Image Caption Generator Using Deep Learning

DISSERTATION

Submitted in partial fulfillment of the
Requirements for the award of the degree

of

Bachelor of Technology

in

Computer Science & Engineering

By:

Gurman Singh (111/CSE2/2019)
Harneet Singh (103/CSE2/2019)
Kuldeep Singh (117/CSE2/2019)
Sahibjot Singh (065/CSE2/2019)

Under the guidance of:

Ms. Jyotsna

Department of Computer Science & Engineering
Guru Tegh Bahadur Institute of Technology

Guru Gobind Singh Indraprastha University
Dwarka, New Delhi
Year 2022-2023

DECLARATION

We hereby declare that all the work presented in the dissertation entitled **"Image Caption Generator using Deep Learning"** in the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in **Computer Science & Engineering,** Guru Tegh Bahadur Institute of Technology, affiliated to Guru Gobind Singh Indraprastha University Delhi is an authentic record of our work carried out under the guidance of **Ms**. **Jyotsna**

Date: 15 October 2022

Gurman Singh (111/CSE2/2019)
Harneet Singh (103/CSE2/2019)
Kuldeep Singh (117/CSE2/2019)
Sahibjot Singh (065/CSE2/2019)

CERTIFICATE

This is to certify that the dissertation entitled **"Image Caption Generator using Deep Learning",** which is submitted by **Mr. Gurman Singh, Mr. Harneet Singh, Mr. Kuldeep Singh, and Mr. Sahibjot Singh** in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in **Computer Science & Engineering**, Guru Tegh Bahadur Institute of Technology, New Delhi is an authentic record of the candidate's work carried out by them under our guidance. The matter embodied in this thesis is original and has not been submitted for the award of any other degree.

Ms. Jyotsna
(Project Guide)

Aashish Bhardwaj
(Head of Department)
Computer Science & Engineering

Date: 15 October 2022

ACKNOWLEDGEMENT

We would like to express our great gratitude towards our supervisor, **Ms. Jyotsna** who has given us support and suggestions. Without their help, we could not have presented this dissertation up to the present standard. We also take this opportunity to give thanks to all others who gave us support for the project or in other aspects of our study at Guru Tegh Bahadur Institute of Technology.

Date: 15 October 2022

Gurman Singh
(111/CSE2/2019)
gurman29singh@gmail.com

Harneet Singh
(103/CSE2/2019)
harneetsinghch110@gmail.com

Kuldeep Singh
(117/CSE2/2019)
kuldeepsingh2670@gmail.com

Sahibjot Singh
(065/CSE2/2019)
sahibjot7180@gmail.com

ABSTRACT

Generating accurate captions for an image has remained one of the major challenges in Artificial Intelligence with plenty of applications ranging from robotic vision to help the visually impaired. Long-term applications also involve providing accurate captions for videos in scenarios such as security systems. “Image caption generator”: the name itself suggests that we aim to build an optimal system that can generate semantically and grammatically accurate captions for an image. Researchers have been involved in finding an efficient way to make better predictions, therefore we have discussed a few methods to achieve good results. We have used deep neural networks and machine learning techniques to build a good model. We have used Flickr 8k dataset which contains around 8000 sample images with five captions for each
image. The Flickr8k dataset is a large dataset of images and their associated textual descriptions, which was compiled for training and evaluating image caption generation systems. The images in the dataset are diverse and cover a wide range of subjects and scenes. Each image in the dataset is accompanied by five different textual descriptions, which were generated by different annotators. The Flickr8k dataset is commonly used as a benchmark for evaluating the performance of image caption generation systems. There are two phases: feature extraction from the image using Convolutional Neural Networks (CNN) and generating sentences by tokenization for framing our sentences from the input images given. The input images are provided by the user via the user interface website, which is developed with ReactJS, Bootstrap, CSS, and HTML. FastAPI is used to connect the AI model and the front end. The final output caption is shown on the website.

CONTENTS

1. INTRODUCTION

Making a computer system detect objects and describe them using natural language processing (NLP) in an age-old problem of Artificial Intelligence. This was considered an impossible task by computer vision researchers till now. With the growing advancements in Deep learning techniques, availability of vast datasets, and computational power, models are often built which will generate captions for an image. Image caption generation is a task that involves image processing and natural language processing concepts to recognize the context of an image and describe them in a natural language like English or any other language.

While human beings are able to do it easily, it takes a strong algorithm and a lot of computational power for a computer system to do so. Many attempts have been made to simplify this problem and break it down into various simpler problems such as object detection, image classification, and text generation. A computer system takes input images as two-dimensional arrays and mapping is done from images to captions or descriptive sentences. In recent years a lot of attention has been drawn toward the task of automatically generating captions for images. However, while new datasets often spur considerable innovation, benchmark datasets also require fast, accurate, and competitive evaluation metrics to encourage rapid progress.

Being able to automatically describe the content of a picture using properly formed English sentences may be a very challenging task, but it could have an excellent impact, as an example by helping visually impaired people better understand the content of images online. This task is significantly harder, for instance than the well-studied image classification or visual perception tasks, which are a main focus within the computer vision community Deep learning methods have demonstrated advanced results on caption generation problems. What is most impressive about these methods is that one end-to-end model is often defined to predict a caption, given a photograph, rather than requiring sophisticated data preparation or a pipeline of specifically designed models.

Deep learning has attracted a lot of attention because it's particularly good at a kind of learning that has the potential to be very useful for real-world applications.

1.1 Python

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released in 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3.

The Python 2 language, i.e. Python 2.7.x, is "sunsetting" on January 1, 2020 (after extension; first planned for 2015), and the Python team of volunteers will not fix security issues or improve it in other ways after that date. With the end-of-life, only Python 3.5.x and later will be supported.

Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open-source reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

Python's large standard library[120] provides tools suited to many tasks and is commonly cited as one of its greatest strengths. For Internet-facing applications, many standard formats and protocols such as MIME and HTTP are supported. It includes modules for creating graphical user interfaces, connecting to relational databases, generating pseudorandom numbers, arithmetic with arbitrary-precision decimals,[121] manipulating regular expressions, and unit testing.

Some parts of the standard library are covered by specifications—for example, the Web Server Gateway Interface (WSGI) implementation `wsgiref` follows PEP

333[122]—but most are specified by their code, internal documentation, and test suites. However, because most of the standard library is cross-platform Python code, only a few modules need altering or rewriting for variant implementations.

As of 14 November 2022, the Python Package Index (PyPI), the official repository for third-party Python software, contains over 415,000[123] packages with a wide range of functionality, including:

- Automation
- Data Analytics
- Databases
- Documentation
- Graphical user interfaces
- Image processing
- Machine learning
- Mobile apps
- Multimedia
- Computer networking
- Scientific computing
- System Administration
- Test frameworks
- Text processing
- Web frameworks
- Web scraping

Most Python implementations (including CPython) include a read–eval–print loop (REPL), permitting them to function as a command line interpreter for which users enter statements sequentially and receive results immediately.

Python also comes with an Integrated development environment (IDE) called IDLE, which is more beginner-oriented.

Other shells, including IDLE and IPython, add further abilities such as improved auto-completion, session state retention, and syntax highlighting. As well as standard desktop integrated development environments, there are Web browser-based IDEs, including SageMath, for developing science- and math-related programs.

1.2 FastAPI

FastAPI is a modern, fast (high-performance), a web framework for building APIs with Python 3.7+ based on standard Python type hints. It is built on top of Starlette for the web parts and Pydantic for the request/response parts. One of the key features of FastAPI is its use of type hints to automatically generate web-based API documentation, as well as automatically validate incoming and outgoing data. This means that you can use FastAPI to build an API with a minimal amount of code, and with a clear separation of the documentation and the implementation.

FastAPI is also fully asynchronous, meaning that it can take advantage of the async/await syntax introduced in Python 3.5 to perform non-blocking I/O operations. This can make your API much more efficient, especially when handling a large number of concurrent requests. In addition to its automatic documentation and validation, FastAPI also includes support for OpenAPI and JSON Schema. This means that you can use FastAPI to build an API that can be easily consumed by a wide range of clients, including those built using Swagger or Postman.

FastAPI is easy to get started with and includes several useful features out of the box. For example, it includes support for dependency injection, which allows you to easily share common functionality between different parts of your API. It also includes support for automatic routing, which means that you can define your API endpoints using simple function decorators. Overall, FastAPI is a powerful and easy-to-use tool for building APIs with Python. Its combination of automatic documentation and validation, async support, and OpenAPI/JSON Schema integration make it a great choice for developers looking to build high-performance, well-documented APIs quickly and easily.

There are several advantages to using FastAPI for building APIs:

1. Fast performance: FastAPI is built on top of Starlette, which is a high-performance ASGI (Asynchronous Server Gateway Interface) framework. This means that FastAPI can handle a large number of concurrent requests efficiently.
2. Automatic documentation: FastAPI uses Python-type hints to automatically generate web-based API documentation. This means that you don't have to write separate documentation for your API, and can focus on writing code.

3. Automatic data validation: FastAPI uses Pydantic to automatically validate incoming and outgoing data. This can help to prevent errors and improve the reliability of your API.
4. Async support: FastAPI is fully asynchronous, which means that it can take advantage of the async/await syntax introduced in Python 3.5 to perform non-blocking I/O operations. This can make your API much more efficient, especially when handling a large number of concurrent requests.
5. OpenAPI/JSON Schema integration: FastAPI includes support for OpenAPI and JSON Schema, which means that your API can be easily consumed by a wide range of clients, including those built using Swagger or Postman.
6. Easy to use: FastAPI is easy to get started with and includes several useful features out of the box, such as dependency injection and automatic routing. This makes it a great choice for developers looking to build APIs quickly and easily.

1.3 HTML

The HyperText Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript.
Web browsers receive HTML documents from a web server or local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.
HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes, and other items.
HTML elements are delineated by tags, written using angle brackets. Tags such as <img /> and <input /> directly introduce content to the page. Other tags such as <p> surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags but use them to interpret the content of the page.
HTML can embed programs written in a scripting language such as JavaScript, which affects the behavior and content of web pages. The inclusion of CSS defines the look and

layout of content. The World Wide Web Consortium (W3C), the former maintainer of the HTML and current maintainer of the CSS standards, has encouraged the use of CSS over explicit presentational HTML since 1997. A form of HTML, known as HTML5, is used to display video and audio, primarily using the <canvas> element, in collaboration with javascript.

<html>: This is called the HTML root element. All other elements are contained within it.

<head>: The head tag contains the "behind the scenes" elements for a webpage. Elements within the head aren't visible on the front end of a webpage. HTML elements used inside the <head> element include:

- <style>-This HTML tag allows us to insert styling into our web pages and make them appealing to look at with the help of CSS.
- <title>-The title is what is displayed on the top of your browser when you visit a website and contains the title of the webpage that you are viewing.
- <base>-It specifies the base URL for all relative URLs in a document.
- <noscript>– Defines a section of HTML that is inserted when the scripting has been turned off in the user's browser.
- <script>-This tag is used to add functionality to the website with the help of JavaScript.
- <meta>-This tag encloses the metadata of the website that must be loaded every time the website is visited. For eg:- the metadata charset allows you to use the standard UTF-8 encoding on your website. This in turn allows the users to view your webpage in the language of their choice. It is a self-closing tag.
- <link>– The 'link' tag is used to tie together HTML, CSS, and JavaScript. It is self-closing.

<body>: The body tag is used to enclose all the visible content of a webpage. In other words, the body content is what the browser will show on the front end.

1.4 CSS

Cascading Style Sheets (**CSS**) is a stylesheet language used to describe the presentation of a document written in HTML or XML (including XML dialects such as SVG, MathML, or XHTML). CSS describes how elements should be rendered on screen, on paper, in speech, or on other media.

CSS is among the core languages of the **open web** and is standardized across Web browsers according to W3C specifications. Previously, the development of various parts of CSS specification was done synchronously, which allowed the versioning of the latest recommendations. You might have heard about CSS1, CSS2.1, or even CSS3. There will never be a CSS3 or a CSS4; rather, everything is now CSS without a version number.

After CSS 2.1, the scope of the specification increased significantly and the progress on different CSS modules started to differ so much, that it became more effective to develop and release recommendations separately per module. Instead of versioning the CSS specification, W3C now periodically takes a snapshot of the latest stable state of the CSS specification and individual modules' progress. CSS modules now have version numbers, or levels, such as CSS Color Module Level 5.

1.4.1 Bootstrap

Bootstrap is a front-end framework for web development that was developed by Twitter. It is a collection of CSS and HTML conventions that are used as a basis for creating web pages. The main goal of Bootstrap is to make it easier for developers to create responsive, mobile-first websites that are consistent across multiple devices and screen sizes. Bootstrap consists of a set of CSS stylesheets and JavaScript files that can be linked-to to the head of an HTML document. These files contain a series of predefined styles, layouts, and components that can be used as a starting point for building a website. Bootstrap also includes a responsive grid system that allows developers to easily create layouts that adapt to different screen sizes. One of the key features of Bootstrap is its use of a 12-column grid system for layout. This grid system is based on the idea of dividing a page into rows and columns, with each row being divided into 12 equal-width columns. By using this grid system, developers can easily create complex layouts that are flexible and responsive. In addition to the grid system, Bootstrap also provides a series of predefined styles and layout classes that can be applied to HTML elements. For example, there are classes for creating buttons, forms, tables, and navigation menus. These styles and layout classes make it easy for developers to quickly create consistent, professional-looking websites. Bootstrap also includes a series of JavaScript components that can be used to add

interactive elements to a website. These components include modal windows, carousels, tabs, and accordions. These components can be easily implemented using Bootstrap's predefined classes and styles.

One of the main advantages of Bootstrap is that it is a widely-used and well-documented framework. This means that there is a large community of developers who are familiar with Bootstrap and who can provide support and guidance when needed. Additionally, Bootstrap is regularly updated with new features and improvements, making it a constantly evolving and reliable tool for web development. Overall, Bootstrap is a powerful and easy-to-use framework that is popular among developers for its flexibility, responsiveness, and wide range of predefined styles and components. It is a great starting point for creating professional and consistent websites that are designed to work across a variety of devices and screen sizes.

1.5 JavaScript

JavaScript is a programming language that is commonly used to create interactive elements on websites. It is an object-oriented language that is supported by all modern web browsers, and it is the primary language used for client-side web development. JavaScript is executed on the client side, which means that it runs in the user's web browser rather than on the server. This makes it an ideal language for creating interactive elements on websites, such as form validation, real-time updates, and asynchronous requests. One of the key features of JavaScript is its support for objects and object-oriented programming. An object in JavaScript is a collection of properties and methods that can be used to represent real-world entities. For example, a JavaScript object might represent a user, a product, or a piece of data. JavaScript also has a number of built-in objects, such as arrays, dates, and math, that can be used to perform common tasks. Additionally, it has a number of features that make it well-suited for web development, such as its support for events, its ability to manipulate the DOM (Document Object Model), and its ability to communicate with servers using AJAX (Asynchronous JavaScript and XML). In summary, JavaScript is a programming language that is commonly used to create interactive elements on websites. It is an object-oriented language that is supported by all modern web

browsers, and it has a number of features that make it well-suited for web development.

1.6 React.js

Let's say one of your friends posted a photograph on Facebook. Now you go and like the image and then you started checking out the comments too. Now while you are browsing over comments you see that the likes count has increased by 100, since you liked the picture, even without reloading the page. This magical count change is because of ReactJS.

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. 'V' denotes the view in MVC. ReactJS is an open-source, component-based front-end library responsible only for the view layer of the application. It is maintained by Facebook.

React uses a declarative paradigm that makes it easier to reason about your application and aims to be both efficient and flexible. It designs simple views for each state in your application, and React will efficiently update and render just the right component when your data changes. The declarative view makes your code more predictable and easier to debug.

A React application is made of multiple components, each responsible for rendering a small, reusable piece of HTML. Components can be nested within other components to allow complex applications to be built out of simple building blocks. A component may also maintain an internal state – for example, a TabList component may store a variable corresponding to the currently open tab.

Note: React is not a framework. It is just a library developed by Facebook to solve some problems that we were facing earlier.

Prerequisites: Download Node packages with their latest version.

How does it work: While building client-side apps, a team of Facebook developers realized that the DOM is slow (The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.). So, to make it faster, React implements a virtual DOM that is a DOM tree representation in JavaScript. So when it needs to read or write to the DOM, it will use the virtual representation of it. Then the virtual DOM will try to

find the most efficient way to update the browser's DOM.
Unlike browser DOM elements, React elements are plain objects and are cheap to create. React DOM takes care of updating the DOM to match the React elements. The reason for this is that JavaScript is very fast and it's worth keeping a DOM tree in it to speed up its manipulation.
Although React was conceived to be used in the browser, because of its design it can also be used in the server with Node.js.

So why should you use React?

- The ability to work with a friendlier and optimized VIrtual Browser (*Virtual-DOM*) as compared to Real DOM.
- You just need **JavaScript knowledge** to work with React.
- React Native allows you to **create Mobile Applications** (iOS and Android).
- React team at Facebook tests all features and versions of React on Facebook.com as a result the React Library is **battle-ready**.
- React allows developers to declare **stateful user interfaces**.

1.7 Keras

Keras is a popular open-source software library that is widely used for developing and training deep learning models. It was developed with the goal of making it easier for developers to build and deploy deep learning applications. One of the main advantages of Keras is its simplicity and ease of use. It has a high-level API that is designed to be easy to understand and use, even for developers who are new to deep learning. This makes it an excellent choice for prototyping and quickly building deep learning models. Keras also has a number of features that make it a powerful tool for deep learning. It has a large collection of pre-trained models that can be easily loaded and used for a variety of tasks, including image classification, natural language processing, and even playing games. It also supports a range of popular deep learning libraries, such as TensorFlow and Theano, allowing developers to choose the one that best fits their needs. Another key advantage of Keras is its flexibility. It can be used to build a wide range of deep learning models, from simple feedforward networks to complex convolutional neural networks. It also supports multiple input and output layers, allowing developers to build models that can handle a variety of tasks. Despite its many strengths, Keras is not without its limitations. It is not as fast as some other deep

learning libraries, such as PyTorch, and it may not be the best choice for very large and complex deep learning projects. However, for most developers, the simplicity and flexibility of Keras make it an excellent choice for building and training deep learning models. In summary, Keras is a popular and powerful open-source software library that is widely used for developing and training deep learning models. It is known for its simplicity, ease of use, and flexibility, making it an excellent choice for developers who are new to deep learning or who need to quickly prototype and build deep learning models.

1.8 CNN

Convolutional neural networks (CNNs) are a type of neural network that is particularly well-suited for image classification and object recognition tasks. They are called "convolutional" neural networks because they use a mathematical operation called convolution to process the input data. At a high level, a CNN consists of an input layer, one or more hidden layers, and an output layer. The input layer is responsible for accepting the raw input data, which is typically an image. The hidden layers are responsible for extracting features from the input data and transforming it into a form that is easier to classify. The output layer is responsible for producing the final classification or prediction. The key to a CNN's ability to perform image classification is its use of convolutional layers. A convolutional layer consists of a set of filters, also called kernels, that are used to scan the input image and extract features from it. Each filter is responsible for identifying a specific pattern or feature in the input data. For example, one filter may be responsible for detecting edges, while another filter may be responsible for detecting corners. As the filters scan the input image, they create a set of feature maps that capture the different patterns and features present in the image. These feature maps are then passed through one or more pooling layers, which reduce the size of the feature maps by aggregating the information in them. This process is repeated until the CNN has extracted all of the relevant features from the input image. Finally, the feature maps are passed through the output layer, which uses them to classify the input image. The output layer typically consists of a series of fully-connected layers, similar to those found in other types of neural networks. These layers use the extracted features to make a final prediction or classification.

1.9 ResNet

ResNet (short for "Residual Network") is a type of convolutional neural network (CNN) that was developed by researchers at Microsoft in 2015. It was designed to address the problem of vanishing gradients, which is a common issue in very deep neural networks. A vanishing gradient occurs when the gradients of the parameters in the deeper layers of a neural network become very small, making it difficult for the network to learn and improve. This can cause the performance of the network to plateau or even degrade as the number of layers increases. ResNet addresses this problem by using skip connections, which allow the gradients to bypass one or more layers and be directly propagated to the earlier layers. This helps to ensure that the gradients remain large and easy to learn from, even in very deep networks. ResNets are typically very deep networks, with hundreds or even thousands of layers. They are commonly used in image classification tasks and have achieved state-of-the-art performance on a number of benchmarks. One of the key advantages of ResNets is their ability to learn from very deep networks without suffering from the problem of vanishing gradients. This allows them to extract more powerful and expressive features from the input data, leading to better performance. Another advantage of ResNets is their simplicity. They are relatively easy to implement and train, and they have a straightforward architecture that is easy to understand. This makes them a popular choice for researchers and practitioners working in the field of deep learning. In summary, ResNet is a type of convolutional neural network that uses skip connections to address the problem of vanishing gradients in very deep networks. It is a powerful and popular tool for image classification tasks and has achieved state-of-the-art performance on a number of benchmarks.

2. Software Requirement Specification (SRS)

2.1 External Interface Requirement:

We classify External Interfaces into 4 types, those are:

1. **User Interface:**

Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, and so on. Define the software components for which a user interface is needed. Details of the user interface design should be documented in a separate user interface specification.

2. **Hardware interface:**

Describe the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware, and the communication protocols to be used.

3. **Software Interface:**

Describe the connections between this product and other specific software components (name and version), including databases, operating systems, tools, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each. Describe the services needed and the nature of communications. Refer to documents that describe detailed application programming interface protocols. Identify data that will be shared across software components. If the data-sharing mechanism must be implemented in a specific way (for example, the use of a global data area in a multitasking operating system), specify this as an implementation constraint.

4. **Communication Interface:**

Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on. Define any pertinent message formatting. Identify any communication standards that will be used, such as FTP or HTTP. Specify any communication security or encryption issues, data transfer rates, and synchronization mechanisms.

2.2 Non-Functional Requirement:

1. Performance Requirements:

If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real-time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.

2. Safety Requirements:

Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product's design or use. Define any safety certifications that must be satisfied.

3. Security Requirements:

Specify any requirements regarding security or privacy issues surrounding the use of the product or protection of the data used or created by the product. Define any user identity authentication requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.

4. Software Quality Attributes:

Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.

2.3 Other Requirements:

- Python 3.7(VSCode, pip)
- BackEnd technologies -Keras, NumPy, pandas, FastAPI, uvicorn, OpenCV, Jupyter Notebook.
- FrontEnd Technologies – React.js, BootStrap

3. Methodology

A. Task

The task is to build a system that will take an image input in the form of a dimensional array and generate an output consisting of a sentence that describes the image and is syntactically and grammatically correct.

B. Corpus

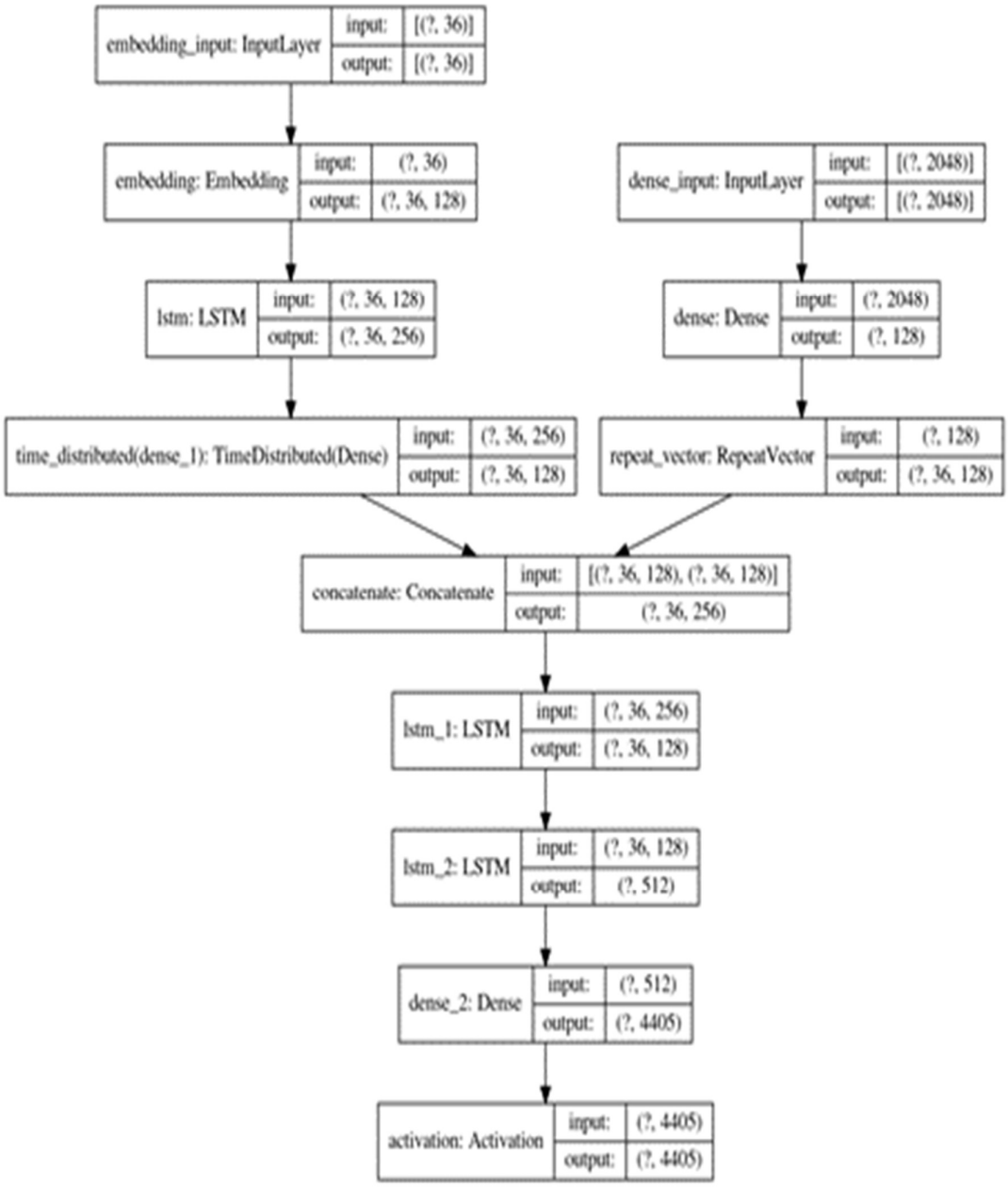
We have used the Flickr 8K dataset as the corpus. The dataset consists of 8000 images and for every image, there are 5 captions. The 5 captions for a single image help in understanding all the various possible scenarios. The dataset has a predefined training dataset Flickr_8k.trainImages.txt (6,000images), a development dataset Flickr_8k.devImages.txt (1,000 images), and a test dataset Flickr_8k.testImages.txt (1,000 images).

C. Preprocessing

Data preprocessing is done in two parts, the images and the corresponding captions are cleaned and pre-processed separately. Image preprocessing is done by feeding the input data to the Xception application of the Keras API running on top of TensorFlow. Xception is pre-trained on ImageNet. This helped us train the images faster with the help of Transfer learning. The descriptions are cleaned using the tokenizer class in Keras, this will vectorize the text corpus and is stored in a separate dictionary. Then each word of the vocabulary is mapped with a unique index value.

D. Model

Deep learning carries out the machine learning process using an artificial neural network that is composed of several levels arranged in a hierarchy. The model is based on deep networks where the flow of information starts from the initial level, where the model learns something simple and then the output of which is passed to layer two of the network, and input is combined into something that is a bit more complex and passes it on to the third level. This process continues as each level in the network produces something more complex from the input it received from the ascendant level.



4. DFD and State chart diagrams for Image Caption Generator System

Here we have shown the DFDs of our system (i.e. Data Flow Diagrams). DFDs provide us with a basic overview of the whole Image Caption Generator System or process being analyzed or modeled.

Fig 1: DFD Diagram Level 0

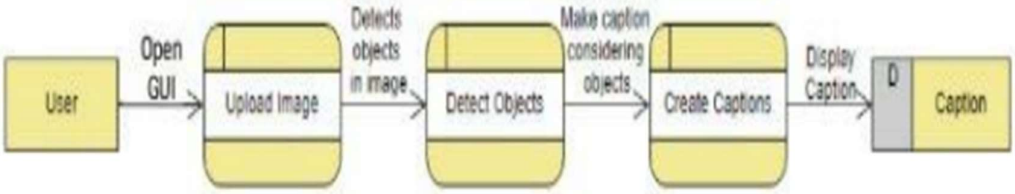


Fig 2: DFD Diagram Level 1

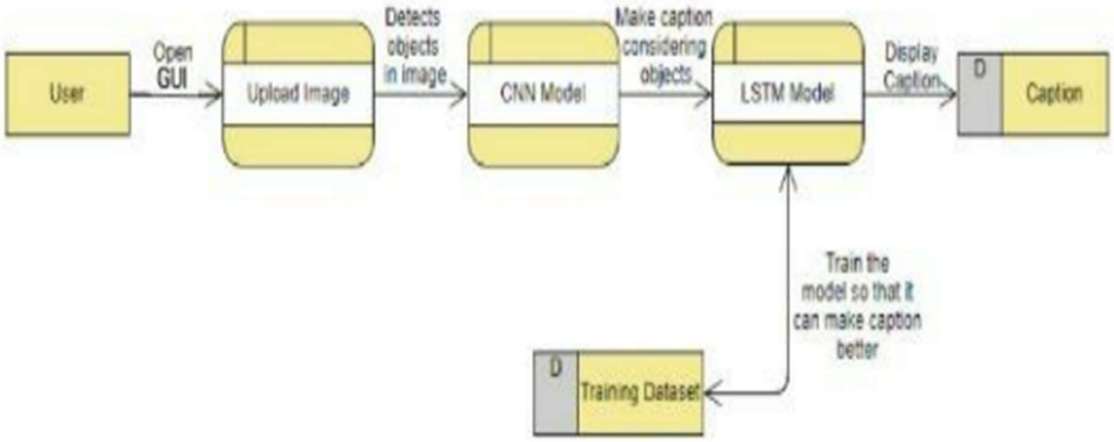


Fig 3: DFD Diagram Level 2

Figure 4. shows the State Chart Diagram of the system. The first user will browse the site. Then he will upload the image, CNN will identify the objects present in the image then ResNet will start preparing captions considering the objects present in the image using [8] Training Dataset, which comprises of Image Data set and Text Data Set, after the training, a suitable caption will be generated and displayed to the user.

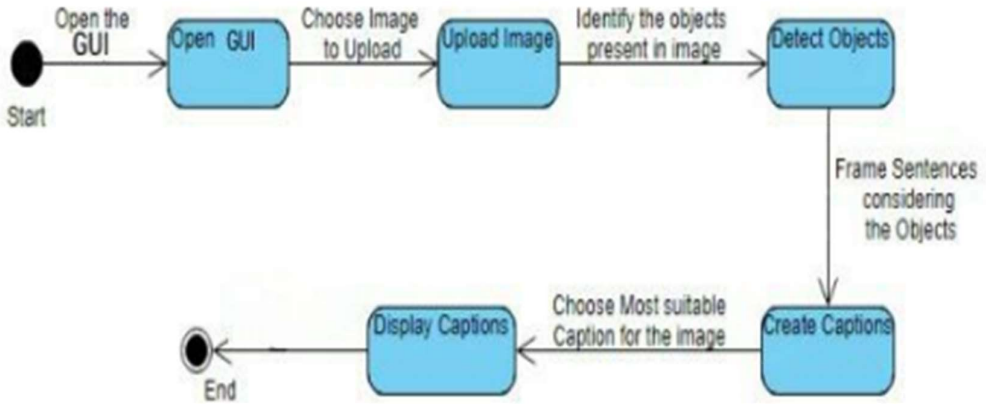


Fig 4: State Chart Diagram of Steps taken by the system

The proposed system of Image Caption Generator has the capability to Generate Captions for the Images, provided during the Training purpose & also for the New images as well. Our Model takes an Image as Input and by analyzing the image it detects objects present in an image and creates a caption that describes the image well enough for any machine to understand what an image is trying to say.

Conclusion

In this advanced Python project, an image caption generator has been developed using a ResNet model. Some key aspects of our project to note are that our model depends on the data, so, it cannot predict the words that are out of its vocabulary. A dataset consisting of 8000 images is used here. But for production-level models i.e. higher accuracy models, we need to train the model on larger than 100,000 image datasets so that better accuracy models can be developed.

Future Scope

The scope of this project is as follows:

1. It will provide great help to visually impaired people to understand image scenes.
2. To classify images based on their captions.
3. To help law by searching and classifying images across social media that match their description keywords.
4. Users can search inside the videos for a specific frame/part.

REFERENCES

[1] CS771 Project Image Captioning by Ankit Gupta, Kartik Hira, and Bajaj Dilip.

[2] ”Every Picture Tells a Story: Generating Sentences from Images.” Computer Vision ECCV (2016) by Farhadi, Ali, Mohsen Hejrati, Mohammad Amin Sadeghi, Peter Young, Cyrus Rashtchian, Julia Hocken-Maier, and David Forsyth

[3] Automatic Caption Generation for News Images by Yansong Feng, and Mirella Lapata, IEEE (2013).

[4] Image Caption Generator Based on Deep Neural Networks by Jianhui Chen, Wenqiang Dong, and Minchen Li, ACM (2014).

[5] Show and Tell: A Neural Image Caption Generator by Oriol Vinyl, Alexander Toshev, Samy Bengio, Dumitru Erhan, IEEE (2015).

[6] Image2Text: A Multimodal Caption Generator by Chang Liu, Changhua Wang, Fuchun Sun, Yong Rui, ACM (2016).

[7] The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions by Sepp Hochreiter.

[8] Where to put the Image in an Image Caption Generator by Marc Tanti, Albert Gatt, Kenneth P. Camilleri.

[9] Sequence to sequence -video to text by Subhashini Venugopalan, Marcus Rohrbach, Jeffrey Donahue, Raymond Mooney, Trevor Darrell, and Kate Saenko.

[10] Learning phrase representations using RNN encoder-decoder for statistical machine translation by K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengi.

[11] TVPRNN for image caption generation.Liang Yang and Haifeng Hu.

[12] Image Captioning in the Wild: How People Caption Images on Flickr Philipp Blandford, Tushar Karayil, Damian Borth, Andreas Dengel, German Institute for Artificial Intelligence, Kaiserslautern, Germany.

[13] Image Caption Generator Based On Deep Neural Networks Jianhui Chen, Wenqiang Dong, Minchen Li, CS Department. ACM 2014.

[14] BLEU: A method for automatic evaluation of machine translation. InACL, 2002 by K. Papineni, S. Roukos, T. Ward, and W. J. Zhu.

[15] HaoranWang, Yue Zhang, and Xiaosheng Yu, “An Overview of Image Caption Generation Methods”, (CIN-2020)

[16] B.Krishnakumar, K.Kousalya, S.Gokul, R.Karthikeyan, and D.Kaviyarasu, “IMAGE CAPTION GENERATOR USING DEEP LEARNING”, (International Journal of Advanced Science and Technology- 2020)

[17] MD. Zakir Hossain, Ferdous Sohel, Mohd Fairuz Shiratuddin, and Hamid Laga, “A Comprehensive Survey of Deep Learning for Image Captioning”,(ACM-2019)

[18] Rehab Alahmadi, Chung Hyuk Park, and James Hahn, “Sequence-to-sequence image caption generator”, (ICMV-2018)

[19] Oriol Vinyals, Alexander Toshev, SamyBengio, and Dumitru Erhan, “Show and Tell: A Neural Image Caption Generator”,(CVPR 1, 2-2015)

[20] Priyanka Kalena, Nishi Malde, Aromal Nair, Saurabh Parkar, and Grishma Sharma, “Visual Image Caption Generator Using Deep Learning”, (ICAST-2019)

[21] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, et al., "Show, attend and tell: Neural image caption generation with visual attention", Proceedings of the International Conference on Machine Learning (ICML), 2015.

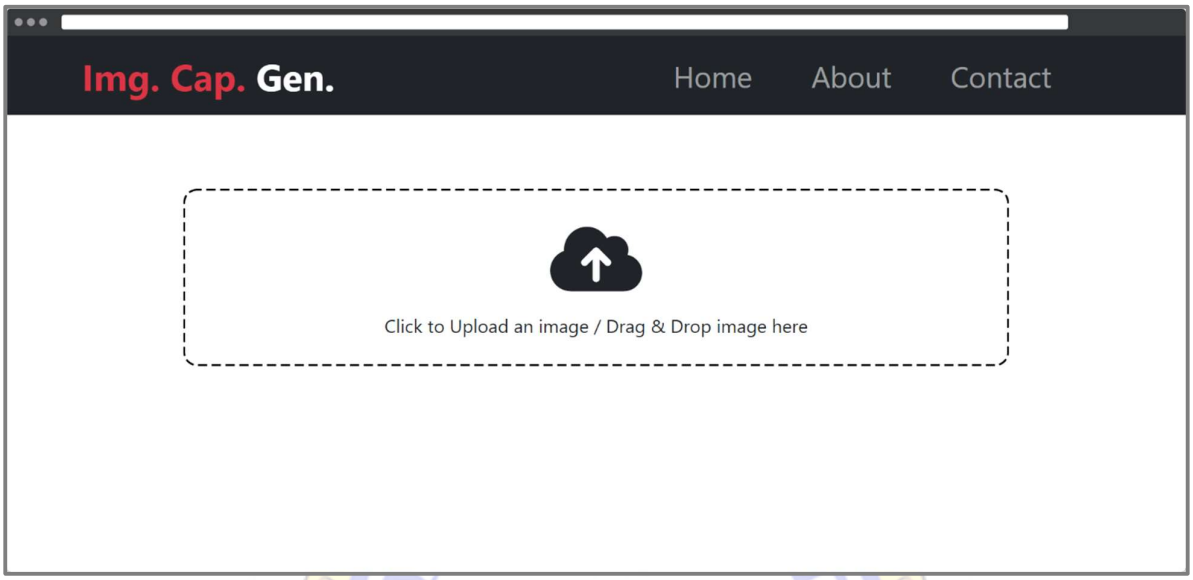
[22] J. Redmon, S. Divvala, Girshick, and A. Farhadi, "You only look once: Unified real-time object detection", Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016

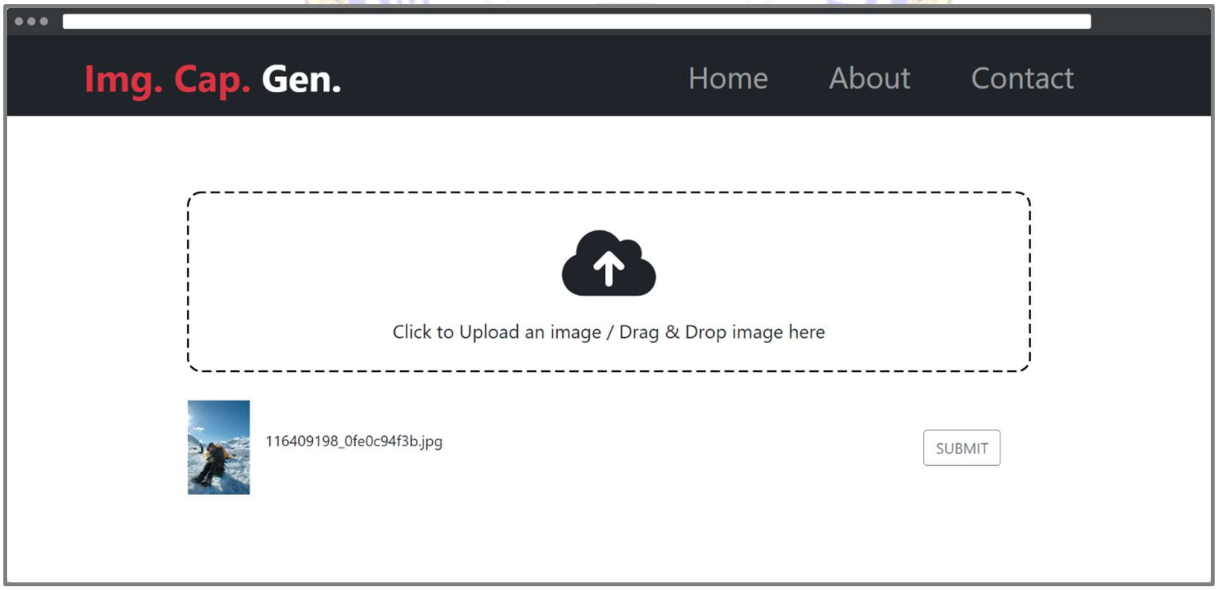
[23] D. Bahdanau, K. Cho, and Y. Bengio. “Neural machine translation by jointly learning to align and translate.arXiv:1409.0473”, 2014.

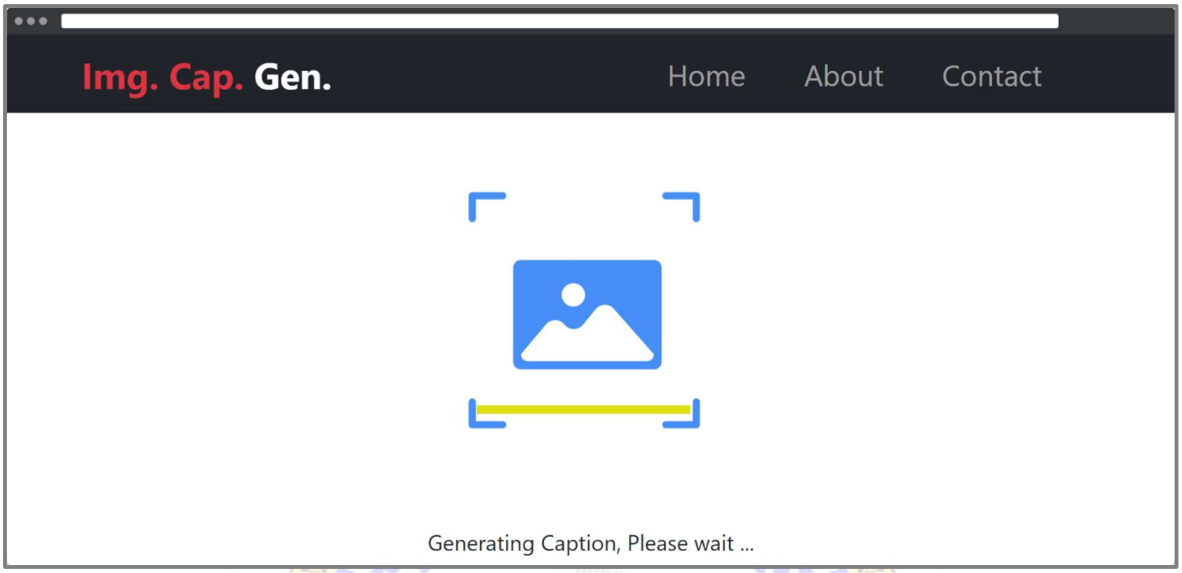
[24] www.kaggle.com/code/quadeer15sh/flickr8k-image-captioning-using-cnns-lstms

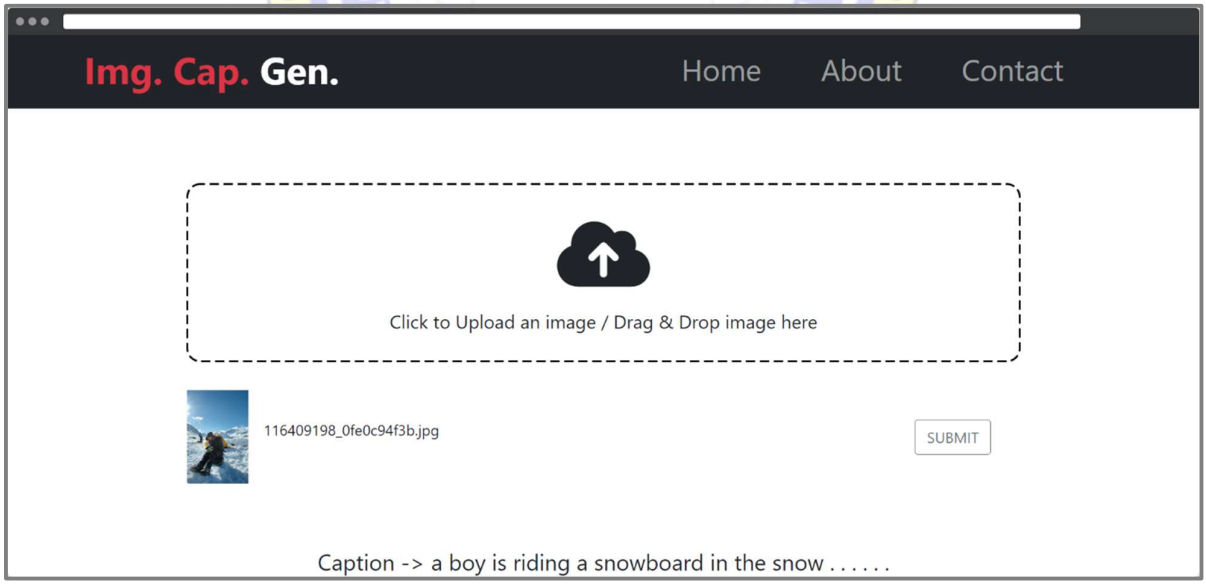
APPENDIX A

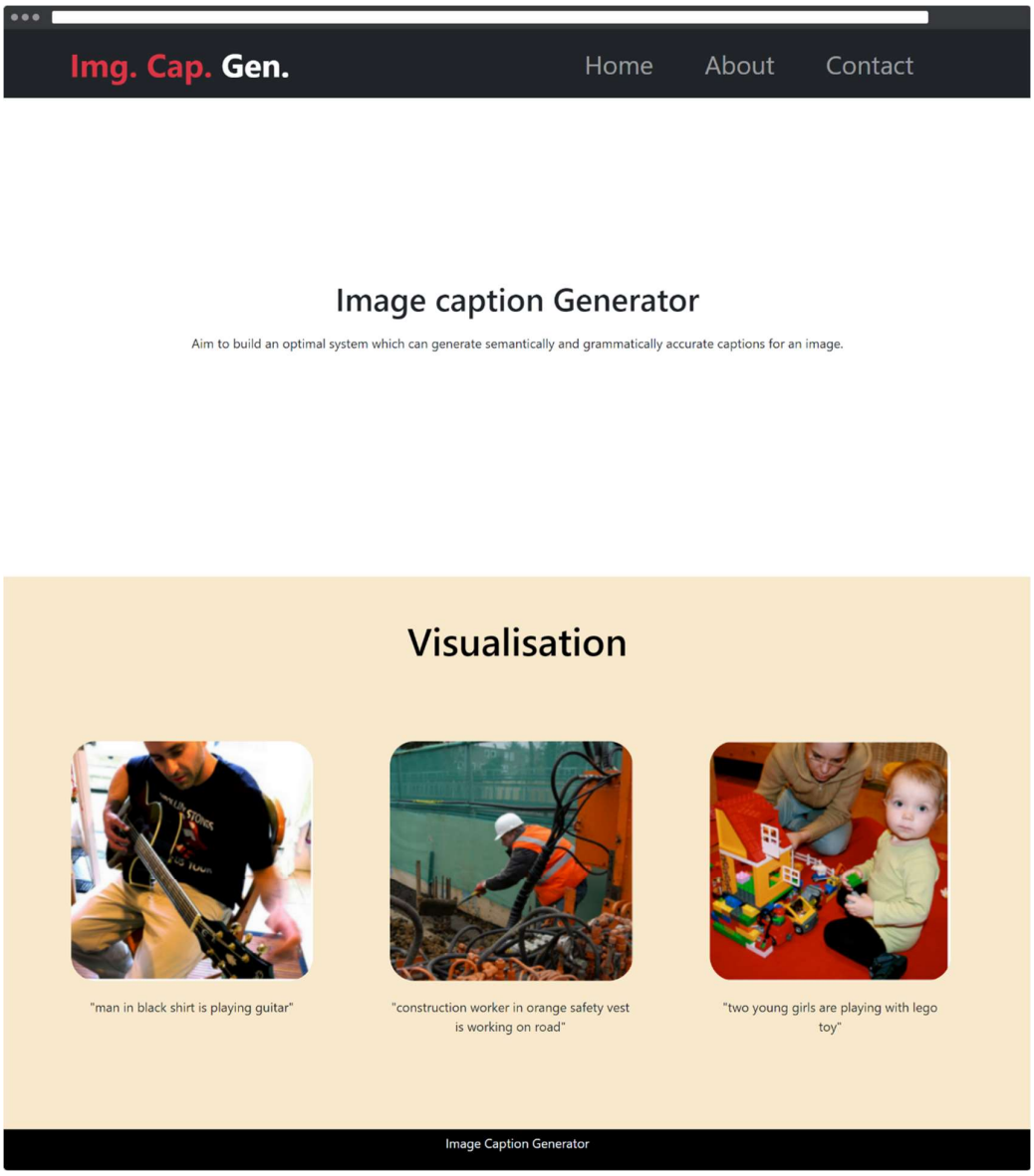
SCREENSHOTS

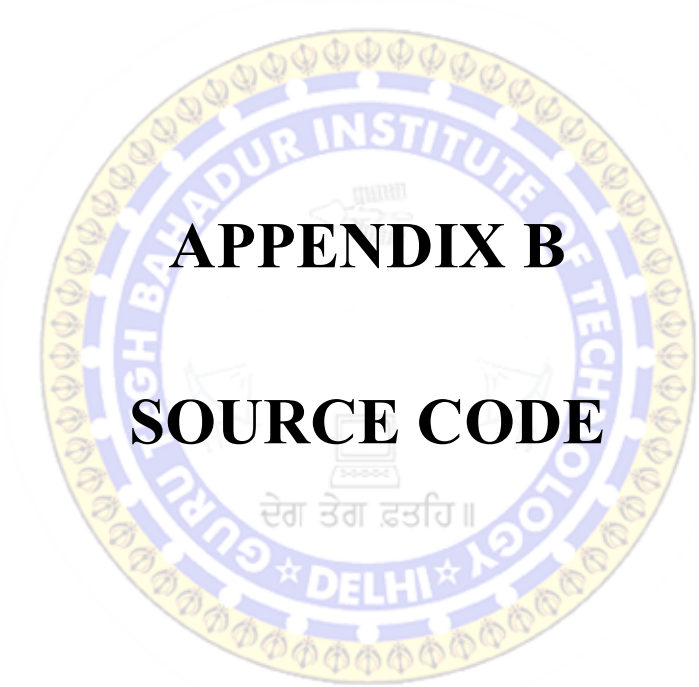

Img. Cap. Gen.
Home
About
Contact
Click to Upload an image / Drag & Drop image here


Img. Cap. Gen.
Home
About
Contact
Click to Upload an image / Drag & Drop image here
116409198_0fe0c94f3b.jpg
SUBMIT


Img. Cap. Gen.
Home
About
Contact
Generating Caption, Please wait ...


Img. Cap. Gen.
Home
About
Contact
Click to Upload an image / Drag & Drop image here
116409198_0fe0c94f3b.jpg
SUBMIT
Caption -> a boy is riding a snowboard in the snow . . . . . .


Img. Cap. Gen.
Home
About
Contact
Image caption Generator
Aim to build an optimal system which can generate semantically and grammatically accurate captions for an image.
Visualisation
"man in black shirt is playing guitar"
"construction worker in orange safety vest is working on road"
"two young girls are playing with lego toy"
Image Caption Generator



APPENDIX B

SOURCE CODE

main.py

```
import uvicorn
import os
from fastapi import FastAPI, responses, File, UploadFile
from fastapi.middleware.cors import CORSMiddleware
from predictor import predict
import cv2
import numpy as np

app = FastAPI(title='Captioning API',
          description="API for generating captions of an image", version="1.0", debug=True)

origins = [
    "http://localhost",
    "http://localhost:3000",
    "http://localhost:8080",
    "http://localhost:5000",
    "http://127.0.0.1:5000",
    "http://192.168.1.41:3000"
]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

vocabPath = os.path.dirname(os.getcwd()).replace(
    '\\', '/') + '/ImageCaptionGenerator/BackEnd/Model/vocab.npy'
weightPath = os.path.dirname(os.getcwd()).replace(
    '\\', '/') + '/ImageCaptionGenerator/BackEnd/Model/mine_model_weights.h5'

predictor = predict(vocabPath, weightPath)


def load_image_into_numpy_array(data):
    npimg = np.frombuffer(data, np.uint8)
    frame = cv2.imdecode(npimg, cv2.IMREAD_COLOR)
    return cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)


@app.post('/genCaption')
async def captiongen(file: UploadFile = File(...)):
    image = load_image_into_numpy_array(await file.read())
    caption = predictor.prediction(image)
    print(caption)
    caption = "Caption -> " + caption
    # return {"Caption - ": caption}
    return responses.JSONResponse(content={"caption": caption})


if __name__ == "__main__":
    uvicorn.run(app, port=5000)
```

predictor.py

```
import cv2
import numpy as np
from keras.applications.resnet import ResNet50
from keras.layers import Dense, LSTM, TimeDistributed, Embedding, Activation, RepeatVector,
Concatenate
from keras.models import Sequential, Model
import cv2
from keras_preprocessing.sequence import pad_sequences
from tqdm import tqdm

class predict:
    def __init__(self, vocabPath, weightPath) -> None:
        self.vocab = np.load(vocabPath, allow_pickle=True)
        self.vocab = self.vocab.item()
        self.inv_vocab = {v: k for k, v in self.vocab.items()}
        embedding_size = 128
        vocab_size = len(self.vocab)
        self.max_len = 40

        image_model = Sequential()

        image_model.add(
            Dense(embedding_size, input_shape=(2048,), activation='relu'))
        image_model.add(RepeatVector(self.max_len))

        language_model = Sequential()

        language_model.add(Embedding(input_dim=vocab_size,
                                     output_dim=embedding_size, input_length=self.max_len))
        language_model.add(LSTM(256, return_sequences=True))
        language_model.add(TimeDistributed(Dense(embedding_size)))

        conca = Concatenate()([image_model.output, language_model.output])
        x = LSTM(128, return_sequences=True)(conca)
        x = LSTM(512, return_sequences=False)(x)
        x = Dense(vocab_size)(x)
        out = Activation('softmax')(x)
        self.model = Model(
            inputs=[image_model.input, language_model.input], outputs=out)

        self.model.compile(loss='categorical_crossentropy',
                           optimizer='RMSprop', metrics=['accuracy'])

        self.model.load_weights(weightPath)

        print("="*150)
        print("MODEL LOADED")

        self.resnet = ResNet50(include_top=False, weights='imagenet',
                               input_shape=(224, 224, 3), pooling='avg')
        print("="*150)
        print("RESNET MODEL LOADED")

    def prediction(self, image):
        # image = cv2.imread(imgPath)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        image = cv2.resize(image, (224, 224))
```

```
        image = np.reshape(image, (1, 224, 224, 3))

        incept = self.resnet.predict(image).reshape(1, 2048)

        print("="*50)
        print("Predict Features")

        text_in = ['startofseq']

        final = ''

        print("="*50)
        print("GETING Captions")

        count = 0
        while tqdm(count < 30):

            count += 1

            encoded = []
            for i in text_in:
                encoded.append(self.vocab[i])

            padded = pad_sequences(
                [encoded], maxlen=self.max_len, padding='post', truncating='post').reshape(1, self.max_len)

            sampled_index = np.argmax(self.model.predict([incept, padded]))

            sampled_word = self.inv_vocab[sampled_index]
            if sampled_word != 'endofseq':
                final = final + ' ' + sampled_word

            text_in.append(sampled_word)
        return final
```

imagecaptioning.ipynb

```
import numpy as np
import pandas as pd
import cv2
import os
from glob import glob

"""# **image Preprocess**"""

images_path = '../input/flickr8k-sau/Flickr_Data/Images/'
images = glob(images_path+'*.jpg')
len(images)

images[:5]

import matplotlib.pyplot as plt

for i in range(5):
    plt.figure()
    img = cv2.imread(images[i])
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```
    plt.imshow(img)

from keras.applications import ResNet50

incept_model = ResNet50(include_top=True)

from keras.models import Model
last = incept_model.layers[-2].output
modele = Model(inputs = incept_model.input,outputs = last)
modele.summary()

images_features = {}
count = 0
for i in images:
    img = cv2.imread(i)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (224,224))

    img = img.reshape(1,224,224,3)
    pred = modele.predict(img).reshape(2048,)

    img_name = i.split('/')[-1]

    images_features[img_name] = pred

    count += 1

    if count > 1750:
        break

    if count % 100 == 0:
        print(count)

len(images_features)

"""# **Text Preprocess**"""

caption_path = '../input/flickr8k-sau/Flickr_Data/Flickr_TextData/Flickr8k.token.txt'

captions = open(caption_path, 'rb').read().decode('utf-8').split('\n')

len(captions)

captions_dict = {}
for i in captions:
    try:
        img_name = i.split('\t')[0][:-2]
        caption = i.split('\t')[1]
        if img_name in images_features:
            if img_name not in captions_dict:
                captions_dict[img_name] = [caption]

            else:
                captions_dict[img_name].append(caption)

    except:
        pass

len(captions_dict)
```

```
"""# **Visualize Images with captions**"""

import matplotlib.pyplot as plt

for i in range(5):
    plt.figure()
    img_name = images[i]


    img = cv2.imread(img_name)

    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.xlabel(captions_dict[img_name.split('/')[-1]])
    plt.imshow(img)

import matplotlib.pyplot as plt

for k in images_features.keys():
    plt.figure()

    img_name = '../input/flickr8k-sau/Flickr_Data/Images/' + k


    img = cv2.imread(img_name)

    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.xlabel(captions_dict[img_name.split('/')[-1]])
    plt.imshow(img)

    break

def preprocessed(txt):
    modified = txt.lower()
    modified = 'startofseq ' + modified + ' endofseq'
    return modified

for k,v in captions_dict.items():
    for vv in v:
        captions_dict[k][v.index(vv)] = preprocessed(vv)

"""# **Create Vocabulary**"""

count_words = {}
for k,vv in captions_dict.items():
    for v in vv:
        for word in v.split():
            if word not in count_words:

                count_words[word] = 0

            else:
                count_words[word] += 1

len(count_words)

THRESH = -1
count = 1
new_dict = {}
for k,v in count_words.items():
    if count_words[k] > THRESH:
```

```
        new_dict[k] = count
        count += 1

len(new_dict)

new_dict['<OUT>'] = len(new_dict)

# captions_backup = captions_dict.copy()

# captions_dict = captions_backup.copy()

for k, vv in captions_dict.items():
    for v in vv:
        encoded = []
        for word in v.split():
            if word not in new_dict:
                encoded.append(new_dict['<OUT>'])
            else:
                encoded.append(new_dict[word])


        captions_dict[k][vv.index(v)] = encoded

# captions_dict

"""----------------------------------------------------------------------------------------------------

# **Build Generator Function**
"""

from keras.utils import to_categorical
from keras.preprocessing.sequence import pad_sequences

MAX_LEN = 0
for k, vv in captions_dict.items():
    for v in vv:
        if len(v) > MAX_LEN:
            MAX_LEN = len(v)
            print(v)

MAX_LEN

# captions_dict

Batch_size = 5000
VOCAB_SIZE = len(new_dict)

def generator(photo, caption):
    n_samples = 0

    X = []
    y_in = []
    y_out = []

    for k, vv in caption.items():
        for v in vv:
            for i in range(1, len(v)):
                X.append(photo[k])

                in_seq= [v[:i]]
```

```
            out_seq = v[i]

            in_seq = pad_sequences(in_seq, maxlen=MAX_LEN, padding='post', truncating='post')[0]
            out_seq = to_categorical([out_seq], num_classes=VOCAB_SIZE)[0]

            y_in.append(in_seq)
            y_out.append(out_seq)

    return X, y_in, y_out

X, y_in, y_out = generator(images_features, captions_dict)

len(X), len(y_in), len(y_out)

X = np.array(X)
y_in = np.array(y_in, dtype='float64')
y_out = np.array(y_out, dtype='float64')

X.shape, y_in.shape, y_out.shape

X[1510]

y_in[2]

"""# **MODEL**"""

from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.utils import plot_model
from keras.models import Model, Sequential
from keras.layers import Input
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Embedding
from keras.layers import Dropout
from keras.layers.merge import add
from keras.callbacks import ModelCheckpoint
from keras.layers import Dense, Flatten,Input, Convolution2D, Dropout, LSTM, TimeDistributed,
Embedding, Bidirectional, Activation, RepeatVector,Concatenate
from keras.models import Sequential, Model

embedding_size = 128
max_len = MAX_LEN
vocab_size = len(new_dict)

image_model = Sequential()

image_model.add(Dense(embedding_size, input_shape=(2048,), activation='relu'))
image_model.add(RepeatVector(max_len))

image_model.summary()

language_model = Sequential()

language_model.add(Embedding(input_dim=vocab_size, output_dim=embedding_size,
input_length=max_len))
language_model.add(LSTM(256, return_sequences=True))
language_model.add(TimeDistributed(Dense(embedding_size)))

language_model.summary()
```

```
conca = Concatenate()([image_model.output, language_model.output])
x = LSTM(128, return_sequences=True)(conca)
x = LSTM(512, return_sequences=False)(x)
x = Dense(vocab_size)(x)
out = Activation('softmax')(x)
model = Model(inputs=[image_model.input, language_model.input], outputs = out)

# model.load_weights("../input/model_weights.h5")
model.compile(loss='categorical_crossentropy', optimizer='RMSprop', metrics=['accuracy'])
model.summary()

model.fit([X, y_in], y_out, batch_size=512, epochs=100)

inv_dict = {v:k for k, v in new_dict.items()}

# model.save('model')

model.save_weights('mine_model_weights.h5')

np.save('vocab.npy', new_dict)
np.save('inv_dict.npy', inv_dict)

def getImage(x):

    test_img_path = images[x]

    test_img = cv2.imread(test_img_path)
    test_img = cv2.cvtColor(test_img, cv2.COLOR_BGR2RGB)

    test_img = cv2.resize(test_img, (299,299))

    test_img = np.reshape(test_img, (1,299,299,3))

    return test_img

"""# **Predictions**"""

for i in range(5):

    no = np.random.randint(1500,7000,(1,1))[0,0]
    test_feature = modele.predict(getImage(no)).reshape(1,2048)

    test_img_path = images[no]
    test_img = cv2.imread(test_img_path)
    test_img = cv2.cvtColor(test_img, cv2.COLOR_BGR2RGB)


    text_inp = ['startofseq']

    count = 0
    caption = ''
    while count < 25:
        count += 1

        encoded = []
        for i in text_inp:
            encoded.append(new_dict[i])

        encoded = [encoded]
```

```
        encoded = pad_sequences(encoded, padding='post', truncating='post', maxlen=MAX_LEN)


        prediction = np.argmax(model.predict([test_feature, encoded]))

        sampled_word = inv_dict[prediction]

        caption = caption + ' ' + sampled_word

        if sampled_word == 'endofseq':
            break

        text_inp.append(sampled_word)

    plt.figure()
    plt.imshow(test_img)
    plt.xlabel(caption)

test_img = cv2.imread('../input/samepe/b8b237ee-7cfe-4eab-b79b-dac389707899.jpg')
test_img = cv2.cvtColor(test_img, cv2.COLOR_BGR2RGB)
test_img = cv2.resize(test_img, (299,299))
test_img = np.reshape(test_img, (1,299,299,3))

test_feature = modele.predict(test_img).reshape(1,2048)

test_img = cv2.imread('../input/samepe/b8b237ee-7cfe-4eab-b79b-dac389707899.jpg')
test_img = cv2.cvtColor(test_img, cv2.COLOR_BGR2RGB)


text_inp = ['startofseq']

count = 0
caption = ''
while count < 25:
    count += 1
    encoded = []
    for i in text_inp:
        encoded.append(new_dict[i])

    encoded = [encoded]

    encoded = pad_sequences(encoded, padding='post', truncating='post', maxlen=MAX_LEN)


    prediction = np.argmax(model.predict([test_feature, encoded]))

    sampled_word = inv_dict[prediction]

    caption = caption + ' ' + sampled_word

    if sampled_word == 'endofseq':
        break

    text_inp.append(sampled_word)

plt.figure()
plt.imshow(test_img)
plt.xlabel(caption)
```

App.js

```
import './App.css';
import {About} from './components/About';
import {Contact} from './components/Contact';
import {Home} from "./components/Home";
import {Navbar} from './components/Navbar';
import { BrowserRouter, Routes, Route, Navigate} from 'react-router-dom';

function App() {
 return (

   <BrowserRouter>
    <Navbar />

    <Routes>
      <Route path='/' element={<Home/>} />
      <Route path='/about' element={<About/>} />
      <Route path='/contact' element={<Contact/>} />
      <Route path = "*" element = {<Navigate to = "/" />} />
    </Routes>

   </BrowserRouter>
 );
}

export default App;
```

Home.js

```
import React, { useState, useCallback } from "react";
import Lottie from "react-lottie";
import Loading from "../assets/Loading.json";
import { useDropzone } from "react-dropzone";
import "./Home.css";

export const Home = () => {
  const [selectedFile, setSelectedFile] = useState(null);
  const [isFilePicked, setIsFilePicked] = useState(false);
  const [data, setData] = useState("");
  const [file, setFile] = useState();
  const [fname, setFname] = useState("");
  const [isLoading, setLoading] = useState(false);

  const onDrop = useCallback((acceptedFiles) => {
    setSelectedFile(acceptedFiles[0]);
    setFname(acceptedFiles[0].name);
    setIsFilePicked(true);
    setFile(URL.createObjectURL(acceptedFiles[0]));
  }, []);

  const { getRootProps, getInputProps } = useDropzone({
    onDrop,
    accept: "image/*",
    multiple: false,
  });
  const setDefaultValues = () => {
    setIsFilePicked(false);
    setData("");
```

```
  setSelectedFile(null);
  setFile(null);
  setFname("");
};

const defaultOptionsLoading = {
  loop: true,
  autoplay: true,
  animationData: Loading,
  rendererSettings: {
    preserveAspectRatio: "xMidYMid slice",
  },
};
const changeHandler = (event) => {
  setSelectedFile(event.target.files[0]);
  setFname(event.target.files[0].name);
  setIsFilePicked(true);
  setFile(URL.createObjectURL(event.target.files[0]));
};

const handleSubmit = (event) => {
  event.preventDefault();
  setData("");
  setLoading(true);
  const formData2 = new FormData();
  formData2.append("file", selectedFile);

  const requestOptions = {
    method: "POST",
    body: formData2,
  };

  fetch("http://127.0.0.1:5000/genCaption", requestOptions)
    .then((response) => response.json())
    .then(function (response) {
      let a = response;
      console.log(a);
      setData(response.caption);
      setLoading(false);
    })
    .catch((err) => {
      setLoading(false);
      setData(
        "Error occured!! Please check your connection or upload a valid image."
      );
    });
};

return (
  <div>
    <center>
      {!isLoading ? (
        <>
          <form onSubmit={handleSubmit}>
            <div className="upload-box" {...getRootProps()}>
              <input
                type="file"
                disabled={isLoading}
                id="image"
                style={{ display: "none" }}
```

```
            {...getInputProps()}
            onChange={changeHandler}
            onClick={setDefaultValues}
           />
           <label for="image">
            <i class="fa-solid fa-cloud-arrow-up"></i>
            <p style={{ fontSize: 20 }}>
             Click to Upload an image / Drag & Drop image here
            </p>
           </label>
          </div>

          {isFilePicked ? (
           <div className="image-box">
            <div className="image-preview">
             <img src={file} />
             <p>{fname}</p>
            </div>

            <button
             type="submit"
             disabled={isLoading}
             class="btn btn-outline-secondary submit-buttion"
            >
             SUBMIT
            </button>
           </div>
          ) : null}
         </form>
         <div class="output">{data}</div>
        </>
       ) : (
        <>
         <Lottie options={defaultOptionsLoading} height={400} width={400} />
         <div className="output">Generating Caption, Please wait ...</div>
        </>
       )}
     </center>
    </div>
  );
};
```

Home.css

```
*{
    margin: 0;
    padding: 0;
}

.upload-box{
    width: 70%;
    height: 12rem;
    background-image: url("data:image/svg+xml,%3csvg width='100%25' height='100%25'
xmlns='http://www.w3.org/2000/svg'%3e%3crect width='100%25' height='100%25' fill='none' rx='15'
ry='15' stroke='black' stroke-width='4' stroke-dasharray='6%2c 9' stroke-dashoffset='48' stroke-
linecap='square'/%3e%3c/svg%3e");
    border-radius: 15px;
    margin-top: 5rem;
    display: flex;
    flex-direction: column;
    align-items: center;
```

```
    justify-content: center;
}

label{
    margin: auto;
    font-size: 5rem;
}
label:hover{
    color: black

}
.image-box{
    margin-top: 1rem;
    width: 70%;
    height:8rem;
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 2rem;
    display: none;
}

.image-box2{
    margin-top: 1rem;
    width: 70%;
    height:8rem;
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 2rem;

}
.image-preview{
    display: flex;
    align-items: center;
    width: 25%;
    justify-content: space-evenly;


}
img{
    width: 100px;
    height:100px;
    margin-right: 1rem;
}
#output{
    font-size: x-large;
}
```

About.js

```
import React, { Component } from 'react';
import img1 from './images/img1.PNG';
import img2 from './images/img2.PNG';
import img3 from './images/img3.PNG';
import './About.css';
export class About extends Component {
   render() {
      return (
         <>

            <div className='main-content'>
               <div className='heading'>
                  <h1>
                     Image caption Generator
                  </h1>
                  <p>
                     Aim to build
                     an optimal system which can generate semantically and grammatically accurate
                     captions for an image.
                  </p>
               </div>
            </div>

            <div className='information-content'>
               <h2>Visualisation</h2>
               <div className='cards'>
                  <div className='card1'>
                     <img src={img1} alt="man in black shirt is playing guitar"/>
                     <p>"man in black shirt is playing guitar"</p>
                  </div>
                  <div className='card1'>
                     <img src={img2} alt="construction worker in orange safety vest is working on
road"/>
                     <p>"construction worker in orange safety vest is working on road"</p>
                  </div>
                  <div className='card1'>
                     <img src={img3} alt="two young girls are playing with lego toy" />
                     <p>"two young girls are playing with lego toy"</p>
                  </div>
               </div>
            </div>

            <div className='footercontent'>
               <p>Image Caption Generator</p>
            </div>
         </>
      );
   }
}
export default About;
```