PROJECT CODE

#(app.py)

```
from flask import Flask, render_template, request, redirect, url_for, flash, session

from flask_sqlalchemy import SQLAlchemy

from werkzeug.security import generate_password_hash, check_password_hash

import os


app = Flask(__name__)

app.secret_key = 'your_secret_key'

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'

app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False


db = SQLAlchemy(app)


class User(db.Model):

    id = db.Column(db.Integer, primary_key=True)

    username = db.Column(db.String(150), unique=True, nullable=False)
```

```
    password = db.Column(db.String(150), nullable=False)


@app.route('/')
def home():
    return render_template('home.html')


@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']


        if User.query.filter_by(username=username).first():
            flash('Username already exists.', 'danger')
            return redirect(url_for('register'))


        hashed_password = generate_password_hash(password,
method='pbkdf2:sha256')


        new_user = User(username=username,
password=hashed_password)
        db.session.add(new_user)
```

```
        db.session.commit()

        flash('Registered successfully! Please login.', 'success')
        return redirect(url_for('login'))

    return render_template('register.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        user = User.query.filter_by(username=username).first()

        if user and check_password_hash(user.password,
password):
            session['user_id'] = user.id
            flash('Login successful!', 'success')
            return redirect(url_for('dashboard'))
        else:
            flash('Invalid credentials. Please try again.', 'danger')
```

```
    return render_template('login.html')

@app.route('/dashboard')
def dashboard():
    if 'user_id' not in session:
        flash('Please login first.', 'warning')
        return redirect(url_for('login'))

    return render_template('dashboard.html')

@app.route('/run_final')
def run_final():
    if 'user_id' not in session:
        flash('Please login first.', 'warning')
        return redirect(url_for('login'))

    os.system('python final.py')
    flash('final.py executed successfully.', 'success')
    return redirect(url_for('dashboard'))

@app.route('/run_main')
```

```
def run_main():
    if 'user_id' not in session:
        flash('Please login first.', 'warning')
        return redirect(url_for('login'))

    os.system('python main.py')
    flash('main.py executed successfully.', 'success')
    return redirect(url_for('dashboard'))

@app.route('/logout')
def logout():
    session.pop('user_id', None)
    flash('Logged out successfully.', 'success')
    return redirect(url_for('home'))

if __name__ == '__main__':
    with app.app_context():
        db.create_all()
    app.run(debug=True)
```

#(main.py)

```
import numpy as np
import cv2
import PySimpleGUI as sg
import os.path

version = '7 June 2020'

prototxt = r'model/colorization_deploy_v2.prototxt'
model = r'model/colorization_release_v2.caffemodel'
points = r'model/pts_in_hull.npy'
points = os.path.join(os.path.dirname(__file__), points)
prototxt = os.path.join(os.path.dirname(__file__), prototxt)
model = os.path.join(os.path.dirname(__file__), model)
if not os.path.isfile(model):
    sg.popup_scrolled('Missing model file', 'You are missing the file "colorization_release_v2.caffemodel"',
                      'Download it and place into your "model" folder', 'You can download this file from this location:\n', r'https://www.dropbox.com/s/dx0qvhhp5hbcx7z/colorization_release_v2.caffemodel?dl=1')
    exit()
```

```
net = cv2.dnn.readNetFromCaffe(prototxt, model)     # load model from disk
pts = np.load(points)


# add the cluster centers as 1x1 convolutions to the model
class8 = net.getLayerId("class8_ab")
conv8 = net.getLayerId("conv8_313_rh")
pts = pts.transpose().reshape(2, 313, 1, 1)
net.getLayer(class8).blobs = [pts.astype("float32")]
net.getLayer(conv8).blobs = [np.full([1, 313], 2.606, dtype="float32")]


def colorize_image(image_filename=None, cv2_frame=None):
    """
    Where all the magic happens.  Colorizes the image provided. Can colorize either
    a filename OR a cv2 frame (read from a web cam most likely)
    :param image_filename: (str) full filename to colorize
    :param cv2_frame: (cv2 frame)
    :return: Tuple[cv2 frame, cv2 frame] both non-colorized and colorized images in cv2 format as a tuple
```

```
    """
    # load the input image from disk, scale the pixel intensities to the range [0, 1], and then convert the image from the BGR to Lab color space
    image = cv2.imread(image_filename) if image_filename else cv2_frame
    scaled = image.astype("float32") / 255.0
    lab = cv2.cvtColor(scaled, cv2.COLOR_BGR2LAB)

    # resize the Lab image to 224x224 (the dimensions the colorization network accepts), split channels, extract the 'L' channel, and then perform mean centering
    resized = cv2.resize(lab, (224, 224))
    L = cv2.split(resized)[0]
    L -= 50

    # pass the L channel through the network which will *predict* the 'a' and 'b' channel values
    'print("[INFO] colorizing image...")'
    net.setInput(cv2.dnn.blobFromImage(L))
    ab = net.forward()[0, :, :, :].transpose((1, 2, 0))
```

```
    # resize the predicted 'ab' volume to the same dimensions as
our input image
    ab = cv2.resize(ab, (image.shape[1], image.shape[0]))

    # grab the 'L' channel from the *original* input image (not
the resized one) and concatenate the original 'L' channel with
the predicted 'ab' channels
    L = cv2.split(lab)[0]
    colorized = np.concatenate((L[:, :, np.newaxis], ab), axis=2)

    # convert the output image from the Lab color space to
RGB, then clip any values that fall outside the range [0, 1]
    colorized = cv2.cvtColor(colorized,
cv2.COLOR_LAB2BGR)
    colorized = np.clip(colorized, 0, 1)

    # the current colorized image is represented as a floating
point data type in the range [0, 1] -- let's convert to an
unsigned 8-bit integer representation in the range [0, 255]
    colorized = (255 * colorized).astype("uint8")
    return image, colorized

def convert_to_grayscale(frame):
```

```
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  # Convert webcam frame to grayscale
    gray_3_channels = np.zeros_like(frame)  # Convert grayscale frame (single channel) to 3 channels
    gray_3_channels[:, :, 0] = gray
    gray_3_channels[:, :, 1] = gray
    gray_3_channels[:, :, 2] = gray
    return gray_3_channels


# ---------------------------------- The GUI ----------------------------------


# First the window layout...2 columns


left_col = [[sg.Text('Folder'), sg.In(size=(25,1), enable_events=True ,key='-FOLDER-'), sg.FolderBrowse()],
            [sg.Listbox(values=[], enable_events=True, size=(40,20),key='-FILE LIST-')],
            [sg.CBox('Convert to gray first',key='-MAKEGRAY-')],
            [sg.Text('Version ' + version, font='Courier 8')]]
```

```
images_col = [[sg.Text('Input file:'),
sg.In(enable_events=True, key='-IN FILE-'),
sg.FileBrowse()],
          [sg.Button('Colorize Photo', key='-PHOTO-'),
sg.Button('Start Webcam', key='-WEBCAM-'),
sg.Button('Save File', key='-SAVE-'), sg.Button('Exit')],
          [sg.Image(filename='', key='-IN-'),
sg.Image(filename='', key='-OUT-')],]

# ----- Full layout -----

layout = [[sg.Column(left_col), sg.VSeperator(),
sg.Column(images_col)]]


# ----- Make the window -----

window = sg.Window('Photo Colorizer', layout,
grab_anywhere=True)


# ----- Run the Event Loop -----

prev_filename = colorized = cap = None

while True:
    event, values = window.read()
    if event in (None, 'Exit'):
        break
    if event == '-FOLDER-':         # Folder name was filled in,
make a list of files in the folder
```

```
        folder = values['-FOLDER-']
        img_types = (".png", ".jpg", "jpeg", ".tiff", ".bmp")
        # get list of files in folder
        try:
            flist0 = os.listdir(folder)
        except:
            continue
        fnames = [f for f in flist0 if os.path.isfile(
            os.path.join(folder, f)) and
f.lower().endswith(img_types)]
        window['-FILE LIST-'].update(fnames)
    elif event == '-FILE LIST-':    # A file was chosen from the
listbox
        try:
            filename = os.path.join(values['-FOLDER-'], values['-
FILE LIST-'][0])
            image = cv2.imread(filename)
            window['-IN-'].update(data=cv2.imencode('.png',
image)[1].tobytes())
            window['-OUT-'].update(data='')
            window['-IN FILE-'].update('')


            if values['-MAKEGRAY-']:
```

```
                gray_3_channels = convert_to_grayscale(image)
                window['-IN-'].update(data=cv2.imencode('.png',
gray_3_channels)[1].tobytes())
                image, colorized =
colorize_image(cv2_frame=gray_3_channels)
            else:
                image, colorized = colorize_image(filename)


            window['-OUT-'].update(data=cv2.imencode('.png',
colorized)[1].tobytes())
        except:
            continue
    elif event == '-PHOTO-':        # Colorize photo button
clicked
        try:
            if values['-IN FILE-']:
                filename = values['-IN FILE-']
            elif values['-FILE LIST-']:
                filename = os.path.join(values['-FOLDER-'],
values['-FILE LIST-'][0])
            else:
                continue
            if values['-MAKEGRAY-']:
```

```
            gray_3_channels =
convert_to_grayscale(cv2.imread(filename))
            window['-IN-'].update(data=cv2.imencode('.png',
gray_3_channels)[1].tobytes())
            image, colorized =
colorize_image(cv2_frame=gray_3_channels)
        else:
            image, colorized = colorize_image(filename)
            window['-IN-'].update(data=cv2.imencode('.png',
image)[1].tobytes())
        window['-OUT-'].update(data=cv2.imencode('.png',
colorized)[1].tobytes())
    except:
        continue
elif event == '-IN FILE-':      # A single filename was
chosen
    filename = values['-IN FILE-']
    if filename != prev_filename:
        prev_filename = filename
        try:
            image = cv2.imread(filename)
            window['-IN-'].update(data=cv2.imencode('.png',
image)[1].tobytes())
```

```
            except:
                continue
        elif event == '-WEBCAM-':       # Webcam button clicked
            sg.popup_quick_message('Starting up your Webcam... this takes a moment....',
auto_close_duration=1,  background_color='red', text_color='white', font='Any 16')
            window['-WEBCAM-'].update('Stop Webcam', button_color=('white','red'))
            cap = cv2.VideoCapture(0) if not cap else cap
            while True:                 # Loop that reads and shows webcam until stop button
                ret, frame = cap.read()     # Read a webcam frame
                gray_3_channels = convert_to_grayscale(frame)
                image, colorized = colorize_image(cv2_frame=gray_3_channels)    # Colorize the 3-channel grayscale frame
                window['-IN-'].update(data=cv2.imencode('.png', gray_3_channels)[1].tobytes())
                window['-OUT-'].update(data=cv2.imencode('.png', colorized)[1].tobytes())
                event, values = window.read(timeout=0)  # Update the window outputs and check for new events
```

```
        if event in (None, '-WEBCAM-', 'Exit'): # Clicked the Stop Webcam button or closed window entirely
            window['-WEBCAM-'].update('Start Webcam', button_color=sg.theme_button_color())
            window['-IN-'].update('')
            window['-OUT-'].update('')
            break
    elif event == '-SAVE-' and colorized is not None:   # Clicked the Save File button
        filename = sg.popup_get_file('Save colorized image.\nColorized image be saved in format matching the extension you enter.', save_as=True)
        try:
            if filename:
                cv2.imwrite(filename, colorized)
                sg.popup_quick_message('Image save complete', background_color='red', text_color='white', font='Any 16')
        except:
            sg.popup_quick_message('ERROR - Image NOT saved!', background_color='red', text_color='white', font='Any 16')
# ----- Exit program -----
window.close()
```