

A Project Report On

# **Efficient Image Captioning with Attention Mechanisms: Leveraging CNN's And Simplified GRU's**

*Major project submitted in partial fulfillment of the requirements for the  
award of the degree of*

**BACHELOR OF TECHNOLOGY  
IN  
INFORMATION TECHNOLOGY  
(2021-2025)  
BY**

**SRIRAM CHILIVERI      21241A1256**

**P. MAHESH BABU      21241A1246**

**YENAMALA VARSHIT    21241A1265**

*Under the Esteemed Guidance  
of*

**DR. R. V. S. S. NAGINI**

**Associate Professor**

**DEPARTMENT OF INFORMATION TECHNOLOGY  
GRIET**



**GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY**

**(AUTONOMOUS)**

**HYDERABAD**

**2024-25**



## **CERTIFICATE**

This is to certify that it is a bonafide record of Major Project work entitled “**Efficient Image Captioning with Attention Mechanisms: Leveraging CNN’s And Simplified GRU’s**” done by **SRIRAM CHILIVERI (21241A1256), P. MAHESH BABU (21241A1246), YENAMALA VARSHIT (21241A1265)** of **B.Tech** in the Department of Information of Technology, **Gokaraju Rangaraju Institute of Engineering and Technology** during the period 2021-2025 in the partial fulfillment of the requirements for the award of degree of **BACHELOR OF TECHNOLOGY IN INFORMATION TECHNOLOGY** from GRIET, Hyderabad.

**Dr. R. V. S. S. S. Nagini**  
Associate Professor  
(Internal Guide)

**Dr. Y J Nagendra Kumar**  
Head of the Department

**(Project External)**

## ACKNOWLEDGEMENT

We take the immense pleasure in expressing gratitude to our Internal guide, **Dr. R. V. S. S. Nagini, Associate Professor, Dept of IT, GRIET**. We express our sincere thanks for his encouragement, suggestions and support, which provided the impetus and paved the way for the successful completion of the project work.

We wish to express our gratitude to **Dr. Y J Nagendra Kumar**, HOD IT, our Project Coordinators **Ms. A. Pavithra** and **Mr. G. Vijendar Reddy** for their constant support during the project.

We express our sincere thanks to **Dr. Jandhyala N Murthy**, Director, GRIET, and **Dr. J. Praveen**, Principal, GRIET, for providing us the conducive environment for carrying through our academic schedules and project with ease.

We also take this opportunity to convey our sincere thanks to the teaching and non-teaching staff of GRIET College, Hyderabad.



**Email:** sriramchiliveri1544@gmail.com

**Contact No:** 8074294033



**Email:** mahesh9949644@gmail.com

**Contact No:** 9177795648



**Email:** varshithyenamala@gmail.com

**Contact No.** 7995140990

## **DECLARATION**

This is to certify that the major-project entitled “**Efficient Image Captioning with Attention Mechanisms: Leveraging CNN’s And Simplified GRU’s**” is a bonafide work done by us in partial fulfillment of the requirements for the award of the degree **BACHELOR OF TECHNOLOGY IN INFORMATION TECHNOLOGY** from Gokaraju Rangaraju Institute of Engineering and Technology, Hyderabad.

We also declare that this project is a result of our own effort and has not been copied or imitated from any source. Citations from any websites, books and paper publications are mentioned in the Bibliography.

This work was not submitted earlier at any other University or Institute for the award of any degree.

**SRIRAM CHILIVERI      21241A1256**

**P. MAHESH BABU        21241A1246**

**YENAMALA VARSHIT    21241A1265**

## **TABLE OF CONTENTS**

	<b>Name</b>	<b>Page no</b>
	<b>Certificates</b>	li
	<b>Contents</b>	V
	<b>Abstract</b>	vii
<b>1</b>	<b>INTRODUCTION</b>	1
1.1	Introduction to project	1
1.2	Existing System	3
1.3	Proposed System	5
<b>2</b>	<b>REQUIREMENT ENGINEERING</b>	6
2.1	Hardware Requirements	6
2.2	Software Requirements	6
<b>3</b>	<b>LITERATURE SURVEY</b>	7
<b>4</b>	<b>TECHNOLOGY</b>	9
<b>5</b>	<b>DESIGN REQUIREMENT ENGINEERING</b>	17
5.1	UML Diagrams	18
5.2	Use-Case Diagram	19
5.3	Class Diagram	20
5.4	Sequence Diagram	21
5.5	Deployment Diagram	22
5.6	System Architecture	23
<b>6</b>	<b>IMPLEMENTATION</b>	24
<b>7</b>	<b>SOFTWARE TESTING</b>	34
7.1	Unit Testing	34
7.2	Integration Testing	34
7.3	Performance Testing	34
7.4	Model Testing	35
7.5	Validation Testing	35
<b>8</b>	<b>RESULTS</b>	36
<b>9</b>	<b>CONCLUSION AND FUTURE</b>	38

<b>10</b>	<b>ENHANCEMENTS BIBLIOGRAPHY</b>	<b>39</b>
-----------	--------------------------------------	-----------

<b>11</b>	<b>LIST OF DIAGRAMS</b>
-----------	-------------------------

<b>S No</b>	<b>Figure Name</b>	<b>Page no</b>
<b>1</b>	Use Case Diagram	19
<b>2</b>	Class Diagram	20
<b>3</b>	Sequence Diagram	21
<b>4</b>	Deployment Diagram	22
<b>5</b>	Architecture	23

## ABSTRACT

Over the last few years, automatic image captioning has emerged as a rapidly growing subfield with potential use in assisting persons with visual impairment, enriching social networks and improving image search results pages. Thus, the possibility of automatic creation of captions that are semantically and syntactically correct for the images offered can revolutionize the use of images. This work aims to propose an automatic image captioning system that utilizes CNNs, RNNs and attention mechanisms for image-to-description translation.

This model in particular employ convolutional neural networks, namely Inception v3 as encoders to extract complex features from image. Indeed, the process of excessive training is computationally expensive whereas by using transfer learning, the model comes already with pretrained weights. For creation of the captions, an RNN with the simplified version of the Gated Recurrent Unit or GRU is used due its smaller memory footprint and improved training time relative to LSTM. The attention mechanism enhances the model by enabling it to selectively attend to regions within the image when generating captions, outlining the capacity to mimic shared visual attention and thereby boosting the models performance.

Having evaluated it on Flickr8k dataset, proved that the model can produce meaningful captions with high BLEU score, which make the proposed approach relevant solution for real-world tasks in accessibility, social networks, and automatized generation of texts. The proposed algorithm offers not only improvements to image captioning but also thinking about the combined application of various deep learning methods to address challenging problems.

**Keywords:** Image Captioning, Attention Mechanisms, Convolutional Neural Network (CNN)  
Gated Recurring Units (GRUs), Transfer Learning

**Domain:** Machine learning

# **1. INTRODUCTION**

## **1.1 Introduction to Project**

The creation of algorithms for the generation of Natural Language Descriptions for images has become an invaluable tool in the contemporary new digital society particularly concerning applications such as accessibility to content, social networks, commerce, and security surveillance among others. The procedure of writing descriptions for the images is called as Image Captioning and it is the confluence of two domain namely Computer Vision and Natural Language Processing. The purpose is to provide semantically significant and syntactically well-formed textual captions that reflect the content of the image , thus, it is a problem solving that comes down to the understanding of the text and using words appropriate to a certain context.

Therefore, deep learning has further enhanced the state of art of image captioning. One of the most popular methodologies include the use of CNNs for feature extraction while RNNs for producing the captions. CNNs, especially those that have been pre-trained on other large image datasets such as Imagenet, can be seen as feature extractors that encode the input image into a new high level feature space. These features are then put through an RNN as LSTM or GRU where the image information is translated into a sequence form of a caption. However, and while traditional models like LSTM also do deliver good results in such cases, they do have their limitations when it comes to, for example, longer sequences or when it comes to modelling the more detailed interactions between the content of the image.

The first and significant step to overcome these challenges is the Attention Mechanism, which is a method based on human cognitive processes within the model where the model attention is directed towards specific areas of the image while generating each word of the caption. This mechanism implements a form of attention which ensures the model does not focus on the entire image but pays attention to regions in the image depending on the context of the words generated. This selective attention makes the generated captions more descriptive and contextually related to the images in the investigated model.



In this work, an image captioning system is designed wherein the InceptionV3 is used for feature extraction (CNN), GRU for indirect sentence generation, and an attention mechanism for augmenting the model's performance. Continuing from the InceptionV3 CNN model that, itself, is pretrained on a large number of images in ImageNet database, the transfer learning intension ensures that the model draws on a large amount of prior knowledge and requires, therefore, relatively fewer training epochs as well as produces better results given limited training images. Since LSTM have drawbacks, GRU is utilized for sequence generation that is the efficient approach than LSTM. This decision decreases the amount of memory used but at the same time does not hamper the potential of the model to generate captions. Attention abstraction is also introduced to improve the focusing ability of the model especially when generating captions for specific aspects of the images.

That is why automated image captioning becomes more essential in the modern world. It has the main function of rendering images to visually impaired people in real time, which makes digital content more friendly. In social media processes, automatic captioning makes sharing of content easy because it generates accurate descriptions for millions of images posted daily. In e-commerce and search engines such systems are useful in improving product descriptions and image searches. Also, image captioning can be used in surveillance, particularly in generating descriptions of video streams.

The purpose of this paper is to propose and decipher an image captioning model with the assistance of CNNs, GRUs, and attention. To support this assertion, we examine the sub-architecture of the design and assess the proposed approach's efficiency using standard datasets such as Flickr8k, concluding that this approach can produce accurate captions with reasonable computational complexity. The standard of the results proves that our method provides a perspective solution for such real-life implementations, providing the right captions that are closely connected with the images, and requiring fewer calculations compared to other deep learning methods for image captioning.

## **1.2 Existing System**

### **1. An Example of the Show and Tell Model - 2015**

The Show and Tell model proposed by Google incorporates CNNs as an encoder and RNNs LSTMs as a decoder. The encoder reads the image part of the dataset and generates an output while the decoder produces captions for each time step.

#### **Limitations:**

- The model uses the same representation for every word, hence does not consider any details at the regional or local level.
- In practical driving, it fails in specific scenes where objects and their activities occur together since it lacks the attention mechanism.
- A small amount of language variation consisting of the primary language and thus generates such qualities like banal and repetitiveness of captions.
- Problems arise from the fact that training is highly dependent on the quality of training data and may provide wrong or biased captions for rare data.
- Capability of RNNs in performing real-time captioning is not as efficient because RNNs work in a sequential manner.

### **2. Explanatory text based on Show, Attend, and Tell (2016)**

The advantage of this model is that they introduce an attention mechanism that enable it to attend to certain regions of the image while generating captions. They employ a soft-attention model for capturing dynamic regions that should be focused during caption generation.

#### **Limitations:**

- Technically challenging because of the decoding where subsequent emphasis is put on different segments of the image.
- If the base model is overfitting the data already, adding attention layers makes it overfit even more so due to increased model capacity especially when the dataset size is small.
- This model is used to fail in capturing long-term dependencies successfully, specifically when it is used in lengthy captions.
- Fails to fully consider linguistic context and as such can from time to time include grammatical mistakes or non- sense sentences.

Overall performance relies at a big level on the ability of the CNN to extract proper features.

### **3. Attention: Bottom-up and Top-Down (2018)**

This model blends object recognition (bottom-up) with global contextual context in the likelihood distribution over the visual input (top-down) for writing captions. For specific descriptions, it uses detected object features.

#### **Limitations:**

- Junior user networks with heavy dependence on pre-trained object detection networks, this impairs performance on seldom bending or novel objects.
- Difficulties in judging the spatial position of an object relative to another object, frequently, unable to identify interactions or context.
- The complexity of the models that are used has grown much higher, such that they demand huge computational power for training as well as at the time of deployment.
- It exhibits diminished performance whenever used on scenes that are highly occluded with other scenes with objects that are on top of each other.
- This leads to a pipeline that has multiple stages, the dependencies of which on separate object detection modules make the pipeline less suited for generalization to end to end systems.

### **4. NIC-V- Neural Image Captioning with Visual Attention (2017).**

This system integrates a visual attention based mechanism with CNN-LSTM for the caption generation. This one determines the attention weights of visual features to augment important parts of the image.

#### **Limitations:**

- Constriction of attention weights; the model gives more importance to insignificant features of the image.
- Some weaknesses are as follows: the method has low transferability in analyzing datasets with various characteristics, because hyperparameters for tuning attention are fixed.
- Difficulties to work with any image that has abstract or conceptual component that does not have referent features.
- Prone to mistakes when it comes to discriminately perceiving details like two similar objects.
- Not designed for edge or low power devices and requires a lot of resources.

### 1.3 Proposed System

The proposed image captioning system integrates InceptionV3, GRU, and an attention mechanism to overcome the limitations of previous models such as Show and Tell, Show, Attend and Tell, Attention: Broadly categorized as Bottom-up and Top-Down, and NIC-V. The enhancements include:

#### **Enhanced Local Context Awareness:**

- Show and Tell Model: Does not support attention mechanisms, in particular, there is only one vector representation per word in the sequence, and no mechanism to self-attention to image details when generating new words in the sequence.
- Our System: Introduces an attention mechanism that is able to ‘focus’ on certain parts of the image at the time of captioning. This results into accurate captions generated with much referencing to the context of the scene where multiple objects or interactions are depicted.

#### **Improved Long-Term Dependency Handling:**

- Show, Attend, and Tell: Failure to memorize captions of long sequences and may create syntactically wrong sentences and sometimes come up with garbage data.
- Our System: Uses GRU (Gated Recurrent Units), which is autonomous and better than LSTM in processing long-term dependency as well as providing real-time captions with comparable quality.

#### **Object Recognition and Contextual Understanding:**

- Attention: Bottom-up and Top-Down: In this method, object detection models are strongly relied on, which does not allow objects to recognize their interactions or spatial dependencies.
- Our System: Connects global and local features by/through attention and helps thus to recognize object interactions and relative location in the image.

#### **Scalability and Flexibility:**

- NIC-V: The setting of hyperparameters for the tuning of attention is fixed to hinder the method’s ability to transfer from one dataset to another.
- Our System: The fact that there’s a fixed feature extractor in InceptionV3 and then, free attention layers make it more versatile to adapt to a large number of datasets without having to rebuild it so heavily or overfit.

In general, for image captioning, the proposed system can be said to be far much better than previous models in terms of resilience, scalability and capability in producing high quality captions whenever the need arises in busy and complex working environments.

## **2. REQUIREMENT ENGINEERING**

### **2.1 Hardware Requirements**

- Processor – i5 and above (64-bit OS).
- Memory – 4GB RAM (Higher specs are recommended for high performance)
- Input devices – Keyboard, Mouse

### **2.2 Software Requirements**

- Anaconda Navigator/Jupyter Notebook
- Python
- Python Libraries
  - 1.pandas
  - 2.numpy
  - 3.matplotlib
  - 4.Tensorflow/Keras
  - 5.Pillow
  - 6.OpenCV
  - 7.NLTK
  - 8.scikit-learn
  - 9.pickle
  - 10.h5py

## 2. LITERATURE SURVEY

1. **Long-term Recurrent Convolutional Networks for Visual Recognition and Description**  
Donahue et al. (2015) introduced the Long-term Recurrent Convolutional Networks (LRCN) model, combining CNNs for feature extraction and LSTMs for generating image captions. This model processes visual data from the Flickr30k dataset, where CNNs capture image features and LSTMs sequentially generate captions. However, the sequential nature of LSTMs leads to slower training and inference times, limiting its use for real-time applications. The authors suggest that transformer-based models could improve the speed and efficiency of caption generation by enabling parallelization of processes.
2. **Deep Visual-Semantic Alignments for Generating Image Descriptions (2015)**  
Karpathy and Fei-Fei (2015) proposed a model that aligns image regions with corresponding words or phrases in sentences using the Flickr30k dataset. The approach leverages pre-trained CNNs for visual feature extraction but lacks sufficient fine-tuning of these CNNs, which leads to less precise alignments between image regions and textual descriptions. The authors recognize that more advanced techniques, such as vision transformers, could improve the model's performance by offering more accurate visual-semantic alignments.
3. **Image Caption with Region-based Attention and Scene Factorization (2016)**  
Jin et al. (2016) introduced a region-based attention mechanism for image captioning, which focuses on specific image regions during caption generation. This approach, applied to the Flickr30k dataset, improves caption accuracy by incorporating both local and global information. However, the region-based attention mechanism struggles to effectively capture the global context, limiting the model's ability to generate coherent captions for complex scenes. The authors suggest that transformer models, with their ability to process both local and global contexts simultaneously, could address this limitation.
4. **Video Captioning with Attention-Based LSTM and Semantic Consistency (2017)**  
Gao et al. (2017) proposed an attention-based LSTM model to generate video captions, with a focus on maintaining semantic consistency. While this model is effective for videos, its complexity is an obstacle for single-image captioning tasks. The authors argue that simpler, more optimized models should be developed for image captioning, avoiding the computational overhead associated with video captioning models.

5. **Looking Deeper and Transferring Attention for Image Captioning (2018)**  
Fang et al. (2018) presented a deep transfer learning approach that integrates attention mechanisms to generate image captions. Their model relies on pre-trained networks for feature extraction, which can lead to performance issues if not fine-tuned correctly. The authors recognize that while transfer learning can be beneficial, models like transformers, which dynamically adjust attention based on the image, may provide more flexibility and accuracy in caption generation.
6. **Predicting Visual Features from Text for Image and Video Caption Retrieval (2018)**  
Dong et al. (2018) introduced a framework for image caption retrieval by aligning visual and textual features. Their approach, based on the Flickr8k dataset, focuses on caption retrieval rather than generation. While effective for retrieval tasks, this model is less suited for generating unique and coherent captions, a limitation that the authors highlight. They suggest that models tailored to caption generation, such as those using transformer-based architectures, would perform better in tasks requiring original captions.
7. **Image Captioning: From Structural Tetrad to Translated Sentences (2019)**  
Guo et al. (2019) proposed a simplified approach to image captioning by using deep learning techniques on the Flickr8k dataset. Their method utilizes a structural tetrad approach but struggles with more complex or abstract visual information. The authors note that advanced language models with greater contextual understanding could outperform their approach in generating detailed and coherent captions.
8. **VD-SAN: Visual-Densely Semantic Attention Network for Image Caption Generation (2019)**  
He et al. (2019) developed the VD-SAN model, which utilizes densely connected semantic attention networks to improve image captioning accuracy on the Flickr30k dataset. While the model achieves high performance in attention to significant image areas, its computational complexity and resource demands make it less suitable for deployment on resource-constrained devices. The authors suggest that transformer-based models, which are more scalable and efficient, could provide better solutions for large-scale applications.

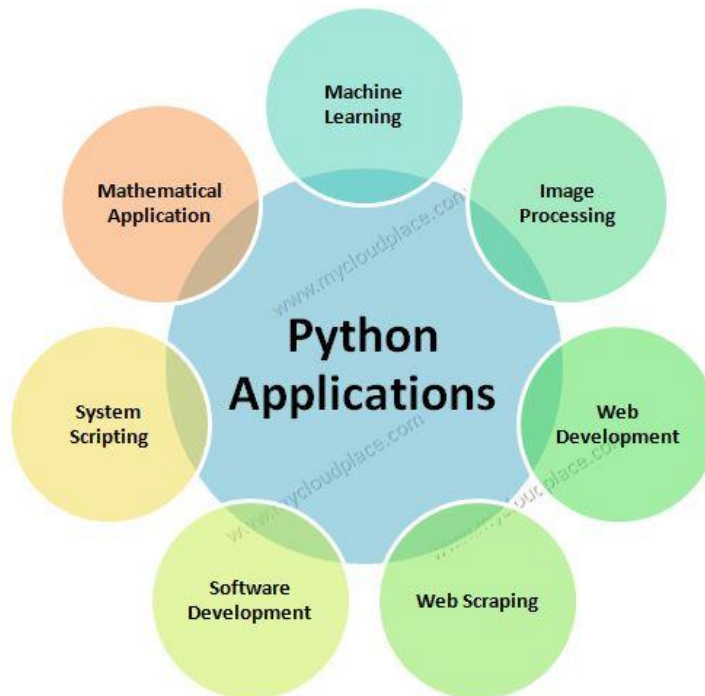
## 4. TECHNOLOGY

### 4.1 ABOUT PYTHON

Python is an adaptable, great level programming language, which is widely used due to its simple syntax. Python was launched in 1991, today it is widely used to develop applications for various fields including web, mathematics and natural sciences. It has rich standard library and allows working in oo, functional and procedural style at the same time is good for both – beginners and professionals. Python language is very plain and with clearly written code, readable to maintain this language in academic community and business world has grown very popular.

### 4.2 APPLICATIONS OF PYTHON

Python is used in many application domains. It makes its presence in every emerging field. It is the fastest-growing programming language and may be used to create any type of application.



*[Fig – 1] Applications of python*



It is used in various fields:

- Web Applications. We can use Python to develop web applications. ...
- Desktop GUI Applications. ...
- Console-based Application. ...
- Software Development. ...
- Scientific and Numeric. ...
- Business Applications. ...
- Audio or Video-based Applications. ...
- 3D CAD Applications.

## **4.3 PYTHON IS WIDELY USED IN DATA SCIENCE**

Python is widely adopted for ML and DL because of its rich line of libraries and frameworks available. Frameworks like NumPy, Pandas, actually are tools for data analysis and data manipulation and other traditional machine learning algorithms are also available in Scikit-learn. In deep learning, high level environment APIs like TensorFlow, Keras and PyTorch offers the better way of implementing neural networks.

The major python libraries used are as follows

### **4.3.1 PANDAS**

A library in python that's used for statistics analyzing, cleaning, exploring and manipulating. generally, dataset incorporates many beneficial and useless statistics. Pandas cause them to readable and relevant.

### **4.3.2 NUMPY**

A library in python that's used for statistics reading, cleaning, exploring and manipulating. commonly dataset carries many useful and vain data. Pandas make them readable and relevant.

### 4.3.3 MATPLOTLIB

A library is used for plotting graphs in python. It built on NumPy arrays. We can plot any graph from basic plot types to bar graph, histogram, scatter and many more.

### 4.3.4 SCIKIT-LEARN

In Python, a library for device gaining knowledge of. device gaining knowledge of and statistical modelling, along with class, regression, and clustering, are completed with Scikit-learn gear.

### 4.3.5 CSV

It is a kind of file that stores tabular records, like a spreadsheet or a database. There are one or extra fields in each entry, that are separated by commas. We use the csv built in module to work with CSV files.

## 4.4 CNN and Transfer Learning:

For this project, Convolutional Neural Networks (CNN) and Transfer Learning are employed to construct an optimal image captioning approach. Below are the key aspects:

**1. CNN as Encoder:** Features from input images are obtained by reshaping the images into a fixed size vector in the case of CNN. A few of them are convolution layers, fully connected layers, pooling layers, and dropout layers in the CNNs. CNN is an essential encoder that encodes the digital images into features for more interpretation in this case.

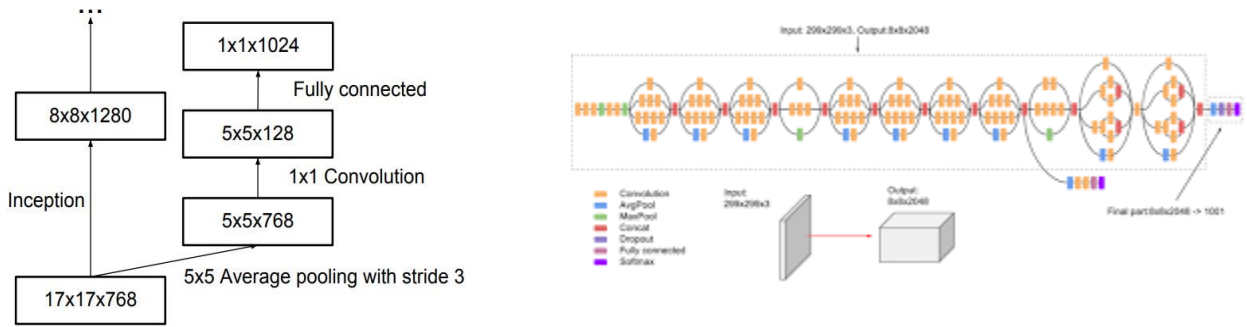
**2. ReLU and Dropout:** The ReLU activation function integrates non-linearity to the network to become  $f(x)=\max(0,x)$  to let the network learn various patterns. The Dropout layer effectively shuts down some percentages of neurons during training in a bid to make the network more generalized.

**3. Transfer Learning:** In transfer learning, information is obtained from models trained on other data, rather than from the start. The use of this method enhances the training of the image captioning

model besides shortening the training time. In this project, Inception-V3 is used and it is a model trained on the ImageNet dataset.

**4. Inception V3 Architecture:** Inception V3 model has 48 layers and the last convolutional layer generated an output shape of  $8 \times 8 \times 2048$ . The architecture concentrates on cutting down the required computations through factorized convolutions, small convolutions and asymmetrical convolutions. Moreover, for training purposes, and additional classifier will be combined to support a regularization mechanism that boosts stability.

Using these components the system is aimed at achieving better results of image captioning with increased speed.



[Fig – 2] Inception v3 Architecture

## 4.5 SEQUENTIAL MODELS: RNN & GRU's

The basic building blocks of recurrent models are Recurrent Neural Networks (RNNs) and Gated Recurrent Units (GRUs) which are precisely designed for working with sequential data including time series analysis, linguistic modeling and much more. The principal difference between RNNs and feed-forward neural networks that operate with all the input data at once and have a fixed number of outputs is that RNNs act step-by-step. While processing each element of the input at each time step, an RNN then does a series of computations and makes an output, which is called the hidden state. The hidden state is then concatenated with next input to proceed the same process which makes the RNN deal with input sequence of any size.

The primary problem that arises when using RNNs is that they have a hard time learning long-term dependencies in sequences because it can be problematic when gradients either vanish or explode. To address this problem, a new form of RNN has been developed known as Gated Recurrent Unit (GRU) developed by Cho et al in 2014. GRUs are similar to RNNs but contain gating mechanisms that controls the transfer of information between cells. The architecture of a GRU allows it to decide what to remember and what to forget within the series, making it easier for the model to get a long-distance view of what is happening rather than what is just happening in the near environment, which is what an ordinary RNN would do. GRUs accomplish this with update and reset gates which determine how much of the prior hidden state as well as new input is incorporated. This lead to better performance in cases when in the sequence there could be intervals with important information several points in time apart.

In general, GRUs are promising as they overcome some of the major issues of RNNs such as, gradient vanishing, giving better results in cases involving significant data sets or sequence problems.

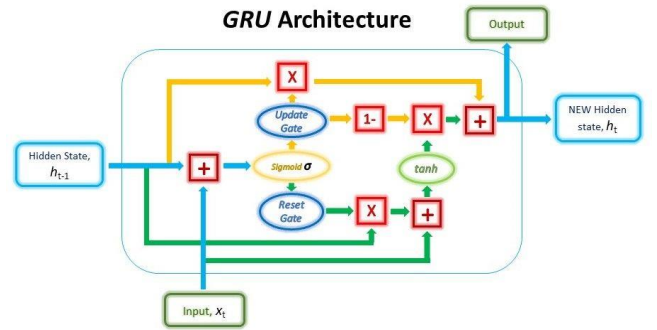
$$gate_{reset} = \sigma(W_{input_{reset}} * X_t + W_{hidden_{reset}} * h_{t-1}) \quad (1)$$

$$r = \tanh(gate_{reset} \oplus (W_{h1} * h_{t-1}) + W_{X1} * X_t) \quad (2)$$

$$gate_{update} = \sigma(W_{input_{update}} * X_t + W_{hidden_{update}} * h_{t-1}) \quad (3)$$

$$u = gate_{update} \oplus h_{t-1} \quad (4)$$

$$h_t = r \oplus (1 - gate_{update}) + u \quad (5)$$

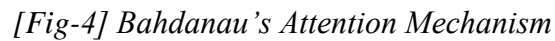


[Fig – 3] GRU Architecture

## 4.6 Attention Mechanism

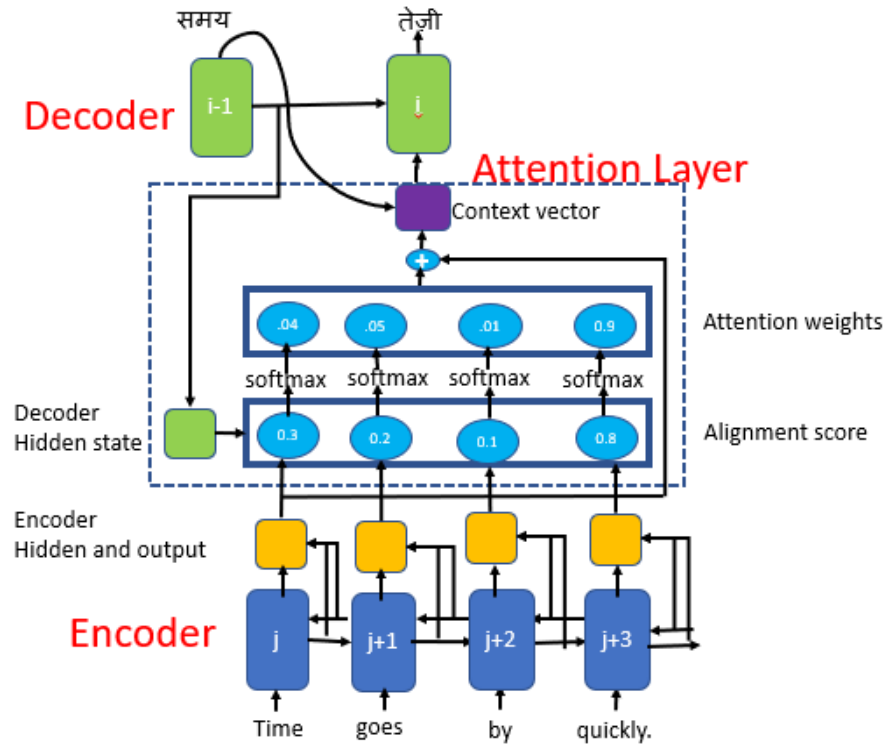
However, seq2seq model is not suitable for long sequences since the only information passed from the encoder to the decoder is the last hidden state of the RNN. Whereas, the Attention Mechanism studies in this do not suffer from this limitation as the model retains and uses all the hidden states of the input sequence during decoding. This mechanism provides a one-to-one correspondence between each output in the decoder and all the hidden states of the encoder, so that the decoder can not only have full access to the entire input sequence but can also selectively attend to the important points for the output.

Bahdanau's Attention or Additive Attention was proposed by Dzmitry Bahdanau to enhance the machine translation with sequence to sequence models. The first and foremost aim was to maintain correspondence of the decoder with critical fragments of the input sequence; in other words, the model is trained to concentrate on specific fragments of the input during the process of generating output.



The attention mechanism involves several key steps:

1. **Producing Encoder Hidden States:** The encoder encodes each element in the input sequence and produces hidden states for these elements.
2. **Calculating Alignment Scores:** The alignment scores are calculated as the dot product between the first argument, the previous decoder hidden state, and the second argument, the encoder's hidden state.
3. **Softmaxing the Alignment Scores:** The scores are then summed to produce a vector similar to the nodes to which they are compared. The weights are normalized by applying a softmax to the resulting vector.
4. **Calculating the Context Vector:** The hidden representations of the encoder are then weighted using the scores in order to produce the context vector.
5. **Decoding the Output:** The context vector is affixed with the previous decoder output and is fed into the Decoder RNN for the next step.
6. **Repeating the Process:** 2-5 are then iteratively traversed within each time-step of the decoder until the final output is generated or till the model reaches some pre-defined maximum length.



[Fig – 5] Attention Mechanism

## 4.7 Dataset Description:

### 1. Flickr8k Dataset:

The Flickr8k dataset consists of approximately 8,000 images sourced from the Flickr platform, commonly used in image captioning research. The images are split into three sets:

- **Training Set:** 6,000 images
- **Development Set:** 1,000 images
- **Test Set:** 1,000 images

Each image in the dataset is annotated with five distinct captions, each consisting of approximately 10-20 words. These captions are diverse in their descriptions, offering a rich variety of linguistic patterns and object interactions. This makes the dataset suitable for training image captioning models that can generate varied and contextually relevant captions for unseen images.

## 2. Flickr30k Dataset:

The Flickr30k dataset is an extended version of the Flickr8k dataset, containing 31,000 images sourced from Flickr. Like Flickr8k, each image in the dataset is paired with five human-generated captions, providing a broader and more detailed range of descriptive language.

### Other Datasets for Image Captioning:

- COCO (Common Objects in Context)
- Microsoft Image Captioning (MS COCO Captions)

**Input size:** (approx) 1.0 GB

Why kaggle dataset...?

1. The dataset can be trained easily as it is small in size.
2. Data is properly labelled.
3. The dataset is available for free.

### Metrics:

Machine learning models for image captioning aim to generate the most probable caption for an image based on the testing dataset. Evaluating the accuracy and quality of generated captions requires reliable metrics. Among the commonly used metrics, the **Bilingual Evaluation Understudy (BLEU)** is one of the simplest and most well-known. BLEU measures the quality of machine-generated text by comparing it to human-provided reference captions.

The BLEU score builds on the concept of **precision**, which calculates how many machine-generated n-grams match the ground truth n-grams. However, BLEU adjusts precision by limiting the count of repeated n-grams to their maximum occurrence in the reference captions. For instance, the repetitive phrase "the the the" would achieve perfect precision but not a perfect BLEU score due to this adjustment. BLEU evaluates matches across multiple n-gram orders, such as unigrams (BLEU-1), bigrams (BLEU-2), and trigrams (BLEU-3), with configurable weights for each.

## **5.DESIGN REQUIREMENT ENGINEERING**

### **CONCEPT OF UML:**

Unified Modeling Language (UML) is a well-defined language used to describe software systems and their components. It is a tool that offers a series of diagrams and shapes or symbols that shows the aspects of a system, parts of it as well as the relations of the various components; used in designing, describing and sharing the structure and the docket of any software development project among the various people involved.

UML assists in detail design of complicated software systems through offering systematic and well understood babel with the developers, analysts, and the stakeholders. Thumbs up for it, as it accommodates the object-oriented ideas and is also good at practicing the number one rules of software engineering and is a good tool for planning, structuring, and organizing the software projects.

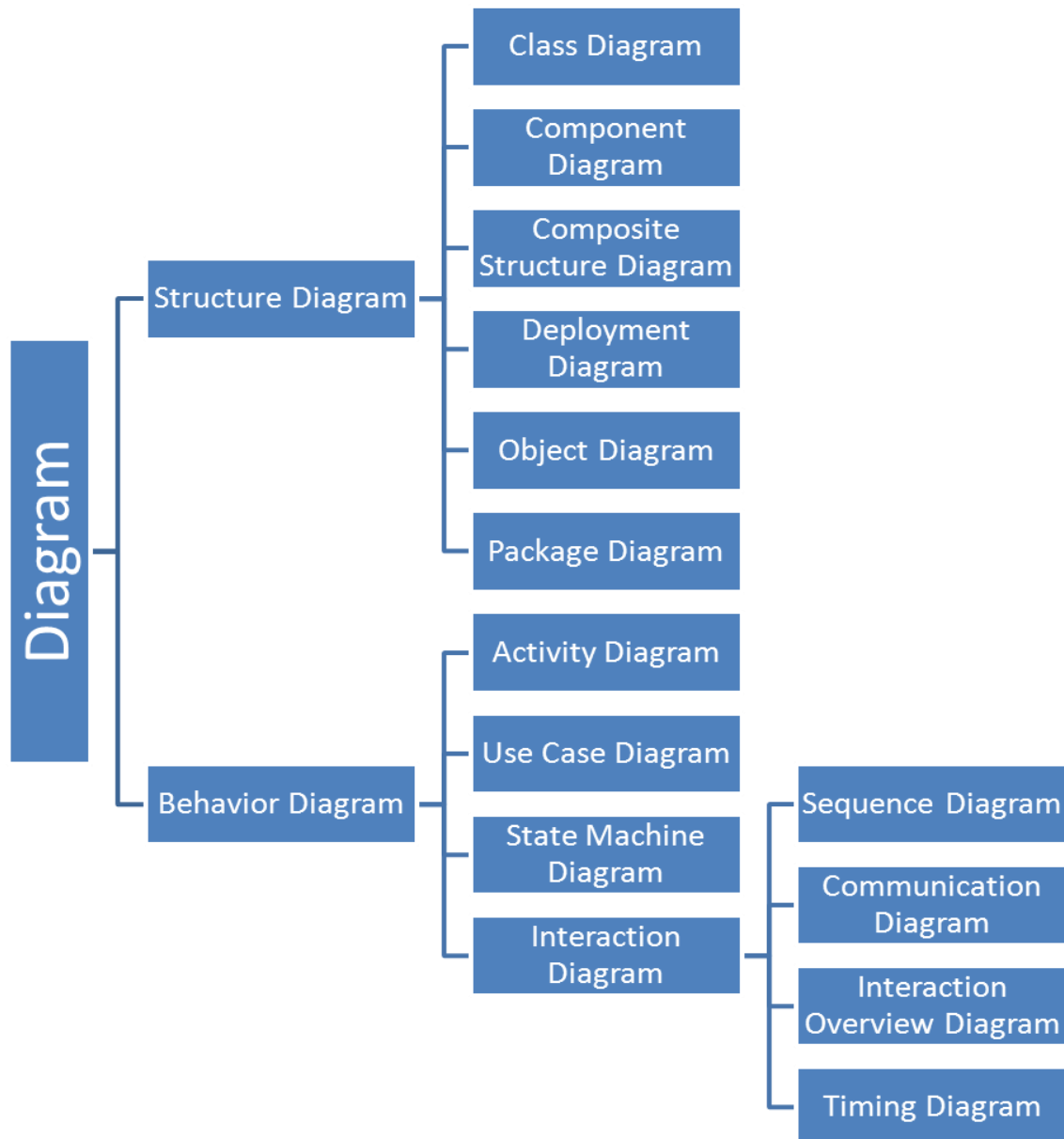
### **UML DIAGRAMS:**

UML is a modelling language that is platform agnostic and can be used with any programming language and with any sort of development methodology and because it is a standard most software developers will be familiar with it. That is to say, although many engineers may not prefer the use of diagrams, they are useful in an Agile development environment as they gradually add value while contributing positively to the flow and pace of work. It will be helpful if you treat UML diagrams not only as additional, and rather fancy, artifacts but as part of documentation. UML diagrams can assist engineering teams in several ways: UML diagrams can assist engineering teams in several ways:

- To welcome a new member on the team or an experienced developer, who joined a new team at the company.
- How to effectively behave for source code.
- This involves coming up with new features in a project before the programming phase of the project is started.
- Enabling easier understanding of the information where there may be an interaction with both the technical and non-technical observers.



- Diagrams in UML can be broadly classified...

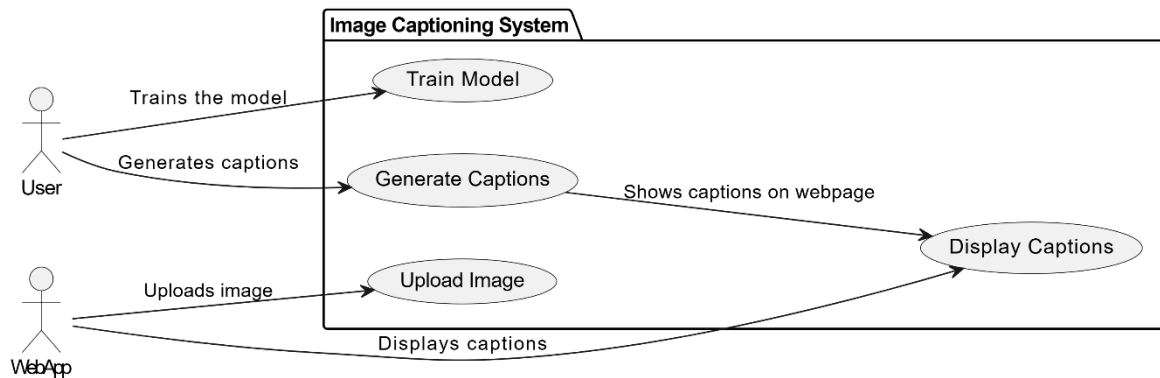


*[Fig-6] Concepts of UML*

## 5.1 Use case Diagram:

The use case diagrams create a picture view of the actors active in software program tool.

As structures that show device skills, they provide help builders contemplate how use instances relate to personas and define what a device is expected to do.

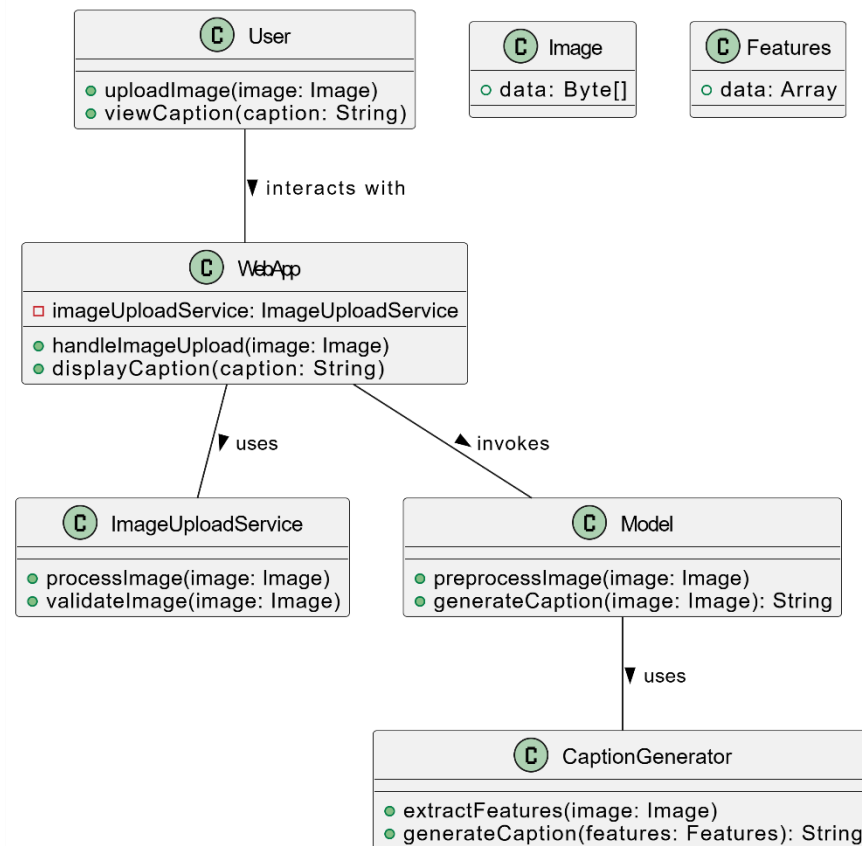


[Fig - 7] Use Case Diagram

The **Use Case Diagram** visualizes the interactions between the actors and the main functionalities of the **Image Captioning System**. The primary actors are the *User* and the *WebApp*. The User can interact with three key use cases: **Train Model**, **Generate Captions**, and **Upload Image**. The WebApp serves as an intermediary between the User and the system. When the User trains the model, it adjusts the underlying algorithms to improve caption generation. When the User uploads an image, the WebApp facilitates its processing, and the system generates captions based on the uploaded content. Finally, captions are displayed to the User. This diagram highlights the roles and actions, making it clear how the system functions from a user's perspective.

## 5.2 Class Diagram:

A UML elegance diagram is an element of a fundamental schema of any object-orientated option. This sketch represents a rigid thing-orientated machine, which defines initiatives supported by using schooling, properties and procedures.

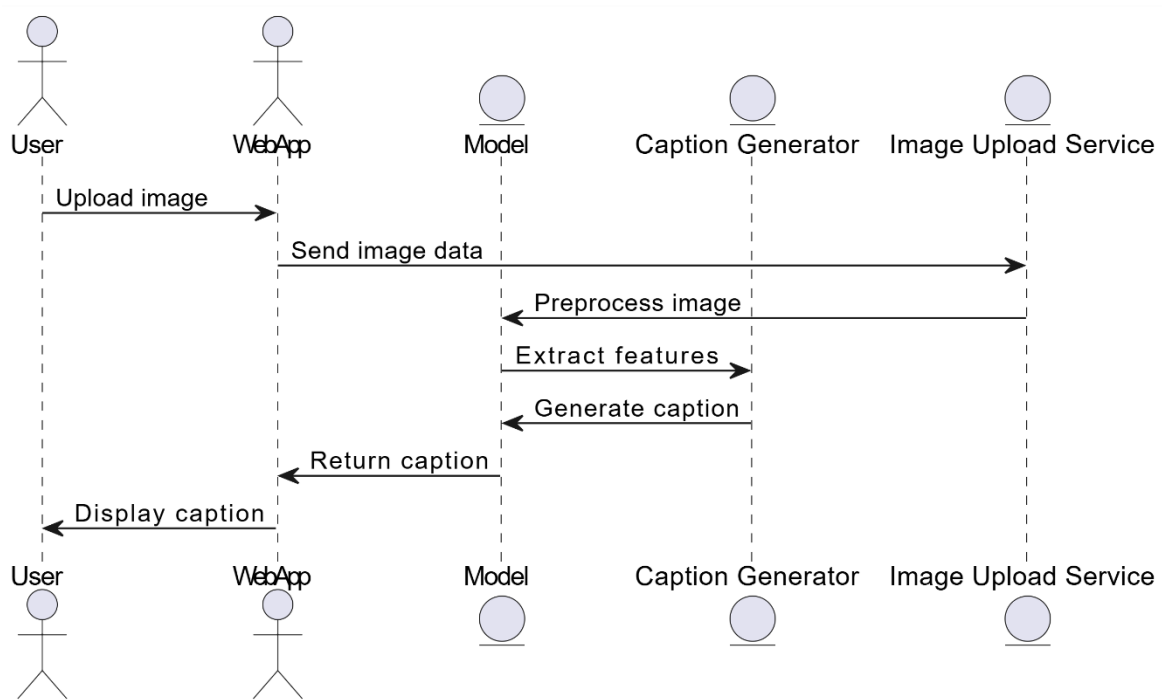


[Fig – 8] Class Diagram

The **Class Diagram** presents the structural components of the system and their relationships. It includes five key classes: User, WebApp, Image, Model, and CaptionGenerator. The User interacts with the system by uploading images and viewing captions. The WebApp uses the ImageUploadService to process and validate the uploaded images. The Model handles image preprocessing and caption generation. The CaptionGenerator works within the Model to extract image features and generate captions from these features. The Image class contains the image data, while the Features class stores processed feature data. The diagram shows how methods like uploadImage, processImage, generateCaption, and extractFeatures connect the system's components, providing a static, structural perspective of the system.

### 5.3 Sequence Diagram:

Another of the UML diagrams of collection is the collection sequence diagrams also frequently known as the occurrence diagrams that present the sequential flow of your gadgets. This includes the very threads of your character pieces and the schemes and dialogues of characters with your gizmos along with the conversations amongst these gizmos to fulfil a particular purpose.

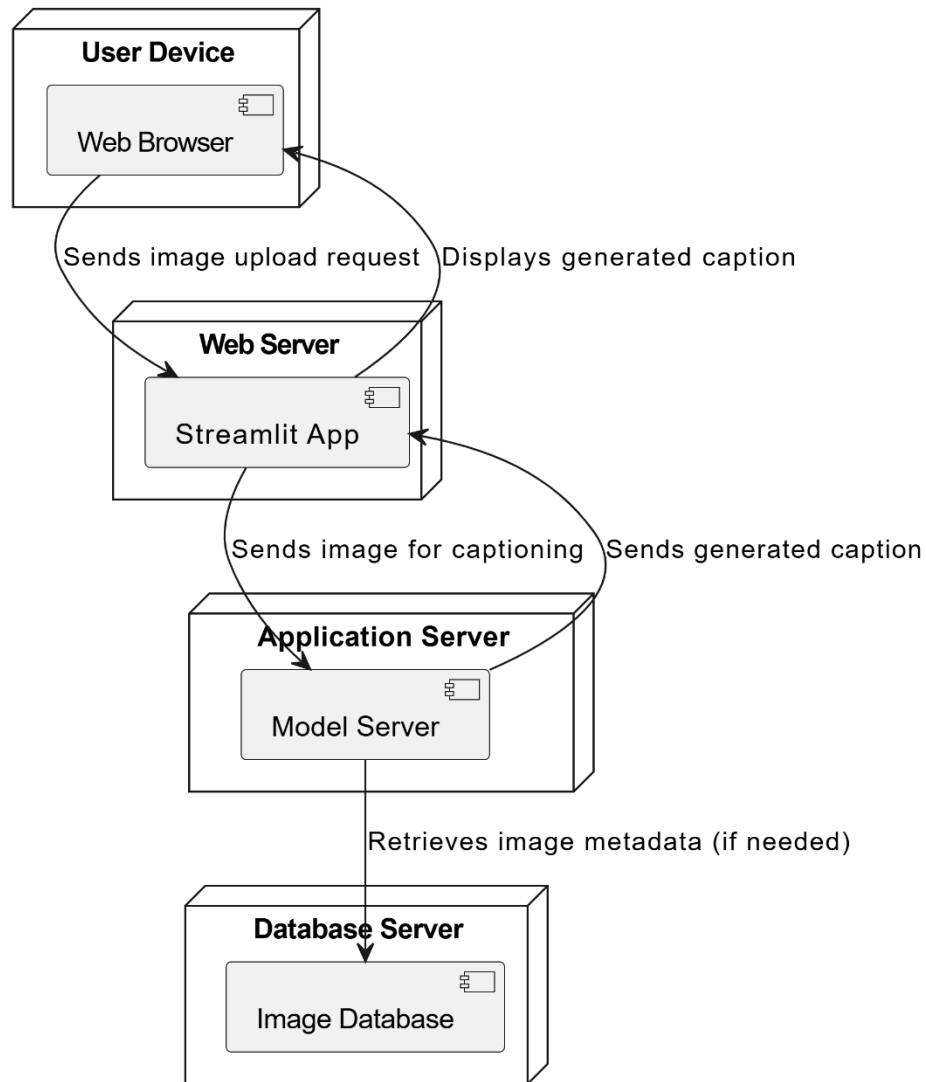


[Fig – 9] Sequence Diagram

The **Sequence Diagram** outlines the step-by-step flow of interactions in the image captioning process. The process starts with the *User* uploading an image via the WebApp. The WebApp sends the image data to the *Model*, which preprocesses the image. Next, the *Caption Generator* extracts features from the image and creates a caption based on the extracted information. The generated caption is then returned to the WebApp, which displays it to the User. This diagram illustrates the chronological order of operations and the interplay between system components such as the Model, Caption Generator, and Image Upload Service, providing a dynamic view of the system's functionality.

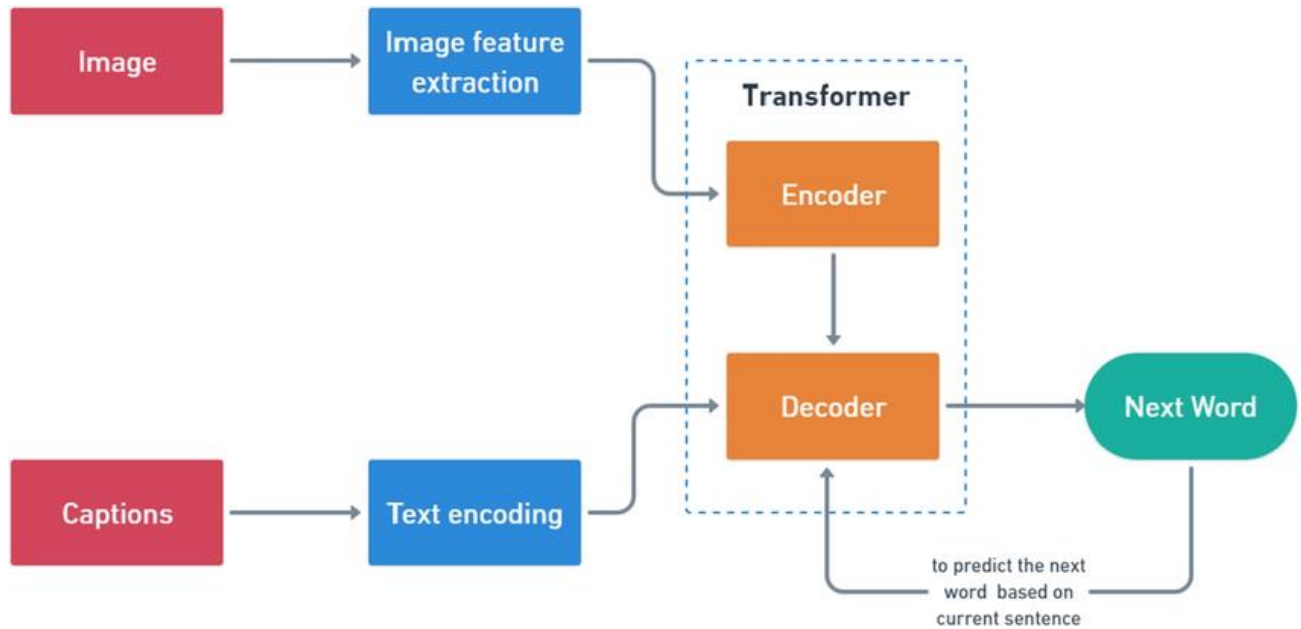
## 5.4 Deployment Diagram:

A deployment diagram show the envisioned distribution of nodes and all the additions that remain there on. In its simplest form, it portrays how the nodes in a distributed environment are joined physically. These diagrams are mainly helpful during the time of designing software that should effectively work in distinct physical platforms.



*[Fig – 10] Deployment Diagram*

## 5.5 System Architecture Diagram:



[Fig – 11] System Architecture

In this model, the interaction begins with the user inputting an image, which is processed through the **Convolutional Neural Network (CNN)** encoder. The CNN, such as a pre-trained Inception-V3, extracts key visual features from the image, converting it into a fixed-size feature vector that captures its essential elements. These feature vectors are passed to the decoder, which is built using a **Recurrent Neural Network (RNN)** or **Gated Recurrent Unit (GRU)**.

The decoder generates a sequence of words to form captions by analyzing the encoded features. To improve accuracy, the model incorporates an **Attention Mechanism**, which dynamically focuses on relevant regions of the image during caption generation, ensuring context-aware descriptions. Transfer Learning is utilized, leveraging the pre-trained CNN to enhance performance and reduce computational requirements.

Finally, the generated captions are evaluated using **BLEU scores**, ensuring quality and alignment with human-like descriptions. This end-to-end system efficiently bridges visual input and textual output.

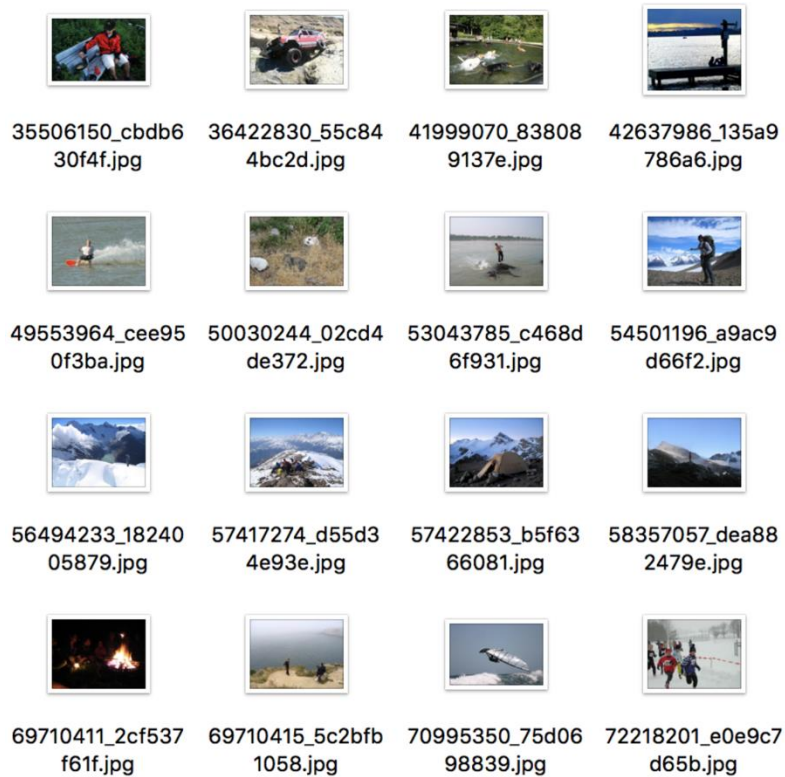
## 6. IMPLEMENTATION

### Sample data for the Problem Statement:

Image captioning is a challenging task that combines computer vision and natural language processing to generate descriptive captions for images. The Flickr8k dataset, consisting of 8,000 images with five captions per image, presents a unique opportunity to create a model capable of generating human-like textual descriptions. However, this task is complex due to several factors.

First, the diverse range of objects, actions, and contexts depicted in the images makes accurate representation difficult. Images often contain multiple interacting elements, and capturing these interactions is critical for meaningful captioning. Additionally, variations in lighting, angles, and backgrounds add further complexity.

The challenge lies in designing a model that can effectively extract visual features, comprehend the image context, and generate coherent captions that align with human perception. Addressing this problem requires a robust and scalable model that bridges the gap between visual understanding and natural language generation, ensuring accuracy and fluency in descriptions.



[Fig – 12] Sample Flickr8k Data

## Model Construction

### Mounting Google Drive , Importing Modules and downloaded Packages

To utilize a module's features, you need to first import the module with an import declaration. The import key-word is accompanied by the module's call in an import declaration. In a Python file, this will be declared on the pinnacle of the code, beneath any shebang strains or standard remarks.

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

Mounted at /content/drive
```

*[Fig – 13] Mounting Google drive*

```
[ ] import matplotlib.pyplot as plt
    import time
    import tensorflow as tf
    from tqdm import tqdm
    from glob import glob
    from PIL import Image
    import nltk
    import string
    import warnings
    import os
    from tensorflow.keras.preprocessing.text import Tokenizer
    from tensorflow.keras.preprocessing.sequence import pad_sequences
    import numpy as np
    from tensorflow.compat.v1 import ConfigProto
    from tensorflow.compat.v1 import InteractiveSession
```

*[Fig – 14] Importinig packages*

### Defining File Paths:

```
[ ] # Dir which contains all the images.
    IMG_FILE_PATH = "/content/drive/MyDrive/archive/Images/"

[ ] # txt file containg the unprocessed captions
    CAP_TEXT_PATH = "/content/drive/MyDrive/Flickr8k_text/Flickr8k.token.txt"
    # txt file containing the names of the train imgs
    TRAIN_TXT_PATH = "/content/drive/MyDrive/Flickr8k_text/Flickr_8k.trainImages.txt"
    # txt file containing the names of the test imgs
    TEST_TXT_PATH = "/content/drive/MyDrive/Flickr8k_text/Flickr_8k.testImages.txt"
    # txt file containing the names of the validation imgs
    VAL_TXT_PATH = "/content/drive/MyDrive/Flickr8k_text/Flickr_8k.devImages.txt"

    # Text file path
    TXT_PATH = "/content/drive/MyDrive/Flickr8k_text/Flickr_8k.trainImages.txt"
```

*[Fig – 15] Path Setting*



## Preprocessing Data:

The preprocessing steps in the image captioning model are crucial for preparing the dataset and ensuring compatibility with the model. For image processing, each image from the Flickr8k dataset is resized to a fixed dimension, typically 299x299 pixels, to align with the input requirements of the InceptionV3 model. Pixel values are normalized to improve training stability. Textual preprocessing involves tokenizing the captions, converting words to lowercase, and removing punctuation to standardize the input. Vocabulary is built by mapping unique words to indices, and captions are padded to a fixed length for uniformity. Finally, images and captions are paired for training.

```
def load_cap(caption_txt_path: str) -> dict:
    """
    To read the text file containing captions and store it in a dict.
    mapping(dict) contains image_name as key and the corresponding captions
    as a list of list of words.
    :param caption_txt_path:
    :return: mapping(dict) contains image_name as key and the
    corresponding captions as a list of captions
    """
    # opening the text file containing the captions
    with open(caption_txt_path, 'r', encoding='utf-8') as caption_txt:
        # each line in the txt file is stored in captions_list(list)
        captions_list = caption_txt.readlines()
    # initializing the dict
    mapping = dict()
    # each line in the text file
    for line in captions_list:
        # split the text by \t
        caption = line.split('\t')
        # the first item of the list
        # -2 ignoring the last two characters and removing the extension
        image_name = caption[0][: -2].split('.')[0]
        # cleaning up the caption(-1 for removing the \n)
        image_caption = clean_cap(caption[-1][: -1])
        # adding start and end of seq
        image_caption = ('startofseq ' + image_caption + ' endofseq')
        # appending the caption with the name as key
        if image_name in mapping:
            mapping[image_name].append(image_caption)
        # if the image_name key is new then creating a key value
        else:
            mapping[image_name] = [image_caption]
    return mapping
```

```

def clean_cap(caption: str) -> str:
    """
    preprocessing the caption
    :param caption: unprocessed caption
    :return: processed caption
    """
    # Removes Punctuations
    cap = ''.join([ch for ch in caption if ch not in string.punctuation])
    # tokenize
    cap = cap.split()
    # convert to lower case
    cap = [word.casefold() for word in cap]
    # remove hanging 's' and 'a'
    cap = [word for word in cap if len(word) > 1]
    # remove tokens with numbers in them
    cap = [word for word in cap if word.isalpha()]
    # Lemmatizing
    lemmatizer = nltk.WordNetLemmatizer()
    cap = [lemmatizer.lemmatize(word) for word in cap]
    # store as string
    return ' '.join(cap)

```

```

def path_cap_list(img_names_set: set, tokenizer: Tokenizer, captions_dict) -> (list, list):
    """
    a list of image paths and a list of captions for images with corresponding values
    Note: the captions will be tokenized and padded in this function
    :param img_names_set: The set on which the processing is done
    :param tokenizer: tokenizer
    :param captions_dict: clean captions for that set without any tokenization
    """
    tokenized_caps_dict = tokenize_cap(tokenizer, captions_dict)
    image_name_list = sorted(img_names_set)
    capt_list = [cap for name in image_name_list for cap in tokenized_caps_dict[name]]
    img_path_list = [img_name_2_path(name) for name in image_name_list for i in range(len(tokenized_caps_dict[name]))]
    return img_path_list, capt_list

```

```

def load_npy(image_path: str, cap: str) -> (str, str):
    """
    :returns image tensor vector with the image path
    :param image_path:
    :param cap:
    """
    img_tensor = np.load(image_path.decode('utf-8') + '.npy')
    return img_tensor, cap

```

```

def create_dataset(img_path_list: str, cap_list: str) -> object:
    """
    :param img_path_list: The ordered list of img paths with duplication acc to number of captions
    :param cap_list: the padded caption list with the curr order
    :return: dataset
    """
    dataset = tf.data.Dataset.from_tensor_slices((img_path_list, cap_list))
    # Use map to load the numpy files in parallel
    dataset = dataset.map(lambda item1, item2: tf.numpy_function(load_npy, [item1, item2], [tf.float32, tf.int32]),
                        num_parallel_calls=tf.data.experimental.AUTOTUNE)
    # Shuffle and batch
    dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE).prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
    return dataset

```

[Fig – 16] Preprocessing steps

## Defining the Model:

```
class BahdanauAttention(tf.keras.Model):
    def __init__(self, units):
        super(BahdanauAttention, self).__init__()
        self.W1 = tf.keras.layers.Dense(units)
        self.W2 = tf.keras.layers.Dense(units)
        self.V = tf.keras.layers.Dense(1)

    def call(self, features, hidden):
        # features(CNN_encoder output) shape == (batch_size, 64, embedding_dim)

        # hidden shape == (batch_size, hidden_size)
        # hidden_with_time_axis shape == (batch_size, 1, hidden_size)
        hidden_with_time_axis = tf.expand_dims(hidden, 1)

        # attention_hidden_layer shape == (batch_size, 64, units)
        attention_hidden_layer = (tf.nn.tanh(self.W1(features) +
                                             self.W2(hidden_with_time_axis)))

        # score shape == (batch_size, 64, 1)
        # This gives you an unnormalized score for each image feature.
        score = self.V(attention_hidden_layer)

        # attention_weights shape == (batch_size, 64, 1)
        attention_weights = tf.nn.softmax(score, axis=1)

        # context_vector shape after sum == (batch_size, hidden_size)
        context_vector = attention_weights * features
        context_vector = tf.reduce_sum(context_vector, axis=1)

        return context_vector, attention_weights
```

```
class CNN_Encoder(tf.keras.Model):
    # Since you have already extracted the features and dumped it
    # This encoder passes those features through a Fully connected layer
    def __init__(self, embedding_dim):
        super(CNN_Encoder, self).__init__()
        # shape after fc == (batch_size, 49, embedding_dim)
        self.fc = tf.keras.layers.Dense(embedding_dim)

    def call(self, x):
        x = self.fc(x)
        x = tf.nn.relu(x)
        return x
```

```
def rnn_type(units):
    # If you have a GPU, we recommend using CuDNNNGRU(provides a 3x speedup than GRU)
    # the code automatically does that.
    if tf.test.is_gpu_available():
        return tf.compat.v1.keras.layers.CuDNNLSTM(units,
                                                    return_sequences=True,
                                                    return_state=True,
                                                    recurrent_initializer='glorot_uniform')
    else:
        return tf.keras.layers.GRU(units,
                                    return_sequences=True,
                                    return_state=True,
                                    recurrent_activation='sigmoid',
                                    recurrent_initializer='glorot_uniform')
```

[Fig – 17]Attention, Encoder model functions

```

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

        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        # self.gru = rnn_type(self.units)
        self.gru = tf.keras.layers.GRU(self.units,
                                       return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')

        self.fc1 = tf.keras.layers.Dense(self.units)
        self.fc2 = tf.keras.layers.Dense(vocab_size)

        self.attention = BahdanauAttention(self.units)

    def call(self, x, features, hidden):
        # defining attention as a separate model
        context_vector, attention_weights = self.attention(features, hidden)

        # x shape after passing through embedding == (batch_size, 1, embedding_dim)
        x = self.embedding(x)

        # x shape after concatenation == (batch_size, 1, embedding_dim + hidden_size)
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

        # passing the concatenated vector to the GRU
        output, state = self.gru(x)

        # shape == (batch_size, MAX_CAP_LEN, hidden_size)
        x = self.fc1(output)

        # x shape == (batch_size * MAX_CAP_LEN, hidden_size)
        x = tf.reshape(x, (-1, x.shape[2]))

        # output shape == (batch_size * MAX_CAP_LEN, vocab)
        x = self.fc2(x)

        return x, state, attention_weights

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

```

*[Fig -18] Decoder model function*

## Training the model:

```
@tf.function
def train_step(img_tensor, target):
    loss = 0

    # initializing the hidden state for each batch
    # because the captions are not related from image to image
    hidden = decoder.reset_state(batch_size=target.shape[0])

    dec_input = tf.expand_dims([tokenizer.word_index['startofseq']] * target.shape[0], 1)

    with tf.GradientTape() as tape:
        features = encoder(img_tensor)

        for i in range(1, target.shape[1]):
            # passing the features through the decoder
            predictions, hidden, _ = decoder(dec_input, features, hidden)

            loss += loss_function(target[:, i], predictions)

            # using teacher forcing
            dec_input = tf.expand_dims(target[:, i], 1)

    total_loss = (loss / int(target.shape[1]))

    trainable_variables = encoder.trainable_variables + decoder.trainable_variables
    gradients = tape.gradient(loss, trainable_variables)
    optimizer.apply_gradients(zip(gradients, trainable_variables))

    return loss, total_loss
```

*[Fig – 19] Training the model*

## Saving Weights

```
if os.path.isfile("Encoder_train_1_8k.h5") is False:
    encoder.save_weights("Encoder_train_1_8k.h5")
    print("Encoder saved")
```

```
if os.path.isfile("Decoder_train_1_8k.h5") is False:
    decoder.save_weights("Decoder_train_1_8k.h5")
    print("Decoder saved")
```

*[Fig – 20] Saving the model*

## Plotting:

```
EPOCHS = 20

for epoch in range(start_epoch, EPOCHS):
    start = time.time()
    total_loss = 0

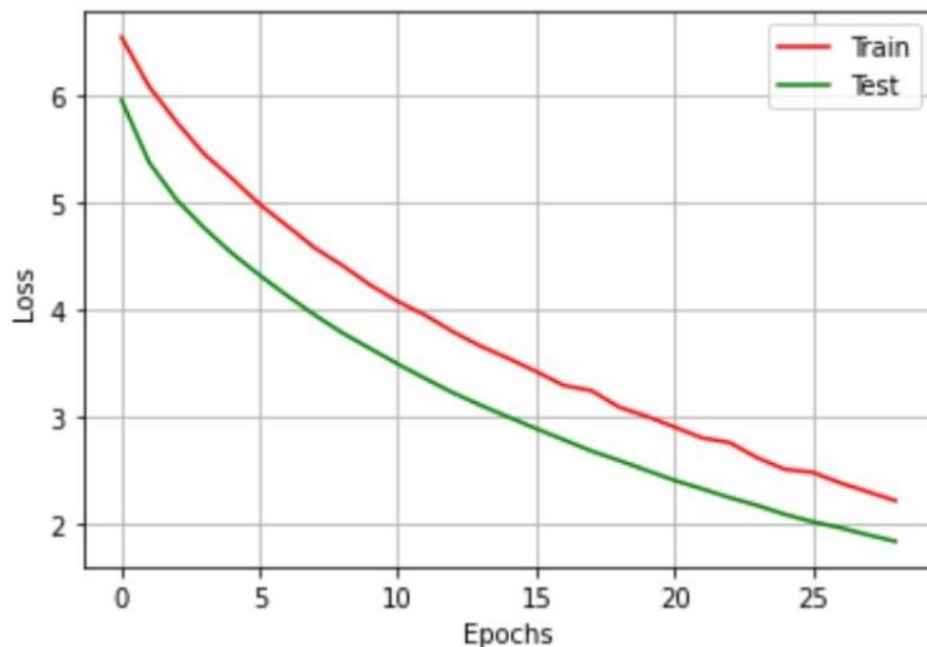
    for (batch, (img_tensor, target)) in enumerate(train_dataset):
        batch_loss, t_loss = train_step(img_tensor, target)
        total_loss += t_loss

        if batch % 100 == 0:
            average_batch_loss = batch_loss.numpy()/int(target.shape[1])
            print(f'Epoch {epoch+1} Batch {batch} Loss {average_batch_loss:.4f}')
    # storing the epoch end loss value to plot later
    loss_plot.append(total_loss / num_steps)

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

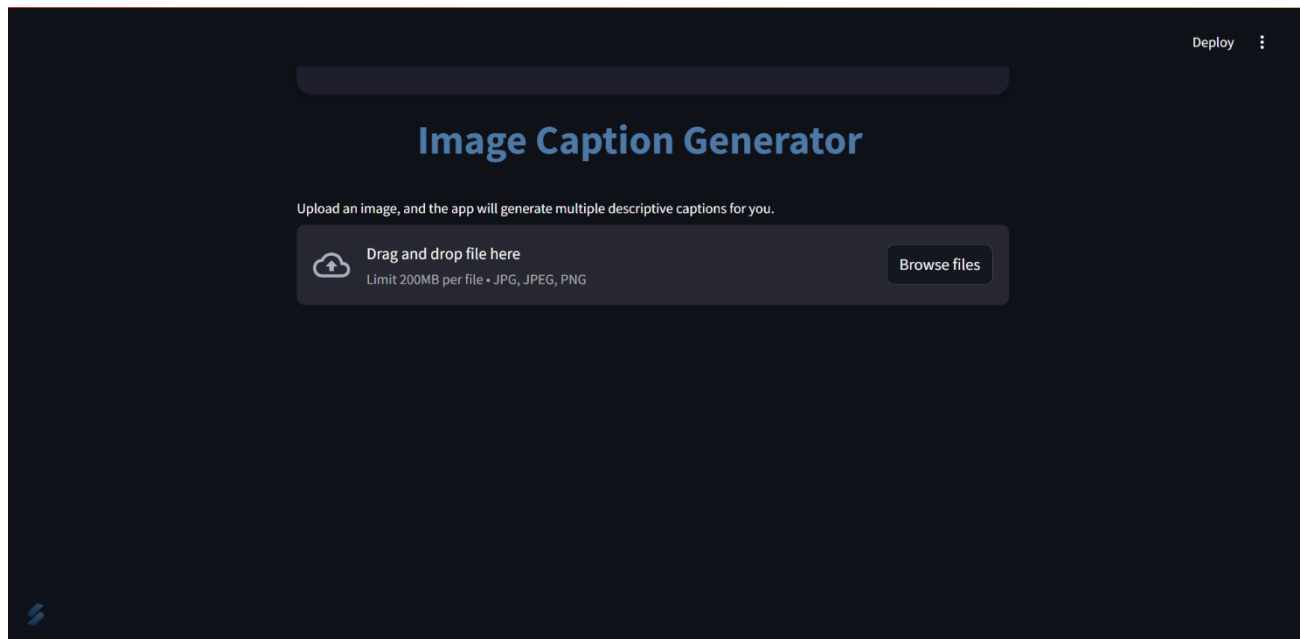
print(f'Epoch {epoch+1} Loss {total_loss/num_steps:.6f}')
print(f'Time taken for 1 epoch {time.time()-start:.2f} sec\n')
```

```
plt.plot(loss_plot)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Plot')
plt.show()
```



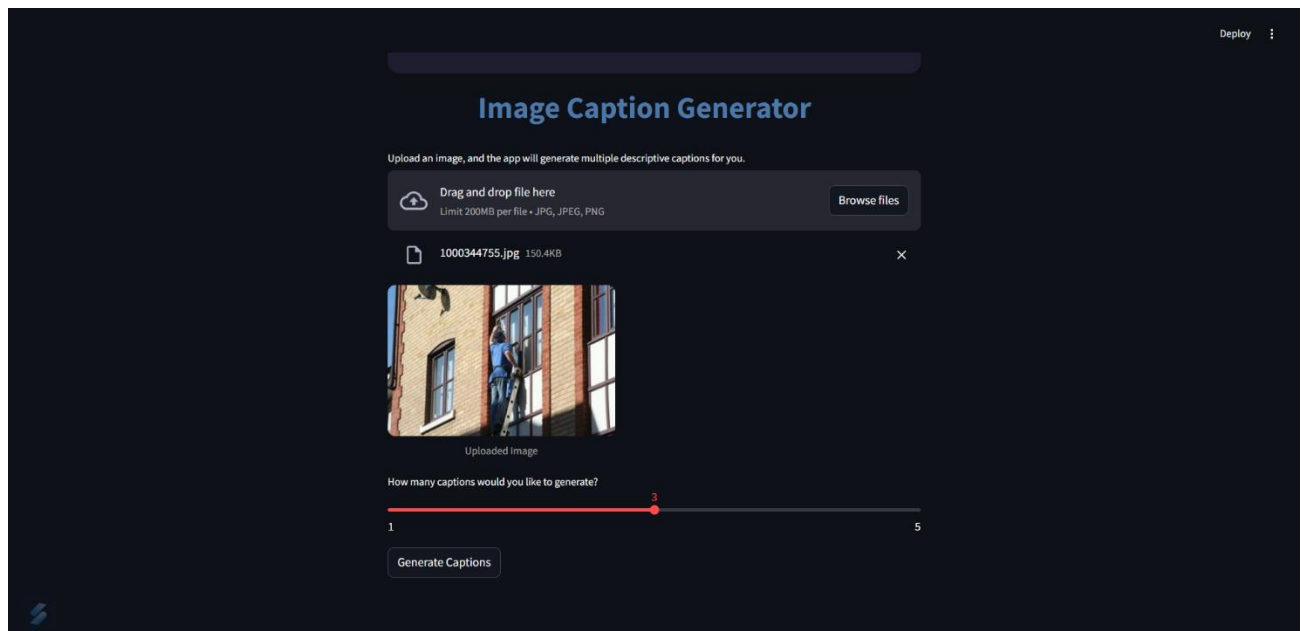
[Fig – 21] Loss vs Epochs graph

## Home page:



[Fig – 22]

## Upload image:




[Fig – 23]


## Generate Captions:


### Image Caption Generator

Upload an image, and the app will generate multiple descriptive captions for you.

 Drag and drop file here  
Limit 200MB per file • JPG, JPEG, PNG

Browse files

1000919630.jpg 114.4KB 



Uploaded Image

How many captions would you like to generate?

1

3

5

Generate Captions

**Generated Captions:**

1. a man holding a teddy bear next to a teddy bear
2. a man holding a teddy bear next to a stuffed animal
3. a man holding a teddy bear while sitting on a chair

[Fig – 24]



## **7. SOFTWARE TESTING**

Software testing is a testing that is done before actual software is completely executed. The main objective for doing the software testing is the requirements of the expected output is free from errors and defects.

### **7.1 Unit Testing:**

Unit testing for the brain stroke prediction project ensures that components such as data preprocessing, feature extraction, data splitting, and model training work correctly. It also verifies such important aspects as dealing with missing values, proper feature selection, accurate training-testing data splits, and correct functioning of models like Random Forest and Logistic Regression. Each part is confirmed to be working properly.

### **7.2 Integration Testing:**

Integration testing checks if the modules of a project, such as data preprocessing, feature extraction, model training and prediction, work together nicely. It tests the flow of data and inter-component interactions in order to search for issues in their integration. For instance, after preprocessing, processed data should pass correctly through the feature extraction module without losing the most important information. Likewise, the features extracted should be integrated into the models without any errors for training and testing. Integration tests also check that the prediction by the trained model matches the expected outcome when unseen data is provided. This will ensure that all parts go along harmoniously from the input raw data to the stroke risk prediction.

### **7.3 Performance Testing:**

The test evaluates the performance of the system regarding efficiency and speed under diverse workloads to ensure satisfactory performance results for meeting expectations. In the case of the brain

stroke prediction project, this means testing how fast the data preprocessing will handle large datasets, ensuring that feature extraction and data splitting can be scaled without bottlenecks. The execution time, memory usage, and computational efficiency of machine learning models, like Random Forest and Logistic Regression, are tested in the training and prediction phases. Stress tests can be done by increasing the dataset size for reliability checkup and system response time, thereby testing and ensuring it runs efficiently both with normal and heavy loads for quick and accurate predictions.

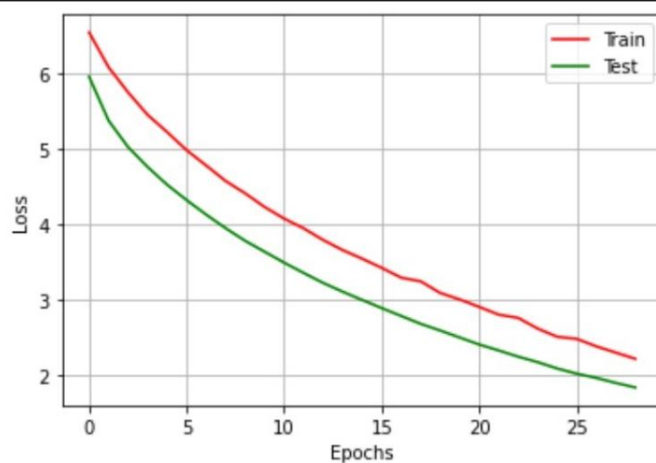
#### **7.4 Model Testing:**

Model testing: This aims at evaluating the machine learning models used in the project about their accuracy, reliability, and robustness. It evaluates the validation of models like Random Forest, Logistic Regression, among others, against test data while checking prediction accuracy, precision, recall, and F1-score. Tests check if the models generalize well to data that has not been seen before and avoid overfitting. For example, edge cases-when input values could be missing or extreme-and the model handled them nicely were tested. Comparison tests to identify the best-performing one were conducted between different models. This assures that the chosen model consistently and reliably predicts strokes.

#### **7.5 Validation Testing:**

Validation testing ensures that the final machine learning model meets the project's requirements to provide accurate and reliable predictions. It tests the model on a validation dataset not used in either training or testing phases to confirm the model's generalization capabilities. To validate performance, metrics like accuracy, precision, recall, and F1-score are used. The testing checks that the model will give appropriate stroke risk predictions for new data, as well as edge-case functionality such as detecting missing or outlier values. It also verifies that the entire end-to-end pipeline-from input data to prediction-is working correctly. This step ensures that the model is ready for release.

## 8. RESULTS

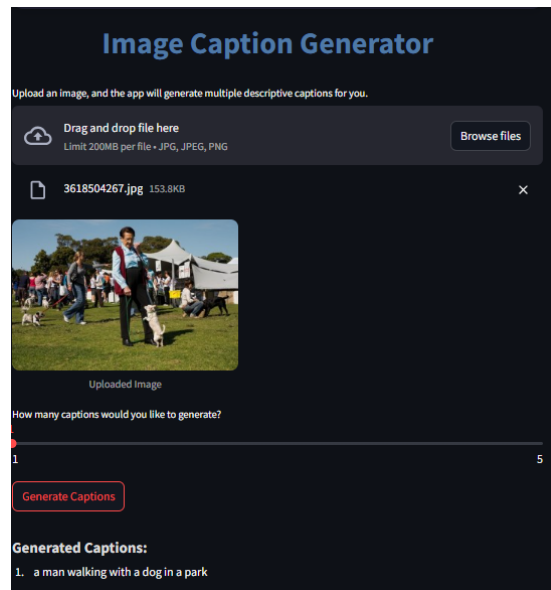


From the graph, we can approximate the values for **training loss** and **validation (test) loss** at the specified number of epochs (20). Here's the observation:

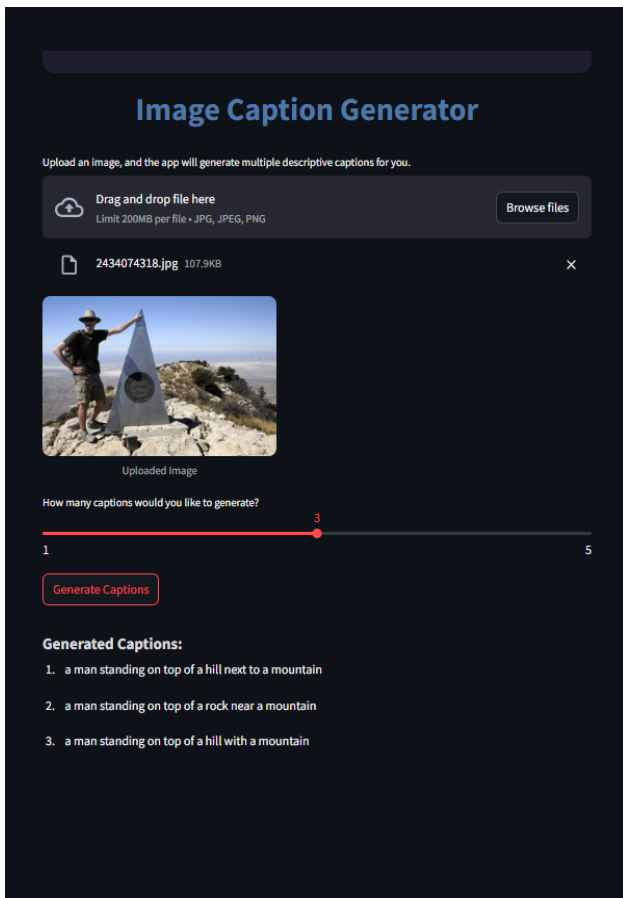
- At epoch 20:
  - **Training loss** (red curve): Approximately **2.8**.
  - **Validation (test) loss** (green curve): Approximately **2.3**.

These values are estimates based on the plotted graph. If you have access to the raw data, that would provide exact values.

## TEST CASE -1:

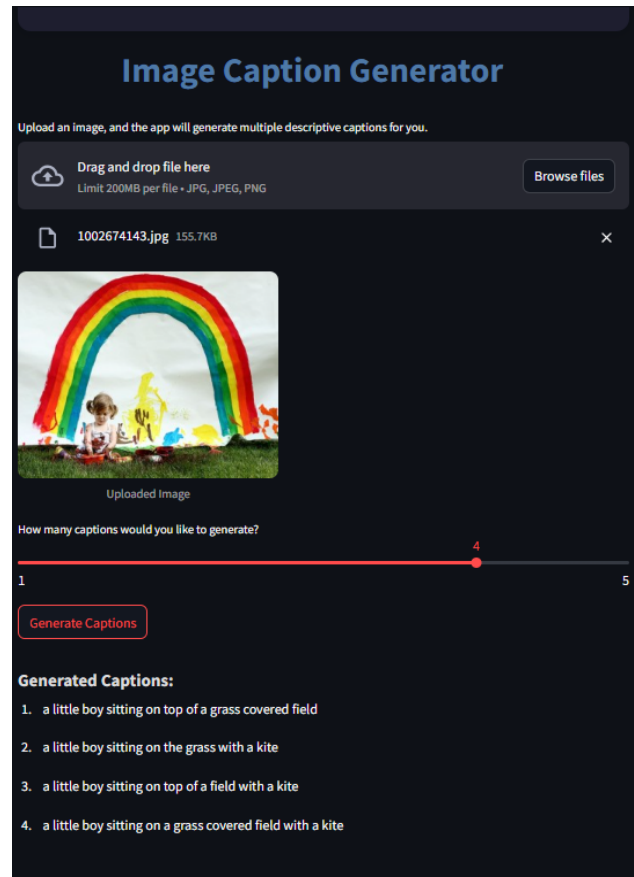


## TEST CASE – 2:



[Fig – 26]

## TEST CASE – 3:



[Fig – 27]

## 9. CONCLUSION AND FUTURE ENHANCEMENTS

In this project, an image captioning model was successfully developed using a combination of Convolutional Neural Networks (CNNs) for feature extraction and Recurrent Neural Networks (RNNs) for sequence generation. By leveraging the Flickr8k dataset and transfer learning through InceptionV3, the model demonstrated its ability to generate meaningful captions for diverse images. Attention mechanisms played a critical role in enhancing the model's focus on relevant image regions, thereby improving the contextual accuracy of captions. BLEU metrics were used to evaluate the model, showcasing satisfactory results for unigram and bigram precision. While the model performed well for straightforward scenes, it faced challenges in handling complex and abstract imagery. Overall, this project highlights the effectiveness of combining deep learning architectures and transfer learning for image captioning, paving the way for further advancements in the field.

### 9.1 Future Enhancements

- **Dataset Expansion:** Use larger datasets like MS COCO or mixed datasets for diverse caption generation.
- **Transformer Models:** Integrate transformers like ViT and BERT for better accuracy and parallel processing.
- **Multilingual Support:** Extend the model to generate captions in multiple languages.
- **Real-Time Captioning:** Implement the model for real-time applications such as assistive technologies.
- **Edge Deployment:** Optimize the model for low-power devices using techniques like quantization.
- **Context Understanding:** Enhance the model's ability to understand abstract or high-level concepts in images.
- **Evaluation Metrics:** Include additional metrics like CIDEr and METEOR for comprehensive performance evaluation.

## 10. BIBLIOGRAPHY

- [1] Long-term Recurrent Convolutional Networks for Visual Recognition and Description;  
Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan,  
Sergio Guadarrama, Kate Saenko, Trevor Darrell
  
- [2] Deep Visual-Semantic Alignments for Generating Image Descriptions;  
Karpathy A, Fei-Fei L
  
- [3] Aligning where to see and what to tell: image caption with region-based attention and  
scene factorization; Junqi Jin, Kun Fu, Runpeng Cui, Fei Sha, Changshui Zhan
  
- [4] Video Captioning With Attention-Based LSTM and Semantic Consistency; : Lianli Gao; Zhao  
Guo; Hanwang Zhang; Xing Xu; Heng Tao Shen
  
- [5] Looking deeper and transferring attention for image captioning; : Fang Fang, Hanli  
Wang, Yihao Chen & Pengjie Tang
  
- [6] Predicting visual features from text for image and video caption retrieval; Jianfeng  
Dong; Xirong Li; Cees G. M. Snoek
  
- [7] Image captioning: from structural tetrad to translated sentences; Guo, R., Ma, S. & Han
  
- [8] VD-SAN: Visual-Densely Semantic Attention Network for Image Caption Generation;  
Xinwei He, Yang Yang , Baoguang Shi, Xiang Bai