Sound Classification with TensorFlow

Having this solution along with an IoT platform allows you to build a smart solution over a very wide area.

By **DataArt** - November 10, 2017

Introduction

There are many different projects and services for human speech recognition like Pocketsphinx, Google's Speech API, and many others. Such applications and services recognize speech to text with pretty good quality, but none of them can determine different sounds captured by the microphone. What was on record: human speech, animal sounds, or music playing?

We were faced with this task and decided to investigate and build sample projects which will be able to classify different sounds using machine learning algorithms. This article describes which tools we have chosen, what challenges we have faced, how we have trained the model for TensorFlow, and how to run our open source project. Also we can supply the recognition results to the DeviceHive, IoT platform, to use them in cloud services for 3rd party application.

Choosing Tools and a Classification Model

At first we need to choose some software to work with neural networks. The first suitable solution that we found was Python Audio Analysis.

The main problem in machine learning is having a good training dataset. There are many datasets for speech recognition and music classification, but not a lot for random sound classification. After some research we found the urban sound dataset.

After some testing we were faced with the following problems:

- pyAudioAnalysis isn't flexible enough. It doesn't take a wide variety of parameters and some of them calculate on the fly. e.g. the number of training experiments based on number of samples and you can't alter this.
- The dataset only has 10 classes and all of them are "urban".

The next solution that we found was Google AudioSet. It is based on labeled YouTube video segments and can be downloaded in two formats:

1. CSV files describing, for each segment, the YouTube video ID, start time, end time, and one or more labels.
2. Extracted audio features that are stored as TensorFlow Record files.

These features are compatible with YouTube-8M models. Also this solution offers the TensorFlow VGGish model as feature extractor. It covered a big part of our requirements, and was therefore the best choice for us.

Training Model

The next task was to figure out how the YouTube-8M interface works. It's designed to work with videos, but fortunately can work with audio as well. This library is pretty flexible, but it has a hardcoded number of sample classes. So we modified this a little bit to pass the number of classes as parameter.

YouTube-8M can work with data of two types: aggregated features and frame features. Google AudioSet can provide data as features as we noted before. Through a little more research we discovered that the features are in frame format. We then needed to choose the model to be trained.

Resources, Time, and Accuracy

GPUs are more suitable choice for machine learning than CPUs. You can find more info about this here. So we will skip this point and go directly to our setup. For our experiments we have PC with one NVIDIA GTX 970 4GB.

In our case the training time didn't really matter. We should mention that 1-2 hours of training was enough to make an initial decision about the chosen model and its accuracy.

Of course we want to get as good accuracy as possible. But to train a more complex model (potentially better accuracy) you need more RAM (video RAM in case of GPU) to fit it in.

Choosing the Model

A full list of YouTube-8M models with descriptions is available here. Because our training data was in frame format, frame-level models had to be be used. Google AudioSet provides us with a data set split into three parts: balanced train, unbalanced train, and evaluation. You can get more info about them here.

A modified version of YouTube-8M was used for training and evaluation. It's available here.

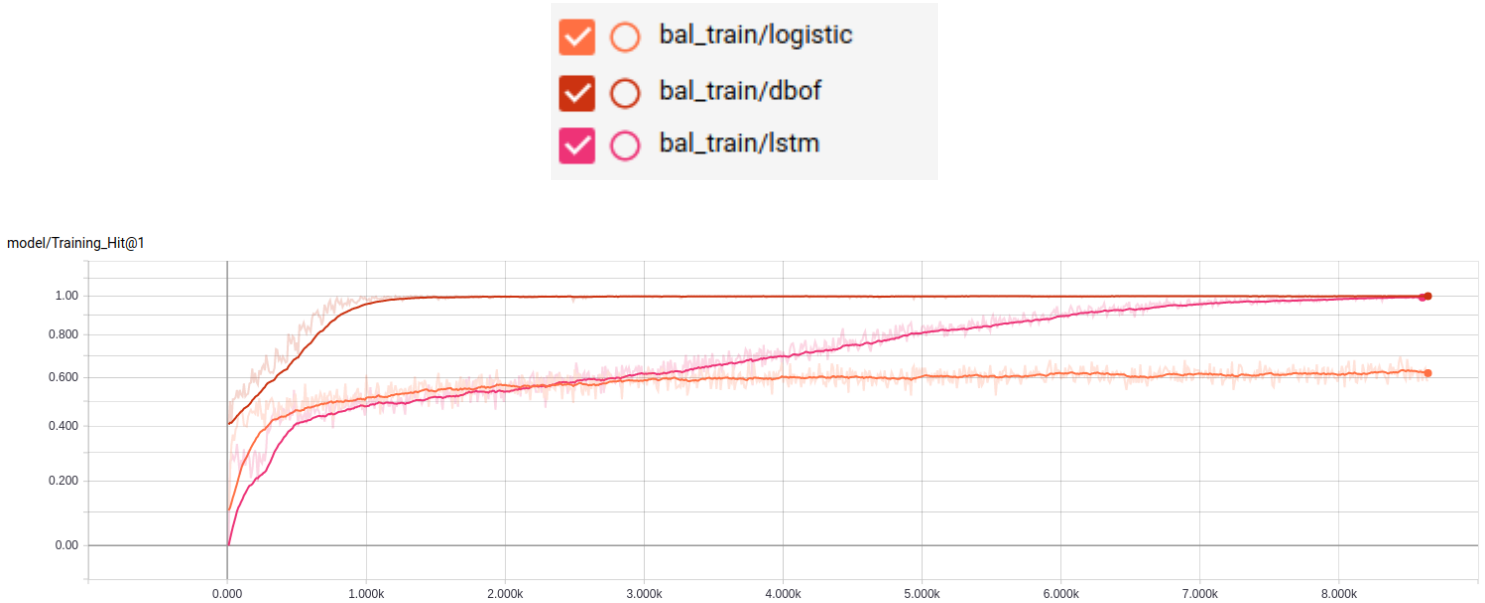
Balanced Train

The training command looks like:

python train.py –
train_data_pattern=/path_to_data/audioset_v1_embeddings/bal_train/.tfrecord –*
num_epochs=100 –learning_rate_decay_examples=400000 –
feature_names=audio_embedding –feature_sizes=128 –frame_features –batch_size=512 –
num_classes=527 –train_dir=/path_to_logs –model=ModelName

For LstmModel we changed the base learning rate to 0.001 as the documentation suggested. Also we changed the default value of lstm_cells to 256 because we didn't have enough RAM for more.

Let's see the training results:

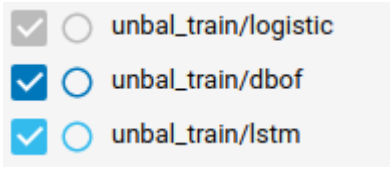


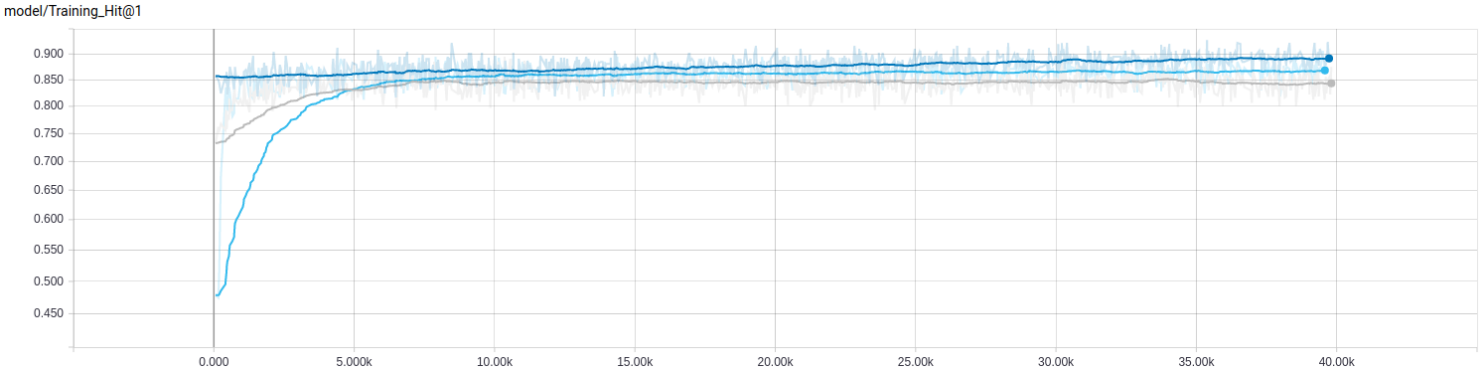
Model name	Training time	Training last step hit	Evaluation average hit
Logistic	14m 3s	0.5859	0.5560
Dbof	31m 46s	1.000	0.5220
Lstm	1h 45m 53s	0.9883	0.4581

As we can see, we got good results during the training step – this doesn't mean we would get good results on the full evaluation.

Unbalanced Train

Let's try the unbalanced train dataset. It has a lot more samples, so we will change the number of training epochs to 10 (should change to 5 at least, because it took significant time to train).





Model name	Training time	Training last step hit	Evaluation average hit
Logistic	2h 4m 14s	0.8750	0.5125
Dbof	4h 39m 29s	0.8848	0.5605
Lstm	9h 42m 52s	0.8691	0.5396

Train Logs

If you want to examine our training logs you can download and extract train_logs.tar.gz. Then run *tensorboard –logdir /path_to_train_logs/* and go to http://127.0.0.1:6006

More About Training

YouTube-8M takes many parameters and a lot of them affect the training process.

For example: You can tune the learning rate and number of epochs that will change the training process a lot. There are also three different functions for loss calculation and many other useful variables that you can tune and change to improve the results.

Using Trained Model with Audio Capture Devices

Now we have some trained models, it's time to add some code to interact with them.

Capture Mic

We need to somehow capture audio data from a microphone. We will use PyAudio. It provides a simple interface and can work on most platforms.

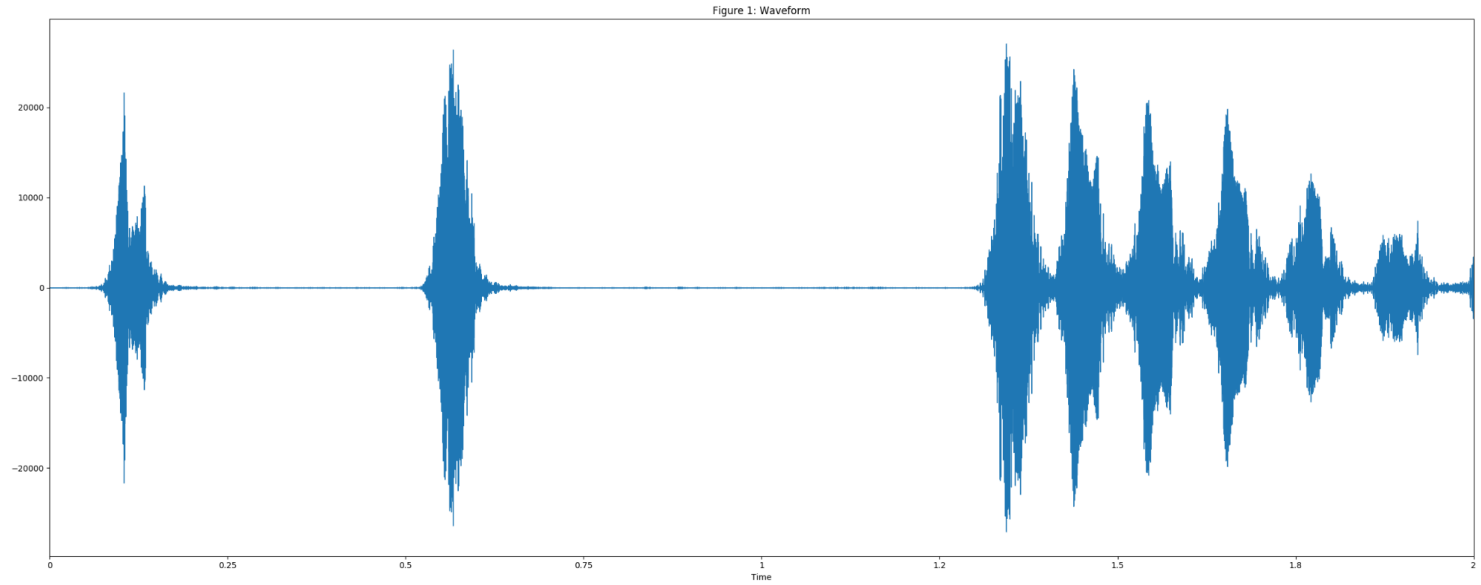
Sound Preparation

As we mentioned before, we will use the TensorFlow VGGish model as the feature extractor.

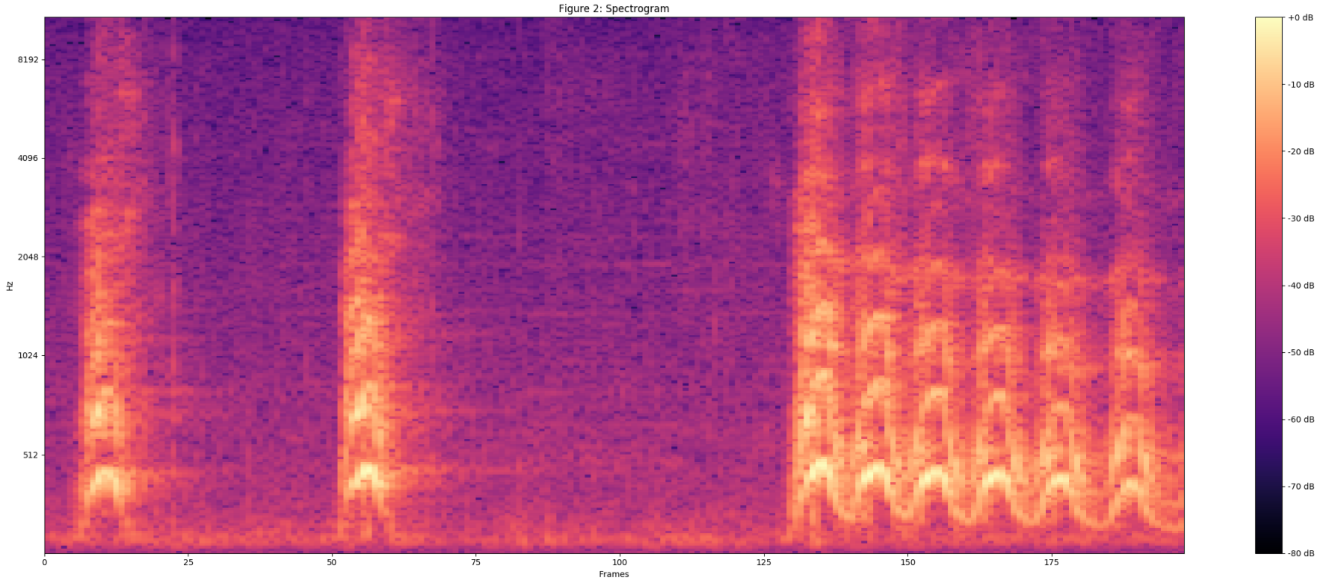
Here is a short explanation of the transformation process:

“Dog bark” example from the UrbanSound dataset was used for visualization.

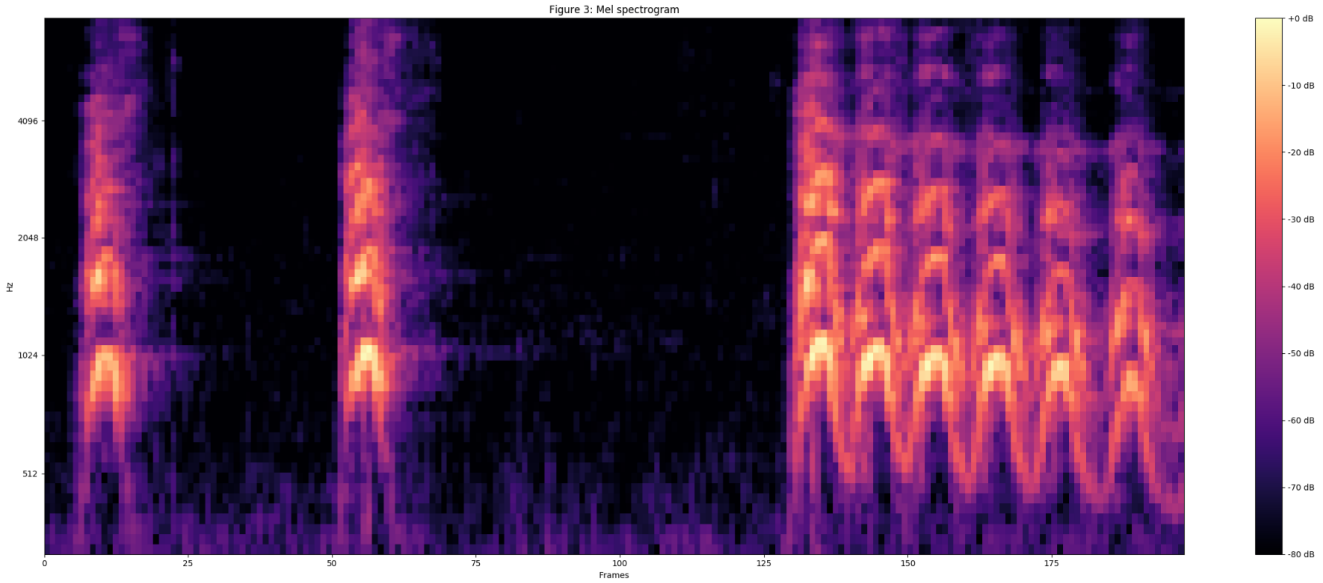
Resample audio to 16 kHz mono.



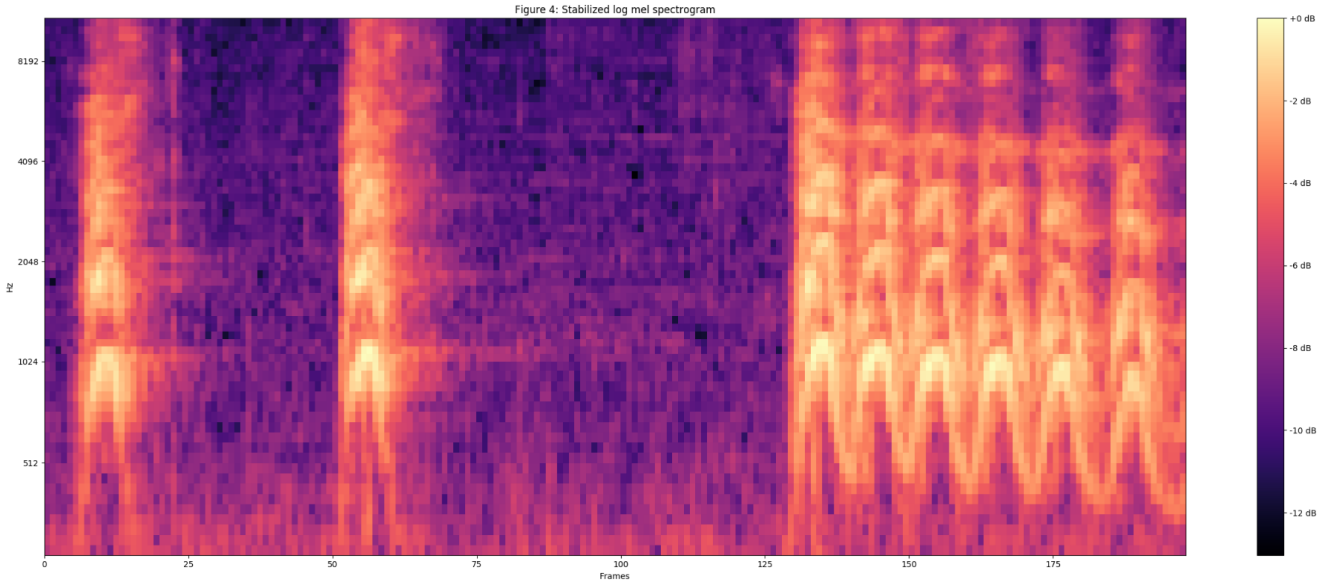
Compute spectrogram using magnitudes of the Short-Time Fourier Transform with a window size of 25 ms, a window hop of 10 ms, and a periodic Hann window.



Compute mel spectrogram by mapping the spectrogram to 64 mel bins.



Compute stabilized log mel spectrogram by applying log(mel-spectrum + 0.01) where an offset is used to avoid taking a logarithm of zero.



These features are then framed into non-overlapping examples of 0.96 seconds, where each example covers 64 mel bands and 96 frames of 10 ms each.

These examples are then fed into the VGGish model to extract embeddings.

Classifying

And finally we need an interface to feed the data to the neural network and get the results.

We will use the YouTube-8M interface as an example, but will modify it to remove the serialization/deserialization step.

Here you can see the result of our work. Let's take a closer look.

Installation

PyAudio uses libportaudio2 and portaudio19-dev so you need to install them to make it work.

Some python libraries are required. You can install them using pip.

pip install -r requirements.txt

Also you need to download and extract to the project root the archive with the saved models. You can find it here.

Running

Our project provides three interfaces to use.

1. Process Prerecorded Audio File

Simply run *python parse_file.py path_to_your_file.wav* and you will see in the terminal something like *Speech: 0.75, Music: 0.12, Inside, large room or hall: 0.03*

The result depends on the input file. These values are the predictions that the neural network has made. A higher value means a higher chance of the input file belonging to that class.

IoT For All Newsletter

Sign up for our weekly newsletter and exclusive content!

Your email address

Sign up

2. Capture and Process Data from Mic

python capture.py starts the process that will capture data from your mic infinitely. It will feed data to the classification interface every 5-7 seconds (by default). You will see the results in the

previous example.

You can run it with *–save_path=/path_to_samples_dir/* in this case all captured data will be stored in the provided directory in *wav* files. This function is useful in case you want to try different models with the same example(s). Use the *–help* parameter to get more info.

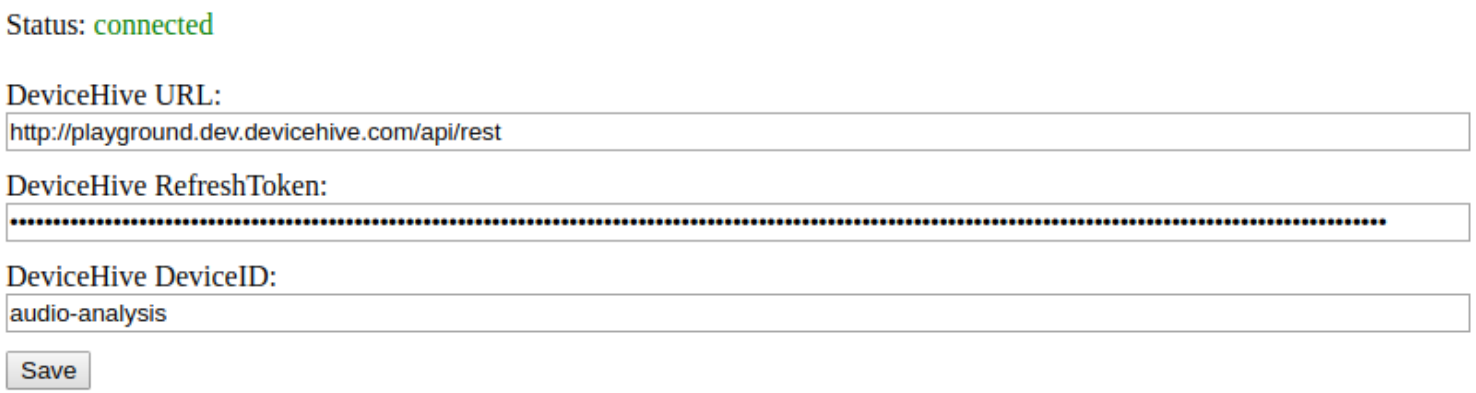
3. Web Interface

python daemon.py implements a simple web interface that is available on http://127.0.0.1:8000 by default. We use the same code as for the previous example. You can see the last ten predictions on the events (http://127.0.0.1:8000/events) page.

2017-11-02 17:29:38	Music: 0.26
2017-11-02 17:29:43	Music: 0.56, Wind chime: 0.20, Speech: 0.13, Chime: 0.12
2017-11-02 17:29:48	Music: 0.51, Wind chime: 0.10
2017-11-02 17:29:53	Music: 0.68, Spray: 0.15, Grunge: 0.12, Speech: 0.11
2017-11-02 17:29:58	Music: 0.56, Speech: 0.13
2017-11-02 17:30:03	Music: 0.56, Speech: 0.15, Snort: 0.13, Crunch: 0.10
2017-11-02 17:30:08	Speech: 0.28, Music: 0.23, Fly, housefly: 0.23, Bee, wasp, etc.: 0.14
2017-11-02 17:30:13	Speech: 0.33, Music: 0.24, Fly, housefly: 0.18, Bee, wasp, etc.: 0.11
2017-11-02 17:30:18	Fly, housefly: 0.28, Speech: 0.26, Music: 0.19, Bee, wasp, etc.: 0.17
2017-11-02 17:30:23	Speech: 0.21, Music: 0.15

IoT Service Integration

Last but not least is integration with the IoT infrastructure. If you run the web interface that we mentioned in the previous section, then you can find the DeviceHive client status and configuration on the index page. As long as the client is connected, predictions will be sent to the specified device as notifications.



Conclusion

TensorFlow is a very flexible tool, as you can see, and can be helpful in many machine learning applications like image and sound recognition. Having such a solution together with an IoT platform allows you to build a smart solution over a very wide area.

Smart cities could use this for security purpose, continuously listening for broken glass, gunfire, and other sounds related with crimes. Even in rainforests, such a solution could be used to track wild animals or birds by analyzing their voices.

The IoT platform can then deliver all such notifications. This solution can be installed on local devices (though it still can be deployed somewhere as a cloud service) to minimize traffic and cloud expenses and be customized to deliver only notifications instead of including the raw audio. Do not forget that this is an open source project, so please feel free to use it.

Written by Igor Panteleyev, Senior Developer at DataArt.

DataArt

https://www.dataart.com/iot

DataArt is a global technology consultancy that designs, develops, and supports unique software solutions. With its deep industry knowledge, DataArt's IoT team created DeviceHive, an open source IoT data platform with a wide range of device integration options. DataArt also provides commercial support services for DeviceHive.

BLOCKCHAIN 99

Why Block Chain is the Next Big Thing In Cybersecurity

Guest Writer - January 16, 2018

CONFERENCE TAKEAWAYS 470

IoT On The Rise — Highlights from CES 2018

Eric Conn - January 15, 2018

TECH 335

IoT for Smart Buildings Isn't What You Think It Is

Matt Ernst - January 12, 2018

Most Recent

The (Smart) Home of Your Dreams

Smart Home 249 January 11, 2018

It's 2028. You have a family and live in a 4-bedroom, 3-bath, 2-car garage detached home. If you live in the US or Canada, your home's value is the equal in...

8 Things I Learned about IoT in 2017

Advice 476 January 10, 2018

2017 marked continued stable growth in both interest and deployment of IoT all over the world. In fact, according to IoT Analytics, there are more connected IoT devices than smartphones and...

HOT NEWS

IoT Dev 3062

IoT Hardware — Introduction and Explanation

Platforms 489

There is No Universal IoT Platform

Consun 11110

Consumer IoT vs. Industrial IoT — What are the Differences?

Industric 1130

Four Investments Driving Digital Transformation in Manufacturing

Choosing the Right Market for AR & VR Products

Advice 158 January 9, 2018

Whether you're a solo developer or you have a host of analysts at your disposal, finding product-to-market fit has always been a balance of art and science for both enterprises and...

Are Prescriptive Applications IIoT Nirvana?

Industrial 158 January 8, 2018

Earlier, I wrote about the value of prescriptive analytics. It's the next wave of analytics. But it still falls short of ensuring successful IIoT applications.Success requires a direct link between analytics...

How Will IoT Impact the Insurance Industry?

Conference Takeaways 290 January 5, 2018

The Internet of Things will disrupt and impact many industries, from automotive to healthcare to energy and beyond. One industry that is already being redefined by IoT, but we don't usually...

Load more