average pooling) in each non-overlapping region of the feature maps. This helps to reduce the number of parameters in the network and prevent overfitting.

Activation functions: These functions introduce non-linearity into the network, allowing it to learn complex patterns in the input data. The most commonly used activation function is the ReLU function, which sets all negative values to zero.

Fully connected layers: These layers connect every neuron in one layer to every neuron in the next layer. They are typically used in the final layers of the network to produce a vector of class probabilities for the input image.

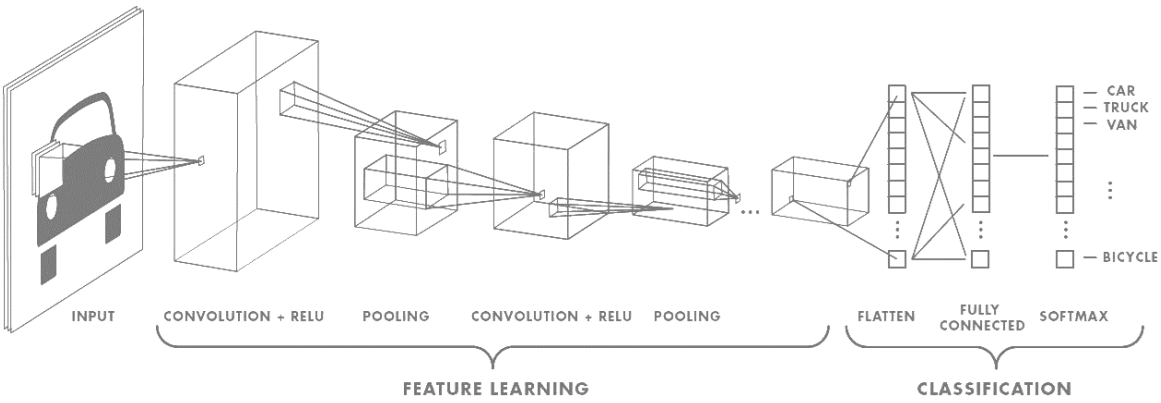

Figure 4.2: Basic CNN Architecture

ResNet50 Architecture

ResNet is a family of deep neural network architectures that use residual learning to enable training of very deep neural networks. ResNet was introduced by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun in 2015.

ResNet50 is a specific implementation of the ResNet architecture and consists of 50 layers, including 49 convolutional layers and one fully connected layer.

Convolutional layer: This layer takes in the input image and applies a set of learnable filters to it, producing a set of feature maps.

Max pooling layer: This layer reduces the spatial dimensions of the feature maps by selecting the maximum value in each non-overlapping region.

Residual blocks: Each residual block contains several convolutional layers with a shortcut connection around them. The shortcut connection simply passes the input directly to the output of the block, allowing gradients to propagate more easily through the network. Each residual block has a set of learnable parameters that allow it to adjust the output of the block.

Fully connected layer: The final layer of the network is a fully connected layer that takes the output of the final residual block and produces a vector of class probabilities.

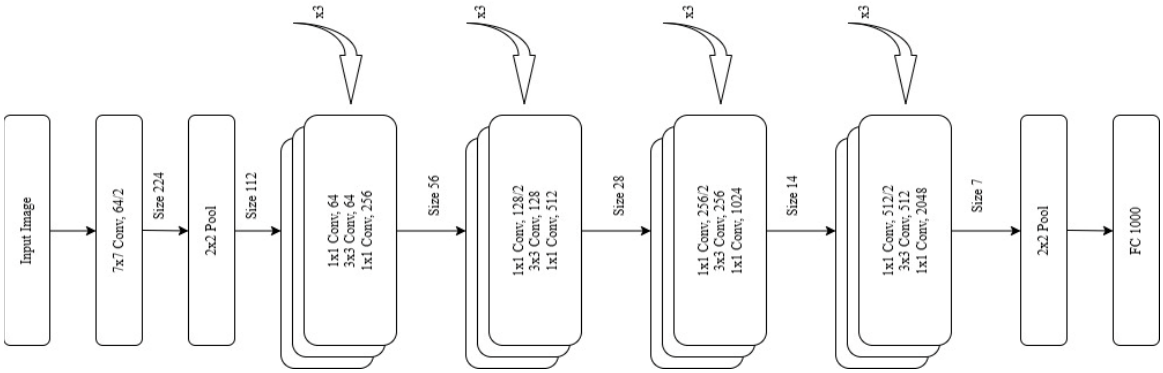

Figure 4.3: Resnet50 basic architecture

The ResNet architecture consists of several residual blocks, each of which contains several convolutional layers followed by a batch normalization layer, a ReLU activation function, and a skip connection.

A residual block is a stack of layers set in such a way that the output of a layer is taken and added to another layer deeper in the block. The non-linearity is then applied after adding it together with the output of the corresponding layer in the main path. This by-pass connection is known as the shortcut or the skip-connection.

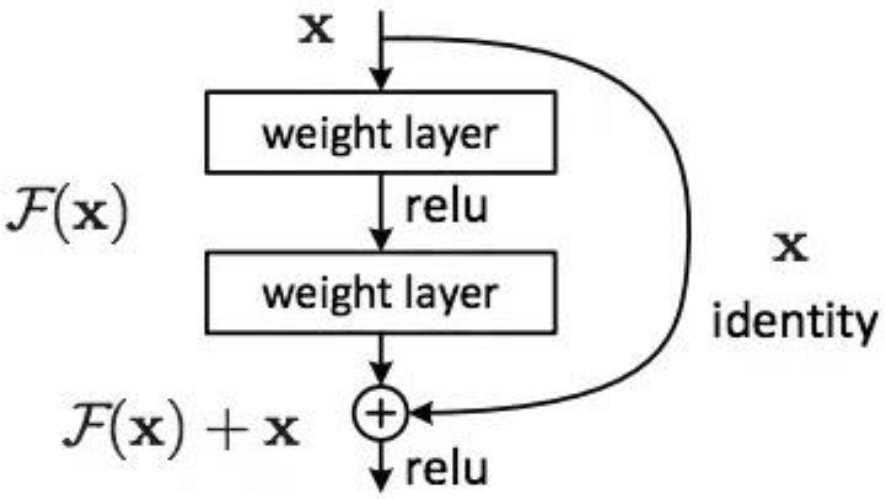

Figure 4.4: Residual learning: a building block

Let x be the input to a layer, and let F(x) be the output of the layer after applying a set of learnable transformations. The output of the layer with a skip connection is given by:

$$H(x) = F(x) + x$$

where, H(x) is the output of the layer with the skip connection, and x is the input to the layer.

The addition operation in this formula is element-wise addition, meaning that each element of F(x) is added to the corresponding element of x.

The skip connection can be visualized as a shortcut path that goes directly from the input to the output of the layer, bypassing the intermediate layers. This allows the input to flow more easily through the network, and helps to mitigate the problem of vanishing gradients that can occur when training very deep networks.

The idea behind residual learning is to add shortcut connections (also known as skip connections or identity mappings) between the layers of the network, allowing the gradient to flow more easily through the network.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 4.5: ResNet Architecture Versions

Recurrent Neural Network

The human brain is evolved in such a way so as to make sense of previous words, and keeping these in mind generate the next words, thus forming a perfect sentence. Basic Neural networks don't have the ability to do this. However, advancements in recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist for a while, by making use of their internal states, thus creating a feedback loop.

RNN works on the principle of saving the output of a particular layer and feeding this back to the input in order to predict the output of the layer.

A recurrent neural network can be thought of as multiple copies of a feedforward network, each passing a message to a successor.

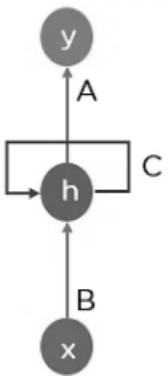

Figure 4.6: Loop in RNN

Here, “x” is the input layer, “h” is the hidden layer, and “y” is the output layer. A, B, and C are the network parameters used to improve the output of the model. At any given time t, the current input is a combination of input at x(t) and x(t-1). The output at any given time is fetched back to the network to improve on the output.

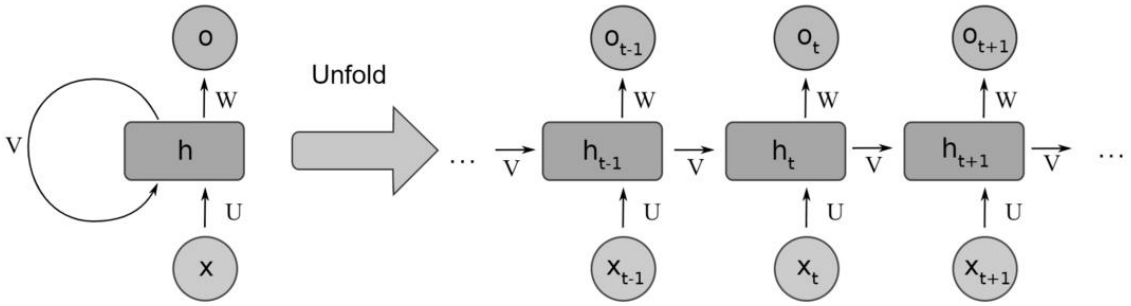

Figure 4.7: Unfolding of RNN

RNNs (Recurrent Neural Networks) work by maintaining a hidden state that captures information from previous inputs in a sequence, and using this hidden state to make predictions for the current input.

Long-Short Term Memory

“LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. Remembering information for long periods is practically their default behavior, and this behavior is controlled with the help of “gates”.

Traditional RNNs suffer from the vanishing gradient problem, which makes it difficult to learn long-term dependencies, but LSTMs are specifically designed to address this problem. While RNNs process single data points, LSTMs can process entire sequences. Not only that, they can learn which point in the data holds importance, and which can be thrown away. Hence, the only relevant information is passed on to the next layer.

The 3 main gates involved are: input gate, output gate and forget gate. These gates decide whether to forget the current cell value, read a value into the cell, or output the cell value. The hidden states play an important role since the previous hidden states are passed to the next step of the sequence. The hidden state acts as the neural network's memory, as it is storing the data that the neural network has seen before. Thus it allows the neural network to function like a human brain trying to form sentences.

Each gate is implemented as a neural network layer that takes as input the current input and the previous hidden state, and produces an output between 0 and 1 that represents the amount of information to be passed through the gate. The output of each gate is then combined with the input and the previous cell state to update the current cell state and hidden state.

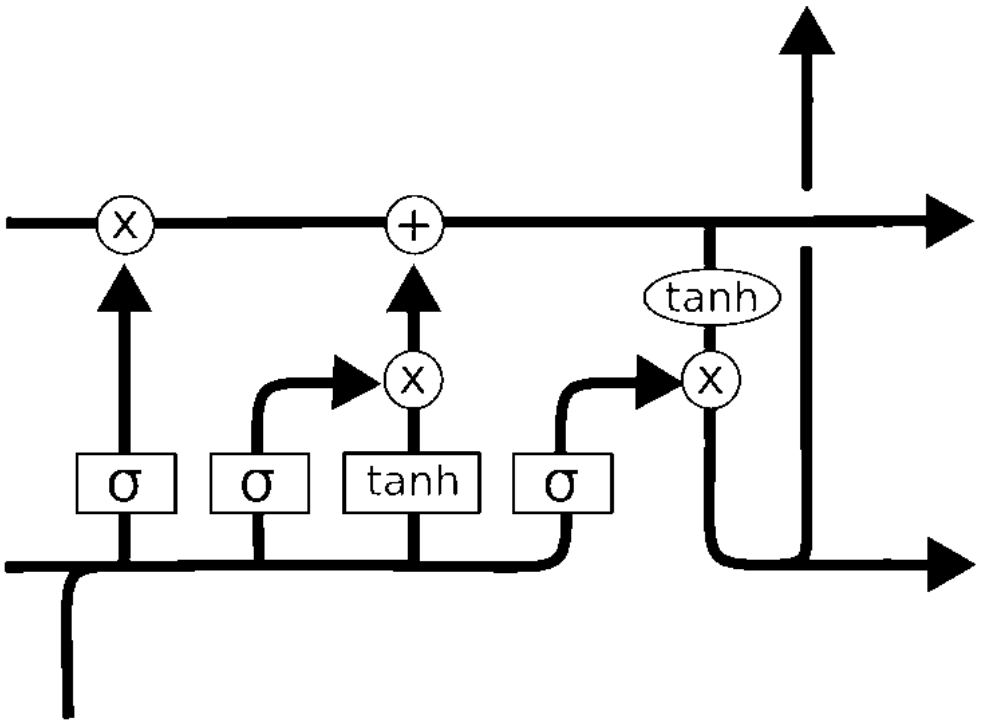

Figure 4.8: LSTM Architecture

The reason behind developing LSTM was, when we go deeper into a neural network if the gradients are very small or zero, then little to no training can take place, leading to poor predictive performance and this problem was encountered when training traditional RNNs. LSTM is way more effective and better compared to the traditional RNN as it overcomes the short term memory limitations of the RNN. LSTM can carry out relevant information throughout the processing of inputs and discards non-relevant information with a forget gate.

The key advantage of LSTMs is that they can selectively update the cell state and hidden state based on the input and the previous state, while also allowing information to be carried forward over long periods of time.

CNN-LSTM Architecture

We utilized CNN and LSTM to take an image as input and output a caption. An “encoder” RNN maps the source sentence (which is of variable length) and transforms it into a fixed-length vector representation, which in turn is used as the initial hidden state of a “decoder” RNN which ultimately generates the final meaningful sentence as a prediction.

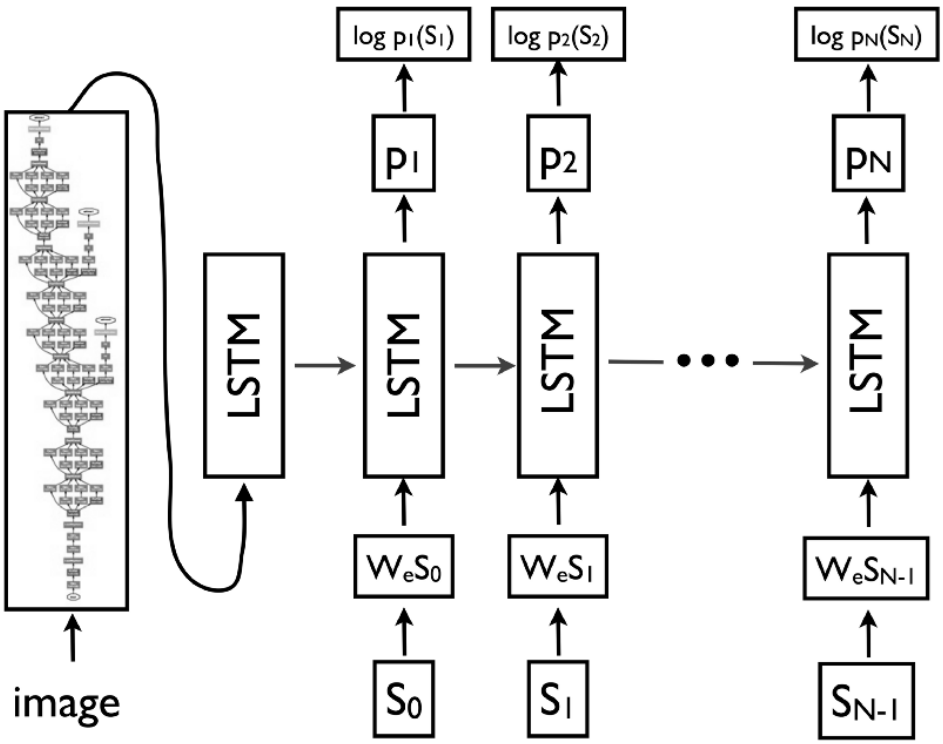

Figure 4.9: CNN-LSTM Architecture

CHAPTER 5: IMPLEMENTATION AND TESTING

5.1 Implementation

We went through data preprocessing steps to process raw input data (both images and captions) into proper format. A pre-trained Convolutional Neural Network architecture as an encoder was used to extract and encode image features into a higher dimensional vector space. An LSTM-based Recurrent Neural Network was used as a decoder to convert encoded features to natural language descriptions.

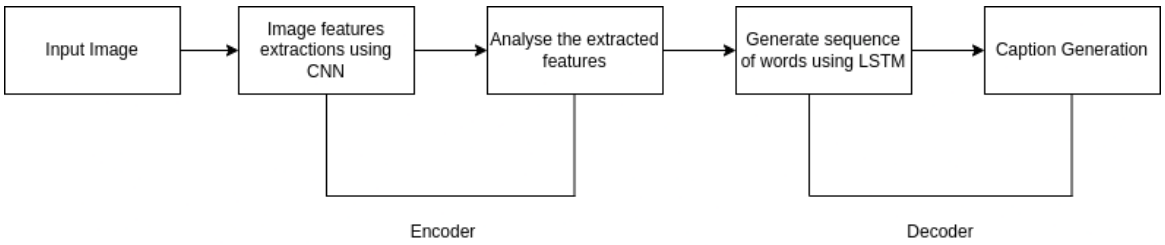

Figure 5.1: Basic Architecture of Image Caption Generator

5.1.1 Tools Used

Development Platform: Google Colaboratory, Kaggle, Visual Studio Code

Programming Language: Python

Libraries: Numpy, OpenCV, Keras, Tensorflow, Streamlit, NLTK

5.1.2 Implementation Details of Module

In the project, we used Flickr30k dataset which contains 31783 images each associated with 5 captions making a total of 158,915 captions as our primary dataset. Caption dataset was read and visualized to have general idea of the dataset such as the number of captions and the format of how the captions are stored in the file.

Data cleaning process include lowercasing all the characters and converting all the non-alphabetic characters (such as '#', '%', '$', '&', '@' etc.), removing them and saving to a file named 'tokens.txt'.

We then visualized the total number of tokens in the caption dataset along with the unique number of tokens.

We have taken two sets of the data one training dataset which we used to train our model and testing dataset which we used for evaluating the performance of the model.

We added two tokens in every caption as: ‘startseq’ which is a start sequence token which will be added at the start of every caption and ‘endseq’ which is an end sequence token which will be added at the end of every caption.

Both training and testing image datasets are then preprocessed and transformed into an array of the suitable format before feeding it into the ResNet50 model.

The features of the images are extracted just before the last layer of classification. The output is an array vector of length 2048 which represents the features extracted from the images. A dictionary with each image tagged to its features is formed and saved. The features thus extracted are dumped into the file and saved in ‘.pkl’ format.

We then encoded each word into a fixed sized vector. We created two python dictionaries namely “word_to_index” and “index_to_word” and dumped the dictionaries in the respective ‘.pkl’ file.

In order to generate a coherent and informative caption, the model needs to understand the semantics of the words in the text. For this we can represent the words as dense vectors, called word embedding, which capture their meaning and context.

The function named data_generator(), train_content, train_encoding, word_to_index, train_encoding, max_len and batch_size transform the data into input-output pairs of data for training the model. There are two input arrays to the model: one for photo features and one for the encoded text. There is one output for the model which is the encoded next word in the text sequence.

We then define the model architecture which contains two input layers one for the vocabulary and the other for the image feature vector. The feature vectors thus extracted from the ResNet50 architecture then go through the layer of dropout and a fully connected layer. Similarly, the caption vector goes through the embedding layer, dropout layer and LSTM layer. The two outputs are then added and goes through a fully connected layer and then to a final output dense layer that makes a softmax prediction over the entire output vocabulary for the next word in the sequence. Greedy search algorithm was the applied to select the words with maximum probability.

The model was trained in batch size of 5 and up to 15 epochs and with every iteration the model was saved. We then created a script that access the saved model and generate the caption of the image uploaded by the user.

For UI part, we made the use of Streamlit library of Python, which contains a simple interface that allows the user to browse the image and then generate and also read the generated caption.

5.2 Testing

5.2.1 Unit Testing

Unit testing refers to the testing of every small modular components of the system, keeping them isolated from other modules. Here we mention testing result of the various part of the system. In unit testing, we design the whole system in modularized pattern and each module was tested. Each of the component of the system that we developed was successfully tested and they were working perfectly.

The other components of the interfaces were also tested and were working properly and as expected.

5.2.2 System Testing

In this testing phase our system as a whole was tested. Every individual component was integrated and tested.

5.3 Result Analysis

For the evaluation of the system, we used a performance metric such as BLEU score and human evaluation. The BLEU (Bilingual Evaluation Understudy) score is a metric used to evaluate the quality of machine-generated translations or text, by comparing them to one or more reference translations.

The sentence-level BLEU score, also known as BLEU-1, computes the precision of the n-grams (sequences of n consecutive words) of the generated sentence with respect to the reference sentence(s), up to a maximum of n-grams of length 4. The BLEU-1 score is then computed as the geometric mean of the precision scores for each of the n-grams, weighted by their lengths.

NLTK provides the sentence_bleu() function for evaluating a generated sequence against one or more reference sentences. The reference sentences must be provided as a list of sentences where each reference is a list of tokens.

To assess the quality of the generated captions, we conducted a human evaluation using a set of pre-defined criteria. We asked our friends to pose as annotators and evaluate the generated captions. Each annotator was asked to rate the quality of the generated captions based on relevance, fluency, grammaticality, and overall coherence.

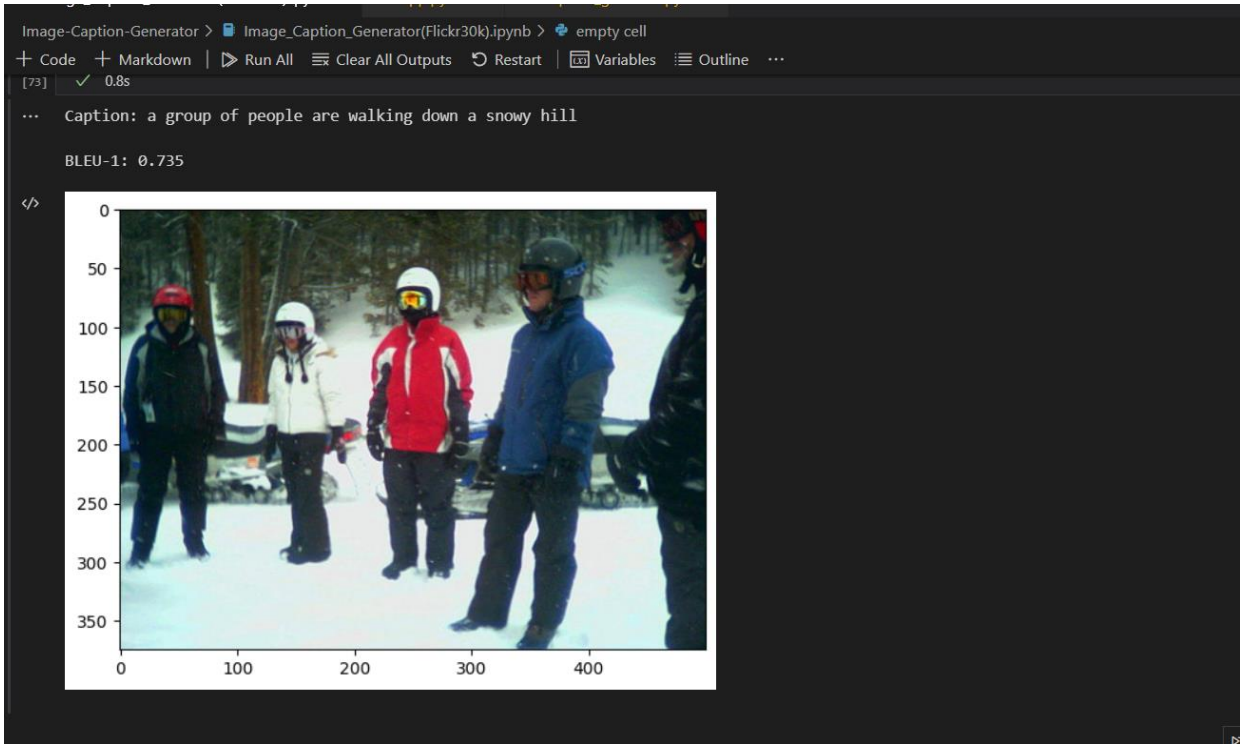

Figure 5.2: Evaluation of the generated caption on the test image dataset

Generated Caption

a group of people are walking down a snowy hill

Original Description

1. Five snowmobile riders all wearing helmets and goggles line up in a snowy clearing in a forest in front of their snowmobiles ; they are all wearing black snow pants and from left to right they are wearing a black coat , white coat , red coat , blue coat , and black coat.
2. Five people wearing winter jackets and helmets stand in the snow , with snowmobiles in the background.
3. Five people wearing winter clothing , helmets , and ski goggles stand outside in the snow.
4. A group of snowmobile riders gather in the snow.
5. Group gathered to go snowmobiling.

Bleu Score

0.735

Figure 5.3: Evaluation of the generated caption on the test image dataset

Generated Description

a group of people are walking down the street

Original Description

1. Several people , including a shirtless man and a woman in purple shorts which say " P.I.N.K. " on the back , are walking through a crowded outdoor area.
2. Shirtless guy staring off in the distance while three woman are walking past a crowd sitting outside a cafe.
3. A shirtless man and three women walk outside near a street cafe.
4. Four people dressed for warm weather out on a street walking.
5. These people are walking in a crowd of people.

Bleu Score

0.799

A higher BLEU score generally indicates that the machine-translated sentence is more similar to the reference sentences, but it does not necessarily guarantee that the translation accurately conveys the intended meaning.

CHAPTER 6: CONCLUSION AND FUTURE RECOMMENDATIONS

6.1 Conclusion

In conclusion, the development of an image caption generator is an exciting and challenging project that combines computer vision and natural language processing techniques. Through our research and implementation, we have successfully created a model that can analyze and understand images, and generate relevant and coherent captions that describe the content of the images.

In this project we have learned and designed a technique of Image Caption Generator which will respond to users with captions or descriptions based on an image. The image based model extracts features of an image and the language based model translates the features and objects extracted by image based model to a natural sentence. Image based model uses CNN whereas language based model uses LSTM.

The workflow is data gathering followed by pre-processing, training model and prediction. The ultimate purpose of an Image Caption Generator is to improve the social media platforms as well as in image indexing and for visually impaired persons with automated generated captions or description.

To conclude, our image caption generator is a promising step towards bridging the gap between images and natural language, and has the potential to greatly benefit various industries and users. With further advancements and refinements, image caption generation can continue to evolve and have a significant impact on how we understand and interact with visual content in the digital era.

6.2 Future Recommendation

We successfully completed what we mentioned in the project proposal, but used a smaller dataset (Flickr30k) due to limited computational power. There can be potential improvements if given more time. First of all, we directly used a pre-trained CNN network, ResNet50 which was trained on ImageNet as part of our pipeline, because of which the network does not adapt to this specific training dataset. Thus, by experimenting with different CNN pre-trained networks and enabling fine-tuning, we expect to achieve a slightly higher BLEU score. Another potential improvement is by training on a combination of Flickr8k, Flickr30k, and MSCOCO.

Also the experiment with other approaches for generating the caption sequence can help improve the performance. In general, the more diverse training dataset the network has seen, the more accurate the output will be.

The caption generator derives literal interpretation of the image while this project can be further explored to expand its application for social media captioning focuses on generating the caption that is more suitable for social media.

Image caption generators have immense potential, and future research can focus on incorporating contextual information, leveraging multimodal techniques, enhancing caption diversity, handling rare or unseen content, addressing bias, integrating user feedback, and exploring real-time captioning to further improve their performance and applicability in various domains.

We all agree that this project ignites our interest in application of Machine Learning knowledge in CV and NLP and expects to explore more in the future.

REFERENCES

[1] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3156–3164.

[2] K. Xu *et al.*, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, PMLR, 2015, pp. 2048–2057.

[3] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[4] R. Kiros, R. Salakhutdinov, and R. Zemel, "Multimodal neural language models," in *International conference on machine learning*, PMLR, 2014, pp. 595–603.

[5] R. Kiros, R. Salakhutdinov, and R. S. Zemel, "Unifying visual-semantic embeddings with multimodal neural language models," *arXiv preprint arXiv:1411.2539*, 2014.

[6] X. Jia, E. Gavves, B. Fernando, and T. Tuytelaars, "Guiding the long-short term memory model for image caption generation," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2407–2415.

[7] M. Soh, "Learning CNN-LSTM architectures for image caption generation," *Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, Tech. Rep*, vol. 1, 2016.

[8] Y. Chu, X. Yue, L. Yu, M. Sergei, and Z. Wang, "Automatic image captioning based on ResNet50 and LSTM with soft attention," *Wirel Commun Mob Comput*, vol. 2020, pp. 1–7, 2020.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[10] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “Bleu: a method for automatic evaluation of machine translation,” in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[12] A. Karpathy and L. Fei-Fei, “Deep visual-semantic alignments for generating image descriptions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3128–3137.

[13] J. Donahue *et al.*, “Long-term recurrent convolutional networks for visual recognition and description,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 2625–2634.

[14] X. Chen and C. L. Zitnick, “Learning a recurrent visual representation for image caption generation,” *arXiv preprint arXiv:1411.5654*, 2014.

[15] L. Yang, H. Hu, S. Xing, and X. Lu, “Constrained lstm and residual attention for image captioning,” *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 16, no. 3, pp. 1–18, 2020.

[16] C. Wang, H. Yang, C. Bartz, and C. Meinel, “Image captioning with deep bidirectional LSTMs,” in *Proceedings of the 24th ACM international conference on Multimedia*, 2016, pp. 988–997.

[17] M. Chen, G. Ding, S. Zhao, H. Chen, Q. Liu, and J. Han, “Reference based LSTM for image captioning,” in *Proceedings of the AAAI conference on artificial intelligence*, 2017.

[18] K. Xu, H. Wang, and P. Tang, “Image captioning with deep LSTM based on sequential residual,” in *2017 IEEE International Conference on Multimedia and Expo (ICME)*, IEEE, 2017, pp. 361–366.

[19] P. Anderson *et al.*, “Bottom-up and top-down attention for image captioning and visual question answering,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6077–6086.

[20] E. K. Wang, X. Zhang, F. Wang, T.-Y. Wu, and C.-M. Chen, “Multilayer dense attention model for image caption,” *IEEE Access*, vol. 7, pp. 66358–66368, 2019.

[21] M. Liu, L. Li, H. Hu, W. Guan, and J. Tian, “Image caption generation with dual attention mechanism,” *Inf Process Manag*, vol. 57, no. 2, p. 102178, 2020.

[22] A. Farhadi *et al.*, “Every picture tells a story: Generating sentences from images,” in *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11*, Springer, 2010, pp. 15–29.

[23] R. Socher, A. Karpathy, Q. V Le, C. D. Manning, and A. Y. Ng, “Grounded compositional semantics for finding and describing images with sentences,” *Trans Assoc Comput Linguist*, vol. 2, pp. 207–218, 2014.

[24] V. Kesavan, V. Muley, and M. Kolhekar, “Deep learning based automatic image caption generation,” in *2019 Global Conference for Advancement in Technology (GCAT)*, IEEE, 2019, pp. 1–6.

[25] D. Matthew Zeiler and F. Rob, “Visualizing and understanding convolutional neural networks,” ECCV, 2014.

APPENDIX

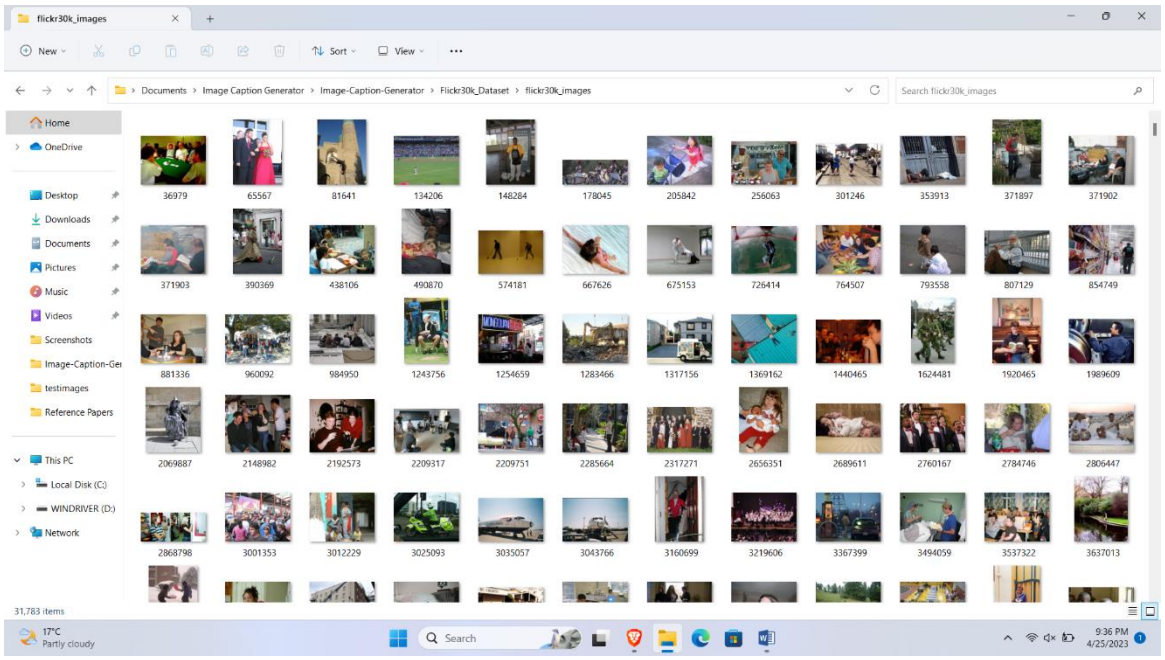

Figure 7.1: A glimpse of Flickr30k image dataset

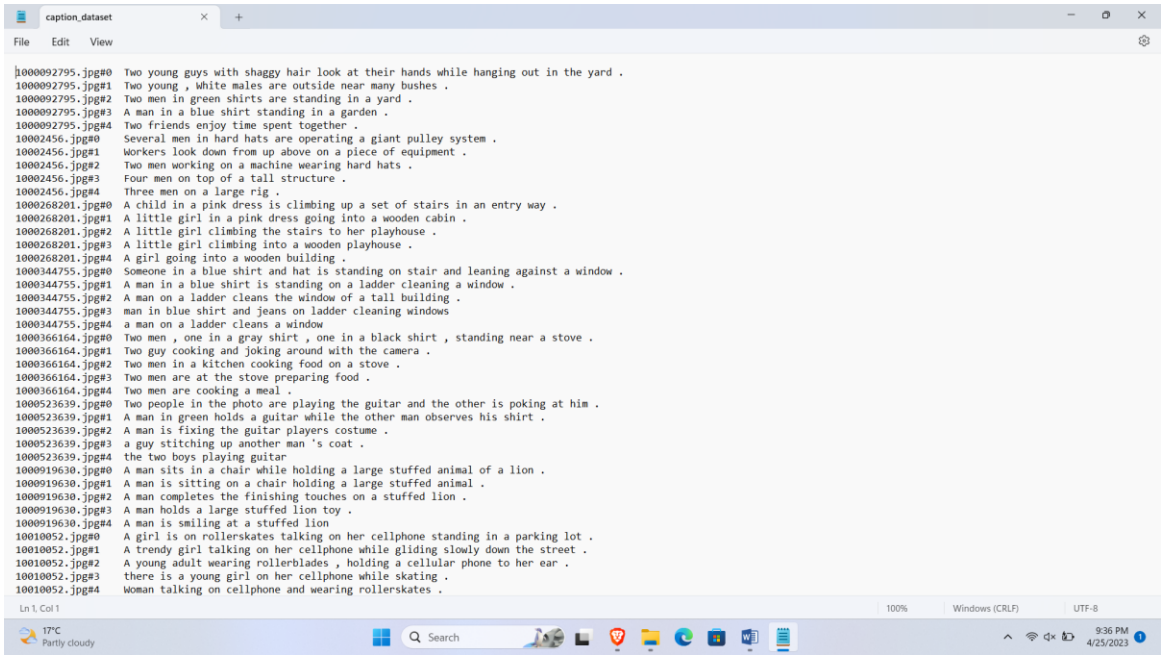

Figure 7.2: A glimpse of Flickr30k caption dataset

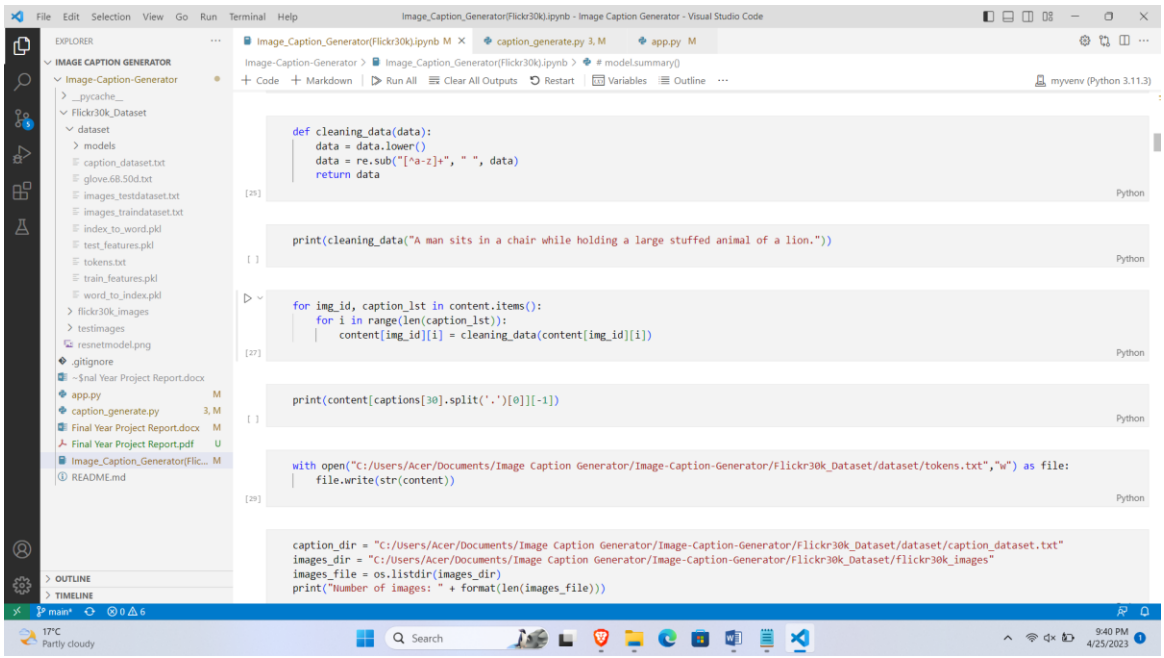

Figure 7.3: Code Snippet of Text Preprocessing Steps

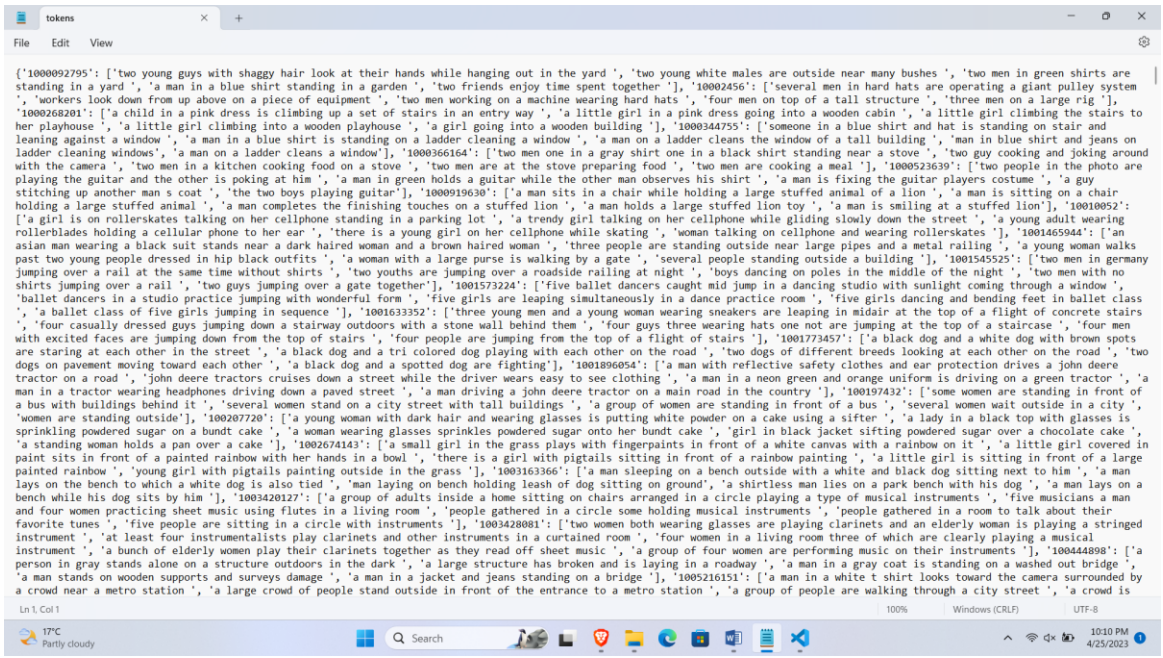

Figure 7.4: A glimpse of preprocessed caption tagged with corresponding image filename

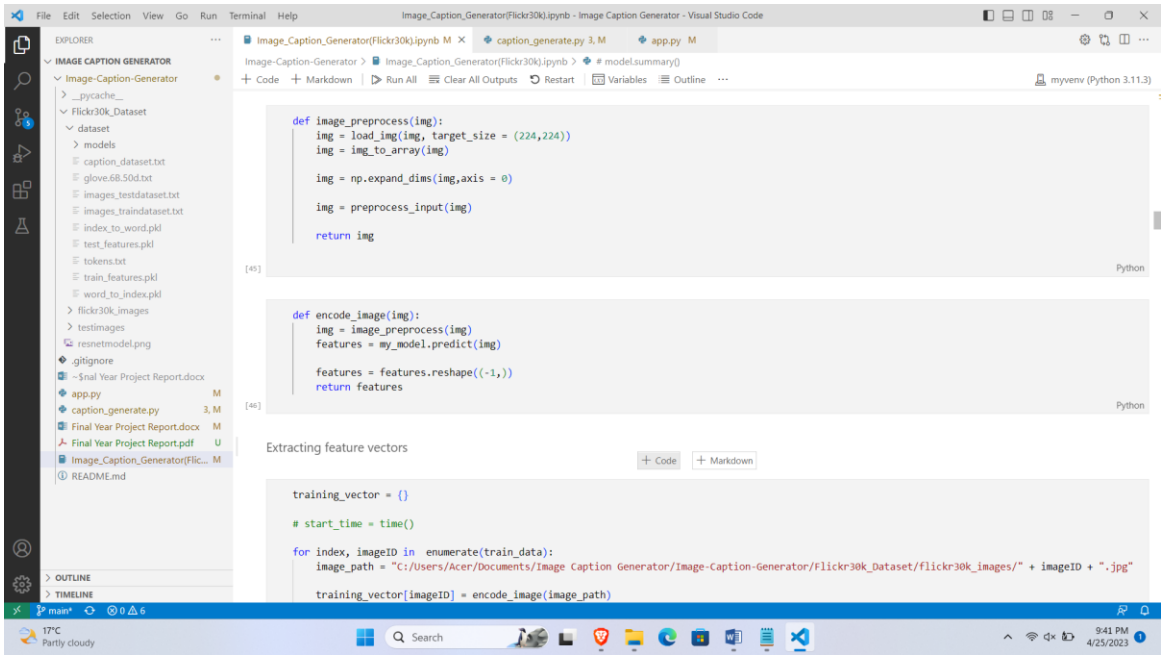

Figure 7.5: Code Snippet of Image Preprocessing Steps

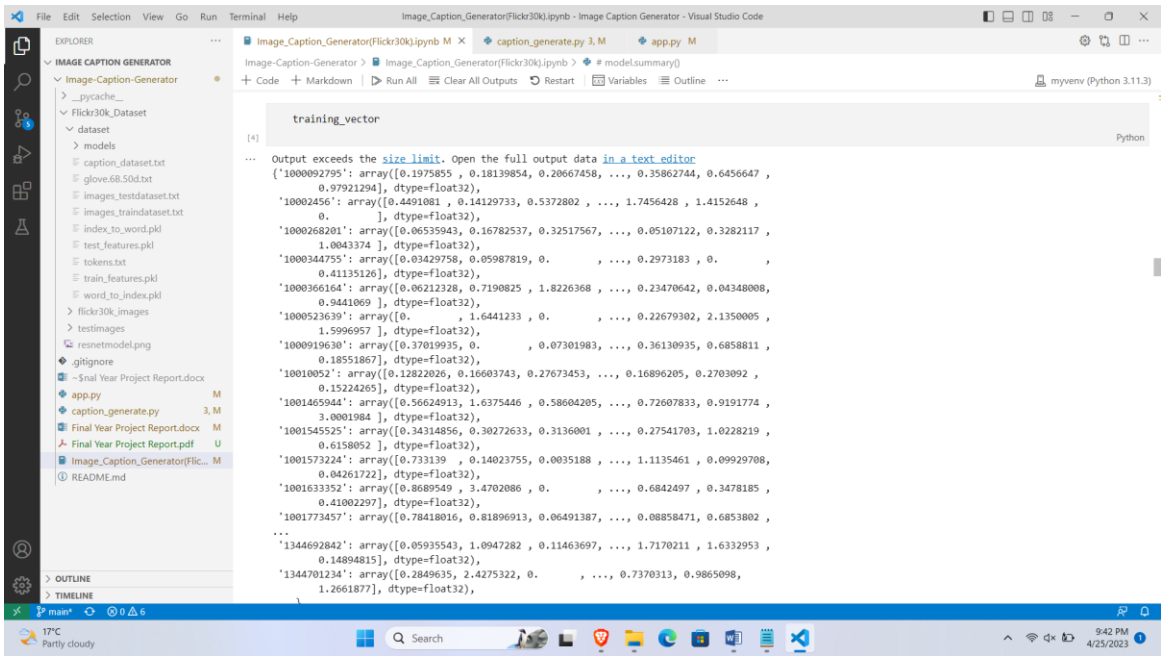

Figure 7.6: A glimpse of image feature vectors

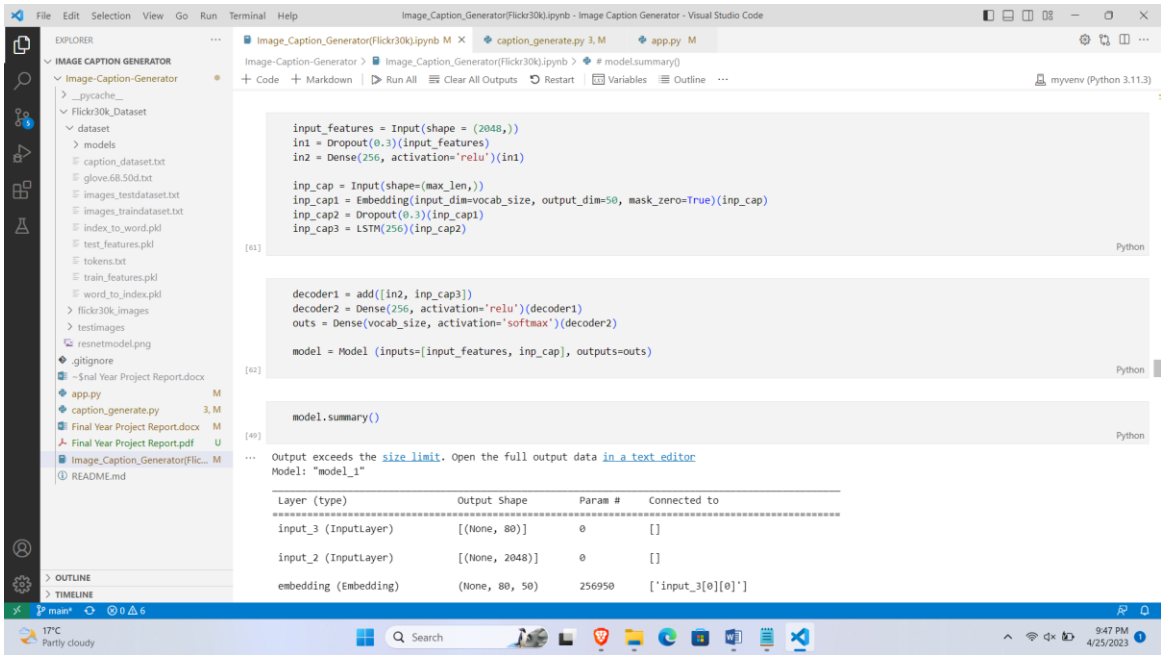

Figure 7.7: Code snippet for defining model

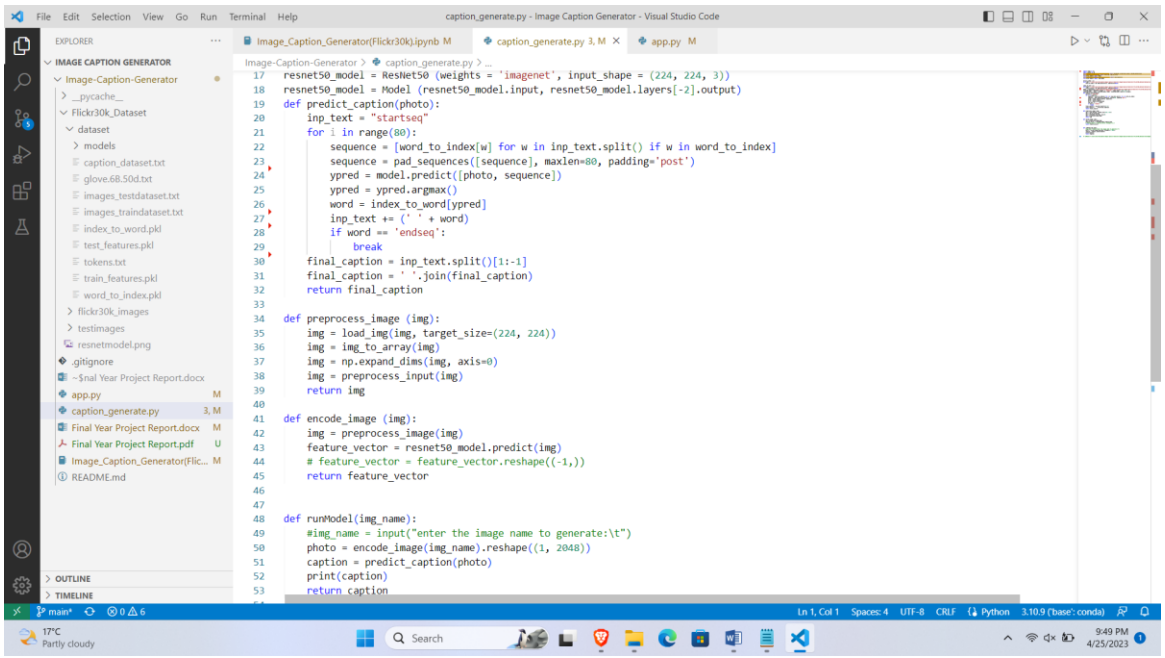

Figure 7.8: Code Snippet of caption generation

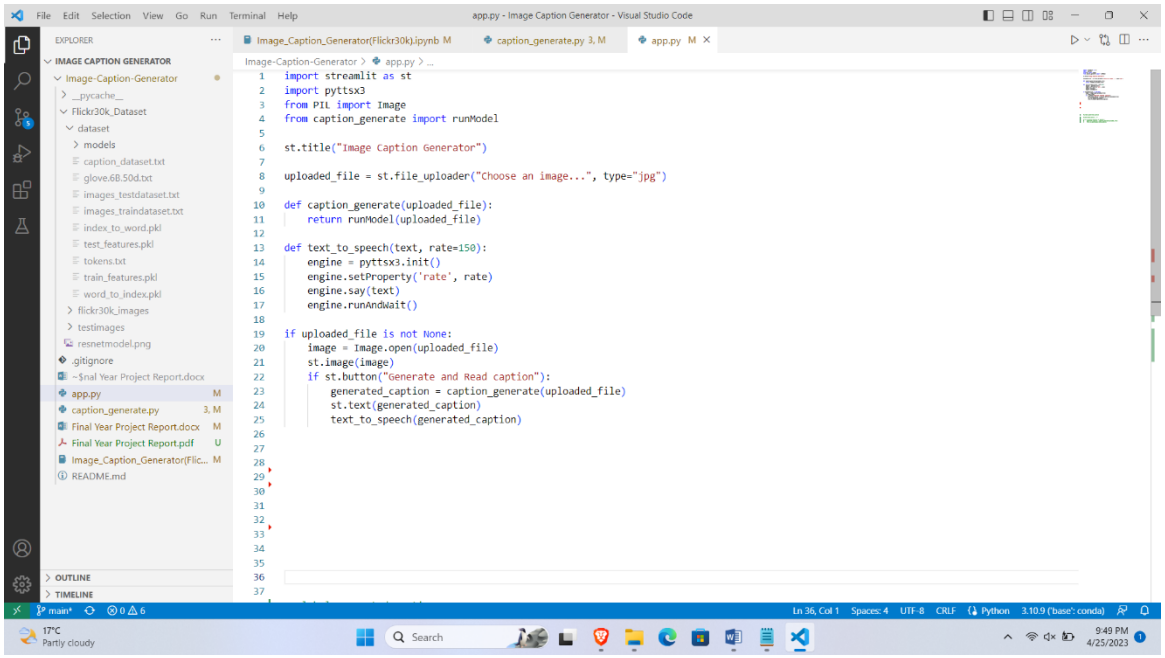

Figure 7.9: Code Snippet of UI using Streamlit

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 80)	0	
input_2 (InputLayer)	(None, 2048)	0	
embedding_1 (Embedding)	(None, 80, 50)	256950	input_3[0][0]
dropout_1 (Dropout)	(None, 2048)	0	input_2[0][0]
dropout_2 (Dropout)	(None, 80, 50)	0	embedding_1[0][0]
dense_1 (Dense)	(None, 256)	524544	dropout_1[0][0]
lstm_1 (LSTM)	(None, 256)	314368	dropout_2[0][0]
add_17 (Add)	(None, 256)	0	dense_1[0][0] lstm_1[0][0]
dense_2 (Dense)	(None, 256)	65792	add_17[0][0]
dense_3 (Dense)	(None, 5139)	1320723	dense_2[0][0]

Figure 7.10: Model Summary

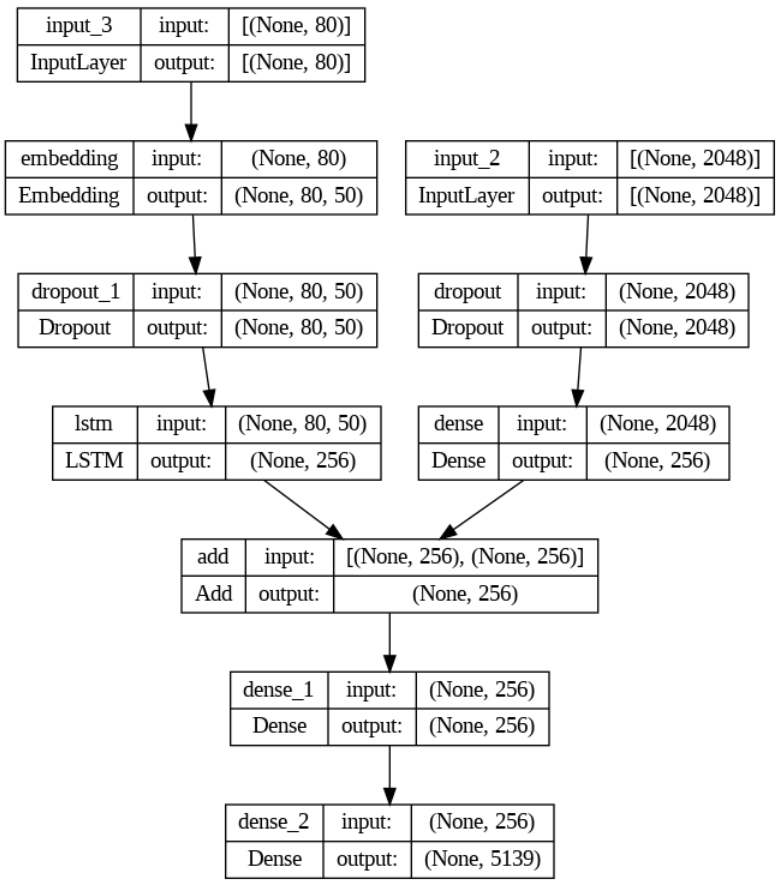

Figure 7.11: Model Visualization

Model: "resnet50"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 224, 224, 3)	0	
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)	0	input_1[0][0]
conv1 (Conv2D)	(None, 112, 112, 64)	9472	conv1_pad[0][0]
bn_conv1 (BatchNormalization)	(None, 112, 112, 64)	256	conv1[0][0]
activation_1 (Activation)	(None, 112, 112, 64)	0	bn_conv1[0][0]
pool1_pad (ZeroPadding2D)	(None, 114, 114, 64)	0	activation_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 64)	0	pool1_pad[0][0]
res2a_branch2a (Conv2D)	(None, 56, 56, 64)	4160	max_pooling2d_1[0][0]
bn2a_branch2a (BatchNormalizati	(None, 56, 56, 64)	256	res2a_branch2a[0][0]
activation_2 (Activation)	(None, 56, 56, 64)	0	bn2a_branch2a[0][0]
res2a_branch2b (Conv2D)	(None, 56, 56, 64)	36928	activation_2[0][0]
bn2a_branch2b (BatchNormalizati	(None, 56, 56, 64)	256	res2a_branch2b[0][0]
activation_3 (Activation)	(None, 56, 56, 64)	0	bn2a_branch2b[0][0]
res2a_branch2c (Conv2D)	(None, 56, 56, 256)	16640	activation_3[0][0]
res2a_branch1 (Conv2D)	(None, 56, 56, 256)	16640	max_pooling2d_1[0][0]
bn2a_branch2c (BatchNormalizati	(None, 56, 56, 256)	1024	res2a_branch2c[0][0]
bn2a_branch1 (BatchNormalizatio	(None, 56, 56, 256)	1024	res2a_branch1[0][0]
add_1 (Add)	(None, 56, 56, 256)	0	bn2a_branch2c[0][0] bn2a_branch1[0][0]
activation_4 (Activation)	(None, 56, 56, 256)	0	add_1[0][0]
res2b_branch2a (Conv2D)	(None, 56, 56, 64)	16448	activation_4[0][0]
bn2b_branch2a (BatchNormalizati	(None, 56, 56, 64)	256	res2b_branch2a[0][0]
activation_5 (Activation)	(None, 56, 56, 64)	0	bn2b_branch2a[0][0]
res2b_branch2b (Conv2D)	(None, 56, 56, 64)	36928	activation_5[0][0]
bn2b_branch2b (BatchNormalizati	(None, 56, 56, 64)	256	res2b_branch2b[0][0]
activation_6 (Activation)	(None, 56, 56, 64)	0	bn2b_branch2b[0][0]
res2b_branch2c (Conv2D)	(None, 56, 56, 256)	16640	activation_6[0][0]
bn2b_branch2c (BatchNormalizati	(None, 56, 56, 256)	1024	res2b_branch2c[0][0]
add_2 (Add)	(None, 56, 56, 256)	0	bn2b_branch2c[0][0] activation_4[0][0]
activation_7 (Activation)	(None, 56, 56, 256)	0	add_2[0][0]
res2c_branch2a (Conv2D)	(None, 56, 56, 64)	16448	activation_7[0][0]
bn2c_branch2a (BatchNormalizati	(None, 56, 56, 64)	256	res2c_branch2a[0][0]

activation_8 (Activation)	(None, 56, 56, 64)	0	bn2c_branch2a[0][0]
res2c_branch2b (Conv2D)	(None, 56, 56, 64)	36928	activation_8[0][0]
bn2c_branch2b (BatchNormalizati	(None, 56, 56, 64)	256	res2c_branch2b[0][0]
activation_9 (Activation)	(None, 56, 56, 64)	0	bn2c_branch2b[0][0]
res2c_branch2c (Conv2D)	(None, 56, 56, 256)	16640	activation_9[0][0]
bn2c_branch2c (BatchNormalizati	(None, 56, 56, 256)	1024	res2c_branch2c[0][0]
add_3 (Add)	(None, 56, 56, 256)	0	bn2c_branch2c[0][0] activation_7[0][0]
activation_10 (Activation)	(None, 56, 56, 256)	0	add_3[0][0]
res3a_branch2a (Conv2D)	(None, 28, 28, 128)	32896	activation_10[0][0]
bn3a_branch2a (BatchNormalizati	(None, 28, 28, 128)	512	res3a_branch2a[0][0]
activation_11 (Activation)	(None, 28, 28, 128)	0	bn3a_branch2a[0][0]
res3a_branch2b (Conv2D)	(None, 28, 28, 128)	147584	activation_11[0][0]
bn3a_branch2b (BatchNormalizati	(None, 28, 28, 128)	512	res3a_branch2b[0][0]
activation_12 (Activation)	(None, 28, 28, 128)	0	bn3a_branch2b[0][0]
res3a_branch2c (Conv2D)	(None, 28, 28, 512)	66048	activation_12[0][0]
res3a_branch1 (Conv2D)	(None, 28, 28, 512)	131584	activation_10[0][0]
bn3a_branch2c (BatchNormalizati	(None, 28, 28, 512)	2048	res3a_branch2c[0][0]
bn3a_branch1 (BatchNormalizatio	(None, 28, 28, 512)	2048	res3a_branch1[0][0]
add_4 (Add)	(None, 28, 28, 512)	0	bn3a_branch2c[0][0] bn3a_branch1[0][0]
activation_13 (Activation)	(None, 28, 28, 512)	0	add_4[0][0]
res3b_branch2a (Conv2D)	(None, 28, 28, 128)	65664	activation_13[0][0]
bn3b_branch2a (BatchNormalizati	(None, 28, 28, 128)	512	res3b_branch2a[0][0]
activation_14 (Activation)	(None, 28, 28, 128)	0	bn3b_branch2a[0][0]

res3b_branch2b (Conv2D)	(None, 28, 28, 128)	147584	activation_14[0][0]
bn3b_branch2b (BatchNormalizati	(None, 28, 28, 128)	512	res3b_branch2b[0][0]
activation_15 (Activation)	(None, 28, 28, 128)	0	bn3b_branch2b[0][0]
res3b_branch2c (Conv2D)	(None, 28, 28, 512)	66048	activation_15[0][0]
bn3b_branch2c (BatchNormalizati	(None, 28, 28, 512)	2048	res3b_branch2c[0][0]
add_5 (Add)	(None, 28, 28, 512)	0	bn3b_branch2c[0][0] activation_13[0][0]
activation_16 (Activation)	(None, 28, 28, 512)	0	add_5[0][0]
res3c_branch2a (Conv2D)	(None, 28, 28, 128)	65664	activation_16[0][0]
bn3c_branch2a (BatchNormalizati	(None, 28, 28, 128)	512	res3c_branch2a[0][0]
activation_17 (Activation)	(None, 28, 28, 128)	0	bn3c_branch2a[0][0]
res3c_branch2b (Conv2D)	(None, 28, 28, 128)	147584	activation_17[0][0]
bn3c_branch2b (BatchNormalizati	(None, 28, 28, 128)	512	res3c_branch2b[0][0]
activation_18 (Activation)	(None, 28, 28, 128)	0	bn3c_branch2b[0][0]
res3c_branch2c (Conv2D)	(None, 28, 28, 512)	66048	activation_18[0][0]
bn3c_branch2c (BatchNormalizati	(None, 28, 28, 512)	2048	res3c_branch2c[0][0]
add_6 (Add)	(None, 28, 28, 512)	0	bn3c_branch2c[0][0] activation_16[0][0]
activation_19 (Activation)	(None, 28, 28, 512)	0	add_6[0][0]
res3d_branch2a (Conv2D)	(None, 28, 28, 128)	65664	activation_19[0][0]
bn3d_branch2a (BatchNormalizati	(None, 28, 28, 128)	512	res3d_branch2a[0][0]
activation_20 (Activation)	(None, 28, 28, 128)	0	bn3d_branch2a[0][0]
res3d_branch2b (Conv2D)	(None, 28, 28, 128)	147584	activation_20[0][0]
bn3d_branch2b (BatchNormalizati	(None, 28, 28, 128)	512	res3d_branch2b[0][0]
activation_21 (Activation)	(None, 28, 28, 128)	0	bn3d_branch2b[0][0]
res3d_branch2c (Conv2D)	(None, 28, 28, 512)	66048	activation_21[0][0]
bn3d_branch2c (BatchNormalizati	(None, 28, 28, 512)	2048	res3d_branch2c[0][0]
add_7 (Add)	(None, 28, 28, 512)	0	bn3d_branch2c[0][0] activation_19[0][0]
activation_22 (Activation)	(None, 28, 28, 512)	0	add_7[0][0]
res4a_branch2a (Conv2D)	(None, 14, 14, 256)	131328	activation_22[0][0]
bn4a_branch2a (BatchNormalizati	(None, 14, 14, 256)	1024	res4a_branch2a[0][0]
activation_23 (Activation)	(None, 14, 14, 256)	0	bn4a_branch2a[0][0]
res4a_branch2b (Conv2D)	(None, 14, 14, 256)	590080	activation_23[0][0]

activation_24 (Activation)	(None, 14, 14, 256)	0	bn4a_branch2b[0][0]
res4a_branch2c (Conv2D)	(None, 14, 14, 1024)	263168	activation_24[0][0]
res4a_branch1 (Conv2D)	(None, 14, 14, 1024)	525312	activation_22[0][0]
bn4a_branch2c (BatchNormalizati	(None, 14, 14, 1024)	4096	res4a_branch2c[0][0]
bn4a_branch1 (BatchNormalizatio	(None, 14, 14, 1024)	4096	res4a_branch1[0][0]
add_8 (Add)	(None, 14, 14, 1024)	0	bn4a_branch2c[0][0] bn4a_branch1[0][0]
activation_25 (Activation)	(None, 14, 14, 1024)	0	add_8[0][0]
res4b_branch2a (Conv2D)	(None, 14, 14, 256)	262400	activation_25[0][0]
bn4b_branch2a (BatchNormalizati	(None, 14, 14, 256)	1024	res4b_branch2a[0][0]
activation_26 (Activation)	(None, 14, 14, 256)	0	bn4b_branch2a[0][0]
res4b_branch2b (Conv2D)	(None, 14, 14, 256)	590080	activation_26[0][0]
bn4b_branch2b (BatchNormalizati	(None, 14, 14, 256)	1024	res4b_branch2b[0][0]
activation_27 (Activation)	(None, 14, 14, 256)	0	bn4b_branch2b[0][0]
res4b_branch2c (Conv2D)	(None, 14, 14, 1024)	263168	activation_27[0][0]
bn4b_branch2c (BatchNormalizati	(None, 14, 14, 1024)	4096	res4b_branch2c[0][0]
add_9 (Add)	(None, 14, 14, 1024)	0	bn4b_branch2c[0][0] activation_25[0][0]
activation_28 (Activation)	(None, 14, 14, 1024)	0	add_9[0][0]
res4c_branch2a (Conv2D)	(None, 14, 14, 256)	262400	activation_28[0][0]
bn4c_branch2a (BatchNormalizati	(None, 14, 14, 256)	1024	res4c_branch2a[0][0]
activation_29 (Activation)	(None, 14, 14, 256)	0	bn4c_branch2a[0][0]
res4c_branch2b (Conv2D)	(None, 14, 14, 256)	590080	activation_29[0][0]
bn4c_branch2b (BatchNormalizati	(None, 14, 14, 256)	1024	res4c_branch2b[0][0]
activation_30 (Activation)	(None, 14, 14, 256)	0	bn4c_branch2b[0][0]
res4c_branch2c (Conv2D)	(None, 14, 14, 1024)	263168	activation_30[0][0]
bn4c_branch2c (BatchNormalizati	(None, 14, 14, 1024)	4096	res4c_branch2c[0][0]
add_10 (Add)	(None, 14, 14, 1024)	0	bn4c_branch2c[0][0] activation_28[0][0]
activation_31 (Activation)	(None, 14, 14, 1024)	0	add_10[0][0]
res4d_branch2a (Conv2D)	(None, 14, 14, 256)	262400	activation_31[0][0]
bn4d_branch2a (BatchNormalizati	(None, 14, 14, 256)	1024	res4d_branch2a[0][0]
activation_32 (Activation)	(None, 14, 14, 256)	0	bn4d_branch2a[0][0]
res4d_branch2b (Conv2D)	(None, 14, 14, 256)	590080	activation_32[0][0]
bn4d_branch2b (BatchNormalizati	(None, 14, 14, 256)	1024	res4d_branch2b[0][0]
activation_33 (Activation)	(None, 14, 14, 256)	0	bn4d_branch2b[0][0]

res4d_branch2c (Conv2D)	(None, 14, 14, 1024)	263168	activation_33[0][0]
bn4d_branch2c (BatchNormalizati	(None, 14, 14, 1024)	4096	res4d_branch2c[0][0]
add_11 (Add)	(None, 14, 14, 1024)	0	bn4d_branch2c[0][0] activation_31[0][0]
activation_34 (Activation)	(None, 14, 14, 1024)	0	add_11[0][0]
res4e_branch2a (Conv2D)	(None, 14, 14, 256)	262400	activation_34[0][0]
bn4e_branch2a (BatchNormalizati	(None, 14, 14, 256)	1024	res4e_branch2a[0][0]
activation_35 (Activation)	(None, 14, 14, 256)	0	bn4e_branch2a[0][0]
res4e_branch2b (Conv2D)	(None, 14, 14, 256)	590080	activation_35[0][0]
bn4e_branch2b (BatchNormalizati	(None, 14, 14, 256)	1024	res4e_branch2b[0][0]
activation_36 (Activation)	(None, 14, 14, 256)	0	bn4e_branch2b[0][0]
res4e_branch2c (Conv2D)	(None, 14, 14, 1024)	263168	activation_36[0][0]
bn4e_branch2c (BatchNormalizati	(None, 14, 14, 1024)	4096	res4e_branch2c[0][0]
add_12 (Add)	(None, 14, 14, 1024)	0	bn4e_branch2c[0][0] activation_34[0][0]
activation_37 (Activation)	(None, 14, 14, 1024)	0	add_12[0][0]
res4f_branch2a (Conv2D)	(None, 14, 14, 256)	262400	activation_37[0][0]
bn4f_branch2a (BatchNormalizati	(None, 14, 14, 256)	1024	res4f_branch2a[0][0]
activation_38 (Activation)	(None, 14, 14, 256)	0	bn4f_branch2a[0][0]
res4f_branch2b (Conv2D)	(None, 14, 14, 256)	590080	activation_38[0][0]
bn4f_branch2b (BatchNormalizati	(None, 14, 14, 256)	1024	res4f_branch2b[0][0]
activation_39 (Activation)	(None, 14, 14, 256)	0	bn4f_branch2b[0][0]
res4f_branch2c (Conv2D)	(None, 14, 14, 1024)	263168	activation_39[0][0]
bn4f_branch2c (BatchNormalizati	(None, 14, 14, 1024)	4096	res4f_branch2c[0][0]
add_13 (Add)	(None, 14, 14, 1024)	0	bn4f_branch2c[0][0] activation_37[0][0]
activation_40 (Activation)	(None, 14, 14, 1024)	0	add_13[0][0]
res5a_branch2a (Conv2D)	(None, 7, 7, 512)	524800	activation_40[0][0]
bn5a_branch2a (BatchNormalizati	(None, 7, 7, 512)	2048	res5a_branch2a[0][0]
activation_41 (Activation)	(None, 7, 7, 512)	0	bn5a_branch2a[0][0]
res5a_branch2b (Conv2D)	(None, 7, 7, 512)	2359808	activation_41[0][0]
bn5a_branch2b (BatchNormalizati	(None, 7, 7, 512)	2048	res5a_branch2b[0][0]
activation_42 (Activation)	(None, 7, 7, 512)	0	bn5a_branch2b[0][0]
res5a_branch2c (Conv2D)	(None, 7, 7, 2048)	1050624	activation_42[0][0]
res5a_branch1 (Conv2D)	(None, 7, 7, 2048)	2099200	activation_40[0][0]
bn5a_branch2c (BatchNormalizati	(None, 7, 7, 2048)	8192	res5a_branch2c[0][0]

bn5a_branch1 (BatchNormalizatio	(None, 7, 7, 2048)	8192	res5a_branch1[0][0]
add_14 (Add)	(None, 7, 7, 2048)	0	bn5a_branch2c[0][0] bn5a_branch1[0][0]
activation_43 (Activation)	(None, 7, 7, 2048)	0	add_14[0][0]
res5b_branch2a (Conv2D)	(None, 7, 7, 512)	1049088	activation_43[0][0]
bn5b_branch2a (BatchNormalizati	(None, 7, 7, 512)	2048	res5b_branch2a[0][0]
activation_44 (Activation)	(None, 7, 7, 512)	0	bn5b_branch2a[0][0]
res5b_branch2b (Conv2D)	(None, 7, 7, 512)	2359808	activation_44[0][0]
bn5b_branch2b (BatchNormalizati	(None, 7, 7, 512)	2048	res5b_branch2b[0][0]
activation_45 (Activation)	(None, 7, 7, 512)	0	bn5b_branch2b[0][0]
res5b_branch2c (Conv2D)	(None, 7, 7, 2048)	1050624	activation_45[0][0]
bn5b_branch2c (BatchNormalizati	(None, 7, 7, 2048)	8192	res5b_branch2c[0][0]
add_15 (Add)	(None, 7, 7, 2048)	0	bn5b_branch2c[0][0] activation_43[0][0]
activation_46 (Activation)	(None, 7, 7, 2048)	0	add_15[0][0]
res5c_branch2a (Conv2D)	(None, 7, 7, 512)	1049088	activation_46[0][0]
bn5c_branch2a (BatchNormalizati	(None, 7, 7, 512)	2048	res5c_branch2a[0][0]
activation_47 (Activation)	(None, 7, 7, 512)	0	bn5c_branch2a[0][0]
res5c_branch2b (Conv2D)	(None, 7, 7, 512)	2359808	activation_47[0][0]
bn5c_branch2b (BatchNormalizati	(None, 7, 7, 512)	2048	res5c_branch2b[0][0]
activation_48 (Activation)	(None, 7, 7, 512)	0	bn5c_branch2b[0][0]
res5c_branch2c (Conv2D)	(None, 7, 7, 2048)	1050624	activation_48[0][0]
bn5c_branch2c (BatchNormalizati	(None, 7, 7, 2048)	8192	res5c_branch2c[0][0]
add_16 (Add)	(None, 7, 7, 2048)	0	bn5c_branch2c[0][0] activation_46[0][0]
activation_49 (Activation)	(None, 7, 7, 2048)	0	add_16[0][0]
avg_pool (GlobalAveragePooling2	(None, 2048)	0	activation_49[0][0]
fc1000 (Dense)	(None, 1000)	2049000	avg_pool[0][0]

Figure 7.12: ResNet50 model summary

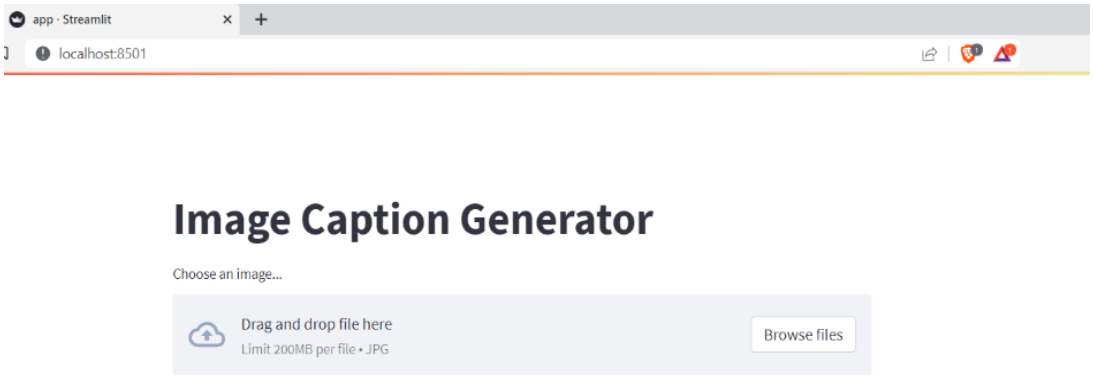

Figure 7.13: User Interface

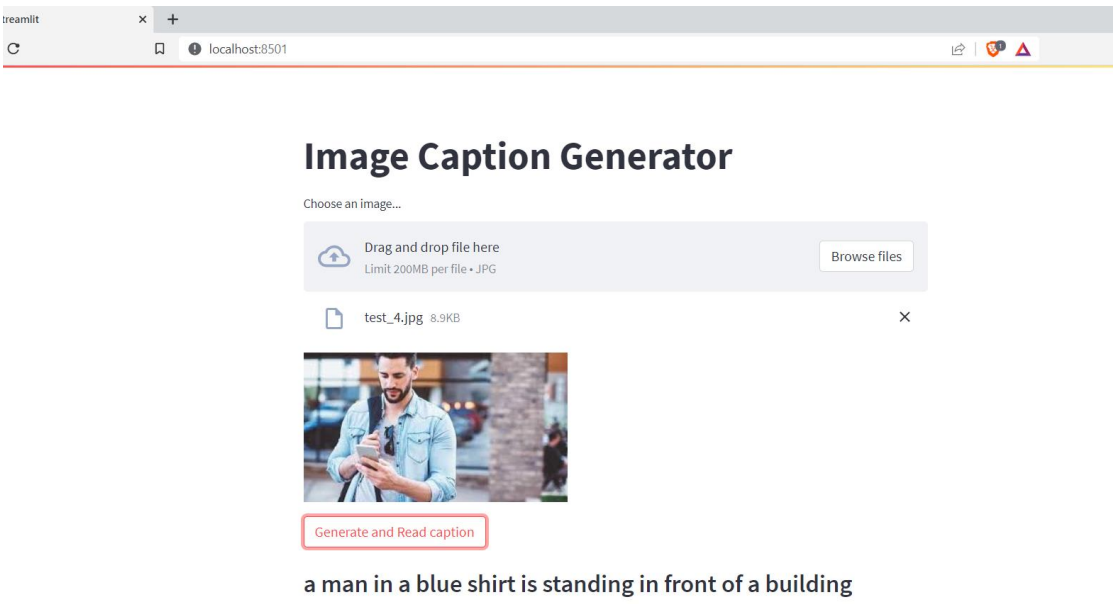

Figure 7.14: Caption Generation of the Uploaded Image 1

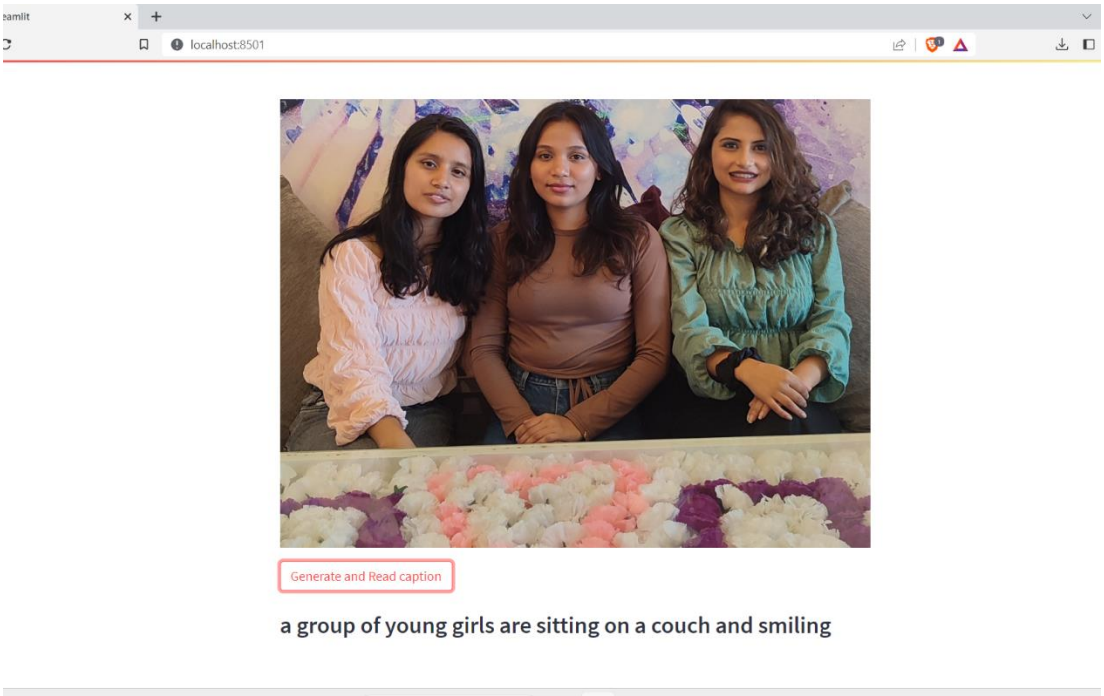

Figure 7.15: Caption Generation of the Uploaded Image 2

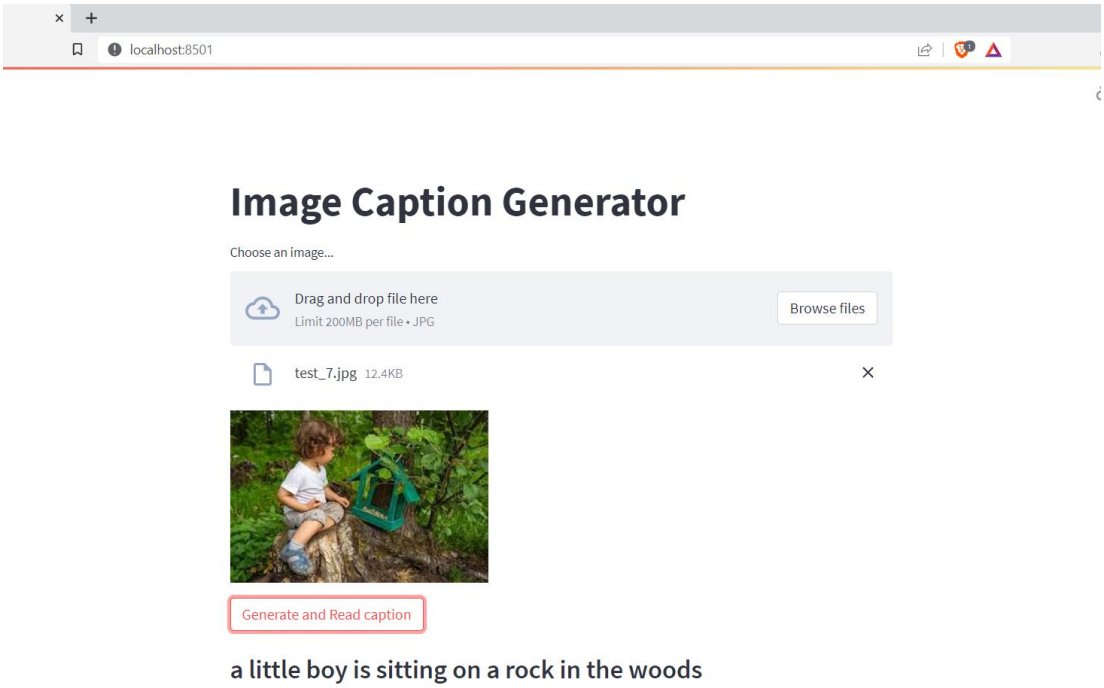

Figure 7.16: Caption Generation of the Uploaded Image 3