Open Ended Problem

Automatic traffic-signal detection using neural network

- ✓ **The ultimate goal here is to identify the situation and make a decision accordingly. This is enabled with the help of various sensors, input devices and machine learning techniques such as convolution neural network, support vector machines.**
- ✓ **We have used convolution neural network image classification technique to train our car to identify various signal and react accordingly.**
- ✓ **CNN is basically machine learning technique which consists of neural that has learnable weights.**
- ✓ **It will identify various signs such as start, stop etc. and according to car will move.**

- **Hardware components required:**
  - Raspberry pi
  - Picamera
  - Car
  - Breadboard
  - Battery
  - Connecting wires

✓ **Code**

```
'''
slow
start = 1
stop = 2
'''
import RPi.GPIO as GPIO
import time
import sys
import numpy as np
import pandas as pd
import matplotlib.pylab as plt
from random import shuffle
```

```
import curses
from picamera import PiCamera
import os
import cv2
GPIO.setmode(GPIO.BOARD)
Motor1A = 16
Motor1B = 18
Motor1E = 22
LEDR = 40
Motor2A = 33
Motor2B = 35
Motor2E = 37

GPIO.setup(Motor1A,GPIO.OUT)
GPIO.setup(Motor1B,GPIO.OUT)
GPIO.setup(Motor1E,GPIO.OUT)
GPIO.setup(LEDR,GPIO.OUT)
GPIO.setup(Motor2A,GPIO.OUT)
GPIO.setup(Motor2B,GPIO.OUT)
GPIO.setup(Motor2E,GPIO.OUT)

def motor_fwd():
   GPIO.output(Motor1B,GPIO.HIGH)
   GPIO.output(Motor1A,GPIO.LOW)
   pwm.ChangeDutyCycle(40)
   GPIO.output(Motor2B,GPIO.HIGH)
   GPIO.output(Motor2A,GPIO.LOW)
   pwm1.ChangeDutyCycle(70)
   GPIO.output(LEDR,GPIO.LOW)
   print("Distance forward ")

def motor_slo():
   GPIO.output(Motor1B,GPIO.HIGH)
   GPIO.output(Motor1A,GPIO.LOW)
   #GPIO.output(Motor1E,GPIO.HIGH)
   pwm.ChangeDutyCycle(25)
   GPIO.output(Motor2B,GPIO.HIGH)
   GPIO.output(Motor2A,GPIO.LOW)
   #GPIO.output(Motor2E,GPIO.HIGH)
   pwm1.ChangeDutyCycle(45)
   GPIO.output(LEDR,GPIO.LOW)
   print("Distance forward slow ")
```

```
def motor_stop1():
   pwm.start(0)
   pwm1.start(0)
   GPIO.output(LEDR,GPIO.HIGH)
   print("Distance stop ")

pwm=GPIO.PWM(22,100)
pwm1=GPIO.PWM(37,100)
pwm.start(0)
pwm1.start(0)

train_path = "/home/pi/Desktop/data_learning/train/"

ROWS = 64
COLS = 64
CHANNELS = 3

images = [img for img in os.listdir(train_path)]
images_slow = [img for img in os.listdir(train_path) if "slow" in img]
images_start = [img for img in os.listdir(train_path) if "start" in img]
images_stop = [img for img in os.listdir(train_path) if "stop" in img]

#only taking a subset (less accuracy but faster training)
train_slow = images_slow[:]
train_start = images_start[:]
train_stop = images_stop[:]

valid_slow = images_slow[3:]
valid_start = images_start[3:]
valid_stop = images_stop[3:]

train_list = train_slow + train_start + train_stop
valid_list = valid_slow + valid_start + valid_stop

shuffle(train_list)

train = np.ndarray(shape=(len(train_list),ROWS, COLS))
train_color = np.ndarray(shape=(len(train_list), ROWS, COLS, CHANNELS),
dtype=np.uint8)
valid = np.ndarray(shape=(len(valid_list), ROWS, COLS))
valid_color = np.ndarray(shape=(len(valid_list), ROWS, COLS, CHANNELS),
dtype=np.uint8)
```

```
labels = np.ndarray(len(train_list))

for i, img_path in enumerate(train_list):
    img_color = cv2.imread(os.path.join(train_path, img_path), 1)
    img_color = cv2.resize(img_color, (ROWS, COLS),
interpolation=cv2.INTER_CUBIC)
    img = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)

    train[i] = img
    train_color[i] = img_color

    if "slow" in img_path:
        labels[i] = 0
    elif "start" in img_path:
        labels[i] = 1
    else:
        labels[i] = 2

valid_labels = np.ndarray(len(valid_list))

for i, img_path in enumerate(valid_list):
    img_color = cv2.imread(os.path.join(train_path, img_path), 1)
    img_color = cv2.resize(img_color, (ROWS, COLS),
interpolation=cv2.INTER_CUBIC)
    img = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)

    valid[i] = img
    valid_color[i] = img_color

    if "slow" in img_path:
        valid_labels[i] = 0
    elif "start" in img_path:
        valid_labels[i] = 1
    else:
        valid_labels[i] = 2

from keras.utils import np_utils

X_train = train_color / 255
X_valid = valid_color / 255
```

```
# one hot encode outputs
y_train = np_utils.to_categorical(labels)
y_valid = np_utils.to_categorical(valid_labels)
num_classes = y_valid.shape[1]

def larger_model():
        # create model
        model = Sequential()
        model.add(Convolution2D(30, 5, 5, border_mode='valid',
input_shape=(64, 64, 3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Convolution2D(15, 3, 3, activation='relu'))

        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.2))
        model.add(Flatten())
        model.add(Dense(128, activation='relu'))
        model.add(Dense(50, activation='relu'))
        model.add(Dense(num_classes, activation='softmax'))
        # Compile model
        model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
        return model

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Convolution2D
from keras.layers.convolutional import MaxPooling2D

# build the model
model = larger_model()

# Fit the model
model.fit(X_train, y_train, validation_data=(X_valid, y_valid), nb_epoch=20,
batch_size=200, verbose=1)
#model.save('model_loaded.h5')

#model = load_model('model_loaded.h5')
def pred():
   test_path = "/home/pi/Desktop/data_learning/test/"
```

```
    images_test = [img for img in os.listdir(test_path)]
    test_list  = images_test[0:]

    test = np.ndarray(shape=(len(test_list),ROWS, COLS))
    test_color = np.ndarray(shape=(len(images_test), ROWS, COLS, CHANNELS),
dtype=np.uint8)

    for i, img_path in enumerate(test_list):
        img_color = cv2.imread(os.path.join(test_path, img_path), 1)
        img_color = cv2.resize(img_color, (ROWS, COLS),
interpolation=cv2.INTER_CUBIC)
        img = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)

        test[i] = img
        test_color[i] = img_color

    X_test  = test_color  / 255

    submission = model.predict_classes(X_test, verbose=0)

    for i in range(0,len(submission)):
        if submission[i] == 0:
            motor_slo()
            print('slow')
        elif submission[i] == 1:
            motor_fwd()
            print('start')
        else:
            motor_stop1()
            print('stop')
    #pd.DataFrame({"id": list(range(1,len(test_color)+1)),
    #              "label":
submission}).to_csv('C:/Users/US/Desktop/data_learning/submission.csv',
index=False,header=True)
    print('completed')

damn = curses.initscr()
while True:
    c = damn.getch()
    if c == 32:
        print "start camera"
```

```
            camera = PiCamera()
            camera.start_preview()
            camera.capture('/home/pi/Desktop/data_learning/test/'+str(1)+'.jpg')
            camera.stop_preview()
            pred()
           # break
        if c == 115:
            break
    #scores = model.evaluate(X_valid, y_valid, verbose=0)
    #print("Classification Error: %.2f%%" % (100-scores[1]*100))
```

✓ **Output:**

- car will react according to signal that it identifies. e.g. start, slow, stop.