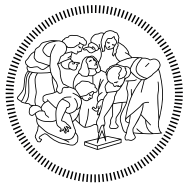


POLITECNICO
MILANO 1863

Towards advertisement clustering: two approaches based on Neural Networks

M.Sc. Thesis of
Andrea Caceffo
ID: 854645

Supervisor:
Prof. Fabio Antonacci

Co-supervisor:
Dr. Guglielmo Zanni - *MakarenaLabs SRL*

Programme:
Mathematical Engineering

School of Industrial and Information Engineering
Politecnico di Milano

Academic Year:
2020-21

Andrea Caceffo
Towards advertisement clustering: two approaches based on Neural Networks - 2020-21

Abstract

The advertisement aired by radio broadcasters is one of their major incomes. The provider, *i.e.* the company which owns the radio station, and the customer, *i.e.* whichever subject interested in advertising a product, sign a contract in which some aspects of the advertisement campaign are specified.

Therefore it is evident the usefulness of an automatic system which is capable of assess the compliance with the contractual terms. A primary requirement for such a system is to be able to univocally and as accurately as possible identify a specific commercial break.

In this Thesis we analyze two Neural Networks approaches to carry out the first step of the identification, *i.e.* the generic recognition of advertisement segments among all the other audio contents. The first model consists of a deep convolutional network which, taking an audio excerpt as input, it is capable of responding with its probability of being an advertisement. The second model is based on the TCN (Temporal Convolutional Network) architecture employed to extract from the audio input the music and speech energies.

Moreover, to accomplish the training of these two Neural Networks, we collected advertising, music and speech audio samples from some italian radio broadcasts, then gathered in MUSPAD (MUsic, SPeech and ADvertisement) dataset.

The first model outperformed the second one in all the experimental results. These noteworthy performances let us think that such a model might be employed, within the advertisement identification system, in an industrial context.

Keywords: Neural Networks, advertisement, clustering, classification

Andrea Caceffo
Towards advertisement clustering: two approaches based on Neural Networks - 2020-21

Abstract in lingua italiana

La pubblicità trasmessa alla radio è una delle principali fonti di ricavo per le società proprietarie delle emittenti radiofoniche. Il fornitore, in questo caso l'emittente radio, e il cliente, cioè qualsiasi entità interessata a pubblicizzare un prodotto sulla piattaforma radiofonica, si accordano in forma contrattuale su vari aspetti della campagna pubblicitaria.

Risulta quindi evidente l'utilità di un sistema automatizzato capace di valutare il rispetto dei termini contrattuali. Requisito fondamentale di tale sistema è quello di essere in grado di identificare in maniera univoca e il più possibile precisa le singole pubblicità.

In questa Tesi analizziamo due approcci a Reti Neurali per portare a termine il primo passo dell'identificazione, cioè il riconoscimento generico di segmenti pubblicitari tra gli altri tipi di contenuti. Il primo modello consiste in una rete convolutiva profonda che preso in ingresso un segmento audio è in grado di stimare la probabilità che esso sia una pubblicità. Il secondo modello si basa sull'architettura TCN (Temporal Convolutional Network), la quale viene impiegata per estrarre dall'audio in ingresso una stima delle energie dovute al contenuto musicale e del parlato.

Inoltre per portare a termine l'addestramento delle due Reti Neurali si è resa necessaria la collezione di campioni radiofonici, da varie emittenti italiane, contenenti pubblicità, musica e parlato, poi raccolti in MUSPAD (MUsic, SPeech and ADvertisement dataset).

I risultati ottenuti vedono il primo modello superare come prestazioni il secondo. I buoni risultati riscontrati dal primo modello inducono a pensare che un'architettura di questo genere possa essere utilizzata, all'interno del sistema di identificazione pubblicitaria, in contesto industriale.

Parole chiave: Reti Neurali, pubblicità, clustering, classificazione

Andrea Caceffo
Towards advertisement clustering: two approaches based on Neural Networks - 2020-21

Contents

List of Figures

List of Tables

Andrea Caceffo
Towards advertisement clustering: two approaches based on Neural Networks - 2020-21

1. Introduction

Advertisement broadcasting constitutes a fundamental source of revenue for the companies which own the radio stations. Radio companies (*providers*) sign contracts with the entities that commission the advertising broadcasting (*customers*). These contracts might set some important characteristics of the advertising campaign, such as, for instance, the number of times that a commercial has to be aired or the particular time of the day in which to broadcast it. In this scenario it is natural to have at disposal a procedure to check if the terms in the contracts are respected. Particularly this aspect is essential in the customers' perspective.

Our work aims to satisfy this need for a way to automatically assess the compliance of the broadcastings with the contracts. Such a machinery requires a section which must be able to correctly identify the specific advertisements.

Previous efforts to solve the task of advertisement identification in a FM radio stream are mainly based on the audio fingerprinting approach. This technique consists of extracting a digest from an audio sample, *i.e.* the fingerprint, and store it in a database together with its metadata. When an unlabeled audio sample is presented to the system, its fingerprint is extracted and matched against those stored in the database. In this way an unlabeled sample can be identified and its metadata can be obtained. Many fingerprinting methods are in use, Cano et al. [1] provided a review of them.

Therefore, since the fingerprinting approach can be exploited for any audio sample, unregarding its content, it has been applied to advertising identification. For instance Cerquides [2] proposed a system for real time advertisement tracking in FM radio, based on the fingerprinting method by Haitsma et al. [3]. Ouyang et al. [4] leveraged both audio and visual fingerprints to detect TV commercial advertisements.

Surely more explored is the field of music recognition. The song identification is essential to check for copyright infringements and to collect the royalties owed to the authors. Moreover this task was shared to the general audience by the renowned application called *Shazam*® which, again, makes use of fingerprinting (Wang [5]) to identify songs in real time.

Another approach to music recognition, and in general to the *MIR* (Music Information Retrieval) tasks, are Neural Networks (NN). This approach, famous for its great success in the field of visual pattern recognition, has been applied also in the audio framework. Neural Networks indeed allow to leverage the representativeness of the features learned by the model, instead of making use of the manually extracted ones (*e.g.* Mel-Frequency Cepstral Coefficient, Zero-Crossing Rate etc.), employed in the classical approaches to MIR.

Various network architectures has been taken into account for several tasks with success. For instance Dong [6] managed to reach human-level accuracy in the

task of music genre classification using Convolutional Neural Networks (CNN). Benito et al. [7] evaluated several architectures, including CNNs and Recurrent Neural Networks (RNN), in the field of music-speech segmentation. Again a CNN is employed by Yu et al. [8] for cover song identification, *i.e.* to identify a new recording produced by someone who is not an original composer or singer.

It is pretty straightforward to apply a Neural Network approach to our identification problem of advertisements broadcasted by radio stations. Such approach let us split the identification task in a first stage of classification, which gets rid of not-advertisement contents, and a second stage to actually identify the specific advertising. Considering that commercial breaks occupy approximately only the 10% of the total time, this procedure could lead to a considerable inference speed-up, excluding at the beginning of the pipeline the not-advertisement content.

In particular in this Thesis we aim to explore the possibility of implementing an automatic identification system based on Deep Neural Network (DNN) models, *i.e.* we want to check if such a system can reach the minimum performances required for a commercial application.

A possible structure for this machinery is composed by a segmentation front-end (segmenter), a DNN-based classifier (classifier) and a DNN which perform the identification task (identifier). In fig. 1.1 we show the general design of our automatic system.

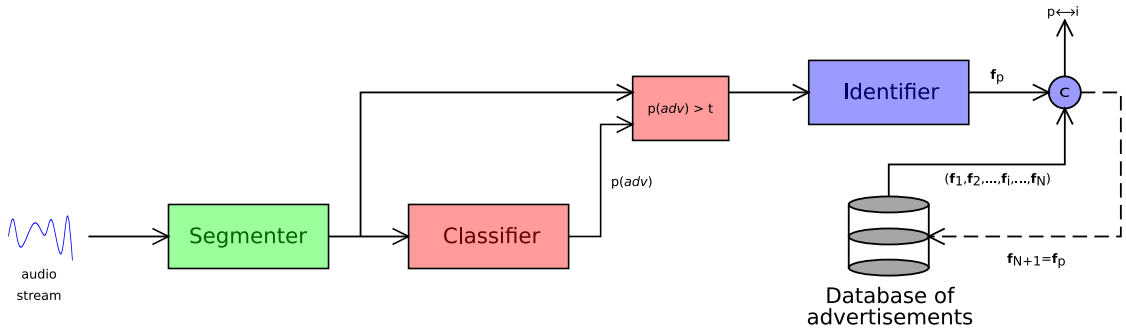


Figure 1.1.: Overall architecture of the identification system

In red we have highlighted the part which is the main object of this Thesis, *i.e.* the DNN-based classifier. We are going to test two different DNN architectures, obtained with some modifications from already existing models, to see how they compare and if, at least one of them, they show promising performance with a view to employ them in a commercial application.

The first model is based on the SoundNet architecture by Aytar et al. [9]. It is modified in order to provide, as output, the probability of a variable-length raw audio segment to be an advertising. This is obtained adding to the 1D CNN block a global pooling layer, to get a fixed length vector of feature, followed by a dense layer with a sigmoidal nonlinearity.

The second model is structurally more complex being composed by two stages. The first stage takes as input a variable-length raw audio and, after some

preprocessing, it extracts estimates for the energies of the music and speech components. The architecture consists of a conversion of the Temporal Convolutional Network (TCN) by Meléndez-Catalán [10] to a regression task. The second stage clearly looks like the last layers of the SoundNet being a stack of convolutional layers plus global pooling and a dense layers. It takes the music and speech energies extracted by the first stage and outputs, again, the probability of facing an advertising sample. In this case the goal is not only assess the performances of such a network, but also check if the music and speech energies alone provide sufficient informations for the advertisement classification task.

Still referring to fig. 1.1, the green box denotes the segmenter stage. The goal of this front-end section is to split a continuous audio stream in shorter segments which present similar features. In an ideal condition the segmenter shoud be able to provide the classifier with semantically coherent samples, *e.g.* dividing a radio broadcasts in songs, news, advertisements etc. This technology is already developed by *MakarenaLabs SRL*®[1], *i.e.* the company which provided this Thesis.

The final stage of the pipeline, in blue, is the identifier. The goal of this last part is to decide whether a sample, which has been classified as an advertising, is already present in the database or, being new, it must be inserted into it. We decided not to employ a fingerprinting approach for this task, but to identify the single advertising performing clustering in the space of features extracted by a DNN. This choice inspired the title of the Thesis. In chapter 6 we give a brief insight of how this clustering might be carried out, being clear that this is not the central argument of the Thesis, but rather a future development.

The following digression is divided in: chapter 2 in which we give a brief introduction of Neural Networks, chapter 3 providing explanation of the assessed models, chapter 4 regarding the datasets employed to train such networks and chapter 5 where we expose the experimental results. Finally conclusions and further developments are dealt with in chapter 6. Theoretical and technical deepenings can be found in chapter A and chapter B.

[1] https://www.makarenalabs.com/

2. Background on Neural Networks

We briefly illustrate the concept of *Neural Networks*, from a mathematical point of view, through the quintessential deep learning models: *Deep Feed-forward Networks*. Before proceeding with the more elaborate models called *Convolutional Neural Networks* and *Temporal Convolutional Networks*, we discuss the network training process from a probabilistic viewpoint. These sections follow the presentation by Bishop [11] chapter 5 and Goodfellow et al. [12] chapter 6.

2.1. Deep Feed-forward Networks

It is straightforward to introduce this argument in the *linear models* framework. Linear models are in general based on linear combinations of fixed nonlinear *basis functions* $\phi_j(\mathbf{x})$ and take the form:

$$y(\mathbf{x}, \mathbf{w}) = f\left(w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})\right) = f\left(\sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x})\right) = f(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})) \tag{2.1}$$

where $f(\cdot)$ is a nonlinear activation function in the case of classification and is the identity in the case of regression, and $\phi_0(\mathbf{x}) = 1$. In the case of regression this type of model is linear both in the parameters and in the basis functions.

Neural networks extend this kind of model making the basis functions $\phi(\mathbf{x})$ depend on parameters, which can be adjusted alongside the coefficients $\{w_j\}$, during training. In neural networks each basis function is a nonlinear function of a linear combination of the inputs, where the coefficients in the linear combination are trainable parameters. This leads to the basic neural network model as a composition of functional transformation, as we see in the following.

We begin by building what is called a *layer*, composed of M *units*. This is done by (1) assembling M linear combinations of the layer input variables $\mathbf{x}^{(l)} = (1, x_1^{(l)}, ..., x_D^{(l)})$ in the form:

$$a_j^{(l)} = \sum_{i=1}^{D} w_{ji}^{(l)} x_i^{(l)} + w_{j0}^{(l)} = \sum_{i=0}^{D} w_{ji}^{(l)} x_i^{(l)} \tag{2.2}$$

for $j = 1, ..., M$ and (2) applying a differentiable, nonlinear *activation function* $h(\cdot)$:

$$z_j^{(l)} = h(a_j^{(l)}) \tag{2.3}$$

The parameters $w_{ji}^{(l)}$ are referred as *weights*, while parameters $w_{j0}^{(k)}$ as *biases*. The quantities $a_j^{(l)}$ are known as *activations*. The superscript l indicates that the quantities belong to the $l-th$ layer. Looking at eq. (2.1) it is clear the correspondence $z_j \leftrightarrow \phi_j(\mathbf{x})$.

Repeatedly applying this construction, assuming as input of each subsequent layer the output of the previous one, we obtain a stack of stages that eventually end up with the output units. The layers stacked inside between the input and the output are called *hidden layers*, and the correspondent units *hidden units*. In particular for the $j-th$ output unit of a network, with $L-1$ hidden layers, we have:

$$\begin{aligned} y_j(\mathbf{x},\mathbf{w}) &= g\left(\sum_{i=1}^{M^{(L-1)}} w_{ji}^{(L)} z_i^{(L-1)} + w_{j0}^{(L)}\right) = g\left(\sum_{i=1}^{M^{(L-1)}} w_{ji}^{(L)} h(a_i^{(L-1)}) + w_{j0}^{(L)}\right) = \\ &= g\left(\sum_{i=1}^{M^{(L-1)}} w_{ji}^{(L)} h\left(\sum_{k=1}^{M^{(L-2)}} w_{ik}^{(L-1)} z_k^{(L-2)} + w_{i0}^{(L-1)}\right) + w_{j0}^{(L)}\right) = \\ &= \ldots = g(\mathbf{f}_L(\mathbf{f}_{L-1}(\ldots \mathbf{f}_1(\mathbf{x};\mathbf{w}_1)\ldots);\mathbf{w}_{L-1});\mathbf{w}_L) = f(\mathbf{x};\mathbf{w}) \end{aligned} \tag{2.4}$$

where $M^{(l)}$ is the number of units of the $l-th$ layer. The meaning of $f(\cdot)$, $\mathbf{f}_l(\cdot)$ and $\mathbf{w}_l$ is clear from the context. Here, for simplicity, we have assumed $h(\cdot)$ and $g(\cdot)$ as activation functions of respectively all the hidden units and all the output units, which is not always the case. This composition of functions can be represented in the form of a network diagram, as shown in fig. 2.1.

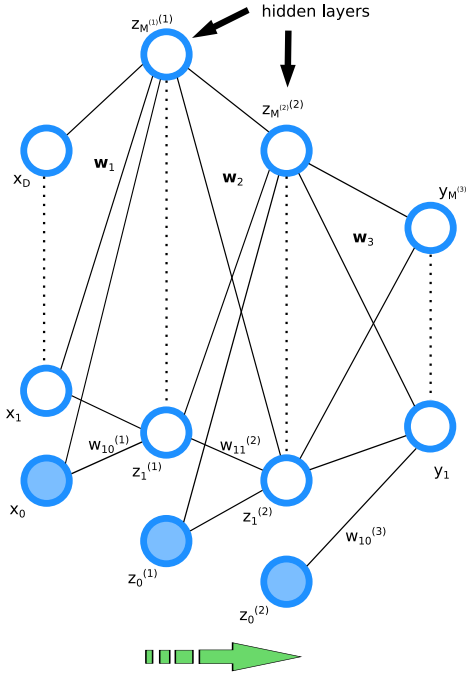


Figure 2.1.: Example of a 3-layers feed-forward network

Approximation property Using nonlinear activation functions on hidden units allows us to approximate any continuous function on a closed and bounded domain to arbitrary accuracy, provided that the network has a sufficiently large number of hidden units and layers. This is true for a wide range of activation functions, but excluding polinomials. This is why neural networks are said to be *universal approximators*. See appendix section A.1 for a proof of this property.

Indeed in order to approximate a function f^*, given the network mapping $\tilde{y} = f(\mathbf{x}; \mathbf{w})$, during the training stage, at each input sample $\mathbf{x}$ feeded from the dataset, we drive $f(\mathbf{x}; \mathbf{w})$ to match $f^*(\mathbf{x})$. This is done by optimizing for $\mathbf{w}$ a measure of the error between $f(\mathbf{x}; \mathbf{w}) = \tilde{y}$ and t, being t the known target value attached to $\mathbf{x}$, *i.e.* an estimate of the true function's value: $t \approx f^*(\mathbf{x})$.

Even though this universality is reassuring, the key problem is to find a suitable set of parameter values, given the finite number of training data. In the next subsection we show the *maximum likelihood* approach to tackle this issue, which is usable since in the supervised framework we have the ground truth values at our disposal.

2.2. Network training

A general view of network training can be obtained by first giving a probabilistic interpretation to the network outputs. We consider the regression and the binary classification tasks, which are faced in this work.

Regression In the regression framework we start considering a vector of target values $\mathbf{t} = (t_1, ..., t_p) \in \mathbb{R}^p$ and we model it as $\mathbf{t} = \mathbf{f}(\mathbf{x}; \mathbf{w}) + \boldsymbol{\epsilon}$, with $\boldsymbol{\epsilon} \sim \mathcal{N}_p(0, \boldsymbol{\Sigma})$ and $\mathbf{f}$ is the network output vector function. Therefore we obtain the following conditional distribution for $\mathbf{t}$:

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \mathcal{N}_p(\mathbf{t}; \mathbf{f}(\mathbf{x}, \mathbf{w}), \boldsymbol{\Sigma}) \tag{2.5}$$

The choice of Gaussian noise might be, in some cases, restrictive, but it is the best starting point for the analysis. Given a dataset consisting of: N *i.i.d* inputs collected in the matrix $\mathbf{X} = (\mathbf{x}_1^T, ..., \mathbf{x}_q^T) \in \mathbb{R}^{N \times q}$ and the corresponding N targets stacked in $\mathbf{T} = (\mathbf{t}_1^T, ..., \mathbf{t}_p^T) \in \mathbb{R}^{N \times p}$, we can construct from eq. (2.5) the correspondent *likelihood* function:

$$p(\mathbf{T}|\mathbf{X}, \mathbf{w}, \boldsymbol{\Sigma}) = \prod_{n=1}^{N} p(\mathbf{t}_n|\mathbf{x}_n, \mathbf{w}, \boldsymbol{\Sigma}) \tag{2.6}$$

and, as usual, the more convenient *log-likelihood* function:

$$-\frac{Np}{2}\log(2\pi) - \frac{N}{2}\log|\boldsymbol{\Sigma}| - \frac{1}{2}\sum_{n=1}^{N}[\mathbf{t}_n - \mathbf{f}(\mathbf{x}_n; \mathbf{w})]^T \boldsymbol{\Sigma}^{-1}[\mathbf{t}_n - \mathbf{f}(\mathbf{x}_n; \mathbf{w})] \tag{2.7}$$

where $|\boldsymbol{\Sigma}|$ is the determinant of the noise covariance matrix. The solution for the maximum likelihood estimate is obtained maximizing eq. (2.7).

With the goal of finding ML estimate of the weights, *i.e.* $\mathbf{w}_{ML}$, we minimize the error function given by:

$$e(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}[\mathbf{t}_n - \mathbf{f}(\mathbf{x}_n;\mathbf{w})]^T\boldsymbol{\Sigma}^{-1}[\mathbf{t}_n - \mathbf{f}(\mathbf{x}_n;\mathbf{w})] \tag{2.8}$$

In the case in which the components of vector $\mathbf{t}$ are supposed independent of each other, conditional on $\mathbf{x}$ and $\mathbf{w}$, with shared known variance, we have a diagonal covariance matrix $\boldsymbol{\Sigma} = \sigma^2\mathbf{I}$. Therefore the minimization of eq. (2.8) reduces to the minimization of the usual sum-of-squares error function:

$$e(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}[\mathbf{t}_n - \mathbf{f}(\mathbf{x}_n;\mathbf{w})]^T[\mathbf{t}_n - \mathbf{f}(\mathbf{x}_n;\mathbf{w})] \tag{2.9}$$

Averaging this quantity over a *mini-batch*[1] we obtain the *Mean Square Error* loss function. In practice, the nonlinearity of $\mathbf{f}(\mathbf{x}_n;\mathbf{w})$ causes the error $e(\mathbf{w})$ to be non-convex, so that local minima of the loss function may be encountered.

Binary classification In the binary classification framework we have a single target variable t such that $t = 0$ denotes class C_1 and $t = 1$ denotes class C_2. In this case our network function $f(\mathbf{x};\mathbf{w})$ is scalar and restricted to the interval $[0, 1]$. This is obtained with a single output unit having a sigmoidal activation function. Moreover we can consider $f(\mathbf{x};\mathbf{w})$ as the conditional probability $p(C_1|\mathbf{x})$. The conditional distribution of the targets given the inputs is then the following Bernoulli distribution:

$$p(t|\mathbf{x},\mathbf{w}) = f(\mathbf{x};\mathbf{w})^t[1 - f(\mathbf{x};\mathbf{w})]^{1-t} \tag{2.10}$$

Considering a training set of independent observations, then the error function to be minimized is given by the negative log-likelihood, *i.e.* the *Binary Cross-entropy* loss function:

$$e(\mathbf{w}) = -\sum_{n=1}^{N}[t_n \log(f(\mathbf{x}_n;\mathbf{w})) + (1 - t_n)\log(1 - f(\mathbf{x}_n;\mathbf{w}))] \tag{2.11}$$

Minimizing the loss function This task of finding a weight vector which minimizes the loss function $e(\mathbf{w})$ cannot be accomplished simply by solving the gradient equation $\nabla e(\mathbf{w}) = 0$.

Indeed the highly nonlinear behaviour of the function may lead to the presence of several stationary points in which the gradient vanishes or it is numerically very small. Hence, for a successful application of neural networks, multiple stationary points need to be "visited" by the algorithm in order to find a suitable minimum, which might not be the global one, but a sufficiently good solution.

[1] A limited amount of samples used to update the gradient. The concept is made clearer in the following.

Moreover there is clearly no hope of finding an analytical solution for the gradient equation.

To tackle this problem *iterative* procedures are employed. The usual scheme is the following: after choosing some initial value $\mathbf{w}^{(0)}$ for the weight vector, at each iteration τ, we apply the update rule

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau+1)} \tag{2.12}$$

until convergence.

Many methods involve the use of the gradient information, for instance the *gradient descent* algorithm. Here the weight update is chosen to comprise a small step in the direction of the negative gradient:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta^{(\tau+1)}\nabla e(\mathbf{w}^{(\tau)}) \tag{2.13}$$

In practice what mostly differentiates the various gradient-based algorithm is the choice of the parameter $\eta^{(\tau)} > 0$, called *learning rate*, which can also change at each iteration. In this work we use the well-known *Adam* algorithm (Kingma et al. [13]), for more information see section B.2.3.

Note that the loss, *i.e.* the error function, is defined with respect to the training set, and so it is the gradient, which is computed from the data. The choice of how many samples employ to evaluate ∇e, which is possible if the cost function is additive like we will suppose in eq. (2.17), distinguishes three type of algorithms:

- *Batch* method: at each step the whole training set it used at once to compute the gradient.
- *Stochastic* methods: at each step the gradient is estimated randomly choosing N samples from the training set. Therefore more than 1 step is needed to complete an *epoch*[2]. They can further be divided in:
  - *Online* method: $N = 1$ so that the gradient is estimated using only one sample.
  - *Mini-batch* method: $N > 1$, this is somewhere in between the previous ones.

In table 2.1 we analyze the choice of the *batch-size* N.

	Pros	Cons
Small batch-size	Faster convergence[i] Avoid redundancies in the dataset Regularizing effect	More step to complete an epoch Multicore arch. underutilized
Large batch-size	More accurate estimate of ∇e	Gain[ii] less than linear *i.e.* $\sim \sqrt{N}$ Memory consumption

[i] In terms of total computation, not in terms of number of updates.

[ii] The standard error of the mean gradient estimated from N samples is given by $\sigma/\sqrt{N}$, where σ is the true standard deviation of the value of the samples.

Table 2.1.: Pros and cons of increasing/decreasing the batch-size

[2] An *epoch* is the passage of the algorithm on the whole training set

Computing the gradient In this paragraph we illustrate an efficient technique for evaluating the gradient of the error funtion $\nabla e(\mathbf{w})$ for a general feed-forward network, which may comprise *skip-connections*[3]. This scheme is called *error back-propagation* or *backprop*, as originally considered by Rumelhart et al. [14]. In a general feed-forward network, each unit computes a weighted sum of the inputs of the form:

$$a_j = \sum_i w_{ji} z_i \tag{2.14}$$

where z_i is the output of a unit (which can be also the input units, *i.e.* $z_i \equiv x_i$) that sends a connection to unit j, and w_{ji} is the weight associated with that connection. Considering, for simplicity, a unique nonlinear activation function $h(\cdot)$ at each hidden unit we have:

$$z_j = h(a_j) \text{ for } j \in J_{hidden} \tag{2.15}$$

As far as output units are concerned we fix:

$$z_j = g(a_j) \text{ for } j \in J_{output} \tag{2.16}$$

We gather all weights, activations and units' outputs respectively in $\mathbf{w}$, $\mathbf{a}$ and $\mathbf{z}$. Many error functions, for instance those defined by maximum likelihood for a set of i.i.d data, can be written as a sum over the samples:

$$e(\mathbf{w}) = \sum_{n=1}^{N} e_n(\mathbf{w}) \tag{2.17}$$

Therefore we can now consider the derivative of $e_n(\mathbf{w})$ with respect to weight w_{ji}. Gathering these terms in a vector we obtain the gradient $\nabla e_n(\mathbf{w})$. From now on each activations a_j, unit outputs z_j and targets t_j are to be considered correspondent to the $n-th$ training input instance $\mathbf{x}_n$. We omit to indicate it to simplify notation.

$$\frac{\partial e_n}{\partial w_{ji}} = \frac{\partial e_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} \tag{2.18}$$

where we have repeatedly applied the chain rule for partial derivatives. Moreover deriving eq. (2.14) yields:

$$\frac{\partial a_j}{\partial w_{ij}} = z_i \tag{2.19}$$

so that:

$$\frac{\partial e_n}{\partial w_{ji}} = \frac{\partial e_n}{\partial a_j} z_i \tag{2.20}$$

[3] A *skip-connection* is a forward link between two units belonging to non-subsequent layers.

We are now required to compute the term $\frac{\partial e_n}{\partial a_j}$, often called *error*. For the output units it is straightforward. Indeed considering, for instance, a sum-of-squares loss function:

$$e_n = \frac{1}{2}\sum_k (z_k - t_k)^2 \text{ for } k \in J_{output} \tag{2.21}$$

then:

$$\frac{\partial e_n}{\partial a_k} = (z_k - t_k)\frac{\partial z_k}{\partial a_k} = [g(a_k) - t_k]g'(a_k) \text{ for } k \in J_{output} \tag{2.22}$$

where t_k as usual indicates the $k-th$ element of the target.
While for a general hidden unit's output we have:

$$\begin{aligned}\frac{\partial e_n}{\partial a_j} &= \sum_i \frac{\partial a_i}{\partial a_j}\frac{\partial e_n}{\partial a_i} = \sum_i \frac{\partial a_i}{\partial z_j}\frac{\partial z_j}{\partial a_j}\frac{\partial e_n}{\partial a_i} = \\ &= h'(a_j)\sum_i w_{ij}\frac{\partial e_n}{\partial a_i}\end{aligned} \tag{2.23}$$

where the sum runs over all units i to which unit j sends connections. The recursive nature of eq. (2.23) is evident. This leads to the following algorithm:

1. Apply an input vector $\mathbf{x}_n$ to the network and feed it forward to calculate all the activations a_j and correspondent outputs z_j of the network.
2. Evaluate $\frac{\partial e_n}{\partial a_k}$ at output units utilizing eq. (2.22).
3. Backpropagate $\frac{\partial e_n}{\partial a_j}$ using eq. (2.23) to obtain this quantity at each hidden unit in the network.
4. Use eq. (2.20) to compute the desired derivatives, *i.e.* the gradient $\nabla e(\mathbf{w})$.

For batch and mini-batch methods the derivative of the total error eq. (2.17) is clearly given by the sum of the derivatives calculated at each sample n. Considering a network having W weight parameters, backpropagation algorithm has an overall cost of $O(W)$ operations which make it very profitable with respect to other methods, like for instance *finite differences* which has a cost of $O(W^2)$ flop.

Regularization Besides optimization, another fundamental issue in the network training is the *regularization*. This controls the balance between *underfitting* and *overfitting*, *i.e.* the model's property of being able to capture the problem complexity and at the same time being general enough to be applied to new data obtaining sensible results. Reporting the definition in Goodfellow et al. [12] regularization is: *any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error*. At a first glance the generalization error might seem a simple function of the number M of hidden units. This usually false due to the presence of local minima in the error function.

Here we give an overview of some regularization methods including some that are used in this work:

- *Early stopping*. While for many of the optimization algorithms the training error is a nonincreasing function of the iteration index, we find that, as far as the validation error is concerned, it shows at the beginning a decreasing pattern followed by an increasing one when the network starts to overfit. Hence we can stop the training at the point of smallest validation error.

- *Parameter norm penalties*. It consists in substituting the usual loss function to optimize with the regularized one, *i.e.* $\tilde{e}(\mathbf{w}) = e(\mathbf{w}) + \alpha\Omega(\mathbf{w})$ where $\Omega(\mathbf{w})$ represent some kind of measure of the magnitute of the weights. Indeed in the neural network framework usually Ω penalizes only the weights, leaving out the biases. Some examples are:
  - L^2 regularization, commonly known as *weight decay*. In this case we add the regularization term $\Omega(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top\mathbf{w} = \frac{1}{2}\|\mathbf{w}\|_2^2$.
  - L^1 regularization. Here the choice of regularization term is $\Omega(\mathbf{w}) = \sum_i |w_i| = \|\mathbf{w}\|_1$.

  Whereas both the regularizer function shrinks the weights $\mathbf{w}$, in comparison L^1 regularization results in a solution that is more *sparse*. In addition these regularizers can be interpreted as prior distributions over the weight vector $\mathbf{w}$, in particular in the case of the L^2 weight decay it is a zero-mean Gaussian prior.

- *Dropout*. First presented by Srivastava et al. [15] this technique has been proved to be more effective than other standard computationally inexpensive regularizers. Moreover it is computationally cheap and applicable to almost any problem and training procedure. Here, each time we load examples into a minibatch, a binary mask is randomly sampled with an inclusion probability whose hyperparameter is fixed before training. This mask is then applied to the non-output units of the network. Such a procedure of including/excluding the units basically leads to the training of an ensemble of subnetworks. The optimal weights are then computed using the *weight scaling inference rule*: the estimated weights are divided by the reciprocal of the inclusion probability, *i.e.* the reciprocal of one minus the *dropout rate*. In this work we make use of the so called *spatial dropout*. In this particular case the mask is randomly computed on the spatial coordinates only, then all the channels correspondent to the selected items are discarded.

- *Dataset Augmentation*. A straightforward approach to improve the capacity of the model to generalize is increasing the size of the training set. For some problem this is possible by creating new fake data, for instance trasforming or injecting noise in the samples already present in the training set.

- *Batch normalization*. See appendix section B.2.4.

2.3. Convolutional Neural Networks

As defined in LeCun et al. [16], and previously investigated in LeCun [17], *convolutional networks* combine three architectural ideas: *local receptive fields*[4], *shared weights* and spatial or temporal *subsampling*. This approach ensures some degree of shift, scale and distortion invariance. Because of that it can be considered a regularization method. Moreover this architecture needs to store fewer parameters, leading to an improvement also in efficiency, and can manipulate variable-length inputs.

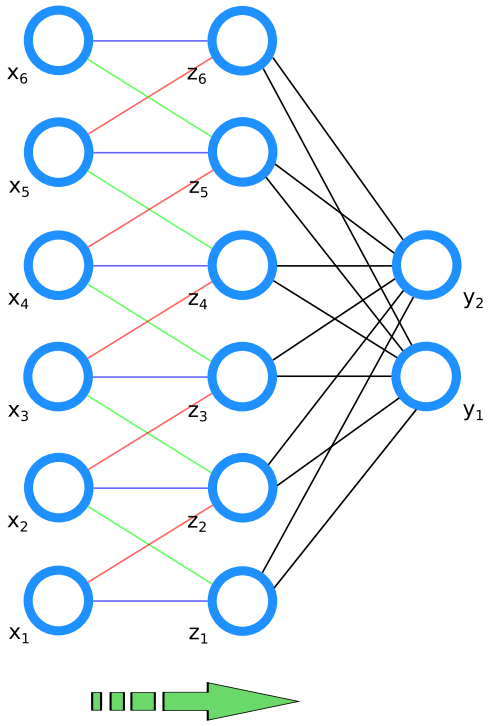


Figure 2.2.: Example of a network with a convolutional and a dense layers

Local receptive fields and weight sharing are made possible by using the *convolution* operation. These two concepts are well illustrated in fig. 2.2: a unit from the hidden layer is affected only by three inputs and the links labeled with the same color are constrained to be equal. From a bayesian point of view, this is equal to choosing an infinitely strong prior distribution over the weight space of a fully connected network[5], imposing some weights to be zero and other to be equal.

We analyze now the 1-D version of the convolution operation. The 2-D version, which is also used in this work, is a pretty straightforward generalization.

Before starting we have to clarify a point: the implementation of convolution

[4]The *receptive field* is defined as the size of the region in the input which produces a particular output, *i.e.* the set of all the inputs which are connected to it through a path of edges.

[5]Network composed only by dense layers, *i.e.* layers in which each input is connected to each output unit.

in many neural network libraries actually features a related function called *cross-correlation*. This is the same as convolution, but without flipping the kernel and consequentially without commutative property. From now on we are going to refer to this function as convolution. The discrete 1-D formula for a finite-length input is given by:

$$s_t = (\mathbf{x} * \mathbf{w})_t = \mathbf{w}^\top \mathbf{x}_{t:t+K} = \sum_{k=0}^{K-1} x_{t+k} w_k \tag{2.24}$$

where K is the *kernel* length, $\mathbf{x}$ is the input and $\mathbf{s}$ is called *feature map*. Usually in the applications we implement a centered version of the convolution with a kernel having odd length like in fig. 2.3:

$$s_t = (\mathbf{s} * \mathbf{w})_t = \mathbf{w}^\top \mathbf{x}_{t-\tilde{K}:t+\tilde{K}} \tag{2.25}$$

where $\tilde{K} = \frac{K-1}{2}$. Once computed the feature map one may or may not add the biases b_t to obtain the activations $a_t = s_t + b_t$, and further apply the nonlinearities $z_t = h(a_t)$.

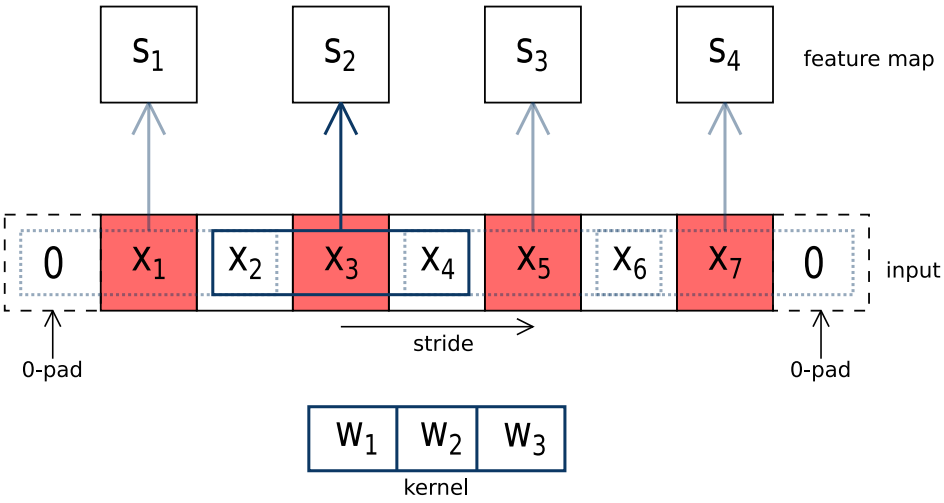


Figure 2.3.: Example of a convolution operation with kernel-size=3 and stride=2

With the help of fig. 2.3 we introduce some of the parameters which define a convolutional layer:

- *kernel size*. It is the shape parameter of the kernel. Bigger kernels means bigger receptive fields but also more weight parameters to optimize.
- *padding*. It is the procedure of adding values, usually equal to zero, at the input's sides to obtain a feature map with the same dimension of the input. Indeed we can clearly notice that, without padding, the application of the convolution operation results in an implicit downsampling, with respect to the input size.

- *stride*. It is the distance between two consecutive values over which the kernel is centered. This is a direct way to control the downsampling factor of the convolution layer.

Since often the inputs include multiple *channels*, *e.g.* RGB image or stereo audio, or anyway, deeper layer may have several channels in input, a correspondent number of different kernel have to be applied. These kernels stacked together in a *tensor*[6] constitute what is called a *filter*. Now we can introduce another, very important, parameter that defines a convolutional layer:

- *number of filters*. Many filters can be applied to allow us to extract different patterns from the input. The chosen number of filter it is equal to the number of channels that the layer is going to output. Check fig. 2.4 for the details.

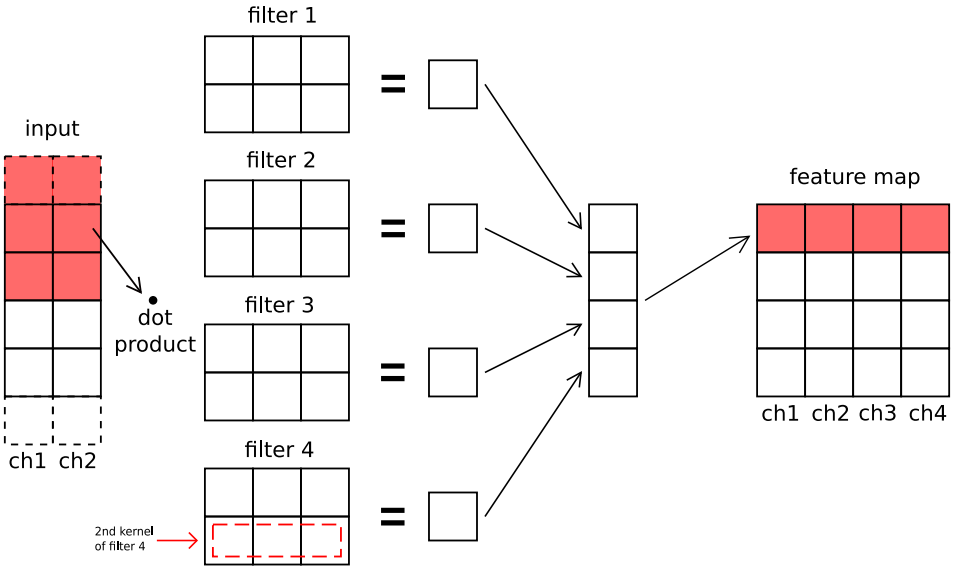


Figure 2.4.: Overview of 1D convolution performed on a 2-channel input with a 4-channel feature map

We discuss now how the third idea combined in neural network, *i.e.* subsampling, is carried out. So far we have seen that both convolution without padding and the stride parameter control the subsampling in the temporal or spatial dimension. Despite that, the most used method to achieve a meaningful subsampling is the so called *pooling*.

Pooling consists in extracting patches through a moving window along temporal or spatial axes and substituting them in the output with single values, which are summary statistic of each patch. This is done for each channel independently. Basically the output of the net at a certain location is substituted with a summary statistic of the nearby outputs. Moreover, to be effective, pooling layer must feature a stride greater than one in order to downsample the input.

[6]Here the generalization of a matrix to n dimensions.

Another important consequence of applying pooling is the invariance of the representation to small translation of the input and the reduction, *i.e.* translation within the kernel size. In addition, subsampling the feature maps let the following convolutional layers to learn high level patterns, indeed pooling increase their unit's receptive field. Finally an implicit advantage is the reduction of the number of parameters which leads to less memory consumpion and faster optimization.

The most common pooling strategy is *max pooling* (Zhou et al. [18]) which consist of extracting the maximum value from the pooling sliding window. This has been proven to be more effective than other pooling approaches, *e.g. average pooling* or L^2 *-norm pooling*.

We conclude this section with an observation on the terminology. Often we call *convolutional layer* the stack of a convolution, a detection and a pooling stages, since these three operation are usually applied together sequentially (Goodfellow et al. [12]). See fig. 2.5.

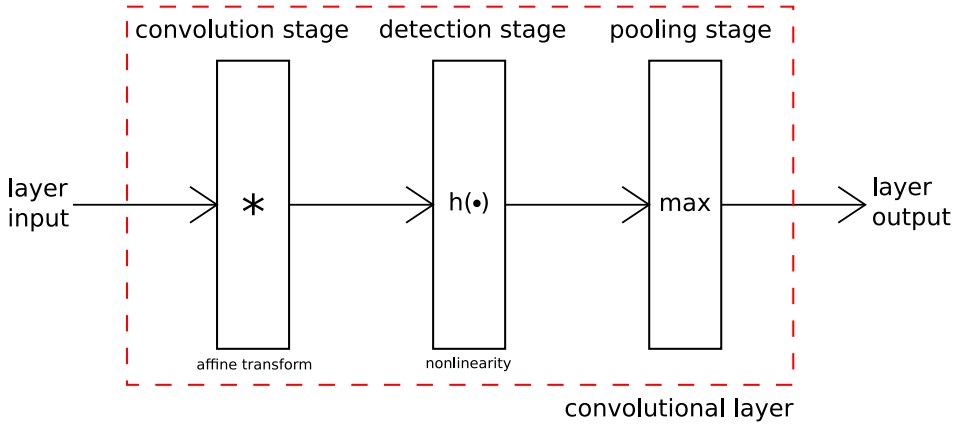


Figure 2.5.: A convolutional layer in complex terminology

2.4. Temporal Convolutional Networks

In Bai et al. [19] *the temporal convolutional network* (TCN) is presented as a family of architectures, for sequences modeling, distinguished by two main characteristics:

- the convolutions in the architecture are causal, meaning that there is no information "leakage" from future to past.
- the architecture can take a sequence of any length and map it to an output sequence of the same length.

While the first point is achieved performing convolutions where an output at time t is convolved only with elements from time t and earlier, the second point is accomplished using a 1D fully-convolutional network (FCN) architecture (Long et al. [20]), where each hidden layer has the same length as the input layer, since

zero padding of length (kernel size - 1) is added to keep subsequent layers the same length as previous ones.

With this basic design, in order to achieve a long effective history size, we need either an extremely deep network or very large filters. To overcome this trade off between longer history size and the extreme deepness of the consequent network, which make the model very difficult and hard to train, TCNs integrate two techniques: *dilated convolutions* and *residual connections*. To explain this two features we follow the Bai et al. [19] exposition.

Dilated convolutions Consider a 1D sequence input $\mathbf{x} \in \mathbb{R}^n$ and a filter $\mathbf{w}$ of length K. The *dilated convolution* is defined as:

$$s_t = (\mathbf{x} *_d \mathbf{w})_t = \sum_{k=0}^{K-1} x_{t-d\cdot k} w_k \tag{2.26}$$

where the multiplier d is the *dilation factor*. Employing this version of the convolution allow us to enable an exponentially large receptive field. Indeed, for our simple stack of convolutional layers, the formula for the this quantity is given by:

$$RF = 1 + \sum_{i=1}^{L} d(i)(K(i) - 1) \tag{2.27}$$

where $\mathbf{d} = (d(1), ..., d(L))$ is the vector of dilation rates and $\mathbf{K} = (K(1), ..., K(L))$ is the vector of kernel lengths. This gives us two ways to augment the receptive field of the TCN: choosing larger filter sizes $\mathbf{K}$ and increase the dilation factors $\mathbf{d}$. The effective history of one of such layers is $(K(i) - 1)d(i)$. Finally, when using dilated convolutions it is common to increase $d(i)$ exponentially with the depth of the network in order be sure that at least one filter hits each input within its effective history. This is clear in fig. 2.6.

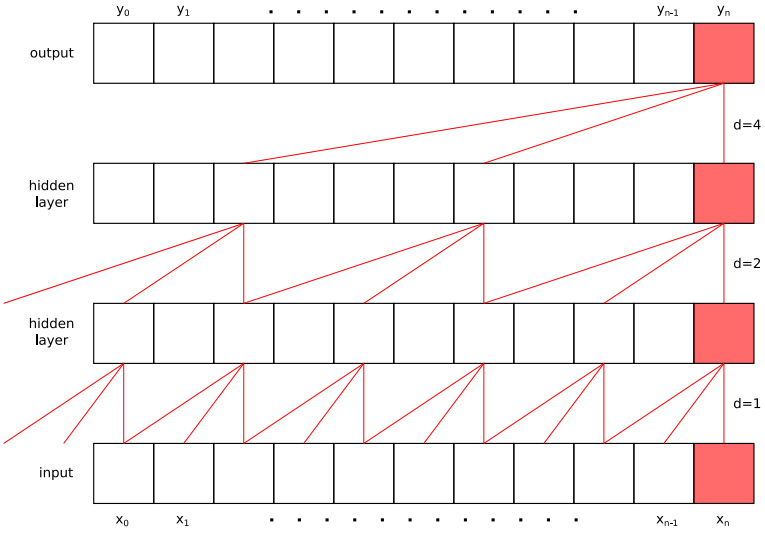


Figure 2.6.: Stack of dilated convolution layers with kernel size=3

Residual connections When deep network are able to start converging, a *degradation* problem has been discovered: with the network depth increasing, accuracy gets saturated and then degrades rapidly. The origin of this phenomenon it is still not clear. It seems not to be caused by overfitting since adding more layers to a deep model leads to higher training error. He et al. [21] (resNet) proposed a method to tackle this issue.

Consider a shallow network and its augmented version obtained by adding more layers on top of it. There exists a solution by contruction to this deeper model: the added layers featuring identity mappings. Obviously in this case the augmented model should not produce higher training error than its shallower counterpart. Experimental results show that our current optimizers are unable to find solutions comparably good with respect to the constructed solution featuring identity mappings. Using *residual connections* we explicitly let the layers to learn modifications to the identity mapping rather than to the entire transformation.

Indeed consider a stack of layers and its overall applied transformation given by $\mathbf{f}(\mathbf{x};\mathbf{w})$. It is considered a *residual block* when the input $\mathbf{x}$ is summed to the transformation within the activation function (see fig. 2.7):

$$\mathbf{z} = h(\mathbf{x} + \mathbf{f}(\mathbf{x};\mathbf{w})) \tag{2.28}$$

The branch that jumps from the input directly into the activation is called *skip connection* or *shortcut connection*.

Since we are summing $\mathbf{x}$ and $\mathbf{f}(\cdot)$ which may have different dimensions, *e.g.* they may have a different number of channels, sometimes a linear projection is needed to match the dimensions. Usually to modify the number of channels 1×1 convolution is applied. Note that the implementation of a skip connection does not need extra parameters (leaving the 1×1 convolution out).

We propose an example of degradation problem and the solution consisting in the usage of residual connections in appendix section A.2.

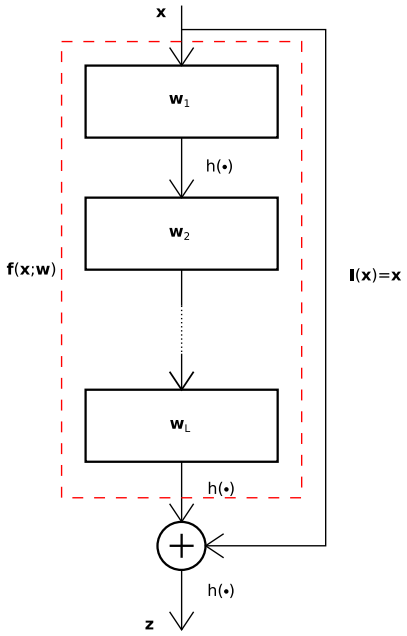


Figure 2.7.: Structure of a residual block, $\mathbf{w} = (\mathbf{w}_1, ..., \mathbf{w}_L)$

To conclude, still following Bai et al. [19] paper, we list some advantages of TCN architecture:

- *Parallelism*. Convolutions can be done in parallel since the same filter is used in each layer. A long input sequence can be processed as a whole without waiting for the predictions relative to previous time steps.
- *Flexible receptive field size*. We can change the receptive fields in multiple ways controlling the number of layers, the dilation factor and the kernel size.
- *Stable gradients*. The backpropagation path is different from the temporal direction of the sequence. This let TCNs to avoid the problem of exploding/vanishing gradients.
- *Low memory requirement for training*. TCN exploit the parameter sharing of the convolutional layers without adding extra parameters.
- *Variable length inputs*. This is also an advantage due to the usage of convolutional layers.

One notable disadvantage is the fact that when we want to transfer the model to a different domain often we will have to change the hyperparameter to satisfy the request of a larger receptive field.

A final clarification: at the beginning of this section we have defined the TCN's kernels to be causal. Actually Lemaire et al. [22] have proven that TCNs featuring non-causal filters (ncTCN) are able to achieve better results in the task of speech and music detection. Obviously this is not possible in all the applications since to compute the convolution output at time t we need samples from instants $t + \tilde{k}$.

3. Models under investigation

In this chapter we detail the two models under investigation with the goal of discriminating between advertisement and other type of radio contents. Hence for both models the framework is the binary classification *advertisement* versus *other*, *i.e.* music and speech, though the second model is more complicated in that it features an energy extractor front-end. Both models are inspired by networks in literature. We highlight the original features and our additions subdividing the sections in two dedicated paragraphs.

3.1. SoundNet-based network

The original architecture We called this first model "SoundNet-based" since it is based on the work of Aytar et al. [9], which dates back to 2016.

While the fields of object recognition, speech recognition, machine translation have been revolutionized by the emergence of massive labeled datasets and learned deep representations, this is not the case as far as natural sound understanding is concerned, which is the goal of the SoundNet project.

The SoundNet utilizes a teacher-student architecture to leverage the natural synchronization between vision and sound in order to learn acoustic representation using two-million unlabeled videos. Unlabeled videos have the advantage that they can be acquired economically at a massive scale and they contain useful signals about sounds. Strong progress in computer vision has enabled machines to recognize scenes and object in images and videos with good accuracy.

The SoundNet architecture transfer this discriminative visual power into sound using unlabeled video as bridge. This model is based on a deep convolutional network that learns directly on raw audio waveforms, which is trained by transferring knowledge from vision into sound. The overall structure can be seen in fig. 3.1.

In their work the authors chose to use videos from *Flickr*[1], an online community where users share images and videos, because they are natural, not professionally edited, short clips recorded with audio in everyday situations. Over two million videos have been downloaded querying for popular tags and dictionary words. The resulting dataset features over one year of continuous natural sound and video. The clips' lengths are variable between few seconds and several minutes. Since they aim to process sound waves in the raw the only post-processing required done on the videos was to convert sound to *mp3* format, fix the sampling rate to 22050 Hz, and transform to a single channel audio. This was done to facilitate the storage and the usage during training of this very large dataset. Finally

[1] www.flickr.com

they scaled the waveform to be in the interval $[-256, 256]$.

As previously mentioned, the model itself consists of a deep convolutional architecture, as far as the sound part of the network is concerned. More specifically a series of 1D convolutions are employed, followed by nonlinearities, *i.e.* ReLU layers, in order to process sound. Between each of these layers a batch-normalization stage is employed to speed-up the training phase.

The convolutional framework had been chosen for a couple of reasons.

Firstly, like for images, the network is needed to be invariant to translations. This necessity leads also to the advantages of reducing the number of parameters to learn and of increasing efficiency.

Secondly, stacking more convolutional layers enables the network to detect higher-level concepts through a series of lower-level detectors. Since the authors desired to work with audio samples which can vary in temporal length, the network must be able to handle this kind of inputs.

The solution implemented is a fully convolutional network, *i.e.* a network composed by convolutional and pooling layers only.

The output layer as well has to be designed to work with variable lengths inputs. Unlike us, they chose not to use a global pooling strategy to squeeze the temporal dimension, that is not to downsample a variable length input to a fixed size vector which may be connected, through a dense layer, to the classification neurons. Instead, since also video is variable length, they use a convolutional output layer to produce an output over multiple timesteps in video. Such a strategy avoids the discard of potential useful informations for high-level representations. Leveraging the large amount of videos in the dataset makes it feasible to use deep architectures without significant overfitting.

The authors experimented with both five-layers and eight-layers networks with the latter which performed better.

We do not deepen into the visual part of the network since it is not useful for our purposes. We simply outline the fact that the network is trained optimizing the well known *Kullback-Leibler divercence* between the visual and the sound outputs, since they can be interpreted as probability distributions over the 1401 possible categories.

Moreover, given the fact that not all of the audio categories that they wish to recognize appear in visual models, the authors trained a linear SVM on a small amount of labeled sound data for the concepts of interest. The SoundNet architecture makes it simple to pick a layer in the network to use its activations as features in order to train the SVM classifier.

The authors evaluated this classifier on three different acoustic scene classification datasets (DCASE, ESC-50 and ESC-10) outperforming existing state-of-the-art methods by around 10% on everyone of the three with respectively 88%, 74.2% and 92.2% accuracy. Moreover they proved the semantic relevance of the features extracted from the 7-th convolutional layer using the t-SNE embedding.

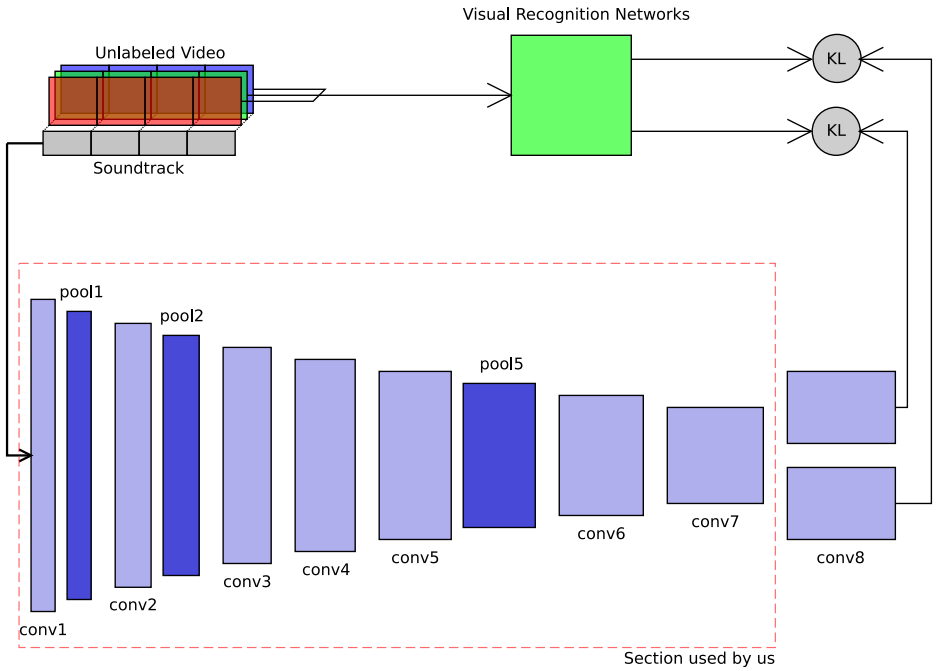


Figure 3.1.: Original SoundNet architecture

Our modified architecture Given the good results and the semantic relevance of the features obtained from this network we decided to mimic its acoustic-dedicated architecture.

As we have seen the SoundNet is dedicated to natural sound classification, or better, to extract relevant features then used for the classification task via SVM. This different dedication prevented us from simply train a dense classification network with the features extracted from the 7-th layer of the SoundNet as inputs, *i.e.* to perform *transfer learning* between natural sounds' domain to ours. Indeed we decided to keep unaltered the convolutional structure and to add on top of that a global pooling layer, followed by a dense layer to perform classification. In fact we want our classifier to be totally based on a neural network approach and do not rely on other methods like SVMs.

This altered structure is to be completely trained from scratch on our dataset. We can look at the described overall architecture in fig. 3.2, while for an insight of the 1D CNN stage see table 3.1. Between each two consecutive layers a batch-normalization layer is inserted which operates along the features axis. It is inserted after the convolutional layer activation and before the activation function. After the batch-normalization, when it is scheduled, a max-pooling layer is put in the pipeline. The global-max-pooling at the end shrinks the whole time axis to unique samples to obtain the 1024 features from the 7-th layer. These features are then connected through a dense layer to a single output unit, provided with a logistic sigmoid activation, as usual for binary classification.

This model is composed by a total of 2,879,384 parameters, of which 4,064 non-trainable, due to batch-normalization layers.

As Aytar et al. [9] have explored in their paper defining the SoundNet, this

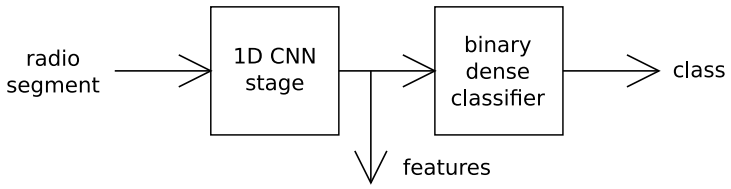


Figure 3.2.: Overall structure of the SoundNet-based model

architecture is easily suitable to the removal or insertion of one or more convolutional blocks in order to extract fewer or more features, depending on the need.

	1st layer	2nd layer	3rd layer	4th layer	5th layer	6th layer	7th layer
filters num	16	32	64	128	256	512	1024
kernel size	64	32	16	8	4	4	4
kernel stride	2	2	2	2	2	2	2
padding	32	16	8	4	2	2	2
activation	ReLu	ReLu	ReLu	ReLu	ReLu	ReLu	ReLu
pool size	8	8	no	no	4	no	GlobMax
pool stride	8	8	no	no	4	no	-

Table 3.1.: Architecture of the 1D CNN stage

3.2. CNN-TCN-based network

The second model that we propose follows a different idea: instead of feeding the network with the raw audio data, we give as input to a CNN-dense classifier the music and speech energies extracted by a CNN-TCN front-end network.

This approach is based on the hypothesis that music and speech energies, and their time course, might encode sufficient information for our advertisement detection task. Indeed listening to advertisement samples suggests that they are usually composed of music and speech segments alternating in time. This fact should let such energy-based network to distinguish between advertisement and music, *i.e.* music does not contain speech, and between advertisement and speech, *i.e.* speech does not contain music.

Nonetheless the situation might be more complicated. That is because for speech we mean also an audio sample in which the focus is clearly on the spoken parts, but music presence is allowed. In fact often in radio broadcasts, expecially the ones focused on music, the radio hosts speak over a music background. This instance is considered speech in our dataset. While sometimes, during broadcasts, such a music background has a rather high volume, this is not the case for the majority of the advertising. Indeed, for the sake of clarity of the advertising message, commercials usually show an alternatig structure in which the most important spoken parts clearly emerge from a quieter music background.

We summarize the observations in support to this claim, about the sufficiency of music and speech energies to carry out our classification task, in table 3.2.

	Foreground music	Only speech	Speech+background music
Advertisement	Presence of speech	Presence of music	Alternating structure with clear emerging speech-only parts

Table 3.2.: Energy-based features that should discriminate between advertising and other contents

A comparison, in terms of performances, with the SoundNet-based model will justify or deny our hypothesis.

In fig. 3.3 we can take a look to the overall pipeline. The total number of parameters of this network is 1,924,299.

In the following we deepen into both the front-end energies-extractor and the back-end classifier.

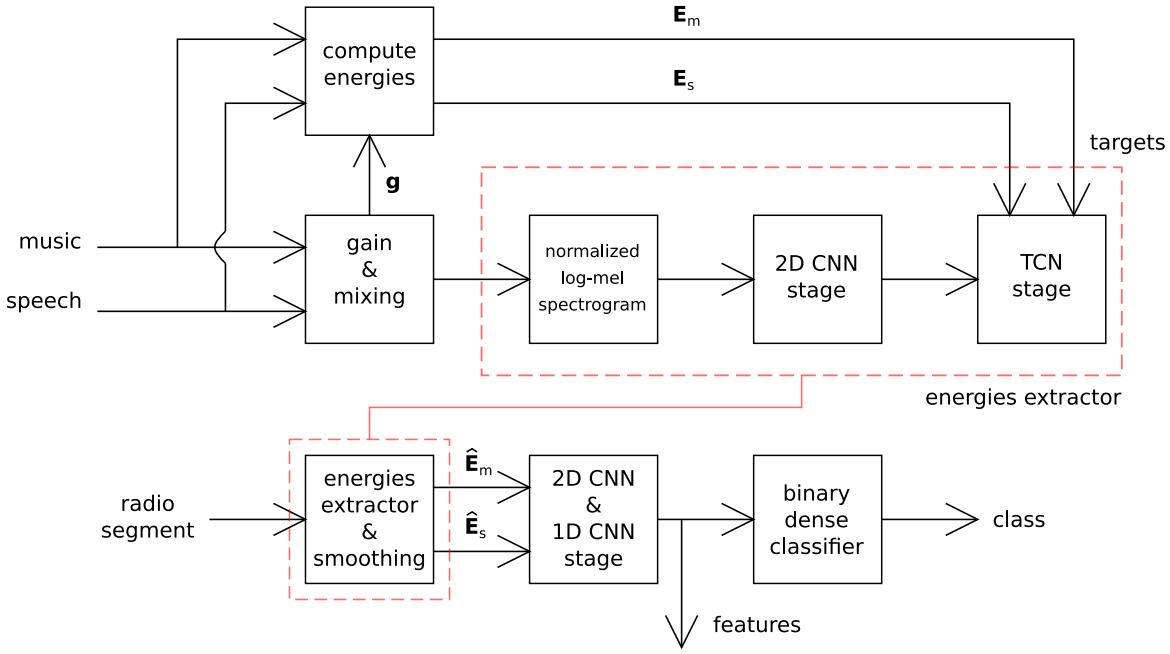


Figure 3.3.: Overall structure of the CNN-TCN based model

3.2.1. CNN-TCN energies extractor

The original architecture The goal of this section of the network is to extract, from the radio broadcasted samples, robust estimates of normalized music and speech energies.

Given the similarity between our and their tasks we employ the architecture proposed by Meléndez-Catalán [10] in 2020. In this paper the authors aim to estimate the *relative music loudness* of an audio source, *i.e.* to divide an audio stream in segments of three classes: foreground music, background music and no music (for instance speech).

This is done employing a Temporal Convolutional Network (see section 2.4) which works, and that is the novelty introduced by the paper, on top of a CNN

front-end. They expect the CNN front-end to work as a feature extraction strategy to achieve a more efficient usage of the network's parameters; while TCNs are a type of architecture with the ability to model temporal context, which is a fundamental characteristic when analyzing temporally correlated signals such as music.

In the case of simple music detection the foreground and background music are not separated in two classes. The detection of background music, distinguished by foreground music, has become relevant in the literature since it is an harder problem. This fact gave birth to the relative music loudness challenge.

Clearly, for our goal, it is essential to implement a network which is able to discriminate between foreground and background music, as we have illustrated previously. Indeed the presence of background music in a sample is a simple index of the fact that we could be in presence of an advertising.

Back to the network structure, the authors claimed that the insertion of the CNN front-end could lead to improvements in the classification performance. To prove that, they compared this CNN+TCN network versus a TCN-only network. Beyond performances, what is sure is the fact that this insertion of a CNN front-end with a proper number of channels yields a model with less parameters than the TCN-only network.

As network inputs the authors use audio with a sample rate of 8000 Hz and 16 bits precision, normalized to have a maximum amplitude value of 1. From the audio data they then compute the power spectrogram with a Hanning window of length 512 samples, *i.e.* 64 ms, and hop size of 128 samples, *i.e.* 16 ms. A Mel filter bank with 128 filters is applied to obtain the Mel-spectrogram. This filter bank converts the usual Hertz scale of frequencies to the Mel scale, which is a perceptual scale of pitches judged by listeners to be equal in distance from one another. This is motivated by the fact that humans do not perceive frequencies on a linear scale. For instance we can easily tell the difference between two low frequencies sounds, but increasing the frequency the same difference becomes harder to tell. Such change of scale allows a more granular sampling of the low frequencies in which sounds humans are used to, *e.g.* voices, are located. Finally the Mel-spectrogram is converted to logarithmic scale and min-max normalized to the interval $[0, 1]$. Since the audio samples have a fixed length of 10 s the resulting inputs are matrices with shape 128×625.

The samples utilized are gathered in the OpenBMAT dataset, composed of over 27 hours of TV-broadcasted audio from various countries distributed over 1647 one-minute long excerpts.

These inputs are fed to the CNN front-end which consists of a stack of 7 blocks that comprehend: (1) a 2D convolutional layer with N_{cnn} 3×3 filters and ReLU activation function, (2) a spatial dropout layer with a dropout rate dr and (3) a max-pooling layer. The max-pooling layer is applied only to the frequency axis reducing its dimensionality by a factor of two until it is equal to one. The TCN reads the output of the CNN as N_{cnn} scalar temporal sequences and models their evolution.

Speaking of which the TCN model is composed by a stack of 6 *residual blocks* (see section 2.4) featuring also spatial-dropout layers with dropout rate again dr.

All the 1D convolutional layers have N_{tcn} non-causal filters of length L. The structures of the convolutional blocks which composes the CNN front-end and of the residual blocks are shown in fig. 3.4.

At the end of the pipeline a 1D convolutional layer with kernel length equal to 1 allows to sum the skip connections coming from the various blocks and outputs, thanks a softmax activation, a probability distribution over the three class, for each time frame.

At this point each time frame is assigned to the class with higher probability. Since classifying at time frame level is very precise, but makes the algorithm susceptible to produce short erraneous segments, a smoothing window is applied. The time frame, on which the window is centered, is assigned to the most represented class in such interval.

Notice that the whole network leaves the time axis invariated in terms of dimension, *i.e.* there is no time downsampling. Such a feature perfectly fits our energy extractor's needs.

During training, firstly, a grid search over hyperparameters was performed, which yielded as best choices: $N_{cnn} = 32$, $N_{tcn} = 16$, $L = 7$ and $dr = 0.15$. Their training results showed also how the presence of a CNN front-end is effective in the task of relative music loudness estimation: in terms of accuracy the CNN+TCN network gains more or less 4% over the TCN-only model reaching the 90%.

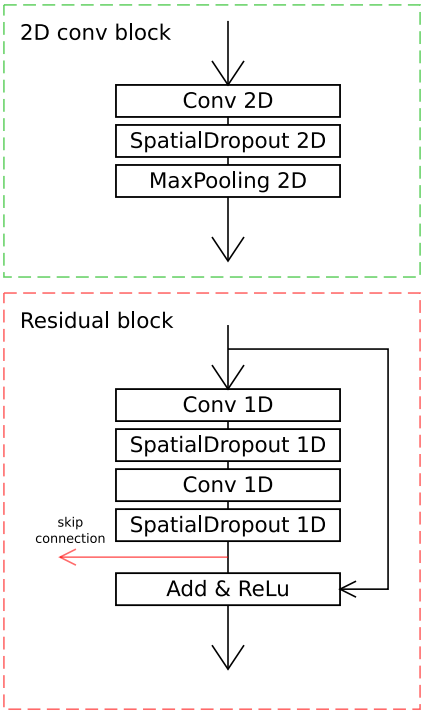


Figure 3.4.: Structure of the 2D convolutional and of the residual blocks

Our modified architecture These encouraging results obtained by Meléndez-Catalán [10] in the task of relative loudness estimation pushed us to employ a

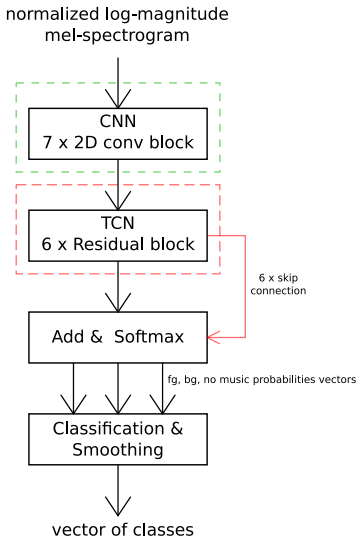
similar architecture, comprising the CNN front-end, for our energies extractor. Indeed the good performances (~85% precision and recall) obtained by the authors in recognizing also background music makes us suppose that a similar architecture may yield a good estimate of music energy, even when it appears in background.

What we have to do is basically convert the paper's 3-classes classification setting to our 2-channels regression framework, *i.e.* one channel represents music energy and the other the speech one. In fact their proposed network features an output that consists of a probability distribution over the three classes for each time-frame in the input, *i.e.* an $N_{frame} \times 3$ matrix, obtained employing soft-max activation functions. Since our desired output is an estimate of normalized music and speech energies, for each time-frame, it has the shape $N_{frames} \times 2$.

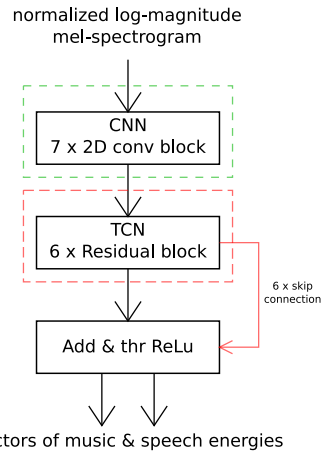
We noticed that the energies normalized in the interval $[0, 1]$ always presented almost-zero positive values. To make better use of the ReLU activation function we extended the normalization range to the interval $[0, 100]$. Therefore, since the output energies are constrained to be positive and less than 100, we apply a thresholded ReLU activation function at each time frame of the 2-channels output.

As far as the inputs are concerned, we employ the ADV-MUSAN dataset which can be created synthetically from the MUSAN corpus, as described in section 4.2. Thus the inputs to the network are mini-batches of normalized log-magnitude mel-spectrogram tensors with dimension $128 \times N_{frames} \times 1$ where the initial single-channel is made explicit. Notice that, contrary to the Meléndez-Catalán [10] case, our inputs are variable in length.

Insights on the utilized architecture can be found, compared to the original network, in fig. 3.5b, along with the blocks' characteristics showed in table 3.3 and table 3.4.



(a) Structure of the original CNN-TCN classifier by Meléndez-Catalán [10]



(b) Structure of the CNN-TCN energies extractor

	1st layer	2nd layer	3rd layer
type	Conv 2D	SpatialDropout 2D	MaxPooling 2D
filters num	32	-	-
kernel size	32×32	-	-
kernel stride	1×1	-	-
padding	"same"	-	-
activation	ReLu	-	-
pool size	-	-	2×1
pool stride	-	-	2×1
dropout	-	dr	-

Table 3.3.: Architecture of a 2D convolutional block

	1st layer	2nd layer	3rd layer	4th layer
type	Conv 1D	SpatialDropout 1D	Conv 1D	SpatialDropout 1D
filters num	16	-	16	-
kernel size	7	-	7	-
kernel stride	1	-	1	-
dilation[i]	2^{b-1}	-	2^{b-1}	-
padding	"same"	-	"same"	-
activation	ReLu	-	linear	-
dropout	-	dr	-	dr

[i] Where $b = 1, ..., 7$ is the index of the current block.

Table 3.4.: Architecture of a residual block

Notice that we kept invariated the design parameters with respect to best setting exposed in the Meléndez-Catalán [10] paper, except for the dropout rate dr. Substituting this choice of parameters in eq. (2.27) yields us a *receptive field* which is 379 time-frames long, *i.e.* approximately 4 seconds. As far as the dropout rate dr is concerned we realized, after testing with few values, that the best choice was to completely remove the spatial dropout layers, *i.e.* $dr = 0$. This is motivated by the fact that our synthetic dataset is arbitrary large and various since it is generated online during the training. Therefore there is no need for such a strong regularizer to mitigate overfitting.

In total the energy extractor is composed by 82,850 trainable parameters.

3.2.2. CNN-dense classifier

The CNN-dense classifier is trained on our MUSPAD dataset. One of the advantages of the CNN-TCN method is that we can storage the sole music and speech energies yielded by the CNN-TCN energy extractor from a segment, instead of the whole audio in raw, saving a great quantity of memory space ($\sim$ 100 times) and working with smaller inputs.

These samples of energies are then smoothed using a moving average approach, before being fed to the classifier network. Its optimal window's length is obtained through the cross-validation approach with 4 folds.

The resulting $N_{frames} \times 2 \times 1$ tensor of energies meets a 2D convolutional block which aim to mix the music and the speech dimensions in a unique one.

Indeed keeping only the valid outputs of the 2D convolution yields a column $\tilde{N}_{frames} \times 1 \times 1$ vector. Basically what we are doing here is performing several linear combinations of the two energies, dependent on the filters' weights, and saving these information in the correspondent feature maps. Looking at table 3.5 notice that we chose not to skip any time frame, *i.e.* the kernel stride is set to 1. The time length of the 2D kernel is 16. We made this choice by progressively increasing the kernel length and noticing that, after 16, the performance of the classifier, regarding validation scores, were not getting any better at the price of more training parameters to be optimized.

Another choice could have been to elaborate the two energies separatedly, considering them as two different inputs. Since before the final dense classification layer the informations extrapolated from the two energies must, in any case, be merged, we prefer to carry this out immediately in the front-end. This approach let us employ less training parameters. Moreover, the already mentioned stride equal to 1 and, in addition, the large number of applied filters, *i.e.* 128, makes us confident that none or a little information might be lost.

A series of three 1D convolutional and pooling layers are then cascaded, until a final dense layer, which performs the binary classification.

This layer structure clearly resembles some aspect of the SoundNet network. Indeed we perform an upsampling in the number of filters to extract higher level feature maps; meanwhile we downsample in time domain employing "valid" padding plus convolution and pooling strides greater than one until a final global-max pooling layer to definitely suppress the time axis. Notice also that the last convolution layer extract 1024 feature maps, exactly like the SoundNet. ReLU activation functions are utilized in all layers, except for the last dense one which, obviously, presents a sigmoidal output. We think that implementing the CNN classifier in this similar fashion is likely to give acceptable results.

To conclude, the total number of parameters of this block of the network is 1,842,049.

	1st layer	2nd layer	3rd layer	4th layer	5th layer
filters num	128	256	512	1024	Dense
kernel size	16×2	8	4	2	-
kernel stride	1×1	2	2	2	-
padding	"valid"	"valid"	"valid"	"valid"	-
activation	ReLu	ReLu	ReLu	ReLu	Sigmoid
pool size	-	4	2	GlobalMax	-
pool stride	-	2	1	-	-

Table 3.5.: Architecture of the CNN classifier

4. Datasets

4.1. *MUSPAD*: the MUsic, SPeech and ADvertisement dataset

A fundamental part of the Thesis has been the fabrication of a suitable dataset. Indeed no such a dataset was found already available. The requirements on data for our binary classification task were very specific. We needed audio samples from radio broadcasts labeled as "advertisement" or "not advertisement". Since it was easy to carry out, we enriched the taxonomy distinguishing between music and speech content, within the already existing "not advertisement" label.

The labeling task has been carried out by the author during the period from the end of June and the beginning of September 2021. Tipically one hour-long radio streams at 22050Hz have been recorded, from 14 renowned italian radio stations, through online openly available links, using *VLC*[1] as recorder. The labeling procedure has been performed using the software *Audacity*[2] resulting in a text file containing timestamps and relative classes associated to the corresponding segment in the recorded *mp3* audio file. Given the small amount of labels to assign and their relatively easy identifiability, typically, the labelling of an hour of raw audio took more or less 50 minutes. The effort strongly depended on the radio station, particularly on the number and duration of the aired advertisings. All the labeling process has been carried out with the maximum possible care, precision in cutting and consistency in labeling.

The assigned labels are:

- *adv* for advertisements. For a human being, advertisement fragments are, almost always, clearly recognizable from the context due to some reasons: (1) the topics covered, (2) the fact that they are usually clustered and sometimes well separated by characteristic jingles or sounds. The temporal course of the audio usually presents only-speech, only-music or both mixed alternating sections. In general the structure may be very varied.

- *speech*. A section is labeled as speech when the focus is clearly on the spoken part. For instance a piece of a radio broadcast in which the host is speaking above a background music is classified as *speech*. Clear examples of speeches are radio news, weather forecasts and traffic forecasts besides any other themed radio program.

[1] VideoLan, 2006. VLC media player, Available at: https://www.videolan.org/vlc/index.html.

[2] Audacity® software is copyright © 1999-2021 Audacity Team. The name Audacity® is a registered trademark.

- *music*. Only complete songs have been timestamped and classified as music. Uncompleted music chunks or other music-like interludes were ignored.

Advertisement and music pieces has always been kept unsegmented. This means that the dataset contains single advertisements and single songs as an unique piece; while in few cases some very long seamless speeches has been divided in more segments.

One important aspect to take into account is the fact that the same specific advertising may appear more than once in the dataset. That is because recording the streams during a rather short period of time makes the chances to encounter the same advertisement, possibly on different radio stations, relatively high. As better explained below, for the advertising clustering purpose we randomly selected approximately one hundred advertisement samples and we labeled them with specific meaningful labels, which summarise their brand and their content. In this selection the worst case was an advertisement repeated four times, in different radio stations. This specific commercial break was clearly part of a summer advertising campaign. We think that this issue does not constitute a problem for training a meaningful model. Conversely it establishes a natural form of data augmentation given the fact that such samples surely are not identical, due to temporal segmentation differences or different streaming quality and filtering between radio stations.

	N of samples	Durations
Advertisement	830	5h 21m 03s
Speech	561	18h 00m 56s
Music	495	1d 1h 47m 30s
Other	1056	1d 19h 48m 26s
Total	1886	2d 1h 09m 29s

Table 4.1.: Complete dataset specifications, Other = Speech + Music

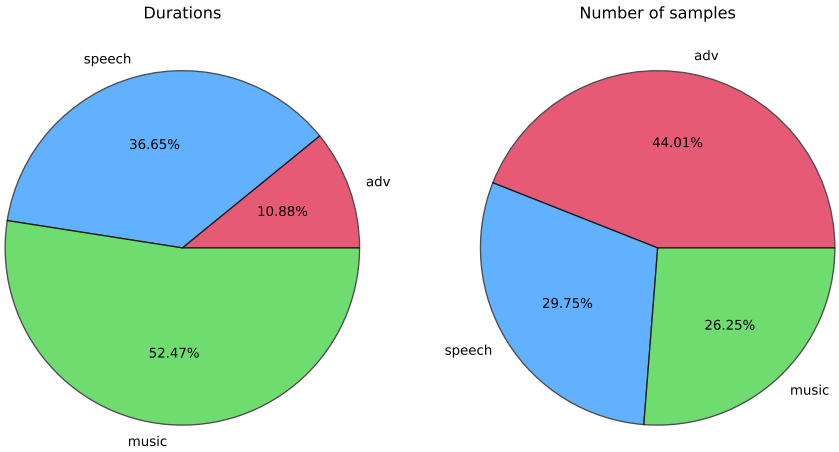


Figure 4.1.: Pie charts of duration and number of samples

The composition of the *MUSPAD* dataset is visualized in table 4.1 and in the pie chart fig. 4.1, according both for the number of samples and for their duration. The dataset is almost naturally balanced as far as the number of samples is concerned, *i.e.* the number of advertisement is almost equal to the number of speeches and songs together. These two quantities differs by approximately the 6%. This fact is interesting since no effort was spent to balance the dataset during the labeling procedure. We limited ourself to classify all the segments in the recorded streams, and then, at the end of the labeling task, we analyzed the dataset composition.

On the contrary a clear asymmetry is present in terms of durations. Indeed the advertisements obviously last less since the commercial message must be transmitted in the shortest possible time to be effective and less expensive. Therefore the total advertising duration only slightly exceeds 10% of the dataset time length. While in term of number of samples music and speech are balanced this is not the case as far as the total duration is concerned. This fact is motivated by the length of the songs that are, in average, longer than spoken parts, and by the dedication, of most of the radio stations recorded, to music broadcasting, particularly during summertime.

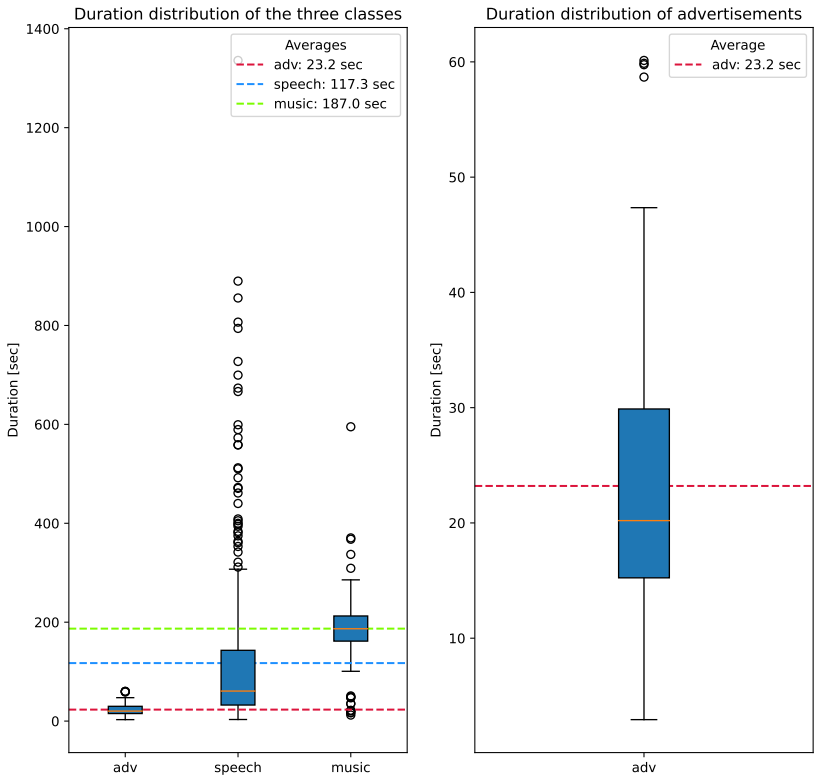


Figure 4.2.: Duration distributions of the complete dataset

In fig. 4.3 we can see the durations exposed in the form of a boxplot. On the right we have the detail of the advertisements durations. We notice at first glance the sorting, based on durations, which shows music in the first place, followed by speech and advertisement. The second annotation is about the dispersion of the samples' durations.

The speech class features several outliers and durations spreaded on a wider range. That is because spoken parts, except perhaps for the news bulletins or weather and traffic forecasts, may vary a lot depending not only on radio stations, but also on the hour of transmission.

The argument is different for music class. Its interval of durations is narrower because songs occupy well definited time slots during the broadcast. Indeed often the songs are edited to fit radio transmission requirements.

A similar discussion can be made about advertisements which must fit in not only fixed but also shorter time slots. From that the even shorter range with a maximum duration of about 1 minute and minimum of less than 5 seconds.

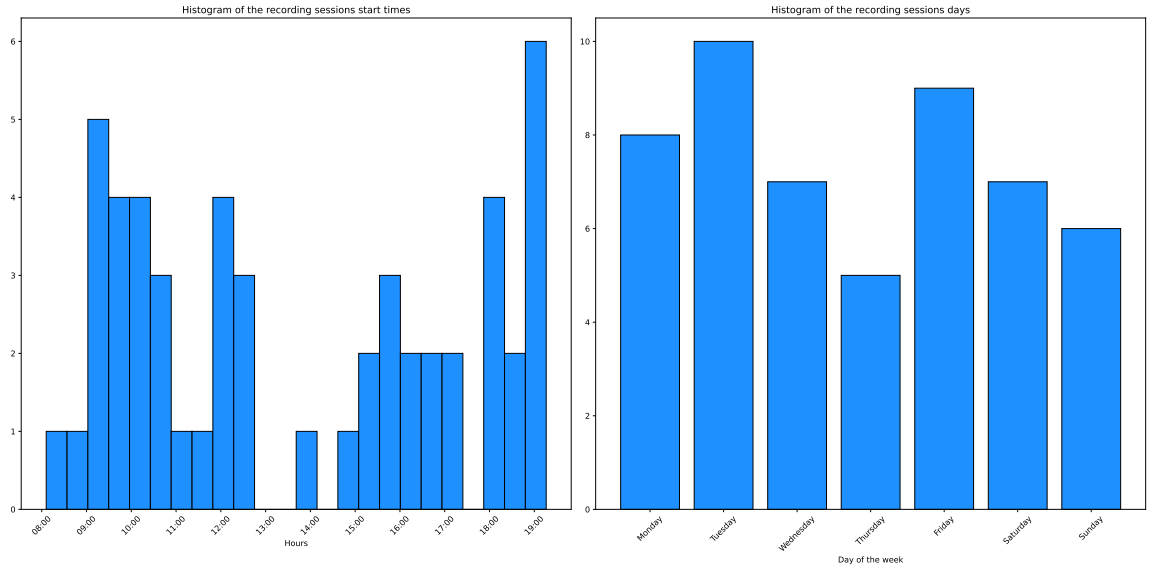


Figure 4.3.: Histograms of the recording sessions distribution in time

As previously mentioned the creation of the dataset has been undertaken during summertime only. Clearly this fact might bias the representativeness of the dataset since it does not contain the whole spectrum of possible advertisements.

To reinforce the generality of the dataset and to employ it in commercial applications it might be recommended to set up other recording and labeling campaigns during different periods of the year. This is motivated by the fact that it is possible that advertising campaigns, specifically aimed to a period of the year, present peculiar audio features. Since we could split music and speech samples to augment our dataset, or, however, music and speech samples are easily available, *e.g.* MUSAN corpus, these additional campaigns can be done taking into account only the advertising, reducing remarkably the time spent in labeling.

Moreover no stream has been recorded during night time. This in our opinion does not constitute a problem since often the night broadcasts are simply recording

of what has been aired during the daytime. As we can see in fig. 4.3 the daytime hours are well covered. Perhaps it is recommandable to fill the gaps in lunch time and in the interval between 7 p.m and 9 p.m.

As far as the weekly coverage is concerned, it seems balanced and every day has at least 5 hours of recorded streams.

Further additions To assess the model's robustness and, moreover, to understand its applicability to different domains, we recorded and labeled a few advertisements from a British radio station. The procedure employed is the same applied for the rest of the dataset, but this time, since we were interested in only the advertising, we recorded only these particular contents. A bunch of 119 commercial breaks were extracted. Since the recording session has been carried out on a single radio station and in a short period of time, *i.e.* within a week, it is likely to find many instances of the same advertising. This fact should not compromise the analysis, given that we are using these samples for testing the models in a realistic situation and not to train them.

The other addition we made to the dataset consists of the timestamps and specific labels obtained for some of the advertisements already present in the corpus. For specific label we mean not just a generic "adv", but a more meaningful tag which is able to uniquely identify the advertisements among the other. In our case such a label contains the brand name and a brief summary of the contents of the advertising. This task has been carried out relistening and specifically relabeling 110 advertisments randomly selected from the MUSPAD dataset. A quantity of 92 samples presents unique labels, *i.e.* 18 samples are duplicates.

Finally we deepen in table 4.2, fig. 4.4 and fig. 4.5 the analyses already carried out, for each radio station.

Radio station	N of samples	Tot duration [sec]
R101Diretta	188	13367
RDS100%GrandiSuccessi	166	13591
RMCRadioMonteCarlo	111	10501
RTL102.5	145	14000
RTLRadiofreccia	143	17152
Radio105	151	12694
Radio24	206	16010
RadioBruno	63	6914
RadioDeejay	166	13074
RadioKissKiss	142	14643
RaiRadio1	111	14465
RaiRadio2	95	12193
RaiRadio3	19	3489
VirginRadio	180	14878

Table 4.2.: Per radio station number of samples and duration

Figure 4.4.: Per radio station pie charts of duration

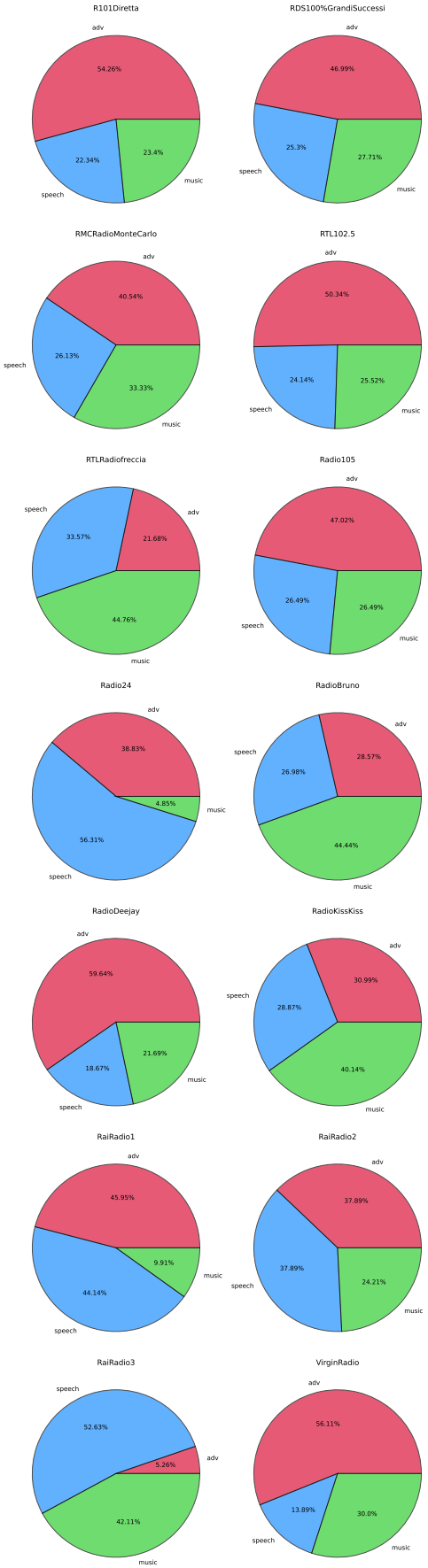


Figure 4.5.: Per radio station pie charts of number of samples

4.2. *ADV-MUSAN*: a dataset of synthetically created advertisement-like samples

In the task of music and speech energies extraction, performed by the CNN-TCN feature extractor network described in section 3.2.1, we employ a synthetic dataset extracted from the well-known *MUSAN* corpus (Snyder et al. [23]).

The idea is to mimic the characteristics of an italian radio-broadcasted advertisement and contemporary leveraging the large amount of samples present in the MUSAN dataset.

To achieve that we take into account only modern music and italian and spanish speeches, discarding the rest. Since the task of the energy extractor is just to extract the music and speech energies of a sample, and not direct advertising recognition, this seems a good approach. Moreover, in the MUSAN corpus, we have at our disposal music and speech samples, not mixed, from which we can separatedly compute the energies to assemble the target variables for our supervised task. The sample-rate of the dataset is 16kHz and all the audio tracks contain values in the interval $[-1, 1]$.

Here we list all the instructions to construct the dataset:

1. Merge all the selected music tracks in a unique stream, do the same for the speech tracks in another unique stream.

2. Extrapolate a segment $\mathbf{x}_n^{(m)}$ from the music stream and one $\mathbf{x}_n^{(s)}$ from the speech stream. The duration of the two segments is the same, randomly extracted from a truncated normal distribution on the interval $[10, 25]$ sec, mean 15 and deviation 5. The two segment's starting points are different and uniformly generated accordingly to the lengths of music and speech streams. The choice of the length was imposed by hardware memory requirements.

3. Compute the energies of the two signals, $\mathbf{x}_n^{(m)}$ and $\mathbf{x}_n^{(s)}$, at each non-overlapping rectangular frame F of duration 10ms: $E_n^{(m)}(F) = \sum_{t \in frame(F)} x_n^{(m)}(t)^2$ and $E_n^{(s)}(F) = \sum_{t \in frame(F)} x_n^{(s)}(t)^2$.

4. Since the speech stream is taken from audio-books the signal level is rather constant. We aim to inject some perturbation, in order to simulate a radio broadcasting, by multiplying the two segments for a gain factor $g_n(F)$ defined for each frame. The gain factor is generated from a uniform distribution over the interval $[0, 1]$ for each frame F. To obtain a smooth transitions between subsequent frames a moving average filter, with a random uniform window size, is applied to the gain vector $\mathbf{g}_n = (g_n(1), ..., g_n(F), ...)$. Then we sum the two segments to obtain the mixed input: $x_n(t) = g_n(F)x_n^{(m)}(t) + [1 - g_n(F)]x_n^{(s)}(t)$ for $t \in frame(F)$.

5. Modify the energies of the two segment to take into account the gain factor and normalize them to the interval $[0, 100]$, *i.e.* divide them for the frame length fl and multiply by 100: $\tilde{E}_n^{(m)}(F) = 100 \times [g_n(F)^2 E_n^{(m)}(F)]/fl$ and

$\tilde{E}_n^{(s)}(F) = 100 \times [(1 - g_n(F))^2 E_n^{(s)}(F)]/fl$. See an example in fig. 4.6. Then stack the two normalized energy vectors in one matrix having two columns: $\mathbf{E}_n = (\tilde{\mathbf{E}}_n^{(m)}, \tilde{\mathbf{E}}_n^{(s)})^T$. More on the choice of the interval $[0, 100]$ in section 3.2.1.

6. Compute the $[0, 1]$-normalized log10-mel-spectrum of the mixed input signal $\mathbf{x}_n = (x_n(1), ..., x_n(t), ...)$ with a 128 filter-bank and STFT's hop-size correspondent to 10ms. In this way the resulting matrix $\mathbf{X}_n$ has the same temporal dimension of the matrix $\mathbf{E}_n$ containing the energies of the musical and speech contributions.

7. Return the couple $(\mathbf{X}_n, \mathbf{E}_n)$ composed by the normalized log10-mel-spectrum of the mixed signal as input and the matrix of normalized music and speech energies as target.

8. Repeat the points, from 2 to 7, N number of times to obtain as many samples as desired.

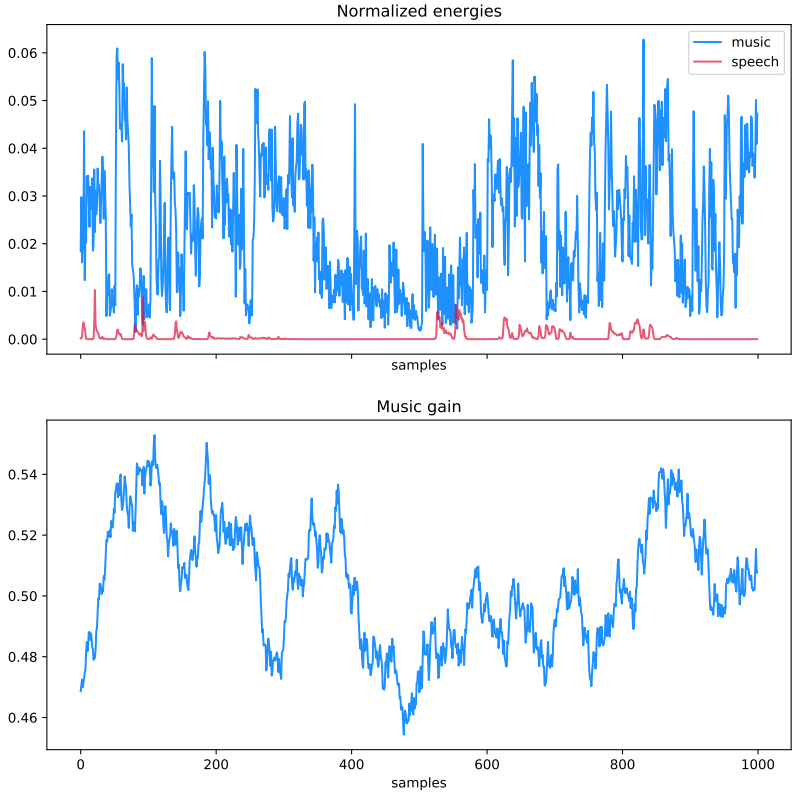


Figure 4.6.: Energies and correspondent gain factor of a mixed sample

5. Experimental results

In this chapter we analyze the experimental results obtained implementing the two models described in chapter 3. The structure of the two chapters presents the same sections' names for the sake of clarity plus the extra section about robustness assessment. We compare the results obtained in the specific last section.

The training instances have been implemented using the deep learning API *Keras* (Chollet et al. [24]) written in *Python*, running on top of the machine learning platform *Tensorflow* (Martı n Abadi et al. [25]). The code has been executed on *Google Colaboratory Pro*[1] cloud-available GPUs and TPUs. We utilized the *Adam* optimizer (see section B.2.3) with default settings for all the training. The embeddings of the features are visualized through the dimension reduction technique called *Uniform Manifold Approximation and Projection* (*UMAP*) by McInnes et al. [26].

5.1. SoundNet-based network

The training of this model has been carried out on a balanced selection of samples randomly taken from our MUSPAD corpus leading to the dataset described in table 5.1. The balancing procedure kept all the advertisements present in the whole dataset and add speech and music segments accordingly, while trying to balancing also within these two latter classes.

	N of samples	Binary label
Speech	415	0
Music	415	0
Other	830	0
Advertisement	830	1
Total	1660	-

Table 5.1.: Balanced dataset specifications: Other = Speech + Music

The dataset is further divided in a 1200-samples training set and a 460-samples test set. The mini-batch size is fixed to 50 samples.

Clearly, to perform the optimization, samples within the same mini-batch must have the same length. To satisfy this requirement we find the longest advertisement within the mini-batch and we truncate or 0-pad the other samples consequentially. This strategy is carried on to avoid the loss of advertising information which is the scarce resource in our problem.

[1] https://colab.research.google.com/

Since we decided to keep the SoundNet network as it is, no validation of parameters, except for the number of training epochs, is needed. A visual inspection of the training and test trends has not given clear sign of overfitting. Hence we run several times the optimization of the binary cross-entropy loss (section B.3.1) for 150 epochs and we kept the best model in terms of test *Precision-Recall Area Under the Curve* (*PR-AUC*). The training curves of this particular model can be seen in fig. 5.1. They have been smoothed for sake of clarity.

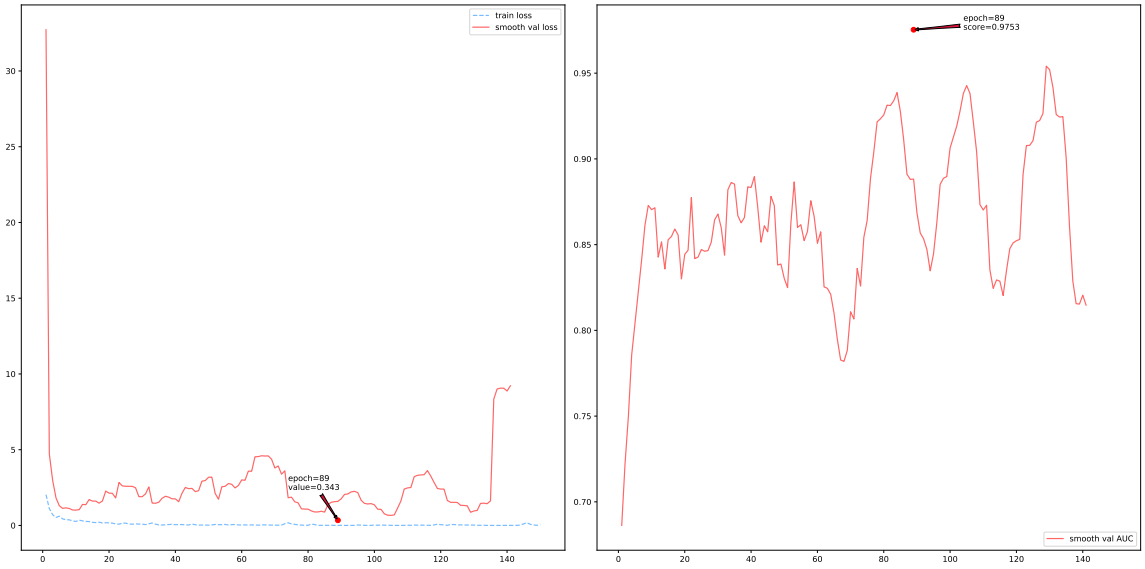


Figure 5.1.: Training curves of the 1024 features model with the best AUC

We deepen the test results showing the confusion matrix in table 5.2 and the test report in table 5.3.

	Other	Advertisement
Other	203	10
Advertisement	29	218

Table 5.2.: Confusion matrix of the 1024 features model. True labels as rows

	Precision	Recall	F1-score	Accuracy	Support
Other	0.88	0.95	0.91	-	213
Advertisement	0.96	0.88	0.92	-	247
Macro avg	0.92	0.92	0.92	-	460
Weighted avg	0.92	0.92	0.92	-	460
Accuracy	-	-	-	0.92	460

Table 5.3.: Test report of the 1024 features model

In fig. 5.2 we can have a look to the test *Precision-Recall* curve of class 1, *i.e.* advertisement. We notice how our usual classification threshold of 0.5 is suitable to obtain a good trade-off between precision and recall.

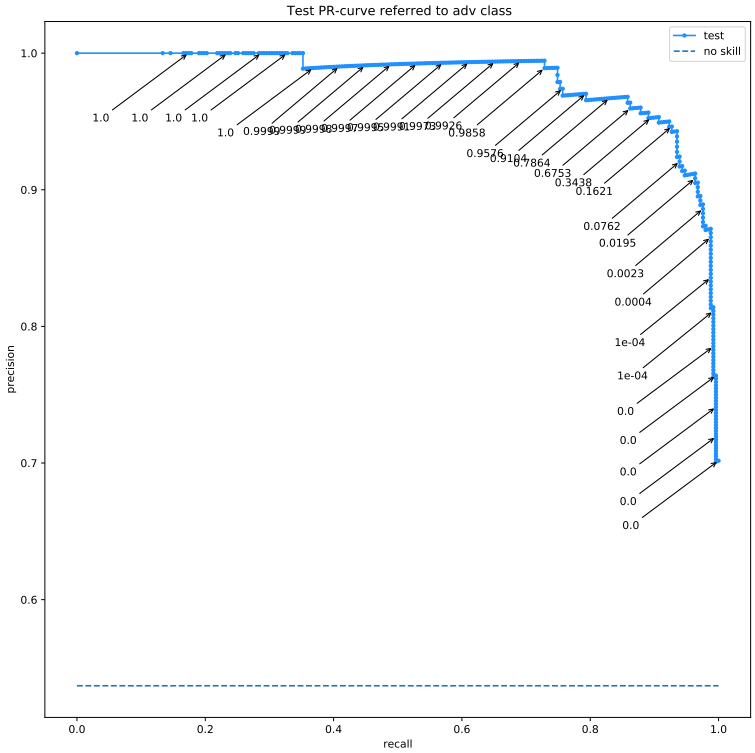


Figure 5.2.: Advertisement class P-R curve of the 1024 features model

In order to show the semantic relevance of the 1024 features, we performed a two dimensional *UMAP* embedding and visualized our balanced dataset in fig. 5.3. This visualization suggests a strong discrimination property of the extracted features, along the horizontal axis, as far as the advertisement's localization is concerned.

Moreover notice that, despite the fact that the classifier has been trained for the binary classification task between advertisement and other, here also music and speech classes are well separated along the vertical axis. Hence we can infer that also the advertisement cluster presents commercials with strong musical content on top, in correspondence with the music cluster, and with strong speech content on bottom. This hypothesis is supported by the presence of a small cluster of songs at the top of advertisement agglomerate and by the almost half-moon shape of the latter. Indeed the top and bottom parts of the advertisement cluster tend to be closer respectively to the music and speech agglomerates. In particular the speech cluster results closer to the advertisement bottom offshoot, indicating greater difficulty in discriminating this two classes.

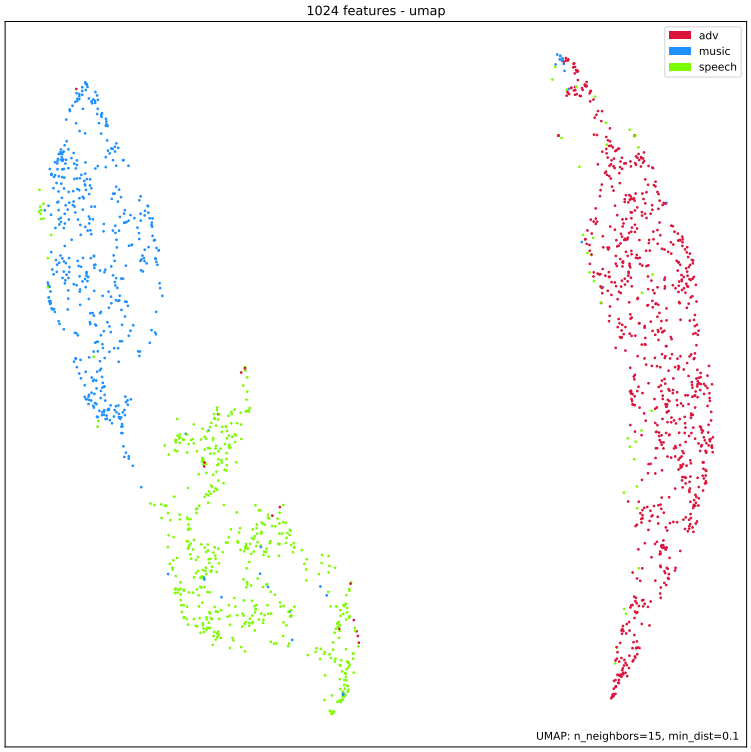


Figure 5.3.: 2D UMAP embedding of the 1024 features model

To answer the question if a lower number of features may be sufficient for the classification purpose we plot in fig. 5.4 the per-class normalized averaged values of the 1024 features extracted by the network. Darker pixels mean higher values.

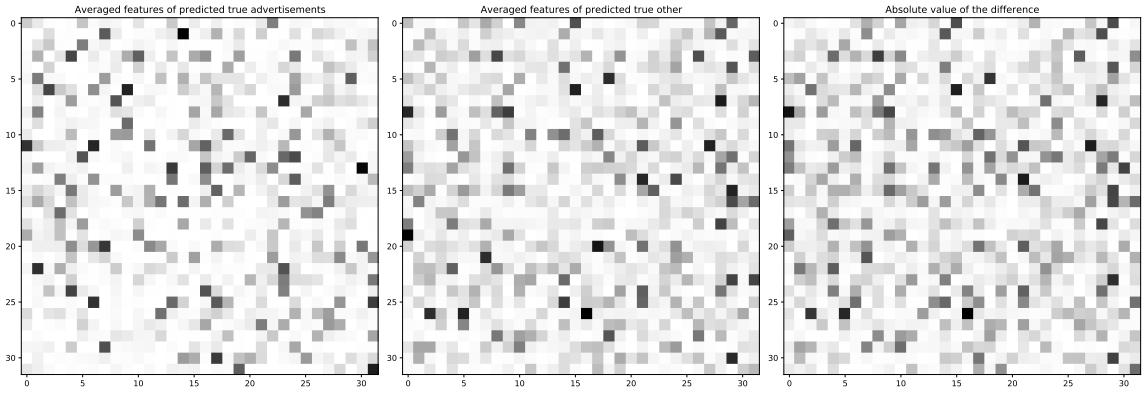


Figure 5.4.: Per-class normalized averaged values of the 1024 features

A visual inspection suggests that a trial with fewer features should be carried out. Indeed a significant amount of features seems to have, in average, low

values in presence of both advertisement and other samples. The straightforward approach to achieve this is removing the 7-th convolutional layer from the network, connecting the 6-th layer's 512 features directly to the dense classification layer.

We can look at the obtained performances in table 5.4 and table 5.5.

	Other	Advertisement
Other	209	26
Advertisement	12	213

Table 5.4.: Confusion matrix of the 512 features model. True labels as rows

	Precision	Recall	F1-score	Accuracy	Support
Other	0.95	0.89	0.92	-	235
Advertisement	0.89	0.95	0.92	-	225
Macro avg	0.92	0.92	0.92	-	460
Weighted avg	0.92	0.92	0.92	-	460
Accuracy	-	-	-	0.92	460

Table 5.5.: Test report of the 512 features model

The result seems promising: it is comparable to the network with 1024 features. To assess the semantic value of this 512 features extracted we show the UMAP embedding fig. 5.5.

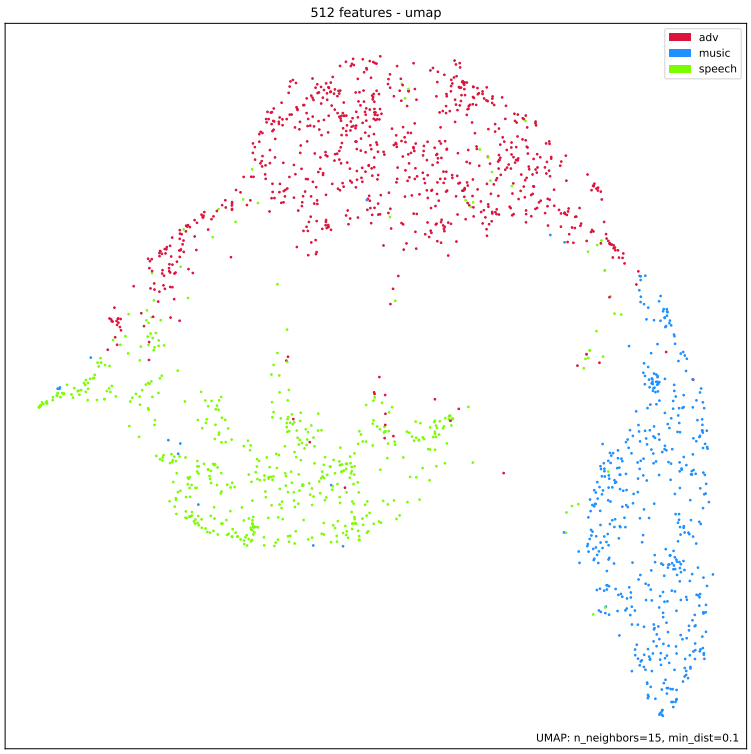


Figure 5.5.: 2D UMAP embedding of the 512 features model

Here we notice the lack of a strong separation, along one of the axis, between advertisement and other classes. A clear leakage of advertisement samples towards music and speech clusters is present respectively in the right and the left parts of the figure. At least, a good discrimination along the x-axis between music and speech classes is again present.

5.2. CNN-TCN-based network

Here we analyze the results obtained by the more complex CNN-TCN-based network. As presented in section 3.2 the goal is the same of the SoundNet-based network with the difference that the actual classification is carried out employing only the estimated music and speech energies.

5.2.1. CNN-TCN energies extractor

We trained the model described in section 3.2.1 with the synthetic dataset in section 4.2.

Leveraging on the large quantity of samples that can be generated in this way, the dropout regularizer, implemented originally, resulted to be unnecessary.

We utilized 10000 samples for training and 5000 for validation purpose to optimize the MSE loss function in eq. (B.15).

Also here, to perform the optimization, samples within the same mini-batch must have the same length. To satisfy this requirement we find the longest advertisement of the mini-batch and we truncate or 0-pad the other samples consequentially.

The loss curves, as we can see in fig. 5.6 reach saturation. We chose the setting which has given the best validation loss.

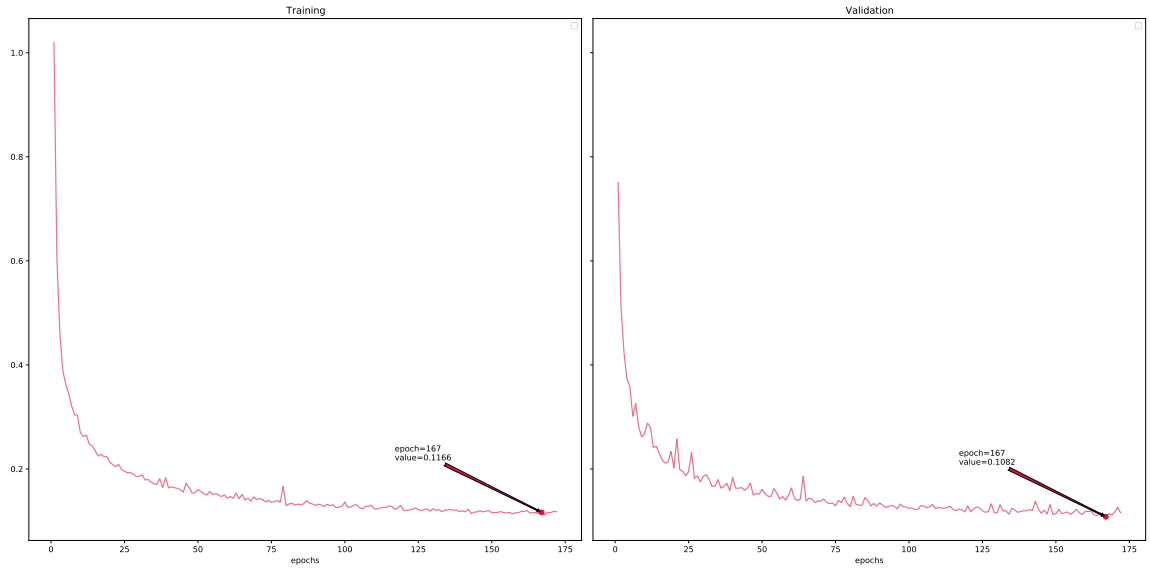


Figure 5.6.: Energies extractor training and validation MSE losses

Moreover this choice yielded also a good result (fig. 5.7) in term of cosine similarity (section B.3.5), which monitors peaks alignement, since it consists in the cosine of the angle between ground thruth and predicted vectors. A test value of -1.83 ensures that the averaged alignement of both music and speech vectors is at least -0.83, where the best possible value is -1.

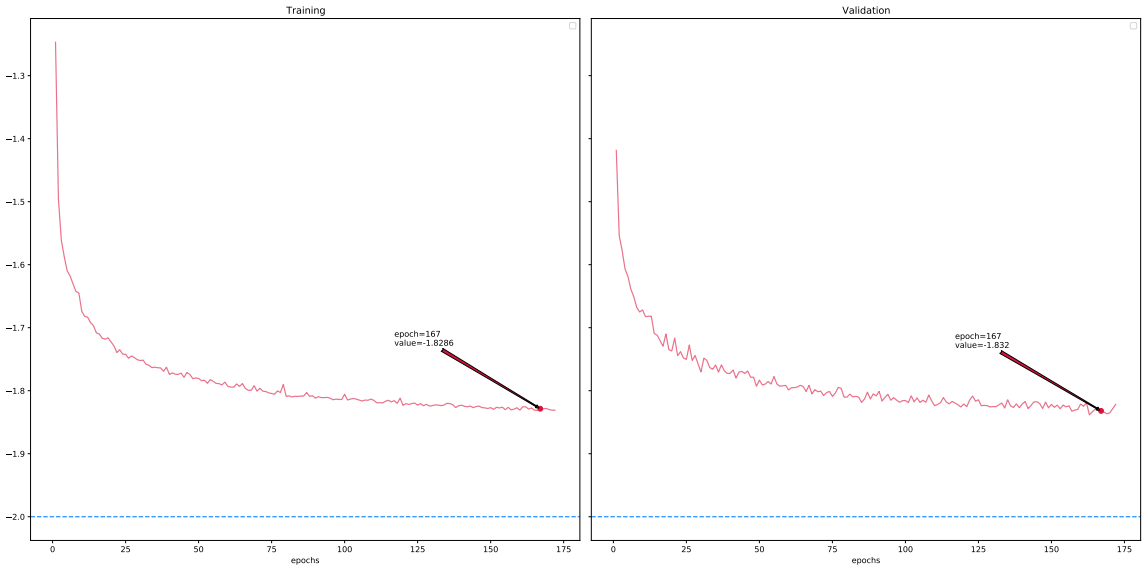


Figure 5.7.: Energies extractor training and validation cosine similarity

Visually we can look at the interpolation property of the selected model on a generated test sample in fig. 5.8: the estimated energies follow rather well the true ones in correspondence of peaks and also across low-energy zones for both music and speech.

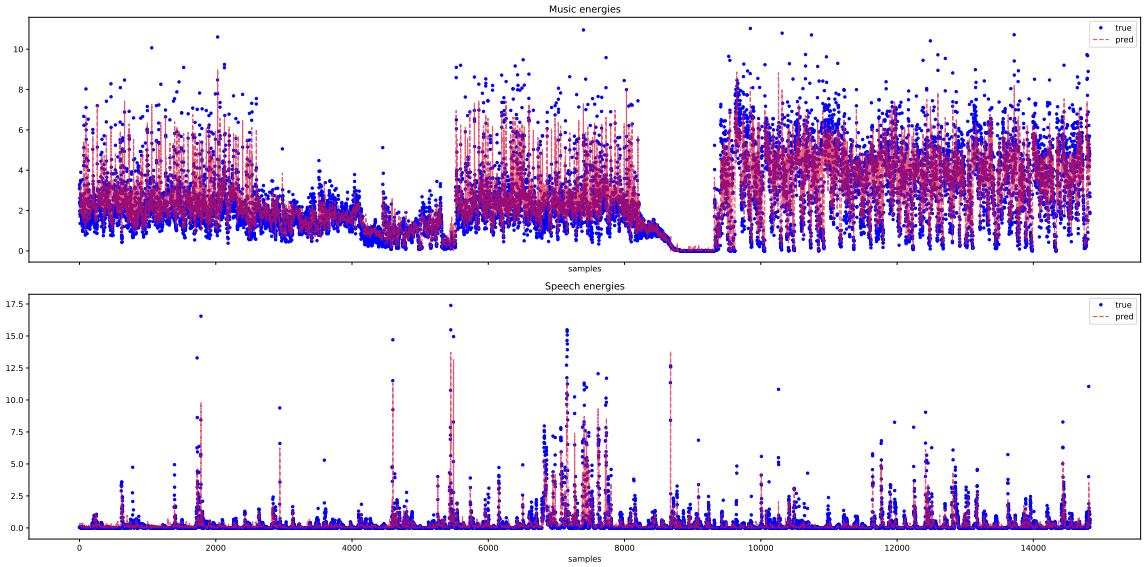


Figure 5.8.: Performance of the energies extractor on a test sample

As far as our MUSPAD dataset is concerned, the estimated energies are consistent with the labels, *i.e.* we have high musical energy in music samples, high speech energy in speech samples and a balanced content in advertisement samples. We can see an instance in fig. 5.9. Notice how in the music sample the speech energy values are almost everywhere lower than the music one. Probably this is a form of noise or, perhaps, the energy extractor interprets the vocals of the song as speech. Conversely the speech sample clearly shows the presence of, a rather loud, background music, as oftens happens in radio broadcasts.

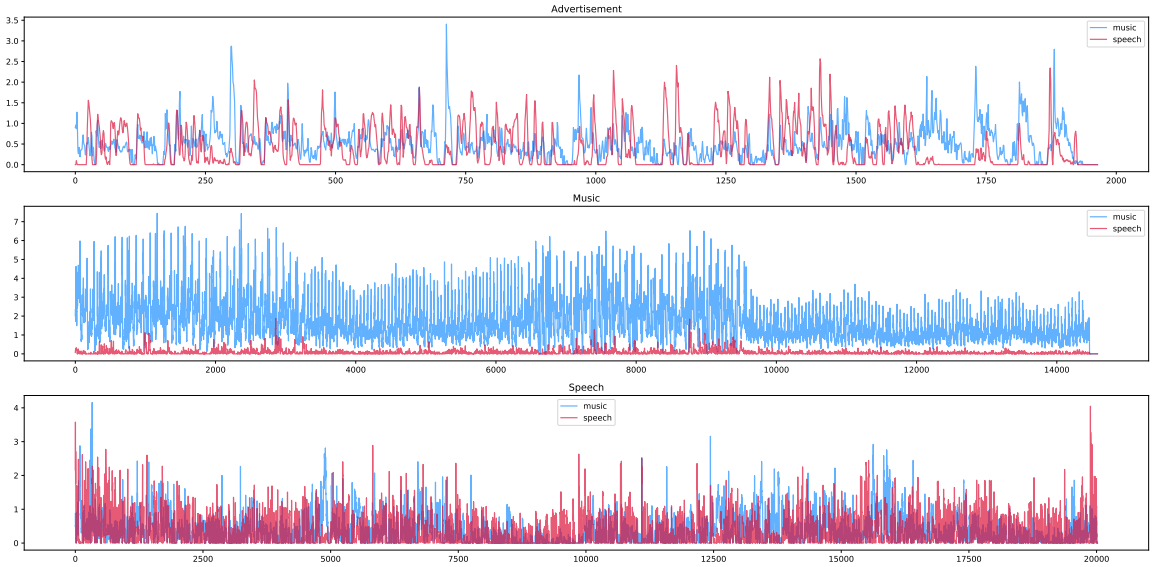


Figure 5.9.: Performance of the energies extractor on a MUSPAD sample

5.2.2. CNN-dense classifier

This portions of the network estimates the membership class of a sample taking as inputs only its music and speech estimated energy vectors.

Before that, a moving average smoothing window of length 29 has been applied to both the energies yielded by the energies extractor. That is equivalent to convolve the energies with the 29-samples long $(1/29, 1/29, ..., 1/29)$ vector. The choice of the window length has been cross-validated with 4 folds on the MUSPAD dataset. Results in terms of loss and binary accuracy can be seen respectively in fig. 5.10 and fig. 5.11. The solid lines represent the baseline window length equal 1 (no-smoothing) in black and the best choice window length equal 29 in magenta. The latter choice of smoothing window's length, though in more epochs, seems to reach a better minimum and maximum concerning respectively the loss and accuracy.

As far as the architecture is concerned, the one which performes best is the one described in section 3.2.2. Increasing the network depth or width, and hence the number of parameters, was useless: no improvements in metrics were gained. Neither did adding some form of regularization.

We optimized the binary cross-entropy loss function (section B.3.1). The results of choosing the model with the lowest such value can be seen in table 5.6

and table 5.7.

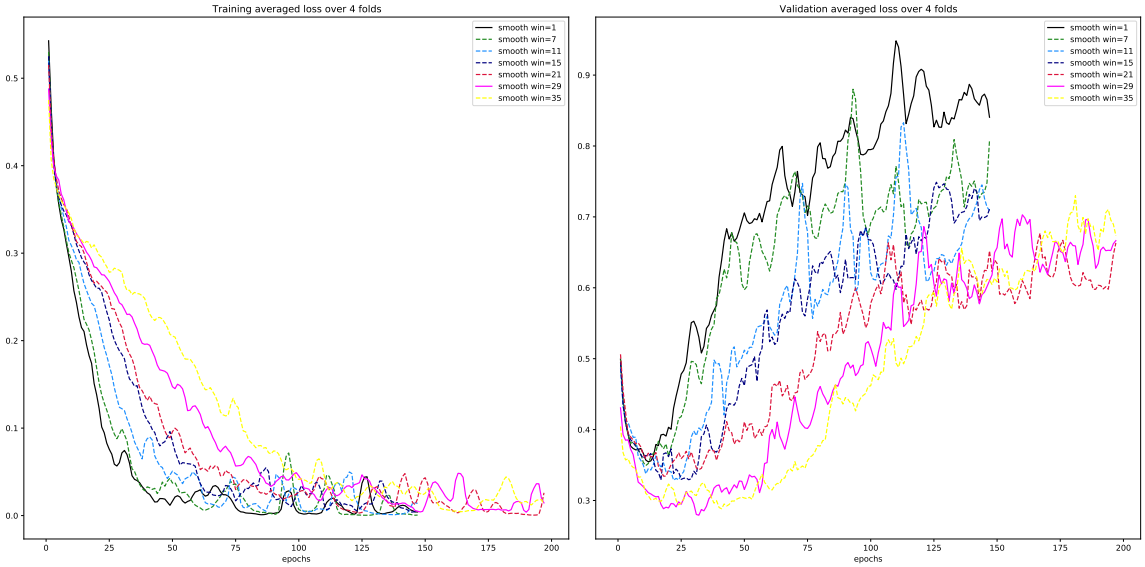


Figure 5.10.: Cross-validation of the smoothing window length - Loss

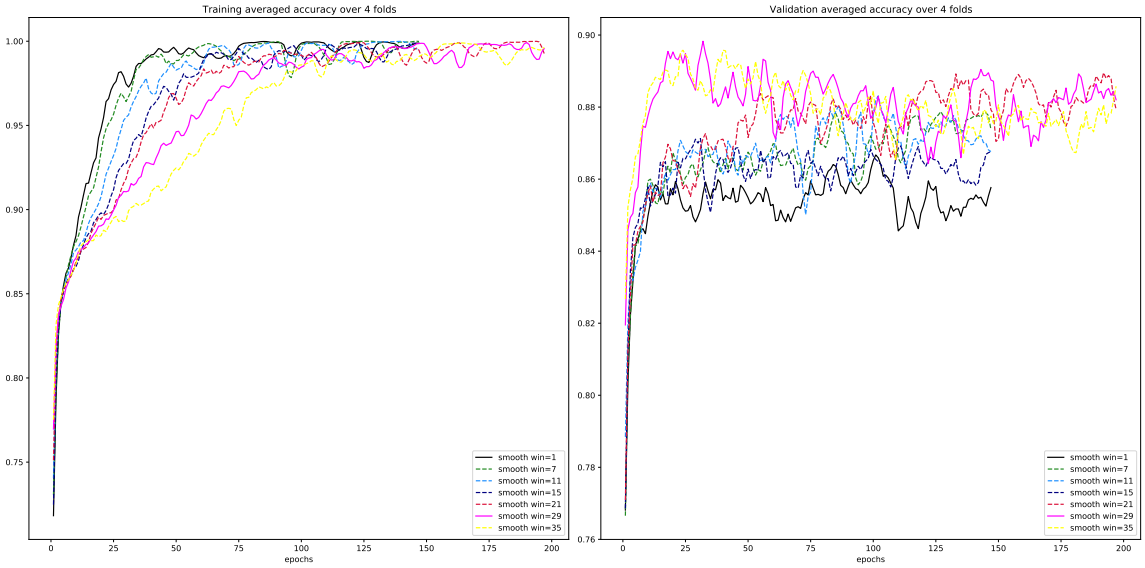


Figure 5.11.: Cross-validation of the smoothing window length - Accuracy

	Other	Advertisement
Other	210	30
Advertisement	39	101

Table 5.6.: Confusion matrix of the chosen model. True labels as rows

	Precision	Recall	F1-score	Accuracy	Support
Other	0.84	0.88	0.86	-	240
Advertisement	0.86	0.82	0.84	-	220
Macro avg	0.85	0.85	0.85	-	460
Weighted avg	0.85	0.85	0.85	-	460
Accuracy	-	-	-	0.85	460

Table 5.7.: Test report of the chosen model

Here in fig. 5.12 we look at the advertisement class Precision-Recall curve: the usual 0.5 threshold seems an equilibrate choice.

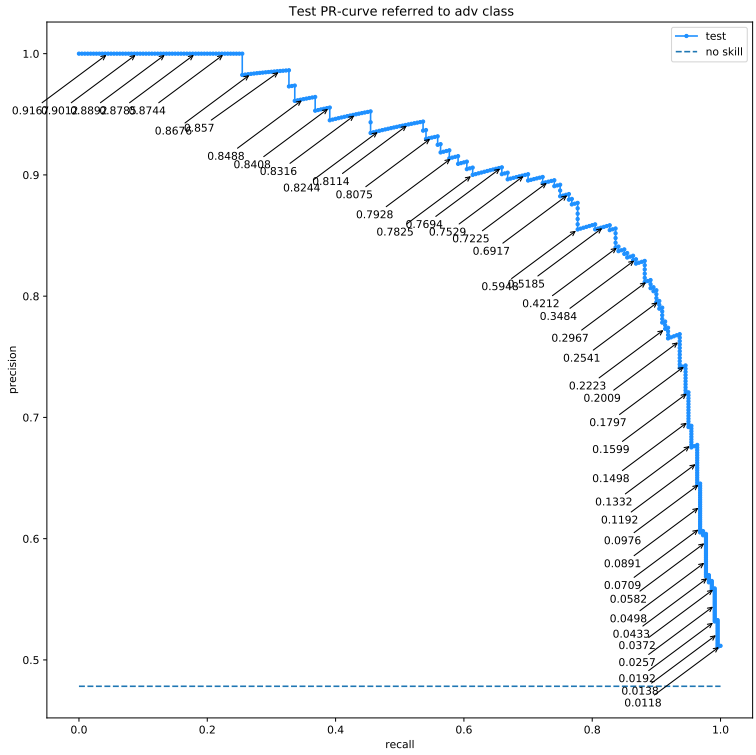


Figure 5.12.: Advertisement class P-R curve of the chosen model

In fig. 5.13 we show the UMAP embedding. The 1024 features extracted by the CNN classifier appears to have poor semantic values. Indeed the two embedding variables both discriminate well between music and speech samples, but the advertisement ones are dispersed, mostly in the speech area.

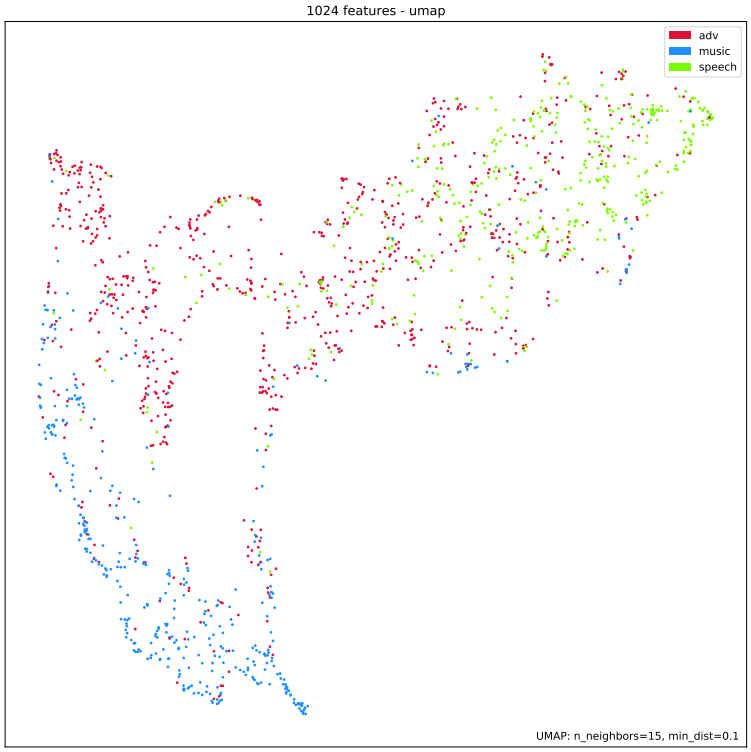


Figure 5.13.: 2D UMAP embedding of the CNN-TCN model's 1024 features

5.3. Robustness analysis

The audio presented as input to the two networks may show some kind of perturbation due to streaming quality or segmentation errors. Moreover these kind of modifications represent basically the differences we may encounter among the same advertising aired on different radio stations. Therefore it is essential, for the sake of clustering, that also perturbed samples are truly classified as advertisement.

To assess the robustness of the networks to these sort of perturbations we take some true-positive samples, *i.e.* advertisement samples which are correctly classified as advertisement by the model, and we modify them in the following ways:

1. *Left crop* of 10% of the audio segment
2. *Right crop* of 10% of the audio segment
3. *Left and right crop* of 10% each of the audio segment
4. *Centered crop* of 10% of the audio segment at a random location
5. *White noise injection* in the whole audio segment
6. *Resampling* at 20% higher and lower sampling rate

We compute the goodness of the model on this 7-times augmented advertisement dataset, *i.e.* the percentage of perturbed samples which are however correctedly classified advertisement, and plot the UMAP embedding of the 1024 features extracted.

5.3.1. SoundNet-based network robustness

We performed the robustness analysis on both the SoundNet-based model with 1024 features and the one with 512. Both the networks appear to be very robust, at least concerning the kind of perturbations applied. Indeed they manage to correctly label as advertisement the 95% and the 96% of the 1400 perturbed segment, correspondent to 200 true-positive original samples.

In fig. 5.14 we can look at the 2D embedding of the original and perturbed samples' features, in the case of the SoundNet-based network extracting 1024 features. We notice how, with few exception, mostly due to resampling, the perturbed samples remain within the original cluster.

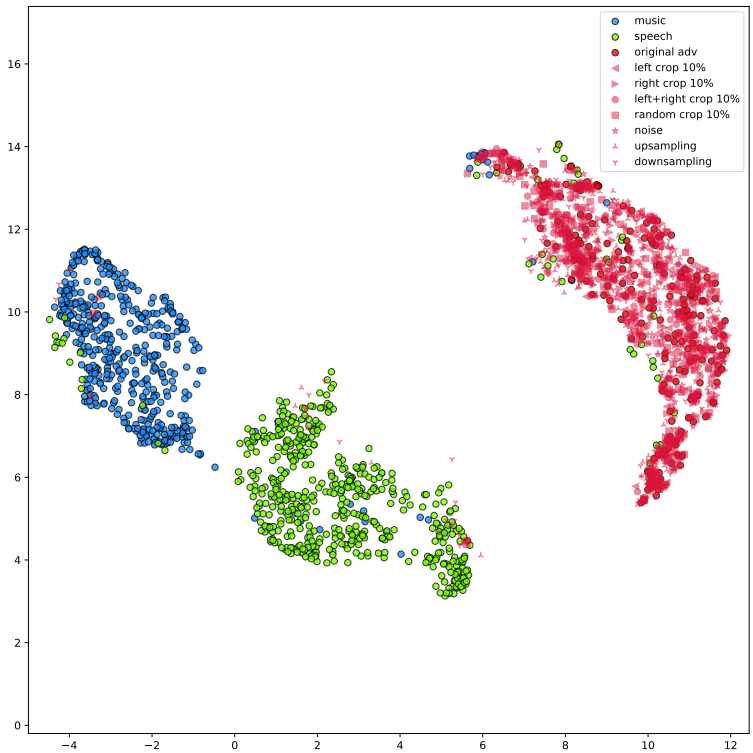


Figure 5.14.: 2D UMAP embedding for robustness analysis of SoundNet-based 1024

5.3.2. CNN-TCN-based network robustness

The selected CNN-TCN-based network performance, on perturbed true-positive samples, shows a considerable degradation. The 75% of the 1400 perturbed samples, correspondent to 200 true-positive original samples are still correctly

classified as advertisement. This is a clear index of sensitivity of this energy-based network to the sort of perturbation applied.

For completeness we show the UMAP embedding in fig. 5.15.

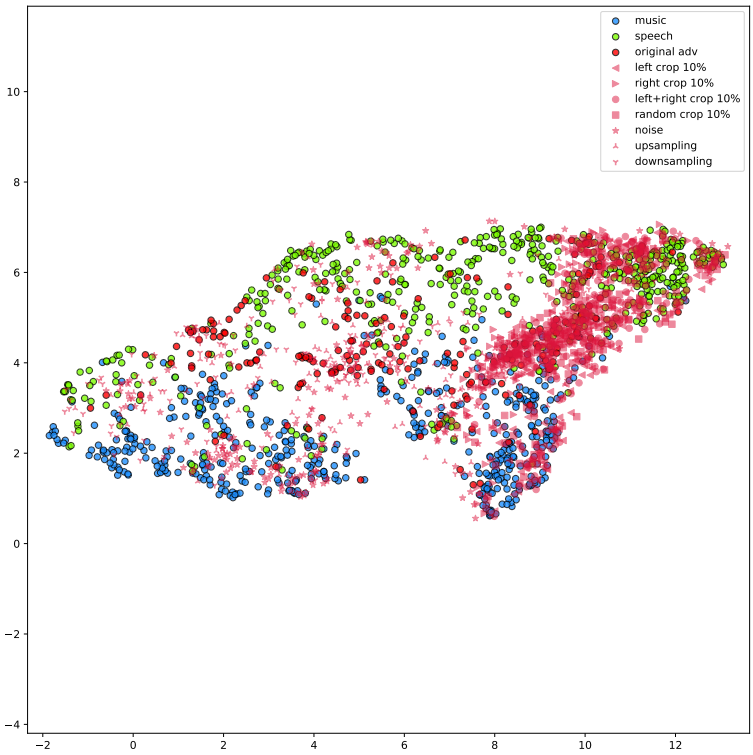


Figure 5.15.: CNN-TCN-based 2D UMAP embedding for robustness analysis

5.3.3. Test with english advertisements

We test our SoundNet-based network giving it as inputs some english advertisements collected from a british radio station, as exposed in section 4.1. This analysis was taken only on the SoundNet-based model because the CNN-TCN-based network already showed critical issues regarding robustness.

The goal of this testing is to assess the possibility to extend the usage of the network to radio broadcasts in other languages, in this case the english speech.

At first glance the british advertisements, or at least the ones here taken into account, seem rather different from the italian ones. It looks like they contain less musical background content and, consequentially, spoken parts which clearly and strongly stand out. Many of them are even composed by basically only speech.

The SoundNet-based model manages to correctly classify as advertisements 100 of the 119 test samples. This means approximately 84% of accuracy.

Given the diversity, exposed before, between the inputted english advertisements and the italian ones, which, by the way, were used for the model training, it seems an interesting result. Indeed it is reasonable to claim that our network may have learnt some features, useful to classify the advertising, which are robust also to a change of language.

We conclude showing in fig. 5.16 the UMAP embedding of the english advertisements. Most of the samples appear correctedly within the advertising cluster, whereas the few mistaken ones are mostly situated in the speech agglomerate.

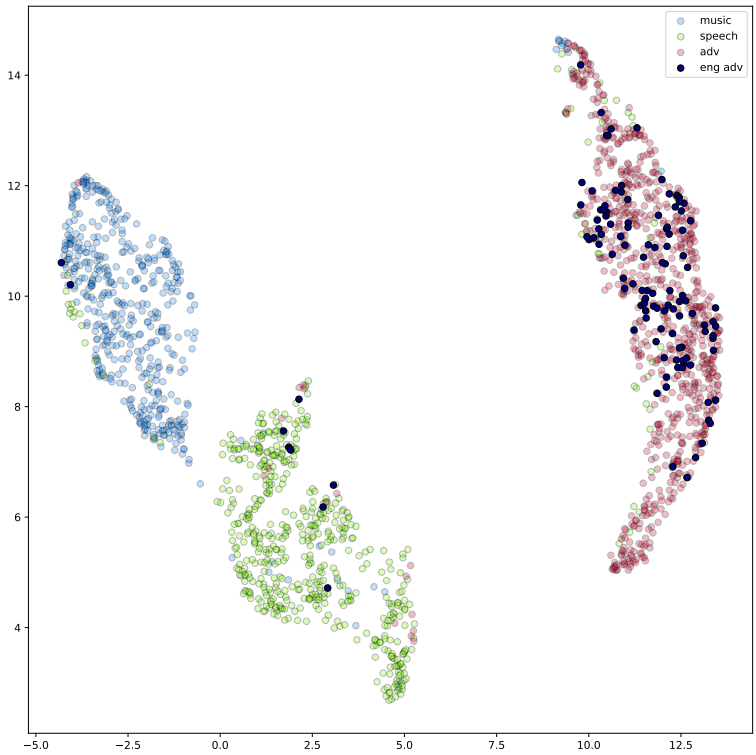


Figure 5.16.: SoundNet-based 2D UMAP embedding of english advertisements

5.4. Discussion of the results

In this section we compare the performances of the SoundNet-based model with seven convolutional layers, which provides 1024 features (SN-1024), the SoundNet-based model with six convolutional layers, which provides 512 features (SN-512), and the CNN-TCN-based network composed by an energies extractor combined with a CNN-dense classifier (CNN-TCN), providing 1024 features as well.

Achitecture We compare the three architectures in terms of width, depth and total number of parameters. SN-1024 and SN-512 share the majority of the network architecture, except for the last convolutional layer, having respectively 7 and 6 1D-convolutional plus one dense layers.

As far as the CNN-TCN is concerned it has 7 2D-convolutional and 12, grouped by two, 1D-convolutional layers due to the energies extractor. The superimposed classifier add 5 more 1D-convolutional layers.

We summarize the depths and the total numbers of parameters in table 5.8.

	2D conv	1D conv	Dense	Tot trainable param
SN-1024	-	7	1	2,875,320
SN-512	-	6	1	774,609
CNN-TCN	7	17	1	1,924,899

Table 5.8.: Comparison among models in terms of depth and number of parameters

While the depth and the total number of parameters are fixed, the widths, and thus the total numbers of units, of the three network depend on the cardinality of the input. A longer audio sample results in a greater number of units at each layer, until the global max-pooler, which squeeze the time dimension to obtain a fixed length vector, *i.e.* the feature vector, to be fed to the dense final layer. Exactly for this reason the audio input lengths have to be fixed within each mini-batch, to build a consistent instance of the network. The previously described techniques to obtain proper mini-batches let the network operate with sufficiently short samples. In this way the network widths correspondent to each mini-batch are well contained, without exceeding the GPU's internal memory, during training. This particular issue is important in the TCN part where the time dimension is left invariate. We managed to train it since in this case our samples can be synthetically generated online (section 4.2) with a feasible length.

Training and testing The training durations of SN-1024 and SN-512 are comparable since they are both around 100 seconds per epoch.

Concerning the CNN-TCN, since the energies extractor and the classifier are trained at different times, we separate the two durations. A training epoch of the energies extractor typically took almost 2000 seconds. This is due to the fact that the residual networks in general require long trainings and the fact that the mini-batches are generated and assembled online. On the contrary the training of the classifier part was very quick at less than 10 seconds per epoch, in average. That is because the network is rather simple and the input energies consist of fewer quantities of data. We can say that the more "effortful" work is done by the energies extractor.

Comparing the training curves in fig. 5.1, fig. 5.6, fig. 5.7 and fig. 5.10, fig. 5.11 we notice that, while the two SoundNet-based model and the energies extractor do not present clear signs of overfitting and they rather reach a sort of saturation of the performances, in the CNN-classifier plots we can clearly identify the point at which the test metric start to worsen. This is because the former ones employ some kind of regularization techique or, in the case of the energies extractor, leverage a large amount of training data, while the latter does not.

In the following table 5.9 we compare the metrics employed in the classification tasks.

A separate discussion has already been taken about regression performances of the energies extractor in section 5.2.1.

	Accuracy	Adv precision	Adv recall	Adv F1-score
SN-1024	0.92	0.96	0.88	0.92
SN-512	0.92	0.89	0.95	0.92
CNN-TCN	0.85	0.86	0.82	0.84

Table 5.9.: Comparison among models in terms of accuracy and the other adv-class metrics

Performances of the two SoundNet-based models are almost identical while they clearly outperform the CNN-TCN network.

Features embedding We plotted the UMAP embedding of the three models in fig. 5.3, fig. 5.5 and fig. 5.13.

At first sight we notice the goodness of the SN-1024's features embedding. As already observed it presents a noticeable separation among the advertisement, music and speech clusters. The SN-512 embedding seem to be less precise, with more leakage between the classes, but still presents a good separation among them.

The worse classification performance of the CNN-TCN model reflects on the UMAP embedding of its extracted features. These features seem to be effective only in separating music and speech clusters, while the advertisement samples are spreaded in-between them.

It is clear that the two SoundNet-based networks are able to learn, from raw audio inputs, a more semantically relevant vector of features than the CNN-TCN from the sole energies extracted. In particuler SN-1024 achieves the best result.

Robustness We perturbed some correctedly classified advertisings to assess the robustness of the networks under scrutiny.

The results are clearly better as far as SN-1024 and SN-512 are concerned. They manage to correctedly recognize respectively 95% and 96% of the perturbed samples as advertisement, while the CNN-TCN achieves only a 75%. We can consider the robustness of SN-1024 and SN-512 comparable.

These results reflect in the UMAP embeddings fig. 5.14 and fig. 5.15. In the SN-1024 features' embedding the perturbed samples appear, with few exceptions, well within the advertisement cluster.

Regarding the CNN-TCN instead, the perturbed samples are mapped all around the space, mostly in the speech agglomerate.

The trial with english advertisings, performed only with SN-1024, yielded a worse, but still reasonable, result. The network is trained only using italian advertisments, therefore 84% accuracy means that italian and english advertising share some, but not all, of the features that SN-1024 is able to extract.

6. Conclusion and future works

In this Thesis we have evaluated the performances of two Deep Neural Network architectures to assess their possible industrial development, in the task of radio-broadcasts advertising classification. We have also assembled *MUSPAD*, a dataset containing music, speeches and advertisements recorded from some italian radio stations. Moreover we explained a possible procedure (*ADV-MUSAN*) to build synthetical advertisement-like samples to be employed in training the energies extractor.

From an industrial point of view the SoundNet-based architecture shows promising results. Both SN-1024 and SN-512's performances in the classification task are remarkable. We think that fine-tuning this architecture could lead to state-of-the-art results feasible for commercial applications. For instance, as we have seen exploring the results, the best number of features extracted may be found within 512 and 1024 to balance between the model dimension and the significance of the embedding.

As far as the CNN-TCN-based network is concerned, it is clearly outperformed by the SoundNet-based model. Perhaps some improvements can be obtained with some more trials and different tunings of both the energies extractor and the classifier stages. However we claim that such a network weakness is given by the fact that the classification is solely based on the estimates of music and speech energies. This means that we are basically forcing a bottle-neck in the flow of informations between the raw audio input and the classifier. Still this is true for at least some kind of advertisements, *i.e.* the ones that induce the approximately 10% difference in performances.

With this in mind it could be interesting to perform an analysis of the SoundNet-based model's interpretability to understand which kind of audio features allow this network to outperform the energy-based one. A proper techinque to carry out this kind of analysis can be *GradCAM* (Selvaraju et al. [27]). Since it was developed for the visual context of object recognition it requires some modification to be effective also in the audio field.

Back to the CNN-TCN-based network, not everything is to be discarded. For instance we have proven that the TCN model with a CNN front-end proposed by Meléndez-Catalán [10], after a slight modification, perfoms rather well in the regression task of estimating music and speech energies. Moreover a further trial could be done using a CNN-TCN novel architecture to directly carry out the classification, without passing through the energies extraction. A CNN-TCN-CNN-Dense pipeline could yield results comparable or even better than the SoundNet-based model.

Furthermore having at disposal TV advertisement samples, *i.e.* samples composed by both visual and audio components, would let us experiment a teacher-

student architecture similar to the original SoundNet which strenghten the learned features using the visual informations.

As we already expressed, improvements can be made also to the MUSPAD dataset. Besides simply expanding it's dimension, this could be done in a meaningful way, *i.e.* programming the recording and labeling campaigns to be spreaded over a whole year. This fact should yield a generalization improvement and thus better representation of the radio advertising's universe. Moreover the recording sessions ought to cover the whole day.

Finally trying to perform classification with english advertisements shows how, to achieve good results, one probably should collect a dataset specific for the language used in the broadcasts in question and then retrain the model. This means that for a commercial deployment in other countries a tailor-shaped dataset should be collected.

Towards advertisement clustering The final goal of the whole automatic system is to identify a specific advertising aired by the radio stations.

The clustering inference consists of assigning an incoming advertisement to a particular cluster, in the space of the features, by means of a specific employed metric. Since this particular cluster represents an advertising, the incoming one is predicted to be such specific advertisement. In the simplest case the cluster is represented by a single advertising stored in a database in the form of a features vector.

How is this algorithm able to tell when a new, *i.e.* still not present in the database, advertisement is encountered? A classic approach is to fix a confidence threshold c to be compared with the computed metric. Basically if the incoming advertisement is at a "sufficient distance" from all the samples present in the database, then it is considered new and directly stored or sent to other processing stages. Such a "sufficient distance" is governed by the confidence threshold. The previously exposed is the inference phase.

The training of the DNN clustering network is carried out employing batches of perturbed and transformed advertisements and minimizing a particular loss. Such a loss should be small when a perturbed sample (*positive*) is compared with its original one (*anchor*) and large when compared with a different advertisement (*negative*). For instance a suitable loss function can be the *triplet loss* (Chechik et al. [28]) utilized, among the others, by the face recognition and clustering network *FaceNet* (Schroff et al. [29]).

To summarise, looking at fig. 1.1 in the Introduction, the trained clustering DNN extracts the vector of features $\mathbf{f}_p$ from an incoming advertisement. A measure of distance is computed between $\mathbf{f}_p$ and all the advertisings present in the database, by means of their features vectors $(\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_i, ..., \mathbf{f}_N)$. If the smallest distance computed is $d(\mathbf{f}_p, \mathbf{f}_i)$, and this quantity is less than the confidence threshold c, then the incoming advertisement is recognized as the $i-th$ advertisement in the database. Otherwise it can be added to the database, *i.e.* $\mathbf{f}_{N+1} = \mathbf{f}_p$.

In the following we show an example of this algorithm at work. Since, with respect to the classification network, the clustering network is a completely distinct model and undergoes another training procedure, the extracted features

are obviously different.

Here we unproperly make use of the features extracted, in the classification framework, by SN-1024 to perform clustering, because they are already at our disposal. Usually the training and tuning of a clustering network take a lot of time and care, and these are not arguments of this Thesis.

We could find useful this exemplary analysis to assess whether the SoundNet architecture might be used for the clustering network as well.

Our setting contemplates a database of 92 true advertisements, which has been relabeled with a specific tag, as exposed in section 4.1. To simulate an incoming batch of advertisings, to be clustered against this database, we pertubate its 92 records, with cropping and noise, obtaining 460 samples. Moreover we inserted in the batch 200 advertisements to simulate the arrival of new samples to be added to the database.

For the cluster assignement we chose to use the simple euclidean distance and the cosine similarity. Therefore we computed the distances between each sample in the batch and each record in the database, more precisely the distances between their features vectors. Then the 460 perturbed incoming advertisements are assigned to their nearest cluster in the database. Of course the nearest cluster in the database is the one represented by the record at the smallest distance, or greatest similarity.

This clustering network based on SN-1024 is able to assign 368 out of 460 incoming advertisements to the correct record in the database, *i.e.* the 80%. This performance is achieved using the euclidean distance as metric. Regarding the cosine similarity the accuracy slightly degrades to the 74%.

We keep track also of the metrics' best values computed for the 200 new incoming advertisements. These should give interesting suggestion for the choice of the confidence threshold c. Indeed have a look at fig. 6.1.

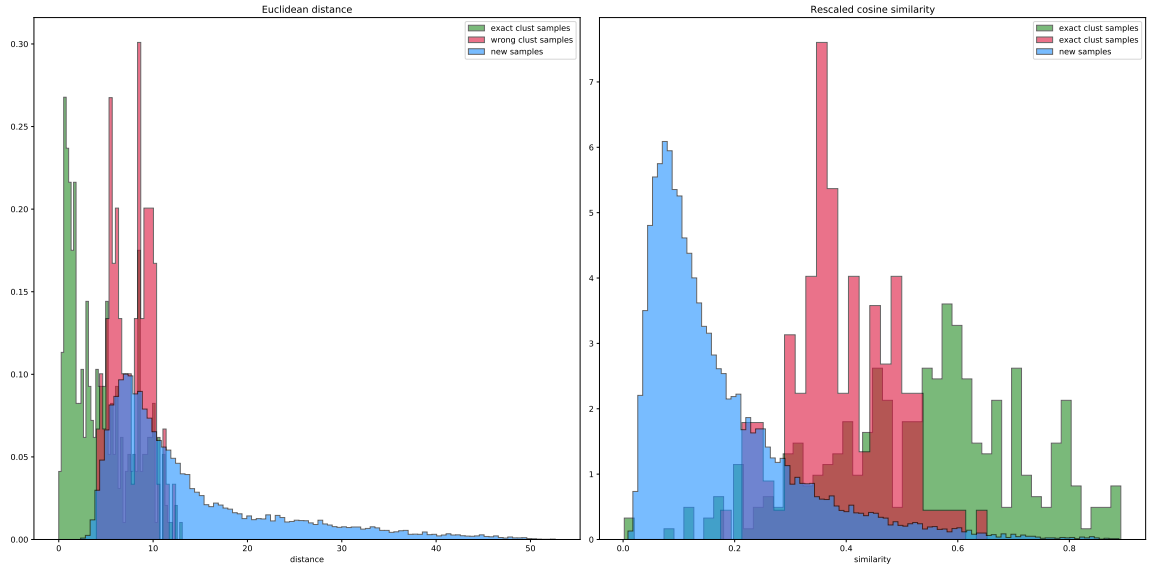


Figure 6.1.: Distributions of the metrics' best values

In green we have the metric's values which led to an exact cluster's assigne-

ment, in red the values that led to a wrong assignement and in blue the best metric's values computed for the new advertisements. Ideally we would like to have the green and the blue distributions as separated as possible and few red samples in the middle.

Again this is a toy example in which the network used for clustering was originally trained for the classification task, and thus the extracted features are not the proper ones.

Such a plot of the distributions may help in choosing a suitable metric d and an efficient confidence threshold c. The latter should be chosen taking into account the application and in particular the cost of assigning a new incoming advertisement to a wrong cluster and, on the contrary, labeling as new an advertisement already present in the database.

In conclusion this experiment suggests that the SoundNet architecture should be a profitable choice also for the clustering network. Indeed, with all the limitations already highlighted, SN-1024, trained for classification, provides features which yielded decent accuracy results also for clustering.

Appendices

A. Theoretical background on Neural Networks

In this chapter of the Appendix we deepen two theoretical aspects of Neural Networks. In the first section we expose the universal approximation property of Neural Networks, in the second a practical example of the residual connections' usefulness is proposed.

A.1. Approximation result

In this section we present a result on the approximation property of a network with linear output units and a single hidden layer. More precisely it has been proven, in different ways by several authors, that such a network can approximate any continuous function uniformly on compacta, by increasing the size of the hidden layer. There are also some results on the rate of approximation, *i.e.* how many units are needed to approximate to a specific accuracy, but they are not very useful in practical problem.

We present the following proposition as exposed by Ripley [36] section 5.7. The above network represents the function:

$$y_k = f_k\left(b_k + \sum_{j=1}^{J} w_{jk} f_j\left(b_j + \sum_{i=1}^{I} w_{ij} x_i\right)\right) \tag{A.1}$$

Consider a function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^p$ linking n inputs to p outputs. We wish to approximate such a function using $\mathbf{g}$ from the class of functions expressed by eq. (A.1). Uniform approximation on compacta means that given a compact set $K \subset \mathbb{R}^n$ and $\epsilon > 0$ we can find a function $\mathbf{g}$, within our class, such that $\|\mathbf{f}(\mathbf{x}) - \mathbf{g}(\mathbf{x})\|_2 < \epsilon$ for all $\mathbf{x} \in K$.

Proposition A.1. *Ripley [36] Any continuous function* $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^p$ *can be approximated uniformly on compacta by functions* $\mathbf{g}$ *of the form eq.* (A.1) *with linear output units and sigmoidal, threshold or ramp units in the hidden layer.*

Proof. The proof proceeds by building up the class of functions that we can approximate.

a) We begin taking $n = p = 1$. Let K be a compact set. Then $K \subset [a, b]$ since a compact set is always contained in a closed interval. Moreover any continuous function f can be uniformly approximated on $[a, b]$ by a step function with steps of size less than $\epsilon/2$. Indeed for each $x \in [a, b]$ define the interval $I(x) = (l(x), u(x))$ where $l(x) = max\{y < x : |f(y) - f(x)| \geq \epsilon/4\}$ and

$u(x) = \min\{y > x : |f(y)-f(x)| \geq \epsilon/4\}$. The open sets $I(x)$ clearly cover $[a, b]$, hence so does a finite collection $I(x_i)$. Sort the x_i in increasing order, and let g take the value $f(x_i)$ on $[l_i, u_i) = [u(x_{i-1}), u(x_i))$. Then by the triangular inequality $|\Delta g(l_i)| = |f(x_{i-1})-f(x_i)| \leq |f(x_{i-1})-f(l_i)|+|f(x_i)-f(l_i)| \leq 2\epsilon/4$. A step function belongs to the class defined by A.1 if the activation functions are thresholds. However sums of sigmoidal or ramp functions can approximate a step function arbitrarily closely, except at the steps, and certainly to within one half of the largest step size.

b) We then extend the result to trigonometric functions of the form $\prod_{i=1}^{n} \cos(\omega_i x+\phi_i)$ for any n. By repeatedly applying the well-known summation formula $2\cos(A)\cos(B) = \cos(A + B) + \cos(A - B)$ this can be written as a sum of the form $\sum_j a_j \cos(\tilde{\omega}_j x + \tilde{\phi}_j)$. Each term in this sum is continuous, so it can be approximated by point (a), thus the whole sum can, as well as linear combinations of these functions, including arbitrary trigonometric polynomials on $\mathbb{R}^n$.

c) From Fourier theory any continuous function $f : \mathbb{R}^n \to \mathbb{R}$ can be approximated by a trigonometric polynomial.

d) Fix K and $\epsilon > 0$. Each component function f_j of $\mathbf{f}$ is continuous, so, from previous points, we can find functions g_j within the class A.1 such that:

$$\sup_{x \in K} |f_j(\mathbf{x}) - g_j(\mathbf{x})| < \frac{\epsilon}{p^{1/2}} \; \forall j \in \{1, ..., p\} \tag{A.2}$$

Recall $\|\mathbf{f}(\mathbf{x}) - \mathbf{g}(\mathbf{x})\|_2 = \left(\sum_{j=1}^{p} |f_j(\mathbf{x}) - g_j(\mathbf{x})|^2\right)^{1/2}$. From eq. (A.2) we have:

$$\begin{aligned} |f_j(\mathbf{x}) - g_j(\mathbf{x})|^2 &< \left(\frac{\epsilon}{p^{1/2}}\right)^2 \; \forall \mathbf{x} \in K, \forall j \in \{1, ..., p\} \\ \sum_{j=1}^{p} |f_j(\mathbf{x}) - g_j(\mathbf{x})|^2 &< p\left(\frac{\epsilon}{p^{1/2}}\right)^2 \\ \|\mathbf{f}(\mathbf{x}) - \mathbf{g}(\mathbf{x})\|_2 &< p^{1/2}\left(\frac{\epsilon}{p^{1/2}}\right) = \epsilon \; \forall \mathbf{x} \in K \end{aligned} \tag{A.3}$$

The function $\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), ..., g_p(\mathbf{x}))$ is within the class eq. (A.1) since we can take separate groups of hidden units for each coordinate function.

□

A.2. Residual connections

In this section, exploring what we have dealt about in section 2.4, we show a toy example of *degradation* effect and its adopted solution, *i.e. residual networks*. Consider the two simple architectures in fig. A.1.

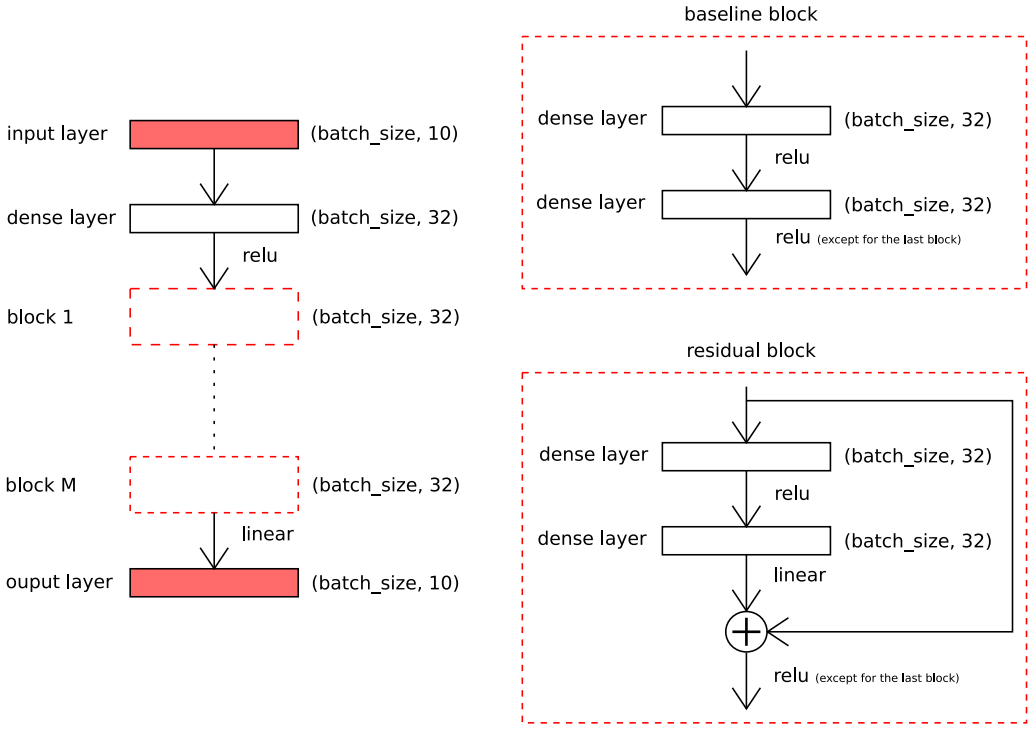


Figure A.1.: Architecture of both baseline and residual networks

They are both built with a stack of M blocks: one featuring baseline blocks composed by two dense layers with ReLU activation, and the other having an additional skip connection from input to output.

Since skip connections are not parametrized, indeed they implement an identity mapping, the total number of parameters is the same for both the models.

We compare this two networks on the task of approximating the cosine function in an interval across $\frac{\pi}{2}$. In fig. A.2 we plot the training loss in function of the total number of layers. The two models are trained with the same number of epochs.

We can clearly see that increasing the number of layers over approximately ten leads a degradation in the performance of the baseline network. Despite this fact the residual network continue to improve well over ten layers. Since we are looking at the training loss this phenomenon cannot be referred to overfitting.

In fig. A.3 we compare the interpolation ability of the two models.

As we have seen in section 2.4, the idea behind adding skip connections is quite simple: models obtained stacking more layers onto a baseline network should not perform worse if we give them the possibility to explicitly learn the identity mapping. Indeed in this way we build a path where the information can directly flow from the baseline network to the output.

Consider a baseline network function $\mathbf{g}(\mathbf{x}; \mathbf{w})$ with input $\mathbf{x}$ and stack on top of it the layers represented by $\mathbf{f}(\cdot; \tilde{\mathbf{w}})$. Calling $\mathbf{y}$ the output, if $\mathbf{y} \approx \mathbf{g}(\mathbf{x}; \mathbf{w})$, which can happen often when the baseline network already provides a good approximation,

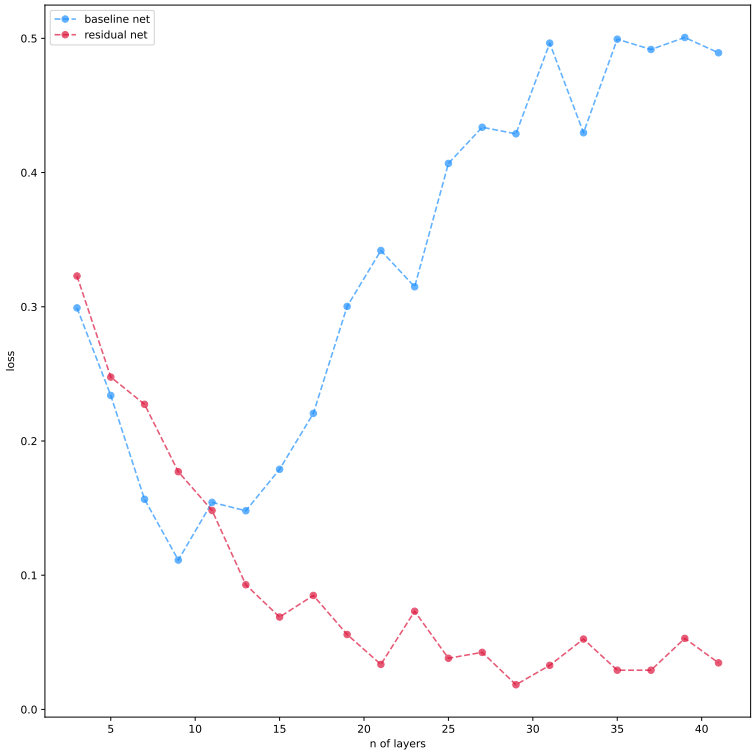


Figure A.2.: Training loss of the baseline and residual networks

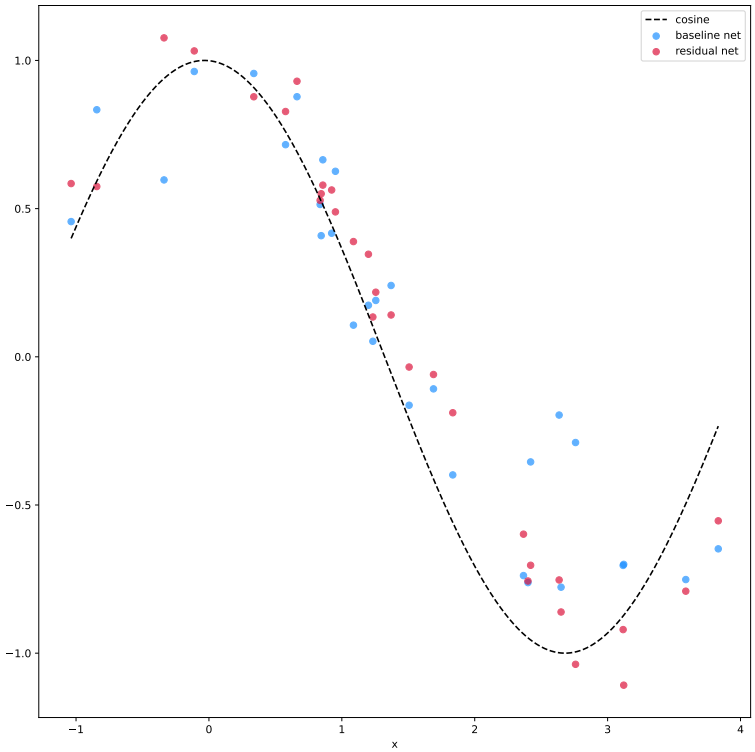


Figure A.3.: Interpolation ability of the best baseline and residual networks

the added layers should learn the identity mapping, *i.e.* $\mathbf{f}(\mathbf{z}; \tilde{\mathbf{w}}) \approx \mathbf{z}$. This task may not be trivial and it is exacerbated by nonlinearities in $\mathbf{f}$ giving rise to the degradation problem.

Adding skip connections we have $\mathbf{y} = \mathbf{f}(\mathbf{g}(\mathbf{x}; \mathbf{w}); \tilde{\mathbf{w}}) + \mathbf{g}(\mathbf{x}; \mathbf{w})$. In this case the solution would be $\mathbf{f}(\mathbf{z}; \tilde{\mathbf{w}}) \approx 0$ which is easier to learn, *i.e.* set all the weights to zero, $\tilde{\mathbf{w}} = 0$.

Nonetheless the reason why residual networks can be efficiently trained is still largely unknown.

One line of research studies empirically the problem. Veit et al. [37] interpret the ResNet as a collection af varying size dependent smaller networks and reveal that these small network alleviate the vanishing gradient problem. Deepening this fact Balduzzi et al. [38] have shown that, in residual networks, the gradients decay sublinearly, in contrast to the exponential decay in feedforward NN. Moreover Li et al. [39] visualize the loss landscape discovering that skip connections yield smoother optimization surfaces.

The other line of research investigates the ResNets theoretically. Hardt et al. [40] show that a linear residual network has no spurious local optima, *i.e.* local optima that yield larger loss values than the global optima. Also Li et al. [41] has proven the absence of spurious local optima and saddle points, for a two-layers ResNet, of which only one unknown, trained using Stochastic Gradient Descent. They are also able to characterize the local convergence of SGD around the global optimum.

However these theoretical results are often considered too optimistic since they are based on oversimplified assumptions, *e.g.* that results obtained for linear residual networks can be well approximate the nonlinear cases.

To conclude we add fig. A.4 and fig. A.5. These are respectively the contour plots and the surface plots of the loss landscapes correspondent to the baseline and the residual architectures, for the problem previously discussed.

The residual network's landscape presents a clearer and deeper mimimum. Note that the loss axis, *i.e.* z axis, is log-scaled. The visualization is obtained with the method proposed by Li et al. [39], *i.e. filter-wise normalization*, along the first two principal components directions in the parameter space.

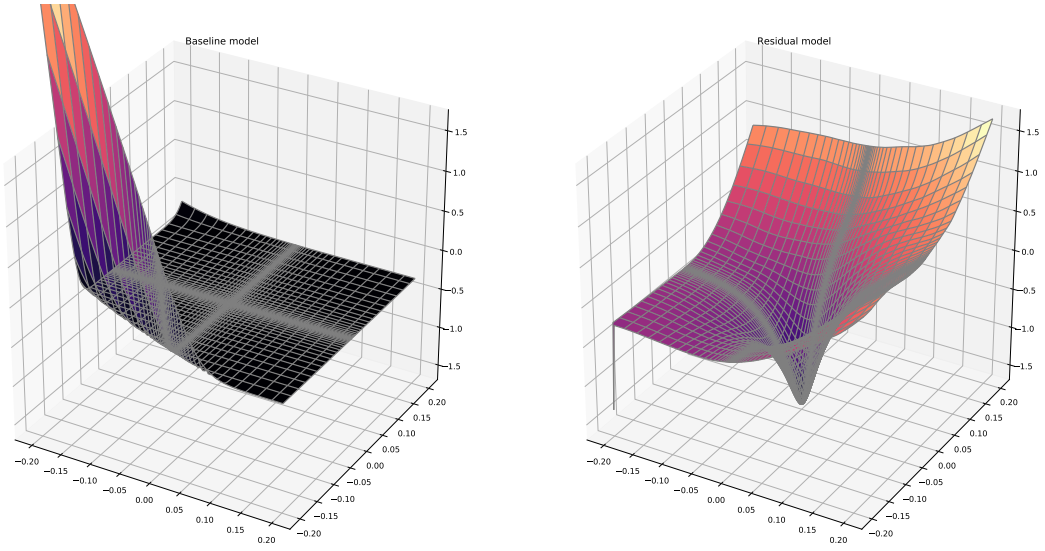


Figure A.4.: Surface plots of the loss landscape for the models with 20 blocks

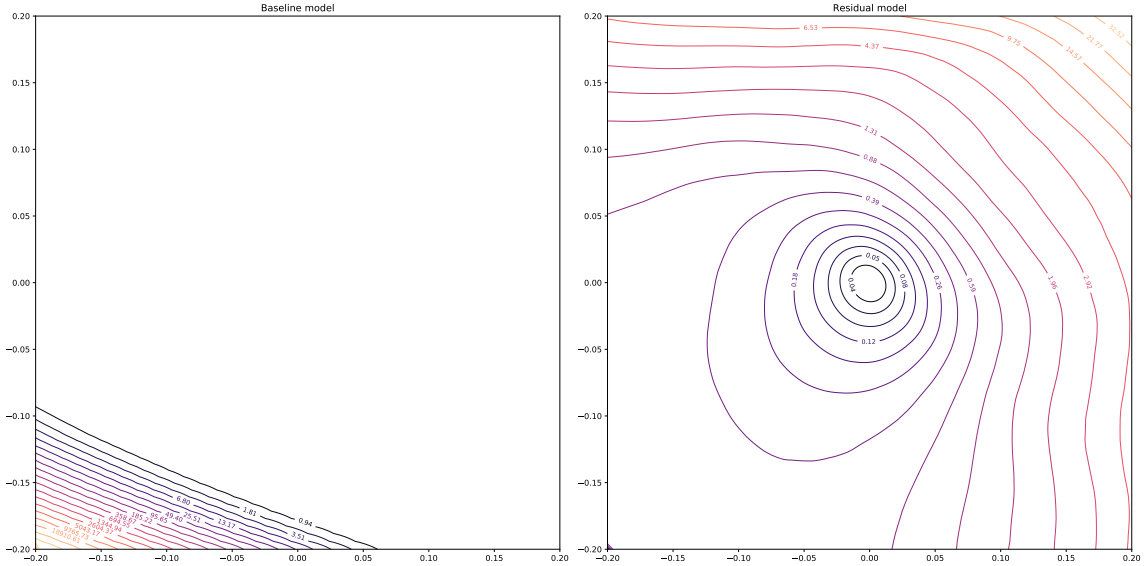


Figure A.5.: Contour plots of the loss landscape for the models with 20 blocks

B. Practical background on Neural Networks

In this chapter of the Appendix we briefly expose the practical notions employed in this work by arguments: the activation functions applied, the metrics used in the networks' trainings and the algorithm utilized for the optimization.

B.1. Activation functions

Recall the definition eq. (2.2) of *activation*:

$$a_j = \sum_{i=1}^{D} w_{ji} x_i + w_{j0} = \sum_{i=0}^{D} w_{ji} x_i \tag{B.1}$$

and the consequent definition eq. (2.3) of *activation function*:

$$z_j = h(a_j) \tag{B.2}$$

There are several choices for the function h. Predicting in advance which one will work best is sometimes difficult. A general property required to be a proper activation function is differentiability. This is a basic requirement to allow the gradient-based optimization algorithm to work fine. Some activation functions are piece-wise differentiable having only a small number of nondifferentiable points. In practice software implementations of neural network usually return one of the one-sided derivatives rather than raising an error.

B.1.1. Rectified Linear Units

The *Rectified Linear Unit*, or *ReLU* (Jarrett et al. [30]; Nair et al. [31]), is an excellent default choice for hidden units. The activation function is defined as:

$$h(a_j) = max\{0, a_j\} \tag{B.3}$$

Since it is similar to linear units, *i.e.* units with activation function $h(a_j) = a_j$, it shares its strengths with it. Indeed when the unit is active, *i.e.* in the positive domain, the derivatives through a ReLU remain large and consistent. Moreover the second derivative is 0 *a.e.* This leads to far more useful gradient directions since no second-order effects are introduced. Some generalization of rectified linear units have been developed to address one drawback: they cannot learn via gradient-based methods on examples for which their output is zero, *i.e.* the activation of the unit is negative. These generalizations guarantee thet they receive

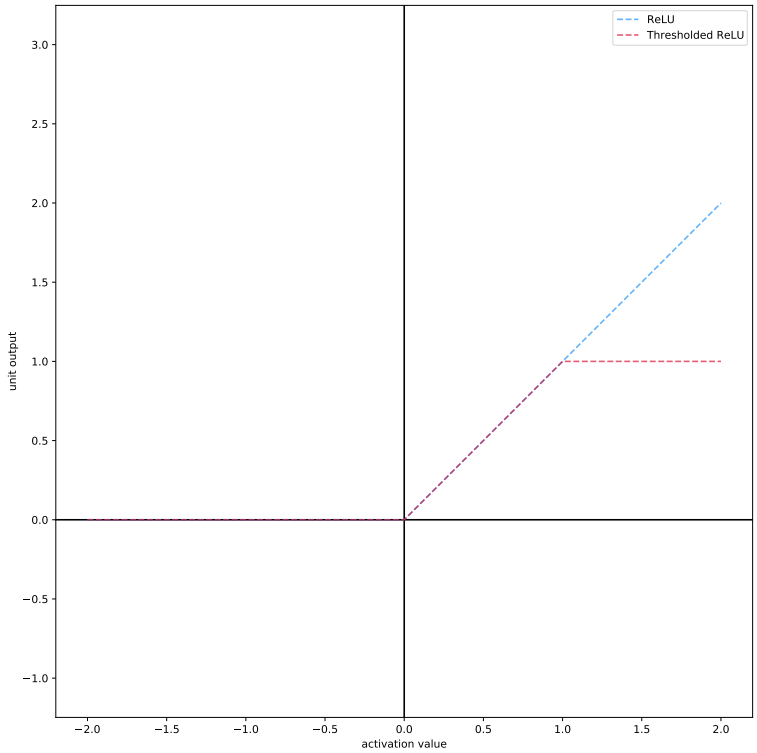


Figure B.1.: ReLU activation function and its thresholded version

gradient everywhere, for instance using a non-zero slope for negative activations, *i.e.* $a_j < 0$. In fig. B.1 we can see the ReLU activation function and a version of it featuring the maximum output value set to 1.

B.1.2. Sigmoidal activation functions

The *logistic activation function* is defined as:

$$h(a_j) = \sigma(a_j) = \frac{1}{1 + e^{-a_j}} \tag{B.4}$$

This type of unit is very useful to output a probability, since it maps $\mathbb{R}$ in $(0, 1)$. The problem is that this function saturates to 1 or 0 across most of its domain. This widespread saturation, characteristic of sigmoidal units, can make gradient-based learning very difficult, *i.e.* the derivative is often numerically zero. That is why its usage in hidden units is discouraged, while, with an appropriate loss function, which can undo the saturation, it is compatible with gradient-based optimization. When the usage of a sigmoidal activation function, in hidden layers, is compulsory, it is preferable to employ the *hyperbolic tangent activation function* defined as:

$$h(a_j) = \tanh(a_j) = \frac{e^{a_j} - e^{-a_j}}{e^{a_j} + e^{-a_j}} \tag{B.5}$$

This is related to the logistic activation function by: $\tanh(x) = 2\sigma(2x) - 1$. It resembles better an identity activation function $h(a_j) = a_j$ since it passes through

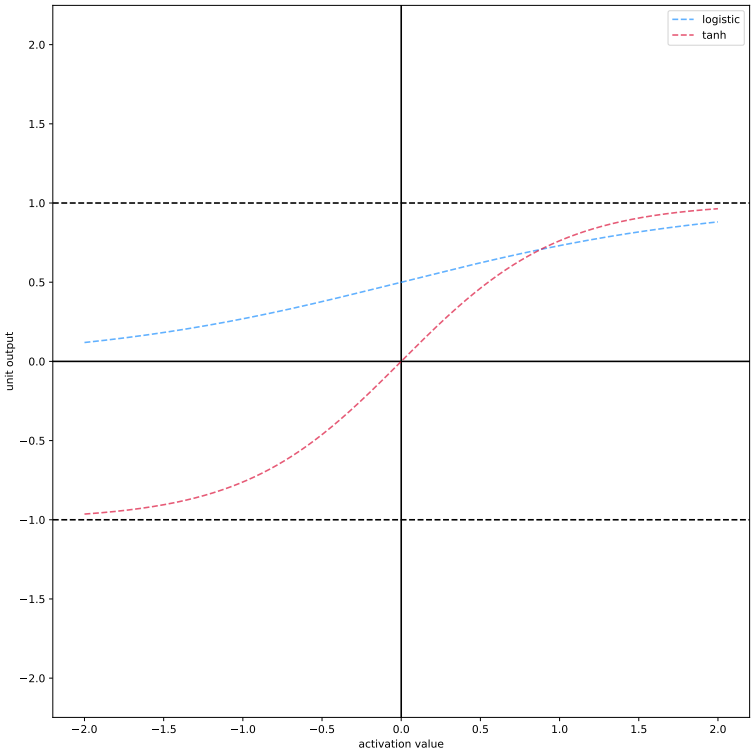


Figure B.2.: Sigmoidal activation functions

the origin, making the training of the network easier. Both activation functions can be seen in fig. B.2.

B.2. Optimization algorithms

In this section we briefly present the algorithms and related techniques used in this work. This overview is mostly taken from Goodfellow et al. [12] chapter 8 and articles directly related to the topics.

B.2.1. Optimization with *momentum*

The method of momentum (Polyak [32]) is designed to accelerate learning.

This algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction. It does so introducing a variable $\mathbf{v}$ that mimics the velocity, *i.e.* the direction and speed at which the parameters $\mathbf{w}$ move through parameter space during the optimization.

The name *momentum* derives from the physical analogy: considering a particle of unitary mass we have that $\mathbf{v}$ may be regarded as its momentum.

The velocity is set to an exponential decaying average over the mini-batch of the negative gradient. A hyperparameter $\alpha \in [0, 1)$ determines how quickly the contributions of previous gradients vanish. The correspondent update rule is:

$$\begin{aligned} \mathbf{v}^{(\tau+1)} &= \alpha \mathbf{v}^{(\tau)} - \frac{\epsilon}{N} \sum_{n=1}^{N} \nabla e_n(\mathbf{w}^{(\tau)}) \\ \mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} + \mathbf{v}^{(\tau+1)} \end{aligned} \tag{B.6}$$

where N is the mini-batch size. The larger α is relative to ϵ, the more previous gradient affect current direction. When many subsequent gradients point in the same direction the step size may grow bigger, hence accelerating the optimization process.

Typically α follows an increasing schedule, while ϵ a decreasing one during learning.

B.2.2. Optimization with *Adaptive* learning rates

Learning rate is one of the most difficult hyperparameter to set because it directly affects the learning performance.

Observing that the loss function is often highly sensitive to some directions in the parameter space and insensitive to others, and supposing that the directions of sensitivity are somehow axis aligned, it could make sense to use separate learning rate for each parameter. These learning rate automatically adapt througout the course of optimization.

B.2.3. *Adam* method

Adam[1] algorithm by Kingma et al. [13] combines adaptive learning rates with momentum.

This method was designed to merge the advantages of two other, previously defined, algorithms: *AdaGrad* (Duchi et al. [33]), which works well with sparse gradients, and *RMSProp* (Tieleman et al. [34]) which works well in online and non-stationary settings.

One of the main differences with the latter algorithm is that in this case the momentum is incorporated directly as an estimate of the first-order moment (with exponential weighting) of the gradient. Moreover Adam includes also bias corrections to the estimates of both first-order and (uncentered) second-order moments to account for their initialization at the origin.

We report here the gradient's first moment $\mathbf{m}^{(\tau)}$ and second moment $\mathbf{v}^{(\tau)}$ update rule:

$$\begin{aligned} \mathbf{m}^{(\tau+1)} &= \beta_1 \mathbf{m}^{(\tau)} + (1-\beta_1)\nabla e(\mathbf{w}^{(\tau)}) \\ \mathbf{v}^{(\tau+1)} &= \beta_2 \mathbf{v}^{(\tau)} + (1-\beta_2)\nabla^2 e(\mathbf{w}^{(\tau)}) \end{aligned} \tag{B.7}$$

where the hyper-parameters $\beta_1, \beta_2 \in [0, 1)$ control the exponential decay rates of these moving averages. As previously mentioned, $\mathbf{m}^{(0)} = \mathbf{v}^{(0)} = 0$.

Adam take this biasing toward zero into account providing a bias correction:

[1] The name derives from the phrase "adaptive moments".

$$\begin{aligned}\hat{\mathbf{m}}^{(\tau)} &= \frac{\mathbf{m}^{(\tau)}}{1-\beta_1^{\tau}} \\ \hat{\mathbf{v}}^{(\tau)} &= \frac{\mathbf{v}^{(\tau)}}{1-\beta_2^{\tau}}\end{aligned} \tag{B.8}$$

Finally the parameters update is given by:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \alpha\frac{\hat{\mathbf{m}}^{(\tau+1)}}{\epsilon + \sqrt{\hat{\mathbf{v}}^{(\tau+1)}}} \tag{B.9}$$

where α called *step-size* approximately bound the effective magnitude of the steps taken in the parameter space at each timestep, *i.e.* $\left|\Delta\mathbf{w}^{(\tau)}\right| \lessapprox \alpha$. This can be understood as establishing a *trust region* around the current parameter value, beyond which the informations carried by the current gradient are not sufficient. ϵ is a small constant for numerical stabilization.

Adam is generally regarded as being fairly robust to the choice of hyperparameters. Default settings, suggested by experimental results, are: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

B.2.4. Batch Normalization

Batch normalization (Ioffe et al. [35]) is not an optimization algorithm, instead it is a method of adaptive reparameterization, motivated by the difficulty of training some very deep models. Indeed the choice of a proper learning rate is made difficult because the effects of an update to the parameters for one layer depend so strongly on all the other layers.

Consider a small step in the parameter space $\delta\mathbf{w}$. In a first-order approximation this leads to a correspondent update of the error function: $\delta e \approx (\delta\mathbf{w})^{\mathsf{T}}\nabla e(\mathbf{w})$. Therefore applying an update rule based only on the gradient: $\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla e(\mathbf{w}^{(\tau)})$, *i.e.* in this case $\delta\mathbf{w} = -\eta\nabla e(\mathbf{w})$, we decrease the loss by $\eta\nabla e(\mathbf{w})^{\mathsf{T}}\nabla e(\mathbf{w})$. Clearly setting the parameter η enables us to control the decrease of the loss function. Such useful control begins to fail when the first-order approximation starts to be too much inaccurate. This fact happens when we have lots of layer with many weights greater than 1. Indeed second and higher order terms depends on the product among the weights belonging to subsequent layers, *i.e.* they take into account the interaction between them.

The reparameterization introduced by batch normalization significantly reduces the problem of coordinating updates across many layers.

Let $\mathbf{A}$ be a mini-batch of activations, arranged in a design matrix, of the layer (hidden or input) that we want to normalize. To do so we replace it with $\tilde{\mathbf{A}}$ defined as:

$$\tilde{\mathbf{A}}_{ij} = \frac{\mathbf{A}_{ij} - \boldsymbol{\mu}_j}{\boldsymbol{\sigma}_j} \tag{B.10}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are the average and deviation statistics computed over the mini-batch, *i.e.* for a size-N batch:

$$\begin{aligned} \boldsymbol{\mu} &= \frac{1}{N}\sum_{n=1}^{N} \mathbf{A}_{n:} \\ \boldsymbol{\sigma} &= \sqrt{\epsilon + \frac{1}{N}\sum_{n=1}^{N}(\mathbf{A}_{n:} - \boldsymbol{\mu})^2} \end{aligned} \tag{B.11}$$

ϵ is a small constant such as 10^{-8} to avoid the computation of the square root of zero. In this way learning the model is now very simple because the parameters at the lower layers do not have an effect in most cases.

The actual implementation of batch normalization replace the $\tilde{\mathbf{A}}_{ij}$ with $\boldsymbol{\beta}_j + \boldsymbol{\gamma}_j \tilde{\mathbf{A}}_{ij}$ for each i. This is done to avoid the reduction of the expressive power of the network, while mantaining it easy to learn with gradient descent. Indeed the mean of $\mathbf{A}$ was determined by a complex interactions among the parameters in the layers below, whereas now it is solely determined by $\boldsymbol{\beta}$. At test time $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ may be replaced by running averages that were stored during training time.

Experimental results show that batch normalization owns a regularization effect as well.

B.3. Metrics

In this section we simply provide definitions and formulas of the performance metrics used in this work. In the following we fix the number of samples per batch to N. Moreover we name $\mathbf{y}$ the output of the network, *i.e.* the estimated value correspondent to the couple input-target $(\mathbf{x}, \mathbf{t})$. When the metric is used as loss function its evaluation is performed batch-by-batch, whereas if we are interested in the metric's value of an epoch we have to apply a further average over the mini-batches.

B.3.1. Binary cross-entropy

The *binary cross-entropy* is a probabilistic metric utilized in the task of binary classification. It is probabilistic in the sense that its inputs, *i.e.* estimated and true labels, are probabilities. In this case t and y are scalars correspondent to a unique output unit. To avoid the problem exposed in section B.1.2 about sigmoidal activation functions, which output probabilities, in *Tensorflow* the actual computation of the metric is done before applying the logit sigmoid. As we have seen in section 2.2 minimizing the binary cross-entropy corresponds to maximizing the log-likelihood function of a Bernoulli distribution. The general definition of the binary cross-entropy loss function is:

$$e(t, y) = -\frac{1}{N}\sum_{n=1}^{N}[y_n \log(t_n) + (1 - y_n)\log(1 - t_n)] \tag{B.12}$$

The actual *Tensorflow* implementation provides a stable version:

$$e(t, y) = -\frac{1}{N}\sum_{n=1}^{N}[max\{z_n, 0\} - z_n t_n + \log(1 + e^{-|z_n|})] \tag{B.13}$$

where z_n is the activation of the output unit, *i.e.* the output value before sigmoidal trasformation is applied. In *Keras* we can apply the metric either to the linear activation or to the sigmoidal output. In this latter case a logit[2] transformation in computed before applying the Tensorflow implementation of the metric in eq. (B.13).

B.3.2. Binary accuracy, precision, recall and AUC

We define together these performance metrics that we utilize for the binary classification task. We use this straightforward notation to denote the following binary quantities: *true positives* (TP), *true negatives* (TN), *false positive* (FP) and *false negatives* (FN). We take their meanings for granted. Hence we have the following formulae:

- *Accuracy*. $ACC = \frac{TP+TN}{TP+TN+FP+FN}$, it is the frequency with which y matches the target t.
- *Precision* or *Positive Predicted Values*. $PPV = \frac{TP}{TP+FP}$, it represents how many of the samples which have been estimated positive are truly positive.
- *Recall* or *Sensitivity* or *True Positive Rate*. $TPR = \frac{TP}{TP+FN}$, it represents how many of the positive samples present in the training or test set are truly detected as positive.
- *Area Under the Curve* (*AUC*). In this work we use the area under the *Precision-Recall* (*PR*) curve, *i.e.* the integral of the curve in the PR plane in which each point represents precision and recall values for a given classification threshold c. It well represents the trade-off between precision and recall and summarise these two metrics. Moreover it remains meaningful even though the dataset is class unbalanced. What is computed during training is an approximation of the curve.

The latter two metrics are referred to the positive class. They can be computed with respect to the negative class in a similar faschion.

B.3.3. Mean Squared Error

The *Mean Squared Error* (*MSE*) is the straightforward metric obtained via the gaussian distribution model of the error described in section 2.2 for regression purpose. Indeed it consists of the average over the sample in the batch of the squared error, in the case in which we ignore the covariance structure. For outputs of length P the *Keras* implementation provides:

[2] $z_n = \log\left(\frac{y_n}{1-y_n}\right)$

$$
\begin{aligned}
e(\mathbf{t}, \mathbf{y}) &= \frac{1}{N} \sum_{n=1}^{N} \|\mathbf{t}_n - \mathbf{y}_n\|_2^2 \\
&= \frac{1}{N} \sum_{n=1}^{N} (\mathbf{t}_n - \mathbf{y}_n)^\top (\mathbf{t}_n - \mathbf{y}_n) \\
&= \frac{1}{NP} \sum_{n=1}^{N} \sum_{p=1}^{P} (t_{np} - y_{np})^2
\end{aligned}
\tag{B.14}
$$

In our work, in the model described in section 3.2, the output of the regression has two channel. We construct a proper loss function simply averaging over the two channels indicated by the superscript:

$$
\begin{aligned}
e(\mathbf{t}^{(1)}, \mathbf{y}^{(1)}, \mathbf{t}^{(2)}, \mathbf{y}^{(2)}) &= \frac{1}{2N} \sum_{n=1}^{N} \left[\left\| \mathbf{t}_n^{(1)} - \mathbf{y}_n^{(1)} \right\|_2^2 + \left\| \mathbf{t}_n^{(2)} - \mathbf{y}_n^{(2)} \right\|_2^2 \right] \\
&= \frac{1}{2N} \sum_{n=1}^{N} [(\mathbf{t}_n^{(1)} - \mathbf{y}_n^{(1)})^\top (\mathbf{t}_n^{(1)} - \mathbf{y}_n^{(1)}) + (\mathbf{t}_n^{(2)} - \mathbf{y}_n^{(2)})^\top (\mathbf{t}_n^{(2)} - \mathbf{y}_n^{(2)})] \\
&= \frac{1}{2NP} \sum_{n=1}^{N} \sum_{p=1}^{P} [(t_{np}^{(1)} - y_{np}^{(1)})^2 + (t_{np}^{(2)} - y_{np}^{(2)})^2]
\end{aligned}
\tag{B.15}
$$

B.3.4. Mean Absolute Error

In this work we use the *Mean Absolute Error* (*MAE*) as an additional metric to check the goodness of the training, we does not employ it as loss function. MAE is known to be robust to outliers, *i.e.* in this case samples for which the difference $|\mathbf{t}_n - \mathbf{y}_n|$ is high. The *Keras* formula is:

$$
e(\mathbf{t}, \mathbf{y}) = \frac{1}{NP} \sum_{n=1}^{N} \sum_{p=1}^{P} |t_{np} - y_{np}| \tag{B.16}
$$

As for the MSE we implement the two-channel version:

$$e(\mathbf{t}^{(1)}, \mathbf{y}^{(1)}, \mathbf{t}^{(2)}, \mathbf{y}^{(2)}) = \frac{1}{2NP} \sum_{n=1}^{N} \sum_{p=1}^{P} [|t_{np}^{(1)} - y_{np}^{(1)}| + |t_{np}^{(2)} - y_{np}^{(2)}|] \tag{B.17}$$

B.3.5. Cosine similarity

The *cosine similarity* is a measure of similarity between two vectors. In particular, as the name suggests, it is equal to the cosine of the angle between the two vectors. This fact shows how this metric takes into account only the alignement of the two vectors neglecting their magnitudes and it is comprised in the interval $[-1, 1]$. The formula is:

$$cs(\mathbf{t}, \mathbf{y}) = \frac{1}{N} \sum_{n=1}^{N} \frac{< \mathbf{t}_n, \mathbf{y}_n >}{\|\mathbf{t}_n\|_2 \|\mathbf{y}_n\|_2} \tag{B.18}$$

where $< \cdot, \cdot >$ denotes the usual scalar product in $\mathbb{R}^P$. When we use it as a loss function we obviously want to minimize the distance, *i.e.* the negative cosine similarity. In *Keras* the implemented loss function is:

$$e(\mathbf{t}, \mathbf{y}) = -\frac{1}{N} \sum_{n=1}^{N} \|\mathbf{t}_n\|_2 \|\mathbf{y}_n\|_2 \tag{B.19}$$

Also in this case, when dealing with a two channels output, we sum the metric over them leading to a metric which outputs values in $[-2, 2]$.

Bibliography

[1] P. Cano, E. Batlle, T. Kalker, and J. Haitsma. "A Review of Audio Fingerprinting". In: *Journal of VLSI signal processing systems for signal, image and video technology* 41 (2005), pp. 271–284 (cit. on p. 1).

[2] R. Cerquides. "A Real Time Audio Fingerprinting System for Advertisement Tracking and Reporting in FM Radio". In: (May 2007), pp. 1–4. DOI: 10.1109/RADIOELEK.2007.371456 (cit. on p. 1).

[3] J. Haitsma and T. Kalker. "A Highly Robust Audio Fingerprinting System With an Efficient Search Strategy". In: *Journal of New Music Research* 32 (June 2003), pp. 211–221. DOI: 10.1076/jnmr.32.2.211.16746 (cit. on p. 1).

[4] J.-q. Ouyang, H. Nie, M. Zhang, Z. li, and Y. Li. "Fusing Audio-Visual Fingerprint to Detect TV Commercial Advertisement". In: *Comput. Electr. Eng.* 37.6 (Nov. 2011), pp. 991–1008. ISSN: 0045-7906. DOI: 10.1016/j.compeleceng.2011.08.004. URL: https://doi.org/10.1016/j.compeleceng.2011.08.004 (cit. on p. 1).

[5] A. L. Wang. "An industrial-strength audio search algorithm". In: (2003), pp. 7–13 (cit. on p. 1).

[6] M. Dong. "Convolutional Neural Network Achieves Human-level Accuracy in Music Genre Classification". In: *CoRR* abs/1802.09697 (2018). arXiv: 1802.09697. URL: http://arxiv.org/abs/1802.09697 (cit. on p. 1).

[7] D. de Benito, A. Lozano-Diez, D. Toledano, and J. Gonzalez-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* 2019 (June 2019). DOI: 10.1186/s13636-019-0152-1 (cit. on p. 2).

[8] Z. Yu, X. Xu, X. Chen, and D. Yang. "Learning a Representation for Cover Song Identification Using Convolutional Neural Network". In: *CoRR* abs/1911.00334 (2019). arXiv: 1911.00334. URL: http://arxiv.org/abs/1911.00334 (cit. on p. 2).

[9] Y. Aytar, C. Vondrick, and A. Torralba. "Soundnet: Learning sound representations from unlabeled video". In: (2016) (cit. on pp. 2, 19, 21).

[10] B. Meléndez-Catalán. "RELATIVE MUSIC LOUDNESS ESTIMATION USING TEMPORAL CONVOLUTIONAL NETWORKS AND A CNN FEATURE EXTRACTION FRONT-END". In: (2020) (cit. on pp. 3, 23, 25–27, 54).

[11] C. M. Bishop. *Pattern Recognition and Machine Learning*. http://research.microsoft.com/en-us/um/people/cmbishop/prml/. Springer, 2006 (cit. on p. 4).

[12] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on pp. 4, 10, 15, 67).

[13] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization." In: *CoRR* abs/1412.6980 (2014). URL: http://dblp.uni-trier.de/db/journals/corr/corr1412.html#KingmaB14 (cit. on pp. 8, 68).

[14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (1986), pp. 533–536 (cit. on p. 9).

[15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html (cit. on p. 11).

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on p. 12).

[17] Y. LeCun. "Generalization and Network Design Strategies". In: (1989). Ed. by R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels (cit. on p. 12).

[18] Y. Zhou and R. Chellappa. "Computation of optical flow using a neural network". In: *IEEE 1988 International Conference on Neural Networks* (1988), 71–78 vol.2 (cit. on p. 15).

[19] S. Bai, J. Z. Kolter, and V. Koltun. "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling." In: *CoRR* abs/1803.01271 (2018). URL: http://arxiv.org/abs/1803.01271 (cit. on pp. 15, 16, 18).

[20] J. Long, E. Shelhamer, and T. Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *CoRR* abs/1411.4038 (2014). URL: http://arxiv.org/abs/1411.4038 (cit. on p. 15).

[21] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). URL: http://arxiv.org/abs/1512.03385 (cit. on p. 17).

[22] Q. Lemaire and A. Holzapfel. "Temporal Convolutional Networks for Speech and Music Detection in Radio Broadcast". In: (2019) (cit. on p. 18).

[23] D. Snyder, G. Chen, and D. Povey. "MUSAN: A Music, Speech, and Noise Corpus". In: *CoRR* abs/1510.08484 (2015). URL: http://arxiv.org/abs/1510.08484 (cit. on p. 36).

[24] F. Chollet et al. *Keras*. https://keras.io. 2015 (cit. on p. 38).

[25] Martı n Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (cit. on p. 38).

[26] L. McInnes, J. Healy, and J. Melville. "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction". In: (2020). arXiv: 1802.03426 [stat.ML] (cit. on p. 38).

[27] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization". In: (2017), pp. 618–626. DOI: 10.1109/ICCV.2017.74 (cit. on p. 54).

[28] G. Chechik, V. Sharma, U. Shalit, and S. Bengio. "Large Scale Online Learning of Image Similarity Through Ranking". In: *J. Mach. Learn. Res.* 11 (Mar. 2010), pp. 1109–1135. ISSN: 1532-4435 (cit. on p. 55).

[29] F. Schroff, D. Kalenichenko, and J. Philbin. "FaceNet: A Unified Embedding for Face Recognition and Clustering". In: *CoRR* abs/1503.03832 (2015). arXiv: 1503.03832. URL: http://arxiv.org/abs/1503.03832 (cit. on p. 55).

[30] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. "What is the best multi-stage architecture for object recognition?" In: (2009), pp. 2146–2153. URL: http://dblp.uni-trier.de/db/conf/iccv/iccv2009.html#JarrettKRL09 (cit. on p. 65).

[31] V. Nair and G. E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: (2010), pp. 807–814 (cit. on p. 65).

[32] B. T. Polyak. "Some methods of speeding up the convergence of iteration methods". In: *Ussr Computational Mathematics and Mathematical Physics* 4 (1964), pp. 1–17 (cit. on p. 67).

[33] J. Duchi, E. Hazan, and Y. Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12.61 (2011), pp. 2121–2159. URL: http://jmlr.org/papers/v12/duchi11a.html (cit. on p. 68).

[34] T. Tieleman, G. Hinton, et al. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31 (cit. on p. 68).

[35] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). URL: http://arxiv.org/abs/1502.03167 (cit. on p. 69).

[36] B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996 (cit. on p. 59).

[37] A. Veit, M. J. Wilber, and S. J. Belongie. "Residual Networks are Exponential Ensembles of Relatively Shallow Networks". In: *CoRR* abs/1605.06431 (2016). URL: http://arxiv.org/abs/1605.06431 (cit. on p. 63).

[38] D. Balduzzi, M. Frean, L. Leary, J. P. Lewis, K. W.-D. Ma, and B. McWilliams. "The Shattered Gradients Problem: If resnets are the answer, then what is the question?" In: *CoRR* abs/1702.08591 (2017). URL: http://arxiv.org/abs/1702.08591 (cit. on p. 63).

[39] H. Li, Z. Xu, G. Taylor, and T. Goldstein. "Visualizing the Loss Landscape of Neural Nets". In: *CoRR* abs/1712.09913 (2017). URL: http://arxiv.org/abs/1712.09913 (cit. on p. 63).

[40] M. Hardt and T. Ma. "Identity Matters in Deep Learning". In: *CoRR* abs/1611.04231 (2016). URL: http://arxiv.org/abs/1611.04231 (cit. on p. 63).

[41] Y. Li and Y. Yuan. "Convergence Analysis of Two-layer Neural Networks with ReLU Activation". In: (2017), pp. 597–607. URL: http://papers.nips.cc/paper/6662-convergence-analysis-of-two-layer-neural-networks-with-relu-activation (cit. on p. 63).

Andrea Caceffo
Towards advertisement clustering: two approaches based on Neural Networks - 2020-21

Aknowledgements:
The first and biggest thanks goes to my family for continuously, and by all means, supporting me in this, rather long, journey at "Politecnico di Milano".
Thanks to all my friends for always backing me up, each one in his own way.
Thanks all the classmates who have shared with me parts of the universitiy's struggles. In particular thank you Artem, best teammate ever, for the funny, and also the difficult, times faced together.
Finally thanks to prof. Antonacci and MakarenaLabs for giving me the opportunity to spend my Thesis on these interesting and exciting arguments. In particular I wish to thank Guglielmo for his availability and the passion for these topics transmitted to me.

Andrea Caceffo
Towards advertisement clustering: two approaches based on Neural Networks - 2020-21