**Project Documentation: Image Recognition Chatbot**

**Project Overview**

The Image Recognition Chatbot project integrates image recognition and natural language processing to create an intelligent system that can analyze images and respond to questions related to them. This document outlines the complete process, including code, file directories, and a summary of each step involved in building and running the project.

**File Directories**

* **Project Root Directory**: E:/2024/SIH Projects/image\_recognition\_chatbot/
  + **data/**
    - **images/**
      * train2017/ (Training images)
      * val2017/ (Validation images)
    - **text/**
      * train.csv (Training data in CSV format)
      * val.csv (Validation data in CSV format)
    - **annotations/**
      * JSON files containing image annotations
  + **models/**
    - image\_recognition\_model.h5 (Saved image recognition model)
    - nlp\_model/ (Directory containing the saved NLP model)
  + **scripts/**
    - gui.py (GUI application code)
    - processing.py (Image processing and recognition code)
    - train\_image\_model.py (Code to train the image recognition model)
    - train\_nlp\_model.py (Code to train the NLP model)
    - integrate\_models.py (Code to integrate image and NLP models)
    - quantize.py (Code to quantize the model)
  + **results/** (Directory for training results)

**Detailed Steps and Code**

**1. Setup and Dependencies**

Ensure that you have the necessary libraries installed:

bash

Copy code

pip install tensorflow transformers datasets pandas pillow

**2. Image Recognition Model Training (train\_image\_model.py)**

This script trains an image recognition model using MobileNetV2 with data augmentation.

**Code:**

python

Copy code

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import MobileNetV2

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D

from tensorflow.keras.models import Model

import os

# Define paths

train\_dir = 'E:/2024/SIH Projects/image\_recognition\_chatbot/data/images/train2017/'

val\_dir = 'E:/2024/SIH Projects/image\_recognition\_chatbot/data/images/val2017/'

# Data Augmentation

train\_datagen = ImageDataGenerator(

rescale=1./255,

rotation\_range=20,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True,

fill\_mode='nearest'

)

val\_datagen = ImageDataGenerator(rescale=1./255)

train\_generator = train\_datagen.flow\_from\_directory(

train\_dir,

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical'

)

val\_generator = val\_datagen.flow\_from\_directory(

val\_dir,

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical'

)

# Build Model

base\_model = MobileNetV2(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

x = base\_model.output

x = GlobalAveragePooling2D()(x)

x = Dense(1024, activation='relu')(x)

predictions = Dense(train\_generator.num\_classes, activation='softmax')(x)

model = Model(inputs=base\_model.input, outputs=predictions)

# Freeze base model layers

for layer in base\_model.layers:

layer.trainable = False

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Train Model

history = model.fit(

train\_generator,

steps\_per\_epoch=train\_generator.samples // train\_generator.batch\_size,

validation\_data=val\_generator,

validation\_steps=val\_generator.samples // val\_generator.batch\_size,

epochs=10

)

# Save the model

model.save('E:/2024/SIH Projects/image\_recognition\_chatbot/models/image\_recognition\_model.h5')

**Summary:**

* Loads images using data augmentation.
* Builds and trains a MobileNetV2-based model.
* Saves the trained model.

**3. NLP Model Training (train\_nlp\_model.py)**

Trains a BERT-based model for question classification.

**Code:**

python

Copy code

from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments

from datasets import load\_dataset

import pandas as pd

# Load dataset

dataset = load\_dataset('csv', data\_files={'train': 'E:/2024/SIH Projects/image\_recognition\_chatbot/data/text/train.csv', 'validation': 'E:/2024/SIH Projects/image\_recognition\_chatbot/data/text/val.csv'})

# Load tokenizer and model

tokenizer = BertTokenizer.from\_pretrained('bert-base-uncased')

model = BertForSequenceClassification.from\_pretrained('bert-base-uncased', num\_labels=len(dataset['train'].features['label'].names))

def tokenize\_function(examples):

return tokenizer(examples['text'], padding="max\_length", truncation=True)

tokenized\_datasets = dataset.map(tokenize\_function, batched=True)

# Define training arguments

training\_args = TrainingArguments(

output\_dir='./results',

evaluation\_strategy="epoch",

per\_device\_train\_batch\_size=8,

per\_device\_eval\_batch\_size=8,

num\_train\_epochs=3,

weight\_decay=0.01,

)

trainer = Trainer(

model=model,

args=training\_args,

train\_dataset=tokenized\_datasets['train'],

eval\_dataset=tokenized\_datasets['validation']

)

# Train Model

trainer.train()

# Save the model

model.save\_pretrained('E:/2024/SIH Projects/image\_recognition\_chatbot/models/nlp\_model')

tokenizer.save\_pretrained('E:/2024/SIH Projects/image\_recognition\_chatbot/models/nlp\_model')

**Summary:**

* Loads and tokenizes text data.
* Trains a BERT model for sequence classification.
* Saves the trained model and tokenizer.

**4. Image and NLP Models Integration (integrate\_models.py)**

Integrates the image recognition and NLP models.

**Code:**

python

Copy code

import tensorflow as tf

from transformers import BertTokenizer, BertForSequenceClassification

import numpy as np

from PIL import Image

import os

# Load models

image\_model = tf.keras.models.load\_model('E:/2024/SIH Projects/image\_recognition\_chatbot/models/image\_recognition\_model.h5')

tokenizer = BertTokenizer.from\_pretrained('E:/2024/SIH Projects/image\_recognition\_chatbot/models/nlp\_model')

nlp\_model = BertForSequenceClassification.from\_pretrained('E:/2024/SIH Projects/image\_recognition\_chatbot/models/nlp\_model')

def recognize\_image(image\_path):

image = Image.open(image\_path).resize((224, 224))

image\_array = np.array(image) / 255.0

image\_array = np.expand\_dims(image\_array, axis=0)

predictions = image\_model.predict(image\_array)

return predictions

def generate\_response(query):

inputs = tokenizer(query, return\_tensors="pt")

outputs = nlp\_model(\*\*inputs)

logits = outputs.logits

predicted\_class = np.argmax(logits.detach().numpy())

return predicted\_class

# Example usage

image\_path = 'path\_to\_image.jpg'

query = 'What is in this image?'

predicted\_object = recognize\_image(image\_path)

response = generate\_response(query)

print(f'Predicted Object: {predicted\_object}')

print(f'Response: {response}')

**Summary:**

* Integrates image recognition and NLP models.
* Provides functions to recognize images and generate responses.

**5. Quantization (quantize.py)**

Quantizes the model for optimization.

**Code:**

python

Copy code

import tensorflow as tf

def quantize\_model(saved\_model\_dir, quantized\_model\_dir):

"""

Applies post-training quantization to a TensorFlow model.

:param saved\_model\_dir: Directory of the saved original model.

:param quantized\_model\_dir: Directory to save the quantized model.

"""

# Load the pre-trained model

converter = tf.lite.TFLiteConverter.from\_saved\_model(saved\_model\_dir)

# Set the optimization strategy to quantize the model

converter.optimizations = [tf.lite.Optimize.DEFAULT]

# Convert the model to TFLite format

quantized\_model = converter.convert()

# Save the quantized model

with open(quantized\_model\_dir, 'wb') as f:

f.write(quantized\_model)

print("Quantized model saved successfully at:", quantized\_model\_dir)

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

# Specify the paths

saved\_model\_dir = "path\_to\_your\_fine\_tuned\_model" # Update with the path to the fine-tuned model

quantized\_model\_dir = "path\_to\_save\_quantized\_model/model.tflite" # Update with the path to save the quantized model

# Perform quantization

quantize\_model(saved\_model\_dir, quantized\_model\_dir)

**Summary:**

* Quantizes the trained model to optimize performance.
* Saves the quantized model.

**6. GUI (gui.py)**

Provides a graphical user interface for interacting with the chatbot.

**Code:**

python

Copy code

import tkinter as tk

from tkinter import filedialog, Label, Button, Text, Listbox, Scrollbar, messagebox, Frame

from PIL import Image, ImageTk

# Initialize lists to keep track of images and questions

uploaded\_images = []

questions = []

def upload\_images():

file\_paths = filedialog.askopenfilenames()

for file\_path in file\_paths:

try:

img = Image.open(file\_path)

img = img.resize((100, 100), Image.LANCZOS) # Resize for thumbnail

img\_tk = ImageTk.PhotoImage(img)

uploaded\_images.append((file\_path, img\_tk))

images\_listbox.insert(tk.END, file\_path)

except Exception as e:

messagebox.showerror("Error", f"Unable to open image: {e}")

def display\_image(event):

selected\_index = images\_listbox.curselection()

if selected\_index:

file\_path = images\_listbox.get(selected\_index)

for path, img\_tk in uploaded\_images:

if path == file\_path:

image\_label.configure(image=img\_tk)

image\_label.image = img\_tk

preview\_label.config(text=f"Image Preview: {file\_path}")

# Set current image for questions

global current\_image

current\_image = file\_path

def submit\_question():

if 'current\_image' not in globals():

messagebox.showwarning("Selection Error", "Please select an image first.")

return

question = query\_entry.get("1.0", tk.END).strip()

if not question:

messagebox.showwarning("Input Error", "Please enter a question.")

return

questions.append((current\_image, question))

questions\_listbox.insert(tk.END, f"{current\_image}: {question}")

# Placeholder for chatbot's response

response = f"Answering your question: '{question}' about {current\_image}"

response\_label.config(text=response)

def clear\_questions():

query\_entry.delete("1.0", tk.END)

response\_label.config(text="")

questions\_listbox.delete(0, tk.END)

def clear\_all():

uploaded\_images.clear()

questions.clear()

images\_listbox.delete(0, tk.END)

query\_entry.delete("1.0", tk.END)

response\_label.config(text="")

image\_label.configure(image='')

global current\_image

current\_image = None

# Main GUI window

root = tk.Tk()

root.title("Image Recognition Chatbot")

root.configure(bg='white')

# Frame for image upload and display

upload\_frame = Frame(root, bg='white', padx=10, pady=10)

upload\_frame.pack(side=tk.TOP, fill=tk.X)

upload\_label = Label(upload\_frame, text="Upload Images", font=('Arial', 14, 'bold'), bg='white', fg='black')

upload\_label.pack(pady=(0, 5))

upload\_button = Button(upload\_frame, text="Upload Images", command=upload\_images, width=20, bg='black', fg='white', activebackground='gray')

upload\_button.pack(pady=5)

# Frame for image preview

preview\_frame = Frame(root, bg='white', padx=10, pady=10)

preview\_frame.pack(side=tk.TOP, fill=tk.BOTH, expand=True)

preview\_label = Label(preview\_frame, text="Image Preview", font=('Arial', 14, 'bold'), bg='white', fg='black')

preview\_label.pack(pady=(0, 5))

image\_label = Label(preview\_frame, bg='lightgray', relief='sunken')

image\_label.pack(padx=10, pady=10)

# Frame for image list

list\_frame = Frame(root, bg='white', padx=10, pady=10)

list\_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

list\_label = Label(list\_frame, text="Uploaded Images", font=('Arial', 14, 'bold'), bg='white', fg='black')

list\_label.pack(pady=(0, 5))

images\_listbox = Listbox(list\_frame, width=50, height=15, bg='white', fg='black', borderwidth=2, relief='groove')

images\_listbox.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

images\_listbox.bind('<<ListboxSelect>>', display\_image)

scrollbar = Scrollbar(list\_frame, orient=tk.VERTICAL, command=images\_listbox.yview)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

images\_listbox.config(yscrollcommand=scrollbar.set)

# Frame for question input and submission

query\_frame = Frame(root, bg='white', padx=10, pady=10)

query\_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

query\_label = Label(query\_frame, text="Ask a Question", font=('Arial', 14, 'bold'), bg='white', fg='black')

query\_label.pack(pady=(0, 5))

query\_entry = Text(query\_frame, height=4, width=40, wrap=tk.WORD, bg='white', fg='black', borderwidth=2, relief='groove')

query\_entry.pack(pady=(0, 10))

button\_frame = Frame(query\_frame, bg='white')

button\_frame.pack(fill=tk.X)

submit\_button = Button(button\_frame, text="Submit Question", command=submit\_question, width=20, bg='black', fg='white', activebackground='gray')

submit\_button.pack(side=tk.LEFT, padx=5, pady=5)

clear\_button = Button(button\_frame, text="Clear Questions", command=clear\_questions, width=20, bg='gray', fg='white', activebackground='darkgray')

clear\_button.pack(side=tk.LEFT, padx=5, pady=5)

clear\_all\_button = Button(button\_frame, text="Clear All", command=clear\_all, width=20, bg='gray', fg='white', activebackground='darkgray')

clear\_all\_button.pack(side=tk.LEFT, padx=5, pady=5)

# Frame for questions list and response

response\_frame = Frame(root, bg='white', padx=10, pady=10)

response\_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

response\_label = Label(response\_frame, text="Chatbot's Response", font=('Arial', 14, 'bold'), bg='white', fg='black')

response\_label.pack(pady=10)

questions\_listbox = Listbox(response\_frame, width=50, height=15, bg='white', fg='black', borderwidth=2, relief='groove')

questions\_listbox.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

scrollbar\_questions = Scrollbar(response\_frame, orient=tk.VERTICAL, command=questions\_listbox.yview)

scrollbar\_questions.pack(side=tk.RIGHT, fill=tk.Y)

questions\_listbox.config(yscrollcommand=scrollbar\_questions.set)

# Set initial image selection to None

current\_image = None

# Start the GUI loop

root.mainloop()

**Summary:**

* Provides a GUI to upload images, ask questions, and display responses.
* Allows interaction with the uploaded images and queries.

**Project Summary**

1. **Training the Image Recognition Model**:
   * Utilized MobileNetV2 for image classification with data augmentation.
   * Saved the trained model for inference.
2. **Training the NLP Model**:
   * Used BERT for question classification.
   * Saved the tokenizer and model.
3. **Integrating Models**:
   * Combined image recognition and NLP models to generate responses based on image analysis.
4. **Quantization**:
   * Optimized the model for performance using TensorFlow Lite.
5. **GUI Application**:
   * Developed a Tkinter-based GUI for user interaction.