ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Informatica

Speech Analysis for Automatic Identification of Mild Cognitive Impairment through Autoencoders

Relatore:
Chiar.mo Prof.
Danilo Montesi

Correlatore:
Ph.D.
Flavio Bertini

Presentata da:
Davide Allevi

Sessione II
Anno Accademico 2018-19

Table of contents

List of figures

Introduction

Nowadays, cognitive decline is unfortunately a non-curable disease which, due to an alteration in brain function, causes the progressive decline of memory, thought and reasoning abilities, so much so that in its most severe state patients reach the complete loss of autonomy.
People suffering from cognitive decline often have difficulty completing daily activities. Sometimes, they may have problems driving the car to a familiar place, managing a budget at work or remembering the rules of a favorite game, they may also have difficulty following or participating in a conversation. These individuals can stop in the middle of a conversation and have no idea how to continue, or they may happen again. They could struggle with the vocabulary, have trouble finding the right word or call things by the wrong name.
Before manifesting itself so clearly, however, cognitive decline goes through a phase that can last several years, during which, although the symptoms are minimal, the disease is at work to determine the decisive brain damage that will lead to the onset of that 'set of disorders that goes by the name of "dementia".
Being able to identify the first signs of cognitive decline in a "pre-symptomatic" phase certainly becomes fundamental in trying to respond significantly to the disease. To succeed in this, the researchers focused on one of the most evolved abilities of the human mind: *the language*.
One of the fundamental reasons for which the language was chosen is surely not to be invasive on people. In particular, some researchers have developed a study that involved 96 patients, some of whom showed symptoms of Mild Cognitive Impairment (MCI), a condition that can occur before Alzheimer's disease. During this experiment each patient is asked to describe an image, a working day and the last dream that was made.
Once the answers were collected, they were analyzed using specific automatic language processing techniques, able to examine the rhythm and sound of words, the use of vocabulary and syntax and other details of linguistic productions . Finally, by checking the results of patients suffering from the disease and those without disorders, they tried to find signs of cognitive impairment that conventional neuropsychological tests are unable to identify.
In this thesis we proposed a method to classify audio files to detect subjects suffering from cognitive decline. In particular we used the *Autoencoder* method that is a type of artificial

neural network used to learn efficient data codings in an unsupervised manner. The specific use of the autoencoder is to use a feedforward approach to reconstitute an output from an input. In our case we have used the software *auDeep* that is software for unsupervised feature learning with deep neural networks (DNNs), which trains an autoencoder for the extraction of features from spectrograms and their classification.
In addition, the *SpecAugment* method was used to increase the number of data to be analyzed. This method involves cutting some frequency and time bands into the spectrograms and using them as new data.
The thesis is structured as follows:

Chapter 1
In the first chapter the state of the art is described making a report on the main techniques to detect subjects suffering from cognitive decline through audio files.

Chapter 2
In the second chapter the notion of "Autoencoding" is introduced, then going on to explain more in detail what an Autoencoder is and how many types of autoencoders can be found.

Chapter 3
In the third chapter a general explanation is given about what the *data augmentation* is and in which fields it can be used. First of all we explain it when it is applied on the images for which it is the most used then we go on to explain in brief how it behaves when it is used for texts. Finally we will make a specific explanation on the data augmentation concerning the audio files, more precisely on the SpecAugment which is precisely the method we will use in our experiment.

Chapter 4
In the fourth chapter we move on to the description of our approach. In particular we are going to explain in detail the dataset as it was built and the work environment that we will use in our experiment. Next we explain in detail the experiment carried out both on the 3 classes (*Control*, *Mild Cognitive Impairment*, *early Dementia*) and on the 2 classes (*Control*, *Decline*). Finally a general summary is made comparing the results obtained by us and those that were obtained in the state of the art.

Conclusions
At the end of the paper, the final conclusions drawn from the various analyzes will be presented.

Chapter 1

State of the art

In the state of the art we started from a study based on the analysis of spontaneous language or rather not with conventional tests, using technologies that also refer to artificial intelligence and big data analysis. The main idea is precisely that of capturing, through spontaneous language, some early signs. To identify these signs it is necessary to use technologies that are able to identify, for example, the spaces between a word and another or the intonation that is given to a sentence, but the things that are altered first seem to be those that transmit emphasis to a speech or when emotions are activated on our speech.
In particular, the language has been chosen among all the faculties of man, because it is a very complex task of our brain, in fact it is known that in the advanced stages of cognitive decline there is a strong decay of this faculty and therefore is the main candidate to find very early markers of the disease before the signs that bring to the diagnosis appear classically.
The goal of this study is certainly to try to prevent Alzheimer's disease in advance so that we can avoid starting treatment too late, of course it will take decades before the symptoms of the disease appear but most likely through early diagnosis it will be possible to slow down the disease, as has been possible in other diseases.
Therefore, as mentioned above, recent studies have suggested that language alterations may be one of the first signs of cognitive decline, often found years before other cognitive deficits occurred. The main goal of the speech analysis is to try to identify features that can be useful for studying the domain that interests us.
These features, as can be seen from the figure below (Fig. 1.1), can be extracted in two ways either *manually* or *automatically*, and subsequently analyzed using known machine learning techniques.

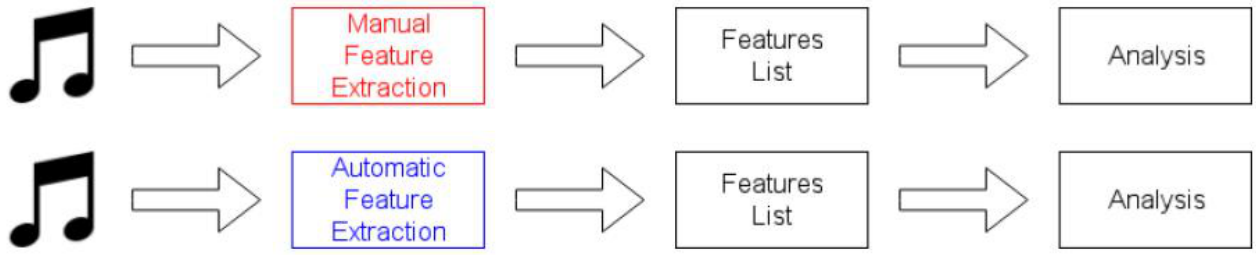

Figure 1.1: Extraction features

1.1 Manual feature extraction

The extraction of the features in manual way are the most frequent and that initially were carried out more frequently in the first experiments. The type of features to be extracted can be different and can be done in different ways. For example in the paper C. Fraser et al. [3] a method with manual extraction is proposed in particular the data have been taken from the DementiaBank corpus, from which 167 patients diagnosed with "possible" or "probable" Alzheimer's disease (AD) provide 240 narrative samples, and 97 controls provide an additional 233. The two groups are not matched for age and education, which is one limitation of these data. Narrative speech was collected using the "Cookie Theft" picture description task from the Boston Diagnostic Aphasia Examination [1]. This protocol instructs the examiner to show the picture to the patient and say, "*Tell me everything you see going on in this picture*".
All speech sample was recorded then manually transcribed at the word level following the TalkBank CHAT [2] protocol. Narratives were segmented into utterances and annotated with filled pauses, paraphasias, and unintelligible words. From the CHAT transcripts have been kept the wordlevel transcription and the utterance segmentation and have been discarded the morphological analysis, dysfluency annotations, and other associated information. Before tagging and parsing the transcripts have been removed automatically short false starts consisting of two letters or fewer and filled pauses (i.e *uh, um, er, ah*). After which these variables are used to train a machine learning classifier to distinguish between participants with AD and healthy controls. The result obtained was an accuracy of over 81% in distinguishing individuals with AD from those without based on short samples of their language on a picture description task. While in Gosztolya.G et al. [1] unlike C. Fraser et al. [3] it carries out the experiment on three classes instead of two. More in detail uses a database as recorded at the Memory Clinic at the Department of Psychiatry of the University of Szeged, Hungary. They have been

[1] https://en.wikipedia.org/wiki/Boston_Diagnostic_Aphasia_Examination
[2] https://talkbank.org/

collected utterances from three categories of subjects: those suffering from MCI, those affected by early-stage AD, and those having no cognitive impairment at the time of recording (control group). Some have not been taken into consideration such as those who have suffered head injuries, depression or psychosis. From some previous studies it has been found that MCI and AD affect the spontaneous speech of the patients more than their planned speech. This is because in the case of planned speech, speakers usually have some time in advance to think about what they would like to say, hence difficulties in word finding (due to memory decline) cannot be reliably detected while in the case of spontaneous speech, speakers are required to speak on the spot, which might truly reflect their difficulties in word finding. for this reason in this paper they have been recorded spontaneous speech.
After the presentation of a specially designed one-minute-long animated film, the subjects were asked to talk about the events seen on the film (*immediate recall*). After the presentation of a second film, the subjects were asked to talk about their previous day (*previous day*). While in the last task the subjects were asked to talk about the second film (*delayed recall*). They have been used the recordings of 25 speakers for each speaker group, resulting in a total of 75 speakers and 225 recordings. Being a very small dataset it is not possible to create a training and a test set, therefore it has been opted for a cross validation (CV) with 5 folds. For this experiment, eight acoustic (speech tempo, length of utterance, duration of silent and filled pauses (hesitation), number of silent and filled pauses and hesitation rate) features were manually extracted and using these features, they performed a new experiment combining MCI and mAD classes. Therefore in this case the class distribution becomes imbalanced with 25 control subjects and 50 subjects having some kind of cognitive disorder.

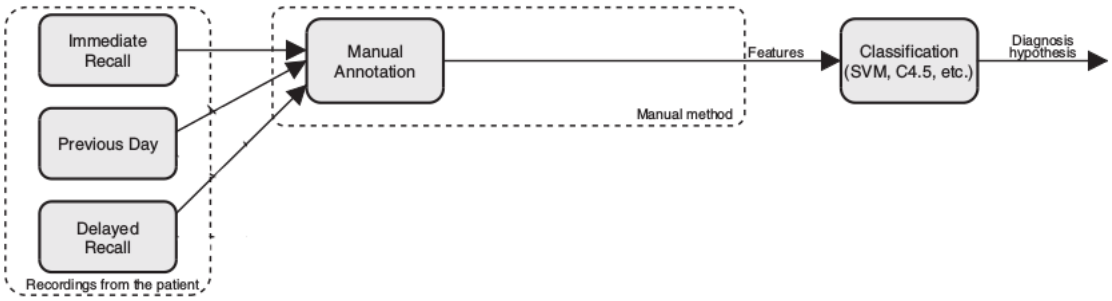

Figure 1.2: Manual extraction in Gosztolya.G et al. [1]

After extracting the features, the respective classes were classified with the following accuracy results:

		3-class	2-class					
Feature extraction	Feature set	Accuracy	Accuracy	UAR	Precision	Recall	Specificity	F_1
Manual	Basic	61.3%	76.0%	74.0%	83.3%	80.0%	63.0%	81.6

Figure 1.3: Results in Gosztolya.G et al. [1]

So in this paper they manage to reach an accuracy of 61.3% for the three classes and 76% for two classes.
But the experiment on which we based our most was Calzà et al. [4]. Starting from this approach we have tried a solution to solve the problems that can be found with an extraction of the features done manually trying to propose an extraction done automatically. For this reason the following paper will be explained in more detail than the others.

Background paper Calzà et al. [4]

This project has the goal to propose actions and methods to prevent fragility and decline and to promote the health of the elderly, developing and elaborating tools and networks for early diagnosis.
In recent years, thanks to the growth of new sophisticated techniques in natural language processing (NLP) it has been possible to use them to analyze written texts, clinically elicited expressions and spontaneous production, with the aim of identifying signs of psychiatric or neurological disorders and extract linguistics derived automatically for the recognition, classification and description of pathologies. This research is part of the OPLON project ("OPportunities for active and healthy LONgevity"). Inside the framerwork OPLON, they have worked to produce methods to identify cognitive fragility at an early stage through the development of spontaneous languages of Italian speakers [4].
Initially they left without a dedicated study so the main objective was to first verify that this approach was feasible.

Data collection

In this project 96 subjects (48 males, 48 females) have been enrolled, between the ages of 50 and 75 and with at least a junior high school certificate or primary school certificate with high intellectual interests throughout the life span. The sample was composed of 48 healthy controls (CON) e 48 subjects with cognitive decline (DEC) [4] [5].

The cognitive decline refers to two categories:

1. *Mild Cognitive Impairment* **MCI**: it causes cognitive changes that are serious enough to be assessed with neuropsychological assessment, but do not affect the individual's ability to carry out everyday activities. Experts classify mild cognitive impairment based on the thinking skills affected:

    a. *amnestic MCI single domain* (a-MCI; 16 subjects: 8 females, 8 males): MCI that primarily affects memory. A person may start to forget important information that he or she would previousl have recalled easily, such as appointments, conversations or recent events.

    b. *mutiple domain MCI* (md-MCI; 16 subjects: 8 females, 8 males): MCI that affects thinking skills other than memory, including the ability to make sound decisions, judge the time or sequence of steps needed to complete a complex task, or visual perception.

2. *Early Dementia* **e-D** (16 subjects: 8 females, 8 males): these patients are affected by cognitive deficits which partially influence everyday life

All the participants were requested to complete the anamnestic interview (anagraphic data; occupation/retirement; children; familiarity with neurodegenerative pathologies; clinical history and pharmacotherapy), a neurological assessment and the traditional cognitive battery aimed to evaluate several cognitive domains: logic, memory, attention, language, visuo-spatial, praxic, and executive functions [6].
After the traditional neuropsychological assessment, to all partecipants were required to record their spontaneous speech during the execution of three tasks, elicited by these input sentences:

- "*Describe this picture*"
- "*Describe your typical working day*"
- "*Describe the last dream you remember*"

Data analysis

Spontaneous speech samples were recorded in un quiet room with an Olympus (Linear PCM Recorder LS-5) in WAV files (44.1 KHZ, 16 bit) placed on a table in front of the subject [4]. Samples were collected during test sessions in the form of audio and were manually transcribed in order to produce the dialogue orthographic transcription through the use of the *Transcriber* software software package. Then choose the "utterance" as the reference unit in the speech

continuum. Utterances are demarcated by prosody in the speech flow so thanks to "prosodic interruptions" we can arrive at the identification of their boundaries. The manual detection of these interruptions was achieved by listening the utterances and splitting the entire dialogues into turns. In addition, during the transcription process have been annotated a series of paralinguistic phenomena such as empty and filled pauses (e.g. "mmmh", "eeh", "ehm"), disfluencies (e.g. hesitation, stuttering, false star, lapsus) and non-verbal phenomena (e.g. inspirations, laughs, coughing fits, throat clearing).
Guidelines were also transcribed in order to force transcribers to respect the developed transcription protocol, and this ensured consistency among them during the project.
Finally all the speech fragments in which the interviewer's speech was present were removed and all expressions that were selected were subjected to automatic morphological and syntactic annotation.
More precisely, all the expressions were automatically PoS-tagged and syntactically parsed with the dependency model used by the Turin University Linguistic Environment - TULE, based on the TUT - Turin University TreeBank tagset in order to explicit all the morphological, syntactic and lexical information. Subsequently all the automatically inserted annotations are checked manually to eliminate all the errors introduced by the automatic tagging procedures.
While for the parameters that come from the discourse acoustic it has been used the "ssvad" Voice Activity Detector especially developed for interview speech, to segment the recordings and identify speech vs. non-speech regions and the forced alignment system belonging to the Kaldi-DNN-ASR package, trained on the APASCI Italian Corpus, for obtaining the temporally aligned phonetic transcriptions needed to compute various rhythmic and acoustic features.
A multidimensional parameter computation is made and the system performs a quantitative analysis of the spoken texts, defining rhythmic, acoustic, lexical, morpho-syntactic and syntactic features.

Experiment

As an evaluation the p-value < 0.05 of the features is determined by using Kolmogorov-Smirnov nonparametric test. They have decided to use such kind of hypothesis testing technique, compared with the T-test or the Wilcoxon-MannWhitney test, because the size of the corpus that has been used is not large enough [4].
For each linguistic task they performed, the features are used as input data for three automatic classifiers available in the Orange Data Mining tool. The classifiers used are:

- *kNN 3-neighbourgs*
- *Logistic Regression*

- *Neural Network classifiers*

The training/test sets has been automatically built by the package by random sampling the entire dataset with a ratio of 80/20% respectively between training and test sets, repeating this procedure 20 times.
For each classifier as a result they recorded the accuracy and the recall obtaining the following results:

- kNN 3-neighbourgs: Accuracy=72%, Recall=70%
- Logistic Regression: Accuracy=75%, Recall=76%
- Neural Network classifiers: Accuracy=76%, Recall=75%

1.2 Automatic feature extraction

In addition to studying cases of manual feature extractions we also looked for some approaches made with automatic extraction. For example, taking the paper Gosztolya.G et al. [1] experiment again, an automatic extraction is done in addition to the manual extraction. As calculating the above acoustic markers manually is quite expensive and requires skilled labor, have thought of using this step. One way of automating it is to use signal processing techniques but this techniques cannot extract all the features and they cannot distinguish filled pauses from speech. Therefore in this experiment automatic speech recognition (ASR) was used to extract the initial features. In particular it was introduced an approach to automatically obtain the time-aligned phoneme sequence of the utterances, serving as the basis of speech marker extraction. Hence it was possible to extend the feature set using other features that can be calculated via the phone labels because now it becomes much cheaper than before.

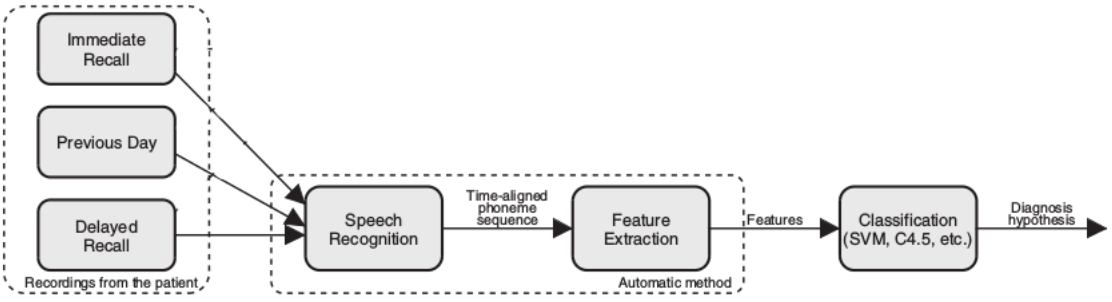

Figure 1.4: Automatic extraction in Gosztolya.G et al. [1]

Below are the results obtained with automatic extraction both with the basic and extended feature set:

		3-class	2-class					
Feature extraction	Feature set	Accuracy	Accuracy	UAR	Precision	Recall	Specificity	F_1
Automatic	Basic	50.7%	64.0%	60.0%	73.5%	72.0%	46.2%	72.7
	Extended	58.7%	73.3%	74.0%	85.7%	72.0%	57.6%	78.3

Figure 1.5: Result automatic extraction in Gosztolya.G et al. [1]

Calculating the feature set automatically is lower than the extraction done manually reaching an accuracy of 50.7% for the three classes and 64% for the two classes. Even with the extended feature set the result improves but remains always lower than the manual in this case reaching an accuracy of 58.7% for thr three classes and 73.3% for the two classes.
Instead in the study Themistocleous.C et al. [2] it has been tried an automated machine learning method, using Deep Neural Network Architectures, that aims to identify MCI. Participants for this study were recruited from the Gothenburg MCI study, and the recordings were conducted in an isolated environment at the University of Gothenburg. A total of 55 subjects were chosen, of which 30 were controls and 25 MCI, between 55 and 79 years old. Each vowel has been segmented in the acoustic signal and vowel formants were measured at multiple positions. In addition to the acoustic features, the model include as predictors information about participants' age and gender. Overall, the classification tasks include the following 24 acoustic and sociophonetic predictors:

1. **Vowel Formants**: measure the first five formant frequencies of vowels
2. **Fundamental frequency (F0)**: measure the F0 across the duration of the vowel and calculated the mean F0, min F0, and max F0.
3. **Vowel duration**: Vowel duration measured in seconds from vowel onset to vowel offset.
4. **Gender**: Participants' gender.
5. **Age**:Participants' age.

During the training phase, the neural network learns the acoustic properties that characterize MCI and HC (healthy controls).
In a "5-fold group cross-validation," the data are randomized and split into five different folds

and the network is trained five times. The data has been split into two parts. The first part consists of the 90% of the data and functions as a training corpus whereas the second part, the remaining 10% functions as an evaluation set. The results are the following:

Model	Accuracy	Precision	Recall	F1 score
M1	67	86	56	63
M2	68	92	56	66
M3	67	100	49	65
M4	68	63	62	62
M5	71	73	71	71
M6	68	73	72	72
M7	75	100	49	65
M8	65	100	49	65
M9	69	100	49	65
M10	66	95	51	64

Figure 1.6: Results in Themistocleous.C [2]

This figure indicates a comparison of the accuracy scores on the training set. The highest accuracy was 75%.

Chapter 2

Autoencoder review

"Autoencoding" is a data compression algorithm where the compression and decompression functions are:

- data-specific
- lossy
- learned automatically from examples rather than engineered by a human

Furthermore, wherever and in any context the name "Autoencoder" is used, the compression and decompression functions are implemented with neural networks [7].
Autoencoders are:

1. *data-specific* which means that they are able to compress only data that is similar to that on which they are trained. Since they learn features specific for the given training data, they are different than a standard data compression algorithm.

2. *lossy* which means that the decompressed output is degraded with respect to the initial input.

3. *learned automatically from data examples* which means that it is easy to train specialized instances of the algorithm that will perform well on a specific type of input. Autoencoders are considered an unsupervised learning technique since they don't need explicit labels to train on, but are self-supervised because they generate their own labels from the training data.

2.1 Definition Autoencoder

An *Autoencoder* is an unsupervised learning technique for neural networks that learns efficient data representations by training the network to ignore signal “noise.”
More precisely, autoencoders work to compress an input into a different representation called **latent-space representation**, trying to reconstruct the output from this representation as close as possible to its original input [8].
This kind of network is composed of two parts :

1. **Encoder**: This is the part of the network that compresses the input into a latent-space representation. It can be represented by an encoding function *h=f(x)*.

2. **Decoder**: This part aims to reconstruct the input from the latent space representation. It can be represented by a decoding function *r=g(h)*.

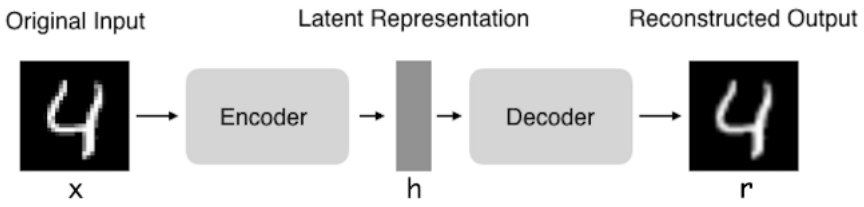

Figure 2.1: Architecture of an Autoencoder

The autoencoder as a whole can thus be described by the function $g(f(x)) = r$ where you want r as close as the original input x.
So to build a autoencoder must take into consider three things: an encoding function, a decoding function, and a distance function between the amount of information loss between the compressed representation of data and the decompressed representation (i.e. a "loss" function). Obviously the parameters of the encoding and decoding functions (in general neural networks), with respect to the distance function so that they can be optimized to minimize the reconstruction loss, using the Stochastic Gradient Descent [1]. The latter is a method to find the optimal parameter configuration for a machine learning algorithm. It iteratively makes small adjustments to a machine learning network configuration to decrease the error of the network. This loss describes the objective that the autoencoder tries to reach. When the goal is reconstruct the input as accurately as possible, there are two main loss functions that are usually taken into consideration:

[1] https://en.wikipedia.org/wiki/Stochastic_gradient_descent

- **Mean Squared Error (MSE)**
- **Kullback-Leibler (KL) divergence**

2.1.1 Mean Squared Error (MSE)

This metric is probably the simplest and most used metric for regression evaluation.
In statistics, the *mean squared error (MSE)* [2] of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errors that is, the average squared difference between the estimated values and what is estimated.
If a vector of N predictions generated from a sample of N data points on all variables, and Y is the vector of observed values of the variable being predicted, with $\bar{Y}_{\mathrm{i}}$ being the predicted values, then the within-sample MSE of the predictor is computed as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \bar{Y}_i)^2 \tag{2.1}$$

In the case of autoencoder the mean squared error is defined as the mean of the squared difference between our network output and the ground truth. For example the encoder output is a grid of values (i.e an image) the MSE between output image $\bar{I}$ and ground truth image I may be defined as [9]:

$$MSE = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}(\bar{I}_{ij} - I_{ij})^2 \tag{2.2}$$

2.1.2 Kullback-Leibler (KL) divergence

In mathematical statistics, the *Kullback–Leibler divergence* [3] is a measure of how one probability distribution is different from a second, reference probability distribution.
It is often conceptualized as measuring some sort of distance between these distributions because first of all the KL divergence is non-negative and secondly because it measures the difference between two distributions. Surely one of the fundamental properties of this metric is that it is non-negative. Indeed the KL divergence is equal to 0 if and only if pp and q are the same distribution in the case of discrete variables, or equal almost everywhere in the case of continuous variables.
Therefore for discrete probability distributions P and Q defined on the same probability space,

[2] https://en.wikipedia.org/wiki/Mean_squared_error
[3] https://en.wikipedia.org/wiki/Kullback-Leibler_divergence

the Kullback–Leibler divergence of Q from P is defined to be [9]:

$$D_{KL}(P\|Q) = -\sum_{x\in X} P(x) log\left(\frac{Q(x)}{P(x)}\right) \tag{2.3}$$

In Machine Learning succeed in minimizing the KL divergence means to make the autoencoder sample its output from a distribution that is similar to the distribution of the input which is just one of the main properties of the autoencoder to satisfy.

2.2 The use of autoencoders

The main objective of an autoencoder is to minimize the reconstruction error between input and output. If this happens then the autoencoder is helped to incorporate important features found in the data. So if a good reconstruction of the input is made then it means that most of the information present in the input has been preserved.
Nowadays *data denoising* and the *reduction of dimensionality* for data visualization are seen as two main interesting practical applications of autoencoders. Autoencoders are learned automatically from data examples, this means that it is relatively easy to train specialized instances of the algorithm that work well on any specific type of input and that it require only the appropriate training data.
However, autoencoders will perform a poor job for image compression even if as the autoencoder is trained on a given set of data, it will achieve better but always poor compression results. In fact the autoencoders are trained to make the new representation have various interesting properties. Different types of autoencoders tend to reach different types of properties.

2.3 Types of Autoencoder

2.3.1 Denoising Autoencoder

The *denoising autoencoder* creates a damaged copy of the input going to insert some noise. This helps to avoid the autoencoders to copy the input to the output without learning features about the data. These autoencoders take an input that is partially damaged in the training phase to try to recover the original corrupted input. The corruption of this input can be done randomly, by making some of the input as zero. Then denoising autoencoder must remove the corruption to generate an output that is similar to the input.
Finally, the output is compared with both the input and the noised input and to minimize the loss function, you go on until you reach convergence. This type of autoencoder tries to minimize the

loss function between the output and the corrupted input. Denoising goes to help autoencoders to learn the latent representation present in the data and ensures that a good representation can be reliably derived from a corrupted input that will be very important for recovering clean input. It is a stochastic autoencoder becouse a stochastic corruption process is used to set some of the inputs to zero [10].

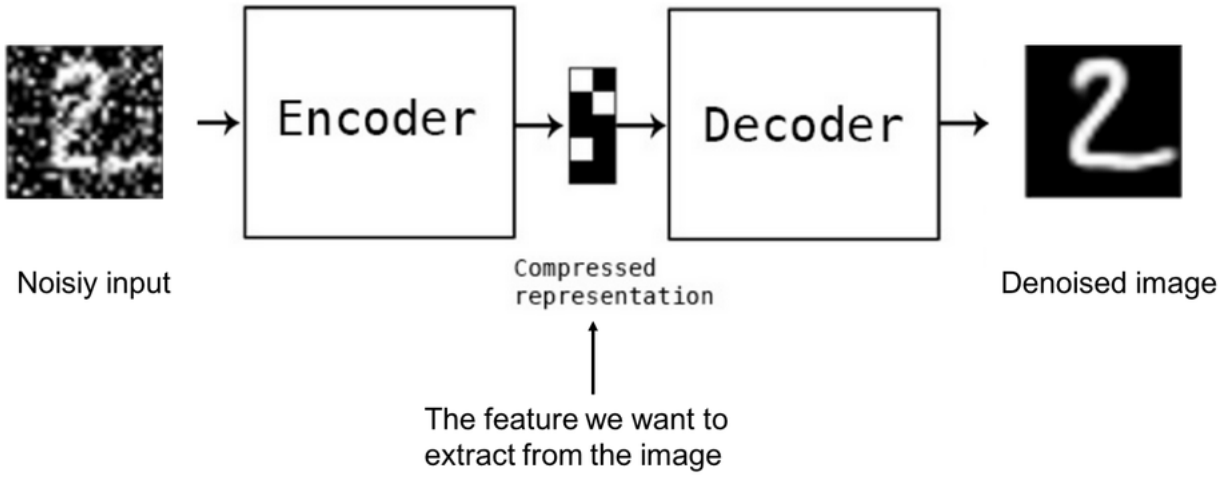

Figure 2.2: Architecture of a Denoising Autoencoder

2.3.2 Sparse Autoencoder

The *sparse autoencoder* consists a single hidden layer, which is connected to the input vector by a weight matrix forming the encoding step. It's may include more hidden units than inputs, but only a small number of the hidden units are allowed to be active at once.
The sparse autoencoders have hidden nodes greater than input nodes, this because can discover important features from the data. Indeed a generic sparse autoencoder the sparsity constraint is included on the hidden layer, all this to prevent output layer copy input data.
An advancement to sparse autoencoders is the k-sparse autoencoder. With this autoencoder k neurons are chosen taking the activation functions with the highest value ignoring other activation functions using ReLU activation functions and adjusting the threshold to find the largest neurons. This tune the value of k to obtain sparsity level best suited for the dataset [11].

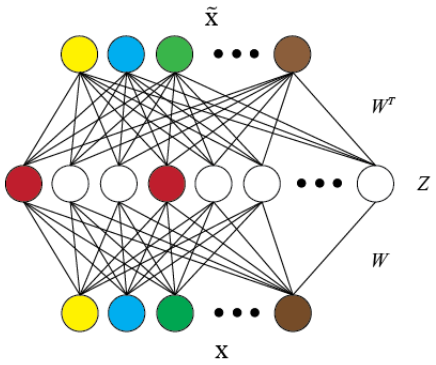

Figure 2.3: Architecture of a Sparse Autoencoder

2.3.3 Contractive Autoencoder

The *contractive autoencoder* objective is to have a robust learned representation which is less sensitive to small variation in the data. To obtain the robustness of this representation for the data is applied a penalty term to the loss of function. This term is Frobenius norm of the Jacobian matrix. Frobenius norm of the Jacobian matrix for the hidden layer is calculated with respect to input and the norm of the Jacobian matrix is the sum of square of all elements.
This autoencoder is another regularization technique like sparse autoencoders and denoising autoencoders. CAE is a better choice than denoising autoencoder to learn useful feature extraction [10].

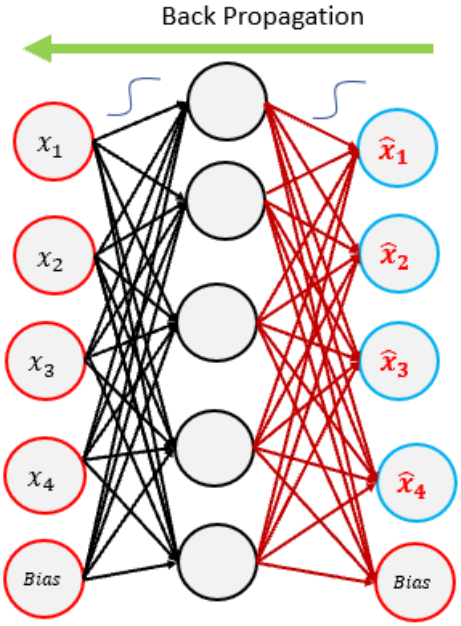

Figure 2.4: Architecture of a Contractive Autoencoder

2.3.4 Undercomplete Autoencoder

The *undercomplete autoencoder* [4] is a type of autoencoder that has the hidden dimension is smaller than the input dimension and this helps to obtain important features from the data. The goal of the autoencoder is to capture the most important features present in the data.
However, using an overparameterized architecture in case of a lack of sufficient training data create overfitting and prevents learning of important features. An advantage of the undercomplete autoencoders is that they do not need any regularization as they maximize the probability of data rather than copying the input to the output [10].

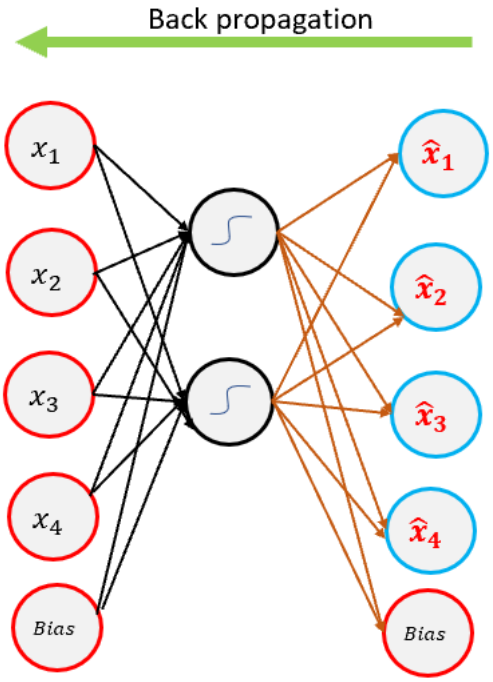

Figure 2.5: Architecture of a Undercomplete Autoencoder - Hidden layer has smaller dimension than input layer

2.3.5 Convolutional Autoencoder

Autoencoders traditionally do not take into consideration the fact that a signal can be seen as a sum of other signals. The *convolutional autoencoders* use the convolution operator to exploit this observation. These autoencoders try to learn how to encode input into a set of very simple signals and and then try to reconstruct the input from them. Once they have been learned, these filters are used on any input to extract features and then to perform any task such as classification.

[4] https://machine-learning-course.readthedocs.io/en/latest/content/deep_learning/autoencoder.html

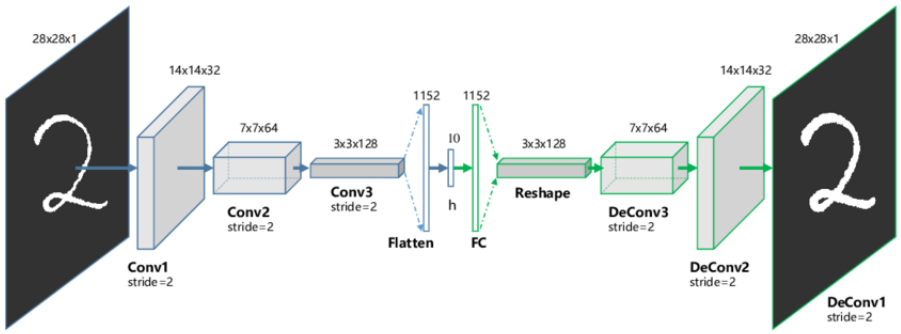

Figure 2.6: Architecture of a Convolutional Autoencoder

2.3.6 Variational Autoencoder

A *variational autoencoder* can be defined as being an autoencoder whose training is regularised to avoid overfitting and ensure that the latent space has good properties that enable generative process [12].
The architecture of this autoencoder is composed of both an encoder and a decoder and that is trained to minimise the reconstruction error between the encoded-decoded data and the initial data. However, to introduce a regularization of the latent space, the coding-decoding process must be modified, even if not in a profound way, since instead of encoding the input as a single point we code it as a distribution on the latent space. The model is then trained as follows:

1. the input is encoded as distribution over the latent space
2. a point from the latent space is sampled from that distribution
3. the sampled point is decoded and the reconstruction error can be computed
4. the reconstruction error is backpropagated through the network

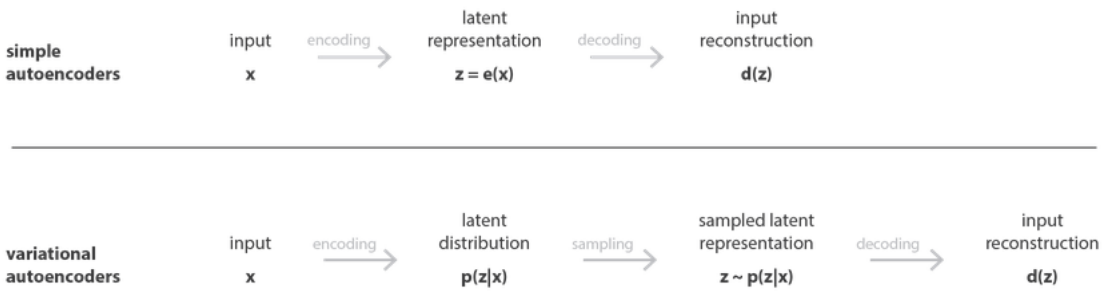

Figure 2.7: Difference between autoencoder (deterministic) and variational autoencoder (probabilistic)

2.3.7 Deep Autoencoder

A *deep autoencoder* [5] is composed of two, symmetrical deep belief networks, one network for encoding and another for decoding. Usually consists of four or five shallow layers representing the encoding half of the net, and second set of four or five layers that make up the decoding half. This layers are *Restricted Boltzmann Machines (RBMs)* [6] which are the building blocks of deep-belief networks, in particular are a two-layered artificial neural network with generative capabilities. RBMs are a special class of Boltzmann Machines and they are restricted in terms of the connections between the visible and the hidden units and this makes it easy to implement them when compared to Boltzmann Machines.

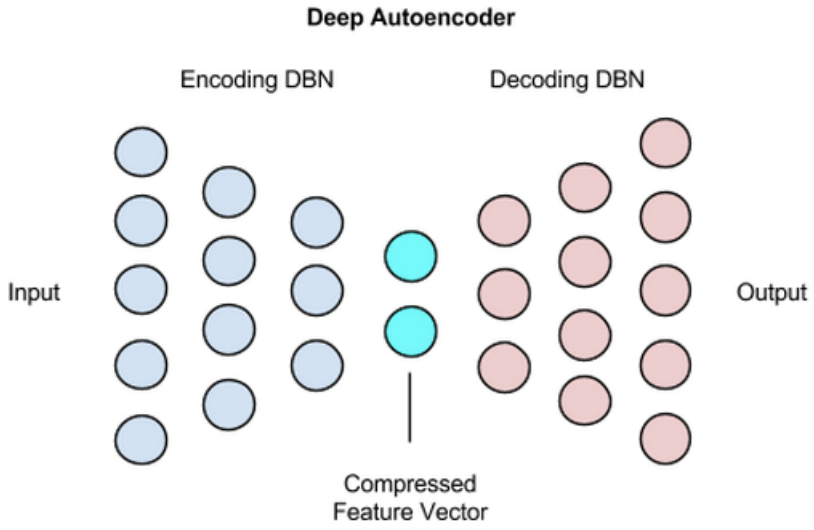

Figure 2.8: Deep Autoencoder)

Deep autoencoders are useful in topic modeling, or statistically modeling abstract topics that are distributed across a collection of documents. Some advantages of these autoencoders are surely that can be used for other types of datasets with real-valued data, on which you would use Gaussian rectified transformations for the RBMs instead and that final encoding is compact and fast.

2.4 Example

Let's see some examples where autoencoders can be used:

[5] https://skymind.ai/wiki/deep-autoencoder
[6] https://en.wikipedia.org/wiki/Restricted_Boltzmann_machine

- *Image coloring*: Autoencoders here are used to convert a black and white image into a colored image. The color obviously depends on what is in the picture.

Figure 2.9: Example of image coloring

- *Feature variation*: It extracts only the required features of an image and generates the output by removing any noise or unnecessary interruption.

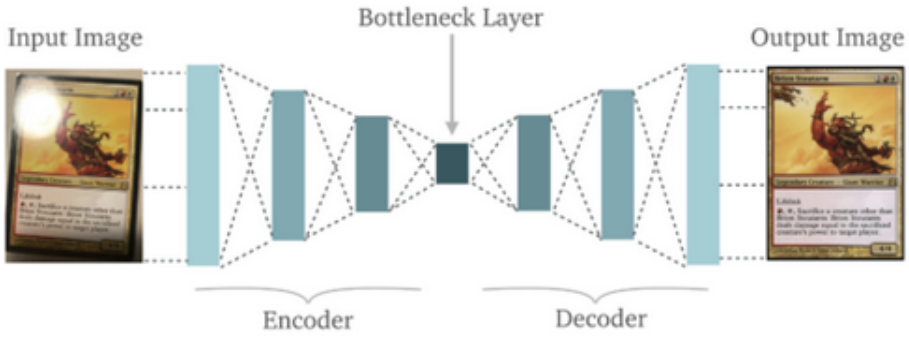

Figure 2.10: Example of feature variation

- *Dimensionality Reduction*: The input image is reconstructed but with reduced dimensions and all this helps to provide the similar image obtained with a reduced pixel value.

Figure 2.11: Example of dimensionality reduction

- *Watermark Removal*: Autoencoders also allow you to remove watermarks from images or to remove anything else while filming a video or a movie.

Figure 2.12: Example of watermark removal

Chapter 3

Data Augmentation

Deep Learning is part of a broader family of machine learning methods based on artificial neural networks. It is is powerful, but they usually need to be trained on massive amounts of data to perform well.
Deep learning models trained on small dataset the low performance of versatility and generalization from the validation and test set [13]. Therefore a frequent problem to which these models suffer is over-fitting. A large number of methods have been proposed to solve this problem, one of which is *Data Augmentation*.
The data augmentation method is very popular in computer vision and increases both the quantity and the diversity of the data. It is a very efficient strategy to reduce over-fitting on the models and improve the diversity of data sets and generalization performances. In particular in the field of image classification there have been clear general improvements, for example like flipping the image horizontally or vertically, translating the image by a few pixels and so on.

3.1 Image data augmentation

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating images with modified versions within the initial data set. Therefore thanks to the augmentation techniques can create variations of the images that can improve the ability of the fit models to generalize what they have learned to new images [14].
By transformations we mean a range of operations from the field of image manipulation, such as shifts, flips, zooms, and much more. For example, a horizontal flip of a picture of a cat may make sense, because the photo could have been taken from the left or right but a vertical flip of the photo of a cat does not make sense because it is not very likely that the model is very unlikely to see a photo of an upside down cat.
Obviously the choice of the specific data augmentation techniques used for a set of training data

must be chosen in a reasonable manner remaining inherent to the context of the training data set and to the knowledge of the problematic domain. Some modern deep learning algorithms such as the convolutional neural network, can learn features that do not vary with respect to their position in the image though augmentation can further aid in this transform invariant approach to learning. Usually this image data augmentation is applied only to the training dataset and not to the validation or test dataset.
Now let's see some basic but powerful augmentation techniques that are popularly used [15].

1. **Flip**: It is possible to flip images horizontally and vertically. Some frameworks do not provide function for vertical flips, but to have a vertical flips it is sufficient to rotate an image 180 degrees.

Figure 3.1: From the left, we have the original image, followed by the image flipped horizontally, and then the image flipped vertically

2. **Rotation**: With this operation it is possible that the size of the initial images may not be preserved. If the image is a square and is rotated at a right angle to the image size is preserved. While if the image is rectangular only rotating it 180 degrees it is possible to preserve the size. The rotation by finer angles changes the final size of the image.

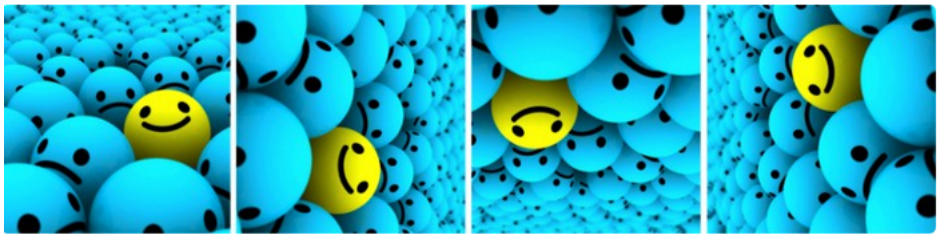

Figure 3.2: The images are rotated by 90 degrees clockwise with respect to the previous one, as we move from left to right

3. **Scale**: The image can be scaled outward or inward. If scaling is done outward, the final size of the image will be larger than the initial size, and most image frameworks will cut out a section from the new image. If scaling is done inward, the image size is reduced.

Figure 3.3: From the left, we have the original image, the image scaled outward by 10%, and the image scaled outward by 20%

4. **Crop**: Unlike scaling, a section of the original image is randomly sampled. Subsequently, the section is resized to the same size as the original image and this method is called *random cropping*.

Figure 3.4: From the left, we have the original image, a square section cropped from the top-left, and then a square section cropped from the bottom-right. The cropped sections were resized to the original image size

5. **Translation**: Translation just involves moving the image along the X or Y direction (or both). In this example we assume that the image has a black background beyond its boundary, and are translated appropriately. This method of augmentation is very useful as most objects can be located at almost anywhere in the image.

Figure 3.5: From the left, we have the original image, the image translated to the right, and the image translated upwards

6. **Gaussian Noise**: Overfitting can occur when the neural network attempts to learn high frequency features that are not useful. Gaussian noise, which has zero mean, essentially has data points in all frequencies, effectively distorting the high frequency features. This means that even low-frequency components are distorted and with the addition of the right amount of noise can improve learning ability.

Figure 3.6: From the left, we have the original image, image with added Gaussian noise, image with added salt and pepper noise

3.2 Text Augmentation

In natural language processing (*NLP*) field is difficult to make a good text augmentation due to high complexity of language. This is because all words cannot be replaced with others and not all words have a synonym. So this means that even a changed word leads to a different context. While generating augmented image as seen in the previous section is certainly simpler because even inserting noise or cropping out portion of image the model can classify the image.
Let us now look at two approaches that are used quite frequently.

3.2.1 Back translation

Back translation [16] is a very popular technique for in-domain data augmentation. In this experiment [17] the Machine Translation on Noisy Text (MTNT) dataset was back-translated which contains parallel sentence pairs with comments crawled from Reddit [1] and manual translations. This dataset contains user-generated text with different kinds of noise (e.g. typos, grammatical errors, emojis, spoken languages, etc.). The MTNT dataset is used as in-domain data, where models are trained with clean data and adapted to noisy data.
This dataset was back-translated using a model fine-tuned on MTNT parallel corpus. However, the goal is to try to improve robustness by producing less noisy outputs, but the data generated

[1] www.reddit.com

with back translation can have noisy target translations (from monolingual data) and less noisy source texts (from back translation).
To avoid increasing the noise level of the output texts, the forward translation used has also been experimented using models fine-tuned on the noisy parallel corpus. Pseudo parallel data generated by forward translation were used for fine tuning models on the same language direction. While to avoid overfitting they have been merged the noisy parallel data of both language directions to produce noisy forward translations. The pseudo parallel data generated by back translation and forward translation have been combined with noisy parallel data and fine-tuned on the baseline model.

3.2.2 Easy Data Augmentation (EDA)

Easy Data Augmentation (EDA) techniques for boosting performance on text classification tasks,in particular they use simple heuristics to augment training data. For a given sentence in the training set, randomly chosen, is performed one of the following operations [18]:

1. **Synonym Replacement (SR)**: Randomly n words are chosen from the sentence excluding the stop words. Replace each of these words with one of its synonyms chosen at random.

2. **Random Insertion (RI)**: A synonym of a random word in the sentence is found excluding a stop word. Insert that synonym into a random position in the sentence. Do this n times.

3. **Random Swap (RS)**: Two words are chosen in the sentence and their positions are exchanged. Do this n times.

4. **Random Deletion (RD)**: Every word in the sentence is removed with probability p.

Since long sentences have more words than short ones, the latter can assimilate more noise while maintaining their original class label. for this the number of words n is changed for Synonym Replacement, Random Insertion and Random Swap based on the sentence length l with the formula n=αl, where α is a parameter that indicates the percent of the words in a sentence are changed (we use p=α for Random Deletion).
Thus, for each original sentence, are generated n_{aug} augmented sentences.

Example

Suppose we have this as the original sentence:

The final project is the main assignment of the course.

- **SR**: The final project is the principal assignment of the course.
- **RI**: The final project is the main assignment of last the course. (last∼final)
- **RS**: The final project is the assignment main of the course.
- **RD**: The final is the main assignment of the course. (project deleted)

Therefore in data augmentation, input data is altered while class labels are maintained. So if the sentences have a different meaning then original class labels may no longer be valid.
Now let's see a visualization approach done in the paper[18] to see if EDA operations significantly change the meanings of augmented sentences. First, an RNN is trained on the pro-con classification task (PC) without augmentation. Then the EDA is applied to the test set by generating nine augmented sentences per original sentence. These are fed into the RNN along with the original sentences, and we extract the outputs from the last dense layer. t-SNE (t-distributed stochastic neighbor embedding) are applied to these vectors and plot their 2-D representations.

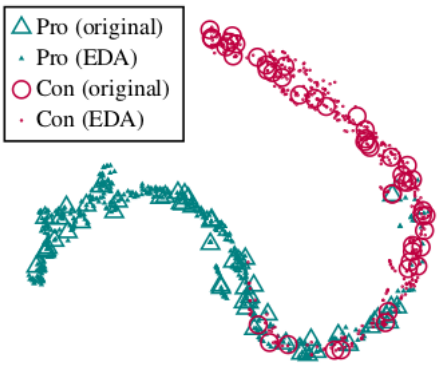

Figure 3.7: Latent space visualization of original and augmented sentences in the Pro-Con dataset

From the figure above it can be seen that the resulting latent space representations for augmented sentences closely surrounded those of the original sentences, which points out that most sentences augmented with EDA conserved the labels of their original sentences.

3.3 SpecAugment

The Automatic Speech Recognition (ASR) [2], is a process of taking an audio input and transcribing it to text that has had a net benefit with the continuous development of deep neural networks. Precisely for this reason it has become present in many modern devices and products such as Google Assistant, Google Home and YouTube.
Despite all this, several challenges still remain in developing deep learning-based ASR systems. The main problem we find is that the ASR models have many parameters and this tends to overfit the training data and therefore consequently they are unable to generalize in order not to see the data when the training set is not extensive enough.
In the absence of a large number of training data it is possible to increase the size of existing data through the process of data augmentation which has contributed as seen previously to improving the performance of deep networks in the domain of image classification and which for this reason has been proposed as a method to generate additional training data for ASR [19].
In speech recognition, data augmentation occurs with the deformation of the audio waveform that was used for training. This deformation can be made of any type, for example by speeding it up or slowing it down or even adding background noise. This allows you to get a data set that is actually wider because more increased versions of a single input are inserted into the network during the training course and also allows the network to become more robust by forcing it to learn relevant features. The problem is that all existing conventional methods introduces additional computational cost and sometimes requires additional data.
In a recent paper "*SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition*" [20] a new approach was used that allows to augmenting audio data but treating them more as a visual problem than an audio problem. So instead of augmenting the input audio waveform as it was done in the previous methods explained above, the new method called *SpecAugment* applies an augmentation policy starting directly from the spectogram of the audio file and not from the audio waveform. This method used has enormous advantages compared to the previous ones, starting from the fact that it is certainly cheaper computationally and does not require further dates than the initial ones.
Thus in the traditional ASR, the audio waveforms are generally encoded as a visual representation such as a spectogram before being inserted later as a training data for the network. However, this augmentation of training data is applied to the waveform audio so before it is transformed into a spectogram, his is because new spectograms can be generated at each iteration to be used. While for the SpecAugment method we study an approach to try increase the data directly from the spectogram rather than from the waveform data.

[2] https://en.wikipedia.org/wiki/Speech_recognition

Since the augmentation is applied directly to the input features of the network, it can be run online during training without significantly impacting training speed.

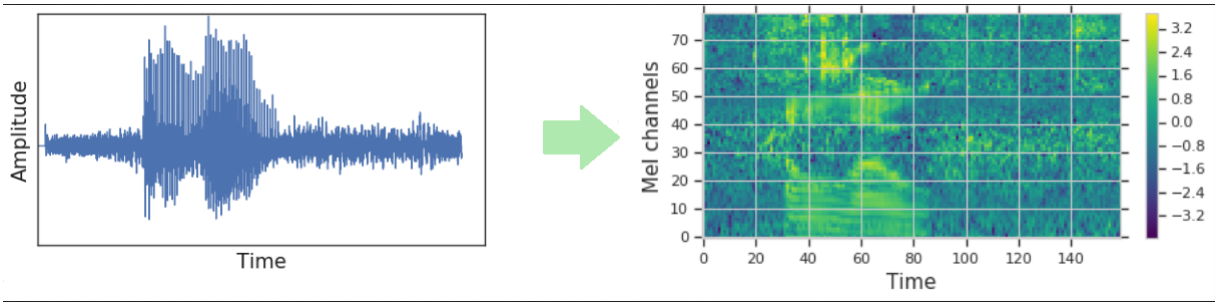

Figure 3.8: A waveform is typically converted into a visual representation before being fed into a network

The augmentation policy that is construct goes to act directly on the log mel spectrogram to give the network learn useful features. Specifying that these features should be robust to deformations in the time direction, partial loss of frequency information and partial loss of small segments of speech have been chosen the following deformations to make up a policy [20]:

1. **Time warping**: Given a log mel spectrogram with τ time steps is seen it as an image where the time axis is horizontal and the frequency axis is vertical. A random point along the horizontal line passing through the center of the image within the time steps (W, τ-W) is to be warped either to the left or right by a distance w chosen from a uniform distribution from 0 to the time warp parameter W along that line.
   So the Time Warp technique simply shifts the spectrogram in time by using interpolation techniques to squeeze and stretch the data in a randomly chosen direction [21].

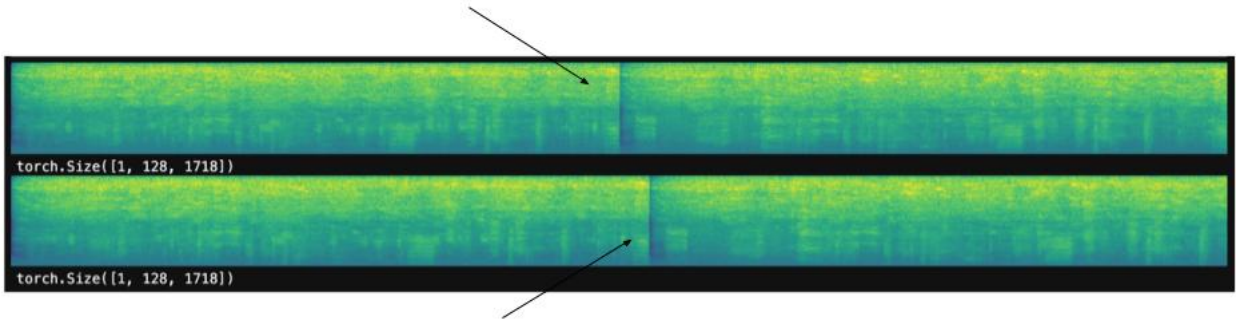

Figure 3.9: Time warping a spectrogram

2. **Frequency masking**: It is applied such that f consecutive mel frequency channels $[f_0, f_0+f)$ are masked, where f is first chosen from a uniform distribution from 0 to the

frequency mask parameter F and f_0 is chosen from [0,v-f). v is the number of mel frequency channels.
So in the frequency masking is masked a randomly chosen band of frequencies with the mean value of the spectrogram or zero.

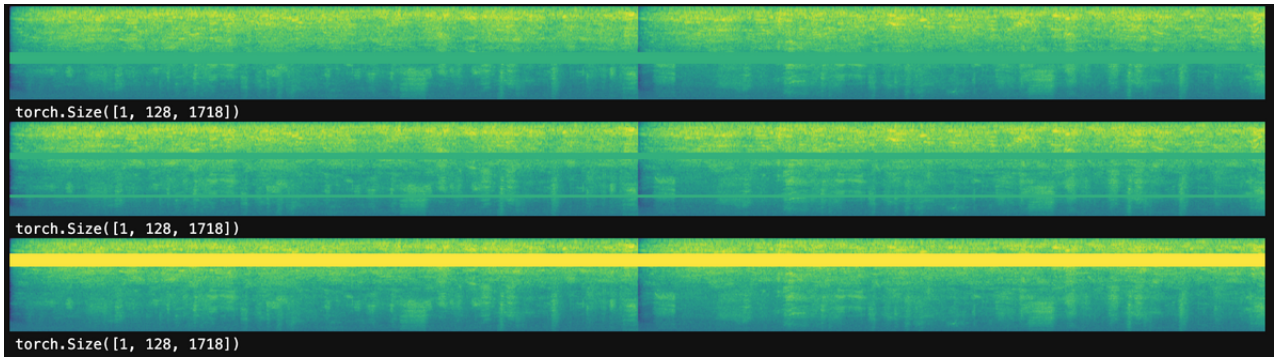

Figure 3.10: Frequency masking a spectrogram

3. **Time masking**: It is applied such that t consecutive time steps [t_0,t_0+t) are masked, where t is first chosen from a uniform distribution from 0 to the time mask parameter T, and t_0 is chosen from [0,τ- t).
Put simple in the time masking is masked a randomly chosen lice of time steps with the mean value of the spectrogram. It appears with time on the X axis and frequency bands on the Y axis, as we can see in the spectogram below.

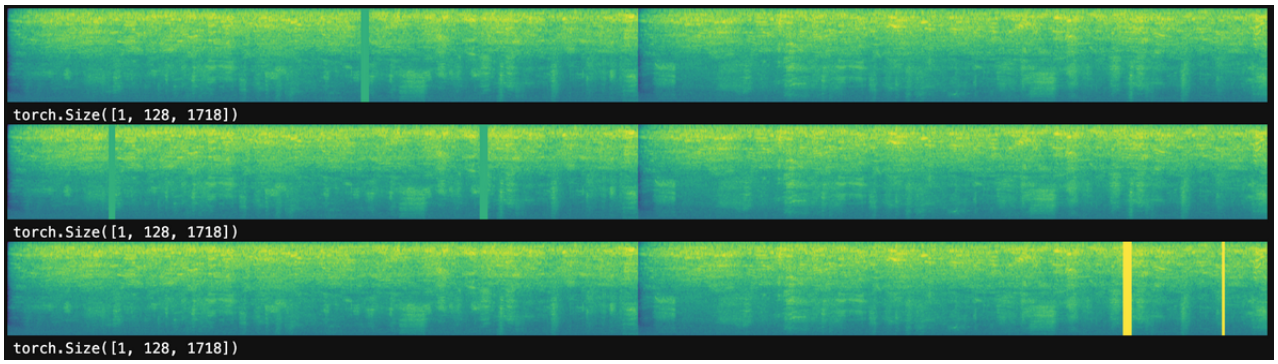

Figure 3.11: Time masking a spectrogram

The final spectrogram with all three techniques applied will be the following:

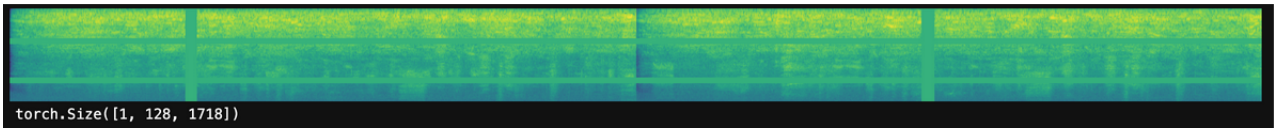

Figure 3.12: All three augmentations combined on a single spectrogram

Chapter 4

Proposed approach to speech analysis for Autoencoder

The main problem in the analysis of audio files is mainly the way in which these must be represented within the analysis system. Typically the audios are converted into a feature vector, obtained manually or automatically, and then analyzed using machine learning techniques.
In the works done previously and explained in the previous chapter in the state of the art, the features were extracted or manually then transcribed through a human or automatically [2] [1] [3] [4].
Our idea was instead to automatically extract features and learn the latter using neural networks in particular with the autoencoders and then classify them through machine learning techniques.Initially we tried to think about the problems that could be created by the extraction of features performed manually.
The problems we found most were:

1. Using manually transcribed texts may cause a loss of useful information of the original data.

2. The manually added features such as pauses, lapsus etc.. can be reported differently if it is not the same person to transcribe them.

3. Therefore a consequence of the previous point implies that this method can be very expansive and therefore not scalable and standardized.

So to try to improve the previous methods we tried to avoid using a manually written text but try to extract the spectograms from the audio files and get through a particular neural network called *Autoencoder* which we will explain in the previous chapter, the features. After that they will be classified. All this will be done using a tool called "**auDeep**" which is a Python toolkit for learning unattended functions with deep neural networks (DNN) and the main focus of this project is feature extraction from audio data with deep recurrent autoencoders.
Furthermore, after trying this method we also tried to expand the starting dataset, not being large. The technique used is that of the *SpecAugment* which does nothing more than create new samples using specific techniques to modify the original data through, as already explained in the previous chapter. Now let's start explaining the datasets we started from and then we'll explain the work that was done by us.

4.1 Dataset

The first point we started from was definitely the dataset. The starting dataset we used is the same that was used in the experiment of the state of the art. The original dataset, provided by the University of Bologna, is composed of 96 participants with an age ranging between 50 and 75 years.
Within it we find 48 participants with cognitive decline in turn divided into 32 with mild cognitive decline and 16 with early dementia, and the other 48 participants without cognitive decline. The information on the dataset in more detail is reserved and not accessible to all, in order to use the dataset a data confidentiality sheet has been signed so that the data will not be disclosed outside the following project.
The dataset folder is called "*test_linguaggio*" and within it there are 96 sub folders that correspond to each participant. Inside each sub-folder we find three audio files in *WAV* format (44.1KHz, 16 bit) that correspond to the description of a figure, of the working day and of the last dream done as already explained above. The length of the audio files varies between approximately 10 seconds to 9 minutes.

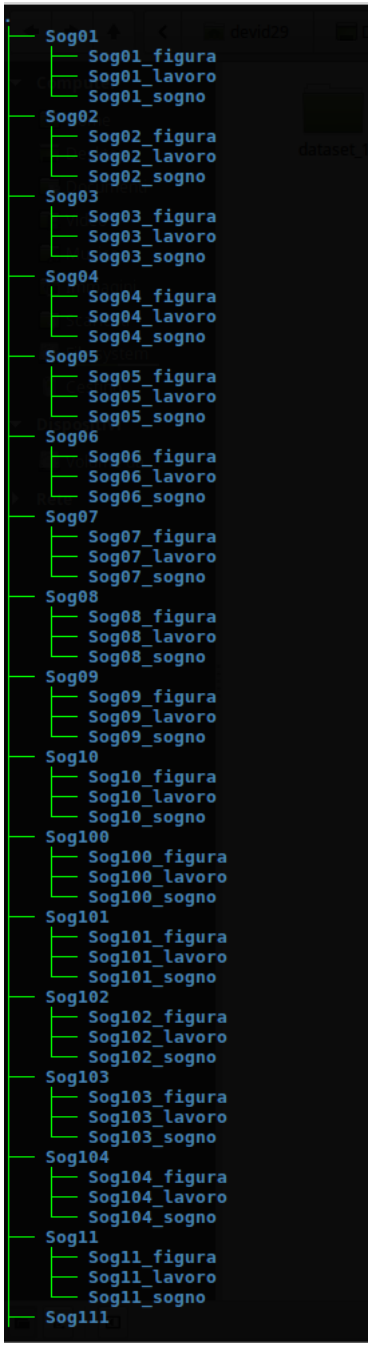

Figure 4.1: "*test_linguaggio*" folder structure

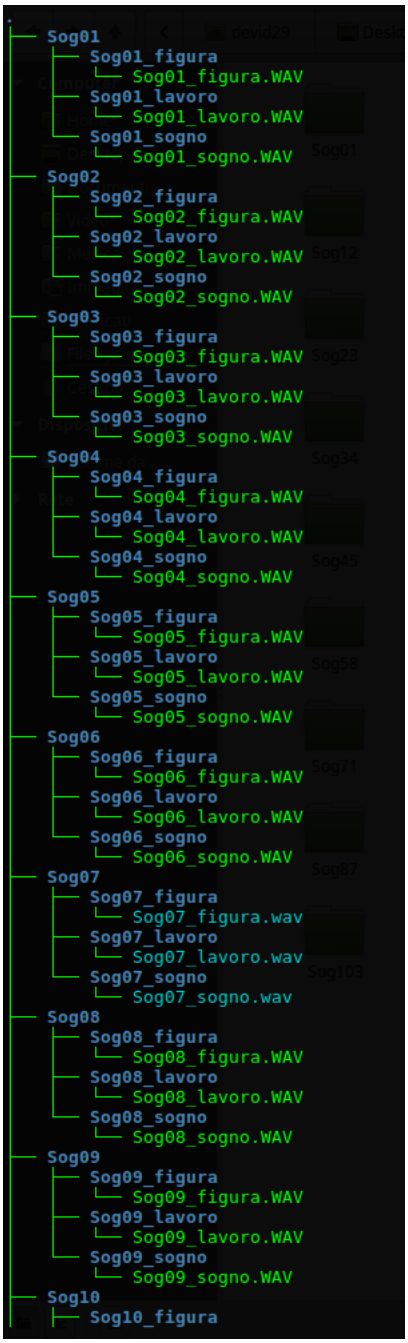

Figure 4.2: "*test_linguaggio*" folder structure with .WAV

For our experiment we started from this dataset and we divided it into two parts, one for the training set and one for the test set. The proportions in which it was divided were 80% for the training set and 20% for the test set. More precisely we have built two types of datasets:

1. **dataset_1**: this dataset is made up of two sub-folders: one for the training set and one for the test set, called "*train*" and "*devel*" respectively. Within them there are respectively three folders that correspond to the three classes Control (**CONT**), Mild Cognitive impairment (**MCI**) and early Dementia (**eD**). The division in training and test sets was done randomly.

Figure 4.3: "*dataset_1*" folder structure

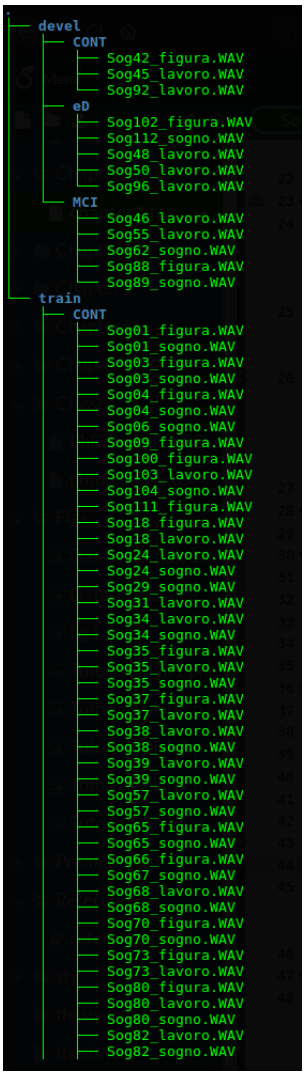

Figure 4.4: "*dataset_1*" folder structure with .WAV

2. **dataset_2**: this dataset is made up of two sub-folders: one for the training set and one for the test set, called "*train*" and "*devel*" respectively. Within them there are respectively two folders that correspond to the two classes Control (**CON**), and Decline (**DEC**). The division in training and test sets was done randomly.

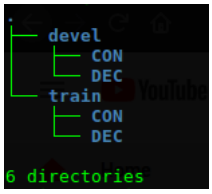

Figure 4.5: "*dataset_2*" folder structure

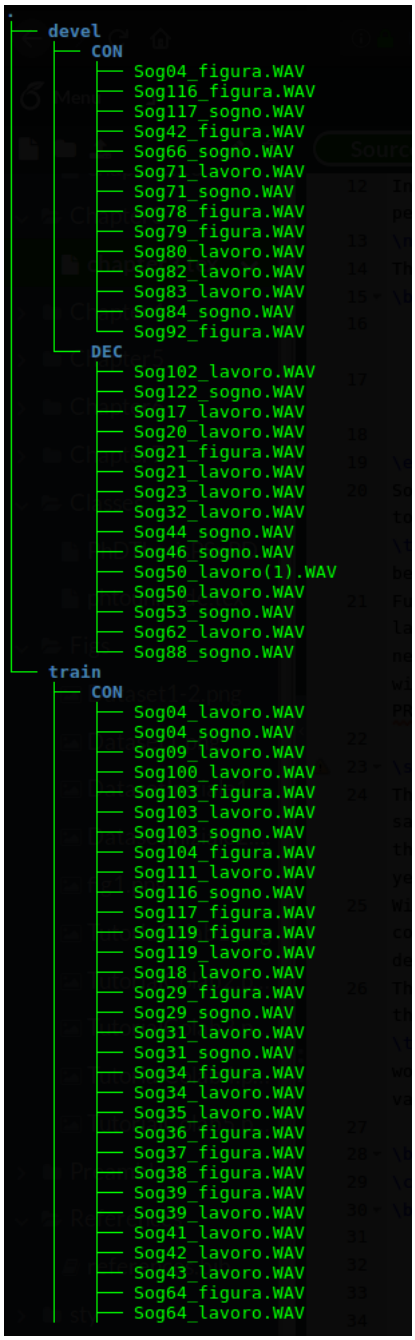

Figure 4.6: "*dataset_2*" folder structure with .WAV

The two datasets have been uploaded to *Google Drive* which is a web service, in a cloud computing environment, for online storage and synchronization.

4.2 Software platform: Google Colab

The platform where we performed our experiments is *Google Colab* [1] also called **Colaboratory**. Google has provided free cool service based on Jupyter Notebooks that supports free GPU. It also allows absolutely anyone to develop deep learning applications using popular libraries such as PyTorch, TensorFlow, Keras, and OpenCV. Google Colab provides free **Tesla K80 GPU of about 12GB** and it's totally free [22].
However, there are limits in fact it only supports python 2.7 and 3.6 but does not support R or Scala. In addition, there is also a limit for sessions and sizes, but can be safely solved by reloading the files. More precisely it is possible to run the session in an interactive Colab notebook for 12 hours. This is because there may be chances that people use it for wrong purposes so after 12 hours, you can restart the session again [23].

4.2.1 Create a folder for notebooks

It is possible to create a Colab Notebook in the following way:

Visit *Google Drive*, *Right Click -> More -> Colaboratory or New -> More -> Colaboratory to start a new Colab Notebook.*

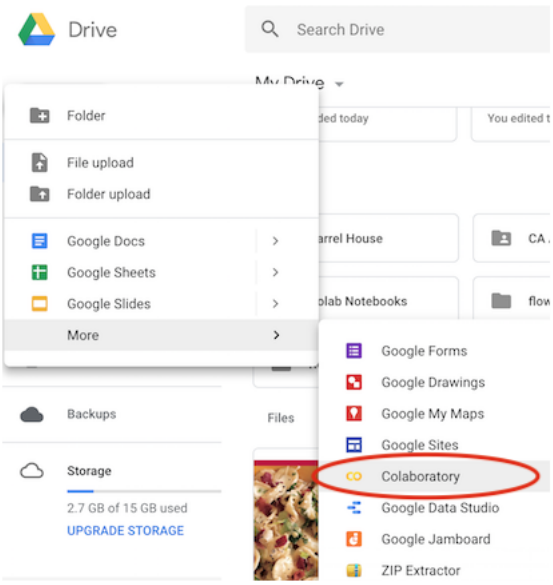

Figure 4.7: Create a folder for your notebooks

[1] https://colab.research.google.com/notebooks/welcome.ipynb

4.2.2 Set up free GPU

By default the runtime type is on NONE, which means that the hardware accelerator [2] would be CPU. Hardware acceleration is the use of computer hardware specially made to perform some functions more efficiently than is possible in software running on a general-purpose CPU. Therefore the implementation of computing tasks in hardware to decrease latency and increase throughput is known as hardware acceleration. So let's see how to switch from CPU to GPU. Open the *Runtime menu -> Change Runtime Type -> Select GPU*.

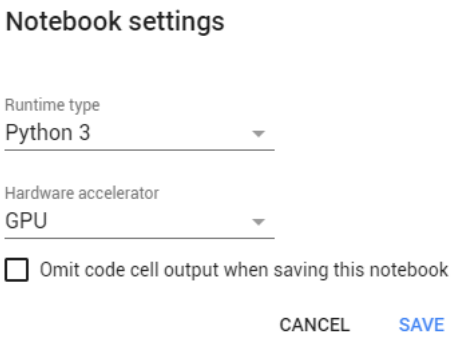

Figure 4.8: Connecting to Server and Setting up GPU Runtime

4.2.3 Mounting Google Drive to Colab Notebook

So, it is important to connect your session to Google Drive as an external storage.
Running the code below, will help you connect to Google Drive. You will be asked to authorize through your Google account.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Figure 4.9: Code to Mount Your Google Drive to Colab Notebook

When you execute the code above, click on the link that appears, select the Google account you wish to connect, copy and paste the authorization code into the box and press Enter.

[2] https://en.wikipedia.org/wiki/Hardware_acceleration

Figure 4.10: Pre-Authorization

Once authorized, you can use Google Colab without problems. To be sure that the authorization has been successful, the following result should appear:

Figure 4.11: Post-Authorization

4.3 Experiment

The software used for our experimentation was **auDeep** [3]. The latter based Python software for deep unsupervised representation learning from acoustic data. All this is achieved using a recurrent sequence to sequence autoencoder approach. auDeep can be used both through its Python API as well as through an extensive command line interface [24].

[3]https://github.com/auDeep/auDeep

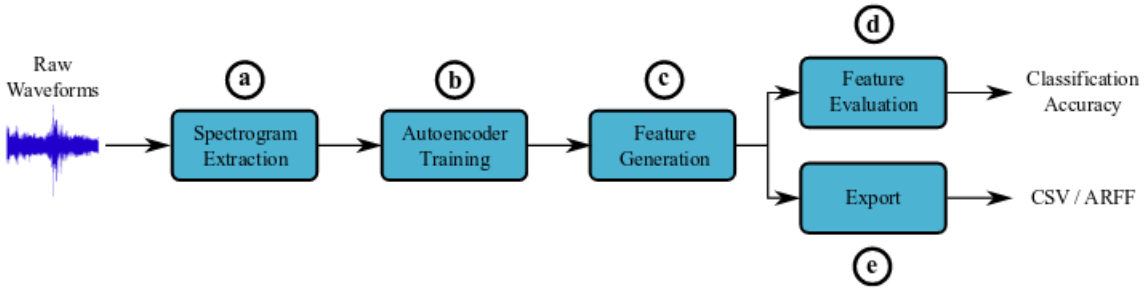

Figure 4.12: Illustration of the feature learning procedure with *auDeep*

auDeep contains at its core a high-performing implementation of sequence to sequence autoencoders which is not specifically constrained to acoustic data, but they are also provided additional functionality for representation learning from audio. It provides a highly modularised Python library for deep unsupervised representation learning from audio. The core sequence to sequence autoencoder models are implemented TENSORFLOW. In particular this implementation extends the built-in sequence to sequence learning capabilities of TensorFlow. Furthermore, diversely structured data sets are handled by the system in a unified way without requiring time-consuming manual adjustments and in addition the topology and parameters of autoencoders are stored as TENSORFLOW checkpoints.
The system is platform-independent, in fact it can be tested on both Windows and different Linux distributions on desktop PCs and in a cluster environment. auDeep is capable of running on CPU only, and GPU-acceleration is leveraged automatically when available. In our case as already mentioned in the previous section thanks to the Google Colab software platform we have a GPU more precisely a *Tesla K80 GPU of about 12GB* totally free.

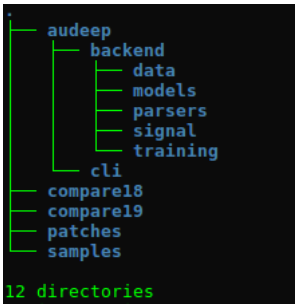

Figure 4.13: *audeep* folder structure

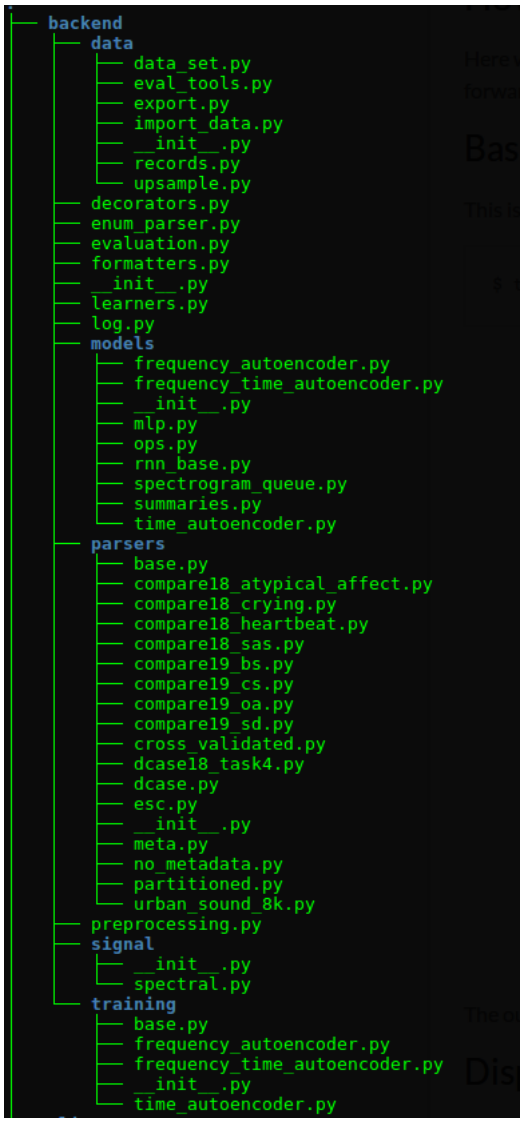

Figure 4.14: *backend* folder structure

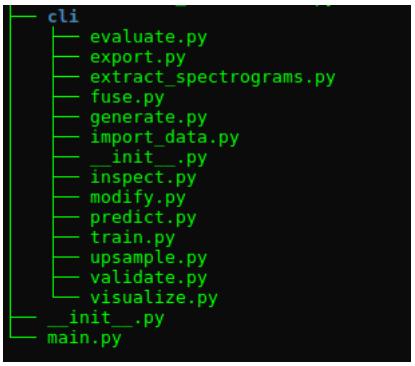

Figure 4.15: *cli* folder structure

4.3.1 Experimental settings

The minimal requirements to install auDeep are listed below.

- *Python 3.5*
- *TkInter* (python3-tk package on Ubuntu, selectable during Python install on Windows)
- *virtualenv* (pip3 install virtualenv)

Now we see below some basic Python dependencies for use that can be installed using the well-known **pip3** command:

- cliff
- liac-arff
- matplotlib
- netCDF4
- pandas
- scipy
- sklearn
- tensorflow or tensorflow-gpu 1.13.0
- xarray

In our case we have decided to upload directly to Google Colab the auDeep tool with the **git clone** command that clones a repository into a newly created directory, creates remote tracking branches for each branch in the cloned repository, and creates and checks out an initial branch that is forked from the cloned repository's currently active branch.

1. In the figure below it is possible to understand how it was cloned on Google Colab:

Figure 4.16: git clone to auDeep

Warning: in the Google Colab platform before the known installation commands must be added *"!"*.

2. Continue by installing auDeep with:

Figure 4.17: Installing auDeep

3. After installing auDeep install the *netCDF4* [4] library. netCDF4-python is a Python interface to the netCDF C library and has many features not found in earlier versions of the library and is implemented on top of HDF5 [5]. This module can read and write files in both the new netCDF 4 and the old netCDF 3 format. The version of netCDF4 that must be installed is 1.4.0 otherwise it will not work:

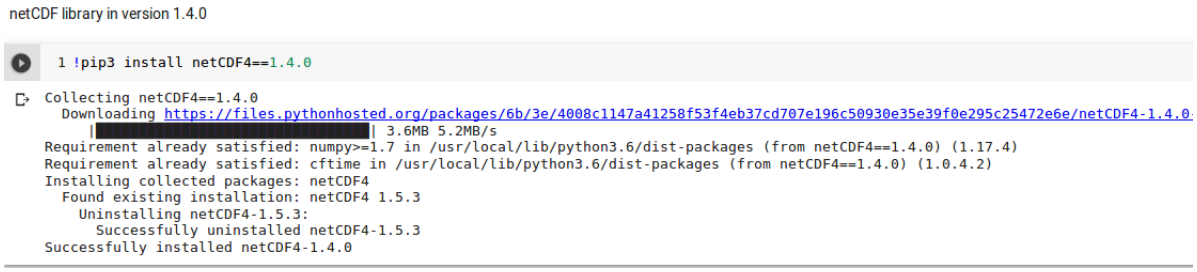

Figure 4.18: Installing netCDF library version 1.4.0

[4] https://en.wikipedia.org/wiki/NetCDF
[5] https://en.wikipedia.org/wiki/Hierarchical_Data_Format

4.3.2 Experiment on three classes (CON, MCI, eD)

Overview

Representation learning with auDeep is performed in in about five distinct stages:

1. Extraction of spectrograms and data set metadata from raw audio files (audeep *preprocess*)
2. Modify data set metadata in various ways (audeep ... *modify*)
3. Training of a DNN on the extracted spectrograms (audeep ... *train*)
4. Feature generation using a trained DNN (audeep ... *generate*)
5. Evaluation of generated features (audeep ... *evaluate*)

Experiment on dataset_1 (without augmentation)

The first point that is made is to extract spectrograms and some metadata from the raw audio files. In order to get a general overview of the audio files contained in a data set you can use this command:

```
!audeep inspect raw  --basedir .../dataset_1 --parser audeep.backend.parsers.
    partitioned.PartitionedParser
```

This will print some logging messages, and a table containing information about the data set:

```
[WARNING] tensorflow - From /usr/local/lib/python3.6/dist-packages/audeep/main.py:98:
     The name tf.logging.set_verbosity is deprecated. Please use tf.compat.v1.logging.
    set_verbosity instead.

[WARNING] tensorflow - From /usr/local/lib/python3.6/dist-packages/audeep/main.py:98:
     The name tf.logging.FATAL is deprecated. Please use tf.compat.v1.logging.FATAL
    instead.

[INFO] InspectRaw - reading audio file information
+-----------------------------------+
| data set information              |
+-----------------------+-----------+
| number of audio files |       285 |
| number of labels      |         3 |
| cross validation folds |         0 |
| minimum sample length |    9.13 s |
| maximum sample length | 537.01 s |
```

```
| sample rate            | 44100 Hz |
| channels               |        1 |
+------------------------+----------+
```

Listing 4.1: *inspect raw*

As we can see from the table the dataset_1 data set contains audio files that are between 9.13 seconds and 537.01 seconds long , contain one channel, and are sampled at 44.1 kHz.
Next, we determine suitable parameters for spectrogram extraction. In general auDeep needs a little larger FTT [6] windows during spectrogram extraction unlike for example the extraction of MFCCs [7]. auDeep works well on mel-spectrograms with a relatively large number of frequency bands. For the our dataset, we would recommend using 160 ms wide FFT windows with overlap 80 ms, and 256 mel frequency bands.
Use the following command to quickly plot a spectrogram with the parameters recommended above:

```
!audeep preprocess --basedir .../dataset\_1 --parser audeep.backend.parsers.
    partitioned.PartitionedParser --window-width 0.16 --window-overlap 0.08 --mel-
    spectrum 256 --fixed-length 5 --clip-below -60 --output spectrograms/dataset1
    -0.08-0.04-128-60.nc
```

Here, the *window-width* 0.16 and *window-overlap* 0.08 options specify the FFT window width and overlap in seconds, respectively. While with the command *mel-spectrum* 256 is indicated that 256 mel frequency bands should be extracted, *fixed-length* 5 option indicates that we want to extract spectrograms from 5 seconds of audio. If samples are shorter than 5 seconds, they are padded with silence, and if they are longer, they are cut to length. Finally *clip-below -60* option is used to filter a part of the background noise by clipping amplitudes below a certain threshold. We recommend a threshold around -45 dB to -60 dB as a starting point.
Once the preprocessing command has been started this will be the result:

```
[WARNING] tensorflow - From /usr/local/lib/python3.6/dist-packages/audeep/main.py:98:
     The name tf.logging.set_verbosity is deprecated. Please use tf.compat.v1.logging.
    set_verbosity instead.

[WARNING] tensorflow - From /usr/local/lib/python3.6/dist-packages/audeep/main.py:98:
     The name tf.logging.FATAL is deprecated. Please use tf.compat.v1.logging.FATAL
    instead.

```

[6] https://en.wikipedia.org/wiki/Window_function
[7] https://en.wikipedia.org/wiki/Mel-frequency_cepstrum

```
[INFO] ExtractSpectrograms - parsing data set at /gdrive/My Drive/Colab Notebooks/
    Tirocinio-Tesi/AutoEncoder_audio/dataset_1
[INFO] ExtractSpectrograms - processing /gdrive/My Drive/Colab Notebooks/Tirocinio-
    Tesi/AutoEncoder_audio/dataset_1/devel/CONT/Sog104_figura.WAV (  1/285)
[INFO] ExtractSpectrograms - processing /gdrive/My Drive/Colab Notebooks/Tirocinio-
    Tesi/AutoEncoder_audio/dataset_1/devel/CONT/Sog104_lavoro.WAV (  2/285)
[INFO] ExtractSpectrograms - processing /gdrive/My Drive/Colab Notebooks/Tirocinio-
    Tesi/AutoEncoder_audio/dataset_1/devel/CONT/Sog111_sogno.WAV (  3/285)
[INFO] ExtractSpectrograms - processing /gdrive/My Drive/Colab Notebooks/Tirocinio-
    Tesi/AutoEncoder_audio/dataset_1/devel/CONT/Sog116_figura.WAV (  4/285)
[INFO] ExtractSpectrograms - processing /gdrive/My Drive/Colab Notebooks/Tirocinio-
    Tesi/AutoEncoder_audio/dataset_1/devel/CONT/Sog116_lavoro.WAV (  5/285)
[INFO] ExtractSpectrograms - processing /gdrive/My Drive/Colab Notebooks/Tirocinio-
    Tesi/AutoEncoder_audio/dataset_1/devel/CONT/Sog116_sogno.WAV (  6/285)
[INFO] ExtractSpectrograms - processing /gdrive/My Drive/Colab Notebooks/Tirocinio-
    Tesi/AutoEncoder_audio/dataset_1/devel/CONT/Sog117_lavoro.WAV (  7/285)
[INFO] ExtractSpectrograms - processing /gdrive/My Drive/Colab Notebooks/Tirocinio-
    Tesi/AutoEncoder_audio/dataset_1/devel/CONT/Sog119_sogno.WAV (  8/285)
[INFO] ExtractSpectrograms - processing /gdrive/My Drive/Colab Notebooks/Tirocinio-
    Tesi/AutoEncoder_audio/dataset_1/devel/CONT/Sog41_lavoro.WAV (  9/285)
[INFO] ExtractSpectrograms - processing /gdrive/My Drive/Colab Notebooks/Tirocinio-
    Tesi/AutoEncoder_audio/dataset_1/devel/CONT/Sog41_sogno.WAV ( 10/285)
[INFO] ExtractSpectrograms - processing /gdrive/My Drive/Colab Notebooks/Tirocinio-
    Tesi/AutoEncoder_audio/dataset_1/devel/CONT/Sog42_figura.WAV ( 11/285)
[INFO] ExtractSpectrograms - processing /gdrive/My Drive/Colab Notebooks/Tirocinio-
    Tesi/AutoEncoder_audio/dataset_1/devel/CONT/Sog43_lavoro.WAV ( 12/285)
[INFO] ExtractSpectrograms - processing /gdrive/My Drive/Colab Notebooks/Tirocinio-
    Tesi/AutoEncoder_audio/dataset_1/devel/CONT/Sog43_sogno.WAV ( 13/285)
[INFO] ExtractSpectrograms - processing /gdrive/My Drive/Colab Notebooks/Tirocinio-
    Tesi/AutoEncoder_audio/dataset_1/devel/CONT/Sog45_figura.WAV ( 14/285)
[INFO] ExtractSpectrograms - processing /gdrive/My Drive/Colab Notebooks/Tirocinio-
    Tesi/AutoEncoder_audio/dataset_1/devel/CONT/Sog45_lavoro.WAV ( 15/285)
[INFO] ExtractSpectrograms - processing /gdrive/My Drive/Colab Notebooks/Tirocinio-
    Tesi/AutoEncoder_audio/dataset_1/devel/CONT/Sog45_sogno.WAV ( 16/285)
[INFO] ExtractSpectrograms - processing /gdrive/My Drive/Colab Notebooks/Tirocinio-
    Tesi/AutoEncoder_audio/dataset_1/devel/CONT/Sog57_figura.WAV ( 17/285)
[INFO] ExtractSpectrograms - processing /gdrive/My Drive/Colab Notebooks/Tirocinio-
    Tesi/AutoEncoder_audio/dataset_1/devel/CONT/Sog71_figura.WAV ( 18/285)
[INFO] ExtractSpectrograms - processing /gdrive/My Drive/Colab Notebooks/Tirocinio-
    Tesi/AutoEncoder_audio/dataset_1/devel/CONT/Sog71_sogno.WAV ( 19/285)
..................
..................
```

Listing 4.2: Step 1: *Preprocess*

After the command finishes, the extracted spectrograms have been stored in netCDF 4 format in a file called **dataset1-0.08-0.04-128-60.nc**.
Before training the autoencoder we are going to apply another command called *modify*:

```
!audeep modify --input spectrograms/dataset1-0.08-0.04-128-60.nc --output
    spectrograms/cross-dataset1-0.08-0.04-128-60.nc --add-cv-setup 20
```

This command modify data set metadata in various ways. the only options required are *input* and *output*, then the rest are optional. in particular we have added the option *add* to randomly generate a cross-validation setup for the data set with NUM_FOLDS evenly-sized non-overlapping folds. In this case as we can see below we used a cross validation with 20 folds:

```
[WARNING] tensorflow - From /usr/local/lib/python3.6/dist-packages/audeep/main.py:98:
     The name tf.logging.set_verbosity is deprecated. Please use tf.compat.v1.logging.
    set_verbosity instead.

[WARNING] tensorflow - From /usr/local/lib/python3.6/dist-packages/audeep/main.py:98:
     The name tf.logging.FATAL is deprecated. Please use tf.compat.v1.logging.FATAL
    instead.

[INFO] audeep.backend.data.data_set - loading data set from spectrograms/dataset1
    -0.08-0.04-128-60.nc
[INFO] audeep.backend.data.eval_tools - label information available - balancing
    classes between folds
[INFO] DataSet - writing data set as netCDF4 to spectrograms/cross-dataset1
    -0.08-0.04-128-60.nc
```

Listing 4.3: Step 2: *Cross Validation*

Therefore once the cross validation is done with the *modify* option we are going to train the autoencoder. More precisely we are going to train a recurrent sequence to sequence autoencoder on the spectrograms extracted in the previous step. Many parameters can be used to fine-tune autoencoder training. In our case we use parameter choices that we have found to work well during our preliminary experiments.
We are going to train an autoencoder with 2 recurrent layers (*num-layers* 2) containing 256 GRU cells (*num-units* 256, GRU is used by default) in the encoder and decoder. The encoder RNN is going to be unidirectional (default setting). When it comes to encoders it has been discovered that depth has not had a bad impact on skill and more surprisingly, a 1-layer unidirectional model performs only slightly worse than a 4-layer unidirectional configuration [25]. While the decoder RNN is going to be bidirectional (*bidirectional-decoder*). The training was carried

out for 128 epochs (*num-epochs* 128) with learning rate 0.001 (*learning-rate* 0.001) and 20% dropout (*keep-prob* 0.8). The batch size during training was chosen 64 (*batch-size* 64).

```
!audeep t-rae train --input spectrograms/cross-dataset1-0.08-0.04-128-60.nc --run-
    name output/dataset1-0.08-0.04-128-60/t-2x256-x-b --num-epochs 128 --batch-size
    64 --learning-rate 0.001 --keep-prob 0.8 --num-layers 2 --num-units 256 --
    bidirectional-decoder
```

The *input* option specifies the spectrogram file which contains training data in this case is **cross-dataset1-0.08-0.04-128-60.nc** which is the dataset after having carried out the cross validation of 20 folds. The *run-name* option specifies a directory for the training run in which models and logging information are stored.

```
[WARNING] tensorflow - From /usr/local/lib/python3.6/dist-packages/audeep/main.py:98:
     The name tf.logging.set_verbosity is deprecated. Please use tf.compat.v1.logging.
    set_verbosity instead.

[WARNING] tensorflow - From /usr/local/lib/python3.6/dist-packages/audeep/main.py:98:
     The name tf.logging.FATAL is deprecated. Please use tf.compat.v1.logging.FATAL
    instead.

[INFO] numexpr.utils - NumExpr defaulting to 2 threads.
[INFO] TrainTimeAutoencoder - created temporary file /tmp/tmpb683lfnx/cross-dataset1
    -0.08-0.04-128-60.nc-0 for data set spectrograms/cross-dataset1-0.08-0.04-128-60.
    nc
[INFO] audeep.backend.data.data_set - loading data set from spectrograms/cross-
    dataset1-0.08-0.04-128-60.nc
[INFO] audeep.backend.data.export - writing data set as TFRecords to /tmp/tmpb683lfnx
    /cross-dataset1-0.08-0.04-128-60.nc-0
[WARNING] TimeAutoencoder - 'decoder_feed_previous_prob' set on bidirectional decoder
     will be ignored. If you have set --decoder-feed-prob 0, or omitted the option,
    the network will behave as expected and you can safely ignore this warning.
[INFO] TimeAutoencoderWrapper - building computation graph
[INFO] GraphWrapper - initializing variables
[INFO] TimeAutoencoderWrapper - preparing input queues
[INFO] TimeAutoencoderWrapper - epoch 1/128, batch 1/5, loss: 0.4403 (4.144 seconds)
[INFO] TimeAutoencoderWrapper - epoch 1/128, batch 2/5, loss: 0.3767 (0.735 seconds)
[INFO] TimeAutoencoderWrapper - epoch 1/128, batch 3/5, loss: 0.3356 (0.609 seconds)
[INFO] TimeAutoencoderWrapper - epoch 1/128, batch 4/5, loss: 0.3236 (0.653 seconds)
[INFO] TimeAutoencoderWrapper - epoch 1/128, batch 5/5, loss: 0.3040 (0.606 seconds)
[INFO] TimeAutoencoderWrapper - epoch 2/128, batch 1/5, loss: 0.2947 (0.565 seconds)
[INFO] TimeAutoencoderWrapper - epoch 2/128, batch 2/5, loss: 0.2872 (0.450 seconds)
[INFO] TimeAutoencoderWrapper - epoch 2/128, batch 3/5, loss: 0.2784 (0.453 seconds)
```

```
[INFO] TimeAutoencoderWrapper - epoch 2/128, batch 4/5, loss: 0.2779 (0.461 seconds)
[INFO] TimeAutoencoderWrapper - epoch 2/128, batch 5/5, loss: 0.2745 (0.453 seconds)
[INFO] TimeAutoencoderWrapper - epoch 3/128, batch 1/5, loss: 0.2640 (0.459 seconds)
[INFO] TimeAutoencoderWrapper - epoch 3/128, batch 2/5, loss: 0.2582 (0.464 seconds)
[INFO] TimeAutoencoderWrapper - epoch 3/128, batch 3/5, loss: 0.2479 (0.457 seconds)
[INFO] TimeAutoencoderWrapper - epoch 3/128, batch 4/5, loss: 0.2539 (0.463 seconds)
[INFO] TimeAutoencoderWrapper - epoch 3/128, batch 5/5, loss: 0.2454 (0.463 seconds)
[INFO] TimeAutoencoderWrapper - epoch 4/128, batch 1/5, loss: 0.2431 (0.439 seconds)
[INFO] TimeAutoencoderWrapper - epoch 4/128, batch 2/5, loss: 0.2376 (0.493 seconds)
[INFO] TimeAutoencoderWrapper - epoch 4/128, batch 3/5, loss: 0.2286 (0.464 seconds)
[INFO] TimeAutoencoderWrapper - epoch 4/128, batch 4/5, loss: 0.2320 (0.464 seconds)
[INFO] TimeAutoencoderWrapper - epoch 4/128, batch 5/5, loss: 0.2270 (0.463 seconds)
[INFO] TimeAutoencoderWrapper - epoch 5/128, batch 1/5, loss: 0.2278 (0.469 seconds)
[INFO] TimeAutoencoderWrapper - epoch 5/128, batch 2/5, loss: 0.2269 (0.458 seconds)
[INFO] TimeAutoencoderWrapper - epoch 5/128, batch 3/5, loss: 0.2210 (0.439 seconds)
[INFO] TimeAutoencoderWrapper - epoch 5/128, batch 4/5, loss: 0.2233 (0.465 seconds)
[INFO] TimeAutoencoderWrapper - epoch 5/128, batch 5/5, loss: 0.2201 (0.454 seconds)
[INFO] TimeAutoencoderWrapper - epoch 6/128, batch 1/5, loss: 0.2171 (0.443 seconds)
[INFO] TimeAutoencoderWrapper - epoch 6/128, batch 2/5, loss: 0.2119 (0.443 seconds)
[INFO] TimeAutoencoderWrapper - epoch 6/128, batch 3/5, loss: 0.2106 (0.448 seconds)
[INFO] TimeAutoencoderWrapper - epoch 6/128, batch 4/5, loss: 0.2055 (0.462 seconds)
[INFO] TimeAutoencoderWrapper - epoch 6/128, batch 5/5, loss: 0.2077 (0.442 seconds)
[INFO] TimeAutoencoderWrapper - epoch 7/128, batch 1/5, loss: 0.2072 (0.452 seconds)
[INFO] TimeAutoencoderWrapper - epoch 7/128, batch 2/5, loss: 0.2041 (0.510 seconds)
[INFO] TimeAutoencoderWrapper - epoch 7/128, batch 3/5, loss: 0.2100 (0.476 seconds)
[INFO] TimeAutoencoderWrapper - epoch 7/128, batch 4/5, loss: 0.2056 (0.451 seconds)
[INFO] TimeAutoencoderWrapper - epoch 7/128, batch 5/5, loss: 0.2040 (0.454 seconds)
[INFO] TimeAutoencoderWrapper - epoch 8/128, batch 1/5, loss: 0.2086 (0.459 seconds)
[INFO] TimeAutoencoderWrapper - epoch 8/128, batch 2/5, loss: 0.2002 (0.458 seconds)
[INFO] TimeAutoencoderWrapper - epoch 8/128, batch 3/5, loss: 0.1989 (0.474 seconds)
[INFO] TimeAutoencoderWrapper - epoch 8/128, batch 4/5, loss: 0.1982 (0.444 seconds)
[INFO] TimeAutoencoderWrapper - epoch 8/128, batch 5/5, loss: 0.1928 (0.461 seconds)
[INFO] TimeAutoencoderWrapper - epoch 9/128, batch 1/5, loss: 0.1964 (0.457 seconds)
[INFO] TimeAutoencoderWrapper - epoch 9/128, batch 2/5, loss: 0.1943 (0.465 seconds)
[INFO] TimeAutoencoderWrapper - epoch 9/128, batch 3/5, loss: 0.1931 (0.515 seconds)
[INFO] TimeAutoencoderWrapper - epoch 9/128, batch 4/5, loss: 0.1942 (0.462 seconds)
[INFO] TimeAutoencoderWrapper - epoch 9/128, batch 5/5, loss: 0.1941 (0.449 seconds)
...........
...........
```

Listing 4.4: Step 3: *Training Autoencoder*

Once that training is finished you can use the trained autoencoder to generate features from spectrograms. If there were no errors using the previous commands we can now execute the following command:

```
!audeep t-rae generate --model-dir output/dataset1-0.08-0.04-128-60/t-2x256-x-b/logs
    --input spectrograms/cross-dataset1-0.08-0.04-128-60.nc --output output/dataset1
    -0.08-0.04-128-60/representations.nc
```

The *model-dir* option specifies the directory containing TensorFlow checkpoints for the trained autoencoder, in our specific case is the *logs* subdirectory of the directory passed to the *run-name* option during autoencoder training. The *input* option specifies the spectrogram file for which we wish to generate features (in this case is **cross-dataset1-0.08-0.04-128-60.nc**), and the *output* option specifies a file where to store the generated features.
This command extracts the learned hidden representation of each spectrogram as its feature vector, and store these features in the output file.

```
[WARNING] tensorflow - From /usr/local/lib/python3.6/dist-packages/audeep/main.py:98:
     The name tf.logging.set_verbosity is deprecated. Please use tf.compat.v1.logging.
    set_verbosity instead.

[WARNING] tensorflow - From /usr/local/lib/python3.6/dist-packages/audeep/main.py:98:
     The name tf.logging.FATAL is deprecated. Please use tf.compat.v1.logging.FATAL
    instead.

[INFO] numexpr.utils - NumExpr defaulting to 2 threads.
[INFO] audeep.backend.data.data_set - loading data set from spectrograms/cross-
    dataset1-0.08-0.04-128-60.nc
[INFO] TimeAutoencoderWrapper - building computation graph
[INFO] GraphWrapper - restoring variables from output/dataset1-0.08-0.04-128-60/t-2
    x256-x-b/logs/model-640
[INFO] TimeAutoencoderWrapper - processed batch 1/1
[INFO] DataSet - writing data set as netCDF4 to output/dataset1-0.08-0.04-128-60/
    representations.nc
```

Listing 4.5: Step 4: *Generate*

Therefore instance labels and a cross-validation setup have been stored, we can now use them to evaluate a simple classifier on the learned representations. The idea is to use the built-in *multilayer perceptron* [8] (MLP) for classification. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron and can distinguish data that is not linearly separable. In the our

[8] https://en.wikipedia.org/wiki/Multilayer_perceptron

case we have built MLP with 4 hidden layers (*num-layers* 4) and 128 hidden units for layer (*num-units* 128). Training was performed for 400 epochs (*num-epochs* 400) with learning rate 0.001 (*learning rate* 0.001) and 20% dropout (*keep-prob* 0.8). No batching is used during MLP training. The *repeat* option repeat evaluation 20 times, and report mean results.

```
!audeep mlp evaluate --input output/dataset1-0.08-0.04-128-60/representations.nc --
    cross-validate --shuffle --num-epochs 400 --learning-rate 0.001 --keep-prob 0.8 --
    num-layers 4 --num-units 128 --repeat 20
```

The *input* option points to a file containing generated features, and while with *cross-validate* option tells the command to perform cross-validated evaluation using the setup stored in that file. The *shuffle* option specifies that the training data should be shuffled between training epochs, which can improve generalization of the network.
The command will print classification accuracy on each cross validation fold, as well as average classification accuracy and a confusion matrix.

```
[WARNING] tensorflow - From /usr/local/lib/python3.6/dist-packages/audeep/main.py:98:
     The name tf.logging.set_verbosity is deprecated. Please use tf.compat.v1.logging.
    set_verbosity instead.

[WARNING] tensorflow - From /usr/local/lib/python3.6/dist-packages/audeep/main.py:98:
     The name tf.logging.FATAL is deprecated. Please use tf.compat.v1.logging.FATAL
    instead.

[INFO] audeep.backend.data.data_set - loading data set from output/dataset1
    -0.08-0.04-128-60/representations.nc
[INFO] CrossValidatedEvaluation - processing cross validation fold 1...
/usr/local/lib/python3.6/dist-packages/xarray/core/dtypes.py:23: FutureWarning:
    Conversion of the second argument of issubdtype from ‘float‘ to ‘np.floating‘ is
    deprecated. In future, it will be treated as ‘np.float64 == np.dtype(float).type‘.

  if np.issubdtype(dtype, float):
/usr/local/lib/python3.6/dist-packages/xarray/core/dtypes.py:25: FutureWarning:
    Conversion of the second argument of issubdtype from ‘int‘ to ‘np.signedinteger‘
    is deprecated. In future, it will be treated as ‘np.int64 == np.dtype(int).type‘.
  elif np.issubdtype(dtype, int):
/usr/local/lib/python3.6/dist-packages/xarray/core/dtypes.py:29: FutureWarning:
    Conversion of the second argument of issubdtype from ‘complex‘ to ‘np.
    complexfloating‘ is deprecated. In future, it will be treated as ‘np.complex128
    == np.dtype(complex).type‘.
  elif np.issubdtype(dtype, complex):
```

```
/usr/local/lib/python3.6/dist-packages/xarray/core/dtypes.py:23: FutureWarning:
    Conversion of the second argument of issubdtype from ‘float‘ to ‘np.floating‘ is
    deprecated. In future, it will be treated as ‘np.float64 == np.dtype(float).type‘.

  if np.issubdtype(dtype, float):
/usr/local/lib/python3.6/dist-packages/xarray/core/dtypes.py:25: FutureWarning:
    Conversion of the second argument of issubdtype from ‘int‘ to ‘np.signedinteger‘
    is deprecated. In future, it will be treated as ‘np.int64 == np.dtype(int).type‘.
  elif np.issubdtype(dtype, int):
/usr/local/lib/python3.6/dist-packages/xarray/core/dtypes.py:29: FutureWarning:
    Conversion of the second argument of issubdtype from ‘complex‘ to ‘np.
    complexfloating‘ is deprecated. In future, it will be treated as ‘np.complex128
    == np.dtype(complex).type‘.
  elif np.issubdtype(dtype, complex):
[INFO] CrossValidatedEvaluation - fold 1 accuracy is 68.75% (UAR 60.28%)
[INFO] CrossValidatedEvaluation - processing cross validation fold 2...
/usr/local/lib/python3.6/dist-packages/xarray/core/dtypes.py:23: FutureWarning:
    Conversion of the second argument of issubdtype from ‘float‘ to ‘np.floating‘ is
    deprecated. In future, it will be treated as ‘np.float64 == np.dtype(float).type‘.

  if np.issubdtype(dtype, float):
/usr/local/lib/python3.6/dist-packages/xarray/core/dtypes.py:25: FutureWarning:
    Conversion of the second argument of issubdtype from ‘int‘ to ‘np.signedinteger‘
    is deprecated. In future, it will be treated as ‘np.int64 == np.dtype(int).type‘.
  elif np.issubdtype(dtype, int):
/usr/local/lib/python3.6/dist-packages/xarray/core/dtypes.py:29: FutureWarning:
    Conversion of the second argument of issubdtype from ‘complex‘ to ‘np.
    complexfloating‘ is deprecated. In future, it will be treated as ‘np.complex128
    == np.dtype(complex).type‘.
  elif np.issubdtype(dtype, complex):
/usr/local/lib/python3.6/dist-packages/xarray/core/dtypes.py:23: FutureWarning:
    Conversion of the second argument of issubdtype from ‘float‘ to ‘np.floating‘ is
    deprecated. In future, it will be treated as ‘np.float64 == np.dtype(float).type‘.

  if np.issubdtype(dtype, float):
/usr/local/lib/python3.6/dist-packages/xarray/core/dtypes.py:25: FutureWarning:
    Conversion of the second argument of issubdtype from ‘int‘ to ‘np.signedinteger‘
    is deprecated. In future, it will be treated as ‘np.int64 == np.dtype(int).type‘.
  elif np.issubdtype(dtype, int):
/usr/local/lib/python3.6/dist-packages/xarray/core/dtypes.py:29: FutureWarning:
    Conversion of the second argument of issubdtype from ‘complex‘ to ‘np.
    complexfloating‘ is deprecated. In future, it will be treated as ‘np.complex128
    == np.dtype(complex).type‘.
  elif np.issubdtype(dtype, complex):
```

```
[INFO] CrossValidatedEvaluation - fold 2 accuracy is 68.75% (UAR 60.28%)
[INFO] CrossValidatedEvaluation - processing cross validation fold 3...
/usr/local/lib/python3.6/dist-packages/xarray/core/dtypes.py:23: FutureWarning:
    Conversion of the second argument of issubdtype from ‘float‘ to ‘np.floating‘ is
    deprecated. In future, it will be treated as ‘np.float64 == np.dtype(float).type‘.

  if np.issubdtype(dtype, float):
/usr/local/lib/python3.6/dist-packages/xarray/core/dtypes.py:25: FutureWarning:
    Conversion of the second argument of issubdtype from ‘int‘ to ‘np.signedinteger‘
    is deprecated. In future, it will be treated as ‘np.int64 == np.dtype(int).type‘.
  elif np.issubdtype(dtype, int):
/usr/local/lib/python3.6/dist-packages/xarray/core/dtypes.py:29: FutureWarning:
    Conversion of the second argument of issubdtype from ‘complex‘ to ‘np.
    complexfloating‘ is deprecated. In future, it will be treated as ‘np.complex128
    == np.dtype(complex).type‘.
  elif np.issubdtype(dtype, complex):
/usr/local/lib/python3.6/dist-packages/xarray/core/dtypes.py:23: FutureWarning:
    Conversion of the second argument of issubdtype from ‘float‘ to ‘np.floating‘ is
    deprecated. In future, it will be treated as ‘np.float64 == np.dtype(float).type‘.

  if np.issubdtype(dtype, float):
/usr/local/lib/python3.6/dist-packages/xarray/core/dtypes.py:25: FutureWarning:
    Conversion of the second argument of issubdtype from ‘int‘ to ‘np.signedinteger‘
    is deprecated. In future, it will be treated as ‘np.int64 == np.dtype(int).type‘.
  elif np.issubdtype(dtype, int):
/usr/local/lib/python3.6/dist-packages/xarray/core/dtypes.py:29: FutureWarning:
    Conversion of the second argument of issubdtype from ‘complex‘ to ‘np.
    complexfloating‘ is deprecated. In future, it will be treated as ‘np.complex128
    == np.dtype(complex).type‘.
  elif np.issubdtype(dtype, complex):
[INFO] CrossValidatedEvaluation - fold 3 accuracy is 68.75% (UAR 57.78%)
[INFO] CrossValidatedEvaluation - processing cross validation fold 4...
..............
..............
```

Listing 4.6: Step 5: *Evaluation*

Below are the results obtained and the related confusion matrix with the dataset_1 in three classes (CON, MCI, eD) without using the data augmentation technique:

```
[INFO] CrossValidatedEvaluation - fold 20 accuracy is 76.92% (UAR 58.33%)
[INFO] MLPEvaluation - cross validation accuracy: 65.23% (+/- 22.64%)
[INFO] MLPEvaluation - cross validation UAR: 56.88% (+/- 26.98%)
[INFO] MLPEvaluation - confusion matrix:
     CON  MCI   eD
CON 2314  400  166
MCI  509 1107  244
 eD  318  342  300

<Figure size 640x480 with 2 Axes>
```

Figure 4.19: *Confusion matrix on dataset_1 (without augmentation)*

Therefore, as an accuracy value with the dataset_1 without augmentation, 65,23% has been reached and is significantly lower than the results seen in the state of the art since it was achieved as the best result on 76% of accuracy value [4] [1].

Experiment on dataset_1 (with augmentation)

As previously seen, the result obtained with the initial dataset using autoencoders did not lead to a good result.For this reason we have decided to use a technique known and used above all for the classification of the images that is the data augmentation. In particular we used a technique explained in the paper Daniel Park et al. [20], called **SpecAugment**. This method increases the size of the dataset by deforming the spectrograms of the audio files through three techniques:

1. *Time warping*: shifts the spectrogram in time by using interpolation techniques to squeeze and stretch the data in a randomly chosen direction.

2. *Time masking*: mask a randomly chosen slice of time steps with the mean value of the spectrogram or zero.

3. *Frequency masking*: mask a randomly chosen band of frequencies with the mean value of the spectrogram or zero.

So we implemented a code able to modify the spectrogram dataset and therefore increasing it with respect to the original. Below we can see the code of the *spec_augment.py*:

```
1 import logging
2 from typing import Union, Sequence, Mapping
3
4 import numpy as np
5 import tensorflow as tf
6 import matplotlib.pyplot as plt
7
8 from audeep.backend.data.data_set import DataSet, Partition, empty
9 from specAugment import spec_augment_tensorflow
```

```
10
11
12 import librosa
13 import librosa.display
14 import tensorflow as tf
15 from tensorflow.contrib.image import sparse_image_warp
16 import numpy as np
17 import matplotlib.pyplot as plt
18
19
20
21
22 def _invert_label_map(label_map: Mapping[str, int]) -> Mapping[int, str]:
23     """
24     Invert the label map of a data set.
25
26     Since we know that label maps are always bijective, no additional checks have to
       be performed.
27
28     Parameters
29     ----------
30     label_map: map of str to int
31         The label map of a data set
32
33     Returns
34     -------
35     map of int to str
36         The inverted label map, which maps numeric label values to nominal label
       values
37     """
38     # noinspection PyTypeChecker
39     return dict(map(reversed, label_map.items()))
40
41
42 def augment(data_set: DataSet,
43             partitions: Union[Partition, Sequence[Partition]] = None) -> DataSet:
44     """
45     Generate new data to balance classes in the specified partitions of the specified
        data set.
46
47     If ‘partitions‘ is set, instances in the specified partitions are repeated so
       that each class has approximately the
48     same number of instances. Any partitions present in the data set, but not
       specified as parameters to this function
```

```
    are left unchanged.

    If ‘partitions‘ is empty or None, the entire data set is upsampled.

    If an instance is upsampled, the string "upsampled.I", where I indicates the repetition index, is appended to the
    filename.

    Parameters
    ----------
    data_set: DataSet
        The data set in which classes should be balanced
    partitions: Partition or list of Partition
        The partitions in which classes should be balanced
    Returns
    -------
    DataSet
        A new data set in which the classes in the specified partitions are balanced
    """
    print("Begin Upsampling")

    log = logging.getLogger(__name__)

    if isinstance(partitions, Partition):
        partitions = [partitions]

    inverse_label_map = _invert_label_map(data_set.label_map)

    if partitions is None:
        keep_data = None
        upsample_data = data_set

        log.debug("upsampling entire data set")
    else:
        partitions_to_keep = [x for x in Partition if x not in partitions]

        # noinspection PyTypeChecker
        log.debug("upsampling partition(s) %s, keeping partition(s) %s", [x.name for x in partitions],
                  [x.name for x in partitions_to_keep])

        keep_data = None if not partitions_to_keep else data_set.partitions(partitions_to_keep)

```

```
90          if keep_data is not None:
91              upsample_data = data_set.partitions(partitions)
92          else:
93              upsample_data = data_set
94
95      labels = upsample_data.labels_numeric
96      unique, unique_count = np.unique(labels, return_counts=True)
97
98
99      upsample_factors = [2,2,2]
100
101     print('Upsample factors:',upsample_factors)
102
103     num_instances = (0 if keep_data is None else keep_data.num_instances) + np.sum(
        upsample_factors * unique_count)
104
105     print('Number of instances:',num_instances)
106
107     log.info("upsampling with factors %s for labels %s, resulting in %d instances
        total", upsample_factors,
108              [inverse_label_map[x] for x in unique], num_instances)
109
110     print('upsampling with factors %s for labels %s, resulting in %d instances total',
         upsample_factors,
111              [inverse_label_map[x] for x in unique], num_instances)
112
113     upsample_map = dict(zip(unique, upsample_factors))
114
115     # noinspection PyTypeChecker
116     new_data = empty(num_instances, list(zip(data_set.feature_dims, data_set.
        feature_shape)), data_set.num_folds)
117     new_data.label_map = data_set.label_map
118
119     new_index = 0
120
121     if keep_data is not None:
122         # just copy instances we are not upsampling
123         for index in keep_data:
124             new_instance = new_data[new_index]
125             old_instance = keep_data[index]
126
127             new_instance.filename = old_instance.filename
128             new_instance.chunk_nr = old_instance.chunk_nr
129             new_instance.label_nominal = old_instance.label_nominal
```

```
            new_instance.cv_folds = old_instance.cv_folds
            new_instance.partition = old_instance.partition
            new_instance.features = old_instance.features

            new_index += 1

    for index in upsample_data:
        old_instance = upsample_data[index]

        print("Primo step")

        for i in range(upsample_map[old_instance.label_numeric]):
            # repeat instance according to upsampling factor for the respective class
            new_instance = new_data[new_index]

            new_instance.filename = old_instance.filename + ".upsampled.%d" % (i + 1)
            new_instance.chunk_nr = old_instance.chunk_nr
            new_instance.label_nominal = old_instance.label_nominal
            new_instance.cv_folds = old_instance.cv_folds
            new_instance.partition = old_instance.partition
            print("Secondo step")
            new_instance.features = spec_augment_tensorflow.spec_augment(old_instance.
    features, time_warping_para=10, frequency_masking_para=10, time_masking_para=10,
    frequency_mask_num=1, time_mask_num=1)
            print("Terzo step")

            new_index += 1


    return new_data



from pathlib import Path

from cliff.command import Command

from audeep.backend.data.data_set import Partition, load
from audeep.backend.data.upsample import upsample
from audeep.backend.enum_parser import EnumType

data_set = load(Path("/home/devid29/Scrivania/SpecAugment/in/dataset1
    -0.08-0.04-128-60.nc")) # initial dataset with three classes

```

```
upsampled_data_set = augment(data_set, None)
print(upsampled_data_set)
upsampled_data_set.save(Path("/home/devid29/Scrivania/SpecAugment/out/augmentttt-
    dataset1-0.08-0.04-128-60.nc")) #augment dataset with three classes
```

Listing 4.7: spec_augment.py

This code was executed locally and not on Google Colab because there were some problems with some libraries. Once the code is executed the new increased dataset is generated:

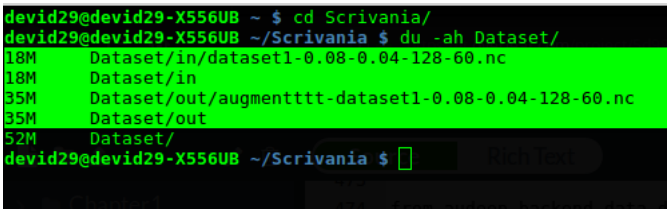

Figure 4.20: *Comparison on the size of the original dataset_1 and dataset_1 augment*

As you can see from the figure above the size of the original 18M dataset becomes 35M thanks to the specAugment method.
After increasing the dataset, we performed the same experiment that was done with the original dataset.

```
[INFO] CrossValidatedEvaluation - fold 20 accuracy is 88.89% (UAR 84.26%)
[INFO] MLPEvaluation - cross validation accuracy: 86.98% (+/- 13.16%)
[INFO] MLPEvaluation - cross validation UAR: 83.28% (+/- 19.89%)
[INFO] MLPEvaluation - confusion matrix:
     CON  MCI   eD
CON 5401  246  113
MCI  435 3131  154
 eD  290  241 1389

<Figure size 640x480 with 2 Axes>
```

Figure 4.21: *Confusion matrix augment dataset_1*

So after doing the new experiment with the new dataset augment we were able to arrive at an accuracy of **86.98%** clearly better than the previous experiment but above all better than the state of the art [4] [1].

	Methods	**Accuracy**	**Recall**
3 classes (CON, MCI, eD)	*Autoencoder*	65.23%	56.88%
	Autoencoder + Augmented dataset	**86.98%**	**83.28%**

Table 4.1: Table with results on dataset_1 with 3 classes (CON, MCI, eD) between original dataset and augment dataset

4.3.3 Experiment on two classes (CON, DEC)

After performing the experiment on three classes Control (CON), Mild Cognitive Impairment (MCI) and early Dementia, we performed the same experiment but only two classes of subjects: *Control* (CON) therefore subjects not affected by cognitive decline and *Decline* (DEC) those subject to cognitive decline.

Experiment on dataset_2 (without augmentation)

First of all, let's extract the spectrograms and some metadata from the raw audio files. We use the command *inspect raw*:

```
!audeep inspect raw  --basedir .../dataset_2 --parser audeep.backend.parsers.
    partitioned.PartitionedParser
```

Immediately after we go to preprocess the audio extracting the spectrograms with the same parameters used in the previous experiment with three classes:

```
!audeep preprocess --basedir .../dataset\_2 --parser audeep.backend.parsers.
    partitioned.PartitionedParser --window-width 0.16 --window-overlap 0.08 --mel-
    spectrum 256 --fixed-length 5 --clip-below -60 --output spectrograms/dataset2
    -0.08-0.04-128-60.nc
```

Then the spectrograms are stored in the netCDF 4 format in a file called **dataset2-0.08-0.04-128-60.nc**.
Before training the autoencoder we execute the *modify* command to modify the metadata of the dataset. In this case we used a cross validation with 20 folds.

```
!audeep modify --input spectrograms/dataset2-0.08-0.04-128-60.nc --output
    spectrograms/cross-dataset2-0.08-0.04-128-60.nc --add-cv-setup 20
```

Now let's train the autoencoder identical to the one used in the previous experiment and with the same parameters.

```
!audeep t-rae train --input spectrograms/cross-dataset2-0.08-0.04-128-60.nc --run-
    name output/dataset2-0.08-0.04-128-60/t-2x256-x-b --num-epochs 128 --batch-size
    64 --learning-rate 0.001 --keep-prob 0.8 --num-layers 2 --num-units 256 --
    bidirectional-decoder
```

Once that training is finished you can use the trained autoencoder to generate features from spectrograms.

```
!audeep t-rae generate --model-dir output/dataset2-0.08-0.04-128-60/t-2x256-x-b/logs
    --input spectrograms/cross-dataset2-0.08-0.04-128-60.nc --output output/dataset2
    -0.08-0.04-128-60/representations.nc
```

Once instance labels and a cross-validation setup have been stored, we can now use them to evaluate a simple classifier on the learned representations.

```
!audeep mlp evaluate --input output/dataset2-0.08-0.04-128-60/representations.nc --
    cross-validate --shuffle --num-epochs 400 --learning-rate 0.001 --keep-prob 0.8 --
    num-layers 4 --num-units 128 --repeat 20
```

Below are the results obtained and the related confusion matrix with the dataset_2 in two classes (CON, DEC) without using the data augmentation technique:

```
[INFO] CrossValidatedEvaluation - fold 20 accuracy is 84.62% (UAR 85.71%)
[INFO] MLPEvaluation - cross validation accuracy: 77.26% (+/- 19.85%)
[INFO] MLPEvaluation - cross validation UAR: 77.35% (+/- 19.89%)
[INFO] MLPEvaluation - confusion matrix:
     CON  DEC
CON 2246  614
DEC  665 2075

<Figure size 640x480 with 2 Axes>
```

Figure 4.22: *Confusion matrix on dataset_2 (without augmentation)*

Therefore, as an accuracy value with the dataset_2 without augmentation, 77.26% has been reached. This result, however, is lower than the state of the art [1] [3] [2] which carries out the same experiment dividing the dataset into two classes as in ours, arriving at 81% accuracy.

Experiment on dataset_2 (with augmentation)

Therefore to improve our accuracy result we tried to use the same technique as previously seen, that of the SpecAugment [20].
Here too we can reuse the same code used for the three classes by modifying just a few lines of code as we see below:

```
"""
ATTENTION: Change the variable upsample_factors = [2,2,2] with upsample_factors =
    [2,2] because in this case we have only two sampling classes.

```

```
upsample_factors = [2,2,2] -----> upsample_factors = [2,2]

"""

from pathlib import Path

from cliff.command import Command

from audeep.backend.data.data_set import Partition, load
from audeep.backend.data.upsample import upsample
from audeep.backend.enum_parser import EnumType

data_set = load(Path("/home/devid29/Scrivania/SpecAugment/in/dataset2
    -0.08-0.04-128-60.nc")) # initial dataset with two classes

upsampled_data_set = augment(data_set, None)
print(upsampled_data_set)
upsampled_data_set.save(Path("/home/devid29/Scrivania/SpecAugment/out/augmentttt-
    dataset2-0.08-0.04-128-60.nc")) #augment dataset with two classes
```

Listing 4.8: spec_augment.py

The only changes to be made are the paths to the dataset folder and the "*upsample_factors*" variable present in the "*augment*" function inside the *specAugment.py* file which must be changed from three to two classes. Then the code can be executed and if there are no errors the new increased dataset will be returned.

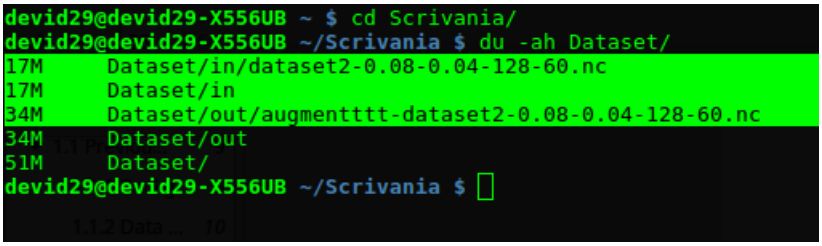

Figure 4.23: *Comparison on the size of the original dataset_2 and dataset_2 augment*

As you can see from the figure above the size of the original 17M dataset becomes 34M thanks to the specAugment method.
After increasing the dataset_2, we performed the same experiment that was done with the original dataset.

```
[INFO] CrossValidatedEvaluation - fold 20 accuracy is 96.30% (UAR 96.15%)
[INFO] MLPEvaluation - cross validation accuracy: 90.57% (+/- 10.76%)
[INFO] MLPEvaluation - cross validation UAR: 90.56% (+/- 10.76%)
[INFO] MLPEvaluation - confusion matrix:
     CON  DEC
CON 5224  496
DEC  561 4919

<Figure size 640x480 with 2 Axes>
```

Figure 4.24: *Confusion matrix augment dataset_2*

So after doing the new experiment with the new dataset augment we were able to arrive at an accuracy of **90.57%** clearly better than the previous experiment but above all better than the state of the art [1] [3] [2].

	Methods	**Accuracy**	**Recall**
2 classes (CON, DEC)	*Autoencoder*	77.26%	77.35%
	Autoencoder + Augmented dataset	**90.57%**	**90.56%**

Table 4.2: Table with results on dataset_2 with 2 classes (CON, DEC) between original dataset and augment dataset

4.4 Results

In summary we can see from the tables below the results obtained by us and those of the state of the art. In the case of the three classes Controls (CON), Mild Cognitive Impairment (MCI) and early dementia (eD) the results were the following:

	Method	***Accuracy***	***Recall***	***Precision***	***F1***
Manual Extraction	K Nearest Neighbour	72.00%	70.00%	72.10%	71.70%
	Logistic Regression	75.00%	76.00%	74.40%	75.30%
	Multi Layer Perceptron	76.00%	75.00%	76.70%	75.90%
	Support Vector Machine	61.30%	n.a	n.a	n.a
Automatic Extraction	*Support Vector Machine*	58.7%	n.a	n.a	n.a
	Autoencoder	65.23%	56.88%	58.59%	57.49%
	Autoencoder + Augmented dataset	**86.98%**	**83.28%**	**86.19%**	**84.63%**

Table 4.3: Comparison of the state of the art and our experiment on three classes (CON,MCI,eD)

Our result with the auto-encoders is worse than almost all the results of the state of the art [4] [1] (green color) but with the addition of the specAugment the accuracy improves decisively with 86% and gets to overcome the state of the art.
The same result was carried out as already explained above also on the two classes Control (CON) and Decline (DEC), we now see in the table below the comparison with the results of the state of the art:

	Method	***Accuracy***	***Recall***	***Precision***	***F1***
Manual Extraction	Logistic Regression	81.92%	n.a	n.a	n.a
	Support Vector Machine	76.00%	80.00%	83.30%	81.6%
Automatic Extraction	*Support Vector Machine*	73.30%	72.00%	85.70%	78.30%
	Deep Sequential Neural Network	83.00%	n.a	n.a	n.a
	Autoencoder	77.26%	77.35%	77.16%	76.44%
	Autoencoder + Augmented dataset	90.57%	90.56%	90.84%	90.30%

Table 4.4: Comparison of the state of the art and our experiment on two classes (CON, DEC)

As expected the result on two classes is clearly better. Initially the result is quite good but with the help of the specAugment it is clearly improved compared to the state of the art [1] [2] [3].

Conclusions

This thesis has had as its main purpose to find a different solution to try to detect the disease of cognitive decline early through the use of audio files. In particular we started from the work carried out by Calzà et al. [4] where it was carried out manually in more detail some precise and manually transcribed features were extracted from the audio files. Finally the extracted features were analyzed using specific software. From our point of view this method has some points to the detriment that made us think of trying another method. One of the most important problems we found was that it was not a standard method and above all not scalable, besides the fact that the use of the transcribed text can lead to a loss of information of the original data. Our proposal was to use instead the transcribed texts the spectrograms of the audio files and extract the features automatically using a particular neural network called *Autoencoder*. So in our approach we extract the spectrograms from all the audio files and train an autoencoder to represent the data in the analysis.
The analysis was performed using the *auDeep* software that trains an autoencoder for extracting features from spectrograms and their classification. In addition to this we have tried to expand our starting dataset using a data augmentation technique that modifies the spectrograms of the audio files called *specAugment*. So our work later was to test this method and the result was pretty good. In particular, both the starting dataset and the one increased through the specAugment were tested and it was found that with a larger dataset the results increase a lot. Therefore we can assume that with a greater dataset the autoencoder technique would have a great potential for improvement. As future developments we could try to use another type of autoencoder with respect to the one that was used for this thesis that maybe could work even better and certainly a greater dataset.

Bibliography

[1] Gábor Gosztolya, Veronika Vincze, László Tóth, Magdolna Pákáski, János Kálmán, and Ildikó Hoffmann. Identifying mild cognitive impairment and mild alzheimer's disease based on spontaneous speech using asr and linguistic features. *Computer Speech & Language*, 53:181–197, 2019.

[2] Charalambos Themistocleous, Marie Eckerström, and Dimitrios Kokkinakis. Identification of mild cognitive impairment from speech in swedish using deep sequential neural networks. *Frontiers in neurology*, 9, 2018.

[3] Kathleen C Fraser, Jed A Meltzer, and Frank Rudzicz. Linguistic features identify alzheimer's disease in narrative speech. *Journal of Alzheimer's Disease*, 49(2):407–422, 2016.

[4] Daniela Beltrami, Laura Calzà, Gloria Gagliardi, Enrico Ghidoni, Norina Marcello, Rema Rossini Favretti, and Fabio Tamburini. Automatic identification of mild cognitive impairment through the analysis of italian spontaneous speech productions. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 2086–2093, 2016.

[5] Daniela Beltrami, Gloria Gagliardi, Rema Rossini Favretti, Enrico Ghidoni, Fabio Tamburini, and Laura Calzà. Speech analysis by natural language processing techniques: a possible tool for very early detection of cognitive decline? *Frontiers in Aging Neuroscience*, 10:369, 2018.

[6] alzheimer's association. Mild Cognitive Impairment (MCI). https://www.alz.org/alzheimers-dementia/what-is-dementia/related_conditions/mild-cognitive-impairment, 2019. Online; 2019.

[7] Francois Chollet. Building Autoencoders in Keras. https://blog.keras.io/building-autoencoders-in-keras.html, 2016. Online; 14 May 2016.

[8] Nathan Hubens. Deep inside: Autoencoders. https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f, 2018. Online; 25 February 2018.

[9] Jannik Zürn. But what is an Autoencoder? https://medium.com/@jannik.zuern/but-what-is-an-autoencoder-26ec3386a2af, 2019. Online; 24 February 2019.

[10] Khandelwal R. Deep Learning - Different Types of Autoencoders. https://medium.com/datadriveninvestor/deep-learning-different-types-of-autoencoders-41d4fa5f7570, 2018. Online; 2 December 2018.

[11] Jonnalagadda V. Sparse, stacked and variational autoencoder. https://medium.com/@venkatakrishna.jonnalagadda/sparse-stacked-and-variational-autoencoder-efe5bfe73b64, 2018. Online; 6 December 2018.

[12] Joseph Rocca. Understanding Variational Autoencoders (VAEs). https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73, 2019. Online; 24 September 2019.

[13] Cheng Lei, Benlin Hu, Dong Wang, Shu Zhang, and Zhenyu Chen. A preliminary study on data augmentation of deep learning for image classification. In *Proceedings of the 11th Asia-Pacific Symposium on Internetware*, page 20. ACM, 2019.

[14] Jason Brownlee. How to Configure Image Data Augmentation in Keras (Deep Learning for Computer Vision). https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/, 2019. Online; 12 April 2019.

[15] Arun Gandhi. Data augmentation | how to use deep learning when you have limited data — part 2. https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/, 2018. Online; November 2018.

[16] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. *arXiv preprint arXiv:1511.06709*, 2015.

[17] Zhenhao Li and Lucia Specia. Improving neural machine translation robustness via data augmentation: Beyond back translation. *arXiv preprint arXiv:1910.03009*, 2019.

[18] Jason W Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*, 2019.

[19] Daniel S. Park, AI Resident and William Chan, Research Scientist. SpecAugment: A New Data Augmentation Method for Automatic Speech Recognition . https://ai.googleblog.com/2019/04/specaugment-new-data-augmentation.html, 2019. Online; 22 April 2019.

[20] Daniel S Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D Cubuk, and Quoc V Le. Specaugment: A simple data augmentation method for automatic speech recognition. *arXiv preprint arXiv:1904.08779*, 2019.

[21] Zach C. State of the art audio data augmentation with google brain's specaugment and pytorch. https://towardsdatascience.com/state-of-the-art-audio-data-augmentation-with-google-brains-specaugment-and-pytorch-d3d1a3ce291e, 2019. Online; 1 May 2019.

[22] Rakshith Ponnappa. Google Colab: Using GPU for Deep Learning. https://python.gotrained.com/google-colab-gpu-deep-learning/, 2019. Online; 27 January 2019.

[23] Anne Bonner. Getting Started With Google Colab. https://towardsdatascience.com/getting-started-with-google-colab-f2fff97f594c, 2019. Online; 1 January 2019.

[24] Michael Freitag, Shahin Amiriparian, Sergey Pugachevskiy, Nicholas Cummins, and Björn Schuller. audeep: Unsupervised learning of representations from audio with deep recurrent neural networks. *The Journal of Machine Learning Research*, 18(1):6340–6344, 2017.

[25] Jason Brownlee. How to configure an encoder-decoder model for neural machine translation. https://machinelearningmastery.com/configure-encoder-decoder-model-neural-machine-translation/, 2019. Online; 7 August 2019.

Ringraziamenti

Prima di tutto vorrei ringraziare i miei genitori che mi hanno permesso di intraprendere questo percorso che ora si sta concludendo. Un ringraziamento speciale va sicuramente a mia mamma Rita che soprattutto in questi ultimi mesi mi è stata vicino e mi ha aiutato a concludere questo persorso di studi nonostante la perdita di mio padre avvenuta poco tempo fa. Ovviamente un rigraziamento speciale va anche a mio padre che nonostante il suo ultimo periodo di vita difficile non ha mai smesso di supportarmi in tutto. Poi vorrei ringraziare anche tutti i miei familiari partendo dagli zii e finendo per tutti i miei cugini che mi sono stati sempre vicini.
Non posso non ringraziare la mia squadra di calcio a 5 il "Drinking Team" con la quale abbiamo passato e spero di passare ancora tante altre esperienze insieme. In particolare la vecchia guardia con la quale abbiamo creato la squadra ma sopratutto abbiamo creato una famiglia dove neanche la distanza ci fa sentire lontani.
Ringrazio anche i miei coinquili ed ex-coinquilini di San Felice che mi hanno permesso di percorrere serenamente il mio percorso sia fuori che dentro l'università.
Ringrazio i ragazzi di Sant'Ilario e Parma che mi hanno sempre fatto sentire a casa e che mi sono stati vicini nei momenti più difficili. Ah già dimenticavo saluto anche il gruppo del momento i "Century Light" composto da Fabio e Mattia due grandi persone ma sopratutto due grandi artisti che meritano le migliori fortune.
Ringrazio infine il gruppo "Giorgio" il gruppo di persone che conosco da 27 anni, dove nemmeno le distanze ci allontanano ma anzi ci rafforzano e ci fanno rimanere uniti.
Un grande ringraziamento va assolutamente al prof. Montesi e Dott. Flavio Bertini per avermi seguito nel periodo di tesi e non solo. Voglio ringraziare tutte le persone che mi sono dimenticato di citare ma che sono stati importanti come gli altri.
Concludo con una frase detta da un mio amico caro una sera: "*Dopo tre anni finalmente riusciamo ad andare alla laurea di Davide Allevi con Davide Allevi...*" cit M.I.