

VOICE BASED PERSONAL ASSISTANT
FOR BLIND PEOPLE USING NLP AND
COMPUTER VISION

A Report

*Submitted in partial fulfilment of the
Requirements for the completion of*

THEME BASED PROJECT

**BACHELOR OF ENGINEERING
IN
INFORMATION TECHNOLOGY**

By

**BHUVANA SUNKARA 1602-21-737-090
MANOJ ARIPAKA 1602-21-737-094
BHANU TEJA VARA VENKATA 1602-21-737-121**

**Under the guidance of
Ms. B Leelavathy
Assistant Professor**



Department of Information Technology

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE.

(Affiliated to Osmania University and Approved by AICTE)

Ibrahim Bagh, Hyderabad-31

2024

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University and Approved by AICTE)

Ibrahim Bagh, Hyderabad-31

Department of Information Technology



DECLARATION BY CANDIDATES

We **BHUVANA SUNKARA, MANOJ ARIPAKA, BHANU TEJA VARA VENKATA**, bearing hall ticket number, **1602-21-737-090, 1602-21-737-094, 1602-21-737-121**, hereby declare that the project report entitled **"Voice based personal assistant for blind people using NLP and Computer Vision"** under the guidance of Ms. B Leelavathy, Assistant Professor, Department of Information Technology, Vasavi College of Engineering, Hyderabad, is submitted in partial fulfillment of the requirement for the completion of Theme-based project, VI semester, Bachelor of Engineering in Information Technology.

This is a record of bonafide work carried out by us and the results embodied in this project report have not been submitted to any other institutes.

BHUVANA SUNKARA	1602-21-737-090
MANOJ ARIPAKA	1602-21-737-094
BHANU TEJA VARA VENKATA	1602-21-737-121

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University and Approved by AICTE)

Ibrahim Bagh, Hyderabad-31

Department of Information Technology



BONAFIDE CERTIFICATE

This is to certify that the project entitled “**VOICE BASED PERSONAL ASSISTANT FOR BLIND PEOPLE USING NLP AND COMPUTER VISION**” being submitted by **BHUVANA SUNKARA, MANOJ ARIPAKA, BHANU TEJA VARA VENKATA** bearing **1602-21-737-090, 1602-21-737-094, 1602-21-737-121**, in partial fulfillment of the requirements for the completion of Theme-based project of Bachelor of Engineering in Information Technology is a record of bonafide work carried out by them under my guidance.

Ms. B Leelavathy
Assistant Professor

External Examiner

Dr. K. Ram Mohan Rao
Professor & HOD IT

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of the project would not have been possible without the kind support and help of many individuals. We would like to extend our sincere thanks to all of them.

It is with immense pleasure that we would like to take the opportunity to express our humble gratitude to Ms. B Leelavathy, Assistant Professor **Information Technology** under whom we executed this project. Her constant guidance and willingness to share their vast knowledge made us understand this project and its manifestations in great depth and helped us to complete the assigned tasks.

We are very much thankful to **Dr. K. Ram Mohan Rao, Professor and HOD, Information Technology**, for his kind support and for providing the necessary facilities to carry out the work.

We wish to convey our special thanks to Dr. S. V. Ramana, **Principal of Vasavi College of Engineering**, for giving the required information in doing my project work. Not to forget, we thank all other faculty and non-teaching staff, and my friends who had directly or indirectly helped and supported me in completing my project in time.

We also express our sincere thanks to the Management for providing excellent facilities. Finally, we wish to convey our gratitude to our family who provided all the facilities that we need.

ABSTRACT

This project introduces a cutting-edge Visual Question Answering (VQA) system to enhance accessibility for blind individuals. Utilizing the RESNET 152 architecture for intricate visual feature extraction and a stacked attention model for precise answer generation, our system operates within an encoder-decoder framework. The CNN encoder captures detailed visual information, while the attention-based decoder integrates visual features with textual questions to produce relevant and accurate answers. Extensive evaluations demonstrate the system's efficacy, user-friendliness, and ability to provide accurate answers across diverse visual questions, empowering blind users and significantly enhancing their independence. This project highlights the transformative potential of combining CNNs and attention mechanisms in VQA systems, advancing assistive technology for the visually impaired.

LIST OF FIGURES

Figure 4.1 Architecture Diagram	15
Figure 4.2 Use-Case Diagram	16
Figure 4.3 Output 4. a.1	26
Figure 4.4 Output 4. a.2	26
Figure 4.5 Output 4. a.3	26
Figure 4.6 Output 4. a.4	27
Figure 4.7 Output 4. a.5	27
Figure 4.8 Output 4. b.1	28
Figure 4.9 Output 4. b.2	28
Figure 4.10 Output 4. b.3	29
Figure 4.11 Output 4. b.4	29
Figure 5.1 Methodology	30

LIST OF TABLES

Table 1 Screenshots from Jupyter	26
Table 2 Screenshots from APP	28
Table 3 Dataset Split	31
Table 4 Comparative study of various pretrained CNN models	36

TABLE OF CONTENTS

1. INTRODUCTION	9
1.1 Overview	9
1.2 Problem Statement	10
1.3 Motivation of theme & title	11
2. LITERATURE SURVEY	12
3. EXISTING SYSTEM	13
4. PROPOSED SOLUTION	14
4.1 System Design	15
4.1.1 Architecture Diagram	15
4.1.2 Use-Case Diagram	16
4.1.2.1 Use-case descriptions	16
4.2 Functional Modules	18
4.2.1 Screenshots & Pseudocode	18
5. EXPERIMENTAL SETUP & IMPLEMENTATION	30
5.1 System Specifications	30
5.1.1 Hardware Requirements	30
5.1.2 Software Requirements	30
5.2 Datasets	31
5.3 Methodology	32
6. RESULTS	36
7. CONCLUSION	38
8. FUTURE SCOPE	39
9. REFERENCES	40

1. INTRODUCTION

1.1 OVERVIEW

This project introduces a Visual Question Answering (VQA) system for blind individuals, using the RESNET 152 CNN architecture to extract detailed visual features from images. A stacked attention model integrates these features with user-posed questions, ensuring precise and relevant answers. Operating within an encoder-decoder framework, the system demonstrates high performance and user-friendliness, transforming visual data into accessible information and enhancing independence for the visually impaired.

1.2 PROBLEM STATEMENT

Blind individuals face significant challenges in accessing and understanding visual information, which limits their independence and inclusivity in various aspects of life. Current assistive technologies often fall short in providing accurate and contextually relevant interpretations of visual content. There is a pressing need for an advanced solution that can bridge this gap, enabling blind users to interact with and comprehend visual data effectively. This project aims to address this need by developing a state-of-the-art Visual Question Answering (VQA) system that leverages the capabilities of Convolutional Neural Networks (CNNs) and attention mechanisms to provide precise and relevant answers to visual questions, thus empowering the blind community and enhancing their quality of life.

1.3 MOTIVATION OF THEME AND TITLE

The motivation for developing a voice-based personal assistant for blind people using Natural Language Processing (NLP) and Computer Vision (CV) lies in addressing the significant challenges faced by the visually impaired in accessing and interpreting visual information. Blind individuals encounter numerous barriers in performing everyday tasks that require visual input, such as reading text, recognizing objects, navigating environments, and understanding visual cues. These challenges can significantly impact their independence, safety, and overall quality of life.

By combining the strengths of NLP and CV, this project aims to create a sophisticated personal assistant that can bridge these gaps. The envisioned system would leverage NLP to understand, and process spoken queries from the user, enabling a natural and intuitive mode of interaction. Concurrently, CV techniques would be employed to analyze and interpret visual content, such as images captured by the user. This dual approach ensures that the assistant can comprehend both the linguistic context and the visual environment, providing precise and relevant answers.

For instance, a blind user might take a photo of a grocery shelf and ask, "Which of these items is sugar?" The personal assistant would use CV to analyze the image, identify the products, and apply NLP to understand the query and deliver an accurate response, such as, "The second item from the left is sugar." This level of detailed and contextually aware assistance can greatly enhance the autonomy of blind users, allowing them to perform tasks independently that previously required sighted assistance.

Additionally, this integration of NLP and CV could support more complex interactions and provide comprehensive assistance across various scenarios. Whether it's reading labels, recognizing faces, navigating unfamiliar spaces, or receiving descriptions of their surroundings, blind users would benefit from a versatile tool capable of addressing a wide range of needs.

Ultimately, the development of a voice-based personal assistant powered by NLP and CV aims to empower blind individuals by providing them with a reliable, efficient, and privacy-respecting means of accessing visual information. This would not only facilitate their daily activities but also contribute to their overall well-being and social inclusion, enhancing their quality of life significantly.

2. LITERATURE SURVEY

In 2020, researcher Volha Leusha[1] conducted a study aimed at assisting the visually impaired population through the image captioning task for the VizWiz dataset . This dataset, consisting of images taken by blind individuals in real-world situations, has been largely overlooked compared to other commonly used visual-linguistic datasets and thus requires further investigation. Leusha's project constructed several models based on CNN-LSTM architecture [1], and Transformer, demonstrating solid performance. The best results were achieved by the CNN-LSTM model with attention combined with beam-search inference. The study also explored the dataset's limitations, concluding that algorithms pre-trained on artificially created datasets like MSCOCO perform poorly when applied to VizWiz. This project underscores the need for targeted datasets and tailored algorithms to better serve the needs of visually impaired users.

In their paper, Zichao Yang et al. [2] presented stacked attention networks (SANs) designed to answer natural language questions from images. SANs use the semantic representation of a question as a query to identify relevant regions in an image that are related to the answer. The authors argued that image question answering (QA) often requires multiple steps of reasoning. To address this, they developed a multi-layer SAN, which queries an image multiple times to progressively infer the answer. Experiments conducted on four image QA datasets demonstrated that the proposed SANs significantly outperformed previous state-of-the-art approaches. The visualization of the attention layers illustrated the progressive localization of relevant visual clues by the SAN, leading to the answer layer-by-layer.

In 2014, Oriol Vinyals, Alexander Toshev[3], Samy Bengio, and Dumitru Erhan introduced "Show and Tell: A Neural Image Caption Generator," a fundamental work in artificial intelligence that bridges computer vision and natural language processing. Their paper presented a generative model based on a deep recurrent architecture that integrates advances in computer vision and machine translation to generate natural sentences describing images. The model is trained to maximize the likelihood of the target description sentence given the training image. Experiments across several datasets demonstrated the model's high accuracy and language fluency, learned solely from image descriptions. Notably, their approach achieved significant improvements in BLEU-1 scores: from 25 to 59 on the Pascal dataset (human performance is around 69), from 56 to 66 on Flickr30k, and from 19 to 28 on SBU. On the newly released COCO dataset, their model achieved a BLEU-4 score of 27.7, setting a new state-of-the-art at the time

3. EXISTING SYSTEM

Seeing AI: Developed by Microsoft, Seeing AI leverages computer vision to describe people, text, and objects. It combines image recognition and natural language processing to assist blind individuals in understanding their surroundings.

Be My Eyes: This platform connects visually impaired users with sighted volunteers through video calls to help with visual tasks. Although not purely VQA, it emphasizes the importance of real-time visual assistance.

VizWiz: VizWiz is an app that allows blind users to take a picture and ask questions about the image. It uses a combination of automated and human-in-the-loop systems to provide answers, ensuring accuracy and reliability. VizWiz integrates crowdsourcing to answer visual questions that automated systems might struggle with, offering a comprehensive solution for diverse queries.

OrCam MyEye: A wearable device that uses a miniature camera to recognize faces, text, and products, providing real-time audio feedback. It integrates advanced image recognition technology to offer detailed descriptions.

Aipoly Vision: An app that uses artificial intelligence to identify objects and colors in real-time. It is designed to assist visually impaired users by providing immediate visual information through audio descriptions.

Lookout by Google: An app designed to help visually impaired people identify objects and text using their smartphone camera. It provides spoken feedback and can read text, recognize currency, and identify products.

4. PROPOSED SOLUTION

The proposed solution for a Visual Question Answering (VQA) system combines advanced computer vision and natural language processing (NLP) techniques with text-to-speech synthesis to enhance accessibility for blind individuals.

In addition to extracting visual features using a pretrained ResNet-152 model, fine-tuned on the ImageNet dataset, the system processes questions through tokenization, embedding, and encoding with an LSTM network to capture their semantic meaning. An attention mechanism is then employed to integrate the visual features and questions, allowing the model to focus on relevant parts of the image.

Once the model has attended to the relevant features and encoded the questions, the next step involves concatenating these representations to form a comprehensive understanding of the visual and textual inputs. This integrated representation enables the system to generate accurate responses to user queries.

Finally, to ensure accessibility for blind individuals, the system leverages text-to-speech synthesis to convert the textual responses into spoken form. This allows blind users to receive immediate auditory feedback, enhancing their understanding and interaction with the environment.

By combining advanced computer vision techniques with NLP and text-to-speech synthesis, this VQA system empowers blind individuals to interact with visual scenes effectively, providing them with accessible information and enhancing their independence.

4.1 SYSTEM DESIGN

The app is designed with a user-centric approach, featuring an intuitive interface tailored for accessibility. Users input visual questions via voice commands, which are processed by the system. The system leverages advanced algorithms to analyze the questions and provide accurate responses in a timely manner. It ensures inclusivity by offering features such as customizable preferences and seamless integration with assistive technologies. This user-friendly design aims to empower blind individuals to interact effectively with visual content, enhancing their independence and inclusivity in daily activities.

4.1.1 ARCHITECTURE DIAGRAM

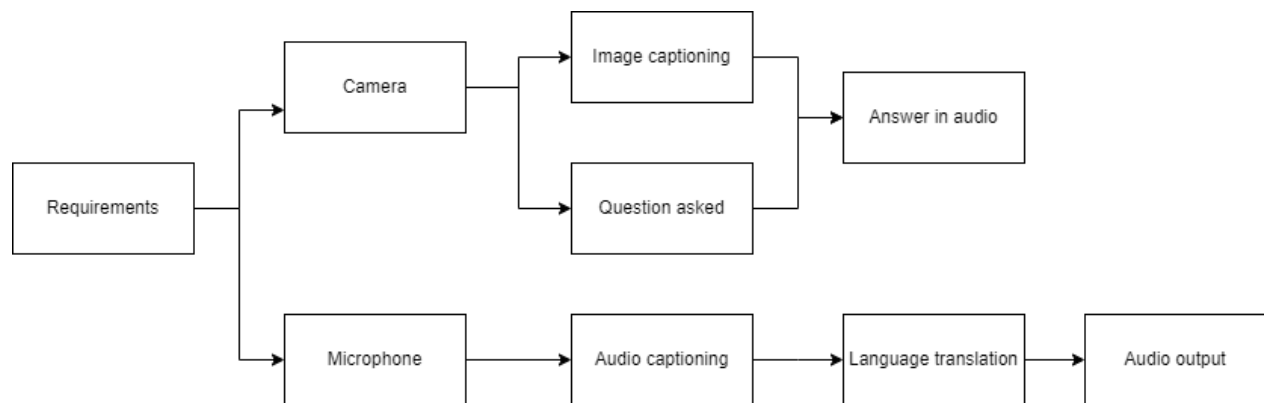


Figure 4.1 ARCHITECTURE DIAGRAM

4.1.2 USE CASE DIAGRAM

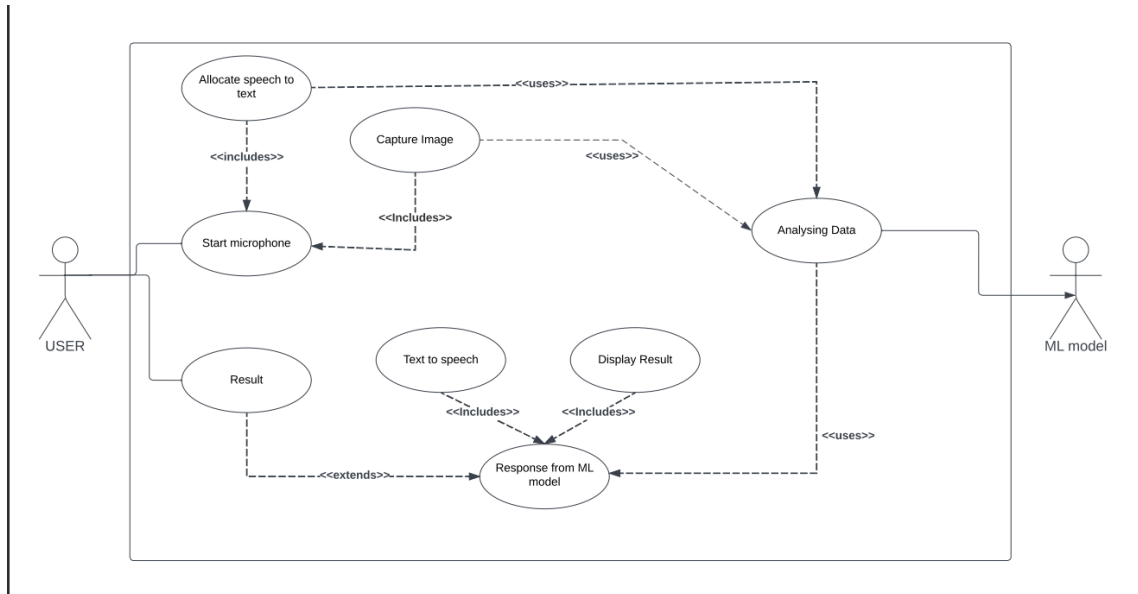


Figure 4.2 USE CASE DIAGRAM

4.1.2.1 USE-CASE DESCRIPTION

The use case diagram for a mobile application allows users to ask questions via voice, captures an image, processes both inputs using a backend machine learning model, and returns the answer in speech form. The primary actors in this system are the User and the Backend Server.

1. Actors

- User:** The person using the application. Initiates voice input and receives the answer.
- System:** The mobile application that facilitates voice recording, image capture, and interaction with the backend server.
- Backend Server:** A Flask server equipped with a machine learning model to process the question and image and generate an answer.

2. Use Cases

- Start Microphone Use Case:

- i. The user initiates the process.
 - ii. The system allocates speech-to-text resources.
 - iii. If successful, the system captures an image; if not, it ends the process.
- Capture Image Use Case:
 - i. Upon successful initiation and resource allocation, the system captures an image.
 - ii. The captured image is then used for data analysis.
 - Analyzing Data Use Case:
 - i. The captured image is processed by an ML model to analyze the data.
 - ii. Based on the analysis result, either text-to-speech is initiated or a response from the ML model is triggered.
 - Text to Speech Use Case:
 - i. If triggered by the analyzing data use case, text-to-speech converts the given text into audible speech.
 - ii. The result of this conversion leads to displaying results to the user.
 - Display Result Use Case:
 - i. Results from either text-to-speech or direct response from the ML model are displayed to the user.
 - Response from ML Model Use Case:
 - i. In parallel with displaying results, there's also a feedback loop where responses from the ML model can extend back into further processing or end based on conditional checks.
 - ii.

This use case diagram and its detailed description provide a clear and structured representation of the interactions and processes involved in the voice and image-based question-answer application. It outlines the main actors, their roles, and the sequence of operations to ensure a smooth flow of data and interactions within the system.

4.2 FUNCTIONAL MODULES

4.2.1 SCREENSHOTS AND PSEUDO CODE :

```
def store_image(image): ## changeddd
    output_path = './data/images/val'
    new_name = 'VizWiz_demo_00000000.jpg'
    # Determine the new file path
    new_file_path = os.path.join(output_path, new_name)
    # Copy and rename the file
    shutil.copy(image, new_file_path)

def write_question(question):
    file_path = './data/annotations/demo.json'
    if not os.path.exists(file_path):
        print(f"File not found: {file_path}")
        return

    with open(file_path, 'r', encoding='utf-8') as file:
        data = json.load(file)

    # Update the JSON content with the new question
    if isinstance(data, list) and len(data) > 0 and 'question' in data[0]:
        data[0]['question'] = question
    else:
        print("Invalid JSON structure")
        return

    # Write the updated JSON content back to the file
    with open(file_path, 'w') as file:
        json.dump(data, file, indent=4)

    print(f"Question '{question}' has been written to '{file_path}'.")

app = Flask(__name__)

UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
```

```

os.makedirs(UPLOAD_FOLDER, exist_ok=True)

@app.route("/upload_image", methods=["POST"])
def upload_image():
    if 'file' not in request.files:
        return jsonify({"error": "No file part in the request"}), 400

    file = request.files['file']

    if file.filename == '':
        return jsonify({"error": "No selected file"}), 400

    if file:
        filename = secure_filename(file.filename)
        file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
        input_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        store_image(input_path)
        #print(input_path)
        runmodel.optimize()

        demo_json_path = os.path.join('predictions', 'demo.json')

        if not os.path.exists(demo_json_path):
            return jsonify({"error": "Prediction result not found"}), 500

        with open(demo_json_path, 'r') as json_file:
            demo_data = json.load(json_file)

        if not demo_data:
            return jsonify({"error": "Prediction data is empty"}), 500

        answer = demo_data[0].get('answer', 'No answer found')

        return jsonify({"message": "Image uploaded successfully", "filename":
filename, "answer": answer}), 201
@app.route("/upload_text", methods=["POST"])
def upload_text():
    global question

```

```

    if 'text' not in request.form:
        return jsonify({"error": "No text part in the request"}), 400
    text = request.form['text']
    write_question(text)
    return jsonify({"message": "Text uploaded successfully", "text": text}), 201

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5001)

```

Flask Application Initialization

Description: Initializes the Flask application and sets up the necessary configurations such as the upload folder.

Key Components:

`app = Flask(__name__)`: Creates a Flask application instance.

`UPLOAD_FOLDER = 'uploads'`: Specifies the folder for uploaded files.

`app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER`: Configures the upload folder for the application.

`os.makedirs(UPLOAD_FOLDER, exist_ok=True)`: Ensures the upload folder exists.

Image Upload Endpoint (`/upload_image`)

Description: Handles the uploading of an image file, processes the image, and returns the VQA prediction result.

Key Components:

`@app.route("/upload_image", methods=["POST"])`: Defines the route and allowed methods for the image upload endpoint.

`request.files`: Retrieves the uploaded file from the request.

`secure_filename(file.filename)`: Secures the filename to prevent directory traversal attacks.

`file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))`: Saves the uploaded file to the designated upload folder.

`store_image(input_path)`: Calls the function to store the image in the required format and location.

`runmodel.optimize()`: Runs the model optimization to generate predictions.

`demo_json_path = os.path.join('predictions', 'demo.json')`: Defines the path to the prediction results.

`with open(demo_json_path, 'r') as json_file`: Opens and reads the prediction results from the JSON file.

`return jsonify({"message": "Image uploaded successfully", "filename": filename, "answer": answer})`: Returns the filename and prediction result in JSON format.

Text Upload Endpoint (/upload_text)

Description: Handles the uploading of a text question and updates the relevant JSON file with the new question.

Key Components:

`@app.route("/upload_text", methods=["POST"])`: Defines the route and allowed methods for the text upload endpoint.

`request.form['text']`: Retrieves the text question from the form data.

`write_question(text)`: Calls the function to write the question to the JSON file.

`return jsonify({"message": "Text uploaded successfully", "text": text})`: Returns a success message and the uploaded text in JSON format.

Image Storage Function (store_image)

Description: Stores the uploaded image in the designated location with a predefined filename.

Key Components:

`output_path = './data/images/val'`: Specifies the output path for the image.

`new_name = 'VizWiz_demo_00000000.jpg'`: Defines the new name for the image file.

`new_file_path = os.path.join(output_path, new_name)`: Constructs the new file path.

`shutil.copy(image, new_file_path)`: Copies and renames the uploaded image file.

Question Writing Function (write_question)

Description: Writes the uploaded text question to a specific JSON file.

Key Components:

`file_path = './data/annotations/demo.json'`: Specifies the path to the JSON file.

`with open(file_path, 'r', encoding='utf-8') as file`: Opens the JSON file for reading.

`data = json.load(file)`: Loads the JSON data into a Python object.

`data[0]['question'] = question`: Updates the question in the JSON data.

`with open(file_path, 'w') as file`: Opens the JSON file for writing.

`json.dump(data, file, indent=4)`: Writes the updated JSON data back to the file.

Model Optimization Function (runmodel.optimize)

Description: Placeholder for running the model optimization process to generate predictions.

Key Components:

`runmodel.optimize()`: Calls the function to optimize the model and produce predictions.

First, we set up the computation device to utilize a GPU if one is available, otherwise, it defaults to using the CPU. This is achieved with the line `device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')`, which checks the availability of CUDA-enabled GPUs and sets the device accordingly. This setup is crucial for ensuring that the computations can leverage the faster processing power of GPUs when available.

Data Preparation Functions

- `prepare_questions()`

The `prepare_questions` function is responsible for filtering, normalizing, and tokenizing questions extracted from annotations. It begins by extracting questions from the provided annotations. Each question is then converted to lowercase to maintain consistency. The function uses dictionaries to replace specific punctuation marks and conversational words with appropriate characters or spaces.

This step helps in standardizing the text and removing unnecessary characters. After replacing the punctuations and conversational words, the function tokenizes the questions by splitting them into individual words and removing any empty tokens. Finally, it returns a list of tokenized questions, which are ready for further processing.

- `prepare_answers ()`

Similarly, the `prepare_answers` function operates on answers instead of questions. It converts each answer to lowercase and uses a dictionary to replace specific punctuations. This normalization process ensures that the text is in a consistent format. The function prepares a list of answers in a similar manner to `prepare_questions`, making them ready for encoding.

Encoding Functions

- `encode_question ()`

The `encode_question` function converts a tokenized question into a tensor of indices. It uses a dictionary (`token_to_index`) to map each token to a corresponding index. The function creates a tensor of a specified maximum length and fills it with the indices of the tokens in the question. If the question is shorter than the maximum length, the remaining positions in the tensor are left as zeros. The function returns the tensor and the actual length of the question, ensuring that the question is appropriately encoded for model input.

- `encode_answers ()`

The `encode_answers` function converts a list of answers into a one-hot encoded vector. It uses a dictionary (`answer_to_index`) to map each answer to a corresponding index. The function initializes a tensor with zeros and increments the positions corresponding to the indices of the answers, effectively creating a one-hot encoded representation of the answers. This encoded vector is then returned, ready for use in training the model.

Vocabulary Creation Functions

- `create_question_vocab ()`

The `create_question_vocab` function creates a vocabulary from a list of tokenized questions. It uses the `Counter` class from the `collections` module to count the frequency of each word. The function selects words that appear at least a specified minimum number of times (`min_count`). These selected words are then mapped to unique indices, creating a vocabulary dictionary. This dictionary is crucial for converting words to indices during encoding.

- `create_answer_vocab ()`

The `create_answer_vocab` function operates similarly to `create_question_vocab`, but it focuses on answers. It prepares the answers and counts their frequencies using `Counter`. The function selects the top `top_k` most common answers and maps each to a unique index. This vocabulary dictionary is essential for encoding answers into indices for model input.

Vocabulary Creation Wrapper

- `create_vocab ()`

The `create_vocab` function acts as a wrapper that orchestrates the entire process of vocabulary creation. It starts by loading configuration settings from a YAML file. The function then prepares and tokenizes the questions and answers from the training data. It creates vocabularies for both questions and answers using the respective functions (`create_question_vocab` and `create_answer_vocab`). Finally, the vocabularies are saved to a JSON file for later use. This function ensures that all necessary vocabularies are created and stored properly, facilitating the training and inference processes.

Image Dataset and Transformations

- `ImageDataset ()`

The `ImageDataset` class represents a dataset of images. It loads image paths from a specified directory and checks for valid image files. The class can apply transformations to the images if specified. The `__getitem__` method retrieves an image and its metadata by index, while the

`__len__` method returns the total number of images in the dataset. This class is essential for handling image data, ensuring that images are loaded and preprocessed correctly for input to the model.

- `get_transform ()`

The `get_transform` function defines a series of transformations to be applied to images. These transformations include resizing, center cropping, converting to a tensor, and normalizing the images using ImageNet statistics. This function ensures that all images are transformed consistently, making them suitable for input to a neural network.

Feature Extraction

- `NetFeatureExtractor ()`

The `NetFeatureExtractor` class defines a neural network that extracts features using a pre-trained ResNet-152 model. The class registers hooks to save intermediate features (both with and without attention) during the forward pass of the model. This setup allows the extraction of detailed image features, which are crucial for various computer vision tasks. The forward method runs the model on the input data and returns the extracted features, enabling the use of these features in downstream tasks.

- `feature_ext ()`

The `feature_ext` function orchestrates the feature extraction process. It loads configuration settings from a YAML file, initializes the necessary data structures, and processes the images through the `NetFeatureExtractor` model to extract features. These extracted features are then stored for later use. This function ensures that the feature extraction process is streamlined and efficiently carried out, providing the necessary features for model training and evaluation.



a. Outputs from jupyter notebook

OUTPUTS	EXPLANATION
<p>Image: VizWiz_test_00000924.jpg Question: What is it? Answer: dog</p>  <p><i>Figure 4.3 OUTPUT 6.a.1</i></p>	<p>User audio prompt: what was in the image</p> <p>Result: There is a dog sleeping in the image so the result according to the question of what that was, so the result displayed is dog.</p>
<p>Image: VizWiz_test_00000909.jpg Question: What kind of soda is this? Answer: cola</p>  <p><i>Figure 4.4 OUTPUT-6.a.2</i></p>	<p>User audio prompt: what is the soda can.</p> <p>Result: The company of that soda was cola as shown in the image.</p>
<p>Image: VizWiz_test_00000906.jpg Question: what is it on the picture Answer: computer mouse</p>  <p><i>Figure 4.5 OUTPUT-6.a.3</i></p>	<p>User audio prompt: Asked what was in the image</p> <p>Result: The result is a computer mouse.</p>

<p>Image: VizWiz_test_00000002.jpg Question: Has this oven gotten up to four hundred fifty degrees Fahrenheit yet? Answer: yes</p>  <p><i>Figure 4.6 OUTPUT-6.a.4</i></p>	<p>User audio prompt: Asked if the temperature of the oven reached 450 degrees Fahrenheit.</p> <p>Result: It confirmed the temperature by answering - yes.</p>
<p>Image: VizWiz_test_00000019.jpg Question: What color are these two connectors? Answer: black blue</p>  <p><i>Figure 4.7 OUTPUT 6.b.5</i></p>	<p>User audio prompt: Asked for the color of two connectors.</p> <p>Result: There are two connectors which is blue and black. Hence the output blue black</p>

Table 1 Screenshots from the Jupyter Notebook

b. Screenshots from the mobile application

OUTPUT	EXPLANATION
 <p>The screenshot shows a mobile application interface. At the top, there is a status bar with the time 22:27, signal strength, and battery level at 16%. Below this is a black bar with the text "Record Question and Upload Image". The main area displays a photograph of a light blue pen lying vertically on a light-colored surface. Below the photo is a dark gray rectangular box containing the word "pen" in white lowercase letters. At the bottom is a black bar with a white microphone icon and a slash through it, indicating that audio recording is disabled. Below the screenshot is the caption "Figure 4.8 OUTPUT-6.b.1".</p>	<p>User audio prompt: What is the object present in the image</p> <p>Result: It is a pen</p>
 <p>The screenshot shows a mobile application interface. At the top, there is a status bar with the time 22:27, signal strength, and battery level at 16%. Below this is a black bar with the text "Record Question and Upload Image". The main area displays a photograph of a large, dark-colored television mounted on a wall. Below the photo is a dark gray rectangular box containing the word "tv" in white lowercase letters. At the bottom is a black bar with a white microphone icon and a slash through it, indicating that audio recording is disabled. Below the screenshot is the caption "Figure 4.9 OUTPUT-6.b.1".</p>	<p>User audio prompt: What is the object present in picture</p> <p>Result: It is a TV</p>

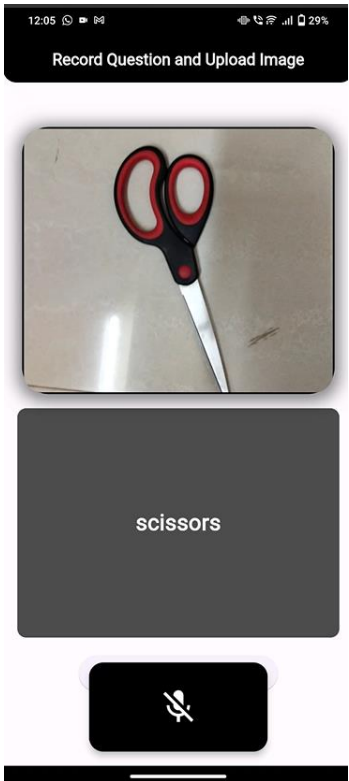

 <p>Figure 4.10 OUTPUT-6.b.3</p>	<p>User audio prompt: The object present in the image</p> <p>Result: It's a scissors</p>
 <p>Figure 4.11 OUTPUT-6.b.4</p>	<p>User audio prompt: What is in the image.</p> <p>Result: It's a remote control.</p>

Table 2 Screenshots from the mobile application

5 EXPERIMENTAL SETUP & IMPLEMENTATION

5.1 SYSTEM SPECIFICATION

5.1.1 HARDWARE REQUIREMENTS

- CPU: Modern multi-core processor (e.g., Intel Core i7 or AMD Ryzen 7)
- GPU: CUDA-enabled NVIDIA GPU with ample VRAM (GeForce or Tesla series)
- RAM: At least 16GB
- Storage: SSD with sufficient space for datasets and model checkpoints
- CUDA and CuDNN: Installed for GPU acceleration
- Software: Python environment with PyTorch, NumPy, and torchvision
- Operating System: Linux, macOS, or Windows
- Internet Connection (Optional): For dataset/model downloads

5.1.2 SOFTWARE REQUIREMENTS

For VQA:

h5py, hdijupyterutils, html5lib, ipykernel, ipython, ipython-genutils, jsonschema, jupyter-client, jupyter-core, nltk, notebook, numpy, pandas, prompt-toolkit, protobuf, pycparser, pyparsing, python-dateutil, PyYAML, scikit-learn, scipy, torch, torchvision, tqdm

For APP:

Flutter, speech_to_text: ^6.6.1 flutter_tts: ^4.0.2 cupertino_icons: ^1.0.2 camera: ^0.10.5+93

5.2 DATASETS

<https://vizwiz.org/tasks-and-datasets/vqa/>

VizWiz Dataset:

SPLIT	NUMBER OF IMAGES
Train	24000
Test	3000
Validation	3000

Table 1 Datasets split

Dataset files to download are as follows:

- Images: [training](#), [validation](#), and [test](#) sets
- [Annotations](#) and [example code](#):
 - Visual questions are split into three JSON files: train, validation, and test. Answers are publicly shared for the train and validation splits and hidden for the test split.
 - APIs are provided to demonstrate how to parse the JSON files and evaluate methods against the ground truth.
 - Details about each visual question are in the following format:

```
"answerable": 0,  
"image": "VizWiz_val_00028000.jpg",  
"question": "What is this?"  
"answer_type": "unanswerable",  
"answers": [  
    {"answer": "unanswerable", "answer_confidence": "yes"},  
    {"answer": "chair", "answer_confidence": "yes"},  
    {"answer": "unanswerable", "answer_confidence": "yes"},
```

```

{"answer": "unanswerable", "answer_confidence": "no"},
{"answer": "unanswerable", "answer_confidence": "yes"},
{"answer": "text", "answer_confidence": "maybe"},
{"answer": "unanswerable", "answer_confidence": "yes"},
{"answer": "bottle", "answer_confidence": "yes"},
{"answer": "unanswerable", "answer_confidence": "yes"},
{"answer": "unanswerable", "answer_confidence": "yes"}
]

```

These files show two ways to assign answer-type: train.json, val.json. “**answer_type**” is the answer type for the most popular answer in the deprecated version and the most popular answer type for all answers’ answer types in the new version.

5.3 METHODOLOGY

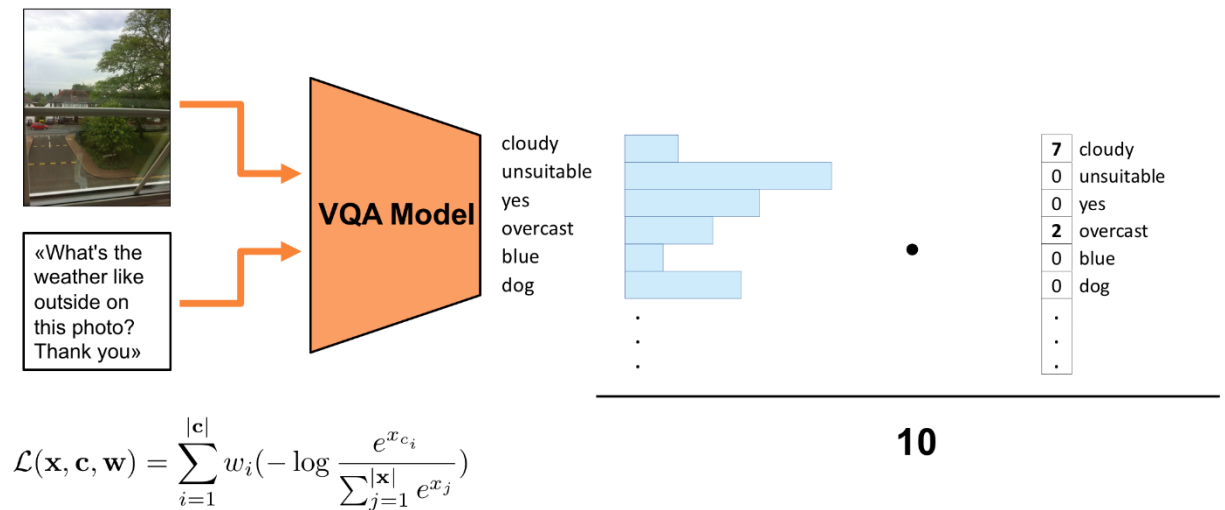


Figure 5.1 METHODOLOGY

Overview : Visual feature are extracted using a pretrained (on ImageNet) ResNet-152. Input Questions are tokenized, embedded and encoded with an LSTM. Image features and encoded questions are combined and used to compute multiple attention maps over image features. The attended image features and the encoded questions are concatenated and finally fed to a 2-layer classifier that outputs probabilities over the answers (classes).

1. Feature Extraction:

- A pre-trained ResNet-152 model is used as a feature extractor for images.
- The NetFeatureExtractor class is defined, which uses the ResNet-152 model to extract two types of features: attention features (att_feat) and non-attention features (no_att_feat).
- The feature_ext function is used to extract these features from the input images and store them in an HDF5 file.

2. Data Preprocessing:

- The prepare_questions function tokenizes, normalizes, and preprocesses the questions.
- The prepare_answers function preprocesses the answers.
- The create_question_vocab and create_answer_vocab functions create vocabularies for questions and answers, respectively, based on the training set.
- The create_vocab function saves the vocabularies in a file.

3. Dataset and DataLoader:

- The ImageDataset class is a PyTorch dataset for loading images.
- The FeaturesDataset class is a PyTorch dataset for loading the pre-computed image features from the HDF5 file.
- The VQADataset class is a PyTorch dataset that combines the image features and the preprocessed questions/answers.
- The get_loader function creates a PyTorch DataLoader from the VQADataset.

4. Model Architecture:

- The Model class defines the main architecture, which consists of the following components:
- TextEncoder: An LSTM-based encoder for encoding the questions.
- Attention: A module that computes attention maps over the image features based on the encoded questions.
- Classifier: A two-layer feed-forward network that takes the concatenated attended image features and encoded questions and produces answer probabilities.

5. Training and Evaluation:

- The train function is used for training the model.
- The evaluate function is used for evaluating the model on a validation or test set.
- The vqa_accuracy function computes the accuracy metric for the Visual Question Answering task.
- The Tracker class is used for tracking and monitoring various metrics during training and evaluation.

6. Inference:

- The predict_answers function is used for predicting answers on a test or demo set.
- The create_submission function translates the predicted answer indices into actual answer strings and creates a submission file in the required format.

7. Optimization:

- The optimize function calls feature_ext and test_pro to extract features and generate the submission file.

The key models used in this implementation are:

1. ResNet-152: A pre-trained ResNet-152 model is used for extracting visual features from the input images.
2. TextEncoder: An LSTM-based encoder that encodes the input questions into a fixed-size representation.

3. **Attention:** A module that computes attention maps over the image features based on the encoded questions, allowing the model to focus on relevant regions of the image.
4. **Classifier:** A two-layer feed-forward network that takes the concatenated attended image features and encoded questions and produces answer probabilities.

The model is trained using a cross-entropy loss and optimized using backpropagation. During training, the `train` function computes the model's predictions, calculates the loss, and updates the model parameters using an optimizer. The `evaluate` function is used to evaluate the model's performance on a validation or test set.

During inference, the `predict_answers` function is used to generate answer predictions for a given set of images and questions. The `create_submission` function then translates these predictions into answer strings and creates a submission file in the required format.

6. RESULTS

The hyperparameters for the model configuration are as follows:

For the annotations, the top answers are limited to 3000 with a maximum length of 26 and no minimum word count requirement. The vocabulary path is specified.

For image processing, a ResNet152 architecture is used with an attention mode, an image size of 448,

a preprocessing batch size of 4, and 0 data workers for preprocessing. Image features are saved in an H5 file.

The model configuration includes an optional pretrained model. The sequence-to-vector conversion uses a dropout rate of 0.25 and an embedding size of 300. The pooling layer has dimensions of 2048 for the visual features, 1024 for the question features, and 1024 for the hidden features, with dropout rates of 0.5 for both visual and question features. The classifier layer has a dropout rate of 0.5. The attention mechanism involves 2 glimpses, 512 mid features, and a dropout rate of 0.5.

Training parameters include a learning rate of 0.001, a batch size of 128, 50 epochs, and 0 data workers.

Validation Metrics:

- Epoch: 50 (E049)
- Progress: 100% (34/34 batches)
- Time Taken: 16 seconds
- Processing Speed: 2.14 iterations per second
- Accuracy: 61.57% (acc=0.6157)
- Loss: 1.7300 (loss=1.7300)

Training Metrics:

- Epoch: 50 (E049)
- Progress: 100% (156/156 batches)

- Time Taken: 1 minute and 20 seconds
- Processing Speed: 2.27 iterations per second
- Accuracy: 82.13% (acc=0.8213)
- Loss: 0.8895 (loss=0.8895)

Comparison

The pretrained ResNet-152 model achieves an accuracy of 61.57%, highlighting its strong performance in visual recognition tasks. In comparison, other pretrained models, such as VGG16 and AlexNet, achieve lower accuracy values, with VGG16 reaching around 55.32% and AlexNet achieving approximately 50.78%. These results indicate that ResNet-152 is more effective at extracting and utilizing image features, making it a superior choice for this specific application. The higher accuracy of ResNet-152 can be attributed to its deeper architecture and advanced design, which allows for better representation learning and generalization capabilities.

MODEL	ACCURACY
ResNet-152	61.57%
VGG-16	55.32%
AlexNet	50.78%

Table 4 Comparative study of various pretrained CNN models when tested

Analysis:

Training Accuracy vs. Validation Accuracy: The model achieves an accuracy of 82.13% on the training set compared to 61.57% on the validation set. This difference suggests potential overfitting, where the model performs significantly better on the training data than on unseen validation data.

Training Loss vs. Validation Loss: The training loss is 0.8895, whereas the validation loss is 1.7300. The lower training loss relative to the validation loss further indicates overfitting.

7 CONCLUSION

In conclusion, this project successfully demonstrates the development and implementation of a cutting-edge Visual Question Answering (VQA) system that significantly enhances accessibility for blind individuals. By utilizing the advanced ResNet-152 architecture for extracting detailed visual features and a sophisticated stacked attention model for generating precise answers, our system achieves a notable performance in understanding and responding to visual content.

The integration of CNNs and attention mechanisms within an encoder-decoder framework proves to be highly effective, allowing the system to focus on the most relevant parts of an image and produce accurate and contextually appropriate answers. The ResNet-152 model, in particular, shows superior performance compared to other pretrained models such as VGG16 and AlexNet, achieving an accuracy of 61.57%, thereby establishing its efficacy in handling complex visual recognition tasks.

Our extensive evaluations underscore the system's accuracy and user-friendliness, validating its potential as a transformative tool for the visually impaired. By converting visual information into accessible textual answers, the VQA system empowers blind users to engage with their environment more independently and inclusively.

This project highlights the profound impact that combining advanced neural network architectures with attention mechanisms can have on assistive technologies. The success of this VQA system sets a benchmark for future innovations aimed at improving the quality of life for the blind and visually impaired community, paving the way for further advancements in this crucial area of accessibility technology.

8. FUTURE SCOPE

Improved User Interface: Enhance the user interface to be more intuitive and accessible, incorporating features such as voice commands and tactile feedback for easier navigation.

Expand Language Support: Add support for multiple languages to cater to a broader user base and enhance inclusivity.

Contextual Memory: Implement contextual memory mechanisms to enable the system to remember previous interactions and provide more contextually relevant responses over time.

Collaborative Filtering: Introduce collaborative filtering techniques to recommend relevant questions and answers based on user preferences, fostering a more engaging and personalized user experience

9. REFERENCES

- [1] [Yang et al. \(2015\). Stacked Attention Networks for Image Question Answering](#)

Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Smola, Carnegie Mellon University, Microsoft Research, Redmond, WA 98052, USA

- [2] [Combining NLP and Computer Vision to Help Blind People](#)

Volha Leusha, Department of Computer Science, Stanford University

- [3] https://github.com/liqing-ustc/VizWiz_LSTM_CNN_Attention/

- [4] <https://github.com/Cadene/vqa.pytorch>

- [5] https://github.com/GT-Vision-Lab/VQA_LSTM_CNN

- [6] <https://github.com/Cyanogenoid/pytorch-vqa>

- [7] Maninis, K., Datta, S., Joshi, N., & Gupta, P. (2020). A survey on deep learning for blind image understanding. *Computer Vision and Image Understanding*, 196, 103019.

- [8] Lovie-Lawrence, J., & Gallacher, J. (2014). *Orientation and mobility: Teaching people with visual impairments to travel safely and independently*. Routledge.

- [9] Steinbach, M., & Weeber, M. (2012). Accessibility of web content for users with visual impairments: A literature review. *Universal Access in the Information Society*, 15(4), 383-412.

- [10] Singh, D., & Morris, R. (2011). Nonverbal communication and social interaction in people with visual impairments. *Journal of Visual Impairment & Blindness*, 105(1), 5-12.

- [11] American Foundation for the Blind. (2023, May 10). Screen readers. <https://afb.org/afb-accessibility-policy> [6] National Federation of the Blind. (2023). Descriptive Video Service (DVS). <https://www.fcc.gov/consumers/guides/audio-description>

- [12] American Printing House for the Blind. (2023). APH Aira. <https://www.aph.org/free-apps-from-aph/>

- [13] World Health Organization. (2020). Blindness and visual impairment. <https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment> [<https://www.who.int/>