import tkinter as tk

from tkinter import filedialog

from PIL import Image, ImageTk

import numpy as np

import cv2

from skimage import exposure

def upload\_image():

"""

Function to upload an image file.

Opens a file dialog to select an image file, resizes the image,

and displays it in the label.

"""

global img, img\_tk, img\_cv

# Open a file dialog to select an image file

file\_path = filedialog.askopenfilename(filetypes=[("Image files", "\*.jpg \*.jpeg \*.png")])

if file\_path:

# Open the image file

img = Image.open(file\_path)

# Convert the image to OpenCV format

img\_cv = cv2.cvtColor(np.array(img), cv2.COLOR\_RGB2BGR)

# Resize the image to fit the display area

img = img.resize((250, 250), Image.LANCZOS)

img\_tk = ImageTk.PhotoImage(img)

# Display the image in the label

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

image\_label.image = img\_tk # Keep a reference to avoid garbage collection

def update\_image\_display():

"""

Function to update the displayed image in the label.

"""

global img, img\_tk

img\_pil = Image.fromarray(cv2.cvtColor(img\_cv, cv2.COLOR\_BGR2RGB))

img = img\_pil.resize((250, 250), Image.LANCZOS)

img\_tk = ImageTk.PhotoImage(img)

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

image\_label.image = img\_tk # Keep a reference to avoid garbage collection

def normalize\_image():

global img\_cv

img\_cv = cv2.normalize(img\_cv, None, 0, 255, cv2.NORM\_MINMAX)

update\_image\_display()

def noise\_reduction():

global img\_cv

img\_cv = cv2.fastNlMeansDenoisingColored(img\_cv, None, 10, 10, 7, 21)

update\_image\_display()

def skull\_stripping():

global img\_cv

gray = cv2.cvtColor(img\_cv, cv2.COLOR\_BGR2GRAY)

blurred = cv2.GaussianBlur(gray, (5, 5), 0)

thresh = cv2.threshold(blurred, 45, 255, cv2.THRESH\_BINARY)[1]

img\_cv = cv2.bitwise\_and(img\_cv, img\_cv, mask=thresh)

update\_image\_display()

def artifact\_removal():

global img\_cv

img\_cv = cv2.medianBlur(img\_cv, 5)

update\_image\_display()

def rotation():

global img\_cv

(h, w) = img\_cv.shape[:2]

center = (w // 2, h // 2)

M = cv2.getRotationMatrix2D(center, 45, 1.0)

img\_cv = cv2.warpAffine(img\_cv, M, (w, h))

update\_image\_display()

def translation():

global img\_cv

(h, w) = img\_cv.shape[:2]

M = np.float32([[1, 0, 25], [0, 1, 25]])

img\_cv = cv2.warpAffine(img\_cv, M, (w, h))

update\_image\_display()

def scaling():

global img\_cv

img\_cv = cv2.resize(img\_cv, None, fx=1.5, fy=1.5, interpolation=cv2.INTER\_LINEAR)

update\_image\_display()

def flipping():

global img\_cv

img\_cv = cv2.flip(img\_cv, 1)

update\_image\_display()

def elastic\_deformation():

pass

def intensity\_adjustment():

global img\_cv

img\_cv = exposure.adjust\_gamma(img\_cv, gamma=0.4, gain=0.9)

update\_image\_display()

def noise\_injection():

global img\_cv

row, col, ch = img\_cv.shape

mean = 0

var = 0.1

sigma = var \*\* 0.5

gauss = np.random.normal(mean, sigma, (row, col, ch))

gauss = gauss.reshape(row, col, ch)

img\_cv = img\_cv + gauss

update\_image\_display()

def shearing():

global img\_cv

M = np.float32([[1, 0.5, 0], [0.5, 1, 0]])

img\_cv = cv2.warpAffine(img\_cv, M, (img\_cv.shape[1], img\_cv.shape[0]))

update\_image\_display()

def random\_cropping():

global img\_cv

h, w = img\_cv.shape[:2]

x = np.random.randint(0, w // 2)

y = np.random.randint(0, h // 2)

img\_cv = img\_cv[y:y + h // 2, x:x + w // 2]

update\_image\_display()

# Create the main window

root = tk.Tk()

root.title("NeuroScan: Advanced Brain Tumor Detection System")

root.geometry("1000x800")

# Create a heading label

heading = tk.Label(root, text="NeuroScan: Advanced Brain Tumor Detection System", font=("Arial", 24, "bold"))

heading.pack(pady=20)

# Create a frame for image uploading and display

upload\_frame = tk.Frame(root)

upload\_frame.pack(pady=20, fill=tk.BOTH, expand=True)

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

upload\_button.pack(pady=10)

image\_label = tk.Label(upload\_frame)

image\_label.pack()

# Create a frame for image processing

image\_processing\_frame = tk.LabelFrame(root, text="Image Processing", font=("Arial", 14, "bold"), padx=10, pady=10)

image\_processing\_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=20, pady=10)

# Create buttons for image processing

normalize\_button = tk.Button(image\_processing\_frame, text="Normalize", command=normalize\_image, width=15)

normalize\_button.grid(row=0, column=0, padx=10, pady=10)

noise\_reduction\_button = tk.Button(image\_processing\_frame, text="Noise Reduction", command=noise\_reduction, width=15)

noise\_reduction\_button.grid(row=0, column=1, padx=10, pady=10)

skull\_stripping\_button = tk.Button(image\_processing\_frame, text="Skull Stripping", command=skull\_stripping, width=15)

skull\_stripping\_button.grid(row=1, column=0, padx=10, pady=10)

artifact\_removal\_button = tk.Button(image\_processing\_frame, text="Artifact Removal", command=artifact\_removal, width=15)

artifact\_removal\_button.grid(row=1, column=1, padx=10, pady=10)

# Create a frame for data augmentation

data\_augmentation\_frame = tk.LabelFrame(root, text="Data Augmentation", font=("Arial", 14, "bold"), padx=10, pady=10)

data\_augmentation\_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=20, pady=10)

# Create buttons for data augmentation

rotation\_button = tk.Button(data\_augmentation\_frame, text="Rotation", command=rotation, width=15)

rotation\_button.grid(row=0, column=0, padx=10, pady=10)

translation\_button = tk.Button(data\_augmentation\_frame, text="Translation", command=translation, width=15)

translation\_button.grid(row=0, column=1, padx=10, pady=10)

scaling\_button = tk.Button(data\_augmentation\_frame, text="Scaling", command=scaling, width=15)

scaling\_button.grid(row=1, column=0, padx=10, pady=10)

flipping\_button = tk.Button(data\_augmentation\_frame, text="Flipping", command=flipping, width=15)

flipping\_button.grid(row=1, column=1, padx=10, pady=10)

elastic\_deformation\_button = tk.Button(data\_augmentation\_frame, text="Elastic Deformation", command=elastic\_deformation, width=15)

elastic\_deformation\_button.grid(row=2, column=0, padx=10, pady=10)

intensity\_adjustment\_button = tk.Button(data\_augmentation\_frame, text="Intensity Adjustment", command=intensity\_adjustment, width=15)

intensity\_adjustment\_button.grid(row=2, column=1, padx=10, pady=10)

noise\_injection\_button = tk.Button(data\_augmentation\_frame, text="Noise Injection", command=noise\_injection, width=15)

noise\_injection\_button.grid(row=3, column=0, padx=10, pady=10)

shearing\_button = tk.Button(data\_augmentation\_frame, text="Shearing", command=shearing, width=15)

shearing\_button.grid(row=3, column=1, padx=10, pady=10)

random\_cropping\_button = tk.Button(data\_augmentation\_frame, text="Random Cropping", command=random\_cropping, width=15)

random\_cropping\_button.grid(row=4, column=0, padx=10, pady=10)

# Configure column weights for responsive design

for frame in [image\_processing\_frame, data\_augmentation\_frame]:

for col in range(2):

frame.grid\_columnconfigure(col, weight=1)

# Run the application

root.mainloop()

Certainly! Let's break down the code for the login page written in Python using Tkinter and PyMySQL. This code creates a graphical login interface and handles user authentication.

### Code Explanation

#### Imports

```python

from tkinter import \*

from PIL import Image, ImageTk

from tkinter import ttk, messagebox

import pymysql

import credentials as cr

from BC220422733\_CS619\_Prototype\_GUI import main\_gui # Import the main GUI function

```

- `tkinter` is imported for GUI creation.

- `PIL` (Pillow) is used to handle images.

- `pymysql` is used for connecting to a MySQL database.

- `credentials` is a module (not shown here) where database credentials are stored.

- `main\_gui` is imported from `BC220422733\_CS619\_Prototype\_GUI`, presumably the main application interface.

#### `Login` Class Definition

```python

class Login:

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

self.window = root

self.window.iconbitmap('Images/apple.ico')

self.window.title("Login")

self.window.geometry("1280x800+0+0")

self.window.config(bg="white")

```

- \*\*Constructor `\_\_init\_\_`\*\*: Initializes the `Login` class with the root window.

- Sets the window's icon, title, size, and background color.

#### Background and Frame

```python

self.bg\_img = ImageTk.PhotoImage(file="Images/hospital\_bg.jpeg")

background = Label(self.window, image=self.bg\_img).place(x=0, y=0, relwidth=1, relheight=1)

frame = Frame(self.window, bg="white")

frame.place(x=350, y=100, width=500, height=550)

```

- Loads and displays a background image.

- Creates a frame for the login form and places it on the window.

#### Title and Form Elements

```python

title1 = Label(frame, text="Login", font=("times new roman", 25, "bold"), bg="white").place(x=20, y=10)

title2 = Label(frame, text="Welcome Back", font=("times new roman", 13), bg="white", fg="gray").place(x=20, y=50)

email = Label(frame, text="Email", font=("helvetica", 15, "bold"), bg="white").place(x=20, y=100)

self.email\_txt = Entry(frame, font=("arial"))

self.email\_txt.place(x=20, y=130, width=420)

password = Label(frame, text="Password", font=("helvetica", 15, "bold"), bg="white").place(x=20, y=180)

self.password\_txt = Entry(frame, font=("arial"), show="\*")

self.password\_txt.place(x=20, y=210, width=420)

self.login\_btn = Button(frame, text="Login", command=self.login\_func, font=("times new roman", 18, "bold"), bd=0, cursor="hand2", bg="green2", fg="white").place(x=120, y=260, width=250)

```

- \*\*Labels\*\* and \*\*Entries\*\*: Used to create the login form with fields for email and password.

- \*\*Button\*\*: Triggers the `login\_func` method when clicked.

#### `login\_func` Method

```python

def login\_func(self):

if self.email\_txt.get() == "" or self.password\_txt.get() == "":

messagebox.showerror("Error!", "All fields are required", parent=self.window)

else:

try:

connection = pymysql.connect(host=cr.host, user=cr.user, password=cr.password, database=cr.database)

cur = connection.cursor()

cur.execute("select \* from student\_register where email=%s and password=%s", (self.email\_txt.get(), self.password\_txt.get()))

row = cur.fetchone()

if row is None:

messagebox.showerror("Error!", "Invalid email or password", parent=self.window)

else:

messagebox.showinfo("Success!", "Login successful", parent=self.window)

self.window.destroy()

root = Tk()

main\_gui(root) # Open the main GUI application

root.mainloop()

except Exception as es:

messagebox.showerror("Error!", f"Error due to {es}", parent=self.window)

```

- \*\*Validation\*\*: Checks if both email and password fields are filled.

- \*\*Database Connection\*\*: Connects to a MySQL database using credentials.

- \*\*Query Execution\*\*: Executes a query to check if the email and password match any record.

- \*\*Response Handling\*\*: Shows appropriate message boxes based on login success or failure.

- \*\*Application Transition\*\*: On successful login, it closes the login window and opens the main GUI.

#### Main Execution

```python

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

root = Tk()

obj = Login(root)

root.mainloop()

```

- \*\*Entry Point\*\*: Initializes and runs the Tkinter main loop to display the login window.

### Summary

This script sets up a Tkinter GUI for user login, validates the user's credentials against a MySQL database, and transitions to the main application interface upon successful login. It includes error handling and user feedback through message boxes.