Teach your Raspberry Pi – "Yeah, world"

Document number: ARM-ECM-0735791 Version: 1.0

Date of Issue: 02/01/2018

Abstract

Program your Raspberry Pi using artificial intelligence to recognize your gestures and respond with a cheer. This beginner-level tutorial provides step-by-step instructions and explanations to turn your Raspberry Pi into a gesture recognition device. Follow this guide and learn how to apply some fundamental principles of machine learning in a fun and interactive way.

Keywords

Machine learning, Artificial intelligence, Raspberry Pi, learning Arm, tutorial, gesture recognition, neural network.

Contents

1 Overview

Did you know that you can train AI on your Raspberry Pi without any extra hardware or accelerators? In this guide we'll use TensorFlow to train a Raspberry Pi to burst into applause whenever you raise your hands in the air using nothing more than a camera and the Pi's on-board Arm CPU. It even works on the Pi Zero!

2 Install libraries

2.1 Install TensorFlow

For a Raspian base install the only dependency that you need to add is TensorFlow from Google's binaries. First, install some TensorFlow prerequisites by entering the following in the command line:

```
sudo apt-get install libblas-dev liblapack-dev python-dev libatlas-base-dev gfortran python-setuptools python-h5py
```

The exact URL of the current TensorFlow build varies between versions. Go to the TensorFlow Raspberry Pi 3 build page or the TensorFlow Raspberry Pi Zero build page to find the current one:

```
# Replace this URL with the correct version for your Pi 3 or Zero:
sudo pip2 install http://ci.tensorflow.org/view/Nightly/job/nightly-pi/lastSuccessfulBuild/artifact/output-artifacts/tensorflow-1.6.0rc1-cp27-none-any.whl
```

2.2 Install Arm's training scripts

Download or clone our ML examples repository from GitHub.

```
git clone https://github.com/ARM-software/ML-examples.git
cd ML-examples/yeah-world
```

There is nothing special about these scripts, they are designed to be easy to understand and modify. Feel free to explore and hack them with your own changes.

2.3 Advanced information

The python source code is designed to be straightforward to follow:

- record.py captures images from the camera and saves them to disk at the end.
- train.py loads saved images, converts them to features and trains a classifier on those features.

- run.py captures images from the camera, converts them to features, classifies them and then plays a random sound if they belong to the first category.

The three helper files are to keep the above three files as readable as possible:

- camera.py initializes the picamera module and optionally fluctuates the exposure and white balance during recording.
- pinet.py loads the pretrained MobileNet with TensorFlow and uses it to convert one image at a time into a set of features.
- randomsound.py uses pygame to play a random sound file from a directory.

3 Train the AI with your own data

Teaching a neural network to reliably recognize gestures from every human at every location and in every light condition on the planet presents a huge challenge that requires lots of varied examples and very careful training. However, teaching a network to recognize your gestures in your environment is very easy.

In fact, it is so easy that we can do it right now on the Raspberry Pi itself, and with no GPU or accelerator to help with the processing required.
The training process has three steps:

1. Set up your camera and environment.
2. Record some new data.
3. Train a network on the data.

4 Set up your camera and environment

Neural networks can learn to detect and generalize small gestural changes that only affect a few dozen pixels on the screen, but it takes much more data and training time than we want to spend.

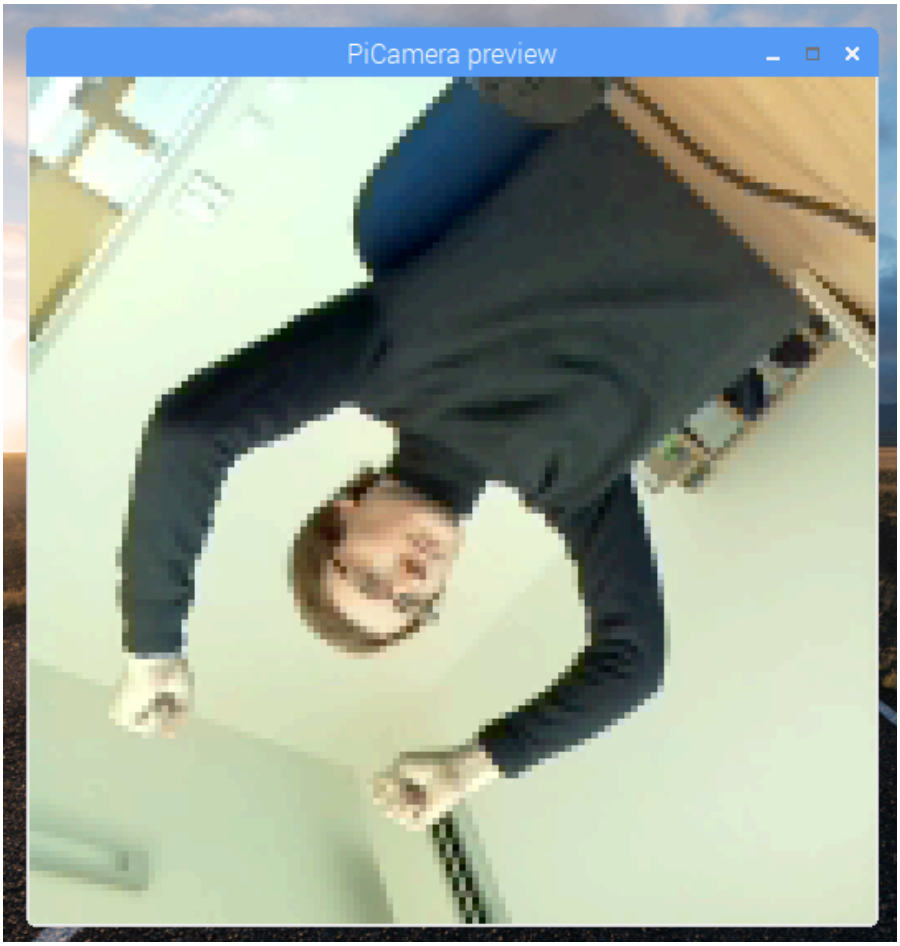
To make it easier for the neural network to learn from limited data, place the camera so that:

- Your gestures are roughly in the center of the camera's view.
- You are close enough that a significant number of pixels are affected by the gesture.

A good rule of thumb is to place the camera in front of you so that your hands can reach the edges of the frame when they are out-stretched. To preview what the record and run scripts will see, use the preview.py script with this command:

```
python preview.py
```

The setup in this example preview is acceptable:



It does not matter if the camera is the right way up as long as it remains consistent. For best results, ensure that:

- There is not too much activity happening in the background.
- There is good lighting and a good view of the subject.
- Your arms are in the picture even when they are extended.

When you are happy with the positioning of the camera and the lighting either close the window or press ctrl-c at the console to exit the preview app.

4.1 Teaching by example

An AI is only as good as its data! When we train an AI we teach it by example – in this case you are giving it one set of pictures with your hands up and another set with your hands down. Each frame of video is one of these pictures.

Ideally the AI learns that the principle difference between these sets of pictures is the position of your hands. However, depending on the examples it might learn to recognise coincidental things, such as:

The angle with which you are leaning, as people tend to lean back slightly when raising their hands while seated.

The overall lighting is brighter or dimmer, if your raised hands cast a shadow on the camera. Whether someone in the background is working in word or watching youtube, if their monitor is visible and they happened to change when you did.

Here are some tips to help you get good results:

- Record variations of the gesture. With your hands in the air lean left and right, wave them higher and lower, twist left and right. Recording variations helps the network recognize more of the slight changes it will see in the real world.
- Record the same variations without the gesture. Try to do the same actions but with your hands down. This teaches the neural network that the chief difference between the cheering and not cheering is the position of your hands and nothing else.
- Record random things you want ignored. These can include things like scratching your head, standing up and walking away, jumping up and down, hiding, and covering the camera.

Above all, be consistent. If you are teaching the network to detect your hands in the air, keep them up there in every frame of the "hands in the air" recording. Vary the position and angle but do not take them all the way down and then put them up again, or you will include frames that look just like the "hands down" frames. This will force the network to assume the difference in those cases is, for example how your clothing is folded, or the reflection of light on your glasses.

5 Record data

If you are using a Raspberry Pi Zero or SSH / VNC, the live preview window can drop the frame rate during recording below 30 fps. That means that less data is collected, which is a bad thing. Run this command in the terminal before entering further commands to ensure recording runs as close to 30 fps as possible:

```
unset DISPLAY
```

If you keep the preview window open during training you will notice that the lighting and color balance fluctuates rapidly. This is intentional! The camera exposure and white balance is set randomly for each frame during training. This technique, which is a variant of domain randomization, shows lots of variance in something that might otherwise trick or confuse the AI (in this case, lighting and color balance). It is actually more common to artificially add this to the data at training time, but this is not covered here.

Follow these steps to record your data:

1. Record some actions with your hands in the air. When you run the following command, it will count down from three and then record you for 15 seconds:

```
python record.py example/yeah 15
```

Suggested actions: wave your hands in the air like you are cheering. Lean left and right while doing so. Turn left and right. Lean forwards and backwards. Keep your hands pumping in the air above your head like a lunatic the whole time.

2. Record the same behavior, but with your hands down after entering this command:

```
python record.py example/sitting 15
```

Suggested actions: Lean left and right, turn left and right, lean forwards and backwards. Keep your hands down or at keyboard height the whole time.

3. Record some random activities that you want the network to ignore and therefore not be surprised by after entering this command:

```
python record.py example/random 15
```

Suggested actions: Cover your face, scratch your head, reach for and cover the camera, stand up and walk away, come back and sit down, get up and walk the other way.

After these steps you will have three files of similar size that contain the three types of data that you recorded, as shown after running the command here:

```
$ ls -lh example/
```

Files:

```
total 166M
-rw-r--r-- 1 pi pi 49M Nov 27 10:59 random
-rw-r--r-- 1 pi pi 64M Nov 27 10:59 sitting
-rw-r--r-- 1 pi pi 54M Nov 27 10:57 yeah
```

6 Train a network on this data

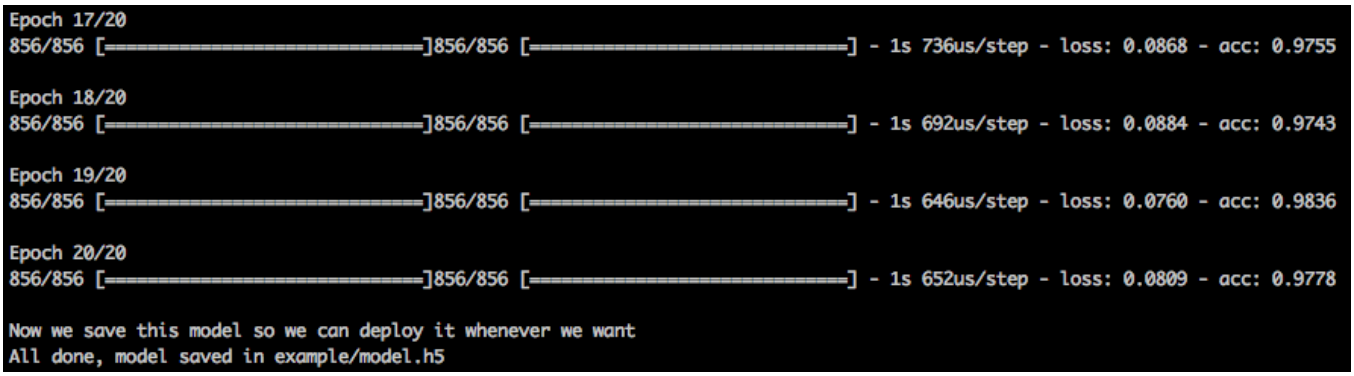
Running the train.py script will load your data files into memory and train a neural network to distinguish between them. There is good support in Keras and TensorFlow for training from disk without loading every example into RAM, but this is a quick and easy way to get us started. Enter the following command to run the script:

```
python train.py example/model.h5 example/yeah example/sitting
example/random
```

The first argument is the name of the model file to save the neural network to, which is model.h5. You then list the different behavior video files that you have recorded. It is important that the first one is

the file that contains your cheering gesture. All of the others are gestures the network should learn to ignore.

The slowest part of training is loading and converting the data. After that it should compute 20 iterations and stop. In the example shown in this screenshot, it converges to an accuracy of 98%.



6.1 What is going on in this process?

Here, the script takes the training files in turn and load all the frames from each. This means that you need enough RAM to hold them all. Once the images are loaded, they are passed through a neural network (MobileNet) that has been pre-trained on the ImageNet dataset.

This has already learned how to extract useful features from images, such as edges, texture, and shapes. This is the slowest part of training, and if we were artificially adding noise and variation to the images (domain randomization) it would be even slower and longer, which is why we took a short cut here.

Having converted every frame into an array of features, these features are used to train a small classifier network that learns to predict the class (in this case the 0-based input file index) it came from.

Each epoch is one cycle through every frame. To prevent it latching on to overly-specific things about individual frames, you add some random noise to the features. This may prevent it from reaching 100% accuracy during training but will make it more accurate when run on previously unseen images, for example, in real use.

6.2 Advanced information

MobileNet was introduced by Google in 2017. It is a convolutional neural network that has been optimized for size and speed at the expense of accuracy. MobileNet is trained on the well-known ImageNet dataset, in which 1 million images are split into 1000 different classes. The MobileNet included here has had that final layer that classifies into 1000 categories removed. Instead, the train.py script uses Keras to build a new layer that classifies into however many categories are passed to train.py.

If you read the code in the train.py file you will notice the classifier model is as simple as can be, the data is only augmented with gaussian noise instead of rotations, transforms and occlusions and the learning rates and batch size are left at Keras defaults. A validation split with early stopping is not used

and domain randomization is applied to the data at source and not varied across epochs during training. There is a lot of room for improvement here so you can experiment to try and improve your results. This example is deliberately clean and sparse to keep it easy to understand and quick to train even on a Pi Zero.

6.3 Run your new network

Now you can run a neural network trained specifically on your environment with a single command:

```
python run.py example/model.h5
```

Wave your hands in the air. Does it cheer? Even if your audio is not configured, the console will print the class predicted and display either:

- 0: Cheering!
- 1: Sitting in view but not cheering.
- 2: Random behaviors such as standing, being out of the picture, unexpected lighting conditions and so on.

There is a lot more that can be done than cheering! You can replace the files in the folder named "sounds" and train a model to recognise completely different gestures, objects or situations. For example, you can teach a Pi to:

- Greet you by name.
- Set off an alarm when someone is in view. For example, try hiding beneath a cardboard box or behind a cushion and sneaking up on it.
- Shout "THE COFFEE IS READY" when the coffee is ready.
- Tell the dog to get off the sofa when the dog gets on the sofa.

With a little python programming, you can also make your Pi do more interesting things in response to camera input such as setting GPIO pins, sending alerts and emails. The applications are limited only by your imagination!

6.4 How did you get on?

Tell us about your successes, failures, and your own machine learning projects. Email us at: ml@arm.com to start the conversation.

6.5 Coming soon

This is the first guide in a series of training your own gesture recognition on the Raspberry Pi using machine learning. Although fun, you will quickly see some of the limitations of this approach:

- Changes location, background and even clothing can throw detection off.
- Recording longer example videos can cause out of memory errors during training.

- Training networks to recognise subtle gestures that don't cause a lot of visual change in the image is difficult.

We will explore these topics in future guides, so look out for guides on:

- Enhancing gesture recognition - training networks to recognise gestures in many situations and how to use larger datasets.
- Teachable Pi - a set of tools for building gesture recognition into Pi-based projects, with installation and usage instructions.