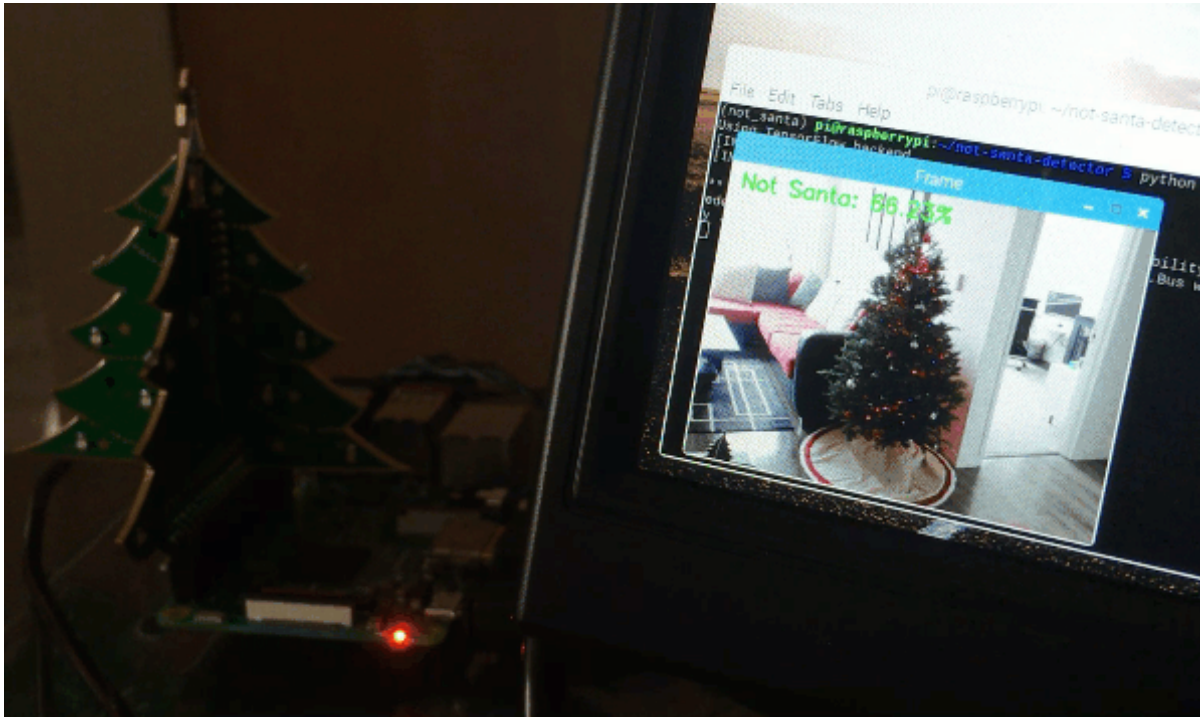


Keras and deep learning on the Raspberry Pi

by [Adrian Rosebrock](#) on December 18, 2017 in [Deep Learning](#), [Keras](#), [Raspberry Pi](#), [Tutorials](#)



Today's blog post is the *most fun* I've ***EVER*** had writing a PyImageSearch tutorial.

It has everything we have been discussing the past few weeks, including:

- Deep learning
- Raspberry Pis
- 3D Christmas trees
- References to HBO's Silicon Valley "Not Hotdog" detector
- **Me dressing up as Santa Clause!**

In keeping with the Christmas and Holiday season, I'll be demonstrating how to take a deep learning model (trained with Keras) and then deploy it to the Raspberry Pi.

But this isn't *any* machine learning model...

This image classifier has been *specifically trained* to detect if Santa Claus is in our video stream.

And if we do detect Santa Claus...

Well. I won't spoil the surprise (but it does involve a 3D Christmas tree and a jolly tune).

Enjoy the tutorial. Download the code. Hack with it.

And most of all, *have fun!*

Looking for the source code to this post?

[Jump right to the downloads section.](#)

Keras and deep learning on the Raspberry Pi

Today's blog post is a *complete guide* to running a deep neural network on the Raspberry Pi using Keras.

I've framed this project as a *Not Santa* detector to give you a practical implementation (and have some fun along the way).

In the first part of this blog post, we'll discuss what a *Not Santa* detector is (just in case you're unfamiliar with [HBO's Silicon Valley "Not Hotdog" detector](#) which has developed a cult following).

We'll then configure our Raspberry Pi for deep learning by installing TensorFlow, Keras, and a number of other prerequisites.

Once our Raspberry Pi is configured for deep learning we'll move on to building a Python script that can:

1. Load our Keras model from disk
2. Access our Raspberry Pi camera module/USB webcam
3. Apply deep learning to detect if Santa Clause is in the frame
4. Access our GPIO pins and play music if Santa is detected

These are my favorite types of blog posts to write here on PyImageSearch as they integrate a bunch of techniques we've discussed, including:

- [Deep learning on the Raspberry Pi](#)
- [Accessing your Raspberry Pi camera module/USB webcam](#)
- [Working with GPIO + computer on the Raspberry Pi](#)

Let's get started!

What is a *Not Santa* detector?



Figure 1: The Not Hotdog detector app from HBO' s Silicon Valley.

A *Not Santa* detector is a play off [HBO' s Silicon Valley](#) where the characters create a smartphone app that can determine if an input photo is a "hot dog" or is "not a hot dog" :

The show is clearly poking fun at the Silicon Valley startup culture in the United States by:

1. Preying on the hype of machine learning/deep learning
2. Satirically remaking on the abundance of smartphone applications that serve little purpose (but the creators are convinced their app will "change the world")

I decided to have some fun myself.

Today we are creating a *Not Santa* detector that will detect if Santa Claus is in an image/video frame.

For those unfamiliar with [Santa Claus](#) (or simply Santa" for short), he is a jolly, portly, white-bearded, fictional western culture figure who delivers presents to boys and girls while they sleep Christmas Eve

However, this application is *not* totally just for fun and satire!

We' ll be learning some practical skills along the way, including how to:

1. Configure your Raspberry Pi for deep learning
2. Install Keras and TensorFlow on your Raspberry Pi
3. Deploy a pre-trained Convolutional Neural Network (with Keras) to your Raspberry Pi
4. Perform a given action once a positive detection has occurred

But before we can write a line of code, let' s first review the hardware we need.

What hardware do I need?



Figure 2: The *Not Santa* detector setup includes the Raspberry Pi 3, speakers, 3D Christmas tree, and a webcam (not pictured). The Pi has LeNet implemented with Keras in a Python script in order to detect Santa.

In order to follow along *exactly* with this tutorial (with no modifications) you'll need:

- A [Raspberry Pi 3](#) (or the [Raspberry Pi 3 Starter Kit](#) which I also highly recommend)
- A [Raspberry Pi camera module](#) or a USB camera. For the purposes of this tutorial, I went with the [Logitech C920](#) as it's a great camera for the price (and having a USB cable gives you a bit of extra room to work with versus the short ribbon on the Pi camera)
- The [3D Christmas Tree](#) for the Raspberry Pi (designed by [Rachel Rayns](#))
- A set of speakers — I recommend [these from Pi Hut](#), the [stereo speakers from Adafruit](#), or if you're looking for something small that still packs a punch, you could get [this speaker from Amazon](#)

Of course, you do not need all these parts.

If you have just a Raspberry Pi + camera module/USB camera you'll be all set (but you will have to modify the code so it doesn't try to access the GPIO pins or play music via the speakers).

Your setup should look similar to mine in **Figure 2** above where I have connected my speakers, 3D Christmas Tree, and webcam (not pictured as it's off camera).

I would also recommend hooking up an HDMI monitor + keyboard to test your scripts and debug them:



Figure 3: My deep learning setup includes the Raspberry Pi and components along with a keyboard, mouse, and small HDMI display. With this setup, we will surely catch Santa delivering presents in front of my tree.

In the image above, you can see my Raspberry Pi, HDMI, keyboard, and Christmas critter friends keeping me company while I put together today's tutorial.

How do I install TensorFlow and Keras on the Raspberry Pi?



Figure 4: We're going to use Keras with the TensorFlow backend on the Raspberry Pi to implement a deep learning Not Santa detector.

Last week, we learned [how to train a Convolutional Neural Network using Keras](#) to determine if Santa was in an input image.

Today, we are going to take the pre-trained model and deploy it to the Raspberry Pi.

As I've mentioned before, the Raspberry Pi is not suitable for training a neural network (outside of "toy" examples). However, the Raspberry Pi can be used to *deploy* a neural network [once it has already been trained](#) (provided the model can fit into a small enough memory footprint, of course).

I'm going to assume you have already installed OpenCV on your Raspberry Pi.

If you have not installed OpenCV on your Raspberry Pi, start by using [this tutorial](#) where I demonstrate how to optimize your Raspberry Pi + OpenCV install for speed (leading to a 30%+ increase in performance).

Note: This guide will not work with Python 3 — **you’ ll instead need to use Python 2.7.** I’ ll explain why later in this section. Take the time now to configure your Raspberry Pi with Python 2.7 and OpenCV bindings. In Step #4 of the [Raspberry Pi + OpenCV installation guide](#), be sure to make a virtual environment with the `-p python2` switch.

From there, I recommend increasing the swap space on your Pi. Increasing the swap will enable you to use the Raspberry Pi SD card for additional memory (a critical step when trying to compile and install large libraries on the memory-limited Raspberry Pi).

To increase your swap space, open up `/etc/dphys-swapfile` and then edit the `CONF_SWAPSIZE` variable:

Keras and deep learning on the Raspberry Pi

Python

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

Notice that I am increasing the swap from 100MB to 1024MB.

From there, restart the swap service:

Keras and deep learning on the Raspberry Pi

Python

```
1 $ sudo /etc/init.d/dphys-swapfile stop
2 $ sudo /etc/init.d/dphys-swapfile start
```

Note: Increasing swap size is a great way to burn out your memory card, so **be sure to revert this change and restart the swap service when you’ re done.** You can read more about large sizes corrupting memory cards [here](#).

Now that your swap size has been increased, let’ s get started configuring our development environment.

To start, create a Python virtual environment named `not_santa` using **Python 2.7** (I’ ll explain why Python 2.7 once we get to the TensorFlow install command):

Keras and deep learning on the Raspberry Pi

Shell

```
1 $ mkvirtualenv not_santa -p python2
```


Notice here how the `-p` switch points to `python2`, indicating that Python 2.7 will be used for the virtual environment.

If you are new to Python virtual environments, how they work, and why we use them, please refer to [this guide to help get you up to speed](#) as well as [this excellent virtualenv primer from RealPython](#).

You' ll also want to make sure you have sym-linked your `cv2.so` bindings into your `not_santa` virtual environment (if you have not done so yet):

Keras and deep learning on the Raspberry Pi

Python

```
1 $ cd ~/.virtualenvs/not_santa/lib/python2.7/site-packages
2 $ ln -s /usr/local/lib/python2.7/site-packages/cv2.so cv2.so
```

Again, make sure you have compiled OpenCV with **Python 2.7** bindings.

You' ll want to double-check your path to your `cv2.so` file as well, just in case your install path is slightly different than mine.

If you compiled Python 3 + OpenCV bindings, created the sym-link, and then tried to `import cv2` into your Python shell, you will get a confusing traceback saying that the import failed.

Important: For these next few `pip` commands, be sure that you' re in the `not_santa` environment (or your Python environment of choice), otherwise you' ll be installing the packages to your Raspberry Pi' s system Python.

To enter the environment simply use the `workon` command at the bash prompt:

Keras and deep learning on the Raspberry Pi

Shell

```
1 $ workon not_santa
```

From there, you' ll see "`(not_santa)`" at the beginning of your bash prompt.

Ensure you have NumPy installed in the `not_santa` environment using the following command:

Keras and deep learning on the Raspberry Pi

Shell

```
1 $ pip install numpy
```

Since we'll be accessing the GPIO pins for this project we'll need to install both [RPi.GPIO](#) and [gpiozero](#):

Keras and deep learning on the Raspberry Pi

Shell

```
1 $ sudo pip install RPi.GPIO gpiozero
```

We are now ready to install TensorFlow on your Raspberry Pi.

The problem is that *there is not an official (Google released) TensorFlow distribution*.

We can follow the long, arduous, painful process of [compiling TensorFlow from scratch](#) on the Raspberry Pi...

Or we can use the pre-compiled binaries created Sam Abrahams (published on [GitHub](#)).

The problem is that there are only two types of pre-compiled TensorFlow binaries:

1. One for Python 2.7
2. And another for Python 3.4

The Raspbian Stretch distribution (the latest release of the Raspbian operating system at the time of this writing) ships with Python 3.5 — we, therefore, have a version mismatch.

To avoid any headaches between Python 3.4 and Python 3.5, I decided to stick with the Python 2.7 install.

While I would have liked to use Python 3 for this guide, the install process would have become more complicated (I could easily write multiple posts on installing TensorFlow + Keras on the Raspberry Pi, but since installation is not the main focus of this tutorial, I decided to keep it more straightforward).

Let's go ahead and install TensorFlow for Python 2.7 using the following commands:

Keras and deep learning on the Raspberry Pi

Shell

```
1 $ wget https://github.com/samjabrahams/tensorflow-on-raspberry-pi/releases/download/1.1.0/tensorflow-1.1.0-cp27-none-linux_armv7l.whl
2 $ pip install tensorflow-1.1.0-cp27-none-linux_armv7l.whl
```

Note: You will need to expand the code block above to copy the full URL. I recommend pressing the "<>" button before you copy the commands.

Once TensorFlow compiles and installs (which took around an hour on my Raspberry Pi) you'll need to install HDF5 and h5py. These libraries will allow us to load our pre-trained model from disk:

Keras and deep learning on the Raspberry Pi

Shell

```
1 $ sudo apt-get install libhdf5-serial-dev
2 $ pip install h5py
```

I Install HDF5 + h5py without the `time` command running so I can't remember the exact amount of time it took to install, but I believe it was around 30-45 minutes.

And finally, let's install Keras and the other prerequisites required for this project:

Keras and deep learning on the Raspberry Pi

Shell

```
1 $ pip install pillow imutils
2 $ pip install scipy --no-cache-dir
3 $ pip install keras==2.1.5
```

The SciPy install in particular will take a few hours so make sure you let the install run overnight/while you're at work.

It is **very important** that you install Keras version 2.1.5 for compatibility with TensorFlow 1.1.0.

To test your configuration, open up a Python shell (while in the `not_santa` environment) and execute the following commands:

Keras and deep learning on the Raspberry Pi

Shell

```
1 $ workon not_santa
2 $ python
3 >>> import h5py
4 >>> from gpiozero import LEDBoard
5 >>> from gpiozero.tools import random_values
6 >>> import cv2
7 >>> import imutils
8 >>> import keras
9 Using TensorFlow backend.
10 >>> print("OpenCV version: {}".format(cv2.__version__))
11 'OpenCV version: 3.3.1'
12 >>> print("Keras version: {}".format(keras.__version__))
13 'Keras version: 2.1.5'
14 >>> exit()
```

If all goes as planned you should see Keras imported using the TensorFlow backend.

As the output above demonstrates, you should also double-check that your OpenCV bindings (`cv2`) can be imported as well.

Finally, ***don' t forget to reduce your swap size*** from 1024MB back down to 100MB by:

1. Opening up `/etc/dphys-swapfile` .
2. Resetting `CONF_SWAPSIZE` to 100MB.
3. Restarting the swap service (as we discussed earlier in this post).

As mentioned in the note above, setting your swap size back to 100MB is important for memory card longevity. If you skip this step, you may encounter memory corruption issues and a decreased lifespan of the card.

Running a Keras + deep learning model on the Raspberry Pi

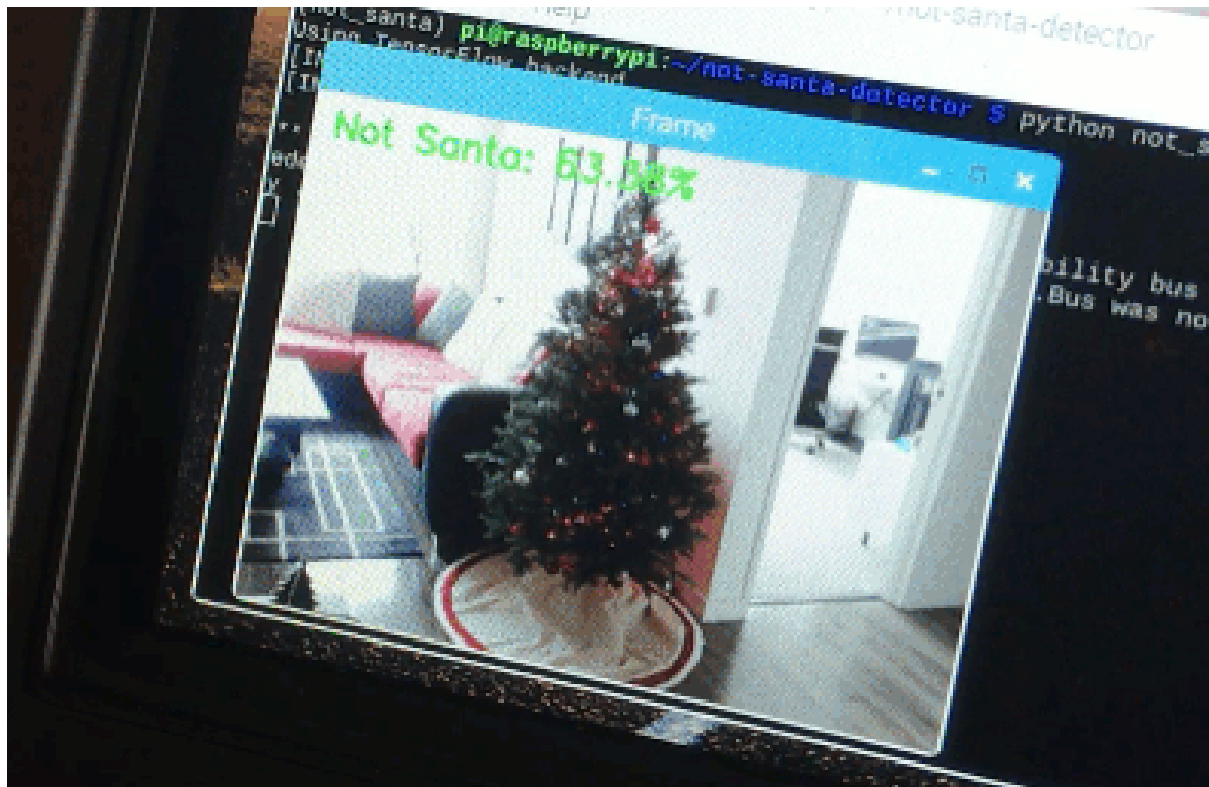


Figure 5: Running a deep learning model on the Raspberry Pi using Keras and Python.

We are now ready to code our *Not Santa* detector using Keras, TensorFlow, and the Raspberry Pi.

Again, I' ll be assuming you have the same hardware setup as I do (i.e., 3D Christmas Tree and speakers) so if your setup is different you' ll need to hack up the code below.

To get started, open a new file, name it `not_santa_detector.py` , and insert the following code:

Keras and deep learning on the Raspberry Pi

Python

```
1 # import the necessary packages
2 from keras.preprocessing.image import img_to_array
3 from keras.models import load_model
4 from gpiozero import LEDBoard
5 from gpiozero.tools import random_values
6 from imutils.video import VideoStream
7 from threading import Thread
8 import numpy as np
9 import imutils
10 import time
11 import cv2
12 import os
```

Lines 2-12 handle our imports, notably:

- `keras` is used to preprocess input frames for classification and to load our pre-trained model from disk.
- `gpiozero` is used to access the 3D Christmas tree.
- `imutils` is used to access the video stream (whether Raspberry Pi camera module or USB).
- `threading` is used for non-blocking operations, in particular when we want to light the Christmas tree or play music while not blocking execution of the main thread.

From there, we' ll define a function to light our [3D Christmas tree](#):

Keras and deep learning on the Raspberry Pi

Python

```
14 def light_tree(tree, sleep=5):
15     # loop over all LEDs in the tree and randomly blink them with
16     # varying intensities
17     for led in tree:
18         led.source_delay = 0.1
19         led.source = random_values()
20
21     # sleep for a bit to let the tree show its Christmas spirit for
22     # santa clause
23     time.sleep(sleep)
24
25     # loop voer the LEDs again, this time turning them off
26     for led in tree:
27         led.source = None
28         led.value = 0
```

Our `light_tree` function accepts a `tree` argument (which is assumed be an `LEDBoard` object).

First, we loop over all LEDs in the `tree` and randomly light each of the LEDs to create a “twinkling” effect (**Lines 17-19**).

We leave the lights on for a period of time for some holiday spirit (**Line 23**) and then we loop over the LEDs again, this time turning them off (**Lines 26-28**).

An example of the 3D Christmas tree lights turned on can be seen below:

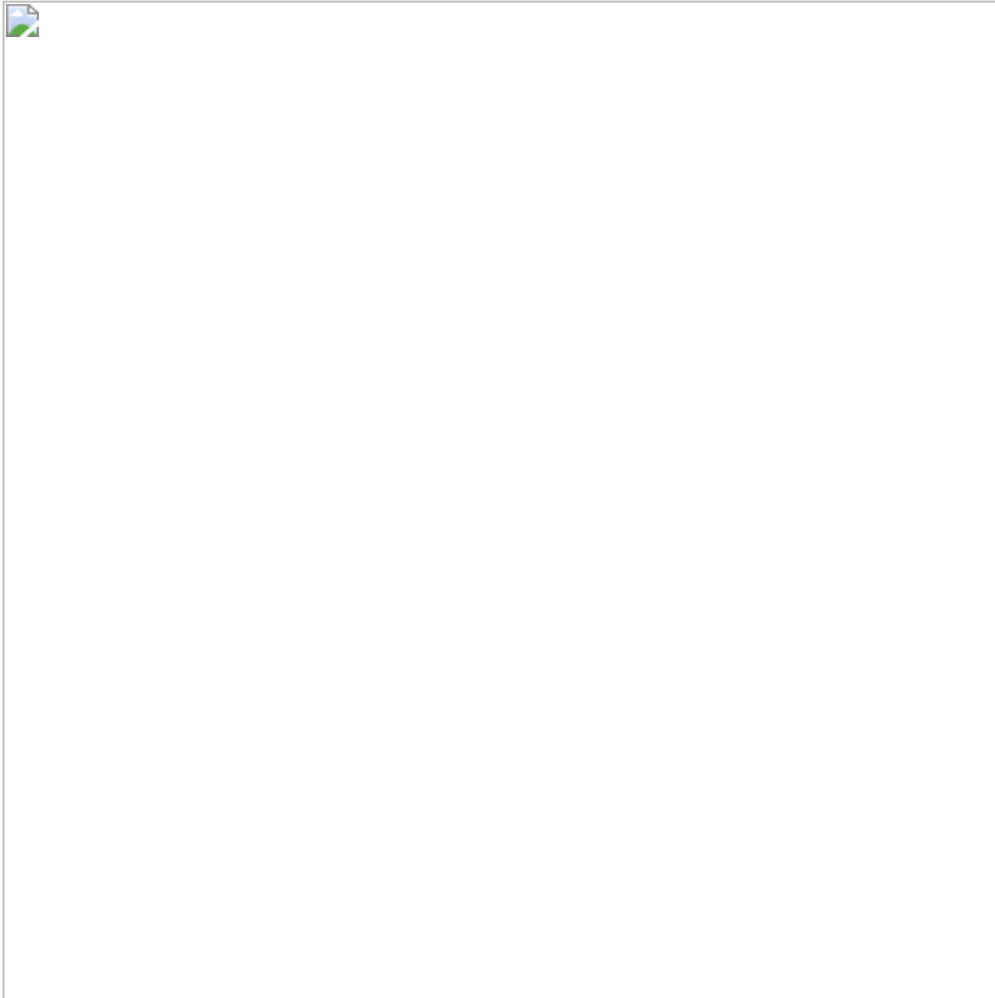


Figure 6: The 3D Christmas tree for the Raspberry Pi. You can get yours from [Pi Hut](#) (photo credit: Pi Hut).

Our next function handles playing music when Santa is detected:

Keras and deep learning on the Raspberry Pi

Python

```
30 def play_christmas_music(p):
31     # construct the command to play the music, then execute the
32     # command
33     command = "aplay -q {}".format(p)
34     os.system(command)
```

In the `play_christmas_music` function, we make a system call to the `aplay` command which enables us to play a music file as we would from the command line.

Using the `os.system` call is a bit of a hack, but playing the audio file via pure Python (using a library such as Pygame) is overkill for this project.

From there, let's hardcode the configurations we'll use:

Keras and deep learning on the Raspberry Pi

Python

```
36 # define the paths to the Not Santa Keras deep learning model and
37 # audio file
38 MODEL_PATH = "santa_not_santa.model"
39 AUDIO_PATH = "jolly_laugh.wav"
40
41 # initialize the total number of frames that *consecutively* contain
42 # santa along with threshold required to trigger the santa alarm
43 TOTAL_CONSEC = 0
44 TOTAL_THRESH = 20
45
46 # initialize is the santa alarm has been triggered
47 SANTA = False
```

Lines 38 and 39 hardcode paths to our [pre-trained Keras model](#) and audio file. Be sure to use the ***“Downloads”*** section of this blog post to grab the files.

We also initialize parameters used for detection which include `TOTAL_CONSEC` and `TOTAL_THRESH`. These two values represent the number of frames containing Santa and the threshold at which we'll both play music and turn on the tree respectively (**Lines 43 and 44**).

The last initialization is `SANTA = False`, a boolean (**Line 47**). We'll use the `SANTA` variable later in the script as a status flag to aid in our logic.

Next, we'll load our pre-trained Keras model and initialize our Christmas tree:

Keras and deep learning on the Raspberry Pi

Python

```
49 # load the model
50 print("[INFO] loading model...")
51 model = load_model(MODEL_PATH)
52
53 # initialize the christmas tree
54 tree = LEDBoard(*range(2, 28), pwm=True)
```

Keras allows us to save models to disk for future use. [Last week](#), we saved our *Not Santa* model to disk and this week we're going to load it up on our Raspberry Pi. We load the model on **Line 51** with the Keras `load_model` function.

Our `tree` object is instantiated on **Line 54**. As shown, `tree` is an `LEDBoard` object from the `gpiozero` package.

Now we'll initialize our video stream:

Keras and deep learning on the Raspberry Pi

Python

```
56 # initialize the video stream and allow the camera sensor to warm up
57 print("[INFO] starting video stream...")
58 vs = VideoStream(src=0).start()
59 # vs = VideoStream(usePiCamera=True).start()
60 time.sleep(2.0)
```

To access the camera, we'll use `VideoStream` from my [imutils](#) package (you can find the [documentation to VideoStream here](#)) on either **Line 58** or **59**.

Important: If you'd like to use the `PiCamera` module (instead of a USB camera) for this project, simply comment **Line 58** and uncomment **Line 59**.

We `sleep` for a brief 2 seconds so our camera can warm up (**Line 60**) before we begin looping over the frames:

Keras and deep learning on the Raspberry Pi

Python

```
62 # loop over the frames from the video stream
63 while True:
64     # grab the frame from the threaded video stream and resize it
65     # to have a maximum width of 400 pixels
66     frame = vs.read()
67     frame = imutils.resize(frame, width=400)
```

On **Line 63** we start looping over video frames until the stop condition is met (shown later in the script).

First, we'll grab a `frame` by calling `vs.read` (**Line 66**).

Then we resize `frame` to `width=400`, *maintaining the aspect ratio* (**Line 67**).

We'll be preprocessing this `frame` prior to sending it through our neural network model. Later on we'll be displaying the frame to the screen along with a text label.

From there let's preprocess the image and pass it through our Keras + deep learning model for prediction:

Keras and deep learning on the Raspberry Pi

Python

```
69 # prepare the image to be classified by our deep learning network
70 image = cv2.resize(frame, (28, 28))
71 image = image.astype("float") / 255.0
72 image = img_to_array(image)
73 image = np.expand_dims(image, axis=0)
74
75 # classify the input image and initialize the label and
```



```

76         # probability of the prediction
77         (notSanta, santa) = model.predict(image)[0]
78         label = "Not Santa"
79         proba = notSanta

```

Lines 70-73 preprocess the `image` and prepare it for classification. To learn more about preprocessing for deep learning, be sure to check out the Starter Bundle of my latest book, [Deep Learning for Computer Vision with Python](#). From there, we query `model.predict` with our `image` as the argument. This sends the `image` through the neural network, returning a tuple containing class probabilities (**Line 77**).

We initialize the `label` to "Not Santa" (we'll revisit `label` later) and the probability, `proba`, to the value of `notSanta` on **Lines 78 and 79**.

Let's check to see if Santa is in the image:

Keras and deep learning on the Raspberry Pi

Python

```

81         # check to see if santa was detected using our convolutional
82         # neural network
83         if santa > notSanta:
84             # update the label and prediction probability
85             label = "Santa"
86             proba = santa
87
88             # increment the total number of consecutive frames that
89             # contain santa
90             TOTAL_CONSEC += 1

```

On **Line 83** we check if the probability of `santa` is greater than `notSanta`. If that is the case, we proceed to update the `label` and `proba` followed by incrementing `TOTAL_CONSEC` (**Lines 85-90**).

Provided enough consecutive "Santa" frames have passed, we need to trigger the Santa alarm:

Keras and deep learning on the Raspberry Pi

Python

```

92         # check to see if we should raise the santa alarm
93         if not SANTA and TOTAL_CONSEC >= TOTAL_THRESH:
94             # indicate that santa has been found
95             SANTA = True
96
97             # light up the christmas tree
98             treeThread = Thread(target=light_tree, args=(tree,))
99             treeThread.daemon = True
100             treeThread.start()
101
102             # play some christmas tunes
103             musicThread = Thread(target=play_christmas_music,
104                                  args=(AUDIO_PATH,))
105             musicThread.daemon = False
106             musicThread.start()

```

We have two actions to perform if `SANTA` is `False` and if the `TOTAL_CONSEC` hits the `TOTAL_THRESH` threshold:

1. Create and start a `treeThread` to twinkle the Christmas tree lights (**Lines 98-100**).
2. Create and start a `musicThread` to play music in the background (**Lines 103-106**).

These threads will run independently without stopping the forward execution of the script (i.e., a *non-blocking* operation).

You can also see that, on **Line 95**, we set our `SANTA` status flag to `True`, implying that we have found Santa in the input frame. In the next pass of the loop, we'll be looking at this value as we did on **Line 93**.

Otherwise (`SANTA` is `True` or the `TOTAL_THRESH` is not met), we reset `TOTAL_CONSEC` to zero and `SANTA` to `False`:

Keras and deep learning on the Raspberry Pi

Python

```
108         # otherwise, reset the total number of consecutive frames and the
109         # santa alarm
110         else:
111             TOTAL_CONSEC = 0
112             SANTA = False
```

Finally, we display the frame to our screen with the generated text label:

Keras and deep learning on the Raspberry Pi

Python

```
114         # build the label and draw it on the frame
115         label = "{}: {:.2f}%".format(label, proba * 100)
116         frame = cv2.putText(frame, label, (10, 25),
117                             cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
118
119         # show the output frame
120         cv2.imshow("Frame", frame)
121         key = cv2.waitKey(1) & 0xFF
122
123         # if the `q` key was pressed, break from the loop
124         if key == ord("q"):
125             break
126
127         # do a bit of cleanup
128         print("[INFO] cleaning up...")
129         cv2.destroyAllWindows()
130         vs.stop()
```

The probability value is appended to the `label` containing either "Santa" or "Not Santa" (**Line 115**).

Then using OpenCV's `cv2.putText`, we can write the label (in Christmas-themed green) on the top of the frame before we display the frame to the screen (**Lines 116-120**).

The exit condition of our infinite while loop is when the 'q' key is pressed on the keyboard (**Lines 121-125**).

If the loop's exit condition is met, we `break` and perform some cleanup on **Lines 129 and 130** before the script itself exits.

That's all there is to it. Take a look back at the 130 lines we reviewed together — this framework/template can easily be used for other deep learning projects on the Raspberry Pi as well.

Now let's catch that fat, bearded, jolly man on camera!

Deep learning + Keras + Raspberry Pi results

In [last week's blog post](#) we tested our *Not Santa* deep learning model using stock images gathered from the web.

But that's no fun — *and certainly not sufficient for this blog post.*

I've always wanted to dress up like good ole' St. Nicholas, so I [ordered a cheap Santa Claus costume](#) last week:



Figure 7: Me, Adrian Rosebrock, dressed up as Santa. I' ll be personally testing our *Not Santa* detector, built using deep learning, Keras, Python, and and OpenCV.

I' m a far cry from the real Santa, but the costume should do the trick.

I then pointed my camera attached to the Raspberry Pi at the Christmas tree in my apartment:



Figure 8: My very own Christmas tree will serve as the background for testing our *Not Santa* detector deep learning model which has been deployed to the Raspberry Pi.

If Santa comes by to put out some presents for the good boys and girls I want to make sure he feels welcome by twinkling the 3D Christmas tree lights and playing some Christmas tunes.

I then started the *Not Santa* deep learning + Keras detector using the following command:

Keras and deep learning on the Raspberry Pi

Shell

```
1 $ python not_santa_detector.py
```

To follow along, make sure you use ***“Downloads”*** section below to download the source code + pre-trained model + audio file used in this guide.

Once the *Not Santa* detector was up and running, I slipped into action:

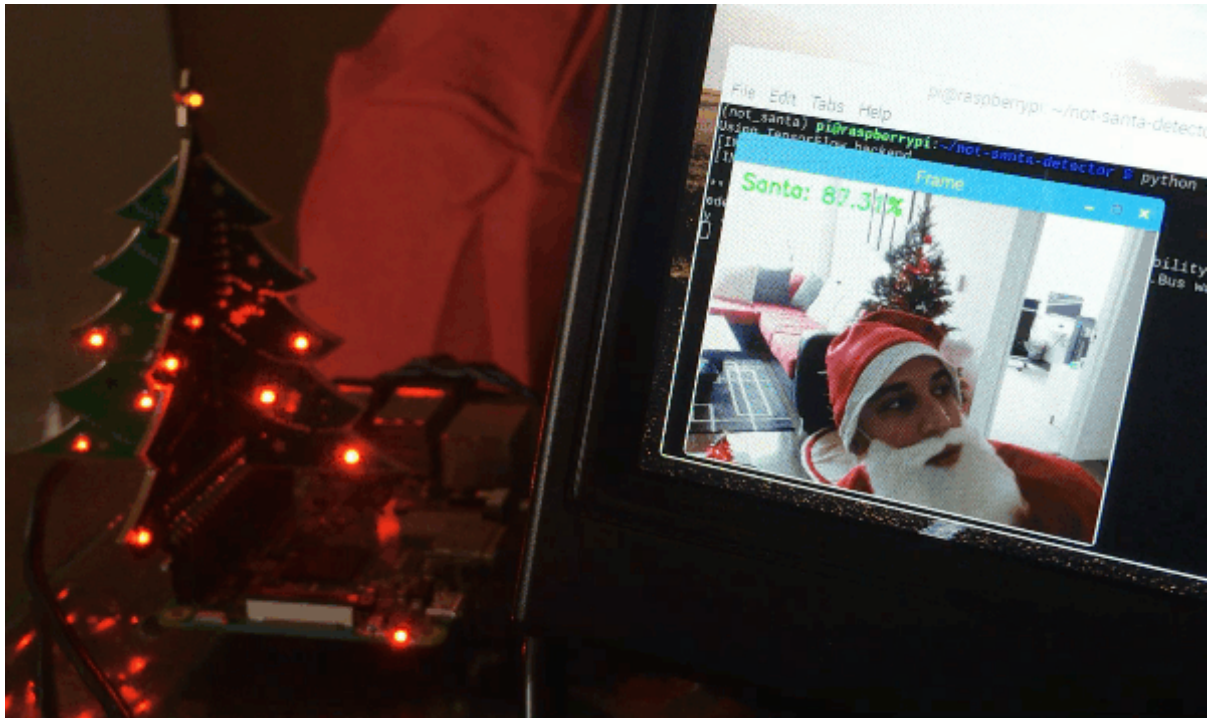


Figure 9: Successfully detecting Santa in a video stream using deep learning, Python, Keras, and a Raspberry Pi.

Whenever Santa is detected the 3D Christmas tree lights up and music starts playing! (which you cannot hear since this is a sample GIF animation).

To see the full *Not Santa* detector (with sound), take a look at the video below:

Whenever Santa enters the scene you'll see the 3D Christmas tree display turn on followed by a jolly laugh emitting from the Raspberry Pi speakers (audio credit to [SoundBible](#)).

Our deep learning model is surprisingly accurate and robust given the small network architecture.

I've been good this year, so I'm sure that Santa is stopping at my apartment.

I'm also more confident than I've ever been about seeing Santa bring some presents with my *Not Santa* detector.

Before Christmas, I'll probably hack this script (with a call to `cv2.imwrite`, or better yet, [save the video clip](#)) to make sure that I save some frames of Santa to disk, as proof. If it is someone else that puts presents under my tree, I'll certainly know.

Dear Santa: If you're reading this, just know that I've got my Pi on you!

Summary

In today's blog post you learned how to run a Keras deep learning model on the Raspberry Pi.

To accomplish this, we first [trained our Keras deep learning model](#) to detect if an image contains "Santa" or "Not Santa" on our laptop/desktop.

We then installed TensorFlow and Keras on our Raspberry Pi, enabling us to take our trained deep learning image classifier and then deploy it to our Raspberry Pi. While the Raspberry Pi isn't suitable for training deep neural networks, it can be used for deploying them — and provided the network architecture is simplistic enough, we can even run our models in real-time. To demonstrate this, we created a *Not Santa* detector on our Raspberry Pi that classifies each individual input frame from a video stream.

If Santa is detected, we access our GPIO pins to light up a 3D Christmas tree and play holiday tunes.

What now?

I hope you had fun learning how to build a *Not Santa* app using deep learning!

If you want to continue studying deep learning and:

- Master the fundamentals of machine learning and neural networks...
- Study deep learning in more detail...
- Train your own Convolutional Neural Networks from scratch...

...then you'll want to take a look at my new book, [Deep Learning for Computer Vision with Python](#).

Inside my book you'll find:

- Super practical **walkthroughs**.
- **Hands-on tutorials** (with lots of code).
- Detailed, thorough guides to help you **replicate state-of-the-art results** from popular deep learning publications.

<https://www.pyimagesearch.com/2017/12/18/keras-deep-learning-raspberry-pi/>