Audio impaired driving assistent

1st Marco Colussi
Università degli studi di milano
Dipartimento di informatica
Milan, Italy
marco.colussi1@studenti.unimi.it

***Abstract*—This document is written to provide an efficient algorithm able to record audio patterns from the surrounding environment of a car, analyze them and recognize sounds like car horns and siren, giving then feedback to the driver.**

The aim is to create a software able to help the drivers to be aware of what's happening around them. To achieve this we tested different machine learning algorithms as KNN, SVM and CNN, and different audio features, as pitch, MFCCs, DELTAS and mel-spectrogram.

***Index Terms*—audio, pattern, recognition, driving, machine learning, neural network, audio feature**

I. Introduction

8% of the American population is affected with hearing impairments, which goes up to 30% when age is over 65 [1]. Audio impaired drivers have an excellent focus and for the less distraction that earing can provide, but in some circumstances, it's necessary to understand what's happening around the car just by hearing it.

The main problem is that is hard to recognize emergency situations without being able to hear clearly. Considering sirens is essential to recognize the arrival of an emergency vehicle on time, and to be able to create a safer condition for the overtaking. Is also important to recognize car horns to understand risky situations or different problem we are not aware of while driving.

With this in mind, we wanted to create a system able to give support and substantial help to the deaf drivers. As a potential application, it could be installed in a car, prepared with microphones and a running system-on-a-chip, able to support the necessary load work, with an output system used to give visual/tactile feedback to the driver.

II. System Overview

A. Audio pre-processing

Before talking about algorithms and results, we need to introduce some theoretical aspects of the audio samples. The main characteristics of audio samples that we are interested in are:

- The number of channels
- The sampling frequency
- The duration

Before giving the audio samples into our algorithms we have to make them as homogeneous as possible. The sampling frequency consists of the number of samples per second in an audio clip, the higher is the frequency the more information we have on about the sample, but growing it also increases the computational cost of elaboration of the sample.

We choose to use 44100 Hz because is the most used format and it has a great amount of information. The first thing to do is to ensure that all the samples in our dataset are consistent, so we need to resample them, if necessary, to have the same amount of information for every sound of the dataset. To do this we use the resample function of Matlab which resamples the audio passed at p/q times the original sample rate, where p and q are extracted with the rat function of 44100/SampleRate.

Secondly, we need to check that the number of channels of our sample is the same, otherwise, one sample could generate different amounts of information based on the number of channels. This will be done directly before extracting the audio feature.

The duration of the samples is not strictly required to be the same for every sample, except for the convolutional neural network, so we created three copy of the same dataset, one without considering the length of the audio samples, one creating only 1-second samples and one with samples long 4 seconds.

Next, the datasets have been divided randomly into two different sets, one for the training (training-set) using the 80% of the data, and one for the evaluation of the accuracy (test-set) using the 20%.

B. Data structures

To store the audio file we use the audioDatastore, Matlab datastores are used to create datastores that contain a large collection of data, giving us the possibility to label differently every sample. To store the enormous amount of data generated from the feature extraction process, we utilized the Tall Arrays [2].

Tall arrays are particularly useful in memory management, they remain unevaluated until you perform a specific operation on them, this makes it possible to work without concerning out-of-memory errors, making it scalable and enabling to work with data which is bigger than the actual memory.

C. Pitch, MFCCs and DELTAs

To create a working machine learning model, we need to understand which data we are going to pass to it. The main representation of an audio sample is his waveform, having on the y-axis the amplitude, and on the x-axis the time.

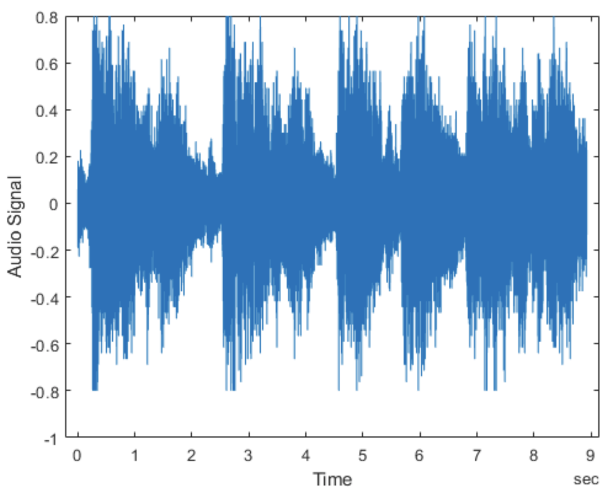


Fig. 1. Waveform example

This kind of data might not be the best for fitting a machine learning model, we have not enough information to differentiate two audio samples. It's important to choose the right feature for the model because the same feature could be perfect for one application but not working for another.

The common features used utilize different domains [3]:

- Time: Energy and Zero-crossing rate
- Frequency: spectral centroid, flux, spectrogram and MFCCs
- Wavelet: PWP

The most performing and spread seems to be the MFCCs (Mel-Frequency Capestral Coefficients). MFCCs are derived by the inverse of the Fourier transform of the logarithm of the signal spectrum. The particularity is that the mel-scale approximates the human system response equally spacing the frequency bands [4],

$$m = 2595 log_{10}(1 + \frac{f}{700})$$

.

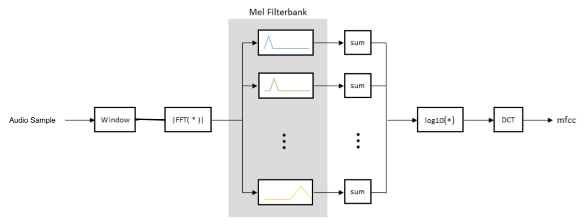


Fig. 2. MFCCs extraction schema

We will use the first 13 MFCCs because it has been seen that they carry adequate information in the classification tasks. The coefficients will be calculated on specific determined sized frames of the sample with a specific overlap time, that because it allows us to work on discrete time-steps. The Delta contains the information change between two consequent coefficients from one frame to another $Delta = currCoeffs - prevCoeffs$. The number of delta coefficients is the same as the MFCCs selected. Adding DELTAS to the features to utilize for the training, increased significantly the memory usage and the computational time without giving a notable increase in performance, this feature was therefore removed and not used during the tests.

Lastly, the pitch seems to be one other possible feature to consider in our application especially for the classification of car horns and syren which works in a particular range frequency with constant changes, to extract the pitch we used the Matlab function pitch, passing as parameters the audio sample, the sampling frequency the number of sample per frame to be considered and the overlap of the samples frame.

D. Mel segmented spectrogram

As pointed out in [5], the best way to train a convolutional neural network for sound event classification is using mel segmented spectrogram as an image that will feed the first layer of the CNN. The mel spectrogram consists of a frequency-domain representation, where on the x-axis we have the time, and on the y-axis, we have the Mel scale.

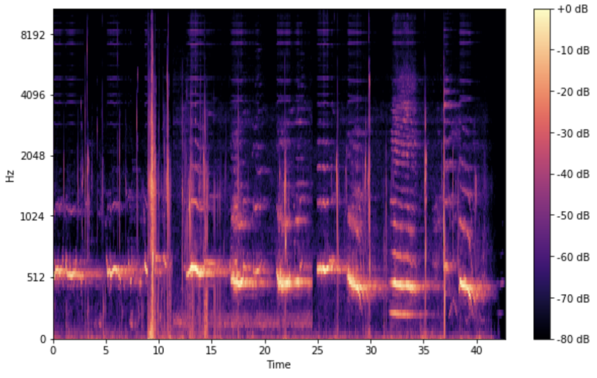


Fig. 3. Mel-spectrogram representation

The segmented part derives from the ability to create small-windowed images to be processed.

III. Models construction

Classifiers are supervised machine learning algorithms, which means that to be able to train a model we need to split our dataset into a training-set used to train the model, and a test-set used to validate and test the prediction of the model.

Another important characteristic to consider is the metric distance. It's important to select the correct metric, in particular for KNN and SVM, where the actual model is just an organization of distances, which is the reason why these are relatively expensive: we need to compute the distances with all the training-set elements.

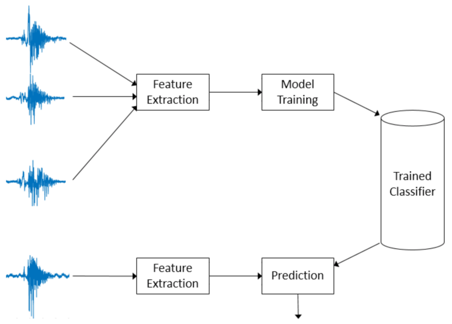


Fig. 4. Algorithms workflow diagram

A. *K-Nearest Neighbor*

A KNN it's used for classification and pattern recognition, it uses just the K ”closest” points to perform the classification, calculated from the distance between the properties of the element and ones of the neighbors, the classification is done calculating the majority of the classes present among the neighbors.

The hyperparameter used are:

- Distance
- K
- Kernel Function

Using the weighted knn we give more weight to points that are closer to our value and less weight to the points farther away.

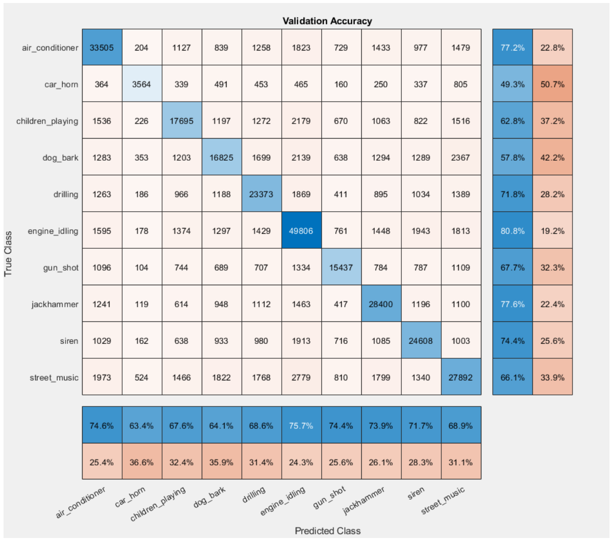
The model is trained using the fitcknn Matlab function, the training process took 1631 seconds (less than 30 minutes). After the training we need to cross-validate the model, to do so we used the k-Fold method using 10 as the parameter. This is used to estimate the skill of a machine learning model, this process is time-consuming, especially with large datasets because it needs to check every point of the elements in the collection.

The evaluation process this time took more than 33 hours, and it resulted in having a validation accuracy of $71,6\%$. More specific results in the following confusion matrix:

B. *Support Vector Machine*

The workflow for training an SVM model it's the same as for the KNN, the only difference lies in the model training process. SVMs are binary classifiers used to separate linearly points in a plane. When data it's not linearly separable we use the kernel trick which transforms the data in a higher-dimensional space where data is now linearly separable.

Applying a SVM to a multi-class problem it's possible, changing the approach from ”one-versus-one” where for each learner one class is positive, one negative and ignores the rest, to ”one-versus-all” where one class is positive and all the other are negative, and applying it one class after the other to classify. If it's positive we have found the class otherwise it takes another class as positive and the rest again negative. To do so we use the fitcecoc function which is designed to fit a multiclass model for SVMs.



	air_conditioner	car_horn	children_playing	dog_bark	drilling	engine_idling	gun_shot	jackhammer	siren	street_music		
air_conditioner	33505	204	1127	839	1258	1823	729	1433	977	1479	77.2%	22.8%
car_horn	364	3564	339	491	453	465	160	250	337	805	49.3%	50.7%
children_playing	1536	226	17695	1197	1272	2179	670	1063	822	1516	62.8%	37.2%
dog_bark	1283	353	1203	16825	1699	2139	638	1294	1289	2367	57.8%	42.2%
drilling	1263	186	966	1188	23373	1869	411	895	1034	1389	71.8%	28.2%
engine_idling	1595	178	1374	1297	1429	49806	761	1448	1943	1813	80.8%	19.2%
gun_shot	1096	104	744	689	707	1334	15437	784	787	1109	67.7%	32.3%
jackhammer	1241	119	614	948	1112	1463	417	28400	1196	1100	77.6%	22.4%
siren	1029	162	638	933	980	1913	716	1085	24608	1003	74.4%	25.6%
street_music	1973	524	1466	1822	1768	2779	810	1799	1340	27892	66.1%	33.9%
	74.6%	63.4%	67.6%	64.1%	68.6%	75.7%	74.4%	73.9%	71.7%	68.9%		
	25.4%	36.6%	32.4%	35.9%	31.4%	24.3%	25.6%	26.1%	28.3%	31.1%		

Fig. 5. Confusion Matrix of validation accuracy

The only parameter we had to change in this algorithm is the coding technique to ”onevsall”. The training of the Multiclass-SVM took 174318 seconds (more than 2 days) and the cross-validation took approximately 3 more days. The result this time is heavy, the overall accuracy of the model is of 12%. Due to the time consumption and the unpromising results we abandoned this method.

C. *Convolutional Neural Network*

A CNN it's a particular artificial network where the layers are feed-forwarded, inspired on the animal visual cortex their main application is in image recognition, classification, and analysis. They are designed to recognize specific patterns directly from the image with minimal processing, they are hardly affected by noise in the dataset due to distortions or geometric transformations of the images.

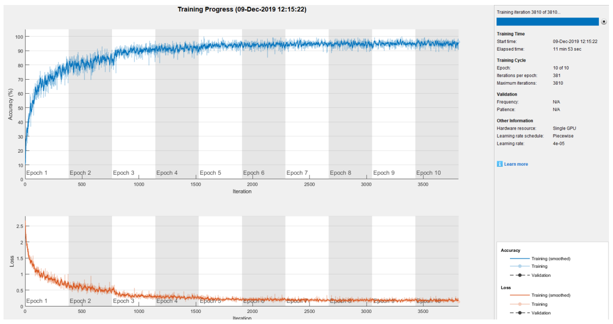
The network architecture and parameters are based on [6], and the more important layers are:

- Image Input layer: takes as input a 2D image
- Convolutional layer: applies sliding convolutional filters
- Max and Average Pooling: performs downsampling and compute the max or average values of each region
- Fully Connected layer: multiplies the input by a weight matrix and adds a bias vector
- Classification Output layer: compute the loss and evaluate the class

The hyperparameter used are:

- Learning Rate: initial learning rate of the net
- Momentum: contribution of the previous step
- MaxEpoch: number of epoch used during training
- Shuffle: when to shuffle the data
- LR Drop period: when to drop the Learning Rate
- LR Drop Factor: how much to drop the Learning Rate

We trained two different models on 2 different input groups. One with one-second length samples and one with four, the initial idea was that, based on the length of the pattern recognized, we could have classified it with one of the two different models. The training timing and accuracy where mostly the same, accuracy over 93% gained in 12-23 minutes.



Training iteration 3810 of 3810...

Training Time
Start time: 09-Dec-2019 12:15:22
Elapsed time: 11 min 53 sec

Training Cycle
Epoch: 10 of 10
Iterations per epoch: 381
Maximum iterations: 3810

Validation
Frequency: N/A
Patience: N/A

Other Information
Hardware resource: Single GPU
Learning rate schedule: Piecewise
Learning rate: 4e-05

Fig. 6. Visual log of the training process

After re-training the models on the extended dataset, the accuracy on the four-second net reached an unpredictable 99.3%, making the use of two distinct models pointless, the accuracy is high enough for every class that we are considering, even when the sounds as car horn occupy a small portion of the full-length sample.

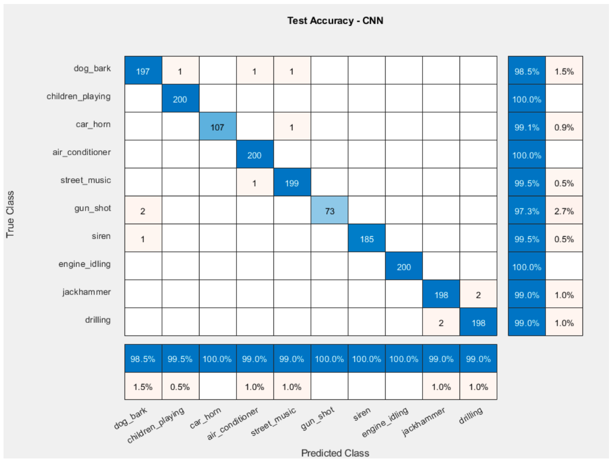


Fig. 7. Confusion matrix of 4 second extended model

IV. Real-Time application

The final application should be able to run in real-time, so just testing our models in samples is not enough to examine it in a ”real-world” case.

The program starts listening to the stream provided (in this case) by the microphone of the computer and when a peak of amplitude it’s found, the program records a stereo 4-second long sample at 44100 Hz. The peak is revealed when $mean(abs(mySignal)) > \tau$, τ is a variable threshold parameter that needs to be tuned every time we switch the input source.

After this audio feature is extracted, in our case the segmented mel-spectrogram and the evaluation is performed. Nevertheless, if the membership of the max class is lower than a threshold of 50%, the prediction is considered too uncertain to signal it to the driver. If the prediction accuracy is high enough and the class predicted is one of the two of interest, car_horn and siren, a visual or tactile feedback will be prompted.

V. Experimental setup and results

All the tests, timings, and results were calculated and performed on a 4-core processor with 16GB of DDR4 ram and the models have been trained on a GTX 1060 6GB NVIDIA graphic card, the last Matlab ($R2019b$) version was used to write the algorithms and the tests.

A. Dataset

The dataset used in our project is the ”URBANSOUND8K” [7], it’s open-source and freely available at [1]. It contains 8732 labeled sound excerpts from 10 classes:

- air_conditioner
- car_horn
- children_playing
- dog_bark
- drilling
- enginge_idling
- gun_shot
- jackhammer
- siren
- street_music

After some experiments, we noticed that car_horn sounds were recognized worst, and that probably because of the under-representation of that class in the dataset. So we proceeded to download more car horn sounds from freesound[2] and integrate them with the old dataset. After this results completely changed passing from a 60/70% accuracy for that specific class to more than 90%.

TABLE I
Extended Dataset

Label	Count
air_conditioner	1000
car_horn	538
children_playing	1000
dog_bark	1000
drilling	1000
engine_idling	1000
gun_shot	374
jackhammer	1000
siren	929
street_music	1000

The samples have many different sampling rates and lengths, so during preprocessing we made them homogeneous. The 10 classes are not all required but it’s good to use all the information we have to prevent underfitting and to better recognize different sounds.

[1]https://urbansounddataset.weebly.com/urbansound8k.html
[2]https://annotator.freesound.org

B. Model Parametrization

1) KNN: The distance metric selected is the euclidean distance, the most used and more proper for numerical values, with the number of neighbors = 10, a value too small can lead to wrong inference due to noise, while a too big can include points from other classes. The kernel function selected is the Inverse-square: $\frac{1}{d^2}$. Finally, we applied 10-fold stratified cross-validation.

2) CNN: The training options for the convolutional neural network have been based on [6] and optimized by [8], subsequently adapted with some tests and error observations.

- Learning Rate: 0.025
- L2Regularization: 0.005
- Momentum: 0.9
- MaxEpoch: 10
- Shuffle: every-epoch
- LR Drop period: 2
- LR Drop Factor: 0.2

VI. Conclusions

In this work, we presented and tested three different classification algorithms applied to audio pattern recognition of an acoustic scene.

We determined that the best model for our purpose is the Convolutional Neural Network using as audio feature the mel-segmented spectrogram. The high-accuracy results ($> 99\%$), and the comparison between the time needed to train and test the various algorithm, which is not comparable, not even in the scale, made the selection surprisingly easy. This technique has also been tested in a simulation of what can be the ”real-time” application, having stimulating results on a possible future real implementation.

VII. Aknowledgments

The project was developed during the Audio Pattern Recognition course held by Professor Stavros Ntalampiras of the University of Milan.

References

[1] Martin E. Gordon, and Justin J. Pearson, “Preliminary Analysis of Roadway Accident Rates for Deaf and Hard-of-Hearing Drivers” in National Academy of Forensic Engineers. Vol. 33 No. 1 June 2016.

[2] The Matworks, ”Tall Arrays for Out-of-Memory Data”, [online] It.mathworks.com. Available at: https://it.mathworks.com/help/matlab/import_export/tall-arrays.html

[3] S. Ntalampiras, “Universal background modeling for acoustic surveillance of urban traffic” in Digital Signal Processing 31 pp: 69–78, Milan, May 2014.

[4] F. Li, J. Ma, and D. Huang, “MFCC and SVM Based Recognition of Chinese Vowels” in CIS 2005, pp 812-819, Bejing, December 2005.

[5] D. De Benito, A. Lozano-Diez, D. Toledano, and J. Gonzales-Rodriguez “Exploring convolutional, recurrent, and hybrid deep neural networks for speech and music detection in a large audio dataset” in EURASIP Journal on Audio Speech and Music Processing. December 2019

[6] Han, Yoonchang, J. Park, and K. Lee. ”Convolutional neural networks with binaural representations and background subtraction for acoustic scene classification.” in DCASE: 1-5, Munich, 16 November 2017.

[7] J. Salamon, C. Jacoby, and J. P. Bello, “A dataset and taxonomy for urban sound research,” in 22nd ACM In- ternational Conference on Multimedia (ACM-MM’14), Orlando, FL, USA, Nov. 2014.

[8] The Matworks, ”Acoustic Scene Recognition Using Late Fusion”, [online] It.mathworks.com. Available at: https://it.mathworks.com/help/audio/examples/acoustic-scene-recognition-using-late-fusion.html