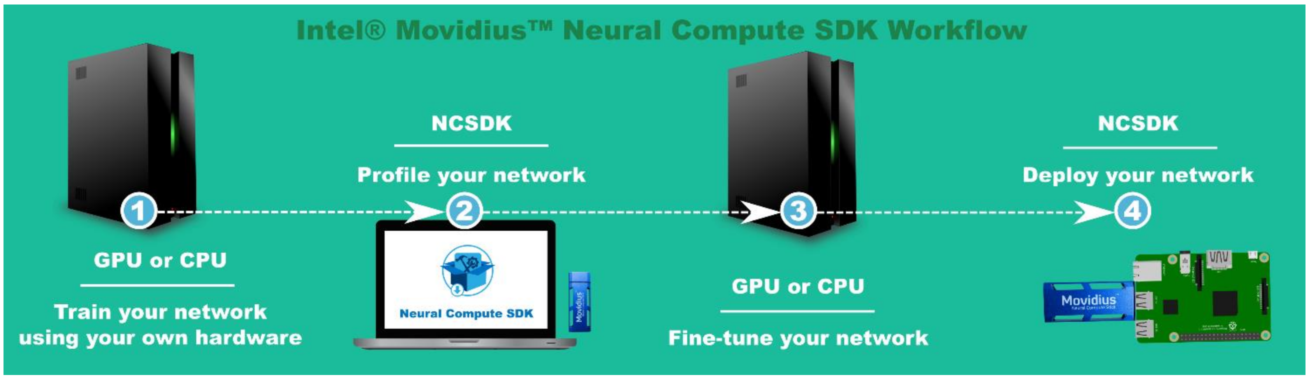
Real Object detection with Neural Computer Stick

Neural Compute stick is a processor or more specifically co-processor which can do neural network network calculations. It is not used to train the model, it is used to process on already deployed neural models. It is mean to run on single board computer (eg. Raspberry Pi), hence power required could be minimum however the processing a neural model would be inappropriate on such single board machine. Hence we train on powerful on computer and use it with rpi for detection.



Below are the materials can be used to as pre requisite to understand the OpenCV and neural compute stick

https://www.pyimagesearch.com/2018/02/12/getting-started-with-the-intel-movidius-neural-compute-stick/

https://www.pyimagesearch.com/2017/10/02/deep-learning-on-the-raspberry-pi-with-opencv/

https://www.pyimagesearch.com/2018/09/26/install-opencv-4-on-your-raspberry-pi/

In our project real Object Detection, we train the model on computer (VirtualBox- Ubuntu) process further on raspberry pi and Movidius NCS.

Hardware Requirement:

Neural Compute Stick (~$75)

Raspberry Pi(~$35)

Male-Female USB wire(~5)
https://www.amazon.com/dp/B00S2N2Q4U/ref=as_li_ss_tl?ie=UTF8&linkCode=sl1&tag=trndingcom-20&linkId=45e7eef30bcbde4b35aa710441038d9a

Pre-Requisite:

Note:Please connect rpi-camera with raspberry pi and make it enabled from raspi-config option

Step 1 : On Raspberry Pi :::Setting up Raspberry pi to upgrade Raspbian OS to "Stretch"

It is very important that the raspberry pi uses latest Raspbian OS "Stretch."

(I tried running in old version of OS and it does not work)

- If raspberry pi is fresh raspberry pi download the Stretch image from https://www.raspberrypi.org/downloads/raspbian/
- Upgrading from old OS to new Stretch OS

This process might take long hours(maybe 5-7 hours).Make sure you complete this successfully as upgraded OS has updated version for OpenCV libraries.

sudo apt-get dist-upgrade

Replacing the jessie keyword with stretch

sudo sed -i /deb/s/jessie/stretch/g /etc/apt/sources.list

sudo sed -I /deb/s/jessie/stretch/g /etc/apt/sources.list.d/*.list

sudo apt-get update

sudo apt-get –simulate upgrade

sudo apt-get upgrade

sudo apt-get dist-upgrade

Restart the raspberry pi to complete installation. Connect keyboard, mouse, and to WIFI connection.

Step 2: On Computer:: Setting up virtual box

Download Ubuntu .iso image as per your machine from http://releases.ubuntu.com/16.04/

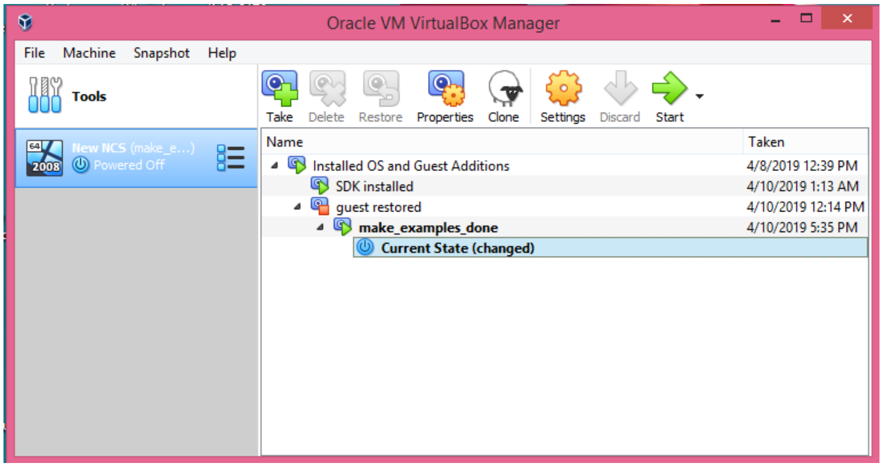
If you don't have virtual box, Download VirtualBox from https://www.virtualbox.org/wiki/Downloads

Follow the wizard prompt to install VirtualBox. It is straight-forward and simple.

We need virtualbox extension pack as we use USB passthrough. Hence download https://www.virtualbox.org/wiki/Downloads

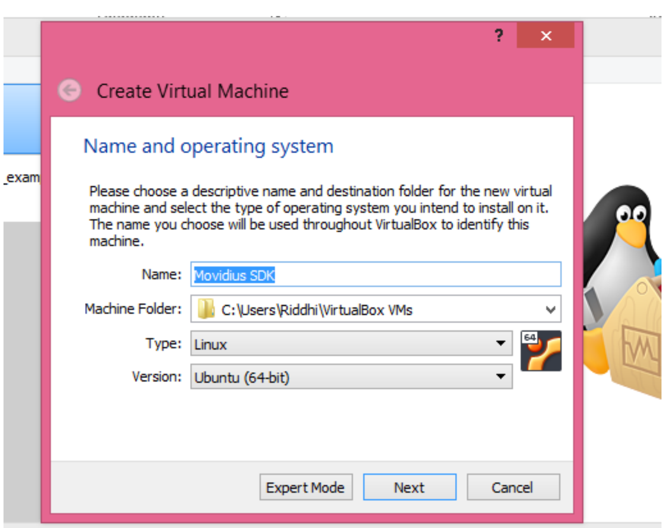
<u>Create VM</u>

Open virtualBox. (For me it shows already one VM which I created for this project)



If no VM installed Go to Tools->Click "New"

Name it "Movidius SDK"

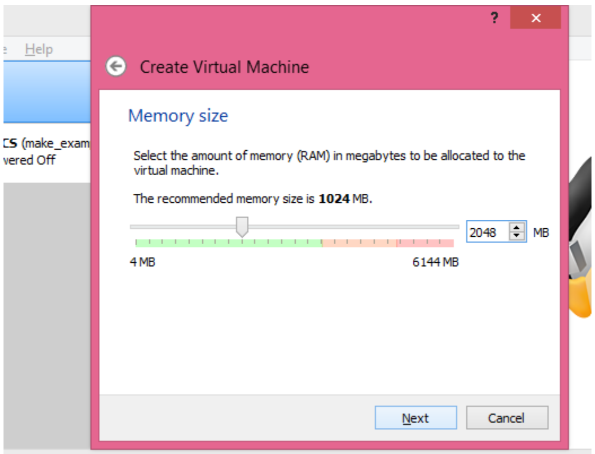


Giving setting for VM

2048MB memory

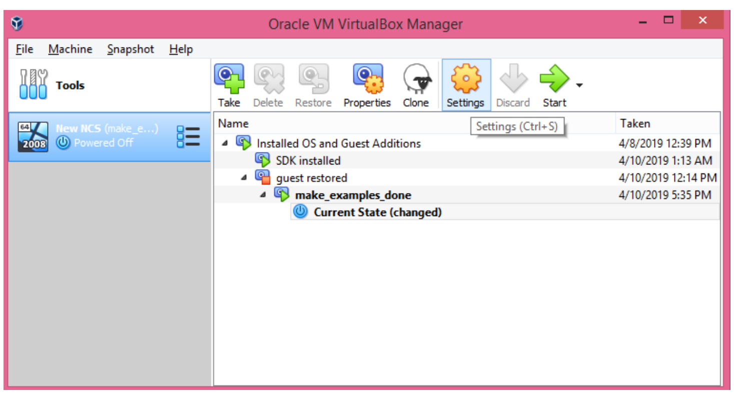
2 virtual CPU

40 GB dynamically allocated VDI

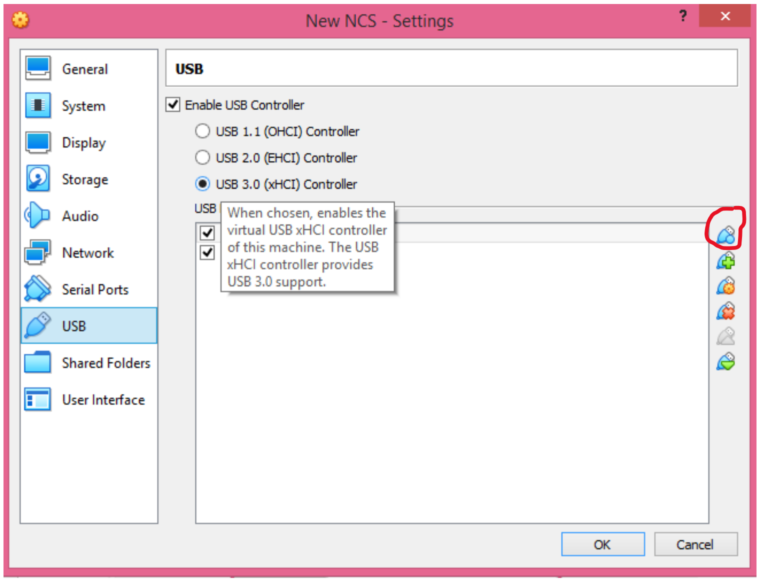


USB passthrough Setting : To access the USB(Neural Compute Stick) through VM-Ubuntu

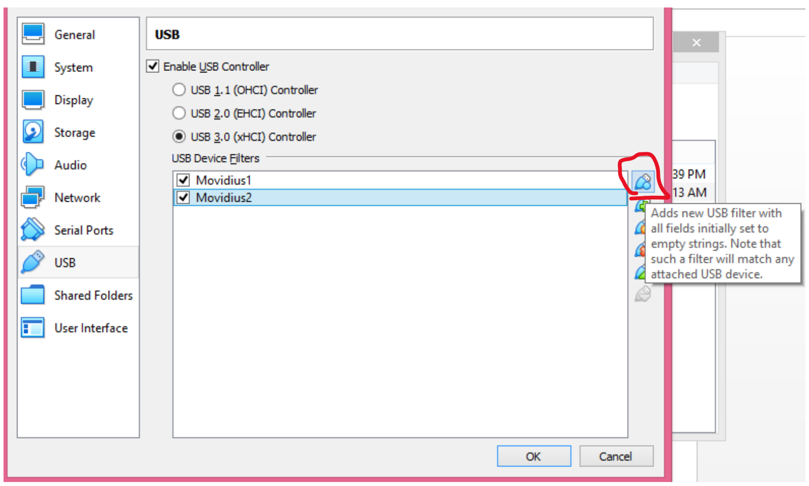
Go to “Setting”



Go to “USB” -> Select USB 3.0 (xHCI) Controller

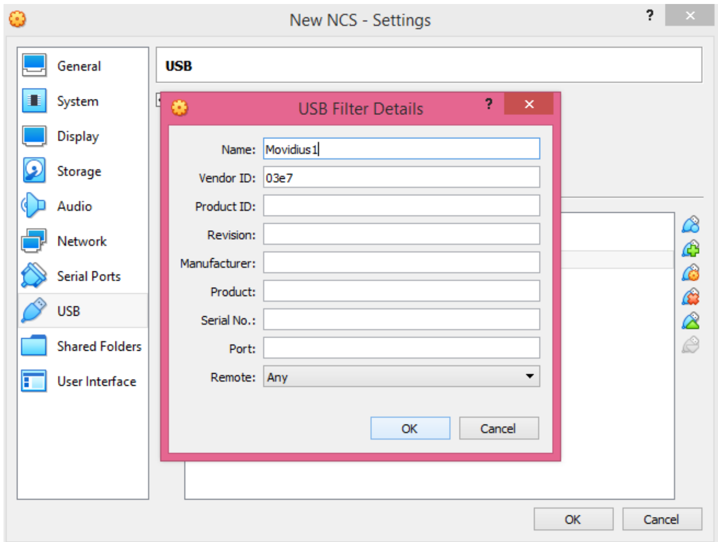


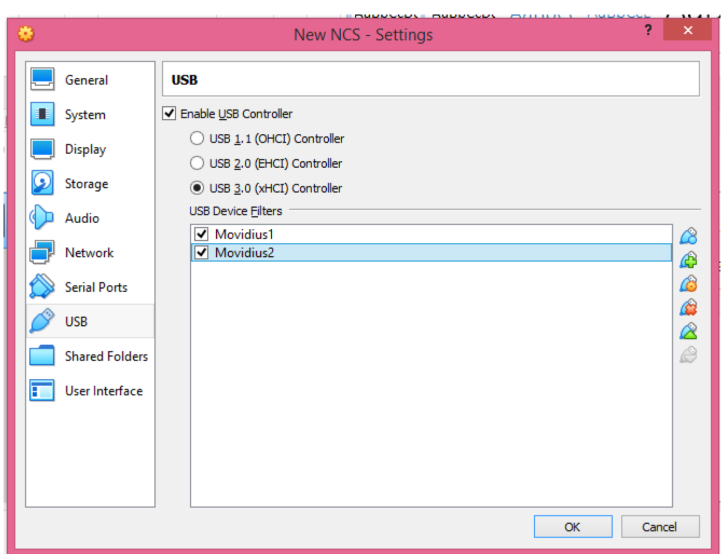
Click right first icon to set filter.



Create two filter for USB and name it as

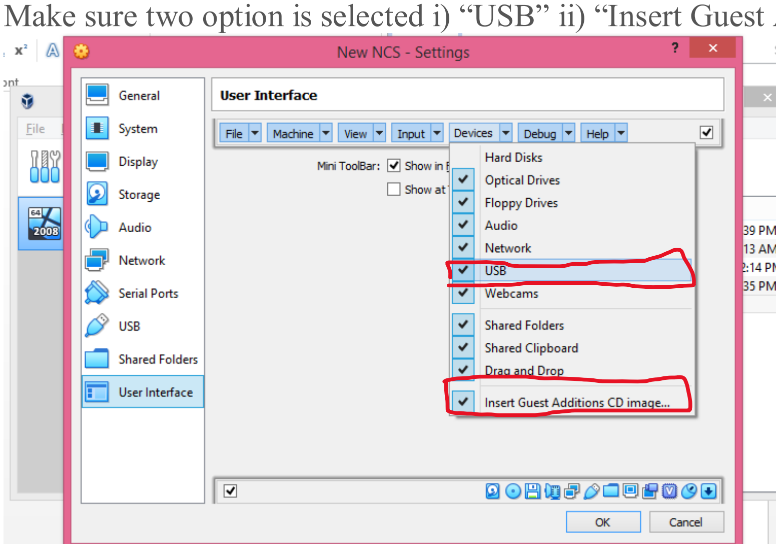
Name: *Movidius1*, **Vendor ID:** *03e7*, **Other fields:** *blank*
Name: *Movidius2*, **Vendor ID:** *040e*, **Other fields:** *blank*



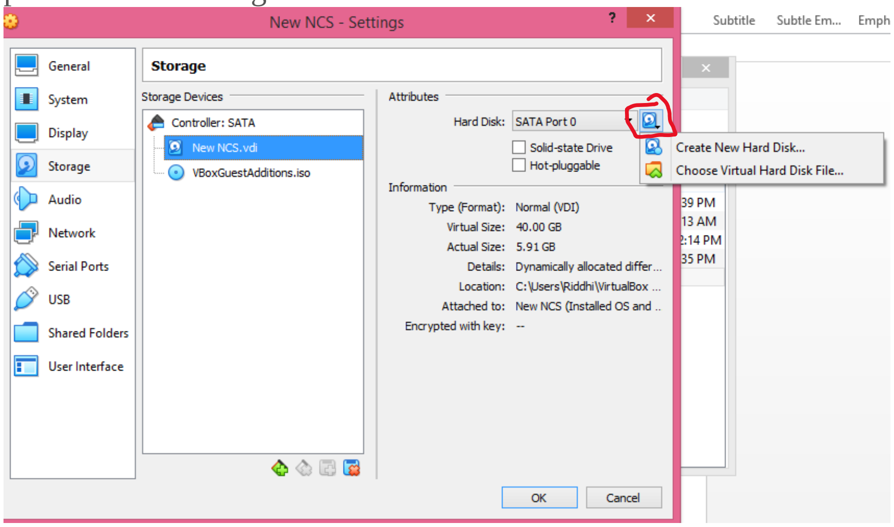


Installing OS on VM

Check User Interface is checked. Go to "Setting" -> "User Interface" - > "Devices" Make sure two option is selected i) "USB" ii) "Insert Guest Addition CD image"



Now to install, Go to “Setting” -> Storage -> Click the small disk shape icon -> Give path to .iso file image



Click “Ok”

Follow the prompt steps to install ubuntu. (You may be ask to erase disk and install ubuntu, you can select Ok as we just created memory for installation!)

After installation of Ubuntu -> Open “terminal” and type

sudo apt-get update && sudo apt-get upgrade

Step3: On computer:: Installing movidius SDK on Ubuntu

We are using the installed trained graphed for our experiment. To use it we will clone the git repository. Hence, first we install git

sudo apt-get install git

Now lets make directory for projectspace.

cd ~

mkdir project

cd project

Clone the repository here

git clone https://github.com/movidius/ncsdk.git

git clone https://githib.com/movidius/ncappzoo.git

cd ~/project/ncsdk

make install

On enter-ing the above command starts the install. It might take around 15-20 minutes.

Step4: On Computer:::Connect Neural compute stick with computer to check connectivity.

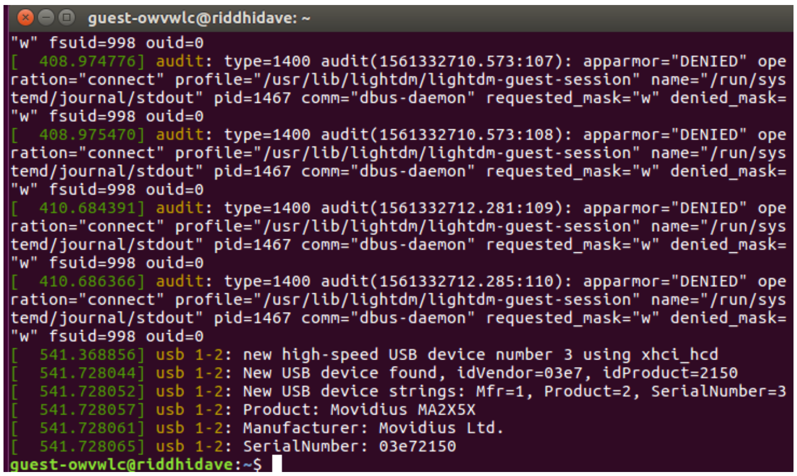
Connect Neural Compute with female-male wire and connect it to computer.

Go to VMbox -> Settings -> User Interface -> Devices check USB if it is not checked.

Now go to terminal on ubuntu

Type

dmesg



It would show output similar to image above.

Step5: On Computer::Generating graph from caffe moel

Graph is created using tool mvNCCompile. This will help to get graph file more quickly. We are working with trained graphs by chuanqui305(website in reference).

Go to terminal and type:

```
mvNCCompile models/MobileNetSSD_deploy.prototxt \
    -w models/MobileNetSSD_deploy.caffemodel \
    -s 12 -is 300 300 -o graphs/mobilenetgraph
```

Detailed description on the arguments on https://www.pyimagesearch.com/2018/02/19/real-time-object-detection-on-the-raspberry-pi-with-the-movidius-ncs/

Step6::On Computer::Object detection with Neural Compute Stick

Writing objection detection python script.

```
import the necessary packages

from mvnc import mvncapi as mvnc
from imutils.video import VideoStream
from imutils.video import FPS
import argparse
import numpy as np
import time
```

```
import cv2
# initialize the list of class labels our network was trained to
# detect, then generate a set of bounding box colors for each class
CLASSES = ["background", "aeroplane", "bicycle", "bird",
	"boat", "bottle", "bus", "car", "cat", "chair", "cow",
	"diningtable", "dog", "horse", "motorbike", "person",
	"pottedplant", "sheep", "sofa", "train", "tvmonitor"]
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))

# frame dimensions should be sqaure
PREPROCESS_DIMS = (300, 300)
DISPLAY_DIMS = (900, 900)

# calculate the multiplier needed to scale the bounding boxes
DISP_MULTIPLIER = DISPLAY_DIMS[0] // PREPROCESS_DIMS[0]
def preprocess_image(input_image):
	# preprocess the image
	preprocessed = cv2.resize(input_image, PREPROCESS_DIMS)
	preprocessed = preprocessed - 127.5
	preprocessed = preprocessed * 0.007843
	preprocessed = preprocessed.astype(np.float16)

	# return the image to the calling function
	return preprocessed
def predict(image, graph):
	# preprocess the image
	image = preprocess_image(image)

	# send the image to the NCS and run a forward pass to grab the
	# network predictions
	graph.LoadTensor(image, None)
	(output, _) = graph.GetResult()

	# grab the number of valid object predictions from the output,
	# then initialize the list of predictions
	num_valid_boxes = output[0]
	predictions = []
	# loop over results
	for box_index in range(num_valid_boxes):
		# calculate the base index into our array so we can extract
		# bounding box information
		base_index = 7 + box_index * 7

		# boxes with non-finite (inf, nan, etc) numbers must be ignored
		if (not np.isfinite(output[base_index]) or
			not np.isfinite(output[base_index + 1]) or
			not np.isfinite(output[base_index + 2]) or
			not np.isfinite(output[base_index + 3]) or
			not np.isfinite(output[base_index + 4]) or
```

```
                not np.isfinite(output[base_index + 5]) or
                not np.isfinite(output[base_index + 6])):
                continue

            # extract the image width and height and clip the boxes to the
            # image size in case network returns boxes outside of the image
            # boundaries
            (h, w) = image.shape[:2]
            x1 = max(0, int(output[base_index + 3] * w))
            y1 = max(0, int(output[base_index + 4] * h))
            x2 = min(w,    int(output[base_index + 5] * w))
            y2 = min(h,    int(output[base_index + 6] * h))

            # grab the prediction class label, confidence (i.e., probability),
            # and bounding box (x, y)-coordinates
            pred_class = int(output[base_index + 1])
            pred_conf = output[base_index + 2]
            pred_boxpts = ((x1, y1), (x2, y2))

            # create prediciton tuple and append the prediction to the
            # predictions list
            prediction = (pred_class, pred_conf, pred_boxpts)
            predictions.append(prediction)

    # return the list of predictions to the calling function
    return predictions
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-g", "--graph", required=True,
    help="path to input graph file")
ap.add_argument("-c", "--confidence", default=.5,
    help="confidence threshold")
ap.add_argument("-d", "--display", type=int, default=0,
    help="switch to display image on screen")
args = vars(ap.parse_args())
# grab a list of all NCS devices plugged in to USB
print("[INFO] finding NCS devices...")
devices = mvnc.EnumerateDevices()

# if no devices found, exit the script
if len(devices) == 0:
    print("[INFO] No devices found. Please plug in a NCS")
    quit()

# use the first device since this is a simple test script
# (you'll want to modify this is using multiple NCS devices)
print("[INFO] found {} devices. device0 will be used. "
    "opening device0...".format(len(devices)))
device = mvnc.Device(devices[0])
```

```
device.OpenDevice()

# open the CNN graph file
print("[INFO] loading the graph file into RPi memory...")
with open(args["graph"], mode="rb") as f:
	graph_in_memory = f.read()

# load the graph into the NCS
print("[INFO] allocating the graph on the NCS...")
graph = device.AllocateGraph(graph_in_memory)
# open a pointer to the video stream thread and allow the buffer to
# start to fill, then start the FPS counter
print("[INFO] starting the video stream and FPS counter...")
vs = VideoStream(usePiCamera=True).start()
time.sleep(1)
fps = FPS().start()
# loop over frames from the video file stream
while True:
	try:
		# grab the frame from the threaded video stream
		# make a copy of the frame and resize it for display/video purposes
		frame = vs.read()
		image_for_result = frame.copy()
		image_for_result = cv2.resize(image_for_result, DISPLAY_DIMS)

		# use the NCS to acquire predictions
		predictions = predict(frame, graph)
		# loop over our predictions
		for (i, pred) in enumerate(predictions):
			# extract prediction data for readability
			(pred_class, pred_conf, pred_boxpts) = pred

			# filter out weak detections by ensuring the `confidence`
			# is greater than the minimum confidence
			if pred_conf > args["confidence"]:
				# print prediction to terminal
				print("[INFO] Prediction #{}: class={}, confidence={}, "
					"boxpoints={}".format(i, CLASSES[pred_class],
pred_conf,
					pred_boxpts))
				# check if we should show the prediction data
				# on the frame
				if args["display"] > 0:
					# build a label consisting of the predicted class and
					# associated probability
					label = "{}: {:.2f}%".format(CLASSES[pred_class],
						pred_conf * 100)

					# extract information from the prediction boxpoints
```

```
                    (ptA, ptB) = (pred_boxpts[0], pred_boxpts[1])
                    ptA = (ptA[0] * DISP_MULTIPLIER, ptA[1] *
DISP_MULTIPLIER)
                    ptB = (ptB[0] * DISP_MULTIPLIER, ptB[1] *
DISP_MULTIPLIER)
                    (startX, startY) = (ptA[0], ptA[1])
                    y = startY - 15 if startY - 15 > 15 else startY + 15

                    # display the rectangle and label text
                    cv2.rectangle(image_for_result, ptA, ptB,
                        COLORS[pred_class], 2)
                    cv2.putText(image_for_result, label, (startX, y),
                        cv2.FONT_HERSHEY_SIMPLEX, 1,
COLORS[pred_class], 3)
        # check if we should display the frame on the screen
        # with prediction data (you can achieve faster FPS if you
        # do not output to the screen)
        if args["display"] > 0:
            # display the frame to the screen
            cv2.imshow("Output", image_for_result)
            key = cv2.waitKey(1) & 0xFF

            # if the `q` key was pressed, break from the loop
            if key == ord("q"):
                break

        # update the FPS counter
        fps.update()

    # if "ctrl+c" is pressed in the terminal, break from the loop
    except KeyboardInterrupt:
        break

    # if there's a problem reading a frame, break gracefully
    except AttributeError:
        break
# stop the FPS counter timer
fps.stop()

# destroy all windows if we are displaying them
if args["display"] > 0:
    cv2.destroyAllWindows()

# stop the video stream
vs.stop()

# clean up the graph and device
graph.DeallocateGraph()
device.CloseDevice()
```

```
# display FPS information
print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))
```

Step7::On RaspberryPi::Movidius NCS + RPi

```
python real_object_detectio.py --graph graph --display 1
```

Congratulation! On successfully running above code it would detect person,background, table and more.