Auto Image Captioning

Submitted in partial fulfillment of the requirements
of the degree of

T. E. Computer Engineering

By

Aishwarya Sreenivasan	**Roll No. 72**	**PID: 172006**
Ankit Jaiswal	**Roll No: 73**	**PID:172015**
Richie Jacob	**Roll No: 74**	**PID:172077**

Guide (s):

Mrs. Safa Hamdare
Professor

Department of Computer Engineering
St. Francis Institute of Technology
(Engineering College)

University of Mumbai

2019-2020

CERTIFICATE

This is to certify that the project entitled **"Auto Image Captioning"** is a bonafide work of **"Aishwarya Shreenivasan (Roll No: 72), Ankit Jaiswal (Roll No: 73), Richie Jacob (Roll No: 74)"** submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of T.E. in Computer Engineering

(Mrs. Safa Hamdare)
Guide

(Dr. Kavita Sonawane)
Head of Department

Project Report Approval for T.E.

This project report entitled ***Auto Image Captioning*** by **Mrs.*Aishwarya Sreenivasan,* *Mr.Ankit Jaiswal, Mr.Richie Jacob*** is approved for the degree of ***T.E. in Computer Engineering.***

Examiners

1.----------

2.----------

Date:

Place:Mumbai

Declaration

I/ We (Make changes in college copy. Individual copy it will be I and College copy it will be We) declare that this written submission represents my ideas in my own words and where other's ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)

Aishwarya Sreenivasan	Roll no: 72
Ankit Jaiswal	Roll no: 73
Richie Jacob	Roll no: 74

Date:

Abstract

Captioning images automatically is a task close to the heart of image understanding. There are various advantages if there is an application which automatically caption the scenes surrounded by them and revert back the caption as a plain message. In this project, we present a model based on CNN-LSTM neural networks which automatically detects the objects in the images and generates descriptions for the images. It uses various pre-trained models such as the VGG16 Model as well as CNN and LSTM to generate the captions. These models can perform two operations. The VGG16 Model is to detect objects in the image using Convolutional Neural Networks and the other is to caption the images using RNN based LSTM (Long Short Term Memory). Interface of the model is developed using flask rest API, which is a web development framework of python.

Caption generation is one of the interesting and focused areas of Artificial Intelligence which has many challenges to pass on. Caption generation involves various complex scenarios starting from picking the dataset, training the model, validating the model, creating pre-trained models to test the images, extracting the features of images and finally generating the captions.

Contents

List of Figures

List of Abbreviations

Sr. No.	Abbreviation	Expanded form
1.	DFD	Data Flow Diagram
2. 3. 4.	CNN LSTM RNN	Convolution Neural Networks Long Short Term Memory Cell Recurrent Neural Network

Chapter 1

Introduction

1.1 Description

Automatic image captioning refers to the problem of constructing a natural language description of an image. This task is more challenging than the image classification and object recognition task, because it not only requires detection of objects within the image, but also requires detection of their relationship, expression, and activity presented in the image. Furthermore, the perceived information must be translated to some human understandable natural language. The main obstacle is the task of detecting the salient visual information that comes naturally to humans. Application can be in search engines where images can be searched by sentence fragments. Apart from the practical applications, image captioning requires the machine learning model to learn image understanding which is a significant computer vision challenge. The image captioning model can be further extended to video captioning which also has many practical applications including alert systems for enhancing security.

Key steps of image captioning task include extracting salient high level features from an image, detecting objects from those features, detecting salient visual information (relationship, interaction, expression, activity) involving those objects, and finally generating a natural language description as a sequence of words to express the content of an image. Some existing works address the image captioning problem by concatenating modules that solve these steps. More recent line of work aims to build an end to end system that uses Convolutional Neural Network (CNN) for salient feature detection and on top of that a Recurrent Neural Network (RNN) that generates sequential words to construct image captions. Several recent methods also proposed semantic attention based neural models for image captioning.

Our approach is different from the existing approaches since we model the image captioning problem as a multi-task learning problem with image activity detection as the auxiliary task. Unlike previous work, our approach does not explicitly fuse the semantic output to the RNN hidden layers, instead, through the auxiliary task of activity detection within an image, a bias is induced to the RNN.

The approach is to use the VGG16 layer so that we remove the classification layer for image captioning topic and use various algorithms to reconstruct the requirements accordingly.

1.2 Problem Formulation

The problem introduces a captioning task, which requires a computer vision system to both localize and describe salient regions in images in natural language. The image captioning task generalizes object detection when the descriptions consist of a single word. Given a set of images and prior knowledge about the content find the correct semantic label for the entire image(s).
Input: An image.
Expected Output: Natural language description of the input image.

1.3 Motivation

Image captioning has many advantages in almost every complex area of Artificial Intelligence. The main use case of our model is to help visually impaired to understand the environment and make them easy to act according to the environment. As this is a complex task to do, with the help of pre trained models and powerful deep learning frameworks like Tensorflow and Keras, we made it possible. This is completely a Deep Learning project, which makes use of multiple Neural Networks like Convolutional Neural Network and Long Short Term Memory to detect objects and caption the images. To deploy our model as a web application, we have used Flask, which is a powerful Python's web framework.

1.4 Proposed Solution

Our model uses two different neural networks to generate the captions. The first neural network is Convolutional Neural Network (CNN), which is used to train the images as well as to detect the objects in the image with the help of various pre-trained models like VGG, Inception or YOLO. The second neural network used is Recurrent Neural Network (RNN) based Long Short Term Memory (LSTM), which is used to generate captions from the generated object keywords. As, there is a lot of data involved to train and validate the model, generalized machine learning algorithms will not work. Deep Learning has evolved from the recent times to solve the data constraints on Machine Learning algorithms. GPU based computing is required to perform the Deep Learning tasks more effectively.

Chapter 2

Review of Literature

Recent methods for object detection and recognition have significantly motivated the image captioning problem. One of the early non-neural approaches on describing images was done by Farhadi et al who proposed a method based on multi-label Markov random field involving an intermediate meaning space to generate short descriptive sentences from images. The proposed approach works in two phases — mapping the image to a meaning space in the format of <object, action, scene>, and mapping the meaning space to a sentence using some predefined templates. Kulkarni et al. also proposed a template based text generation method for describing images. The limitation in this approach is that it only describes the relative position of various objects detected in an image. Hence, the method is only capable of capturing the spatial relationship among objects. Being a neural model, our approach has much more capacity than these non-neural models and is able to learn from the provided captions to produce versatile captions.

More recent line of work on image captioning involves a deep convolutional neural network layer for high level feature extraction from images. Vinyals et al. proposed Neural Image Caption (NIC) — a generative model based on deep recurrent architecture that maximizes the likelihood of generating the target caption given an input image. In NIC, CNN is used as an image encoder to produce a fixed length feature vector to represent high level features of the input images. The idea is to chop off the final output layer of a CNN based image classification task and use the output of the last hidden layer as input to the caption generator recurrent network for sequence modeling. NIC combines pre-trained sub-networks for vision and language models. Our approach shares some similarity in the lower level of NIC, however, the NIC model is an end-to-end single objective task, where our approach involves multi-task learning. Moreover, NIC makes use of a language model that is trained from external corpora where we do not assume such external knowledge.

We use a slightly modified version of NIC as our baseline approach. We describe the NIC model in detail here. NIC uses a probabilistic model to maximize the probability of the correct caption. The formulation is provided in Equation 1. Here θ denotes the model parameters, I is the input image and S is the corresponding target caption. The tuple (I,S) denotes a training image and caption pair.

$$\theta^* = \text{argmax}_{\theta} \sum_{(I,S)} \log P(S|I;\theta) \qquad (1)$$

Since S denotes a sequence of words $< S_0,S_1,...,S_N >$ of length N, the joint probability $P(S|I;\theta)$ can be expressed using chain rule as shown in Equation 2.

$$\log P(S|I;\theta) = \sum_{t=1}^{t=N} \log P(S_t|I,S_0,S_1,...,S_{t-1};\theta) \qquad (2)$$

For image representation NIC uses CNN and for word representation word embedding model is used. For modeling the structure in Equation 2, NIC uses Long Short Term Memory (LSTM) [6], which is very suitable for sequence prediction. For training, the output of t-th LSTM is fed as the

input to (t + 1)-th LSTM. For prediction, the authors mention two approaches — (1) Sampling words from the probability distribution obtained in the LSTM output layer at each time step, and (2) BeamSearch that generates top k sentences at time $t - 1$ to generate sentences at time t and keeps top-k among them. The modified approach we use as the baseline in this report uses the ground truth captions instead of previous time step's LSTM output as input to the LSTM at next time step.

Karpathy et al. proposed a model that is built with a combination of CNN, bi-directional RNN, and a structured objective. Unlike other approaches that limit the image description to a sentence, the proposed approach aims at generating dense description of images. The key idea in their approach is to align sentence fragments in image description to corresponding image regions to better learn visual information in images. For text generation from input images, they introduced a multimodal RNN architecture. This model is particularly useful to generate descriptions of previously unseen combinations of known image regions. Instead of image regions, our proposed approach focuses on activity of an image that is overlooked by many state-of-the art approaches.

Xu et al. and You et al. proposed attention based image captioning models for better describing content of images. The approach focuses on spatial attention and the authors have shown visually how the trained model can fix its gaze on salient objects while generating sentences as captions. The model consists of CNN for extracting salient image features and RNN for learning word sequence generation. The key idea in this approach is incorporating attention that mimics the human visual system while constructing the caption. Unlike the static features that are generated at once from the CNN, the attention allows features to be produced dynamically. Since the RNN is capable of accepting sequential inputs, the dynamic salient features can improve sentence generation as the RNN can attend different objects during different temporal steps. The approach described in can attend multiple salient objects with differently assigned weights and dynamically switch attention among objects. A shortcoming of such attention based models is that they mostly attend to objects, but not activities within an image. Since objects will dominate activities in an image, without explicit supervision, it is unlikely that the salient features will represent activities. However, activities play an important role when it comes to caption generation. Our proposed approach can be augmented to such attention based models to attend activities.

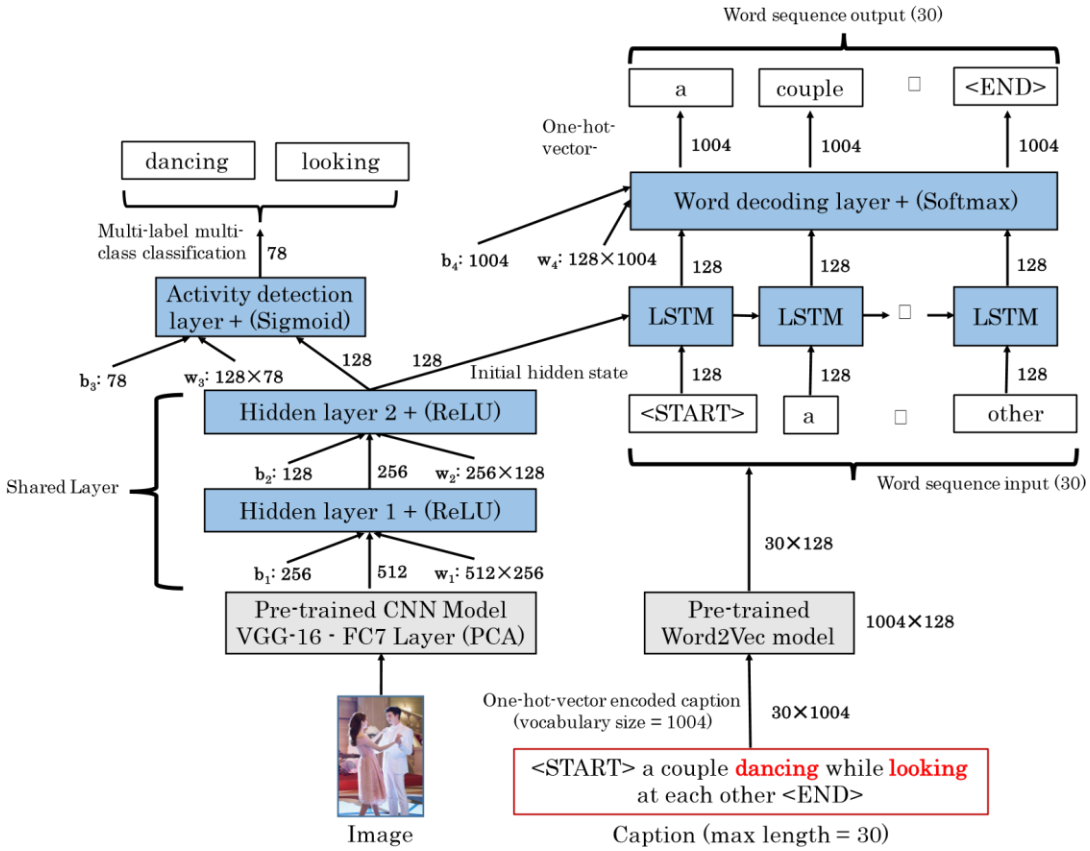

Fig1: Encoding and Decoding of Image Identifier and Description.

Chen et al. proposed a method for bidirectional mapping between images and captions using RNN. The model aims at both generating description from input image and reconstructing visual features from given description. Johnson et al. proposed a dense captioning model that both localizes and describes salient image regions via producing rich annotations of objects. Lu et al. [12] proposed a modified version of attention based captioning where the model is not forced to always attend the visual features. There exists several other notable works in image captioning [1, 14, 11, 17, 4] that include attention based mechanism, nearest neighbor based approach, bidirectional LSTM etc. Our proposed approach is novel in two ways — (1) it incorporates multi-task learning with a relevant and well suited auxiliary task of activity detection, and (2) it addresses the issue of extracting salient features representing activities within an image that was not addressed by any of the previous works.

Chapter 3

System Analysis

System Study and Environment.

In order to tackle the image captioning task, recent work shows it is in one's interest to utilize neural networks. This frequently used term dates back to the 1950s when notions such as the Perceptron Learning Algorithm were introduced. Modern neural networks draw on notions discovered in the era of a Perceptron. In this section, we rst de ne a neuron as a fundamental part of modern neural networks. Then we elaborate on Convolutional Networks and Recurrent Networks.

3.1 Functional Requirements

Major functionalities associated with the user are:

- Enable users to enter into his log by logging in the system with his specific username and password.
- The user can upload and find captions for the same as well as they can save it in the log.

Major functionalities associated with the system are as follows:

- Enable system to give caption after obtaining image identifier and tokenizing to find appropriate prediction.

Interface Requirements:

- Proper Format of the Image.
- Clear Image (Blurry will be accepted with a bad result).

3.2 Non Functional Requirements

3.2.1 Performance

The computer running the software did require a powerful GPU and TPU environment to train the models, it required a 64 bit operating system for execution of a program in order to call inbuilt packages like flask,sklearn etc.

3.2.2 Reliability

The system takes in the inputs as the image file without any error and predicts the expected response approximately so that the users of the system can get proper response.

3.2.3 Usability

The system is easy to handle and navigates in the most expected way with not much delays. The system interacts with the user in a very friendly manner making it easier for the users to use the system.

3.3 Specific Requirements

3.3.1 User Interfaces

- Front-end Software: Django,HTML, CSS, Web browser.
- Back-end Software: Sklearn,pandas and various other requirements that have been solved by using Google colab.

3.3.2 Hardware Requirements

- CPU Type : Intel Core or above
- Clock speed : 1.0GHz
- Ram size : 1GB and above
- Hard Disk capacity : 100GB and above
- Working keyboard

3.3.2 Software Requirements

- Operating System : Windows
- Python 3.5 or above

3.3.3 Communication Interfaces

This system will be completely based on a local system of the user.

3.4 UML Diagram

Unified Modeling Language:

The Unified Modeling Language permits the technologist to specific AN analysis model mistreatment the modeling notation that's ruled by a group of grammar linguistics and pragmatic rules. A UML system is diagrammatical mistreatment 5 completely different views that describe the system from a clearly different perspective. Every read is outlined by a group of diagrams, that is as follows. It represents the dynamic of behavior as elements of the system, portraying the interactions of assortment between varied structural components delineated within the user model and structural model read. Use case Diagrams represent the practicality of the system from a user's purpose of read. Use cases are used throughout needs induction and analysis to represent the practicality of the system. Use cases specialize in the behavior of the system from external purposes of read. Actors are external entities that move with the system. Samples of actors embody users like administrator, bank client …etc., or another system like central info.

Use Case Diagrams:

A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and the use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a "System" is something being developed and operated, such as a website. The "Actors" are people or entities operating under defined roles within the system.

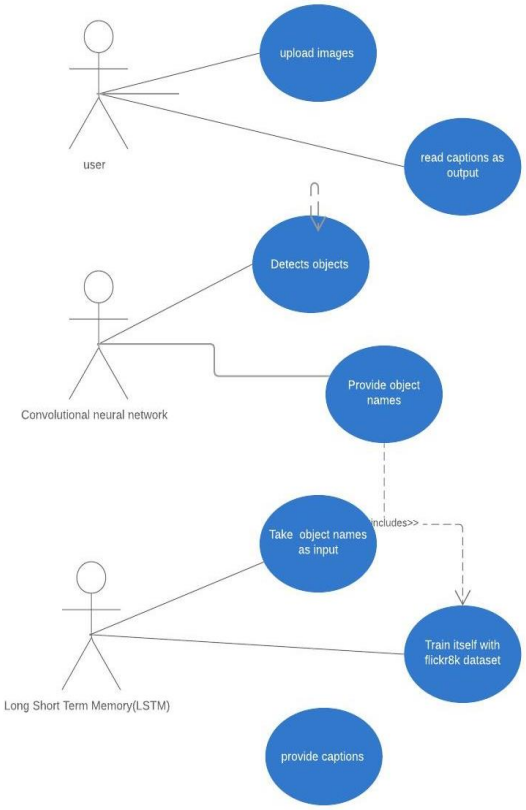

Fig 2: Use Case diagram

Chapter 4

Analysis Modeling

4.1 Class Diagram

Class diagrams are the backbone of virtually each object-oriented methodology as well as UML. They describe the static structure of a system. Categories represent associate degree abstraction of entities with common characteristics. Associations represent the relationships between categories.

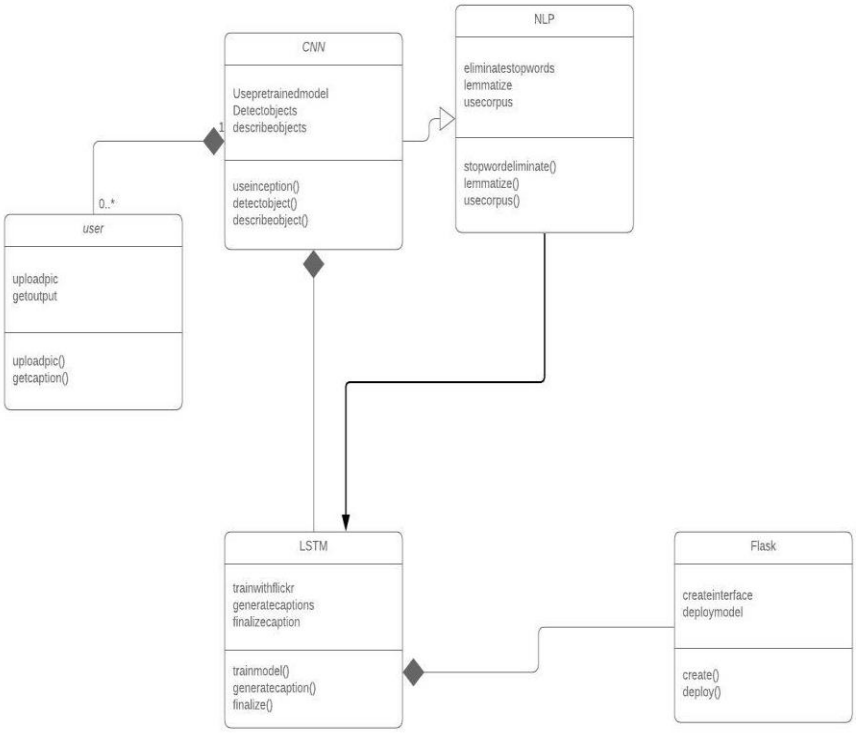

Fig 4.1: Class Diagram

Component Diagram:

An element diagram describes the organization of the physical elements in a very system. An element could be a physical building block of the system. It's pictured as a parallelogram with tabs.

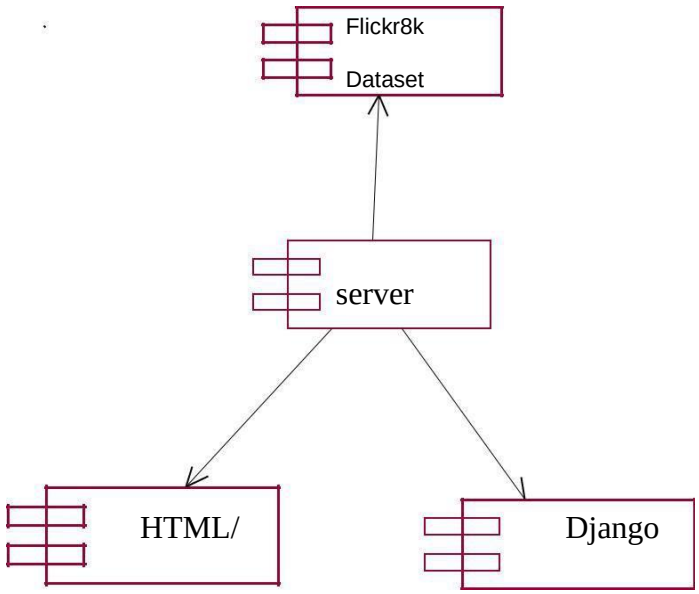

Fig 4.2: Component Diagram

Deployment Diagram:

Deployment diagrams depict the physical resources in an exceedingly system as well as nodes, components, and connections. A node may be a physical resource that executes code parts.

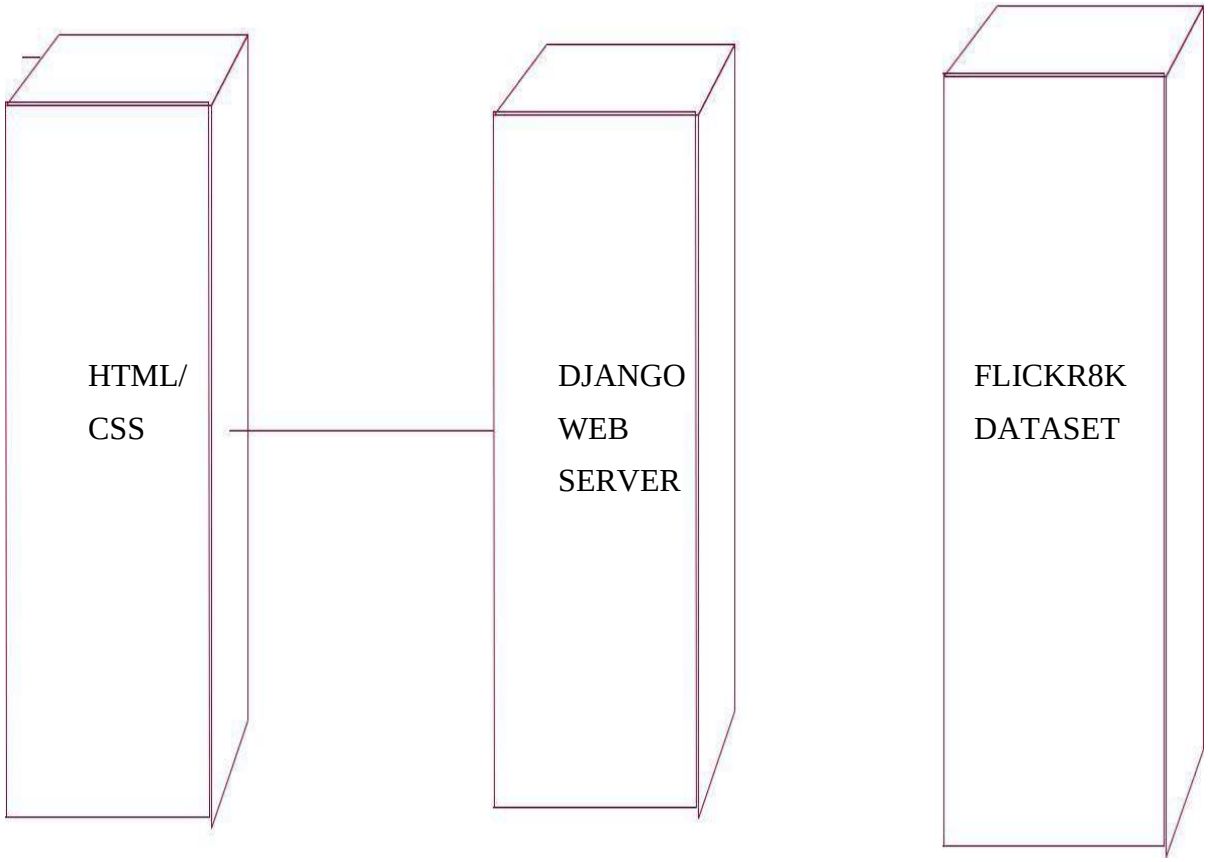

Fig 4.3: Deployment Diagram

Sequence Diagram:

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

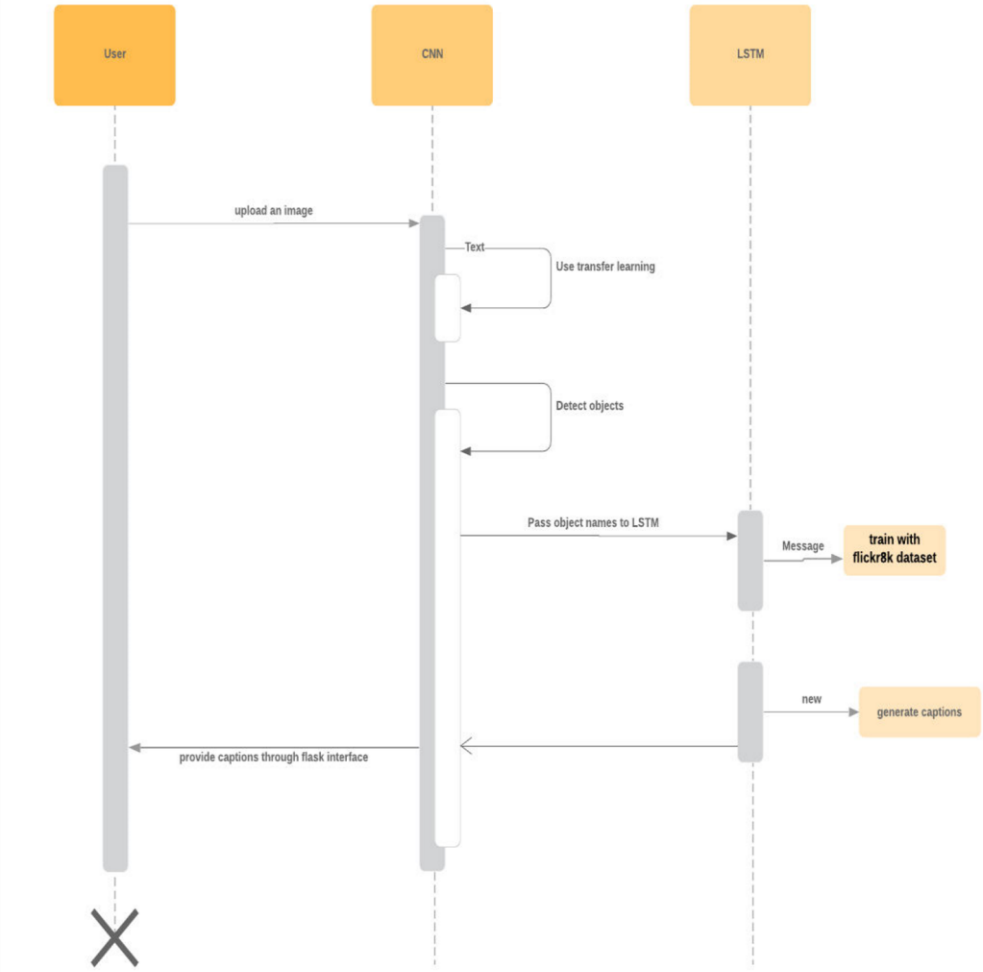

Fig 4.4: Sequence Diagram

Activity Diagram:

An activity diagram illustrates the dynamic nature of a system by modeling the flow of management from activity to activity. An activity represents AN operation on some category within the system that leads to an amendment within the state of the system. Typically, activity diagrams are accustomed model progress or business processes and internal operation. As a result of AN activity diagram may be a special quiet state chart diagram, it uses a number of constant modeling conventions.

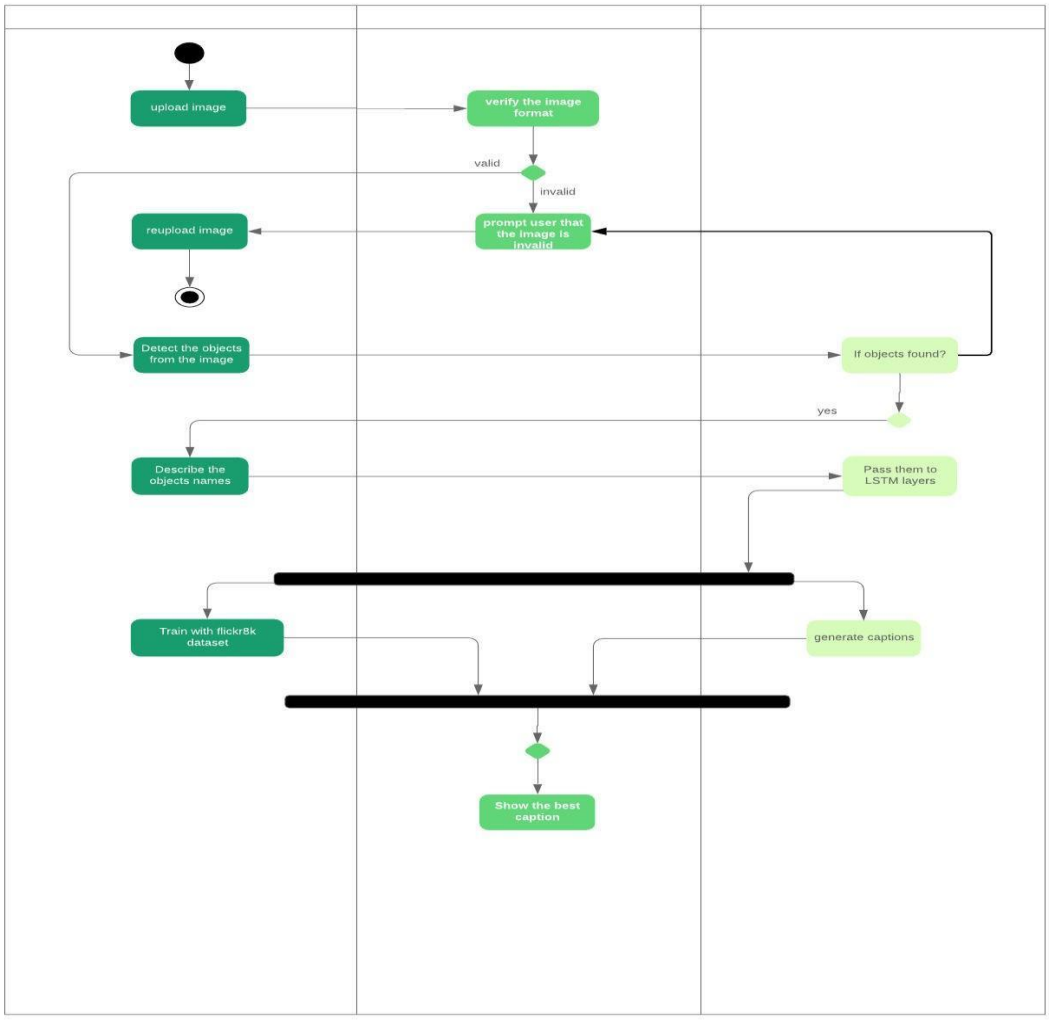

Fig 4.5: Activity Diagram

4.2 Functional Modelling

Data Flow Diagram:

A data-flow diagram is a way of representing a flow of a data of a process or a system The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops.

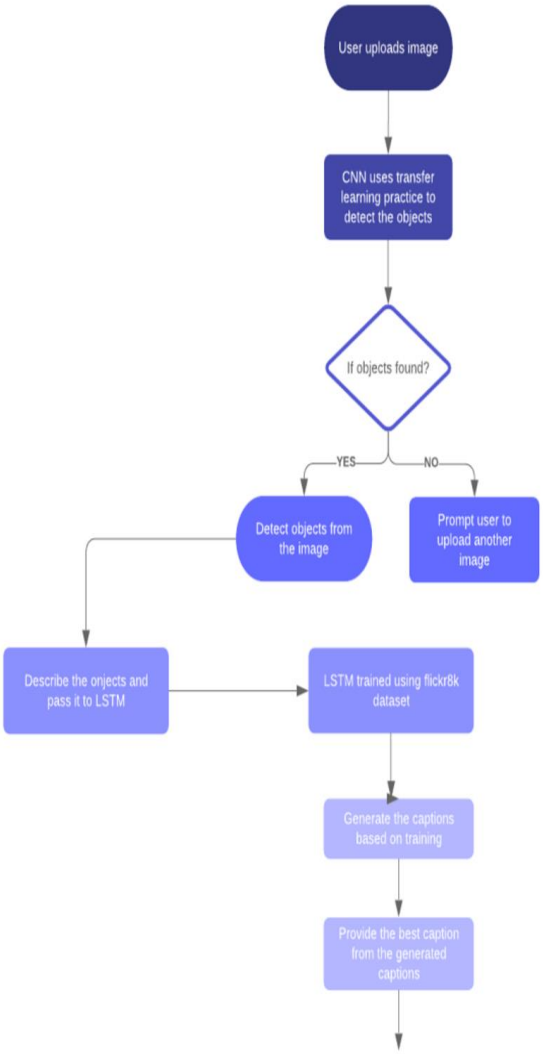

Fig 4.6: Data Flow Diagram

Chapter 5

5.Design

5.1 System Architecture

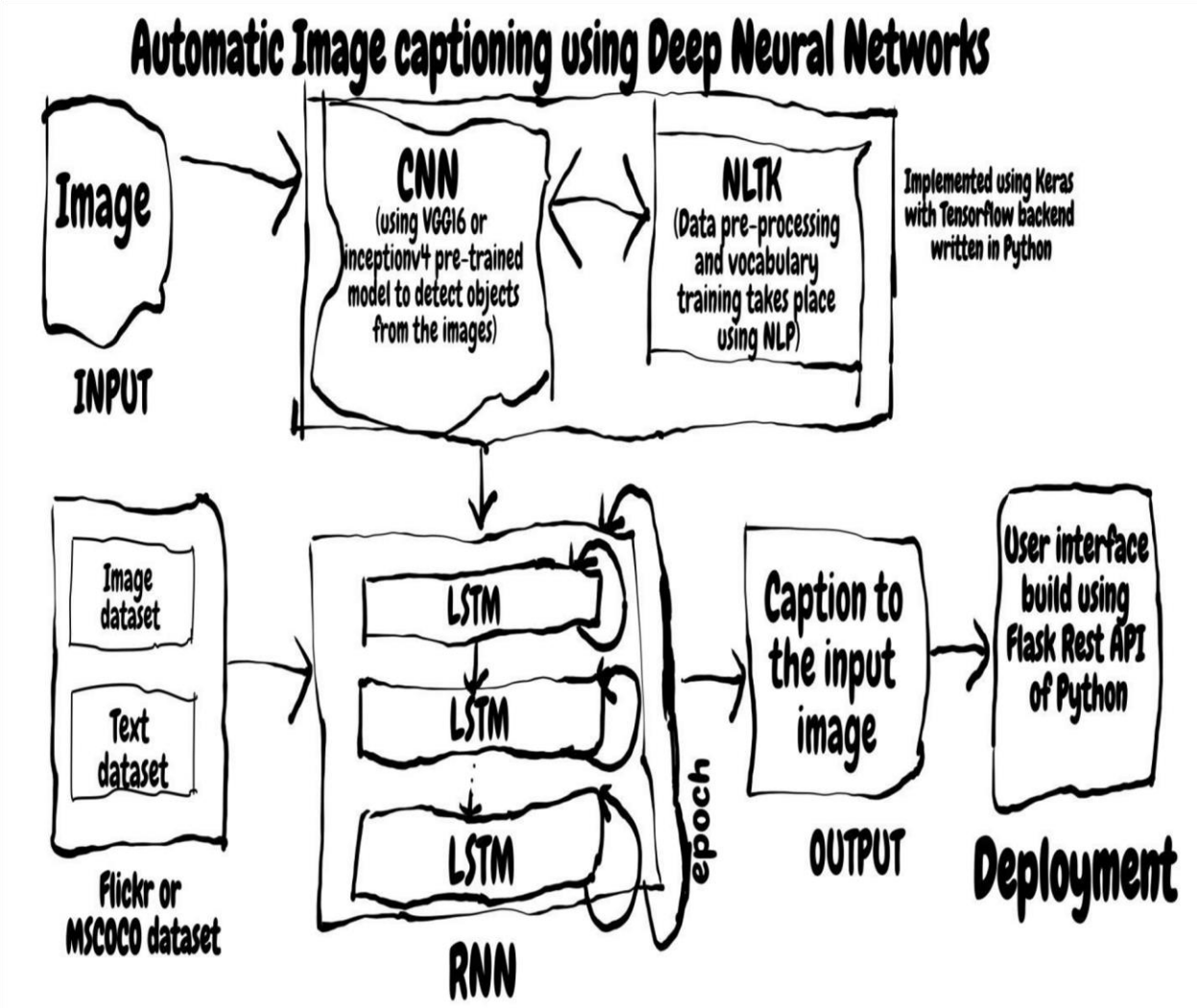

Fig 5.1: System Architecture

5.2 User Interface Design

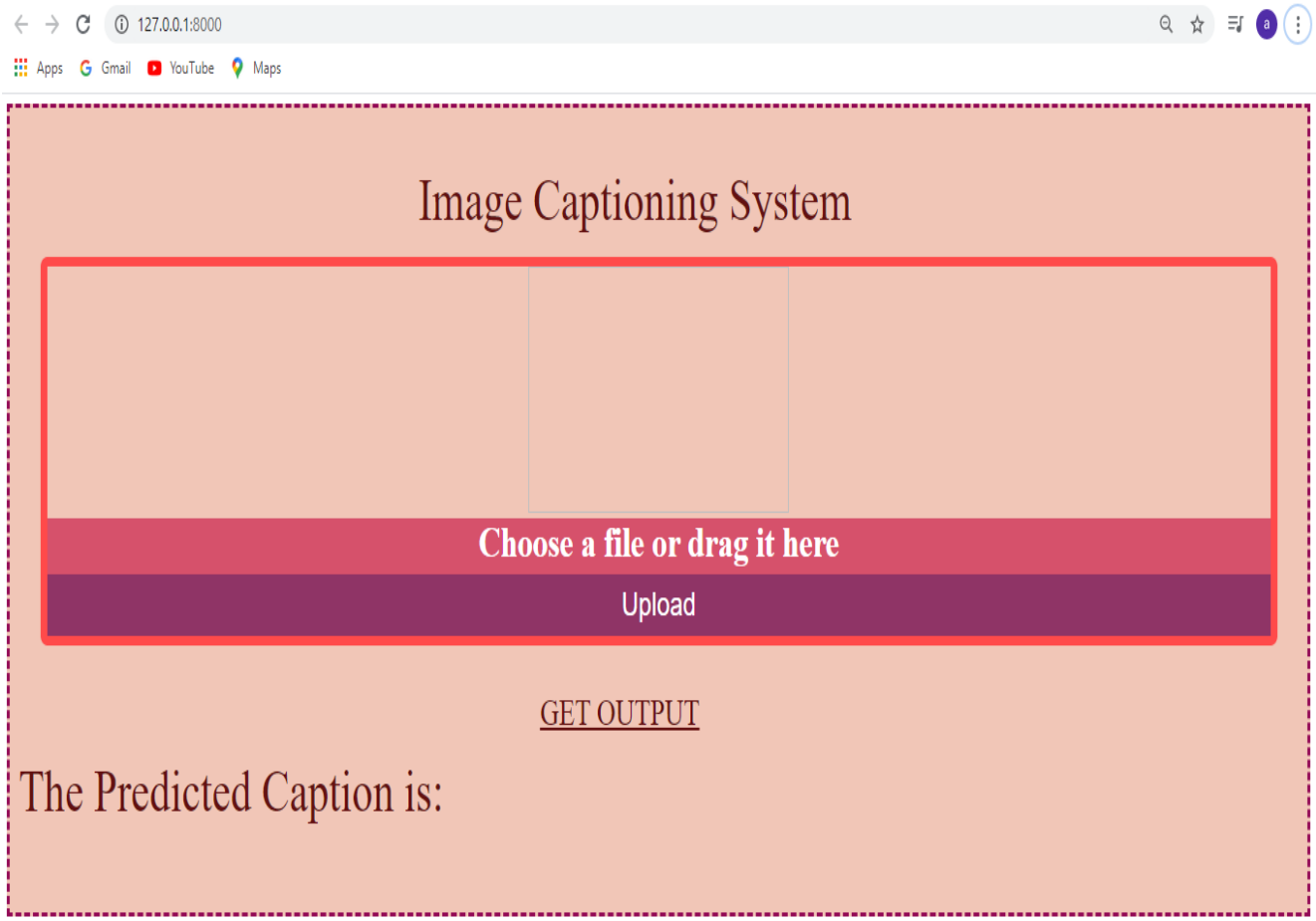

Fig: 5.2

Chapter 6

Implementation

6.1 Methods Used

➢ **CNN**

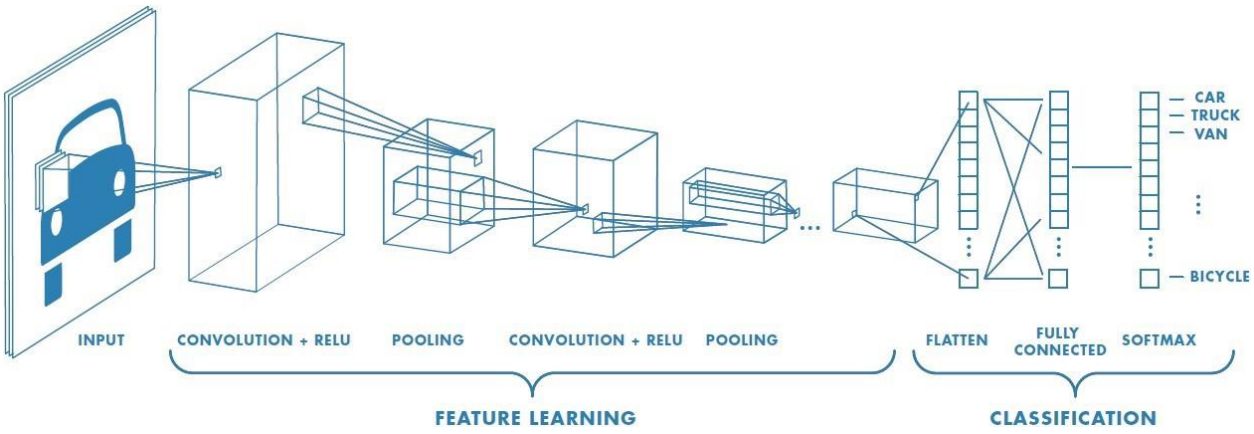

Fig: 6.1

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

The CNN consists of:

- **Convolution Layer**

  Purpose of convolution is to extract features from the input image. It preserves the spatial relationship between pixels by learning image features using small squares of input data. It is usually followed by Relu.

- **Relu**

  It is an element-wise operation that replaces all negative pixel values in the feature map by zero. Its purpose is to introduce non-linearity in the convolution network.

- **Pooling**

  Also called downsampling, reduces the dimensionality of each feature map but retains important data.

- **Fully-Connected Layer**

  Its a multi-layer perceptron that uses softmax function in the output layer. Its purpose is to use features from previous layers by classifying the input image into various classes based on training data.

➢ **LSTM**

LSTM stands for Long short term memory, they are a type of RNN (recurrent neural network) which is well suited for sequence prediction problems. Based on the previous text, we can predict what the next word will be. It has proven itself effective from the traditional RNN by overcoming the limitations of RNN which had short term memory. LSTM can carry out relevant information throughout the processing of inputs and with a forget gate, it discards non-relevant information.

➢ **Workflow**

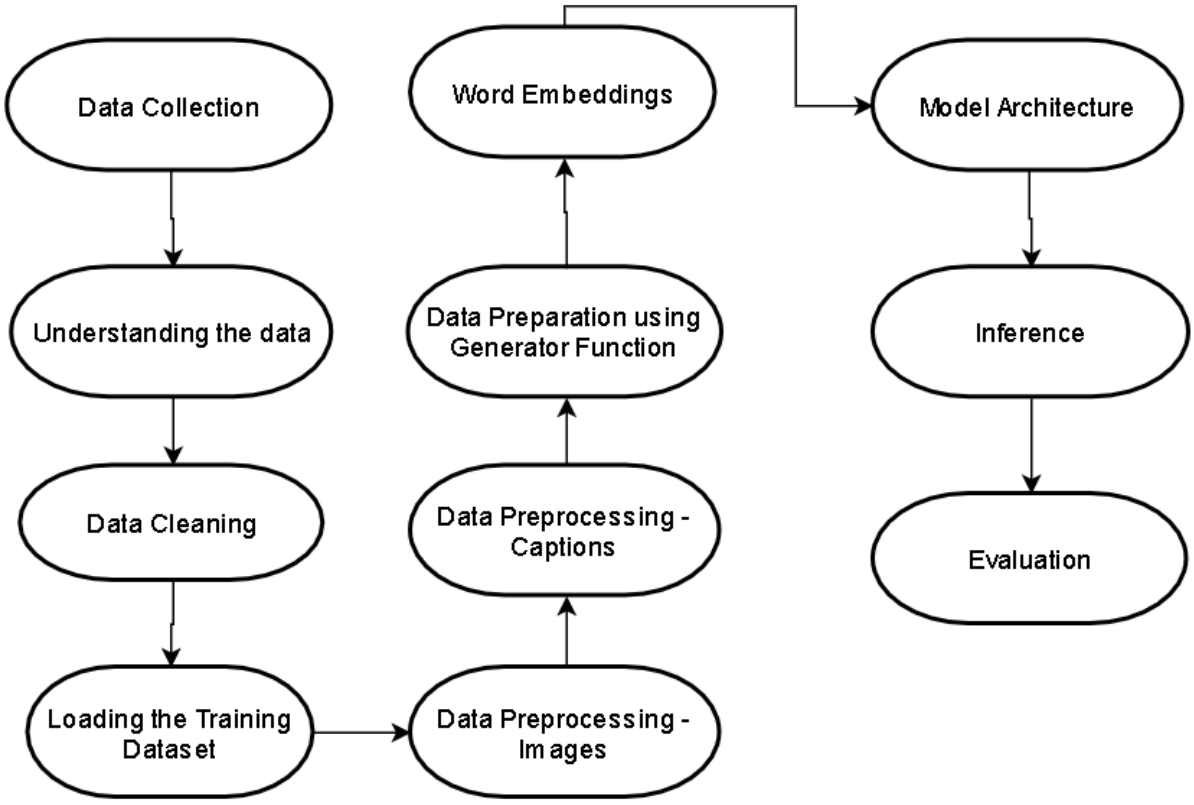

Fig: 6.2

6.2 Working of Project

Code:

```
"""Image_caption_Project.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1C4bnUlgqCw7Bvb7bIBcOmqsH63LVTewQ
"""

from os import listdir
from pickle import dump
from keras.applications.vgg16 import VGG16
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.models import Model

def extract_features(directory):
        model = VGG16()
        model.layers.pop()
        model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
        print(model.summary())
        features = dict()
        for name in listdir(directory):
                filename = directory + '/' + name
                image = load_img(filename, target_size=(224, 224))
                image = img_to_array(image)
                image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
                image = preprocess_input(image)
                feature = model.predict(image, verbose=0)
                image_id = name.split('.')[0]
                features[image_id] = feature
                print('>%s' % name)
        return features

def to_vocabulary(descriptions):
        all_desc = set()
        for key in descriptions.keys():
                [all_desc.update(d.split()) for d in descriptions[key]]
        return all_desc

def save_descriptions(descriptions, filename):
        lines = list()
        for key, desc_list in descriptions.items():
                for desc in desc_list:
                        lines.append(key + ' ' + desc)
        data = '\n'.join(lines)
        file = open(filename, 'w')
        file.write(data)
        file.close()

filename = 'Flickr8k.token.txt'
doc = load_doc(filename)
descriptions = load_descriptions(doc)
print('Loaded: %d ' % len(descriptions))
clean_descriptions(descriptions)
vocabulary = to_vocabulary(descriptions)
print('Vocabulary Size: %d' % len(vocabulary))
save_descriptions(descriptions, 'descriptions.txt')

from pickle import load

def load_doc(filename):
        file = open(filename, 'r')
        text = file.read()
        file.close()
        return text
```

```
def load_set(filename):
        doc = load_doc(filename)
        dataset = list()
        for line in doc.split('\n'):
                if len(line) < 1:
                        continue
                identifier = line.split('.')[0]
                dataset.append(identifier)
        return set(dataset)

def load_clean_descriptions(filename, dataset):
        doc = load_doc(filename)
        descriptions = dict()
        for line in doc.split('\n'):
                tokens = line.split()
                image_id, image_desc = tokens[0], tokens[1:]
                if image_id in dataset:
                        if image_id not in descriptions:
                                descriptions[image_id] = list()
                        desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
                        descriptions[image_id].append(desc)
        return descriptions

def load_photo_features(filename, dataset):
        all_features = load(open(filename, 'rb'))
        features = {k: all_features[k] for k in dataset}
        return features

filename = 'Flickr_8k.trainImages.txt'
train = load_set(filename)
print('Dataset: %d' % len(train))
train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))
train_features = load_photo_features('features.pkl', train)
print('Photos: train=%d' % len(train_features))
train_descriptions
```

```
from numpy import array
import tensorflow
from pickle import load
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.utils import plot_model
from keras.models import Model
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


def load_doc(filename):
        file = open(filename, 'r')
        text = file.read()
        file.close()
        return text

def load_set(filename):
        doc = load_doc(filename)
        dataset = list()
        for line in doc.split('\n'):
                if len(line) < 1:
                        continue
                identifier = line.split('.')[0]
                dataset.append(identifier)
        return set(dataset)
```

```
def load_clean_descriptions(filename, dataset):
        doc = load_doc(filename)
        descriptions = dict()
        for line in doc.split('\n'):
                tokens = line.split()
                image_id, image_desc = tokens[0], tokens[1:]
                if image_id in dataset:
                        if image_id not in descriptions:
                                descriptions[image_id] = list()
                        desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
                        descriptions[image_id].append(desc)
        return descriptions

def load_photo_features(filename, dataset):
        all_features = load(open(filename, 'rb'))
        features = {k: all_features[k] for k in dataset}
        return features

def to_lines(descriptions):
        all_desc = list()
        for key in descriptions.keys():
                [all_desc.append(d) for d in descriptions[key]]
        return all_desc

def create_tokenizer(descriptions):
        lines = to_lines(descriptions)
        tokenizer = Tokenizer()
        tokenizer.fit_on_texts(lines)
        return tokenizer

tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: %d' % vocab_size)

def max_length(descriptions):
        lines = to_lines(descriptions)
        return max(len(d.split()) for d in lines)

def create_sequences(tokenizer, max_length, desc_list, photo):
        X1, X2, y = list(), list(), list()
        for desc in desc_list:
                seq = tokenizer.texts_to_sequences([desc])[0]
                for i in range(1, len(seq)):
                        in_seq, out_seq = seq[:i], seq[i]
                        in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                        out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                        X1.append(photo)
                        X2.append(in_seq)
                        y.append(out_seq)
        return array(X1), array(X2), array(y)

def define_model(vocab_size, max_length):
        inputs1 = Input(shape=(4096,))
        fe1 = Dropout(0.5)(inputs1)
        fe2 = Dense(256, activation='relu')(fe1)

        inputs2 = Input(shape=(max_length,))
        se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
        se2 = Dropout(0.5)(se1)
        se3 = LSTM(256)(se2)

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

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

        print(model.summary())
        return model
```

```
def data_generator(descriptions, photos, tokenizer, max_length):
	while 1:
		for key, desc_list in descriptions.items():
			photo = photos[key][0]
			in_img, in_seq, out_word = create_sequences(tokenizer, max_length, desc_list, photo)
			yield [[in_img, in_seq], out_word]

filename = 'Flickr_8k.trainImages.txt'
train = load_set(filename)
print('Dataset: %d' % len(train))

train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))

train_features = load_photo_features('features.pkl', train)
print('Photos: train=%d' % len(train_features))

tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: %d' % vocab_size)

max_length = max_length(train_descriptions)
print('Description Length: %d' % max_length)

model = define_model(vocab_size, max_length)

epochs = 20
steps = len(train_descriptions)
for i in range(epochs):
	generator = data_generator(train_descriptions, train_features, tokenizer, max_length)
	model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
	model.save('model_' + str(i) + '.h5')
```

```
from numpy import argmax
from pickle import load
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import load_model
from nltk.translate.bleu_score import corpus_bleu

def load_doc(filename):
	file = open(filename, 'r')
	text = file.read()
	file.close()
	return text

def load_set(filename):
	doc = load_doc(filename)
	dataset = list()
	for line in doc.split('\n'):
		if len(line) < 1:
			continue
		identifier = line.split('.')[0]
		dataset.append(identifier)
	return set(dataset)

def load_clean_descriptions(filename, dataset):
	doc = load_doc(filename)
	descriptions = dict()
	for line in doc.split('\n'):
		tokens = line.split()
		image_id, image_desc = tokens[0], tokens[1:]
		if image_id in dataset:
			if image_id not in descriptions:
				descriptions[image_id] = list()
			desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
			descriptions[image_id].append(desc)
	return descriptions

def load_photo_features(filename, dataset):
	all_features = load(open(filename, 'rb'))
	features = {k: all_features[k] for k in dataset}
	return features
```

```
def evaluate_model(model, descriptions, photos, tokenizer, max_length):
        actual, predicted = list(), list()
        for key, desc_list in descriptions.items():
                yhat = generate_desc(model, tokenizer, photos[key], max_length)
                references = [d.split() for d in desc_list]
                actual.append(references)
                predicted.append(yhat.split())
        print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
        print('BLEU-2: %f' % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
        print('BLEU-3: %f' % corpus_bleu(actual, predicted, weights=(0.3, 0.3, 0.3, 0)))
        print('BLEU-4: %f' % corpus_bleu(actual, predicted, weights=(0.25, 0.25, 0.25, 0.25)))

filename = 'Flickr_8k.trainImages.txt'
train = load_set(filename)
print('Dataset: %d' % len(train))

train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))

tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: %d' % vocab_size)

max_length = max_length(train_descriptions)
print('Description Length: %d' % max_length)

filename = 'Flickr_8k.testImages.txt'
test = load_set(filename)
print('Dataset: %d' % len(test))

test_descriptions = load_clean_descriptions('descriptions.txt', test)
print('Descriptions: test=%d' % len(test_descriptions))

test_features = load_photo_features('features.pkl', test)
print('Photos: test=%d' % len(test_features))

filename = 'model_18.h5'
model = load_model(filename)

evaluate_model(model, test_descriptions, test_features, tokenizer, max_length)

from pickle import load
from numpy import argmax
from keras.preprocessing.sequence import pad_sequences
from keras.applications.vgg16 import VGG16
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.models import Model
from keras.models import load_model

def extract_features(filename):
        model = VGG16()
        model.layers.pop()
        model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
        image = load_img(filename, target_size=(224, 224))
        image = img_to_array(image)
        image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
        image = preprocess_input(image)
        feature = model.predict(image, verbose=0)
        return feature

tokenizer = load(open('tokenizer.pkl', 'rb'))
max_length = 34
model = load_model('model_18.h5')
photo = extract_features('Sample_Image.jpg')
description = generate_desc(model, tokenizer, photo, max_length)
print(description)

query = description
stopwords = ['startseq','endseq']
querywords = query.split()

resultwords  = [word for word in querywords if word.lower() not in stopwords]
result = ' '.join(resultwords)

print(result)
```

Output:

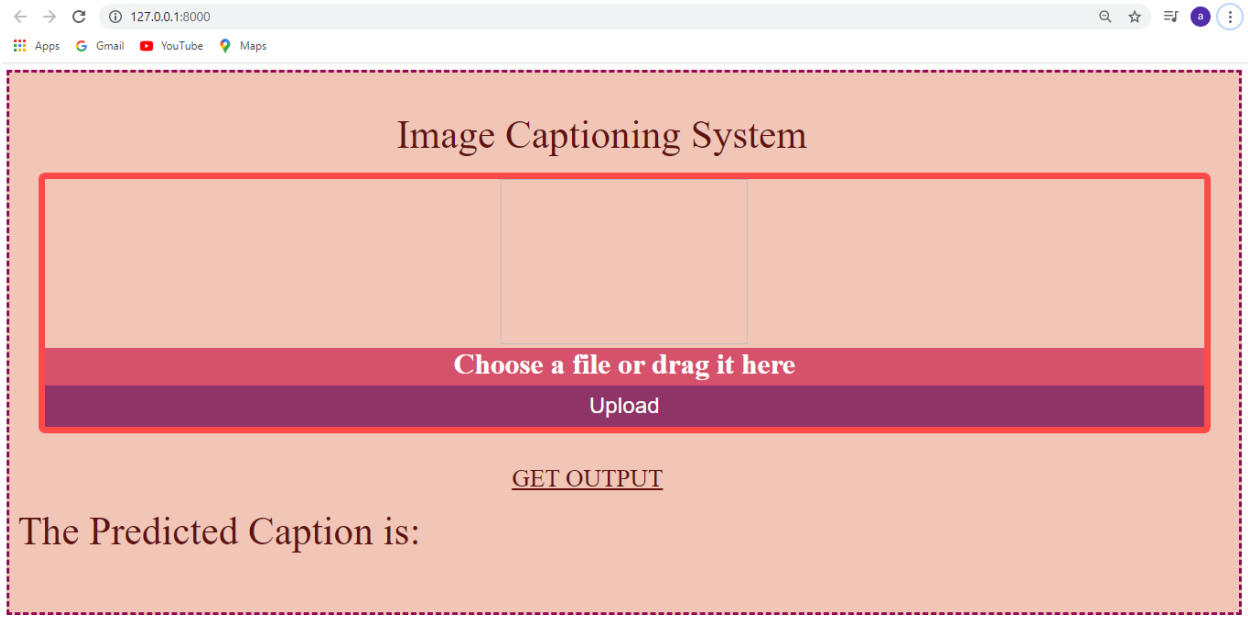

Fig: 6.3

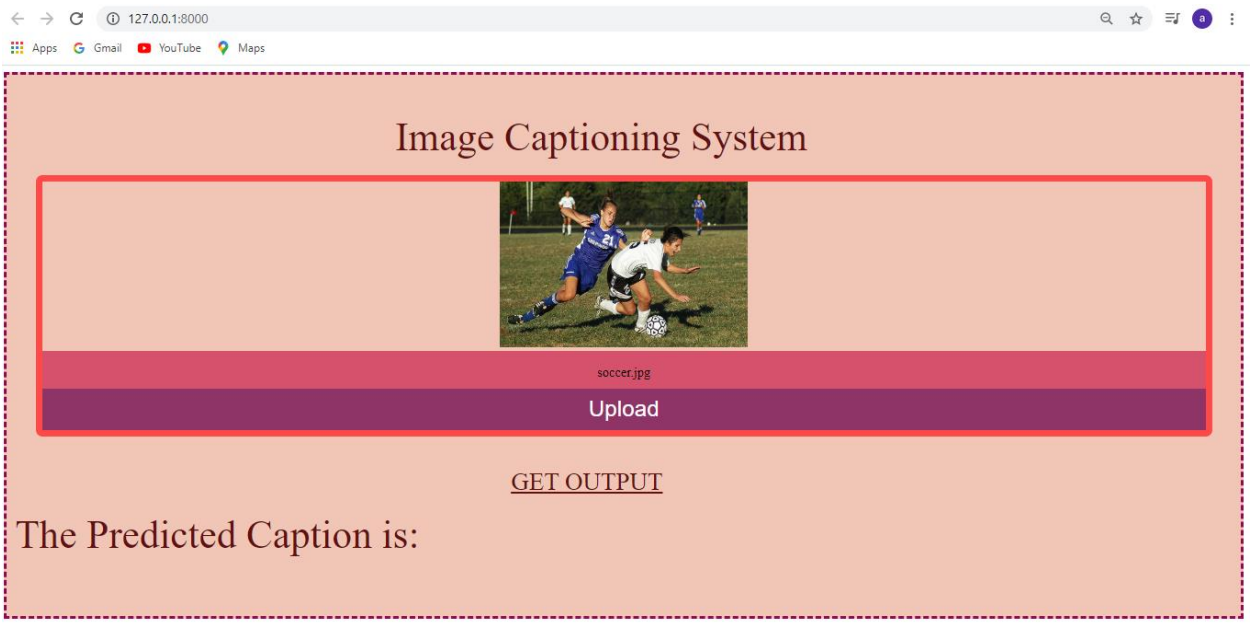

Fig: 6.4

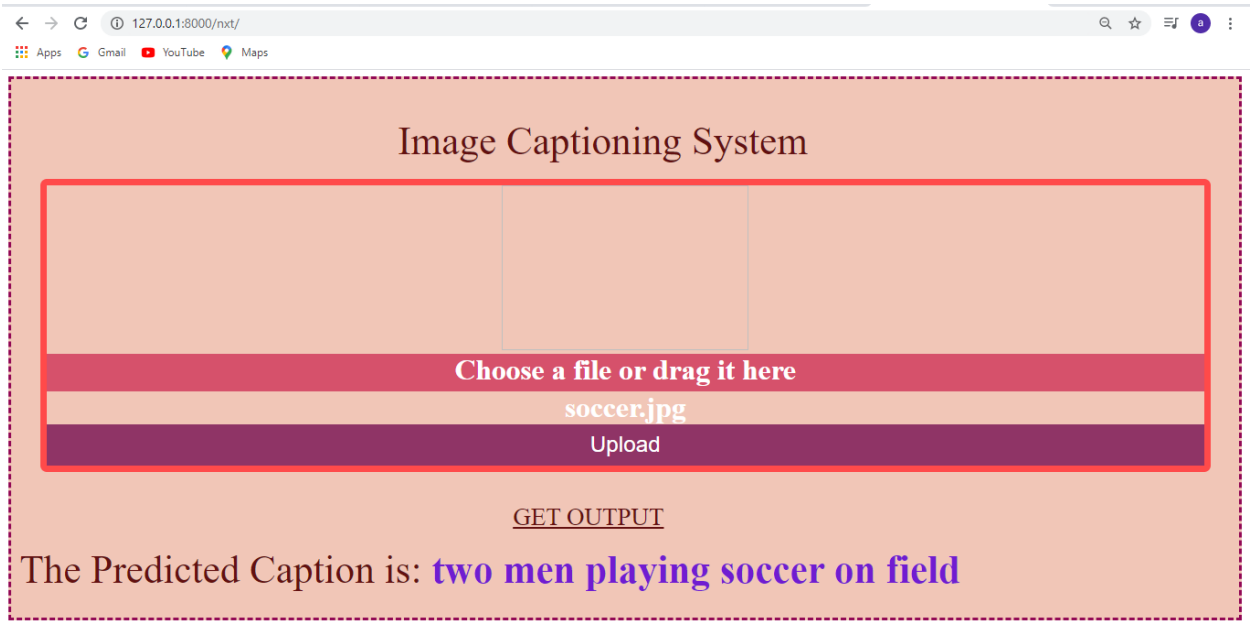
127.0.0.1:8000/nxt/
Apps
Gmail
YouTube
Maps
Image Captioning System
Choose a file or drag it here
soccer.jpg
Upload
GET OUTPUT
The Predicted Caption is: two men playing soccer on field

Fig: 6.5

Chapter 7

Conclusion and Future Scope.

Image captioning has many advantages in almost every complex area of Artificial Intelligence. The main use case of our model is to help visually impaired to understand the environment and make them easy to act according to the environment. As this is a complex task to do, with the help of pre trained models and powerful deep learning frameworks like Tensorflow and Keras, we made it possible. This is completely a Deep Learning project, which makes use of multiple Neural Networks like Convolutional Neural Network and Long Short Term Memory to detect objects and caption the images. To deploy our model as a web application, we have used Flask, which is a powerful Python's web framework.

We are going to extend our work in the next higher level by enhancing our model to generate captions even for the live video frame. Our present model generates captions only for the image, which itself is a complex task and captioning live video frames is much complex to create. This is completely GPU based and captioning live video frames cannot be possible with the general CPUs. Video captioning is a popular research area in which it is going to change the lifestyle of the people with the use cases being widely usable in almost every domain. It automates the major tasks like video surveillance and other security tasks.

References

[1] P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, and L. Zhang. Bottom-up and top-down attention for image captioning and VQA. *CoRR*, abs/1707.07998, 2017.

[2] X. Chen, H. Fang, T. Lin, R. Vedantam, S. Gupta, P. Dollár, and C. L. Zitnick. Microsoft COCO captions: Data collection and evaluation server. *CoRR*, abs/1504.00325, 2015.

[3] X. Chen and C. L. Zitnick. Mind's eye: A recurrent visual representation for image caption generation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015 , Boston, MA, USA, June 7-12, 2015*, pages 2422–2431, 2015.

[4] J. Devlin, S. Gupta, R. Girshick, M. Mitchell, and C. L. Zitnick. Exploring nearest neighbor approaches for image captioning. *arXiv preprint arXiv:1505.04467*, 2015.

[5] A. Farhadi, S. M. M. Hejrati, M. A. Sadeghi, P. Young, C. Rashtchian, J. Hockenmaier, and D. A. Forsyth. Every picture tells a story: Generating sentences from images. In *Computer Vision - ECCV 2010, 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV*, pages 15–29, 2010.

[6] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780 , Nov. 1997.

[7] J. Johnson, A. Karpathy, and L. Fei-Fei. Densecap: Fully convolutional localization networks for dense captioning. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 4565–4574, 2016.

[8] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):664–676, 2017.

[9] A. Karpathy and F. Li. Deep visual-semantic alignments for generating image descriptions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 3128–3137, 2015.

[10] G. Kulkarni, V. Premraj, V. Ordonez, S. Dhar, S. Li, Y. Choi, A. C. Berg, and T. L. Berg. Babytalk: Understanding and generating simple image descriptions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(12):2891–2903, 2013.

[11] C. Liu, J. Mao, F. Sha, and A. L. Yuille. Attention correctness in neural image captioning. In *AAAI*, pages 4176–4182, 2017.

[12] https://towardsdatascience.com/image-captioning-with-keras-teaching-computers-to-describe-pictures-c88a46a311b8

[13] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[14] K. Tran, X. He, L. Zhang, J. Sun, C. Carapcea, C. Thrasher, C. Buehler, and C. Sienkiewicz. Rich image captioning in the wild. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2016.

[15] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015 , Boston, MA, USA, June 7-12, 2015*, pages 3156–3164, 2015.

[16] https://data-flair.training/blogs/python-based-project-image-caption-generator-cnn/

[17] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: Lessons learned from the 2015 mscoco image captioning challenge. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):652–663, April 2017.

[18] C. Wang, H. Yang, C. Bartz, and C. Meinel. Image captioning with deep bidirectional lstms. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 988–997. ACM, 2016.

[19] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2048–2057, 2015.

[20] You, H. Jin, Z. Wang, C. Fang, and J. Luo. Image captioning with semantic attention. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Acknowledgements

We take the opportunity to thank all those people who have helped and guided us through this project and make this experience worthwhile for us. We wish to sincerely thank our reverend **Bro. Jose Thuruthiyil** and principal **Dr. Sincy George** for giving us this opportunity for making a project in Third Year of Engineering. We would also like to thank HOD of Computer department **Dr. Kavita Sonawane** and all teaching and non teaching staff for their immense support and cooperation.

Last but not the least we would like to thank **Mrs. Safa Hamdare** for guiding us throughout the project and encouraging us to explore in this domain.