A PROJECT REPORT ON

IMAGE CAPTION GENERATOR

SUBMITTED TOWARDS PARTIAL FULFILMENT OF THE ACADEMIC REQUIREMENT FOR THE AWARD OF DEGREE OF

MASTER OF COMPUTER APPLICATION

Submitted By

ANKITA SAHOO

2224100041

Under the esteemed guidance of

Dr. Jibitesh Mishra

Professor, DEAN of students,

and

Dr. Subasish Mohapatra

Ass. Professor

School of Computer Science,

SCHOOL OF COMPUTER SCIENCES

ODISHA UNIVERSITY OF TECHNOLOGY AND RESEARCH

(Techno Campus, Ghatikia, Mahalaxmi Vihar, Bhubaneswar-751029)

Academic Year 2022-2024

ODISHA UNIVERSITY OF TECHNOLOGY AND RESEARCH

(Techno Campus, Ghatikia, Mahalaxmi Vihar, Bhubaneswar-751029)

SCHOOL OF COMPUTER SCIENCES

CERTIFICATE

This is to certify that the minor project entitled **IMAGE CAPTION GENERATOR** is being submitted by **ANKITA SAHOO** with regd.no **222410041** in partial fulfillment of the requirement for the award "Master of Computer Application" degree is an authentic record of the work done by her under my supervision and guidance. The matter embodied in the project has not been submitted to any other University for the award of any degree or diploma to the best of my knowledge. The work carried out by her during the project period is original and her performance during the compilation of the project was appreciable.

INTERNAL GUIDE	INTERNAL GUIDE	HEAD OF SCHOOL
Dr. Jibitesh Mishra	**Dr. Subasish Mohapatra**	**Dr. Ranjan Kumar Dash**
Professor	Ass. Professor,	SCOS
DEAN of students	SOCS	

EXTERNAL EXAMINER

ODISHA UNIVERSITY OF TECHNOLOGY AND RESEARCH

(Techno Campus, Ghatikia, Mahalaxmi Vihar, Bhubaneswar-751029)

SCHOOL OF COMPUTER SCIENCES

DECLARATION

We hereby declare that the minor project work entitled "**IMAGE CAPTION GENERATOR**" submitted to the Odisha University of Technology and Research, Bhubaneswar is a record of original work done by me under the guidance of **Dr. Jibitesh Mishra Professor, DEAN of students** and **Dr.Subasish Mohapatra, Assistant professor, School of Computer Science** and this project work is submitted in the partial fulfillment of the award of the degree of Master of Computer Applications. The results embodied in this report have not been submitted to any other University or Institute for the award of any degree.

Date:

Student Name

Ankita Sahoo

ODISHA UNIVERSITY OF TECHNOLOGY AND RESEARCH

(Techno Campus, Ghatikia, Mahalaxmi Vihar, Bhubaneswar-751029)

SCHOOL OF COMPUTER SCIENCES

ACKNOWLEDGMENT

What I write or mention in this sheet will hardly be adequate in return for the amount of help and cooperation I have received from all the people who have contributed to making this project a reality. This project owes its existence to these people's help, support, and inspiration.

We would like to express our sincere gratitude to **Dr. Jibitesh Mishra Professor, DEAN of students** and **Dr.Subasish Mohapatra, Assistant professor, School of Computer Science**, whose knowledge and guidance have motivated me to achieve goals we never thought possible. He has consistently been a source of motivation, encouragement, and inspiration. The time we have spent working under his supervision has truly been a pleasure.

We also want to convey our deep regards to all other faculty members of MCA Dept. (School of Computer Science), who have bestowed their great effort and guidance at appropriate times without which it would have been challenging on our part to finish this work. Finally, I also want to thank our friends for their advice and for pointing out our mistakes.

Date: **ANKITA SAHOO**

Place: OUTR, BBSR **2224100041**

ABSTRACT

This project aims to develop an advanced Image Caption Generator by integrating deep learning techniques from available domain of NLP and computer vision. Utilizing the Flickr8k dataset, the model effectively generates descriptive captions for images by accurately interpreting visual content. The workflow encompasses meticulous preprocessing of images and captions, where images are encoded using a pre-trained ResNet50 model, and captions are vectorized to form structured data inputs. The innovative model architecture features separate processing branches for images and text, which are subsequently concatenated to yield a unified output. The image processing branch employs Dense layers with activation functions, dropout, and batch normalization to optimize performance and mitigate overfitting. Concurrently, the language processing branch leverages Bidirectional LSTMs to capture complex syntactic and semantic patterns in captions.

The combined model undergoes extensive training and validation on the dataset, with its efficacy evaluated through accuracy and loss metrics. This project highlights the synergistic potential of convolutional neural networks (CNNs) and recurrent neural networks (RNNs) to construct a robust system capable of generating coherent, contextually relevant captions for images.

Image captioning has significant applications across various sectors. It enhances accessibility for visually impaired individuals by providing textual descriptions of visual content, facilitating a better understanding of their surroundings and digital content. In media and publishing, automatic captioning aids in efficient content management, allowing for streamlined indexing, searching, and retrieval of images based on descriptive metadata. Social media platforms benefit from this technology by automating the captioning process, enhancing user engagement and interaction through more engaging and contextually appropriate descriptions. In e-commerce, online retail platforms can leverage image captioning to generate detailed product descriptions, enhancing product discoverability and improving the shopping experience for users. Additionally, real-time image captioning can be applied in surveillance systems to provide immediate, descriptive feedback on monitored scenes, aiding security personnel in quick and informed decision-making.

Image captioning bridges the gap between visual and textual information, enabling a richer interaction with digital content. It serves as a critical tool for enhancing user experience, accessibility, and operational efficiency across various sectors. By combining the strengths of CNNs in visual recognition with RNNs' proficiency in language generation, this project not only demonstrates the current capabilities of AI in understanding and describing visual content but also sets a foundation for future advancements in multimedia content processing and AI-driven user interfaces. The continual improvement in image captioning models will lead to more intelligent systems capable of seamlessly integrating visual and textual data, thereby enhancing the usability and accessibility of digital content in an increasingly image-centric world.

TABLE OF CONTENTS

1. CHAPTER 1:
   i. Introduction ---------- 07
2. CHAPTER 2:
   i. Software and Hardware Requirements ---------- 10
3. CHAPTER 3:
   i. Problem statement ---------- 12
   ii. Objective and Motivation ---------- 15
   iii. Literature Survey ---------- 16
4. CHAPTER 4:
   i. Data Set ---------- 19
5. CHAPTER 5:
   i. Proposed Work-Flow ----------
      22
   ii. Model Architecture ---------- 24
      a) Image Feature Extraction ---------- 24
      b) Image Model Architecture ---------- 30
      c) Language Model Architecture ---------- 32
      d) Concatenated Model Architecture ---------- 34
6. CHAPTER 6:
   i. Implementation ---------- 37
      a) Preprocess Image ---------- 38
      b) Feature Extraction ---------- 38
      c) Preprocess Text/ caption ---------- 39
      d) Model Implementation ---------- 41
      e) Training ---------- 44
7. CHAPTER 7:
   i. Result
      a) Training result ---------- 48
      b) Accuracy plot ---------- 50
      c) Model comparisons ---------- 51
      d) Captioned Image Result ---------- 53
8. CHAPTER 8:
   i. Limitations ---------- 61
   ii. Future Scope ---------- 61
9. CHAPTER 9:
   i. Conclusion ---------- 64
   ii. References ---------- 64

CHAPTER - 1

Introduction:

The task of generating descriptive captions for images is a fascinating intersection of computer vision and natural language processing (NLP). As a subset of artificial intelligence (AI), this challenge requires a model to not only recognize objects within an image but also understand the context and produce relevant textual descriptions. Image captioning has numerous practical applications, including assisting visually impaired individuals, enhancing image search engines, and automating the annotation of vast image datasets.

Traditionally, image captioning was tackled using rule-based systems that relied heavily on predefined templates and limited vocabulary. However, these systems lacked flexibility and often failed to produce natural-sounding descriptions. The advent of deep learning has revolutionized this field by enabling the development of models that can learn from data, improving the quality and accuracy of generated captions significantly.

With the rapid progress in Deep Learning over the past few decades, Neural Networks have enabled solutions to numerous complex problems. The rise of Convolutional Neural Networks (CNNs) and Long Short-Term Memory networks (LSTMs) has shown significant potential in processing image and text data. Tasks like Image Caption generation, which once appeared beyond the capabilities of machines, are now becoming increasingly achievable.

Image captioning is the task of generating descriptive and relevant sentences for a given image. This task has two sub-tasks:

- Understanding the context of the given image.
- Represent that understanding in the form of textual description.

This project uses the Flickr8k dataset, which consists of 8,091 images, each image have five different captions. This diverse dataset provides a robust foundation for training a model to understand and describe various scenes and objects.

The model architecture combines the strengths of convolutional neural networks (CNNs) and recurrent neural networks (RNNs). CNNs, like ResNet50, are highly effective in extracting rich features from images. ResNet50, in particular, is known for its deep architecture, incorporating residual connections to mitigate the vanishing gradient problem and improve training efficiency. On the other hand, RNNs, especially LSTMs (Long Short-Term Memory networks), are well-suited for sequence modeling tasks like language generation. Bidirectional LSTMs enhance this capability by considering context from both past and future sequences, leading to more coherent sentence generation.

In this project, the ResNet50 model is employed to extract a 2048-dimensional feature vector from each image. These features are then fed into a sequential model that includes Dense layers with ReLU activation functions, dropout, and batch normalization to prevent overfitting and stabilize training. Simultaneously, captions are preprocessed by tokenizing the text, creating word-to-index mappings, and padding sequences to ensure uniform length. These sequences are then used to train an embedding-based language model, incorporating Bidirectional LSTMs to capture the sequential nature of the data.

The combined model, which merges the image and language models, is trained using the categorical cross-entropy loss function and the RMSprop optimizer. The training process involves feeding the image features and the corresponding padded caption sequences into the model, allowing it to learn the mapping between visual features and textual descriptions.

This report provides a detailed overview of the project's various stages, including data preprocessing, model architecture, training, and evaluation. The results are visualized through accuracy and loss plots, demonstrating the model's performance and potential areas for further improvement.

Image captioning is a crucial and relevant field of research with far-reaching implications and applications. Its ability to enhance accessibility, improve search and retrieval, facilitate content understanding, and support multimodal interaction makes it an essential technology in today's digital landscape. The ongoing research and development in image captioning are driving advancements in AI and machine learning, fostering interdisciplinary collaboration, and providing valuable benchmarks for evaluating progress. With a wide range of use cases, image captioning is poised to continue making significant contributions to various industries and improving the user experience across different domains.

CHAPTER - 2

SOFTWARE AND HARDWARE REQUIREMENTS

1. Software requirement

1.1. Jupyter Notebook / Google colab
1.2. Python libraries
    1.2.1. Matplotlib
    1.2.2. Seaborn
    1.2.3. Numpy
    1.2.4. Pandas
    1.2.5. Pip
1.3. SK-learn
1.4. TensorFlow, Keras
1.5. ResNet 50
1.6. IPython

2. HARDWARE REQUIREMENTS

1.1. Windows 10 64 Bit

1.2. Intel Processor i5 10th Generation

1.3. VRAM: 16 GB GDDR6

1.4. GPU Tesla K80, 2496 CUDA cores , 12GB GDDR5 VRAM

1.5. GPU Architecture: NVIDIA Turing

1.6. CUDA Cores per GPU: 2560

1.7. Tensor Cores per GPU: 320

CHAPTER - 3

PROBLEM STATEMENT

Generating accurate and meaningful captions for images is a challenging task that requires a deep understanding of both visual content and natural language. The problem can be broken down into several key challenges:

- Visual Feature Extraction: The model must be able to recognize and extract relevant features from images, distinguishing between different objects, actions, and scenes. This involves dealing with varying lighting conditions, perspectives, and occlusions.

- Language Understanding: The model needs to generate grammatically correct and contextually appropriate sentences. This requires a comprehensive understanding of vocabulary, grammar, and the ability to form coherent sentences that accurately describe the visual content.

- Contextual Relevance: Ensuring that the generated captions are contextually relevant and accurately reflect the content of the image. This involves not only identifying objects but also understanding their relationships and the overall context of the scene.

Examples:

Simple Object Recognition:

Figure-1 : Example image 01

Image: A dog playing with a ball.
Caption: "A dog is playing with a blue ball in a field."

Complex Scenes:

Figure-2 : Example image 02

Image: A group of people sitting around a table having a meal.
Caption: "A group of people are sitting around a table enjoying a meal together."

Ambiguous Contexts:

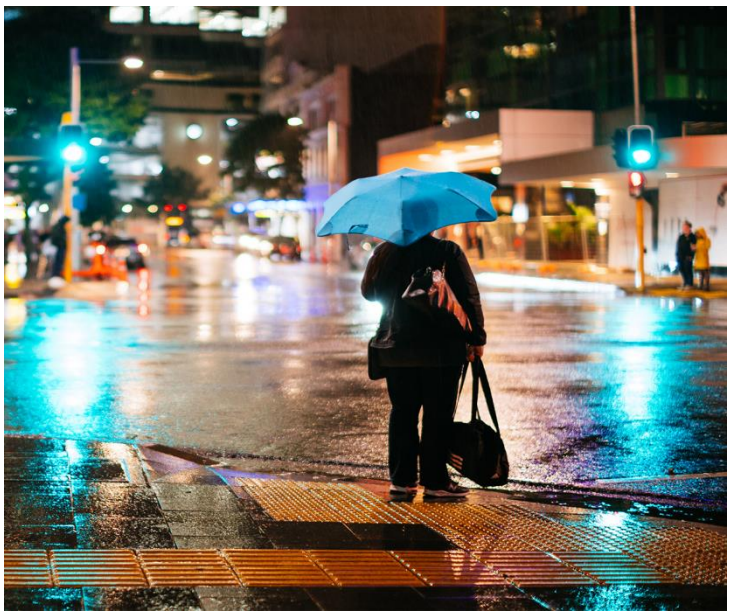


Figure-3 : Example image 03

Image: A person holding an umbrella on a busy street.
Caption: "A person is holding a blue umbrella in a busy street."

The challenge lies in ensuring that the model can handle diverse and complex scenes, generating captions that are both accurate and meaningful. The objective of this project is to develop a robust image caption generator that can effectively address these challenges, providing a scalable solution for various applications, such as aiding visually impaired individuals, improving image search engines, and automating image annotation processes.

Objective and Motivation:

The primary objective of this project is to develop an efficient and accurate image caption generator using deep learning techniques. The specific goals include:

- Feature Extraction: Utilize a pre-trained ResNet50 model to extract rich and relevant features from images, ensuring that the model can handle a wide variety of visual content.

- Caption Generation: Implement a Bidirectional LSTM-based language model that can generate grammatically correct and contextually appropriate captions for the given images.

- Model Integration: Combine the image and language models into a unified architecture that leverages the strengths of both CNNs and RNNs, ensuring that the final model is capable of producing high-quality captions.

- Training and Evaluation: Train the model on the Flickr8k dataset, validate its performance, and fine-tune the hyperparameters to achieve optimal results. Evaluate the model's performance using accuracy and loss metrics, and visualize the training progress.

- Practical Applications: Demonstrate the practical applications of the developed model, showcasing its ability to generate captions for a variety of images, and explore potential use cases in assisting visually impaired individuals, enhancing image search engines, and automating image annotation.

Motivation

The motivation behind this project stems from the profound impact that accurate image captioning can have across various domains. Understanding and describing visual content is a fundamental aspect of human communication, and replicating this ability in machines can bring about significant advancements and benefits.

Applications of Image caption Generators are.

1. Self driving cars — Automatic driving is one of the biggest challenges and if we can properly caption the scene around the car, it can give a boost to the self driving system.
2. Aid to the blind — We can create a product for the blind which will guide them travelling on the roads without the support of anyone else. This can be done by converting the image into text and then the text to speech. Refer this link where its shown how Nvidia research is trying to create such a product.
3. CCTV cameras integration: CCTV are everywhere today, but along with viewing the world, if we can also generate relevant captions, then we can raise alarms as soon as there is some malicious activity going on somewhere. This could probably help reduce some crime and/or accidents.

4. Enhance search engine: Automatic Captioning can help, make Google Image Search as good as Google Search, as then every image could be first converted into a caption and then search can be performed based on the caption.

LITERATURE SURVEY:

The development of image captioning systems has been an area of active research, driven by advancements in computer vision and NLP. Early methods primarily focused on template-based approaches, which lacked flexibility and often produced repetitive and unnatural captions. These methods were limited by their reliance on predefined rules and the inability to generalize to new or unseen images.

The introduction of deep learning, particularly CNNs and RNNs, marked a significant shift in the field. CNNs like AlexNet and VGGNet demonstrated superior performance in image recognition tasks, paving the way for their application in feature extraction for image captioning. Simultaneously, RNNs, with their capability to model sequential data, provided a natural fit for generating coherent sentences.

One of the seminal works in this domain was by Vinyals et al. (2015), who proposed the "Show and Tell" model. This approach used a CNN to encode images and an LSTM to decode these features into sentences. The model was trained end-to-end using a large dataset, demonstrating the potential of combining vision and language models.

Karpathy and Fei-Fei (2015) introduced the "Deep Visual-Semantic Alignments" model, which aligned segments of sentences with regions of images. This model leveraged a CNN-RNN architecture to learn the relationship between visual and textual data, further improving caption generation accuracy.

Subsequent research has explored various enhancements to these models. Xu et al. (2015) proposed an attention mechanism that allows the model to focus on specific parts of the image while generating each word of the caption. This approach mimics human visual attention and significantly improves the quality of generated captions.

The use of pre-trained models, such as ResNet50, has also become a common practice. ResNet50, with its deep architecture and residual connections, provides rich feature representations that are highly beneficial for image captioning tasks. These features are often combined with advanced RNN architectures, like Bidirectional LSTMs, to capture the sequential nature of language data effectively.

Despite these advancements, challenges remain. Generating captions that are both syntactically correct and semantically meaningful is a complex task. Models often struggle with understanding context and producing diverse descriptions for the same image. Additionally, the evaluation of generated captions is inherently subjective, making it difficult to measure performance accurately.

This project builds upon these advancements by integrating ResNet50 for image feature extraction and Bidirectional LSTMs for language modeling. The model aims to generate coherent and contextually relevant captions by leveraging the strengths of both CNNs and RNNs, incorporating dropout and batch normalization to enhance training stability and performance.

CHAPTER - 4

DataSet:

The Flickr8k dataset is a widely used benchmark in the field of computer vision and natural language processing, specifically for image captioning tasks. It comprises 8,000 images sourced from Flickr, each paired with five different captions that describe the content of the image. This dataset has become a standard for evaluating image captioning models due to its diverse and challenging nature. The dataset's creation involved meticulous collection and annotation processes, ensuring high-quality data for research and development in image-captioning technologies.

The Flickr8k dataset was introduced as part of a broader effort to advance the field of image captioning, which involves generating descriptive textual content based on visual inputs. The motivation behind creating this dataset was to provide a robust and standardized benchmark that researchers could use to develop and evaluate their models.

The Flickr8k dataset is a widely used benchmark in the field of image captioning and contains the following key components:

Structure of Flickr8k Dataset

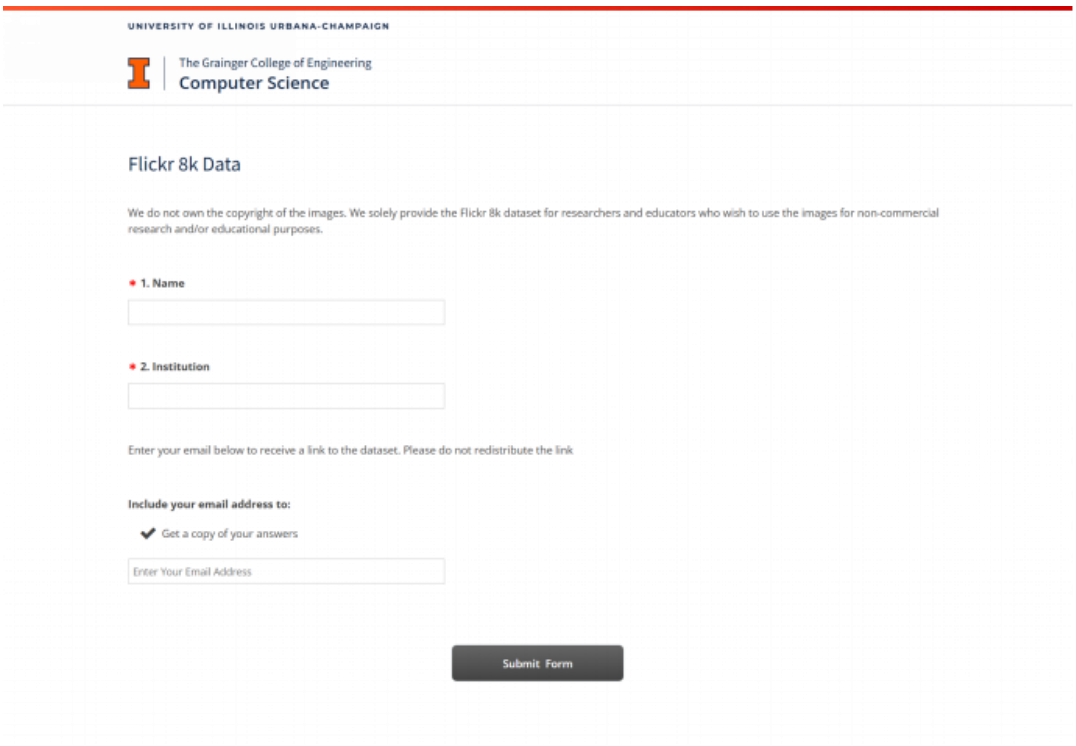
Images:
Total Images: 8,000
Source: The images are sourced from the photo-sharing website Flickr. They are diverse in nature, including various scenes, objects, people, and actions.
Captions:

Total Captions: 40,000
Captions per Image: Each image is annotated with five different captions. These captions describe the contents of the images in a detailed and varied manner, providing multiple perspectives on the same image.
Annotation:

The captions are collected through crowd-sourcing on Amazon Mechanical Turk, where human annotators are tasked with describing the images.
The annotations aim to provide a comprehensive description of the scene, objects, actions, and interactions within each image.

The Flickr8k dataset is a cornerstone in the field of image captioning, offering a robust and diverse benchmark for researchers. Its creation involved careful selection, annotation, and quality control processes to ensure high-quality data. The dataset's impact on research has been profound, driving advancements in image captioning and related fields. As the field continues to evolve, the Flickr8k dataset remains a valuable resource for developing and evaluating innovative models and techniques.

The data set is collected by filling out the form provided by the University of Illinois at Urbana-Champaign


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

The Grainger College of Engineering
Computer Science

Flickr 8k Data

We do not own the copyright of the images. We solely provide the Flickr 8k dataset for researchers and educators who wish to use the images for non-commercial research and/or educational purposes.

* 1. Name

* 2. Institution

Enter your email below to receive a link to the dataset. Please do not redistribute the link

Include your email address to:

Get a copy of your answers

Enter Your Email Address

Submit Form

CHAPTER - 5

Proposed architecture

The proposed architecture for the image caption generator involves several integrated steps, starting with the preprocessing of both image and caption data. The image data is first loaded and processed using the ResNet50 model, from which the top classification layer is removed. This adjustment enables the model to act as a feature extractor, capturing essential visual elements of the images. Concurrently, caption data undergoes preprocessing where word-index mappings are created, and the maximum caption length is calculated to standardize the sequence lengths. The images are then preprocessed and fed into the modified ResNet50 to extract feature vectors, which are subsequently saved for further use.

In parallel, caption sequences are padded to ensure uniformity in length and are one-hot encoded to convert words into a numerical format suitable for model training. The image model and language model are defined separately at this stage. The image model processes the encoded image features, transforming them into a dense vector representation. The language model, on the other hand, processes the padded caption sequences, learning the structure and context of the language used in the captions.

These two models are then concatenated to form a combined model that integrates both visual and textual information. This combined model is compiled and trained using the prepared data, optimizing for a loss function suitable for sequence prediction tasks. During training, the model learns to associate image features with the corresponding caption sequences, gradually improving its ability to generate accurate and relevant captions.

Post-training, the model's performance is evaluated by visualizing the training results, plotting accuracy, and loss curves. For generating captions for new images, the same preprocessing steps are applied: the image is processed using ResNet50 to extract features, which are then fed into the trained model to predict the caption sequence. The final output includes the display of the image alongside the generated caption, demonstrating the model's ability to interpret and describe new visual inputs accurately.
This architecture showcases a comprehensive approach to bridging computer vision and natural language processing, creating a robust system for automatic image caption generation.

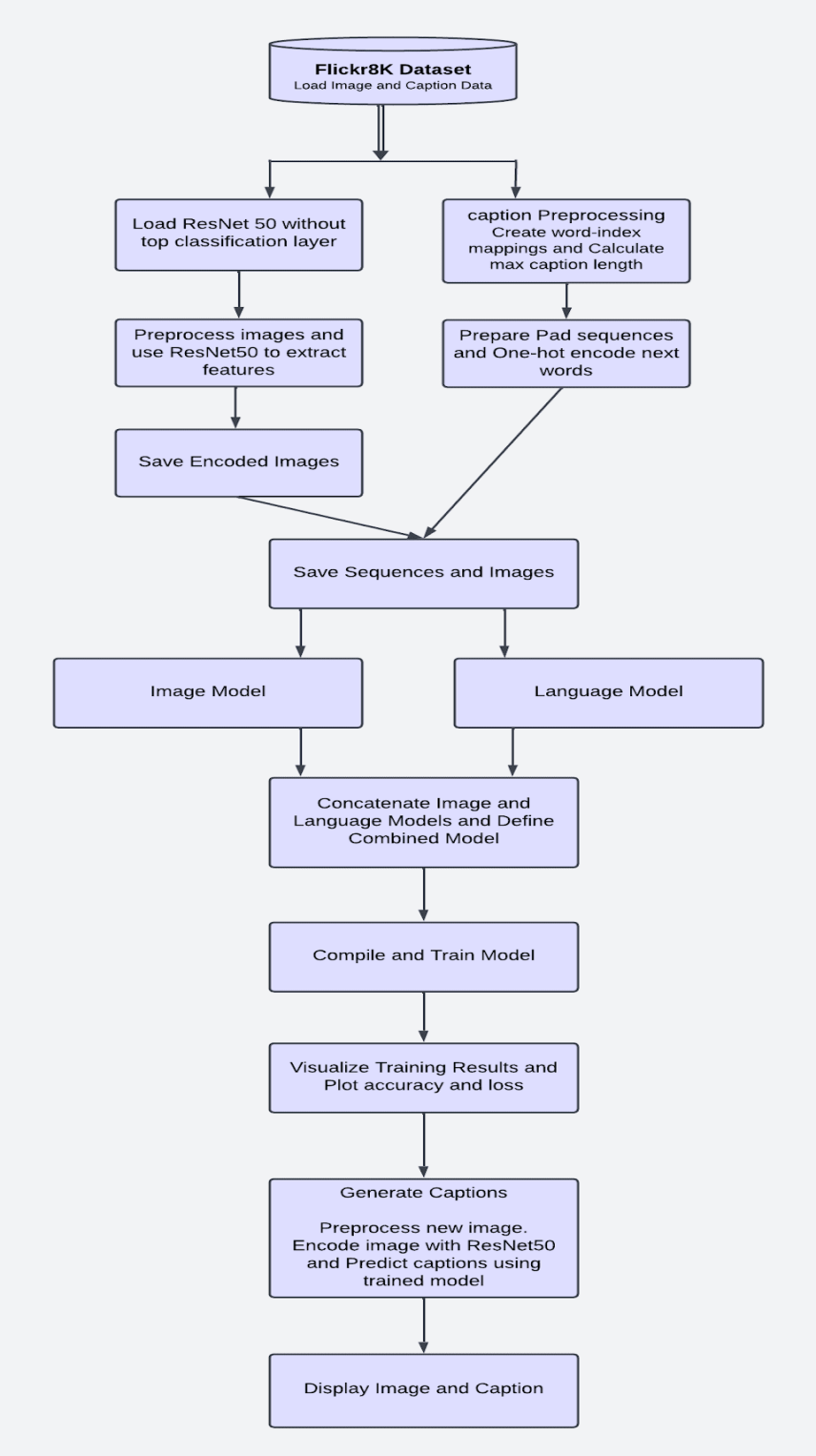


Figure-4 : Workflow Diagram

Model Architecture:

Image feature extraction:

1. ResNet 50 Architecture:

ResNet 50 (Residual Networks) is a powerful deep learning architecture known for its exceptional performance in image classification and other computer vision tasks, by Kaiming He et al. in the paper "Deep Residual Learning for Image Recognition" in 2015. The core idea behind ResNet is the introduction of residual learning, which helps in training very deep neural networks by addressing the vanishing gradient problem.

A standard ResNet architecture is composed of several stacked residual blocks. Each residual block includes a series of convolutional layers and a skip connection (or shortcut connection) that bypasses one or more layers. This skip connection allows the gradient to flow directly through the network, facilitating the training of much deeper networks than was previously possible.

Functionality of ResNet:
ResNet50 processes an input image through a series of convolutional layers, each followed by batch normalization and activation layers. The network learns to detect various features such as edges, textures, objects, and their spatial hierarchies. By the time the image reaches the global average pooling layer, it has been transformed into a high-level representation that captures the essential visual elements needed for further processing in the image captioning pipeline.

The purpose of using the ResNet50 model in the image caption generator project is to leverage its advanced capability in feature extraction from images. ResNet50, which stands for Residual Network with 50 layers, is a deep convolutional neural network (CNN) that has demonstrated exceptional performance in image recognition tasks. Here are the key reasons for using ResNet50:

- Deep Architecture with Residual Learning:

ResNet50's deep architecture consists of 50 layers, which allows it to learn complex features from images. The introduction of residual connections, or shortcuts, helps in addressing the vanishing gradient problem by allowing gradients to flow through the network more effectively during training. This enables the model to be very deep without suffering from degradation in performance.

- Feature Extraction:

For the image captioning task, the goal is to extract meaningful features that represent the visual content of an image. ResNet50 has been trained on a large dataset (such as ImageNet), allowing it to learn a rich set of features. By removing the top classification layer, ResNet50 can serve as a powerful feature extractor. The output from the penultimate layer provides a feature vector that

encapsulates the image's essential information, which can then be used by the subsequent language model to generate captions.

- Transfer Learning:

Using a pre-trained ResNet50 model leverages transfer learning, where the model's weights, pre-trained on a large dataset, are utilized for a new task. This significantly reduces the need for extensive computational resources and large datasets for training from scratch, enabling more efficient and effective training of the caption generator model.

- Robustness and Accuracy:

ResNet50 is a state of art model and known for its robustness and high accuracy in image classification and feature extraction. Its architecture is designed to capture various levels of abstraction in the image data, from low-level edges and textures to high-level object parts and configurations. This comprehensive feature extraction is crucial for generating detailed and accurate image captions.

- Standardization and Proven Success:

ResNet50 is a well-established and widely adopted model in the field of computer vision. Its success and reliability make it a go-to choice for many image-related tasks. Using a standardized model like ResNet50 ensures that the results are consistent and comparable with other works in the field.

Summery of ResNet 50 model:

Model: "resnet50"

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[(None, 224, 224, 3)	0	
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)	0	input_3[0][0]
conv1_conv (Conv2D)	(None, 112, 112, 64)	9472	conv1_pad[0][0]
conv1_bn (BatchNormalization)	(None, 112, 112, 64)	256	conv1_conv[0][0]
conv1_relu (Activation)	(None, 112, 112, 64)	0	conv1_bn[0][0]
pool1_pad (ZeroPadding2D)	(None, 114, 114, 64)	0	conv1_relu[0][0]
pool1_pool (MaxPooling2D)	(None, 56, 56, 64)	0	pool1_pad[0][0]
conv2_block1_1_conv (Conv2D)	(None, 56, 56, 64)	4160	pool1_pool[0][0]
conv2_block1_1_bn (BatchNormali	(None, 56, 56, 64)	256	conv2_block1_1_conv[0][0]
conv2_block1_1_relu (Activation	(None, 56, 56, 64)	0	conv2_block1_1_bn[0][0]
conv2_block1_2_conv (Conv2D)	(None, 56, 56, 64)	36928	conv2_block1_1_relu[0][0]
conv2_block1_2_bn (BatchNormali	(None, 56, 56, 64)	256	conv2_block1_2_conv[0][0]
conv2_block1_2_relu (Activation	(None, 56, 56, 64)	0	conv2_block1_2_bn[0][0]
conv2_block1_0_conv (Conv2D)	(None, 56, 56, 256)	16640	pool1_pool[0][0]
conv2_block1_3_conv (Conv2D)	(None, 56, 56, 256)	16640	conv2_block1_2_relu[0][0]

conv2_block1_0_bn (BatchNormali (None, 56, 56, 256) 1024 conv2_block1_0_conv[0][0]

conv2_block1_3_bn (BatchNormali (None, 56, 56, 256) 1024 conv2_block1_3_conv[0][0]

conv2_block1_add (Add) (None, 56, 56, 256) 0 conv2_block1_0_bn[0][0]
conv2_block1_3_bn[0][0]

conv2_block1_out (Activation) (None, 56, 56, 256) 0 conv2_block1_add[0][0]

conv2_block2_1_conv (Conv2D) (None, 56, 56, 64) 16448 conv2_block1_out[0][0]

conv2_block2_1_bn (BatchNormali (None, 56, 56, 64) 256 conv2_block2_1_conv[0][0]

conv2_block2_1_relu (Activation (None, 56, 56, 64) 0 conv2_block2_1_bn[0][0]

conv2_block2_2_conv (Conv2D) (None, 56, 56, 64) 36928 conv2_block2_1_relu[0][0]

conv2_block2_2_bn (BatchNormali (None, 56, 56, 64) 256 conv2_block2_2_conv[0][0]

conv2_block2_2_relu (Activation (None, 56, 56, 64) 0 conv2_block2_2_bn[0][0]

conv2_block2_3_conv (Conv2D) (None, 56, 56, 256) 16640 conv2_block2_2_relu[0][0]

conv2_block2_3_bn (BatchNormali (None, 56, 56, 256) 1024 conv2_block2_3_conv[0][0]

conv2_block2_add (Add) (None, 56, 56, 256) 0 conv2_block1_out[0][0]
conv2_block2_3_bn[0][0]

conv2_block2_out (Activation) (None, 56, 56, 256) 0 conv2_block2_add[0][0]

conv2_block3_1_conv (Conv2D) (None, 56, 56, 64) 16448 conv2_block2_out[0][0]

conv2_block3_1_bn (BatchNormali (None, 56, 56, 64) 256 conv2_block3_1_conv[0][0]

conv2_block3_1_relu (Activation (None, 56, 56, 64) 0 conv2_block3_1_bn[0][0]

conv2_block3_2_conv (Conv2D) (None, 56, 56, 64) 36928 conv2_block3_1_relu[0][0]

conv2_block3_2_bn (BatchNormali (None, 56, 56, 64) 256 conv2_block3_2_conv[0][0]

conv2_block3_2_relu (Activation (None, 56, 56, 64) 0 conv2_block3_2_bn[0][0]

conv2_block3_3_conv (Conv2D) (None, 56, 56, 256) 16640 conv2_block3_2_relu[0][0]

conv2_block3_3_bn (BatchNormali (None, 56, 56, 256) 1024 conv2_block3_3_conv[0][0]

conv2_block3_add (Add) (None, 56, 56, 256) 0 conv2_block2_out[0][0]
conv2_block3_3_bn[0][0]

conv2_block3_out (Activation) (None, 56, 56, 256) 0 conv2_block3_add[0][0]

conv3_block1_1_conv (Conv2D) (None, 28, 28, 128) 32896 conv2_block3_out[0][0]

conv3_block1_1_bn (BatchNormali (None, 28, 28, 128) 512 conv3_block1_1_conv[0][0]

conv3_block1_1_relu (Activation (None, 28, 28, 128) 0 conv3_block1_1_bn[0][0]

conv3_block1_2_conv (Conv2D) (None, 28, 28, 128) 147584 conv3_block1_1_relu[0][0]

conv3_block1_2_bn (BatchNormali (None, 28, 28, 128) 512 conv3_block1_2_conv[0][0]

conv3_block1_2_relu (Activation (None, 28, 28, 128) 0 conv3_block1_2_bn[0][0]

conv3_block1_0_conv (Conv2D) (None, 28, 28, 512) 131584 conv2_block3_out[0][0]

conv3_block1_3_conv (Conv2D) (None, 28, 28, 512) 66048 conv3_block1_2_relu[0][0]

conv3_block1_0_bn (BatchNormali (None, 28, 28, 512) 2048 conv3_block1_0_conv[0][0]

conv3_block1_3_bn (BatchNormali (None, 28, 28, 512) 2048 conv3_block1_3_conv[0][0]

conv3_block1_add (Add) (None, 28, 28, 512) 0 conv3_block1_0_bn[0][0]
conv3_block1_3_bn[0][0]

conv3_block1_out (Activation) (None, 28, 28, 512) 0 conv3_block1_add[0][0]

conv3_block2_1_conv (Conv2D) (None, 28, 28, 128) 65664 conv3_block1_out[0][0]

conv3_block2_1_bn (BatchNormali (None, 28, 28, 128) 512 conv3_block2_1_conv[0][0]

conv3_block2_1_relu (Activation (None, 28, 28, 128) 0 conv3_block2_1_bn[0][0]

```
conv3_block2_2_conv (Conv2D)    (None, 28, 28, 128)  147584      conv3_block2_1_relu[0][0]
conv3_block2_2_bn (BatchNormali (None, 28, 28, 128)  512         conv3_block2_2_conv[0][0]
conv3_block2_2_relu (Activation (None, 28, 28, 128)  0           conv3_block2_2_bn[0][0]
conv3_block2_3_conv (Conv2D)    (None, 28, 28, 512)  66048       conv3_block2_2_relu[0][0]
conv3_block2_3_bn (BatchNormali (None, 28, 28, 512)  2048        conv3_block2_3_conv[0][0]
conv3_block2_add (Add)          (None, 28, 28, 512)  0           conv3_block1_out[0][0]
                                                                 conv3_block2_3_bn[0][0]
conv3_block2_out (Activation)   (None, 28, 28, 512)  0           conv3_block2_add[0][0]
conv3_block3_1_conv (Conv2D)    (None, 28, 28, 128)  65664       conv3_block2_out[0][0]
conv3_block3_1_bn (BatchNormali (None, 28, 28, 128)  512         conv3_block3_1_conv[0][0]
conv3_block3_1_relu (Activation (None, 28, 28, 128)  0           conv3_block3_1_bn[0][0]
conv3_block3_2_conv (Conv2D)    (None, 28, 28, 128)  147584      conv3_block3_1_relu[0][0]
conv3_block3_2_bn (BatchNormali (None, 28, 28, 128)  512         conv3_block3_2_conv[0][0]
conv3_block3_2_relu (Activation (None, 28, 28, 128)  0           conv3_block3_2_bn[0][0]
conv3_block3_3_conv (Conv2D)    (None, 28, 28, 512)  66048       conv3_block3_2_relu[0][0]
conv3_block3_3_bn (BatchNormali (None, 28, 28, 512)  2048        conv3_block3_3_conv[0][0]
conv3_block3_add (Add)          (None, 28, 28, 512)  0           conv3_block2_out[0][0]
                                                                 conv3_block3_3_bn[0][0]
conv3_block3_out (Activation)   (None, 28, 28, 512)  0           conv3_block3_add[0][0]
conv3_block4_1_conv (Conv2D)    (None, 28, 28, 128)  65664       conv3_block3_out[0][0]
conv3_block4_1_bn (BatchNormali (None, 28, 28, 128)  512         conv3_block4_1_conv[0][0]
conv3_block4_1_relu (Activation (None, 28, 28, 128)  0           conv3_block4_1_bn[0][0]
conv3_block4_2_conv (Conv2D)    (None, 28, 28, 128)  147584      conv3_block4_1_relu[0][0]
conv3_block4_2_bn (BatchNormali (None, 28, 28, 128)  512         conv3_block4_2_conv[0][0]
conv3_block4_2_relu (Activation (None, 28, 28, 128)  0           conv3_block4_2_bn[0][0]
conv3_block4_3_conv (Conv2D)    (None, 28, 28, 512)  66048       conv3_block4_2_relu[0][0]
conv3_block4_3_bn (BatchNormali (None, 28, 28, 512)  2048        conv3_block4_3_conv[0][0]
conv3_block4_add (Add)          (None, 28, 28, 512)  0           conv3_block3_out[0][0]
                                                                 conv3_block4_3_bn[0][0]
conv3_block4_out (Activation)   (None, 28, 28, 512)  0           conv3_block4_add[0][0]
conv4_block1_1_conv (Conv2D)    (None, 14, 14, 256)  131328      conv3_block4_out[0][0]
conv4_block1_1_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block1_1_conv[0][0]
conv4_block1_1_relu (Activation (None, 14, 14, 256)  0           conv4_block1_1_bn[0][0]
conv4_block1_2_conv (Conv2D)    (None, 14, 14, 256)  590080      conv4_block1_1_relu[0][0]
conv4_block1_2_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block1_2_conv[0][0]
conv4_block1_2_relu (Activation (None, 14, 14, 256)  0           conv4_block1_2_bn[0][0]
conv4_block1_0_conv (Conv2D)    (None, 14, 14, 1024) 525312      conv3_block4_out[0][0]
conv4_block1_3_conv (Conv2D)    (None, 14, 14, 1024) 263168      conv4_block1_2_relu[0][0]
conv4_block1_0_bn (BatchNormali (None, 14, 14, 1024) 4096        conv4_block1_0_conv[0][0]
conv4_block1_3_bn (BatchNormali (None, 14, 14, 1024) 4096        conv4_block1_3_conv[0][0]
conv4_block1_add (Add)          (None, 14, 14, 1024) 0           conv4_block1_0_bn[0][0]
                                                                 conv4_block1_3_bn[0][0]
```

conv4_block1_out (Activation) (None, 14, 14, 1024) 0 conv4_block1_add[0][0]

conv4_block2_1_conv (Conv2D) (None, 14, 14, 256) 262400 conv4_block1_out[0][0]

conv4_block2_1_bn (BatchNormali (None, 14, 14, 256) 1024 conv4_block2_1_conv[0][0]

conv4_block2_1_relu (Activation (None, 14, 14, 256) 0 conv4_block2_1_bn[0][0]

conv4_block2_2_conv (Conv2D) (None, 14, 14, 256) 590080 conv4_block2_1_relu[0][0]

conv4_block2_2_bn (BatchNormali (None, 14, 14, 256) 1024 conv4_block2_2_conv[0][0]

conv4_block2_2_relu (Activation (None, 14, 14, 256) 0 conv4_block2_2_bn[0][0]

conv4_block2_3_conv (Conv2D) (None, 14, 14, 1024) 263168 conv4_block2_2_relu[0][0]

conv4_block2_3_bn (BatchNormali (None, 14, 14, 1024) 4096 conv4_block2_3_conv[0][0]

conv4_block2_add (Add) (None, 14, 14, 1024) 0 conv4_block1_out[0][0]
conv4_block2_3_bn[0][0]

conv4_block2_out (Activation) (None, 14, 14, 1024) 0 conv4_block2_add[0][0]

conv4_block3_1_conv (Conv2D) (None, 14, 14, 256) 262400 conv4_block2_out[0][0]

conv4_block3_1_bn (BatchNormali (None, 14, 14, 256) 1024 conv4_block3_1_conv[0][0]

conv4_block3_1_relu (Activation (None, 14, 14, 256) 0 conv4_block3_1_bn[0][0]

conv4_block3_2_conv (Conv2D) (None, 14, 14, 256) 590080 conv4_block3_1_relu[0][0]

conv4_block3_2_bn (BatchNormali (None, 14, 14, 256) 1024 conv4_block3_2_conv[0][0]

conv4_block3_2_relu (Activation (None, 14, 14, 256) 0 conv4_block3_2_bn[0][0]

conv4_block3_3_conv (Conv2D) (None, 14, 14, 1024) 263168 conv4_block3_2_relu[0][0]

conv4_block3_3_bn (BatchNormali (None, 14, 14, 1024) 4096 conv4_block3_3_conv[0][0]

conv4_block3_add (Add) (None, 14, 14, 1024) 0 conv4_block2_out[0][0]
conv4_block3_3_bn[0][0]

conv4_block3_out (Activation) (None, 14, 14, 1024) 0 conv4_block3_add[0][0]

conv4_block4_1_conv (Conv2D) (None, 14, 14, 256) 262400 conv4_block3_out[0][0]

conv4_block4_1_bn (BatchNormali (None, 14, 14, 256) 1024 conv4_block4_1_conv[0][0]

conv4_block4_1_relu (Activation (None, 14, 14, 256) 0 conv4_block4_1_bn[0][0]

conv4_block4_2_conv (Conv2D) (None, 14, 14, 256) 590080 conv4_block4_1_relu[0][0]

conv4_block4_2_bn (BatchNormali (None, 14, 14, 256) 1024 conv4_block4_2_conv[0][0]

conv4_block4_2_relu (Activation (None, 14, 14, 256) 0 conv4_block4_2_bn[0][0]

conv4_block4_3_conv (Conv2D) (None, 14, 14, 1024) 263168 conv4_block4_2_relu[0][0]

conv4_block4_3_bn (BatchNormali (None, 14, 14, 1024) 4096 conv4_block4_3_conv[0][0]

conv4_block4_add (Add) (None, 14, 14, 1024) 0 conv4_block3_out[0][0]
conv4_block4_3_bn[0][0]

conv4_block4_out (Activation) (None, 14, 14, 1024) 0 conv4_block4_add[0][0]

conv4_block5_1_conv (Conv2D) (None, 14, 14, 256) 262400 conv4_block4_out[0][0]

conv4_block5_1_bn (BatchNormali (None, 14, 14, 256) 1024 conv4_block5_1_conv[0][0]

conv4_block5_1_relu (Activation (None, 14, 14, 256) 0 conv4_block5_1_bn[0][0]

conv4_block5_2_conv (Conv2D) (None, 14, 14, 256) 590080 conv4_block5_1_relu[0][0]

conv4_block5_2_bn (BatchNormali (None, 14, 14, 256) 1024 conv4_block5_2_conv[0][0]

conv4_block5_2_relu (Activation (None, 14, 14, 256) 0 conv4_block5_2_bn[0][0]

conv4_block5_3_conv (Conv2D) (None, 14, 14, 1024) 263168 conv4_block5_2_relu[0][0]

conv4_block5_3_bn (BatchNormali (None, 14, 14, 1024) 4096 conv4_block5_3_conv[0][0]

conv4_block5_add (Add) (None, 14, 14, 1024) 0 conv4_block4_out[0][0]
conv4_block5_3_bn[0][0]

conv4_block5_out (Activation) (None, 14, 14, 1024) 0 conv4_block5_add[0][0]

conv4_block6_1_conv (Conv2D) (None, 14, 14, 256) 262400 conv4_block5_out[0][0]

conv4_block6_1_bn (BatchNormali (None, 14, 14, 256) 1024 conv4_block6_1_conv[0][0]

conv4_block6_1_relu (Activation (None, 14, 14, 256) 0 conv4_block6_1_bn[0][0]

conv4_block6_2_conv (Conv2D) (None, 14, 14, 256) 590080 conv4_block6_1_relu[0][0]

conv4_block6_2_bn (BatchNormali (None, 14, 14, 256) 1024 conv4_block6_2_conv[0][0]

conv4_block6_2_relu (Activation (None, 14, 14, 256) 0 conv4_block6_2_bn[0][0]

conv4_block6_3_conv (Conv2D) (None, 14, 14, 1024) 263168 conv4_block6_2_relu[0][0]

conv4_block6_3_bn (BatchNormali (None, 14, 14, 1024) 4096 conv4_block6_3_conv[0][0]

conv4_block6_add (Add) (None, 14, 14, 1024) 0 conv4_block5_out[0][0]
conv4_block6_3_bn[0][0]

conv4_block6_out (Activation) (None, 14, 14, 1024) 0 conv4_block6_add[0][0]

conv5_block1_1_conv (Conv2D) (None, 7, 7, 512) 524800 conv4_block6_out[0][0]

conv5_block1_1_bn (BatchNormali (None, 7, 7, 512) 2048 conv5_block1_1_conv[0][0]

conv5_block1_1_relu (Activation (None, 7, 7, 512) 0 conv5_block1_1_bn[0][0]

conv5_block1_2_conv (Conv2D) (None, 7, 7, 512) 2359808 conv5_block1_1_relu[0][0]

conv5_block1_2_bn (BatchNormali (None, 7, 7, 512) 2048 conv5_block1_2_conv[0][0]

conv5_block1_2_relu (Activation (None, 7, 7, 512) 0 conv5_block1_2_bn[0][0]

conv5_block1_0_conv (Conv2D) (None, 7, 7, 2048) 2099200 conv4_block6_out[0][0]

conv5_block1_3_conv (Conv2D) (None, 7, 7, 2048) 1050624 conv5_block1_2_relu[0][0]

conv5_block1_0_bn (BatchNormali (None, 7, 7, 2048) 8192 conv5_block1_0_conv[0][0]

conv5_block1_3_bn (BatchNormali (None, 7, 7, 2048) 8192 conv5_block1_3_conv[0][0]

conv5_block1_add (Add) (None, 7, 7, 2048) 0 conv5_block1_0_bn[0][0]
conv5_block1_3_bn[0][0]

conv5_block1_out (Activation) (None, 7, 7, 2048) 0 conv5_block1_add[0][0]

conv5_block2_1_conv (Conv2D) (None, 7, 7, 512) 1049088 conv5_block1_out[0][0]

conv5_block2_1_bn (BatchNormali (None, 7, 7, 512) 2048 conv5_block2_1_conv[0][0]

conv5_block2_1_relu (Activation (None, 7, 7, 512) 0 conv5_block2_1_bn[0][0]

conv5_block2_2_conv (Conv2D) (None, 7, 7, 512) 2359808 conv5_block2_1_relu[0][0]

conv5_block2_2_bn (BatchNormali (None, 7, 7, 512) 2048 conv5_block2_2_conv[0][0]

conv5_block2_2_relu (Activation (None, 7, 7, 512) 0 conv5_block2_2_bn[0][0]

conv5_block2_3_conv (Conv2D) (None, 7, 7, 2048) 1050624 conv5_block2_2_relu[0][0]

conv5_block2_3_bn (BatchNormali (None, 7, 7, 2048) 8192 conv5_block2_3_conv[0][0]

conv5_block2_add (Add) (None, 7, 7, 2048) 0 conv5_block1_out[0][0]
conv5_block2_3_bn[0][0]

conv5_block2_out (Activation) (None, 7, 7, 2048) 0 conv5_block2_add[0][0]

conv5_block3_1_conv (Conv2D) (None, 7, 7, 512) 1049088 conv5_block2_out[0][0]

conv5_block3_1_bn (BatchNormali (None, 7, 7, 512) 2048 conv5_block3_1_conv[0][0]

conv5_block3_1_relu (Activation (None, 7, 7, 512) 0 conv5_block3_1_bn[0][0]

conv5_block3_2_conv (Conv2D) (None, 7, 7, 512) 2359808 conv5_block3_1_relu[0][0]

conv5_block3_2_bn (BatchNormali (None, 7, 7, 512) 2048 conv5_block3_2_conv[0][0]

```
conv5_block3_2_relu (Activation (None, 7, 7, 512)    0          conv5_block3_2_bn[0][0]
conv5_block3_3_conv (Conv2D)    (None, 7, 7, 2048)   1050624    conv5_block3_2_relu[0][0]
conv5_block3_3_bn (BatchNormali (None, 7, 7, 2048)   8192       conv5_block3_3_conv[0][0]
conv5_block3_add (Add)          (None, 7, 7, 2048)   0          conv5_block2_out[0][0]
                                                                conv5_block3_3_bn[0][0]
conv5_block3_out (Activation)   (None, 7, 7, 2048)   0          conv5_block3_add[0][0]
avg_pool (GlobalAveragePooling2 (None, 2048)         0          conv5_block3_out[0][0]
==================================================================================================
```

Total params: 23,587,712
Trainable params: 23,534,592
Non-trainable params: 53,120

Changes Made to ResNet:

For our image captioning model, we utilize the ResNet50 architecture, which includes 50 layers. However, we modify the standard ResNet50 by removing its final classification layer. This alteration allows us to use ResNet50 as a feature extractor rather than a classifier. Specifically, we use the output from the global average pooling layer, which provides a 2048-dimensional feature vector for each image. These feature vectors serve as input to our image captioning model, providing a rich representation of the visual content of each image.

2. Image Model Architecture:

The image model is designed to process the 2048-dimensional feature vector extracted from ResNet50 and transform it into an embedding suitable for the captioning task. The architecture of the image model is as follows:

I. Dense Layer with ReLU Activation:

Input: 2048-dimensional feature vector. Output: 512-dimensional vector. This layer reduces the dimensionality of the feature vector and applies the ReLU activation function to introduce non-linearity.

II. Dropout Layer:

Dropout rate: 0.5. This layer helps prevent overfitting by randomly setting a fraction of input units to zero at each update during training.

III. Batch Normalization Layer:

This layer normalizes the input to the next layer, stabilizing and accelerating the training process.

IV. Dense Layer with ReLU Activation:

Input: 512-dimensional vector. Output: 256-dimensional vector. This layer further reduces the dimensionality and applies the ReLU activation function.

V. Dropout Layer:

Dropout rate: 0.5. This layer helps prevent overfitting.

VI. Batch Normalization Layer:

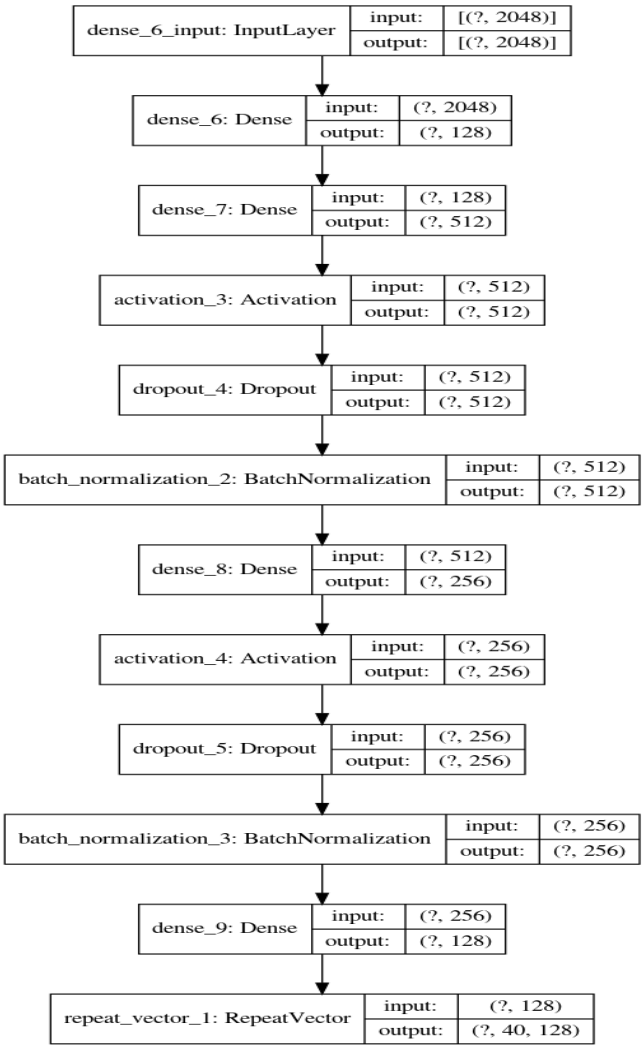
This layer normalizes the input to the next layer.

VII. Dense Layer:

Input: 256-dimensional vector. Output: 128-dimensional embedding vector. This layer maps the input to a 128-dimensional embedding space.

VIII. RepeatVector Layer:

This layer repeats the input embedding vector max_len times to match the length of the caption sequences. The output of the image model is a sequence of 128-dimensional vectors, each corresponding to a time step in the caption sequence.



:

Figure - 5 : image model architecture

3. Language Model Architecture:

The language model is designed to process the caption sequences and generate corresponding word embeddings. The architecture of the language model is as follows:

I. Embedding Layer:

Input: Caption sequences of shape (batch_size, max_len). Output: Sequences of 128-dimensional word embeddings. This layer converts each word in the caption to its corresponding embedding vector.

II. Bidirectional LSTM Layer:

Number of units: 256. This layer processes the embedding sequences in both forward and backward directions, capturing context from both ends of the sequence. Return sequences: True, to produce output for each time step.

III. Dropout Layer:

Dropout rate: 0.5. This layer helps prevent overfitting.

IV. Bidirectional LSTM Layer:

Number of units: 256. This layer processes the output of the previous TimeDistributed layer in both forward and backward directions. Return sequences: True.

V. Dropout Layer:

Dropout rate: 0.5. This layer helps prevent overfitting.

VI. TimeDistributed Dense Layer:

Output: 128-dimensional vector for each time step. This layer applies a dense transformation to each time step individually. The output of the language model is a sequence of 128-dimensional vectors, each corresponding to a time step in the caption sequence.

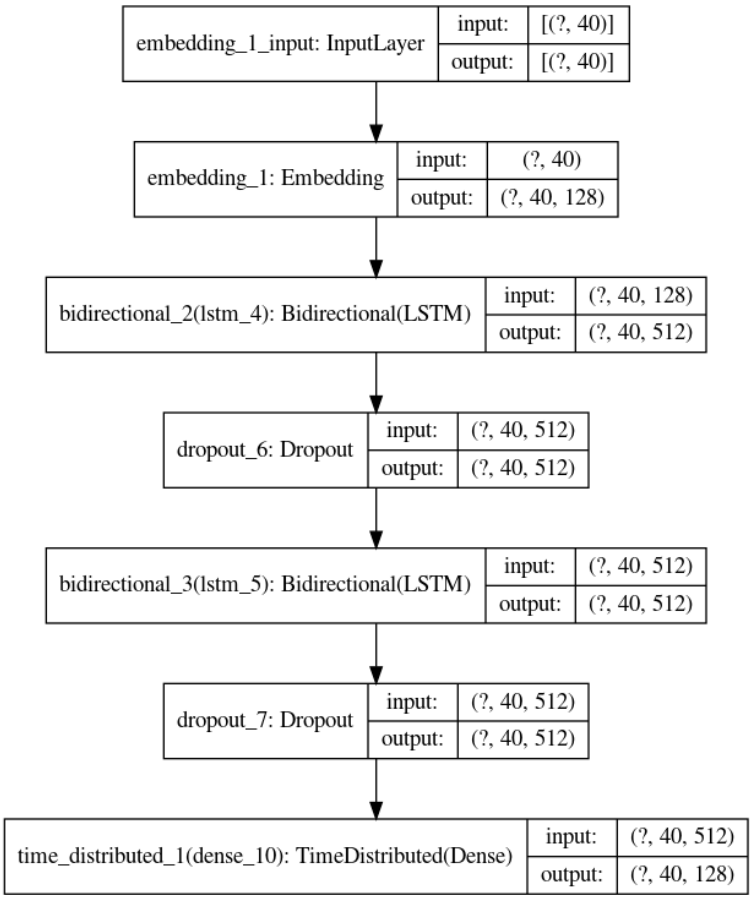


Figure - 6 : Language model architecture

4. Merged Model Architecture:

Merging Image and Language Models:

The outputs of the image model and the language model are concatenated to form a combined sequence of embeddings. This combined sequence serves as input to the next stage of the model:

I. Concatenate Layer:

This layer concatenates the output of the image model (repeated image embeddings) with the output of the language model (caption embeddings).

II. LSTM Layer:

Number of units: 128. This LSTM processes the combined sequence and returns the output for each time step.

III. LSTM Layer:

Number of units: 512. This LSTM processes the combined sequence and returns the output for the final time step only.

IV. Dense Layer:

Output: vocab_size-dimensional vector. This layer maps the LSTM output to the vocabulary space, producing logits for each word in the vocabulary.

V. Softmax Activation:

This layer applies the softmax function to produce a probability distribution over the vocabulary, indicating the likelihood of each word being the next word in the caption.

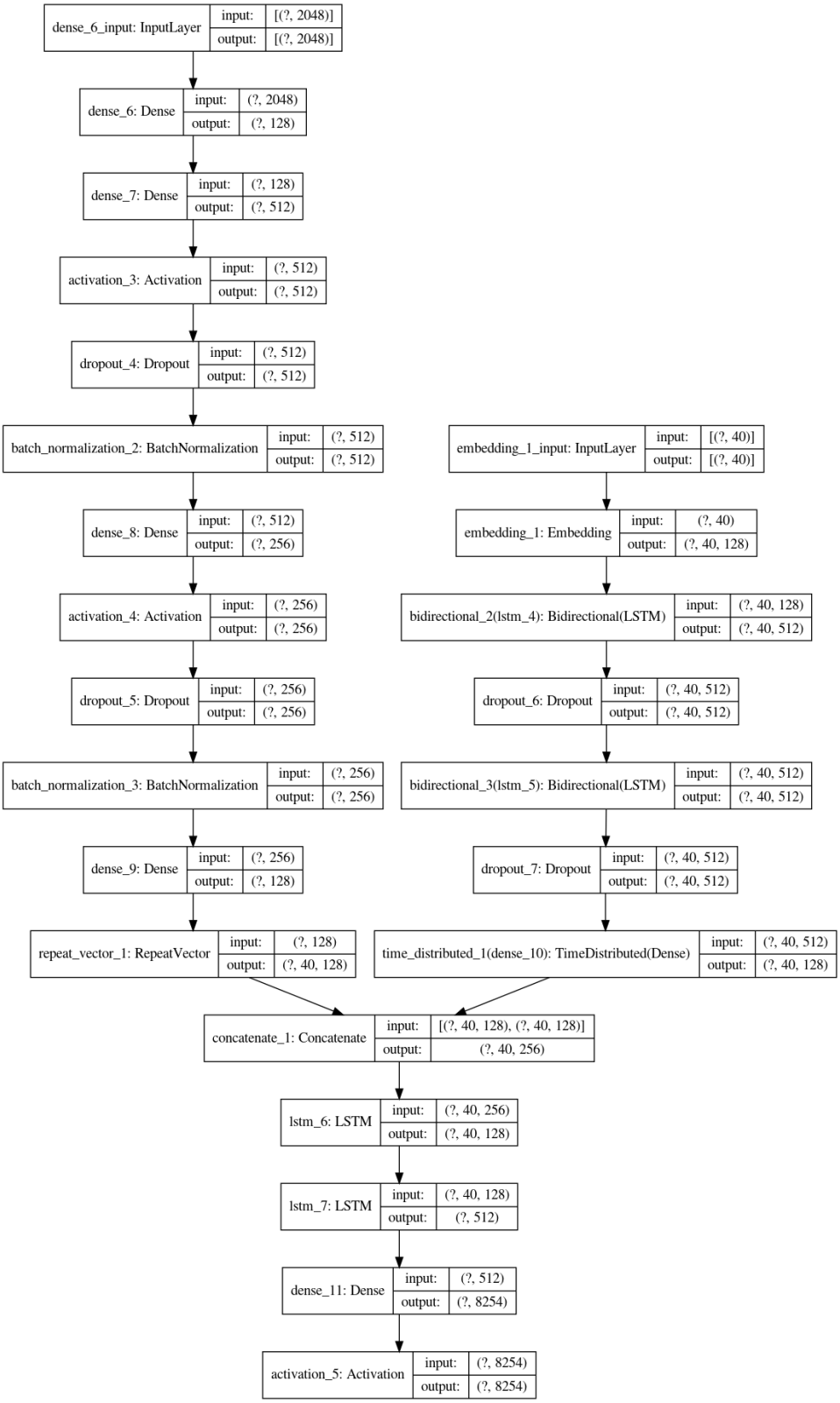


Figure - 7 : Complete Model architecture

CHAPTER - 6

Implementation:

Some major steps to implementation of the image caption generator involves:

Loading Libraries

The first step is to import the necessary libraries and modules. This includes libraries for data manipulation, image processing, neural network construction, and text processing.

```
# Load libraries
import matplotlib.pyplot as plt
import pandas as pd
import pickle
import numpy as np
import os
from keras.applications.resnet50 import ResNet50
from keras.optimizers import Adam
from keras.layers import Dense, Flatten,Input, Convolution2D,
Dropout, LSTM, TimeDistributed, Embedding, Bidirectional, Activation,
RepeatVector,Concatenate, BatchNormalization
from keras.models import Sequential, Model
from keras.utils import np_utils, plot_model
import random
from keras.preprocessing import image, sequence
import matplotlib.pyplot as plt
from IPython.display import Image, display
```

Loading Image Data: Load the image files from the dataset.
Loading Captions: Load and preprocess the text captions associated with the images.

```
# Load data
images_dir =
os.listdir("../input/flickr8k/145129_343604_upload_Flickr_Data/Flickr
_Data")

images_path =
'../input/flickr8k/145129_343604_upload_Flickr_Data/Flickr_Data/Image
s/'
captions_path='../input/flickr8k/145129_343604_upload_Flickr_Data/Fli
ckr_Data/Flickr_TextData/Flickr8k.token.txt'
train_path='../input/flickr8k/145129_343604_upload_Flickr_Data/Flickr
_Data/Flickr_TextData/Flickr_8k.trainImages.txt'
```

```
val_path='../input/flickr8k/145129_343604_upload_Flickr_Data/Flickr_Data/Flickr_TextData/Flickr_8k.devImages.txt'
test_path='../input/flickr8k/145129_343604_upload_Flickr_Data/Flickr_Data/Flickr_TextData/Flickr_8k.testImages.txt'
captions = open(captions_path, 'r').read().split("\n")
x_train = open(train_path, 'r').read().split("\n")
x_val = open(val_path, 'r').read().split("\n")
x_test = open(test_path, 'r').read().split("\n")
```

Preprocessing:

Image preprocessing follows,

Loading Images:
Images from the Flickr8k dataset are loaded from the directory. The images are in various formats and sizes, so they need to be standardized.
Resizing Images:
Each image is resized to a fixed dimension of 224x224 pixels, which is the required input size for the ResNet50 model. This ensures uniformity across all input images.
Normalization:
The pixel values of the images are normalized to the range [0, 1]. Normalization helps in speeding up the training process and achieving better convergence.

```
# Helper function to process images
def preprocessing(img_path):
    im = image.load_img(img_path, target_size=(224,224,3))
    im = image.img_to_array(im)
    im = np.expand_dims(im, axis=0)   # Add batch dimension
    return im
```

Feature Extraction using ResNet50:

The preprocessed images are fed into a pre-trained ResNet50 model (without the final classification layer). This model has been trained on the ImageNet dataset and is used to extract meaningful features from the images.
The ResNet50 model outputs a 2048-dimensional feature vector for each image, which captures high-level representations useful for generating captions.

```
# Loading 50 layer Residual Network Model
from IPython.core.display import display, HTML
model =
```

```
ResNet50(include_top=False,weights='imagenet',input_shape=(224,224,3),pooli
ng='avg')
```

Storing Image Features:
The extracted features are stored in a dictionary, with image filenames as keys and the corresponding feature vectors as values. This allows efficient retrieval of features during training and caption generation.

```
train_data = {}
ctr=0
for ix in x_train:
    if ix == "":
        continue
    if ctr >= 3000:
        break
    ctr+=1
    if ctr%1000==0:
        print(ctr)
    path = images_path + ix
    img = preprocessing(path)
    pred = model.predict(img).reshape(2048)
    train_data[ix] = pred
```

Language preprocessing follows,

Loading Captions:
Captions associated with each image are loaded from the dataset. Each image typically has multiple captions that describe its content.
These captions are stored in a dictionary where the keys are image filenames and the values are lists of captions.

Tokenization:
The captions are tokenized using the Keras Tokenizer. This step involves converting words into unique integer indices, creating a vocabulary of all the words present in the captions.

```
# Loading captions as values and images as key in dictionary
tokens = {}
for ix in range(len(captions)-1):
    temp = captions[ix].split("#")
    if temp[0] in tokens:
```

```
        tokens[temp[0]].append(temp[1][2:])
    else:
        tokens[temp[0]] = [temp[1][2:]]
```

Padding Sequences:
Calculate maximum length and then Captions are padded to a fixed length to ensure uniform input size for the model. Shorter captions are padded with zeros, while longer ones are truncated.
Creating Sequences for Training:
For each caption, input-output pairs are created where the input is the sequence of words up to a certain point, and the output is the next word in the sequence. This helps the model learn to predict the next word in a caption based on the preceding words.

```
# calculates the maximum length of captions in terms of the number of words
max_len = 0

for i in sentences:
    i = i.split()
    if len(i) > max_len:
        max_len = len(i)
print(max_len)
padded_sequences, subsequent_words = [], []
for ix in range(ds.shape[0]):
    partial_seqs = []
    next_words = []
    text = ds[ix, 1].split()
    text = [word_2_indices[i] for i in text]
    for i in range(1, len(text)):
        partial_seqs.append(text[:i])
        next_words.append(text[i])
    padded_partial_seqs = sequence.pad_sequences(partial_seqs, max_len,
padding='post')
    next_words_1hot = np.zeros([len(next_words), vocab_size],
dtype=np.bool)
    #Vectorization
    for i,next_word in enumerate(next_words):
        next_words_1hot[i, next_word] = 1

    padded_sequences.append(padded_partial_seqs)
    subsequent_words.append(next_words_1hot)

padded_sequences = np.asarray(padded_sequences)
subsequent_words = np.asarray(subsequent_words)
print(padded_sequences.shape)
print(subsequent_words.shape)
```

Image Model define:

```
image_model = Sequential()
image_model.add(Dense(embedding_size, input_shape=(2048,), activation='relu'))
# Add Dense layer with ReLU activation function
image_model.add(Dense(512, input_shape=(2048,)))
image_model.add(Activation('relu'))
# Add Dropout layer to prevent overfitting
image_model.add(Dropout(0.5))
# Add BatchNormalization layer to stabilize and accelerate training
image_model.add(BatchNormalization())
# Add another Dense layer with ReLU activation function
image_model.add(Dense(256))
image_model.add(Activation('relu'))
# Add Dropout layer
image_model.add(Dropout(0.5))
# Add BatchNormalization layer
image_model.add(BatchNormalization())
# Add the final Dense layer
image_model.add(Dense(embedding_size))
image_model.add(RepeatVector(max_len))
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	262272
dense_1 (Dense)	(None, 512)	66048
activation (Activation)	(None, 512)	0
dropout (Dropout)	(None, 512)	0
batch_normalization (BatchNo	(None, 512)	2048
dense_2 (Dense)	(None, 256)	131328
activation_1 (Activation)	(None, 256)	0
dropout_1 (Dropout)	(None, 256)	0
batch_normalization_1 (Batch	(None, 256)	1024
dense_3 (Dense)	(None, 128)	32896
repeat_vector (RepeatVector)	(None, 40, 128)	0

Total params: 495,616
Trainable params: 494,080
Non-trainable params: 1,536

Language model:

```
language_model = Sequential()
language_model.add(Embedding(input_dim=vocab_size,
output_dim=embedding_size, input_length=max_len))
# Add Bidirectional LSTM layer with 256 units and return sequences
language_model.add(Bidirectional(LSTM(256, return_sequences=True)))
# Add Dropout layer to prevent overfitting
language_model.add(Dropout(0.5))
# Add another Bidirectional LSTM layer
language_model.add(Bidirectional(LSTM(256, return_sequences=True)))
# Add Dropout layer
language_model.add(Dropout(0.5))
# Add TimeDistributed Dense layer
language_model.add(TimeDistributed(Dense(embedding_size)))
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 40, 128)	1056512
bidirectional (Bidirectional	(None, 40, 512)	788480
dropout_2 (Dropout)	(None, 40, 512)	0
bidirectional_1 (Bidirection	(None, 40, 512)	1574912
dropout_3 (Dropout)	(None, 40, 512)	0
time_distributed (TimeDistri	(None, 40, 128)	65664

Total params: 3,485,568
Trainable params: 3,485,568
Non-trainable params: 0

Concatenate model:

```
conca = Concatenate()([image_model.output, language_model.output])
x = LSTM(128, return_sequences=True)(conca)
x = LSTM(512, return_sequences=False)(x)
x = Dense(vocab_size)(x)
out = Activation('softmax')(x)
model = Model(inputs=[image_model.input, language_model.input],
outputs = out)
model.compile(loss='categorical_crossentropy', optimizer='RMSprop',
metrics=['accuracy'])
model.summary()
```

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
dense_input (InputLayer)	[(None, 2048)]	0	
dense (Dense)	(None, 128)	262272	dense_input[0][0]
dense_1 (Dense)	(None, 512)	66048	dense[0][0]
activation (Activation)	(None, 512)	0	dense_1[0][0]
dropout (Dropout)	(None, 512)	0	activation[0][0]
batch_normalization (BatchNorma	(None, 512)	2048	dropout[0][0]
embedding_input (InputLayer)	[(None, 40)]	0	
dense_2 (Dense)	(None, 256)	131328	batch_normalization[0][0]
embedding (Embedding)	(None, 40, 128)	1056512	embedding_input[0][0]
activation_1 (Activation)	(None, 256)	0	dense_2[0][0]
bidirectional (Bidirectional)	(None, 40, 512)	788480	embedding[0][0]
dropout_1 (Dropout)	(None, 256)	0	activation_1[0][0]
dropout_2 (Dropout)	(None, 40, 512)	0	bidirectional[0][0]
batch_normalization_1 (BatchNor	(None, 256)	1024	dropout_1[0][0]
bidirectional_1 (Bidirectional)	(None, 40, 512)	1574912	dropout_2[0][0]
dense_3 (Dense)	(None, 128)	32896	batch_normalization_1[0][0]
dropout_3 (Dropout)	(None, 40, 512)	0	bidirectional_1[0][0]
repeat_vector (RepeatVector)	(None, 40, 128)	0	dense_3[0][0]
time_distributed (TimeDistribut	(None, 40, 128)	65664	dropout_3[0][0]
concatenate (Concatenate)	(None, 40, 256)	0	repeat_vector[0][0] time_distributed[0][0]
lstm_2 (LSTM)	(None, 40, 128)	197120	concatenate[0][0]
lstm_3 (LSTM)	(None, 512)	1312768	lstm_2[0][0]
dense_5 (Dense)	(None, 8254)	4234302	lstm_3[0][0]
activation_2 (Activation)	(None, 8254)	0	dense_5[0][0]

Total params: 9,725,374
Trainable params: 9,723,838
Non-trainable params: 1,536

Model Training:

```
# hist = model.fit([images, captions], next_words, batch_size=512,
epochs=180)
hist = model.fit([images, captions], next_words, batch_size=512,
epochs=190, validation_split=0.2)
```

Visualize performance:

```
import matplotlib.pyplot as plt
# Plot training & validation accuracy values
plt.figure(figsize=(12, 6))
# Accuracy plot
plt.plot(hist.history['accuracy'], label='Train Accuracy')
plt.plot(hist.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='upper left')
# Plot training & validation loss values
plt.figure(figsize=(12, 6))
# Loss plot
plt.plot(hist.history['loss'], label='Train Loss')
plt.plot(hist.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```

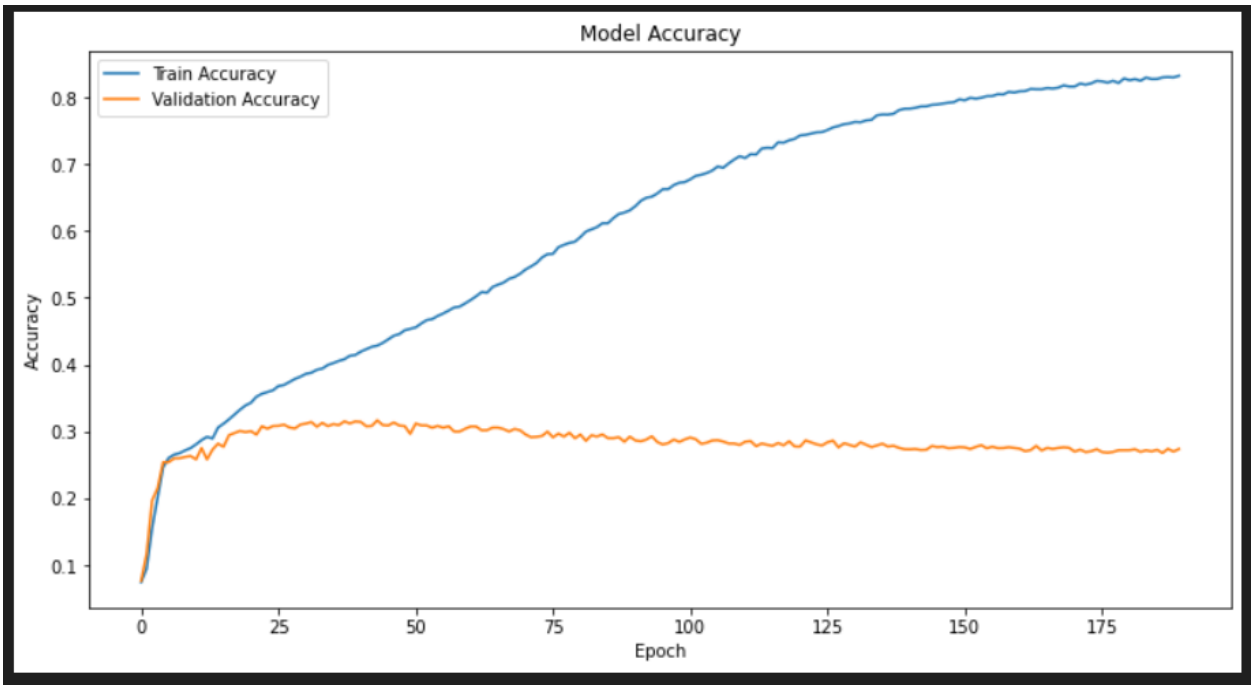


Figure - 8 : Model Accuracy Graph

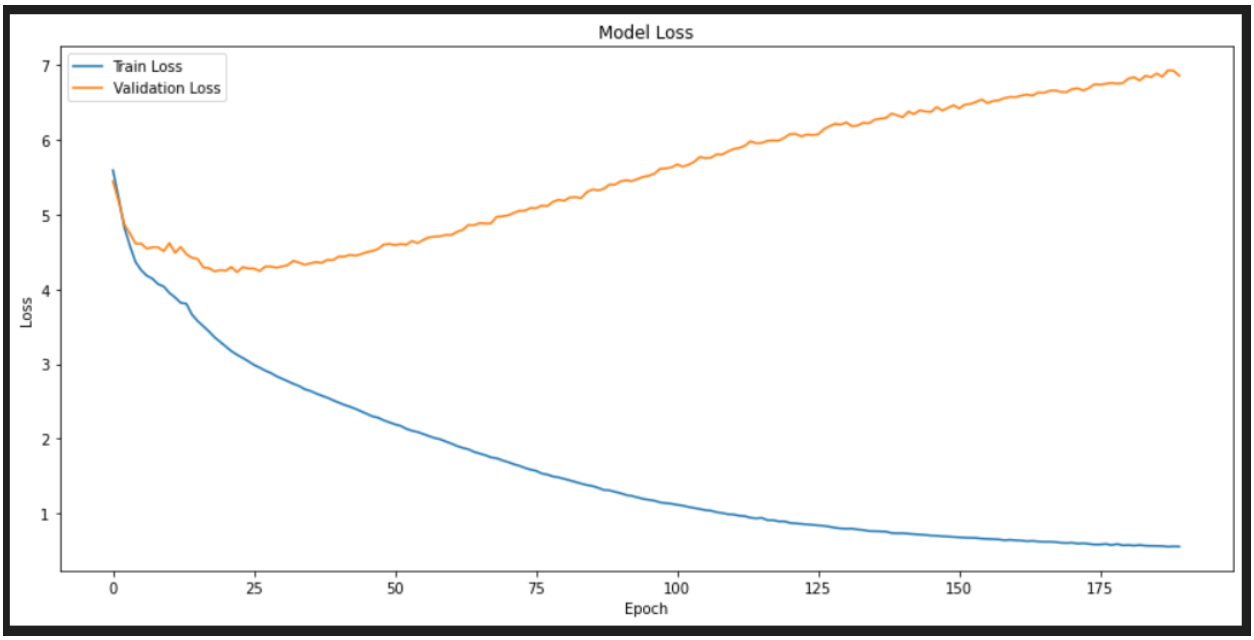


Figure - 9 : Model Loss Graph

Generate caption :

Generate captions for new images using the trained model.

```
def predict_captions(image):
    start_word = ["<start>"]
    while True:
        par_caps = [word_2_indices[i] for i in start_word]
        par_caps = sequence.pad_sequences([par_caps], maxlen=max_len,
padding='post')
        preds = model.predict([np.array([image]), np.array(par_caps)])
        word_pred = indices_2_word[np.argmax(preds[0])]
        start_word.append(word_pred)
        if word_pred == "<end>" or len(start_word) > max_len:
            break

    return ' '.join(start_word[1:-1])

Argmax_Search = predict_captions(test_img)

img="/kaggle/input/flickr8k/145129_343604_upload_Flickr_Data/Flickr_Data/Im
ages/1075716537_62105738b4.jpg"
test_img = get_encoding(resnet, img)
Argmax_Search = predict_captions(test_img)

z = Image(filename=img)
display(z)
print(Argmax_Search)
```

CHAPTER - 7

Result:

Model Training and Accuracy Progress:

The model was trained for 180 epochs, and the following summarizes the loss and accuracy metrics at each epoch:

Loss and Accuracy Analysis

Initial Phases (Epochs 1-10):		
Loss	**Accuracy**	**Observation**
Starting from 5.3723 and dropping to 3.4073	Improved significantly from 0.0987 to 0.3526	Rapid improvement in both loss and accuracy indicates that the model is learning effectively from the data.

Table - 1: Initial Phase result

Mid Phases (Epochs 11-50)		
Loss	**Accuracy**	**Observation**
Continued to decrease from 3.3197 to 2.0979.	Improved from 0.3604 to 0.5043.	The learning rate appears appropriate as the model continues to learn effectively without much significant fluctuations in loss.

Table - 2: Mid Phase result

Later Phases (Epochs 51-100):		
Loss	**Accuracy**	**Observation**
Dropped from 2.0758 to 1.2369	Increased from 0.5083 to 0.6671	Consistent decrease in loss and steady increase in accuracy shows the model is still learning, though at a slower pace, which is typical as it approaches convergence.

Table - 3: Later Phase result

Final Phases (Epochs 101-180)		
Loss	**Accuracy**	**Observation**
Decreased from 1.2297 to 0.7305	Improved from 0.6679 to 0.8505.	The model shows signs of nearing convergence with smaller incremental improvements in both metrics.

Table - 1: Final Phase result

Review:

The model training over 170 epochs demonstrated a significant improvement in both loss and accuracy metrics. Initially, the model exhibited high loss values and low accuracy, with an accuracy of 0.0987 and a loss of 5.3723 at epoch 1. As training progressed, the model's performance improved steadily. By epoch 85, the accuracy reached 0.6264 with a corresponding

loss of 1.4205. This upward trend continued until the final epoch, showcasing the effectiveness of the training process in enhancing the model's capability to generalize from the training data.

- Steady Improvement: Both loss and accuracy improve consistently, indicating a well-tuned model and effective learning rate.
- Convergence Trend: Towards the final epochs, the rate of improvement slows, suggesting the model is converging.

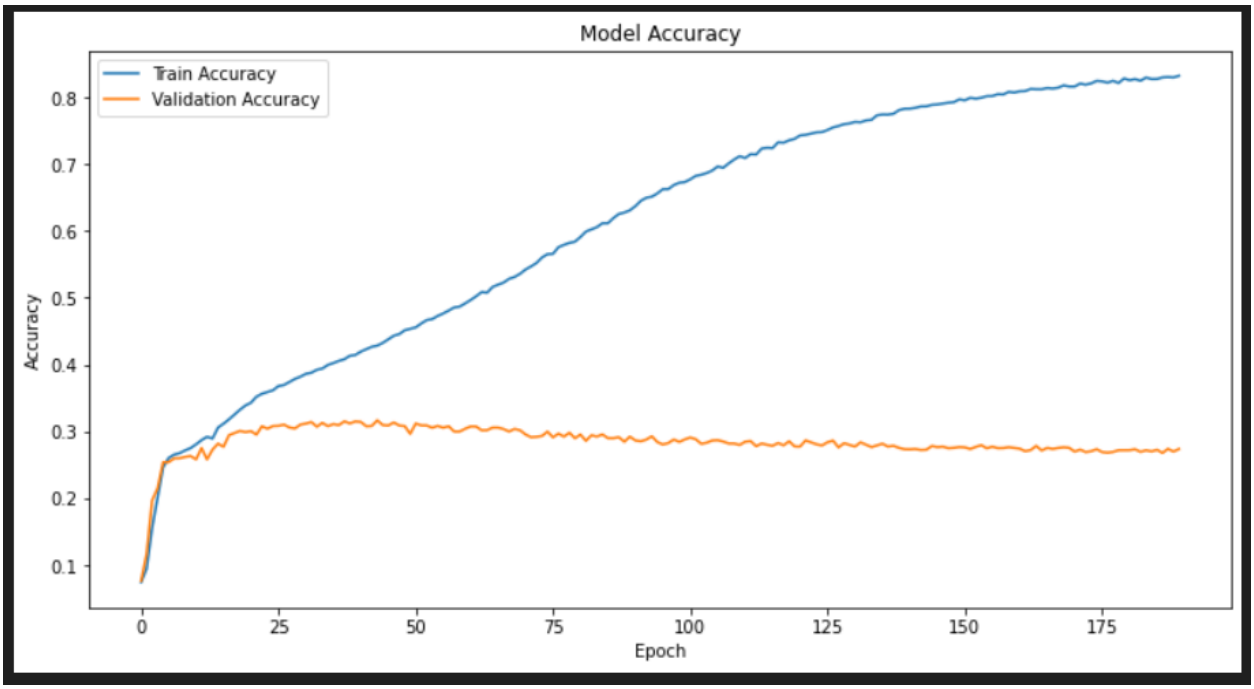


Figure - 8: Model Accuracy Graph

The consistency in the reduction of loss and the increase in accuracy indicates that the model successfully learned the underlying patterns in the data. The final results suggest that the model is well-trained, achieving a high level of accuracy. This progress reflects the robustness of the chosen architecture and the efficacy of the training regimen. Future work could explore further fine-tuning and validation to ensure the model's reliability and performance on unseen data.

ResNet-50 vs Modified ResNet-50 with Custom Image Model

Aspect	**Standard ResNet-50**	**Modified ResNet-50 with Custom Image Model**
Primary Function	Image Classification	Feature Extraction and Advanced Custom Tasks
Top Layer Configuration	Fully connected layer followed by Softmax Layer	Top layer removed; custom layers added
Output	Class Probabilities	Task-specific outputs (e.g., descriptive captions)
Flexibility and Customization	Original design is limited to predefined classification tasks and not easily adaptable for tasks that require outputs other than class probabilities	Highly adaptable for various applications. Offers significant flexibility, allowing customization of the additional layers to fit various complex tasks. This makes it a powerful tool for research and applications in domains where standard classification is insufficient.
Feature Utilization	Directly for classification	Input for additional processing (e.g., sequence generation)
Architecture and Design	Employs a fixed architecture with a softmax layer for classification, making it an excellent tool for tasks where predefined class labels are sufficient.	The softmax layer is removed, and the feature maps are fed into a custom model designed for specific tasks such as image captioning. This setup leverages ResNet-50's robust feature extraction while allowing additional processing for more complex outputs.

Comparison of Model Architectures

Aspects	Show and Tell Model	Deep Visual-Semantic Alignments Model	Proposed Model
Feature Extraction	Uses GoogLeNet (Inception v1) to extract high-level image features.	Utilizes VGG-16 for extracting image features.	Adopts ResNet50, removing the final classification layer to use it as a feature extractor.
Image Model	Image features are fed into an LSTM network to generate captions.	Extracted features are aligned with words using a multimodal embedding space combining CNN and RNN.	ResNet50's 2048-dimensional feature vector is transformed through dense layers before integration with the language model.
Language Model	Employs a single-layer LSTM for decoding image features into words.	Uses a bidirectional RNN for generating a multimodal embedding that aligns visual features with words.	Implements an LSTM-based language model processing preprocessed caption sequences, concatenated with image feature vectors.
Image Representation	Represents the image as a fixed-length vector summarizing content.	Represents the image through region-based features for detailed alignment with sentence fragments.	Uses a 2048-dimensional feature vector transformed through dense layers with ReLU activations and dropout.
Training Objective	Maximizes the likelihood of the target sentence given the image.	Maximizes alignment score between image regions and sentence fragments.	Integrates visual and textual information, optimizing a loss function suited for sequence prediction tasks.
Dataset Used	Evaluated primarily on the MSCOCO dataset.	Evaluated on Flickr8K, Flickr30K, and MSCOCO datasets.	Evaluated on Flickr8K
Key Characteristics	Simplicity and efficiency ideal for quick caption generation.	Detailed alignment approach provides accurate and context-aware captions.	Combines ResNet50 for feature extraction with dense layers and LSTM for descriptive and relevant captions.

Comparison of Model Performance

Aspects	**Show and Tell Model**	**Deep Visual-Semantic Alignments Model**	**Proposed Model**
Dataset Used	MS COCO	MS COCO, Flickr30K	Flickr8k
Accuracy	BLEU-4: 27.7%	BLEU-4: 25.7%	85%
Training Epochs	Not explicitly mentioned	Not explicitly mentioned	180 epochs
Loss Function	Cross-entropy loss	Bidirectional ranking loss	Cross-entropy loss
Final Loss Value	Not Provided	Not Provided	Decreased from 5.3723 to 0.7305
Model Type	CNN (GoogleNet) + LSTM	CNN + RNN (BRNN)	CNN (ResNet50) + Bi-directional LSTM
Architecture	Encoder-Decoder	Multi-modal Embedding Model	Combined CNN and Bi-directional LSTM

Captioned Image Results

GOOD Captions:

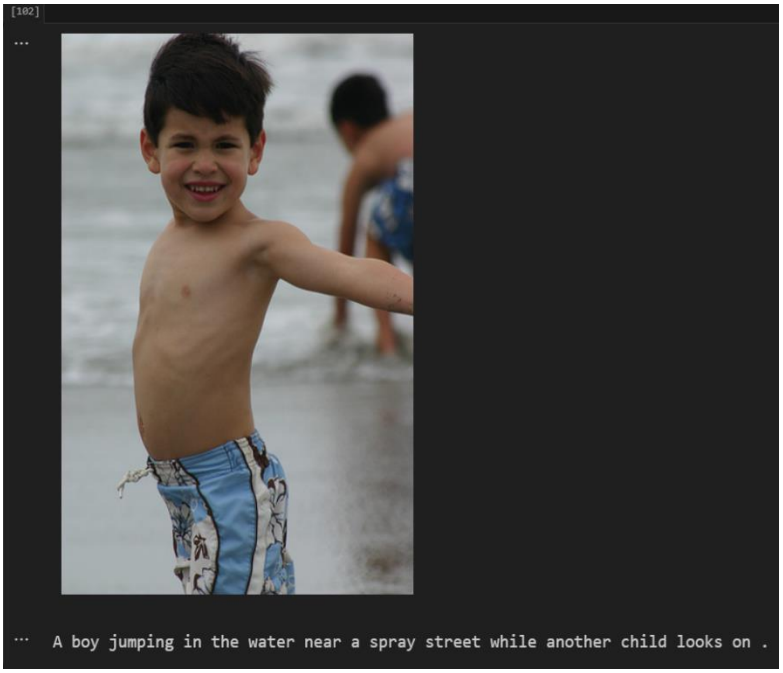


Figure - 10 : Result image 1
Caption: A boy jumping in the water near a spray street while another child looks on .

Figure - 11 : Result image 2

Caption: A fluffy dog is jumping and catching a toy

Figure - 12 : Result image 3
Caption: A man and a woman stand close together

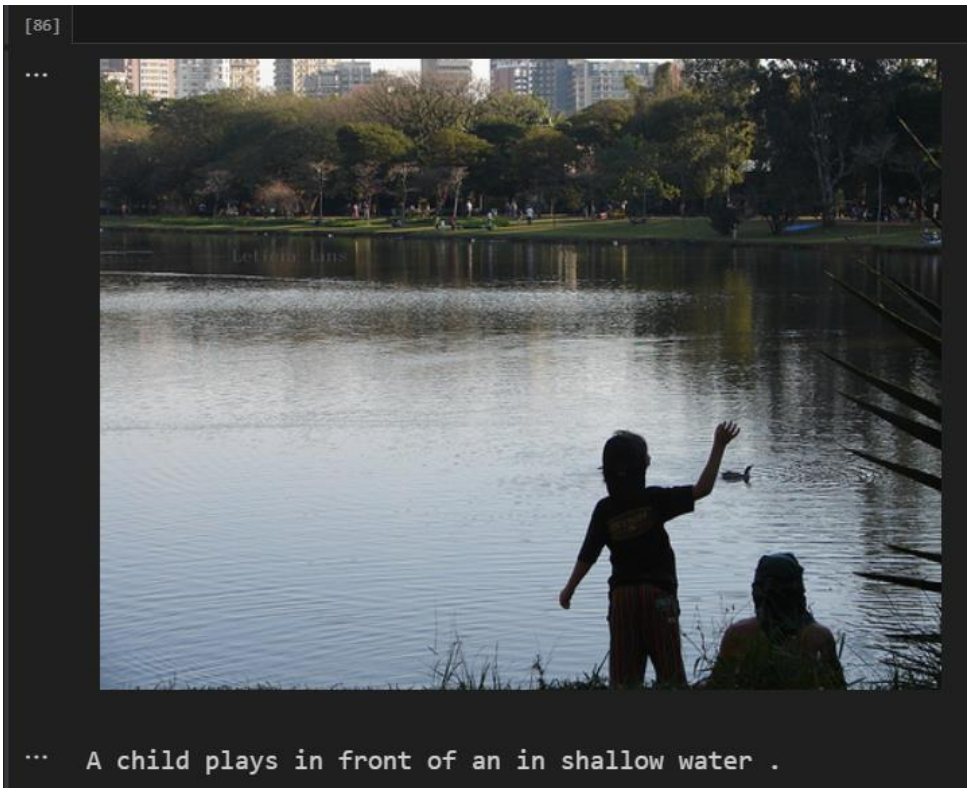


Figure - 13 : Result image 4
Caption: A child plays in front of an in shallow water .

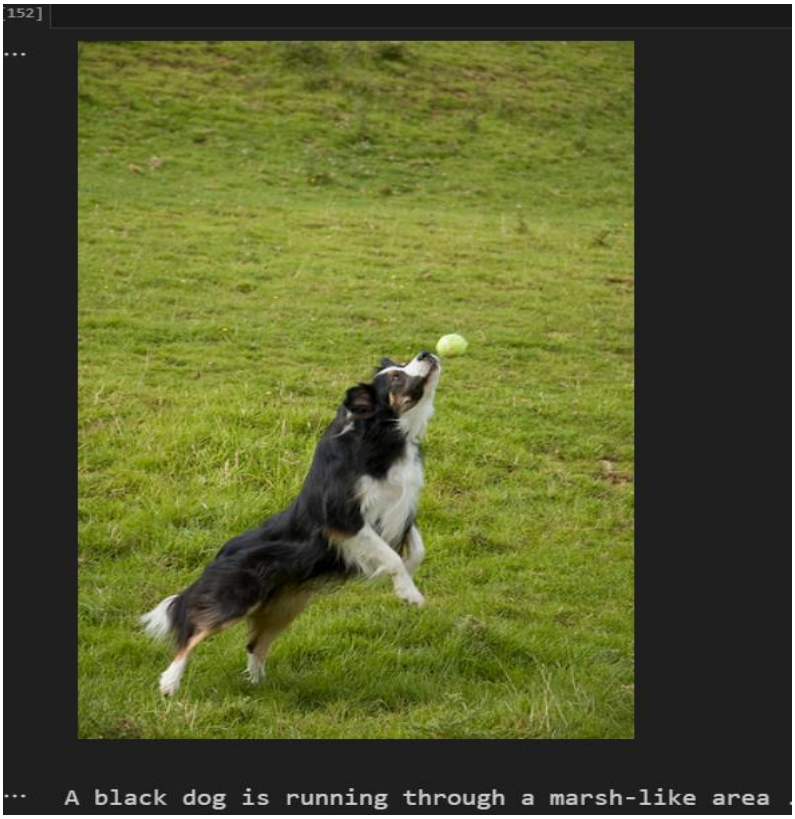


Figure - 14 : Result image 5

A black dog is running through a marsh-like area.

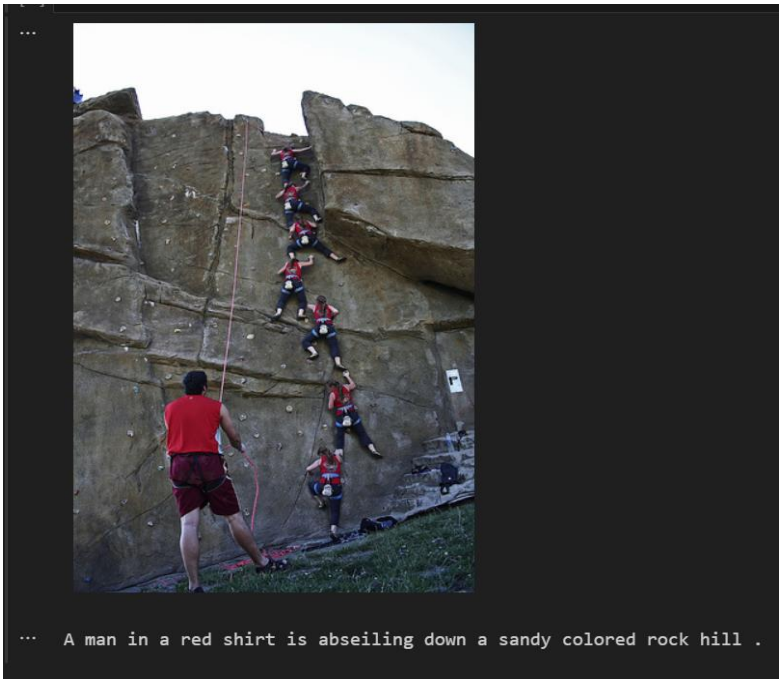


Figure - 15 : Result image 6

A man in a red shirt is abseiling down a sandy colored rock hill.

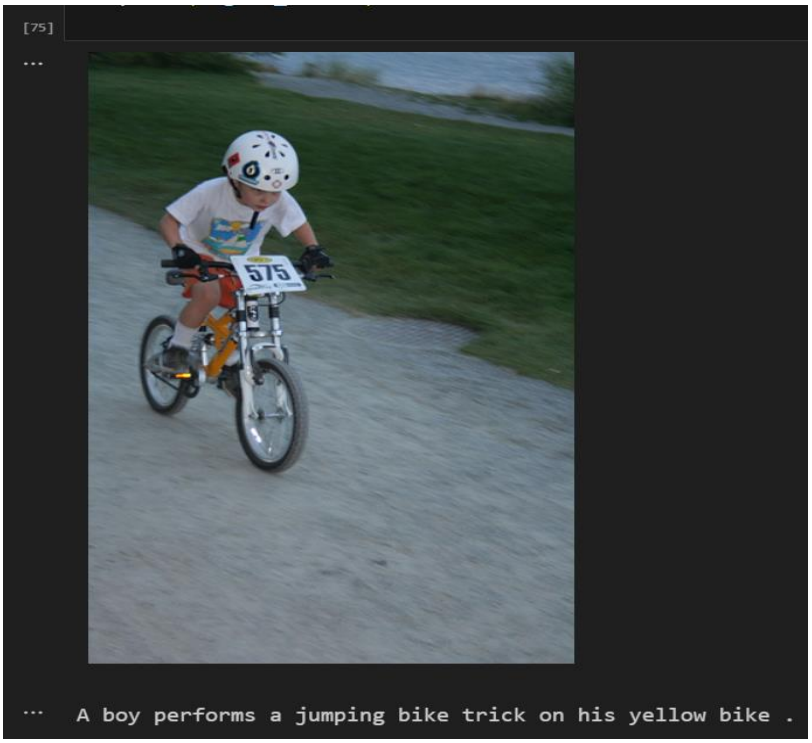


Figure - 16 : Result image 7

A boy performs a jumping bike trick on his yellow bike .

Not So accurate captions:

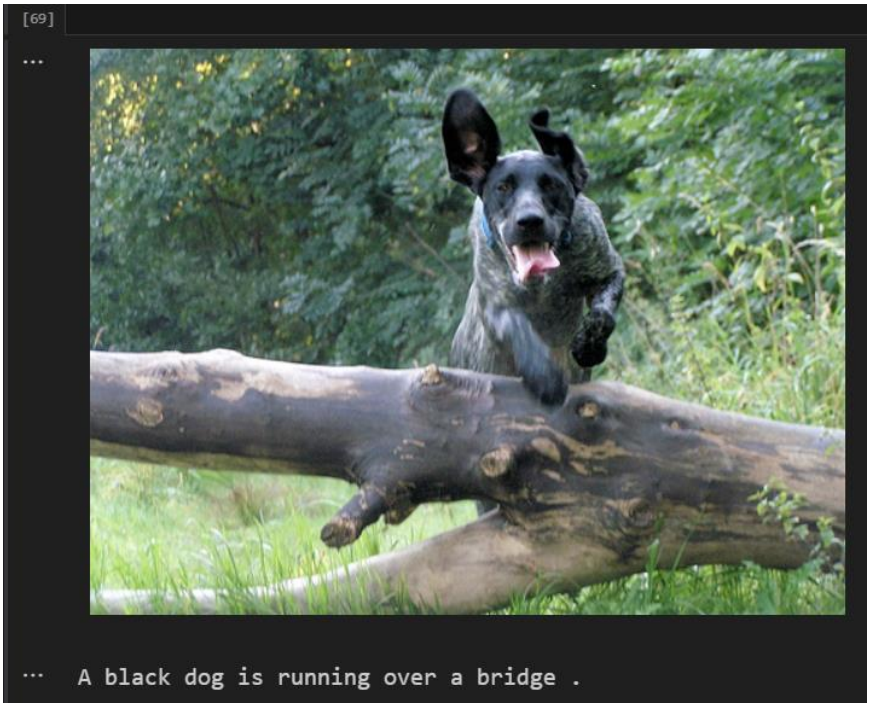


Figure - 17 : Result image 8

A black dog is running over a bridge .

Figure - 18 : Result image 9

A women with a backpack is looking at a fishing pole in the snow.

Figure - 19 : Result image 10

Two dogs , one brown and the other black , play in the sand

Figure - 20 : Result image 11

A group of people wearing life jackets riding a raft through rough waters .

Figure - 21 : Result image 12

A dog is running through a Frisbee

CHAPTER - 8

Limitations:

- Fixed-Length Encoding:

The image model uses a fixed-length encoding for the entire image, which may not capture detailed and localized features as effectively as attention mechanisms.

- Sequential Generation:

The language model generates captions sequentially without focusing on specific parts of the image, leading to less coherent and contextually relevant captions.

- Lack of Dynamic Focus:

Attention models dynamically focus on different parts of the image during each step of the caption generation, providing more accurate and descriptive captions.

- Overfitting:

The complex architecture and lack of attention mechanism is leading to overfitting, especially with smaller datasets, resulting in poor generalization to new images.

- Computational Efficiency:

Although attention models can be computationally intensive, their ability to focus on relevant parts of the image can make them more efficient in generating relevant captions, potentially reducing the need for very deep architectures.

Future Scopes:

The image caption generator model described has significant future scope in various applications:

1. Accessibility:

The model can be integrated into screen readers and other assistive technologies to provide visually impaired users with descriptions of images they encounter on websites, social media, and digital documents.This enhances web accessibility, ensuring that visually impaired individuals can understand and engage with visual content.

2. Content Management:

Automating the generation of descriptive metadata for images stored in large databases such as those used by museums, news agencies, and stock photo websites. Improved metadata enhances searchability and organization, making it easier to retrieve specific images based on their content.

3. Social Media:

Platforms like Instagram, Facebook, and Twitter can use the model to auto-generate captions for user-uploaded images. This feature can help users quickly generate engaging content, improve accessibility, and increase interaction by providing relevant hashtags and descriptions.

4. E-commerce:

Online retail platforms can leverage the model to automatically generate detailed product descriptions from images. Enhanced product descriptions can improve user experience, facilitate product discovery through better search engine optimization (SEO), and ultimately drive sales.

5. Surveillance:

Real-time image captioning for surveillance footage can help security personnel by providing immediate, descriptive summaries of scenes and activities captured by security cameras. This application can enhance monitoring efficiency, support rapid decision-making, and improve incident response by highlighting potential security threats.

CHAPTER - 9

Conclusion:

The image caption generator model showcases a promising avenue for various impactful applications. It can revolutionize accessibility tools, helping visually impaired individuals comprehend visual content through descriptive captions. In content management, it automates metadata generation, improving image retrieval and organization in large databases. Social media platforms can leverage it for auto-generating engaging captions, boosting user interaction. E-commerce websites benefit from enhanced product descriptions, aiding product discovery and sales. Additionally, in surveillance, it provides real-time scene descriptions, aiding security personnel in monitoring and quick decision-making. Future enhancements, particularly through attention mechanisms, will further refine its accuracy and utility, cementing its relevance across these domains.

References:

1. Wang, Haoran, Yue Zhang, and Xiaosheng Yu. "An overview of image caption generation methods." Computational intelligence and neuroscience 2020.
2. Bai, Shuang, and Shan An. "A survey on automatic image caption generation." *Neurocomputing* 311 (2018).
3. Xu, Kelvin, et al. "Show, attend and tell: Neural image caption generation with visual attention." *International conference on machine learning*. PMLR, 2015.
4. Karpathy, Andrej, and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." 2015
5. Waswani, A., et al. "Attention is all you need." *NIPS*. 2017.
6. Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2015). Show and tell: A neural image caption generator.
7. Chen, Xinlei, and C. Lawrence Zitnick. "Mind's eye: A recurrent visual representation for image caption generation." 2015.
8. Chen, Xinlei, and C. Lawrence Zitnick. "Learning a recurrent visual representation for image caption generation." (2014).
9. Amritkar, Chetan, and Vaishali Jabade. "Image caption generation using deep learning technique.", IEEE, 2018.
10. Amirian, Soheyla, et al. "A short review on image caption generation with deep learning." *(IPCV)*, Computer Engineering and Applied Computing (WorldComp), 2019.