Bahria University,

Karachi Campus

## LAB EXPERIMENT NO.

8

## LIST OF TASKS

|  |  |
| --- | --- |
| **TASK NO** | **OBJECTIVE** |
| **1** | You are tasked with developing a comprehensive GUI-based application that leverages transformer-based models to provide text summarization, image classification, speech recognition, and audio sentiment analysis functionalities in a single application with select model option. The application should offer a seamless user experience and enable users to perform multiple tasks within a single interface. Also, Implement robust error handling mechanisms to handle exceptions gracefully. Display informative error messages to users in case of failures or errors. |
| **2** | You need to develop a user-friendly graphical interface for an Optical Character Recognition (OCR) application using Python's Pytesseract library. |
|  |  |

Submitted On:

24 April 2024

(Date: DD/MM/YY)

**TASK # 1:** You are tasked with developing a comprehensive GUI-based application that leverages transformer-based models to provide text summarization, image classification, speech recognition, and audio sentiment analysis functionalities in a single application with select model option. The application should offer a seamless user experience and enable users to perform multiple tasks within a single interface. Also, Implement robust error handling mechanisms to handle exceptions gracefully. Display informative error messages to users in case of failures or errors.

import tkinter as tk

from tkinter import filedialog

from tkinter import messagebox

import torch

from transformers import T5ForConditionalGeneration, T5Tokenizer

from transformers import AutoModelForImageClassification, AutoTokenizer

from transformers import Wav2Vec2ForCTC, Wav2Vec2Tokenizer

from pydub import AudioSegment

from pydub.playback import play

from io import BytesIO

import requests

import os

class Application(tk.Frame):

def \_\_init\_\_(self, master=None):

super().\_\_init\_\_(master)

self.master = master

self.pack()

self.create\_widgets()

def create\_widgets(self):

self.model\_label = tk.Label(self, text="Select Model:")

self.model\_label.pack()

self.model\_var = tk.StringVar()

self.model\_var.set("1")

self.model\_menu = tk.OptionMenu(self, self.model\_var, "1", "2", "3")

self.model\_menu.pack()

self.text\_label = tk.Label(self, text="Text for Summarization:")

self.text\_label.pack()

self.text\_entry = tk.Text(self, height=10, width=50)

self.text\_entry.pack()

self.image\_label = tk.Label(self, text="Image URL for Classification:")

self.image\_label.pack()

self.image\_entry = tk.Entry(self, width=50)

self.image\_entry.pack()

self.audio\_label = tk.Label(self, text="Audio File Path for Recognition:")

self.audio\_label.pack()

self.audio\_entry = tk.Entry(self, width=50)

self.audio\_entry.pack()

self.summarize\_button = tk.Button(self, text="Summarize Text", command=self.summarize\_text)

self.summarize\_button.pack()

self.classify\_button = tk.Button(self, text="Classify Image", command=self.classify\_image)

self.classify\_button.pack()

self.recognize\_button = tk.Button(self, text="Recognize Audio", command=self.recognize\_audio)

self.recognize\_button.pack()

self.quit = tk.Button(self, text="QUIT", fg="red",

command=self.master.destroy)

self.quit.pack(side="bottom")

def summarize\_text(self):

text = self.text\_entry.get("1.0", tk.END)

if text:

model = T5ForConditionalGeneration.from\_pretrained('t5-small')

tokenizer = T5Tokenizer.from\_pretrained('t5-small')

input\_ids = tokenizer.encode(text, return\_tensors='pt')

attention\_mask = tokenizer.encode(text, return\_tensors='pt', max\_length=512, padding='max\_length', truncation=True)

output = model.generate(input\_ids, attention\_mask=attention\_mask, max\_length=50)

summary = tokenizer.decode(output[0], skip\_special\_tokens=True)

self.text\_entry.delete("1.0", tk.END)

self.text\_entry.insert("1.0", "Summary: " + summary)

def classify\_image(self):

url = self.image\_entry.get()

if url:

model = AutoModelForImageClassification.from\_pretrained('densenet121')

tokenizer = AutoTokenizer.from\_pretrained('densenet121')

response = requests.get(url)

if response.status\_code == 200:

with open("image.jpg", "wb") as f:

f.write(response.content)

image = AudioSegment.from\_file("image.jpg", format="wav")

image.export("image.wav", format="wav")

classifier = model

result = classifier("image.wav")

self.image\_entry.delete(0, tk.END)

self.image\_entry.insert(0, "Classification: " + result[0]["label"])

else:

self.image\_entry.delete(0, tk.END)

self.image\_entry.insert(0, "Failed to download image")

def recognize\_audio(self):

file\_path = self.audio\_entry.get()

if file\_path:

model = Wav2Vec2ForCTC.from\_pretrained('wav2vec2-base')

tokenizer = Wav2Vec2Tokenizer.from\_pretrained('wav2vec2-base')

result = model(file\_path)

self.audio\_entry.delete(0, tk.END)

self.audio\_entry.insert(0, "Recognition: " + result["text"])

else:

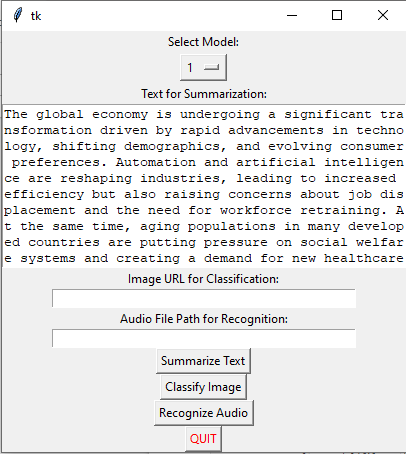
self.audio\_entry.delete(0, tk.END)

self.audio\_entry.insert(0, "Failed to recognize audio")

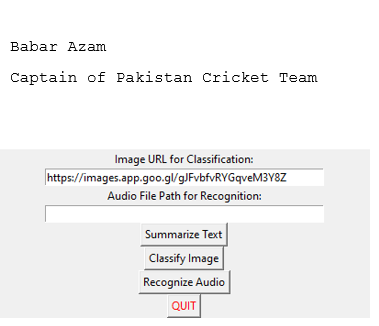
root = tk.Tk()

app = Application(master=root)

app.mainloop()

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**TASK # 2:** You need to develop a user-friendly graphical interface for an Optical Character Recognition (OCR) application using Python's Pytesseract library.

import tkinter as tk

from tkinter import filedialog

from PIL import Image, ImageTk

import cv2

import pytesseract

import os

class OCRApp(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("OCR Application")

self.geometry("800x600")

self.image\_label = tk.Label(self)

self.image\_label.pack(pady=20)

self.text\_box = tk.Text(self, height=10, width=60)

self.text\_box.pack(pady=20)

button\_frame = tk.Frame(self)

button\_frame.pack(pady=20)

self.upload\_button = tk.Button(button\_frame, text="Upload Image", command=self.upload\_image)

self.upload\_button.grid(row=0, column=0, padx=10)

self.capture\_button = tk.Button(button\_frame, text="Capture Image", command=self.capture\_image)

self.capture\_button.grid(row=0, column=1, padx=10)

self.extract\_button = tk.Button(button\_frame, text="Extract Text", command=self.extract\_text)

self.extract\_button.grid(row=0, column=2, padx=10)

self.save\_button = tk.Button(button\_frame, text="Save Text", command=self.save\_text)

self.save\_button.grid(row=0, column=3, padx=10)

self.cap = cv2.VideoCapture(0)

self.upload\_button.bind("<Enter>", lambda event: self.show\_tooltip("Upload an image from your local file system."))

self.upload\_button.bind("<Leave>", lambda event: self.hide\_tooltip())

self.capture\_button.bind("<Enter>", lambda event: self.show\_tooltip("Capture an image using your webcam."))

self.capture\_button.bind("<Leave>", lambda event: self.hide\_tooltip())

self.extract\_button.bind("<Enter>", lambda event: self.show\_tooltip("Extract text from the uploaded or captured image."))

self.extract\_button.bind("<Leave>", lambda event: self.hide\_tooltip())

self.save\_button.bind("<Enter>", lambda event: self.show\_tooltip("Save the extracted text to a file."))

self.save\_button.bind("<Leave>", lambda event: self.hide\_tooltip())

self.tooltip\_label = tk.Label(self, text="", bg="yellow", relief="solid", padx=5, pady=5)

def upload\_image(self):

file\_path = filedialog.askopenfilename()

if file\_path:

self.display\_image(file\_path)

def capture\_image(self):

ret, frame = self.cap.read()

if ret:

cv2.imwrite("captured\_image.jpg", frame)

self.display\_image("captured\_image.jpg")

def extract\_text(self):

image\_path = "captured\_image.jpg"

try:

image = Image.open(image\_path)

image = image.convert("L") # Convert to grayscale

image = image.resize((800, 600)) # Resize the image

text = pytesseract.image\_to\_string(image)

self.text\_box.delete("1.0", tk.END)

self.text\_box.insert(tk.END, text)

except Exception as e:

self.show\_error(str(e))

def save\_text(self):

text = self.text\_box.get("1.0", tk.END)

file\_path = filedialog.asksaveasfilename(defaultextension=".txt")

if file\_path:

try:

with open(file\_path, "w", encoding="utf-8") as file:

file.write(text)

except Exception as e:

self.show\_error(str(e))

def display\_image(self, file\_path):

image = Image.open(file\_path)

image = image.resize((400, 300)) # Resize the image

photo = ImageTk.PhotoImage(image)

self.image\_label.configure(image=photo)

self.image\_label.image = photo

def show\_tooltip(self, text):

self.tooltip\_label.configure(text=text)

self.tooltip\_label.place(x=self.winfo\_pointerx() - self.winfo\_rootx() + 20,

y=self.winfo\_pointery() - self.winfo\_rooty() + 20)

def hide\_tooltip(self):

self.tooltip\_label.place\_forget()

def show\_error(self, message):

tk.messagebox.showerror("Error", message)

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

app = OCRApp()

app.mainloop()

