***A Major-Project Report***

*On*

ENSEMBLE MODEL: CNN-LSTM FOR IMAGE CAPTION GENERATION

*Submitted in partial fulfilment for the Degree of B. Tech.*

*In*

***Artificial Intelligence***

*By*

*ALURU SAI HARSHITHA [20911A3505] TEJASWI SATTARSHETTY [20911A3554]*

*Y. KEERTHI CHOWDARY [20911A3559] ZAIB UNNISA NAYEEM [20911A3560]*

***Under the guidance of***

Dr. A. Obulesh

Associate Professor - Department of AI



**DEPARTMENT OF ARTIFICIAL INTELLIGENCE**

**VIDYA JYOTHI INSTITUTE OF TECHNOLOGY**

(An Autonomous Institution)

**[Accredited by NAAC & NBA, Approved by AICTE New Delhi & Permanently Affiliated to JNTUH]**

# 2024 – 2025



**[Accredited by NAAC & NBA, Approved by AICTE New Delhi & Permanently Affiliated to JNTUH]**

**CERTIFICATE**

This is to certify that the project report entitled **“Ensemble Model : CNN-LSTM for Image Caption Generation”** submitted by **Aluru Sai Harshitha (20911A3505) , Tejaswi Sattarshetty (20911A3554), Yalamaddi Keerthi Chowdary (20911A3559) and Zaib Unnisa Nayeem (20911A3560)** to Vidya Jyothi Institute of Technology(An Autonomous Institution), Hyderabad, in partial fulfilment for the award of the degree of **B. Tech. in Artificial Intelligence** a *bonafide* record of project work carried out by us under my supervision. The contents of this report, in full or in parts, have not been submitted to any other Institution or University for the award of any degree.

**Signature of Supervisor Dr. A. Obulesh**

**Associate Professor**

**Department of Artificial Intelligence**

**Signature of Head of the Department Dr. A. Obulesh**

**Associate Professor**

**Department of Artificial Intelligence**

**Signature of External Examiner**

# DECLARATION

We declare that this project report titled **Ensemble Model: CNN-LSTM for Image Caption Generation** submitted in partial fulfilment of the degree of B. Tech in Artificial Intelligence is a record of original work carried out by us under the supervision of **Dr. A. Obulesh**, and has not formed the basis for the award of any other degree or diploma, in this or any other Institution or University. In keeping with the ethical practice of reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

Aluru Sai Harshitha (20911A3505) Tejaswi Sattarshetty (20911A3554)

Y. Keerthi Chowdary (20911A3559)

**Date:** Zaib Unnisa Nayeem (20911A3560)

# ACKNOWLEDGEMENTS

We extend our heartfelt thanks to **Dr. A. Obulesh,** Associate Professor, for his invaluable mentorship and guidance throughout the project. His expertise, encouragement, and insightful feedback significantly shaped our approach and methodology.

We wish to express our heartful thanks to **Dr. A. Obulesh, HOD**, for guiding and supporting us throughout the course.

We sincerely extend our heartfelt gratitude to Principal **Dr. A. Srujana**, Dean Accreditations & Rankings **Dr. A. Padmaja**, and Dean Academic Coordinator **Mrs. G. Sreelatha** for their invaluable support and unwavering encouragement throughout the course.

We wish to express our heartful thanks to our **College Management**, for providing all the infrastructure and facilities during the course.

We would like to express our sincere gratitude to our **Parents and Friends,** for their unwavering support throughout the completion of this major project.

Aluru Sai Harshitha (20911A3505) Tejaswi Sattarshetty (20911A3554)

Y. Keerthi Chowdary (20911A3559) Zaib Unnisa Nayeem (20911A3560)

# ABSTRACT

This project focuses on implementing an Image Captioning system, combining Long Short- Term Memory networks (LSTMs) and Convolutional Neural Networks (CNNs). The goal is to automatically describe natural scenes, bridging computer vision and natural language processing. While current methods predominantly use CNNs to extract visual features, this work explores the fusion of text within images for more nuanced captioning. The proposed model integrates deep CNNs and LSTMs to enhance image captioning accuracy by incorporating textual information present in images. This fusion of text and visual features goes beyond existing approaches, offering a more detailed understanding of scenes. The application domains of this model are diverse, including medical imaging for detailed image descriptions aiding healthcare professionals, improved visual search in e-commerce, contributions to tourism, environmental monitoring, cultural heritage preservation, news reporting, and virtual reality. The versatility of the architecture extends to autonomous systems, contributing to the development of self-navigating vehicles through a deeper understanding of the visual environment.

# TABLE OF CONTENTS

DESCRIPTION PAGE NO. CERTIFICATE ii

[DECLARATION iii](#_TOC_250022)

[ACKNOWLEDGEMENTS iv](#_TOC_250021)

[ABSTRACT v](#_TOC_250020)

[LIST OF FIGURES vii](#_TOC_250019)

[LIST OF TABLES viii](#_TOC_250018)

[CHAPTER 1 INTRODUCTION 1](#_TOC_250017)

* 1. [Introduction 1](#_TOC_250016)
  2. Problem description 2
  3. [Motivation 2](#_TOC_250015)
  4. [Objective 3](#_TOC_250014)

[CHAPTER 2 LITERATURE SURVEY 4](#_TOC_250013)

[CHAPTER 3 METHODOLOGY 7](#_TOC_250012)

* 1. [Dataset 7](#_TOC_250011)
  2. [Long Short Term Memory 7](#_TOC_250010)
  3. [Proposed model for image captioning 8](#_TOC_250009)
  4. [Visual Features 10](#_TOC_250008)
  5. [Text Features 11](#_TOC_250007)
  6. [Feature fusion 12](#_TOC_250006)
     1. [LSTM based caption generator 13](#_TOC_250005)

CHAPTER 4 RESULTS AND DISCUSSIONS 16

* 1. [Evaluation metrics 17](#_TOC_250004)
  2. [Implementation details 18](#_TOC_250003)
  3. Comparison with baseline 19
  4. [User Manual 21](#_TOC_250002)

[CHAPTER 5 CONCLUSION 52](#_TOC_250001)

[CHAPTER 6 REFERENCES 53](#_TOC_250000)

# LIST OF FIGURES

**FIGURE NO. TITLE PAGE NO.**

|  |  |  |
| --- | --- | --- |
| 1.1 | Samples of natural scene images with text | 3 |
| 3.1 | Proposed model block diagram | 10 |
| 3.2 | Proposed LSTM graphical representation | 15 |
| 4.1 | Text detection from Flickr8k | 19 |
| 4.2 | Generated captions for images without text | 20 |
| 4.3 | Generated captions for images with text | 21 |
| 4.4 | Output 1- Image captions (without text in image) | 20 |
| 4.5 | Output 2- Image captions (with text in image) | 21 |

# LIST OF TABLES

**TABLE NO. TITLE PAGE NO.**

* 1. Literature survey 5
  2. Statistics of dataset 7
  3. Assessment of model with baseline 20

# CHAPTER 1 INTRODUCTION

## Introduction

Effortlessly capturing the essence of a natural scene with a mere glance, humans possess an innate ability to articulate a plethora of details within seconds. Mimicking this capability, image captioning has emerged as a pivotal process, aiming to generate linguistically acceptable and semantically correct descriptions that align with human understanding. However, this task poses significant challenges, necessitating a deep comprehension of image content and human-like sentence generation. The implications of successful caption generation are far-reaching, spanning applications such as information retrieval, human-robot communication, and assistance for visually impaired individuals.

Recent advancements have leveraged deep neural networks for feature extraction in caption generation, predominantly following an encoder-decoder pipeline methodology. While these methods have demonstrated moderate success compared to traditional approaches, they primarily rely on visual features alone. Yet, natural scenes harbor abundant semantic information that remains untapped, presenting an opportunity for fine-grained image captioning. Notably, salient text elements—like roadside signboards, car number plates, building names, and business place identifiers—contribute significantly to the scene's semantics and can enhance caption quality.

In response to this opportunity, our major project, titled "Transformative Image Captioning through LSTM-CNN," endeavours to pioneer a novel approach for fine-grained caption generation. We propose a model that integrates text features extracted from the scene alongside visual features. By fusing these modalities, our model aims to generate captions that resonate more closely with human perception and understanding. As depicted in Figure 1.1, traditional methods may identify the main object as simply a "building" based on visual features alone. However, by incorporating text features such as "Livestrong Sporting Park," "Hotel," and "Parkway Cinema," our model can produce more meaningful captions, akin to human visual processing.

By harnessing the synergies between advanced Long Short-Term Memory networks (LSTMs) and Convolutional Neural Networks (CNNs), our project aims to pave the way for

transformative applications across various domains, revolutionizing how we perceive and interact with visual information.

## Problem Definition

The problem at hand is to develop an effective Image Captioning model using LSTM- CNN architecture to bridge the gap between visual understanding and natural language processing. Currently, image captioning approaches heavily rely on Convolutional Neural Networks (CNNs) for extracting visual features, overlooking potentially valuable textual information present within images. The objective is to address this limitation by creating a model that seamlessly integrates both visual and text features, leveraging the strengths of CNNs and Long Short-Term Memory networks (LSTMs) for enhanced image captioning accuracy.

Through this fusion, the aim is to enable computers to provide accurate and contextually relevant descriptions for images, thereby advancing the capabilities of computer vision systems.

## Motivation

Our motivation stems from the limitations of current image captioning methods, which overlook valuable textual information within images. By integrating LSTM-CNN architecture, we aim to bridge the gap between visual understanding and natural language processing. This fusion promises more accurate and contextually rich captions, unlocking new possibilities in healthcare, e-commerce, tourism, and autonomous systems. Moreover, improved image captioning can enhance accessibility for visually impaired individuals and facilitate more intuitive human-computer interaction, driving advancements in AI systems' capabilities. Figure 1.1 shows the natural scene which shows ‘‘building’’ as the main object if we consider only visual features. However, if we fuse the text features extracted from the scene such as ‘‘Livestrong sporting park,’’ ‘‘Hotel,’’ and ‘‘Parkway Cinema’’ with the visual feature, the generated caption will be more meaningful as mimicked by the human visual system.

## Objective

* + 1. The objective of the proposed work is to generate a semantically meaningful caption to the images.
    2. Integrating textual feature with visual features using Deep CNN and LSTM.



Figure 1.1: Samples of natural scene images with text.

# CHAPTER 2 LITERATURE SURVEY

Image captioning has been a challenging trade in the arena of computer vision and artificial intelligence. Recent advancements have leveraged deep neural networks for feature extraction in caption generation [[1](#_bookmark0)-5], predominantly following an encoder-decoder pipeline methodology. While these methods have demonstrated moderate success compared to traditional approaches, they primarily rely on visual features alone. Flickr8k [6] and Flickr30k

[7] are said to be benchmark datasets for the task of image captioning. Yet, natural scenes harbour abundant semantic information that remains untapped, presenting an opportunity for fine-grained image captioning. Notably, salient text elements—like roadside signboards, car number plates, building names, and business place identifiers—contribute significantly to the scene's semantics and can enhance caption quality [8].

Before incorporating neural network in caption generation, there are two categories of image captioning: first is retrieval based and second is template based. In retrieval-based methods [9–11] for a given scene image, a caption is generated from pre-specified sentence collection. In template-based methods [12–14], image captions are produced on the basis of a syntactically and semantically controlled procedure. Incredible success in the deep learning

[15] field, for automatic image captioning task the ongoing work starts to depend on deep neural networks. Socher et al. [16] proposed an approach for representing the sentences as compositional vectors using dependency tree recursive neural networks to produce a caption for an input image. A deep neural network is utilized as a visual model to extract visual features. Common space is used to be represented as a max-margin objective function for mapping multimodal features. In this common space, the sentence and correct image pairs are selected on basis of an inner product after training the data.

Table 2.1: Literature survey.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.**  **No.** | **Title** | **Methodology** | **Metrics** | **Limitations** |
| 1 | Automatic image caption generation using deep learning  - *Akash Verma1 ·Arun Kumar Yadav1 · Mohit Kumar1 ·Divakar Yadav2* | The proposed  methodology employs an encoder-decoder model using VGG16 Hybrid Places 1365 as an encoder and LSTM as a decoder to generate captions for images | BLEU, METEOR, GLEU | Generating diverse and contextually relevant captions for various images. |
| 2 | Growing random forest on deep convolutional neural networks for scene categorization.  *- Bai S* | The methodology used by these neural machine translation models is encoder–decoder pipeline | BLEU, GLEU | Only visual features are  exploited for caption generation. |
| 3 | Im2text: describing images using 1 million captioned photographs.  -*Ordonez V, Kulkarni G, Berg TL* | For a given scene image, a caption is generated from pre-specified sentence collection. In template- based methods | BLEU | Textual Features are not captured. |
| 4 | ImageNet classification with deep convolutional neural networks.  -*Krizhevsky A, Sutskever I, Hinton GE* | A deep convolutional neural network is used for extracting visual feature from an image and a multimodal recurrent neural network (RNN) is used to generate caption depending on context words and features of an image | BLEU | LSTM are proved to perform better than RNN |

At last, caption generation is performed on similarities index between sentences and representation of images in common space. Karpathy et al. [17] suggested the sentence ranking system. In this system for a given query image, the sentences and image are embedded into a common space. They use region convolution neural network approach [18] for an image and dependency tree relation for the sentence. Feature vectors represent both sentences and images, and the max-margin objective function is designed by the authors to map data into a common space which contains a global ranking term. The common space has fragment similarities computed on the basis of similarities between images and sentences. As a result, at a finer level sentence ranking can be shown.

A multimodal convolutional neural network [19] is proposed by Ma et al. who consider interaction at various levels in order to calculate the similarities between sentences and images. Multimodal convolutional neural network is used which determines the absolute similarity score between a sentence and an image. In this approach, the deep convolutional neural network is applied to do feature extraction on the image, and further extracted features are forwarded to the model of neural language which maps words and image features into the common space and generates the words which depend on the previously produced context words and image feature. Another neural language model was proposed by Kiros et al. [20] for generating captions of an image which is conditioned on the input image. In their approach, multimodal methods are replaced by the log-bilinear language model.

Conditional probability is used for generating a word depending on the previous-step- produced words for a language model in natural language processing. To produce original captions for images, recurrent neural network (RNN) sequence model was given by Mao et al. [21]. This model used the approach where the given image and the previous word are conditioned for getting the probability of generating a word. In this model, a deep convolutional neural network (CNN) [22] is used for extracting the visual feature from an image and a multimodal recurrent neural network (RNN) is used to generate caption depending on context words and features of an image. Vinyals et al. [23] proposed similar motivation as that of neural machine translation, to encode images deep convolutional neural network (CNN) as an encoder and the long short-term memory (LSTM) is used to decode image features into the caption.

# CHAPTER 3 METHODOLOGY

## Dataset:

Flickr8k [6] benchmark datasets are used for the evaluation of the proposed approach. In these datasets, there are five annotated sentences for every image. The Flickr8k dataset is extracted from Flickr, and it contains 8000 images. For practical implementation of this dataset, the standard split is used for training-6000 images, validation-1000 images, and testing-1000 images.. Further the benchmark datasets Flickr8k contain images having both visual and textual cues, which are approxi mately 1722, and a remaining number of images are having only visual cues as shown in table below

Table 3.1: Statistics of dataset.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Dataset**  **Name** | **Size** | | | **Total images** | **Visual + Textual cue images** | **Visual only cue images** |
|  | **Train** | **Validation** | **Test** |
| Flickr8k | 6091 | 1000 | 1000 | 8091 | 1722 | 6369 |

## Long short-term memory:

Long Short-Term Memory (LSTM) is a crucial component in various tasks such as machine translation and sequence generation. At the heart of the LSTM model lies the memory cell (denoted as 'c'), which stores essential information at each step of the sequence. This information is dependent on the input received in the previous step.

The functionality of the LSTM model is regulated by its 'gates.' These gates serve to control the flow of information within the memory cell. Each gate determines whether certain information should be retained or discarded based on its relevance. Typically, there are three primary gates employed in LSTM architecture: the input gate, the output gate, and the forget gate.

The forget gate (denoted as 'fg') plays a pivotal role in determining whether the current cell value should be forgotten or retained for further processing. It evaluates the significance of the current information in relation to the overall context.

On the other hand, the input gate (denoted as 'ig') is responsible for assimilating new input data into the memory cell. It decides which parts of the input should be incorporated

into the cell's current state.

Finally, the output gate (denoted as 'og') controls the generation of new cell values based on the processed information. It determines what information should be outputted from the memory cell to the subsequent layers or as the final result of the LSTM model.

In essence, the interplay between these gates enables the LSTM model to effectively capture long-range dependencies in sequential data and make informed decisions at each step of the sequence. This capability makes LSTM networks particularly well-suited for tasks involving sequential data processing, such as natural language processing and time series prediction.

The mathematical formulation of these gates and memory cell is given below in (eq. 3.1 – 3.6) :

igt = σ(Wigxxt + Wigmmt−1) (3.1)

fgt = σ(Wfgxxt + Wfgmmt−1) (3.2)

ogt = σ(Wogxxt + Wogmmt−1) (3.3)

|  |  |  |
| --- | --- | --- |
| cgt = fgt ⊗ cgt−1 + igt ⊗ h(Wcgxxt + Wcgmmt−1) | | (3.4) |
| mt = ogt ⊗ cgt |  | (3.5) |
| Pt+1 = Softmax(mt) |  | (3.6) |

where σ(.) is sigmoid activation function to generate nonlinearity and h(.) is hyperbolic tangent activation function, trained parameters are several W matrices, and product with a gate value is denoted by ⊗; to train the LSTM network, these gates are used which deal with the problem of vanishing and exploding gradients. Softmax is used at the end step of LSTM for computing the probability pt over the words after getting input mt at the previous step.

## Proposed model for image captioning:

In the realm of image captioning, recent advancements have predominantly relied on visual features alone to generate captions, resulting in descriptions that often lack specificity and fail to capture nuanced details of a scene. To address this limitation, a novel model has been proposed, as depicted in Figure 3.1, which integrates both visual image features and textual features for fine-grained image captioning of natural scenes. This approach aims to

produce captions that not only provide generic scene descriptions but also incorporate specific details, enhancing the overall quality and richness of the generated captions.

The proposed model comprises three key modules. Firstly, the model leverages the Visual Geometry Group (VGG) net [24], a deep convolutional neural network (CNN) model, to extract visual features from input images. This initial module enables the model to capture intricate visual information inherent in the images, laying the foundation for more detailed caption generation. Following the extraction of visual features, the second module employs a text extraction model to identify and extract textual features from the images. By incorporating textual features, the model gains the ability to recognize and highlight important textual elements within the scene, enriching the contextual understanding of the images. Finally, the third module employs Long Short-Term Memory (LSTM), a powerful recurrent neural network architecture, to fuse the extracted visual and textual features and generate comprehensive captions for the input images. This fusion mechanism enables the model to synergistically combine both visual and textual cues, resulting in more informative and contextually relevant image captions that encompass both visual and textual aspects of the scene. Overall, by integrating visual and textual features, the proposed model offers a promising approach for enhancing image captioning capabilities, facilitating the generation of more detailed and contextually rich descriptions of natural scenes.

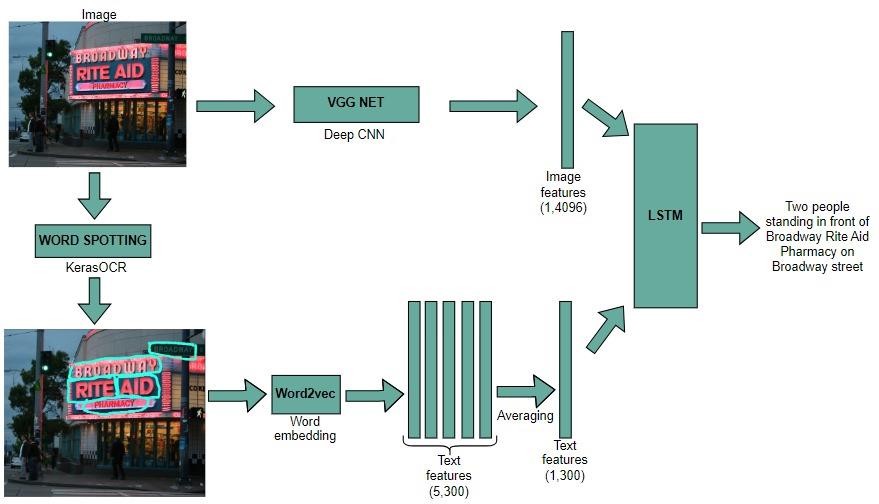


Figure 3.1: Proposed model block diagram.

## Visual Features:

Convolutional Neural Networks (CNNs) have emerged as a cornerstone technique in computer vision, particularly for extracting meaningful image features. In recent years, the ability of CNNs to learn rich visual representations from vast image datasets has revolutionized various computer vision tasks. As highlighted in [25], these learned visual features offer significant utility across a spectrum of applications. By training a CNN model on large-scale image datasets, such as ImageNet, these models can effectively capture intricate visual patterns and semantics present in images. Moreover, the versatility of CNN- based features allows for their seamless integration into diverse tasks, ranging from image classification to object detection and beyond.

An essential aspect of leveraging CNNs in practical applications is the availability of pre-trained models, which offer a compelling advantage over training from scratch. Fine- tuning a pre-trained CNN model allows for the adaptation of learned features to specific tasks or domains, thereby yielding improved performance with reduced computational overhead. In the proposed framework, the widely adopted VGG Net [24] pre-trained model serves as the foundation for generating image features. Trained on millions of images from the ImageNet dataset [26], VGG Net architecture consists of 3x3 convolutional layers stacked in increasing depth, augmented by max-pooling for downsampling. The network culminates in two fully connected layers with 4096 nodes each, followed by a Softmax classifier responsible for classifying images into 1000 classes.

For image captioning tasks, the proposed model harnesses the power of the pre-trained VGG Net by extracting a 4096-dimensional feature vector from the second last layer of the network. This feature vector encapsulates the high-level visual representation of the input image, learned through the hierarchical processing of convolutional layers in VGG Net. After fine-tuning, the resulting visual feature vector, denoted as 'x', serves as a compact yet comprehensive representation of the image's visual content. Thus, by integrating pre-trained CNN models like VGG Net into the image captioning pipeline, the proposed framework capitalizes on the wealth of visual information encoded within these models, facilitating more robust and contextually relevant caption generation for diverse images.

## Text features:

In the proposed image captioning framework, the textual information extracted using the kerasOCR approach [27] undergoes further processing to facilitate its integration into the

caption generation process. To represent the textual content of the given image IM in a format conducive to computational analysis, word embedding algorithms such as Word2Vec [28] are employed. These algorithms map words into a continuous vector space, where semantically similar words are positioned closer together. By adopting Word2Vec, the spotted words within the image are encoded into fixed-length vectors, enabling efficient manipulation and analysis of textual features within the proposed model.

Each spotted word Ti within the image is represented using a pre-trained word vector fti of 300 dimensions, obtained through Word2Vec embedding. Consequently, for an image containing N spotted words, the resulting text feature size becomes 300 × N. The total size of spotted words is predetermined for both training and testing phases and is denoted by Nmax. Consequently, for each image, the textual information is represented by a fixed-size feature vector of dimensions 300 × Nmax, encapsulating the salient textual elements extracted from the image.

Given that not all spotted words may be equally relevant to the image context, it becomes crucial to assign appropriate weights to each word to ensure the correlation between textual and visual features. To achieve this, different weights are assigned based on the strength of the relationship between the spotted words and the image content. Words that exhibit a strong correlation with the image are assigned higher weights, reflecting their significance in contributing to the overall understanding of the scene. Conversely, words with weaker associations or potential noise are assigned smaller weights, mitigating their influence on the caption generation process. This weighting mechanism ensures that only the most relevant textual information contributes substantially to the final caption, enhancing the coherence and relevance of the generated descriptions. Overall, by incorporating such weighting strategies, the proposed model effectively balances the influence of textual features, thereby improving the fidelity and relevance of the generated image captions.

The following equation 3.7 is used to evaluate the weight wi:

𝑤𝑖 = exp(𝑓𝑇 ⋅ 𝑈 ⋅ 𝑓

𝑣

𝑡𝑖

) (3.7)

where U is the bilinear parameter matrix and to normalize all words T = {Ti} in Softmax way the exponent is used, and all the Nmax word features are fused using weighted sum-pooling methods. The text features represented as z are used as fused word features as follows in (eq. 3.8):

𝑧 = ∑𝑁𝑚𝑎𝑥 𝑤

⋅ 𝑓

(3.8)

𝑖=1

𝑖 𝑡𝑖

To align the dimensions of the Nmax features with those of the visual features, which are represented as 1,4096, the Nmax features are averaged. This averaging process ensures that each Nmax feature contributes equally to the final representation, maintaining consistency across the dimensions. Consequently, the resulting textual features are reduced to a dimensionality of 1,300. This reduction in dimensionality enables efficient integration and comparison with the visual features, facilitating seamless multimodal analysis and interpretation. By harmonizing the dimensions of textual and visual features, the combined representation becomes more compact and manageable, enhancing the overall effectiveness of the multimodal analysis framework.

## Feature fusion:

The proposed method introduces a multimodal fusion approach that combines both visual and textual features to enhance the generation of fine-grained image captions. Initially, visual features denoted as 'x' are extracted from the input images using a pre-trained VGG Net model [24]. These visual features comprise a 4096-dimensional vector, encoding high-level visual representations learned from the images. Concurrently, textual features denoted as 'z' are extracted using the Word2Vec algorithm [28], which maps spotted words within the images into fixed-length vectors in a continuous vector space.

Once both visual and textual feature vectors are extracted, the proposed model employs Long Short-Term Memory (LSTM), a powerful recurrent neural network architecture, to fuse these modalities and generate comprehensive image captions. LSTM is particularly well-suited for sequential data processing tasks due to its ability to capture long- range dependencies and contextual information effectively. In the fusion process, the extracted visual and textual feature vectors are fed into the LSTM model, which dynamically integrates and weighs the contributions of each modality over the course of caption generation. By leveraging the rich contextual information encoded within both visual and textual features, the LSTM model generates coherent and contextually relevant image captions that encapsulate both visual and semantic aspects of the input images.

Overall, the multimodal fusion approach proposed in this method harnesses the complementary nature of visual and textual information to produce fine-grained image

captions that are both descriptive and informative. By integrating information from multiple modalities, the model achieves a more comprehensive understanding of the input images, resulting in captions that capture both the visual content and textual context, thereby enhancing the overall quality and richness of the generated captions.

## LSTM-based caption generator:

The Long Short-Term Memory (LSTM) [29] model for sequence generation [30], as described in previous sections, is characterized by its memory cell (c) and the gating mechanisms that regulate its behavior. These gating mechanisms, including the input gate, output gate, and forget gate, play a crucial role in controlling the flow of information within the LSTM model. Traditionally, the inputs to these gates consist of the current visual feature

(x) and the previous hidden state (mt-1), representing the contextual information from the previous time step.

In the proposed model, it has been recognized that incorporating additional semantic information, such as text features, can contribute to the generation of more meaningful captions. To facilitate the integration of this semantic information into the LSTM model, a context vector 'z' is introduced. This context vector represents the textual features extracted from the input image, providing additional semantic context that complements the visual information.

By incorporating the context vector 'z' as an additional input to the LSTM gates, the proposed model enriches the information available to the LSTM model during caption generation. This enables the model to consider not only the visual features of the image but also the semantic context provided by the textual features. As a result, the LSTM gates can make more informed decisions, dynamically adjusting the flow of information based on both visual and semantic cues.

Overall, the introduction of the context vector 'z' as an additional input to the LSTM gates enhances the model's ability to generate more meaningful and contextually relevant captions by leveraging both visual and textual information. This multimodal approach enables the model to capture a broader range of semantic nuances present in the input images, ultimately leading to improved caption quality and descriptive accuracy. The proposed model’s mathematical formulation of these gates and memory cell is given below (eq. 3.9 – 3.14):

igt = σ(Wigxxt fgt = σ(Wfgxxt ogt = σ(Wogxx

|  |  |
| --- | --- |
| + Wigmmt−1 + WigzZ) | (3.9) |
| + Wfgmmt−1 + WfgzZ) | (3.10) |
| t + Wogmmt−1 + WogzZ) | (3.11) |

cgt = fgt ⊗ cgt−1 + igt ⊗ h(Wcgxxt + Wcgmmt−1 + WcgzZ) (3.12)

mt = ogt ⊗ cgt (3.13)

Pt+1 = Softmax(mt) (3.14)

where z is the context vectors that represent the text features, rðÞ is sigmoid activation function to generate nonlinearity, hðÞ is hyperbolic tangent activation function, trained parameters are several W matrices, and product with a gate value is denoted by ⊗ to train the LSTM network, these gates are used which deal with the problem of vanishing and exploding gradients. Softmax is used at the end step of LSTM for evaluating the probability pt over the words after getting input mt at the previous step. P(St | IM ; z; S0; ...; St-1) has been defined for predicting individual word of the sentence. Since LSTM is a recurrent network, each step shares the common parameters (x and z) and at time t the input of the LSTM is the output mt- 1 at time stamp t - 1 as shown in Fig. 3.2. In the unrolled form, the feed-forward network is the transformation of all recurrent networks. Two special words S0 and SN are used for the purpose of designating the start word and the stop word of the sentence. LSTM uses the stop word to indicate the accomplishment of the generated caption. The inputted features of image IM are mapped to the common space.

The reformation of LSTM parameters, CNN result, and word embedding algorithm can minimize the loss.

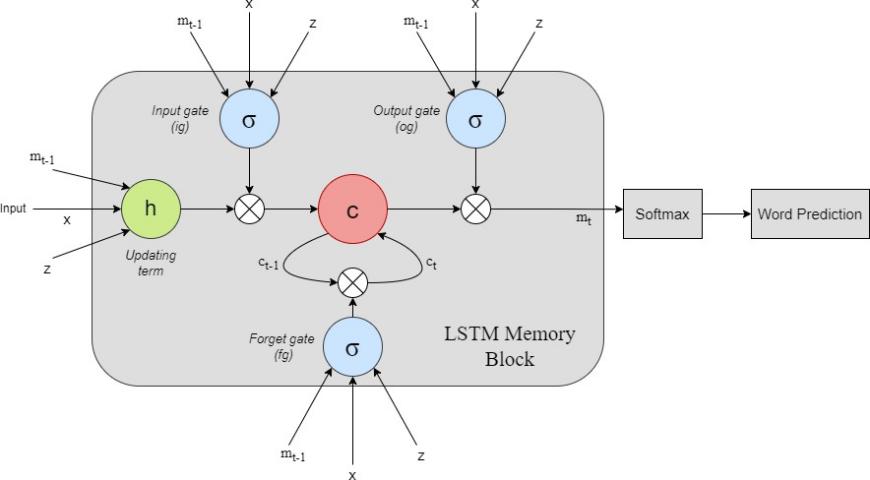


Figure 3.2: Proposed LSTM graphical representation.

# CHAPTER 4

**RESULTS & DISCUSSIONS**

In this section, we delve into a comprehensive examination of the proposed image captioning approach, where both quantitative and qualitative analyses are conducted to assess its efficacy. To begin, we outline the experimental setup and methodology used to validate the impact of Long Short-Term Memory (LSTM) on image captioning. Our experiments are designed to rigorously evaluate the proposed model's performance against a visual baseline method, ensuring a thorough comparison across various benchmark datasets.

Quantitative evaluation forms a cornerstone of our analysis, involving objective metrics to quantify the model's performance in generating captions. Metrics such as BLEU (Bilingual Evaluation Understudy), BLEU-1; BLEU-2; BLEU-3 and BLEU-4 are employed to measure the quality, relevance, and linguistic diversity of the generated captions. Through systematic experimentation and comparison, we aim to provide empirical evidence of the proposed model's superiority over baseline methods and its competitiveness with existing state-of-the-art approaches (using only visual cues).

In addition to quantitative assessments, qualitative analysis is conducted to offer insights into the nuanced aspects of caption generation by the proposed model. Visual inspection of the generated captions alongside their corresponding ground truth captions allows for a qualitative assessment of the model's ability to capture scene details, context, and linguistic fluency. Through visual examination, we aim to elucidate the strengths and limitations of the proposed approach, providing a holistic understanding of its performance beyond numerical metrics.

Furthermore, extensive ablation studies are conducted to investigate the specific contributions of LSTM to the image captioning process. By systematically varying the model architecture and evaluating its performance under different configurations, we aim to elucidate the impact of LSTM on caption quality and coherence. These experiments shed light on the role of LSTM in capturing long-range dependencies, contextual information, and semantic coherence in image captioning tasks.

Overall, our comprehensive evaluation framework encompasses a combination of quantitative metrics, qualitative assessments, and ablation studies to provide a thorough understanding of the proposed image captioning approach. By comparing against baselines

and state-of-the-art methods across benchmark datasets, we aim to demonstrate the effectiveness and robustness of the proposed model in generating high-quality and contextually relevant captions for diverse images.

## Evaluation Metrics:

In the validation of our proposed image captioning approach, we employ two key evaluation metrics: the Bilingual Evaluation Understudy (BLEU) [31]. Traditionally used in assessing automatic machine translation, this metric provide objective measures to evaluate the quality and fidelity of generated captions in image captioning tasks. Similar to machine translation, image captioning relies on the semantic information extracted from the input image and reference sentences, making BLEU well-suited for evaluating the effectiveness of caption generation.

BLEU (Bilingual Evaluation Understudy) is a metric frequently used for assessing the quality of machine-generated text, commonly employed in tasks like machine translation and image captioning. It measures the similarity between a generated text, such as a caption, and one or more reference texts, typically ground truth captions, by computing the precision of N- grams. The N-grams represent sequences of N consecutive words in the text.

1. **BLEU-1 to BLEU-4:** BLEU calculates precision scores for N-grams of varying lengths, ranging from unigrams (BLEU-1) to four-grams (BLEU-4). Each BLEU score indicates how closely the generated text matches the reference text in terms of N-gram overlap.
2. **BLEU-1 (Unigram Precision):** This evaluates the precision of individual words (unigrams) in the generated text compared to the reference text. BLEU-1 is useful for assessing the accuracy of single words.
3. **BLEU-2 (Bigram Precision):** It measures the precision of word pairs (bigrams) in the generated text relative to the reference text. BLEU-2 captures the correctness of word sequences.
4. **BLEU-3 (Trigram Precision):** Similar to BLEU-2, but it focuses on sequences of three consecutive words (trigrams). BLEU-3 provides insights into the correctness of longer word sequences.
5. **BLEU-4 (Four-gram Precision):** Extending the evaluation to sequences of four consecutive words (four-grams), BLEU-4 assesses how well the generated text captures longer and more complex phrases.

## Implementation details:

In the practical implementation setup of our proposed model, we adopt a multimodal LSTM method with an adaptive learning rate algorithm, specifically the Adam algorithm [32], to train the model. This optimization technique enables efficient training by dynamically adjusting the learning rate based on the gradients of the loss function, facilitating faster convergence and improved model performance. By leveraging the Adam algorithm, we aim to optimize the parameters of our model effectively while mitigating the risk of getting stuck in local minima.

To extract visual features from input images, we utilize the Oxford Visual Geometry Group (VGG) Net [24], a deep convolutional neural network (CNN) model pre-trained on the ImageNet dataset. The VGG Net architecture is renowned for its effectiveness in capturing high-level visual representations from images, making it well-suited for feature extraction tasks in computer vision applications. Specifically, we extract the 4096-dimensional visual feature vector generated at the second last layer of the VGG Net model, capturing rich visual information encapsulated within the images.

In addition to visual feature extraction, our proposed approach incorporates textual feature extraction to enrich the caption generation process. We employ a kerasOCR model

[26] to identify and extract salient text regions from the input images. This model is trained to detect text regions based on their prominence and significance within the image context. Subsequently, we utilize the Word2Vec algorithm to convert the detected textual information into a fixed-length textual feature vector. Word2Vec maps words into a continuous vector space, facilitating semantic analysis and comparison of textual features.

Once both visual and textual feature vectors are extracted, they are fused using Long Short-Term Memory (LSTM), a powerful recurrent neural network architecture. LSTM dynamically integrates the visual and textual information, capturing dependencies and relationships between them to generate comprehensive image captions. By leveraging the complementary nature of visual and textual features, the LSTM model produces captions that effectively capture both visual content and semantic context, enhancing the overall descriptive quality and relevance of the generated captions.

Overall, the practical implementation setup of our proposed model encompasses a systematic pipeline involving visual and textual feature extraction, fusion using LSTM, and optimization using the Adam algorithm, enabling efficient and effective generation of fine- grained image captions. Through this setup, we aim to demonstrate the feasibility and

effectiveness of our approach in generating meaningful and contextually relevant captions for diverse images.

## Comparison with baselines:

To provide a fair and comprehensive conclusion, we conducted experiments comparing the proposed method with visual baseline image captioning techniques. The evaluation was performed on two subsets of the dataset: one containing only visual cues and the other including both visual and textual cues. The benchmark datasets used for evaluation were Flickr8k [6]

Tables 4.1 present the results of these experiments, demonstrating that the proposed model, leveraging both visual and textual cues, outperforms the visual baseline method on the benchmark datasets. By incorporating textual information alongside visual features, our model achieves superior performance in generating image captions that are more descriptive, contextually relevant, and semantically rich.

The comparison against visual methods highlights the effectiveness of incorporating textual cues in the image captioning process. While visual cues alone provide valuable information, the inclusion of textual information enhances the model's ability to capture nuanced details and semantic context within the images. As a result, the proposed model surpasses baseline methods in caption quality and accuracy, as evidenced by the evaluation results.

The comparison results validate the effectiveness of the proposed method in leveraging both visual and textual cues for image captioning. By surpassing visual baseline methods, our model demonstrates its superiority in capturing the richness and complexity of image content, ultimately advancing the state-of-the-art in image captioning research.



Figure 4.1: Text detection from Flickr8k.

Table. 4.1: Assessment of model with baseline.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **BLEU-1** | **BLEU-2** | **BLEU-3** | **BLEU-4** |
| Visual Cues | 50.1 | 30.47 | 19.49 | 11.65 |
| Visual + Textual Cues | 56.5 | 34.35 | 22.02 | 13.5 |

Figure 4.1 present the text detected from dataset. Figure 4.2 present the caption results of these experiments on the dataset which doesn’t have text The proposed (right column) describes the image content more accurately with the integration of textual cues even though text is not present. Figure 4.3 present the caption results of these experiments on the dataset which doesn’t have text The proposed (right column) describes the image content more accurately with the integration of textual cues even though text is not present.

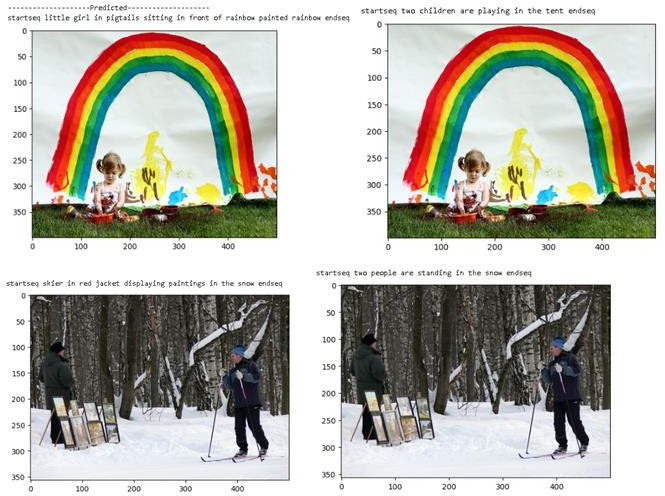


Figure 4.2: Generated captions for images without text.

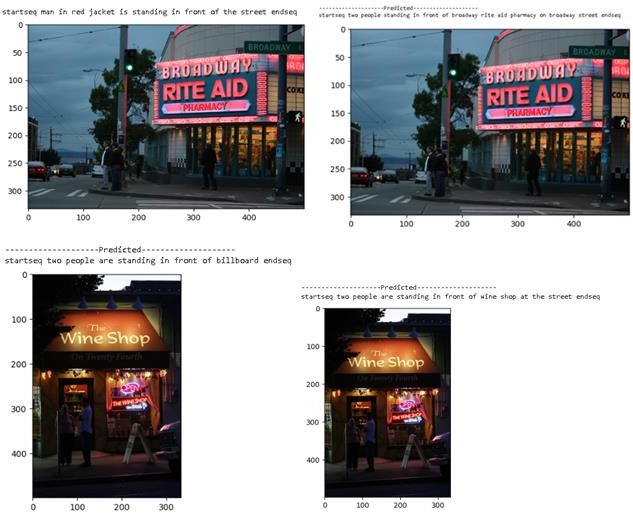


Figure 4.3: Generated captions for images with text.

## User Manual

* + 1. **Visual features extraction from images and storing in ‘features.pkl’ file**

# -\*- coding: utf-8 -\*- """CNN\_LSTM.ipynb

Automatically generated by Colaboratory.

Original file is located at https://colab.research.google.com/drive/1WbdWi8z5nEC6iiN\_cKtd53J\_Vdcw4hGB

"""

from google.colab import drive drive.mount('/content/drive')

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, load\_model

from tensorflow.keras.utils import to\_categorical, plot\_model

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

!unzip /content/drive/MyDrive/major\_project/flickr8k.zip

BASE\_DIR = '/content/flickr8k'

WORKING\_DIR = '/content/drive/MyDrive/major\_project/Working'

# load vgg16 model model = VGG16()

# restructure the model

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

print(model.summary())

# extract features from image features = {}

directory = os.path.join(BASE\_DIR, 'Images')

for img\_name in tqdm(os.listdir(directory)): # load the image from file

img\_path = directory + '/' + img\_name

image = load\_img(img\_path, target\_size=(224, 224)) # convert image pixels to numpy array

image = img\_to\_array(image) # reshape data for model

image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2])) # preprocess image for vgg

image = preprocess\_input(image) # extract features

feature = model.predict(image, verbose=0) # get image ID

image\_id = img\_name.split('.')[0] # store feature features[image\_id] = feature

# store features in pickle

pickle.dump(features, open(os.path.join(WORKING\_DIR, 'features.pkl'), 'wb'))

# load features from pickle

with open(os.path.join(WORKING\_DIR, 'features.pkl'), 'rb') as f: features = pickle.load(f)

print(features.keys()) a=features['1000268201\_693b08cb0e'] print(a.shape)

## Mapping captions to images and storing in ‘mapping.pkl’ file

with open(os.path.join(BASE\_DIR, 'captions.txt'), 'r') as f: next(f)

captions\_doc = f.read()

# create mapping of image to captions mapping = {}

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)

len(mapping)

def clean(mapping):

for key, captions in mapping.items(): for i in range(len(captions)):

# take one caption at a time caption = captions[i]

# preprocessing steps

# convert to lowercase caption = caption.lower()

# delete digits, special chars, etc., caption = caption.replace('[^A-Za-z]', '') # delete additional spaces

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

# add start and end tags to the caption

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

endseq'

captions[i] = caption

# before preprocess of text mapping['1000268201\_693b08cb0e']

# preprocess the text clean(mapping)

# after preprocess of text mapping['1000268201\_693b08cb0e']

* + 1. **Text extraction from images**

# -\*- coding: utf-8 -\*- """8k\_KerasOCR.ipynb

Automatically generated by Colaboratory.

Original file is located at https://colab.research.google.com/drive/1g9Y0I4G7ZIH6M9deV9n0G0nmxE6FnqXE

"""

from google.colab import drive drive.mount('/content/drive')

import os

!pip install -q keras-ocr

import keras\_ocr

import matplotlib.pyplot as plt

# Path to the directory containing the Flickr8k images

flickr8k\_images\_dir = '/content/drive/MyDrive/major\_project/8k/harshi/Images'

# Path to the directory where you want to save the extracted text

output\_dir = '/content/drive/MyDrive/major\_project/8k/harshi/flickr8k\_harshi\_text' # Function to extract text from an image using OCR

pipeline = keras\_ocr.pipeline.Pipeline() def extract\_text\_from\_image(image\_path):

#image = Image.open(image\_path)

#text = pytesseract.image\_to\_string(image) print(image\_path)

images = [keras\_ocr.tools.read(img) for img in [image\_path, ]] # predicted\_image = pipeline.recognize(images) prediction\_groups = pipeline.recognize(images)

predicted\_image = prediction\_groups[0] p=[]

for text, box in predicted\_image: p.append(text)

#print(text) return p

# Function to save the extracted text to a text file def save\_text\_to\_file(text2, image\_filename):

txt\_filename = os.path.splitext(image\_filename)[0] + ".txt" # with open(os.path.join(output\_dir, txt\_filename), "w") as f: # f.write(text)

with open(os.path.join(output\_dir, txt\_filename), "w") as f: # Iterate over the elements of the list

for element in text2:

# Write each element to the file, followed by a newline character f.write(element + "\n")

def text\_file\_exists(image\_filename):

txt\_filename = os.path.splitext(image\_filename)[0] + ".txt" return os.path.exists(os.path.join(output\_dir, txt\_filename))

# Iterate through each image in the Flickr8k dataset and extract text for filename in os.listdir(flickr8k\_images\_dir):

if filename.endswith(".jpg"):

image\_path = os.path.join(flickr8k\_images\_dir, filename) if not text\_file\_exists(filename):

text1 = extract\_text\_from\_image(image\_path) text\_list=[]

for sublist in text1: text\_list.append(sublist)

# Print the resulting list of text print(text\_list) save\_text\_to\_file(text\_list, filename) else:

print('already extracted') continue

text\_files\_dir = '/content/drive/MyDrive/major\_project/8k/harshi/flickr8k\_harshi\_text' images = os.listdir(text\_files\_dir)

print(len(images))

'''# Path to the directory containing the extracted text files

text\_files\_dir = '/content/drive/MyDrive/major\_project/8k/harshi/flickr8k\_harshi\_text' # Function to read text from text files and print if not "FF"

def print\_text\_from\_files(text\_files\_dir): for filename in os.listdir(text\_files\_dir):

if filename.endswith(".txt"):

text\_file\_path = os.path.join(text\_files\_dir, filename) with open(text\_file\_path, "r") as file:

extracted\_text = file.read().strip()

print(f"Text from {filename}: {extracted\_text}") print\_text\_from\_files(text\_files\_dir)'''

## Generating word embedding from the extracted text using Word2Vec and storing in ‘text\_features.pkl’ file

# -\*- coding: utf-8 -\*- """Word2Vec2.ipynb

Automatically generated by Colaboratory.

Original file is located at https://colab.research.google.com/drive/1pp97xDfkkOAVVhgrnXe\_ipuHg24Brsip

"""

from google.colab import drive drive.mount('/content/drive')

!pip install nltk

!pip install -q keras-ocr

import nltk nltk.download('punkt')

import os

import keras\_ocr

import matplotlib.pyplot as plt

from gensim.models import KeyedVectors import os

from gensim.models import Word2Vec from nltk.tokenize import word\_tokenize

folder\_path = '/content/drive/MyDrive/major\_project/30k/Divided\_Images/part1\_text' WORKING\_DIR = '/content/drive/MyDrive/major\_project/Working/30k'

# Load text from each file in the folder words = []

for filename in os.listdir(folder\_path): if filename.endswith('.txt'):

with open(os.path.join(folder\_path, filename), 'r') as file: text = file.read()

words.append(text) print(words) print(len(words))

# Load text from each file in the folder data = {}

for filename in os.listdir(folder\_path): if filename.endswith('.txt'):

with open(os.path.join(folder\_path, filename), 'r') as file: text = file.read()

data[filename] = text print(data)

print(len(data))

from gensim.models import Word2Vec import numpy as np

# Preprocess the text data sentences = []

for file\_name, text in data.items(): words = text.split('\n')

words = [word for word in words if word] # Remove empty strings sentences.append(words)

print(sentences)

# Train Word2Vec model

model = Word2Vec(sentences, vector\_size=300, window=5, min\_count=0, workers=4)

import re

# Preprocess the text data and extract individual words words = []

for file\_name, text in data.items(): file\_words = text.split('\n')

file\_words = [re.sub(r'\W+', '', word) for word in file\_words] # Remove non-word characters

file\_words = [word for word in file\_words if word] # Remove empty strings words.extend(file\_words)

# Train Word2Vec model

model = Word2Vec([words], vector\_size=300, window=5, min\_count=1, workers=4)

# Save the trained model model.save("word2vec\_model")

# Load the trained model

# model = Word2Vec.load("word2vec\_model")

# Obtain embeddings for the words in the dictionary word\_embeddings = {}

for file\_name, text in data.items(): file\_words = text.split('\n')

file\_words = [re.sub(r'\W+', '', word) for word in file\_words] # Remove non-word characters

file\_words = [word for word in file\_words if word] # Remove empty strings

file\_embeddings = [] for word in file\_words:

try:

embedding = model.wv[word] except KeyError:

embedding = np.zeros(300) file\_embeddings.append(embedding)

word\_embeddings[file\_name] = file\_embeddings

# Print word embeddings for each file

for file\_name, embeddings in word\_embeddings.items(): print(file\_name, embeddings)

print(len(word\_embeddings))

import numpy as np

# Create a new dictionary to store reshaped embeddings reshaped\_embeddings = {}

# Iterate over items in the dictionary

for key, value in word\_embeddings.items(): if not value: # If the value is an empty list

reshaped\_embeddings[key] = np.zeros((0, 300)) # Create a (0, 300) array else:

# Convert the list of embeddings to a NumPy array embeddings\_array = np.array(value)

# Reshape the array to the desired shape reshaped\_shape = (len(value), len(value[0]))

reshaped\_array = embeddings\_array.reshape(reshaped\_shape)

# Store the reshaped array in the new dictionary reshaped\_embeddings[key] = reshaped\_array

# Print the reshaped embeddings

for key, value in reshaped\_embeddings.items(): print(key, value.shape)

import pickle

pickle\_filename = WORKING\_DIR+'/text\_features.pkl'

# Load existing data from the pickle file with open(pickle\_filename, 'rb') as f:

existing\_data = pickle.load(f)

# Assume new\_data is the additional data you want to add

new\_data = reshaped\_embeddings # Change this to the new data you want to add

# Update existing\_data with new\_data existing\_data.update(new\_data)

# Dump the combined data back into the pickle file with open(pickle\_filename, 'wb') as f:

pickle.dump(existing\_data, f)

'''import pickle

pickle\_filename = WORKING\_DIR+'/text\_features.pkl'

# print("Pickle file updated successfully:", pickle\_filename)

# Dump the additional dictionary into the pickle file

with open(os.path.join(WORKING\_DIR, 'text\_features.pkl'), "wb") as f: # Use "ab" (append binary) mode

pickle.dump(reshaped\_embeddings, f)

print("Additional values added to pickle file:", pickle\_filename)'''

# load features from pickle

with open(os.path.join(WORKING\_DIR, 'text\_features.pkl'), 'rb') as f: features = pickle.load(f)

print(len(features)) print(features.keys())

a=features['967719295\_3257695095.txt'] print(a.shape)

print(a) print('============================================================

==========================================')

a=features['2021613437\_d99731f986.txt'] print(a.shape)

print(a) print('============================================================

==========================================')

a=features['3336808362\_c17837afd8.txt'] print(a.shape)

print(a) a=features['2831672255\_d779807c14.txt'] print(a.shape)

print(a)

## Image captioning through LSTM with only visual cues (cnn\_lstm.py) file

# -\*- coding: utf-8 -\*- """CNN\_LSTM.ipynb

Automatically generated by Colaboratory.

Original file is located at https://colab.research.google.com/drive/1WbdWi8z5nEC6iiN\_cKtd53J\_Vdcw4hGB

"""

from google.colab import drive

drive.mount('/content/drive')

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

# !unzip /content/drive/MyDrive/major\_project/flickr8k.zip

BASE\_DIR = '/content/flickr8k'

WORKING\_DIR = '/content/drive/MyDrive/major\_project/Working'

# load vgg16 model model = VGG16()

# restructure the model

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

print(model.summary())

import pickle

pickle\_filename = WORKING\_DIR+"/visual\_features.pkl"

with open(pickle\_filename, 'rb') as f: vis\_feat = pickle.load(f)

print(vis\_feat.keys())

print("Length of the loaded dictionary:", len(vis\_feat)) vis\_feat['1001773457\_577c3a7d70'].shape pickle\_filename = WORKING\_DIR+"/mapping.pkl"

with open(pickle\_filename, 'rb') as f: mapping = pickle.load(f)

print(mapping.keys())

print("Length of the loaded dictionary:", len(mapping)) mapping['1001773457\_577c3a7d70']

all\_captions = [] for key in mapping:

for caption in mapping[key]: all\_captions.append(caption)

len(all\_captions)

all\_captions[:10]

# tokenize the text tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(all\_captions) vocab\_size = len(tokenizer.word\_index) + 1

vocab\_size

# get maximum length of the caption available

max\_length = max(len(caption.split()) for caption in all\_captions) max\_length

image\_ids = list(mapping.keys()) split = int(len(image\_ids) \* 0.90)

train = image\_ids[:split] test = image\_ids[split:]

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

# 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)

from keras.callbacks import EarlyStopping

epochs = 200

batch\_size = 32

train\_steps = len(train\_keys) // batch\_size val\_steps = len(val\_keys) // batch\_size

early\_stopping = EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True)

train\_generator = data\_generator(train\_keys, mapping, features, tokenizer, max\_length, vocab\_size, batch\_size)

val\_generator = data\_generator(val\_keys, mapping, features, tokenizer, max\_length, vocab\_size, batch\_size)

model.fit(train\_generator, epochs=epochs, steps\_per\_epoch=train\_steps, validation\_data=val\_generator, validation\_steps=val\_steps, verbose=1, callbacks=[early\_stopping])

model.save\_weights('no\_text\_30k\_best\_weights.h5')

model.load\_weights('best\_weights.h5')

model.save(WORKING\_DIR+'/best\_model\_final.h5')

new\_model = load\_model(WORKING\_DIR+'/best\_model\_final.h5') new\_model.summary()

def idx\_to\_word(integer, tokenizer):

for word, index in tokenizer.word\_index.items(): if index == integer:

return word return None

# generate caption for an image

def predict\_caption(new\_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 = new\_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

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(new\_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)))

print("BLEU-3: %f" % corpus\_bleu(actual, predicted, weights=(0.33, 0.33, 0.33, 0)))

print("BLEU-4: %f" % corpus\_bleu(actual, predicted, weights=(0.25, 0.25, 0.25, 0.25)))

from PIL import Image import matplotlib.pyplot as plt

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]

print(' Actual ') for caption in captions:

print(caption)

# predict the caption

y\_pred = predict\_caption(new\_model, features[image\_id], tokenizer, max\_length) print(' Predicted ')

print(y\_pred) plt.imshow(image)

generate\_caption('/content/1329832826\_432538d331.jpg')

generate\_caption("1002674143\_1b742ab4b8.jpg")

generate\_caption("101669240\_b2d3e7f17b.jpg")

vgg\_model = VGG16() # restructure the model

vgg\_model = Model(inputs=vgg\_model.inputs, outputs=vgg\_model.layers[-2].output)

image\_path = '/content/1370615506\_2b96105ca3.jpg' # load image

image = load\_img(image\_path, target\_size=(224, 224)) # convert image pixels to numpy array

image = img\_to\_array(image) # reshape data for model

image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2])) # preprocess image for vgg

image = preprocess\_input(image) # extract features

feature = vgg\_model.predict(image, verbose=0) # predict from the trained model

predict\_caption(new\_model, feature, tokenizer, max\_length)

## Image captioning through LSTM with visual+textual cues (cnn\_lstm.py) file

# -\*- coding: utf-8 -\*- """CNN\_TEXT\_LSTM.ipynb

Automatically generated by Colaboratory.

Original file is located at https://colab.research.google.com/drive/1ud5L7O0OszY4pxIEeXFOJUQIyctyR6UK

"""

from google.colab import drive drive.mount('/content/drive')

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

# !unzip /content/drive/MyDrive/major\_project/flickr8k.zip

BASE\_DIR = '/content/flickr8k'

WORKING\_DIR = '/content/drive/MyDrive/major\_project/Working'

# load vgg16 model model = VGG16()

# restructure the model

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

print(model.summary())

import pickle

pickle\_filename = WORKING\_DIR+"/visual\_features.pkl"

with open(pickle\_filename, 'rb') as f: vis\_feat = pickle.load(f)

print(vis\_feat.keys())

print("Length of the loaded dictionary:", len(vis\_feat)) vis\_feat['1001773457\_577c3a7d70'].shape

import pickle

pickle\_filename = WORKING\_DIR+"/textual\_features.pkl"

with open(pickle\_filename, 'rb') as f: tf = pickle.load(f)

print(tf.keys())

print("Length of the loaded dictionary:", len(tf)) tf['1001773457\_577c3a7d70'].shape

import pickle import numpy as np

pickle\_filename = WORKING\_DIR + "/textual\_features.pkl"

with open(pickle\_filename, 'rb') as f: tf = pickle.load(f)

print(tf.keys())

print("Length of the loaded dictionary:", len(tf))

text\_feat = {}

for key, arr in tf.items():

# If the array is not empty if arr.size > 0:

# Compute the mean along the first axis (rows) averaged\_arr = np.mean(arr, axis=0)

else:

# If the array is empty, create a zero-filled array of shape (300,) averaged\_arr = np.zeros(300)

text\_feat[key] = averaged\_arr

print("Number of files with textual features:", len(text\_feat)) # Example: Print shape of textual feature for a specific key

print("Shape of textual feature for key '1000268201\_693b08cb0e':", text\_feat['1000268201\_693b08cb0e'].shape) #print(text\_feat['1001773457\_577c3a7d70'])

pickle\_filename = WORKING\_DIR+"/mapping.pkl"

with open(pickle\_filename, 'rb') as f: mapping = pickle.load(f)

print(mapping.keys())

print("Length of the loaded dictionary:", len(mapping)) mapping['1001773457\_577c3a7d70']

all\_captions = [] for key in mapping:

for caption in mapping[key]: all\_captions.append(caption)

len(all\_captions)

all\_captions[:10]

# tokenize the text tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(all\_captions) vocab\_size = len(tokenizer.word\_index) + 1

vocab\_size

# get maximum length of the caption available

max\_length = max(len(caption.split()) for caption in all\_captions) max\_length

image\_ids = list(mapping.keys()) split = int(len(image\_ids) \* 0.80) train = image\_ids[:split]

test = image\_ids[split:]

print(train)

def data\_generator(data\_keys, mapping, vis\_features, text\_features, tokenizer, max\_length, vocab\_size, batch\_size):

X1, X2, X3, y = [], [], [], []

n = 0

while True:

for key in data\_keys: n += 1

captions = mapping[key] for caption in captions:

seq = tokenizer.texts\_to\_sequences([caption])[0] for i in range(1, len(seq)):

in\_seq, out\_seq = seq[:i], seq[i]

in\_seq = pad\_sequences([in\_seq], maxlen=max\_length)[0] out\_seq = to\_categorical([out\_seq], num\_classes=vocab\_size)[0]

X1.append(vis\_features[key][0]) # Visual features X2.append(text\_features[key]) # Textual features X3.append(in\_seq) # Input sequence

y.append(out\_seq) # Output sequence

if n == batch\_size: '''print(np.array(X1).shape) print(np.array(X2).shape) print(np.array(X3).shape)'''

yield [np.array(X1), np.array(X2), np.array(X3)], np.array(y) X1, X2, X3, y = [], [], [], []

n = 0

from keras.layers import concatenate, Input, Dense, Dropout, LSTM, Embedding from keras.models import Model

# Encoder for visual input inputs1 = Input(shape=(4096,)) fe1 = Dropout(0.4)(inputs1)

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

# Encoder for textual input inputs2 = Input(shape=(300,))

# No changes needed for textual input encoding

# Encoder for sequential input inputs3 = Input(shape=(max\_length,))

se1 = Embedding(vocab\_size, 256, mask\_zero=True)(inputs3) se2 = Dropout(0.4)(se1)

se3 = LSTM(256)(se2)

# Decoder

decoder1 = concatenate([fe2, inputs2, se3]) # Concatenating visual, textual, and sequence features

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

outputs = Dense(vocab\_size, activation='softmax')(decoder2)

model = Model(inputs=[inputs1, inputs2, inputs3], outputs=outputs) model.compile(loss='categorical\_crossentropy', optimizer='adam')

# plot the model

plot\_model(model, show\_shapes=True, to\_file='model\_plot.png')

from sklearn.model\_selection import train\_test\_split

early\_stopping = EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True)

# Split the data into training and validation sets

train\_keys, val\_keys = train\_test\_split(train, test\_size=0.2, random\_state=42)

# Train the model with early stopping

epochs = 10

batch\_size = 32

train\_steps = len(train\_keys) // batch\_size val\_steps = len(val\_keys) // batch\_size

model.load\_weights('/content/drive/MyDrive/major\_project/best\_weights3.h5') for i in range(epochs):

# create data generators for training and validation

train\_generator = data\_generator(train\_keys, mapping, vis\_feat, text\_feat, tokenizer, max\_length, vocab\_size, batch\_size)

val\_generator = data\_generator(val\_keys, mapping, vis\_feat, text\_feat, tokenizer, max\_length, vocab\_size, batch\_size)

# fit for one epoch with early stopping

model.fit(train\_generator, epochs=1, steps\_per\_epoch=train\_steps, validation\_data=val\_generator, validation\_steps=val\_steps, verbose=1, callbacks=[early\_stopping])

# Save the trained model weights model.save\_weights('best\_weights4.h5')

model.save\_weights('/content/drive/MyDrive/major\_project/best\_weights3.h5')

model.load\_weights('/content/best\_weights5.h5')

def idx\_to\_word(integer, tokenizer):

for word, index in tokenizer.word\_index.items(): if index == integer:

return word return None

# generate caption for an image

def predict\_caption(model, v\_image, t\_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([v\_image, t\_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

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] a = text\_feat[key]

arr\_reshaped = np.reshape(a, (1, 300)) text\_feat[key] = arr\_reshaped

y\_pred = predict\_caption(model, vis\_feat[key], text\_feat[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)))

print("BLEU-3: %f" % corpus\_bleu(actual, predicted, weights=(0.33, 0.33, 0.33, 0)))

print("BLEU-4: %f" % corpus\_bleu(actual, predicted, weights=(0.25, 0.25, 0.25, 0.25)))

from PIL import Image import matplotlib.pyplot as plt

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]

print(' Actual ') for caption in captions:

print(caption)

# predict the caption

a = text\_feat[image\_id]

arr\_reshaped = np.reshape(a, (1, 300)) text\_feat[image\_id] = arr\_reshaped

y\_pred = predict\_caption(model, vis\_feat[image\_id], text\_feat[image\_id], tokenizer, max\_length)

print(' Predicted ') print(y\_pred)

plt.imshow(image)

generate\_caption("1001773457\_577c3a7d70.jpg") generate\_caption("101669240\_b2d3e7f17b.jpg") generate\_caption("1355945307\_f9e01a9a05.jpg") generate\_caption("1370615506\_2b96105ca3.jpg")

## OUTPUT:

Figure 4.4 represents the output of image captions of the discussed experiments on the dataset which doesn’t have text The proposed (right column) describes the image content more accurately with the integration of textual cues even though text is not present. Figure 4.5 present the output of image captions of these experiments on the dataset which doesn’t have text The proposed (right column) describes the image content more accurately with the integration of textual cues even though text is not present.

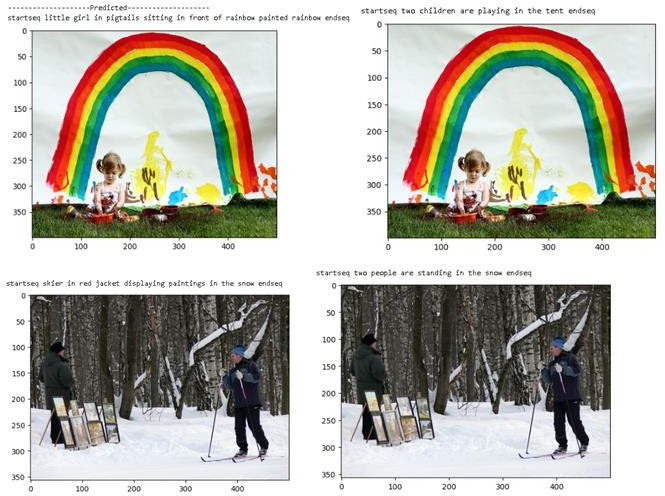


Figure 4.4: Output 1- Image captions (without text in image)

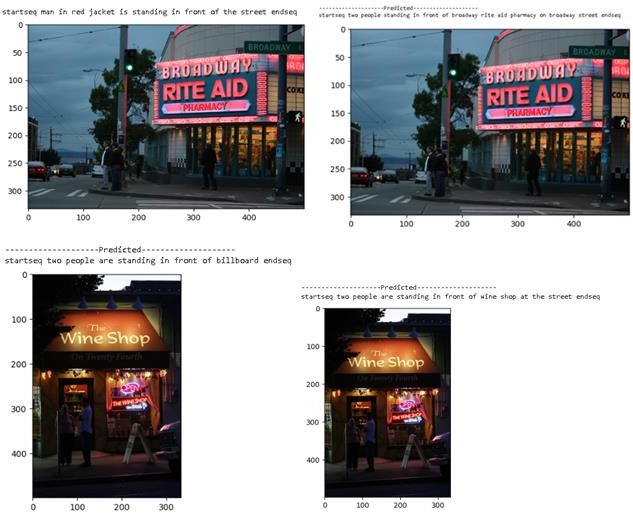


Figure 4.5: Output 2- Image captions (with text in image)

# CHAPTER 5 CONCLUSION

* 1. **Conclusion**

In this project, we have introduced a novel model aimed at generating fine-grained image captions by integrating both visual and textual features extracted from the images. Our investigation revealed that relying solely on visual features for caption generation often leads to insufficient and generic descriptions of scenes. To address this limitation, we proposed leveraging textual information extracted as salient text from the images to enhance the meaningfulness of the generated captions. In our proposed model, we perform feature fusion by incorporating the textual feature vector alongside visual features, injecting it into the initial layer of the Long Short-Term Memory (LSTM) architecture. This fusion mechanism allows the model to leverage both visual and textual cues throughout the caption generation process, enabling it to capture a more comprehensive understanding of the image content and produce more nuanced and contextually relevant captions. Through comprehensive experiments conducted on benchmark datasets, we have demonstrated the effectiveness of our proposed approach. The results showcase the superior performance of our model compared to visual baseline methods and other state-of-the-art caption generation techniques. By outperforming existing methods, our model establishes itself as a leading approach in the field of image captioning, offering enhanced capabilities in generating descriptive and insightful captions for diverse images.

* 1. **Future Work**

One potential direction is to explore task-specific caption generation, tailoring the model to generate captions optimized for specific applications or domains. Additionally, further research could focus on enhancing the fusion mechanism to better integrate visual and textual features and explore novel architectures for more effective caption generation. Overall, our proposed approach lays the groundwork for future advancements in image captioning research and holds promise for applications requiring detailed and contextually relevant image descriptions.

# CHAPTER 6 REFERENCES

1. Goh H, Thome N, Cord M, Lim JH (2014) Learning deep hier archical visual feature coding. IEEE Trans Neural Netw Learn Syst 25(12):2212–2225
2. Zhang N, Ding S, Zhang J, Xue Y (2017) Research on point-wise gated deep networks.

Appl Soft Comput 52:1210–1221

1. Papa JP, Scheirer W, Cox DD (2016) Fine-tuning deep belief networks using harmony search. Appl Soft Comput 46:875–885
2. Bai S (2017) Growing random forest on deep convolutional neural networks for scene categorization. Expert Syst Appl 71:279–287
3. Cho K, Van Merrie¨nboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078
4. Rashtchian C, Young P, Hodosh M, Hockenmaier J. (2010) Collecting image annotations using Amazon’s Mechanical Turk. In: Proceedings of the NAACL HLT 2010 workshop on creating speech and language data with Amazon’s Mechanical Turk. Association for Computational Linguistics, pp 139–147
5. Young P, Lai A, Hodosh M, Hockenmaier J (2014) From image descriptions to visual denotations: new similarity metrics for semantic inference over event descriptions. Trans Assoc Comput Linguist 2:67–78
6. Gupta N, Jalal AS (2017) A comparison of visual attention models for the salient text content detection in natural scene. In: 2017 conference on information and communication technology (CICT). IEEE, pp 1–5
7. Farhadi A, Hejrati M, Sadeghi MA, Young P, Rashtchian C, Hockenmaier J, Forsyth D (2010) Every picture tells a story: generating sentences from images. In: European conference on computer vision. Springer, Berlin, pp 15–29
8. Ordonez V, Kulkarni G, Berg TL (2011) Im2text: describing images using 1 million captioned photographs. In: Advances in neural information processing systems, pp 1143– 1151
9. Gupta A, Verma Y, Jawahar CV (2012) Choosing linguistics over vision to describe images. In: AAAI, p 1
10. Kulkarni G, Premraj V, Ordonez V, Dhar S, Li S, Choi Y et al (2013) Babytalk: understanding and generating simple image descriptions. IEEE Trans Pattern Anal Mach Intell 35(12):2891–2903
11. Mitchell M, Han X, Dodge J, Mensch A, Goyal A, Berg A et al (2012) Midge: generating image descriptions from computer vision detections. In: Proceedings of the 13th conference of the European chapter of the association for computational linguistics.

Association for Computational Linguistics, pp 747–756

1. Ushiku Y, Yamaguchi M, Mukuta Y, Harada T (2015) Common subspace for model and similarity: phrase learning for caption generation from images. In: Proceedings of the IEEE interna tional conference on computer vision, pp 2668–2676
2. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436
3. Socher R, Karpathy A, Le QV, Manning CD, Ng AY (2014) Grounded compositional semantics for finding and describing images with sentences. Trans Assoc Comput Linguist 2(1):207–218
4. Karpathy A, Joulin A, Fei-Fei LF (2014) Deep fragment embeddings for bidirectional image sentence mapping. In: Advances in neural information processing systems, pp 1889–1897
5. Girshick R, Donahue J, Darrell T, Malik J (2014) Rich feature hierarchies for accurate object detection and semantic segmen tation. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 580–587
6. Ma L, Lu Z, Shang L, Li H (2015) Multimodal convolutional neural networks for matching image and sentence. In: Proceed ings of the IEEE international conference on computer vision, pp 2623–2631
7. Kiros R, Salakhutdinov R, Zemel R (2014) Multimodal neural language models. In:

International conference on machine learning, pp 595–603

1. Mao J, Xu W,Yang Y,WangJ,Yuille AL (2014) Explain images with multimodal recurrent neural networks. arXiv preprint arXiv: 1410.1090
2. Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classi f ication with deep convolutional neural networks. In: Advances in neural information processing systems, pp 1097–1105
3. Verma, Akash, Arun Kumar Yadav, Mohit Kumar, and Divakar Yadav. "Automatic image caption generation using deep learning." *Multimedia Tools and Applications* 83, no. 2 (2024): 5309-5325.
4. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv: 1409.1556
5. Sharif Razavian A, Azizpour H, Sullivan J, Carlsson S (2014) CNN features off-the- shelf: an astounding baseline for recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition workshops, pp 806–813
6. Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S et al (2015) Imagenet large scale visual recognition challenge. Int J Comput Vis 115(3):211–252 123 Neural Computing and Applications.
7. Dome, Saurabh, and Asha P. Sathe. "Optical character recognition using tesseract and classification." 2021 International Conference on Emerging Smart Computing and Informatics (ESCI). IEEE, 2021.
8. Mikolov T, Chen K, Corrado G, Dean J (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781
9. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
10. Graves A (2013) Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850
11. Papineni K, Roukos S, Ward T, Zhu WJ (2002). BLEU: a method for automatic evaluation of machine translation. In: Proceedings of the 40th annual meeting on association for computational linguistics. Association pp 311–318 for Computational Linguistics, pp 311–318
12. Kingma DP, Ba J (2014) Adam: a method for stochastic opti mization. arXiv preprint arXiv:1412.6980