Bachelor of Technology (B. Tech.)

Computer and Communication Engineering (CCE)

Amrita School of Engineering

Coimbatore Campus (India)

Academic Year – 2022 - 23

Team 06

S. No	Name	Roll Number
1	B R Aravind Ragav	CB.EN.U4CCE20008
2	Hariviyaas.B	CB.EN.U4CCE20023
3	Parthiv.V	CB.EN.U4CCE20041
4	Sudhan Saravanan	CB.EN.U4CCE20061

Fifth Semester

19CCE301 Internet of Things (IoT) Project

IoT-Based Gesture recognition using Raspberry Pi

Faculty In-charge – Peeyush K P

TABLE OF CONTENTS

Introduction

With the advent of technology and the increase in the amount of electronic devices being used all over the world, it is necessary to bridge the communication gap between machines and people with hearing impairments, deaf people, and those with speaking disabilities can only communicate with people that are skilled in sign language, this project tries to make it easier for people to converse freely, another way that this project can be implemented is with gesture recognition, where a recognized gesture can be used to control a device or perform a task on the raspberry pi device.

Abstract

This project proposes a way to communicate with people who have speech and ear impairments. People who are impaired adopt sign language as to communicate with people. However, not everyone is knowledgeable with sign language. With the intervention of Deep learning Image processing and the help of Raspberry Pi we have adopted a way to detect the gestures made which lets us understand what each sign means.

Requirements

Hardware:

- Raspberry Pi (Ver 3/4)
- PiCamera module
- 16 pin LCD display (2x16 display)
- Connecting wires

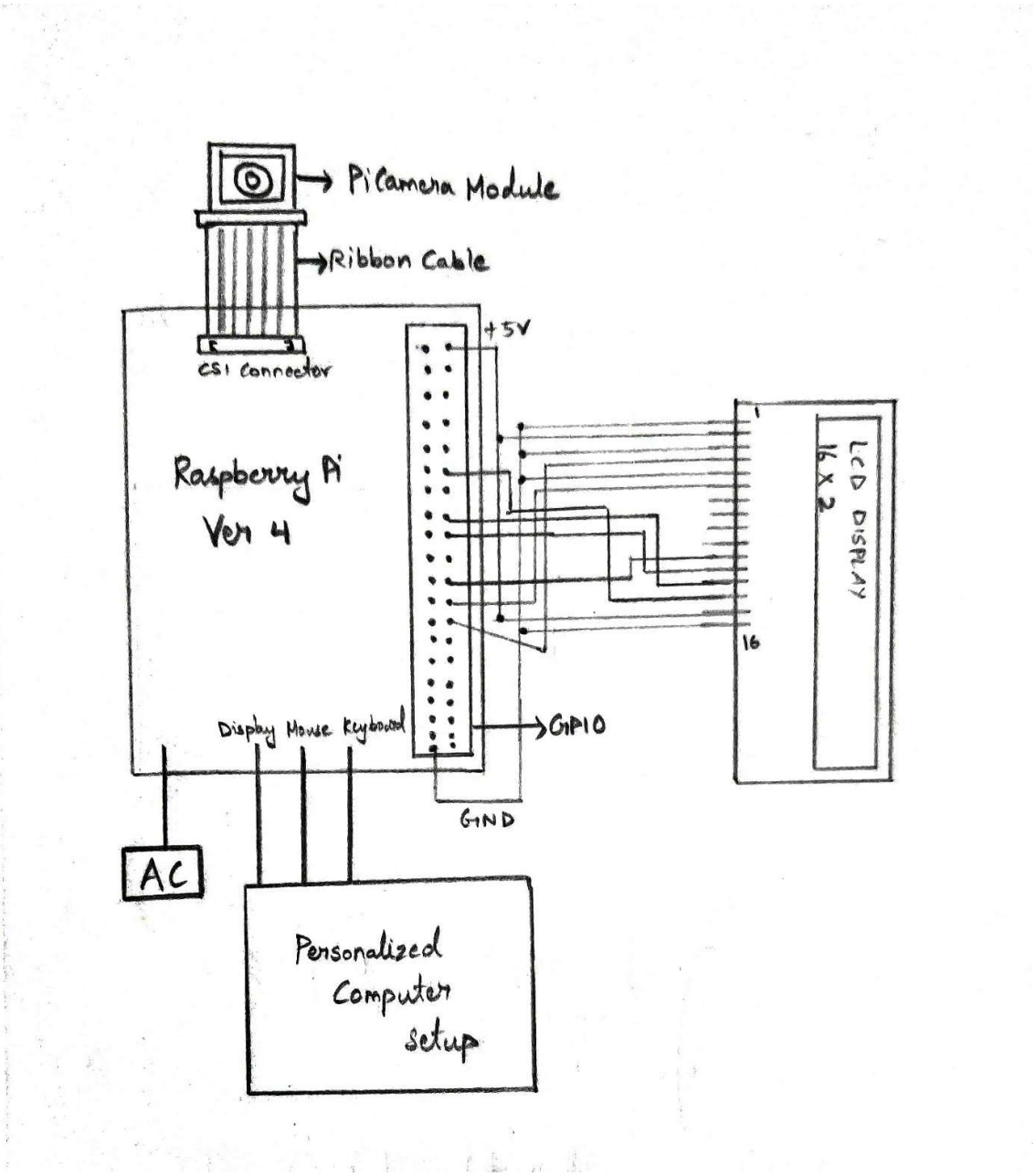
Software:

- Spyder IDE for Python
- Thonny IDE

Modules:

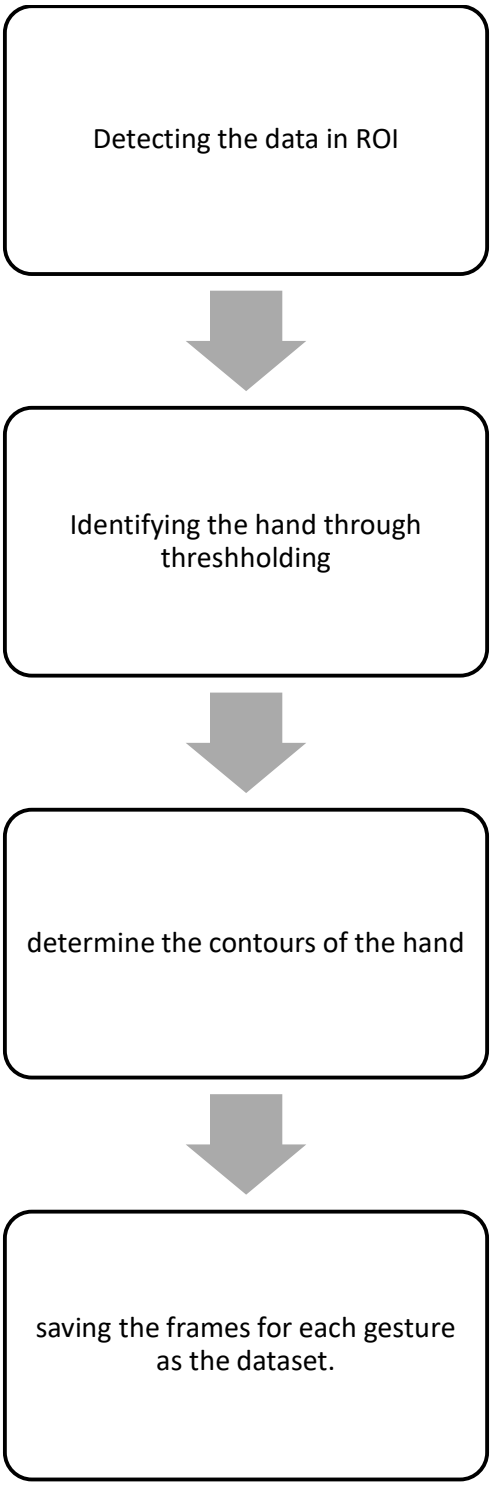
- Tensorflow
- OpenCV
- Keras
- ZMQ
- Board
- DigitialIO
- AdaFruit

Block Diagram

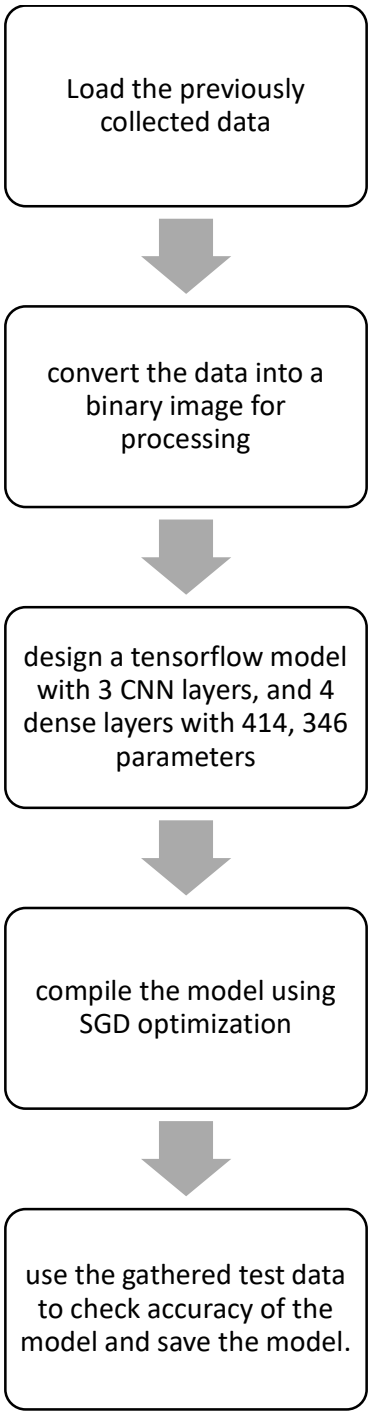


Flow Chart

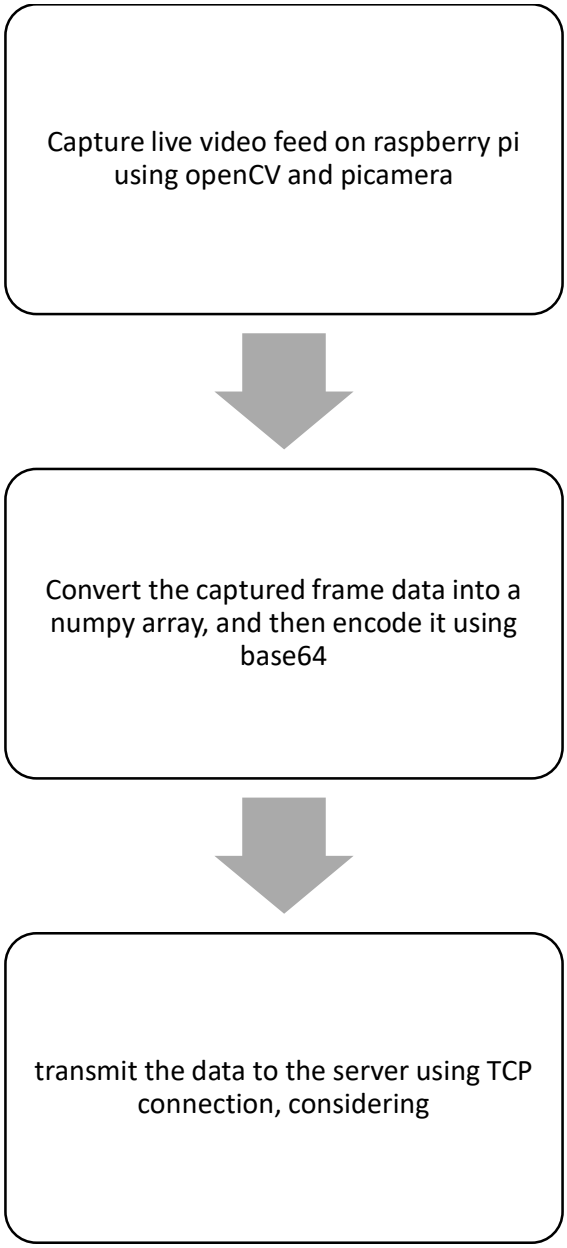
1. Creating the dataset



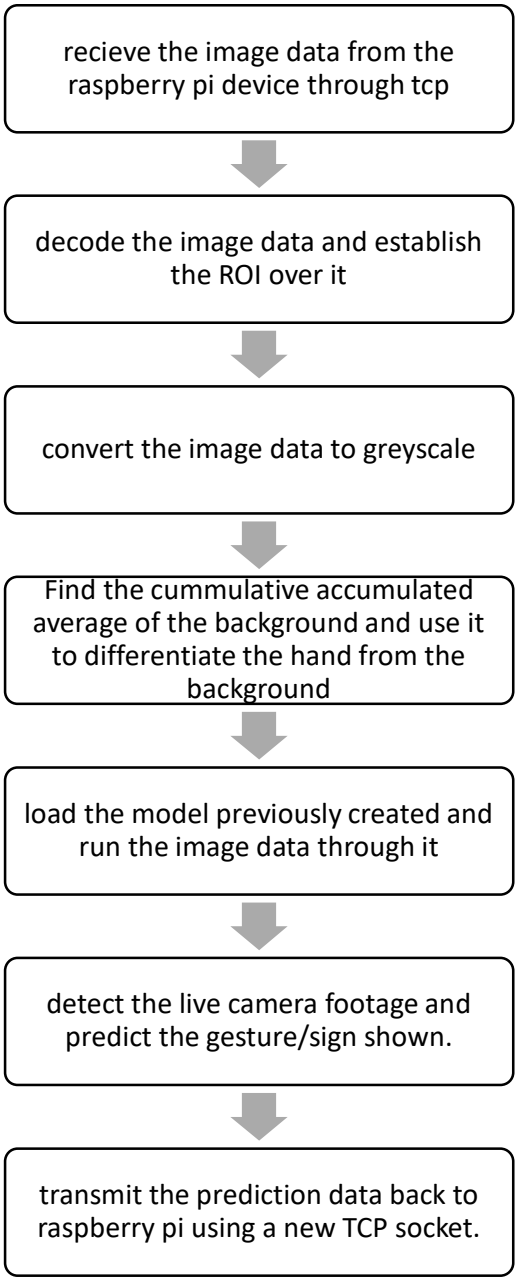
2. Training a deep-learning model



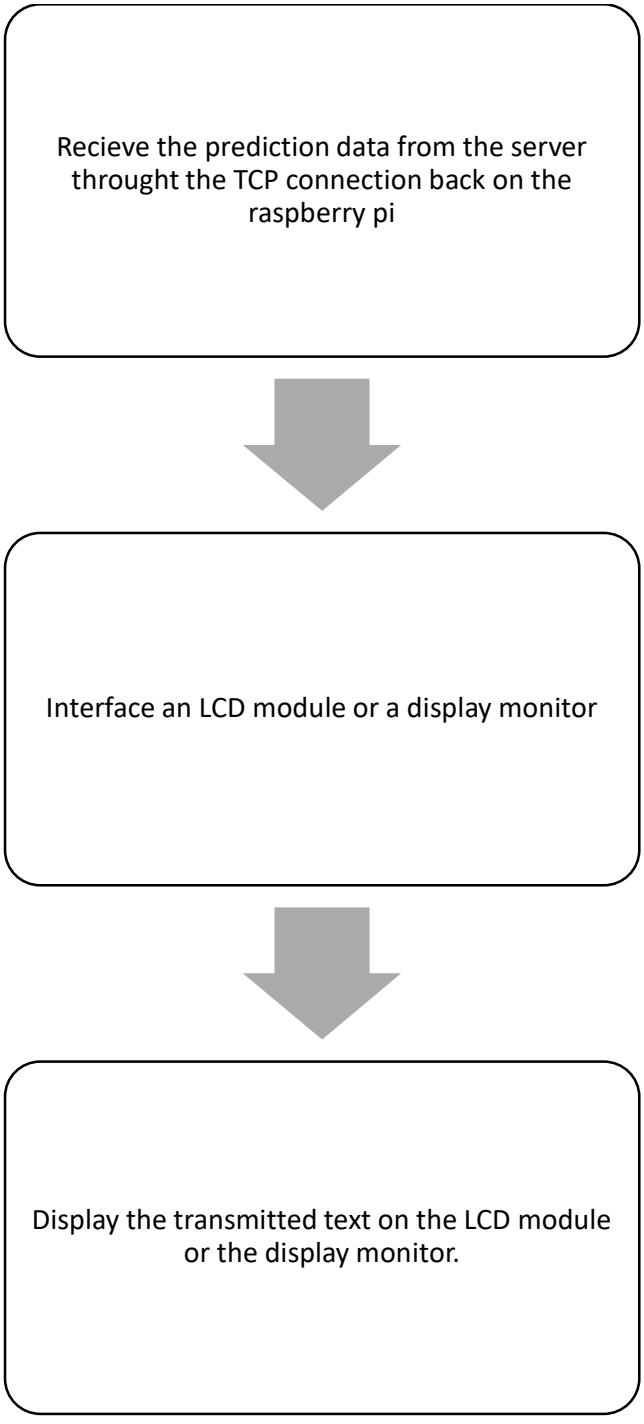
3. Transmission of live feed from raspberry pi to server



4. Gesture/Sign prediction



5. Reception and display of prediction data



Working

a. First the dataset is created by capturing images and defining the region of interest (ROI) the image data within the ROI is converted to greyscale and the background is calculated using the cumulative accumulated average. The background data is then removed from the further frames through the use of thresholding, which allows for accurate capture of the sign data, taking 300 frames for each sign as both test and train, totaling about 6000 images for a total of 10 signs.

b. The previously created dataset is put through a CNN to train a model, here, with three separate CNN layers and a total of 4,14,346 parameters, the model is then compiled with a reduced learning rate and early stop enabled, with a final accuracy of 98.67% stopping at 4 epochs under the influence of early stop, the test data is then run through and the images are classified based on their similarity to the 10 signs, the created model is saved for future use.

c. Initially the image data is captured utilizing the PiCamera module interfaced with the raspberry pi device, this data is then converted into a numpy array using the OpenCV library and then encoded using base64, this data is then transmitted to the server using TCP transmission through ZMQ python module.

d. The transmitted data is received at the server side and decoded back into image data, this image data is then processed and the ROI is defined, the data within the ROI is again converted to greyscale and the background removed as before, the data is then put through the previously saved model to predict the image data, the prediction is then sent back to the raspberry device, through a new TCP socket.

e. At the raspberry pi device, the predicted data is received through the TCP connection and is displayed on both the monitor and the interfaced LCD module.

Code

Collection of Dataset and test train compilation

```
import cv2
import numpy as np

background = None
accumulated_weight = 0.5

ROI_top = 100
ROI_bottom = 300
ROI_right = 150
ROI_left = 350


def cal_accum_avg(frame, accumulated_weight):
    global background

    if background is None:
        background = frame.copy().astype("float")
        return None

    cv2.accumulateWeighted(frame, background, accumulated_weight)


def segment_hand(frame, threshold=25):
    global background

    diff = cv2.absdiff(background.astype("uint8"), frame)

    _, thresholded = cv2.threshold(diff, threshold, 255, cv2.THRESH_BINARY)

    # Grab the external contours for the image
    contours, hierarchy = cv2.findContours(thresholded.copy(),
cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    if len(contours) == 0:
        return None
    else:

        hand_segment_max_cont = max(contours, key=cv2.contourArea)

        return (thresholded, hand_segment_max_cont)


cam = cv2.VideoCapture(0)

num_frames = 0
element = 1
num_imgs_taken = 0

while True:
    ret, frame = cam.read()

    # filpping the frame to prevent inverted image of captured frame...
    frame = cv2.flip(frame, 1)

    frame_copy = frame.copy()
```

```
    roi = frame[ROI_top:ROI_bottom, ROI_right:ROI_left]

    gray_frame = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    gray_frame = cv2.GaussianBlur(gray_frame, (9, 9), 0)

    if num_frames < 60:
        cal_accum_avg(gray_frame, accumulated_weight)
        if num_frames <= 59:
            cv2.putText(frame_copy, "FETCHING BACKGROUND...PLEASE WAIT",
(80, 400), cv2.FONT_HERSHEY_SIMPLEX, 0.9,
                        (0, 0, 255), 2)
            # cv2.imshow("Sign Detection",frame_copy)

    # Time to configure the hand specifically into the ROI...
    elif num_frames <= 300:

        hand = segment_hand(gray_frame)

        cv2.putText(frame_copy, "Adjust hand...Gesture for" + str(element),
(200, 400), cv2.FONT_HERSHEY_SIMPLEX, 1,
                    (0, 0, 255), 2)

        # Checking if hand is actually detected by counting number of
contours detected...
        if hand is not None:
            thresholded, hand_segment = hand

            # Draw contours around hand segment
            cv2.drawContours(frame_copy, [hand_segment + (ROI_right,
ROI_top)], -1, (255, 0, 0), 1)

            cv2.putText(frame_copy, str(num_frames) + "For" + str(element),
(70, 45), cv2.FONT_HERSHEY_SIMPLEX, 1,
                        (0, 0, 255), 2)

            # Also display the thresholded image
            cv2.imshow("Thresholded Hand Image", thresholded)

    else:

        # Segmenting the hand region...
        hand = segment_hand(gray_frame)

        # Checking if we are able to detect the hand...
        if hand is not None:

            # unpack the thresholded img and the max_contour...
            thresholded, hand_segment = hand

            # Drawing contours around hand segment
            cv2.drawContours(frame_copy, [hand_segment + (ROI_right,
ROI_top)], -1, (255, 0, 0), 1)

            cv2.putText(frame_copy, str(num_frames), (70, 45),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
            cv2.putText(frame_copy, str(num_frames) + "For" + str(element),
(70, 45), cv2.FONT_HERSHEY_SIMPLEX, 1,
                        (0, 0, 255), 2)
            # cv2.putText(frame_copy, str(num_imgs_taken) + 'images' +"For"
+ str(element), (200, 400), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)
```

```
            # Displaying the thresholded image
            cv2.imshow("Thresholded Hand Image", thresholded)
            if num_imgs_taken <= 300:
                cv2.imwrite(r"D:\gesture\test\\" + str(element) + "\\" +
str(num_imgs_taken + 300) + '.jpg',
                            thresholded)
                # cv2.imwrite(r"D:\\gesture\\x"+"\\" + str(num_imgs_taken)
+ '.jpg', thresholded)
            else:
                break
            num_imgs_taken += 1
        else:
            cv2.putText(frame_copy, 'No hand detected...', (200, 400),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

    # Drawing ROI on frame copy
    cv2.rectangle(frame_copy, (ROI_left, ROI_top), (ROI_right, ROI_bottom),
(255, 128, 0), 3)

    cv2.putText(frame_copy, "DataFlair hand sign recognition_ _ _", (10,
20), cv2.FONT_ITALIC, 0.5, (51, 255, 51), 1)

    # increment the number of frames for tracking
    num_frames += 1

    # Display the frame with segmented hand
    cv2.imshow("Sign Detection", frame_copy)

    # Closing windows with Esc key...(any other key with ord can be used
too.)
    k = cv2.waitKey(1) & 0xFF

    if k == 27:
        break

# Releasing camera & destroying all the windows...

cv2.destroyAllWindows()
cam.release()
```

Creation of TensorFlow Model

```
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Activation, Dense, Flatten, BatchNormalization,
Conv2D, MaxPool2D, Dropout
from keras.optimizers import Adam, SGD
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator
import itertools
import random
import warnings
import numpy as np
import cv2
from keras.callbacks import ReduceLROnPlateau
from keras.callbacks import ModelCheckpoint, EarlyStopping
warnings.simplefilter(action='ignore', category=FutureWarning)
```

```
import matplotlib.pyplot as plt

train_path = r'D:\gesture\train'
test_path = r'D:\gesture\test'

train_batches =
ImageDataGenerator(preprocessing_function=tf.keras.applications.vgg16.prepr
ocess_input).flow_from_directory(directory=train_path, target_size=(64,64),
class_mode='categorical', batch_size=10,shuffle=True)
test_batches =
ImageDataGenerator(preprocessing_function=tf.keras.applications.vgg16.prepr
ocess_input).flow_from_directory(directory=test_path, target_size=(64,64),
class_mode='categorical', batch_size=10, shuffle=True)

imgs, labels = next(train_batches)


#Plotting the images...
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 10, figsize=(30,20))
    axes = axes.flatten()
    for img, ax in zip( images_arr, axes):
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        ax.imshow(img)
        ax.axis('off')
    plt.tight_layout()
    plt.show()


plotImages(imgs)
print(imgs.shape)
print(labels)

model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu',
input_shape=(64,64,3)))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding
= 'same'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu',
padding = 'valid'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Flatten())

model.add(Dense(64,activation ="relu"))
model.add(Dense(128,activation ="relu"))
model.add(Dropout(0.2))
model.add(Dense(128,activation ="relu"))
model.add(Dropout(0.3))
model.add(Dense(10,activation ="softmax"))


# In[23]:


model.compile(optimizer=Adam(learning_rate=0.001),
```

```
loss='categorical_crossentropy', metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=1,
min_lr=0.0001)
early_stop = EarlyStopping(monitor='val_loss', min_delta=0, patience=2,
verbose=0, mode='auto')



model.compile(optimizer=SGD(learning_rate=0.001),
loss='categorical_crossentropy', metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=1,
min_lr=0.0005)
early_stop = EarlyStopping(monitor='val_loss', min_delta=0, patience=2,
verbose=0, mode='auto')


history2 = model.fit(train_batches, epochs=10, callbacks=[reduce_lr,
early_stop],  validation_data = test_batches)#, checkpoint])
imgs, labels = next(train_batches) # For getting next batch of imgs...

imgs, labels = next(test_batches) # For getting next batch of imgs...
scores = model.evaluate(imgs, labels, verbose=0)
print(f'{model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]}
of {scores[1]*100}%')


#model.save('best_model_dataflair.h5')
model.save('best_model_dataflair3.h5')

print(history2.history)

imgs, labels = next(test_batches)

model = keras.models.load_model(r"best_model_dataflair3.h5")

scores = model.evaluate(imgs, labels, verbose=0)
print(f'{model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]}
of {scores[1]*100}%')

model.summary()

scores #[loss, accuracy] on test data...
model.metrics_names

word_dict = {0:'peace',1:'I love
you',2:'great',3:'luck',4:'ok',5:'stop',6:'why',7:'phone',8:'where',9:'wate
r'}

predictions = model.predict(imgs, verbose=0)
print("predictions on a small set of test data--")
print("")
for ind, i in enumerate(predictions):
    print(word_dict[np.argmax(i)], end='   ')

plotImages(imgs)
print('Actual labels')
for i in labels:
    print(word_dict[np.argmax(i)], end='   ')

print(imgs.shape)
history2.history
```

Sending image data from Raspberry pi to server

```
import cv2
import numpy as np
import base64
import zmq

context = zmq.Context()
footage_socket = context.socket(zmq.PUB)
footage_socket.connect('tcp://192.168.80.251:5000')

cam = cv2.VideoCapture(0)
ROI_top = 100
ROI_bottom = 300
ROI_right = 150
ROI_left = 350

while (True):
    try:
        ret, frame = cam.read()
        frame = cv2.rotate(cv2.flip(frame, 1), cv2.ROTATE_180)
        frame_copy = frame.copy()
        encoded, buffer = cv2.imencode('.jpg', frame_copy)
        roi = frame[ROI_top:ROI_bottom, ROI_right:ROI_left]
        gray_frame = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
        jpg_as_text = base64.b64encode(buffer)
        footage_socket.send(jpg_as_text)


    except KeyboardInterrupt:
        cam.release()
        cv2.destroyAllWindows()
        break
```

Processing of image data at server side

```
import numpy as np
import cv2
import keras
from keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
import zmq
import base64

model = keras.models.load_model(r"D:\best_model_dataflair3.h5")

context = zmq.Context()
footage_socket = context.socket(zmq.SUB)
footage_socket.bind('tcp://*:5000')
footage_socket.setsockopt_string(zmq.SUBSCRIBE, np.unicode(''))

cont = zmq.Context()
data_socket = context.socket(zmq.PUB)
data_socket.connect('tcp://192.168.80.69:6000')

background = None
accumulated_weight = 0.5
```

```
ROI_top = 100
ROI_bottom = 300
ROI_right = 150
ROI_left = 350
word_dict = {0: 'peace', 1: 'I love you', 2: 'great', 3: 'luck', 4: 'ok',
5: 'stop', 6: 'why', 7: 'phone', 8: 'where',
             9: 'water'}


def cal_accum_avg(frame, accumulated_weight):
    global background

    if background is None:
        background = frame.copy().astype("float")
        return None

    cv2.accumulateWeighted(frame, background, accumulated_weight)


def segment_hand(frame, threshold=25):
    global background

    diff = cv2.absdiff(background.astype("uint8"), frame)

    _, thresholded = cv2.threshold(diff, threshold, 255, cv2.THRESH_BINARY)

    # Fetching contours in the frame (These contours can be of hand or any
other object in foreground) ...
    contours, hierarchy = cv2.findContours(thresholded.copy(),
cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    # If length of contours list = 0, means we didn't get any contours...
    if len(contours) == 0:
        return None
    else:
        # The largest external contour should be the hand
        hand_segment_max_cont = max(contours, key=cv2.contourArea)

        # Returning the hand segment(max contour) and the thresholded image
of hand...
        return (thresholded, hand_segment_max_cont)


cam = cv2.VideoCapture(0)
num_frames = 0
while True:
    fr = footage_socket.recv_string()
    img = base64.b64decode(fr)
    npimg = np.fromstring(img, dtype=np.uint8)
    source = cv2.imdecode(npimg, 1)

    # filpping the frame to prevent inverted image of captured frame...
    frame = cv2.flip(source, 1)

    frame_copy = frame.copy()

    # ROI from the frame
    roi = frame[ROI_top:ROI_bottom, ROI_right:ROI_left]

    gray_frame = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    gray_frame = cv2.GaussianBlur(gray_frame, (9, 9), 0)
```

```
    if num_frames < 70:

        cal_accum_avg(gray_frame, accumulated_weight)

        cv2.putText(frame_copy, "FETCHING BACKGROUND...PLEASE WAIT", (80,
400), cv2.FONT_HERSHEY_SIMPLEX, 0.9,
                    (0, 0, 255), 2)

    else:
        # segmenting the hand region
        hand = segment_hand(gray_frame)

        # Checking if we are able to detect the hand...
        if hand is not None:
            thresholded, hand_segment = hand

            # Drawing contours around hand segment
            cv2.drawContours(frame_copy, [hand_segment + (ROI_right,
ROI_top)], -1, (255, 0, 0), 1)

            cv2.imshow("Thesholded Hand Image", thresholded)

            thresholded = cv2.resize(thresholded, (64, 64))
            thresholded = cv2.cvtColor(thresholded, cv2.COLOR_GRAY2RGB)
            thresholded = np.reshape(thresholded, (1, thresholded.shape[0],
thresholded.shape[1], 3))

            pred = model.predict(thresholded)
            cv2.putText(frame_copy, word_dict[np.argmax(pred)], (170, 45),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

            data_socket.send_string(word_dict[np.argmax(pred)])

    # Draw ROI on frame_copy
    cv2.rectangle(frame_copy, (ROI_left, ROI_top), (ROI_right, ROI_bottom),
(255, 128, 0), 3)

    # incrementing the number of frames for tracking
    num_frames += 1

    # Display the frame with segmented hand
    cv2.putText(frame_copy, "DataFlair hand sign recognition_ _ _", (10,
20), cv2.FONT_ITALIC, 0.5, (51, 255, 51), 1)
    cv2.imshow("Sign Detection", frame_copy)

    # Close windows with Esc
    k = cv2.waitKey(1) & 0xFF

    if k == 27:
        break

# Release the camera and destroy all the windows
cam.release()
cv2.destroyAllWindows()
```

Displaying Data on LCD screen

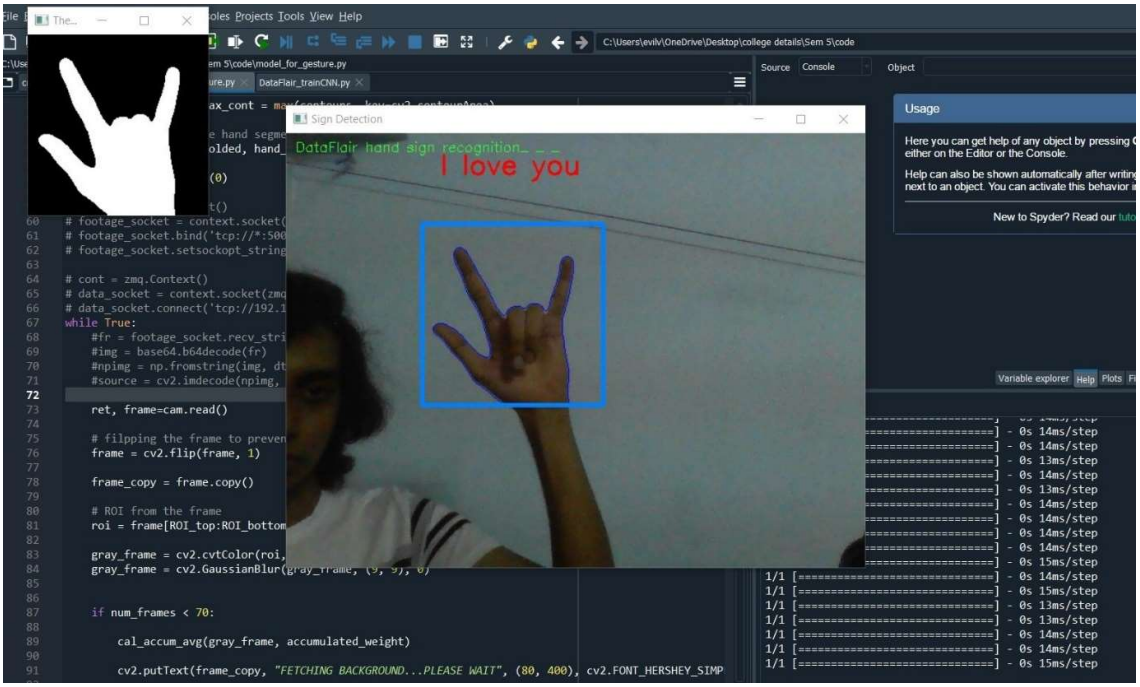
```
import zmq
import numpy as np

import board
import time
import digitalio
import adafruit_character_lcd.character_lcd as characterlcd


context = zmq.Context()
footage_socket = context.socket(zmq.SUB)
footage_socket.bind('tcp://*:6000')
footage_socket.setsockopt_string(zmq.SUBSCRIBE, np.unicode(''))


lcd_rs = digitalio.DigitalInOut(board.D7)#26
lcd_en = digitalio.DigitalInOut(board.D8)#24
lcd_d7 = digitalio.DigitalInOut(board.D18)#12
lcd_d6 = digitalio.DigitalInOut(board.D23)#16
lcd_d5 = digitalio.DigitalInOut(board.D24)#18
lcd_d4 = digitalio.DigitalInOut(board.D25)#22

lcd_columns = 16
lcd_rows = 2


lcd = characterlcd.Character_LCD_Mono(lcd_rs, lcd_en, lcd_d4, lcd_d5,
lcd_d6, lcd_d7, lcd_columns, lcd_rows)


while True:
    data = footage_socket.recv_string()
    print(data)
    lcd.clear()
    lcd.cursor = True
    lcd.message=data #send msg to my LCD
    time.sleep(1) #delay of 5 seconds
    lcd.cursor = False
    lcd.clear()
```

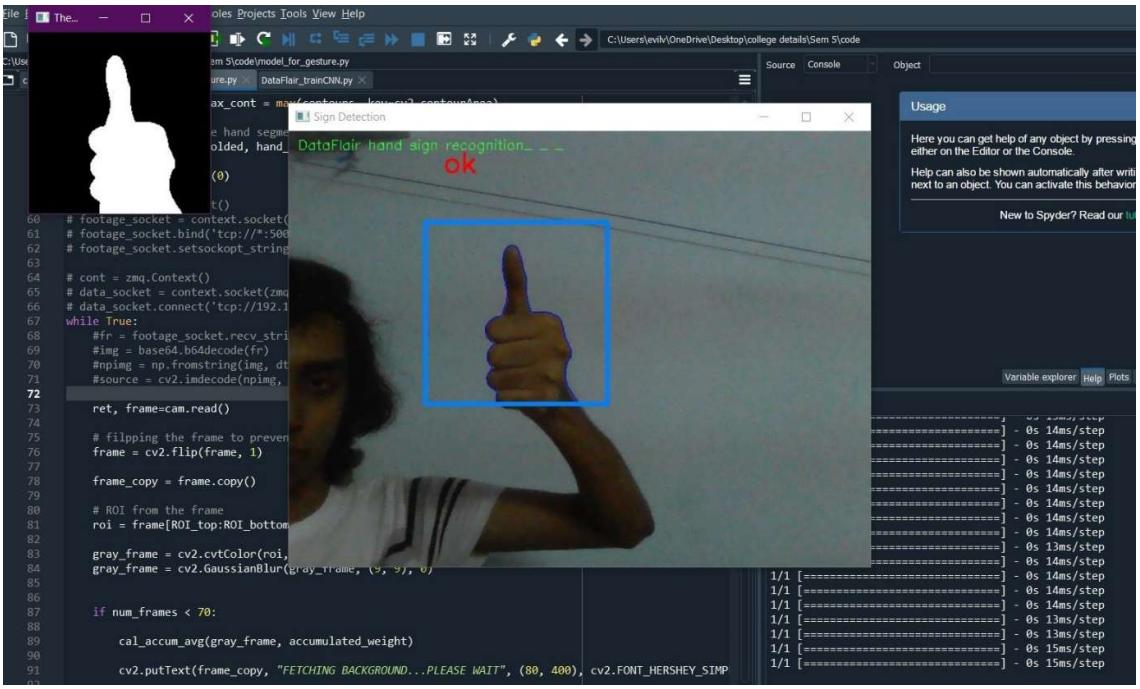
Output Images

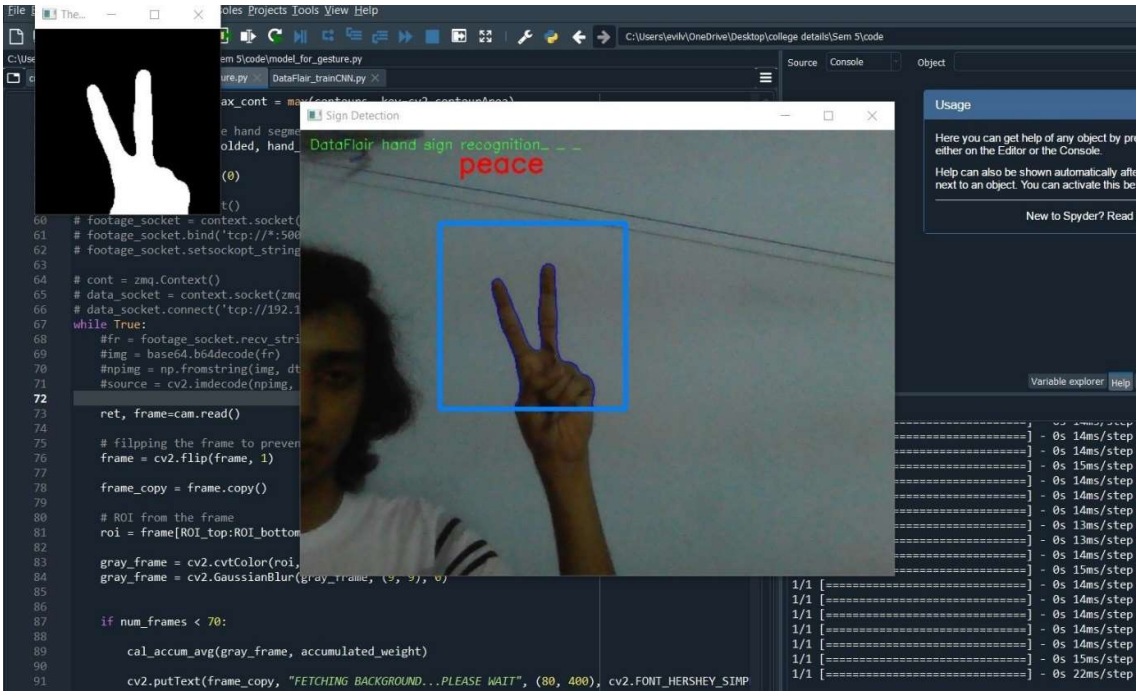
Dataset example:

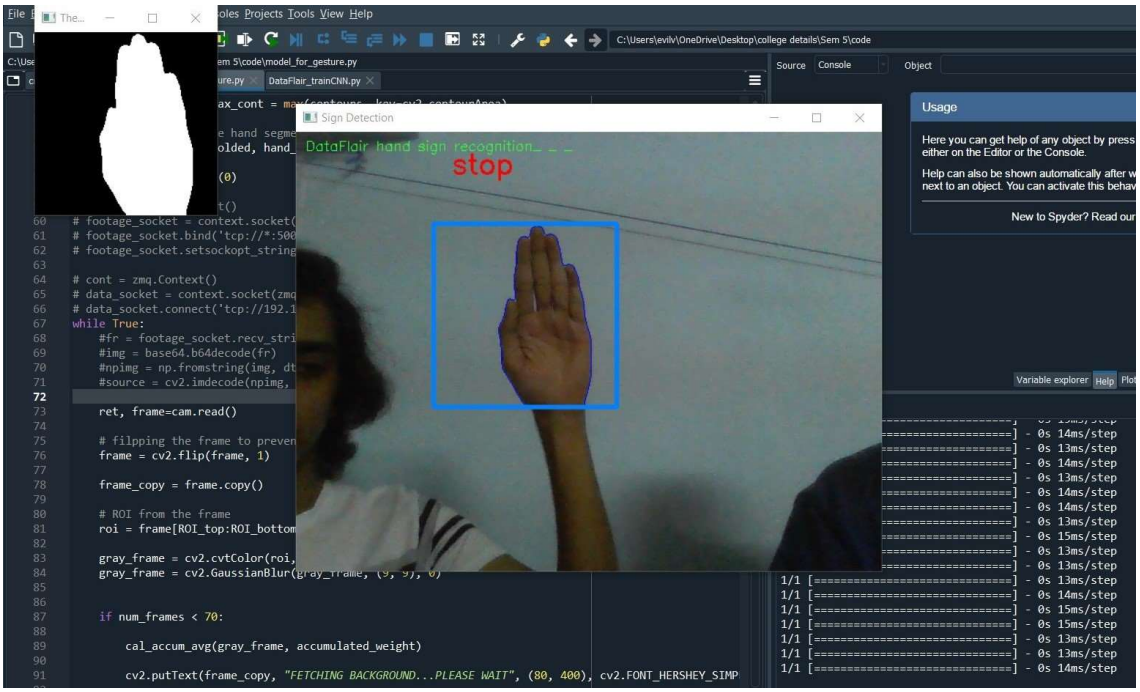
Model creation test data:

Sign detection outputs:

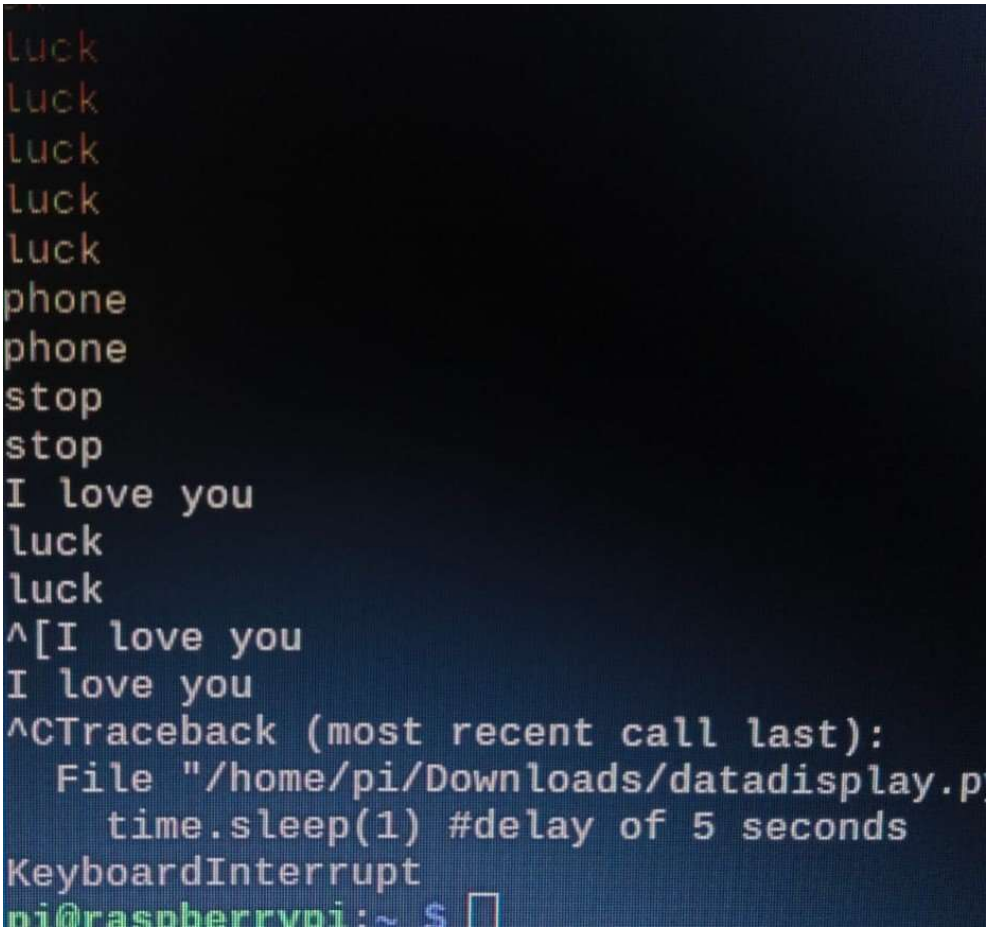



Sign Detection
DataFlair hand sign recognition_ _ _
ok
Usage
Here you can get help of any object by pressing
either on the Editor or the Console.
New to Spyder? Read our
Variable explorer
Help
Plots


Sign Detection
DataFlair hand sign recognition_ _ _
peace
Usage
Here you can get help of any object by pr
either on the Editor or the Console.
New to Spyder? Read
Variable explorer
Help



Raspberry pi monitor output:



LCD output: