الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي

UNIVERSITE BADJI MOKHTAR - ANNABA
BADJI MOKHTAR – ANNABA UNIVERSITY

جامعة باجي مختار – عنابـــــــــــــة

Faculté : Sciences de L'ingéniorat
Département : Electronique
Domaine : Sciences et Techniques
Filière : Télécommunication
Spécialité : Réseaux et Télécommunication

Mémoire

Présenté en vue de l'obtention du Diplôme de Master

Thème :

Convolutional time-domain audio separation network: case of bird sounds

Présenté par : *Abour Abdennour*
Aloui Mohamed Lamine

Encadrant : *A. Boulmaiz* *MCB* *UBM ANNABA*

Jury de Soutenance :

YAHI A.	MCB	UBM Annaba	Président
BOULMAIZ A.	MCB	UBM Annaba	Encadrant
ZERMI N.	MCA	UBM Annaba	Examinateur

Année Universitaire : 2020/2021

Acknowledgement

First of all, we would like to thank the director of this thesis, Mrs. BOULMAIZ Amira, for trusted us, guided us, encouraged us and advised us. We are also grateful to her for the considerable time she gave us, her pedagogical and scientific qualities, her frankness and her sympathy. we have learned a lot at his side and we are grateful to her for all that.

We address sincere thanks to Mrs. ZERMI N. and

Mrs. YAHI A. for having done us the honor of participating in the jury of defense

Finally, we would like to thank all the people who participated in our research and in the elaboration of this thesis

Dedications

May this work bear witness to my respects to my parents,

My brothers and sisters and all my family

Thanks to their encouragement and their great sacrifices. No dedication could express my respect, consideration and deep feelings towards them. I pray to ALLAH to bless them, to watch over them, hoping that they will always be proud of me.

To all my teachers

Their generosity and support obliges me to show them my deep respect and my loyal consideration.

To my friends and colleagues

And to all those who are dear to me

They will find here the expression of a fidelity and an infinite friendship, of my feelings of gratitude for the support that they did not cease to carry to me.

Find in this modest work my sincere gratitudes and recognition.

This work is yours.

Abdennour et Mohamed Lamine

Abstract

Birds are strong indicator for the health of an ecosystem. It is one of the most endangered animal species in the world. This make birds very important and need special attention. Identify different species and detect bird sounds in mixture recording help to preserve these species from extinction and therefore preserve the balance of the ecosystem.

In this thesis, we propose to apply a fully-convolutional neural network system in time-domain named (Conv-TasNet) for bird sounds separation. The studied system is composed of three main blocks: an encoder, a separation model based on the neural network CNN 1D and a decoder

Experimental results for the bird sounds separation, demonstrate that the methodology adopted to separate bird sounds is effective and works satisfactorily. In addition, separation performance and environmental noise immunity are significantly improved after the application of the Conv-TasNet method, indicating that it is an appropriate approach for acoustic recognition of birds in complex environments.

Key words: Bird sounds separation, convolutional neural network, CNN, Conv-TasNet, deep learning, time-domain.

ملخص :

الطيور هي مؤشر قوي على صحة النظام البيئي. إنها واحدة من أكثر أنواع الحيوانات المهددة بالانقراض في العالم. هذا يجعل الطيور مهمة للغاية وتحتاج إلى عناية خاصة. تحديد الأنواع المختلفة واكتشاف أصوات الطيور في تسجيل المزيج يساعد في الحفاظ على هذه الأنواع من الانقراض وبالتالي الحفاظ على توازن النظام البيئي.

في هذه الأطروحة، نقترح تطبيق نظام شبكة عصبية ملتفة في المجال الزمني يسمى (Conv-TasNet) لفصل أصوات الطيور. يتكون النظام المدروس من ثلاث عناصر رئيسية: جهاز التشفير، ونموذج فصل يعتمد على الشبكة العصبية (CNN 1D) و جهاز فك التشفير.

أظهرت النتائج التجريبية لفصل أصوات الطيور أن المنهجية المعتمدة لفصل أصوات الطيور فعالة وتعمل بشكل مرض. بالإضافة إلى ذلك، تم تحسين أداء الفصل والحصانة من الضوضاء البيئية بشكل ملحوظ بعد تطبيق طريقة Conv-TasNet، مما يشير إلى أنه نهج مناسب للتعرف الصوتي للطيور في البيئات المعقدة.

الكلمات المفتاحية : فصل أصوات الطيور، الشبكة العصبية الملتفة، CNN، Conv-TasNet، التعلم العميق، المجال الزمني.

Résumé

Les oiseaux sont de puissants indicateurs de la santé d'un écosystème. C'est l'une des espèces animales les plus menacées au monde. Cela rend les oiseaux très importants et nécessite une attention particulière. L'identification des différentes espèces et la détection des sons d'oiseaux dans l'enregistrement du mélange aident à préserver ces espèces de l'extinction et donc à préserver l'équilibre de l'écosystème.

Dans cette thèse, nous proposons d'appliquer un système de réseau de neurones entièrement convolutionnel dans le domaine temporel nommé (Conv-TasNet) pour la séparation des sons d'oiseaux. Le système étudié est composé de trois blocs principaux : un encodeur, un modèle de séparation basé sur le réseau de neurones CNN 1D et un décodeur.

Les résultats expérimentaux pour la séparation des sons d'oiseaux, démontrent que la méthodologie adoptée pour séparer les sons d'oiseaux est efficace et fonctionne de manière satisfaisante. De plus, les performances de séparation et l'immunité au bruit environnemental sont considérablement améliorées après l'application de la méthode Conv-TasNet, ce qui indique qu'il s'agit d'une approche appropriée pour la reconnaissance acoustique des oiseaux dans des environnements complexes.

Mots clés : Séparation des sons d'oiseaux, réseau de neurones convolutifs, CNN, Conv-TasNet, apprentissage profond, domaine temporel.

List of Figures

List of Tables

List of Symbols

$\mathbf{A}$	Mixing matrix
$*$	Conjugation
$\{\bullet\}$	Indicator function
$\times$	Multiplication
$\forall$	For all
$\boldsymbol{\nabla}$	Gradient
$\odot$	Element-wise product
$\otimes$	Convolution
$\sum_{i=1}^{N}$	Summation over discrete index i from 1 to N Transpose
v	Vector
$\boldsymbol{v_i}$ *or* $\boldsymbol{v(i)}$	Element of a vector
$\boldsymbol{a}^{(\boldsymbol{L})}$	Activation function matrix for the L-th layer
$\boldsymbol{a}_{\boldsymbol{i}}^{(\boldsymbol{L})}$	Activation function in the L-th layer for the i-th neuron
α	Learning rate
$\tilde{\alpha}$	Average room absorption coefficient
$\boldsymbol{b}$	Bias vector
$\boldsymbol{b}^{(\boldsymbol{c})}$	Bias vector of a memory cell of a LSTM
$\boldsymbol{b}^{(\boldsymbol{L})}$	Bias vector for the L-th layer of a fully-connected network
$\boldsymbol{b}^{(\boldsymbol{f})}$	Bias vector for the forget gate of a LSTM
$\boldsymbol{B_s}$	Batch size
$\boldsymbol{\beta}$	Sparsity penalty term
$\boldsymbol{c}$	sound wave velocity when travelling through the air
$\boldsymbol{\gamma}$	Phase
$\boldsymbol{F}$	Number of frequency bins
$\boldsymbol{e}_{artifacts}$	Artifacts error

$\boldsymbol{e}_{interference}$	Interference error
$\boldsymbol{e_{noise}}$	Noise errorN
$\boldsymbol{f_{crit}}$	F-value critical for the ANOVA test
$\boldsymbol{f_{value}}$	F-value for the ANOVA test
$\boldsymbol{d}$	Size of the irregularities of a surface
$\boldsymbol{d_f(t)}$	Linear correlation coefficient
$\boldsymbol{k}$	Number of neurons in the hidden layer
$\boldsymbol{L}$	Number of layers of a fully-connected network
$\boldsymbol{N}$	Number of DNNs

List of Abbreviations

ASA	**A**uditory **S**cene **A**nalysis
BSD	**B**ird **S**ound **D**ataset
BBS	**B**lind **S**ource **S**eparation
CASA	**C**omputational **A**uditory **S**cene **A**nalysis
CNN	**C**onvolutional **N**eural **N**etwork
Conv-TasNet	**Conv**olutional **T**ime-Domain **A**udio **S**eparation **N**etwork
cLN	**c**umulative **L**ayer **N**ormalization
DPCL	**D**eep **C**lustering
DL	**D**eep **L**earning
DNN	**D**eep **N**eural **N**etwork
DRL	**D**eep **R**einforcement **L**earning
FastICA	**F**ast **I**ndependent **C**omponent **A**nalysis
FC	**F**ully **C**onnected
GRUs	**G**ated **R**ecurrent **U**nit**s**
GMMs	**G**aussian **M**ixture **M**odel**s**
GANs	**G**enerative **A**dversarial **N**etwork**s**
GAP	**G**lobal **A**verage **P**ooling
gLN	**g**lobal **L**ayer **N**ormalization
HMMs	**H**idden **M**arkov **M**odel**s**
HPC	**H**igh **P**erformance **C**omputing
IBMs	**I**deal **B**inary **M**ask**s**
ICA	**I**ndependent **C**omponent **A**nalysis
iSTFT	**i**nverse **S**hort-**T**ime **F**ourier **T**ransform

LLNL	**L**awrence **L**ivermore **N**ational **L**aboratory
LSTM	**L**ong **S**hort-**T**erm **M**emory
ML	**M**achine **L**earning
MLP	**M**ulti-**L**ayer **P**erceptron
NLP	**N**atural **L**anguage **P**rocessing
NMF	**N**on-negative **M**atrix **F**actorization
PReLU	**P**arametric **R**ectified **L**inear **U**nit
Pinv	**P**seudo-**inv**erse
ReLU	**R**ectified **L**inear **U**nit
RNN	**R**ecurrent **N**eural **N**etwork
RAAM	**R**ecursive **A**uto-**A**ssociative **M**emory
RvNN	**R**ecursive **N**eural **N**etwork
RL	**R**epresentation **L**earning
SI-SNR	**S**cale **I**nvariant **S**ource to **N**oise **R**atio
STFT	**S**hort-**T**ime **F**ourier **T**ransform
SDRi	Signal to **D**istortion **R**atio **i**mprovement
SNR	**S**ignal to **N**oise **R**atios
TCN	**T**emporal **C**onvolutional **N**etwork
T-D	**T**ime-**D**omain
TasNet	**T**ime-Domain **A**udio **S**eparation **N**etwork
T-F	**T**ime-**F**requency
TL	**T**ransfer **L**earning
UPGMA	**U**nweighted **P**air **G**roup **M**ethod with **A**rithmetic mean
WSJ0	**W**all **S**treet **J**ournal

Table of contents

GENERAL INTRODUCTION

Birds play a vital role in providing ecosystem functions. However, many birds have been under threat due to human intrusion on their habitat.

Therefore, there is an urgent need to a method that helps experts to protect birds.

There have been many previous approaches identify bird, but they are limited like field observation and identifying bird species from their sound. However, the captured audio recordings are composed of a mixture of different sounds (bird sound, environmental noise...etc)

Several researchers have been interested in the audio separation of speech but according to our research very few have studied the separation of bird sounds

Audio source separation algorithms can be roughly divided into two categories, time frequency (T-F) methods and time-domain (T-D) (or end-to-end) methods. T-F audio source separation algorithms aim to estimate the enhanced spectrum of each individual source from the mixture spectra [1], and then rebuild the wave- form via the inverse short-time Fourier Transform (iSTFT), by combining the enhanced magnitude spectrum with either noisy or the modified phase of the mixture. On the downside, the erroneous estimation of the phase limits the upper bound of separation performance. To solve the phase problem, end-to-end audio source separation methods such as the fully-convolutional time-domain audio separation network (Conv-TasNet) [2] and Wave-u-net [3] are proposed.

In this work, we suggest to use Conv-TasNet model for bird sound separation. The proposed system is based of three processing stages. The first stage consists an encoder module used to transform short segments of the mixture waveform into their corresponding representations. This representation is then used to estimate a multiplicative function (mask) for each source at each time step in the separation stage. Finally, the modified encoder representations are then inverted back to the waveforms using a decoder module [2]. The experiments were performed on a dataset of birds that we composed from xeno-canto.org.

0.1. Objectives:

The main objective of this work is to develop a system based on CNN for bird sound separation. Long term objective in the project is to develop this system until that could identify bird species by their sound in field conditions.

Others objectives are:

- Forming a dataset of bird sounds
- Learn how deep learning and Convolutional neural network work.
- Learn how to train and test a Convolutional neural network.
- Implementing of Conv-TasNet model.

0.2. Thesis structure:

The remainder of the thesis is organized as follows:

Chapter I: Audio separation methods: a state of the art. The first chapter focused on audio source separation and presents the previous methods for separation

Chapter II: Convolutional Neural Network. This chapter is divided into two parts. The first part of this chapter presents the deep learning with her different techniques. The second part is devoted to the different types of DL networks, Especially CNN.

Chapter III: Conv-TasNet for bird sound separation. The third chapter explained the Conv-TasNet method and implement it for bird sound separation.

Chapter IV: Experiments and results. This chapter is focused on the results obtained for our bird sound separation system

CHAPTER I

Audio separation methods: a state of art

I.1. Introduction

Audio source separation methods have recently seen great progress. However, the accuracy, latency, and computational cost of such methods remain insufficient. The majority of the previous methods have formulated the separation problem through the time-frequency representation of the mixed signal, which has several drawbacks, including the decoupling of the phase and magnitude of the signal, the sub-optimality of time-frequency representation for audio source separation, and the long latency in calculating the spectrograms.

Therefore, we investigate end-to-end source separation in the time-domain, which allows modelling phase information and avoids fixed spectral transformations. Due to high sampling rates for audio, employing a long temporal input context on the sample level is difficult, but required for high quality separation results because of long-range temporal correlations.

I.2. Audio source separation

Audio source separation aims to extract individual sources from mixtures of multiple sound sources, e.g. audio source, noise and music. The problem of separating multiple sound sources from sound mixtures is also known as the cocktail party problem [4]: one can imagine a situation in which two friends are in a party, with loud music in the background and other people around talking simultaneously. Humans have an innate ability to separate audio source and sounds in a sound mixture. However, this is not a trivial task for computers.

I.3. Problem Formulation

The audio source separation problem has been considered classically under the framework of Blind Source Separation (BSS), which aims to separate the sources with no (or very little) information about the sources and mixing channels.

In the case of R sound sources, the general problem (without noise) to be solved is:

$$x(\tilde{t}) = \mathbf{A}\, \mathrm{s}(\tilde{t}) \tag{I.1}$$

Where $S(\tilde{t}) = (s_1(\tilde{t}). . . s_R(\tilde{t}))^T$ contains the R independent source signals at the discrete time index $\tilde{t}$, $x(\tilde{t}) = (x_1(\tilde{t}). . . x_p(\tilde{t}))^T$ contains the P recorded microphone signals and **A** is the so-called mixing matrix, whose dimension is $P \times R$. In general, both **A** and are unknown, and need to be estimated, given $x(\tilde{t})$.

Taking into account the number of microphones and sources, three scenarios can be distinguished:

- *Over-determined case*: The number of microphones is larger than the number of sources (R< P).
- *Determined case:* The number of microphones is equal to the number of sources (R = P).
- *Under*-determined case: The number of sources is larger than the number of microphones (R > P). In many situations, the number of sound sources is not known and may be greater than the number of mixtures. The separation of under-determined mixtures requires additional assumptions on the source signals than in the over-determined case, in order to reduce the possible solutions in the solutions space.

Depending on the level of surface reflections in an acoustic environment, three types of mixtures can be distinguished:

- *Instantaneous mixtures*: Each microphone records a signal which is a linear combination of the source signals. In other words, A is a scalar matrix.
- *Anechoic mixtures*: Each microphone picks up the mixtures of the direct sound from the sources. A is a scalar matrix and $s(\tilde{t}) \rightarrow s(\tilde{t} - \delta)$, where δ is the time taken for the source to reach the microphone.
- *Convolutive mixtures*: Room reverberation affects the audio collected by microphones, leading to superposition of direct sound sources and time-delayed source component due to room reflections. In this case, A is a matrix of filters. Room reflections also depend on the experimental methodology employed. Other factors to take into account are measurement noise (electrical and acoustical), room geometry, acoustic treatment, source directivity, air flow, temperature, humidity, homogeneity and even the presence of people and other scatterers.

Depending on whether the measurement process is linear, we have:

- *Linear case:* The mixture is a linear combination of the sources as represented by model
- *Non-linear case*: The relation between $\mathbf{x}$ and $\mathbf{s}$ is characterized by a non-linear function, such as $\boldsymbol{x} = \exp(\boldsymbol{s})$, where $\exp(\cdot)$ is an exponential function.

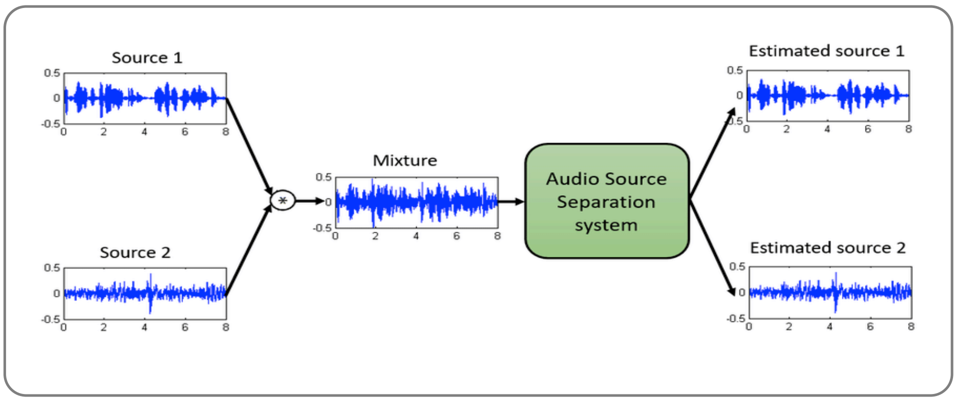


Figure I.1: Audio Source Separation

I.4. Conventional Methods of Audio Source Separation

Several techniques have been successfully applied to the audio source separation problem. For example, binary and soft time-frequency masking, have been applied to the sound mixture in the time-frequency domain. Independent component analysis [5, 6] is based on the statistical independence between the sound sources and works for over-determined or determined mixtures, while Computational Auditory Scene Analysis [7] tackles the problem from a human perception point of view. Non-negative Matrix Factorization (NMF) [8] factorizes the mixture spectrogram into a dictionary of source components weighted by activation coefficients and works well in the case of unsupervised monaural source separation. Dictionary learning based sparse representation technique has been used for under-determined sound separation.

The following sub-sections will briefly present several main approaches that have been developed for audio source separation.

I.4.1. Independent Component Analysis (ICA)

In Independent component analysis [5, 6], each mixture **x** and source **s** in (I.1), are assumed to be a random variable, with zero mean (otherwise they can always be centred by subtracting the sample mean). Moreover, the sources s are assumed to be statistically independent, which means that the following relation on the probabilities holds for the two variables y_i, y_j:

$$p(y_i, y_j) = (y_i)(y_j) \tag{I.2}$$

Where p indicates the joint density function. When the two variables are Gaussian distributed, $p(y_i, y_j)$ would be itself Gaussian, therefore it would not contain any information on the directions of the columns of the mixing matrix **A**: this implies that the ICA model cannot be used for separating multiple Gaussian sources.

Two main problems arise when using the ICA techniques: considering that both **A** and ***s*** are unknown, it is impossible to estimate the variances of the independent components and their order. While on one hand ICA guarantees unique solutions subject to scale and permutation ambiguities, the weak point of ICA-based techniques is that, in order to be effective, it is necessary to first estimate the number of unknown sources from the mixture before performing source separation. Moreover, ICA is not suitable for sources which are mutually dependent.

The problem of separating mixed audio source signals using ICA has been widely investigated. Early papers [9, 10] attempt to recover multiple unknown source signals from multiple observed signals that are mixtures of the sources. However, the method in [10] is successful in those cases where [9] fails, e.g. for weak signals in a high level of noise. In more recent papers, such as [11], ICA is applied to convolutive mixtures of two audio sources, picked-up by two-microphones. The sources are extracted one by one in a decreasing order of negentropy from the mixed signal.

A different approach consists in using ICA techniques for the estimation of Ideal Binary Masks (IBMs) in [12, 13]. More specifically, in [12], a method for instantaneous mixing model is proposed, which assumes closely spaced microphones. IBMs estimated from the outputs of an ICA algorithm are used to extract an arbitrary number of audio source signals. In [13], ICA is used to estimate the IBMs for separating the source signals from two-microphone recordings of convolutive audio source mixtures, but includes an additional step, which introduces cepstral smoothing, in order to reduce musical noise caused by the T-F masking. Other works [14] aim for a lower computational complexity and a faster convergence compared to standard ICA methods [6], where a target audio is extracted and recognized from noisy stereo mixtures.

I.4.2. Fast Independent Component Analysis (FastICA)

An efficient and popular algorithm for ICA, named FastICA [6], is employed in several works [15–18], offering fast convergence, guaranteed global convergence for certain mixing conditions and contrasts, and robustness in presence of noise. Convolutive mixtures are separated in [15] by using the FastICA algorithm. This algorithm combines multi-channel whitening with fixed-point iterations, which allows the sources to be reconstructed as they appear in the observed mixtures. In [16], an enhanced FastICA is employed, showing less artifacts compared to the ICA method in [11]. A different approach is followed in [17], where ICA is used for identifying the active components of Hidden Markov Models (HMMs).

The FastICA algorithm in [10] is here used to build independent voice space for talker and environment adaptation. The work in [18] is based on the maximisation of non-Gaussianity technique using Gradient ICA algorithm and FastICA algorithm, and then compares the results of both methods. While FastICA needs less execution time compared to Gradient ICA, the latter provides a higher efficiency in separating audio signals. FastICA is also combined with sparse component analysis, and applied to over and under-determined mixtures.

I.4.3. Computational Auditory Scene Analysis (CASA)

In Computational Auditory Scene Analysis [7] the task is to separate mixtures of sound sources like human listeners do. In the same way in which an image is analyzed and processed as a whole by sensing the single features, such as edges, textures and colors, the sound reaching the human ear is subject to Auditory Scene Analysis (ASA).
A standard CASA system consists of four stages [7]:

1) *Peripheral analysis*: the input signal is processed using an auditory model, resulting in a cochleagram (a T-F representation).
2) *Auditory features extraction:* some features are generated.
3) *Segmentation*: the system generates a collection of segments or contiguous regions in a cochleagram.
4) *Grouping*: those segments which are likely to come from the same source are grouped into a perceptual structure, called stream, corresponding to how the source is mentally perceived by the listener.

CASA attempts to construct a machine that approaches human performance in ASA by using one or two microphones recordings of the acoustic scene, in order to extract individual source streams. A typical example in CASA aims to estimate an ideal time-frequency mask, built e.g. by using a model of peripheral auditory system called cochleagram, which emulates the human frequency selectivity. CASA is one of the first attempts to imitate the human auditory system for the purpose of creating an audio source separation system.

I.5. Categories of Audio Source Separation

Audio source separation algorithms can be roughly divided into two categories, time-frequency T-F methods and T-D (or end-to-end) methods. T-F audio source separation algorithms aim to estimate the enhanced spectrum of each individual source from the mixture spectra [1], and then rebuild the wave- form via the iSTFT, by combining the enhanced magnitude spectrum with either noisy or the modified phase of the mixture. On the downside, the erroneous estimation of the phase limits the upper bound of separation performance. To solve the phase problem, end-to-end audio source separation methods such as the fully-convolutional time-domain audio separation network (Conv-TasNet) [2] and Wave-u-net [3] are proposed.

I.5.1. Time-frequency methods

Having a conversation in a complex acoustic environment, with multiple noise sources and competing background speakers, is a task humans are remarkably good at. The problem that humans solve when they focus their auditory attention towards one audio signal in a complex mixture of signals is commonly known as the cocktail party problem.

Since the cocktail party problem was initially formalized, a large number of potential solutions have been proposed, and the most popular techniques originate from the field of Computational Auditory Scene Analysis. In CASA, different segmentation and grouping rules are used to group T-F units that are believed to belong to the same speaker the grouped T-F units are then used to extract a particular speaker from the mixture signal. Another popular technique for multi-talker audio source separation is NMF.

The NMF technique uses non-negative dictionaries to decompose the spectrogram of the mixture signal into speaker specific activations, and from these activations an isolated target signal can be approximated using the dictionaries. For multi-talker audio source separation, both CASA and NMF have led to limited success and the most successful techniques, before the deep learning era, are based on probabilistic models, such as factorial Gaussian mixture models-Hidden Markov models (GMM-HMM) , that model the temporal dynamics and the complex interactions of the target and competing audio source signals. Unfortunately, these models assume and only work under closed-set speaker conditions, i.e. the identity of the speakers must be known *a priori*.

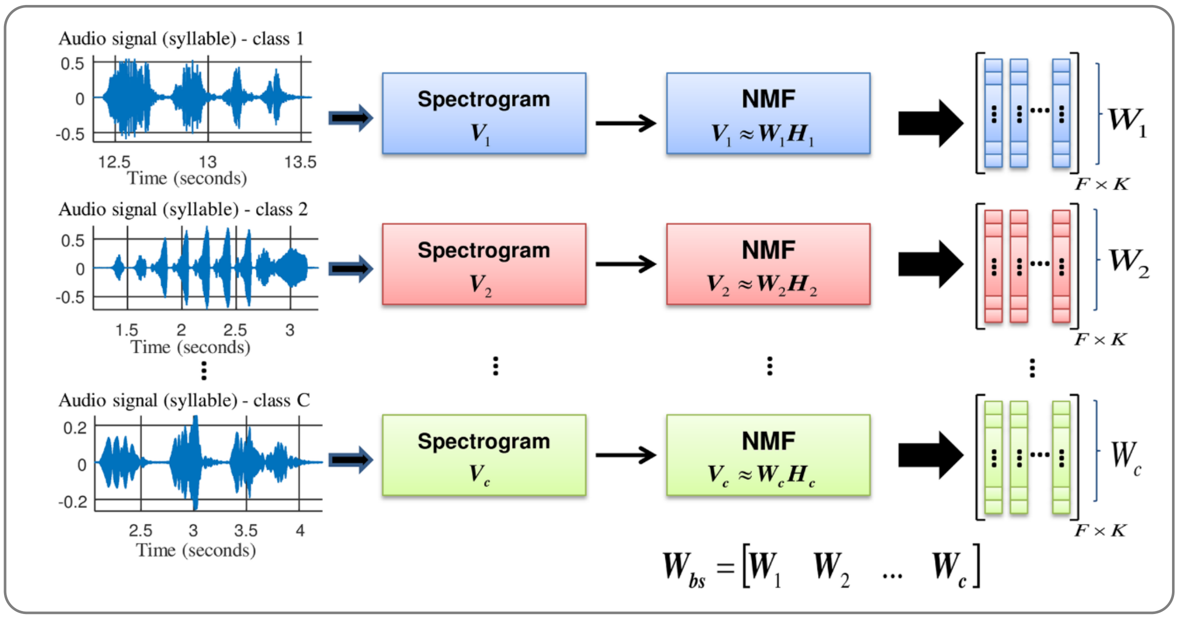

Audio signal (syllable) - class 1
Time (seconds)
Audio signal (syllable) - class 2
Time (seconds)
Audio signal (syllable) - class C
Time (seconds)
Spectrogram V_1
Spectrogram V_2
Spectrogram V_c
NMF $V_1 \approx W_1 H_1$
NMF $V_2 \approx W_2 H_2$
NMF $V_c \approx W_c H_c$
W_1
W_2
W_c
$F \times K$
$W_{bs} = [W_1 \ \ W_2 \ \ \dots \ \ W_c]$

Figure I.2: Bird sound spectrogram decomposition through NMF

There are other methods proposed from some scientists:

In Weng *et al.* [18], which proposed the best performing system in the 2006 monaural audio source separation and recognition challenge , the instantaneous energy was used to determine the training label assignment, which alleviated the label permutation problem and allowed separation of unknown speakers. Although this approach works well for two-speaker mixtures, it is hard to scale up to mixtures of three or more speakers.

Hershey *et al.* [20] have made significant progress with their Deep Clustering (DPCL) technique. In their work, a deep Recurrent Neural Network (RNN) is used to project the audio source mixture into an embedding space, where T-F units belonging to the same speaker form a cluster. In this embedding space a clustering algorithm (e.g. K means) is used to identify the clusters. Finally, T-F units belonging to the same clusters are grouped together and a binary mask is constructed and used to separate the speakers from the mixture signal. To further improve the model, another RNN is stacked on top of the first DPCL RNN to estimate continuous masks for each target speaker. Although DPCL show good performance, the technique is potentially limited because the objective function is based on the affinity between the sources in the embedding space, instead of the separated signals themselves. That is, low proximity in the embedding space does not necessarily imply perfect separation of the sources in the signal space.

I.5.2. Time-domain methods

A. Wave-u-net

The Wave-U-Net is a neural network applicable to audio source separation tasks, which directly separates a time-domain signal into source signals and works directly on the raw audio waveform. The Wave-U-Net is an adaptation of the U-Net architecture to the one-dimensional time domain to perform end-to-end audio source separation and has an encoder-decoder architecture.

As shown in Figure (**I.3**), Wave-U-Net architecture computes an increasing number of higher-level features on coarser time scales using down-sampling blocks. These features are combined with the earlier computed local, high-resolution features using up-sampling blocks, yielding multi-scale features which are used for making predictions. The network has L levels in total, with each successive level operating at half the time resolution as the previous one. For K sources to be estimated, the model returns predictions in the interval$(-1;\ 1)$, one for each source audio sample [3].

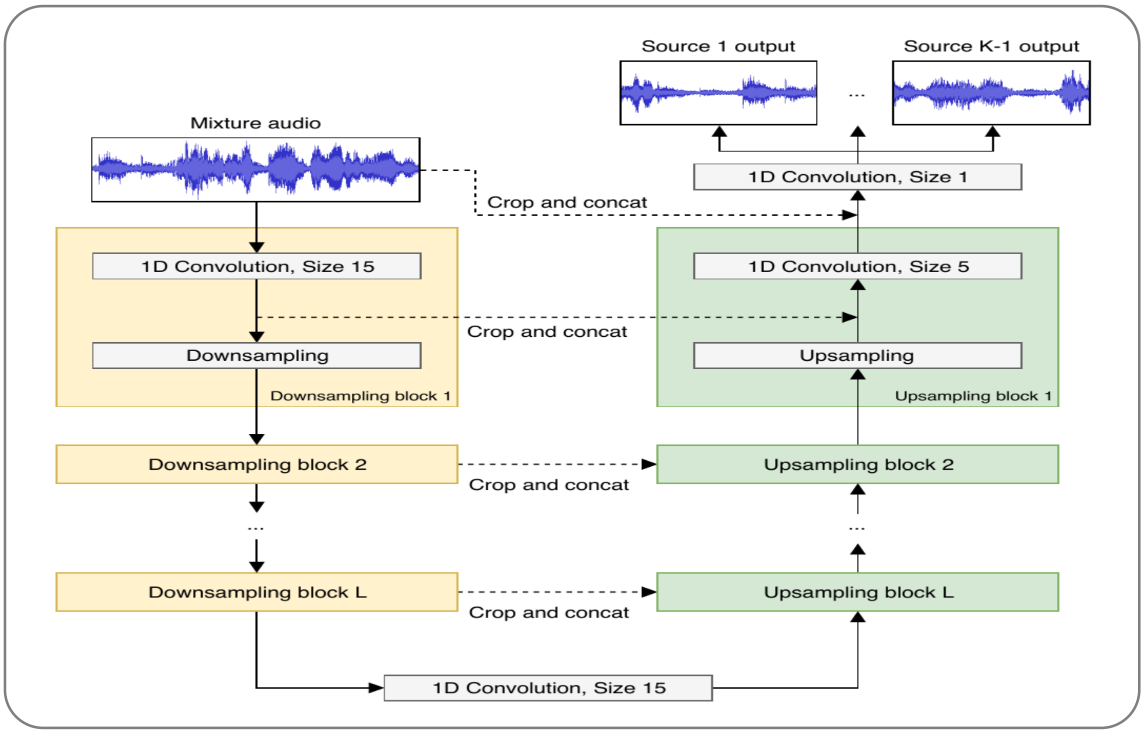


Figure I.3: Wave-u-Net architecture

B. TASNET

Time-domain Audio Separation Network (TasNet) is a neural network that directly models the mixture waveform using an encoder-decoder framework, and performs the separation on the output of the encoder. In this framework, the mixture waveform is represented by a nonnegative weighted sum of basis signals, where the weights are the outputs of the encoder, and the basis signals are the filters of the decoder. The separation is done by estimating the weights that correspond to each source from the mixture weight. Because the weights are nonnegative, the estimation of source weights can be formulated as finding the masks which indicate the contribution of each source to the mixture weight, similar to the T-F masks that are used in Short-Time Fourier Transform (STFT) systems. The source waveforms are then reconstructed using the learned decoder. Since TasNet operates on waveform segments that can be small, the system can be implemented in real-time with very low latency. Also TasNet outperforms the state-of-art STFT-based system in applications that do not require real-time processing.

The structure of the network contains three parts: an encoder for estimating the mixture weight, a separation module, and a decoder for source waveform reconstruction. The combination of the encoder and the decoder modules construct a nonnegative auto encoder for the waveform of the mixture, where the nonnegative weights are calculated by the encoder and the basis signals are the 1-D filters in the decoder. The separation is performed on the mixture weight matrix using a subnetwork that estimates a mask for each source. As shown in Figure **(I.4)** [21].

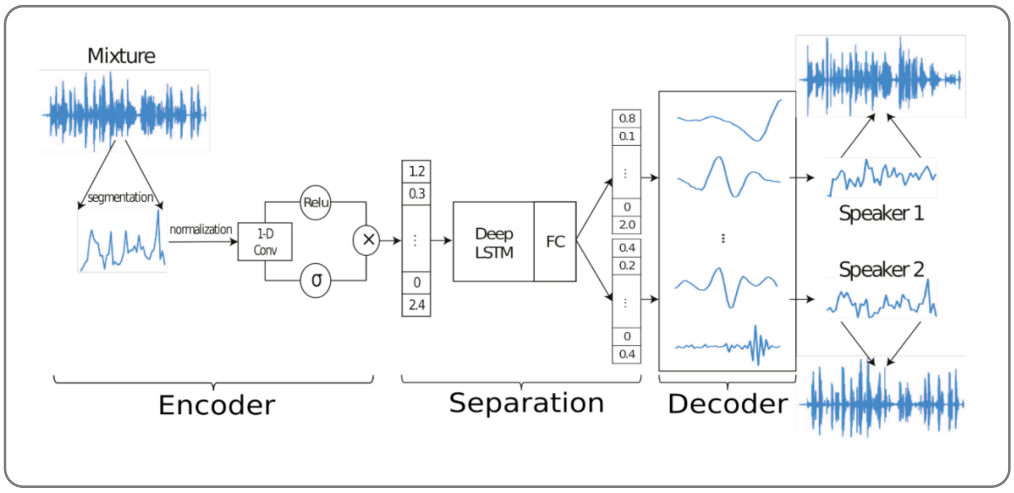


Figure I.4: Time-domain Audio Separation Network

I.6. Conclusion

Powerful audio source processing in a real sound environment usually requires automatic audio source separation. Due to the importance of this research topic to language processing technology, many methods to solve this problem have been proposed as we saw in the first chapter. However, with this remarkable progress in audio source separation methods, they are still insufficient, because we haven't seen a method for separating some other audio sources. Therefore, until today there is no separation method for bird sounds.

In next chapter, we will present the deep learning with her different techniques. In addition, we will study the different types of DL networks, Especially CNN.

CHAPTER II

Convolutional Neural Network

II.1. Introduction

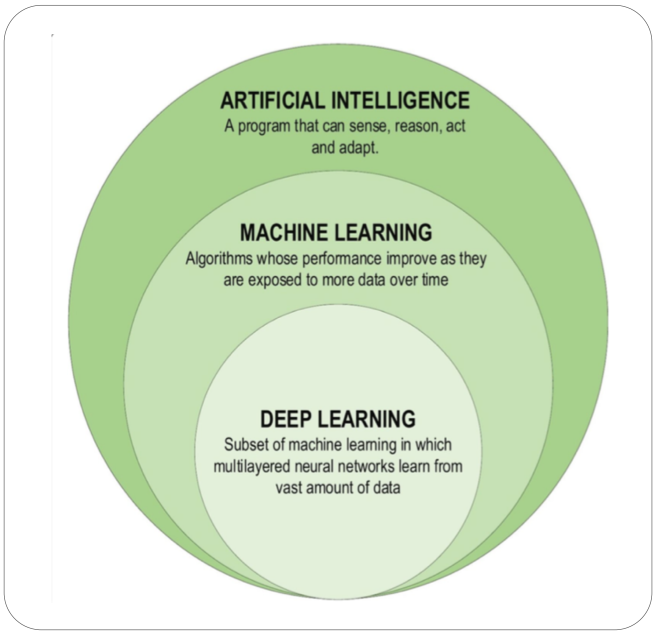
In this chapter, we present machine learning in general, deep learning, 2D convolutional neural networks CNN, and finally 1D convolutional neural network.

II.2. Machine and Deep learning

Recently, machine learning (ML) has become very widespread in research and has been incorporated in a variety of applications, including text mining, spam detection, video recommendation, image and audio classification, and multimedia concept retrieval [22]. Among the different ML algorithms, deep learning (DL) is very commonly employed in these applications. Another name for DL is representation learning (RL). The continuing appearance of novel studies in the ends of deep and distributed learning is due to both the unpredictable growth in the ability to obtain data and the amazing progress made in the hardware technologies, e.g. High Performance Computing (HPC) [23].

DL is derived from the ML but considerably outperforms its predecessors. Moreover, DL employs transformations and graph technologies simultaneously in order to build up multi-layer learning models. The most recently developed DL techniques have obtained good outstanding performance across a variety of applications, including audio and audio source processing, visual data processing, natural language processing (NLP), among others [24].

In the field of ML, DL, due to its considerable success, is currently one of the most prominent research trends. In this section, an overview of DL is presented that adopts various perspectives such as the main concepts, architectures, challenges, applications, computational tools and evolution matrix. Convolutional neural network (CNN) is one of the most popular and used of DL networks. Because of CNN, DL is very popular nowadays. The main advantage of CNN compared to its predecessors is that it automatically detects the significant features without any human supervision which made it the most used. Therefore, we have dug in deep with CNN by presenting the main components of it.


ARTIFICIAL INTELLIGENCE
A program that can sense, reason, act and adapt.
MACHINE LEARNING
Algorithms whose performance improve as they are exposed to more data over time
DEEP LEARNING
Subset of machine learning in which multilayered neural networks learn from vast amount of data

. **Figure II.1.** Deep learning family

II.2.1. When to apply deep learning?

Machine intelligence is useful in many situations which is equal or better than human experts in some cases, meaning that DL can be a solution to the following problems:

- Cases where human experts are not available.
- Cases where humans are unable to explain decisions made using their expertise (language understanding, medical decisions, and audio recognition).
- Cases where the problem solution updates over time (price prediction, stock preference, weather prediction, and tracking).
- Cases where solutions require adaptation based on specific cases (personalization, biometrics).
- Cases where size of the problem is extremely large and exceeds our inadequate reasoning abilities (sentiment analysis, matching ads to Facebook, calculation webpage ranks).

II.2.2. Why deep learning?

Several performance features may answer this question, e.g.

- *Universal Learning Approach:* Because DL has the ability to perform in approximately all application domains, it is sometimes referred to as universal learning.

- *Robustness:* In general, precisely designed features are not required in DL techniques. Instead, the optimized features are learned in an automated fashion related to the task under consideration. Thus, robustness to the usual changes of the input data is attained.
- *Generalization:* Different data types or different applications can use the same DL technique, an approach frequently referred to as transfer learning (TL) which explained in the latter section. Furthermore, it is a useful approach in problems where data is insufficient.
- *Scalability:* DL is highly scalable. ResNet [25], which was invented by Microsoft, comprises 1202 layers and is frequently applied at a supercomputing scale. Lawrence Livermore National Laboratory (LLNL), a large enterprise working on evolving frameworks for networks, adopted a similar approach, where thousands of nodes can be implemented.

II.2.3. Classification of DL approaches

DL techniques are classified into three major categories: unsupervised, partially supervised (semi-supervised) and supervised. Furthermore, deep reinforcement learning (DRL), also known as RL, is another type of learning technique, which is mostly considered to fall into the category of partially supervised (and occasionally unsupervised) learning techniques.

A. Deep supervised learning

This technique deals with labeled data. When considering such a technique, the environs have a collection of inputs and resultant outputs $(x_t, y_t) \sim \rho$. For instance, the smart agent guesses $\hat{y}_t = f(x_t)$ if the input is x_t and will obtain $\iota(\hat{y}_t, y_t)$ as a loss value. Next, the network parameters are repeatedly updated by the agent to obtain an improved estimate for the preferred outputs. Following a positive training outcome, the agent acquires the ability to obtain the right solutions to the queries from the environs.

For DL, there are several supervised learning techniques, such as recurrent neural networks (RNNs), convolutional neural networks (CNNs), and deep neural networks (DNNs). In addition, the RNN category includes gated recurrent units (GRUs) and long short-term memory (LSTM) approaches.

The main advantage of this technique is the ability to collect data or generate a data output from the prior knowledge. However, the disadvantage of this technique is that decision boundary might be overstrained when training set doesn't own samples that should be in a class. Overall, this technique is simpler than other techniques in the way of learning with high performance.

B. Deep semi-supervised learning

In this technique, the learning process is based on semi-labeled datasets. Occasionally, generative adversarial networks (GANs) and DRL are employed in the same way as this technique. In addition, RNNs, which include GRUs and LSTMs, are also employed for partially supervised learning. One of the advantages of this technique is to minimize the amount of labeled data needed. On other the hand, one of the disadvantages of this technique is irrelevant input feature present training data could furnish incorrect decisions. Text document classifier is one of the most popular examples of an application of semi-supervised learning. Due to difficulty of obtaining a large amount of labeled text documents, semi-supervised learning is ideal for text document classification task.

C. Deep unsupervised learning

This technique makes it possible to implement the learning process in the absence of available labeled data (i.e. no labels are required). Here, the agent learns the significant features or interior representation required to discover the unidentified structure or relationships in the input data. The main disadvantages of unsupervised learning are unable to provide accurate information concerning data sorting and computationally complex. One of the most popular unsupervised learning approaches is clustering [26].

II.2.4. Types of DL networks

The most famous types of deep learning networks are discussed in this section: these include recursive neural networks (RvNNs), RNNs, and CNNs. RvNNs and RNNs were briefly explained in this section while CNNs were explained in deep due to the importance of this type in our work. Furthermore, it is the most used in several applications among other networks.

A. Recursive neural networks

RvNN can achieve predictions in a hierarchical structure also classify the outputs utilizing compositional vectors [27]. Recursive auto-associative memory (RAAM) is the primary inspiration for the RvNN development. The RvNN architecture is generated for processing objects, which have randomly shaped structures like graphs or trees. This approach generates a fixed-width distributed representation from a variable-size recursive-data structure.

B. Recurrent neural networks

RNNs are a commonly employed and familiar algorithm in the discipline of DL. RNN is mainly applied in the area of audio processing and NLP contexts [28].

Unlike conventional networks, RNN uses sequential data in the network. Since the embedded structure in the sequence of the data delivers valuable information, this feature is fundamental to a range of different applications. For instance, it is important to understand the context of the sentence in order to determine the meaning of a specific word in it. Thus, it is possible to consider the RNN as a unit of short-term memory, a typical unfolded RNN diagram is illustrated in Figure II.2.

Pascanu et al. [29] introduced three different types of deep RNN techniques, namely “Hidden-to-Hidden”, “Hidden-to-Output”, and “Input-to-Hidden”. A deep RNN is introduced that lessens the learning difficulty in the deep network and brings the benefits of a deeper RNN based on these three techniques.

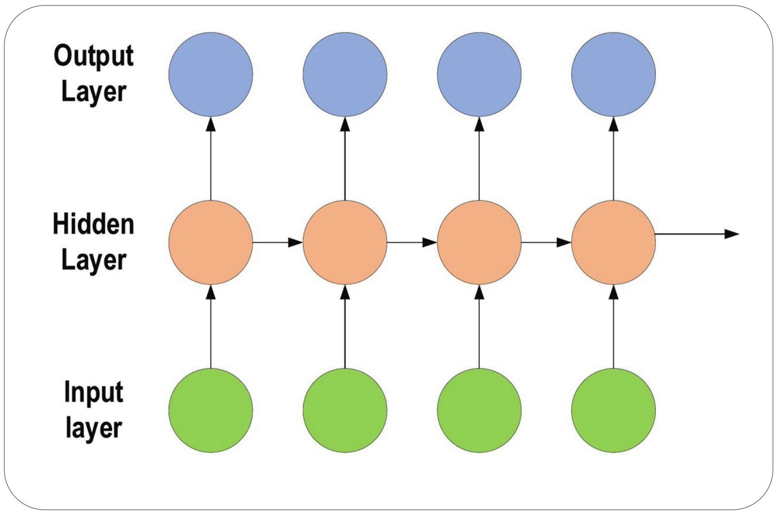


Figure II.2. Typical unfolded RNN diagram

However, RNN’s sensitivity to the exploding gradient and vanishing problems represent one of the main issues with this approach. More specifically, during the training process, the reduplications of several large or small derivatives may cause the gradients to exponentially explode or decay. With the entrance of new inputs, the network stops thinking about the initial ones; therefore, this sensitivity decays over time. Furthermore, this issue can be handled using LSTM [30].

C. Long short term memory networks

Long-Short Time Memory networks are special kind of RNN, networks with loops in them, allowing information to persist. LSTMs are capable of learning long-term dependencies between input samples. Furthermore, they are great remembering information for long time periods.

LSTMs have a chain like structure with a repeating module. In one of the most common implementations, each module is composed by a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell [30].

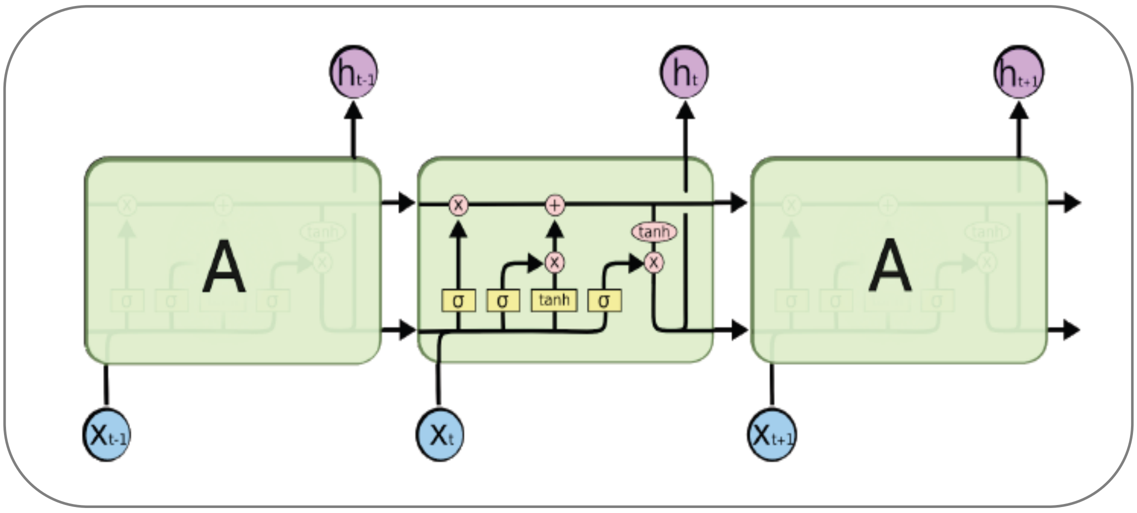


Figure II.3. LSTM implementation

CNN is considered to be more powerful than RNN and LSTM. RNN includes less feature compatibility when compared to CNN.

II.3. Convolutional neural networks

In the field of DL, the CNN is the most famous and commonly employed algorithm [31]. The main benefit of CNN compared to its predecessors is that it automatically identifies the relevant features without any human supervision. CNNs have been extensively applied in a range of different fields, including computer vision, speech processing, Face Recognition, etc. The structure of CNNs was inspired by neurons in human and animal brains, similar to a conventional neural network. More specifically, in a cat's brain, a complex sequence of cells forms the visual cortex; this sequence is simulated by the CNN [32].

Goodfellow et al. [33] identified three key benefits of the CNN: equivalent representations, sparse interactions, and parameter sharing. Unlike conventional fully connected (FC) networks, shared weights and local connections in the CNN are employed to make full use of 2D input-data structures like image signals or mixture of 1D input-data.

This operation utilizes an extremely small number of parameters, which both simplifies the training process and speeds up the network. This is the same as in the visual cortex cells. Notably, only small regions of a scene are sensed by these cells rather than the whole scene (i.e., these cells spatially extract the local correlation available in the input, like local filters over the input).

A commonly used type of CNN, which is similar to the multi-layer perceptron (MLP), consists of numerous convolution layers preceding sub-sampling (pooling) layers, while the ending layers are FC layers. An example of CNN architecture for image classification is illustrated in figure II.4.

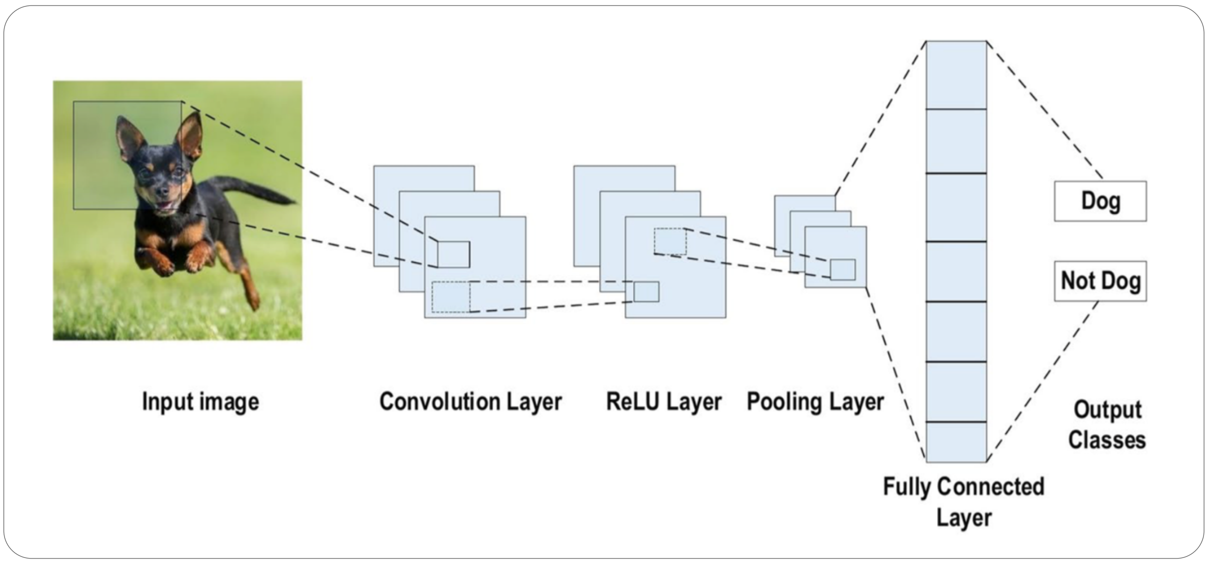


Figure II.4. An example of CNN architecture for image classification

II.3.1. Benefits of employing CNNs

The benefits of using CNNs over other traditional neural networks in the computer vision environment are listed as follows:

1) The main reason to consider CNN is the weight sharing feature, which reduces the number of trainable network parameters and in turn helps the network to enhance generalization and to avoid overfitting.
2) Concurrently learning the feature extraction layers and the classification layer causes the model output to be both highly organized and highly reliant on the extracted features.
3) Large-scale network implementation is much easier with CNN than with other neural networks.

II.3.2. CNN layers

The CNN architecture consists of a number of layers (or so-called multi-building blocks). Each layer in the CNN architecture, including its function, is described in detail below.

1. **Convolutional Layer:** In CNN architecture, the most significant component is the convolutional layer. It consists of a collection of convolutional filters (so-called kernels). In our case, the input mixture of bird sounds, expressed as N-dimensional metrics, is convolved with these filters to generate the output feature map.
   - *Kernel definition:* A grid of discrete numbers or values describes the kernel. Each value is called the kernel weight. Random numbers are assigned to act as the weights of the kernel at the beginning of the CNN training process. In addition, there are several different methods used to initialize the weights. Next, these weights are adjusted at each training era; thus, the kernel learns to extract significant features.
   - *Convolutional Operation:* Initially, the CNN input format is described. The vector format is the input of the traditional neural network, while the image is the input of the CNN. To understand the convolutional operation, let us take an example of a 4 × 4 gray-scale image with a 2 × 2 random weight-initialized kernel. First, the kernel slides over the whole image horizontally and vertically. In addition, the dot product between the input image and the kernel is determined, where their corresponding values are multiplied and then summed up to create a single scalar value, calculated concurrently. The whole process is then repeated until no further sliding is possible. Note that the calculated dot product values represent the feature map of the output. Figure 8 graphically illustrates the primary calculations executed at each step. In this figure, the light green color represents the 2 × 2 kernel, while the light blue color represents the similar size area of the input image. Both are multiplied; the end result after summing up the resulting product values (marked in a light orange color) represents an entry value to the output feature map.
   - However, padding to the input image is not applied in the previous example, while a stride of one (denoted for the selected step-size over all vertical or horizontal locations) is applied to the kernel. Note that it is also possible to use another stride value. In addition, a feature map of lower dimensions is obtained as a result of increasing the stride value.
   - On the other hand, padding is highly significant to determining border size

information related to the input mixture sounds. By contrast, the border side-features moves carried away very fast. By applying padding, the size of the input will increase, and in turn, the size of the output feature map will also increase. Core Benefits of Convolutional Layers.

- *Sparse Connectivity:* Each neuron of a layer in FC neural networks links with all neurons in the following layer. By contrast, in CNNs, only a few weights are available between two adjacent layers. Thus, the number of required weights or connections is small, while the memory required to store these weights is also small; hence, this approach is memory-effective. In addition, matrix operation is computationally costlier than the dot operation ⊙ in CNN.

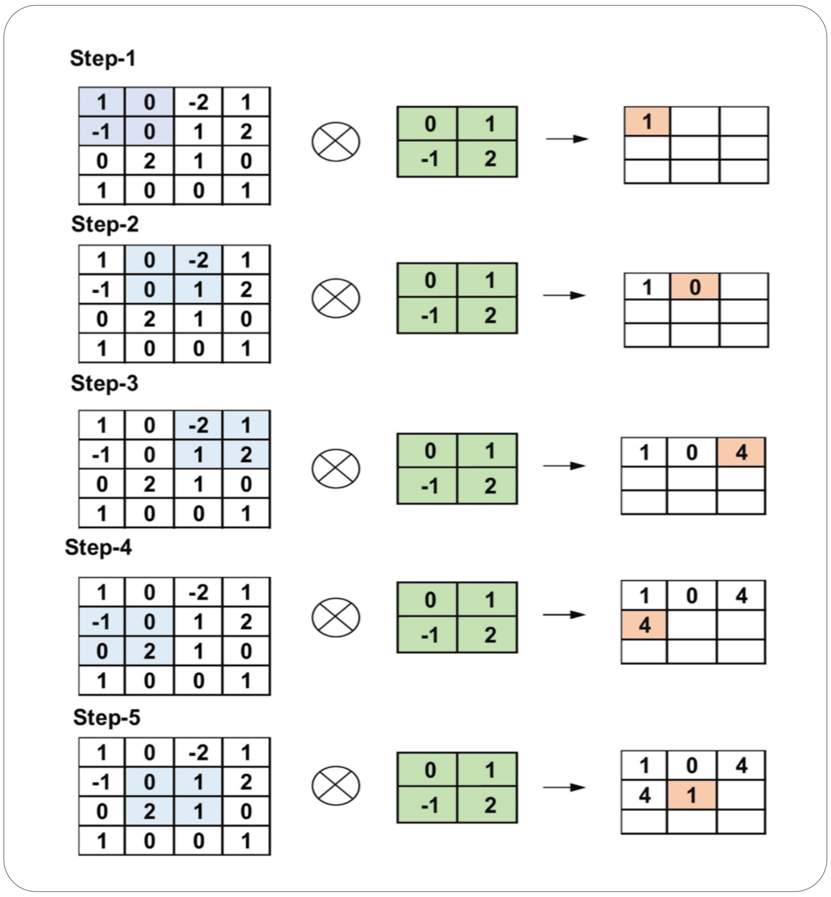


Figure II.5. Primary calculations executed at each step of convolutional layer

- *Weight Sharing:* There are no allocated weights between any two neurons of neighboring layers in CNN, as the whole weights operate with one and all pixels of the input matrix. Learning a single group of weights for the whole input will significantly decrease the required training time and various costs, as it is not necessary to learn additional weights for each neuron.

2. **Pooling Layer:** The main task of the pooling layer is the sub-sampling of the feature maps. These maps are generated by following the convolutional operations. In other words, this approach shrinks large-size feature maps to create smaller feature maps. Concurrently, it maintains the majority of the dominant information (or features) in every step of the pooling stage. In a similar manner to the convolutional operation, both the stride and the kernel are initially size-assigned before the pooling operation is executed. Several types of pooling methods are available for utilization in various pooling layers. These methods include tree pooling, gated pooling, average pooling, min pooling, max pooling, global average pooling (GAP), and global max pooling. The most familiar and frequently utilized pooling methods are the max, min, and GAP pooling. Sometimes, the overall CNN performance is decreased as a result; this represents the main shortfall of the pooling layer, as this layer helps the CNN to determine whether or not a certain feature is available in the particular input, but focuses exclusively on ascertaining the correct location of that feature. Thus, the CNN model misses the relevant information

3. **Activation Function (non-linearity):** Mapping the input to the output is the core function of all types of activation function in all types of neural network. The input value is determined by computing the weighted summation of the neuron input along with its bias (if present). This means that the activation function makes the decision as to whether or not to fire a neuron with reference to a particular input by creating the corresponding output. Non-linear activation layers are employed after all layers with weights (so-called learnable layers, such as FC layers and convolutional layers) in CNN architecture. This non-linear performance of the activation layers means that the mapping of input to output will be non-linear; moreover, these layers give the CNN the ability to learn extra-complicated things. The activation function must also have the ability to differentiate, which is an extremely significant feature, as it allows error back-propagation to be used to train the network. The following types of activation functions are most commonly used in CNN and other deep neural networks.

- **Sigmoid:** The input of this activation function is real numbers, while the output is restricted to between zero and one. The sigmoid function curve is S-shaped and can be represented mathematically by Eq. (II.1).

$$f\ (x)_{sigm}\ = \frac{1}{1+e^{-x}} \tag{II.1}$$

- **Tanh:** It is similar to the sigmoid function, as its input is real numbers, but the output is restricted to between −1 and 1. Its mathematical representation is in Eq. (II.2).

$$f\ (x)_{tanh} = \frac{e^{x} - e^{-x}}{e^{x} +\ e^{-x}} \tag{II.2}$$

- **ReLU:** The mostly commonly used function in the CNN context. It converts the whole values of the input to positive numbers. Lower computational load is the main benefit of ReLU over the others. Its mathematical representation is in Eq. (II.3).

$$f\ (x)_{ReLU} =\ max(0, x) \tag{II.3}$$

Occasionally, a few significant issues may occur during the use of ReLU. For instance, consider an error back-propagation algorithm with a larger gradient flowing through it. Passing this gradient within the ReLU function will update the weights in a way that makes the neuron certainly not activated once more. This issue is referred to as "Dying ReLU". Some ReLU alternatives exist to solve such issues. The following discusses some of them.

- **Leaky ReLU:** Instead of ReLU down-scaling the negative inputs, this activation function ensures these inputs are never ignored. It is employed to solve the Dying ReLU problem. Leaky ReLU can be represented mathematically as in Eq. (II.4).

$$f(x)_{LekyReLU} = \begin{cases} x, if\ x\ > 0 \\ mx,\ \ x \leq 0 \end{cases} \tag{II.4}$$

Note that the leak factor is denoted by m. It is commonly set to a very small value, such as 0.001.

- **Noisy ReLU:** This function employs a Gaussian distribution to make ReLU noisy. It can be represented mathematically as in Eq. (II.5).

$$f(x)_{NoisyReLU} = max(x + Y), with\, Y \sim N(0, \sigma(x)) \qquad \text{(II.5)}$$

- **Parametric Linear Units:** This is mostly the same as Leaky ReLU. The main difference is that the leak factor in this function is updated through the model training process. The parametric linear unit can be represented mathematically as in Eq. (II.6).

$$f\,(x)_{\text{ParametricLinear}} = \begin{cases} x\,, if\; x > 0 \\ ax\,, \quad if\; x \leq 0 \end{cases} \qquad \text{(II.6)}$$

Note that the learnable weight is denoted as a

4. **Fully Connected Layer:** Commonly, this layer is located at the end of each CNN architecture. Inside this layer, each neuron is connected to all neurons of the previous layer, the so-called FC approach. It is utilized as the CNN classifier. It follows the basic method of the conventional multiple-layer perceptron neural network, as it is a type of feed-forward ANN. The input of the FC layer comes from the last pooling or convolutional layer. This input is in the form of a vector, which is created from the feature maps after flattening.

5. **Loss Functions:** The previous section has presented various layer-types of CNN architecture. In addition, the final classification is achieved from the output layer, which represents the last layer of the CNN architecture. Some loss functions are utilized in the output layer to calculate the predicted error created across the training samples in the CNN model. This error reveals the difference between the actual output and the predicted one. Next, it will be optimized through the CNN learning process.
However, two parameters are used by the loss function to calculate the error. The CNN estimated output (referred to as the prediction) is the first parameter. The actual output (referred to as the label) is the second parameter.

II.4. 1D Convolutional neural networks

The conventional deep CNNs presented in the previous section are designed to operate exclusively on 2D data such as images and videos. This is why they are often referred to as, ‘‘2D CNNs”. As an alternative, a modified version of 2D CNNs called 1D Convolutional Neural Networks (1D CNNs) have recently been developed [74]. These studies have shown that for certain applications 1D CNNs are advantageous and thus preferable to their 2D counterparts in dealing with 1D signals due to the following reasons:

- There is a significant difference in terms of computational complexities of 1D and 2D convolutions, i.e., an image with NxN dimensions convolve with KxK kernel will have a computational complexity $\sim O(N^2K^2)$ while in the corresponding 1D convolution (with the same dimensions, N and K) this is $\sim O(NK)$.. This means that under equivalent conditions (same configuration, network and hyper parameters) the computational complexity of a 1D CNN is significantly lower than the 2D CNN.
- As a general observation especially over the recent studies most of the 1D CNN applications have used compact (with 1–2 hidden CNN layers) configurations with networks having<10 K parameters whereas almost all 2D CNN applications have used ‘‘deep” architectures with more than 1 M (usually above 10 M) parameters. Obviously, networks with shallow architectures are much easier to train and implement.
- Usually, training deep 2D CNNs requires special hardware setup (e.g. Cloud computing or GPU farms). On the other hand, any CPU implementation over a standard computer is feasible and relatively fast for training compact 1D CNNs with few hidden layers (e.g. 2 or less) and neurons (e.g. < 50).
- Due to their low computational requirements, compact 1D CNNs are well-suited for real-time and low-cost applications especially on mobile or hand-held devices [34].

II.5. Conclusion

In this chapter, we presented some generalities about machine learning, then we introduced the notion of deep learning (DL). We investigated classification of DL approaches and different types of DL networks. We then looked in detail at the 2D convolutional neural network and finally the 1D CNN network that we will need for the rest of our work.

CHAPTER III

Conv-TasNet for bird sound separation

III.1. Introduction

Fully-connected DNNs are an early and commonly used type of neural networks applied to audio source separation. However, many works show that these networks may be limited, especially when the recordings contain some levels of reverberation.

Conv-TasNet is a recently proposed fully-convolutional time-domain audio separation network based deep neural network that achieves state-of-the-art performance in audio source source separation. Its architecture consists of a learnable encoder/decoder and a separator that operates on top of this learned space. Various improvements have been proposed to Conv-TasNet. However, they mostly focus on the separator, leaving its encoder/decoder as a (shallow) linear operator.

The fully-convolutional time-domain audio separation network (Conv-TasNet), a deep learning framework for end-to-end time-domain audio source separation. Conv-TasNet uses a linear encoder to generate a representation of the audio source waveform optimized for separating individual sounds.

III.2. Convolutional Time-Domain Audio Separation Network

The fully-convolutional time-domain audio separation net- work (Conv-TasNet) consists of three (03) processing stages, as shown in figure III.1: encoder, separation, and decoder.

First, an encoder module is used to transform short segments of the mixture waveform into their corresponding representations in an intermediate feature space. This representation is then used to estimate a multiplicative function (mask) for each source at each time step. The source waveforms are then reconstructed by transforming the masked encoder features using a decoder module. We describe the details of each stage in this section.

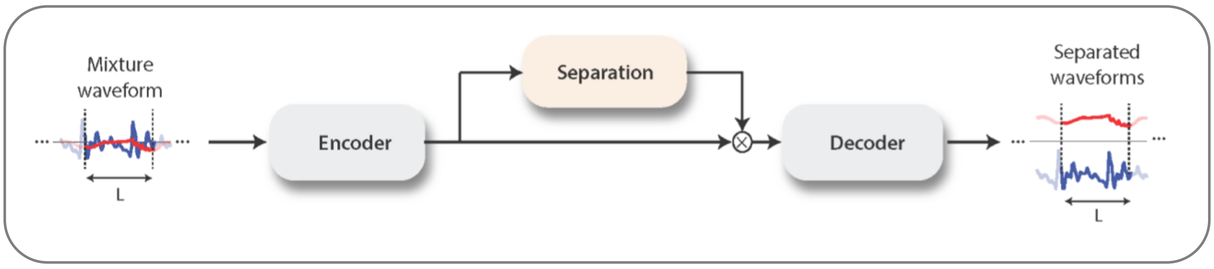


Figure III.1. The block diagram of the TasNet system

III.2.1. Time-domain audio source separation

The problem of single-channel audio source separation can be formulated in terms of estimating C sources $s_1(t), \dots, s_c(t) \in \mathbb{R}^{1\times T}$, given the discrete waveform of the mixture $x(t) \in \mathbb{R}^{1\times T}$, where

$$x(t) = \sum_{i=1}^{C} s_i(t) \tag{III.1}$$

In time-domain audio separation, we aim to directly estimate $s_i(t), i = 1, \dots, C,$ from $x(t)$.

III.2.2. Convolutional encoder-decoder

The input mixture sound can be divided into overlapping segments of length L, represented by $x_k \in \mathbb{R}^{1\times L}$, where $k = 1, \dots, \hat{T}$ denotes the segment index and $\hat{T}$ denotes the total number of segments in the input. x_k is transformed into a N-dimensional representation, $w \in \mathbb{R}^{1\times N}$ by a 1-D convolution operation, which is reformulated as a matrix multiplication (the index k is dropped from now on):

$$w = \mathcal{H}(xU) \tag{III.2}$$

Where $U \in \mathbb{R}^{N\times L}$ contains N vectors (encoder basis functions) with length L each, and $\mathcal{H}(\cdot)$ is an optional nonlinear function. In [21], $\mathcal{H}(\cdot)$ was the rectified linear unit (ReLU) to ensure that the representation is non-negative. The decoder reconstructs the waveform from this representation

Using a 1-D transposed convolution operation, which can be reformulated as another matrix multiplication:

$$\hat{x} = wV \tag{III.3}$$

Where $\hat{x} \in \mathbb{R}^{1\times L}$ is the reconstruction of x, and the rows in $V \in \mathbb{R}^{N\times L}$ are the decoder basis functions, each with length L. The overlapping reconstructed segments are summed together to generate the final waveforms.

Although we reformulate the encoder/decoder operations as matrix multiplication, the term “convolutional auto-encoder” is used because in actual model implementation, convolutional and transposed convolutional layers can more easily handle the overlap between segments and thus enable faster training and better convergence.

III.2.3. Estimating the separation masks

The separation for each frame is performed by estimating C vectors (masks) $m_i \in \mathbb{R}^{1\times N}$, $i = 1,\dots,C$ where C is the number of speakers in the mixture that is multiplied by the encoder output w. The mask vectors m_i have the constraint that $m_i \in [0,1]$. The representation of each source, $d_i \in \mathbb{R}^{1\times N}$, is then calculated by applying the corresponding mask, m_i to the mixture representation w:

$$d_i = w \odot m_i \tag{III.4}$$

Where $\odot$ denotes element-wise multiplication. The waveform of each source $\hat{s}_i$, $i = 1,\dots,C$ is then reconstructed by the decoder:

$$\hat{s}_i = d_i V \tag{III.5}$$

The unit summation constraint in [21], $\sum_{i=1}^{C} m_i = 1$, was applied based on the assumption that the encoder-encoder architecture can perfectly reconstruct the input mixture. In next chapter, we will examine the consequence of relaxing this unity summation constraint on separation accuracy.

III.2.4. Convolutional separation module

Motivated by the temporal convolutional network [35], we propose a fully-convolutional separation module that consists of stacked 1-D dilated convolutional blocks, as shown in figure III.2. Temporal convolutional network (TCN) was proposed as a replacement for RNNs in various sequence modeling tasks. Each layer in a TCN consists of 1-D convolutional blocks with increasing dilation factors. The dilation factors increase exponentially to ensure a sufficiently large temporal context window to take advantage of the long-range dependencies of the audio source signal, as denoted with different colors in figure III.2. In Conv-TasNet, M convolutional blocks with dilation factors $1,2,4,\dots,2^{M-1}$ are repeated R times. The input to each block is zero padded accordingly to ensure the output length is the same as the input. The output of the TCN is passed to a convolutional block with kernel size block $1(1\times1-conv$ block, also known as pointwise convolution) for mask estimation. The $1\times1-conv$ block together with a nonlinear activation function estimates C mask vectors for the C target sources.

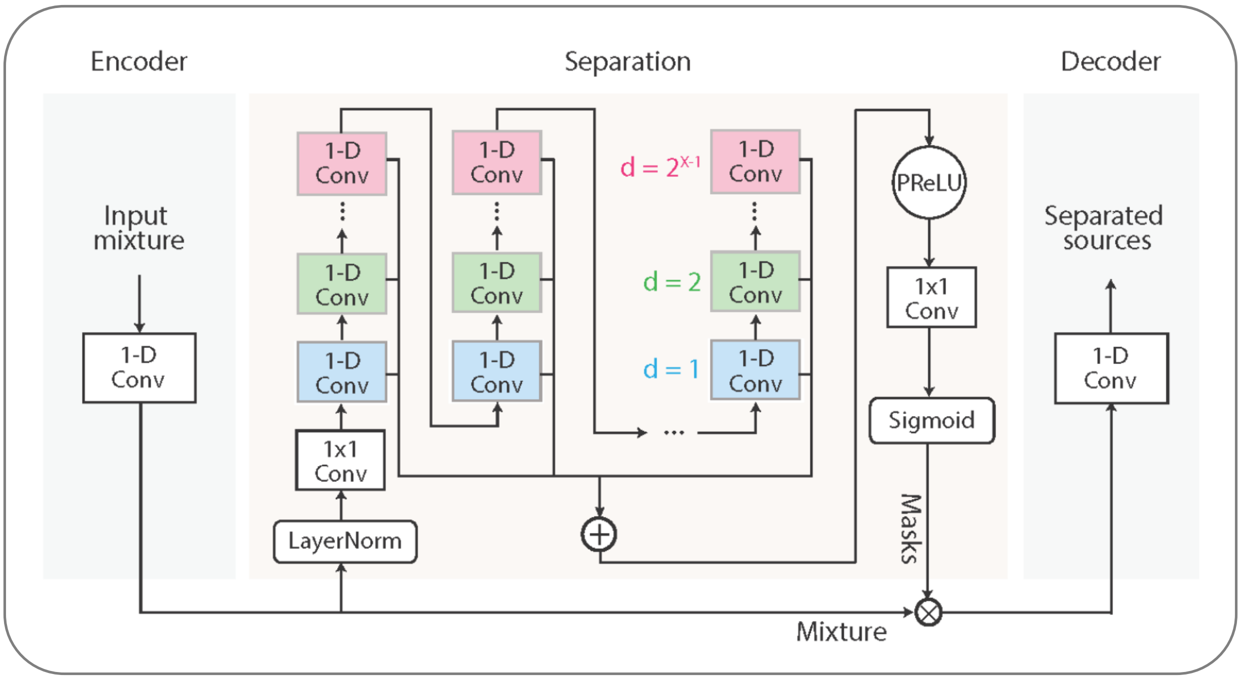


Figure III.2. A flowchart of the proposed system

Figure III.3 shows the design of each 1-D convolutional block. The design of the 1-D convolutional blocks follows [36], where a residual path and a skip-connection path are applied: the residual path of a block serves as the input to the next block, and the skip-connection paths for all blocks are summed up and used as the output of the TCN. To further decrease the number of parameters, depthwise separable convolution ($S-conv(\cdot)$) is used to replace standard convolution in each convolutional block.

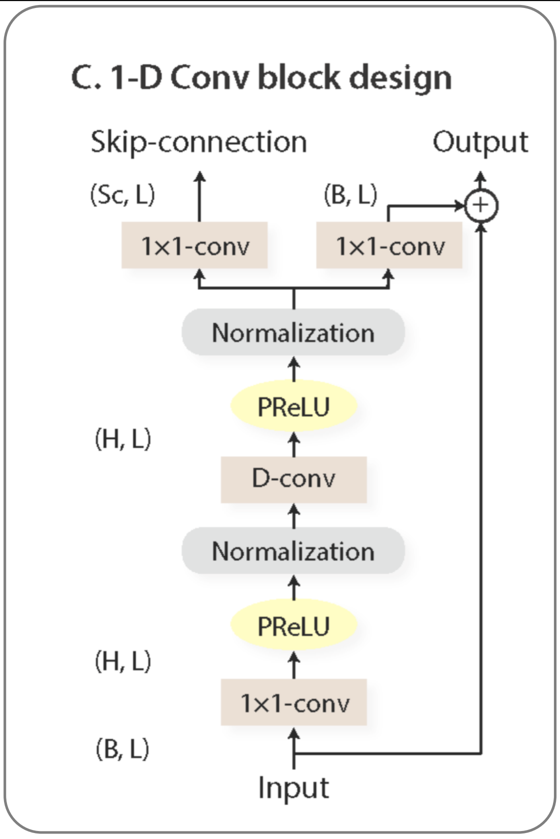


Figure III.3. The design of 1-D convolutional block

Depthwise separable convolution (also referred to as separable convolution) has proven effective in image processing tasks [37] and neural machine translation tasks [38]. The depthwise separable convolution operator decouples the standard convolution operation into two consecutive operations, a depthwise convolution ($D - conv(\cdot)$) followed by pointwise convolution ($1\times1 - conv(\cdot)$):

$$D - conv(\boldsymbol{Y}, \boldsymbol{K}) = concat(\boldsymbol{y}_j \circledast \boldsymbol{k}_j), \quad j = 1, \dots, N \tag{III.6}$$

$$S - conv(\boldsymbol{Y}, \boldsymbol{K}, \boldsymbol{L}) = D - conv(\boldsymbol{Y}, \boldsymbol{K}) \circledast \boldsymbol{L} \tag{III.7}$$

Where $\boldsymbol{Y} \in \mathbb{R}^{G\times M}$ is the input to $S - conv(\cdot)$, $\boldsymbol{K} \in \mathbb{R}^{G\times P}$ is the convolution kernel with size P, $\boldsymbol{y}_j \in \mathbb{R}^{1\times M}$ and $\boldsymbol{k}_j \in \mathbb{R}^{1\times P}$ are the rows of matrices **Y** and **K** , respectively, $\boldsymbol{L} \in \mathbb{R}^{G\times H\times 1}$ is the convolution kernel with size 1, and $\circledast$ denotes the convolution operation. In other words, the $D - conv(\cdot)$ operation convolves each row of the input Y with the corresponding row of matrix K, and the $1\times1 - conv$ block linearly transforms the feature space. In comparison with the standard convolution with kernel size $\widehat{K} \in \mathbb{R}^{G\times H\times P}$ depthwise separable convolution only contains $G{\times}P + G{\times}H$ parameters, which decreases the model size by a factor of $\frac{H \times P}{H+P} \approx P$ when $H \gg P$.

A nonlinear activation function and a normalization operation are added after both the first $1{\times}1 - conv$ and $D - conv$ blocks respectively. The nonlinear activation function is the parametric rectified linear unit (PReLU) [39]:

$$PReLU(x) = \begin{cases} x, & if\ x \geq 0 \\ \alpha x, & otherwise \end{cases} \tag{III.8}$$

Where $\alpha \in \mathbb{R}$ is a trainable scalar controlling the negative slope of the rectifier. The choice of the normalization method in the network depends on the causality requirement. For noncausal configuration, we found empirically that global layer normalization (gLN) outperforms all other normalization methods. In gLN, the feature is normalized over both the channel and the time dimensions:

$$gLN(\boldsymbol{F}) = \frac{F - E[\boldsymbol{F}]}{\sqrt{Var[\boldsymbol{F}] + \epsilon}} \odot \gamma + \beta \tag{III.9}$$

$$E[\boldsymbol{F}] = \frac{1}{NT} \sum_{NT} \boldsymbol{F} \tag{III.10}$$

$$Var[\boldsymbol{F}] = \frac{1}{NT} \sum_{NT} (F - E[\boldsymbol{F}])^2 \tag{III.11}$$

Where $\boldsymbol{F} \in \mathbb{R}^{N \times T}$ is the feature, $\gamma, \beta \in \mathbb{R}^{N \times 1}$ are trainable parameters, and ϵ is a small constant for numerical stability. This is identical to the standard layer normalization applied in computer vision models where the channel and time dimension correspond to the width and height dimension in an image [40]. In causal configuration, gLN cannot be applied since it relies on the future values of the signal at any time step. Instead, we designed a cumulative layer normalization (cLN) operation to perform step-wise normalization in the causal system:

$$cLN(\boldsymbol{f}_k) = \frac{\boldsymbol{f}_k - E[\boldsymbol{f}_{t\leq k}]}{\sqrt{Var[\boldsymbol{f}_{t\leq k}] + \epsilon}} \odot \gamma + \beta \tag{III.12}$$

$$E[\boldsymbol{f}_{t\leq k}] = \frac{1}{Nk}\sum_{Nk} \boldsymbol{f}_{t\leq k} \tag{III.13}$$

$$Var[\boldsymbol{f}_{t\leq k}] = \frac{1}{Nk}\sum_{Nk} (\boldsymbol{f}_{t\leq k} - E[\boldsymbol{f}_{t\leq k}])^2 \tag{III.14}$$

Where $\boldsymbol{f}_k \in \mathbb{R}^{N\times 1}$ is the k-th frame of the entire feature **F**, $\boldsymbol{f}_{t\leq k} \in \mathbb{R}^{N\times k}$ corresponds to the feature of k frames $[\boldsymbol{f}_1, \dots, \boldsymbol{f}_k]$ and $\gamma, \beta \in \mathbb{R}^{N\times 1}$ are trainable parameters applied to all frames. To ensure that the separation module is invariant to the scaling of the input, the selected normalization method is applied to the encoder output **w** before it is passed to the separation module.

At the beginning of the separation module, a linear $1\times 1 - conv$ block is added as a bottleneck layer. This block determines the number of channels in the input and residual path of the subsequent convolutional blocks. For instance, if the linear bottleneck layer has B channels, then for a 1-D convolutional block with H channels and kernel size P, the size of the kernel in the first $1\times 1 - conv$ block and the first $D - conv$ block should be $\boldsymbol{O} \in \mathbb{R}^{B\times H\times 1}$ and $\boldsymbol{K} \in \mathbb{R}^{H\times P}$ respectively, and the size of the kernel in the residual paths should be $\boldsymbol{L}_{R_s} \in \mathbb{R}^{H\times B\times 1}$. The number of output channels in the skip-connection path can be different than B, and we denote the size of kernels in that path as $\boldsymbol{L}_{S_c} \in \mathbb{R}^{H\times S_c\times 1}$.

III.3. Conclusion

In this chapter, we introduced the Conv-TasNet audio source separation system. We have presented in detail the different Conv-Tasnet component blocks namely the encoder/decoder and the separation module based on the CNN 1D convolutional neural network.

In the next chapter, we will implement this system and apply it on a database of popular speech mixtures, then we will apply it on our own database of bird sounds.

CHAPTER IV

Experiment and results

IV.1. Introduction

In this last chapter, we will present the implementation of Conv-TasNet on our dataset of birds that we composed from xeno-canto.org. We will train our system and test it using a python code. Next, we will discuss the results achieved and the performance of our bird sound separation system.

IV.2. Programming Languages

The programing and development of the project has been made using Python and run the code using Shell.

Python 87.5 %
Shell 12.5 %

IV.2.1. Presentation of programming languages:

A. What is Shell language?

At its base, a shell is simply a macro processor that executes commands. The term macro processor means functionality where text and symbols are expanded to create larger expressions.

A Unix shell is both a command interpreter and a programming language. As a command interpreter, the shell provides the user interface to the rich set of GNU utilities. The programming language features allow these utilities to be combined. Files containing commands can be created, and become commands themselves. These new commands have the same status as system commands in directories such as /bin, allowing users or groups to establish custom environments to automate their common tasks.

Shells provide a small set of built-in commands (builtins) implementing functionality impossible or inconvenient to obtain via separate utilities. like, cd, bash [1], pwd.

Shell language is available on Linux and Mac OS and don't exist on Windows.

[1] Bash is a command language interpreter, for the GNU operating system.

B. What is Python?

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source the fast edit-test-debug cycle makes this simple approach very effective.

DL has become one of fastest-moving areas of IT and one of Python’s star use cases. The vast majority of the libraries used for DL have Python interfaces, making the language the most popular high-level command interface to for DL libraries and other numerical algorithms.

C. Installation of Python

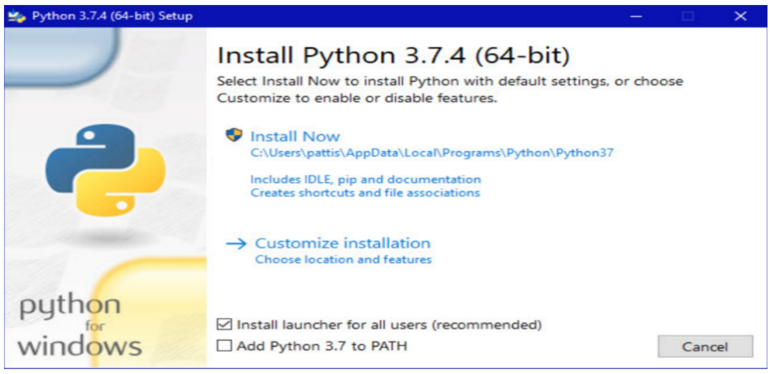
As we know python have two different versions (python 2 and python 3) .In this section we will explain briefly how to download and install python3 for windows.

To install Python for Microsoft Windows (7, 8,10) download latest version of python from here : *http://www.python.org/downloads/windows*

If your browser asks whether to save or keep your file, choose to save it.

Once you've downloaded the Python for Windows installation file you should be prompted to run it. If not, open your Downloads folder and double-click the file. Now, follow the installation instructions:

- Select Add python to path and then click Install Now.
- For the optional Features you should select pip[2].
- When asked whether to allow the program to make changes to your computer, choose Next
- Click Close once installation finishes, and you should see a Python 3 entry in your Windows Start menu.


Python 3.7.4 (64-bit) Setup

Install Python 3.7.4 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\pattis\AppData\Local\Programs\Python\Python37

Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☐ Add Python 3.7 to PATH

Cancel

Figure IV.1. Python installation

When installing Python, you'll also install the IDLE program, which is the Integrated DeveLopment Environment that lets you write programs for Python.

See the annex, there is a detailed explanation of how to install python for Mac OS, You can also install Anaconda, It contains python and DL Tools. See installation steps from here: https://docs.anaconda.com/anaconda/install/

[2] Pip is the package installer for Python. Which can download and install other python packages.

D. Used packages

The following packages were used in our work:

- *PyTorch*: module is a Python based framework for programming neural networks, that contains implementation of linear, LSTM, BLSTM, convolutional layers, backpropagation algorithms and loss functions.
- *CUDA:* NVIDIA's CUDA Python provides a driver and runtime API for existing toolkits and libraries to simplify GPU-based accelerated processing.
- *Asteroid*: PyTorch-based audio source separation toolkit for researchers.
- *Soundfile:* package is used for loading mixtures from sound files in dataset folders and for writing outputs of testing.
- *Numpy:* package is used for math.
- *Pandas*: a Python package that provides fast, flexible, and expressive data structures designed to make working with structured and time series data both easy and intuitive.
- *Torch*: A Tensors and Dynamic neural networks in Python with strong GPU acceleration.
- *CUDA:* NVIDIA's CUDA Python provides a driver and runtime API for existing toolkits and libraries to simplify GPU-based accelerated processing.
- *Data loader :* This module provides loading files from the dataset which is stored in the folder structure

IV.2.2. PRESENTATION OF ASTEROID:

Asteroid is a Pytorch-based audio source separation toolkit that enables fast experimentation on common datasets. It comes with a source code that supports a large range of datasets and architectures, and a set of recipes to reproduce some important papers.

Asteroid has several target usage:

- Use asteroid in your own code, as a package.
- Use available recipes to build your own separation model.
- Use pretrained models to process your files.
- Hit the ground running with your research ideas

A. Installation

Asteroid is on the python package index (PyPI), you can install the latest release with the instruction:

```
o pip install asteroid
```

To run Asteroid on GPU, you will need a CUDA-enabled PyTorch installation.

B. Functionality

Asteroid follows the encoder-masker-decoder approach, and provides various choices of filterbanks, masker networks, and loss functions. It also provides training and evaluation tools and recipes for several datasets.

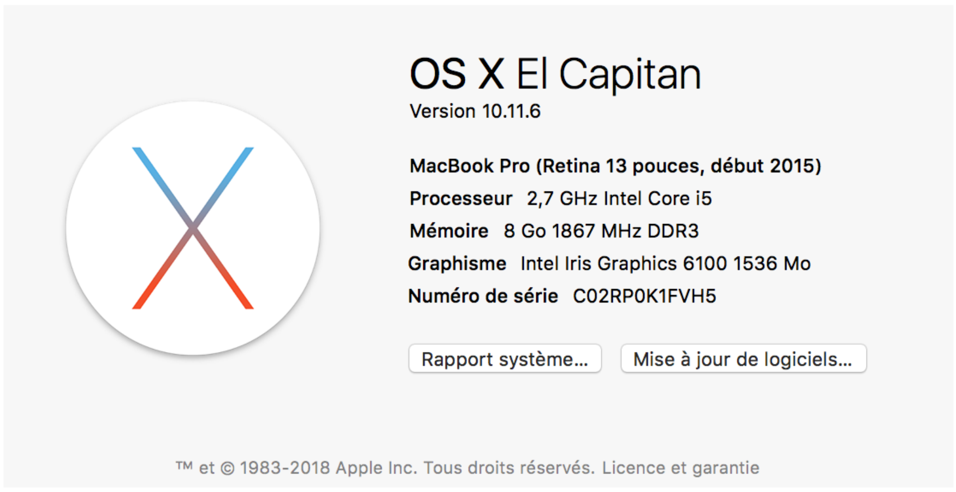
C. Supported Datasets

The following is a list of supported datasets, sorted by task. As you can use asteroid on your own dataset and for your own task.

- Speech separation
  - WSJ0-Mix dataset
  - WHAM dataset
  - WHAMR dataset
  - Kinect-WSJ dataset
  - LibriMix dataset
  - SMS_WSJ dataset
- Speech enhancement
  - DNS Challenge's dataset
- Music source separation
  - MUSDB18 Dataset
  - DAMP-VSEP dataset
- Audio-visual source separation
  - AVSpeech dataset
- Environmental sound separation
  - FUSS dataset

IV.2.3. Implementation

The experiments were performed on a MacBook Pro a computer with i7 core 8G RAM and 256G ROM , the reason for that is that windows can't run shell scripts and also the experiment is a deep learning project to realize it we will need a computer with high hardware configuration.



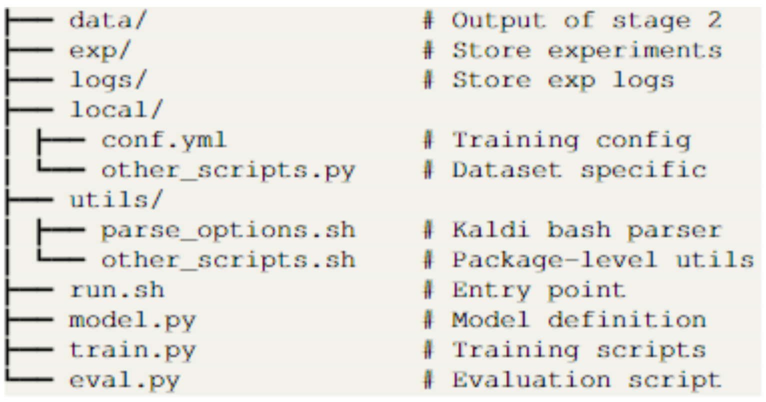
OS X El Capitan
Version 10.11.6

MacBook Pro (Retina 13 pouces, début 2015)
Processeur 2,7 GHz Intel Core i5
Mémoire 8 Go 1867 MHz DDR3
Graphisme Intel Iris Graphics 6100 1536 Mo
Numéro de série C02RP0K1FVH5

™ et © 1983-2018 Apple Inc. Tous droits réservés. Licence et garantie

Figure IV.2. The characteristics of the machine used

Our system is implemented in Python using all libraries mentioned in used package. All audios are resampled to 16kHz.all others parameters were mentioned in next sections.



```
├── data/                    # Output of stage 2
├── exp/                     # Store experiments
├── logs/                    # Store exp logs
├── local/
│   ├── conf.yml             # Training config
│   └── other_scripts.py     # Dataset specific
├── utils/
│   ├── parse_options.sh     # Kaldi bash parser
│   └── other_scripts.sh     # Package-level utils
├── run.sh                   # Entry point
├── model.py                 # Model definition
├── train.py                 # Training scripts
└── eval.py                  # Evaluation script
```

Figure IV.3. Typical directory of a recipe

Implementation has 4 stages as follows:

- Stage 1: Generate mixture.
- Stage 2: Gather data information into text files.
- Stage 3: Train the source separation system.
- Stage 4: Separate test mixtures and evaluate.

First, we create our dataset. Then, we use the mixture script to generate the data. All the information required by the dataset's `DataLoader` such as file names and paths, sounds lengths is then gathered into text files under `data/`. The training stage is finally followed by the evaluation stage.

The model class, which is a direct subclass of `PyTorch`'s `nn.Module`, is defined in `model.py`. It is imported in both training and evaluation scripts. Instead of defining constants in `model.py` and `train.py`, most of them are gathered in a YAML configuration file `conf.yml`. An argument parser is created from this configuration file to allow modification of these values from the command line, with `run.sh` passing arguments to `train.py`. The resulting modified configuration is saved in `exp/` to enable future reuse. Other arguments such as the experiment name, the number of GPUs, etc., are directly passed to `run.sh`.

Full python code is written in the annex.

Model architecture:

The encoder consists of just one convolution and the decoder, symmetrically, of a transposed convolution. The separation module is a temporal convolutional network, composed of stacks of convolutional blocks with exponentially increasing dilation factors, which have both skip connections and residual connections, and ending with a 1x1 convolution followed by a sigmoid that produces the masks for each of the sources. A convolutional block uses 1x1 convolutions and a depthwise convolution, to reduce the number of parameters, as well as group layer normalization and PReLU activations. The model is trained using SI-SDR as the cost function.

IV.3. Experiment:

Dataset: Bird Sound Dataset (BSD)

We decided to gather our data from xeno-canto.org, which is a popular open source bird recording website, with over 500,000 recordings. We liked this source because we could specifically pick the birds, size, and quality of our data.

Through a Python wrapper from Github we downloaded our dataset from xeno-canto.org composed of bird sounds. We chose to build our dataset to look like the Wall Street Journal (WSJ0) dataset. With same characteristic exist in WSJ0.

We evaluated our system on two-bird and three-bird sound separation problems using the BSD-2mix and BSD-3mix datasets. 3 hours of training and 1 hour of validation data are generated from birds in datasets. The sound mixtures are generated by randomly selecting vocalizations from different birds in the bird sound dataset (BSD) and mixing them at random signal-to-noise ratios (SNR) between -5 dB and 5 dB. 30 min evaluation set is generated in the same way. All the waveforms are resampled at 8 kHz.

IV.3.1. Experiment configurations:

The networks are trained for 10 epochs on 4-second long segments. The initial learning rate is set to $1e^{-3}$. The learning rate is halved if the accuracy of validation set is not improved in 3 consecutive epochs. Adam [3][41] is used as the optimizer. A 50% stride size is used in the convolutional autoencoder (i.e. 50% overlap between consecutive frames). Gradient clipping with maximum L_2-norm of 5 is applied during training. The hyperparameters of the network are shown in table I

[3] Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.

Symbol	Description
N	Number of filters in autoencoder
L	Length of the filters (in samples)
B	Number of channels in bottleneck and the residual paths' $1 \times 1 - conv$ blocks
Sc	Number of channels in skip-connection paths' $1 \times 1 - conv$ blocks
H	Number of channels in convolutional blocks
P	Kernel size in convolutional blocks
X	Number of convolutional blocks in each repeat
R	Number of repeats

Table IV.1. Hyperparameters of the network

Training objective :

The objective of training the end-to-end system is maximizing the scale-invariant source-to noise ratio (SI-SNR), which has commonly been used as the evaluation metric for source separation replacing the standard source-to-distortion ratio. SI-SNR is defined as:

$$\begin{cases} s_{target} := \frac{\langle \hat{s}, s \rangle s}{\|s\|} \\ e_{\ noise} := \hat{s} - s_{target} \\ SI - SNR := 10log_{10} \frac{\|s_{target}\|^2}{\|e_{noise}\|^2} \end{cases}$$

Where $\hat{s} \in \mathbb{R}^{1\times T}$ and $s \in \mathbb{R}^{1\times T}$ are the estimated and original clean sources, respectively, and $\|s\|^2 = \langle s, s \rangle$ denotes the signal power. Scale invariance is ensured by normalizing $\hat{s}$ and s to zero-mean prior to the calculation. Audio-level permutation invariant training (uPIT) is applied during training to address the source permutation problem.

Evaluation metrics:

We report the scale-invariant signal-to-noise ratio improvement (SI-SNRi) and signal-to-distortion ratio improvement (SDRi) as objective measures of separation accuracy. The reported improvements in tables indicate the additive values over the original mixture. In addition to the distortion metrics,

IV.4. Results

Figure IV.3 visualizes all the internal variables of Conv-TasNet for one example mixture sound with two overlapping birds sound (denoted by red and blue). The encoder and decoder basis functions are sorted by the similarity of the Euclidean distance of the basis functions found using the unweighted pair group method with arithmetic mean (UPGMA) method [42]. The basis functions show a diversity of frequency and phase tuning. The representation of the encoder is colored according to the power of each bird's sound at the corresponding basis output at each time point, demonstrating the sparsity of the encoder representation. As can be seen in Figure IV.3, the estimated masks for the two sounds highly resemble their encoder representations, which allows for the suppression of the encoder outputs that correspond to the interfering sound and the extraction of the target sound in each mask. The separated waveforms for the two birds sound are estimated by the linear decoder, whose basis functions are shown in Figure IV.3. The separated waveforms are shown on the right.

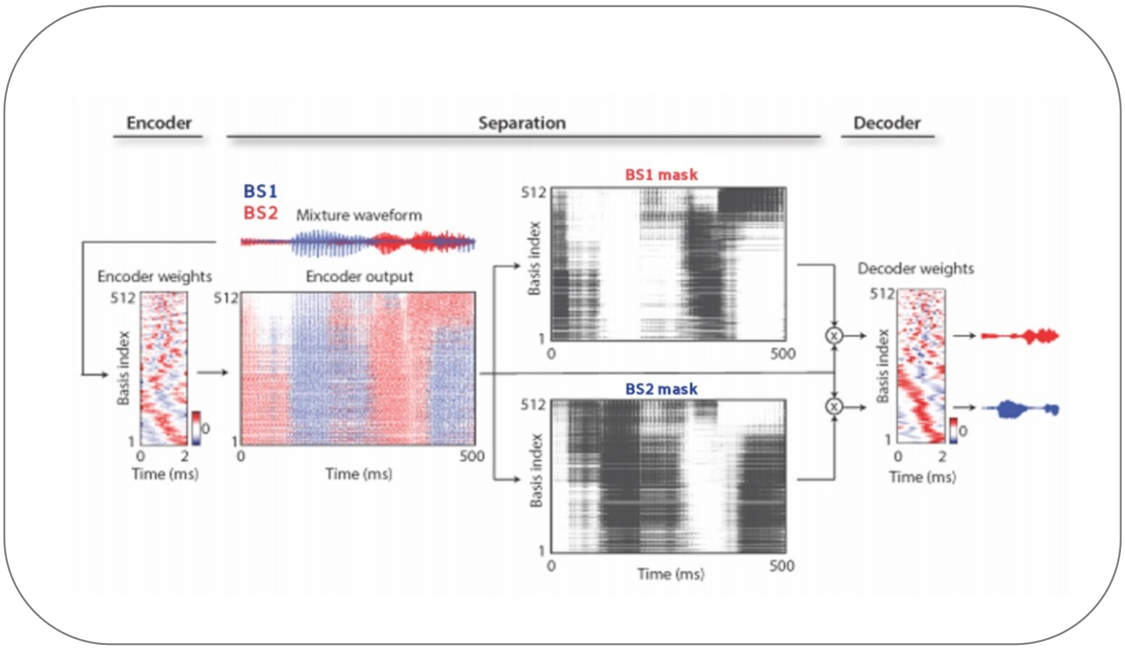


Figure IV.3. Sound Separation Architecture

Figure IV.3. demonstrate a visualization of the encoder and decoder basis functions, encoder representation, and source masks for a sample 2-birds sound mixture. The birds are shown in red and blue. The encoder representation is colored according to the power of each birds at each basis function and point in time. The basis functions are sorted according to their Euclidean similarity and show diversity in frequency and phase tuning

Non-negativity of the encoder output

The non-negativity of the encoder output was enforced in [21], [43] using a rectified-linear nonlinearity (ReLU) function. This constraint was based on the assumption that the masking operation on the encoder output is only meaningful when the mixture and speaker waveforms can be represented with a non-negative combination of the basis functions, since an unbounded encoder representation may result in unbounded masks. However, by removing the nonlinear function H, another assumption can be made: with an unbounded but highly overcomplete representation of the mixture, a set of non-negative masks can still be found to reconstruct the clean sources. In this case, the overcompleteness of the representation is crucial.

If there exist only a unique weight feature for the mixture as well as for the sources, the non-negativity of the mask cannot be guaranteed. Also note that in both assumptions, we put no constraint on the relationship between the encoder and decoder basis functions U and V, meaning that they are not forced to reconstruct the mixture signal perfectly.

One way to explicitly ensure the autoencoder property is by choosing V to be the pseudo-inverse of U (i.e. least square reconstruction). The choice of encoder/decoder design affects the mask estimation: in the case of an autoencoder, the unit summation constraint must be satisfied; otherwise, the unit summation constraint is not strictly required. To illustrate this point, we compared five different encoder-decoder configurations:

1) Linear encoder with its pseudo-inverse (Pinv) as decoder, i.e. $w = x(V^TV)^{-1}V^T$ and $\hat{x} = wV$, with Softmax function for mask estimation.
2) Linear encoder and decoder where $w = xU$ and $\hat{x} = wV$, with Softmax or Sigmoid function for mask estimation.
3) Encoder with ReLU activation and linear decoder where $w = ReLU\,(xU)$ and $\hat{x} = wV$, with Softmax or Sigmoid function for mask estimation.

Separation accuracy of different configurations in table III shows that pseudo-inverse autoencoder [4]leads to the worst performance, indicating that an explicit autoencoder configuration does not necessarily improve the separation score in this framework. The performance of all other configurations is comparable. Because linear encoder and decoder

[4] Autoencoders are a special class of neural networks designed to find an efficient representation of data by learning correlations between the data points.

with Sigmoid function achieves a slightly better accuracy over other methods, we used this configuration in all the following experiments.

Encoder	Mask	Model Size	SI-SNRI (dB)	SDRi (dB)
Pinv	Softmax		12.1	12.4
Linear	Softmax		12.9	13.2
	Sigmoid	**1.5M**	**13.1**	**13.4**
ReLU	Softmax		13.0	13.3
	Sigmoid		12.9	13.2

Table IV.2. Separation score for different system configurations

IV.4.1. Optimizing the network parameters:

We evaluate the performance of Conv-TasNet on two bird separation tasks as a function of different network parameters. Table II shows the performance of the systems with different parameters, from which we can conclude the following statements:

- Encoder/decoder: Increasing the number of basis signals in the encoder/decoder increases the overcompleteness of the basis signals and improves the performance.
- Hyperparameters in the 1-D convolutional blocks: A possible configuration consists of a small bottleneck size B and a large number of channels in the convolutional blocks H. where the ratio between the convolutional block and the bottleneck H/B was found to be best around 5. Increasing the number of channels in the skip-connection block improves the performance while greatly increases the model size. Therefore, we selected a small skip-connection block as a trade-off between performance and model size.
- Number of 1-D convolutional blocks: When the receptive field is the same, deeper networks lead to better performance, possibly due to the increased model capacity.
- Size of receptive field: Increasing the size of receptive field leads to better performance, which shows the importance of modeling the temporal dependencies in the audio signal.

- Length of each segment: Shorter segment length consistently improves performance. Note that the best system uses a filter length of only 2 ms ($\frac{L}{f_s} = \frac{16}{8000} = 0{:}\,002s$),
- Causality: Using a causal configuration leads to a significant drop in the performance. This drop could be due to the causal convolution and/or the layer normalization operations.

N	**L**	**B**	**H**	***Sc***	**P**	**X**	**R**	Normali-zation	Causal	Receptive field (s)	Model size	SI-SNRi (dB)	SDRi (dB)
128	40	128	256	128	3	7	2	gLN	×	1.28	1.5M	13.0	13.2
256	40	128	256	128	3	7	2	gLN	×	1.28	1.5M	13.1	13.3
512	40	128	256	128	3	7	2	gLN	×	1.28	1.7M	13.3	13.5
512	40	128	256	256	3	7	2	gLN	×	1.28	2.4M	13.0	13.3
512	40	128	512	128	3	7	2	gLN	×	1.28	3.1M	13.3	13.6
512	40	128	512	512	3	7	2	gLN	×	1.28	6.2M	13.5	13.8
512	40	128	256	256	3	7	2	gLN	×	1.28	3.2M	13.0	13.3
512	40	128	512	256	3	7	2	gLN	×	1.28	6.0M	13.4	13.7
512	40	128	512	512	3	7	2	gLN	×	1.28	6.5M	13.2	13.5
512	40	128	512	128	3	6	4	gLN	×	1.27	5.1M	14.1	14.4
512	40	128	512	128	3	4	6	gLN	×	0.46	5.1M	13.9	14.2
512	40	128	512	128	3	8	3	gLN	×	3.83	5.1M	14.4	14.7
512	32	128	512	128	3	8	3	gLN	×	3.06	5.1M	14.7	15.0
512	16	128	512	128	3	8	3	gLN	×	1.53	5.1M	**15.1**	**15.4**
512	16	128	512	128	3	8	3	cLN	**YES**	1.53	5.1M	10.6	11.0

Table IV.3. The effect of different configurations in Conv-TasNet

IV.4.2. Comparison of Conv-TasNet on other datasets

We compared the separation accuracy of Conv-TasNet method using SDRi and SI-SNRi on other datasets (WSJ0-2Mix, MiniLibriMix) with our dataset BSD. Unlike our dataset, the two other datasets composed of utterances from different speakers (speech)

We list the best results that have been obtained in the experiments. Except for wsj0 dataset, we listed the results that have been reported in literature .The configurations of Conv-TasNet is the same in all experiments. Table IV.4 compares the performance of Conv-TasNet on the three datasets.

Dataset	Method	SI-SNRi (dB)	SDRi (dB)
BSD	Conv-TasNet	15,1	15.4
MiniLibriMix		14.4	14.7
WSJ0-2Mix		**15.3**	**15.6**

Table IV.4. Comparison of Conv-TasNet on other datasets.

The table IV.4 shows that the MiniLibriMix have the worst performance because of her small size. For this dataset size is important to assure a better performance when we train the network. As shown too that the using of WSJ0-2Mix leads to the best performance and the performance of our dataset is comparable.

IV.5. Conclusion

In this last chapter we studied the fully-convolutional time-domain audio separation network (Conv-TasNet) for bird sounds separation. A deep learning audio separation system that directly operates on the sound. We implement the network, train and evaluate it and show the results. This allowed to a better understanding of the system studied (Conv-TasNet). Experiments results shown that the performance of separation sounds is significantly improved especially in noisy conditions. In addition the effectiveness of the Conv-TasNet system for separation bird sounds has been proven and given an acceptable results compared to other sounds especially with speech.

GENERAL CONCLUSION

In order to achieve good performance in identifying bird species based on their sounds, the separation of bird sounds is a very important and crucial step

In this thesis, we have presented a robust bird sound separation system. a fully-convolutional neural network system in time-domain named (Conv-TasNet) composed of three main blocks: an encoder, a separation model based on the neural network CNN 1D and a decoder

This work was based on four main chapters:

In the first chapter, we studied the audio source separation. After that, we made a state of the art about the related methods on audio separation found in the literature.

In the second chapter, we first presented the deep learning with her different techniques, then, we studied the different types of DL networks and focused on the CNN.

In the third chapter, we explained in detail each step of the proposed bird sound separation system the Conv-TasNet.

In the last chapter, we implemented the Conv-TasNet system. Then, we evaluated the performance proposed system. after that, we presented the results obtained and compared it with similar works

The results obtained demonstrated the effective of the Conv-TasNet system and works satisfactorily for bird sound separation.

In conclusion, Conv-TasNet represents a significant step toward the realization of audio source separation algorithms and opens many future research directions that would further improve its accuracy, speed, and computational cost, which could eventually make automatic audio source separation a common and necessary feature of every audio processing technology designed for real-world applications.

APPENDIX

A. How to Install Python on Macintosh

Python is one of the oldest programming languages around. However, with the onset of Machine Learning, Python has been given a new lease of life. It has become a popular tool for both Machine Learning and Deep Learning.

Currently, Python is available as two distinct versions. That is Python 2 and Python 3. In this tutorial, you are going to learn how install both versions on your system.

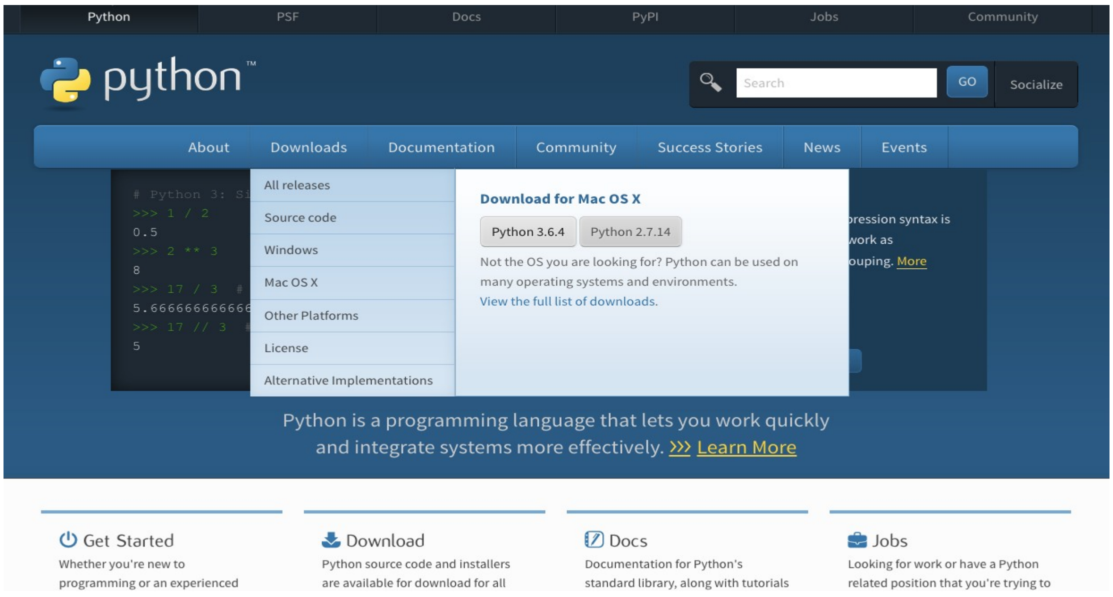
Selecting a Version of Python

With two great choices, how do you decide on which one to use? It all comes down to compatibility. Certain programmes or libraries that you want to use in your project may only be compatible with one version of Python.

So keep your end goal in mind when selecting a version. If you want to have more flexibility when it comes to Python projects, just download both versions.

Begin Installation

1. Visit https://www.python.org/



```
# Python 3: Si
>>> 1 / 2
0.5
>>> 2 ** 3
8
>>> 17 / 3  #
5.666666666666
>>> 17 // 3  #
5
```

Download for Mac OS X

Not the OS you are looking for? Python can be used on many operating systems and environments. View the full list of downloads.

pression syntax is work as ouping. More

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

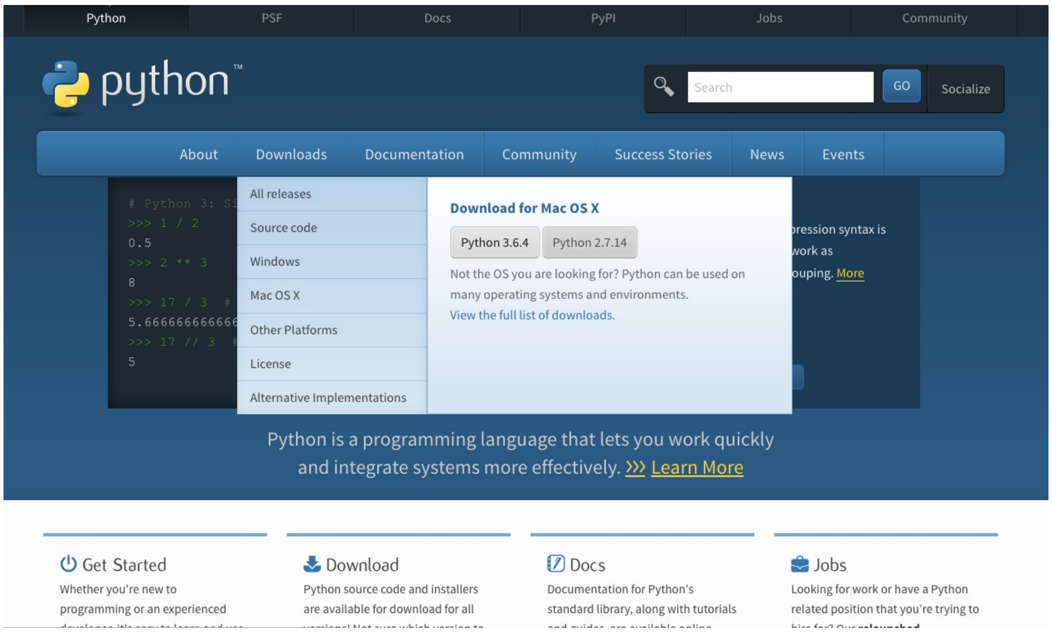
Whether you're new to programming or an experienced

Python source code and installers are available for download for all

Documentation for Python's standard library, along with tutorials

Looking for work or have a Python related position that you're trying to

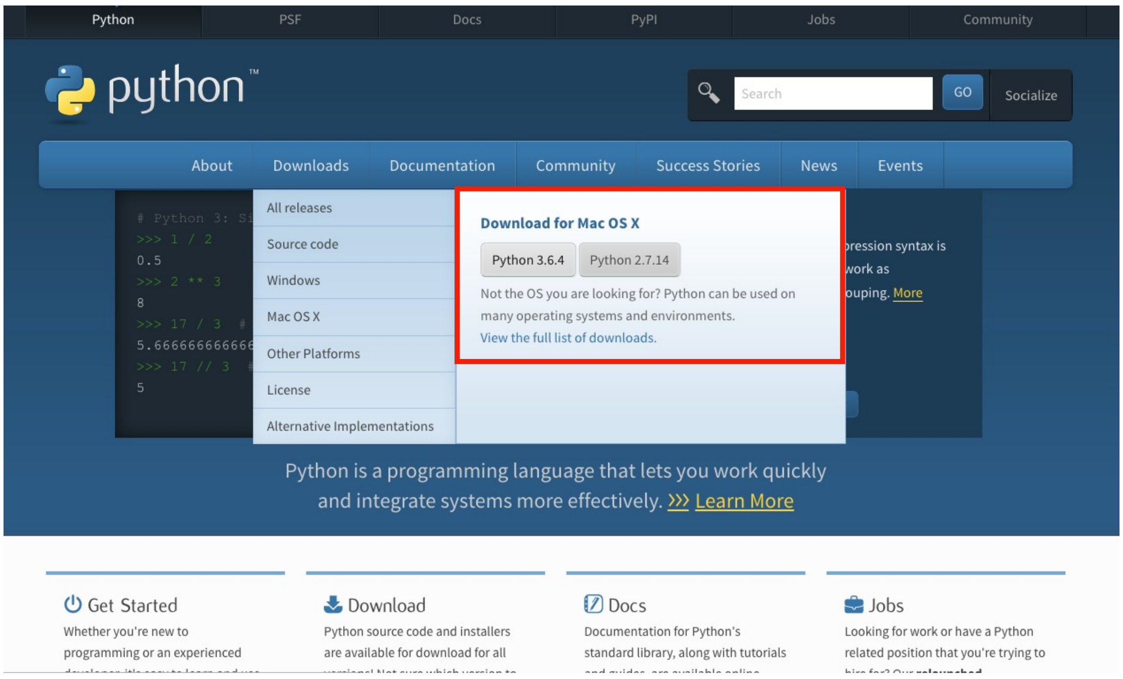
2. Select the 'Downloads' tab. A drop down menu will appear.


Download for Mac OS X

Not the OS you are looking for? Python can be used on many operating systems and environments. View the full list of downloads.

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

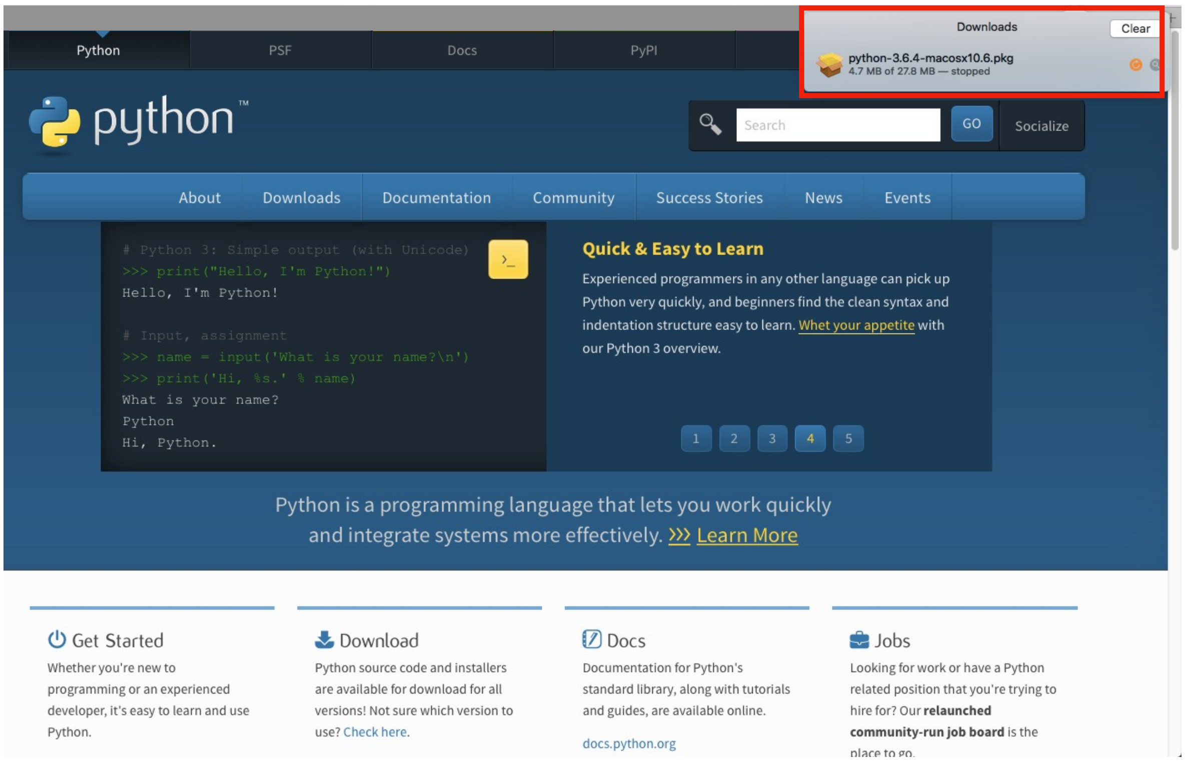
3. To the right of the drop down menu, you will see the latest versions of python that are available for MAC. The first button provides the latest version of Python 3 and the second button provides the option for the latest version of Python 2. Once you click an option, the download will begin.



Not the OS you are looking for? Python can be used on many operating systems and environments. View the full list of downloads.

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

4. Once installation is complete, double click the package in the download bar. This will start the installation process.



```
# Python 3: Simple output (with Unicode)
>>> print("Hello, I'm Python!")
Hello, I'm Python!

# Input, assignment
>>> name = input('What is your name?\n')
>>> print('Hi, %s.' % name)
What is your name?
Python
Hi, Python.
```

Quick & Easy to Learn

Experienced programmers in any other language can pick up Python very quickly, and beginners find the clean syntax and indentation structure easy to learn. Whet your appetite with our Python 3 overview.

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

Get Started: Whether you're new to programming or an experienced developer, it's easy to learn and use Python.

Download: Python source code and installers are available for download for all versions! Not sure which version to use? Check here.

Docs: Documentation for Python's standard library, along with tutorials and guides, are available online. docs.python.org

Jobs: Looking for work or have a Python related position that you're trying to hire for? Our **relaunched community-run job board** is the place to go.

5. In the dialogue box that pops up you will be shown a welcome notice, select "Continue".
6. In the new dialogue box, you will be presented with important information regarding the changes made to Python, once again select "Continue".
7. Now you will be shown the terms and conditions for using Python. Select "Continue".
8. A mini dialogue box will appear requesting you to agree to the terms and conditions listed. Select "Agree".
9. Finally you will be told how much memory will be used on your system. Select "install"
10. For security purposes, the system will request you to enter your user name and password. Enter the details, then select "Install Software"
11. Now the dialogue box will display a progress bar to indicate how much of the installation is complete. This should only take a few minutes depending on your system's memory and speed.
12. Once the installation is complete, you will be presented with a dialogue box indicating that your installation was successful.

Test your Installation

Now that you have downloaded Python, test it to make sure it is working correctly.

- Open the Mac Terminal.
- Type python2 (or python3, depending on the version you installed) and press enter.
- The version of your Python installation should show up

B. Asteroid:

Installation :

By following the instructions below, first install PyTorch and then Asteroid (using either pip/dev install). We recommend the development installation for users likely to modify the source code.

CUDA and PyTorch

Asteroid is based on PyTorch. To run Asteroid on GPU, you will need a CUDA-enabled PyTorch installation. Visit this site for the instructions :

Pip

Asteroid is regularly updated on PyPI, install the latest stable version with:

```
pip install asteroid
```

Development installation

For development installation, you can fork/clone the GitHub repo and locally install it with pip:

```
git clone https://github.com/asteroid-
team/asteroid cd asteroid

pip install-e.
```

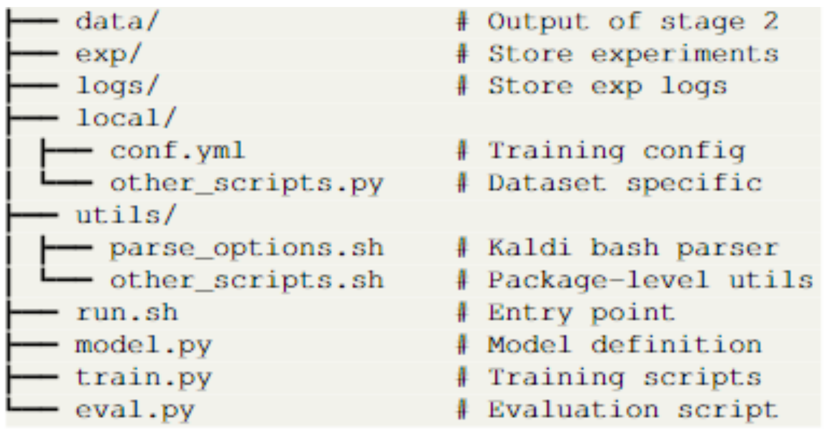
This is an editable install (-e flag), it means that source code changes (or branch switching) are automatically taken into account when importing asteroid.

You can also use `conda env create -f environment.yml` to create a `Conda env` directly.

Asteroid Recipe:

A recipe is a set of scripts that use Asteroid to build a source separation system. our directory corresponds to BSD dataset and each subdirectory corresponds corresponds to our system build on this dataset.

Our recipe is organized as follows:



```
├── data/                  # Output of stage 2
├── exp/                   # Store experiments
├── logs/                  # Store exp logs
├── local/
│  ├── conf.yml            # Training config
│  └── other_scripts.py    # Dataset specific
├── utils/
│  ├── parse_options.sh    # Kaldi bash parser
│  └── other_scripts.sh    # Package-level utils
├── run.sh                 # Entry point
├── model.py               # Model definition
├── train.py               # Training scripts
└── eval.py                # Evaluation script
```

How our recipe work:

As we said we created our recipe in the same way exist in asteroid. Now we explain how our work:

- There is a master file, `run.sh`, from which all the steps are run (install dependencies, download data, create dataset, train a model evaluate it and so on...).
  - We change some variables in the top of the file (comments are around it) like data directory, python path etc.
  - This script is controlled by several arguments. Among them, `stage` controls from where do you start the script. For example you already generated the data? No need to do it again, set `stage=3`!
  - All steps until training are dataset-specific and the corresponding scripts are stored in `./local`.

- The training and evaluation scripts are then called from `run.sh`

  - There is a script, `model.py`, where the model should be defined along with the System subclass used for training (if needed).
  - We wrap the model definition in one function (`make_model_and_optimizer`). The function receives a dictionary which is also saved in the experiment folder. This make checkpoint restoring easy without any additional constraints.
  - We also write a function to load the best model (`load_best_model`) after training. This is useful to load the model several time (evaluation, separation of new examples. . .).

- The arguments flow through bash/python/yaml in a specific way, which was designed by us and suits our use- cases until now:

- The very first step is the `local/conf.yml` files where is a hierarchical configuration file.
- **On the python side**: This file is parsed as a dictionary of dictionary in `training.py` From this dict, we create an argument parser which can accept all the second-level keys from the dictionary (so second-level keys should be unique) as arguments and has the default values from the `conf.yml` file.
- **On the bash side**: we also parse arguments from the command line (using `utils/parse_options.sh`). The arguments above the line. `utils/parse_options.sh` can be parsed, the rest are fixed. Most arguments will be passed to the training script. Others control the data preparation, GPU usage etc.
- In light of all this the config file should have sensible default values that shouldn't be modified by hand much. The quickly configurable part of the recipe is added to `run.sh` (you want to experiment with the batch size, add an argument in and pass it to python. If you want it fixed, no need to put it in bash, the conf.yml file keeps it for you.) This makes it possible to directly identify the important parts of the experiment, without reading lots of lines of argparser or bash arguments.

C. Python codes :

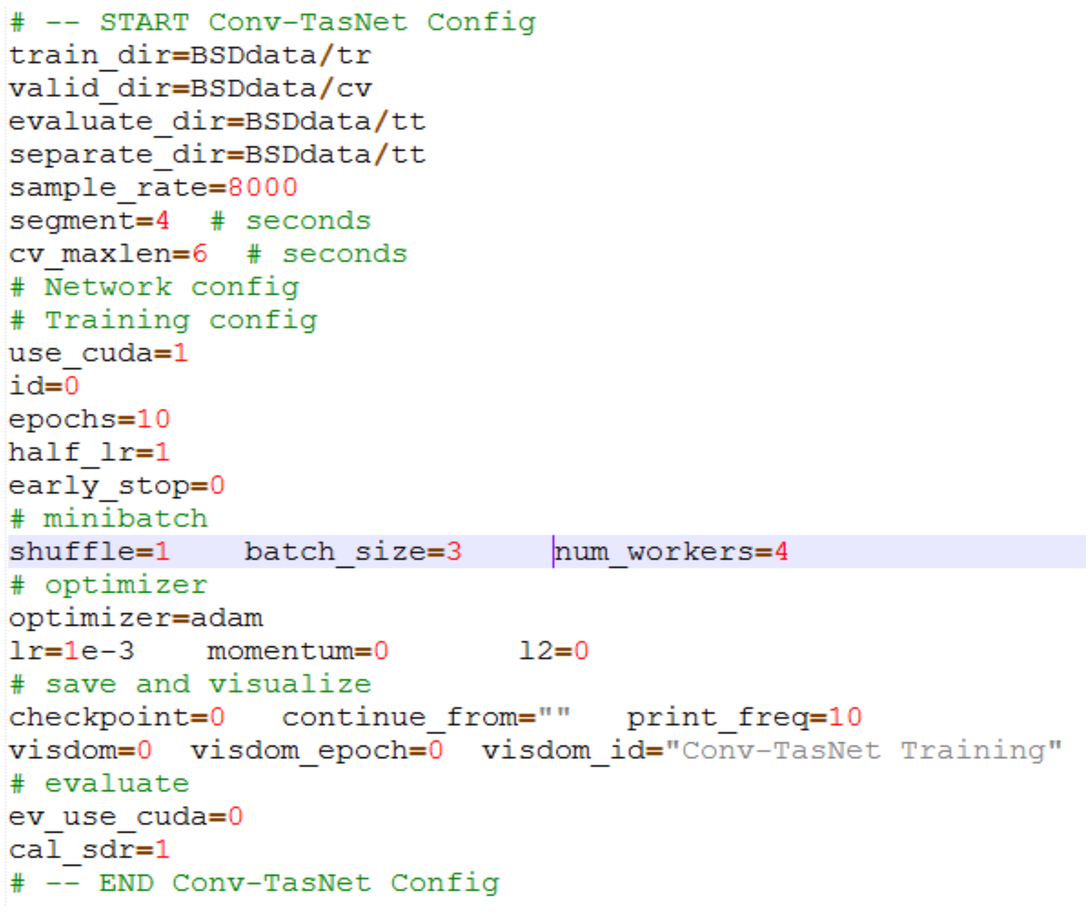
Encoder/Decoder :

```python
class Encoder(nn.Module):
    """Estimation of the nonnegative mixture weight by a 1-D conv layer. """
    def __init__(self, L, N):
        super(Encoder, self).__init__()
        # Hyper-parameter
        self.L, self.N = L, N
        # Components
        # 50% overlap
        self.conv1d_U = nn.Conv1d(1, N, kernel_size=L, stride=L // 2, bias=False)

    def forward(self, mixture):
        """
        Args:     mixture: [M, T], M is batch size, T is #samples
        Returns:    mixture_w: [M, N, K], where K = (T-L)/(L/2)+1 = 2T/L-1
        """
        mixture = torch.unsqueeze(mixture, 1)  # [M, 1, T]
        mixture_w = F.relu(self.conv1d_U(mixture))   # [M, N, K]
        return mixture_w

class Decoder(nn.Module):
    def __init__(self, N, L):
        super(Decoder, self).__init__()
        # Hyper-parameter
        self.N, self.L = N, L
        # Components
        self.basis_signals = nn.Linear(N, L, bias=False)

    def forward(self, mixture_w, est_mask):
        """
        Args:
          mixture_w: [M, N, K]
            est_mask: [M, C, N, K]
        Returns:      est_source: [M, C, T]
        """
        # D = W * M
        source_w = torch.unsqueeze(mixture_w, 1) * est_mask  # [M, C, N, K]
        source_w = torch.transpose(source_w, 2, 3) # [M, C, K, N]
        # S = DV
        est_source = self.basis_signals(source_w)  # [M, C, K, L]
        est_source = overlap_and_add(est_source, self.L//2) # M x C x T
        return est_source
```

Network architecture :

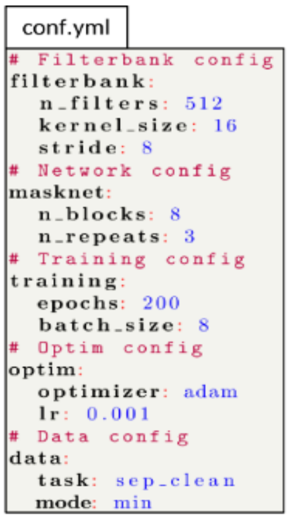
```python
# Network architecture
parser.add_argument('--N', default=512, type=int,
                    help='Number of filters in autoencoder')
parser.add_argument('--L', default=16, type=int,
                    help='Length of the filters in samples (40=5ms at 8kHZ)')
parser.add_argument('--B', default=128, type=int,
                    help='Number of channels in bottleneck 1 × 1-conv block')
parser.add_argument('--H', default=512, type=int,
                    help='Number of channels in convolutional blocks')
parser.add_argument('--P', default=3, type=int,
                    help='Kernel size in convolutional blocks')
parser.add_argument('--X', default=8, type=int,
                    help='Number of convolutional blocks in each repeat')
parser.add_argument('--R', default=3, type=int,
                    help='Number of repeats')
parser.add_argument('--C', default=2, type=int,
                    help='Number of speakers')
parser.add_argument('--norm_type', default='gLN', type=str,
                    choices=['gLN', 'cLN', 'BN'], help='Layer norm type')
parser.add_argument('--causal', type=int, default=0,
                    help='Causal (1) or noncausal(0) training')
parser.add_argument('--mask_nonlinear', default='Sigmoid', type=str,')
```

Conv-TasNet Config: (Training config, optimizer….)



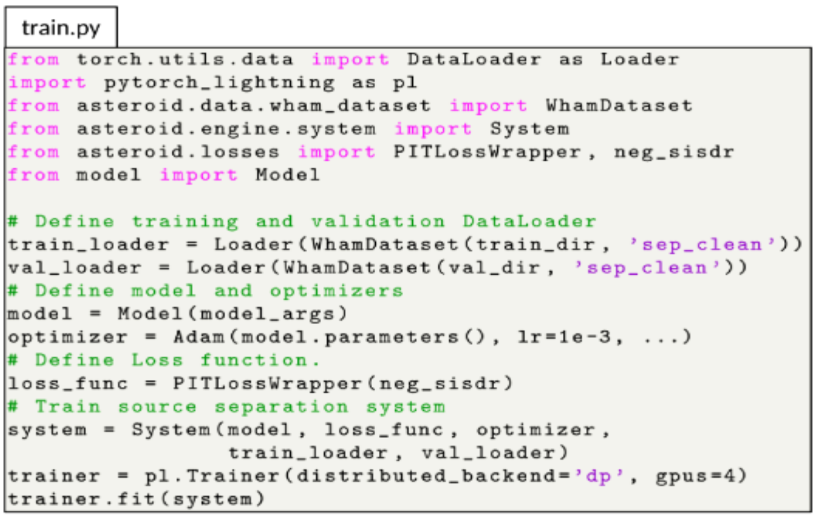
```
# -- START Conv-TasNet Config
train_dir=BSDdata/tr
valid_dir=BSDdata/cv
evaluate_dir=BSDdata/tt
separate_dir=BSDdata/tt
sample_rate=8000
segment=4  # seconds
cv_maxlen=6  # seconds
# Network config
# Training config
use_cuda=1
id=0
epochs=10
half_lr=1
early_stop=0
# minibatch
shuffle=1     batch_size=3      num_workers=4
# optimizer
optimizer=adam
lr=1e-3     momentum=0       l2=0
# save and visualize
checkpoint=0    continue_from=""    print_freq=10
visdom=0   visdom_epoch=0   visdom_id="Conv-TasNet Training"
# evaluate
ev_use_cuda=0
cal_sdr=1
# -- END Conv-TasNet Config
```

Exemple for train.py and conf.yml :



conf.yml

```
# Filterbank config
filterbank:
   n_filters: 512
   kernel_size: 16
   stride: 8
# Network config
masknet:
   n_blocks: 8
   n_repeats: 3
# Training config
training:
   epochs: 200
   batch_size: 8
# Optim config
optim:
   optimizer: adam
   lr: 0.001
# Data config
data:
   task: sep_clean
   mode: min
```



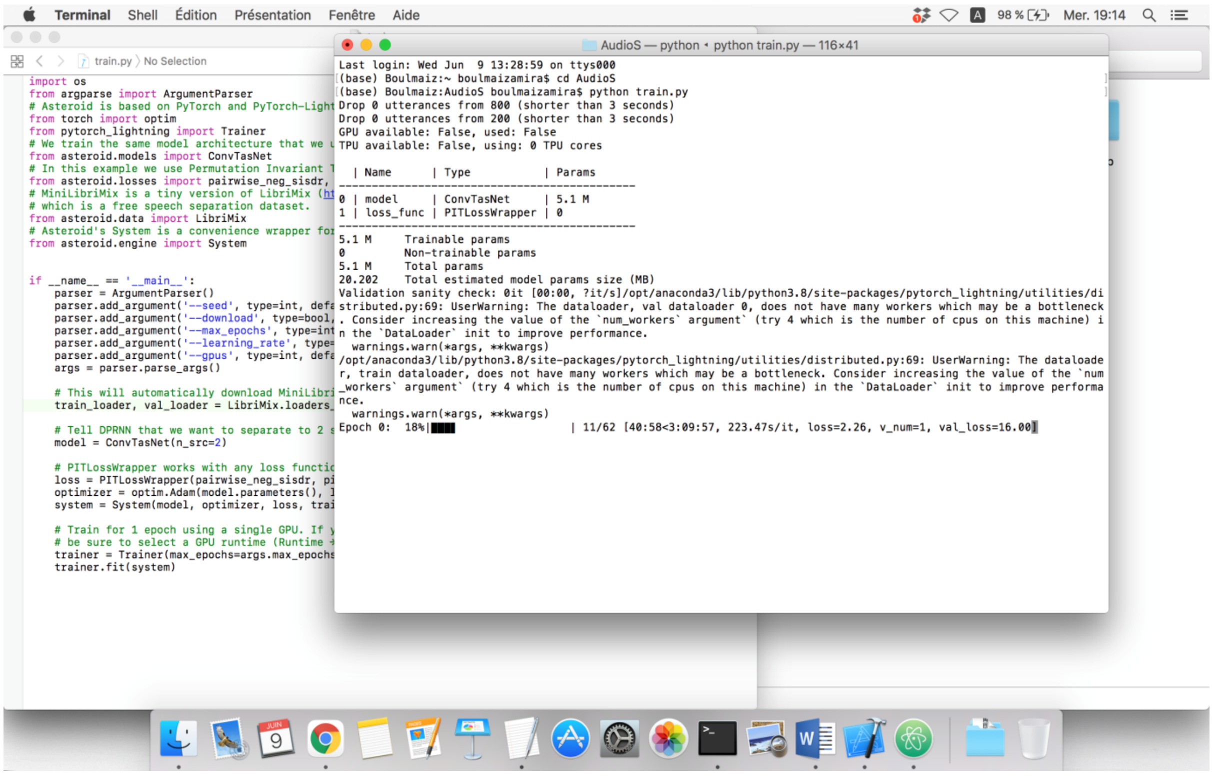
train.py

```
from torch.utils.data import DataLoader as Loader
import pytorch_lightning as pl
from asteroid.data.wham_dataset import WhamDataset
from asteroid.engine.system import System
from asteroid.losses import PITLossWrapper, neg_sisdr
from model import Model

# Define training and validation DataLoader
train_loader = Loader(WhamDataset(train_dir, 'sep_clean'))
val_loader = Loader(WhamDataset(val_dir, 'sep_clean'))
# Define model and optimizers
model = Model(model_args)
optimizer = Adam(model.parameters(), lr=1e-3, ...)
# Define Loss function.
loss_func = PITLossWrapper(neg_sisdr)
# Train source separation system
system = System(model, loss_func, optimizer,
                train_loader, val_loader)
trainer = pl.Trainer(distributed_backend='dp', gpus=4)
trainer.fit(system)
```

For more details about scripts visit:

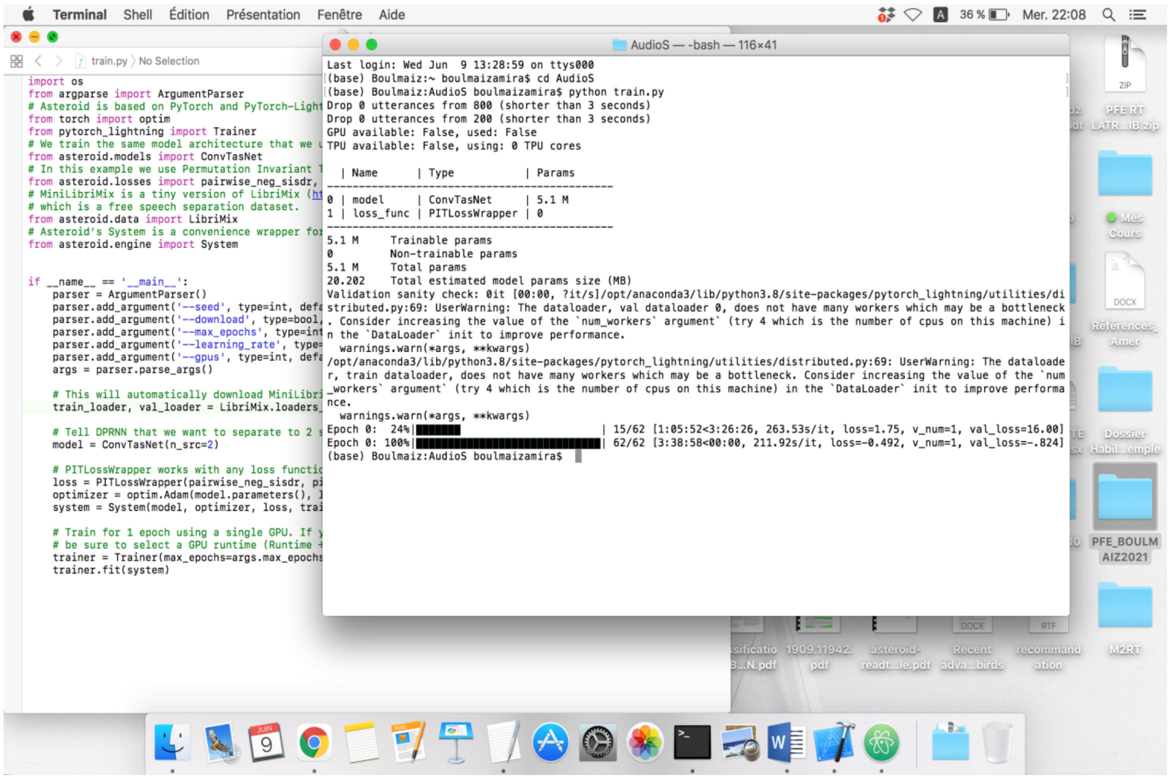
- https: //github.com/asteroid-team/asteroid
- https: //github.com/kaituoxu/Conv-TasNet

Training experiment: (training of MiniLibriMixDataset)



```
import os
from argparse import ArgumentParser
# Asteroid is based on PyTorch and PyTorch-Light
from torch import optim
from pytorch_lightning import Trainer
# We train the same model architecture that we u
from asteroid.models import ConvTasNet
# In this example we use Permutation Invariant T
from asteroid.losses import pairwise_neg_sisdr,
# MiniLibriMix is a tiny version of LibriMix (ht
# which is a free speech separation dataset.
from asteroid.data import LibriMix
# Asteroid's System is a convenience wrapper for
from asteroid.engine import System


if __name__ == '__main__':
    parser = ArgumentParser()
    parser.add_argument('--seed', type=int, defa
    parser.add_argument('--download', type=bool,
    parser.add_argument('--max_epochs', type=int
    parser.add_argument('--learning_rate', type=
    parser.add_argument('--gpus', type=int, defa
    args = parser.parse_args()

    # This will automatically download MiniLibri
    train_loader, val_loader = LibriMix.loaders_

    # Tell DPRNN that we want to separate to 2 s
    model = ConvTasNet(n_src=2)

    # PITLossWrapper works with any loss functic
    loss = PITLossWrapper(pairwise_neg_sisdr, pi
    optimizer = optim.Adam(model.parameters(), l
    system = System(model, optimizer, loss, trai

    # Train for 1 epoch using a single GPU. If y
    # be sure to select a GPU runtime (Runtime →
    trainer = Trainer(max_epochs=args.max_epochs
    trainer.fit(system)
```

```
Last login: Wed Jun  9 13:28:59 on ttys000
(base) Boulmaiz:~ boulmaizamira$ cd AudioS
(base) Boulmaiz:AudioS boulmaizamira$ python train.py
Drop 0 utterances from 800 (shorter than 3 seconds)
Drop 0 utterances from 200 (shorter than 3 seconds)
GPU available: False, used: False
TPU available: False, using: 0 TPU cores

  | Name      | Type           | Params
---------------------------------------------
0 | model     | ConvTasNet     | 5.1 M
1 | loss_func | PITLossWrapper | 0
---------------------------------------------
5.1 M     Trainable params
0         Non-trainable params
5.1 M     Total params
20.202    Total estimated model params size (MB)
Validation sanity check: 0it [00:00, ?it/s]/opt/anaconda3/lib/python3.8/site-packages/pytorch_lightning/utilities/di
stributed.py:69: UserWarning: The dataloader, val dataloader 0, does not have many workers which may be a bottleneck
. Consider increasing the value of the `num_workers` argument` (try 4 which is the number of cpus on this machine) i
n the `DataLoader` init to improve performance.
  warnings.warn(*args, **kwargs)
/opt/anaconda3/lib/python3.8/site-packages/pytorch_lightning/utilities/distributed.py:69: UserWarning: The dataloade
r, train dataloader, does not have many workers which may be a bottleneck. Consider increasing the value of the `num
_workers` argument` (try 4 which is the number of cpus on this machine) in the `DataLoader` init to improve performa
nce.
  warnings.warn(*args, **kwargs)
Epoch 0:  18%|████                 | 11/62 [40:58<3:09:57, 223.47s/it, loss=2.26, v_num=1, val_loss=16.00]
```



```
import os
from argparse import ArgumentParser
# Asteroid is based on PyTorch and PyTorch-Light
from torch import optim
from pytorch_lightning import Trainer
# We train the same model architecture that we u
from asteroid.models import ConvTasNet
# In this example we use Permutation Invariant T
from asteroid.losses import pairwise_neg_sisdr,
# MiniLibriMix is a tiny version of LibriMix (ht
# which is a free speech separation dataset.
from asteroid.data import LibriMix
# Asteroid's System is a convenience wrapper for
from asteroid.engine import System


if __name__ == '__main__':
    parser = ArgumentParser()
    parser.add_argument('--seed', type=int, defa
    parser.add_argument('--download', type=bool,
    parser.add_argument('--max_epochs', type=int
    parser.add_argument('--learning_rate', type=
    parser.add_argument('--gpus', type=int, defa
    args = parser.parse_args()

    # This will automatically download MiniLibri
    train_loader, val_loader = LibriMix.loaders_

    # Tell DPRNN that we want to separate to 2 s
    model = ConvTasNet(n_src=2)

    # PITLossWrapper works with any loss functic
    loss = PITLossWrapper(pairwise_neg_sisdr, pi
    optimizer = optim.Adam(model.parameters(), l
    system = System(model, optimizer, loss, trai

    # Train for 1 epoch using a single GPU. If y
    # be sure to select a GPU runtime (Runtime →
    trainer = Trainer(max_epochs=args.max_epochs
    trainer.fit(system)
```

```
Last login: Wed Jun  9 13:28:59 on ttys000
(base) Boulmaiz:~ boulmaizamira$ cd AudioS
(base) Boulmaiz:AudioS boulmaizamira$ python train.py
Drop 0 utterances from 800 (shorter than 3 seconds)
Drop 0 utterances from 200 (shorter than 3 seconds)
GPU available: False, used: False
TPU available: False, using: 0 TPU cores

  | Name      | Type           | Params
---------------------------------------------
0 | model     | ConvTasNet     | 5.1 M
1 | loss_func | PITLossWrapper | 0
---------------------------------------------
5.1 M     Trainable params
0         Non-trainable params
5.1 M     Total params
20.202    Total estimated model params size (MB)
Validation sanity check: 0it [00:00, ?it/s]/opt/anaconda3/lib/python3.8/site-packages/pytorch_lightning/utilities/di
stributed.py:69: UserWarning: The dataloader, val dataloader 0, does not have many workers which may be a bottleneck
. Consider increasing the value of the `num_workers` argument` (try 4 which is the number of cpus on this machine) i
n the `DataLoader` init to improve performance.
  warnings.warn(*args, **kwargs)
/opt/anaconda3/lib/python3.8/site-packages/pytorch_lightning/utilities/distributed.py:69: UserWarning: The dataloade
r, train dataloader, does not have many workers which may be a bottleneck. Consider increasing the value of the `num
_workers` argument` (try 4 which is the number of cpus on this machine) in the `DataLoader` init to improve performa
nce.
  warnings.warn(*args, **kwargs)
Epoch 0:  24%|███████                       | 15/62 [1:05:52<3:26:26, 263.53s/it, loss=1.75, v_num=1, val_loss=16.00]
Epoch 0: 100%|██████████████████████████████| 62/62 [3:38:58<00:00, 211.92s/it, loss=-0.492, v_num=1, val_loss=-.824]
(base) Boulmaiz:AudioS boulmaizamira$ 
```

BIBLIOGRAPHY

[1] Kolbæk, M., Yu, D., Tan, Z. H., & Jensen, J. (2017). Multitalker speech separation with utterance-level permutation invariant training of deep recurrent neural networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *25*(10), 1901-1913.

[2] Luo, Y., & Mesgarani, N. (2019). Conv-tasnet: Surpassing ideal time–frequency magnitude masking for speech separation. *IEEE/ACM transactions on audio, speech, and language processing*, *27*(8), 1256-1266.

[3] Stoller, D., Ewert, S., & Dixon, S. (2018). Wave-u-net: A multi-scale neural network for end-to-end audio source separation. *arXiv preprint arXiv:1806.03185*.

[4] Cherry, E. C. (1953). Some experiments on the recognition of speech, with one and with two ears. *The Journal of the acoustical society of America*, *25*(5), 975-979.

[5] Comon, P., & Jutten, C. (Eds.). (2010). *Handbook of Blind Source Separation: Independent component analysis and applications*. Academic press.

[6] Hyvärinen, A., & Oja, E. (2000). Independent component analysis: algorithms and applications. *Neural networks*, *13*(4-5), 411-430.

[7] Wang, D., & Brown, G. J. (2006). *Computational auditory scene analysis: Principles, algorithms, and applications*. Wiley-IEEE press.

[8] Plumbley, M. D., Cichocki, A., & Bro, R. (2010). Non-negative mixtures. In *Handbook of Blind Source Separation* (pp. 515-547). Academic Press.

[9] Jutten, C., & Herault, J. (1991). Blind separation of sources, part I: An adaptive algorithm based on neuromimetic architecture. *Signal processing*, *24*(1), 1-10.

[10] Cichocki, A., Bogner, R. E., Moszczyński, L., & Pope, K. (1997). Modified Herault–Jutten algorithms for blind separation of sources. *Digital signal processing*, *7*(2), 80-93.

[11] Prasad, R., Saruwatari, H., & Shikano, K. (2004). An ICA algorithm for separation of convolutive mixture of speech signals. *International Journal of Information Technology*, *2*(4), 273-283.

[12] Pedersen, M. S., Wang, D., Larsen, J., & Kjems, U. (2008). Two-microphone separation of speech mixtures. *IEEE Transactions on Neural Networks*, *19*(3), 475-492.

[13] Jan, T., Wang, W., & Wang, D. (2011). A multistage approach to blind separation of convolutive speech mixtures. *Speech Communication*, *53*(4), 524-539.

[14] Kim, M., & Park, H. M. (2015). Efficient online target speech extraction using DOA-constrained independent component analysis of stereo data for robust speech recognition. *Signal Processing*, *117*, 126-137.

[15] Douglas, S. C., Gupta, M., Sawada, H., & Makino, S. (2007). Spatio–Temporal FastICA algorithms for the blind separation of convolutive mixtures. *IEEE transactions on audio, speech, and language processing*, *15*(5), 1511-1520.

[16] Koldovský, Z., & Tichavský, P. (2007). Time-domain blind audio source separation using advanced ICA methods. In *Eighth Annual Conference of the International Speech Communication Association*.

[17] Hsieh, H. L., Chien, J. T., Shinoda, K., & Furui, S. (2009, April). Independent component analysis for noisy speech recognition. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing* (pp. 4369-4372). IEEE.

[18] Mohanaprasad, K., & Arulmozhivarman, P. (2013). Comparison of Fast ICA and gradient algorithms of independent component analysis for separation of speech signals. *Int. J. Eng. Technol*, *5*(4), 3196-3202.

[19] Weng, C., Yu, D., Seltzer, M. L., & Droppo, J. (2015). Deep neural networks for single-channel multi-talker speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *23*(10), 1670-1679.

[20] Hershey, J. R., Chen, Z., Le Roux, J., & Watanabe, S. (2016, March). Deep clustering: Discriminative embeddings for segmentation and separation. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 31-35). IEEE.

[21] Luo, Y., & Mesgarani, N. (2018, April). Tasnet: time-domain audio separation network for real-time, single-channel speech separation. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 696-700). IEEE.

[22] Rozenwald, M. B., Galitsyna, A. A., Sapunov, G. V., Khrameeva, E. E., & Gelfand, M. S. (2020). A machine learning framework for the prediction of

chromatin folding in Drosophila using epigenetic features. *PeerJ Computer Science*, *6*, e307.

[23] Potok, T. E., Schuman, C., Young, S., Patton, R., Spedalieri, F., Liu, J., ... & Chakma, G. (2018). A study of complex deep learning networks on high-performance, neuromorphic, and quantum computers. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, *14*(2), 1-21.

[24] Koppe, G., Meyer-Lindenberg, A., & Durstewitz, D. (2021). Deep learning for small and big data in psychiatry. *Neuropsychopharmacology*, *46*(1), 176-190.

[25] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[26] Saeed, M. M., Al Aghbari, Z., & Alsharidah, M. (2020). Big data clustering techniques based on Spark: a literature review. *PeerJ Computer Science*, *6*, e321.

[27] Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., & Potts, C. (2013, October). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing* (pp. 1631-1642).

[28] Batur Dinler, Ö., & Aydin, N. (2020). An optimal feature parameter set based on gated recurrent unit recurrent neural networks for speech segment detection. *Applied Sciences*, *10*(4), 1273.

[29] Sadr, H., Pedram, M. M., & Teshnehlab, M. (2019). A robust sentiment analysis method based on sequential combination of convolutional and recursive neural networks. *Neural Processing Letters*, *50*(3), 2745-2761.

[30] Gao, C., Yan, J., Zhou, S., Varshney, P. K., & Liu, H. (2019). Long short-term memory-based deep recurrent neural networks for target tracking. *Information Sciences*, *502*, 279-296.

[31] Zhou, D. X. (2020). Universality of deep convolutional neural networks. *Applied and computational harmonic analysis*, *48*(2), 787-794.

[32] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., ... & Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern Recognition*, *77*, 354-377.

[33] Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning* (Vol. 1, No. 2). Cambridge: MIT press.

[34] Kiranyaz, S., Ince, T., Hamila, R., & Gabbouj, M. (2015, August). Convolutional neural networks for patient-specific ECG classification. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)* (pp. 2608-2611). IEEE.

[35] Bai, S., Kolter, J. Z., & Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271.*

[36] Oord, A. V. D., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... & Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499.*

[37] Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1251-1258).

[38] Kaiser, L., Gomez, A. N., & Chollet, F. (2017). Depthwise separable convolutions for neural machine translation. *arXiv preprint arXiv:1706.03059.*

[39] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision* (pp. 1026-1034).

[40] Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450.*

[41] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

[42] Sokal, R. R. (1958). A statistical method for evaluating systematic relationships. *Univ. Kansas, Sci. Bull.*, *38*, 1409-1438.

[43] Luo, Y., & Mesgarani, N. (2018, September). Real-time single-channel dereverberation and separation with time-domain audio separation network. In *Interspeech* (pp. 342-346).