Self-Driving Toy Car

Self-driving car is a vehicle that is capable of sensing its environment and navigating without human input.

SAE international has categorized we are going to build level 2 autonomous vehicle.

SAE Level 2

Name Partial Automation

Definition This driving mode-specific execution by one or more driver assistance systems of *both steering and acceleration/deceleration* using information about the driving environment.

Self Driving car uses combination of:

- Cameras
- Lidar
- Ultra Sonic Sensors
- GPS

Every sensor have its drawbacks like lidar cannot sense glass and cannot distinguish between colors, Radar mainly used for sensing metal and cameras can be fooled by images

We are going to use cameras in our self-driving toy car so here are some advantages to use a camera:

- Can see a clear picture of environment around car.
- By fitting cam in all directions we can get 360 degree view around car.
- Tesla gets 2D image and make 3D mapping and get through environment.
- Camera can used to pick image of environment and can detect lanes, roads, sign etc.
- Camera can see all colors.

Lidar costs 10's of thousands while camera can be of thousands need processing power to analyze data can be expensive but still less than LIDAR making it suitable candidate to adopt. Mobile Eye an Iserali Company states "**If a human can drive on vision alone so can a computer**". Tesla and Mobile Eye believe that cams can be solo source of vision for cars.

Equipment required for building self-driving car:

- USB Camera (any resolution)
- Raspberry Pi (3B recommended)
- H-bridge Motor Driver (L298N Module recommended)
- Remote Control Toy car with steerable front wheels
- Power Source (Power Bank and batteries for motors)
- Cardboards
- A4 size papers

After buying toy car open your toy car figure out which wires are driving forward steering motor and wires which are driving back motor. Cut that wires and extend those wires by soldering some piece of wires with existing wires make a hole in body and get pairs of wires out of body and close your car case. Connect steering motor wires to OUT3 and OUT4 of motor driver. Connect other pair of wire to OUT1 and OUT2 of L298N motor driver.

Install Respbian software in Raspberry-Pi and get it running.

Install Libraries Open-CV2 in python3, Keras, Pygame and TensorFlow in raspberry pi.

Pip3 install keras

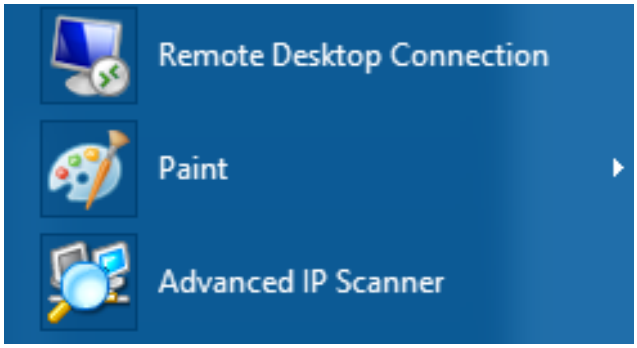
Pip3 install tensorflow

Remotely Connecting Raspberry Pi

We connect a monitor a mouse or keyboard while training, testing or data collection parts so it will be handy to use laptop to do all this stuff.

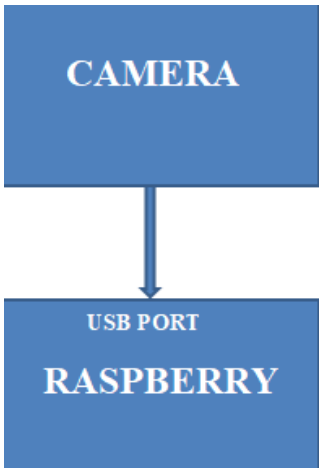
Install advance IP scanner after installing make sure your laptop and raspberry pi both are connected to same network otherwise you will not be able to connect remotely to your raspberry pi.

Open advance ip scanner start search and find out IP of Raspberry pi note down that IP after that open Remote Desktop Connection it is already installed in most of the computers put that IP address there after that a windows pops up use ‘pi’ as name and as password use ‘raspberry’ and raspberry pi environment will be initiated.



Capturing Image Frames using Open-Cv2 and saving in Memory:

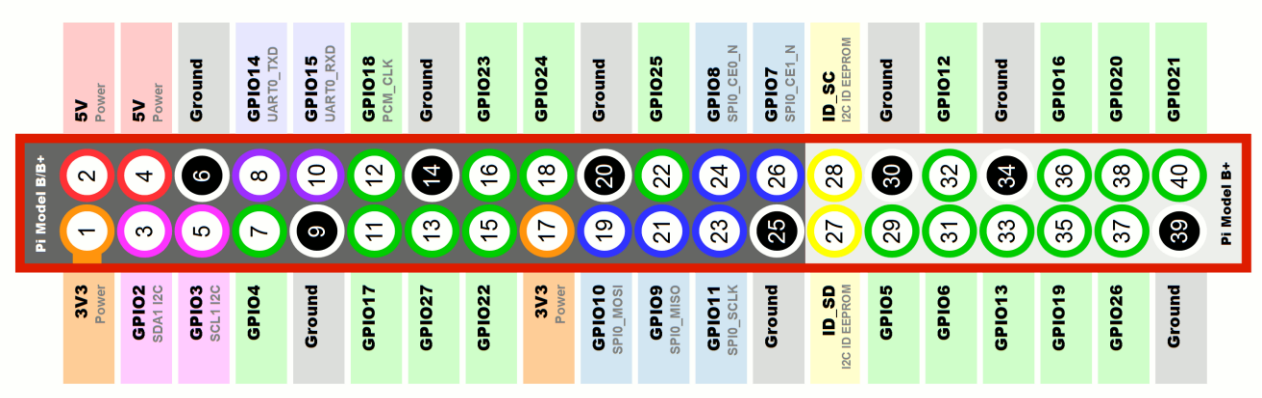
Following code is capturing image frames and saving those images in local memory. Connect Usb Camera with Raspberry-Pi Usb port following code will capture video in 64x64 size in XVID format while images frames size depends upon the resolution of your usb camera by pressing key 'w' you can store images with name Forward{#counter-value}.png and counter will decide the number of image, here counter is used so new frame which is being saved will not overwrite the pervious frame.



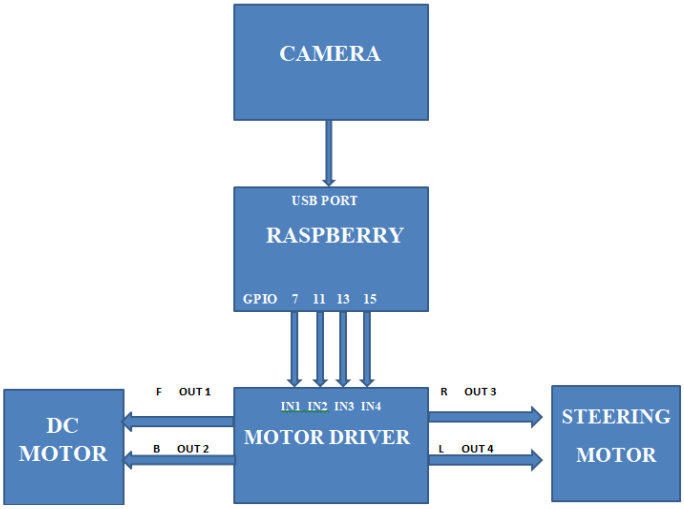

```
import cv2
import numpy as np

cam = cv2.VideoCapture(0)

fourcc = cv2.VideoWriter_fourcc(*'XVID')
vid = cv2.VideoWriter('output.avi', fourcc, 6, (64,64))

counts = 0
countf = 0
```

```
while(True):

   tf, frame = cam.read()
   #print tf
   key = cv2.waitKey(1)
   Raw = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)



   cv2.imshow('Single Frame', Raw)

   if key == 27: #esc key
      break

   elif key == ord('w'):
     Forward = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

      cv2.imshow('Single Frame', Forward)
      cv2.imwrite("forward{}.png".format(countf), Forward)
      countf+=1

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

Running Motors using GPIO pins



Above picture shows the layout of GPIO pins of raspberry pi. Connect pin number 7 with IN1, connect 11,13 and 15 to IN2,IN3 and IN4 respectively to L298N motor driver IC.



In following code we are first defining pins 7,11,13,15 as output using PWM to drive motors, defining functions for every possible move left forward, right forward, forward , stop and backward.

Code:

```
import RPi.GPIO as io

io.setwarnings(False)
io.setmode(io.BOARD)

in1_pin1=7
in2_pin1=11
in1_pin2=13
in2_pin2=15

io.setup(in1_pin1,io.OUT)
p1=io.PWM(in1_pin1,50)
p1.start(0)

io.setup(in2_pin1,io.OUT)
p2=io.PWM(in2_pin1,50)
p2.start(0)

io.setup(in1_pin2,io.OUT)
p3=io.PWM(in1_pin2,50)
p3.start(0)

io.setup(in2_pin2,io.OUT)
p4=io.PWM(in2_pin2,50)
p4.start(0)


def left():
```

```
    p1.start(70)
    p2.start(0)
    p3.start(0)
    p4.start(100)

def right():
    p1.start(60)
    p2.start(0)
    p3.start(100)
    p4.start(0)


def backward():
    p1.start(0)
    p2.start(100)
    p3.start(0)
    p4.start(0)


def forward():
    p1.start(55)
    p2.start(0)
    p3.start(0)
    p4.start(0)
    print('Forwarding')

def stop():
    p1.start(0)
    p2.start(0)
    p3.start(0)
    p4.start(0)
```

Using Py-Game library to control via Keyboard

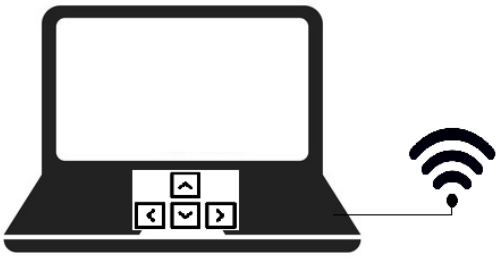
Pygame is used in following code we are using it to send a key to Pi and control motor accordingly. It is a module basically used to make games in python but here we are using this library to control motors in data collection part. You can also use Pynput.keyoard or getch libraries.

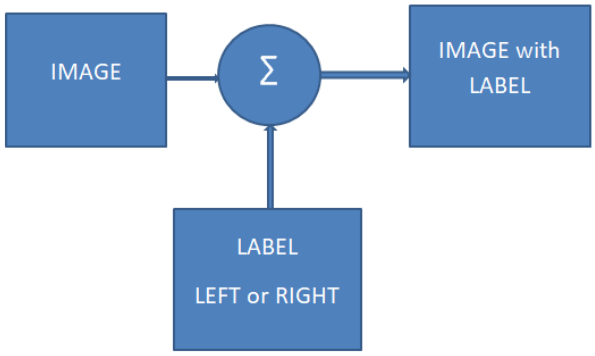
```
init()
screen = display.set_mode((320,40))
display.set_caption('SDC')

endProgram =0
```

```
while(endProgram == 0):


   for e in event.get():
      if e.type == KEYDOWN:
         if(e.key == K_UP):
           forward()


         if(e.key == K_DOWN):
            stop()


         if(e.key == K_RIGHT):
            right()


         if(e.key == K_LEFT):
            left()

         if(e.key == K_ESCAPE):
            endProgram=1

        if(e.key == K_q):
            backward()

         else:
            print(e.key)

quit()
```

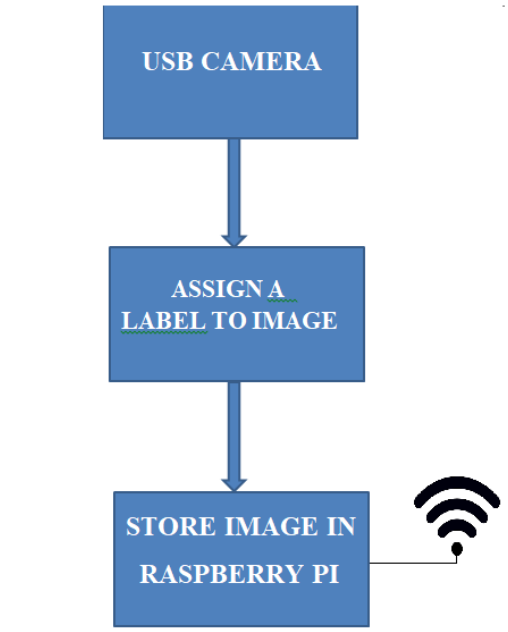
Data Collection:

In data collection part we are using all of the above codes combines them to send key controlling motor and in the mean time we keep on capturing image frames and save it with respect to key pressed. The following code key on checking for keys which are coming from laptop keyboard turn on the flags and according to these flags images are stored and counters are incremented. Here we are using Forward,Left and Right counters named (countf,countl and countr) when any of Up,Left or right key is pressed on of the flag from FC,RC or LC is turned to 1 and other are turned to 0.





LABEL ASSIGNMENT



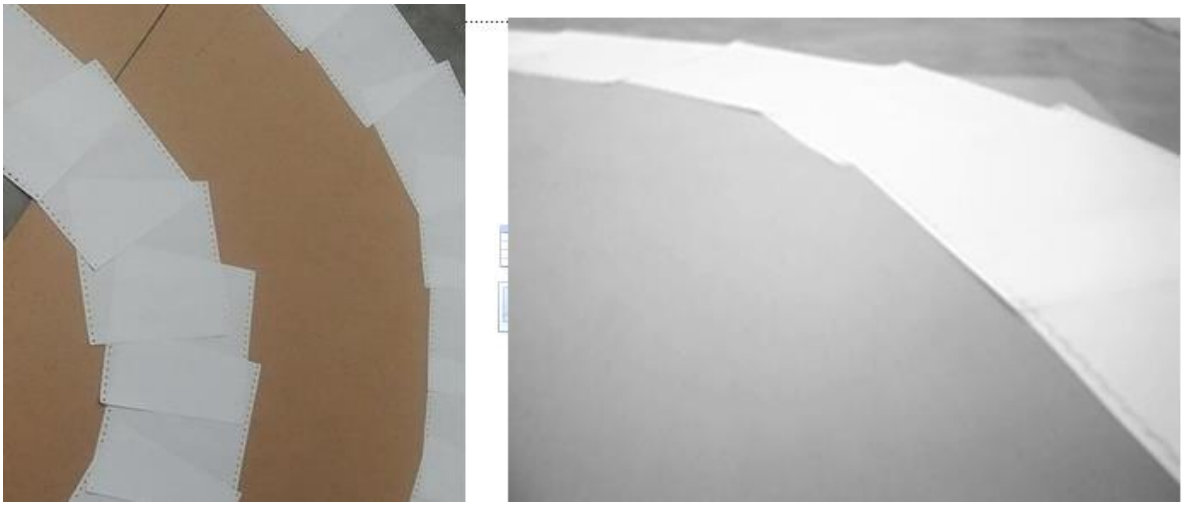
STORING IMAGE IN RASPBERRY WITH LABEL

```
import RPi.GPIO as io
import cv2
from pygame import *

io.setwarnings(False)
io.setmode(io.BOARD)

in1_pin1=7
in2_pin1=11
in1_pin2=13
in2_pin2=15

io.setup(in1_pin1,io.OUT)
p1=io.PWM(in1_pin1,50)
p1.start(0)

io.setup(in2_pin1,io.OUT)
p2=io.PWM(in2_pin1,50)
p2.start(0)

io.setup(in1_pin2,io.OUT)
p3=io.PWM(in1_pin2,50)
p3.start(0)

io.setup(in2_pin2,io.OUT)
p4=io.PWM(in2_pin2,50)
p4.start(0)


def left():
   p1.start(70)
   p2.start(0)
   p3.start(0)
   p4.start(100)

def right():
   p1.start(60)
   p2.start(0)
   p3.start(100)
   p4.start(0)


def backward():
   p1.start(0)
   p2.start(100)
   p3.start(0)
   p4.start(0)
```

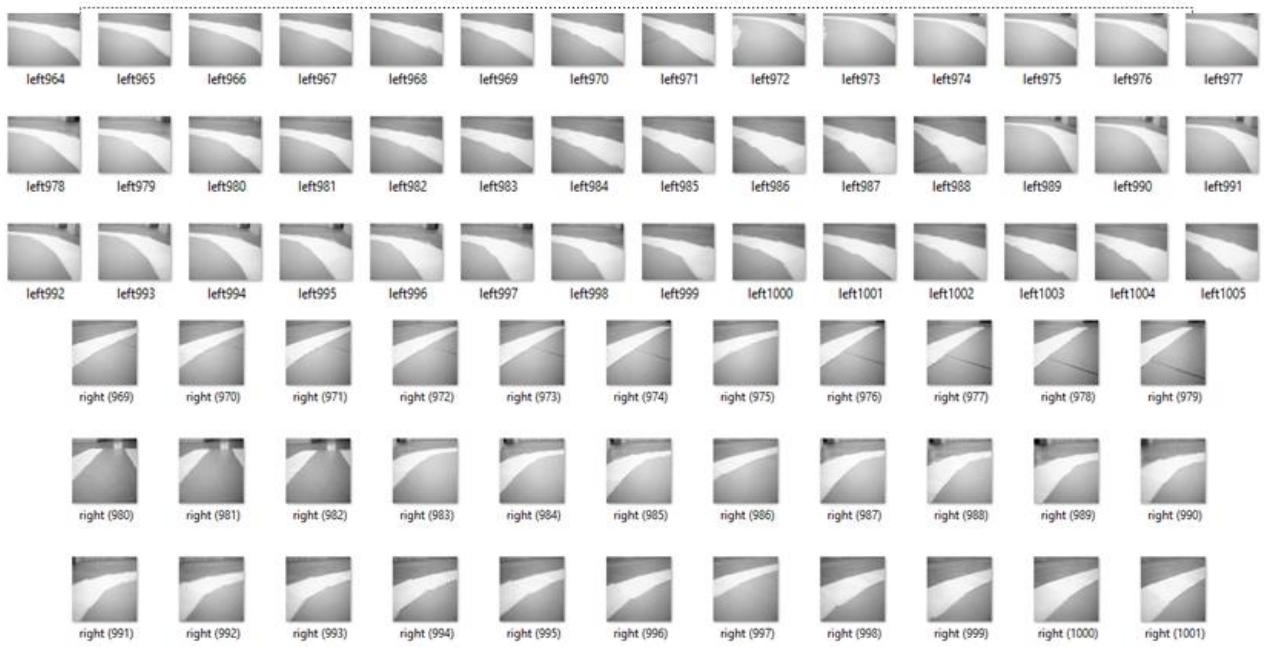
```
def forward():
   p1.start(55)
   p2.start(0)
   p3.start(0)
   p4.start(0)
   print('Forwarding')

def stop():
   p1.start(0)
   p2.start(0)
   p3.start(0)
   p4.start(0)


cam = cv2.VideoCapture(0)

init()
screen = display.set_mode((320,40))
display.set_caption('SDC')

countf = 2452
countr = 0
countl = 1010
FC=0
RC=0
LC=0
endProgram =0

while(endProgram == 0):
   tf, frame = cam.read()
   Forward = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

   for e in event.get():
      if e.type == KEYDOWN:
         if(e.key == K_UP):
            forward()
            FC=1
            RC=0
            LC=0

         if(e.key == K_DOWN):
             stop()
             FC=0
             RC=0
             LC=0

         if(e.key == K_RIGHT):
```

```
            right()
            FC=0
            RC=1
            LC=0

        if(e.key == K_LEFT):
            left()
            FC=0
            LC=1
            RC=0
        if(e.key == K_ESCAPE):
            endProgram=1
        if(e.key == K_q):
            backward()
        else:
            print(e.key)

    if(FC == 1):
        cv2.imwrite("Images/forward{}.png".format(countf), Forward)
        countf+=1
        print('Saving Forward Pic ',countf)
    if(RC == 1):
        cv2.imwrite("Images/right{}.png".format(countr), Forward)
        countr+=1
        print('Saving Right Pic')
    if(LC==1):
        cv2.imwrite("Images/left{}.png".format(countl), Forward)
        countl+=1
        print('Saving Left Pic ', countl)



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

After that get some piece of cardboard and white A4 size build a track with different curves where cardboard shows the path and papers shows the boundaries.



Take almost 1000 samples of every class forward, left and right.



Training

(You can do training by copying image frames to laptop because it is faster than Raspberry-Pi you can still train on Pi but it is slower)

In training we first resize our image because we don't need such a high resolution so we will resize that image to 16*16 and save it.

Create a folder named 'Train' and copy all those image frames in that folder and create Train16X16 to store resize images.

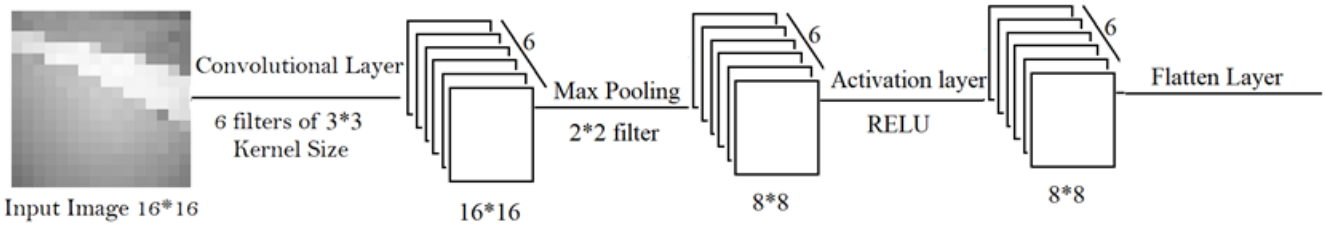
```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
import cv2
import glob

images = glob.glob('Train/*.*')

left = []
right = []
forward = []

i=0
size=16
#feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)

for image in images:
   if 'left' in image:
      imag = cv2.imread(image)
      if(imag==None):
         imagee=image

      else:
         img = cv2.resize(imag, (size,size))
         cv2.imwrite('Train16X16/left'+str(i)+'.jpg', img)
         i=i+1
   elif 'forward' in image :
      imag = cv2.imread(image)
      if(imag==None):
         imagee=image

      else:
         img = cv2.resize(imag, (size,size))
         cv2.imwrite('Train16X16/forward'+str(i)+'.jpg', img)
         i=i+1
   elif 'right' in image :
      imag = cv2.imread(image)
      if(imag==None):
         imagee=image

      else:
         img = cv2.resize(imag, (size,size))
         cv2.imwrite('Train16X16/right'+str(i)+'.jpg', img)
         i=i+1
```

```
print("Done!")
```

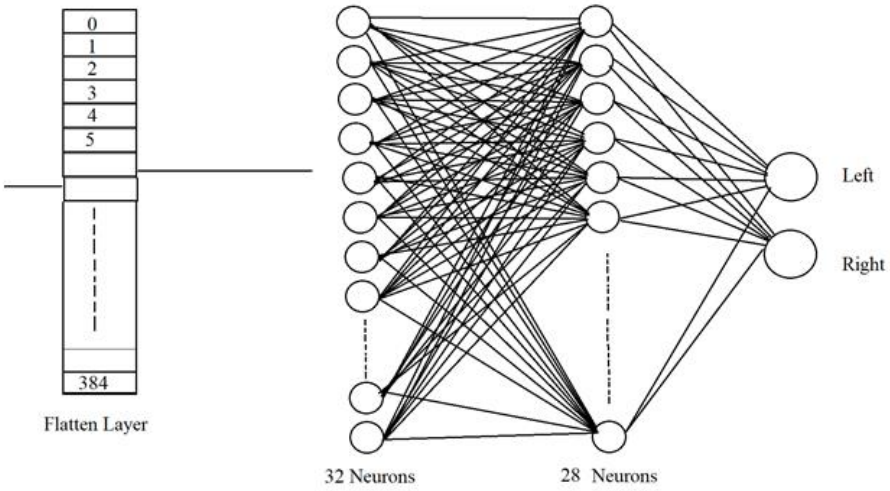
Copy last 100 images of each classes and paste in a new folder named 'Test 16X16'.

Our neural architecture for this problem is as follow:

- Feature Extraction



- Classification



In this picture we have shown only two neurons in last layer but actually there are three neurons Forward, Left and Right.

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, Convolution2D, MaxPooling2D, SpatialDropout2D
from keras.layers.normalization import BatchNormalization
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping
```

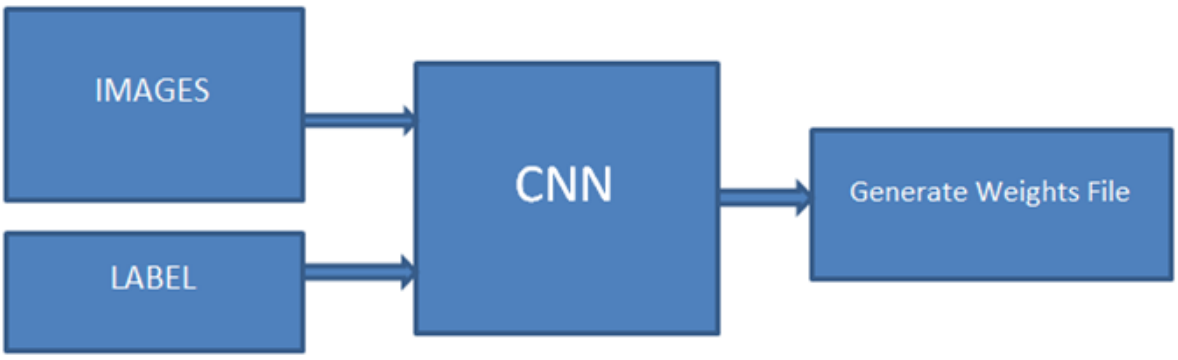
```
import numpy as np
import cv2
import glob
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import scipy
from scipy import ndimage
from scipy.misc import imresize

from sklearn.model_selection import train_test_split

images = glob.glob('Train16X16/*.jpg')

left = []
right = []
forward = []


for image in images:
   if 'left' in image:
      left.append(cv2.imread(image))
   elif 'right' in image:
      right.append(cv2.imread(image))
   elif 'forward' in image:
      forward.append(cv2.imread(image))


print("left: ",len(left))
print("right: ",len(right))
print("forward: ",len(forward))



print("Concatenating Training Classes & Generating Labels...")
X_train = np.concatenate((left,right,forward), axis=0).astype(np.uint8)
y_train = np.concatenate( ( np.zeros(len(left)), np.zeros(len(right))+1,
np.zeros(len(forward))+2),axis=0).astype(np.uint8)

print("Training Data Ready.")
print("Training Data Shape: ",X_train.shape)
print("Training Labels Shape:",y_train.shape)

images = glob.glob('Test16X16/*.jpg')
```

```
left = []
right = []
forward = []


for image in images:
    if 'left' in image:
        left.append(cv2.imread(image))
    elif 'right' in image:
        right.append(cv2.imread(image))
    elif 'forward' in image:
        forward.append(cv2.imread(image))



print("left: ",len(left))
print("right: ",len(right))
print("forward: ",len(forward))




print("Concatenating Testing Classes & Generating Labels...")
X_test = np.concatenate((left,right, forward), axis=0).astype(np.uint8)
y_test = np.concatenate( ( np.zeros(len(left)), np.zeros(len(right))+1,
np.zeros(len(forward))+2),axis=0).astype(np.uint8)

print("Test Data Ready.")
print("Test Data Shape: ",X_test.shape)
print("Test Labels Shape: ",y_test.shape)

from sklearn.utils import shuffle
X_train, y_train = shuffle(X_train, y_train)

print("Data Shuffled!")

def normalize_grayscale(image_data):
    a = -0.5
    b = 0.5
    grayscale_min = 0
    grayscale_max = 255
```

```
    return a + ( ( (image_data - grayscale_min)*(b - a) )/( grayscale_max - grayscale_min ) )

X_normalized = normalize_grayscale(X_train)
print("Data Normalized!")

from sklearn.preprocessing import LabelBinarizer
label_binarizer = LabelBinarizer()
y_one_hot = label_binarizer.fit_transform(y_train)

print("Labels One Hot Encoded!")

# TODO: Build a model
from keras.models import Sequential
from keras.models import Sequential
from keras.layers.core import Dense, Activation, Flatten, Dropout
from keras.layers.convolutional import Convolution2D
from keras.layers.pooling import MaxPooling2D

model = Sequential()

model.add(Convolution2D(6, 3, 3, border_mode="same",  input_shape=(28,28, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Activation('relu'))

model.add(Flatten())


model.add(Dense(32))
model.add(Activation('relu'))

model.add(Dense(28))
model.add(Activation('relu'))

model.add(Dense(4))
model.add(Activation('softmax'))
model.summary()

from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping

# Compiling model with Adam optimizer and learning rate of .0001
#adam = Adam(lr=0.0001)
#model.compile(loss='categorical_crossentropy
sparse_categorical_crossentropy',optimizer=adam)
```

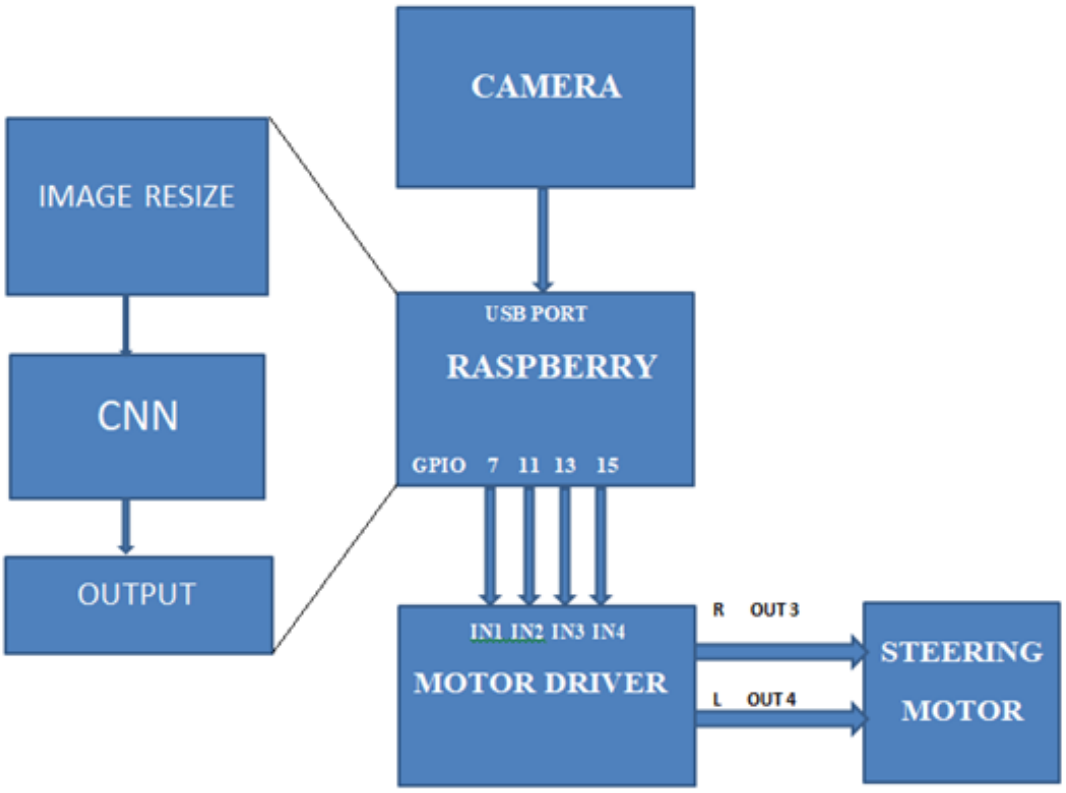
```
model.compile('adam', 'categorical_crossentropy', ['accuracy'])
# Model saves the weights whenever validation loss improves
checkpoint = ModelCheckpoint(filepath = 'model.h5', verbose = 1, save_best_only=True, monitor='val_loss')

# Discontinue training when validation loss fails to decrease
callback = EarlyStopping(monitor='val_loss', patience=3, verbose=1)

# Training model for 10 epochs and a batch size of 128
model.fit(X_normalized,y_one_hot,nb_epoch=50,
verbose=1,batch_size=128,shuffle=True,validation_split=0.05,
        callbacks=[checkpoint,callback])
#history = model.fit(X_normalized, y_one_hot, nb_epoch=20, validation_split=0.05)

print('Model Trained Successfully!')

X_normalized_test = normalize_grayscale(X_test)
y_one_hot_test = label_binarizer.fit_transform(y_test)

metrics = model.evaluate(X_normalized_test, y_one_hot_test)
for metric_i in range(len(model.metrics_names)):
   metric_name = model.metrics_names[metric_i]
   metric_value = metrics[metric_i]
   print('{}: {}'.format(metric_name, metric_value))
```

Model will saved in the same directory named model.h5 that is our weight file which we are going to use for predictions in testing section.



Testing

In testing copy that model.h5 file back to raspberry pi if you have trained your neural network in your laptop. Capture image using usb camera feed frame captured to CNN, it will give you an output, based on that output send controls signal to motor driver and control motors accordingly.




```
import RPi.GPIO as io
import cv2
import numpy as np
from keras.models import load_model
import glob

io.setwarnings(False)
io.setmode(io.BOARD)


in1_pin1=7
in2_pin1=11
in1_pin2=13
in2_pin2=15
```

```
io.setup(in1_pin1,io.OUT)
p1=io.PWM(in1_pin1,50)
p1.start(0)

io.setup(in2_pin1,io.OUT)
p2=io.PWM(in2_pin1,50)
p2.start(0)

io.setup(in1_pin2,io.OUT)
p3=io.PWM(in1_pin2,50)
p3.start(0)

io.setup(in2_pin2,io.OUT)
p4=io.PWM(in2_pin2,50)
p4.start(0)


def left():
   p1.start(70)
   p2.start(0)
   p3.start(0)
   p4.start(100)

def right():
   p1.start(60)
   p2.start(0)
   p3.start(100)
   p4.start(0)


def backward():
   p1.start(0)
   p2.start(100)
   p3.start(0)
   p4.start(0)


def forward():
   p1.start(55)
   p2.start(0)
   p3.start(0)
   p4.start(0)
   print('Forwarding')

def stop():
   p1.start(0)
   p2.start(0)
   p3.start(0)
```

```
    p4.start(0)


cam = cv2.VideoCapture(0)

new_model = load_model('model.h5')
new_model.compile('adam', 'sparse_categorical_crossentropy', ['accuracy'])



while(True):
     img = cv2.imread(img)
    img = cv2.resize(img,(16,16))
    img = np.reshape(img,[1,16,16,3])

    cls = new_model.predict_classes(img)

    if cls==0:
        left()
    elif cls==1:
        right()
    elif cls==2:
        forward()
    if key==27:
       break

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

Implement above code in raspberry pi and get it running.