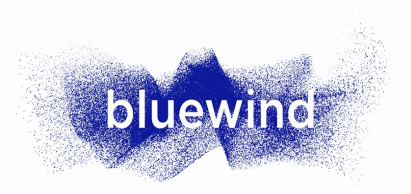
Project Report

For internal use only

Vivaldi Project Overview

Summary



Acknowledgments

Bluewind would like to express special thanks to the partners involved in the development of the Vivladi project.

Danilo Pietro Pau, Senior Principal Engineer and Senior Member of Technical Staff Advanced System Technology at **STMicroeletronics**, supported the Vivaldi team from the beginning. Danilo provided steady support from the hardware selection to the neural network model optimization and integration phases. His highly valuable help proved to be fundamental in the overall application development.

1) Introduction

This document talks about the **Vivaldi Car Detection Application Demo developed at Bluewind** in 2018.

The aim of the document is to highlight the main results and the major issues encountered during technical development, as well as to depict the current state and the future perspectives of the application.

This document should also be regarded as a short tutorial about the development of **machine learning based application for STM32** targets.

In particular, the next chapters deal with the major topics or phases of the Vivaldi application development:

- supervised machine learning application for low power microcontrollers
- a world of sounds: audio feature extraction and preprocessing
- useful software tools for AI coding in C

Each chapter is standalone and there is no need to read them all; find below what each chapter talks about and just skip to the ones you are interested in:

2) Vivaldi application demo

This section gives an overview of the application demo developed at Bluewind. The strengths of the application are described at the beginning of the section. The second paragraph talks about the fields of application of the Vivaldi device. Following will be a brief description of the working principle and of the hardware features.

The Vivaldi application demo's purpose is to highlight the opportunities made available by the adoption of unconventional programming techniques in the world of embedded systems, smart sensors and IoT. The aim of the demo is to show some of the benefits obtained from integrating machine learning techniques in the embedded software, such as:

- the **required memory capacity is decreased**. This happens whenever meaningful data are made available right after raw data collection (instant processing)
- high information density data are exchanged between the device and the remote server only: data are exchanged occasionally → **great impacts on energy savings** arise (less and less frequently)

"*The physics of moving data around just seems to require a lot of energy. There seems to be a rule that the energy an operation takes is proportional to how far you have to send the bits. CPUs and sensors send bits a few millimeters, and is cheap, radio sends them meters or more and is expensive.*"

Pete Warden's blog, 11 June 2018

- **safety is increased** since the system operates without the need of being steadily connected to a host (twice safer: operational safety and cyber security)

Before discovering how the Vivaldi application does integrate these features, an overview of the task it carries out and of its working principle is outlined.

2.1) Current and future scenarios

The aim of the Vivaldi device is to collect statistics about urban traffic by counting the number of cars passing by. This product is intended to be installed by the side of the street and is suitable for whoever needs to collect statistics on traffic flows and road conditions.

The key features of the device are listed below:

- cheapness
- small dimensions

- low power consumption
- safety
- promptness

The Vivaldi device is designed to help people who manage urban spaces and design transportation systems, providing them with the data they need to take decisions and act. It could help urban architects and authorities who will transform the places we live in into more livable and sustainable cities.

SDG n.11

The demo version and its future releases fall within the **ITS** (intelligent transportation systems) field of application, which includes each infrastructure or technological device involved in traffic and mobility management. An ITS application is, for instance, traffic light networks optimization using real-time data to maximize vehicles throughput.

“Sustainable development cannot be achieved without significantly transforming the way we build and manage our urban spaces."

Transforming our world: the 2030 agenda for sustainable development, UN

Another field of application of the Vivaldi device is **autonomous driving**. Such a sensor is capable of detecting cars flowing nearby. These sort of data could feed a decision-taking algorithm of an autonomous vehicle: a use case example is the one of a car approaching a crossroad in the need to know whether another vehicle is doing the same, before having the capability of seeing it by mean of its on board camera-like devices. Furthermore, this data reinforce decision taking in case of lack of visibility.

Self-driving vehicle sensors

2.2) Working principle

Should you analyze a street's traffic flow or detect the proximity of a flowing vehicle, which devices would you suggest? How would you then manage the overall information flow?

Today’s solutions are based on video cameras and displacing a net of devices through several roads is highly expensive and energy consuming.

Vivaldi device relies on **audio stream analysis** and **sound event classification**. This solution gives the possibility to use small, cheap and low power consuming sensors and MCUs.

The overall work-flow of the runtime application is:

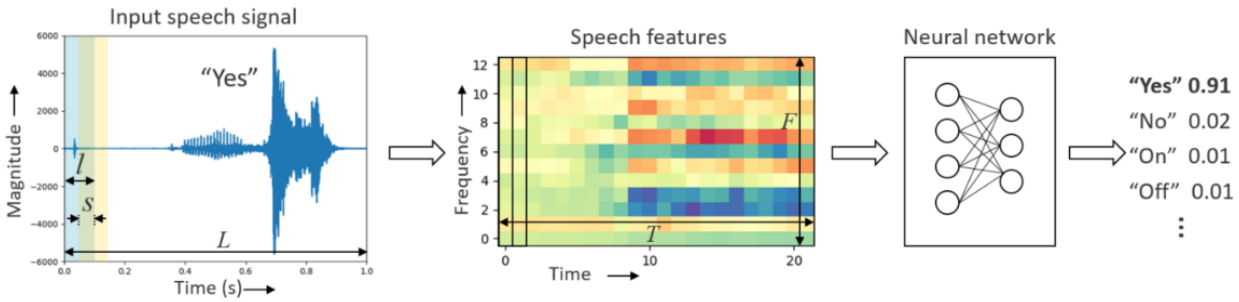
- audio **acquisition** over a 2s time window
- audio **preprocessing** for MFCC extraction
- sound event **classification** based on pre-trained CRNN algorithm (Convolutional Recurrent Neural Network)

Preprocessing and classification execution together take less than 0.5s. This does not affect the device performances due to the fact that the flowing car is sensed within more than 50 yards, therefore the chances of not hearing it are null.

It is known that using raw audio samples to feed a neural network is not feasible in terms of model’s accuracy and dimensions. Therefore audio is preprocessed to extract an MFCCs (Mel-frequency Cepstral Coefficients) matrix, which stands as input to the classification algorithm.

Lastly, a CRNN model was selected due to its well-known high accuracy reached in the tasks of audio streams analysis, such as key word spotting or sound event classification. The application demo embeds a pre-trained CRNN model where net training was performed off line.

You will find more about audio processing and model architecture in the next chapters.



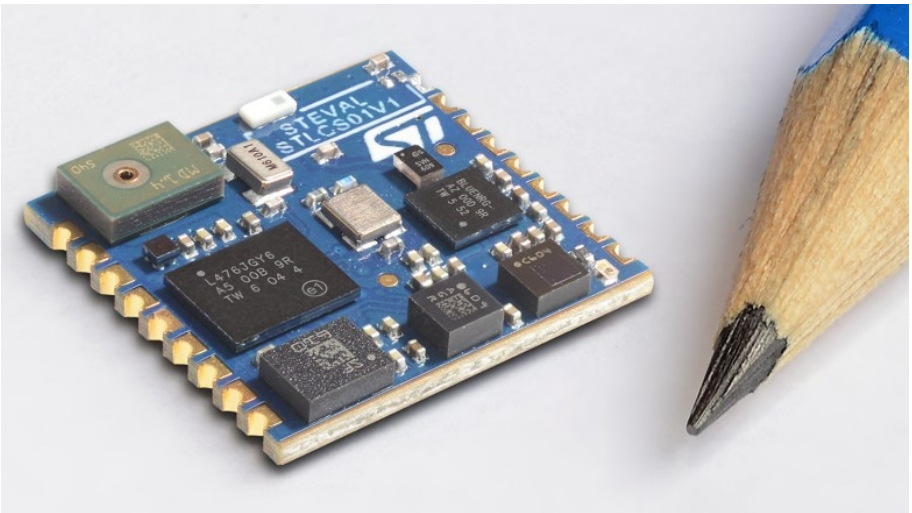
Audio classification: the keyword spotting pipeline

2.2) Hardware

The hardware on which the application is based is the Sensortile development kit from STMicroelectronics equipped with the following components:

- a MEMS microphone
- STM32L476JG MCU
- audio DAC
- LEDs
- others (not used)

STM32L476JG is an ultra low power MCU based on the ARM Cortex-M4 32-bit architecture. It embeds high-speed memories with 1Mbyte of flash and 128 Kbyte of SRAM. These resources proved to be enough for a well pruned network to run. Furthermnore, MEMS sensors and a low power MCU ensure very low power consumptions, thus extending the life of the device.



The SensorTile board

3) Optimized supervised learning model

This chapter deals with the choice of the machine learning algorithm architecture. The model optimization for reaching the target capabilities (memory and timings) will be presented as well. The last paragraph talks about the dataset creation and the application's scalability.

In this chapter you'll go through the development phases of the application architecture and you will get to know why a binary classification algorithm was not enough to reach the desired output. Moreover the following question will be answered: which is the physical principle which triggers the Vivaldi device? Or in other words, what did we teach it?

As per each supervised learning application, data are the starting point: the more you have, the higher accuracy the model will reach. The great care taken during the dataset creation is described in the last section of this chapter.

3.2) CRNN architecture

Modeling and training of the classification algorithm was performed with **Keras** v2.1.5 framework running over Tensorflow.

Convolution recurrent neural network is a hybrid of CNN (Convolution neural net) and RNN (recurrent neural net), which takes advantages of both. It exploits the local temporal/spatial correlation using convolution layers and global temporal dependencies in the speech features using recurrent layers.

The model is fed with the MFCC matrices of size 20x44 supplied every 2s by the preprocessing stage. The instances' features are each of the 880 cepstral coefficients stored in the matrix. The calculation algorithm and the meaning of MFCCs matrix will be deepened in the next chapter.

The adopted CRNN model starts with two convolution layers each followed by a max pooling layer for dimensionality reduction and batch normalization for vanishing/exploding gradients avoidance. Dropout was adopted to enhance generalization.
It is not the purpose of this report to talk about machine learning optimization and generalization techniques: the referenced books dealing with these topics represent great guides and studying material.

After the convolution layers a recurrent layer made of 10 LSTM cells (long short-term memory) is used to encode the signal. Again batch norm and dropout were used for model convergence and for generalization enhancing.

Two fully connected layers map the information which at last reaches the softmax layer.

The model was trained off line using the categorical cross-entropy loss and Adagrad optimizer with a learning rate of 0.01 and batch size of 30 over 20 epochs. The so trained model was evaluated based on the classification

accuracy on the test set.

To fit the target capabilities several pruning and optimization iterations were performed. Each iteration required a model adjustment in terms of convolution filter numbers and kernel size, as well as in pooling layer dimensions. The final model features, fitting the target, are:

- required ROM: 480 kBytes
- required RAM: 98 kBytes
- number of tunable parameters: ~4600

3.3) Dataset creation and scalability

The first aim of the Vivaldi team was to develop an application able to classify instances into two sounds categories: “car” and “something else”. The hypothesis behind this choice was that an approaching car would make the softmax probability of the class “car” to increase towards 1 with succeeding audio acquisitions, while a receding car would make the “car” probability to decrease towards 0.

Therefore, simple post processing rules would have given the possibility to find the “car” probability peak and trigger the +1 counting for each flowing car, avoiding false positives generation (namely counting the same car twice or more).

The dataset was created by collecting audio samples with the Sensortile (Audioloop application provided by STMicroelectronics), reproducing the real working conditions of the device. Each instance was stored in the category it belonged to and in two weeks the dataset was filled with 3000 samples for the “something else” category and 4000 for the “car” category.

Training reached 98% accuracy on the validation set and the first field tests were run in order to find the required post processing rules. What turned to be an issue was to define effective rules: setting a probability threshold, monitoring the probability’s moving average or calculating the probability’s time integral all proved not to be accurate enough.

The reason why such an application continuously generated false positives was that the “car” probability instantly stepped from 0 to 1 as soon as a flowing car was heard: the hypothesis of gradual transition was wrong.

The solution came by switching from a binary to a **multi-class** algorithm losing few points in the validation accuracy. The dataset was rearranged into three categories: “approaching car”, “receding car” and “something else”. This required each sample to be listened again and split into two halves after having detected the critical point. The loss in theoretical accuracy was a minor pain and a real gain with regards to application’s efficiency.

The multi classification version passed the field tests: the Vivaldi device’s LED is toggled when the softmax higher probability switches from the class

"approaching car" to the "receding car" one.

In this sense the Vivaldi sensor was taught to respond to the **Doppler effect** of a flowing car.

Later field tests reached accuracies over 99% in the car detection purpose.

The training dataset is design to give the chance of adding new categories. Different vehicles could be counted for instance: "approaching truck", "receding truck", "approaching motorcycle" and "receding motorcycle". The number of classes of the model impact on the softmax layer dimension only, thus the overall model dimensions is not affected.

4) The world of sounds

You are watching television on your sofa with your dog, waiting for some family members to join you for lunch. The noise of a car breaks the silence, you suppose it's your aunt's car noise as she usually arrives earlier than the others. That's a supposition based on experience but your dog can extract greater information from that input. You watch "Tokyo" as it raises its ears listening carefully to the noise for a bunch of seconds and after that it starts barking at you. You stand up and watch outside the window until you realize it's your son's car. You look back and say: "you're right Tokyo, I was wrong".

We use to rely mainly on our sight, the most powerful sense we are given. But **the world of sounds is full of information** as well and audio sensors are much smaller, much cheaper and much more energy saving. Today, sound detection is an important capability for use cases such as the exposure of illegal logging, diagnosis of medical or neurological conditions such as Parkinson's disease or predictive maintenance of public transportation systems.

But basing a classifier on raw audio samples would be highly inefficient: this chapter deals with the audio stream processing and features extraction, a crucial point in the overall Vivaldi application workflow.

4.1) MFCCs feature extraction

The MFCCs matrix is the output of the feature extraction process, its size is 20X44, totalling 880 floating point input features.

Mel-frequency cepstral coefficients (**MFCCs**) are derived from a type of cepstral representation of the audio stream (a nonlinear "spectrum-of-a-spectrum"). The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal cepstrum, these techniques. This frequency warping can allow for better representation of sound.

In the Vivaldi application MFCCs are derived as follows:

1. Acquire a 2s audio signal at 11025 Hz frequency.
2. Take the Fourier transform of a 180 ms windowed excerpt of a signal (with a stride of 40 ms).
3. Map the powers of the spectrum obtained above onto the mel scale, using 20 triangular overlapping windows.
4. Take the logs of the powers at each of the mel frequencies.
5. Take the discrete cosine tranform (DCT) of the list of mel log powers, as if it were a signal.
6. The MFCCs are the amplitudes of the resulting spectrum.

Python packages **LibROSA** and **Scipy** provide useful tools for music and audio

analysis. The default **librosa.feature.mfcc()** function was used with its default parameters for off-line preprocessing and it was then translated in C to be integrated in the embedded application.

5) C coding tools

Machine Learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed (see https://www.expertsystem.com/machine-learning-definition/). But how should a deep learning pre-trained neural network model be part of an embedded software application targeting STM32 execution? How to improve programmer productivity by avoiding to develop time consuming hand crafted code to run the neural network inference engine on STM32 ?

The solution comes from **STM32CUBE.AI**: a STMicroelectonics AI software extension for the well-known STM32CubeMX configuration and code generation tool (see https://www.youtube.com/watch?v=ORojJPPhprU/).

Based on the chosen STM32 target the tool takes the pre-trained model (for instance a .h5 format model generated with Keras) as input and automatically generates a memory optimized and computationally efficient NN library, checking for memory size consistency, being fully validated and providing static and run-time information. ST public APIs are exported as well to expose the NN library resources to the high level application.

At the time of development the Keras supported versions were 2.1.5 and earlier (this is subject to change).

Embedded software was developed with Atollic TrueSTUDIO IDE for STM32 v.9.0.1.

6) Conclusions

The Vivaldi sensor's task is to detect flowing cars. The application is based on audio analysis with sound event classification techniques. The device is designed to be cheap, small and energy saving. The fields of application of the Vivaldi sensor are:

- traffic monitoring and **ITS** (intelligent transportation systems)
- **autonomous driving vehicles**

The Vivaldi car detection device stands as **a real case example of embedded machine learning application**. The power of deep learning makes it possible to perform a task which would hardly be achieved by conventional programming techniques. Furthermore, deploying **on the edge ML** brings several other advantages, such as:

- application' s **safety** and **stability**
- **low power consumptions** (no communication with server)
- **memory optimization** (meaningful data are available right after raw data collection)

The target hardware is the Sensortile development kit from STMicroelectronics, which holds small and cheap sensors and an ultra low power MCU based on ARM Cortex-M4 architecture, featuring 1Mb of Flash and 128Mb of SRAM. An optimized model running on board and efficient coding make it possible to preserve the information and the classification accuracy during execution.

The STM32CUBE.AI package was used for neural network software integration and deployment on STM32 within sensor tile. The availability of this tool brings to be an innovative approach in the designing of smart IoT infrastructures which relies upon **distributed AI** instead of fully centralized systems.

7) References

- “Hands-On Machine Learning with Scikit-Learn and TensorFlow”Aurelién Géron [2017]
- “The 2030 agenda for sustainable development” UN General Assembly [2015]
- “Human and Machine Hearing: Extracting Meaning from Sound” Richard F. Lyon [2017]
- “Hello Edge: Keyword Spotting on Microcontrollers” Yundong Zhang, Naveen Suda, Liangzhen Lai, Vikas Chandra [2018]
- “Interpreting and Explaining Deep Neural Networks for Classification of Audio Signals” Sören Becker, Marcel Ackermann, Sebastian Lapuschkin, Klaus-Robert Müller, Wojciech Samekb [2018]
- “Deep Learning for IoT Big Data and Streaming Analytics: A Survey” Mehdi Mohammadi, Ala Al-Fuqaha, Sameh Sorour [2018]
- “Notes from the AI frontier: applying AI for social good” McKinsey Global Institute [2018]