ВЕСТНИК ТОМСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

2018 Управление, вычислительная техника и информатика № 45

УДК 004.8, 638.14
DOI: 10.17223/19988605/45/8

V.A. Kulyukin, S. Mukherjee, Yu.B. Burkatovskaya

CLASSIFICATION OF AUDIO SAMPLES BY CONVOLUTIONAL NETWORKS IN AUDIOBEEHIVE MONITORING

In the investigation, we consider the problem of classification of audio samples resulting from the audio beehive monitoring. Audio beehive monitoring is a key component of electronic beehive monitoring (EBM) that can potentially automate the identification of various stressors for honeybee colonies. We propose to use convolutional neural networks (ConvNets) and compare developed ConvNets in classifying audio samples from electronic beehive monitors deployed in live beehives. As a result, samples are placed in one of the three non-overlapping categories: bee buzzing (B), cricket chirping (C), and ambient noise (N). We show that ConvNets trained to classify raw audio samples perform slightly better than ConvNets trained to classify spectrogram images of audio samples. We demonstrate that ConvNets can successfully operate in situ on low voltage devices such as the credit card size raspberry pi computer.

Keywords: deep learning; machine learning; convolutional neural networks; audio classification; audio processing; electronic beehive monitoring.

Beekeepers ascertain the state of their honey bee colonies by listening to them, because bee buzzing carries information on colony behavior and phenology. While experienced beekeepers can tell audio changes in sounds produced by stressed colonies, they may not always be able to determine the exact causes of the changes without hive inspections. Unfortunately, hive inspections can be quite invasive in that they disrupt the life cycle of bee colonies and put additional stress on the bees.

Audio beehive monitoring is a key component of EBM that attracts considerable research and development effort, because honey bees generate specific sounds when exposed to stressors such as failing queens, predatory mites and airborne toxicants. Bromenschenk et al. [1] designed a system for profiling acoustic signatures of honeybee colonies, analyzing these signatures, and identifying various stressors for the colonies from this analysis. A foam insulated microphone probe assembly is inserted through a hole drilled in the back of the upper super of a Langstroth hive that consists of two supers. The probe's microphone is connected via a microphone amplifier to a computer with an audio card and audio processing software that performs the Fast Fourier Transform (FFT) of captured wav files. The frequency spectrum is processed into a running average and an overall average to associate sound spectra with acoustic variations. In an experiment, bee colonies treated with naptha, ammonia, or toluene produced statistically different acoustic signatures.

Ferrari et al. [2] assembled a system for monitoring swarm sounds in beehives. The system consists of a microphone, a temperature sensor, and a humidity sensor placed in a beehive and connected to a computer in a nearby barn via underground cables. In a field experiment, the sounds were recorded at a sample rate of 2 kHz and analyzed with Matlab. The researchers monitored three beehives for 270 hours and observed that swarming was indicated by an increase of the buzzing frequency at about 110 Hz with a peak at 300 Hz when the swarm left the hive. Another finding was that a swarming period correlated with a rise in temperature from 33° C to 35° C with a temperature drop to 32° C at the actual time of swarming.

Mezquida and Martinez [3] developed a distributed audio monitoring system for apiaries. The system consists of nodes and wireless sensor units with one node per apiary and one sensor unit per hive. A node is an embedded solar-powered computer. A sensor unit consists of an omnidirectional microphone and a temperature sensor. Sensor units are placed at the bottom of each hive protected by a grid to prevent propolization. The system was tested in an apiary of 15 Langstroth hives where up to 10 hives were monitored continuously by taking 8 second audio samples every hour with a sampling rate of 6,250 Hz. The timestamped frequency spectra obtained with Fourier transform and temperature data were logged in a SQL database from May 2008 to April 2009. The researchers reported that sound volume and sound intensity at medium and low frequencies showed no identifiable daily patterns in the winter but exhibited distinguishable daily patterns in the spring.

Kulyukin et al. [4] designed an algorithm for digitizing bee buzzing signals with harmonic intervals into A440 piano note sequences to obtain symbolic representations of audio signals over specific time periods. When viewed as a time series, such sequences correlate with other timestamped data such as estimates of bee traffic levels or temperatures [5]. The note range detected by the proposed algorithm on 3421.52MB of 30-second wav files contained the first four octaves, with the lowest note being A0 and the highest note being F#4. In a 24-hour period, the peaks in the frequency counts started in the first octave (D1), shifted higher to C3 and C#3 in the third octave, and returned back to the first octave at the end of the selected time period. Several notes in the fourth octave, e.g., F#4 and C#4, were also detected but their frequency counts were substantially lower than those of the peaks in the first three octaves.

Ramsey et al. [6] presented a method to analyze and identify honeybee swarming events through a computational analysis of acoustic vibrations captured with accelerometers placed on outside walls of hives. The researchers placed two accelerometers on the outer walls of two Langstroth hives with Apis mellifera honey bees, one accelerometer per hive. A cavity was drilled in the center of the back wall of each hive. Both hives were located in close proximity to each other and approximately 10 meters away from a house with a PC running a Linux distribution. The accelerometers were placed in the cavities and connected to a dual channel conditioner placed between the hives and encased in a waterproof acrylic box. The conditioner was coupled to the indoor PC with two coaxial cables through which the captured vibration samples were saved on the PC's hard disk. An ad hoc roof was placed above both hives to control for vibrations caused by rain drops. The vibration samples, each one hour long, were logged from April to June 2009. Averaged frequency spectra were computed for a frequency resolution of 20 Hz and an averaging time of 510 seconds. The spectra were combined into one day long spectrograms which were analyzed with the Principal Component Analysis (PCA) for feature extraction. The witnessed swarming events were juxtaposed with the corresponding spectrograms and were found to exhibit a unique set of features.

In this article, we contribute to the body of research on audio beehive monitoring by designing and evaluating ConvNets that classify audio samples from microphones located above landing pads of Langstroth beehives. The proposed audio processing methods are applicable to any audio samples obtained from microphones placed inside or outside beehives. To ensure the replicability of our findings reported in this article we have made public our source code [7] and our curated dataset of 9110 audio samples [8].

1. Materials and methods

Hardware. All audio data were captured by BeePi, a multi-sensor electronic beehive monitoring system we designed and built in 2014. A BeePi monitor consists of a raspberry pi computer, a miniature camera, a microphone splitter, 3 or 4 microphones connected to the splitter, a solar panel, a temperature sensor, a battery, and a hardware clock. The solar version of BeePi is shown in Fig 1. All hardware components, except for solar panels, are placed in a Langstroth super (see Fig 2). The solar panel is placed either on top of or next to a beehive, as shown in Fig 1. We used the Pi Model B+ 512MB RAM models, Pi T-Cobblers, half-size breadboards, waterproof DS18B20 digital temperature sensors, and Pi cameras. For solar harvesting, we used the Renogy 50 watts 12 Volts monocrystalline solar panels, Renogy 10 Amp PWM solar charge controllers, and Renogy 10ft 10AWG solar adaptor kits. For power storage, we used two types of rechargeable batteries: the UPG 12V 12Ah F2 lead acid AGM deep cycle battery and the Anker Astro E7 26800mAh battery. Each BeePi unit is equipped with four Neewer 3.5mm mini lapel microphones that are placed either above the landing pad or embedded in beehive walls. In this study, the microphones were placed 10cm above the landing pad with two microphones on each side, as shown in Fig 3.

BeePi monitors have so far had four field deployments. The first deployment was in Logan, UT (Sept. 2014) when one BeePi unit was placed into an empty hive and ran exclusively on solar power for two weeks. The second deployment was in Garland, UT (Dec. 2014 – Jan. 2015) when one BeePi unit was placed in a hive with overwintering bees and successfully operated for nine out of the fourteen days of deployment exclusively on solar power and collected about 200 MB of data. The third deployment was in North Logan, UT (Apr. 2016 – Nov. 2016) where four BeePi units were placed into four beehives at two small apiaries and collected 20GB of data. The fourth deployment was in both Logan and North Logan, UT (Apr. 2017 – Sept. 2017) when four BeePi units again placed into four beehives at two small apiaries to collect 220GB of audio, video, and temperature data.

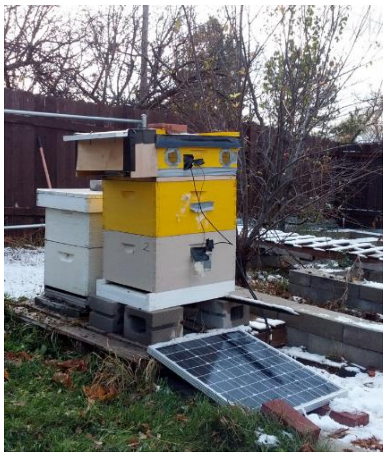
Fig. 1. Deployed BeePi monitor

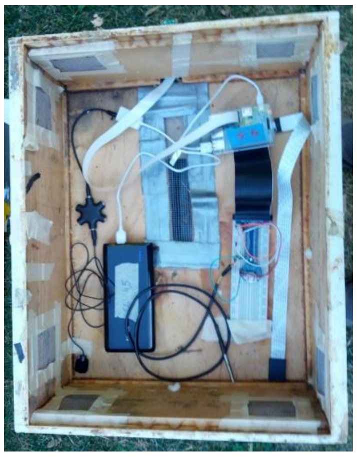
Fig. 2. BeePi hardware components

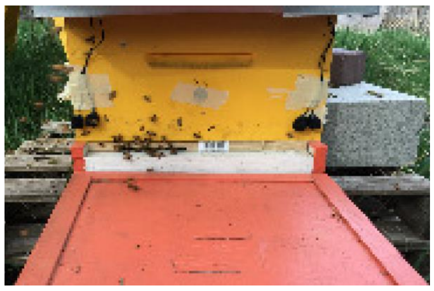
Fig. 3. Four microphones above a landing pad

Audio Data. The training and testing audio data were captured by two of our four BeePi monitors deployed in Logan, UT and North Logan, UT from April 2017 to September 2017 in four Langstroth beehives with Italian honey bee colonies. Each monitor saved a 30-second audio wav file every 15 minutes recorded with the four USB microphones placed 10cm above the beehives' landing pads and plugged into a raspberry pi computer. Each 30-second audio sample was segmented into 2-second wav samples with a 1-second overlap, resulting in 28 2-second wav samples per one 30-second file.

We obtained the ground truth classification by manually labeling 9110 2-second audio samples by listening to each sample and placing it into one of the three non-overlapping categories: bee buzzing (B), cricket chirping (C), and ambient noise (N). The B category consisted of the samples where we could hear the buzzing of bees. The C category consisted of the files where we could hear the chirping of crickets. The N category included all samples where we could not clearly hear either bee buzzing or cricket chirping. The final labeled dataset included 3,000 B samples, 3,110 C samples, and 3,000 N samples.

Convolutional Neural Networks. Representation learning is a branch of AI that investigates automatic acquisition of representations for detection and classification from raw signals. Standard machine learning techniques are limited in their ability to acquire representations from raw data, because they require considerable feature engineering to develop robust feature extractors that convert raw signals into feature vectors used in classification methods [9]. ConvNets are deep networks designed to acquire multi-layer representations from raw data. Starting from raw input, each layer is a transformation function that transforms its input to the one acceptable for the next layer. Many features of these layers are learned automatically by a general purpose procedure known as backpropagation [10]. Compositions of these transformations emulate complex classification functions. ConvNets have been successfully applied to image classification [11, 12], speech recognition [13], and bioinformatics [14].

In ConvNets, filters of various sizes are convolved with a raw input signal to obtain a stack of filtered signals. This stack is referred to as the convolutional layer. The convolved signals are normalized. A standard choice for normalization is the rectified linear units (ReLUs) that convert all negative values to 0's [15]. This layer is called the normalization layer. The size of each signal from the convolutional layer or the normalization layer can be reduced in a pooling layer where a window and a stride are used to shift the window in stride steps across the input signal retaining the maximum value from each window. The fourth type of layer in a ConvNet is the fully connected layer (FC) where the stack of processed signals is construed as a 1D vector where each value is connected via a synapse (a weighted connection that transmits a value from one unit to another) to each node in the next layer. In addition to standard layers, ConvNets can have custom layers that implement arbitrary transformation functions. The architectural features of a ConvNet that cannot be dynamically changed through backpropagation are called hyperparameters and include the number and size of features in convolutional layers, the window size and stride in maxpooling layers, and the number of neurons in FC layers.

A fundamental research objective of this study is to compare ConvNets trained on spectrograms obtained through Fourier analysis with ConvNets trained on raw audio data. Toward that end, two ConvNet architectures were designed and trained to handle two types of input signal. The first architecture (SpectConvNet) was designed to classify spectrogram images. The second architecture (RawConvNet) was designed to classify raw audio waveforms.

SpectConvNet designed to classify spectrogram images is shown in Fig 4. The input signal is a 100×100×32 RGB image of the spectrogram of a 2-second wav audio sample. The signal goes through a 100×100×100 convolutional (Conv-2D) layer with 100 filters and then through a 50×50×100 maxpooling (Maxpool-2D) layer with a kernel size of 2. The first Maxpool-2D layer is followed by a stack of two Conv-2D layers with 100 and 200 filters, respectively. The output of the second Conv-2D layer is processed by a 25×25×200 Maxpool-2D layer with a kernel size of 2 and then by a stack of two fully connected (FC) layers.

The first FC layer has fifty neurons whereas the second FC layer has three neurons corresponding to the B, C, and N classes. The first FC layer (FC-50) uses the rectified linear unit (ReLU) activation function. To avoid overfitting, random dropout with a keep probability of 0.5 is applied to the output of the first FC layer. The second FC layer (FC-3) has the softmax activation function [16]. The softmax function is used to represent a probability distribution over *k* different possible outcomes by mapping a *k*-dimensional vector of arbitrary real values into another *k*-dimensional vector whose values are in the range (0,1) that add up to 1. The softmax function emphasizes the largest values and de-emphasizes the smallest values. Specifically, if $X=[x_1,...,x_k]$ is a *k*-dimensional vector, then the softmax function maps each element x_i of X to the real value specified in Eq. 1. The softmax function is used to transform a *k*-dimensional vector X to a *k*-dimensional vector Z, as shown in Eq. 2.

$$S(x_i) = e^{x_i} \Big/ \sum_{j=1}^{k} e^{x_j}, \quad \text{where } 1 \le i \le k; \tag{1}$$

$$Z = \left[S(x_1),...,S(x_k) \right]. \tag{2}$$

In multiclass classification, given a sample vector X and its score vector Z, the probability of X belonging to class j is given in Eq.3, where Z_i is the i-th element of Z.

$$P(X = i | Z) = e^{Z_i} \Big/ \sum_{j=1}^{k} e^{Z_j}. \tag{3}$$

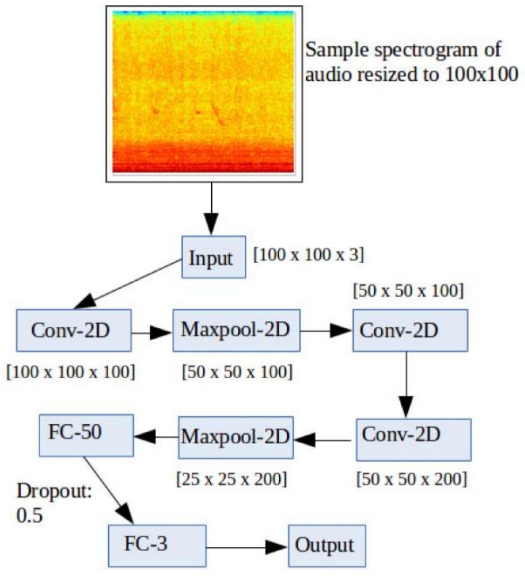

Fig. 4. SpectConvNet architecture

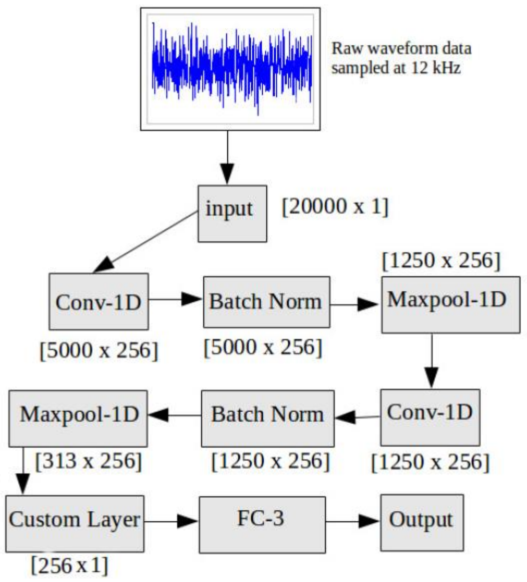

Fig. 5. RawConvNet architecture

RawConvNet designed to classify raw audio samples is shown in Fig 5. The input signal to RawConvNet is a raw audio wav file downsampled to 12 245 kHz and normalized to have a mean of 0 and a variance of 1. The input signal goes to a 5000×256 Conv-1D layer with 256 filters with a filter size *n*, where $n \in \{3, 10, 30, 80, 100\}$, and a stride of 4. To discover the optimal sizes of receptive fields for the convolutional layers, the following receptive field sizes for both layers were used during training and evaluation: 3×1, 10×1, 30×1, 80×1, and 100×1. The impact of the receptive field's size on the ConvNet's performance is described in the section about the experiments. The output of the Conv-1D layer goes into a 5000×256 batch normalization layer to reduce internal covariate shift [17]. The normalized output is given to a 1250×256 Maxpool-1D layer with a kernel size of 4. After the first Maxpool-1D layer the signal goes through a 1250×256 Conv-1D layer with a stride of 1 the output of which is normalized with another 1250×256 batch normalization layer and subsequently maxpooled with a 313×256 Maxpool-1D layer. The output of this Maxpool-1D layer is processed by a 256×1 custom layer that calculates the mean of the tensor along the first axis. A tensor is an *n*-dimensional array. For example, a 1D tensor is a vector, a 2D tensor is a matrix, a 3D tensor is a cube, and a 4D tensor is a vector of cubes, a 5D tensor is a matrix of cubes and so on. The custom layer is designed to calculate the global average over each feature map, thus reducing each feature map tensor into a single float value.

The output of the custom layer goes into a FC layer with 3 neurons (FC-3) that correspond to the B, C, and N classes. Both convolutional layers use ReLUs. Each convolutional layer has L_2 regularization and a weight decay of 0.001. Batch normalization is applied to the output of each convolutional layer to reduce the computational costs and accelerate the learning process. Batch normalization forces the activations in feature maps to have a standard deviation of 1 and a mean of 0. The output of each convolutional layer is a tensor of rank 4, i.e., $[B, H, W, D]$, where B is a batch size, $H\times W$ is a feature map size, and D is the number of channels or dimensions. The output of the batch normalization layer is controlled by two parameters that are learned to best represent the activation of the feature maps.

2. Experiments

Both ConvNet architectures were trained on a Xeon(R) CPU W3565 computer with 5.8GiB of RAM and 64-bit Ubuntu 14.04 LTS. Both architectures were trained with the Adam optimizer and categorical cross-entropy [18]. The Adam optimizer is a variant of stochastic gradient descent that can handle sparse gradients on noisy problems. Both architectures were trained for 100 epochs with a batch size of 128 and a 70/30 train/test split of the 9110 audio samples described in the section on audio data, which resulted in 6377 audio samples used for training (70% of 9110 audio samples) and 2733 audio samples used for testing (30% of 9110 audio samples). Table 1 gives the performance results comparing SpectConvNet with RawConvNet with different sizes of the receptive field. The results in the table show that when the validation accuracy is lowest (98.87%) when the size of the receptive field is 3 and highest (99.93%) when the size of the receptive field is 80. The runtime per epoch is slowest when the receptive field's size is 3. Figures 5 and 6 show the validation loss and accuracy curves for RawConvNet for different receptive field sizes, respectively. As the number of training epochs increases the validation losses decrease while the validation accuracies increase for each receptive field size.

Table 1

Performance Summary of Two ConvNet Architectures

Model Name	Training Loss	Training Acc	Validation Loss	Validation Acc	Runtime per Epoch
Spectrogram ConvNet	0,00619	99,04%	0,00702	99,13%	690 secs
RawConvNet (*n* = 3)	0,02759	99,24%	0,03046	98,87%	460 secs
RawConvNet (*n* = 10)	0,01369	99,74%	0,01429	99,60%	462 secs
RawConvNet (*n* =30)	0,00827	99,91%	0,00679	99,71%	465 secs
RawConvNet (*n* = 80)	0,00692	99,85%	0,00432	99,93%	462 secs
RawConvNet (*n* = 100)	0,00456	99,97%	0,00785	99,74%	505 secs

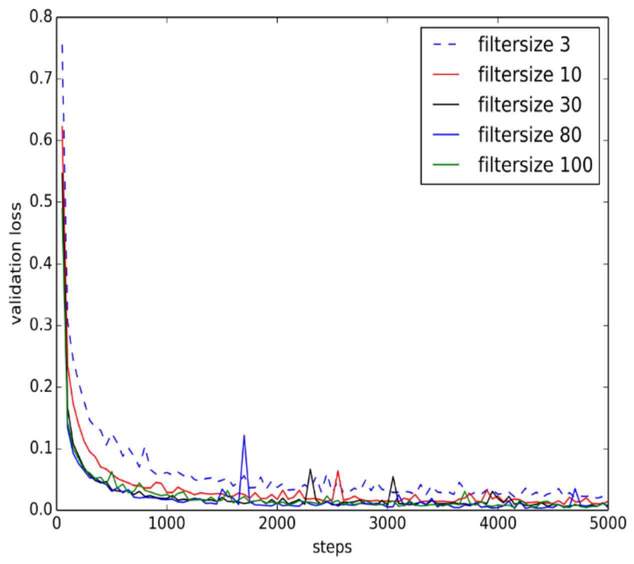

Fig. 6. Validation Loss Curves for RawConvNet

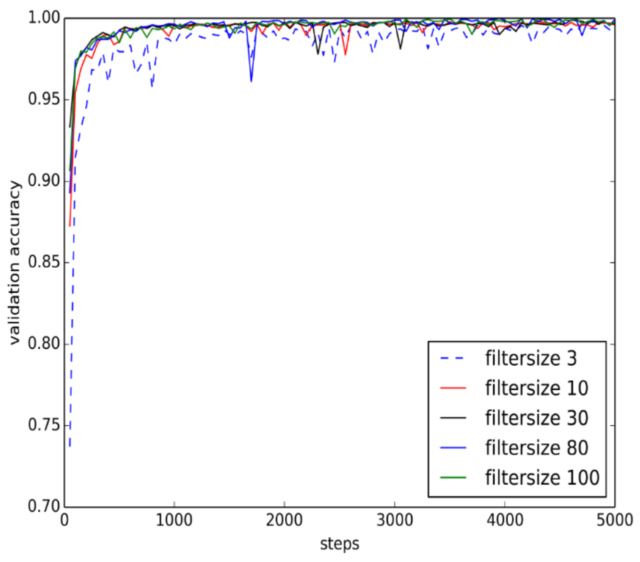

Fig. 7. Validation Accuracy Curves for RawConvNet

To estimate the contribution of the custom layer, three deeper models were designed where the custom layer was replaced with various combinations of FC and convolutional layers. In the first ConvNet (ConvNet 1), the first three layers and the fifth layer are identical to RawConvNet and the fourth layer is replaced with an FC layer with 256 neurons. In the second ConvNet (ConvNet 2), the first three layers are identical to RawConvNet. In layer 4, batch normalization is performed on the output of layer 3 and is given to layer 5 consisting of an FC layer with 256 neurons with a dropout of 0.5. Layer 6 is identical to the last layer in RawConvNet and ConvNet 1 and consists of an FC layer with 3 neurons that correspond to the B, C, and N classes with the softmax activation function. In the third ConvNet (ConvNet 3), the first three layers are identical as the three layers of ConvNets 1 and 2. In layer 4, the output of layer 3 is convolved with 256 filters with a filter size of 3 and a stride of 1. In layer 5, batch normalization is performed on the output of layer 4 before passing it to layer 6 that consists of an FC layer with 256 neurons with a dropout of 0.5. Layer 7 is the same as in ConvNets 1 and 2. Table 2 gives the results of comparing RawConvNet with ConvNet 1, ConvNet 2, and ConvNet 2. All four networks were trained for 100 epochs. The results indicate that the validation accuracies of ConvNets 1, 2, and 3 are slightly lower than the validation accuracy of the custom ConvNet. However, ConvNets 1, 2, and 3 have substantially higher validation losses. The results indicate that increasing the number of layers and adding more depth may not necessarily result in higher classification performance.

Table 2

Comparing RawConvNet with ConvNets 1, 2, and 3 after 100 Epochs

Model Name	Training Loss	Training Acc	Validation Loss	Validation Acc	Runtime per Epoch
RawConvNet (*n* = 80)	0,00619	99,04%	0,00702	99,13%	690 secs
ConvNet 1	0,02759	99,24%	0,03046	98,87%	460 secs
ConvNet 2	0,01369	99,74%	0,01429	99,60%	462 secs
ConvNet 3	0,00827	99,91%	0,00679	99,71%	465 secs

RawConvNet was persisted in a Python pickle file on the Linux computer where it was trained and then saved on the sdcard of a raspberry pi 3 model B v1.2. The raspberry pi was powered with a fully charged Anker Astro E7 26800mAh portable battery.

Two experiments were then performed on the raspberry pi to estimate how feasible it is to do in situ audio classification with the ConvNet on the raspberry pi. Two hundred thirty second raw audio samples from the audio data set were placed in a local folder on the pi computer. In the first experiment, a script was implemented in Python 2.7 to run every 15 minutes. The script would load RawConvNet from the pickle file, load an audio sample from the local folder, split it into non-overlapping two second segments, and classify each two second segment with RawConvNet. In the first experiment, the fully charged battery supported audio classification for 40 hours during which 162 30-second samples were processed. It took the algorithm, on average, 13.66 seconds to process one 30-second audio sample.

In the second experiment, the script was modified to process four 30-second audio files once every 60 minutes to estimate whether a batch approach to in situ audio classification would result in better power efficiency, because the persisted ConvNet would be loaded into memory only once per every four thirty second audio samples. In the second experiment, the fully charged Anker battery supported audio classification for 43 hours during which 172 thirty second audio samples were processed. It took the algorithm 37.68 seconds, on average, to classify a batch of four 30-second audio sample, which results in 9.42 seconds per a 30-second audio sample.

3. Conclusion

Our experiments indicate that that it is possible to build ConvNets that classify raw audio samples as well as ConvNets that classify audio samples by converting them into spectrograms. On our data set of 9110 audio samples, RawConvNet, our ConvNet trained to classify raw audio samples, achieved a validation accuracy of 99.93%, which is slightly higher than the validation accuracy of 99.13% achieved by SpectConvNet, our ConvNet trained to classify spectrograms. While it may be possible to experiment with different ConvNet architectures to improve the validation accuracy of SpectConvNet, the validation accuracy of 99.93% achieved by RawConvNet does not leave much room for improvement.

We investigated whether making ConvNets deeper by adding more layers results in a higher classification performance than a single custom layer. Our experiments suggest that adding more layers to ConvNets does not necessarily improve classification performance. Thus, deeper is not necessarily better. Specifically, our experimental results show that, on our data set, RawConvNet, a shallower ConvNet with a single custom layer, achieved a higher validation accuracy (99.93%) than each of the validation accuracies of the three deeper ConvNets without a custom layer (i.e., 97.59%, 98.21%, 98.02%). Our experiments show that RawConvNet also achieved a validation loss (0.00432), which is smaller than each of the validation losses of three ConvNets without a custom layer (i.e., 0.57427, 0.59022, 0.59429). This empirical evidence suggests that ConvNets trained to classify raw audio samples perform slightly better than ConvNets trained to classify spectrogram images of audio samples.

The deeper ConvNets required more time and consequently more power for training. Thus, RawConvNet took, on average, 462 seconds per epoch for training whereas ConvNets 1, 2, and 3 took, on average, 545 seconds, 532 seconds, and 610 seconds, respectively.

REFERENCES

1. Bromenschenk, J.J., Henderson, C.B., Seccomb, R.A., Rice, S.D. & Etter, R.T. (2007) Honey bee acoustic recording and analysis system for monitoring hive health. *U.S. Patent 2007/0224914 A1*. Sep. 27.
2. Ferrari, S., Silva, M., Guarino, M. & Berckmans, D. (2008) Monitoring of swarming sounds in bee hives for early detection of the swarming period. *Computers and Electronics in Agriculture*. 64(1). pp. 72–77. DOI: 10.1016/j.compag.2008.05.010
3. Atauri, M.D. & Llorente, M.J. (2009) Platform for beehives monitoring based on sound analysis: A perpetual warehouse for swarm's daily activity. *Spanish Journal of Agricultural Research*. 7(4). pp. 824–828. DOI: 10.5424/sjar/2009074-1109
4. Kulyukin, V.A., Putnam, M. & Reka, S.K. (2016) Digitizing buzzing signals into A440 piano note sequences and estimating forage traffic levels from images in solar-powered, electronic beehive monitoring. *Lecture Notes in Engineering and Computer Science: Proc. of the International MultiConference of Engineers and Computer Scientists*. 1. pp. 82–87.
5. Kulyukin, V.A. & Reka, S.K. (2016) Toward sustainable electronic beehive monitoring: algorithms for omnidirectional bee counting from images and harmonic analysis of buzzing signals. *Engineering Letters*. 24(3). pp. 317–327.
6. Ramsey, M., Bencsik, M. & Newton, M.I. (2017) Long-term trends in the honeybee 'whooping signal' revealed by automated detection. *PLoS One*. 12(2). DOI: 10.1371/journal.pone.0171162

7. Mukherjee, S. & Kulyukin, V.A. (2017) *Python source code for audio classification experiments with convolutional neural networks*. [Online] Available from: https://github.com/sarba-jit/EBM_Audio_Classification.
8. Kulyukin, V.A., Mukherjee, S. & Amlathe, P. (2017) *BUZZ1: A database of audio samples from live beehives captured with BeePi monitors*. [Online] Available from: https://usu.app.box.com/v/BeePiAudioData.
9. Bengio, Y., Courville, A. & Vincent, P. (2013) Representation learning: a review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 35(8). pp. 1798–1828. DOI: 10.1109/TPAMI.2013.50
10. Mitchell, T.M. (1997). *Machine learning*. McGraw-Hill.
11. Krizhevsky, A., Sutskever, I. & Hinton, G.E. (2012) ImageNet classification with deep convolutional neural networks. *Proc. of the 25th International Conference on Neural Information Processing Systems.* 1. pp. 1097–1105.
12. Farabet, C., Couprie, C., Najman, L. & LeCun, Y. (2013) Learning hierarchical features for scene labeling. *IEEE Trans Pattern Anal Mach Intell*. 35(8). pp. 1915–1929. DOI: 10.1109/TPAMI.2012.231
13. Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A.R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. & Kingsbury, B. (2012) Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. *Signal Processing Magazine*. 29(6). pp. 82–97. DOI: 10.1109/MSP.2012.2205597
14. Leung, M.K.K., Xiong, H.Y., Lee, L.J. & Frey, B.J. (2014). Deep learning of the tissue-regulated splicing code. *Bioinformatics*. 30(12). pp. 121–129. DOI: 10.1093/bioinformatics/btu277
15. Nair, V. & Hinton, G.E. (2010) Rectified linear units improve restricted boltzmann machines. *Proc. of the 27th International Conference on Machine Learning*. pp. 807–814.
16. Sutton, R.S. & Barto, A.G. (1998). *Introduction to Reinforcement Learning*. MIT Press.
17. Ioffe, S. & Szegedy, C. (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. *JMLR Workshop and Conference*. 37. pp. 448–456.
18. Kingma, D. & Ba, J. (2015) Adam: A Method for Stochastic Optimization. *Proc. of the 3rd International Conference on Learning Representations (ICLR)*.

Received: May 2, 2018

Kulyukin V.A., Mukherjee S., Burkatovskaya Yu.B. CLASSIFICATION OF AUDIO SAMPLES BY CONVOLUTIONAL NETWORKS IN AUDIO BEEHIVE MONITORING. *Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie vychislitelnaja tehnika i informatika* [Tomsk State University Journal of Control and Computer Science]. 45. pp. 68–75

DOI: 10.17223/19988605/45/8

Кулюкин В.А., Мукерджи С., Буркатовская Ю.Б. КЛАССИФИКАЦИЯ АУДИОФАЙЛОВ С ПОМОЩЬЮ СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ В ЭЛЕКТРОННОМ АУДИОМОНИТОРИНГЕ УЛЬЕВ. *Вестник Томского государственного университета. Управление, вычислительная техника и информатика.* 2018. № 45. С. 68–75

В работе решается задача классификации аудиофайлов, полученных в результате аудиомониторинга пчелиных ульев. Аудиомониторинг улья является ключевым компонентом электронного мониторинга ульев, с помощью которого потенциально можно автоматизировать идентификацию различных стрессовых факторов для пчелиных семей. Предлагается использовать сверточные нейронные сети, которые сравниваются по качеству классификации образцов звука от систем электронного мониторинга ульев, развернутых в живых ульях. В результате файлы относятся к одной из трех непересекающихся категорий: жужжание пчел (B), стрекот сверчков (C) и шум (N). Показано, что сети, обученные для классификации необработанных аудиофайлов, лучше, чем сети, обученные для классификации спектрограмм. Продемонстрирована успешная работа сетей in situ на низковольтных устройствах, таких как компьютер raspberry pi.

Ключевые слова: глубокое обучение; сверточные нейронные сети; классификация аудио; обработка аудио; электронный мониторинг пчелиных ульев.

KULYUKIN Vladimir Alekseevich (Candidate of Technical Sciences, Associate Professor of the Department of computer science Utah State University, Logan, USA).
E-mail: vladimir.kulyukin@usu.edu

MUKHERJEE Sarbajit (PhD Student of the Department of computer science Utah State University, Logan, USA).
E-mail: sarbajit.mukherjee@aggiemail.usu.edu

BURKATOVSKAYA Yulia Borisovna (Candidate of Physical and Mathematical Sciences, Associate Professor, National Research Tomsk Polytechnic University, Russian Federation).
E-mail: tracey@tpu.ru