VOICE BOT FOR ASSISTIVE VISION

MINI PROJECT REPORT

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS TO

RGUKT- SRIKAKULAM

FOR THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY IN

COMPUTER SCIENCE AND ENGINEERING

Submitted By:

K MANGA RAJU	**S190317**
T PRAVALLIKA	**S190318**
A DEEPIKA	**S190334**
J ANUSHA	**S190336**

Under the Esteemed Guidance of

Mrs.K.Sita Durga

Assistant Professor

Department of Computer Science and Engineering,

RGUKT-SRIKAKULAM, ETCHERLA- July 2023

CERTIFICATE

This is to certify that the mini project report titled “VOICE BOT FOR ASSISTIVE VISION” was successfully completed by K MANGA RAJU (**S190317**), T PRAVALLIKA (**S190318**), A DEEPIKA (**S190334**) , J ANUSHA (**S190336**) under the guidance of Mrs.K.SITA DURGA Assistant Professor In partial fulfilment of the requirements for the Mini Project in Computer Science and Engineering of **Rajiv Gandhi University of Knowledge Technologies** under my guidance and output of the work carried out is satisfactory.

Project Guide	**Head of the Department**
Mrs.K.Sita Durga	Mrs.Ch.Lakshmi Bala
Assistant Professor	Assistent Professor

DECLARATION

We declared that this thesis work titled “VOICE BOT FOR ASSISTIVE VISION” is carried out by me during the year 2023-2024 in partial fulfilment of the requirements for the Mini Project in **Computer Science and Engineering.**

We further declare that this dissertation has not been submitted elsewhere for any Degree. The matter embodied in this dissertation report has not been submitted elsewhere for any other degree. Furthermore, the technical details furnished in various chapters of this thesis are purely relevant to the above project and there is no deviation from the theoretical point of view for design, development and implementation.

K. Manga Raju (S190317)

T. Pravallika (S190318)

A. Deepika (S190334)

J. Anusha (S190336)

ACKNOWLEDGEMENT

We would like to articulate my profound gratitude and indebtedness to our project guide **Mrs.Sita Durga** Mam, who has always been a constant motivation and guiding factor throughout the project time. It has been a great pleasure for us to get an opportunity to work under her guidance and complete the thesis work successfully.

We wish to extend our sincere thanks to **Mrs.Ch.Lakshmi Bala** mam Head of the Computer Science and Engineering Department, for her constant encouragement throughout the project.

We are also grateful to other members of the department without their support our work would have not been carried out so successfully.

I thank one and all who have rendered help to me directly or indirectly in the completion of my thesis work .

Project Associate

K. Manga Raju (S190317)

T. Pravallika (S190318)

A. Deepika (S190334)

J. Anusha (S190336)

ABSTRACT

Visually challenged people find it difficult to lead their life as they cannot see things and scenes around them like normal people. Blind people are always in need of a person to help them know about what is happening around. Technologies like mobile, camera have become a part of our daily life. So, these technologies can be used to make visually challenged people be aware of what is happening around them and enjoy life like normal people.The system aims to recognize the context of an image and describe them in natural language like English and convert it to voice so that visually challenged people can know about their surroundings. The features from the image are extracted with the help of Convolutional Neural Network(CNN). Long Short-Term Memory (LSTM), which is well suited for predicting the sequence is fed with the word embeddings of partial sentences.This approach makes use of merge architecture for generating image captions. The caption generated is converted to audio using the Google Text To Speech (GTTS) module in python.Finally the code is deployed as a REST API using the FLASK

Table of Contents:

1.INTRODUCTION

1.1 Introduction

Many people with disabilities still find it difficult to fully participate in society, but they are still a valuable and important part of our society. As a result, they have been hampered in their social and economic advancement, and they have little or no desire to contribute to our economic prosperity. Our goal is to assist in bridging this ever-widening gap between the two groups. These technological advancements will assist us in achieving this goal. A person without visual impairments can deduce the scene description and content of an image, but the blind in our society do not have this ability. This ability to provide visual content descriptions in the form of naturally spoken sentences could be extremely beneficial to the visually impaired. If you want to imagine a world where no one is limited by their visual abilities, you can have access to the visual medium without having to see the objects themselves. Their goal is to use an automated method of capturing visual content and producing natural language sentences to empower the visually impaired. This ability was one of the most difficult for a computer to achieve on its own before recent advances in the field of computer vision. Image descriptions are therefore more difficult than object recognition and classification because they must capture more than just the objects themselves. To provide a visual representation and understanding, the visual and linguistic models must be understood.

1.2 Problem Statement

People with visual impairments struggle to understand images in our digital world. Current tools don't provide real-time, accurate descriptions, making daily tasks harder. There is a need for a new technology that describes images and turns them into audio. This would help visually impaired individuals understand their surroundings better.

1.3 Objective

The main objective of the project Voice Bot for Assistive Vision is to develop a system that can automatically generate descriptive captions for images and convert them in the form of audio to assist individuals with visual impairments.

1.4.Importance

For the past few years, researchers have been focusing on the issue of translating visual content into descriptions in natural language forms. They are vulnerable to attack and have a limited set of capabilities because of certain constraints.

A new image captioning model known as "domain specific image caption generator" replaces the general caption's specific words with those that are specific to the domain. This model is referred to as a "domain-specific image caption generator" (DSIG). The image caption generator was put to the test in terms of both quality and quantity. This model does not allow for the implementation of a semantic ontology from beginning to end.

1.5 Motivation For The Work

This project stems from the desire to enhance the quality of life for visually challenged individuals. By providing them with a means to understand their surroundings through image recognition and natural language descriptions, we aim to empower them with greater independence and a more comprehensive understanding of their environment.

2.LITERATURE SURVEY

No.	Authors	Research Paper	Publication Year	Dataset and methodology	Conclusions
1.	Pranay Mathur, Aman Gill, Nand Kumar Bansode, Anurag Mishra	Camera2Caption: A real-time image caption generator	2017	Dataset: MSCOCO Method: Advanced deep reinforcement learning based on NLP and Computer vision	The model proposed generates the real time environment high quality captions with the help of tensorflow.
2.	Simao Herdade, Armin Kappeler, Kofi Boakye, Joao Soares	Automatic Image Captioning using Convolution Neural Networks and LSTM	2019	Dataset: MS COCO Method: architecture model using CNN as well as NLP techniques	Using CNN and LSTM models the image's caption is generated.
3.	Manish Raypurkar,	Deep learning-based	2021	Dataset: Flickr_8k	Proposed model

	Abhishek Supe, Pratik Bhumkar, Pravin Borse, Dr. Shabnam Sayyad	image caption generator		Method: CNN and LSTM model to extract features and sequence the words and finally generating captions.	is based on multi label Neural networks
4.	B.Krishnakumar,K.Kousalya, S.Gokul,R.Karthikeyan, D.Kaviyarasu	Image caption Generator using Deep Learning	2020	Method: Deep learning-based model using CNN to identify featured objects with the help of OpenCv.	proposed model could generate captions successfully in Jupyter Notebook using keras as well as tenserflow
5.	R. Subash	Automatic Image Captioning using	2019	Dataset: MS COCO	Using CNN-

Convolution Neural Networks and LSTM	Method: NLP and CNN-LSTM based model	LSTM and NLP techniques the model for image captioning is generated

3.SYSTEM ANALYSIS

3.1 Proposed System

Recognizing the objects and scenes in images, understanding their context, and then generating descriptive natural language captions in user convenient language and transform it into audio to convey the image content.

3.2 System Requirements

3.2.1 Software Requirements

- RNN-Recurrent Nueral Network- LSTM(Long Short term memory).
- Tensorflow,Keras,pillow
- CNN
- Numpy,matplotlib

3.2.2 Hardware Requirements

OS: Windows 10.

CPU: Intel processor with 64-bit support.

Disk Storage: 8GB of free disk space.

4.PRELIMINARIES

4.1 IM

AGES:

A dataset of images is required for training and testing the computer vision models. The images should represent various real-world scenarios and contexts to ensure that the system can recognize and describe a wide range of visual content. Once the dataset is collected or obtained, it needs to be appropriately preprocessed, which may involve resizing the images, normalizing pixel values, and splitting the dataset into training and testing sets.

4.2 CAPTIONS:

caption data is required for training and evaluating the image captioning model in the " Voice assistance for images " project. Caption data consists of textual descriptions or annotations that correspond to the images in the dataset.

When using pre-labeled datasets or crowdsourced data, it's essential to provide proper attribution to the original sources and comply with any licensing requirements associated with the caption data.

Remember to preprocess the caption data, such as cleaning the text, tokenizing the captions into individual words or tokens, and creating appropriate data structures for training the captioning model.

4.3 FEATURE EXTRACTION:

Feature extraction is a crucial step in image captioning, where the visual content of an image is transformed into a representative numerical feature vector. This feature vector captures the relevant visual information that will be used by the captioning model to generate captions for the images.

In the context of image captioning, feature extraction is typically performed using Convolutional Neural Networks (CNNs). CNNs are deep learning models that are well-suited for extracting meaningful features from images. Popular CNN architectures for feature extraction include VGG-16 OR Xception.

4.4 MODELS

4.4.1 CNN (CONVUTIONAL NUERAL NETWORK)

CNN mainly has three parts:

4.4.1.1 convolution

4.4.1.2 Pooling

4.4.1.3 fully connected layers

Convolution layer:

In this layer, filters are applied to extract features from images. The most important parameters are the size of the kernel and stride. This is the first step in the process of extracting valuable features from an image. A convolution layer has several filters that perform the convolution operation. Every image is considered as a matrix of pixel values.

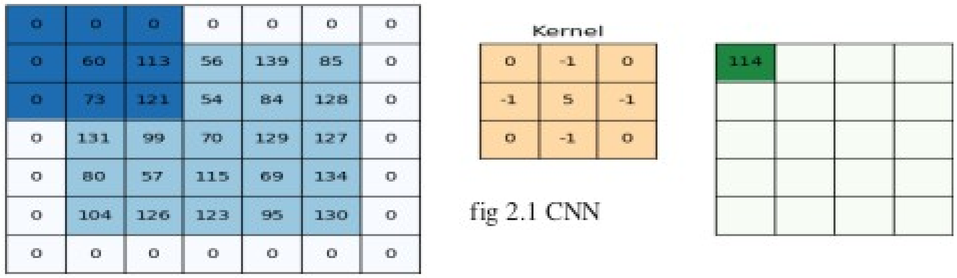
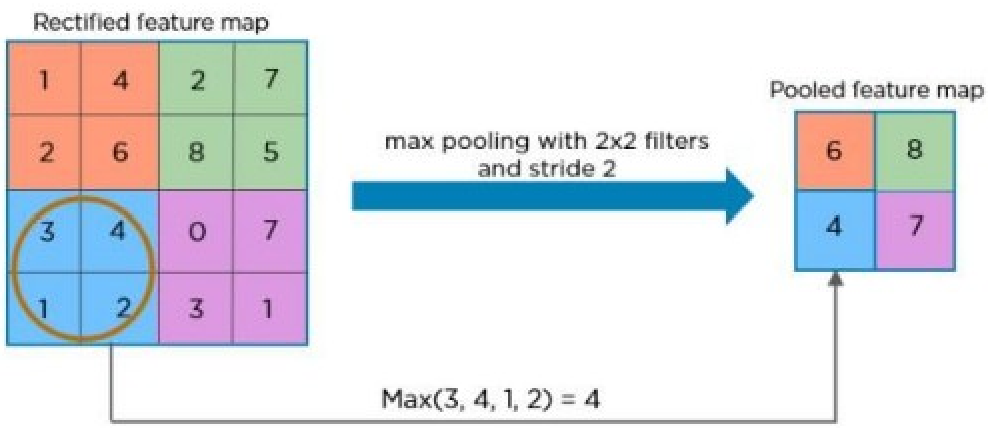


fig 2.1 CNN

Pooling layer:

Its function is to reduce the spatial size to reduce the number of parameters and computation in a network. Pooling layer summarizes the features present in a region of the feature map generated by a convolution layer.b



Fully Connected:

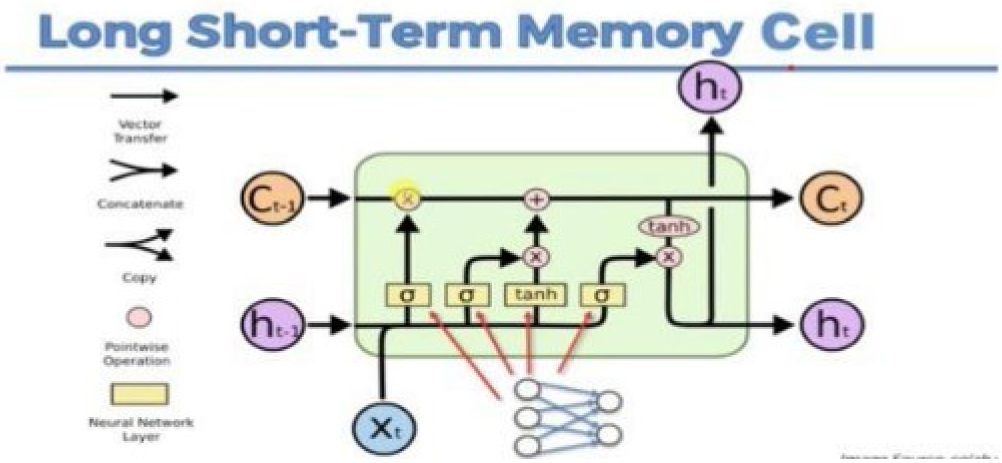
These are fully connected connections to the previous layers as in a simple neural network.

VGG16:

The VGG-16 model is a convolutional neural network (CNN) architecture that was proposed by the Visual Geometry Group (VGG) at the University of Oxford. It is characterized by its depth, consisting of 16 layers, including 13 convolutional layers and 3 fully connected layers. VGG-16 is renowned for its simplicity and effectiveness, as well as its ability to achieve strong performance on various computer vision tasks, including image classification and object recognition. The model’s architecture features a stack of convolutional layers followed by max-pooling layers, with progressively increasing depth. This design enables the model to learn intricate hierarchical representations of visual features, leading to robust and accurate predictions. Despite its simplicity compared to more recent architectures, VGG-16 remains a popular choice for many deep learning applications due to its versatility and excellent performance.

4.4.2 LSTM (Long Short-Term Memory)

It is a variety of recurrent neural networks (RNNs) that are capable of learning long-term dependencies, especially in sequence prediction problems. LSTM has feedback connections, i.e., it is capable of processing the entire sequence of data, apart from single data points such as images. This finds application in speech recognition, machine translation, etc. LSTM is a special kind of RNN, which shows outstanding performance on a large variety of problems.The central role of an LSTM model is held by a memory cell known as a ‘cell state’ that maintains its state over time. The cell state is the horizontal line that runs through the top of the below diagram. It can be visualized as a conveyor belt through which information just flows ,unchanged.



4.4.3 GTTS (Google Text-to-Speech)

The Text-to-Speech API enables developers to generate human-like speech. The API converts text into audio formats such as WAV,lMP3, or OggOpus. Google TTS simply checks the content information and matches it with its data set and essentially plays a sound yield. In the event that something isn't found in the data set it attempts to talk, for example at the point when some Indian attempts to talk familiar French (simply a model). Actually speaking, If you have seen a word reference, every single word is appointed an elocution. Likewise TTS checks its information base for the articulation and arranges the discourse yield. Furthermore, assuming no such word exists in is word reference, it will articulate straight, that is, Raam will be articulated as in the event that you would "Ram" (Hindi).

5.METHODOLOGY

5.1 METHODOLOGY

This project involves several key steps to enable visually challenged individuals to perceive their surroundings through image recognition and natural language understanding. The following is a high-level overview of the methodology:

1. Data Collection:

- Gather a dataset of images that represent various scenes, objects, and activities.
- Collect caption data for the images, either through manual annotation, pre-labeled datasets,or crowdsourcing.

2. Data Preprocessing:

- Clean and preprocess the caption data, including lowercasing, removing punctuation, and tokenizing the captions into individual words.
- Preprocess the images, such as resizing, normalizing pixel values, and applying any necessary transformations.

3. Feature Extraction:

- Utilize a pre-trained Convolutional Neural Network (CNN) model, such as Xception or VGG- 16, to extract visual features from the images.
- The output of a specific layer in the CNN is used as the feature vector that represents the visual content of the image.

4.Caption Generation Model:

- Design and implement a caption generation model, typically based on Recurrent Neural Networks (RNN) or Transformer architectures.
- The model takes the extracted image features as input and learns to generate captions by sequentially predicting words or tokens.

5. Training the Model:

- Split the dataset into training and validation sets.
- Feed the image features and corresponding preprocessed captions into the caption generation model for training.
- Optimize the model using suitable loss functions and optimization techniques, adjusting the model's parameters to minimize the loss and improve caption generation performance.

6. Evaluation:

- Evaluate the trained model using appropriate metrics, such as BLEU to measure the quality and accuracy of the generated captions.
- Fine-tune the model if necessary based on the evaluation results.

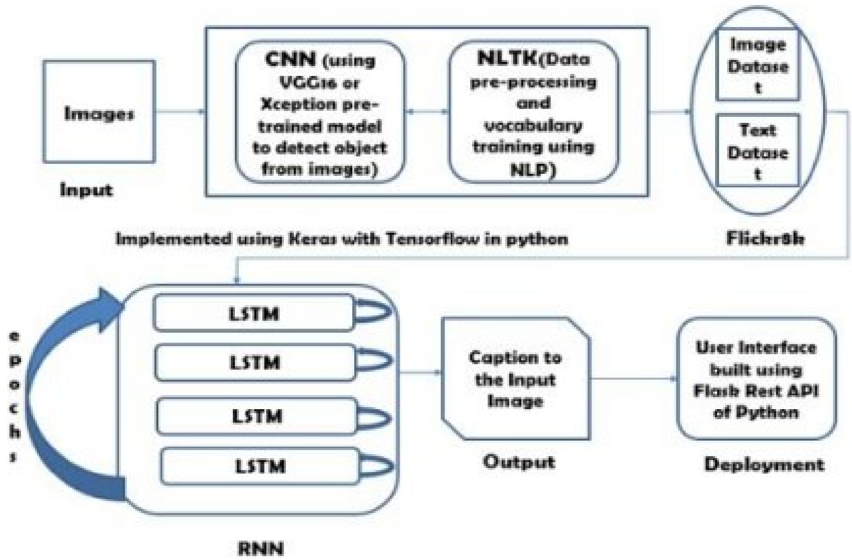
7. Caption Generation and Audio Conversion:

- Utilize the trained model to generate captions for new or unseen images.
- Convert the generated captions into audio format using text-to-speech synthesis libraries or services, such as Google Text-to-Speech (GTTS).
- The audio output provides visually challenged individuals with auditory descriptions of the image content.

8. Deployment:

- Deploy the captioning system as a REST API using a web framework like Flask.
- Provide a user-friendly interface or mobile application to allow visually challenged users to capture or upload images, receive audio descriptions, and interact with the system.

5.2 System Architecture

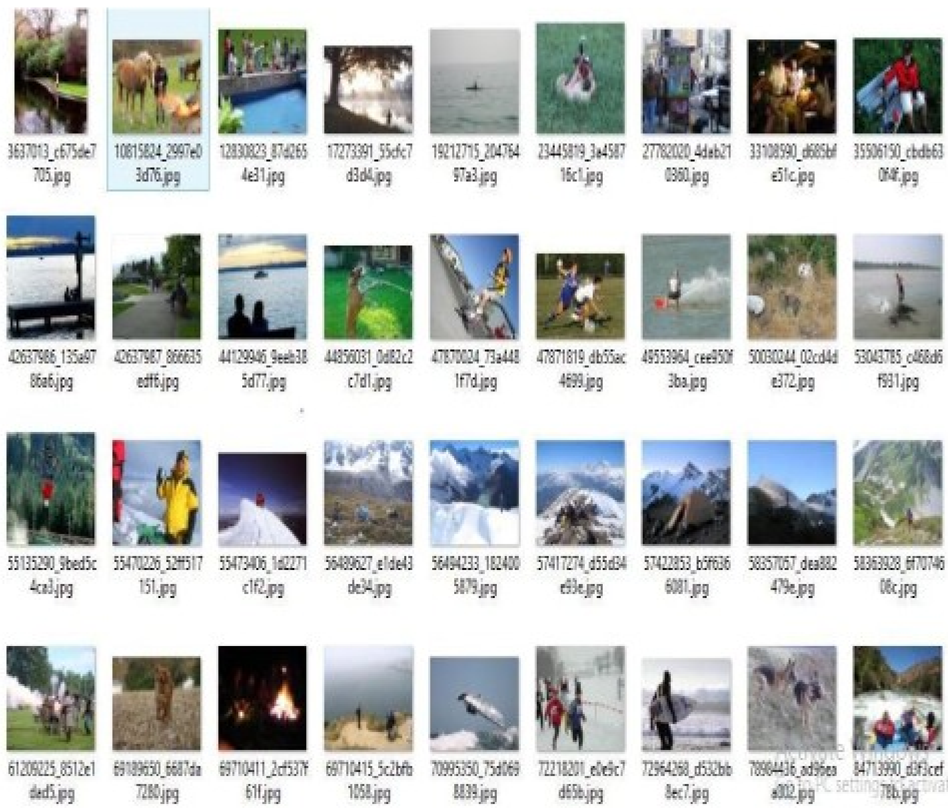


5.2 DATA SETS

- FLICKR 8K IMAGES
- FLICKR CAPTION DATA

5.2.1 FLICKR 8K IMAGES

The " Voice Bot For Assistive Vision" project can utilize the Flickr8K dataset, which is a popular and widely used dataset for image captioning tasks. The dataset consists of 8,000 images collected from the photo-sharing platform Flickr. Each image in the dataset is accompanied by five human-generated captions, providing diverse and descriptive textual annotations for the images.



5.2.2CAPTIONS

Caption data is a crucial component of the image captioning task. It consists of textual descriptions or captions that provide a detailed explanation or interpretation of the visual content present in an image. In the context of the "Voice Bot For Assistive Vision " project, caption data is required to train a model that can generate captions for images, enabling visually challenged individuals to understand the visual content through textual descriptions.

One image contains five different type of captions

8000*5 =40000 captions

5.3 Preprocessing

Preprocessing steps for image captioning:

1. Image Preprocessing:

- Image resizing: Images are resized to a fixed size to ensure consistency in the input dimensions for the model.
- Normalization: Pixel values of the images are normalized to a specific range (e.g., 0 to 1 or -1 to 1) to facilitate better model convergence.
- Feature extraction: A pre-trained convolutional neural network (CNN) model is used to extract visual features from the images. The output features represent the high-level visual information of the images.

2. Caption Preprocessing:

• Tokenization: Each caption is split into individual words or tokens. This step breaks down the sentences into smaller units that can be processed by the model.

• Lowercasing: All words in the captions are converted to lowercase to ensure consistency and avoid duplication based on capitalization.

•Removing punctuation: Punctuation marks, such as commas, periods, and quotation marks, are removed from the captions as they are not necessary for the model's understanding.

•Vocabulary creation: A vocabulary is built by collecting all unique words from the captions. This vocabulary is used to map words to numeric indices for input and output encoding.

•Start and end tokens: Special start and end tokens (e.g., "startseq" and "endseq") are added to the beginning and end of each caption. These tokens help the model learn the caption generation process.

3. Data Splitting:

•The preprocessed data is split into training, validation, and test sets. The training set is used to train the model, the validation set is used for hyperparameter tuning and model selection, and the test set is used for evaluating the final model's performance.

5.4 FEATURE EXTRACTION

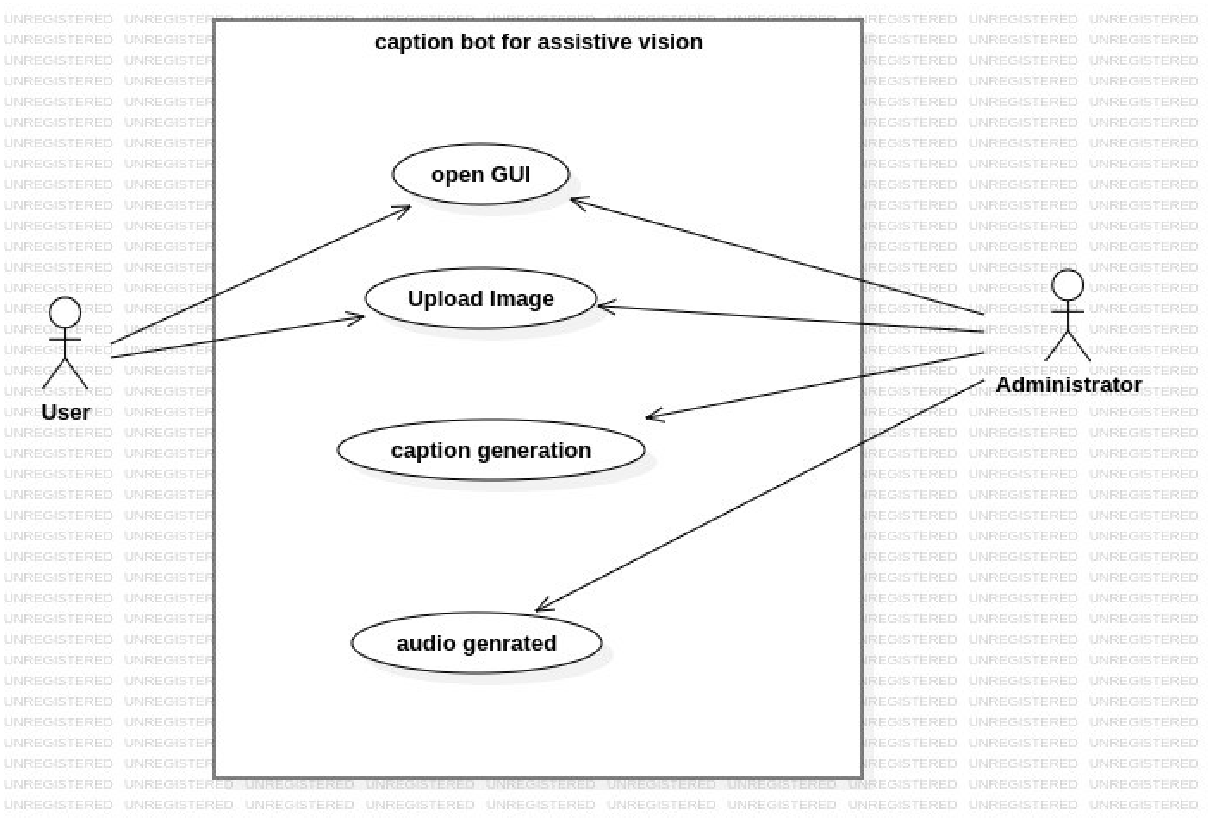
Feature extraction is a crucial step in image captioning, where the visual information present in an image is transformed into a numerical representation that can be understood by a machine learning model. In image captioning, the goal is to generate textual descriptions of images based on their visual content. Feature extraction helps to capture the relevant visual features from the images, which can then be used by the model to generate accurate and contextually relevant captions.

The output of the pre-trained CNN model is extracted at a certain layer. This layer is typically the last convolutional or pooling layer before the fully connected layers. The output of this layer represents a high-dimensional feature vector that encodes the visual information of the image.

Identify and localize the signs, and then applies a trained model for sign classification and interpretation. The module communicates with the server backend to receive image or video data, process it, and send back the detected signs along with their meanings.

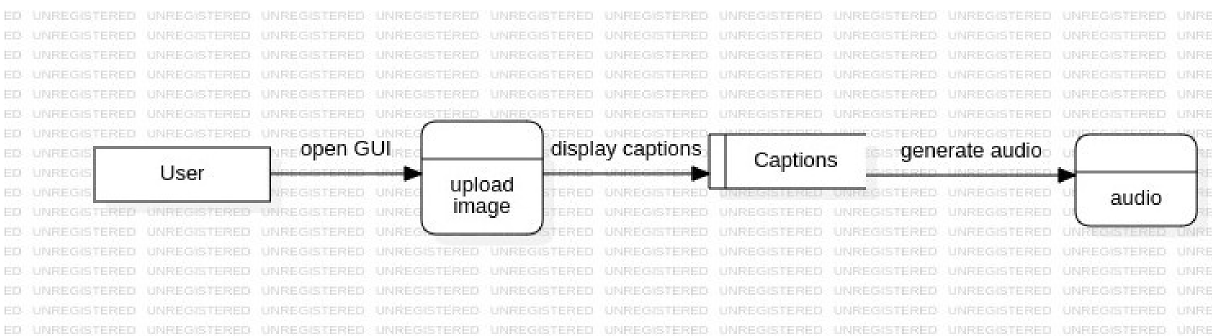
6.List Of Illustrations

6.1 Use Case Diagram



Usecase Diagram

6.2 Data Flow Diagram



Data Flow diagram for level 0

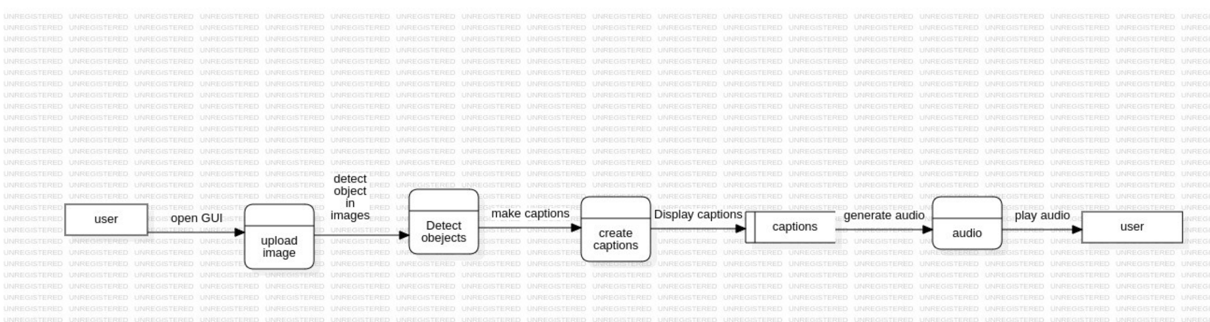


Fig.Data flow diagram for level1

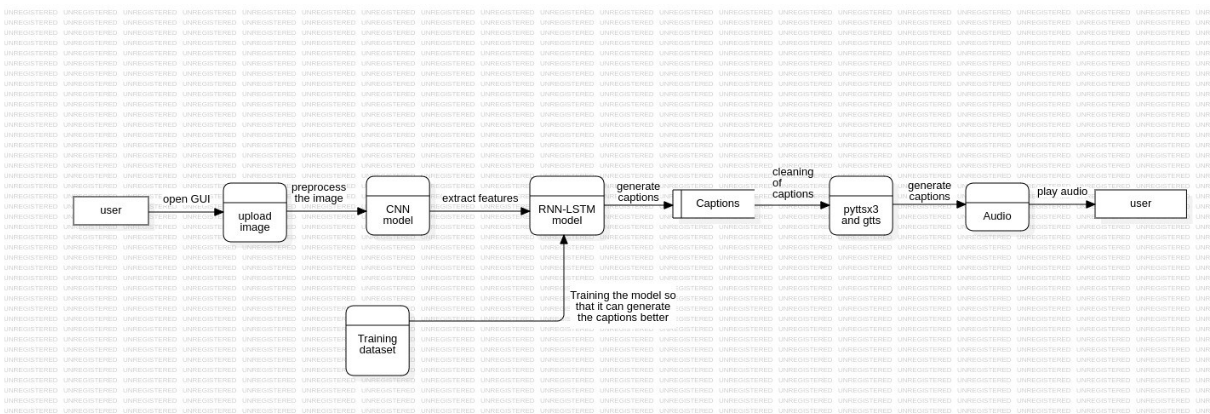


Fig.Data flow diagram for level2

6.3 State Chart Diagram

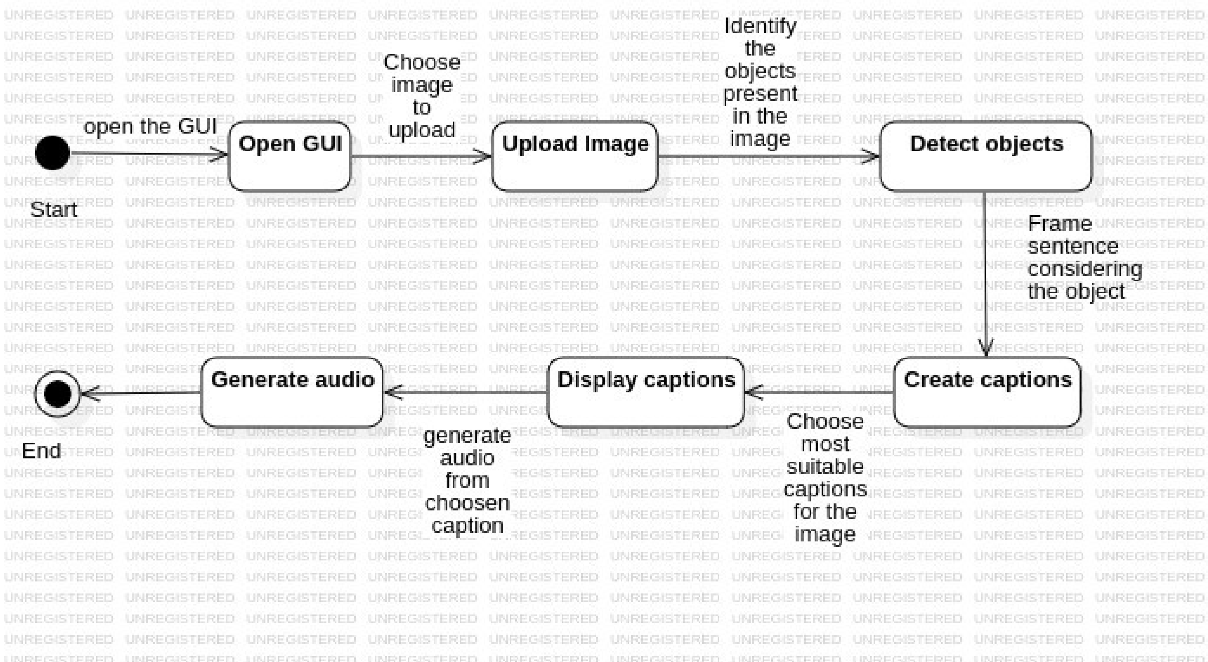


Fig.Statechart diagram

7.IMPLEMENTATION

7.1 REQUIRED LIBRARIES AND PACKAGES:

```
import os
import pickle
import numpy as np
from tqdm.notebook import tqdm
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout,
add
```

EXPLANATION:

- import string: Imports the string module, which provides a collection of string constants and helper functions.
- import numpy as np: Imports the numpy library and aliases it as np, which is commonly used for numerical computing in Python.
- from PIL import Image: Imports the Image module from the Python Imaging Library (PIL),which is used for opening, manipulating, and saving many different image file formats.
- import os: Imports the os module, which provides functions for interacting with the

operating system, such as reading or writing files and directories.

- from pickle import dump, load: Imports the dump and load functions from the pickle module, which are used for serializing and deserializing Python objects.
- import numpy as np: Re-imports the numpy library, possibly redundant as it was already imported earlier in the script.
- from keras.applications.vgg16 import vgg16, preprocess_input: Imports the Xception model architecture and a function for preprocessing input data specific to the Xception model from the Keras library. Xception is a pre-trained deep learning model commonly used for image-related tasks

•from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input: Imports the VGG16 model architecture and a function preprocess_input for preprocessing images to be compatible with the VGG16 model.

- from tensorflow.keras.preprocessing.image import load_img, img_to_array: These functions
- from tensorflow.keras.preprocessing.sequence import pad_sequences:

  `pad_sequences` is used for padding sequences to a maximum length
- from tensorflow.keras.models import Model: `Model` is imported for creating a Keras model using the functional API, which allows defining complex models with multiple

inputs and outputs.

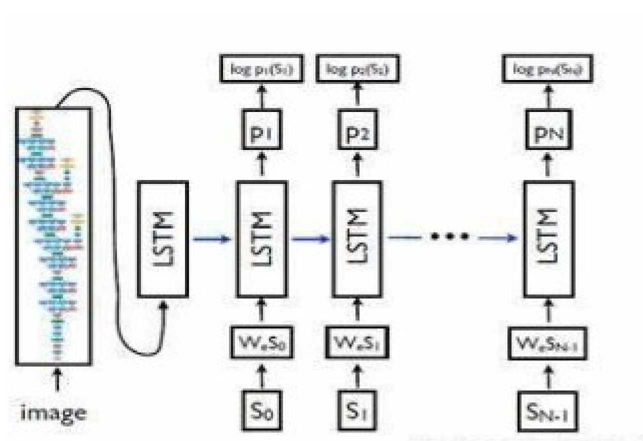
- from tensorflow.keras.utils import to_categorical, plot_model: `to_categorical` is used to convert class vectors (integers) to binary class matrices (one-hot encoded). `plot_model` allows visualizing the architecture of a Keras model as a graph.
- from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, add: These are various types of layers commonly used in neural network architectures:
  - `Input`: Defines the input layer of the neural network.
  - `Dense`: Fully connected layer.
  - `LSTM`: Long Short-Term Memory layer, commonly used for sequence prediction tasks.
  - `Embedding`: Converts integers (tokenized words) into fixed-size dense vectors.
  - `Dropout`: A regularization technique where randomly selected neurons are ignored during training.
  - `add`: Allows adding layers or merging multiple layers.

7.2 IMPLEMENTATION OF LSTM MODEL:

The implementation of an LSTM (Long Short-Term Memory) model for image captioning involves several steps.

Here's a general outline of the implementation process:

- **Load the dataset:** Load the preprocessed image features and caption data. This includes loading the image features extracted using a pre-trained CNN and the corresponding captions.
- **Prepare the caption data:** Tokenize the captions into individual words and create a vocabulary.Assign a unique index to each word in the vocabulary. Convert the caption data into numerical sequences by replacing each word with its corresponding index.
- **Split the dataset:** Split the dataset into training and validation sets. This is important for evaluating the performance of the model and preventing overfitting.
- **Define the LSTM model architecture:** Create the LSTM model using the Keras or TensorFlow library. The model consists of an input layer, an embedding layer to transform the word indices into dense vectors, one or more LSTM layers, and a dense output layer.



CLEANING THE CAPTION DATA:

```
def clean(mapping):
    for key,captions in mapping.items():
        for i in range(len(captions)):
        # take one caption at a time
            caption = captions[i]
        # preprocessing steps
        # convert to lowercase
            caption = caption.lower()
        # delete digits, special chars, etc.,
            caption = caption.replace('[^A-Za-z]', '')
        # delete additional spaces
            caption = caption.replace('\s+', ' ')
        # add start and end tags to the caption
            caption = 'startseq ' + " ".join([word for word in caption.split() if len(word)>1]) + '
             endseq'
            captions[i] = caption
```

Explanation:

Iterating through `mapping` Dictionary:

- `for key, captions in mapping.items():`: Iterates through each key-value pair in the `mapping` dictionary, where `key` is typically an image ID, and `captions` is a list of captions associated with that image.

- **Iterating through Captions List**:

`for i in range(len(captions)):`: Iterates through each caption in the `captions` list associated with the current `key`.

- **Preprocessing Steps**:
- **Lowercasing**:
  - `caption = caption.lower()`: Converts the entire caption to lowercase to ensure uniformity and simplify tokenization and comparison.
- **Removing Digits, Special Characters, etc.**:

  `caption = caption.replace('[^A-Za-z]', '')`: This line attempts to remove any characters that are not letters (both uppercase and lowercase) from the caption. However, the usage of `replace` here with a regular expression pattern (`[^A-Za-z]`) won't work as intended. To correctly remove non-alphabetic characters, you would typically use regular expression substitution with `re.sub.`

Feature Extraction:

extract features from image

features = {}

directory = os.path.join('images')

```
for img_name in tqdm(os.listdir(directory)):
    # load the image from file
    img_path = directory + '/' + img_name
    path = directory + '/' + img_name
    image = load_img(img_path, target_size=(224, 224))
    # convert image pixels to numpy array
    image = img_to_array(image)
    # reshape data for model
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    # preprocess image for vgg
    image = preprocess_input(image)
    # extract features
    feature = model.predict(image, verbose=0)
    # get image ID
    image_id = img_name.split('.')[0]
    # store feature
    features[image_id] = feature
```

Explanation:

Initialization:

- `features = {}`: This dictionary will store the extracted features of each image, where the keys are image IDs (derived from the filenames) and the values are the extracted features.

- `directory = os.path.join('images')`: Defines the directory containing the images. Assuming there is a folder named `'images'` where the images are stored.

2. **Iterating through Images**:

- `for img_name in tqdm(os.listdir(directory)):`: Iterates through each file name in the directory `'images'`. `tqdm` is used here to show a progress bar in the notebook environment, providing visual feedback on the loop's progress.

3. **Loading and Preprocessing Images**:

- `img_path = directory + '/' + img_name`: Constructs the full path to the image file.
- `image = load_img(img_path, target_size=(224, 224))`: Loads the image from the file and resizes it to a target size of 224x224 pixels. `load_img` is from `keras.preprocessing.image`.
- `image = img_to_array(image)`: Converts the loaded image to a NumPy array representation.
- `image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))`: Reshapes the image data to conform to the input shape required by the VGG16 model, which is `(batch_size, height, width, channels)`. Here, `batch_size` is set to 1, indicating a single image input.

4. **Preprocessing for VGG16 Model**:

- `image = preprocess_input(image)`: Preprocesses the image data according to the requirements of the VGG16 model. This includes mean subtraction and scaling to ensure compatibility with the model's pre-trained weights.

5. **Extracting Features**:

   - `feature = model.predict(image, verbose=0)`: Uses the VGG16 model (which was previously modified to output the second-to-last layer's activations) to predict features for the preprocessed image. The resulting `feature` variable contains the extracted feature vector for the current image.

6. **Storing Features**:

   - `image_id = img_name.split('.')[0]`: Extracts the image ID from the filename by removing the file extension (assuming filenames are in the format `image_id.jpg`).
   - `features[image_id] = feature`: Stores the extracted feature vector in the `features` dictionary, using the `image_id` as the key.

Create Mapping Of Images:

```
# create mapping of image to captions
mapping = {}
# process lines
for line in tqdm(captions_doc.split('\n')):
    # split the line by comma(,)
    tokens = line.split(',')
```

```
if len(line) < 2:
    continue
image_id, caption = tokens[0], tokens[1:]
# remove extension from image ID
image_id = image_id.split('.')[0]
 # convert caption list to string
caption = " ".join(caption)
# create list if needed
if image_id not in mapping:
    mapping[image_id] = []
# store the caption
mapping[image_id].append(caption)
```

Explaination:

Initialization:

- `mapping = {}`: Initializes an empty dictionary `mapping` that will store image IDs as keys and lists of captions as values.

Processing Lines from CSV File:

- `for line in tqdm(captions_doc.split('\n')):`: Iterates through each line of text in the `captions_doc` variable. `tqdm` is used to show a progress bar in the notebook environment.

Splitting and Parsing Each Line:

- `tokens = line.split(',')`: Splits each line by commas (`,`). This assumes that the CSV file separates fields by commas.
- `if len(line) < 2: continue`: Skips the line if its length is less than 2 characters, typically used to skip empty lines or lines that might not contain valid data.

Extracting Image ID and Captions:

- `image_id, caption = tokens[0], tokens[1:]`: Assigns the first token (`tokens[0]`) to `image_id`, which presumably contains the filename of the image. The rest (`tokens[1:]`) are assumed to be captions associated with that image.
- `image_id = image_id.split('.')[0]`: Removes the file extension from `image_id`. This assumes filenames are in the format `image_id.jpg` or similar.

Formatting Captions:

- `caption = " ".join(caption)`: Joins the list of caption tokens into a single string with spaces in between. This assumes captions are split into multiple parts by commas in the CSV file.

Storing in the Mapping Dictionary:

- `if image_id not in mapping: mapping[image_id] = []`: Checks if `image_id` already exists as a key in `mapping`. If not, initializes an empty list as its value.
- `mapping[image_id].append(caption)`: Appends the formatted caption to the list associated with `image_id` in the `mapping` dictionary.

create data generator to get data in batch

create data generator to get data in batch (avoids session crash)

```
def data_generator(data_keys, mapping, features, tokenizer, max_length, vocab_size,
 batch_size):
    # loop over images
    X1, X2, y = list(), list(), list()
    n = 0
    while 1:
        for key in data_keys:
            n += 1
            captions = mapping[key]
            # process each caption
            for caption in captions:
                # encode the sequence
                seq = tokenizer.texts_to_sequences([caption])[0]
                # split the sequence into X, y pairs
                for i in range(1, len(seq)):
                    # split into input and output pairs
                    in_seq, out_seq = seq[:i], seq[i]
                    # pad input sequence
                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                    # encode output sequence
                    out_seq = to_categorical([out_seq],     num_classes=vocab_size)[0]
```

```
            # store the sequences
            X1.append(features[key][0])
            X2.append(in_seq)
            y.append(out_seq)
        if n == batch_size:
            X1, X2, y = np.array(X1), np.array(X2), np.array(y)
            yield {"image": X1, "text": X2}, y
            X1, X2, y = list(), list(), list()
            n = 0
```

Explaination:

`data_generator` function is to provide an efficient way to generate batches of data for training a neural network. This is particularly useful when dealing with large datasets that cannot fit into memory all at once. By using a generator, batches of data are generated on-the-fly during training, allowing for continuous model training without the risk of running out of memory.

7.3 Train Model:

```
# train the model
epochs = 20
batch_size = 32
steps = len(train) // batch_size
```

```
for i in range(epochs):
    # create data generator
    generator = data_generator(train, mapping, features, tokenizer, max_length, vocab_size,
 batch_size)
    # fit for one epoch
    model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1)
```

Explanation:

Training Parameters:

- `epochs`: Number of times the model will iterate over the entire training dataset.
- `batch_size`: Number of samples per gradient update during training.

Training Loop:

- **Epoch Loop (`for i in range(epochs):`)**: Iterates over the specified number of epochs (`epochs`).
- **Data Generator Creation**:
  - Initializes a data generator using the `data_generator` function previously defined. This generator will yield batches of training data continuously.
- **Model Training**:
  - Fits the model for one epoch (`epochs=1`) using the data generator (`generator`).
  - `steps_per_epoch=steps` specifies how many batches to draw

from the generator for each epoch.

- `verbose=1` indicates that training progress will be displayed during training.

```
28/28 ━━━━━━━━━━━━━━━━━━━━ 56s 2s/step - loss: 6.6431
28/28 ━━━━━━━━━━━━━━━━━━━━ 50s 2s/step - loss: 5.4005
28/28 ━━━━━━━━━━━━━━━━━━━━ 53s 2s/step - loss: 4.9906
28/28 ━━━━━━━━━━━━━━━━━━━━ 52s 2s/step - loss: 4.5970
28/28 ━━━━━━━━━━━━━━━━━━━━ 53s 2s/step - loss: 4.1905
28/28 ━━━━━━━━━━━━━━━━━━━━ 52s 2s/step - loss: 3.8346
28/28 ━━━━━━━━━━━━━━━━━━━━ 55s 2s/step - loss: 3.5261
28/28 ━━━━━━━━━━━━━━━━━━━━ 50s 2s/step - loss: 3.2625
28/28 ━━━━━━━━━━━━━━━━━━━━ 51s 2s/step - loss: 3.0188
28/28 ━━━━━━━━━━━━━━━━━━━━ 50s 2s/step - loss: 2.8208
28/28 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - loss: 2.6440
28/28 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - loss: 2.5444
28/28 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - loss: 2.4513
28/28 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - loss: 2.3610
28/28 ━━━━━━━━━━━━━━━━━━━━ 48s 2s/step - loss: 2.2726
28/28 ━━━━━━━━━━━━━━━━━━━━ 44s 2s/step - loss: 2.2011
28/28 ━━━━━━━━━━━━━━━━━━━━ 44s 2s/step - loss: 2.1154
28/28 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - loss: 2.0487
28/28 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - loss: 1.9934
28/28 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - loss: 1.9502
```

- `28/28`: Indicates the current batch number out of the total batches per epoch. For instance,`28/28` means the 28th batch out of 28 batches in that epoch.
- ━━━━━━━━━━━━━━━━━━━━: Visual representation of progress bar.
- `50s`: Time taken for processing that batch.
- `2s/step`: Average time taken per batch (`2s` here).
- `- loss: X.XXXX`: Loss value (`X.XXXX`) computed after processing that batch.

Generate Caption For An Image:

```
def predict_caption(model, image, tokenizer, max_length):

    # add start tag for generation process

    in_text = 'startseq'

    # iterate over the max length of sequence

    for i in range(max_length):

        # encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]

        # pad the sequence

        sequence = pad_sequences([sequence], maxlen=max_length)

        # predict next word

         yhat = model.predict([image, sequence], verbose=0)

        # get index with high probability
```

```
        yhat = np.argmax(yhat)
        # convert index to word
        word = idx_to_word(yhat, tokenizer)
        # stop if word not found
        if word is None:
            break
        # append word as input for generating next word
        in_text += " " + word
        # stop if we reach end tag
        if word == 'endseq':
            break
    return in_text
```

Explanation:

Parameters:

- `model`: The trained Keras model used for generating captions, which takes both image features and partial captions as inputs.
- `image`: The feature representation of the image, typically extracted using a pre-trained CNN like VGG16 and preprocessed.
- `tokenizer`: The tokenizer object that maps words to indices and vice versa.
- `max_length`: The maximum length of the caption that the function will generate.

- **Function Logic**:
  - **Initialize Input Text**: Starts with `'startseq'` as the initial input to kickstart

the caption generation process.

- **Caption Generation Loop**:
  - Iterates over `max_length` to generate each word in the caption sequence.
- **Text Encoding and Padding**:
  - `sequence = tokenizer.texts_to_sequences([in_text])[0]`: Converts the current `in_text` (caption sequence so far) into a sequence of integers using the tokenizer.
  - `sequence = pad_sequences([sequence], maxlen=max_length)`: Pads the sequence to ensure it has the same length (`max_length`), required by the model.
- **Prediction**:
  - `yhat = model.predict([image, sequence], verbose=0)`: Predicts the next word in the sequence given the image features (`image`) and the current partial sequence (`sequence`).
  - `yhat = np.argmax(yhat)`: Determines the index of the word with the highest predicted probability.
- **Word Conversion and Update**:
  - `word = idx_to_word(yhat, tokenizer)`: Converts the predicted index (`yhat`) into a word using the `idx_to_word` function.
  - `in_text += " " + word`: Appends the predicted word to the current `in_text` for generating the next word in the sequence.
- **Stopping Conditions**:
  - Stops generating words if no valid word is found (`word is None`).

- Stops if the end sequence tag (`'endseq'`) is predicted, indicating the end of the caption.

- **Output**:
    - Returns the generated caption (`in_text`), starting with `'startseq'` and potentially ending with `'endseq'` or when no more valid words can be predicted.

Text To Speech Conversion:

```
import pyttsx3

# Inilialize the TTS engine

engine = pyttsx3.init()

# Set properties (optional)

engine.setProperty('rate', 110)  # Speed of speech

engine.setProperty('volume', 1.0)  # Volume level (0.0 to 1.0)

# Text to be converted to speech

text = x

# Convert text to speech

engine.say(text)# Wait for the speech to finish

engine.runAndWait()
```

8.1 SUMMARY

The project aims to develop an assistive vision system for visually challenged individuals. The system utilizes image recognition and natural language processing techniques to provide visually impaired people with a description of their surroundings.

8.2 FUTURE WORK

As a result, a system was created to assist the blind and visually impaired in achieving their goals and contributing more to society as a whole. To create a mapping between images and sentences, a CNN network is trained first, followed by an LSTM network. The quantitative validation of our model yielded promising results. The model's precision and efficiency will be improved in the future. A server-client model for the blind to use in any environment can also be added. As the size of the dataset grows larger, overfitting becomes less of a problem. Furthermore, we believe that our research could pave the way for a more general form of AI.

8.3 REFERENCES

•Seung-Ho Han and Ho-Jin Choi,” Domain-Specific Image Caption Generator with Semantic

Ontology" IEEE 2020

•Kurt Shuster, Samuel Humeau, Hexiang Hu, Antoine Bordes, Jason Weston, "Engaging ImageCaptioning via Personality" IEEE 2019

• Image Caption using CNN LSTM by Ali ashraf Mohamed [1]

• Image Caption Generator using CNN-LSTM by Preksha Khant, Vishal Deshmukh, Aishwarya Kude, Prachi Kiraula [2]