DCASE 2016 ACOUSTIC SCENE CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS

Michele Valenti[*1], *Aleksandr Diment*[2], *Giambattista Parascandolo*[2], *Stefano Squartini*[1], *Tuomas Virtanen*[2]

[1]Università Politecnica delle Marche, Department of Information Engineering, Ancona, Italy

[2]Tampere University of Technology, Department of Signal Processing, Tampere, Finland

ABSTRACT

This workshop paper presents our contribution for the acoustic scene classification (ASC) task proposed for the "detection and classification of acoustic scenes and events" (DCASE) 2016 challenge. We propose the use of a convolutional neural network trained to classify short sequences of audio, represented by their log-mel spectrogram. We also propose a training method that can be used when the system validation performance saturates as the training proceeds. The system is evaluated on the public ASC development dataset provided for the DCASE 2016 challenge. The best accuracy score obtained by our system on a four-fold cross-validation setup is 79.0% which constitutes a 8.8% relative improvement with respect to the baseline system.

Index Terms— Acoustic scene classification, convolutional neural networks, DCASE, computational audio processing

1. INTRODUCTION

When we talk of ASC we refer to the capability of a human or an artificial system to understand an *audio context*, either from an on-line stream or from a recording. "Context" or "scene" are concepts that humans commonly use to identify a particular acoustic environment, *i.e.* the ensamble of background noises and sound events that we associate to a specific audio scenario, like a restaurant or a park. For humans this may look like a simple task: complex calculations that our brain is able to perform and our extensive life experiences allow us to easily associate these ensembles of sounds to specific scenes. However, this task is not trivial for artificial systems. The interest in computational ASC lies in its many possible applications, like context-aware computation [1], intelligent wearable interfaces [2] and mobile robot navigation [3]. In the field of machine learning different models and audio feature representations have been proposed to deal with this task, especially in the last few years, thanks to the contributions to the previous DCASE challenge in 2013 [4]. Some examples of classifiers used in the previous challenge are Gaussian mixture models (GMMs) [5], support vector machines [6] and tree bagger classifiers [7].

Nowadays, application of convolutional neural networks (CNNs) for audio-related tasks is becoming more and more widespread, for example in speech recognition [8], environmental sound classification [9] and robust audio event recognition [10]. To the best of our knowledge this is the first work introducing a CNN-based classifier specifically designed for ASC.

Our system is designed to output class prediction scores for short audio sequences. During training a recently-proposed regularization technique is used, *i.e.* batch normalization. In addition, we propose a training procedure that will allow the classifier to achieve a good generalizing performance on the development dataset. Hence our intent is to propose a system capable of improving the baseline system, represented by a GMM [11] classifier, and to give a novel contribution for future development in the use of neural networks for the ASC task.

In Section 2 a brief background about CNNs is given. Then, a detailed description of the proposed system is reported in Section 3. Finally, results for the proposed model and comparisons between different configurations are presented in Section 4 and our conclusions are reported in Section 5.

2. CONVOLUTIONAL NEURAL NETWORKS

In a CNN inputs are processed by small computational units (neurons) organized in a layered structure. The most remarkable feature that makes CNNs a particular subset of feed-forward neural networks is the presence of convolutional layers. Convolutional layers are characterized by neurons (called kernels) that perform subsequent non-linear filtering operations along small context windows of the input, *i.e.* their receptive fields (RFs). This localized filtering is a feature known as local connectivity and it represents one key characteristic for obtaining invariance against input pattern shifts. Parameters defining the RF are its width (L), height (H), depth (D) and stride. The area ($L \times H$) defines the dimension of the kernel's context window and the stride defines how much this window will slide between two filtering operations. Both area and stride are free parameters, whereas the depth is the same as the input's. For example, if the input is a three-channel (RGB) picture the network will be fed with a tensor with depth $D = 3$. In an audio analysis scenario the input can be a spectrogram, represented by a bi-dimensional matrix, which has unitary depth dimension ($D = 1$). Moreover, if the convolutional layer is acting on the output of another convolutional layer, then D will be equal to the number of kernels of the former layer.

Non-linear filtering consists of two steps. In the first place an output is calculated through a linear combination of each pixel currently seen by the RF. Then, this output is fed into a non-linear function, *i.e.* the activation function. Activation functions that are typically used are the sigmoid, the hyperbolic tangent or the rectifier function. Finally, subsequent kernel outputs are collected in matrices that are called feature maps.

Pooling layers are usually placed after each convolutional layer to reduce each feature map dimensions and to enhance the network invariance to input pattern shifts. The most common pooling layer is formed by filters that operate with non-overlapping windows by extracting the highest value from the area, *i.e.* max-pooling. Therefore, the stacking of multiple convolutional and pooling layers will make

[*]This work has been done during an internship at Tampere University of Technology.

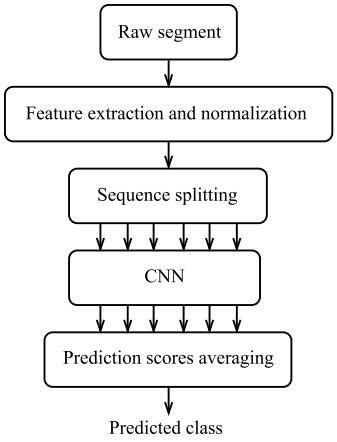

Figure 1: Block diagram of the proposed method.

the network able to extract features with a gradual increment of the input overview. The output layer is usually a fully-connected softmax layer. So, if we define y_i as the output of neuron i in the last layer we will have:

$$y_i = \text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{N}(\exp(x_j))}, \tag{1}$$

where N is the number of possible classes, x_i the input to the non-linearity and y_i the prediction score for the input sequence to belong to the i^{th} class. Hence the overall output is a vector $\mathbf{y}$ containing all network's prediction scores associated to each class. If we let y_j to be the highest of these scores, the predicted class for the input sequence will be the j^{th} class. In order to optimize the network parameters a comparison between the prediction vector $\mathbf{y}$ and a target vector is performed in terms of a loss function. Typically used loss functions are the mean squared error or the categorical cross-entropy.

3. PROPOSED METHOD

In this section we describe our method and the architecture chosen for the proposed system. The main steps are represented as a block diagram in Figure 1.

3.1. Feature representation and preprocessing

The feature representation we choose for our system is the log-mel spectrogram. To calculate it we apply a short-time Fourier transform (STFT) over windows of 40 ms of audio with 50% overlap and Hamming windowing. We then square the absolute value of each bin and apply a 60-band mel-scale filter bank. Finally, the logarithmic conversion of the mel energies is computed. The whole feature extraction process has been implemented in Python using the librosa [12] library.

After the extraction process we normalize each bin by subtracting its mean and dividing by its standard deviation, both calculated on the whole training set of each fold. We then split normalized spectrograms into shorter spectrograms, which we will call sequences hereafter. Tests with different sequence lengths are reported in Section 4. Unlike frames used for the STFT, we choose sequences to be non-overlapping. At the end of this process the input to the CNN is a matrix which can be treated as a mono-channel image.

3.2. Proposed architecture

The proposed model consists of a deep CNN and it is represented in Figure 2. Parameters we report here are chosen as a result of experiments aimed to test different kernel numbers and different RF areas.

The first layer performs a convolution over the input spectrogram with 128 kernels characterized by 5×5 RFs and unitary depth and stride in both dimensions. The obtained feature maps are then sub-sampled with a max-pooling layer which operates over 5×5 non-overlapping squares. The second convolutional layer is the same as the first one, with the exception that more kernels (256) are used in order to grant higher level representation. The second and last sub-sampling is then performed aiming to the "destruction" of the time axis. Therefore, we use a max-pooling layer which operates over the entire sequence length and, on the frequency axis, only over four non-overlapping frequency bands. The activation function used for kernels in both convolutional layers is the rectifier function, therefore kernels are usually called rectifier linear units (ReLUs) [13]. Finally, since the classification involves 15 different classes, the last is a softmax layer composed of 15 fully-connected neurons.

The classification for the whole segment is obtained by averaging all prediction scores obtained for its sequences. Recalling the notation introduced in Section 2, the CNN output $\mathbf{y}^{(i)}$ is now a vector containing all class-wise prediction scores for the i^{th} sequence. Then, the predicted class c^* for the whole segment is calculated as:

$$c^* = \arg\max_c \left[\frac{1}{M} \sum_{i=1}^{M} y_c^{(i)} \right], \tag{2}$$

where M is the number of sequences into which the segment is split and $y_c^{(i)}$ is the c^{th} entry of $\mathbf{y}^{(i)}$. In other words, the predicted class is the position of the maximum entry y_{c^*} in the vector given by the average of all prediction vectors output for each sequence.

The system is implemented with the Keras library (vers. 1.0.4) [14] for Python and its training is performed with an Nvidia Tesla K80 GPU showing an average training time of 50 s per epoch. The loss function we use for training is the categorical cross-entropy and the optimization algorithm we choose for its minimization is the adaptive momentum (adam) [15]. Basing on preliminary experiments we propose to use the optimizer default parameter configuration.

3.3. Regularization and model training

Batch normalization, introduced in [16], is a technique that addresses the issue described by Shimoidara *et al.* [17], known as internal covariate shift. We can look at batch normalization as an intermediate layer placed after each of the two convolutional layers in order to whiten the output of such layers. As showed in [16] and in our preliminary experiments, this practice can drastically reduce the training convergence time. A batch normalization layer applies a linear transformation $\text{BN}_{\beta,\gamma}$ to its input x as follows:

$$\text{BN}_{\beta,\gamma}(x) = \frac{\gamma}{\sqrt{\text{Var}[x] + \epsilon}} \cdot x + \left(\beta - \frac{\gamma \cdot \text{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} \right), \tag{3}$$

where $\text{E}[x]$ and $\text{Var}[x]$ are the mean and the variance of the input to the batch normalization layer, calculated for a batch of samples. Moreover, γ and β represent the transformation parameters that will be learned during training. We use batch normalization to normalize kernel outputs, therefore we have one γ and one β for each kernel.

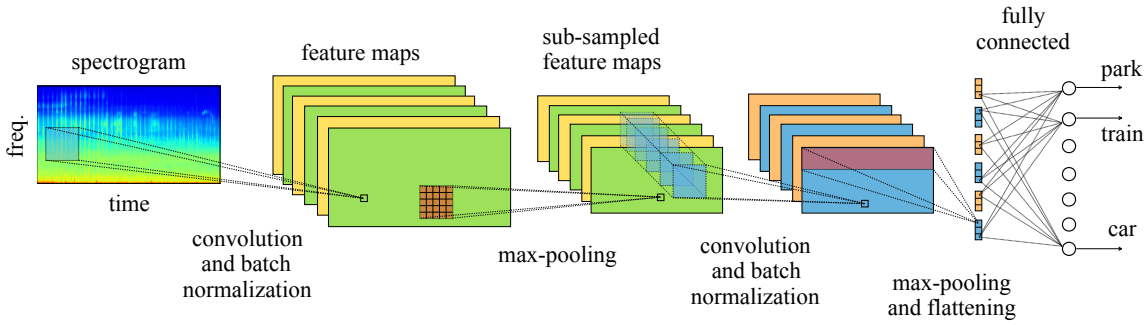

Figure 2: Block scheme of the used convolutional neural network. Max-pooling windows are represented in red, kernel RFs are in light blue.

By using batch normalization we increase the model complexity, but preliminary experiments showed better performance and a drastic reduction of the number of epochs needed for training convergence.

The proposed training method consists of two phases. The first, called *non-full training*, starts with a splitting of the whole training data into two subsets: one for training and one for validation. Every epoch we collect training spectrograms into class-wise feature lists so to randomly shuffle and time-shift them before the sequence splitting. This is done in order to increase the input variability, hence showing the network always slightly different sequence spectrograms. Then, every five epochs we check the segment-wise performance on both training and validation sets according to the metrics described in Section 4. After the check, we save the network parameters if the segment-wise validation score has improved. Finally, we stop the training if no improvement is recorded after 100 epochs. With this setup it is possible to notice that the segment-wise validation performance is prone to saturate. This means that the score starts to oscillate around a fixed, stable value. When this happens we say that the system has converged, therefore it is possible to proceed to the second phase, which we call *full training*. In this phase we decide to re-train the network on all the training data for a fixed number of epochs. This number is chosen by looking at the convergence time of segment-wise validation accuracies during the non-full training. A model trained this way will reach a convergence state without excessive overfit and making the best of all the available training data. This fact may turn out to be particularly desirable when dealing with small datasets, such as in this case.

4. EVALUATION

4.1. Dataset and metrics

For our evaluation we use the DCASE 2016 [11] development dataset as provided at the beginning of the challenge, when mobile phone interferences were not annotated. We do not consider this a problem, since only less than the 1% of the total audio duration is affected. The dataset consists of 1170 audio segments of thirty seconds, equally distributed between 15 different classes, and of text files including both the annotated ground truth and the recommended data subdivision. The 15 classes are: beach, bus, café/restaurant, car, city center, forest path, grocery store, home, library, metro station, office, park, residential area, train and tram.

Groups of segments have been obtained by splitting a longer audio file recorded in a single location. Due to this fact it is important not to train and evaluate the network performance on segments coming from the same location, since this would falsify the generalization score. Because of this, we decide to use the training and test subsets recommended in the dataset annotations. The train/test split gives us approximatively 880 training segments for each fold, some folds having fewer segments. This is due to the fact that different long recordings have been split into a variable number of segments, ranging from three to ten. This means that the distribution of per-location segments is not uniform. Similarly to what was done for the recommended sets, for the training/validation split we decide how many locations to use for validation and then pick all segments coming from those locations.

The model is evaluated according to a four-fold cross-validation scheme. For the evaluation of each fold, per-class accuracies are initially calculated on the test set on a segment-wise level. These accuracies are obtained by dividing the number of correctly classified segments by the total number of segments belonging to the class. Accuracies for each fold are then obtained by averaging all the 15 per-class accuracies. Finally, the overall accuracy is calculated by averaging the four per-fold accuracies.

4.2. Classification accuracy and sequence length

Our ability to distinguish different scenes from acoustic information is influenced by the length of the sequence we can listen to. Hence, the main focus of this section is a comparison between accuracies obtained with different sequence lengths. The average convergence time we estimate, hence the chosen full training period, is 200 epochs. For full training configurations we compute mean accuracies over four experiments involving different random weights initializations. Results we report in Table 1 apparently highlight that medium-length sequences, like three or five seconds, perform better than extremely short or long sequences. The best accuracy is achieved by the three-second configuration, with an average accuracy of 75.9% with the non-full training configuration. The accuracy rises to 79.0% if full training is performed.

A deeper insight into which classes are mostly misclassified can be obtained by looking at the confusion matrix in Figure 3, which we obtained by grouping results of all folds. What emerges is that some classes — *e.g.* “park” and “residential area”, or “bus” and “train” — are often confused by the system. This may indicate that our model is relying more on the background noise of the sequence rather then on acoustic event occurrences. We believe that this can also explain why classes with very similar background noises are confused even when 30-second sequences are used. Due to results in Table 1, we choose three seconds as the sequence length used for the challenge evaluation results. For the final training we use the whole development dataset and we estimate the new convergence time to be 400 epochs. With this configuration our model reaches a 86.2% accuracy score, therefore ranking the sixth place out 57 systems submitted in the first task of

Table 1: Accuracy comparison for the proposed model trained with different sequence lengths (seq. len.) and training modes ("non-full" and "full"). Standard deviations refer to full training accuracies.

seq. len. (s)	*accuracy* (%)		
	non-full	*full*	± *s.d.*
0.5	68.2	75.4	0.75
1.5	74.0	78.4	1.19
3	75.9	**79.0**	0.68
5	74.1	78.3	0.86
10	71.5	77.3	0.88
30	74.0	75.6	0.44

Table 2: Accuracy comparison for different systems and training modes ("non-full" and "full"). Neural architectures are identified by their number of hidden layers.

system	*seq. len.* (s)	*accuracy* (%)	
		non-full	*full*
two-layer MLP (log-mel)	-	66.6	69.3
one-layer CNN (log-mel)	3	70.3	74.8
two-layer CNN (log-mel)	3	75.9	**79.0**
two-layer CNN (MFCC)	5	67.7	72.6
baseline GMM (MFCC)	-	-	72.6

the DCASE 2016 challenge.

4.3. Comparison of other systems

Here we report new comparisons with other systems that have been tested during our development. All parameter configurations are chosen in order to give each system a representation capacity that can be comparable with the proposed network's.

The first system we introduce is a multi-layer perceptron (MLP) with two hidden layers. The input layer consists of 900 nodes to which we apply log-mel features for a context window of 15 frames. We then stack two hidden layers with 512 ReLU units (both batch-normalized) and an output layer with 15 softmax neurons. The difference from the proposed network is that the output is now the class associated to the central frame of the context window, which is sliding with unitary stride along the spectrogram. Segment-wise classification is performed as described in Eq. (2), but now M represents the number of frames in a segment. With the non-full training setup the proposed model achieves a 66.6% accuracy with a convergence time of 100 epochs. A full training for 100 epochs lets the model achieve a 69.3% accuracy.

The second model we use for the comparison consists of a CNN whose input is a three-second sequence. The input is processed by only one hidden convolutional layer with 2048 ReLU kernels whose RFs cover all the mel-energy bands and a context window of 15 frames. In this very first step the frequency dimension is narrowed, hence the first layer outputs are matrices with unitary height. Then, after a batch normalization layer, we stack a max-pooling layer to shrink also the time dimension, hence obtaining a single number for each feature map. This structure is very similar to a MLP, since each kernel looks at all the mel-energy bands of a single context window at the same time. The key difference is represented by the max-pooling, which, similarly to the proposed model's, emphasizes the occurrence of a particular input pattern no matter where it appears in the sequence. This network achieves a 70.3% accuracy with the non-full training setup, which rises to 74.8% if a full training is performed for 100 epochs.

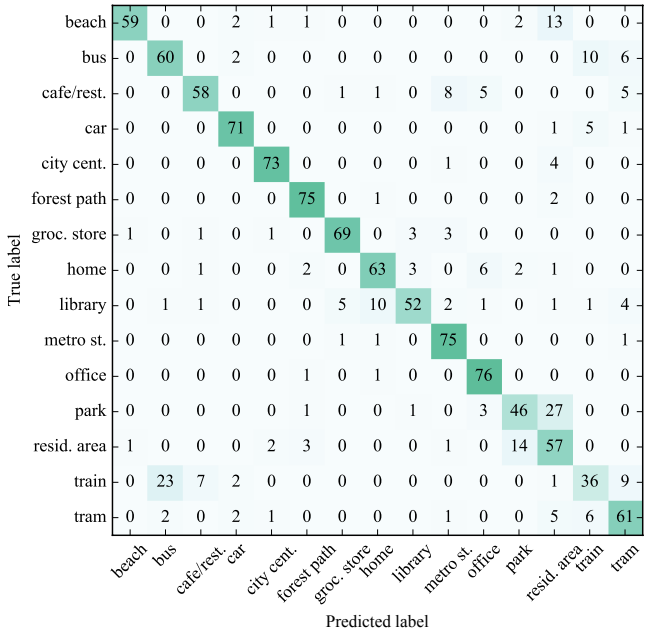

Figure 3: Confusion matrix for the proposed CNN evaluated on the four folds. Three-second sequences are used.

The last result we report intends to compare our architecture and the baseline system when both are trained on mel-frequency cepstral coefficients (MFCCs). Details about the feature parameters are reported in [11]. The baseline system trains 15 different mixtures of 16 Gaussians in order to model each of the 15 classes. Classification is performed by comparing each mixture to the test audio file in terms of log-probabilities of the data under each model. The most similar mixture gives the chosen class. We choose a sequence length of five seconds for this comparison. Our model reaches a 67.7% overall accuracy with the non-full training setup and the average convergence time on the validation data is 100 epochs. The performance reaches a 72.6% overall accuracy with a full training setup, which equals the baseline accuracy.

5. CONCLUSIONS

Our work proposes one out of many possible ways of approaching acoustic scene classification with CNNs. In doing so we reach a 79.0% accuracy on the DCASE 2016 development dataset, demonstrating that a two-layered convolutional network can achieve higher accuracies if compared to a two-layer MLP (9.7% more), a one-layer CNN (4.2% more) and a GMM-MFCC system (6.4% more). We observed also that, under particular circumstances, training the network without monitoring its generalization performance can lead to a relevant accuracy improvement. This is true especially when the lack of training data is a narrow bottleneck for the network generalizing performance.

6. ACKNOWLEDGMENTS

The authors wish to acknowledge CSC — IT Center for Science, Finland, for generous computational resources.

7. REFERENCES

[1] B. Schilit, N. Adams, and R. Want, "Context-aware computing applications," in *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*. IEEE, 1994, pp. 85–90.

[2] Y. Xu, W. J. Li, and K. K. Lee, *Intelligent wearable interfaces*. John Wiley & Sons, 2008.

[3] S. Chu, S. Narayanan, C.-C. J. Kuo, and M. J. Mataric, "Where am I? Scene recognition for mobile robots using audio features," in *2006 IEEE International Conference on Multimedia and Expo*. IEEE, 2006, pp. 885–888.

[4] D. Giannoulis, E. Benetos, D. Stowell, M. Rossignol, M. Lagrange, and M. D. Plumbley, "Detection and classification of acoustic scenes and events: An IEEE AASP challenge," in *Applications of Signal Processing to Audio and Acoustics (WASPAA), 2013 IEEE Workshop on*. IEEE, 2013, pp. 1–4.

[5] M. Chum, A. Habshush, A. Rahman, and C. Sang, "IEEE AASP scene classification challenge using hidden markov models and frame based classification," *IEEE AASP Challenge on Detection and Classification of Acoustic Scenes and Events*, 2013.

[6] J. T. Geiger, B. Schuller, and G. Rigoll, "Large-scale audio feature extraction and SVM for acoustic scene classification," in *Applications of Signal Processing to Audio and Acoustics (WASPAA), 2013 IEEE Workshop on*. IEEE, 2013, pp. 1–4.

[7] E. Olivetti, "The wonders of the normalized compression dissimilarity representation," *IEEE AASP Challenge on Detection and Classification of Acoustic Scenes and Events*, 2013.

[8] O. Abdel-Hamid, L. Deng, and D. Yu, "Exploring convolutional neural network structures and optimization techniques for speech recognition." in *INTERSPEECH*, 2013, pp. 3366–3370.

[9] K. J. Piczak, "Environmental sound classification with convolutional neural networks," in *Machine Learning for Signal Processing (MLSP), 2015 IEEE 25th International Workshop on*. IEEE, 2015, pp. 1–6.

[10] H. Phan, L. Hertel, M. Maass, and A. Mertins, "Robust audio event recognition with 1-max pooling convolutional neural networks," *arXiv preprint arXiv:1604.06338*, 2016.

[11] A. Mesaros, T. Heittola, and T. Virtanen, "TUT database for acoustic scene classification and sound event detection," *In 24th Acoustic Scene Classification Workshop 2016 European Signal Processing Conference (EUSIPCO)*, 2016.

[12] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th Python in Science Conference*, 2015.

[13] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[14] F. Chollet, "keras," https://github.com/fchollet/keras, 2015.

[15] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[16] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[17] H. Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function," *Journal of statistical planning and inference*, vol. 90, no. 2, pp. 227–244, 2000.