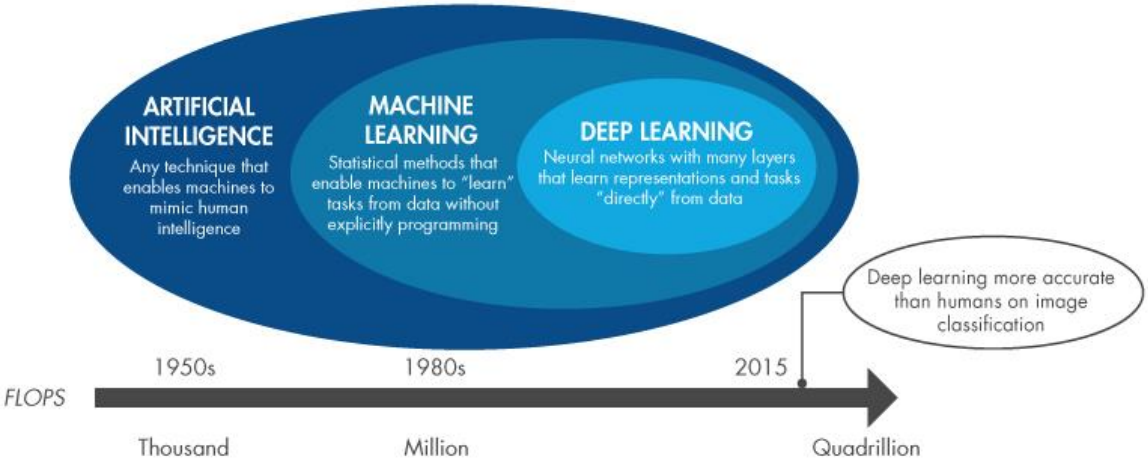
Tutorial 8: How to use a pre-trained (Deep Learning) object detection model

Today, we are learning how to use a pre-trained deep learning model to detect complex objects. You already learned how to detect objects using low-level image features. However, low-level features are not sufficient to understand complex objects and scenes. We use high-level features generated by a trained neural network and detect objects. The image classifier you learned in Tutorial 7 was based on HOG features. The deep learning model you are learning today was generated following a relatively similar approach. Your Tutorial 7 classification model belongs to Machine Learning. This object detection model belongs to Deep Learning which is a sub-domain of Machine Learning.

Let's start by learning some terminologies. These were taken from MathWorks Deep Learning eBook (https://explore.mathworks.com/machine-learning-vs-deep-learning/chapter-1-129M-100NU.html)



AI: Artificial intelligence (AI) is a computer system trained to perceive its environment, make decisions, and take action. AI systems rely on learning algorithms, such as machine learning and deep learning, along with large sets of sensor data with well-defined representations of objective truth.

Machine learning: We use *machine learning* as shorthand for "traditional machine learning"—the workflow in which you manually select features and then train the model. When we refer to *machine learning* we exclude deep learning. Common machine learning techniques include decision trees, support vector machines, and ensemble methods.

Deep learning: A subset of machine learning modelled loosely on the neural pathways of the human brain. *Deep* refers to the multiple layers between the input and output layers. In deep learning, the algorithm automatically learns what features are useful. Common deep learning techniques include convolutional neural networks (CNNs), recurrent neural networks (such as long short-term memory, or LSTM), and deep Q networks.

Algorithm: The set of rules or instructions that will train the model to do what you want it to do.

Model (or network): The trained program that predicts outputs given a set of inputs.

If you are interested in learning Deep Learning, there are plenty of free resources available. This short tutorial is not enough to cover everything in Deep Learning. The following Forbes article lists 6

free online courses available today. These well-explained courses are offered by the experts in the field.

https://www.forbes.com/sites/bernardmarr/2018/04/16/the-6-best-free-online-artificial-intelligence-courses-for-2018/#307b6dad59d7

Today tutorial is based-on the blog-post named "OpenVINO, OpenCV, and Movidius NCS on the Raspberry Pi" (https://www.pyimagesearch.com/2019/04/08/openvino-opencv-and-movidius-ncs-on-the-raspberry-pi/).

This blog-post was published by a reputed scientist in the computer vision field. This is a well explained tutorial. There are links given to related topics inside the tutorial. You are advised to check those links and understand the fundamentals.

Hardware needed

- Raspberry Pi 3B+ (or Raspberry Pi 3B)
- Movidius NCS 1
- A short USB extension cable (NCS is bulky and can damage RPi USB ports if connected directly)
- PiCamera V2 (or USB webcam)
- 32GB microSD card with Raspbian Stretch freshly flashed (16GB would likely work as well)
- HDMI screen + keyboard/mouse (at least for the initial WiFi configuration)

Intel Movidius Vision Processing Unit (VPU)

There are two versions of Intel Neural Compute Sticks (NCS):

- Intel Movidius Neural Compute Stick
- Intel Neural Compute Stick 2

We use Intel Movidius NCS.

This unit enables visual intelligence at a high compute per watt. It supports camera processing, computer vision, and deep learning inferences.

Common uses:

- Optimizing imaging, computer vision, and neural network pipelines
- Delivering high-performance, on-device deep learning inferences
- Furnishing data flow for machine intelligence workloads
- Supplying low power situations such as smart cameras or small compute devices

Works best for:

- Autonomous service robots and drones
- Digital security and surveillance
- Smart home and intelligent wearables
- Augmented reality (AR) and virtual reality (VR) all-in-one headsets

Supported operating systems:

- Ubuntu 16.04.3 TLS (64 bit)
- CentOS 7.4 (64 bit)
- Windows 10 (64 bit)
- Raspbian (target only)

The OpenVINO (Open Visual Inference and Neural network Optimization) toolkit

- Intel's OpenVINO is an acceleration library for optimized computing with Intel's hardware portfolio.
- OpenVINO supports Intel CPUs, GPUs, FPGAs, and VPUs.
- Deep learning libraries such as TensorFlow, Caffe, and mxnet are supported by OpenVINO.
- The OpenVINO documentation available here is provided to you.

OpenCV library

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

Important !

- You are advised to use a fresh SD card (32GB) for the installation.
- In the reference blog-post, the Raspbian Stretch OS is downloaded from its original source. You should download the Raspbian Stretch OS from this Robotis link (this image comes with all packages installed for TurtleBot):

http://emanual.robotis.com/docs/en/platform/turtlebot3/raspberry_pi_3_setup/#install-linux-based-on-raspbian

- Follow the Robotis SBC setup instructions and modify the .bashrc file by including your TurtleBot model.
- Next, follow the reference blog-post and install OpenVINO toolkit. This might take few hours (assuming you read all instructions/comments/taking notes).
- You can ssh to Pi from Windows command line. This way, you can easily copy-paste the commands to your command window (Ctrl+C copies the text, Right Click on command line pastes the text).
- If you get an error, check your command (spelling) again or Google it. This code was already tested on the specified setting.

Possible installation errors:

Error:
```
$ pip install “picamera[array]”
```
This command is not working.

Solution: Note the difference in double quotation marks. Run the following command.
```
sudo pip install "picamera[array]"
```

Error:
```
fatal: could not create work tree dir 'ncappzoo': No space left on device
```

Solution:
Reboot the Pi. Maybe you forgot to reboot after memory expansion.

How to run the code?

In the reference blog-post, the python packages are installed in a virtual environment. You can create any number of virtual environments and install the required versions of packages. For example, let’s say you have program A and program B, and program A depends of package C version 1 and program B depends on package C version 2. You can create two virtual environments for programs A and B, and install the required versions of package C separately.

- The MobileNet-SSD files and python files used in the blog-post are given to you. The easiest way to get those files is by direct downloading from your learn online website. You can connect your TurtleBot to UNISA WI-FI and download the files.

- First, you have to activate your openvino virtual environment.

```
pi@raspberrypi:~/Downloads/openvino-pi-object-detection $ workon openvino
```

- Then, using ‘cd’ command, go to the folder where the trained Caffe model is saved. In the show example, it was saved in ‘Downloads’ folder. Run the object detector.

```
(openvino) pi@raspberrypi:~/Downloads/openvino-pi-object-detection $ python openvino_real_time_object_detection.py --prototxt
```

```
MobileNetSSD_deploy.prototxt -m/ --model
MobileNetSSD_deploy.caffemodel
```

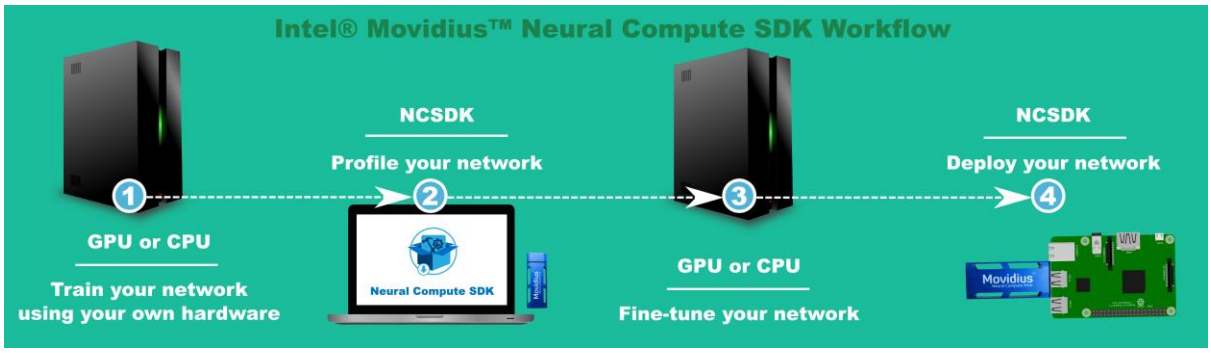
- To stop, press 'Q'. Then you can see the speed – frames per second (FPS) value.
- If you are using a USB camera instead of Pi camera, you have to modify the following line

```
vs = VideoStream(usePiCamera=True).start()
```
to
```
vs = VideoStream(src=0).start()
```

If you are using MobileNet SSD model for your project, you are required to DO/know the followings:

- You are free to use any pre-trained network including MobileNet-SSD. However, using a pre-trained network should not be a black-box approach. You should understand the theory behind it. Following questions will help you to better understand this approach.
- What is the typical structure of a Convolutional Neural Network?
- Understand how to train/test deep neural network.
- What is classification/object detection?
- How can you use a pre-trained network as a classifier or detector?
- What are the popular pre-trained neural networks?
- What are the popular datasets?
- What are the Deep Learning applications in robotics?
- How to use a pre-trained models?
- What is MobileNet SSD?
- Popular object detectors are YOLO, RCNN and MobileNet SSD. Compare their speeds, architecture.
- You should understand all the steps given in the provided blog-post link.

These are the steps involve in creating an object detector:

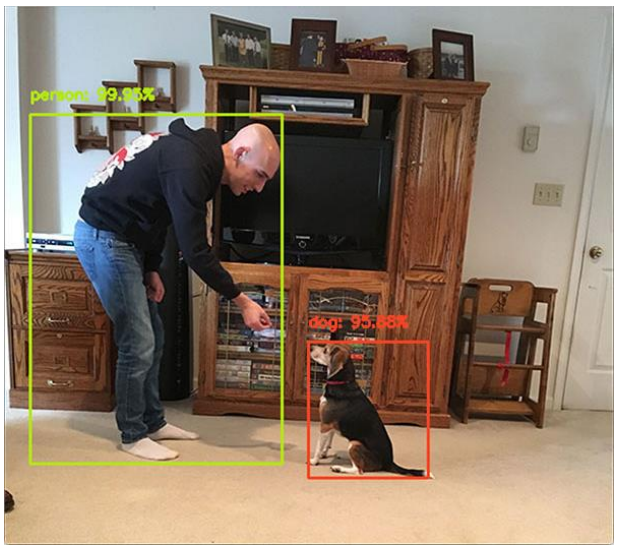


1. Train - Neural network selection, dataset preparation, and training
2. Profile - Analyze the neural network for bandwidth, complexity, and execution time
3. Fine tune - Modify the neural network topology to gain better execution time
4. Deploy - Deploy the customized neural network on an edge device powered by NCS

When we use a pre-trained network we skip above first three steps and execute only the last step.

These are the classes the MobileNet SSD can detect:

"background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow", "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"



Object detection example with MobileNet-SSD (Image source: PyImageSearch).

After the detector is working…

- The detector provides the object class and the bounding box.
- You should create a ROS publisher to publish the class and bounding box.
- Use MATLAB to extract that information and use them to simulate a practical scenario.
- Possible use cases (ideas for your project) are
  - Detect a person and follow him/her using the color thresholding (e.g. person wears a red color shirt). There are other red color objects in the background, and robot should track/follow only the person. This can be combined with lidar obstacle avoidance.
  - Give commands to the robot. Detect multiple people. Out of them, detect the commander using the blue color hat/vest he/she is wearing. Then, use the color gloves/signs for commands. After completing the command, rotate around, find and face toward the commander for next command.
  - Create a traffic scenario using toy cars, busses, bicycles, motorbikes, trains, persons etc. Use a toy car/bus (remote controlled) to track/follow. This can be combined with obstacle avoidance with vision (for shorter objects not detected by lidar).
- You can connect NCS to your Virtual Machine and do the same. Compare the speeds.

If you are planning to use NCS for your project, in addition to above project ideas you can find some relatively similar ideas on Robot Software Architectures website at UNSW. They also use TurtleBots.

https://webcms3.cse.unsw.edu.au/COMP3431/18s2/resources/19946