Simultaneous Localization and Classification of Environmental Sounds Using Deep Learning

Pierre MARCENAC

August 26, 2016
Version: 0.1

Technical University of Berlin

Electrical Engineering and Computer Science
Software Engineering and Theoretical Computer Science
Neural Information Processing

Master's thesis

Simultaneous Localization and Classification of Environmental Sounds Using Deep Learning

Pierre Marcenac

1. Reviewer Prof. Dr. Klaus Obermayer
Neural Information Processing
Software Engineering and Theoretical Computer Science

2. Reviewer Youssef Kashef
Neural Information Processing
Software Engineering and Theoretical Computer Science

Supervisors Prof. Dr. Klaus Obermayer and Youssef Kashef

August 26, 2016

Pierre MARCENAC

Simultaneous Localization and Classification of Environmental Sounds Using Deep Learning

Master's thesis, August 26, 2016

Reviewers: Prof. Dr. Klaus OBERMAYER and Youssef KASHEF

Supervisors: Prof. Dr. Klaus OBERMAYER and Youssef KASHEF

Technical University of Berlin

Neural Information Processing

Software Engineering and Theoretical Computer Science

Electrical Engineering and Computer Science

Marchstrasse 23

D-10587 and Berlin

Abstract

The purpose of the present thesis is the simultaneous classification and localization of sounds using bio-inspired computing, including deep learning and binaural recording. This falls within the scope of computational auditory scene analysis, where a machine-hearing agent shall identify and localize sounds from its environment.

The data is a mixture of binaural recordings related to a fire emergency (alarm, fire noise, people shouting, etc.) along with their related labels (for classification) and azimuths (for localization). The extracted features that are fed into the machine-hearing agent are similar to these used by the actual auditory system, such as ratemaps.

Representing sounds mainly by their spectrogram images allows us to train our data on convolutional neural networks, which appear to be particularly suitable for pattern recognition on images. Appropriate metrics help select and assess the best architectures along with a random search on the hyperparameter space.

Clean sounds can be highly accurately identified (88.8% accuracy) and localized (88.7% accuracy). Although mixed sounds identification is also achieved (71.0% balanced accuracy for the best class), their localization has proved a much harder task (60.0% balanced accuracy for the best azimuth). Side conclusions can also be drawn with respect to feature transferability in deep networks, supporting the fact that layers become increasingly more task-specific as depth increases.

Keywords: bio-inspired artificial intelligence, deep learning, convolutional neural networks, supervised classification, multitask learning, machine-hearing system.

Acknowledgment

I want to express my gratitude to my supervisor, Youssef KASHEF, who has guided me through the whole master thesis work and has given invaluable support during the project. The importance of his mentorship, his patience and his teachings could not be overestimated. I also want to express my thankfulness to my professor and supervisor, Prof. Dr. Klaus OBERMAYER, for his trust and the opportunity to accomplish my master's thesis work within his research group.

This work was supported by the Deutsche Forschungsgemeinschaft (GRK1589/2). We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X GPU used for this research.

Contents

1 Introduction

1.1 Problem setting and approach

The present thesis builds on the European Union FP7 project Two!Ears. The latter project aims at developing a computational framework for modeling active exploratory listening that assigns meaning to auditory scenes in order to help a machine listening agent extract knowledge from its direct environment. Using labeled past data and machine learning, it has to simultaneously perform identification and localization of the current observed cues in real time. In the particular case of a fire rescue scenario, a robot evolving in a building shall for example identify environmental sounds, their relative positions and move accordingly to help rescue persons. The environmental sounds are taken from real-life situations: running engine, crash, footsteps, piano, barking dog, phone, knocking, burning fire, crying baby, alarm, female speech, male speech, screams, as well as other various general sounds. In addition to these single cues, challenging acoustic scenarios require to consider mixtures of cues.

We here seek to mimic the human auditory system by separating the left and right cues to create a stereo signal, on the base of which localization is possible. Inspired by biology, we thus intend to exploit the fact that human listeners can easily recognize and localize multiple sound sources and even complex mixtures of cues in noisy and reverberant environments. We first engineer features from binaural recordings. Convolutional neural networks are then used to map the binaural features to the source class (identification) and azimuth (localization).

After presenting the problem and some elements of the underlying theory of machine learning in the introduction, we derive the methods used in terms of architecture and training of deep learning models. This then allows us to draw more general conclusions and present results on architecture selection, multitask learning and performances of the models.

1.2 Data and environmental sounds

Two types of data coming from the same dataset were used during the project. They both helped selecting the proper architectures of the neural networks and drawing conclusions on multitask learning. On one hand, clean environmental sounds were easily generated and allowed a fast prototyping. On the other hand, actual mixtures of sounds correspond to real-life cases for which appropriate architectures and models were also selected and assessed.

All sounds come from the NI General Sounds (NIGENS) database. It provides audio recordings for 14 event classes taken from everyday's life: running engine, crash, footsteps, piano, barking dog, phone, knocking, burning fire, crying baby, alarm, female speech, male speech, screams (for a total of 745 *.wav files), and a so-called "general" class (305 *.wav files of varied events, going from nature sounds to animal or human sounds). The recordings come from the commercial stock sound provider `stockmusic.com`, speech classes from the GRiD and TIMIT corpora (COOKE et al., 2006; GAROFOLO et al., 1993), and the scream sounds as well as a few general sounds from `freesound.org`. The recordings are also taken in isolation, that is without superposition of noise or other sources.

Clean sounds encompass 12 different classes taken from the NIGENS database (alarm, baby, crash, dog, engine, female speech, fire, footsteps, general, knock, phone, piano). Using the Two!Ears binaural simulator, the sounds are then rotated along 72 azimuths (ordered from $-180°$to $+180°$with a 5-degree resolution). Each sound is clean in that it is composed of one class localized at one azimuth with no reverberation,—no mixtures are allowed just as in the NIGENS database.

Mixed sounds are proper mixtures of one, two, three or four cues with reverberations and coming from 13 different classes (same classes as clean sounds as well as female and male screams and male speech) and from 72 different azimuths with various signal-to-noise ratio between the considered sources.

The methods for features extraction and data formating are further explained in section 2.1.

1.3 Deep learning for machine hearing

We here present the underlying theory on deep learning and convolutional neural networks, as well as the implementation methods we chose and the software we used for training the machine learning models.

1.3.1 Deep learning and convolutional neural networks

Deep learning refers to artificial neural networks (HAYKIN, 2004) that are composed of many layers, as well as the corresponding techniques to learn from large datasets. Whereas traditional methods of deep learning mainly perform supervised learning through the use of projections (e.g. principal component analysis) or changes of variables (e.g. kernel classifiers), feedforward neural networks allow to automatically construct the latter techniques through a stack of successive layers whose parameters are learnt as data is fed into the network. These networks are said to be deep when the number of layers is increased. They can be used for classification and regression tasks. Convolutional neural networks (CNN) in particular perform very well in most visual recognition tasks such as image or speech multiclass and multilabel classification (LECUN and BENGIO, 1995).

In CNN, a convolutional stage is defined as a filter layer (convolution), a dimension reduction layer (feature pooling) and a non-linear layer (non-linearity function). A CNN is a stack of convolutional stages. Once the new features are constructed through this stack, a classical multilayer perceptron with linear layers (inner products) classifies them. An activation layer eventually outputs the corresponding probabilities.

In the present context of supervised learning, backpropagation computes the gradients for each weight while gradient descent allows to optimize the weights of each layer given the architecture and the desired outputs corresponding to each training example. The error to minimize between the desired label and the actual output of the network is calculated using the cross-entropy loss L:

$$L(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} H(y_n, \hat{y}_n) = -\frac{1}{N}\sum_{n=1}^{N} [y_n \log \hat{y}_n + (1 - y_n)\log(1 - \hat{y}_n)] \tag{1.1}$$

with H the entropy measuring the similarity between y_n the label ground truth and $\hat{y}_n$ the prediction, averaged along each of the N observations.

The techniques used for training the models include in particular (LECUN, BOTTOU, et al., 2012):

- regularization (through the use of dropout (SRIVASTAVA et al., 2014), weight decay, momentum, or weight sharing between layers),
- rectified linear units (ReLUs) for the non-linear units,
- stochastic gradient descent for minimizing the loss function,—it is a stochastic approximation of the more general gradient descent optimization method, as batch learning is computationally not possible given the size of the dataset,

- Xavier's initialization of weights (GLOROT and BENGIO, 2010) which randomly draw initial weights from a distribution with zero mean and a specific variance depending on the network's architecture—it is particularly important to make sure the initial weights are properly chosen, since long training times often did not allow us to iterate on several weight initializations, especially for mixed sounds.

1.3.2 Deep learning tools

Models were trained using the deep learning framework Caffe (JIA et al., 2014). Caffe is an open-source well-maintained C++-based software that supports OpenCL and CUDA, thus exploiting parallel computing on Nvidia graphics processing units (GPU). It is originally developed by the Berkeley Vision and Learning Center and is now supported by an active community (3800+ commits and 200+ contributors on `github.com` as of November 2016). In Caffe, the solver and the network architecture are defined separately as Google protobuf models (*.prototxt). They can be defined, automated and controlled via command line or a Python interface, which makes it easier to integrate with other machine learning libraries, such as SciKit Learn for Python.

The solver protobuf defines key parameters for the training and testing phases (type of gradient descent, learning rate policy, regularization parameters, total number of iterations) and the testing phase (test intervals during the training).

The network protobuf defines the underlying architecture of the neural networks, as a series of layers (`Convolution`, `Pooling`, `ReLU`, `InnerProduct`, etc) with their respective hyperparameters. The training process then consists in a back and forth flow along the obtained graph. Blobs are arrays for communicating information from the data to the output loss, storing data, derivatives and parameters at all time. At each moment of the training, all hyperparameters, weights and state of the solver can be saved as a HDF5 or LMDB file. The corresponding snapshots can then be resumed for later use for training or production.

Training was achieved on the research group servers (one with 6 Gb of GPU RAM on an Nvidia GeForce GTX 980 Ti card and 12 Intel Core i7 powering the CPU at 3.50 GHz; another one with 12 Gb of GPU RAM on an Nvidia GeForce GTX Titan X).

1.4 Literature on machine hearing and multitask learning

We now examine past literature on sound identification and localization with deep learning techniques, as well as multitask learning for CNN.

1.4.1 Sound identification

PICZAK (PICZAK, 2015a) points out convolutional neural networks as a viable solution to environmental sound classification tasks achieving an accuracy of 73.1% on a public urban sounds dataset with 10 sound classes. It evaluates a network with two convolutional layers followed by a two-layer perceptron on three public standardized datasets of short environmental recordings (ESC-50 (PICZAK, 2015b), ESC-10 (PICZAK, 2015b) and UrbanSound8K (SALAMON et al., 2014)). The model outperforms state-of-the-art techniques of machine learning, even on small datasets after data augmentation. The features used for the audio data are mainly segmented spectrograms.

SONG and CAI (SONG and CAI, n.d.) see convolutional neural networks as a promising alternative to Gaussian mixtures and Hidden Markov models for automatic speech recognition. Using four convolutional layers with ReLUs and pooling, two inner products with ReLUs and a softmax layer, they successfully handle phone recognition with a test error of 22.1% on the TIMIT dataset, which closely matches the state-of-the-art. The features they used are mel-log filter banks.

1.4.2 Sound localization

Bio-inspired methods in machine hearing mostly use binaural localization (MA et al., 2015). The human auditory system indeed relies on two main cues to determine the azimuth of a sound source: interaural time differences (ITD, lag in cross-correlation function between the left and right ears) and interaural level differences (ILD, energy ratio between the left and right ears).

WOODRUFF and WANG (WOODRUFF and WANG, 2012; WOODRUFF and WANG, 2010) propose a binaural model based on ILD and ITD to localize mutliple sources with prior knowledge of the number of recorded cues. The method uses Gaussian mixture models and is said to be computationally complex, thus limiting the number of simultaneous sources.

MA et al. (MA et al., 2015) achieve robust localization of multiple sources in reverberant conditions, using only the cross-correlations and ILD features fed into a regular deep neural networks with 8 hidden layers (inner products). In particular, in this study, ILD are shown to significantly improve the front-back confusion resulting from the similarity of ITD and ILD in the front and rear hemifields.

1.4.3 Multitask learning for convolutional neural networks

There are already existing methods for transfer and multitask learning for optimizing a single network for several tasks while preserving performance (LI and HOIEM, 2016).

Feature extraction uses the activations of a layer as extracted features to be fed into a regular classifier, such as support vector machine or even logistic regression.

Fine-tuning allows to modify the parameters of an existing CNN to train a new task. Only the earlier layers are used with the existing optimized weights, while the output layers are optimized for the new tasks.

Multitask learning *per se* consists in optimizing all tasks in parallel with the same anterior layers but different output layers. The ratio between the shared and task-specific parts of the network can be adapted. The network may be pre-trained on a single task before the global optimization.

2 Methods

2.1 Feature extraction and data preparation

After we extract features from the data described in section 1.2, it must be labeled, balanced and split.

2.1.1 Data labeling

Supervised machine learning requires labeled data to train the models on. Sounds from the NIGENS database come already labeled, so there is no extra work to do here. However labels must be arranged in a readable and adapted format.

For each sound, the class is stored both as a scalar from 0 to 11, as well as an array of size 12 containing only zeros and a one for the active class. The corresponding localization is also stored as scalars (in radians and degrees) and as an array of size 72. Both the scalar and the vector representations were used in the training and testing process of clean sounds.

Mixed sounds were labeled using a technique for exploiting multitask learning and combining both information as shown in figure 2.1. Localization vectors are presented horizontally for each row. A void class bin completes the information, stating whether the class is active (0) or inactive (1) in the mixed sound. The resulting matrix is of size (13×73) (for a total of 949 parameters).

2.1.2 Feature description

The features to feed into the machine learning algorithm are ratemaps, amplitude modulation spectrogram based features and interaural level differences. They all are computed using the Two!Ears auditory front-end (SCHYMURA et al., n.d.) and all have two channels for each ear signal, except for interaural level differences which only has one channel that sums up both signals information. We changed the feature computation parameters when switching from clean sounds to mixed sounds in order to speed up computation in the system and reuse the features for

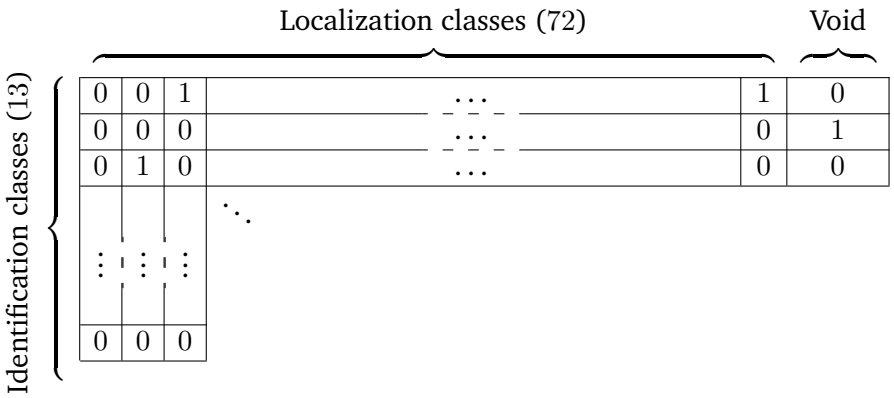


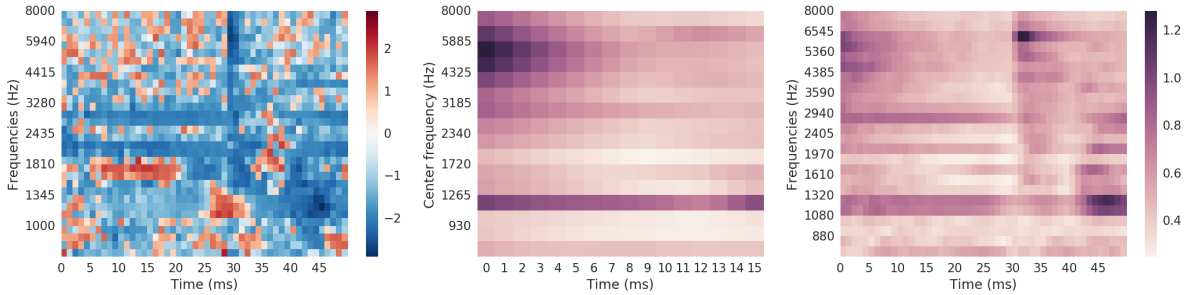
Fig. 2.1: Labeling methods for mixed sounds combining localization and identification

other purposes. This explains why the obtained features may not have the same size for clean and mixed sounds.

Ratemaps. Log-scaled spectrograms are extracted from all recordings (truncated in windows of 0.5 s) with Hann-window step size of 32 ms along 63 frequency channels between 80 Hz and 8 kHz for clean sounds, and with Hann-windows step size of 21 ms along 50 frequency channels for mixed sounds. Ratemaps can thus be stored as an array of size (16×63) for clean sounds and size (24×50) for mixed sounds. In the context of bio-inspired methods, ratemaps accurately picture the neural activity in that they represent a map of auditory nerve firing rates (SCHYMURA et al., n.d.).

Amplitude modulation spectrograms based features (AMS). AMS contain several channels for each ear with information of both center frequencies and modulation frequencies for each time frame, which mimics the auditory cortex of mammals (LANGNER, 1992). The detection of envelope fluctuations is indeed a central aptitude of the human auditory system (MORITZ et al., 2011). They are obtained after both a short-term frequency analysis and a modulation spectrogram (SCHYMURA et al., n.d.). AMS can be stored as arrays of size $(18 \times 16 \times 63)$ for clean sounds and of size $(16 \times 16 \times 16)$ for mixed sounds.

Interaural level differences (ILD). ILD are a standard feature in machine hearing. It relies on the fact that a sound emitted on the right side of the head will have a higher level at the right ear than at the left ear. ILD are estimated for each individual frequency channel by comparing the frame-based energy of the left and right-ear envelope (SCHYMURA et al., n.d.). ILD can be stored as arrays of size (32×50) for mixed sounds. ILD were not computed for clean sounds.



(a) ILD feature **(b)** Left-channel first AMS **(c)** Left-channel ratemap

Fig. 2.2: Features representation for a "phone" recording (ILD, AMS and ratemap)

2.1.3 Balancing

All types of classes are originally strongly imbalanced for both types of sounds. That means some classes are outnumbered by other classes. That could possibly lead to a bias towards the majority classes, treating minority classes as outliers or even not considering rare events at all. This phenomenon applies to identifications classes (class count in figure 2.3a), localization classes (azimuths count in figure 2.3c) and number of sources (counted for one specific class in figure 2.3b).

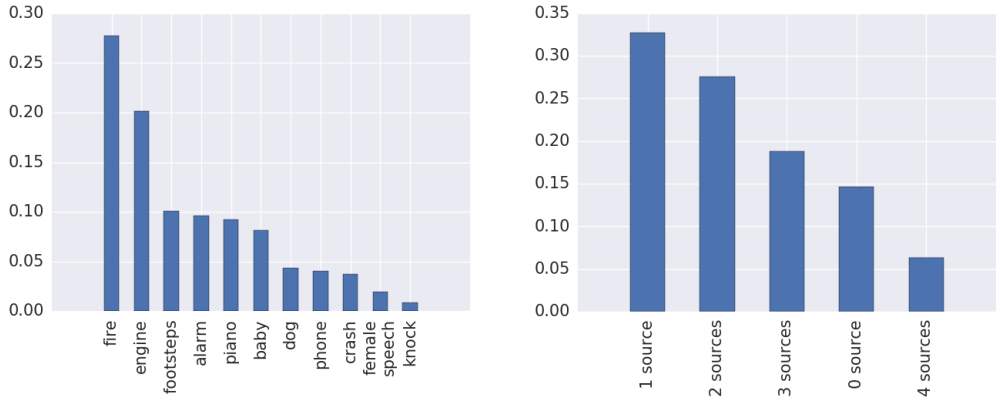
Balancing is performed by counting each class and fixing a target as the exact class count we want to reach for each class after balancing. Minority classes are artificially oversampled by repeating elements from the class; majority classes are subsampled.

Although balancing was performed for identification classes, it has not been solved on mixed sounds in the azimuth plane. Balancing one class type (*e.g.* azimuths) would indeed tend to imbalance another one (*e.g.* number of sources). That is why azimuth imbalance was not tackled for mixed sounds.

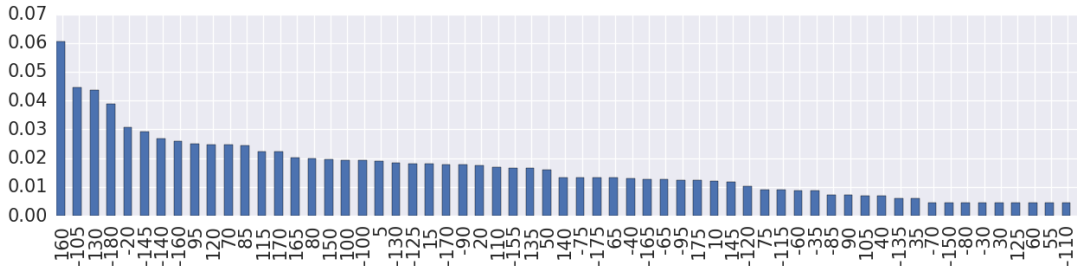
2.1.4 Train-test split

After balancing, clean sounds count $9{,}000{,}000$ data points and mixed sounds count $17{,}227{,}800$ data points. They are first randomly shuffled in order to not bias the gradient with entire batches of highly correlated examples, which could lead to a poor convergence (see the gradient descent algorithm in section 1.3). Data points are then split into training and testing sets in the proportion 67%-33%.

Training and testing datasets are saved under separate HDF5 files (*.h5). We chose HDF5 format over LMDB for its simplicity in read/write actions and its portability in Python through the `h5py` library.



(a) Identification classes for clean sounds **(b)** Co-occurent sources for mixed sounds



(c) Distribution of azimuths for the class "alarm" of the mixed sounds (sorted according to frequencies of localization classes)

Fig. 2.3: Percentage ratio of several classes for clean and mixed sounds (while some classes stand out, others are outnumbered)

2.2 Evaluation metrics

Machine learning model evaluation requires to work with appropriate evaluation metrics. We here present the metrics that we used for picking the best models.

2.2.1 Learning curve

As described in section 1.3.1, train and test losses (also called cost errors) are direct outputs of a neural network, so they do not require extra calculations. They are evaluated between the expected ground truths and the obtained labels either on the train set or on the test set with the chosen loss. Plotting the train and test losses as a function of the number of learning iterations is called visualizing the learning curve.

The train loss characterizes the network's learning progression. A flat learning error represents a slow-learning network. When the train loss reaches a plateau, one can say the network is no longer learning. The test loss shows the network's generalization capacity on unseen data.

The test-train difference between the train loss and the test loss characterizes the bias-variance trade-off (GEMAN et al., 1992). A high bias corresponds to a high training error and a small test-train difference, whereas a high variance is characterized by a huge gap between the test and the train losses (*i.e.* a big test-train difference). The test-train difference can be qualitatively estimated on the learning curve, or quantitatively expressed as a relative difference:

$$\text{Test-train difference} = \min_{\mathbf{w}} \frac{|L_{\text{train}}(\mathbf{w}) - L_{\text{test}}(\mathbf{w})|}{\max\left(L_{\text{train}}(\mathbf{w}), L_{\text{test}}(\mathbf{w})\right)} \tag{2.1}$$

2.2.2 Confusion matrix

A confusion matrix allows to visualize the performance of an algorithm by counting the actual labels *versus* the predicted labels for each given class, as presented in figure 2.4. From that, we can define sensitivity (also called recall, or true positive

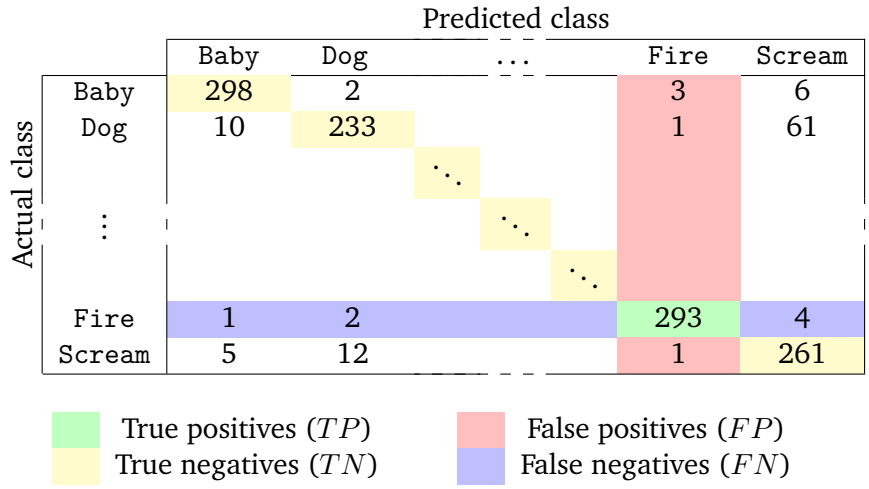


Fig. 2.4: Example of confusion matrix with TP, TN, FP and FN for the class "Fire"

rate), specificity (true negative rate) and precision (positive predictive value), such that:

$$\begin{cases} \text{Sensitivity} = \text{Recall} = \dfrac{TP}{TP + FN} & \text{(true positive rate)} \\ \text{Specificity} = \dfrac{TN}{TN + FP} & \text{(true negative rate)} \\ \text{Precision} = \dfrac{TP}{TP + FP} & \text{(positive predictive value)} \end{cases} \tag{2.2}$$

Sensitivity illustrates the classifier's benefits by counting well-classified cases. Specificity depicts the model's ability to classify other classes relatively to a given class. Sensitivity and specificity are inversely proportional, meaning that as the sensitivity increases, the specificity falls and *vice versa*. The precision of the classifier is the

percentage of sounds predicted for a given class who actually belong to this class. Precision and sensitivity are also inversely related. The trade-offs between sensitivity *versus* specificity and precision *versus* sensitivity have specific metrics explored in section 2.2.4.

2.2.3 Overall and balanced accuracy

Overall accuracy is defined for classifiers as the portion of correctly predicted labels. Accuracy makes sense when used with balanced datasets and softmax activation layers, in which case accuracy reflects the actual performance of the classifier:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.3}$$

Accuracy requires a properly balanced dataset (which works fine in the case of clean sounds). Imbalanced domains would indeed be biased by a trivial classifier that would always predict the majority classes and ignore rare events. Accuracy can therefore only be computed when dealing with clean sounds and using a softmax activation layer (see equation 2.5) which favors one class over all the others for both identification and localization. In the case of imbalanced datasets and/or multilabel classification, other metrics should be used to assess the classification performance, such as balanced accuracy or precision and recall.

Balanced accuracy relates the portion of correctly predicted labels to the number of actual positives and negatives, as in:

$$\text{Balanced accuracy} = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right) \tag{2.4}$$

Balanced accuracy can be viewed as the arithmetic mean of sensitivity and specificity (true positive and negative rates). It can be used with mixed sounds. For those classifiers, other metrics were also used such as the receiver operating characteristic and the precision-recall curves.

2.2.4 Trade-off between sensitivity, specificity and precision

For the mixed sounds, the classifier outputs a score for each identification and localization class, then converted to a probability through the sigmoid activation layer. A class is considered as active if the obtained probability goes beyond a given threshold. This threshold is arbitrarily 0.5 by default, but can be changed in order to improve the classifier's performance. Finding the most appropriate threshold can be achieved by plotting the two following curves that aim at finding trade-offs between sensitivity, specificity and precision.

Receiver operating characteristic (ROC). For each class, the ROC curve is defined by the FP rate and the sensitivity as x and y-axis for each probability threshold used by the classifier. ROC depicts the trade-off between false positive ($= 1 -$ specificity, i.e. the costs) and true positive ($=$ sensitivity, i.e. the benefits) rates.

Precision-recall (PR). For each class, the PR curve represents the precision and the recall (also called sensitivity) as x and y-axis for several probability thresholds. PR thus depicts the trade-off between the classifier's precision and sensitivity.

For both curves, the area under the curve (AUC) is considered an appropriate metrics of the corresponding trade-off. The AUC is calculated by using an average of a number of trapezoidal approximations on several chosen thresholds.

DAVIS and GOADRICH (DAVIS and GOADRICH, 2006) argue that when a problem suffers from class imbalance, using the PR AUC as evaluation metrics is better than the ROC AUC, so the PR AUC should be favored in case of ambiguity. Both were computed for mixed sounds.

2.3 Architecture selection

The architecture selection aims at picking the best models in terms of accuracy. This requires to select the:

- features that will be used (feature selection): ratemaps, AMS or ILD features, or any combination of them,
- network's depth for the convolutional layers and the inner products,
- hyperparameter for the layers (e.g. number of filters, size of filters and stride for convolutional layers) and the solver (e.g. learning rate and momentum used by the stochastic gradient descent).

2.3.1 Convolutional neural networks

As illustrated on the example of the figure 2.5, we focus on the class of neural networks of the following form:

Convolutional stages. We stack from one to three convolutional stages with their filter layers (hyperparameters: number of filters, size of filters, stride), non-linear layers (no hyperparameter for ReLUs) and dimension reduction layers using max pooling (hyperparameter: stride).

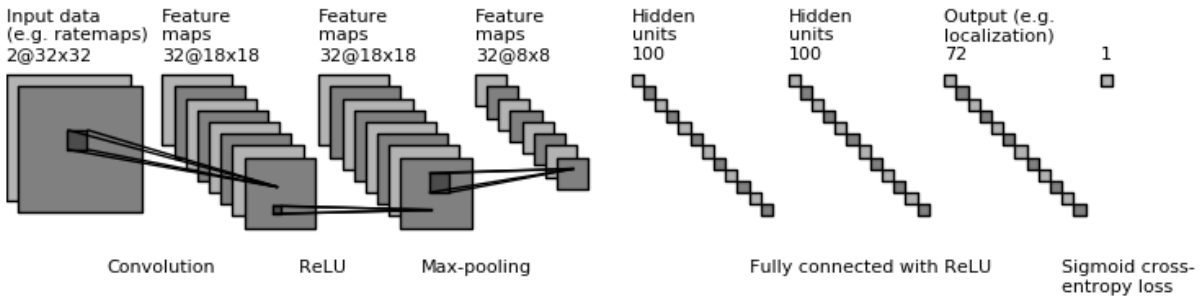


Fig. 2.5: Simple CNN architecture with one convolutional stage and two inner products

Fully connected layers. We stack from one to two inner products (hyperparameter: number of nodes) including dropout in the training phase.

Activation layers. They are sigmoid layers for the mixed sounds and softmax layers for the clean sounds. The softmax function allows to make a class stand out through the use of exponential coefficients,—thus achieving multiclass unilabel classification. The sigmoid function on the contrary only converts values from the last fully connected layer to related probabilities in the range $[0, 1]$. In mathematical terms, we have for $j = 1, \ldots, K$ (indices of the last inner product):

$$\begin{cases} \text{Sigmoid: } \sigma(\mathbf{z})_j = \dfrac{1}{1 + e^{-\mathbf{z}_j}} \\ \text{Softmax: } \sigma(\mathbf{z})_j = \dfrac{e^{\mathbf{z}_j}}{\sum_{k=1}^{K} e^{\mathbf{z}_k}} \end{cases} \tag{2.5}$$

2.3.2 Methods for hyperparameter optimization

Manual optimization. Going from coarse to fine on the hyperparameter ranges, we manually update the hyperparameters and check for the obtained accuracy. In addition to being relatively slow, this search is not optimal and only gives a sense of what may happen underneath. It requires an expert experience, as well as the knowledge of unwritten rules of thumbs.

Grid search consists of an exhaustive search on the whole hyperparameter space, each combination of hyperparameters being mapped to their respective accuracy. While this technique is particularly relevant for most machine learning algorithms, it is not well suited to neural networks where hyperparameters are numerous and lie in a highly dimensional space.

Bayesian optimization aims at efficiently navigating within the hyperparameter space. Spearmint (SNOEK et al., 2012) is an open-source Python library that can be used for that purpose. The Bayesian strategy treats the evaluation metrics as a

random function with a prior over it (pre-knowledge). As models are trained, the prior is updated to form the posterior distribution which is in turn used to find the most appropriate hyperparameters that determine the next model to train.

Random search consists of a random search on the hyperparameter space. It is proved more efficient in time and computing power than a regular exhaustive grid search, because not all hyperparameters are equally important to tune (BERGSTRA and BENGIO, 2012). We implemented a random search for Caffe and used it for models with clean and mixed sounds. For each architecture to investigate, the hyperparameters are randomly initialized, the network is trained for a given number of iterations and its performance is assessed through well-chosen evaluation metrics. When selecting models with random search, we proceed from coarse to fine in terms of depth and complexity of the tested architectures. In this manner, we can reuse the weights at each random iteration and need less learning iterations before convergence.

3 Results

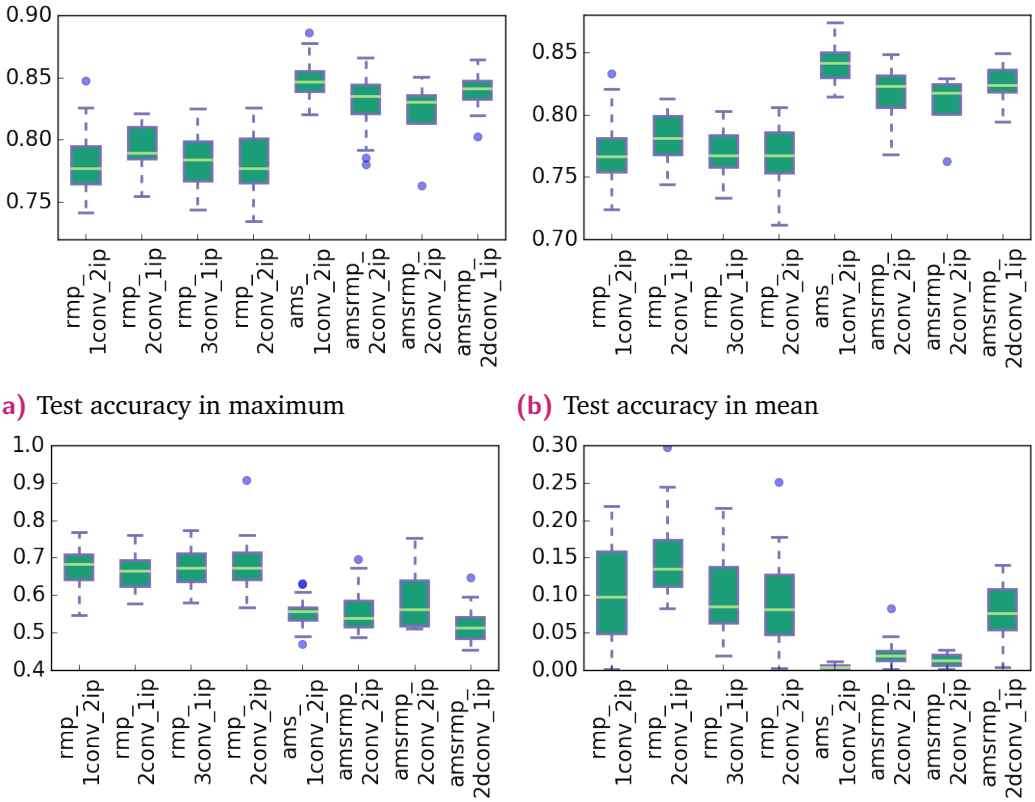
3.1 Architecture selection on clean sounds

For the architecture selection, we use random search as described in section 2.3.2. Several architectures can thus be tested and their relative performances compared. The metrics we used are the test accuracy (in mean and maximum), the test loss (in mean) and the test-train difference (in relative difference). For the architecture selection, we favored identification over localization as localization is proved easier to learn as seen in section 3.2.

The tested architectures are presented on table 3.1 and compared on figures 3.1a to 3.1d.

`rmp_1conv_2ip`	Using only ratemaps, one convolutional stage followed by two inner products
`rmp_2conv_1ip`	Using only ratemaps, two convolutional stages followed by one inner product
`rmp_2conv_2ip`	Using only ratemaps, two convolutional stages followed by two inner products
`rmp_3conv_1ip`	Using only ratemaps, three convolutional stages followed by one inner product
`ams_1conv_2ip`	Using only AMS, one convolutional stage followed by two inner products
`amsrmp_2conv_2ip`	Using all available features (AMS and ratemaps), two convolutional stages followed by two inner products
`amsrmp_2conv_2ip`	Using AMS and ratemaps, two convolutional stages followed by two inner products
`amsrmp_2dconv_1ip`	Using AMS and ratemaps, two parallel convolutional stages for both features followed by a concatenation stage and one single inner product; the models learn different weights for the two convolutions; the parameters of the convolution however are the same, so that they produce outputs of the same size that can be concatenated

Tab. 3.1: Tested architectures for clean sounds



(a) Test accuracy in maximum

(b) Test accuracy in mean

(c) Test loss in mean on final plateau

(d) Test-train difference

Fig. 3.1: Architecture selection results for clean sounds using several features and depths

The architecture using only AMS with one convolutional stage and two inner products (`ams_1conv_2ip`) clearly stands out for all metrics with an accuracy mean of 87.4% for identification.

3.2 Multitask learning on clean sounds

We here intend to compare the learning speed and final performance of different multitask learning techniques, based on the same best model of section 3.1. The tested architectures are presented on table 3.2 and compared on figures 3.2a to 3.2f.

All experiments were launched with the same training and testing parameters, so they can be properly compared. We took a moving average of the training loss on the last 30 iterations, in order to smooth the learning curve and extract the global pattern.

First of all, the convergence speeds to the desired plateau accuracies in 3.2a and 3.2b confirm that identification is easier to learn for the network than localization. We also notice that fine-tuning for multitask learning beginning with identification (i.e.

`Id, loc`	Sequential multitask learning starting with identification before learning both tasks
`Loc, id`	Sequential multitask learning starting with localization before learning both tasks
`Joint`	Joint multitask learning with all branches in common; only the output layers are task-specific; both losses have the same weight and are simultaneously optimized
`Joint, sep`	Joint multitask learning with the last layers optimized simultaneously, but separately (i.e. with different learnt weights)

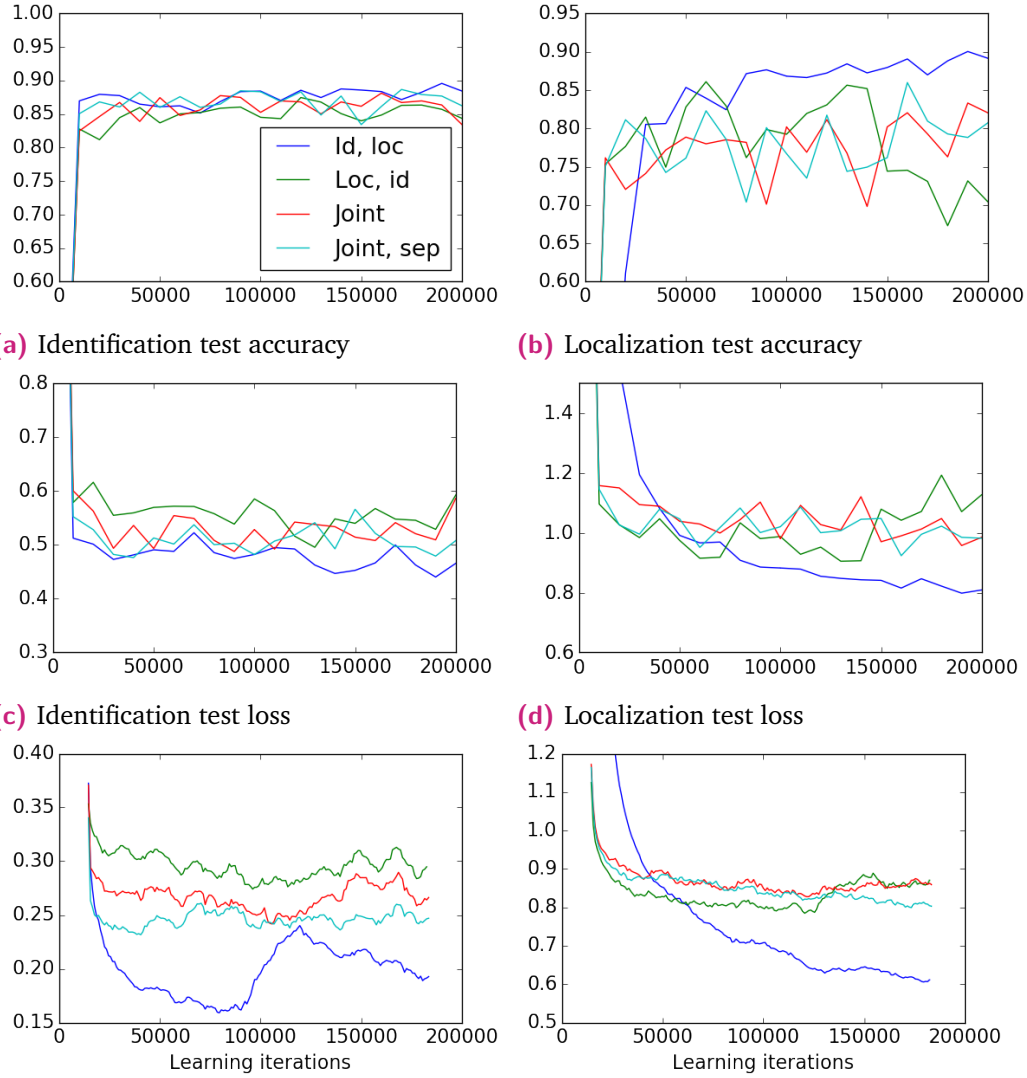
Tab. 3.2: Tested architectures for multitask learning

the "easiest" task in terms of learning) seems more efficient, in that its test accuracy (the blue line in 3.2a and 3.2b) converges much faster to the optimal accuracy for the desired tasks. Using fine-tuning would also allow to prioritize one task over another, in particularly by decreasing the learning rate or increasing the momentum after having trained one task. The second task to be trained will take some time to adjust as seen in 3.2e and 3.2f. We finally achieve best plateau accuracies (best mean on the last 20% plateau of the accuracy function) of 88.8% for identification and 88.7% for localization.

Secondly, for both identification and localization, there is a difference in the learning speed between the joint learning and the joint learning on separate branches as seen on 3.2e and 3.2f. In mean, the training loss using full joint learning is 9% higher than the training loss using separate branches for identification, and 5% higher for localization. This means that the last layers of a CNN would construct features that are less transferable than early layers. As YOSINSKI et al. (YOSINSKI et al., 2014) also argue, this supports the fact that first-layer features are general while last-layer features are more specific to the trained task. However, in our case, further experiments should be led with various architectures, in order to draw more general conclusions. In particularly, this phenomenon should be proved to be independent from the weight and bias initializations, which are currently set to random in all our experiments, as customary in deep learning. In that matter, averaging results on several learning and test phases can lead to more robust results.

3.3 Architecture selection on mixed sounds

As for mixed sounds, we also use random search as described in section 2.3.2. We did not rely on the architectures we found for clean sounds, as the data, the features and the complexity of the problem are all very different. The several models are



(a) Identification test accuracy

(b) Localization test accuracy

(c) Identification test loss

(d) Localization test loss

(e) Averaged identification train loss

(f) Averaged localization train loss

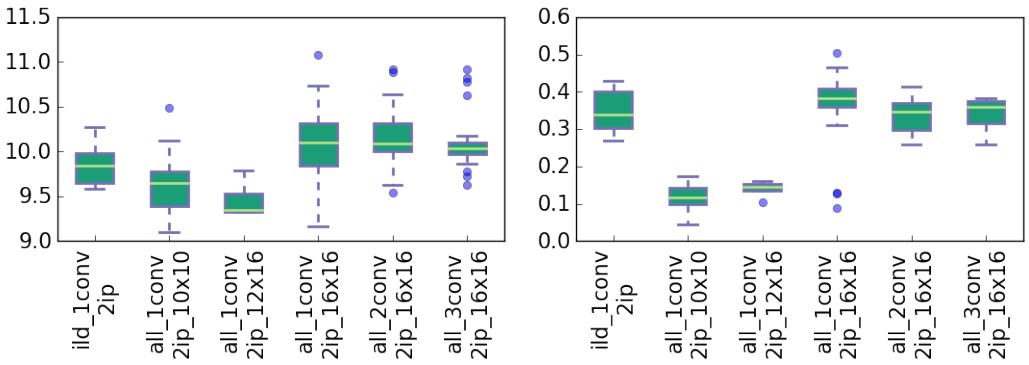
Fig. 3.2: Learning curves with different multitask learning techniques for both tasks (identification on the left, localization on the right)

compared by their test losses (in mean) and test-train loss differences (in relative difference), as other metrics require more expensive computations which are not compatible with random search.

The tested architectures are presented on table 3.3 and compared on figures 3.3a and 3.3b. We first tested each feature taken individually (results using only ratemaps and only AMS are relatively too poor to appear on the figures), then all features with architectures of increasing depths. As features have different sizes, the first convolutions are arranged to output features of same sizes that can then be processed in the rest of the network. For this purpose, we exploit the relationship in

`ild_1conv2ip`	Using ILD features only, one convolutional stage followed by two inner products
`all_1conv2ip_10x10`	Using all available features (AMS, ILD and ratemaps), one convolutional stage followed by two inner products; the first convolutions are adjusted so that it produces outputs of the same size 10×10 for each feature, which can then be concatenated and fed into the inner products
`all_1conv2ip_12x16`	With all features, one convolutional stage followed by two inner products; the first convolution outputs features of size 12×16
`all_1conv2ip_16x16`	With all features, one convolutional stage followed by two inner products; the first convolutions output features of size 16×16
`all_2conv2ip_16x16`	With all features, two convolutional stages followed by two inner products; the first convolution is as described above
`all_3conv2ip_16x16`	With all features, three convolutional stages followed by two inner products; the first convolution is as described above

Tab. 3.3: Tested architectures for mixed sounds



(a) Test loss in mean on final plateau **(b)** Test-train difference

Fig. 3.3: Architecture selection results for mixed sounds using several features and depths

a convolution between the output size W_{ouput}, the input size W_{input}, the size F of the convolution filter, the stride S and the padding P with which they are applied:

$$W_{\text{output}} = \frac{W_{\text{input}} - F + 2P}{S + 1} \tag{3.1}$$

In this case, the test-train difference makes less sense because random search does not allow to train long enough to make training and test losses comparable. We retain three architectures with the three best test losses (`all_1conv2ip_16x16`, `all_2conv2ip_16x16` and `all_3conv2ip_16x16`).

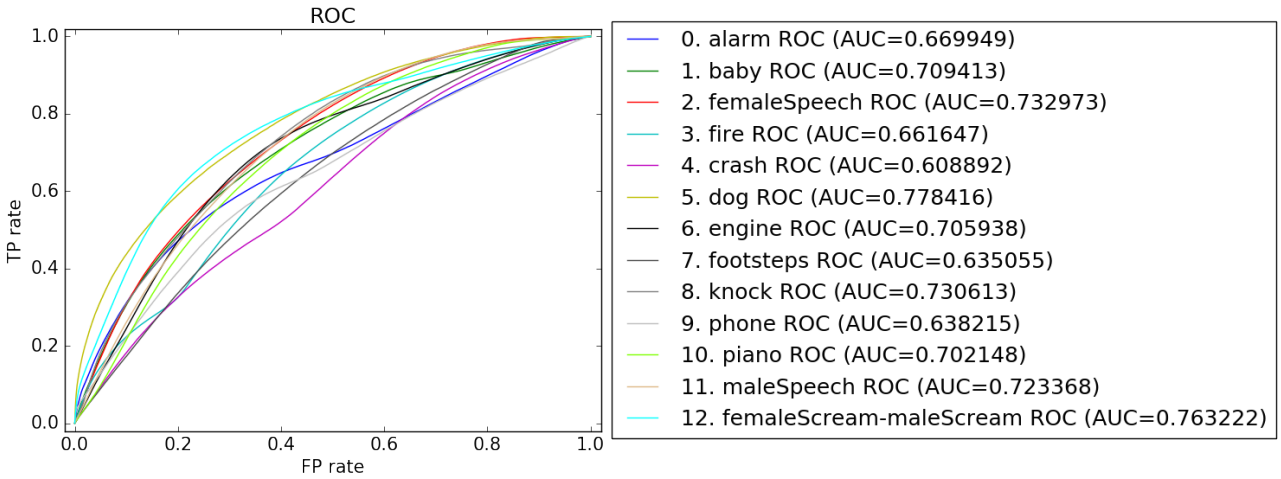


Fig. 3.4: Area under the curve of the receiving operating curve (AUC ROC) for identification classes

3.4 Multitask learning performances on mixed sounds

The best performing model from section 3.3 is picked to be assessed using the specific metrics of section 2.2 (Evaluation metrics).

The best-identified classes by increasing order of ROC AUC are as shown on figure 3.4: dog (0.778 ROC AUC), female and male screams (0.763), female speech (0.733), knock (0.731), male speech (0.723), baby (0.709), engine (0.706), piano (0.702), alarm (0.67), fire (0.662), phone (0.638), footsteps (0.635) and crash (0.609).

The maximal balanced accuracy calculated at the best classifier threshold also gives similar results as shown on figure 3.5: female and male screams (70.95% balanced accuracy), dog (69.99%), knock (67.13%), engine (67.1%), male speech (66.71%), female speech (66.59%), baby (65.7%), piano (65.36%), alarm (63.88%), fire (62.4%), phone (61.5%), footsteps (59.93%) and crash (57.56%). With both metrics, screams are the best-identified class.

As for localization, balanced accuracy clearly indicates that the agent only achieves a near-chance performance, averaging at 53.15% balanced accuracy, as shown in figure 3.6a. However the best-localized classes are: -160.0° (60.05% balanced accuracy), -150.0° (59.23%), -100.0° (57.93%), -65.0° (56.66%), -90.0° (56.29%), -105.0° (55.63%), -180.0° (55.39%), -170.0° (55.23%), -140.0° (54.9%) and -130.0° (54.07%). However, figure 3.6b implies there is a correlation between the frequency of apparition of the azimuth in the training set and the obtained balanced accuracy

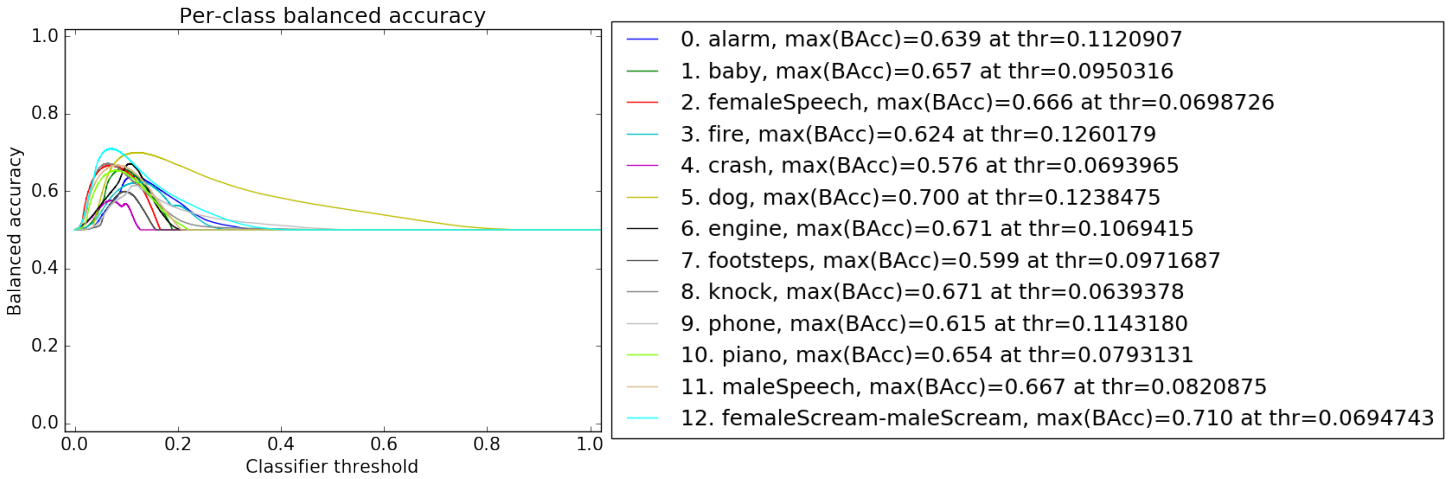
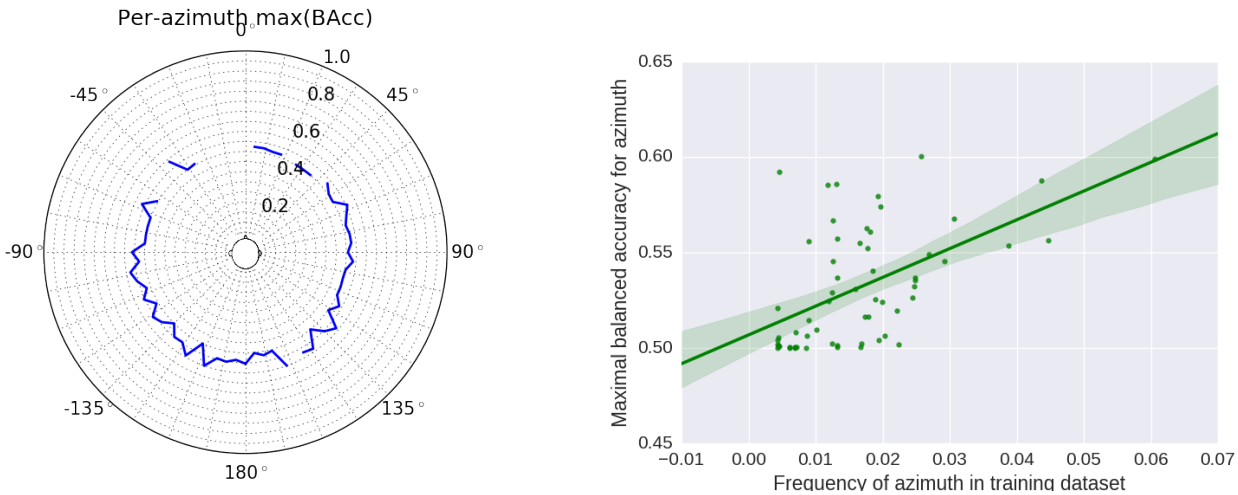


Fig. 3.5: Identification classes balanced accuracy calculated for each classifier threshold

for the corresponding azimuth. The dataset imbalance in terms of localization classes (as seen on figure 2.3c of section 2.1.3) thus seems to penalize outnumbered classes. Sufficient amounts of data is indeed essential for CNN to perform well and that can explain the gap between some localization classes.



(a) Maximal balanced accuracy at best classifier threshold for localization classes

(b) Correlation between the dataset imbalance and the obtained balanced accuracy for each azimuth (regression line and 95% confidence interval)

Fig. 3.6: Results for localization of mixed sounds: balanced accuracy for each azimuth and correlation with localization class percentage ratio

4 Conclusion

In the present thesis, we were to apply canonical methods of machine and deep learning (such as feature engineering, metrics selection and hyperparameter optimization) to simultaneously localize and classify environmental sounds. After extracting adapted features from regular audio recordings, proper training and testing datasets were prepared for both clean and mixed sounds. Specific evaluation metrics were picked to tackle a multiclass multilabel classification problem. The machine-hearing agent is based on convolutional neural networks whose architectures were iteratively determined through random search.

The real-scenario's complexity (with a variable number of sounds from more identification classes at different azimuths), as well as the presence of reverberation and the strong dataset imbalance make the learning process much more difficult than in the case of clean sounds where we achieve very good levels of accuracy for both identification (88.8% accuracy) and localization (88.7% accuracy). For mixed sounds, localization turned out a much harder task (53.2% balanced accuracy on average) while much better results were achieved for identification (65.0% balanced accuracy on average).

Further investigations can be done in the future, especially to achieve better results when dealing with mixed sounds. One possibility would be to use deeper networks or longer training time with lowered learning rates. The fact that some localization classes clearly stand out in terms of balanced accuracy indicate better results can also be achieved for other azimuths. Dataset imbalance for mixed sounds was proven to play a role in the poorer results for the localization of some azimuths, so gathering more data for the outnumbered localization classes could help reach better results. Another possibility for achieving better results is to make use of more regularization. An idea for regularization is to convolve in the channel's direction for AMS and ILD features before convolving on each spectrogram. One could also explore feature transferability by expanding the experiments from section 3.2. In particularly, testing several architectures with multiple random initializations could lead to broader conclusions on layers roles in function of their depths.

Bibliography

BERGSTRA, James and Yoshua BENGIO (2012). „Random search for hyper-parameter optimization“. In: *Journal of Machine Learning Research* 13.Feb, pp. 281–305 (cit. on p. 15).

COOKE, Martin, Jon BARKER, Stuart CUNNINGHAM, and Xu SHAO (2006). „An audio-visual corpus for speech perception and automatic speech recognition“. In: *The Journal of the Acoustical Society of America* 120.5, pp. 2421–2424 (cit. on p. 2).

DAVIS, Jesse and Mark GOADRICH (2006). „The relationship between Precision-Recall and ROC curves“. In: *Proceedings of the 23rd international conference on Machine learning*. ACM, pp. 233–240 (cit. on p. 13).

GAROFOLO, John, Lori LAMEL, William FISHER, et al. (1993). „TIMIT acoustic-phonetic continuous speech corpus“. In: *Linguistic data consortium, Philadelphia* 33 (cit. on p. 2).

GEMAN, Stuart, Elie BIENENSTOCK, and René DOURSAT (1992). „Neural networks and the bias/variance dilemma“. In: *Neural computation* 4.1, pp. 1–58 (cit. on p. 11).

GLOROT, Xavier and Yoshua BENGIO (2010). „Understanding the difficulty of training deep feedforward neural networks.“ In: *Aistats*. Vol. 9, pp. 249–256 (cit. on p. 4).

HAYKIN, Simon (2004). „Neural Networks: A comprehensive foundation“. In: *Neural Networks* 2.2004 (cit. on p. 3).

JIA, Yangqing, Evan SHELHAMER, Jeff DONAHUE, et al. (2014). „Caffe: Convolutional architecture for fast feature embedding“. In: *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, pp. 675–678 (cit. on p. 4).

LANGNER, Gerald (1992). „Periodicity coding in the auditory system“. In: *Hearing research* 60.2, pp. 115–142 (cit. on p. 8).

LECUN, Yann and Yoshua BENGIO (1995). „Convolutional networks for images, speech, and time series“. In: *The handbook of brain theory and neural networks* 3361.10, p. 1995 (cit. on p. 3).

LECUN, Yann, Léon BOTTOU, Genevieve ORR, and Klaus-Robert MÜLLER (2012). „Efficient backprop“. In: *Neural networks: Tricks of the trade*. Springer, pp. 9–48 (cit. on p. 3).

LI, Zhizhong and Derek HOIEM (2016). „Learning without Forgetting“. In: *arXiv preprint arXiv:1606.09282* (cit. on p. 6).

MA, Ning, Guy BROWN, and Tobias MAY (2015). „Exploiting deep neural networks and head movements for binaural localisation of multiple speakers in reverberant conditions“. In: *INTERSPEECH 2015*, pp. 3302–3306 (cit. on pp. 5, 6).

MORITZ, Niko, Jörn ANEMÜLLER, and Birger KOLLMEIER (2011). „Amplitude modulation spectrogram based features for robust speech recognition in noisy and reverberant environments“. In: *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 5492–5495 (cit. on p. 8).

PICZAK, Karol (2015a). „Environmental sound classification with convolutional neural networks“. In: *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, pp. 1–6 (cit. on p. 5).

– (2015b). „ESC: Dataset for environmental sound classification“. In: *Proceedings of the 23rd ACM international conference on Multimedia*. ACM, pp. 1015–1018 (cit. on p. 5).

SALAMON, Justin, Christopher JACOBY, and Juan Pablo BELLO (2014). „A dataset and taxonomy for urban sound research“. In: *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, pp. 1041–1044 (cit. on p. 5).

SCHYMURA, C, T WALTHER, H WIERSTORF, F WINTER, and S SPORS. „Two! Ears–Integral interactive model of auditory perception and experience“. In: (cit. on pp. 7, 8).

SNOEK, Jasper, Hugo LAROCHELLE, and Ryan ADAMS (2012). „Practical bayesian optimization of machine learning algorithms“. In: *Advances in neural information processing systems*, pp. 2951–2959 (cit. on p. 14).

SONG, William and Jim CAI. „End-to-End Deep Neural Network for Automatic Speech Recognition“. In: (cit. on p. 5).

SRIVASTAVA, Nitish, Geoffrey HINTON, Alex KRIZHEVSKY, Ilya SUTSKEVER, and Ruslan SALAKHUTDINOV (2014). „Dropout: a simple way to prevent neural networks from overfitting.“ In: *Journal of Machine Learning Research* 15.1, pp. 1929–1958 (cit. on p. 3).

WOODRUFF, John and DeLiang WANG (2010). „Sequential organization of speech in reverberant environments by integrating monaural grouping and binaural localization“. In: *IEEE transactions on audio, speech, and language processing* 18.7, pp. 1856–1866 (cit. on p. 5).

– (2012). „Binaural localization of multiple sources in reverberant and noisy environments“. In: *IEEE Transactions on Audio, Speech, and Language Processing* 20.5, pp. 1503–1512 (cit. on p. 5).

YOSINSKI, Jason, Jeff CLUNE, Yoshua BENGIO, and Hod LIPSON (2014). „How transferable are features in deep neural networks?“ In: *Advances in neural information processing systems*, pp. 3320–3328 (cit. on p. 19).

List of Figures and Tables

Declaration

Statutory declaration. I declare that I have authored this thesis independently, that I have not used other than the declared sources & resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Eidesstattliche Erklärung. Hierbei erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne unerlaubte fremde Hilfe angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Berlin, August 26, 2016

Pierre MARCENAC