In [ ]:
```python
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from PIL import Image, ImageOps
import pandas as pd
import numpy as np
import os
from os import listdir
import cv2
import time
import mediapipe
```

In [ ]:
```python
# import and format the training and testing (validation) sets
# for this example, using just numbers
# code for letters is the same

training = pd.read_csv("dataset/asl_dataset/numbers_mp_train.csv")
training.sample(frac=1) # shuffle the images, order fed into model matters

testing = pd.read_csv("dataset/asl_dataset/numbers_mp_test.csv")
testing.sample(frac=1)

trainY = np.array(training.iloc[:,0]) # label = first element of each csv entry (row)
trainX = np.array(training.iloc[:,1:]) # rest of the elements = feature vector

testY = np.array(testing.iloc[:,0])
testX= np.array(testing.iloc[:,1:])
```

In [ ]:
```python
# create the machine learning model

model = models.Sequential()
model.add(layers.Conv1D(13, 15, padding="same",activation='relu', input_shape=(40, 1)))
model.add(layers.MaxPooling1D(2))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.2))
model.add(layers.Conv1D(25, 9, activation='relu'))
model.add(layers.MaxPooling1D(2))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.2))
model.add(layers.Conv1D(50, 5, activation='relu'))
model.add(layers.MaxPooling1D(2))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.2))

model.add(layers.Flatten())
model.add(layers.Dense(80, activation="relu"))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.2))
model.add(layers.Dense(10, activation="softmax")) # 10 for just numbers, 25 for just letters
```

In [ ]:
```python
# compile the machine learning model and set parameters for training

model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
```

In [ ]:
```python
# train the model using the training and testing (validation) sets

model.fit(trainX, trainY, validation_data=(testX, testY), epochs=15)
```

In [ ]:
```python
# generate feature vector from array of hand landmark coordinates
# same function as the one used to generate the training/validation dataset

def generateFeatures(coords):
    relative = []
    x0, y0 = coords[0]
    x1, y1 = coords[1]
    d = np.sqrt((x1-x0)*(x1-x0)+(y1-y0)*(y1-y0))
    for i in range(20):
        x1, y1 = coords[i+1]
        relative.append((x1-x0)/d)
        relative.append((y1-y0)/d)
    return np.array(relative)
```

In [ ]:

```python
# program to test the classification and code off-device, uses computer webcam

drawingModule = mediapipe.solutions.drawing_utils
handsModule = mediapipe.solutions.hands

cap = cv2.VideoCapture(0)

features = []
answer = -1

coords = []
with handsModule.Hands(static_image_mode=True, min_detection_confidence=0.7, min_tracking_confidence=0.7, max_num_hands=1) as hands:

    while True:
        _, frame = cap.read()
        frame = cv2.resize(frame, (640, 480))

        results = hands.process(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))

        # used to obtain hand bounding box
        minx, miny = 640, 480
        maxx, maxy = 0, 0

        if results.multi_hand_landmarks != None:
            for handLandmarks in results.multi_hand_landmarks:
                found = 0
                coords = []
                for point in handsModule.HandLandmark:
                    normalizedLandmark = handLandmarks.landmark[point]
                    pixelCoordinatesLandmark= drawingModule._normalized_to_pixel_coordinates(normalizedLandmark.x, normalizedLandmark.y, 640, 480)
                    if(pixelCoordinatesLandmark):
                        coords.append(pixelCoordinatesLandmark)
                        found += 1
                        x, y = pixelCoordinatesLandmark
                        if x > maxx: maxx = x
                        if x < minx: minx = x
                        if y > maxy: maxy = y
                        if y < miny: miny = y
                    else: break

                if found==21: #don't classify/draw anything unless all 21 hand landmarks are identified
                    frame = cv2.rectangle(frame, (minx-20, miny-20), (maxx+20, maxy+20), (255,0,255), 2) #bounding box
                    frame = cv2.circle(frame, coords[0], 2, (0,0,255), 3) # palm location
                    frame = cv2.circle(frame, coords[9], 2, (255,0,0), 3) # base of middle finger
                    for i in range(20):
                        frame = cv2.line(frame, coords[i], coords[i+1], (255, 255, 255), 2) # draw lines of hand
                    features = generateFeatures(coords) # obtain feature vector from detected hand
                    answer = np.argmax(model.predict(np.array([features,]))) # classify using trained model
                else: answer = -1
        else: answer = -1

        frame = cv2.flip(frame, flipCode = 1)
        if answer>= 0: cv2.putText(frame, f'{answer}', (20, 70), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

        cv2.imshow("Frame", frame);
        key = cv2.waitKey(1) & 0xFF
        if key == ord("q"):
            break

cv2.destroyAllWindows()
```

In [ ]:

```python
# Convert the model from tensorflow to tensorflow lite to be able to run on RPi

converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model
with open('model_numbers.tflite', 'wb') as f:
  f.write(tflite_model)
```