A project report on

IMAGE CAPTION GENERATOR

Submitted in partial fulfillment for the award of the degree of

B.Tech

by

CHOWKACHERLA SABAREESH REDDY (19BCE7210)

VIT-AP UNIVERSITY

SCOPE

June, 2023

IMAGE CAPTION GENERATOR

Submitted in partial fulfillment for the award of the degree of

B.Tech

by

CHOWKACHERLA SABAREESH REDDY (19BCE7210)

SCOPE

June, 2023

DECLARATION

I here by declare that the thesis entitled “IMAGE CAPTION GENERATOR” submitted by me, for the award of the degree of B.Tech. It is a record of bonafide work carried out by me under the supervision of Chandru

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Amaravati

Date:22-05-2023

Signature of the Candidate

Ch. Sabareesh

CERTIFICATE

This is to certify that the Internship titled "**IMAGE CAPTION GENERATOR**" that is being submitted by **CHOWKACHERLA SABAREESH REDDY (19BCE7210)** is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma, and the same is certified.

ORIANA INFORMATION TECHNOLOGIES LLP
CHENNAI
600 119

Chandru

External Guide

The Thesis is Satisfactory

Internal Examiner **Internal Examiner**

Approved by

PROGRAM CHAIR **DEAN**

B. Tech. CSE School Of Computer Science Engineering

ORIANA
Enlighten your Destiny

20th May 2023

To Whom It May Concern

This is to certify that **Mr. CHOWKACHERLA SABAREESH REDDY (Reg.No. 19BCE7210)** B.Tech.,(Computer Science and Engineering), student of **Vellore Institute of Technology –Amravati,** has successfully completed his Project titled on **"Image Caption Generator"** in the platform of **DATA SCIENCE** in our company from **January 2023 to June 2023.**

All necessary details were provided from our side for the establishment of this project. We have noticed that, during the period, he has shown keen interest in his assignments and was also regular in attendance.

Oriana Information Technologies LLP.,

[illegible]

ORIANA INFORMATION TECHNOLOGIES LLP ★ CHENNAI 600 119

Authorised Signature

+91 9176333393
hr@orianainfotech.com
www.orianainfotech.com

ABSTRACT

This final report describes the development and evaluation of an image caption generator, an innovative system that combines convolutional neural networks (CNN) and long short-term memory (LSTM) networks to generate descriptive text captions for images. This project aims to bridge the gap between computer vision and natural language processing and provide a comprehensive solution for automatic image understanding.

Image caption generators use pre-trained CNNs to extract high-level image features and capture the spatial information present in images. These features feed into LSTM networks that model sequential dependencies and leverage their ability to generate consistent labels. The LSTM is trained using a dataset of images combined with corresponding captions using both supervised and reinforcement learning techniques to optimize the language model goal. Standard metrics such as BLEU score are used to evaluate the performance of image caption generators. The results demonstrate the effectiveness of the proposed model in generating accurate and meaningful captions for various image types. In addition, attentional mechanisms are investigated to improve the model's ability to focus on relevant image regions when generating captions, thereby improving the overall quality and contextual understanding of the generated captions.

The developed image caption generator has great potential for practical applications such as automatic image captioning, accessibility support for the visually impaired, and content search of image databases. This project contributes to advances in computer vision and natural language processing, providing a robust and versatile solution for image understanding and captioning tasks.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Chandru, Oriana Information Technology LLP, for his constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavor. My association with him / her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Python.

Our sincere thanks to Our Honorable Founder and Chancellor, DR.G. VISWANATHAN, for educating us in his premier institution. We would like to express special gratitude and thanks to Our Beloved Vice Presidents SANKAR VISWANATHAN, Dr. SEKAR VISWANATHAN, G.V. SELVAM for providing us such a platform to complete our project and SUDHA SV, Scope for providing with an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to Saroj Panigrahy, Ph.D., all teaching staff and members working as limbs of our university for their not-self-centered enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Amaravati

Date:22-05-2023 **Chowkacherla Sabareesh Reddy**

TABLE OF CONTENTS

S.No.	Chapter	Title	Page Number
1.		Abstract	
2.		Acknowledgement	
3.		List of Figures and Table	
4.	1	Introduction	10
	1.1	Objectives	11
	1.2	Literature Survey	12
5.	2	Image Caption Generator	13
	2.1	Existing System	13
	2.2	Proposed System	14
	2.3	Requirement Analysis	15
	2.4	Requirement Specification	16
	2.4.1	Functional Requirements	16
	2.4.2	Software Requirements	16
	2.4.3	Supported Operating Systems	17
	2.4.4	Hardware Specifications	17
	2.5	Working Methodology	18
	2.6	Frameworks, Libraries &Modules	20
6.	3	Implications and Limitations	23
	3.1	Recommendations for Future Work	25
7.	4	Results and Discussions	26
8.	5	Conclusion & Future Work	34
9.	6	Appendix	35
10.	7	References	61

LIST OF FIGURES

Figure Number	Name	Page Number
1	Image Caption Generator	15
2	Restructuring the VGG Model for Extraction	26
3	Extracting the Features from the Image Dataset	27
4	Before Preprocessing the Text data	28
5	After Preprocessing the Text data	28
6	Representation of the Model in the Graph	29
7	Training the Model with Approx 20 Epochs	30
8	Calculating the BLEU-1 and BLUE-2 Scores	31
9	Predicted Image Caption for the Image	32
10	Image Caption Generation Actual vs Predicted	32
11	Another Example	33

CHAPTER-1

INTRODUCTION

This project focuses on constructing an Image Caption Generator using deep learning techniques, specifically Convolutional Neural Networks (CNN) for visual feature extraction and Long Short-Term Memory (LSTM) networks for sequential modeling and language generation. The primary objective is to generate accurate and meaningful captions for images, bridging the gap between computer vision and natural language processing.

Automated image captioning has practical applications in various domains, including accessibility support, automated image description, and content retrieval in image databases. To achieve an effective and precise Image Caption Generator, the model is trained using a dataset comprising images and their corresponding captions. Evaluation metrics such as BLEU are employed to assess the quality and similarity of the generated captions compared to human-generated captions.

By combining CNNs for visual feature extraction and LSTM networks for sequential modeling, this project aims to develop a robust and efficient system capable of comprehending and interpreting visual content. The resulting Image Caption Generator will contribute to advancing the fields of computer vision and natural language processing, facilitating improved human-machine interaction and the development of intelligent systems with a comprehensive understanding of visual information.

1.1 OBJECTIVES

The Image Caption Generator project has two main objectives. Firstly, it aims to create a reliable and efficient system capable of generating descriptive image captions. This involves utilizing Convolutional Neural Networks (CNN) for extracting visual features and Long Short-Term Memory (LSTM) networks for sequential modeling and language generation. The goal is to develop a robust model that can accurately interpret visual content and generate meaningful captions.

Secondly, the project aims to improve the quality and relevance of the generated captions. This will be achieved by exploring advanced techniques like attention mechanisms and beam search decoding. Attention mechanisms allow the model to focus on specific regions of the image, enhancing coherence and contextual understanding. Beam search decoding enables the generation of diverse and contextually rich captions by considering multiple caption hypotheses.

The project also emphasizes evaluating and optimizing the performance of the Image Caption Generator. This includes assessing caption quality using evaluation metrics such as BLEU. The objective is to continuously refine and enhance the model through iterative training and fine-tuning, incorporating techniques from supervised learning and reinforcement learning.

Overall, the project aims to develop a practical and adaptable Image Caption Generator applicable in various domains, such as automated image description and accessibility support. By achieving these objectives, the project contributes to advancing computer vision and natural language processing, enabling improved human-machine interaction and the development of intelligent systems.

1.2 LITERATURE REVIEW

Image caption generation has witnessed notable progress with the adoption of Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. CNNs excel at extracting relevant visual features from images, while LSTM networks effectively model the sequential nature of language.

The integration of CNNs and LSTM networks has resulted in improved accuracy and coherency in generated captions. CNNs analyze image content to extract meaningful features, which are then used by LSTM networks to generate contextually relevant captions.

To evaluate caption quality, metrics such as BLEU are commonly used. These metrics compare generated captions with human-authored references, providing a quantitative measure of the performance of CNN and LSTM-based image caption generators.

Large-scale datasets like MSCOCO, Flickr30k and Flickr8k have played a crucial role in training and evaluating CNN and LSTM-based models. These datasets contain diverse image-caption pairs, enabling researchers to develop and benchmark their models effectively.

However, challenges persist in image caption generation, including generating diverse and accurate captions, handling rare objects or unseen scenarios, and addressing caption length variations. Ongoing research focuses on refining the integration of CNNs and LSTMs to enhance caption generation capabilities.

In summary, the integration of CNNs and LSTMs has significantly advanced image caption generation. The combination of visual feature extraction and sequential modeling has improved caption accuracy and coherence. Ongoing research aims to overcome challenges and further enhance CNN and LSTM-based image caption generators.

CHAPTER-2

IMAGE CAPTION GENERATOR

2.1 EXISTING SYSTEM:

The development of various existing systems has significantly advanced the field of image caption generation. These systems have played a crucial role in implementing deep learning techniques to generate descriptive captions for images. For example, "Show and Tell" introduced an end-to-end deep learning model that combined Convolutional Neural Networks (CNN) for visual feature extraction and Long Short-Term Memory (LSTM) networks for language modeling, paving the way for further advancements.

Another notable system, the "Neural Image Caption" approach, incorporated attention mechanisms to improve the alignment between visual content and textual descriptions. By allowing the model to focus on different image regions during caption generation, it achieved more accurate and contextually relevant results.

Transformer-based models, inspired by their success in natural language processing, have also been applied to image captioning. These models replaced LSTM networks with Transformer layers, enabling parallel processing and capturing long-range dependencies in captions. Transformer-based models have shown promising results in generating coherent and contextually rich captions.

Reinforcement learning approaches have been explored to optimize caption generation by fine-tuning generated captions using reward-based training. These approaches directly optimize evaluation metrics or employ reinforcement learning techniques like Policy Gradient to enhance caption quality and diversity.

In summary, these existing systems demonstrate the progress achieved in image caption generation by incorporating deep learning, attention mechanisms, multimodal approaches, and reinforcement learning. They provide a solid foundation for the proposed Image Caption Generator project, which aims to further enhance caption accuracy, relevance, and diversity.

2.2 PROPOSED SYSTEM:

The proposed system for the Image Caption Generator utilizes Convolutional Neural Networks (CNN) for extracting image features and Long Short-Term Memory (LSTM) networks for generating captions. CNNs are widely used in computer vision tasks and are effective in capturing visual patterns and structures from images. By extracting image features using CNNs, the proposed system can obtain a rich representation of the visual content, which serves as the foundation for generating descriptive captions.

In addition to CNNs, LSTM networks are employed for language modeling and caption generation. LSTM networks are a type of recurrent neural network that can capture the sequential nature of language and model the dependencies between words. By incorporating LSTM networks into the system, it can generate coherent and contextually relevant captions based on the extracted image features.

To train and evaluate the proposed system, the Flickr8k dataset is utilized. This dataset consists of a large collection of images paired with human-generated captions, providing a diverse and annotated set of data for training the model. The availability of such a dataset ensures that the system can learn from a wide range of images and generate captions that are representative and accurate.

The primary evaluation metric used in the proposed system is the BLEU score. BLEU (Bilingual Evaluation Understudy) is a widely used metric for evaluating the quality of machine-generated text by comparing it to human-generated reference text. The BLEU score measures the similarity between the generated captions and the reference captions, providing a quantitative measure of the quality and accuracy of the generated captions.

Overall, the proposed system aims to leverage the power of deep learning, specifically CNNs and LSTM networks, along with the utilization of the Flickr8k dataset, to develop an accurate and contextually relevant Image Caption Generator. The evaluation of the system's performance using the BLEU score ensures that the generated captions are of high quality and comparable to human-generated captions.

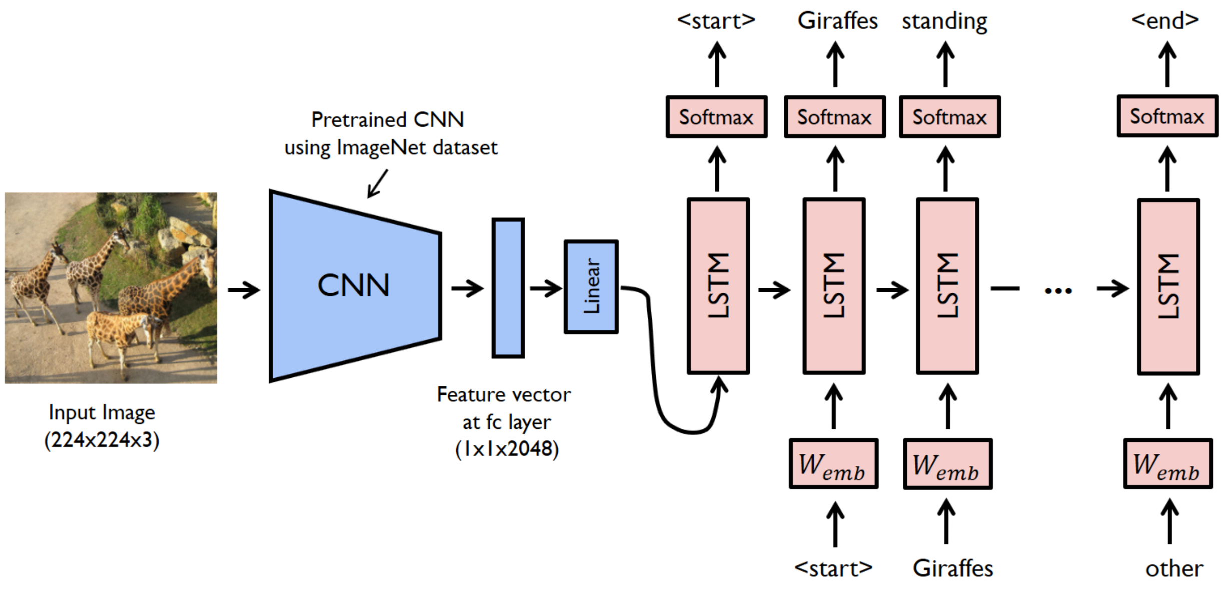


Fig 1: Image Caption Generator

2.3 REQUIREMENT ANALYSIS:

The proposed Image Caption Generator system has specific requirements to fulfill for effective functioning. Firstly, it should be capable of handling the Flickr8k dataset, which consists of diverse images paired with human-generated captions. This dataset serves as a valuable resource for training and evaluating the system's performance.

To extract relevant image features, the system will utilize the VGG model, a well-known Convolutional Neural Network (CNN) architecture widely used for image processing tasks. Leveraging the capabilities of the VGG model, the system can extract high-level visual features from input images, enabling a comprehensive understanding of the visual content.

For generating captions, the system will employ an LSTM network or a similar architecture. LSTM networks excel in capturing the sequential nature of language and generating coherent and contextually relevant captions. Training the LSTM network on the extracted image features enables the system to produce descriptive captions that accurately represent the image content.

Efficiency is a key consideration, and the system will implement techniques such as batch processing and runtime optimizations to enhance speed and efficiency in image processing and caption generation.

The system will include a user interface that allows users to input images and view the generated captions. The interface should be intuitive, user-friendly, and provide a seamless experience, enabling easy interaction and caption generation for user-supplied images.

By addressing these requirements, the Image Caption Generator system aims to leverage the Flickr8k dataset and the VGG model for accurate and meaningful caption generation. With a focus on efficiency, usability, scalability, and ethical considerations, the system aims to provide a reliable and valuable tool for automated image captioning.

2.4 REQUIREMENT SPECIFICATION:

2.4.1 Functional Requirements

Graphical User Interface: The Image Caption Generator should have a user-friendly graphical interface that allows easy interaction with the system. Users should be able to input images and view the generated captions in a visually appealing manner.

Caption Generation: The primary functionality of the system is to generate descriptive captions for images. It should be capable of processing images and generating accurate and contextually relevant captions using deep learning techniques.

2.4.2 Software Requirements

Python: The Image Caption Generator relies on the Python programming language as its development environment. Python offers a wide range of libraries and tools for machine learning, deep learning, and natural language processing tasks.

TensorFlow: TensorFlow, an open-source deep learning framework, is essential for implementing and training neural network models, including those used for image captioning. Its high-level APIs and computation capabilities support efficient deep learning processing.

Flickr8k Dataset: The Image Caption Generator utilizes the Flickr8k dataset, which provides a diverse collection of images paired with human-generated captions. The dataset should be integrated effectively to provide a representative set of images for caption generation.

VGG Model: The VGG model is employed for extracting visual features from images. It is an integral part of the Image Caption Generator, responsible for capturing meaningful and representative visual information used in generating accurate and descriptive captions.

2.4.3 Supported Operating Systems

The Image Caption Generator is compatible with the following operating systems:

Windows 7 and above: The system can be installed and run on Windows 7 and newer versions of the Windows operating system.

Linux: The system can be installed and operated on various Linux distributions, such as Ubuntu, Fedora, and CentOS.

2.4.4 Hardware Specifications

The Image Caption Generator requires the following hardware specifications:

Processor: A multi-core processor with a minimum clock speed of 2.0 GHz or higher is recommended for efficient image processing and deep learning computations.

Memory (RAM): At least 8 GB of RAM is recommended to handle the computational demands of the deep learning models and datasets.

Storage: Sufficient storage space is needed to store the dataset, pre-trained models, and generated captions. A minimum of 100 GB of free disk space is recommended.

Graphics Processing Unit (GPU): Although not mandatory, having a compatible GPU with CUDA support can significantly accelerate the training and inference processes in deep learning.

By meeting these software and hardware requirements, the Image Caption Generator can effectively utilize Python, TensorFlow, the Flickr8k dataset, and the VGG model to generate accurate and contextually relevant captions for images. The compatibility with multiple operating systems ensures flexibility and accessibility, while the recommended hardware specifications ensure optimal performance and resource utilization.

2.5 WORKING METHODOLOGY

Step-1 Dataset Preparation:
Download the Flickr8k dataset, which consists of a large collection of images paired with human-generated captions.
Preprocess the dataset by resizing the images to a specific resolution and extracting the captions associated with each image.

Step-2 Image Feature Extraction:
Utilize the VGG (Visual Geometry Group) model, pre-trained on a large image classification dataset, to extract visual features from the input images.
Preprocess the images by converting them into a suitable format and feed them into the VGG model.
Extract the features from the last fully connected layer of the VGG model and store them using a suitable data structure, such as NumPy arrays or pickle files.

Step-3 Text Preprocessing:
Preprocess the captions by removing unnecessary punctuation, converting words to lowercase, and splitting them into individual tokens.
Create a vocabulary from the captions by tokenizing the words and assigning unique numerical indices to each token.

Step-4 Model Training:
Build a deep learning model using TensorFlow, comprising an LSTM (Long Short-Term Memory) network.
Initialize the model with the pre-trained VGG image features and the preprocessed captions.
Split the dataset into training and validation sets.
Train the model using the training data, adjusting the model parameters to minimize the loss function.

Monitor the model's performance on the validation set by calculating evaluation metrics such as loss, perplexity, and accuracy.

Step-5 Caption Generation:
Load the trained model and input an image for which a caption needs to be generated.
Preprocess the image by resizing it to the appropriate resolution and extracting its visual features using the VGG model.
Initialize the LSTM network with the pre-trained model weights.
Generate captions by iteratively feeding the image features and previous word embeddings into the LSTM network, predicting the next word at each step.
Apply decoding techniques such as beam search to improve the diversity and coherence of the generated captions.

Step-6 Evaluation and Metrics:
Evaluate the quality of the generated captions by comparing them with the reference captions from the dataset.
Calculate metrics such as BLEU (Bilingual Evaluation Understudy to assess the performance of the caption generator. Monitor and analyze the metrics to understand the strengths and limitations of the model.

Step-7 Fine-tuning and Iterative Training:
Perform fine-tuning on the model by adjusting hyperparameters, exploring different architectures, or incorporating regularization techniques.
Iterate the training process by repeating steps 4 to 7, incorporating any necessary modifications based on the evaluation results.

By following this detailed working procedure, the project can effectively utilize the Flickr8k dataset, VGG for image feature extraction, and TensorFlow for model training to generate accurate and contextually relevant captions for images.

2.6 FRAMEWORKS, LIBRARIES & MODULES:

TensorFlow:

- TensorFlow is an open-source machine learning framework widely recognized and utilized by developers worldwide. Developed by Google, it offers a comprehensive range of features and tools to facilitate the development, training, and deployment of machine learning models.

- TensorFlow simplifies the process of constructing and training neural networks and other machine learning models. Its high-level APIs, such as Keras, provide an intuitive interface that abstracts away the intricacies of low-level implementation, making it accessible even to users with limited machine learning expertise.

- An important advantage of TensorFlow is its ability to leverage both CPU and GPU resources efficiently. This allows for accelerated computation during model training and inference, enabling faster and more efficient processing. Additionally, TensorFlow provides visualizations and monitoring capabilities that aid in the analysis and evaluation of model perform.

OS:

- The OS module is a built-in Python library that provides a range of functions for interacting with the operating system. It allows you to perform tasks such as reading and writing files, creating and deleting directories, and interacting with the environment variables and command-line arguments.

NumPy:

- NumPy is a fundamental library in Python for numerical computing. It provides high-performance multidimensional array objects and a collection of mathematical functions to efficiently manipulate large datasets and perform complex mathematical operations.

- Multidimensional Arrays: NumPy's main feature is its ND array (n-dimensional array) object, which allows you to store and manipulate large arrays of homogeneous data efficiently. These arrays can have any number of dimensions and provide a flexible data structure for representing and working with numerical data.

- Mathematical Functions: NumPy provides a vast collection of mathematical functions that operate element-wise on arrays. These functions cover a wide range of mathematical operations, including basic arithmetic, trigonometry, logarithmic functions, statistical computations, and linear algebra operations.

Tqdm:

- tqdm.notebook is a Python library that provides a progress bar for iterative tasks, allowing you to monitor the progress of a loop or operation in real-time. It offers a simple and intuitive way to visualize the progress of your code, making it easier to track the completion status and estimated time remaining.

- Notebook Integration: tqdm.notebook is specifically designed for integration with Jupiter Notebook, providing an interactive progress bar that can be displayed directly in the notebook environment. This allows you to monitor the progress of your code execution while working in a notebook, providing a convenient way to track the advancement of lengthy computations or data processing tasks.

VGG:

- VGG16 is composed of 16 layers, including 13 convolutional layers and 3 fully connected layers. The network architecture is known for its simplicity and uniformity, with small 3x3 filters and max pooling layers used throughout the network. The convolutional layers are followed by three fully connected layers, which are responsible for classification.

- Image Classification: VGG16 is primarily used for image classification tasks. It has been trained on large-scale image classification datasets such as ImageNet, where it achieved state-of-the-art performance at the time of its introduction. The network is capable of recognizing a wide range of object categories in images with high accuracy.

PYTHON IMAGE LIBRARY(PIL):

- The Python Imaging Library (PIL) is a library for working with images in Python. It provides support for reading and writing a wide range of image file formats, as well as basic image manipulation functions such as cropping, resizing, and rotating image. Once you have installed the library
- You can use it in your Python code by importing the Image module.

NLTK:

- NLTK provides a wide range of tools and resources for various NLP tasks, including tokenization, stemming, lemmatization, part-of-speech tagging, parsing, and named entity recognition. These functionalities enable developers to perform advanced text processing and analysis on textual data.

- Corpus and Language Resources: NLTK offers a vast collection of corpora and language resources that facilitate language-specific analysis and modeling. These resources include pre-processed text collections, lexicons, wordlists, and grammars for different languages. NLTK allows users to access and utilize these resources for building NLP models and conducting linguistic research.

MATPLOTLIB:

- Data Visualization: Matplotlib provides a wide range of functions and tools for creating high-quality visualizations of data. It offers support for various types of plots, including line plots, scatter plots, bar plots, histograms, pie charts, and more. With Matplotlib, users can customize and enhance their plots with different colors, markers, line styles, and annotations.

- Publication-Quality Plots: Matplotlib allows users to create publication-ready plots with fine-grained control over plot elements. It provides options to adjust the figure size, aspect ratio, axis limits, and tick labels. Users can add titles, axis labels, legends, and annotations to make their plots informative and visually appealing.

CHAPTER – 3

3 IMPLICATIONS AND LIMITATIONS

Image caption generators are powerful tools that combine computer vision and natural language processing techniques to automatically generate textual descriptions for images. there are several implications and limitations associated with image caption generators:

Implications

Accessibility: Image caption generators can greatly improve accessibility for visually impaired individuals by providing them with textual descriptions of images. This enables them to perceive and understand visual content that would otherwise be inaccessible.

Content indexing and retrieval: Generating captions for images allows for better indexing and retrieval of visual content. By associating descriptive text with images, search engines can effectively index and categorize the content, making it easier for users to find relevant images based on their descriptions.

Content summarization: Image caption generators can be used to summarize the content of an image, providing a concise textual representation that captures the main elements and context. This can be particularly useful in applications where quick and efficient understanding of visual content is required, such as social media platforms.

Multimodal understanding: Image caption generators bridge the gap between visual and textual modalities, promoting research and development in multimodal understanding. This has implications for various applications, including human-robot interaction, content generation, and more.

Contextual understanding: Image caption generators may struggle with capturing the contextual nuances of images. Understanding the relationship between objects, their spatial arrangement, and the overall scene context can be challenging, leading to less accurate or contextually inappropriate captions.

Lack of real-time adaptability: Image caption generators typically require processing time to analyze an image and generate captions. This lack of real-time adaptability can be a limitation in applications that require immediate and dynamic captioning, such as live video captioning or real-time object recognition.

Limitations

Subjectivity and ambiguity: Generating accurate and contextually appropriate captions for images is a challenging task. The interpretation of images and the generation of captions can be subjective and influenced by biases present in the training data. Different individuals may provide varying captions for the same image, leading to ambiguity.

Lack of detailed understanding: While image caption generators can provide high-level descriptions, they often lack fine-grained understanding and detailed analysis of the visual content. The generated captions may miss specific objects, attributes, or relationships present in the image, resulting in incomplete descriptions.

Limited creativity and adaptability: Image caption generators typically rely on pre-trained models and fixed datasets, which may limit their creativity and adaptability to novel or unseen images. They may struggle to generate captions for complex, abstract, or uncommon images that are not well-represented in the training data.

Vocabulary and language limitations: Image caption generators are constrained by the vocabulary and language used during training. If an image contains objects, scenes, or concepts that are not covered in the training data, the generator may struggle to accurately describe them or resort to generic or incorrect terms.

Sensitivity to input quality: The quality of the generated captions can be influenced by the quality and clarity of the input image. Poor image resolution, low contrast, or occluded objects may lead to less accurate or less informative captions.

Ethical considerations: Image caption generators can inadvertently reinforce biases present in the training data. If the training data contains biased or discriminatory content, the generator may generate captions that reflect those biases. Careful consideration and mitigation of biases are necessary to ensure fairness and inclusivity.

Understanding these implications and limitations is crucial for the responsible and effective use of image caption generators in various applications. Improve the performance and capabilities of these systems.

3.1 RECOMMENDATIONS FOR FUTURE WORK

Enhanced context understanding: Develop techniques that enable image caption generators to better capture contextual information, spatial relationships, and scene understanding. This can involve integrating techniques from visual reasoning, commonsense reasoning, and incorporating contextual information from surrounding text or multimodal inputs.

Fine-grained image understanding: Improve the generators' ability to detect and describe fine-grained details, specific attributes, and relationships between objects in images. This can be achieved through more sophisticated object recognition, segmentation, and reasoning models.

Personalization and user preferences: Explore methods to personalize generated captions based on individual user preferences, taking into account their demographic information, cultural background, and language style. This can involve developing adaptive models that learn from user feedback and adapt the generated captions accordingly.

Real-time adaptation and dynamic captioning: Investigate techniques for real-time image captioning, enabling the generators to process and generate captions for images on the fly. This can be valuable for applications such as live video captioning, augmented reality, and interactive media

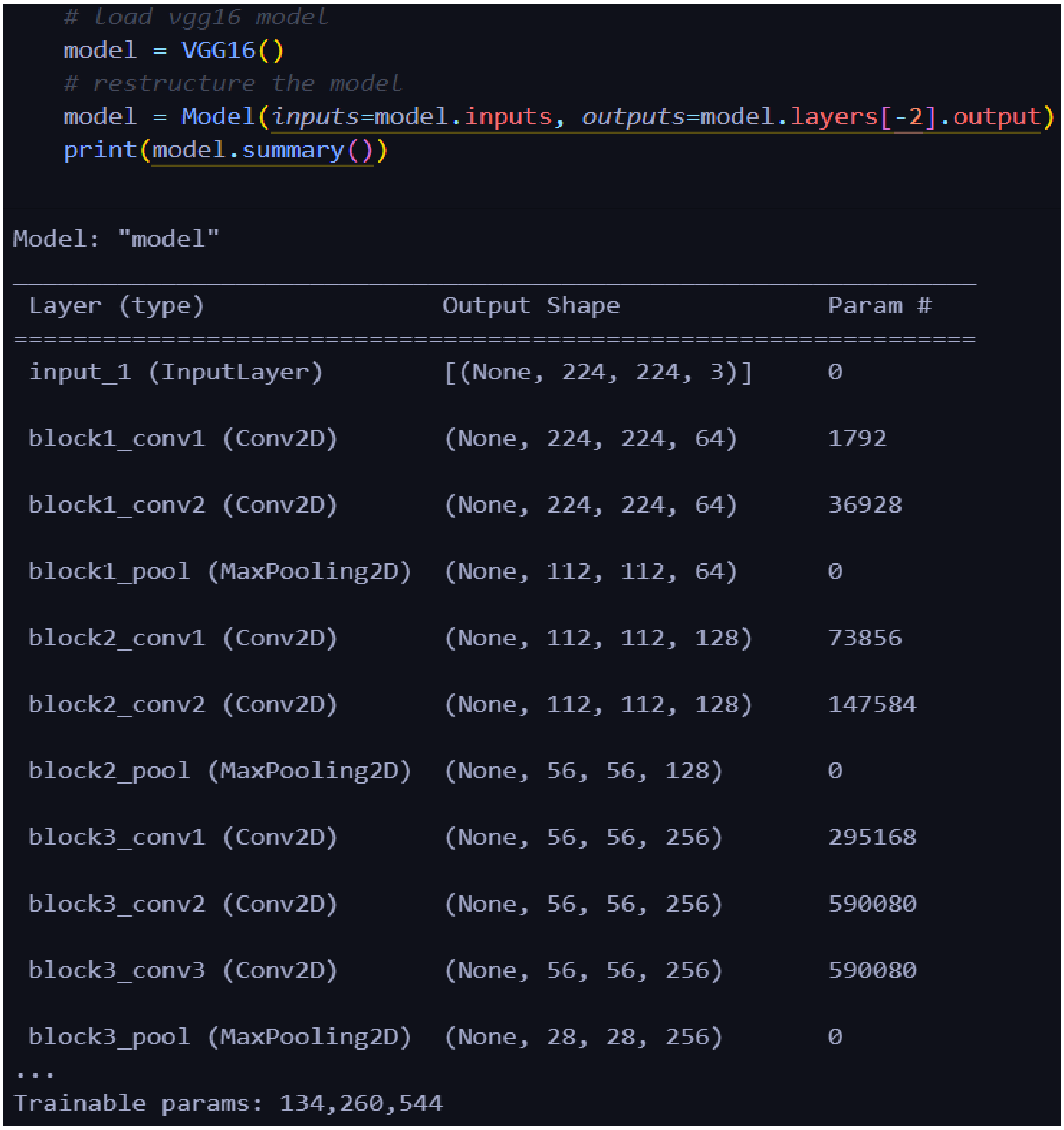
Mitigating biases: Address the issue of biases in training data and generated captions. Develop techniques to identify and mitigate biases, promote fairness, and ensure inclusivity in the generated captions. This can involve careful curation of training data, algorithmic approaches to debiasing, and incorporating ethical considerations throughout the development process.

Incorporating user feedback: Explore methods to incorporate user feedback on generated captions to improve their quality and relevance. This can involve interactive caption generation, where users can provide corrections or suggestions, and the model adapts based on this feedback.

Dataset diversity and benchmarking: Expand and diversify the datasets used for training and evaluation of image caption generators. Include a wider range of image types, scenes, and objects to ensure that the models generalize well to real-world scenarios. Develop standardized benchmarks and evaluation metrics that capture different aspects of caption quality and performance.

CHAPTER – 4

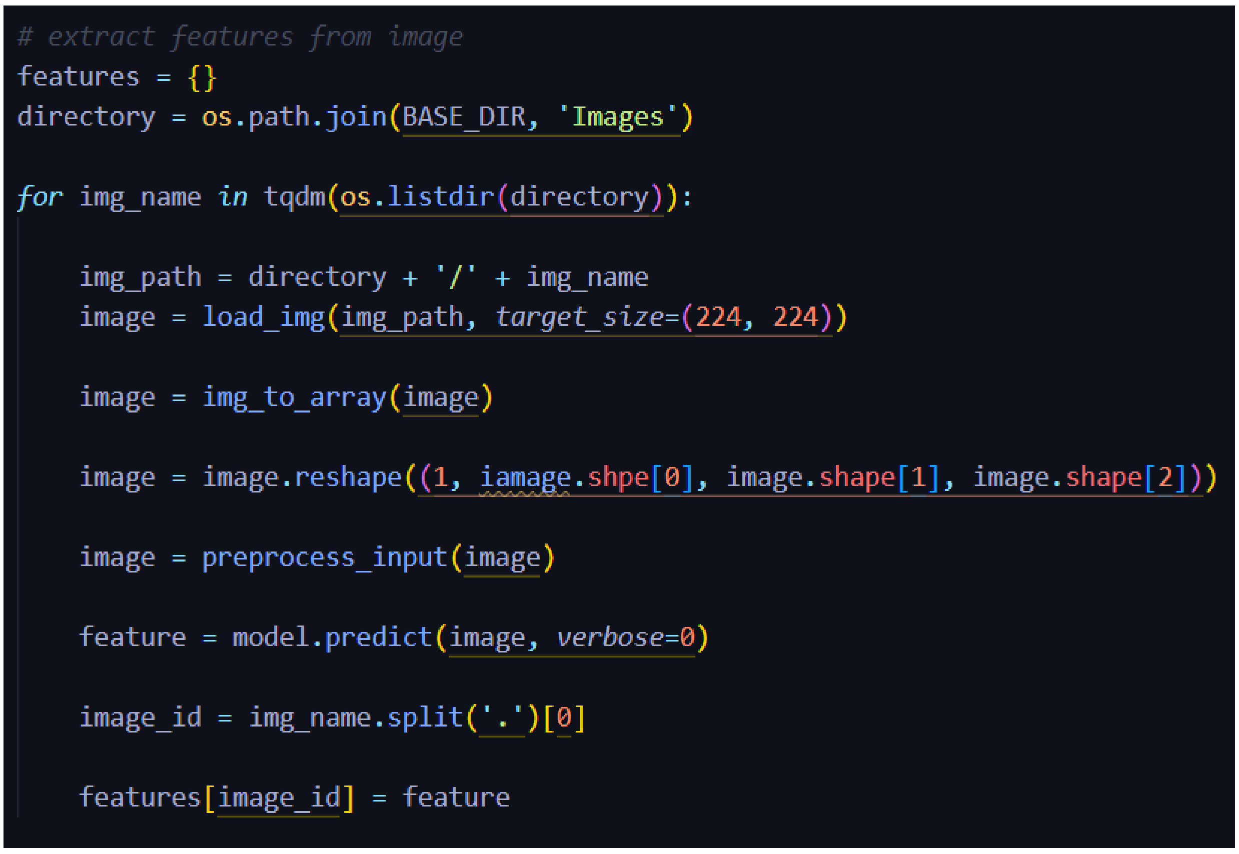
RESULTS AND DISCUSSION



```
    # load vgg16 model
    model = VGG16()
    # restructure the model
    model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
    print(model.summary())
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0
...
Trainable params: 134,260,544
```

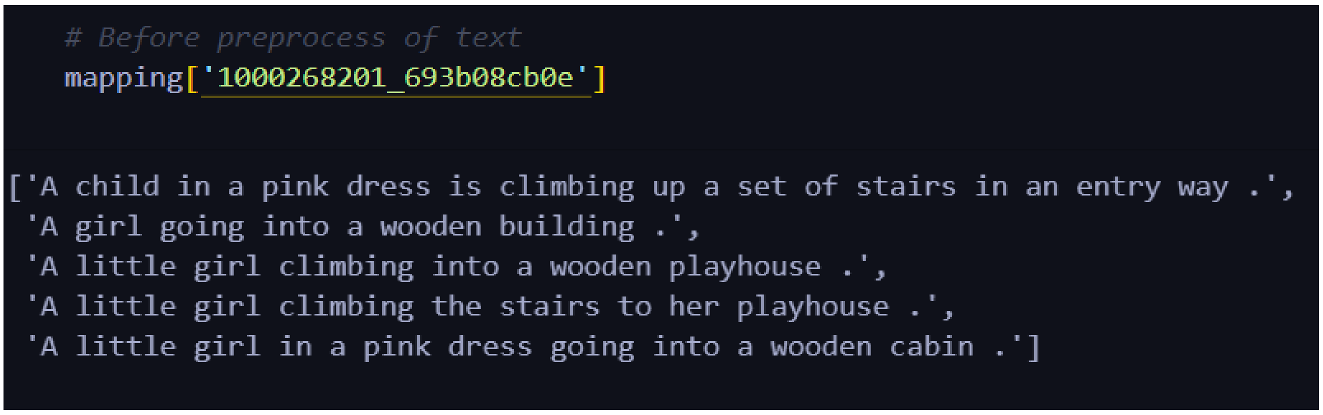
Fig 2: Restructuring the VGG Model for Extraction



```
# extract features from image
features = {}
directory = os.path.join(BASE_DIR, 'Images')

for img_name in tqdm(os.listdir(directory)):

    img_path = directory + '/' + img_name
    image = load_img(img_path, target_size=(224, 224))

    image = img_to_array(image)

    image = image.reshape((1, iamage.shpe[0], image.shape[1], image.shape[2]))

    image = preprocess_input(image)

    feature = model.predict(image, verbose=0)

    image_id = img_name.split('.')[0]

    features[image_id] = feature
```

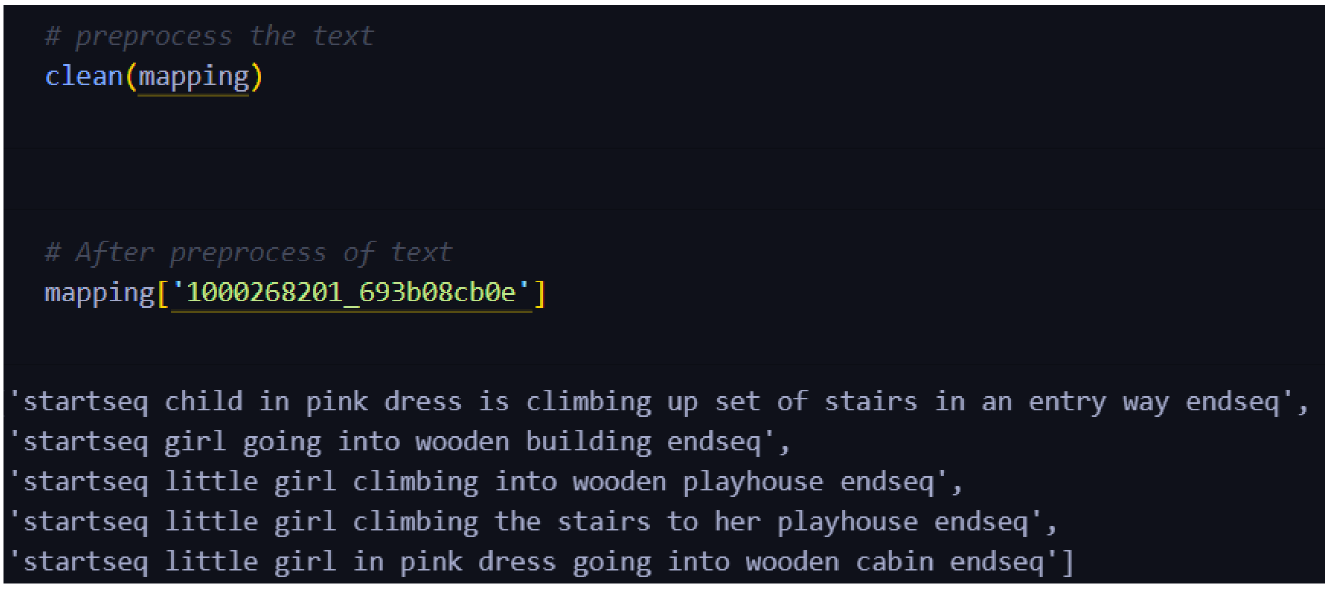
```
100% ████████████ 8091/8091 [11:40&lt;00:00, 12.32it/s]
```

Fig 3: Extracting the Features from the Image Dataset



```
# Before preprocess of text
mapping['1000268201_693b08cb0e']

['A child in a pink dress is climbing up a set of stairs in an entry way .',
 'A girl going into a wooden building .',
 'A little girl climbing into a wooden playhouse .',
 'A little girl climbing the stairs to her playhouse .',
 'A little girl in a pink dress going into a wooden cabin .']
```

Fig 4: Before Preprocessing the Text data



```
# preprocess the text
clean(mapping)

# After preprocess of text
mapping['1000268201_693b08cb0e']

'startseq child in pink dress is climbing up set of stairs in an entry way endseq',
'startseq girl going into wooden building endseq',
'startseq little girl climbing into wooden playhouse endseq',
'startseq little girl climbing the stairs to her playhouse endseq',
'startseq little girl in pink dress going into wooden cabin endseq']
```

Fig 5: After Preprocessing the Text data

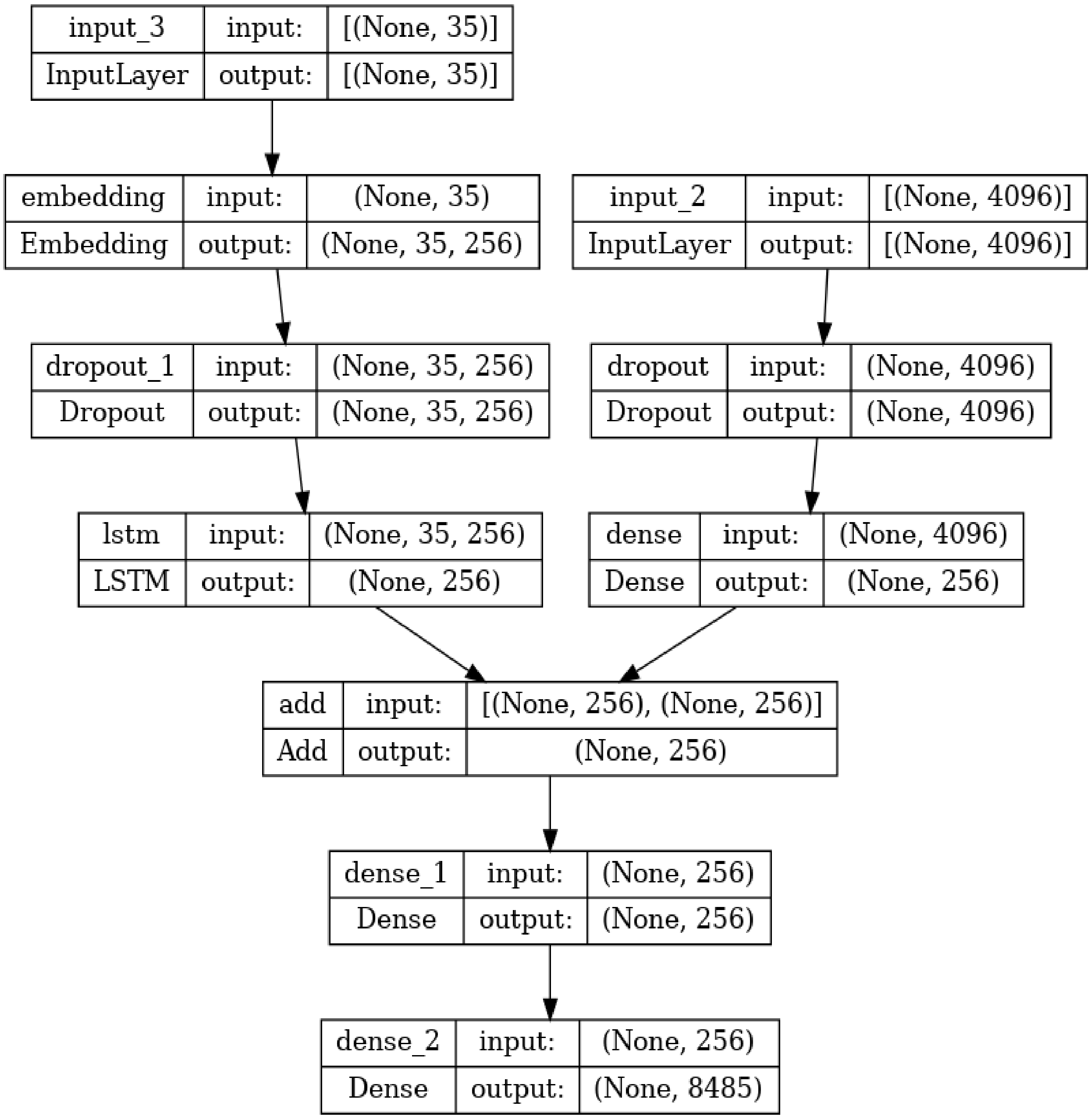
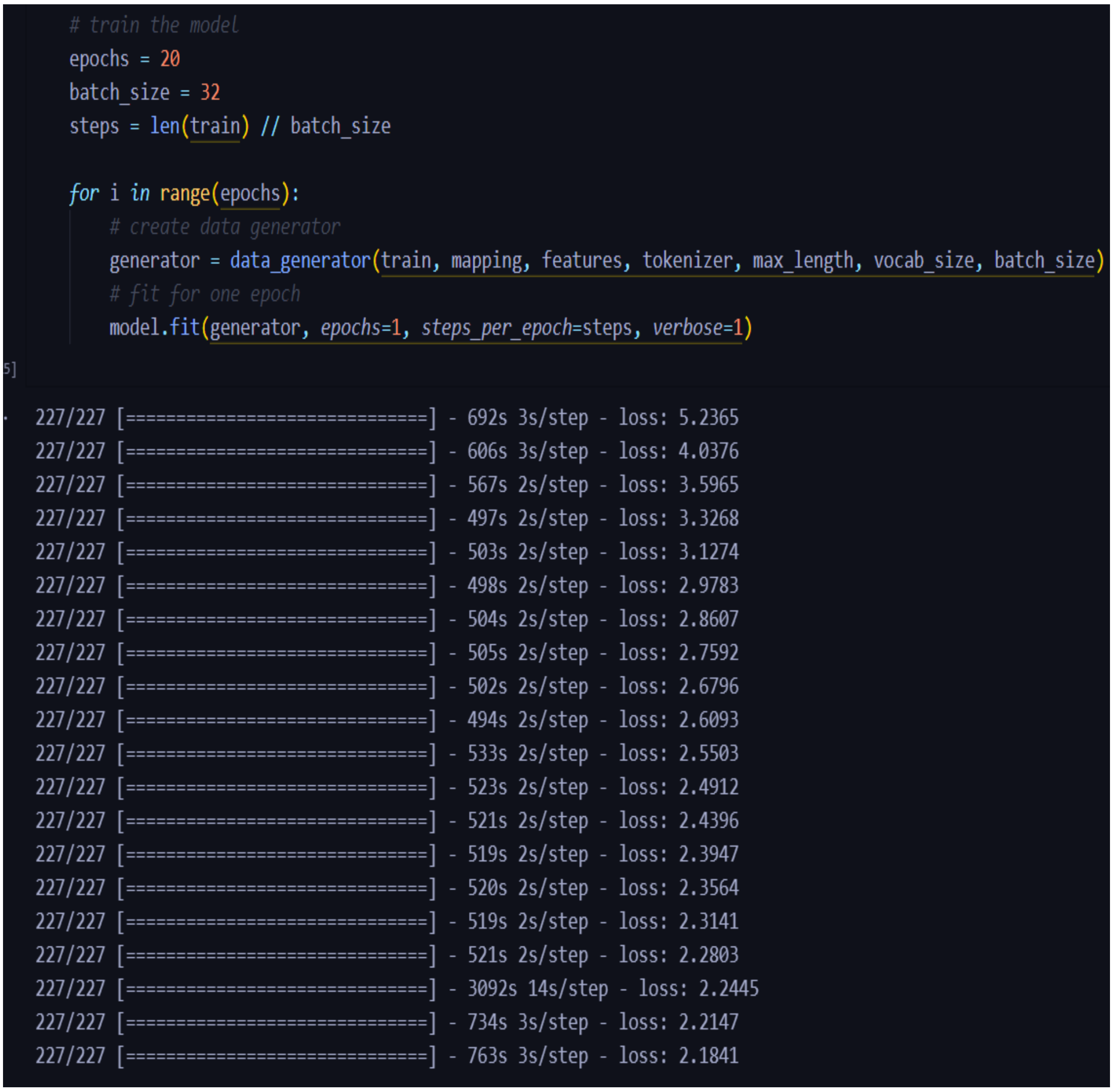

input_3 | input: | [(None, 35)]
InputLayer | output: | [(None, 35)]
embedding | input: | (None, 35)
Embedding | output: | (None, 35, 256)
input_2 | input: | [(None, 4096)]
InputLayer | output: | [(None, 4096)]
dropout_1 | input: | (None, 35, 256)
Dropout | output: | (None, 35, 256)
dropout | input: | (None, 4096)
Dropout | output: | (None, 4096)
lstm | input: | (None, 35, 256)
LSTM | output: | (None, 256)
dense | input: | (None, 4096)
Dense | output: | (None, 256)
add | input: | [(None, 256), (None, 256)]
Add | output: | (None, 256)
dense_1 | input: | (None, 256)
Dense | output: | (None, 256)
dense_2 | input: | (None, 256)
Dense | output: | (None, 8485)

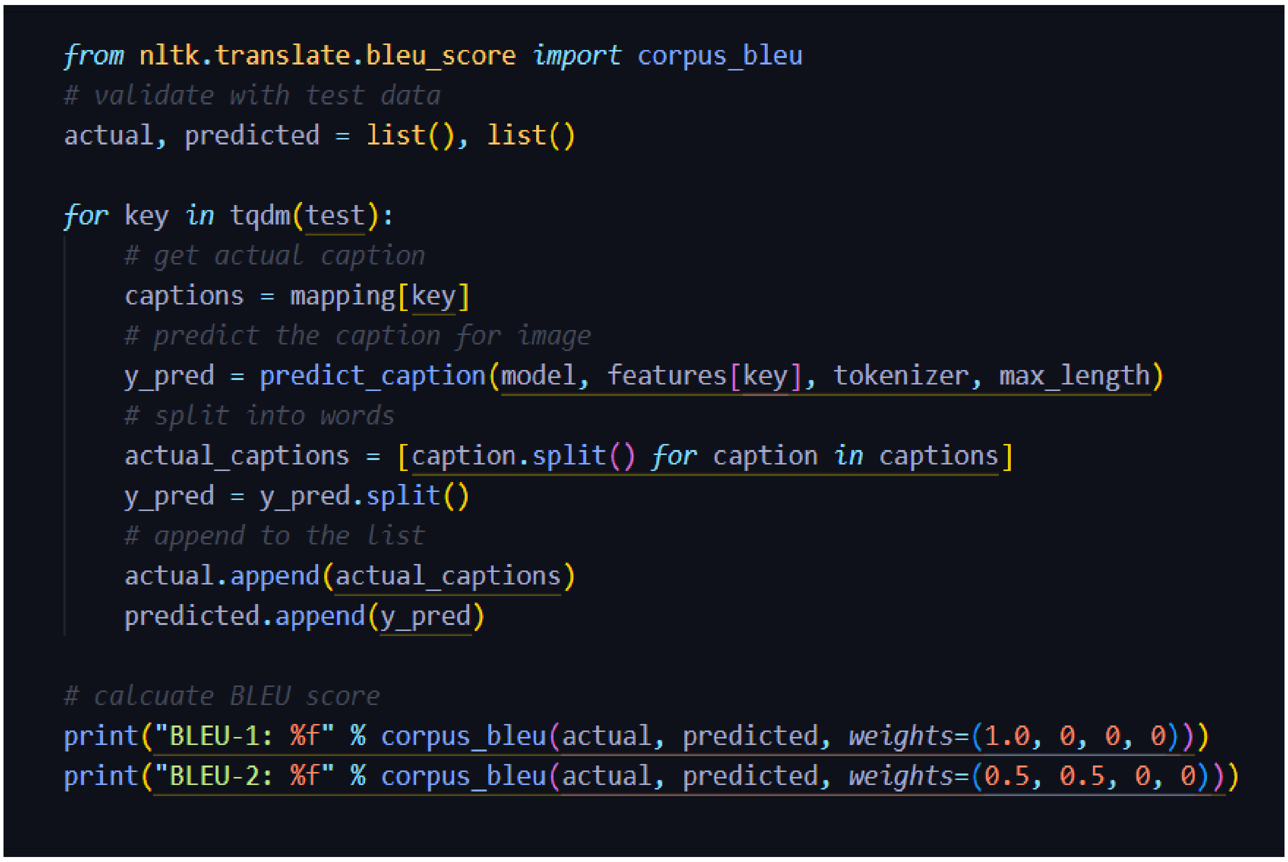
Fig 6: Representation of the Model in the Graph



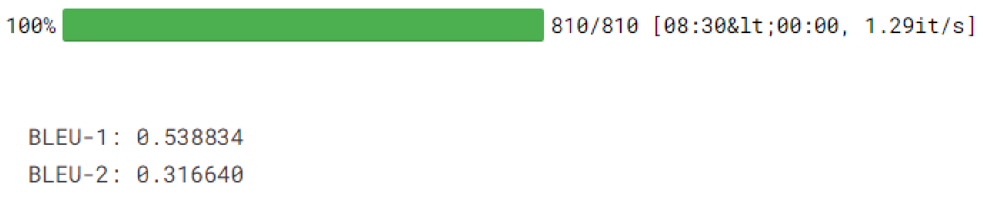
```
# train the model
epochs = 20
batch_size = 32
steps = len(train) // batch_size

for i in range(epochs):
    # create data generator
    generator = data_generator(train, mapping, features, tokenizer, max_length, vocab_size, batch_size)
    # fit for one epoch
    model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1)
```

```
227/227 [==============================] - 692s 3s/step - loss: 5.2365
227/227 [==============================] - 606s 3s/step - loss: 4.0376
227/227 [==============================] - 567s 2s/step - loss: 3.5965
227/227 [==============================] - 497s 2s/step - loss: 3.3268
227/227 [==============================] - 503s 2s/step - loss: 3.1274
227/227 [==============================] - 498s 2s/step - loss: 2.9783
227/227 [==============================] - 504s 2s/step - loss: 2.8607
227/227 [==============================] - 505s 2s/step - loss: 2.7592
227/227 [==============================] - 502s 2s/step - loss: 2.6796
227/227 [==============================] - 494s 2s/step - loss: 2.6093
227/227 [==============================] - 533s 2s/step - loss: 2.5503
227/227 [==============================] - 523s 2s/step - loss: 2.4912
227/227 [==============================] - 521s 2s/step - loss: 2.4396
227/227 [==============================] - 519s 2s/step - loss: 2.3947
227/227 [==============================] - 520s 2s/step - loss: 2.3564
227/227 [==============================] - 519s 2s/step - loss: 2.3141
227/227 [==============================] - 521s 2s/step - loss: 2.2803
227/227 [==============================] - 3092s 14s/step - loss: 2.2445
227/227 [==============================] - 734s 3s/step - loss: 2.2147
227/227 [==============================] - 763s 3s/step - loss: 2.1841
```

Fig 7: Training the Model with Approximately 20 Epochs

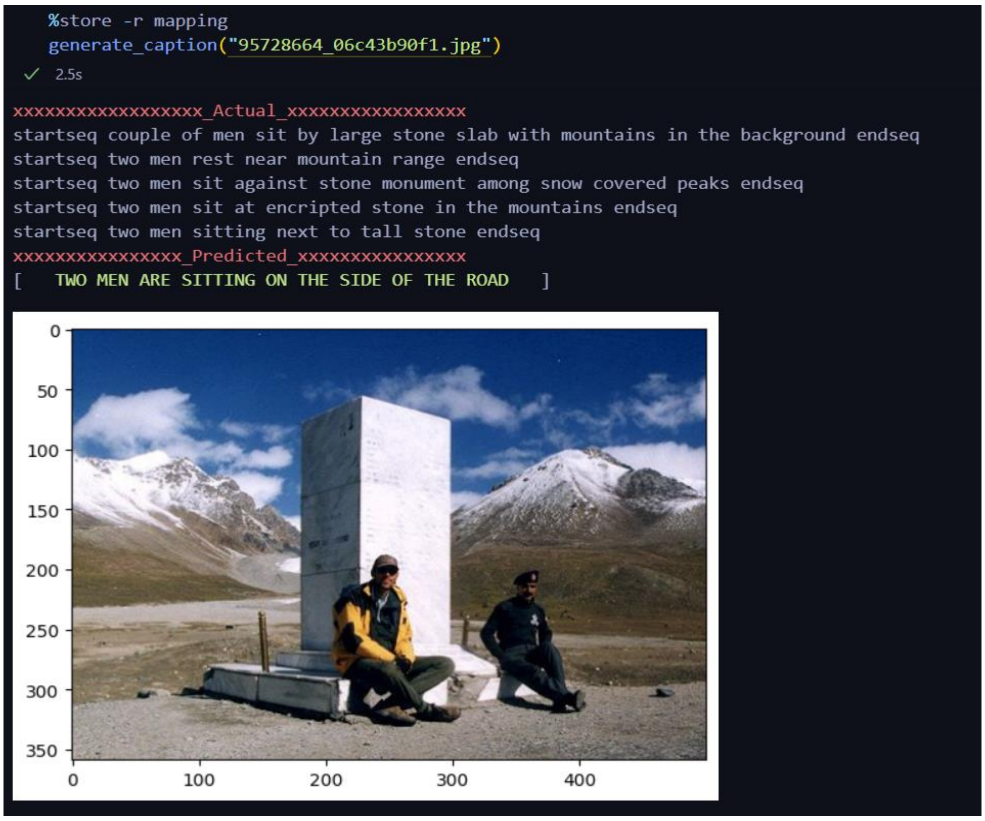


```python
from nltk.translate.bleu_score import corpus_bleu
# validate with test data
actual, predicted = list(), list()

for key in tqdm(test):
    # get actual caption
    captions = mapping[key]
    # predict the caption for image
    y_pred = predict_caption(model, features[key], tokenizer, max_length)
    # split into words
    actual_captions = [caption.split() for caption in captions]
    y_pred = y_pred.split()
    # append to the list
    actual.append(actual_captions)
    predicted.append(y_pred)

# calcuate BLEU score
print("BLEU-1: %f" % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
print("BLEU-2: %f" % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
```



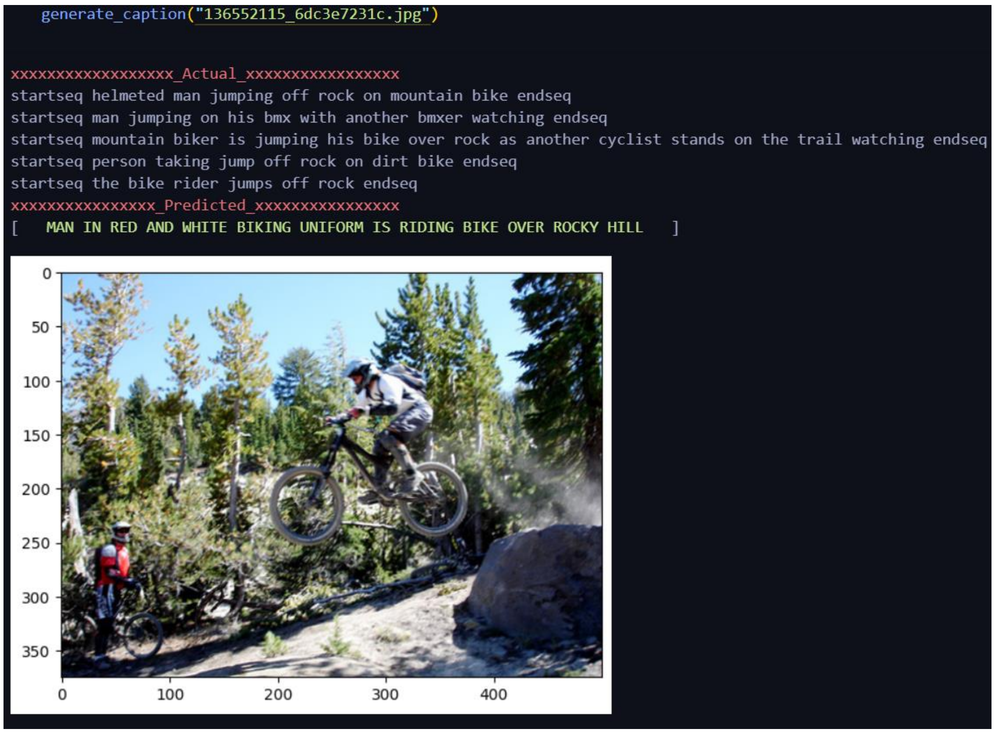
```
100% ████████████████████ 810/810 [08:30&lt;00:00, 1.29it/s]


 BLEU-1: 0.538834
 BLEU-2: 0.316640
```

Fig 8: Calculating the BLEU-1 and BLUE-2 Scores



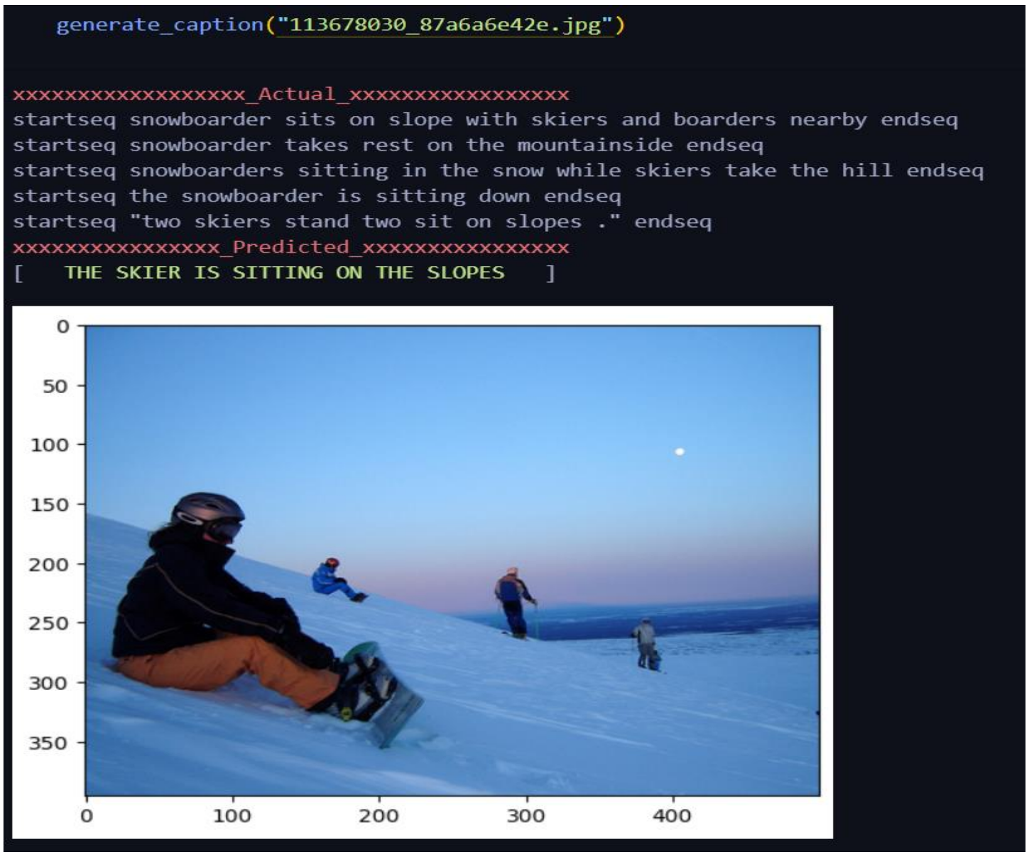
```
%store -r mapping
generate_caption("95728664_06c43b90f1.jpg")
✓ 2.5s

xxxxxxxxxxxxxxxxxx_Actual_xxxxxxxxxxxxxxxxx
startseq couple of men sit by large stone slab with mountains in the background endseq
startseq two men rest near mountain range endseq
startseq two men sit against stone monument among snow covered peaks endseq
startseq two men sit at encripted stone in the mountains endseq
startseq two men sitting next to tall stone endseq
xxxxxxxxxxxxxxxx_Predicted_xxxxxxxxxxxxxxxx
[   TWO MEN ARE SITTING ON THE SIDE OF THE ROAD    ]
```

Fig 9: Predicted Image Caption for the Image



```
generate_caption("136552115_6dc3e7231c.jpg")

xxxxxxxxxxxxxxxxxx_Actual_xxxxxxxxxxxxxxxxx
startseq helmeted man jumping off rock on mountain bike endseq
startseq man jumping on his bmx with another bmxer watching endseq
startseq mountain biker is jumping his bike over rock as another cyclist stands on the trail watching endseq
startseq person taking jump off rock on dirt bike endseq
startseq the bike rider jumps off rock endseq
xxxxxxxxxxxxxxxx_Predicted_xxxxxxxxxxxxxxxx
[   MAN IN RED AND WHITE BIKING UNIFORM IS RIDING BIKE OVER ROCKY HILL    ]
```

Fig 10: Image Caption Generation Actual vs Predicted



```
generate_caption("113678030_87a6a6e42e.jpg")

xxxxxxxxxxxxxxxxxx_Actual_xxxxxxxxxxxxxxxxx
startseq snowboarder sits on slope with skiers and boarders nearby endseq
startseq snowboarder takes rest on the mountainside endseq
startseq snowboarders sitting in the snow while skiers take the hill endseq
startseq the snowboarder is sitting down endseq
startseq "two skiers stand two sit on slopes ." endseq
xxxxxxxxxxxxxxxx_Predicted_xxxxxxxxxxxxxxxx
[    THE SKIER IS SITTING ON THE SLOPES    ]
```

Fig 11: Another Example

CHAPTER -5

CONCLUSION AND FUTURE WORK

In conclusion, image caption generators have important implications in terms of accessibility, content indexing, retrieval, and content summarization. They bridge the gap between visuals and text, promoting multimodal understanding. However, they also come with limitations like subjectivity, lack of detailed understanding, and reliance on training data.

Future work for image caption generators includes improving contextual understanding, capturing finer image details, personalizing captions, enabling real-time adaptation, mitigating biases, incorporating user feedback, integrating multiple modalities, diversifying datasets, addressing ethical considerations, and supporting multilingual and cross-cultural captioning. These advancements aim to enhance the generators' accuracy, adaptability, fairness, and utility across various applications.

Continuous research and development in these areas are crucial for maximizing the potential of image caption generators, ensuring their reliability, inclusivity, and effectiveness in generating relevant and appropriate descriptions for images. By tackling these challenges, we can advance the capabilities of image caption generators and unlock new possibilities in the fields of computer vision and natural language processing.

Furthermore, future work for image caption generators should focus on addressing the challenge of generating captions that capture the overall narrative or story within a sequence of images. This involves developing methods for understanding temporal relationships and contextual coherence across multiple images, enabling the generators to generate captions that describe the sequence as a whole.

Lastly, it is crucial to continue refining and expanding benchmark datasets and evaluation metrics for image caption generation. This will facilitate fair and consistent evaluation of different models and approaches, fostering healthy competition and driving further advancements in the field.

By addressing these additional points, future work on image caption generators can further enhance their capabilities, leading to more accurate, creative, interpretable, and efficient systems that can cater to a wider range of applications and user needs.

CHAPTER – 6

APPENDIX

Image Caption Generator.Py

```
import os

import pickle

import numpy as np

from tqdm.notebook import tqdm



from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input

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

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad_sequences

from tensorflow.keras.models import Model

from tensorflow.keras.utils import to_categorical, plot_model

from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, add




# In[2]:

import tensorflow as tf

print(tf.__version__)
```

```
# In[2]:

BASE_DIR =  'Dataset'

WORKING_DIR = 'Working dir'




# In[4]:

print(BASE_DIR)


# ## Extract Image Features

# In[4]:

# load vgg16 model

model = VGG16()

# restructure the model

model = Model(inputs=model.inputs, outputs=model.layers[-2].output)

print(model.summary())




# In[5]:

 # extract features from image

features = {}

directory = os.path.join(BASE_DIR, 'Images')
```

```
for img_name in tqdm(os.listdir(directory)):

    img_path = directory + '/' + img_name

    image = load_img(img_path, target_size=(224, 224))

    image = img_to_array(image)

    image = image.reshape((1, iamage.shpe[0], image.shape[1], image.shape[2]))

    image = preprocess_input(image)

     feature = model.predict(image, verbose=0)

    image_id = img_name.split('.')[0]

    features[image_id] = feature




# In[6]:

# store features in pickle

pickle.dump(features, open(os.path.join(WORKING_DIR, 'features.pkl'), 'wb'))




# In[11]:

# load features from pickle

with open(os.path.join(WORKING_DIR, 'features.pkl'), 'rb') as f:

    features = pickle.load(f)
```

```
# ## Load the Captions Data


# In[ ]:



with open(os.path.join(BASE_DIR, 'captions.txt'), 'r') as f:

    next(f)

    captions_doc = f.read()



# In[9]:

# create mapping of image to captions

mapping = {}

# process lines

for line in tqdm(captions_doc.split('\n')):


    tokens = line.split(',')

    if len(line) < 2:

        continue

    image_id, caption = tokens[0], tokens[1:]

    # remove extension from image ID
    image_id = image_id.split('.')[0]

    # convert caption list to string

    caption = " ".join(caption)

    # create list if needed

    if image_id not in mapping:

        mapping[image_id] = []

    # store the caption

    mapping[image_id].append(caption)




# In[10]:

len(mapping)




# ## Preprocess Text Data

# In[11]:

def clean(mapping):

    for key, captions in mapping.items():

        for i in range(len(captions)):



            caption = captions[i]

            # preprocessing steps
        caption = caption.lower()

        caption = caption.replace('[^A-Za-z]', '')

        caption = caption.replace('\s+', ' ')

        caption = 'startseq ' + " ".join([word for word in caption.split() if len(word)>1]) + ' endseq'

        captions[i] = caption




# In[12]:

# Before preprocess of text

mapping['1000268201_693b08cb0e']



# In[13]:

# preprocess the text

clean(mapping)



# In[14]:

# After preprocess of text

mapping['1000268201_693b08cb0e']




# In[15]:
```

```
all_captions = []

for key in mapping:

    for caption in mapping[key]:

        all_captions.append(caption)



# In[16]:

len(all_captions)



# In[17]:

all_captions[:10]





# In[9]:

# Tokenize the text

get_ipython().run_line_magic('store', '-r all_captions')

tokenizer = Tokenizer()

tokenizer.fit_on_texts(all_captions)

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

```
# In[19]:

vocab_size




# In[11]:

# get maximum length of the caption available

max_length = max(len(caption.split()) for caption in all_captions)

max_length




# ## Train Test Split

# In[21]:

image_ids = list(mapping.keys())

split = int(len(image_ids) * 0.90)

train = image_ids[:split]

test = image_ids[split:]




# In[23]:

# create data generator to get data in batch (avoids session crash or the Kernal Crash)

def data_generator(data_keys, mapping, features, tokenizer, max_length, vocab_size, batch_size):

    # loop over images
```

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

n = 0

while 1:

    for key in data_keys:

        n += 1

        captions = mapping[key]

        # process each caption

        for caption in captions:

            # encode the sequence

            seq = tokenizer.texts_to_sequences([caption])[0]

            # split the sequence into X, y pairs

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

                # split into input and output pairs

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

                # pad input sequence

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

                # encode output sequence

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



                # store the sequences

                X1.append(features[key][0])

                X2.append(in_seq)
            y.append(out_seq)
        if n == batch_size:
            X1, X2, y = np.array(X1), np.array(X2), np.array(y)
            yield [X1, X2], y
            X1, X2, y = list(), list(), list()
            n = 0



# ## Model Creation
# In[24]:
# encoder model
# image feature layers
inputs1 = Input(shape=(4096,))
fe1 = Dropout(0.4)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
# sequence feature layers
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
se2 = Dropout(0.4)(se1)
se3 = LSTM(256)(se2)
```

```
# decoder model
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)

model = Model(inputs=[inputs1, inputs2], outputs=outputs)
model.compile(loss='categorical_crossentropy', optimizer='adam')

# plot the model
plot_model(model, show_shapes=True)


# In[25]:
# Train the model
epochs = 20
batch_size = 32
steps = len(train) // batch_size

for i in range(epochs):
    # create data generator
    generator = data_generator(train, mapping, features, tokenizer, max_length, vocab_size, 
batch_size)
```

```
    # fit for one epoch

    model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1)




# In[26]:

# Save the Best model

model.save(WORKING_DIR+'/best_model.h5')




# ## Generate Captions for the Image

# In[13]:

def idx_to_word(integer, tokenizer):

    for word, index in tokenizer.word_index.items():

        if index == integer:

            return word

    return None




# In[8]:

# generate caption for an image

def predict_caption(model, image, tokenizer, max_length):

    # add start tag for generation process
```

```
in_text = 'startseq'
# iterate over the max length of sequence
for i in range(max_length):
    # encode input sequence
    sequence = tokenizer.texts_to_sequences([in_text])[0]
    # pad the sequence
    sequence = pad_sequences([sequence], max_length)
    # predict next word
    yhat = model.predict([image, sequence], verbose=0)
    # get index with high probability
    yhat = np.argmax(yhat)
    # convert index to word
    word = idx_to_word(yhat, tokenizer)
    # stop if word not found
    if word is None:
        break
    # append word as input for generating next word
    in_text += " " + word
    # stop if we reach end tag
    if word == 'endseq':
        break
  return in_text
```

```
# In[31]:

from nltk.translate.bleu_score import corpus_bleu

# validate with test data

actual, predicted = list(), list()


for key in tqdm(test):

    # get actual caption

    captions = mapping[key]

    # predict the caption for image

    y_pred = predict_caption(model, features[key], tokenizer, max_length)

    # split into words

    actual_captions = [caption.split() for caption in captions]

    y_pred = y_pred.split()

    # append to the list

    actual.append(actual_captions)

    predicted.append(y_pred)


# calcuate BLEU score

print("BLEU-1: %f" % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))

print("BLEU-2: %f" % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
```

```
# ## Visualize the Results

# In[9]:

from PIL import Image

import os

import matplotlib.pyplot as plt

def generate_caption(image_name):

    # load the image

    # for Example : image_name = "101654506_8eb26cfb60.jpg"

    image_id = image_name.split('.')[0]

    img_path = os.path.join(BASE_DIR, "Images", image_name)

    image = Image.open(img_path)

    captions = mapping[image_id]

    print('---------------------Actual---------------------')

    for caption in captions:

        print(caption)

    # prediction of  the caption

    y_pred = predict_caption(model, features[image_id], tokenizer, max_length)

    print('--------------------Predicted--------------------')

    print(y_pred)

    plt.imshow(image)
```

```
# In[ ]:

get_ipython().run_line_magic('store', '-r mapping')

generate_caption("1002674143_1b742ab4b8.jpg")


# In[13]:

generate_caption("101654506_8eb26cfb60.jpg")


# In[35]:

generate_caption("101669240_b2d3e7f17b.jpg")


# ## Test with Real Image


# In[4]:

vgg_model = VGG16()

vgg_model = Model(inputs=vgg_model.inputs, outputs=vgg_model.layers[-2].output)


# In[ ]:

from tensorflow.keras.models import load_model

# load the saved model

model = load_model(r'Working dir\best_model.h5')
```

```
image_path = r"C:/Users/SabareeshReddy/Desktop/rainbow.png"

# load image

image = load_img(image_path, target_size=(224, 224))

image = img_to_array(image)

image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))

image = preprocess_input(image)

feature = vgg_model.predict(image, verbose=0)

result = predict_caption(model, feature, tokenizer, max_length)

result = result.replace("startseq", "").replace("endseq", "")

print(result)

image = Image.open(image_path)

image.show()


# ## Store all the Variables Present in the Session & Use it for later use.

# In[69]:

get_ipython().run_line_magic('store', 'model mapping all_captions')

get_ipython().run_line_magic('store', '')




# # Run Final code

# ## Run with Real Time Image
# In[1]:

import os

import pickle

import numpy as np

from tqdm.notebook import tqdm



from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input

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

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad_sequences

from tensorflow.keras.models import Model

from tensorflow.keras.utils import to_categorical, plot_model

from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, add

from tensorflow.keras.models import load_model



vgg_model = VGG16()

# restructure the model

vgg_model = Model(inputs=vgg_model.inputs, outputs=vgg_model.layers[-2].output)



get_ipython().run_line_magic('store', '-r all_captions')

tokenizer = Tokenizer()
```

```
tokenizer.fit_on_texts(all_captions)

vocab_size = len(tokenizer.word_index) + 1

max_length = max(len(caption.split()) for caption in all_captions)


from PIL import Image

import os

import matplotlib.pyplot as plt

def generate_caption(image_name):

    # load the image

    image_id = image_name.split('.')[0]

    img_path = os.path.join(BASE_DIR, "Images", image_name)

    image = Image.open(img_path)

    captions = mapping[image_id]

    print('---------------------Actual---------------------')

    for caption in captions:

        print(caption)

    # predict the caption

    y_pred = predict_caption(model, features[image_id], tokenizer, max_length)

    print('--------------------Predicted--------------------')

    print(y_pred)

    plt.imshow(image)
```

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


def predict_caption(model, image, tokenizer, max_length):
    # add start tag for generation process
    in_text = 'startseq'
    # iterate over the max length of sequence
    for i in range(max_length):
        # encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # pad the sequence
        sequence = pad_sequences([sequence], max_length)
        # predict next word
        yhat = model.predict([image, sequence], verbose=0)
        # get index with high probability
        yhat = np.argmax(yhat)
        # convert index to word
        word = idx_to_word(yhat, tokenizer)
        # stop if word not found
        if word is None:
            break
        # append word as input for generating next word
        in_text += " " + word
        # stop if we reach end tag
        if word == 'endseq':
            break

    return in_text


# load the saved model
model = load_model(r'Working dir\best_model.h5')


image_path = r"C:\Users\SabareeshReddy\Desktop\download.jpeg"
# load image
image = load_img(image_path, target_size=(224, 224))
image = img_to_array(image)
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
image = preprocess_input(image)
feature = vgg_model.predict(image, verbose=0)
result = predict_caption(model, feature, tokenizer, max_length)
result = result.replace("startseq", "").replace("endseq", "")
result = result.upper()

red_start = "\033[31m"

reset_color = "\033[0m"

output = f"Predicted Caption:- [{red_start}{result}{reset_color}]"

print(output)




image = Image.open(image_path)

image.show()




# # Actual Vs Predicted



# In[3]:





from PIL import Image

import os

import pickle

import matplotlib.pyplot as plt



get_ipython().run_line_magic('store', '-r all_captions')
```

```
tokenizer = Tokenizer()

tokenizer.fit_on_texts(all_captions)

vocab_size = len(tokenizer.word_index) + 1

max_length = max(len(caption.split()) for caption in all_captions)


with open(os.path.join(WORKING_DIR, 'features.pkl'), 'rb') as f:

    features = pickle.load(f)


def idx_to_word(integer, tokenizer):

    for word, index in tokenizer.word_index.items():

        if index == integer:

            return word

    return None


def predict_caption(model, image, tokenizer, max_length):

    # add start tag for generation process

    in_text = 'startseq'

    # iterate over the max length of sequence

    for i in range(max_length):

        # encode input sequence

        sequence = tokenizer.texts_to_sequences([in_text])[0]

        # pad the sequence
        sequence = pad_sequences([sequence], max_length)
        # predict next word
        yhat = model.predict([image, sequence], verbose=0)
        # get index with high probability
        yhat = np.argmax(yhat)
        # convert index to word
        word = idx_to_word(yhat, tokenizer)
        # stop if word not found
        if word is None:
            break
        # append word as input for generating next word
        in_text += " " + word
        # stop if we reach end tag
        if word == 'endseq':
            break

    return in_text


def generate_caption(image_name):
    # load the image
    # image_name = "1001773457_577c3a7d70.jpg"
    image_id = image_name.split('.')[0]
    img_path = os.path.join(BASE_DIR, "Images", image_name)

    image = Image.open(img_path)

    captions = mapping[image_id]

    red_start = "\033[31m"

    reset_color = "\033[0m"

    print(red_start +'xxxxxxxxxxxxxxxxxxx_Actual_xxxxxxxxxxxxxxxxxx'+ reset_color)

    for caption in captions:

        print(caption)

    # predict the caption

    y_pred = predict_caption(model, features[image_id], tokenizer, max_length)

    print(red_start +'xxxxxxxxxxxxxxxxx_Predicted_xxxxxxxxxxxxxxxxx'+ reset_color)

    y_pred= y_pred.replace("startseq", "").replace("endseq", "")

    y_pred = y_pred.upper()

    yellow_start = "\033[32m"

    reset_color = "\033[0m"

    output = f"[{yellow_start}  {y_pred}  {reset_color}]"

    print(output)

    plt.imshow(image)


BASE_DIR =  'Dataset'

WORKING_DIR = 'Working dir'
```

```
directory = os.path.join(BASE_DIR, 'Images')

get_ipython().run_line_magic('store', '-r mapping')

generate_caption("95728664_06c43b90f1.jpg")


# In[4]:

generate_caption("136552115_6dc3e7231c.jpg")


# In[5]:

generate_caption("113678030_87a6a6e42e.jpg")
```

CHAPTER – 7

REFERENCES

Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2015). Show and tell: A neural image caption generator. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 3156-3164).

Xu, K., Ba, J., Kiros, R., Courville, A., Salakhudinov, R., Zemel, R., & Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In International Conference on Machine Learning (ICML) (pp. 2048-2057).

Johnson, J., Karpathy, A., & Fei-Fei, L. (2016). Densecap: Fully convolutional localization networks for dense captioning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 4565-4574).

Anderson, P., He, X., Buehler, C., Teney, D., Johnson, M., Gould, S., & Zhang, L. (2018). 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 (CVPR) (pp. 6077-6086).

Lu, J., Xiong, C., Parikh, D., & Socher, R. (2018). Knowing when to look: Adaptive attention via a visual sentinel for image captioning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 375-384).