INNOMATICS
RESEARCH LABS

PROJECT REPORT ON:

Building AI Powered Solution for Assisting Visually Impaired Individuals

SUBMITTED BY:

Waheed Ullah

UNIQUE INTERN ID:

IN9240885

Building AI Powered Solution for Assisting Visually Impaired Individuals

Problem Statement:

- Visually impaired individuals often face challenges in understanding their environment, reading visual content, and performing tasks that rely on sight.
- This project leverages Generative AI to assist visually impaired individuals by providing functionalities like real-time scene understanding, text-to-speech conversion, and object detection.

Objectives:

- Provide real-time scene understanding.

- Extract and convert text from images to speech.

- Identify objects and obstacles for safe navigation.

- Offer personalized assistance for daily tasks.

Implementation Details

Technologies Used

1. ***LangChain:*** Conversational AI capabilities.

2. ***Streamlit:*** User-friendly web application interface.

3. ***Google Generative AI (Gemini API):*** Generates detailed scene descriptions.

4. ***Pytesseract:*** Optical Character Recognition (OCR) to extract text from images.

5. ***pyttsx3:*** Text-to-Speech engine for audio output.

Functionalities Implemented:

1. Real-Time Scene Understanding

- Generates detailed scene descriptions using Google Generative AI.

- Output includes objects, overall scene description, and safety suggestions.

2. Text-to-Speech Conversion

- Extracts text from images using OCR.

- Converts the extracted text to audible speech for accessibility.

User Interaction Flow:

1. ***Image Upload:*** Users upload an image through the Streamlit app.

2. ***Feature Selection:*** Choose between scene description, text extraction, or text-to-speech.

3. ***Processing and Output:***

   - Scene Understanding: Detailed descriptions generated.

   - OCR and Speech: Text extracted and read aloud.

Outputs:

Example 1: Scene Understanding

- Uploaded Image: Two children playing.
- Generated Description:
    - Two young children are sitting on a white floor playing with colorful building blocks. The boy, on the left, has blond curly hair and wears a brightly striped polo shirt in rainbow colors and blue pants. The girl, on the right, has light brown hair adorned with a colorful hair clip. She's wearing a rainbow-striped t-shirt and white shorts. They are collaboratively building a tall tower of various shaped and sized blocks. The blocks are primarily square and rectangular, but some are also triangular and cylindrical. They come in vibrant hues of red, orange, yellow, green, blue, and purple. The blocks are spread out on the floor around them. In the background, slightly out of focus, is a white stool with a colorful abacus on top and a glimpse of a light blue stuffed animal. The setting appears to be a bright playroom or nursery. "

Example 2: Text Extraction and Speech

- Uploaded Image: A python programming code quiz
- Extracted Text: "Question 5: Which of the following statements is true?
  Choose the best answer(s):
  A) Statement A and Statement B are both true. C) Statement A is false, but Statement B is true. O B) Statement A is true, but Statement B is false.

D) Statement A and Statement B are both false.."

- Speech Output: Text is read aloud as audio.

Implementation Code:

```python

import streamlit as st
import google.generativeai as genai
from PIL import Image, ImageDraw
import pytesseract
import pyttsx3
import torch
import os
from torchvision import transforms
from torchvision.models.detection import fasterrcnn_resnet50_fpn

import google.generativeai as genai

genai.configure(api_key="AIzaSyC2ayB9fYcTTQUbytcGX6A6lsYgwvu2enU")
response = genai.GenerativeModel('gemini-1.5-pro').generate_content("Hello, AI!")
print(response.text)


# Load environment variables
from dotenv import load_dotenv
# Set Tesseract OCR path
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'
load_dotenv()

# Streamlit App
st.set_page_config(page_title="👁️ VisionCare AI", layout="centered", page_icon="🌟")


# response function
```

```
def get_response(input_prompt, image_data):
    model = genai.GenerativeModel("gemini-1.5-pro")
    response = model.generate_content([input_prompt, image_data[0]])
    return response.text


# function to convert image to bytes
def image_to_bytes(uploaded_file):
    try:
        bytes_data = uploaded_file.getvalue()

        image_parts = [{"mime_type": uploaded_file.type, "data": bytes_data}]

        return image_parts

    except Exception as e:
        raise FileNotFoundError(
            f"Failed to process image. Please try again. Error: {e}"
        )


# function to extract text from image
def extract_text_from_image(uploaded_file):
    try:
        img = Image.open(uploaded_file)

        # pytesseract to extract text
        extracted_text = pytesseract.image_to_string(img)

        if not extracted_text.strip():
            return "No text found in the image."

        return extracted_text

    except Exception as e:
        raise ValueError(f"Failed to extract text. Error: {e}")


# function for text to speech


def text_to_speech_pyttsx3(text):
    try:
        # Initialize TTS engine
        engine = pyttsx3.init()

        # Speak the text
        engine.say(text)
        engine.runAndWait()

        engine.stop()  # Stop the engine after use

    except Exception as e:
        raise RuntimeError(f"Failed to convert text to speech. Error: {e}")


# Load object detection model (Faster R-CNN)
@st.cache_resource
def load_object_detection_model():
    model = fasterrcnn_resnet50_fpn(pretrained=True)
    model.eval()
    return model


object_detection_model = load_object_detection_model()


def detect_objects(image, threshold=0.5, iou_threshold=0.5):
    try:
        # Transform image to tensor
        transform = transforms.Compose([transforms.ToTensor()])
        img_tensor = transform(image)

        # Get predictions
        predictions = object_detection_model([img_tensor])[0]

        # Perform Non-Maximum Suppression
        keep = torch.ops.torchvision.nms(
            predictions["boxes"], predictions["scores"], iou_threshold
        )

        # Filter results based on NMS and score threshold
        filtered_predictions = {
            "boxes": predictions["boxes"][keep],
            "labels": predictions["labels"][keep],
            "scores": predictions["scores"][keep],
        }

        return filtered_predictions
    except Exception as e:
        raise RuntimeError(f"Failed to detect objects. Error: {e}")


# COCO class labels (91 categories)
COCO_CLASSES = [
    "__background__",
    "person",
    "bicycle",
    "car",
    "motorcycle",
    "airplane",
    "bus",
    "train",
    "truck",
    "boat",
    "traffic light",
    "fire hydrant",
    "N/A",
    "stop sign",
    "parking meter",
    "bench",
    "bird",
    "cat",
    "dog",
    "horse",
    "sheep",
    "cow",
    "elephant",
    "bear",
    "zebra",
    "giraffe",
    "N/A",
    "backpack",
    "umbrella",
    "N/A",
    "N/A",
    "handbag",
    "tie",
    "suitcase",
    "frisbee",
    "skis",
    "snowboard",
    "sports ball",
    "kite",
    "baseball bat",
    "baseball glove",
    "skateboard",
    "surfboard",
    "tennis racket",
    "bottle",
    "N/A",
    "wine glass",
    "cup",
    "fork",
    "knife",
    "spoon",
    "bowl",
    "banana",
    "apple",
    "sandwich",
    "orange",
    "broccoli",
    "carrot",
    "hot dog",
    "pizza",
    "donut",
    "cake",
    "chair",
    "couch",
    "potted plant",
    "bed",
    "N/A",
    "dining table",
    "N/A",
    "N/A",
    "toilet",
    "N/A",
    "tv",
    "laptop",
    "mouse",
    "remote",
    "keyboard",
    "cell phone",
    "microwave",
    "oven",
    "toaster",
    "sink",
    "refrigerator",
    "N/A",
    "book",
    "clock",
    "vase",
    "scissors",
    "teddy bear",
    "hair drier",
    "toothbrush",
]


# Highlight detected objects in the image
def draw_boxes(image, predictions, threshold=0.5):
    draw = ImageDraw.Draw(image)
    labels = predictions["labels"]
    boxes = predictions["boxes"]
    scores = predictions["scores"]

    for label, box, score in zip(labels, boxes, scores):
        if score > threshold:
            x1, y1, x2, y2 = box
            class_name = COCO_CLASSES[label.item()]  # Map label ID to class
name
            draw.rectangle([x1, y1, x2, y2], outline="yellow", width=3)
            draw.text((x1, y1), f"{class_name} ({score:.2f})", fill="black")
    return image


# Response function for Personalized Assistance
def get_assistance_response(input_prompt, image_data):
    model = genai.GenerativeModel("gemini-1.5-pro")
    response = model.generate_content([input_prompt, image_data[0]])
    return response.text


# Prompt Engineering
input_prompt = """
```

```
You are an AI assistant designed to assist visually impaired individuals
by analyzing images and providing descriptive outputs.
Your task is to:
- Analyze the uploaded image and describe its content in clear and simple language.
- Provide detailed information about objects, people, settings, or activities in the scene.
"""



st.title("👁️ VisionCare AI: Vision Assistant for All")

st.markdown(
    """
    **Empowering Accessibility with Cutting-Edge Vision Intelligence**
    **Features:**
    - 📷 **Scene Analysis**: Understand complex visuals.
    - 🚧 **Object Detection**: Identify objects and navigate obstacles.
    - 🧑‍🤝‍🧑 **Personalized Assistance**: For specific tasks like label reading.
    - 📝 **Text-to-Speech**: Converts text into clear, natural speech.
"""
)
st.markdown("---")
# st.markdown(
#     """
# **Features:**
# - **Real-Time Scene Analysis**: Describe scenes from uploaded images.
# - **Object and Obstacle Detection**: Detect objects/obstacles for safe navigation.
# - **Personalized Assistance**: Provide task-specific guidance.
# - **Text-to-Speech Conversion**: Convert text to audio descriptions.
# """
# )

# File uploader
st.sidebar.header("📂 **Upload an Image**")
uploaded_file = st.sidebar.file_uploader(
    "Choose an image:", type=["jpg", "jpeg", "png", "webp"]
)

if uploaded_file:
    st.sidebar.image(uploaded_file, caption="Uploaded Image",
use_container_width=True)

# Buttons
stop_audio_button = st.button("Stop Audio ⏹️")

bt1, bt2, bt3, bt4 = st.columns(4)
scene_analysis_button = bt1.button("🏞️ Describe Scene")
object_detection_button = bt2.button("🚧 Detect Objects")
assist_button = bt3.button("🤖 Assist Tasks")
text_tts_button = bt4.button("📝 Extract Text")

# Object Detection
if object_detection_button and uploaded_file:
    with st.spinner("🔍 Detecting objects..."):
        st.subheader("🚧 Detected Objects:")
        image = Image.open(uploaded_file)
        predictions = detect_objects(image)
        image_with_boxes = draw_boxes(image.copy(), predictions)
        st.image(
            image_with_boxes,         caption="Objects         Highlighted",
use_container_width=True
        )

# Personalized Assistance
if assist_button and uploaded_file:
    with st.spinner("🤔 Providing task-specific assistance..."):
        st.subheader("🤖 Assistance Output:")
        image_data = image_to_bytes(uploaded_file)
        assist_prompt = """
        You are a helpful AI assistant. Analyze the uploaded image and identify
tasks
        you can assist with, such as recognizing objects or reading labels.
        """
        response = get_assistance_response(assist_prompt, image_data)
        st.write(response)

        # Convert response to audio
        text_to_speech_pyttsx3(response)

# Scene Analysis
if scene_analysis_button and uploaded_file:
    with st.spinner("🖼️ Analyzing Image..."):
        st.subheader("🏞️ Scene Description:")
        image_data = image_to_bytes(uploaded_file)
        response = get_response(input_prompt, image_data)
        st.write(response)

        # Convert response to audio
        text_to_speech_pyttsx3(response)

# Extract Text and TTS
if text_tts_button and uploaded_file:
    with st.spinner("📝 Extracting text from image..."):
        text = extract_text_from_image(uploaded_file)

        # Display the text
        st.write(text)

        # Convert text to audio
        if text.strip():
            text_to_speech_pyttsx3(text)

if stop_audio_button:
    try:
        # Initialize TTS engine if not already initialized
        if "tts_engine" not in st.session_state:
            st.session_state.tts_engine = pyttsx3.init()

        # Stop the audio playback
        st.session_state.tts_engine.stop()
        st.success("Audio playback stopped.")
    except Exception as e:
        st.error(f"Failed to stop the audio. Error: {e}")
```

Evaluation Criteria

- Uniqueness of Implementation: The project integrates cutting-edge Generative AI and OCR with a simple UI.

- Successful Features: Implements at least two core functionalities.
- Technical Accuracy: Uses Google Generative AI, OCR, and TTS effectively.
- Documentation: Provides clear code and explanation.

Conclusion

- This project demonstrates an effective application of AI to assist visually impaired individuals. It combines real-time scene understanding with OCR and text-to-speech capabilities, enabling enhanced accessibility and independence.