

# Automatic Speech Detection on a Smart Beehive's Raspberry Pi

Pascal Janetzky<sup>1</sup>, Philip Lissmann<sup>1</sup>, Andreas Hotho<sup>1</sup> and Anna Krause<sup>1</sup>

<sup>1</sup>University of Würzburg, Department of Computer Science, CAIDAS, Chair for Data Science, 97074 Würzburg, Germany

## Abstract

The we4bee project has deployed 100 smart hives all over Germany. These hives are equipped with microphones, among other sensors. Beekeepers and bee researchers have observed the importance of sounds when monitoring bee hives, but audio can only be recorded in accordance with privacy laws. To prevent saving recordings of human voices, our aim is to deploy a pre-trained deep learning model on the Raspberry Pi 3B computer controlling the smart hive. This model has to classify recorded data in real-time.

In this technical report, we document the process of setting up the software on the Raspberry Pi, the adaptations required for existing code to run in the new environment, and the necessity of modifying the trained models for deployment on the mini-computer. We find that in both standard operation conditions and under various artificial levels of high CPU and I/O load, the model's inference runs in real-time.

## Keywords

TensorFlow, machine learning, audio classification, speech detection, mobile computing

## 1. Introduction

The we4bee project<sup>1</sup> runs around 100 smart beehives all over Germany. Since 2019, these hives collect data in and around the hives with 16 sensors and two cameras. The sensor system is powered by a Raspberry Pi 3B mini computer, which handles data collection and transfer to the central database at the University of Würzburg. In our earliest work, we relied on inside temperatures to detect events in the hive via anomaly detection [1], but audio recorded in beehives is also a valuable data source for precision beekeeping [2, 3, 4]. In recent work, we have thus leveraged audio data for event detection [5]. However, since written consent is mandatory to record audio data, audio is currently recorded in a single beehive only.

In order to record audio data in more hives, Janetzky et al. [6] trained deep learning models that identify human speech in audio recordings obtained from the we4bee hive. In this technical report, we directly deploy the best-performing model from our earlier research on the Raspberry Pi in the smart hives. For successful deployment, we require the model to be real-time capable: a 60 s recording has to be classified in less than a minute so that data without speech is buffered or uploaded before successive samples are recorded.


---

LWDA'23: *Lernen, Wissen, Daten, Analysen*. October 09–11, 2023, Marburg, Germany

✉ janetzky@informatik.uni-wuerzburg.de (P. Janetzky); philip.lissmann@stud-mail.uni-wuerzburg.de (P. Lissmann); hotho@informatik.uni-wuerzburg.de (A. Hotho); anna.krause@informatik.uni-wuerzburg.de (A. Krause)



© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

<sup>1</sup>we4bee.org

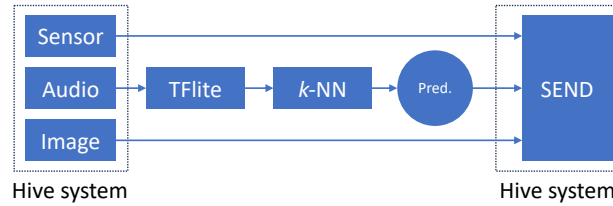


Figure 1: An overview of the existing data collection system (dotted boxes) and our speech detection pipeline. Sensory and image data is automatically collected and uploaded as is. For the audio data, we obtain a recording’s embeddings through TensorFlow Lite (TFlite) and predict their class. Data is only uploaded if no speech is detected.

## 2. Detecting Human Speech

In our previous work [6], we evaluated three Siamese neural networks, *Saeed* [7], *ESC* [8, 9] and *Bulbul* [10], followed by a  $k$ -Nearest Neighbor ( $k$ -NN) classifier [11, 12], on the detection of speech in audio recordings obtained from beehives. Of these networks, *Bulbul* showed the best performance. This Siamese network consists of four times a block of convolution, leaky ReLU [13], and max pooling layers. Afterwards, the output is flattened and followed by two blocks of dropout [14], dense, and leaky ReLU layers.

The Siamese network has been trained on a total of 200 labeled samples of 60 s, of which we created random data pairs. The network was then trained to minimize the Euclidean distance between audio pairs from the same class (e.g., speech-speech) and to maximize the distance between pairs of different classes. From the trained network, the embeddings of the training data were then extracted from an intermediate layer and used to train a  $k$ -NN classifier to predict a sample’s class.

## 3. On-device Classification

To be able to record audio in more than one beehive, we want to identify and discard audio recordings with human speech directly on the beehive’s Raspberry Pi. To this end, we selected an exemplary smart beehive and completed the following tasks: installing the TensorFlow Lite library on the Raspberry Pi; migrating our Python environment and scripts; migrating the model; and enabling real-time classification of incoming audio data. The proposed approach to real-time speech detection is visualized in fig. 1. The existing system, visualized on the left, records sensor and image data and uploads them directly. For the detection of speech, we use the pipeline visualized in the middle, where only no-speech data is stored for upload. The remainder of this report will delve into the necessary adaptations in more detail.

### 3.1. Adaptions to the new environment

Since our earlier research was conducted using the TensorFlow library [15], TensorFlow is required on the Raspberry Pi. While the full library offers all features related to machine

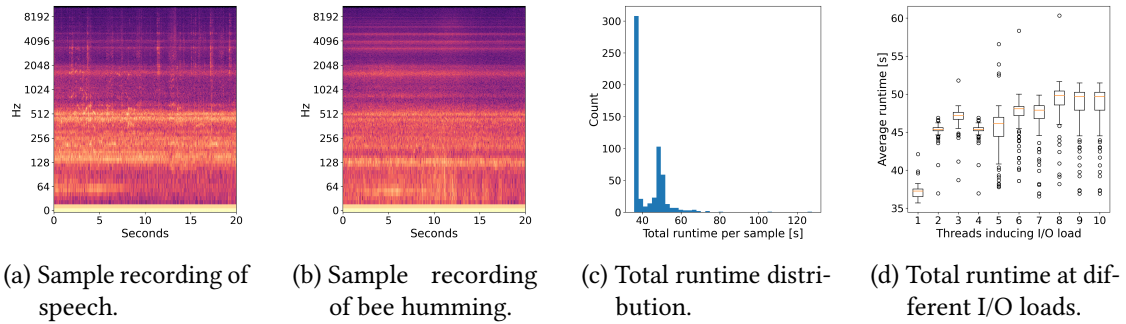


Figure 2: Sample recordings (a, b), total inference runtime per sample (c) and the average total runtime at different I/O loads (d). Note the vertical lines indicating speech in a), and the difference in the lower frequencies between a) and b). The total runtime is most strongly determined by Bulbul 's forward pass, which takes 37.30 s on average. Increasing the I/O load also increases runtime.

learning research, the 1 MB small TensorFlow Lite (TFLite) package focuses on inference and deployment and is sufficient for our purpose.

We verified the successful installation by running an official audio classification tutorial [16]. In addition to the steps in the tutorial, we had to add our user to the audio group to gain access to the microphone. Following these configuration changes, we successfully executed the official audio classification tutorial and thereby confirmed the installation and functionality of TFLite.

After installing TFLite, we migrated the Python environment and adapted the scripts. The methodology outlined in [6], requires `librosa` for audio processing. Installing it was not possible on Raspbian 11 Bullseye OS due to incompatible dependencies. Therefore, `librosa` was replaced by the `soundfile` 0.12.1 package [17], which also supports processing audio data.

The last step was migrating the best-performing model from our earlier research. Bulbul relies on the `kapre` library [18] for transforming raw audio input data to a spectrogram during the forward pass. Two spectrograms of speech and bee humming are given in fig. 2a and fig. 2b, respectively, which show different intensities in the lower frequencies. In fig. 2a, vertical lines also indicate human speech. The audio conversion to spectrogram takes place in custom layers requiring non-standard TFLite operations. To avoid installing full TensorFlow, we replaced the spectrogram and magnitude-scaling layer of the original model with TFLite-compatible ones and transferred trained weights. After conversion to the `.tflite` format, we confirmed that the model performance did not suffer by re-running experiments from [6]. The  $k$ -NN model does not run out-of-the-box as well, and was re-initialized on the device with  $k = 5$ . A tutorial of the setup process including code is available at <https://professor-x.de/beepi-speech>.

### 3.2. Performance boundaries

The smart beehive runs various sensor recording services, whose average CPU load over 15 min is 25.48 % (std 15.05). The memory usage generally is small, ranging from 5.78 % to 12.59 %, indicating that around 100 MB of the 1 GB RAM are occupied. These statistics show that the Raspberry Pi has enough resources available for real-time audio classification and uploading

data without speech. To evaluate the boundary at which this is no longer feasible, we loaded the Bu1bu1 and  $k$ -NN model, and separately timed the audio pre-processing, embedding extraction, and prediction over 117 files uploaded to our research beehive. For that, we disabled the sensor recording services and used the Linux commands `stress` to induce and `nice` to prioritize artificial CPU load. Despite these severe restrictions, the system scheduler ensures that our classification runs in real time. Further, we also evaluated the prediction performance under varying I/O loads using the `stress --io n_threads` command with  $\{1..10\}$  threads. The results in fig. 2d show that for more than 5 threads inducing I/O load, more outliers arise, and the average total runtime and its standard deviations increase. However, while these increases indicate that our script has more waiting time, the system is still capable of real-time inference.

### 3.3. Real-time Classification of Incoming Audio Data

To verify the performance in a controlled, realistic setting, we re-activated the sensor recording system and ran the inference five times in sequence, yielding 585 measurements in total. A histogram of the overall runtimes is given in fig. 2c, which shows that the majority is predicted in less than 60 s, with the  $k$ -NN having negligible influence. Of the 585 tested files, only 19 take longer than 60 s to classify. Of these delays, according to logs, 10 are caused by the camera recording, 4 by measuring the finedust concentration, and for 5 the source is unclear. Apart from this, the parallel recording of the sensor modalities had no negative impact on model runtimes. In summary, the results show that we only have around 20 min of audio data that remain unclassified over a period of roughly 10 h. On average, this translates to two audio snippets per hour that cannot be classified in real-time. Further data loss through connection failures is prevented by buffering up to 60 h of classified, non-speech data.

For evaluating the model performance on a different hive, we asked one male and one female volunteer to perform different activities (talking, playing music and singing, laughing and rough-housing, and staying silent for a fixed time period) at 1 m, 5 m 10 m distance to the hive. Our observations show that all activities are well-detected at all distances to the hive, including quiet speech.

Lastly, we let the automated audio classification run over one week, logging the predictions. The speech detected by our system can generally be mapped to real events, as consultation with the hive's owners revealed. For example, one morning, between 7 and 7:15 am, speech is detected when they take their dog for a walk. In another instance, they prepare for travel.

## 4. Conclusion

In this technical report, we describe the process of deploying a speech detection model onto a Raspberry Pi 3B in a smart beehive. We show that, after making adaptations to the Python code and model architectures, we can deploy and run the model on the mobile computer. On this hardware, our setup can predict the class of a one-minute audio recording in less than 60 s, *i.e.*, in real-time. Even when high CPU and I/O load simulate extreme scenarios, the setup is still capable of real-time inference. Lastly, using controlled activities with two volunteers as a test case, we showed that our model can predict human speech at various distances from the recording device. The next step is rolling out the models to all hives of the we4bee project.

## References

- [1] P. Davidson, M. Steininger, F. Lautenschlager, K. Kobs, A. Krause, A. Hotho, Anomaly detection in beehives using deep recurrent autoencoders, CoRR abs/2003.04576 (2020). URL: <https://arxiv.org/abs/2003.04576>. arXiv:2003.04576.
- [2] A. Žgank, Acoustic monitoring and classification of bee swarm activity using mfcc feature extraction and hmm acoustic modeling, in: 2018 ELEKTRO, IEEE, 2018, pp. 1–4.
- [3] S. Cecchi, A. Terenzi, S. Orcioni, F. Piazza, Analysis of the sound emitted by honey bees in a beehive, in: Audio Engineering Society Convention 147, Audio Engineering Society, 2019.
- [4] I. Nolasco, A. Terenzi, S. Cecchi, S. Orcioni, H. L. Bear, E. Benetos, Audio-based identification of beehive states, in: ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2019, pp. 8256–8260.
- [5] P. Janetzky, M. Schaller, A. Krause, A. Hotho, Swarming detection in smart beehives using auto encoders for audio data, in: 2023 30th International Conference on Systems, Signals and Image Processing (IWSSIP), 2023, pp. 1–5. doi:10.1109/IWSSIP58668.2023.10180253.
- [6] P. Janetzky, P. Davidson, M. Steininger, A. Krause, A. Hotho, Detecting presence of speech in acoustic data obtained from beehives., in: DCASE, 2021, pp. 26–30.
- [7] A. Saeed, Urban Sound Classification, 2016. URL: <https://github.com/aqibsaeed/Urban-Sound-Classification>, accessed: 2023-09-07.
- [8] K. J. Piczak, Esc: Dataset for environmental sound classification, in: Proceedings of the 23rd ACM international conference on Multimedia, 2015, pp. 1015–1018.
- [9] K. J. Piczak, Environmental sound classification with convolutional neural networks, in: 2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP), IEEE, 2015, pp. 1–6.
- [10] T. Grill, J. Schlüter, Two convolutional neural networks for bird detection in audio signals, in: 2017 25th European Signal Processing Conference (EUSIPCO), IEEE, 2017, pp. 1764–1768.
- [11] N. S. Altman, An introduction to kernel and nearest-neighbor nonparametric regression, *The American Statistician* 46 (1992) 175–185.
- [12] E. Fix, J. L. Hodges, Discriminatory analysis. nonparametric discrimination: Consistency properties, *International Statistical Review/Revue Internationale de Statistique* 57 (1989) 238–247.
- [13] A. L. Maas, A. Y. Hannun, A. Y. Ng, et al., Rectifier nonlinearities improve neural network acoustic models, in: Proc. icml, volume 30, Atlanta, GA, 2013, p. 3.
- [14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, *The journal of machine learning research* 15 (2014) 1929–1958.
- [15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu,

- X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL: <https://www.tensorflow.org/>, software available from tensorflow.org.
- [16] Tensorflow lite python audio classification example with raspberry pi, [https://github.com/tensorflow/examples/tree/master/lite/examples/audio\\_classification/raspberry\\_pi](https://github.com/tensorflow/examples/tree/master/lite/examples/audio_classification/raspberry_pi), 2022. Accessed: 2023-07-13.
- [17] B. Bechtold, soundfile audio library, <https://pypi.org/project/soundfile/>, 2023. Accessed: 2023-07-13.
- [18] K. Choi, D. Joo, J. Kim, Kapre: On-gpu audio preprocessing layers for a quick implementation of deep neural network models with keras, in: Machine Learning for Music Discovery Workshop at 34th International Conference on Machine Learning, ICML, 2017.