AI based PPE detector

ELE5MPA - Project report

Fernando J. Galetto
Student ID: 19158643
Project supervisor: Dennis Deng

Department of Engineering
La Trobe University
Melbourne - Australia
Jun 2019

Abstract

The purpose of this project is to create an application to demonstrate how an Intel Neural Compute Stick can accelerate inference on edge devices such as raspberry pi using OpenVINO toolkit and following Intel's workflow.

Constructions sites are busy places and can constitute a very dangerous environment when many contractors are working at the same time, especially when they are not using appropriate personal protective equipment (PPE), the system described in this document detects whether workers are wearing appropriate PPEs or not running Tiny-YOLOv3 on a raspberry Pi with an Intel Neural Computer Stick 2 to accelerate the inference process.

Contents

Part I
Theory

1 Introduction

1.1 The problem to solve...

OHS is a very important subject in Australia, government agencies such as Work Safe have the responsibility to improve work health and safety while the government oversees the enforcement and creation of laws to guarantee the health and safety of workers.

Construction sites are very busy environments where sometimes many contractors work at the same time, coordination and planning is essential to create a safe workplace.

Personal protective equipment (PPE) is equipment designed to protect workers from injuries or illness, PPEs are required as a last resort where there is no other way to control the risk [1], so when they are required its very important to wear them.

Sometimes workers don't use their required PPEs for different reasons:

- They are uncomfortable
- Unattractive
- Decreases productivity
- They weren't provided.
- They forgot them
- They don't know they are required.

Control the use of PPE is a stressful and a full-time job that can be automatized using AI, this project is about developing a system that can detect people in construction sites that are not wearing PPE and notify their supervisor or control entities. As a first approach, only hard hat and high visibility vest is detected but the system has been thought with the possibility to add new element such as hard boots or safety glasses if needed.

1.2 AI at the edge

Artificial intelligence computing nowadays is done mostly on cloud-based datacentres such as Google cloud or Amazon Web Services (AWS) but recently a new trend has arisen, AI on Edge devices have captive the attention of the giants Google INC and INTEL, AI on edge devices allow the developers to create products that doesn't necessarily connect to a network keeping the data locally with a relatively low cost [2].

Machine learning development is done in two phases, first the model is created and trained (this process requires huge computational power) and the second and last phase is Inference, where the pre-trained network is implemented. AI on Edge leaves the inference phase to the EDGE devices, for this low power, small

form factor and relatively low-cost hardware accelerators such as Intel Movidius Neural compute Stick (NCS2) or Google CORAL have been introduced, these processors are designed and optimized to run neural network inference [2, 3].

2 Object detection with YOLO

2.1 Overview

YOLO stands for "You Only Look Once", it is considered for many authors a state-of-the-art object detection technique, it allows to detect multiple bounding boxes and object probabilities with a single convolutional neural network [4]. YOLO it is based in the fact that humans can identify objects and its positions in an image instantly just looking at it once [4], it was first introduced in 2016 as a single stage method for object detection [5] and it has become very popular since then, different applications such as censing animal poplation [6], detection and dlassification of the Breast Abnormalities on medical images [7], real time forest fire detection [8] or Automatic License Plate Recognition [9] demonstrated it effectiveness and reliability.

There are 3 YOLO version published by Redmon et all, YoloV2 can support more classes (up to 9000), it is faster and stronger than YOLOv1 [10], YOLOv3 has a slight improvement on accuracy over YOLOv2 specially on small object detection [11], also each YOLO version has its faster version called Tiny-YOLO which consists in a smaller network that performs faster than the original by scarifying accuracy, on this project tiny-YOLOv3 was used because the target hardware has considerable processing limitations.

Yolo has been designed using darknet, Darknet is an open source framework to create neural networks in C [12], since YOLO has become very popular there are many open source repositories to implement YOLO on TensorFlow [13] or caffe [14].

2.2 How it works...

YOLO divides the input image into an SxS grid cell, each cell is responsible for detecting an object cantered in that cell, each cell can predict a certain number B of boxes called anchor boxes, this boxes have 4 normalized parameter associated with its coordenates x, y, h, w where x and y are the coordenates for the center of the box relatives to the center of the cell, h and w are height and width of the box relative to the whole image [4].

The following 447x447 image (figure 1) illustrates a 3x3 grid cell where the center cell detects an object.

In addition to those 4 parameters per bounding box the network gives another value from 0 to 1 that belong to the confidence prediction (Pr(Object)) which indicates the presence of an object of any class [4].

Each grid cell also predicts a number C of conditional class probabilities, they are called conditional because they are only valid if the cell contains an

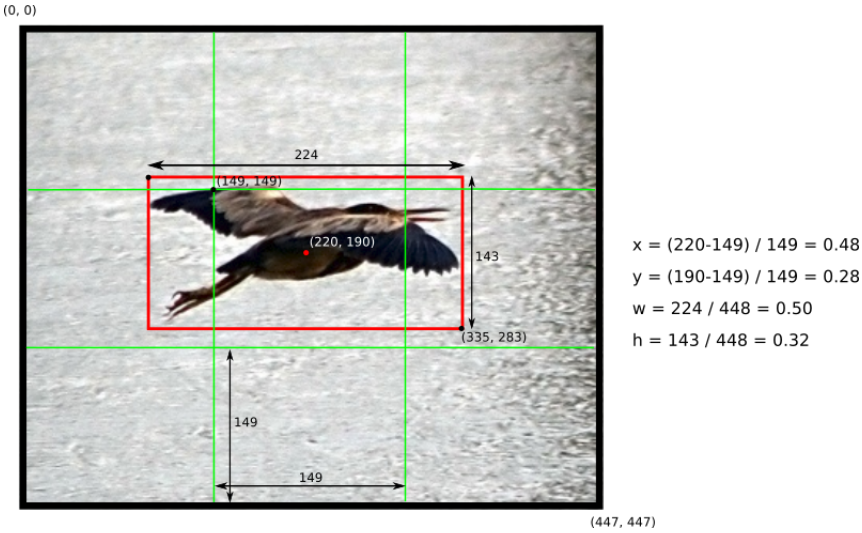


Figure 1: YOLO 3x3 grid cell example

object. To obtain the confidence for a specific class we just need to multiply the object confidence by the class probability [4]:

$$\Pr(\text{Class}_i|\text{Object}) * \Pr(\text{Object}) * \text{IOU}^{\text{truth}}_{\text{pred}} = \Pr(\text{Class}_i) * \text{IOU}^{\text{truth}}_{\text{pred}}$$

Putting all together the output size for YOLO is:

$$[S, S, (B * (5 + C))]$$

where S: grid side, B: Anchor boxes number, C: Class Number

The following image (figure 2) illustrate a YOLO output with a 7x7 grid cell, 2 bounding boxes and 20 classes.

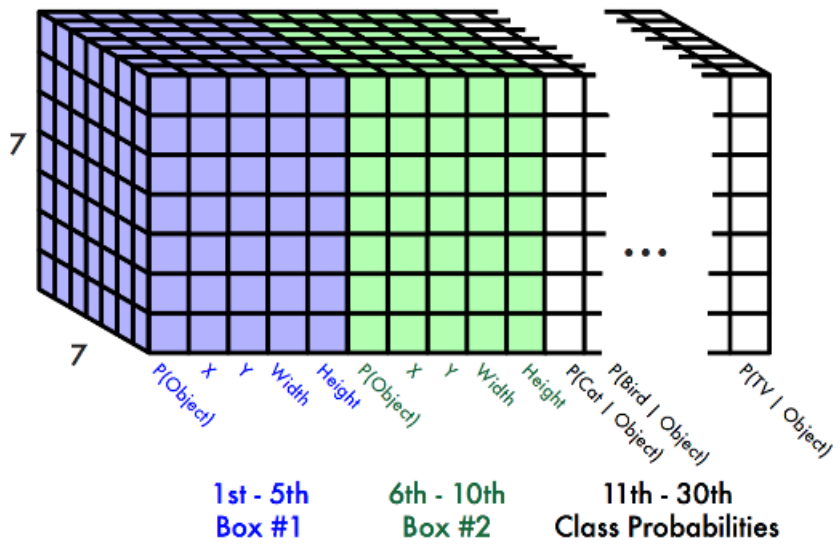


Figure 2: YOLO output example

Small objects could not be detected when using a small grid size, but the speed will be jeopardized with a large number of cells, so YOLO uses multiple grid sizes at once to overcome this issue , YOLOv3 uses 13x13, 26x26 and 32x32 grid sizes and tiny-yolov3 uses only 13x13 and 26x26 sizes, reason why it is faster and less accurate [11].

For this project I used an input size of 416x416x3, 3 anchor boxes and 3 classes, which gives me the following 2 output sizes:

$$[S, S, (B * (5 + C))] = [13, 13, (3 * (5 + 3))] = [13, 13, 24]$$
$$[S, S, (B * (5 + C))] = [26, 26, (3 * (5 + 3))] = [26, 26, 24]$$

The network is extremely fast and can detect SxSxB bounding boxes simultaneously if all the boxes are plotted in the image there will be more than one bounding box for each object detected as shown in the figure 3 below [4].

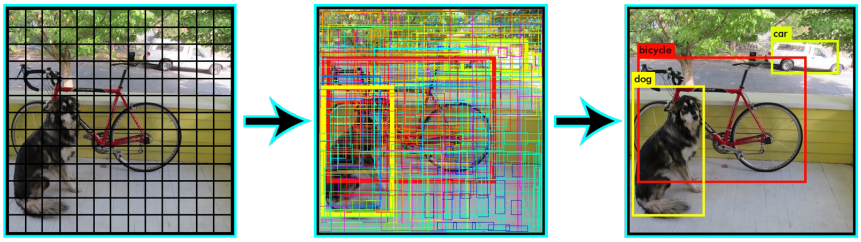


Figure 3: YOLO Non-Maximum Algorithm example

To reduce the number of bounding boxes in order to get only one for each object in the frame as picture c on last image non-maximum suppression (NMS) algorithm should be implemented [15]:

1. Eliminate all the bounding boxes with a confidence level lower than a certain threshold "Confidence threshold"
2. Eliminate all the overlapping box with an IOU higher than a desired threshold "IOU threshold" keeping the box with higher confidence level.

Sometimes these 2 steps are not enough, and extra manipulation must be performed, for instance when 2 classes can overlap this algorithm will keep only the class with higher confidence level ignoring the other, an example of overlapping classes are: Person, woman or Vest and Person. The last section of this report describes how the NMS algorithm was implemented on this project and how the problem of overlapping classes was solved.

layer	filters	size	input	output
0 conv	16	3 x 3 / 1	416 x 416 x 3	416 x 416 x 16
1 max		2 x 2 / 2	416 x 416 x16	208 x 208 x16
2 conv	32	3 x 3 / 1	208 x 208 x 16	208 x 208 x 32
3 max		2 x 2 / 2	208 x 208 x32	104 x 104 x32
4 conv	64	3 x 3 / 1	104 x 104 x32	104 x 104 x64
5 max		2 x 2 / 2	104 x 104 x64	52 x52 x64
6 conv	128	3 x 3 / 1	52 x52 x64	52 x52 x 128
7 max		2 x 2 / 2	52 x52 x 128	26 x26 x 128
8 conv	256	3 x 3 / 1	26 x26 x 128	26 x26 x 256
9 max		2 x 2 / 2	26 x26 x 256	13 x13 x 256
10 conv	512	3 x 3 / 1	13 x13 x 256	13 x13 x 512
11 max		2 x 2 / 1	13 x13 x 512	13 x13 x 512
12 conv	1024	3 x 3 / 1	13 x13 x 512	13 x13 x1024
13 conv	256	1 x 1 / 1	13 x13 x1024	13 x13 x 256
14 conv	512	3 x 3 / 1	13 x13 x 256	13 x13 x 512
15 conv	24	1 x 1 / 1	13 x13 x 512	13 x13 x24
16 yolo				
17 route		13		
18 conv	128	1 x 1 / 1	13 x13 x 256	13 x13 x 128
19 upsample			2x13 x13 x 128	26 x26 x 128
20 route		19 8		
21 conv	256	3 x 3 / 1	26 x26 x 384	26 x26 x 256
22 conv	24	1 x 1 / 1	26 x26 x 256	26 x26 x24
23 yolo				

Table 1: Tiny-YOLOv3 network architecture

The object detection is done thanks to a single CNN that performs the feature extraction, YOLOv2 uses a CNN called Darknet-19 to extract feature

from frames to perform the detection but in YOLOv3 a new and improved network is implemented, with 53 convolutional layers is called Darknet-53, this network is much more accurate and robust [10, 11].

The architecture of tiny YOLOv3 can be seen on the .cfg file using darknet, table 1 indicates each of the layers of the model implemented in this project with its respective input and output sizes.

As we can see the network only has 13 convolutional layers which is significantly smaller than Darknet-53 making Tiny-YOLOv3 impressively faster.

2.3 Training YOLO

Once that the network architecture is designed YOLO can be trained, this can be easily done in darknet by running a simple command specifying the path to the dataset folder, configuration file and classes names as it will be seen in later sections, during the training phase all the weights and hyper-parameters of the network are modified automatically in order to reduce the detection losses and improve the accuracy [5, 16].

Training a deep learning model from scratch is a processing heavy task because a huge number of weights and Hyper-parameters need to be adjusted [5], training YOLO can take several hours or days depending on the hardware and there is no guaranty that the model will converge, "transfer learning" overcome this issue by starting the training using weights from a similar trained application and tune them until they satisfy the new application [16], for this project YOLO weights trained on MS COCO dataset [17] were used as a starting point.

YOLO is trained by using full annotated images containing objects in a dataset, those images are pre-processed and feed into the network in groups called batches, the detection resulting is used with the ground true to calculate the losses gradients to adjust the weights [4].

The batch size plays an important tole during training due to the parallelism on GPUs multiples images can be feed into the network before the weights are adjusted, the problem arises when the GPU memory is not enough to keep up feeding the network, then the batch size needs to be reduced. Larger batch sizes reduce the training time also it has been demonstrated that increases the accuracy and generalization of the network [5].

2.4 Dataset

As mentioned before YOLO is trained using a dataset with full images and those images can have multiple object on it, this makes YOLO hard to implement at large scale due that dataset labelling is very expensive and time demanding, annotating implies giving the coordenates for each object in the image in the following format:

$$< class - id >< x >< y >< width >< height >$$

Class id indicates the number corresponding to the class labelled, x and y are the normalize coordinates to the center of the bounding box, width and height

are the normalized dimensions for the bounding box, a text file is associated with each image in the dataset containing one label per line, fortunately there are many software tools that helps reducing the annotation time [18] and a some new researches that involve artificial intelligence are trying to overcome this issue [19], also the "data augmentation technique" allow us to increase the training set size by only changing a few parameters of the existing images such as size, bright, noise, etc [5].

Figure 4: Labelling tool for image annotation

There is no an specific number of images to create a good dataset but the bigger and more diverse the better, also there are a wide variety of popular datasets available on Internet such as MS COCO [17], PASCAL VOC [20], mnist [21], imagenet [22] or Open Images [23], these datasets can help to reduce the preparation time specially when the objects to detect are simple and popular such as people, faces, animal, plants, balls, etc.

When creating a dataset for object detection the following images should be included:

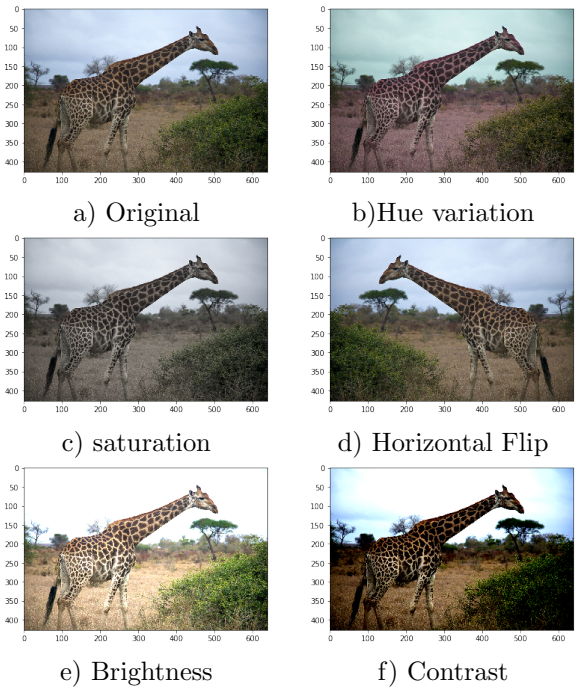
1. the number of images for each class must be similar.
2. the object from different angles and poses
3. the object in different backgrounds
4. diverse lighting and exposure levels.
5. the object at different distances.
6. the resolution equal or higher than the network input.
7. the object in all possible configurations.
8. representative data with the object in a similar environment it will be during the application.

2.5 Data augmentation:

As mentioned before the data augmentation objective is to increase the number of images in the training-set to increase diversity and about over-fitting by performing transformation on the existing data [24] without the need of expending extra hours labelling new images [5].

The following transformation are some of the transformations that can be performed individually or in group to increase the dataset:

- Flipping and Rotation
- Cropping
- shifting
- colour jittering
- Noise addition
- scaling
- Brightness variation



a) Original b)Hue variation

c) saturation d) Horizontal Flip

e) Brightness f) Contrast

Figure 5: Data Augmentation example

On Figure 5, 3 basic different transformations and the original image are shown:

Those transformations are very easy to code, it just need simple image processing operations, there are a few tools available online that perform data augmentation such as [25], Darknet also has an inbuilt tool for this purpose, this tool is configured through YOLO's configuration file, the cfg file has some parameters such as angle, saturation, exposure, hue and scale that can be edited and used during training to get a more robust and accurate network avoiding overfitting [25].

3 OpenVINO

3.1 Overview

OpenVINO stands for Open Visual Inferencing and Neural Network Optimization, it's a toolkit designed by Intel used to deploy visual oriented solutions on intel platforms such as CPUs, GPUs, VPUs and FPGAs.

The OpenVINO toolkit allows deep learning inferencing on the Edge and reduces developing times supporting execution on multiple devices such as CPUs, intel integrated graphics, intel Movidius compute stick and includes calls for OpenCV, OpenCL and OpenVX [26].

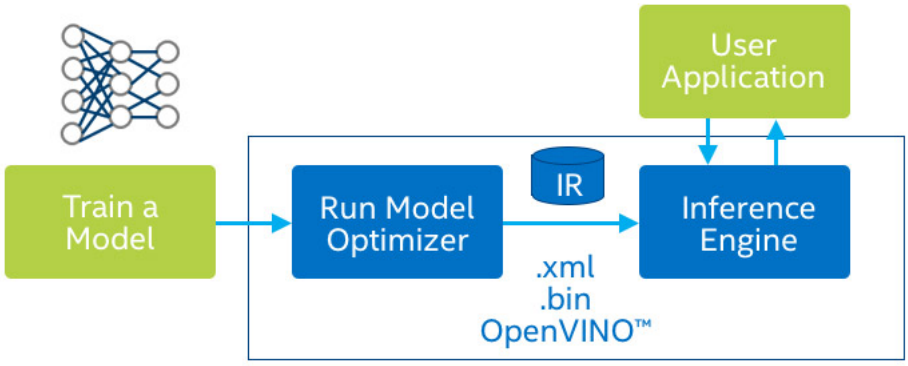


Figure 6: OpenVINO Workflow [27]

The 2 main components of OpenVINO toolkit are:

- **The model Optimizer**: A platform that create the IR of models to be used by the inference engine, the IR is hardware agnostic. The model optimizer supports a wide variety of frameworks such as TensorFlow and Caffe. The model optimizer also has tools to improve and modify models [27].

- **The inference engine**: An API to perform deep learning inference in multiple hardware types. The model needs to be converted to its IR before performing inference [26].

Figure 6 shows the steps to optimize and deploy a model, note that the OpenVINO toolkit does not train the model it was designed focused on inference only.

A trained model is the input of the model optimizer, OpenVINO supports models from the most popular frameworks such as TensorFlow or caffe as long as all the dependencies has been pre-installed, there are many public pre-trained models available online [28] that can be download and used.

The model optimizer output is an Intermediate Representation of the network which describes the model, this representation can be used to deploy the neural network in different platforms without modifying it.

Two files compose the IR:

1. .xml file that contain the topology of the network
2. .bin file that contain the weights and biases.

Once that the IR representation is generated the inference engine allows the deployment for the user application.

3.2 Inference Engine Python API.

On this section I will review briefly on how to use Python API provided in the OpenVINO toolkit installation package, the Python API is available from OpenVINO™ toolkit R1.2 however I used OpenVINO toolkit 2019 R1.

Intel Documentation about the python API for OpenVINO can be found in the following links

- Python API overview
- Python API example

Intel clarifies repeatedly that this API is an evaluation version and the structure and the API itself can be changed in the future [29, 30].

The Python API is an interface for the Inference engine, it allows us to perform the following tasks [30]:

- Handle the models.
- Load and configure the Plugin depending on the device.
- Perform inference.

It works with Python 3.5 and 3.6 on Ubuntu and Windows 10, to see the full compatibility list please refer to Intel documentation.

Once that OpenVINO is installed we need to set up the environmental variables before running the inference engine, OpenVINO toolkit provides a script file that does that for us [29]:

```
#On Ubuntu 16.04:
source <INSTALL_DIR>/bin/setupvars.sh

#On Windows 10:
call <INSTALL_DIR>\deployment tools\inference engine\python_api\
    setenv.bat
```

Listing 1: set environmental variables

This script detects the latest python version installed and set the environmental variables if the version is supported. A typical flowgraph for the python API is shown in figure 7:

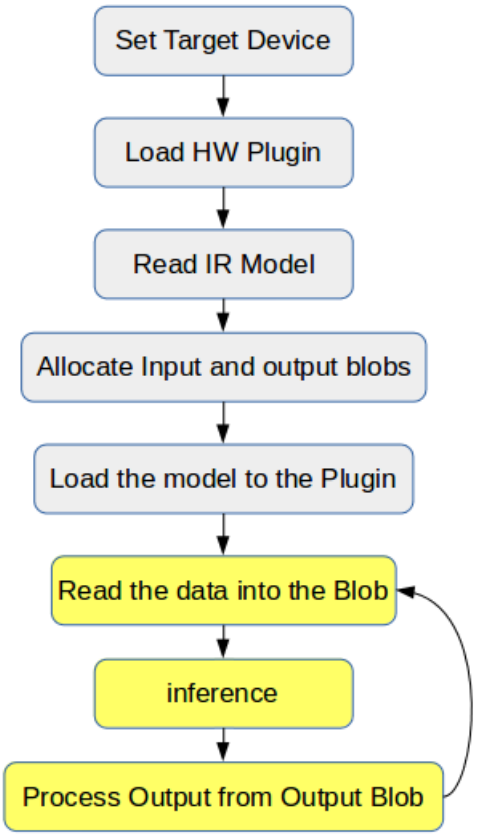


Figure 7: Inference Engine Python API Flowgraph

First we need to define the device where the inference will be performed (MYRIAD, CPU, FPGA or GPU) to load its plugin, then the Intermediate Representation of the network is read (.xml and .bin files) and load into the plugin to generate an executable network, finally each frame is read and preformatted to perform the inference and process the output, this last step is

performed repeatedly for each frame.

This completely flowgraph can be performed using only 3 classes [30]: IENetwork, IEPlugin and ExecutableNetwork which are briefly described below:

A) IENetwork

This class provide the information about the model, the two input parameters are the .xml and .bin file (the IR of the model), this class allows to see each layer of the network and modify some parameters such as batch size, layer affinity, etc [30].

```
#import the class:
from openvino.inference_engine import IENetwork, IEPlugin

#read the model from the IR files:
net = IENetwork(model=xml_file_path, weights=bin_file_path)
```

One of the class attributes I used is inputs, it is a dictionary that maps the input layer names into an InputInfo object, this attribute allows us to get the shape of the input layer, the following code show how to get the input layer shape of the Tiny-YOLOv3 IR:

```
from openvino.inference_engine import IENetwork, IEPlugin
 net = IENetwork(model="/home/fernando/
    frozen_darknet_yolov3_mode_90000.xml", weights="/home/fernando/
    frozen_darknet_yolov3_mode_90000.bin")
net.inputs
# Output
# {'inputs': <openvino.inference_engine.ie_api.InputInfo object at
    0x7fbfd2d7f9b0>}
net.inputs['inputs'].shape
# Output
# [1, 3, 416, 416]
```

B) IEPlugin

This class is used to configure and run the plugin, the main property this class has is the device name, as we know OpenVINO inference engine can perform inference in multiple devices such as CPU, GPU, FPGA, MYRIAD or HETERO. The load method is used to create an executable network from a network object (net=IENetwork(...)) after loading the IR to the plugin [30].

```
# set the device name property on plugin.
plugin = IEPlugin(device="MYRIAD")
# read the IR model.
net = IENetwork(model=xml_file_path, weights=bin_file_path)
# retrieve the input layer information.
input = next(iter(net.inputs))
# load the network into the plugin.
exec_net = plugin.load(network=net)
```

C) ExecutableNetwork

An executable network is a network ready for inference, the network instance was previously loaded into the plug in and now is ready to perform

inferences: the executable network instance is obtained after using the IEPlugin.load() method. The method infer() is used to perform the inference, is a simplified version of infer() method of InferRequest class, it receives as a parameter the data for the input layer as a numpy.ndarray with the proper shape and returns a numpy.ndarray with the output data [30]. Example:

```
# import the class:
from openvino.inference_engine import IENetwork, IEPlugin

# Read the model from IR.
net = IENetwork(model=xml_file_path, weights=bin_file_path) # read
    the IR model.

# configure and load the plugin
plugin = IEPlugin(device="MYRIAD") # set the device name property
    on plugin.
exec_net = plugin.load(network=net) # load the network into the
    plugin.

# read input layer information;
input = net.inputs

# perform inference.
outputs = exec_net.infer(inputs={input: img})
```

Part II
Project Description

4 Overview

The project is divided in 3 main blocks, Hardware platform, model developing and deployment, figure 8 shows the system block diagram, the followings sections describe each of the 3 main blocks involved.

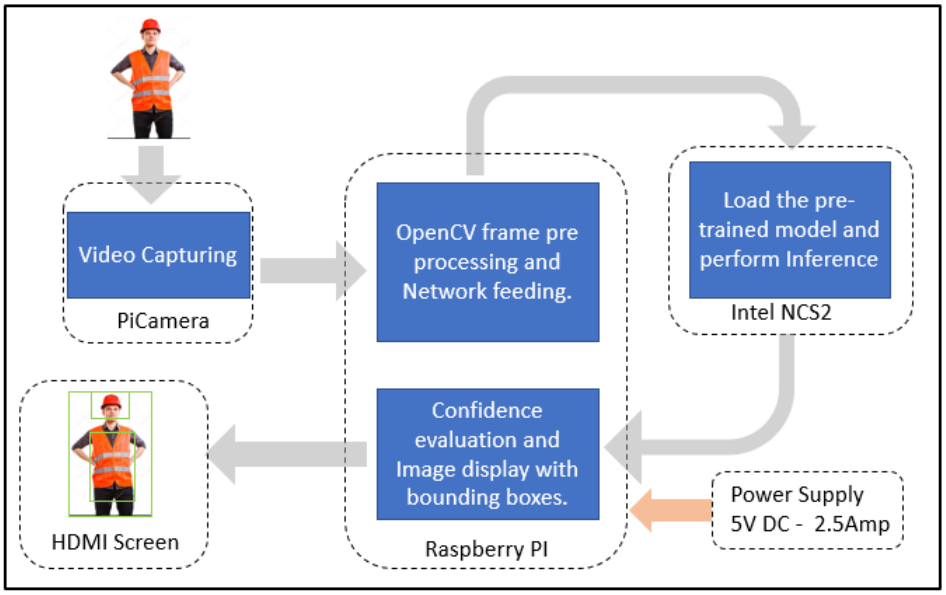


Figure 8: System block diagram

5 Hardware platform.

The hardware for this project is composed by only 4 main items: a single board computer, a camera, a NCS2 and a power supply, which made the development process simple, some changes could be done for the design to be ready for mass production, but the model and software will be basically the same.

The output of the system can be displayed on a remote PC, so no screen is needed. Below I've detailed each of the main components and why they were selected.

5.1 Raspberry Pi 3 Model B+:

As mentioned before, the project includes a single board computer based on a quad-core ARM Cortex CPU, table 2 has the following specifications:

SoC	Broadcom BCM2837
CPU	4 x ARM Cortex A53 1.2GHz
GPU	Broadcom Video Core IV
RAM	1GB LPDDR2 (900 MHz)
Networking	10/100 Ethernet, 2.4GHz 802.11n wireless
Bluetooth	Bluetooth 4.1 Classic, Bluetooth Low Energy **Storage** microSD
GPIO	40-pin header populated
Ports:	HDMI, 3.5mm analogue audio-video jack, 4x USB 2.0, Ethernet RJ45, Camera Serial Interface (CSI), Display Serial Interface (DSI)

Table 2: Raspberry Pi 3 model B+ specs [31]

The raspberry Pi OS is a Linux distribution called Raspbian Stretch desktop, this OS comes with pre-installed software for education, programming and general use. This board was selected for the following reasons:

- Medium range power consumption 5W
- Raspbian OS supports OpenVINO 2019 R1
- Small form factor.
- Wide variety of resources available online.
- Has been in the market for long time.

The Raspberry Pi CPU handles all the operations but the inference, it oversees the pre-processing of the frames from the camera to be feed into the network and handling the result and interface to the user.

5.2 Raspberry Pi Camera V2:

A camera is needed to capture and input images from the construction site to the system, the requirements to be fulfil are listed below:

- Small form factor
- Compatible with Raspbian OS
- Video capable.
- Resolution at least 640x480
- Low cost.

- No additional hardware to control it.

The camera chosen is the Raspberry Pi camera V2 which exceeds the requirements, it is supported in Raspbian OS and the raspberry pi already counts with a dedicated port for this camera so no extra IOs are required, the connection to the pi is done through a ribbon cable which is highly beneficial because the NCS2 physically block 3 USB ports. The following table list the specifications of the Pi Camera V2:

Size	25 x 24 x 9 mm
Weight	3g
Still resolution	8 Megapixels
Video modes	1080p30, 720p60 and 640 x 480p60/90
Linux integration	V4L2 driver available
C programming API	OpenMAX IL and others available
Sensor	Sony IMX219
Sensor resolution	3280 x 2464 pixels
Sensor image area	3.68 x 2.76 mm (4.6 mm diagonal)
Pixel size	1.12 μm x 1.12 μm
Optical size	1/4”
Focal length	3.04 mm
Horizontal field of view	62.2 degrees
Vertical field of view	48.8 degrees
Focal ratio (F-Stop)	2

Table 3: Raspberry Pi camera specs [31]

5.3 INTEL Neural Compute Stick 2

Intel NCS2 has a Vision Processing Unit (VPU) called Intel® Movidius™ Myriad™ X which has a dedicated Neural Compute Engine, 16 High performance SHAVE Cores, enhanced ISP with 4K support and vision accelerators including stereo depth [32].

The Raspberry Pi has a relative powerful CPU and GPU, but it is a bottleneck for computer vision applications that involve deep learning inference, for that reason an inference accelerator was needed, there are a few in the market such as Google Coral or Intel Neural compute Stick version 1 and 2, Intel NCS2 has been in the market for longer time so more documentation and examples can be found online [2, 28].

The NCS2 has been built for computer vision and AI inference applications [3, 33], it has USB3 connection, low power consumption (1 watt) and a very small form factor which make it ideal for Edge devices, it will handle all the inference processing during the model deployment.

Intel NCS2 is based on a MYRIAD X VPU, the main features of this device are listed below [32]:

- Dedicated Neural Compute Engine
- 16 High Performance SHAVE Cores
- Enhanced ISP with 4K support
- New Vision Accelerators including Stereo Depth
- Low Power consumption 1Watt

Although the price is a little bit high it is still affordable compared with other options such as powerful CPUs or GPUs.

5.4 Power supply

Because this is a prototype the power supply is just a wall AC-DC converter from 240VAC to 5VDC 2.5 Amp which is enough to supply the current needed for the entire system avoiding the Raspberry Pi to restart for undervoltage caused by draining excessive current from the power supply.

The following table shows the power consumption of each of the hardware components involved:

Item	Max power consumption [Watt]
Raspberry Pi 3 model b+	5
Intel Neural Compute Stick 2	1
Raspberry PI Camera V2	1
Total =	7

Table 4: Power Consumption

Please note that the values on the table are approximated values obtained by different benchmarks found online, there is no official data-sheet showing these values.

$$P_min = 7Watt$$
$$V = 5Volts$$
$$I_min = P/I = 7Watt/5Volts = 1.2Amp$$

The minimum current that the power supply must source to guarantee the proper function of the system is 1.2 Amp, to be 100% sure and considering future modification or improvements the value is doubled.

$$I = 2xI_min = 2.4Amps$$

5.5 Enclosure

A demo enclosure was designed and printed with a 3d printer with ABS filament, this enclosure is only used as a demo, it is not weather proof as required for the final product but is enough to demonstrate the concept.

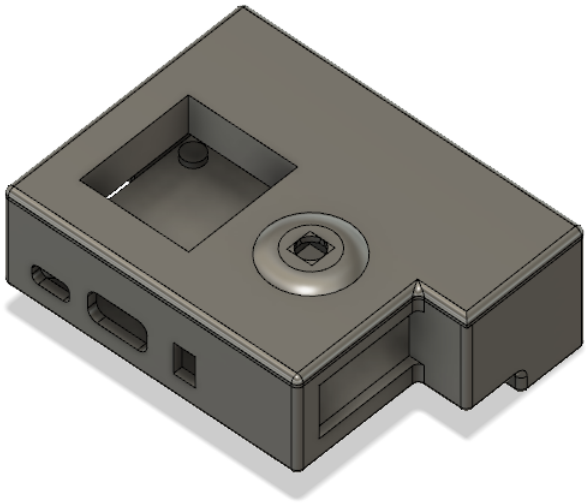


Figure 9: Enclosure

6 Model developing

On this section all the steps to develop the model are described, starting from installing the software required, followed by preparing the dataset and training the network and finalizing with how to obtain the intermediate representation using OpenVINO Model Optimizer, the resulting IR will be used to deploy the detection on the Raspberry PI and NCS2.

6.1 Installing required software:

This section describes how to install the software required to train the model and to obtain the IR which will be used in the deployment phase, the OS used during this stage was Ubuntu 16.04.

6.1.1 Installing Darknet on Ubuntu 16.04:

The procedure to install Darknet framework can be found on its official website [12], however this is a simplified version of it that works perfect on Ubuntu 16.04.

install darknet clone and make the following repository:

```
git clone https://github.com/pjreddie/darknet.git
cd darknet
make
```

the installation can be confirm executing the following command:

```
./darknet
# the output should be the following:
usage: ./darknet <function>
```

To add OpenCV to darknet the Makefile needs to be edited and enable OpenCV:

```
# in darknet folder edit Makefile
sudo nano Makefile
# set OPENCV=1
# press ctrl+o and enter.
Make
```

to test OpenCV execute the following command, a few windows showing a bird must open:

```
./darknet imtest data/eagle.jpg
```

if the verification does not work is probably because the environmental variable PKG_CONFIG_PATH is not set, it can be check using the following command:

```
echo $PKG_CONFIG_PATH
```

to set the environmental variable python-dev needs to be installed:

```
sudo apt=get install python-dev
export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
```

6.1.2 Installing OpenVINO Toolkit

The procedure to install OpenVINO Toolkit on a Linux machine can be found on Intel's official website www.openvinotoolkit.org, the installation process consists of 4 main steps.

1. Install Intel OpenVINO toolkit
2. Install External software and dependencies.
3. Set OpenVINO environmental variables
4. configure the model optimizer

1) install Intel OpenVINO toolkit:
Go to Intel's official website and download the latest OpenVINO distribution in /Downloads, then run the commands below:

```
cd ~/Downloads/
tar -xvzf l_openvino_toolkit_p_<version>.tgz
cd l_openvino_toolkit_p_<version>
sudo ./install_GUI.sh
```

Follow the steps indicated on the screen. the default directory where OpenVINO is installed is: "/opt/intel/openvino_version/"

2) install External software dependencies.

The model optimizer and inference engine need these external dependencies to operate properly.

```
cd /opt/intel/openvino/install_dependencies
sudo -E ./install_openvino_dependencies.sh
```

3) Set environmental variables.

These variables need to be set before running OpenVINO. The variables can be set using the following command:

```
source /opt/intel/openvino/bin/setupvars.sh
```

optionally, the file .bashrc can be modify so the previous command executes every time a terminal is opened.

```
echo "source /opt/intel/openvino/bin/setupvars.sh" >> ~/.bashrc
```

4) Configure the model optimizer.

The model optimizer needs to be configured according to the framework used on the original model, to configure the model optimizer to work with all the frameworks supported by openvino execute the following command:

```
cd /opt/intel/openvino/deployment_tools/model_optimizer/
    install_prerequisites
sudo ./install_prerequisites.sh
```

alternatively, the MO can be configured to work only with TensorFlow models executing:

```
cd /opt/intel/openvino/deployment_tools/
cd model_optimizer/install_prerequisites
sudo ./install_prerequisites_tf.sh
```

to test the installation openvino counts with 2 examples, to run one of them:

```
cd /opt/intel/openvino/deployment_tools/demo/
./demo_squeezenet_download_convert_run.sh
```

this demo classifies a picture of a car and gives the 10 top labels resulting, it uses the model optimizer and the inference engine, so both are fully tested when the script is executed successfully.

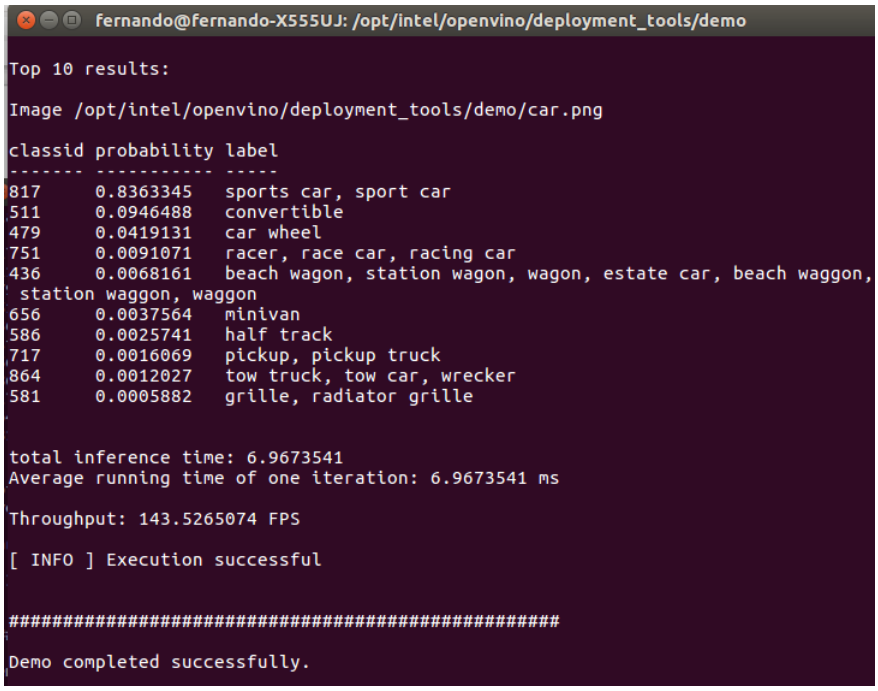


Figure 10: OpenVINO demo squeezenet output

6.2 Dataset preparation

As mentioned before to train YOLOv3 model each picture on the dataset must have a txt file associated to it which contains each of the bounding boxes coordinates for the object on each picture.

The txt file must have the same name as the picture and each line should have the following format:

$$< class\_id >< x >< y >< width >< height >$$

There is a few open-source software that make the labelling processes easier, in this case I use a GitHub repository made in python [18].

https://github.com/tzutalin/labelImg

I compiled around 3000 images in this dataset, every picture in the dataset contains at least one of the classes I wanted to detect, to make the system as robust as possible I tried to include this kind of pictures:

- People in different angles.
- people in different locations or backgrounds

- Include both genders and different ages.
- People wearing and not wearing Hat and vests.
- multiple colours and types of hats.
- multiple colour and type of vests.

Figure 11 shows the graphical user interface for the labelling tool used.

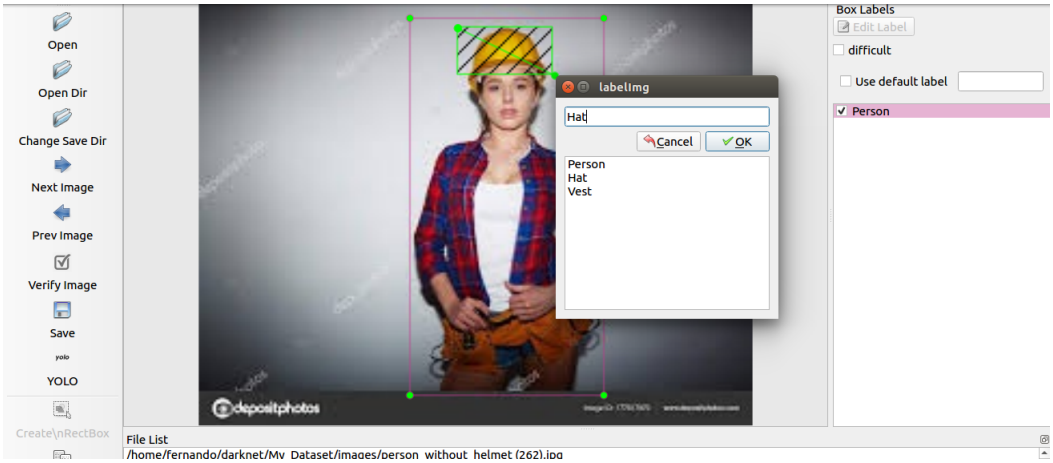


Figure 11: Labelling tool

The following steps shows how to use the labelling tool:

1. Create a file called images and save all the images in it:

```
mkdir images
mkdir labels
```

2. go to darknet directory and download the zip file from this link:

```
git clone https://github.com/tzutalin/labelImg
cd labelImg
```

3. run the following commands:

```
# Please note these steps are for Python 3 + Qt5
sudo apt-get install pyqt5-dev-tools
sudo pip3 install -r requirements/requirements-linux-
python3.txt
make qt5py3
python3 labelImg.py
```

4. Run the script

```
python3 labelImg.py
```

5. Click open Dir and select the folder where all the images are.
6. Click Change sav dir and select the labels directory.
7. Select YOLO instead of VOC and start labelling the pictures, remember to use always the same labels

6.3 Configure darknet and Yolo files

Before starting with the training, the model needs to be configured, as mentioned before the official YOLO files are prepared to detect 80 object classes with in a maximum image size of 416x416x3 so we need to adjust some parameters to adequate the network.

The files we need to create/edit are the following:

1. configuration file yolov3-tiny.cfgfile
2. objects.labels
3. trainer.data
4. train.txt and test.txt
5. tiny-yolov3 pre-trained weights

The following procedure is an adaptation and modification of the procedure published on YOLO official website to detect 3 classes, the original procedure can be found in the following link:

https://pjreddie.com/darknet/yolo/

1) Edit cfg file.

The configuration file has all the information about each of the layer that compose tiny YOLO network, for this project I need to modify this file to detect 3 classes:

```
#download the yolov3-tiny.cfgfile:
cd ~/darknet
mkdir custom
cd custom/
wget https://github.com/pjreddie/darknet/blob/master/cfg/yolov3-
    tiny.cfg

# Open the file
sudo nano yolov3-tiny.cfg
```

Edit the beginning of the configuration file as below:

```
[net]
# Testing
#batch=1ytho
#subdivisions=1
#Training
batch=1
subdivisions=1
#input size 416,416,3
width=416
height=416
channels=3
#data augmentation.
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1
```

Batch and subdivisions are used for training, leave it in 1 if you don't have a good GPU (4GB+)

width and height are the sizes for the input layer, the bigger the better accuracy of the network but it will be slower as well [11], I chose 416x416 to get at least 4fps with the raspberry pi.
Channels = 3 because I use coloured images.
The rest of the parameters are used to configure the data augmentation tool.
we need to set the number of classes to 3 (line 135 and 177) and the number of filters to 24 (line 127 and 171) on each YOLO layer .

2) create train.txt and test.txt file.

These files contain all the images paths that will be used for training and all the images path that will be used for evaluation. The file should have a structure similar as shown in figure 12.

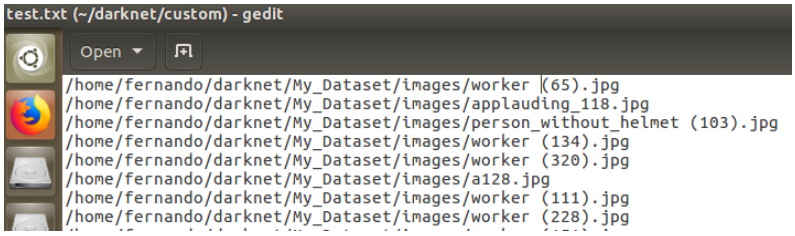


Figure 12: train.txt and test.txt file structure

Keep in mind that the train and validation images must not be the same, I used 10% of the total images for validation.

3) create object.names file This file contains the list of classes that the system will detect. The order on this file must be the same used to annotate

the dataset.

```
sudo nano object.names
# add the following content
Person
hat
vest
#press ctrl+o then Enter
#press ctrl+x then Enter
```

4) create train.data file This file contains all the information that the trainer needs, number of classes, location of the file train.txt, object.names and the folder where the weights are going to be stored.

```
Sudo nano train.data
```

Add the following content:

```
classes= 3
train = custom/train.txt
valid = custom/test.txt
names = custom/objects.names
backup = backup/
```

5) Download tiny yolo pre-trained weights to use as starting point

```
wget https://pjreddie.com/media/files/darknet53.conv.74
```

6.4 Train the model with darknet

Darknet has a function to train the detector, it only needs 3 parameters: the configuration file, the trainer.data file (which has the path to the dataset, backup folder, class number, etc) and the starting point pre-trained weights path (transfer learning) [12].

To train the network run the following command:

```
./darknet detector train custom/trainer.data custom/yolov3-tiny.cfg
    darknet53.conv.74
```

The weights will be saved automatically every 100 iterations until 900 then they will be saved every 10000 iterations.

It is recommended to send the console output to a log file, so we can plot the losses vs iteration later, to save the output to a training.log file run the following command instead:

```
./darknet detector train custom/trainer.data custom/yolov3-tiny.cfg
    darknet53.conv.74 >> log/training.log
```

The training can be stop using ctrl+c and then resumed using the following:

```
./darknet detector train custom/trainer.data custom/yolov3-tiny.cfg
    backup/yolov3-tiny.backup
```

To see the training progress, open the log file on /darknet/log/training.log, train the network until the avg loss is less than 1 or until the avg value is not changing anymore.

6.4.1 Plot the losses vs iterations

To evaluate the accuracy of the network we can analyse the avg loss of the detections, this avg loss is calculated based on the detection results obtained with the current weights, as can be seen in figure 13 the avg value is decreasing over iterations:

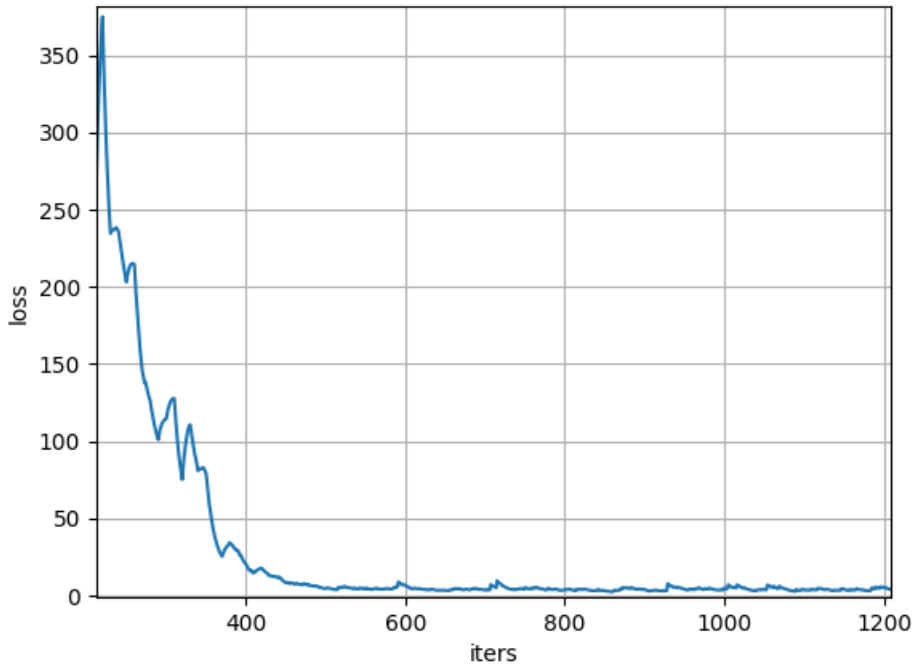


Figure 13: Loss vs iterations

Although the losses are decreasing the model could be performing worse when tested with the validation data, this is due to overfitting, we want our model to generalize so it is recommended to test the model periodically to find out the most appropriate weights

There are multiple ways to plot the losses vs iterations from the log file, it can be done using Microsoft excel or any spreadsheet software. The following GitHub repository implements a Python code to plot the graph [34]:

```
cd ~/darknet/
wget https://github.com/Jumabek/darknet_scripts/blob/master/
plot_yolo_log.py
python3 plot_yolo_log.py log/training.log
```

6.4.2 Test darknet weights using darknet

To test the training results we can use the inbuilt test in darknet, run the following commands to perform object detection on an image:

```
./darknet detector test custom/trainer.data custom/yolov3-tiny.cfg
    backup/yolov3-tiny_10000.weights RELATIVE_PATH_TO_IMAGE
```

example:

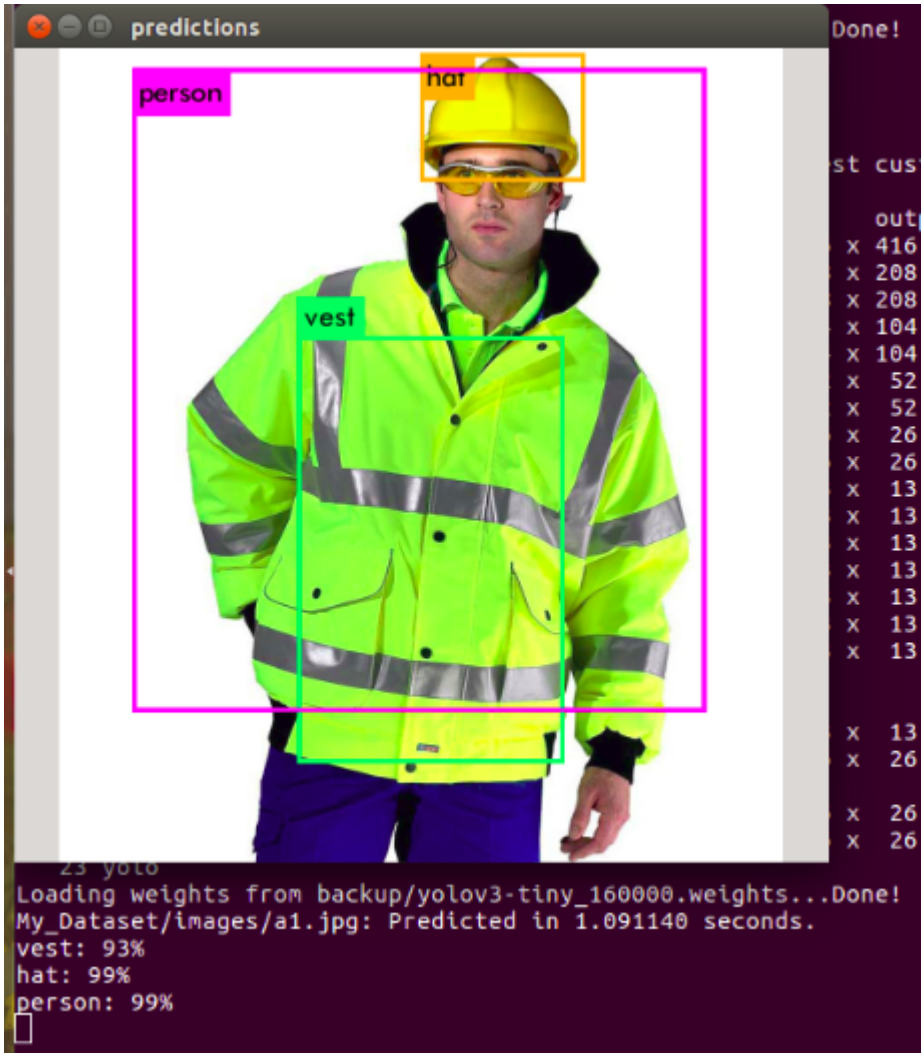


Figure 14: Darknet detection result

6.5 Converting YOLO weights to TensorFlow model.

As we know, OpenVINO model optimizer can generate the IR from models with different structures, it supports the most popular frameworks such as TensorFlow, caffe and others [27], but YOLO is written using C language and Darknet [4], which is not supported by the MO so as a first instance we needed to convert the darknet model to TensorFlow, Darknet models are dynamic, so the model needs to be freeze and stored using TensorFlow structure (extension .pb) [35].

Requirements [27]:

- Python 3.6
- TensorFlow 1.2
- OpenVINO Toolkit 2019 R1
- YOLOv3 weights, cfg files and class names.

We can find different public repositories on GitHub to convert from yolo weights to TensorFlow implementation (.pb file) in this opportunity I used the repository recommended by Intel [35, 36]:

```
# Download the repository:
git clone https://github.com/mystic123/tensorflow-yolo-v3.git
cd tensorflow-yolo-v3

# convert to .pb
python3 convert_weights_pb.py
    --class_names My_files/objects.names
    --data_format NHWC
    --weights_file My_files/yolov3-tiny.backup
    --tiny
    --output_graph ~/darknet/custom/frozen_tiny_yolo_model.pb
```

The frozen model file will be created in the following location

```
~/darknet/custom/frozen_tiny_yolo_model.pb
```

6.5.1 Test the .pb model

To test the model the repository [36] has a python code that load a file into the network and produce an output.jpg image with bounding boxes on detected objects.

```
python3 ./demo.py
    --input_img /home/fernando/Documents/extended_dataset/a1.jpg
    --output_img resultado.jpg
    --frozen_model frozen_darknet_yolov3_model.pb
    --class_names My_files/objects.names
    --tiny
```

The result will be a jpg image with the bounding boxes and labels as this:

As we can see in figure 15 the detection is similar using both TensorFlow and darknet.

Figure 15: TensorFlow model detection result

6.6 Generate the IR using MO

Once that the YOLO weights from darknet were converted into a TensorFlow frozen graph the model optimizer can be used, before running the model optimizer we need to modify a Json file to be able to convert the YOLO layer into its IR [35].

6.6.1 Edit Json File.

The Json file contains all the attributes of the YOLO layer, the recommended tiny-yolov3 Jason file can be found at:

```
<OPENVINO_INSTALL_DIR>/deployment_tools/model_optimizer/extensions/
    front/tf
```

To adapt the provided Json file it is necessary to edit the custom attributes fields, all the item in this category except for entry_points can be found in the cfg file, entry_points don't need to be modified [35].

```
[
  {
    "id": "TFYOLOV3",
    "match_kind": "general",
    "custom_attributes": {
      "classes": 3,
      "anchors": [10,14,  23,27,  37,58,  81,82,  135,169,
    344,319],
      "coords": 4,
      "num": 6,
      "mask": [0, 1, 2],
      "entry_points": ["detector/yolo-v3-tiny/Reshape", "detector/
    yolo-v3-tiny/Reshape_4"]
    }
  }
]
```

Package inputenc Error: Keyboard character used is undefined

6.7 Run model optimizer:

After the TensorFlow frozen model was created and the Jason file modified it is time to run the model optimizer to generate the IR of the network, the model optimizer can be found in the following folder:

```
cd /opt/intel/openvino/deployment_tools/model_optimizer
```

because it is a TensorFlow model I used mo_tf.py:

```
python3 /opt/intel/openvino/deployment_tools/model_optimizer/mo_tf.
    py
    --input_model tensorflow_files/frozen_tiny_yolo_model_160000.pb
     --tensorflow_use_custom_operations_config tensorflow_files/
    yolo_v3_tiny.json
    --batch 1
    --data_type FP32
    -o openvino_IR/FP16/
    -n tiny_yolo_IR_160000_FP16
```

where:

- input_model is the frozen model in TensorFlow format .pb.
- tensorflow_use_custom_operations_config is used for custom layers, in this case the YOLO layer.
- batch: number of frames that will be entered to the network at the same time
- data_type: use FP16 for NCS2 and FP32 for CPU.

- o: output directory
- n: name of the file.

After running the previous command two file called -n.xml and -n.bin will be stored into the directory -o those files correspond to the IR of the network.

openvino_IR/FP16/tiny_yolo_IR_160000_FP16.xml
openvino_IR/FP16/tiny_yolo_IR_160000_FP16.bin

7 Deployment

Once that the model's IR is obtained it can be used to perform inference on a NCS2 connected to the Raspberry Pi using OpenVINO inference engine Python API [26]. This section describes the steps to install OpenVINO toolkit on the raspberry Pi and shows the Python code implemented for this project explaining the main parts of it.

7.1 Installing OpenVINO on Raspberry Pi

This topic shows how to install OpenVINO on a raspberry Pi 3 B+ running Raspbian stretch OS with desktop.

The following link allows to download Raspbian Stretch OS with desktop and recommended software which is the version used for this project.

https://downloads.raspberrypi.org/raspbian_full_latest

Although Intel has its own procedure to install OpenVINO toolkit here [37], this procedure shows a simpler way to install OpenVINO on the raspberry PI using Raspbian Stretch.

It is important to download the same OpenVINO version as used on the PC where the model optimizer was run, for this project the 2019 R1 version was used, previous versions do not support YOLOv3 on NCS2 due to a software bug.

```
# Download OpenVINO from INTEL
cd ~/Downloads
wget https://download.01.org/opencv/2019/openvinotoolkit/R1/
    l_openvino_toolkit_raspbi_p_2019.1.094.tgz

# Expand the swapfile partition to 1024 MB
sudo nano /etc/dphys-swapfile

# Press ctrl+o and Enter to save then Ctrl+x to close the file.

# Install cmake
sudo apt-get install cmake

# Create the installation folder and unpack the file
sudo mkdir -p /opt/intel/openvino
```

```
sudo tar -xf l_openvino_toolkit_raspbi_p_2019.1.094.tgz --strip 1 -
    C /opt/intel/openvino

# Modify the script setupvars.sh
sudo sed -i "s|<INSTALLDIR>|/opt/intel/openvino|" /opt/intel/
    openvino/bin/setupvars.sh
```

7.1.1 Environmental variables:

The environmental variables need to be set every time before using OpenVINO, the best way to do it is modifying the .bashrc file which is executed every time a new terminal is open [37].

```
echo "source /opt/intel/openvino/bin/setupvars.sh" >> ~/.bashrc
```

To verify that the changes have been applied open a new terminal, the first line should show the following message.

```
[setupvars.sh] OpenVINO environment initialized
```

7.1.2 USB Rules

To perform inference on the Intel® Movidius™ Neural Compute Stick or Intel® Neural Compute Stick 2, install the USB rules as follows [37]:

```
# Add the current user to users group
sudo usermod -a -G users "$(whoami)"
sh /opt/intel/openvino/install_dependencies/install_NCS_udev_rules.
    sh
```

7.1.3 Test OpenVINO with Object Detection Sample

OpenVINO toolkit comes with a few samples to test and evaluate the toolkit installation [28], the following steps shows how to run a pre-trained face detection network.

1. Create a new directory

   ```
   mkdir build && cd build
   ```

2. Build the sample

   ```
   make -DCMAKE_BUILD_TYPE=Release -DCMAKE_CXX_FLAGS="-march=
   armv7-a" /opt/intel/openvino/deployment_tools/
   inference_engine/samples
   make -j2 object_detection_sample_ssd
   ```

3. Download the pre-trained model to the computer,

```
# This file contains the weights of our model
wget --no-check-certificate https://download.01.org/opencv
    /2019/open_model_zoo/R1/models_bin/face-detection-adas
    -0001/FP16/face-detection-adas-0001.bin

# Download the network topology with the following command:
wget --no-check-certificate https://download.01.org/opencv
    /2019/open_model_zoo/R1/models_bin/face-detection-adas
    -0001/FP16/face-detection-adas-0001.xml
```

4. Save a bmp image in any desired location, make sure that the image contains at least one face on it.

5. Connect the NCS and run the sample specifying the path to the image.

```
./armv7l/Release/object_detection_sample_ssd -m face-detection
    -adas-0001.xml -d MYRIAD -i <path_to_image>
```

6. Wait until the sample finish and verify that has been executed successfully, the shell should show the following message:

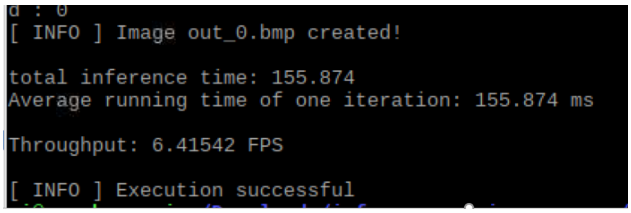


Figure 16: Execution Successful message

7. A New image with the name out_0.bmp is created in the working directory, open it and verify the result

Figure 17: Face detection sample result

7.2 Python Code.

This section shows the python code used to perform the detections and describes its more important blocks.

7.2.1 User interface Flow graph

The following flow graph shows the basic structure for the user application code, additional features could be added in the future to make it more robust and avoid workers to fool the system.

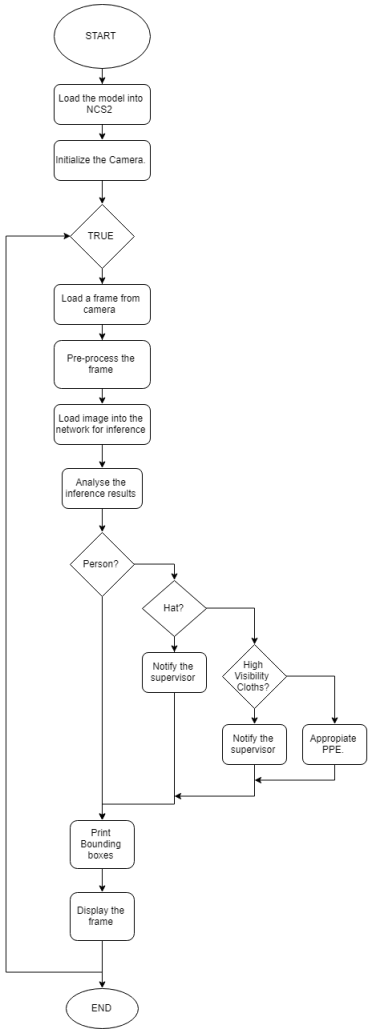


Figure 18: User interface flow graph

7.2.2 Acquire frames from Raspberry Pi camera:

Source

To be able to use the raspberry Pi camera (PiCamera) first it needs to be enabled from the configuration panel:

```
sudo raspi-config
# go to interface and select camera
# press enter and test the camera
raspistill -v -o test.jpg

# a file called test.jpg will be saved on the working directory.
```

Once that the HW has been enabled we need to download the module that allow using the PiCamera with python and OpenCV, for that reason we install "PiCamera[array]" this module ant its sub-module "array" allow us to interface the PiCamera with Python getting NumPy arrays to work with OpenCV.

```
  pip install "picamera[array]"
```

hence, to start acquiring raw images from the PiCamera to load them into the network:

```
# import the necessary packages
from picamera.array import PiRGBArray
from picamera import PiCamera
import cv2

# initialize the camera and grab a reference to the raw camera
    capture
RPi_Camera = PiCamera()
RPi_Camera.resolution = (640, 480)
Rpi_Camera.framerate = 32
raw = PiRGBArray(RPi_Camera, size=(640, 480))

# capture frames from the camera
for cap in RPi_Camera.capture_continuous(raw, format="bgr",
    use_video_port=True):
  # grab the image as NumPy array
  frame = cap.array

  ...

  # the code for the detections goes here.
  ...

   #Display Image
   cv2.imshow("Window_tittle", frame)

      # clear the stream in preparation for the next frame
      rawCapture.truncate(0)

   if cv2.waitKey(1)&0xFF == ord('q'):
       break

cv2.destroyAllWindows()
```

Because the resolution of the YOLO network is 416x416 a resize of the frame needs to be done keeping the aspect ratio, this is very simple to perform with OpenCV:

```
# constant declarations
yolo_input_size = 416
resolution = [640,480]

# compute new resolution
resized_resolution = [yolo_input_size,int(resolution[1] * (
    yolo_input_size/resolution[0]))]

#resize image (this should be perfrmed in the loop)
resized_image = cv2.resize(image, resized_resolution)
```

Now that the image has a resolution smaller than 416x416 can be prepared to enter in the network, as discussed before the model accept an input array of (1,3,416,416) which is (batch number, width (colour image), height, width) known as NCHW format. To obtain the 4-dimension array to input in the network:

```
# create an array full of 0s with size (416,416,3)
blob = np.full((yolo_input_size, yolo_input_size, 3), 0)

#copy the image to the new vector
blob[0:resized_resolution[1],:,  :] = resized_image

# add a new axis
blob = blob[np.newaxis, :, :, :]

#convert from NHWC to NCHW
blob = blob.transpose((0, 3, 1, 2))
```

The array is ready to be entered into the network.

7.2.3 Detection Object class

The data for each detection will be saved in a class which contains all the data required to plot the result on the screen: bounding box coordinates, class id and confidence.

This class also has a method to initialize the values since the box parameters obtained from the network are normalized and we need to transform them into absolute [38].

```
class Detected_Obj():
    xmin = 0
    ymin = 0
    xmax = 0
    ymax = 0
    class_id = 0
    confidence = 0.0

    def __init__(self, x, y, h, w, class_id, confidence, h_scale,
    w_scale):
```

```
        self.xmin = int((x - w / 2) * w_scale)
        self.ymin = int((y - h / 2) * h_scale)
        self.xmax = int(self.xmin + w * w_scale)
        self.ymax = int(self.ymin + h * h_scale)
        self.class_id = class_id
        self.confidence = confidence
```

7.2.4 Model implementation.

As described in one of the initial sections OpenVINO 2019 R1 has a Python API which can be used as an interface to the inference engine, this API allow us to read the model from the IR, configure the HW and plug in and execute the detection.

The aim of next lines is to show and describe how the OpenVINO inference Engine was implemented in this project.

```
#import the class:
from openvino.inference_engine import IENetwork, IEPlugin

#read IR model from xml and bin files.
model_xml = "frozen_darknet_yolov3_model_160000.xml"
model_bin = os.path.splitext(model_xml)[0] + ".bin"
net = IENetwork(model=model_xml, weights=model_bin)

# set the target device
plugin = IEPlugin(device="MYRIAD")

# Allocate input blob.
input_blob = next(iter(net.inputs))

# Load the model to the plugin.
 exec_net = plugin.load(network=net)

—>> FRAME LOOP STARTS HERE <<—

  ...

  # read the data into the blob.

  # perform the inference
        outputs = exec_net.infer(inputs={input_blob: prepimg})

  ...

  #process the output.
# delete all the objects
del net
del exec_net
del plugin
```

7.2.5 Output interpretation.

YOLOv3 divides the input image into a SxS grid (13x13 and 26x26 for tiny-YOLOv3), each cell of the grid can predict a certain number of bounding boxes

and 3 class probabilities in this case, each of the boxes predicted has 5 parameters (x, y, h, w and object probability) the first 4 parameters are normalized and belongs to the centre, Hight and width of the object detected, then the 5th parameter belong to the probability or confidence that an object exist on that cell without saying to which class it belongs

The class predictions indicate the percentage of confidence that the object detected in that cell belong to certain class.

The following Python code process the output vector generated after the inference and produces a list of objects detected, these objects have attached the information for the bounding box as well as the confidence and class id.

```
# This constant comes from YOLO configuration file.
number_of_classes = 3
coords = 4
num = 3
anchors = [10,14, 23,27, 37,58, 81,82, 135,169, 344,319]

# this function helps to find an index in the flatten array.
def EntryIndex(s_sqr, lcoords, lclass, location, entry):
    n = int(location / s_sqr)
    loc = location % s_sqr
    return int(n * s_sqr * (lcoords + lclass + 1) + entry * s_sqr +
     loc)

# This function interprets the output of YOLO network and return
# a list of detected objects

def YOLOV3 (blob, resized_resolution, resolution, thrld, objects):
    # get the side lenght
    side = blob.shape[2]
    anchor_offset = 6
    s_sqr = side * side

    # flat the output matrix into a vector.
    output_blob = blob.flatten()

    # To check each cell for each num and each class.
    for k in range(s_sqr):
      # get the row and column for each cell
        row = int(k / side)
        col = int(k % side)
        for n in range(num):
            obj_index = EntryIndex(s_sqr, coords, number_of_classes
    , n * s_sqr + i, coords)
            box_index = EntryIndex(s_sqr, coords, number_of_classes
    , n * s_sqr + i, 0)
      # get the object probability
            scale = output_blob[obj_index]
            if (scale < thrld):
                continue
      # get the center coordenates.
            x = (col + output_blob[box_index + 0 * s_sqr]) / side *
     resized_resolution[0]
            y = (row + output_blob[box_index + 1 * s_sqr]) / side *
     resized_resolution[1]
```

```
            height = math.exp(output_blob[box_index + 3 * s_sqr]) *
     anchors[anchor_offset + 2 * n + 1]
            width = math.exp(output_blob[box_index + 2 * s_sqr]) *
    anchors[anchor_offset + 2 * n]
      # get the class probability.
            for j in range(number_of_classes):
                class_index = EntryIndex(s_sqr, coords,
    number_of_classes, n * s_sqr + i, coords + 1 + j)
                object_probability = scale * output_blob[
    class_index]
                if object_probability < thrld:
                    continue
        # Create the object and append it to the list.
                obj = Detected_Obj(x, y, height, width, j, prob, (
    resolution[1] / resized_resolution[1]), (resolution[0]/
    resized_resolution[0]))
                objects.append(obj)
  # return the list of detected objects.
    return objects
```

7.2.6 Intersection over union (IOU)

This functions it is used later to remove duplicated detections, it receives two detected objects and calculates the intersection over union between them, first it computes the area of intersection, then doing the addition of both boxes areas minus the intersection area it computes the area of union, finally it calculates the IOU factor doing the intersection_area over the union_area, it returns 0 if the boxes don't intersect.

```
def IOU(boxA, boxB):
    # determine the coordenates of the intersection rectangle
    xA = max(boxA.xmax, boxB.xmax)
    yA = max(boxA.ymax, boxB.ymax)
    xB = min(boxA.xmin, boxB.xmin)
    yB = min(boxA.ymin, boxB.ymin)

    # compute the intersection area
    intersection_area = max(0, xB - xA) * max(0, yB - yA)

    # compute each box area
    boxA_area = (boxA.ymax - boxA.ymin)  * (boxA.xmax - boxA.xmin)
    boxB_area = (boxB.ymax - boxB.ymin)  * (boxB.xmax - boxB.xmin)

    #compute the area of union.
    union_area = boxA + boxB - intersection_area

    iou = 0.0
    if area_of_union > 0:
        iou = (intersection_area / union_area)
    return iou
```

7.2.7 Non-Maximum suppression algorithm

As was discussed before, YOLO divide the input frame into a grid and each cell can detect an object, sometimes multiple cells can detect the same object, and this will produce multiple bounding boxes surrounding the same object [4], Non-maximal suppression algorithm (NMS) get rid of this issue by eliminating all the boxes that detect the same object [15].

The NMS algorithm implemented uses the Intersection Over Union value which indicates the lever of overlapping between two boxes, if two boxes have a certain IOU higher than a IOU_threshold means that they are referring to the same object and only the box with a higher confidence level will be kept, this also applies if two different classes are detected by the same cell, the NMS algorithm will keep only the box with the highest confidence score [15].

For classes which have some level of overlapping for instance: Person, woman or Person, Vest this will be a problem because only the class with highest confidence will be considered, reason why the detections are classified into classes first and then the NMS algorithm is applied for each class separately.

Duplicated boxes are eliminated following these steps:

1. Make a list of objects detected for each class (3 lists)
2. Remove all the detections with a confidence score lower than "conf_threshold"
3. Eliminate Overlapping objects with an IOU higher than "IOU_threshold"

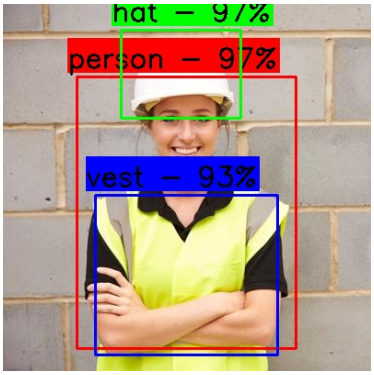
```
# Separate detected object by classes

        for i in range(len(objects)):
            if(objects[i].class_id == 0):
                People.append(objects[i])

            if(objects[i].class_id == 1): # Hat
                Hats.append(objects[i])
            if(objects[i].class_id == 2):
                Vests.append(objects[i])

# Eliminate overlapping Hats.
        for i in range(len(Hats)):
            if (Hats[i].confidence == 0.0):
                continue
            for j in range(i + 1, len(Hats)):
               if (IOU(Hats[i], Hats[j]) >= IOU_threshold):
                    if Hats[i].confidence < Hats[j].confidence:
                        Hats[i], Hats[j] = Hats[j], Hats[i]
                    Hats[j].confidence = 0.0

# same procedure is done for people and vest
```

8 Results

The following images show the inference results after applying YOLO's to a bunch of images, every object detected is automatically bounded with a box and labelled with its class id and class probability.

There is a color code that allows a quick object identification (red=person, blue=vest, green=hat).

It is important to mention that none of these images belong to the training set so we can appreciate the degree of generalization that the network have.



Most of the images in the data-set correspond to a construction site workers reason why i assume the prediction for non-worker persons are weaker (low class confidence).

The code used to obtain these samples takes images from a folder, resize them keeping the aspect-ratio until they resolution fit YOLOs input size 416x416, the inference time for all the images is bout 40ms when running on a laptop CORE i7 with a GPU nvidia Geforce 920M, this timing will produce a 25fps video detection.

The performance on video was as fast as expected, it was possible to reach up to 4 Fps on the raspberry pi + NCS2 with only utilizing 20% of the Raspberry Pi's CPU.

Although the timing was between the desired boundaries there was always a few frames where no objects were detected, reason why i decided to decrease the confidence_threshold to 20%, this means all the objects detected with a confidence level of 20% or more will be displayed on the screen, Also i wait for

10 frames before taking a decision about the worker on the screen, so if a person is detected and there is no helmet or vest detected in the following 10 frames the system notifies the supervisor that the person is not wearing appropriate PPEs.

Although the system could detect multiple people, hats or vests such as in the following figures, i had to limit the number to 1 people for frame in order to be able to apply the fix mentioned in the previous paragraph.

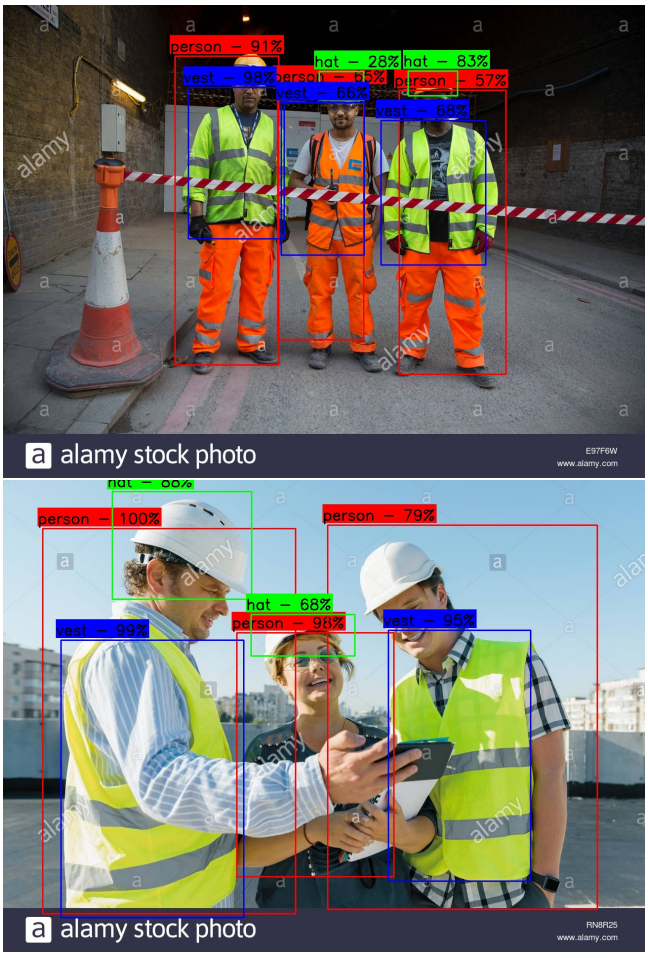


Figure 19: Multiple objects detected on a image

Decreasing the threshold had some disadvantages, some false-positives appeared while testing, specially with helmets, the solution to this issue will involve increasing the dataset and further train the network.

The NCS2 demonstrated to be very stable, the network was tested for 24hs

Figure 20: Hat false detection

with a 100% up-time. A different test was used to measure the temperature over 2 hs reached a maximum of 55C which is acceptable for such a set up but it is advisable to add a fan to the case to produce air circulation, the temperature was taken using the temperature sensor on the sense hat,

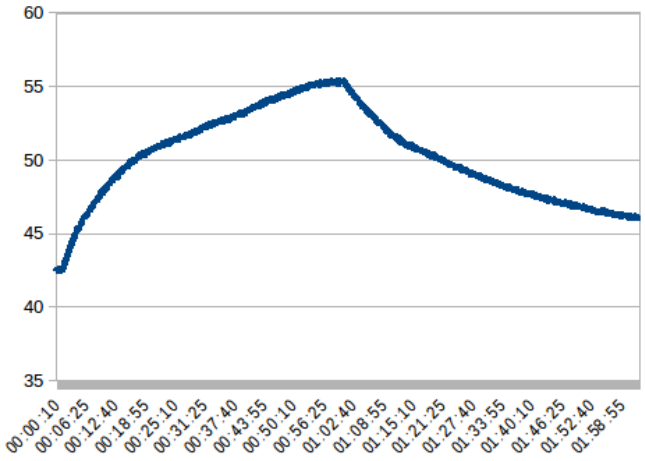


Figure 21: Raspberry Pi Temperature [C] vs Time [hh:mm:sss]

9 Conclusion

During this project i performed object detection using Tinny-YOLOv3 model to detect 3 classes (Person, hat and vest), the inference was deployed on an Intel Neural Compute stick obtaining satisfactory results, it was notable the fact that the Raspberry Pi CPU usage was around 20% while running the network inference on the NCS2, this give developers a wide resource margin to run additional processes.

The model was created and trained using Darknet, then converted to a frozen tensorflow model to finally use OpenVINO model optimizer to generate the intermediate representation of the network, this workflow can be followed by other applications.

The biggest time consumer task on this project was the data-set preparation, image annotation takes long time although there are a wide range of resources such as open and public data-sets or software that facilitate the task.

The PPE detector achieved 4fps running on a Raspberry Pi with the NCS2, this demonstrates that Movidius is a great tool to accelerate the inference on edge devices with a reasonable power consumption.

References

[1] S. W. Australia, "Personal protective equipment," Mar 2017, [Accessed 6 Apr. 2019]. [Online]. Available: https://www.safeworkaustralia.gov.au/ppe

[2] A. M. Khan, I. Umar, and P. H. Ha, "Efficient compute at the edge: Optimizing energy aware data structures for emerging edge hardware," in *2018 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2018, pp. 314–321.

[3] "Press release: Intel unveils the intel neural compute stick 2 at intel ai devcon beijing for building smarter ai edge devices," Nov 14 2018, name - Nasdaq Stock Market Inc; Movidius; Intel Corp; Copyright - Copyright Dow Jones Company Inc Nov 14, 2018; Last updated - 2018-11-15. [Online]. Available: http://ez.library.latrobe.edu.au/login?url=https://search-proquest-com.ez.library.latrobe.edu.au/docview/2133181072?accountid=12001

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[5] S. Agarwal, J. O. D. Terrail, and F. Jurie, "Recent advances in object detection in the age of deep convolutional neural networks," *arXiv preprint arXiv:1809.03193*, 2018.

[6] J. R. Parham, J. Crall, C. Stewart, T. Berger-Wolf, and D. Rubenstein, "Animal population censusing at scale with citizen science and photographic identification," in *2017 AAAI Spring Symposium Series*, 2017.

[7] M. A. Al-masni, M. A. Al-antari, J. Park, G. Gi, T.-Y. Kim, P. Rivera, E. Valarezo, S.-M. Han, and T.-S. Kim, "Detection and classification of the breast abnormalities in digital mammograms via regional convolutional neural network," in *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2017, pp. 1230–1233.

[8] S. Wu and L. Zhang, "Using popular object detection methods for real time forest fire detection," in *2018 11th International Symposium on Computational Intelligence and Design (ISCID)*, vol. 1. IEEE, 2019, pp. 280–284.

[9] D. M. Izidio, A. P. Ferreira, and E. N. Barros, "An embedded automatic license plate recognition system using deep learning," in *2018 VIII Brazilian Symposium on Computing Systems Engineering (SBESC)*. IEEE, 2019, pp. 38–45.

[10] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[11] ——, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[12] J. Redmon, "Darknet: Open source neural networks in c," http://pjreddie.com/darknet/, 2013–2016.

[13] Thtrieu, "Darkflow: Translate darknet to tensorflow," Mar 2018. [Online]. Available: https://github.com/thtrieu/darkflow

[14] Xingwangsfu, "Yolo in caffe," Dec 2016. [Online]. Available: https://github.com/xingwangsfu/caffe-yolo

[15] R. Rothe, M. Guillaumin, and L. Van Gool, "Non-maximum suppression for object detection by passing messages between windows," in *Asian Conference on Computer Vision*. Springer, 2014, pp. 290–306.

[16] F. Chollet, *Deep Learning with Python*, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2017.

[17] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[18] Tzutalin, "labelimg: a graphical image annotation tool." May 2019. [Online]. Available: https://github.com/tzutalin/labelImg

[19] O. Russakovsky, L.-J. Li, and L. Fei-Fei, "Best of both worlds: human-machine collaboration for object annotation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 2121–2131.

[20] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[21] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[23] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, T. Duerig *et al.*, "The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale," *arXiv preprint arXiv:1811.00982*, 2018.

[24] A. Fawzi, H. Samulowitz, D. Turaga, and P. Frossard, "Adaptive data augmentation for image classification," in *2016 IEEE International Conference on Image Processing (ICIP)*. Ieee, 2016, pp. 3688–3692.

[25] srp 31, "Data augmentation for object detection(yolo)," Jul 2018. [Online]. Available: https://github.com/srp-31/Data-Augmentation-for-Object-Detection-YOLO-

[26] Intel, "Inference engine developer guide." [Online]. Available: https://docs.openvinotoolkit.org/latest/_docs_IE_DG_Deep_Learning_Inference_Engine_DevGuide.html

[27] ——, "Model optimizer developer guide." [Online]. Available: https://docs.openvinotoolkit.org/latest/_docs_MO_DG_Deep_Learning_Model_Optimizer_DevGuide.html

[28] ——, "Pretrained models — intel® distribution of openvino™ toolkit," May 2018. [Online]. Available: https://software.intel.com/en-us/openvino-toolkit/documentation/pretrained-models

[29] Bryankbr, "Use an inference engine api in python* to deploy the intel® distribution of openvino™ toolkit," Jun 2018. [Online]. Available: https://software.intel.com/en-us/articles/use-an-inference-engine-api-in-python-to-deploy-the-openvino-toolkit

[30] Intel, "Overview of inference engine python* api." [Online]. Available: https://docs.openvinotoolkit.org/latest/_inference_engine_ie_bridges_python_docs_api_overview.html#ieplugin-class

[31] RaspberryPi, "Raspberry pi hardware." [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/

[32] Intel, "Intel® movidius™ myriad™ x vpu." [Online]. Available: https://www.movidius.com/myriadx

[33] ——, "Intel® neural compute stick 2: Smarter, faster, plug-and-play ai at the edge," Feb 2019. [Online]. Available: https://www.intel.ai/intel-neural-compute-stick-2-smarter-faster-plug-and-play-ai-at-the-edge/#gs.dvwil3

[34] Jumabek, "Darknet scripts," May 2018. [Online]. Available: https://github.com/Jumabek/darknet_scripts

[35] Intel, "Converting yolo* models to the intermediate representation (ir)." [Online]. Available: https://docs.openvinotoolkit.org/latest/_docs_MO_DG_prepare_model_convert_model_tf_specific_Convert_YOLO_From_Tensorflow.html

[36] mystic123, "Tensorflow-yolo-v3," Dec 2018. [Online]. Available: https://github.com/mystic123/tensorflow-yolo-v3

[37] Intel, "nstall openvino™ toolkit for raspbian* os - openvino toolkit." [Online]. Available: https://docs.openvinotoolkit.org/latest/_docs_install_guides_installing_openvino_raspbian.html

[38] PINTO0309, "Openvino-yolov3," Apr 2019. [Online]. Available: https://github.com/PINTO0309/OpenVINO-YoloV3