Collecting Image

In [ ]:
```
import os
import cv2

DATA_DIR = './data'
if not os.path.exists(DATA_DIR):
    os.makedirs(DATA_DIR)

number_of_classes = 4  # Assuming you have 26 classes
dataset_size = 100

cap = cv2.VideoCapture(0)

try:
    # if not cap.isOpened():
    #     print("Error: Could not open camera.")
    #     exit()

    for j in range(number_of_classes):
        if not os.path.exists(os.path.join(DATA_DIR, str(j))):
            os.makedirs(os.path.join(DATA_DIR, str(j)))

        print('Collecting data for class {}'.format(j))

        while True:
            ret, frame = cap.read()
            cv2.putText(frame, 'Ready? Press "S" to start capturing, "N" to skip, or "q" to exit!', (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0, 255, 0), 3, cv2.LINE_AA)
            cv2.imshow('frame', frame)
            key = cv2.waitKey(25)

            if key == ord('s'):
                print("Capturing data for class {}".format(j))
                break
            elif key == ord('n'):
                print("Skipping class {}".format(j))
                break
            elif key == ord('q'):
                print("Exiting the program.")
                cap.release()
                cv2.destroyAllWindows()



        if key == ord('n'):
            continue  # Skip to the next class

        counter = 0
        while counter < dataset_size:
            ret, frame = cap.read()
            cv2.imshow('frame', frame)
            cv2.waitKey(25)
```

```
            cv2.imwrite(os.path.join(DATA_DIR, str(j), '{}.jpg'.format(counter)), frame)
            counter += 1

except Exception as e:
    print("An error occurred:", e)

finally:
    cap.release()
    cv2.destroyAllWindows()
```

In [ ]:
```
cap.release()
cv2.destroyAllWindows()
```

Create DataSet

Referred

In [ ]:
```
import os   # for directory operations
import pickle # This module implements binary protocols for serializing and de-serializing Python objects. It's used for saving and loading objects to/from files.

import mediapipe as mp #
import cv2 #
import matplotlib.pyplot as plt # for plotting images

# Load the hand tracking model
mp_hands = mp.solutions.hands # for hand tracking
mp_drawing = mp.solutions.drawing_utils # for drawing landmarks
mp_drawing_styles = mp.solutions.drawing_styles # for drawing styles
```

In [ ]:
```
# Static image mode is used for processing a single image. min_detection_confidence is the minimum confidence value ([0.0, 1.0]) for hand detection to be considered successful.

hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.3)
```

Displaying Raw Images

In [ ]:
```
for dir_ in os.listdir(DATA_DIR): # loop through the classes
    for img_path in os.listdir(os.path.join(DATA_DIR, dir_))[:1]: # loop through the images
        img = cv2.imread(os.path.join(DATA_DIR, dir_, img_path)) # read the image
        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # convert the image to RGB

        plt.figure() # create a new figure
        plt.imshow(img_rgb) # display the image
```

Displaying annotated Images

In [ ]:
```
# To show the land marks on the image (only last frame "Just for Understanding")

DATA_DIR = './data' # path to the data directory

for dir_ in os.listdir(DATA_DIR): # loop through the classes
```

```
    for img_path in os.listdir(os.path.join(DATA_DIR, dir_))[:1]: # loop through the images
        img = cv2.imread(os.path.join(DATA_DIR, dir_, img_path)) # read the image
        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # convert the image to RGB

        results = hands.process(img_rgb) # this will process the image and return the landmarks
        if results.multi_hand_landmarks: # if there are hands in the image
            for hands_landmarks in results.multi_hand_landmarks: # loop through the hands
                mp_drawing.draw_landmarks(img_rgb, # image to draw
                                          hands_landmarks, # Model output
                                          mp_hands.HAND_CONNECTIONS, # hand connections
                                          mp_drawing_styles.get_default_hand_landmarks_style(), # drawing styles
                                          mp_drawing_styles.get_default_hand_connections_style()) # drawing styles



        plt.figure() # create a new figure
        plt.imshow(img_rgb) # display the image
```

In [ ]:

```
DATA_DIR = './data' # path to the data directory

# Create empty lists to store the data and labels
data = []
labels = []


for dir_ in os.listdir(DATA_DIR):  # loop through the classes
    for img_path in os.listdir(os.path.join(DATA_DIR, dir_)): # loop through the images
        data_aux = [] # create an empty list to store the landmarks

        x_ = [] # create an empty list to store the x coordinates
        y_ = [] # create an empty list to store the y coordinates

        img = cv2.imread(os.path.join(DATA_DIR, dir_, img_path)) # read the image
        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # convert the image to RGB

        results = hands.process(img_rgb) # this will process the image and return the landmarks

        if results.multi_hand_landmarks: # if there are hands in the image
            for hand_landmarks in results.multi_hand_landmarks: # loop through the hands
                for i in range(len(hand_landmarks.landmark)): # loop through the landmarks
                    x = hand_landmarks.landmark[i].x # x coordinate of the landmark
                    y = hand_landmarks.landmark[i].y # y coordinate of the landmark

                    x_.append(x) # append the x coordinate to the x_ list
                    y_.append(y) # append the y coordinate to the y_ list

                for i in range(len(hand_landmarks.landmark)): # loop through the landmarks
                    x = hand_landmarks.landmark[i].x # x coordinate of the landmark
                    y = hand_landmarks.landmark[i].y # y coordinate of the landmark
                    data_aux.append(x - min(x_)) # append the normalized x coordinate to the data_aux list
                    data_aux.append(y - min(y_)) # append the normalized y coordinate to the data_aux list

            data.append(data_aux) # append the data_aux list to the data list
            labels.append(dir_) # append the label to the labels list

f = open('data.pickle', 'wb') # open a file in binary write mode
```

```
pickle.dump({'data': data, 'labels': labels}, f) # save the data and labels to the file
f.close() # close the file
```

Confirmation [Recommended to included only 2 class for understanding]

In [ ]:
```
# Total number of classes [Do only for 2 classes]
dir_
```

In [ ]:
```
# There are 42 floating point numbers for each image. This is because we have 21 landmarks and each landmark has an x and y coordinate. We can reshape the data to have 21 rows and 2 columns.
# data_aux hold the data for the last image from the lastclass
data_aux
```

Extracting data from pickle file and confirmation

In [ ]:
```
import pandas as pd
datafrompickle = pd.read_pickle('data.pickle') # read the data from the pickle file
```

In [ ]:
```
# label represent the class

print(datafrompickle['labels'][0:100]) # start from 0 to 100 labels not including 100
```

In [ ]:
```
print(datafrompickle['labels'][100:200]) # get the labels from index 100 till 200 (not include 200)
```

In [ ]:
```
# showing the end of list
print(datafrompickle['labels'][200]) # since only 2 classes, we have 0th label for 0th class and 1st label for 1st class totally 200 labels only
```

In [ ]:
```
list = datafrompickle['labels'][100:200] # get the labels from index 100 till 200 (not include 200)
# Example list

# Count the number of ones
num_ones = list.count('1')

print("Number of ones in the list:", num_ones, "--> hence each class has 100 image and each are labelled correctly") # print the number of ones in the list
```

In [ ]:
```
# Display the data
print(datafrompickle['data'][0:199]) # Display the first 200 rows of the data because the we collected 100 images for each class and if we have 2 classes then we have 200 images
```

In [ ]:
```
# Compare the stored data with the original data
# data is original data before dumping to pickle
# datafrompickle is the data after loading from pickle

if datafrompickle['data'][199] == data_aux:
    print('The data is the same as the original data')
else:
    print('The data is not the same as the original data')
```

In [ ]:
```
%pip install openpyxl
```

```
Requirement already satisfied: openpyxl in c:\users\sbsa5\miniconda3\envs\handfeb\lib\site-packages (3.1.2)
Requirement already satisfied: et-xmlfile in c:\users\sbsa5\miniconda3\envs\handfeb\lib\site-packages (from openpyxl) (1.1.0)
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:
```python
import pandas as pd

# Load the data from the pickle file
data_from_pickle = pd.read_pickle('data.pickle')

# Create a DataFrame from the data
df = pd.DataFrame(data_from_pickle['data'], columns=[f'x_{i}' for i in range(21)] + [f'y_{i}' for i in range(21)])

# Add the label column to the DataFrame
df['label'] = data_from_pickle['labels']

# Save the DataFrame to an Excel file
df.to_excel('data.xlsx', index=False)
```

In [ ]:
```python

```