Traffic Sign Recognition system

CH B T G V PRASAD

COMPUTER VISION

Introduction:

This Traffic Sign Recognition (TSR) project showcases a robust system designed to detect and classify traffic signs in real-time, employing a combination of Python, OpenCV, and machine learning methodologies. Leveraging a webcam or processing images from a specified folder, the system captures traffic scenes and preprocesses them to ensure standardized analysis. Through a trained machine learning model, stored using pickle, the system predicts the class of each detected sign, drawing from a diverse dataset encompassing various traffic sign types. Utilizing OpenCV, the system overlays the predicted class and associated probability onto the original image, enabling seamless visual feedback. Furthermore, for enhanced user interaction, a text-to-speech engine (Pyttsx3) audibly announces the class of the detected sign, underscoring its practical utility in real-world scenarios like driver assistance systems and traffic management.

This project underscores the significance of TSR systems in enhancing road safety and facilitating efficient traffic management. By leveraging state-of-the-art technologies and machine learning algorithms, the system demonstrates its efficacy in real-time traffic sign detection and classification tasks. Its versatility, whether in webcam-based real-time monitoring or batch processing of images from designated folders, underscores its adaptability to diverse application scenarios. With seamless integration of visual and auditory feedback mechanisms, the project underscores a user-centric approach to traffic sign recognition, poised to contribute significantly to advancements in driver assistance systems and intelligent transportation systems.

Abstract:

The Traffic Sign Recognition (TSR) project presents a comprehensive system for real-time detection and classification of traffic signs, crucial for enhancing road safety and traffic management. Leveraging Python, OpenCV, and machine learning techniques, the system seamlessly processes input from either a webcam or designated image folders. Through meticulous preprocessing, the captured images are standardized for analysis. A trained machine learning model, stored using pickle, enables precise prediction of traffic sign classes, drawing from an extensive dataset encompassing various sign types. OpenCV facilitates the overlay of predicted class labels and associated probabilities onto the original images, providing immediate visual feedback. Moreover, integrating a text-to-speech engine (Pyttsx3) ensures auditory announcements of detected sign classes, enhancing user interaction. This project underscores the system's versatility, applicability to real-world scenarios, and potential contributions to driver assistance systems and intelligent transportation systems, emphasizing its pivotal role in promoting road safety and efficient traffic management.

Objective:

The objective of the provided code appears to be implementing a traffic sign recognition system using a pre-trained model. This system allows users to either use a webcam to capture real-time video for traffic sign recognition or select a folder containing images for batch processing.

Here's a breakdown of the main objectives:

1. **Load Pre-trained Model**: The code loads a pre-trained model from a pickle file. This model is likely trained on a dataset of traffic sign images and can classify them into different categories representing various traffic signs.
2. **Real-time Recognition (Webcam)**: The `process_webcam()` function continuously captures video from the webcam, preprocesses each frame, and then uses the pre-trained model to make predictions on each frame. Predicted class names and probabilities are overlaid on the video stream, and the result is displayed in a window.
3. **Batch Processing (Folder)**: The `process_folder()` function allows users to select a folder containing images. It then processes each image in the folder similarly to the webcam process, overlaying predicted class names and probabilities on each image. Additionally, the system utilizes pyttsx3 to audibly announce the recognized traffic sign for each image.
4. **User Interface (GUI)**: The code provides a simple tkinter-based GUI with buttons for selecting between using the webcam or processing images from a folder. This GUI makes the system user-friendly and accessible.

Problem Statement:

Problem Statement: Traffic Sign Recognition System

Description: Develop a Traffic Sign Recognition System (TSRS) to enhance road safety and navigation for autonomous vehicles or driver assistance systems. The TSRS should be capable of accurately detecting, classifying, and interpreting traffic signs in real-time, providing crucial information to vehicle control systems.

Key Objectives:

1. **Detection:** Implement algorithms to detect traffic signs within images or video streams captured by vehicle-mounted cameras. The detection process should accurately identify the location and boundaries of traffic signs in varying environmental conditions.
2. **Classification:** Develop a classifier capable of categorizing detected traffic signs into predefined classes representing different types of signs (e.g., speed limits, stop signs, yield signs). The classification model should achieve high accuracy and reliability in recognizing various sign types.
3. **Interpretation:** Provide semantic interpretation of recognized traffic signs, conveying their meanings, regulations, or warnings to vehicle control systems. This includes associating recognized signs with appropriate textual descriptions or symbolic representations for driver awareness and decision-making.
4. **Real-time Performance:** Ensure that the TSRS operates with minimal latency and high throughput, enabling timely response to detected traffic signs in dynamic driving scenarios. Real-time performance is critical for supporting autonomous vehicle control systems and driver assistance features.
5. **Integration:** Integrate the TSRS seamlessly into existing vehicle platforms or driver assistance systems, allowing it to interface with other onboard sensors and control modules. The TSRS should provide actionable information to enhance vehicle navigation, adapt to changing traffic conditions, and improve overall road safety.

6. **Accuracy and Robustness:** Evaluate the TSRS's accuracy and robustness under diverse environmental conditions, including variations in lighting, weather, signage placement, and occlusions. Robust performance ensures reliable operation across different driving scenarios and geographic locations.
7. **User Interface (Optional):** Optionally, develop a user interface to visualize the TSRS's outputs, including detected traffic signs, classification results, and interpreted meanings. The user interface may provide real-time feedback to drivers or autonomous vehicle operators, enhancing situational awareness and trust in the system.

Deliverables:

- Software Implementation: Develop the TSRS software, including algorithms for detection, classification, and interpretation of traffic signs.
- Documentation: Provide comprehensive documentation detailing the TSRS architecture, algorithms, implementation details, and usage instructions.
- Evaluation Report: Conduct thorough testing and evaluation of the TSRS's performance, presenting results on detection accuracy, classification accuracy, real-time performance, and robustness.
- Integration Support: Assist in the integration of the TSRS into vehicle platforms or driver assistance systems, providing technical support and guidance as needed.

Potential Extensions:

- Advanced Sign Recognition: Explore advanced techniques, such as deep learning-based approaches, for improved traffic sign recognition performance.
- Multi-sensor Fusion: Investigate methods for integrating data from multiple sensors (e.g., cameras, LiDAR, radar) to enhance traffic sign detection and interpretation.
- Adaptive Learning: Implement mechanisms for adaptive learning and continuous improvement of the TSRS's performance over time, incorporating feedback from real-world driving scenarios.
- Regulatory Compliance: Ensure that the TSRS complies with relevant traffic regulations and standards, facilitating its adoption in autonomous vehicles and commercial automotive applications.

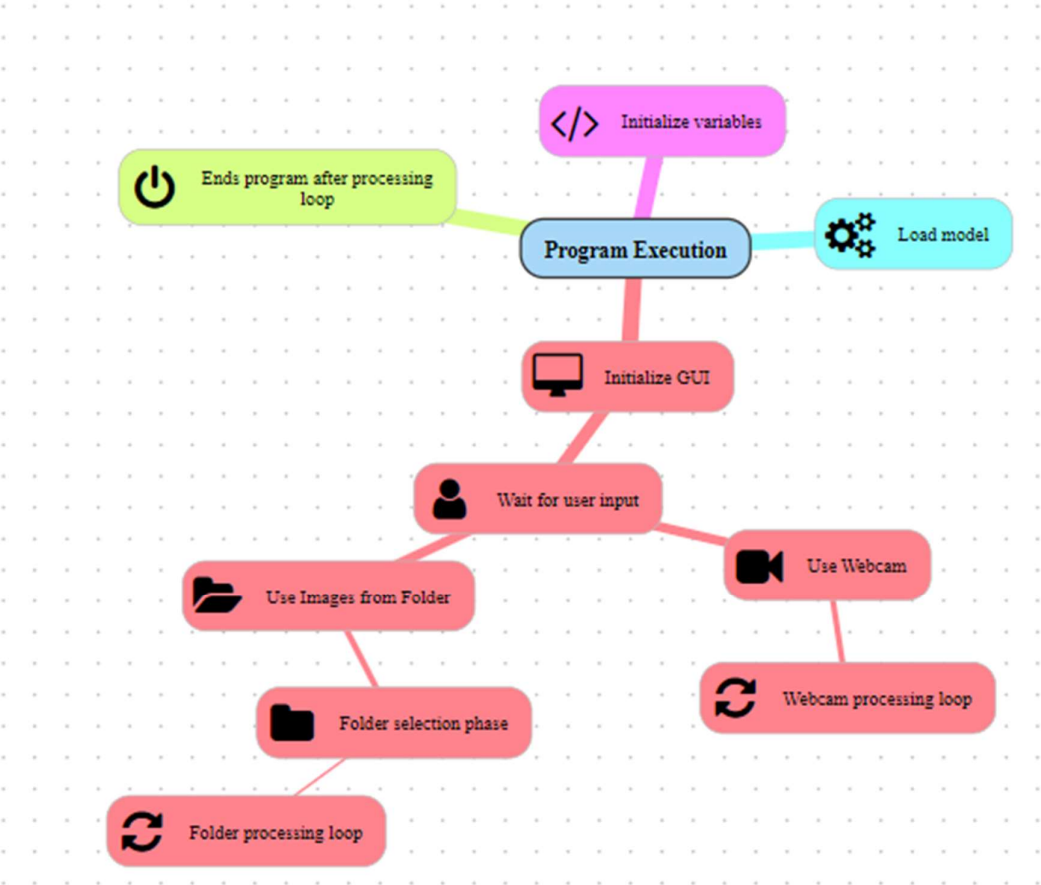
traffic sign recognition system using OpenCV for image processing, TensorFlow (assuming the `model` was trained with it) for prediction, and Tkinter for the GUI. Below is a breakdown of the key components and the overall architecture:

Key Components:

1. **Model Loading:**
   - The trained model is loaded using `pickle`.
2. **Preprocessing Function:**
   - Resizes images to 32x32 pixels.
   - Converts images to grayscale.
   - Normalizes pixel values.
3. **Class Name Function:**
   - Returns the class name based on the class index.
4. **Webcam Processing Function:**

- Captures video from the webcam.
- Processes each frame to predict the traffic sign.
- Displays the class and probability on the frame.

5. **Folder Processing Function:**
   - Allows the user to select a folder of images.
   - Processes each image to predict the traffic sign.
   - Uses text-to-speech to announce the detected sign.
6. **GUI with Tkinter:**
   - Provides options to use the webcam or process images from a folder.

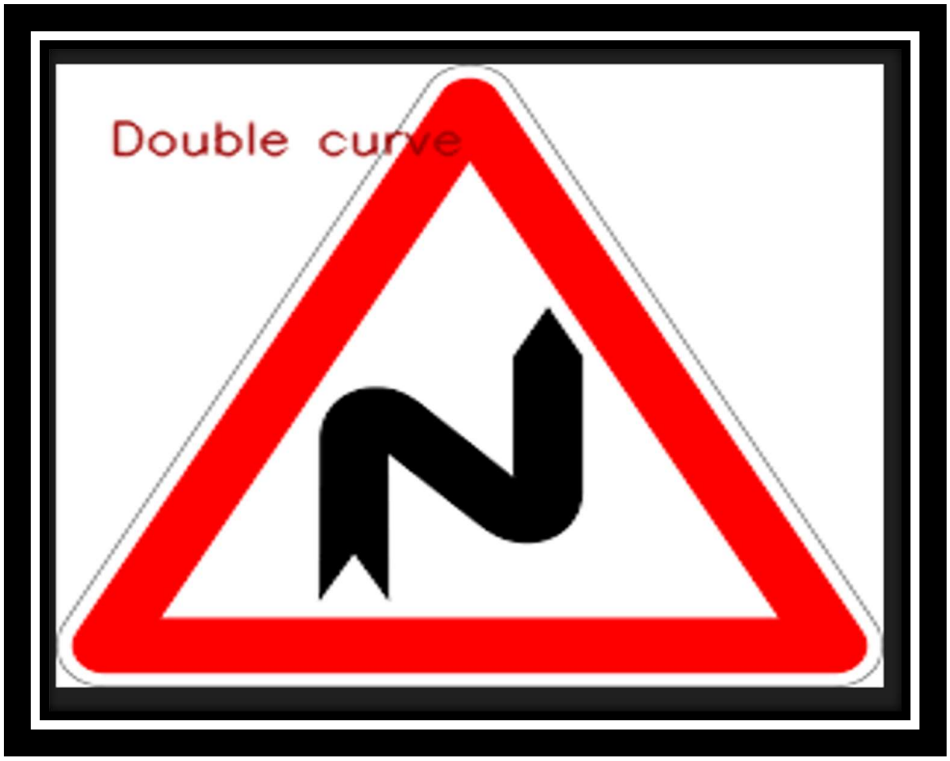


Process:

- **Start:** The beginning of the flowchart.
- **Initialize Variables and Load Model:** Set configurations and load the trained model.
- **GUI Initialization:** Create the Tkinter window with buttons for user options.
- **User Input:** Decision point where the user chooses between the webcam or folder processing.

- **Webcam Processing Loop:** Continuous processing of webcam frames until the user decides to quit.
- **Folder Selection:** Allow the user to select a folder of images.
- **Folder Processing Loop:** Process each image in the selected folder.
- **End:** The end of the flowchart indicating the completion of the processes.

Output_images:



Double curve

Children crossing

Slippery road

Code : (for data tranning)

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D,
MaxPooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import cv2
from sklearn.model_selection import train_test_split
import pandas as pd
import os
import random
import pickle


path = r"myData"
labelFile = r'labels/labels.csv'
batch_size_val = 50

epochs_val = 50
imageDimesions = (32, 32, 3)
testRatio = 0.2
validationRatio = 0.2

count = 0
images = []
classNo = []
myList = os.listdir(path)
print("Total Classes Detected:", len(myList))
noOfClasses = len(myList)
print("Importing Classes.....")
for x in range(0, len(myList)):
    myPicList = os.listdir(path + "/" + str(count))
    for y in myPicList:
        curImg = cv2.imread(path + "/" + str(count) + "/" + y)
        if curImg is not None:
            curImg = cv2.resize(curImg, (imageDimesions[1],
imageDimesions[0]))  # Resize the image
            images.append(curImg)
            classNo.append(count)
        else:
            print(f"Warning: Image {path + '/' + str(count) + '/' + y}
could not be read.")
    print(count, end=" ")
    count += 1
print(" ")
images = np.array(images)
classNo = np.array(classNo)


X_train, X_test, y_train, y_test = train_test_split(images, classNo,
test_size=testRatio)
X_train, X_validation, y_train, y_validation = train_test_split(X_train,
y_train, test_size=validationRatio)
```

```
print("Data Shapes")
print("Train", end="")
print(X_train.shape, y_train.shape)
print("Validation", end="")
print(X_validation.shape, y_validation.shape)
print("Test", end="")
print(X_test.shape, y_test.shape)
assert (X_train.shape[0] == y_train.shape[0])
assert (X_validation.shape[0] == y_validation.shape[0])
assert (X_test.shape[0] == y_test.shape[0])
assert (X_train.shape[1:] == imageDimesions)
assert (X_validation.shape[1:] == imageDimesions)
assert (X_test.shape[1:] == imageDimesions)

data = pd.read_csv(labelFile)
print("data shape ", data.shape, type(data))


num_of_samples = []
cols = 5
num_classes = noOfClasses
fig, axs = plt.subplots(nrows=num_classes, ncols=cols, figsize=(5, 300))
fig.tight_layout()
for i in range(cols):
    for j, row in data.iterrows():
        x_selected = X_train[y_train == j]
        axs[j][i].imshow(x_selected[random.randint(0, len(x_selected) - 1),
:, :], cmap=plt.get_cmap("gray"))
        axs[j][i].axis("off")
        if i == 2:
            axs[j][i].set_title(str(j) + "-" + row["Name"])
            num_of_samples.append(len(x_selected))

print(num_of_samples)
plt.figure(figsize=(12, 4))
plt.bar(range(0, num_classes), num_of_samples)
plt.title("Distribution of the training dataset")
plt.xlabel("Class number")
plt.ylabel("Number of images")
plt.show()



def grayscale(img):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return img


def equalize(img):
    img = cv2.equalizeHist(img)
    return img


def preprocessing(img):
    img = grayscale(img)
    img = equalize(img)
    img = img / 255
    return img
```

```
X_train = np.array(list(map(preprocessing, X_train)))
X_validation = np.array(list(map(preprocessing, X_validation)))
X_test = np.array(list(map(preprocessing, X_test)))
cv2.imshow("GrayScale Images", X_train[random.randint(0, len(X_train) -
1)])

X_train = X_train.reshape(X_train.shape[0], X_train.shape[1],
X_train.shape[2], 1)
X_validation = X_validation.reshape(X_validation.shape[0],
X_validation.shape[1], X_validation.shape[2], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2],
1)

dataGen = ImageDataGenerator(width_shift_range=0.1,
                             height_shift_range=0.1,
                             zoom_range=0.2,
                             shear_range=0.1,
                             rotation_range=10)
dataGen.fit(X_train)
batches = dataGen.flow(X_train, y_train, batch_size=20)
X_batch, y_batch = next(batches)

# TO SHOW AUGMENTED IMAGE SAMPLES
fig, axs = plt.subplots(1, 15, figsize=(20, 5))
fig.tight_layout()
for i in range(15):
    axs[i].imshow(X_batch[i].reshape(imageDimesions[0], imageDimesions[1]),
cmap='gray')
    axs[i].axis('off')
plt.show()

y_train = to_categorical(y_train, noOfClasses)
y_validation = to_categorical(y_validation, noOfClasses)
y_test = to_categorical(y_test, noOfClasses)


def myModel():
    no_Of_Filters = 60
    size_of_Filter = (5, 5)
    size_of_Filter2 = (3, 3)
    size_of_pool = (2, 2)
    no_Of_Nodes = 500
    model = Sequential()
    model.add(Conv2D(no_Of_Filters, size_of_Filter,
input_shape=(imageDimesions[0], imageDimesions[1], 1),
                     activation='relu'))
    model.add(Conv2D(no_Of_Filters, size_of_Filter, activation='relu'))
    model.add(MaxPooling2D(pool_size=size_of_pool))

    model.add(Conv2D(no_Of_Filters // 2, size_of_Filter2,
activation='relu'))
    model.add(Conv2D(no_Of_Filters // 2, size_of_Filter2,
activation='relu'))
    model.add(MaxPooling2D(pool_size=size_of_pool))
    model.add(Dropout(0.5))

    model.add(Flatten())
    model.add(Dense(no_Of_Nodes, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(noOfClasses, activation='softmax'))
    model.compile(Adam(lr=0.001), loss='categorical_crossentropy',
```

```
metrics=['accuracy'])
    return model



model = myModel()
print(model.summary())
steps_per_epoch_val = len(X_train) // batch_size_val  # Adjust
steps_per_epoch
history = model.fit(dataGen.flow(X_train, y_train,
batch_size=batch_size_val),
                    steps_per_epoch=steps_per_epoch_val, epochs=epochs_val,
                    validation_data=(X_validation, y_validation),
shuffle=True)


plt.figure(1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['training', 'validation'])
plt.title('Loss')
plt.xlabel('Epoch')
plt.figure(2)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training', 'validation'])
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.show()
score = model.evaluate(X_test, y_test, verbose=0)
print('Test Score:', score[0])
print('Test Accuracy:', score[1])

# STORE THE MODEL AS A PICKLE OBJECT
with open("model_trained.p", "wb") as pickle_out:
    pickle.dump(model, pickle_out)

cv2.waitKey(0)
```

for code :

```
import numpy as np
import cv2
import pickle
import tkinter as tk
from tkinter import filedialog
import os
import pyttsx3

frameWidth = 640
frameHeight = 480
brightness = 180
threshold = 0.5
font = cv2.FONT_HERSHEY_SIMPLEX

with open("model_trained1.p", "rb") as pickle_in:
```

```
    model = pickle.load(pickle_in)

engine = pyttsx3.init()


def preprocessing(img):
    img = cv2.resize(img, (32, 32))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = img / 255
    return img


def getClassName(classNo):
    classes = {
        0: 'Speed Limit 20 km/h', 1: 'Speed Limit 30 km/h', 2: 'Speed Limit
50 km/h',
        3: 'Speed Limit 60 km/h', 4: 'Speed Limit 70 km/h', 5: 'Speed Limit
80 km/h',
        6: 'End of Speed Limit 80 km/h', 7: 'Speed Limit 100 km/h', 8:
'Speed Limit 120 km/h',
        9: 'No passing', 10: 'No passing for vehicles over 3.5 metric
tons',
        11: 'Right-of-way at the next intersection', 12: 'Priority road',
13: 'Yield',
        14: 'Stop', 15: 'No vehicles', 16: 'Vehicles over 3.5 metric tons
prohibited',
        17: 'No entry', 18: 'General caution', 19: 'Dangerous curve to the
left',
        20: 'Dangerous curve to the right', 21: 'Double curve', 22: 'Bumpy
road',
        23: 'Slippery road', 24: 'Road narrows on the right', 25: 'Road
work',
        26: 'Traffic signals', 27: 'Pedestrians', 28: 'Children crossing',
        29: 'Bicycles crossing', 30: 'Beware of ice/snow', 31: 'Wild
animals crossing',
        32: 'End of all speed and passing limits', 33: 'Turn right ahead',
        34: 'Turn left ahead', 35: 'Ahead only', 36: 'Go straight or
right',
        37: 'Go straight or left', 38: 'Keep right', 39: 'Keep left',
        40: 'Roundabout mandatory', 41: 'End of no passing',
        42: 'End of no passing by vehicles over 3.5 metric tons'
    }
    return classes.get(classNo, 'Unknown')


def process_webcam():
    cap = cv2.VideoCapture(0)
    cap.set(3, frameWidth)
    cap.set(4, frameHeight)
    cap.set(10, brightness)

    while True:
        success, imgOrignal = cap.read()

        if not success:
            break
        img = preprocessing(imgOrignal)

        img = img.reshape(1, 32, 32, 1)

        predictions = model.predict(img)
```

```
        classIndex = np.argmax(predictions)
        probabilityValue = np.amax(predictions)
        if probabilityValue > threshold:
            cv2.putText(imgOrignal, str(classIndex) + " " +
str(getClassName(classIndex)), (20, 35), font, 0.75,
                        (5,90, 255), 2, cv2.LINE_AA)
            cv2.putText(imgOrignal, str(round(probabilityValue * 100, 2)) +
"%", (20, 75), font, 0.75, (0,10, 255), 2,
                        cv2.LINE_AA)
        cv2.imshow("Result", imgOrignal)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()


def process_folder():
    root = tk.Tk()
    root.withdraw()
    folder_selected = filedialog.askdirectory()

    for filename in os.listdir(folder_selected):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            imgOrignal = cv2.imread(os.path.join(folder_selected,
filename))
            if imgOrignal is not None:
                img = preprocessing(imgOrignal)

                img = img.reshape(1, 32, 32, 1)

                predictions = model.predict(img)
                classIndex = np.argmax(predictions)
                probabilityValue = np.amax(predictions)
                if probabilityValue > threshold:
                    class_name = getClassName(classIndex)
                    cv2.putText(imgOrignal, str(classIndex) + " " +
class_name, (20, 35), font, 0.75,
                                (0, 0, 255), 2, cv2.LINE_AA)
                    cv2.putText(imgOrignal, str(round(probabilityValue *
100, 2)) + "%", (20, 75), font, 0.75,
                                (0, 0, 255), 2,
                                cv2.LINE_AA)

                    engine.say(class_name)
                    engine.runAndWait()

                cv2.imshow("Result", imgOrignal)
                cv2.waitKey(0)

    cv2.destroyAllWindows()


root = tk.Tk()
root.title("Select Option")
root.geometry("300x100")

webcam_button = tk.Button(root, text="Use Webcam", command=process_webcam)
webcam_button.pack(pady=10)
```

```
folder_button = tk.Button(root, text="Use Images from Folder",
command=process_folder)
folder_button.pack(pady=10)

root.mainloop()
```