Deep Learning in Low Cost Embedded Systems

Bruno Bueno Furquim
Luiz Antonio Celiberto Junior
Universidade Federal do ABC.

Edson Coaru Kitani
Faculdade de Tecnologia de Santo André.

ABSTRACT

In the topic of autonomous (electro) mobility and embedded vehicle electronics, object detection becomes inevitable. Yours importance is significant, because it covers other sectors such as manufacturing, agriculture and administrative processes. However, the image processing procedure suffers from the complexity of the training process, being necessary to pay attention to the hardware requirements. Given this, the paper is dedicated to presenting questions and manipulations of microprocessors, to make not only able to implement learning methods, but also effective, reaching higher levels of abstraction in low cost and high accessibility embedded systems. Thus, will be shown with the results generated, the best pre-trained deep learning algorithms available on the internet and effective processing techniques.

To avoid using a graphics processing unit - GPU, a Raspberry Pi microprocessor will be used, due to the processing capacity, cost and other factors. That is, it will be possible to show the possibility or at least the limits, to work with elements responsible for vehicle autonomy, through easily accessible resources and reduced cost.

INTRODUCTION

Object detection has been good enough for a variety of applications (although image segmentation is a much more accurate result, it suffers from the complexity of creating training data. Usually takes a human note 12x more time to segment an image than to draw bounding boxes this is more anecdotal and lacks a source) [9]. Furthermore, after detecting objects, it is possible to segment the bounding box object separately.

Using object detection: Object detection is of significant importance and has been used in several sectors. Some of the examples are mentioned below:

-Manufacturing;
-Agriculture;
-Augmented Reality;
-Automation in the Work Environment;
-Autonomous Vehicles; [1]

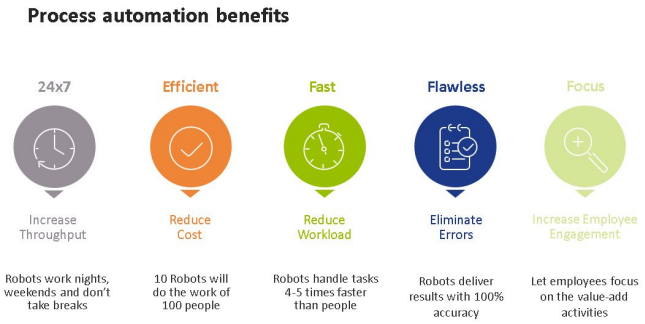

Figure 1. Benefits of RPA [3]

Object identification can be used to answer a variety of questions. These are the broad categories:

-Is an object present in my image or not? for example, there is an intruder in my house;

-Where is an object in the image? For example, when a car is trying to navigate the world, it is important to know where an object is.

-How many objects are there in an image? Object detection is one of the most efficient ways to count objects. For example, how many boxes in a rack within a warehouse;

-What are the different types of objects in the image? For example, which animal exists in which part of the zoo?

-How big is an object?

-How are different objects interacting with each other? For example, how does training on a football field affect the outcome?

-Where is an object in relation to time (Tracking an Object). For example, tracking a moving object like a train and calculating its speed, etc. [1]

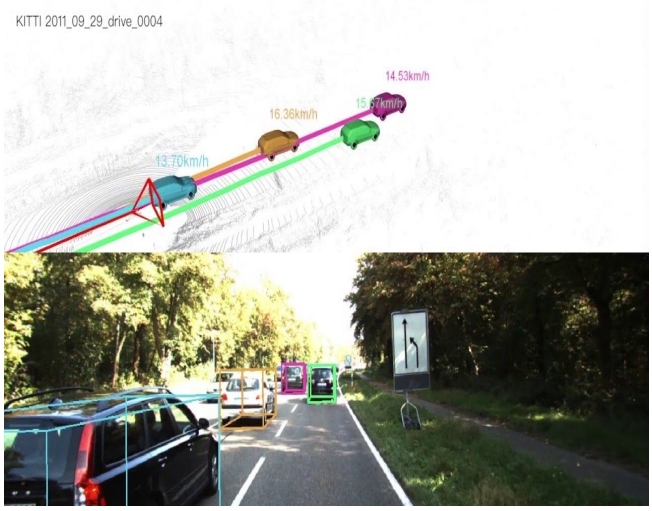

Figure 2. Tracking for Autonomous Driving [4]

In view of these reasons, this work is dedicated to present, through stages, the questioning and improvement from the point of view of Hardware, in order to enable the execution of learning methods and thus reaching higher levels of abstraction in low-level embedded systems. cost and high accessibility.

It is expected to present, at the end of this work, with the results generated by offline tests, the best techniques for processing a deep learning algorithm in our system. In order to avoid the use of a graphics processing unit - GPU, a Raspberry Pi microprocessor will be used, due to the processing capacity, cost and among other factors. That is, it will be possible to show the possibility, or at least the limits, to work with elements responsible for vehicle autonomy, through easily accessible resources and reduced cost.

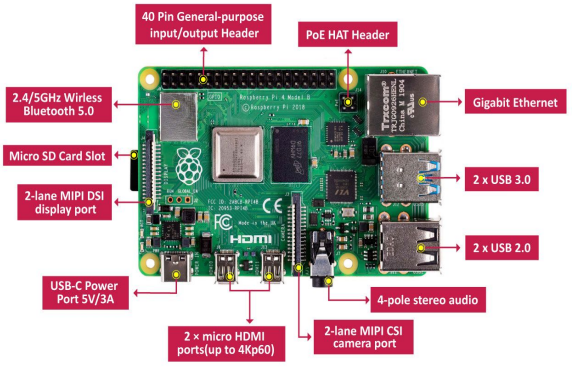

Figure 3. Raspberry Pi 4

THEORETICAL REFERENCE

Regarding autonomous technology, Deep Learning is an essential element, as it allows the system to extract and process the maximum possible amount of data generated in ever increasing variety and speed, allowing to create models and achieve high levels of accuracy, related to operational dynamics.

A low-cost microprocessor will be used to process the Deep Learning algorithm and widely disseminated on the Internet, together with software and libraries available through Github based on the Open Source initiative, which makes its licensed source code available with an open source license in which the copyright provides the right to study, modify and distribute the software for free for anyone and for any purpose [5]. This facilitates the study because it allows to reduce time spent with the development of intermediate steps. In addition, it indicates paths, such as the creation of clusters as well as used in automation, encryption software and others, already existing, favorable to obtain the expected results only with the use of the Raspberry Pi 3 and peripherals.

The project will feature especially the OpenCV and Deep Learning tutorials, resources and guides provided by Adrian Rosebrock, which are available through his personal website called PyImageSearch, which is dedicated to helping other programmers understand how search engines work. images. Although many computer vision concepts are theoretical in nature. Its objective is to bring your life experiences in the creation of image search engines in concise and easy to understand and learn examples, all based on the “learn by example” method [6].

METHODOLOGY

Essentially, experimental research consists of determining an object of study, selecting the variables that would be able to influence it, defining the forms of control

and observation of the effects that the variable produces on the object [8].

When the objects under study are physical entities, such as portions of liquids, bacteria or mice, no major limitations are identified as to the possibility of experimentation. However, when it comes to experimenting with social objects, that is, with people, groups or institutions, the limitations become quite evident. Ethical and human considerations prevent experimentation from being carried out efficiently in the humanities, which is why experimental procedures are only suitable for a small number of situations. However, experiments in the humanities are increasingly frequent, especially in Psychology (for example: learning), in Social Psychology (for example: measuring attitudes, studying the behavior of small groups, analyzing the effects of advertising, etc.) and in Sociology of Work (for example: influence of social factors on productivity) [8].

The project will also seek a Methodware methodology, thus distributing the process in: Planning; Execution; Monitoring; Control; Closing [9].

First, a bibliographic review will be sought and the study of what has been worked on this theme in a more recent way, that is, to recognize the State of the Art of Deep Learning and resources as it can be used in a small processor like the Raspberry Pi . In this way it is possible to proceed with the subject in the most updated way possible.

During the execution of Deep Learning algorithms it is essential to follow a workflow that consists of 6 main steps, which are divided into 3 parts:

Collecting training data:
- Camera: Image capture;
- Annotation: Annotate the images.

Training the model:
- Pre-processing by Raspberry Pi;
- Train the Algorithm.

New image predictions:
- Random device: Capture the image;
- Random device: Predict the Image [10].

PHASE 1 - Collecting training data

Step 1 - Collect Images (at least 100 per Object):
For this task, we need 100 images per object.

Step 2 - Make a note (draw boxes on these Images manually):

Draw bounding boxes on the images. It is necessary to use a tool like labelImg. Usually people are needed who will work to annotate the images. This is a very intense and time-consuming task.

However, it is already possible to obtain these pre-selected images in a database made available via Github, making it possible to save a lot of time and complete this phase in a few minutes.

PHASE 2 - Training a model on the Raspberry Pi

Step 1 - Find a pre-trained model for transfer learning: A pre-trained model is needed to be able to reduce the amount of data needed to train. Without this, some 100k images would be needed to train the model. However, it is possible to find several pre-trained models again on Github.

Step 2 - Training the Raspberry Pi using Open Source software and approaches, once again all the documentation and tips are available via Gihub. To train a model, we will need to select the right “hyper parameters”. The art of “Deep Learning” involves a little bit of trial and error to find out which are the best parameters to obtain the highest precision for our model - This will be one of the first points on which approaches will be made to improve the processing of the Raspberry Pi.

Still in Step 2: “Quantize Model”

This model allows to reduce the size to fit in a small device like the Raspberry Pi due to its low memory and reduced computing capacity when compared to NVidia's GPUs and others. Here is one of the main fields of action of the IC, that is, to look for ways to work and to overcome the deficiencies of the Raspberry Pi.

Neural network training is done by applying many tiny stimuli, and these small increments usually need floating point precision to work. By taking a pre-trained model and running the inference we hope to have a very different result.

One of the qualities of Deep Neural Networks is that they tend to handle very high levels of noise at their inputs very well. Therefore, when quantifying neural network models, it is possible to decrease file sizes, storing the minimum and maximum of each layer, and then each floating value is compressed into an eight-bit integer. Consequently, the file size is reduced by 75%, since the nodes and weights of a neural network are originally stored as 32-bit floating point numbers.

PHASE 3 - New image predictions using the Raspberry Pi

Step 1 - Capture a new image via the camera

For this step we will have a camera connected to the Raspberry Pi which both the tutor and the student have

available, and not being restricted to simple cameras like Open MV, but also 3D, for example the Kinect.

Figure 4. OpenMV camera

Then the new image will be captured to give permission and start the last step.

Step 2 - Predicting a new image

Download the model through the link mentioned above.
Install TensorFlow on the Raspberry Pi.
Perform training on the model.

Finally, you can even use methods found on the internet that allow you to work with the high volume of processing required through mechanisms such as: encrypting the signal and reading through software that can be easily found on the internet, due to the initiative Open-Source.

As an example, it will be briefly described one of the methodologies easily found which can be used in CI:

-A Python routine checks whether a new data package exists in a specific directory (a package consists of a .PRO file (prologue), several data files and a .EPI file (epilogue). When the file arrived. I know that a data package has been downloaded. The new data is moved to a temporary directory.

-Another routine in Python runs the utility to unzip the images. This utility has the source code released, so it is possible to compile for the ARM platform.

-From the uncompressed data, image processing can be done using MPOP

-Finally, the images are reprojected.

Through these methods, indoor and online approaches will be carried out without the need to use a mobile robot, that is, in the laboratory we will only work with the presentation of objects and images to the system and obtain results from your response. Depending on the response time and time available, it will be sought to work, later, on a mobile system which will be discussed during the project.

DEVELOPMENT

APPLICATION 1 - OpenMV with Raspberry:

The OpenMV project deals with the creation of low cost “Machine vision” modules, extensible and powered by Python, and aims to become the Arduino of “Machine vision”. Our goal is to bring “Machine vision” algorithms closer to manufacturers and enthusiasts. We made the algorithm work difficult and time consuming for you, leaving more time for your creativity! [11]

OpenMV Cam is like a super powerful Arduino with an integrated camera that you program in Python. We make it easy to run “Machine vision” algorithms on what OpenMV Cam sees, so you can track colors, detect faces and more in seconds and control I / O pins in the real world. [11]

To run OpenMV on Raspberry, just access Raspberry's official website, go to the “Downloads” tab and click on “DOWNLOAD NOW FOR RASPBERRY PI 1,2,3 OR LATER”. Then just unzip and read the instructions in “Readme.txt”

There you will find how to download the OpenMV IDE where you will have access to a vast number of examples

APPLICATION 2 - Deep Learning with OpenCV:

When OpenCV 3+ was released, it brought with it an enhanced deep learning (dnn) module. This module supports a larger number of frameworks like Caffe, TensorFlow, and Torch / PyTorch. This way, together with the possibility of making use of languages like Python, it becomes simple to classify images as much as:

- Load the Deep Learning model.

- Pre-process the image.

- Pass the image through the network and obtain its classification at the exit [6].

As we have already seen, with OpenCV 4.1, which was used in this project - and therefore will be shortened only to OpenCV in the rest of the work -, it is possible to make use of pre-trained networks with more popular frameworks. The advantage is that it will not be necessary to spend a lot of time training the network. OpenCV is not (and is not intended to be) a tool for training networks - there are already great frameworks available for this purpose. As a network (like a CNN) can be used as a

classifier, it makes logical sense that OpenCV has a Deep Learning module that we can easily take advantage of within the OpenCV ecosystem [12].

Installing OpenCV on Raspberry
OBS.: I used the Raspbian non-NOOB version

1st. Step - Install Prerequisites

```
$ sudo apt-get update # Faz Update de qualquer pacote já instalado

$ sudo apt-get upgrade # Faz Upgrade de qualquer pacote já instalado
```

```
# Instalar a ferramenta para ajustes no Build, assim como a Cmake

$ sudo apt-get install build-essential cmake git pkg-config
```

```
# Instala bibliotecas e pacotes que permitem ler vários tipos de imagens

$ sudo apt-get install libjpeg8-dev libtiff5-dev libjasper-dev libpng12-dev
```

```
# Instala algumas bibliotecas que permitem ler vários tipos de vídeos

$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev

$ sudo apt-get install libxvidcore-dev libx264-dev
```

```
# Instala o GTK para que possamos usar os recursos da GUI do OpenCV
$ sudo apt-get install libgtk2.0-dev
```

```
# Instala os pacotes que otimizam algumas funções do OpenCV, como as matrizes
$ sudo apt-get install libatlas-base-dev gfortran

//Instala o python e seus “development headers” e bibliotecas
$ sudo apt-get install python3-dev
```

2nd. Passo - OpenCV4 Download

```
$ cd ~
$ wget -O opencv.zip https://github.com/opencv/opencv/archive/4.0.0.zip
$ wget -O opencv_contrib.zip https://github.com/opencv/opencv_contrib/archive/4.0.0.zip

# Agora será necessário fazer o unzip dos downloads
$ unzip opencv.zip

$ unzip opencv_contrib.zip

# Por fim, para fins de facilidade faremos a troca dos nomes
$ mv opencv-4.0.0 opencv
$ mv opencv_contrib-4.0.0 opencv_contrib
```

It is important to use a virtual environment for Python 3 so that it is possible to maintain different environments for each step, and thus not generate conflicts or system overload as it is necessary to deal with hardware restrictions. For this, virtualenv and virtualenvwrapper will be used so that it is allowed to use Python 3 in the virtual environment, as will be seen below:

```
$ sudo pip install virtualenv virtualenvwrapper
$ sudo rm -rf ~/get-pip.py ~/.cache/pip
```

Now it will be necessary to relocate them inside the folder (~ / .profile).

```
# Ao invés de acessar o ~/.profile via “nano”, basta usar os comandos abaixo para fazer o ajuste via “bash” mesmo.
$ echo -e "\n# virtualenv and virtualenvwrapper" >> ~/.profile
$ echo "export WORKON_HOME=$HOME/.virtualenvs" >> ~/.profile
$ echo "export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3" >> ~/.profile
$ echo "source /usr/local/bin/virtualenvwrapper.sh" >> ~/.profile
```

To ensure that changes occur, simply enter in sequence: $ source ~ / .profile

Finally, just create the virtual environment so that you can allocate OpenCV and additional packages: $ mkvirtualenv cv -p python3

Note: The procedures above allow you to access the virtual environment using only with the $ workon cv command only if we do the steps above for ~ / .bashrc.

Finally, now that the virtual environment has been created, we can install the right libraries, prepare OpenCv for compilation and execute our Deep Learning model.

OpenCV makes use of vectors (Arrays) to represent the images and therefore it will be necessary to have the NumPy library installed in the virtual environment:

```
$ pip install numpy // Instala o Numpy
$ pip install imutils // Instala o Imutils
```

It will be necessary to download the "opencv_contrib" repository Without this repository, we will not have access to the "keypoint detectors" and "local invariant descriptors" (such as SIFT, SURF etc.) that were available in the OpenCV 2.4.X version. We will also be missing out on some of the latest features of OpenCV, such as text detection in natural images.

3rd Step - Configuring and compiling OpenCv within the virtual environment.

```
//Tenha certeza que os processos abaixo serão executados
dentro do cv
$ cd ~/opencv  // Retornando ao repositório opencv
$ mkdir build // criando a pasta "build"
$ cd build // Acessando o "build"

$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D
OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/mo
dules \
    -D ENABLE_NEON=ON \
    -D ENABLE_VFPV3=ON \
    -D BUILD_TESTS=OFF \
    -D OPENCV_ENABLE_NONFREE=ON \
    -D INSTALL_PYTHON_EXAMPLES=OFF \
    -D BUILD_EXAMPLES=OFF ..
```

At the end of the cmake run, the configuration should be in the following format:

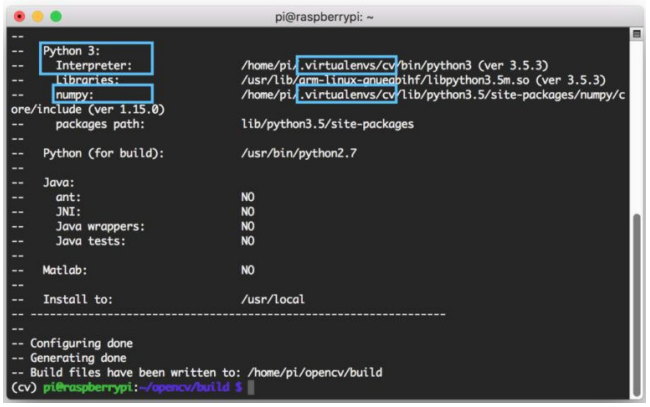

Figure 5 - End of Cmake run

Before you begin the compile I would suggest increasing your swap space. This will enable you to compile OpenCV with all four cores of the Raspberry Pi without the compile hanging due to memory exhausting.

Open up your /etc/dphys-swapfile file:
Install OpenCV 4 on your Raspberry PiShell
$ sudo nano /etc/dphys-swapfile
…and then edit the CONF_SWAPSIZE variable:

Install OpenCV 4 on your Raspberry PiShell

```
# set size to absolute value, leaving empty (default) then
uses computed value
#   you most likely don't want this, unless you have an
special disk situation
# CONF_SWAPSIZE=100
CONF_SWAPSIZE=1024
```

Notice that I'm increasing the swap from 100MB to 2048MB.
If you do not perform this step it's very likely that your Pi will hang.

From there, restart the swap service:
Install OpenCV 4 on your Raspberry PiShell
$ sudo /etc/init.d/dphys-swapfile stop
$ sudo /etc/init.d/dphys-swapfile start

Note: Increasing swap size is a great way to burn out your Raspberry Pi microSD card. Flash-based storage has a limited number of writes you can perform until the card is essentially unable to hold the 1's and 0's anymore. We'll only be enabling large swaps for a short period of time, so it's not a big deal. Regardless, be sure to backup your .img file after installing OpenCV + Python just in case your card dies unexpectedly early. You can read more about large swap sizes corrupting memory cards on this page. [13]

4th Step - Compiling the OpencCV
If this step is a problem, run it again but without the "-j4"

```
$ make -j4 // o comando j4 especifica que existe 4 núcleos de
processamento
```

5th Step - Installing OpenCV
Once executed DO NOT EXECUTE again

```
$ sudo make install
$ sudo ldconfig
```

6th Step - Sym-link between OpenCV and the virtual environment with Python

```
$ cd ~/.virtualenvs/cv/lib/python3.7/site-packages/
$ ln -s
/usr/local/python/cv2/python-3.7/cv2.cpython-35m-arm-linux-
gnueabihf.so cv2.so
$ cd ~
```

7th - OpenCV works test.

```
$ workon cv // executa o ambiente virtual
$ python
>>> import cv2
>>> cv2.__version__
'4.0.0'
```

```
>>> exit()
```

When using Raspberry Pi for Deep Learning there are two problems that we need to address:
-Restricted Memory;
- Limited processing speed.

Thus, one of the first paths to follow is the use of more efficient neural networks from a computational point of view, bringing less need for memory and processing. It is possible to find some options on the internet, but for the Raspberry Pi there are two which Adrian, on his personal website PyImageSearch, recommends, such as MobileNet and SqueezeNet. For practical reasons, SqueezeNet will be used in a first approach.

SqueezeNet is a convolutional neural network trained in more than one million images from the ImageNet database [1]. The network has 18 layers and can classify images in 1000 categories of objects, such as keyboard, mouse, pencil and many animals. As a result, the network learned resource-rich representations for a wide variety of images. The network has an image input size of 227 by 227. [15]

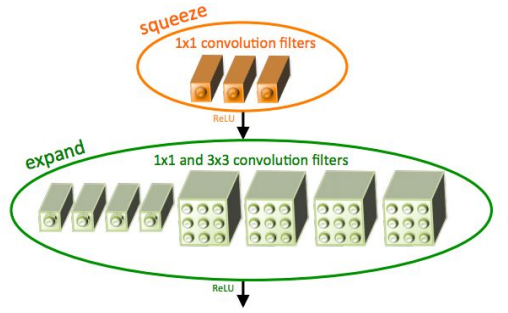

Figure 6. SqueezeNet diagram [1]

The most important thing is to highlight the work of Iandola et al. which stands out by its name: “SqueezeNet: AlexNet-level accuracy with 50x few parameters and <0.5MB model size”. According to his article, it is possible to reduce the size of the model by applying a new use of 1 × 1 and 3 × 3 convolutions, without fully connected layers. The end result is a model weighing 4.9 MB, which can be further reduced to <0.5 MB by the processing method (also called "weight pruning" and "sparsifying a model") [14].

To start this process we will first create a document with the suggestive name “pi_deep_learning.py” where it will have the following code:

```
# Inserindo as bibliotecas necessárias
import numpy as np
import argparse
import time
import cv2

# Construindo o “argument parse” e “parse the arguments”
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
	help="path to input image")
ap.add_argument("-p", "--prototxt", required=True,
	help="path to Caffe 'deploy' prototxt file")
ap.add_argument("-m", "--model", required=True,
	help="path to Caffe pre-trained model")
ap.add_argument("-l", "--labels", required=True,
	help="path to ImageNet labels (i.e., syn-sets)")
args = vars(ap.parse_args())
```

Attention:
--image : The path to the input image;
--prototxt: The path to a Caffe prototxt file which is essentially a plaintext configuration file following a JSON-like structure.
--model : The path to a pre-trained Caffe model. As stated above, you’ll want to train your model on hardware which packs much more punch than the Raspberry Pi — we can, however, leverage a small, pre-existing model on the Pi;
--labels : The path to class labels, in this case ImageNet “syn-sets” labels.

```
# load the class labels from disk
rows = open(args["labels"]).read().strip().split("\n")
classes = [r[r.find(" ") + 1:].split(",")[0] for r in rows]

# load the input image from disk
image = cv2.imread(args["image"])

# our CNN requires fixed spatial dimensions for our input
image(s)
# so we need to ensure it is resized to 227x227 pixels while
# performing mean subtraction (104, 117, 123) to normalize
the input;
# after executing this command our "blob" now has the shape:
# (1, 3, 227, 227)
blob = cv2.dnn.blobFromImage(image, 1, (227, 227), (104,
117, 123))

# load our serialized model from disk
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"],
args["model"])

# set the blob as input to the network and perform a
forward-pass to
# obtain our output classification
net.setInput(blob)
start = time.time()
preds = net.forward()
end = time.time()
print("[INFO] classification took {:.5} seconds".format(end -
start))

# sort the indexes of the probabilities in descending order
(higher
# probabilitiy first) and grab the top-5 predictions
preds = preds.reshape((1, len(classes)))
```

```
idxs = np.argsort(preds[0])[::-1][:5]

# loop over the top-5 predictions and display them
for (i, idx) in enumerate(idxs):
        # draw the top prediction on the input image
        if i == 0:
                text = "Label: {},
{:.2f}%".format(classes[idx],
                        preds[0][idx] * 100)
                cv2.putText(image, text, (5, 25),
cv2.FONT_HERSHEY_SIMPLEX,
                        0.7, (0, 0, 255), 2)

        # display the predicted label + associated probability
to the
        # console
        print("[INFO] {}. label: {}, probability:
{:.5}".format(i + 1,
                classes[idx], preds[0][idx]))

# display the output image
cv2.imshow("Image", image)
cv2.waitKey(0)
```

In general, you should: Never use your Raspberry Pi to train a neural network. Only use your Raspberry Pi to deploy a pre-trained deep learning network.
The Raspberry Pi does not have enough memory or CPU power to train these types of deep, complex neural networks from scratch.

In fact, the Raspberry Pi barely has enough processing power to run them — as we'll find out in next week's blog post you'll struggle to get a reasonable frame per second for video processing applications.

If you're interested in embedded deep learning on low cost hardware, I'd consider looking at optimized devices such as NVIDIA's Jetson TX1 and TX2. These boards are designed to execute neural networks on the GPU and provide real-time (or as close to real-time as possible) classification speed.

We'll be benchmarking our Raspberry Pi for deep learning against two pre-trained deep neural networks:
- GoogLeNet
- SqueezeNet

RESULTS

GoogLeNet took about 1.7304 seconds and SqeezeNet just 0.92073 seconds.

```
$ python pi_deep_learning.py --prototxt
models/bvlc_googlenet.prototxt \
        --model models/bvlc_googlenet.caffemodel
--labels synset_words.txt \
        --image images/barbershop.png
```

```
[INFO] loading model...
[INFO] classification took 1.7304 seconds
[INFO] 1. label: barbershop, probability: 0.70508
[INFO] 2. label: barber chair, probability: 0.29491
[INFO] 3. label: restaurant, probability: 2.9732e-06
[INFO] 4. label: desk, probability: 2.06e-06
[INFO] 5. label: rocking chair, probability: 1.7565e-06
```

```
$ python pi_deep_learning.py --prototxt
models/squeezenet_v1.0.prototxt \
        --model models/squeezenet_v1.0.caffemodel
--labels synset_words.txt \
        --image images/barbershop.png
[INFO] loading model...
[INFO] classification took 0.92073 seconds
[INFO] 1. label: barbershop, probability: 0.80578
[INFO] 2. label: barber chair, probability: 0.15124
[INFO] 3. label: half track, probability: 0.0052873
[INFO] 4. label: restaurant, probability: 0.0040124
[INFO] 5. label: desktop computer, probability:
0.0033352
```

The results from the experiments show that running object detection on low end CPU devices is very slow. The purpose of this paper was to find out if a Raspberry Pi is suitable to use as hardware in a real time object detection system. But different applications have different requirements in speed and accuracy. In one instance you might need fast detection but don't care about small or distant objects. In another situation, speed might be less important but better detection is. When implementing an object detector on a low end device, this speed and accuracy trade-off most likely must be done and the results shown [16].

As our results demonstrated we were able to get up to 0.9 frames per second, which is not fast enough to constitute real-time detection. That said, given the limited processing power of the Pi, 0.9 frames per second is still reasonable for some applications [14].

If your use case involves low traffic object detection where the objects are slow moving through the frame, then you can certainly consider using the Raspberry Pi for deep learning object detection. However, if you are developing an application that involves many objects that are fast moving, you should instead consider faster hardware.

It is important to note that the Raspberry Pi may not perform in the same way as a high-cost device, however it is a valid tool to embark on pre-trained deep learning and provide satisfactory results for people who are willing to delve into the area of artificial intelligence.

REFERENCES

1. Marcello Santos da Fonseca. URL: http://www.ic.uff.br/~aconci/limiarizacao.htm.

2. GAEA. URL: https://gaea.com.br/afinal-o-que-e-deep-learning/ .

3. Nuno Ferreira and Philip Costa-Hibberd. URL: https://zanders.eu/en/latest-insights/rpa-cutting-through-the-noise/ .

4. Peiliang Li, Tong Qin, and Shaojie Shen. URL: https://arxiv.org/abs/1807.02062.

5. OPENSOURCE. URL: https://opensource.org/.

6. Adrian Rosebrock. URL: https://www.pyimagesearch.com/about/.

7. Sarthak Jain. URL:https://medium.com/nanonets/how-to-easily-detect-objects-with-deep-learning-on-raspberrypi-225f29635c74.

8. Antonio Carlos Gil. URL:http://www.urca.br/itec/images/pdfs/modulo%20v%20-%20como_elaborar_projeto_de_pesquisa_-_antonio_carlos_gil.pdf.

9. Carlos Magno da Silva Xavier. URL:https://pmkb.com.br/artigos/as-metodologias-de-gerenciamento-de-projetos/.

10. Mariana González . URL: https://blog.idwall.co/o-que-e-machine-learning/.

11. OPENMV. URL: https://openmv.io/.

12. Adrian Rosebrock. URL: https://www.pyimagesearch.com/2017/08/21/deep-learning-with-opencv/.

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

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

15. MATHWORKS. URL: https://www.mathworks.com/help/deeplearning/ref/squeezenet.html

16. Adam Gunnarsson. URL: https://www.diva-portal.org/smash/get/diva2:1361039/FULLTEXT01.pdf

ABOUT ME

Complete name of Author: Bruno Bueno Furquim
E-mail: brunoquim@hotmail.com
Institutional e-mail: bruno.furquim@aluno.ufabc.edu.br

ACKNOWLEDGMENTS

I would like to thank the support and attention given by the co-authors and the Federal University of ABC. To my brother Felipe, my father Eduardo and my mother Marcia for the inspiration.