LUISS Guido Carli
LIBERA UNIVERSITÀ INTERNAZIONALE DEGLI STUDI SOCIALI

Business and Management department
Management and Computer Science

Deep Learning Methods for Speech-to-Text Systems

SUPERVISOR
Prof. Marco Querini

CANDIDATE
Jacopo Giannetti
232431

Academic Year 2020/2021

Deep Learning Methods for Speech-to-Text Systems
Jacopo Giannetti
Department of Business and Management
Luiss Guido Carli

Abstract

Machine learning is one of the most interesting innovations of the twenty-first century. Born from the neuroscience studies of the early forties and evolved over time, it has become what most similar to an artificial intelligence humanity has been able to create. Machine Learning is based on complex structures called neural networks, which mimic the functioning of neurons in our brain combined with optimization algorithms to allow a machine to learn from the observation of results. This thesis explores one of the many uses of these structures that is the study of the voice, focusing on speech recognition methods and analyzing their functioning in details.

CONTENTS

“Nobody phrases it this way, but I think that artificial intelligence is almost a humanities discipline. It's really an attempt to understand human intelligence and human cognition.”

Sebastian Thrun

1

Introduction

This thesis deals with the analysis of the current status, progress and major criticalities of a Speech-to-Text translation carried out through the use of Artificial Neural Networks. From self-driving cars to systems capable of understanding and analyzing brain neuronal activations, Artificial Intelligence has indeed made incredible progress in the last decade, principally thanks to the large availability of data related to the digitalization of our lives. Of utmost importance, in this modern society that through digitization pushes us to break the barriers that separate individuals, is the ability to translate the human voice into a state that can be understood and analyzed by a computer. This could open the doors for countless possible applications that arise from the study, analysis and understanding of human voice. In particular, through the use of neural networks, it is possible to create systems capable of understanding and modelling the human voice in an increasingly accurate and reliable way, making transcription, translation and real-time analysis of human language a reality that is day by day more concrete.

1.1 Motivation

Given the incredible implications that the application of neural networks is having on our society and given the increasingly recurrent use of these techniques for the analysis of the human voice, which seems to improve exponentially over time with the technological advancement, this thesis aims to analyze and confront the current methods used in deep learning for speech recognition, through the study, the analysis and experimentation on different models. For this analysis, I decided to study some of the most advanced system and works available at the moment, in order to understand what are the weakness and the strong point of this type of voice modelling. In the end there will be also a practical experimentation, in which I built a model able to recognize some keyword thorough the use of a convolutional neural network.

1.2 Methodology

I started my work by analyzing the current state of the art of those systems, looking at the most advanced network aimed at speech recognition developed by some of the biggest company on the planet. All of these systems have very complex implementation and are trained through huge dataset, which require extreme computational power and efficient algorithms in order to be processed. This research on the state of the art is described in chapter 2.

Chapter 3, instead, will be a very detailed explanation of the background behind the functioning of neural networks. In the first part of the chapter is going to be described the history of neural networks, form the first perceptron to more advanced models. The functioning of these models will then be analyzed in detail, with clear description of their functioning and of all the technique used for their training and modelling. In the final part of the chapter are also explained some of the techniques used in particular for the study of the voice.

Chapter four will be a detailed description of two of the most interesting works that I researched and study, involving a Speech-to-text system and a more complex model that deals with direct Speech-to-Speech translation.

The study of these models is very useful to fully understand the possible application and uses of neural networks for the study of the voice, and also a very interesting way to analyze and observe important experimentation made on these models by their creators, in order to see and understand in detail, their functioning and structure.

In chapter 5 will be described instead a practical application of a speech recognition system that I decided to build, try and experiment on my own. The code, entirely written using Python, inspired by the work of Valerio Velardo (1), through the use of Convolutional Neural Network is able to recognize a spoken word out of a list of ten possible. In chapter 5.4.2 the model will also be expanded to handle 35 different words. The program is able to generate an extremely efficient network, which after being trained on a dataset containing about 3,800 audio files for each of the 10 words, for a total of 38.000 audio files (increased to 133.000 during my experimentation) is able to predict one of these with considerable certainty and accuracy. After the creation and description of the model I also decide the make some experiments on it, several different models have been created by varying the hyperparameters in order to optimize the model in the most appropriate way. This experimentation part will also be described in chapter 5 (5.4).

The last chapter will instead contain the conclusion of this work.

2

State of the Art

In this chapter we analyze related work to our problem. If the purpose of this thesis is a practical comparison between models that use convolutional neural network and recurrent neural network to carry out Speech to text translation tasks, in this part we will describe the best systems on earth that deal with this problem.

2.1 Systems

In this section we introduce some of the best software systems capable of accomplishing a Speech to text translation. Many different companies are developing speech recognition technology today. Tech giants like Google, Microsoft and IBM are at the forefront of bringing these technologies to their platforms and services. At the same time, they offer the technology to use as part of other companies' solutions. Most of today's automated transcription services are based on one of the aforementioned technologies from the tech giants[1].

Google API

Google's main interests in the development of speech recognition technology are voice input for mobile phones, voice search on the desktop and transcription and translation of YouTube, which are the major platforms in which the giant operates.
The impressive technological development, in particular the development of neural networks has brought immense improvements in the Google system, which as a pioneer of this new technology can now be considered one of the greatest exponents in the field of speech to text, with the greatest part of the "home-made" projects that rely on its

[1]Bohouta, Gamal & Këpuska, Veton (2017). "Comparing Speech Recognition Systems (Microsoft API, Google API And CMU Sphinx)". *Journal of Engineering Research and Application. 2248-9622. 20-24.* .

API. To get the impression of the improvement brought by google in the world of speech to text systems, one can think that the error rate of the Google API was 8% in 2015, 23% lower than in 2013[2]. According to Pichai, senior vice president of Android, Chrome, and Apps at Google, “We have the best investments in machine learning over the past many years. Indeed, Google has acquired several deep learning companies over the years, including DeepMind, DNNresearch, and Jetpac ".

Microsoft API

Microsoft develops its voice API primarily to implement the technology in its Windows operating system. Microsoft recently announced that its speech recognition technology has also reached the 95% artificial transcription accuracy threshold. It is an interesting fact about Microsoft Speech API that the development of the technology started as early as 1993. Microsoft has focused with a growing emphasis on speech recognition systems and has improved its Speech API (SAPI) by using a context-dependent Deep Neural Network Hidden Markov Model (Bohouta, Gamal & Këpuska, Veton, 2017). Just recently Microsoft announced: "Historic Achievement: Microsoft Researchers Achieve Human Parity in Conversational Speech Recognition" (2).

CMU Sphinx

CMU Sphinx is the generic term to describe a group of speech recognition systems developed at Carnegie Mellon University. These include a set of speech recognizers and an acoustic model trainer. Currently "CMU Sphinx has an extensive vocabulary, speaker independent speech recognition code base and its code is available for download and use"[3].

Silero

Silero is a pre-trained enterprise grade speech to text and text to speech model.
It provides one of the best STT models in circulation which, the company said, would achieve similar, if not superior, performance to the models offered by Google. Since the model is pre trained there is not much experimentation possible to be executed on it.

[2] Filippidou F, Moussiades L. (2020) “A Benchmarking of IBM, Google and Wit Automatic Speech Recognition Systems, Artificial Intelligence Applications and Innovations”.

[3] Carnegie Mellon University (2019). “CMUSphinx Tutorial for Developers”.

2.2 Python libraries

Tensorflow

TensorFlow is an end-to-end open source platform designed for machine learning. The Tensorflow library contains functions capable of executing neural networks in the most precise and efficient way possible. It also has excellent documentation and a good set of examples that make it one of the best choices for this type of project. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications (3). The majority of the home-made systems that execute a simple STT task, as the project analyzed for this work, are usually built with the help of Tensorflow 2.

Pythorch

PyTorch is a Python-based computing package that uses the power of graphics processing units. GPUs are in fact optimized for training artificial intelligence and deep learning models as they can process multiple computations at the same time. This is due to the fact that they have many cores, which allows for better computation of multiple parallel processes. PyTorch is one of the most widely used deep learning research platforms, built to provide maximum flexibility and speed. Given its affinity in the use of RNN, in the second project treated in this thesis PyTorch was chosen as the reference library

Keras

With the increasing number of users entering the world of deep nerual networks, more and more common in the functioning of programs in today's society, the complexity of the latter has always been a barrier to new programmers. There have been many proposals for simplified APIs for modeling neural networks, and Keras is one of them. Keras is a leading high-level neural network API written in Python that supports multiple back-end neural network computation engines. It has been chosen as the official TensorFlow 2 high level API.

Scikit-Learn

Scikit-learn is a Python library that provides many unsupervised and supervised learning algorithms. It uses other very useful libraries such as NumPy, pandas and Matplotlib.
Features provided by scikit-learn includes: Regression (such as Linear and Logistic Regression), Classification (such as K-Nearest Neighbors), Clustering (such as K-Means and K-Means), Model Selection and Pre-Processing (such as Normalization Min-Max)

3

Background

In this section we introduce the History, the theory and the recent advancements within the machine learning area that make a speech to text translation system possible. We start by describing what is are neural networks and how do they work, while towards the end we describe with more specificity the techniques used in particular for a speech to text (STT) system.

3.1 The history of Artificial Neural Networks

The success of artificial neural networks has undoubtedly sparked a revolution in the world of artificial intelligence in the last ten years.
What many people don't know, however, is that the basic idea of neural networks has been around since 1950s. In 1957, Frank Rosenblatt invented the perceptron, a type of neural network where binary neurons units are connected via adjustable weights.
Rosenblatt was inspired by the work of neuroscience in 1940s which led him to create a crude replication of the neurons in the brain. Although the idea was theoretically revolutionary and despite the countless efforts to find more efficient model layouts or better learning algorithms, the computational power of the computers of the time did not allow Rosenblatt's perceptron to obtain convincing results. Strongly convinced of his idea, Rosenblatt did not give up easily, and tried to build a real machine, with a similar functioning to that of a computer, built solely for the purpose of making the hypothesized perceptron work.
The machine was made up of adjustable resistors controlled by small motors that turned the resistors off and on while the machine "learned". When completed, Rosenblatt's machine was able to classify images of simple shapes or letters. The New York Times of July 13, 1958, reported the discovery with these words (4):

"The Navy last week demonstrated the embryo of an electronic computer named the Perceptron which, when completed in about a year, is expected to be the first non-living mechanism able to "perceive, recognize and identify its surroundings without human training or control."

The New York Times Archives, July 13, 1958
(Section E, Page 9)

In 1969 the book "Perceptrons: an introduction to computational geometry", by Marvin Minsky and Seymour Papert[4], was published, it was a harsh critic on Rosenblatt's perceptrons, demonstrating that perceptrons sometimes failed even on extremely simple task. At the time it was already clear that the conjunction of multiple levels to the Rosenblatt model would have allowed not only to significantly improve the reliability of the results of the system, but also to considerably increase the complexity of the tasks that the perceptron was able to perform. Still, at the time, there was no algorithm capable of training such a network, and the project was abandoned for several years. It took seventeen years until such an algorithm, now known as "back-propagation" was deviced. The new algorithm, invented by Rumelhart, D., Hinton, G. and Williams[5], which theoretically allowed neural networks to approximate any function, ensured that in the field of artificial intelligence, great things were about to happen. However, despite the great revolution that seemed to have changed things, after a short time most scholars moved back to the study of other methods, such as Support Vector Machine (SVM) or Bayesian style methods, since neural networks, for some unknown reasons, still did not give the expected results.
The answer arrived several years later, the problem that prevented neural networks from functioning as expected seemed evident: the lack of data and computation power. For reference, at the beginning of 2002 the total amount of data produced worldwide per year was estimated to be approximately 5 Exabytes, while the data generated annually in the year 2019 is estimated to be 10 Zetabytes, an increase with the factor of 2000[6].

[4] Minsky, M., Papert, S. (1969). "Perceptrons: An Introduction to Computational Geometry". Cambridge, MA, USA: MIT Press.

[5] Rumelhart, D., Hinton, G. & Williams, R. (1986). "Learning representations by back-propagating errors". *Nature* 323, 533–536.

[6] Peter Lyman and Hal R. (2003). "*How much information?* coordinator: Kirsten Swearingen, School of Information Management and Systems at the University of California at Berkeley.

In 2012, the annual ILSVRC-2012 competition was won for the first time using a neural network, achieving significantly better results than the runner-up[7].

"In the ILSVRC-2012 competition we achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry"

Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton

Computer capability and data availability allowed deep neural networks to outperform classical approaches in many different fields, paving the way for a future in which artificial intelligence will increasingly become the basis in the functioning of our society.

3.2 Neural networks

The basis of this thesis largely depends on the subfield within machine learning called artificial neural networks, generally referred to as neural networks. In 1989, Dr. Robert Hecht-Neilsen defined neural networks as "... an accomplished computer system composed of a series of simple and highly interconnected processing elements, which process information based on their dynamic state of response to external inputs"[8].
Figure 1 show an example of a simple artificial neural network, and it will be used in this section in order to give an idea of the general structure of such a network.

There are three type of layer that form a neural network as seen in Figure 3.1. The input layer is formed by the nodes which we feed our input data into. The output layer contains the final nodes of the network and gives the output generated by the network based on the given input in the input layer[9]. For an SST system the input that the input layer receives is an audio file while the output of the network is transcription of the audio file. The middle layer is also called the hidden layer, and it is basically the "core"

[7] Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012), "Imagenet classification with deep convolutional neural networks, Advances in neural Information processing systems*". (p./pp. 1097--1105)*.

[8] Maureen Caudill (1989). "Neural nets primer". *Part vi. AI Expert, 4(2):61–67.*

[9] Wang SC. (2003). "Artificial Neural Network. In: Interdisciplinary Computing in Java Programming". *The Springer International Series in Engineering and Computer Science, vol 743.* Springer, Boston, MA

of the network, where the input is processed in order to give an output. Roughly, the size and number of hidden layers can determine how much information the network can distinguish between. This is how the basic structure of a neural network is built, now we will focus on how such a network can learn and train itself.
Inside of each layer we can found a specific set of nodes. From each node in one layer there is an edge that connects to each of the nodes in the next layer, just as shown in Figure 3.1. Each node performs a predefined calculation based on the input from the edges. A classic, yet simple, type of node in neural networks is the McCulloch-Pitts node[10]. An illustration of this node, or neuron as McCulloch and Pitt prefer to call them, can be seen in figure 3.2. The calculations performed in the McCulloch-Pitts node are basically a sigmoid function. They basically add up the inputs and, if these are above a certain threshold, they produce 1, otherwise they produce 0. Obviously, there are also much more complex representations of these nodes, but the McCulloch-Pitts neuron is a good start point to understand the basics of neural networks.

Together with the aforementioned nodes, a neural network is also formed by the so-called "edges", which we refer to when we talk about network weights. What they basically do is multiply the output of a node by their weight before passing it on to the next neuron they are connected to. By updating these weights, depending on the generated output of an example, we can teach the network to distinguish which input data should generate which output data. The procedure for updating weights is called backpropagation and will be described in the next section.

[10]Warren S. McCulloch and Walter Pitts. (1943). "A logical calculus of the ideas immanent in nervous activity". *The Bulletin of Mathematical Biophysics, 5(4):115–133.*

.

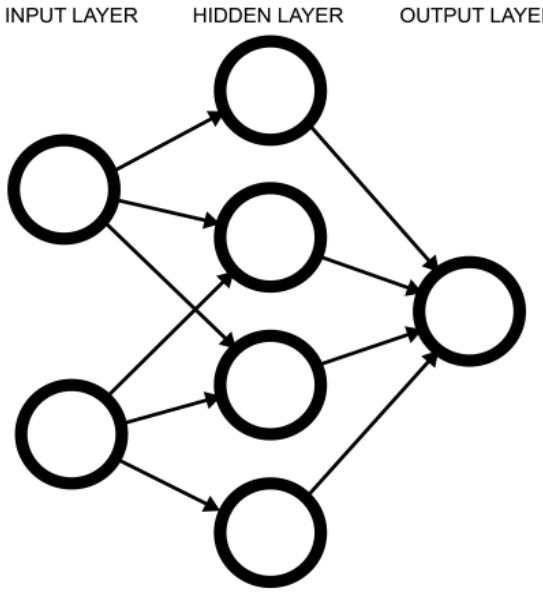


Figure 3.1: A simple example of the structure of a neural network

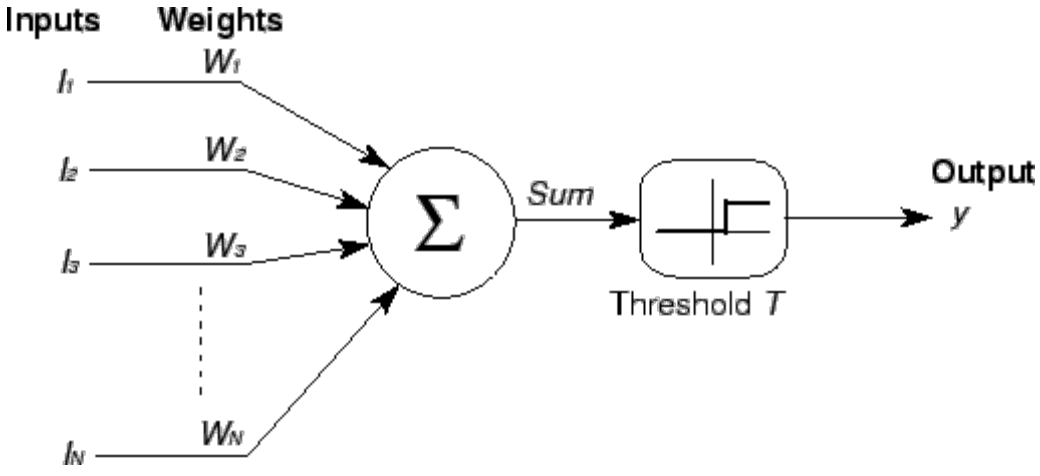


Figure 3.2: The McCulloch-Pitts neuron

3.2.1 Backpropagation

The name back propagation comes from the term employed by Rosenblatt (1962) for his attempt to generalize the perceptron learning algorithm to multiple layers, even if none of the multiple attempts to do so during the 1960s and 1970s were especially successful[11]. The backpropagation algorithm is one of the most important tools of an artificial neural network, it is specifically the part which deals with the training of the network, i.e. where it actually learns.
During this process the network updates the weights of all the edges to make it perform the correct output given a specific input. The inner workings of the backpropagation algorithm are explained by Rumelhart et al.[12] and we will give an overview of the concept in this section.

In a simplified way, the backpropagation algorithm takes care of observing the network output, comparing it with the expected output and slightly modifying the network weights so that the output approaches the expected output. To carry out this process, the difference between the network output and the expected output is calculated. The function that takes care of calculating this error is called the *loss function*. Being it an approximation, the loss function uses its derivative, and must therefore be differentiable by definition in order to work with the backpropagation algorithm.
We therefore want to update the network weights so that the output is more similar to the expected output the next time the input is given to the network. We start by calculating the partial derivative of the loss function with respect to the edges entering the exit node. Each derivative expresses how much the output of the loss function depends on the weight of each input. The weights are then updated and in the same way it is possible to update the error of the activation functions in the nodes of the previous layer. This update is recursively calculated down to the input level to update the entire network[13].

[11] Yves Chauvin, David E. Rumelhart (1995). "*Backpropagation: Theory, Architectures, and Applications".*

[12] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams (1985). "Learning internal representations by error propagation". Technical report, DTIC Document.

[13] Frederik Bredmar (2017). "Speech-to-speech translation using deep learning". Department of Computer Science and Engineering, University of Gothenburg.
.

3.3 Deep learning

Deep learning enables computational models composed of multiple levels to learn data representations with multiple levels of abstraction. These methods have improved the state of the art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Deep learning is capable of discovering complex patterns in large datasets, using the backpropagation algorithm, mentioned in section 3.2.1, which enables the network to modify its internal parameters used to compute the representation in each level from the representation in the previous level. Deep convolutional networks have led to breakthroughs in image, video, speech and audio processing, whereas recurrent nets have shone light on sequential data such as text and speech[14].

3.4 Convolutional neural network

A convolutional neural network is a subclass of neural networks that have at least one convolution layer. They are great for capturing local information (eg neighboring pixels in an image or surrounding words in text) as well as reducing model complexity (faster training, requires fewer samples, reduces the chance of overfitting). We can think of a Convolutional Neural Network (CNN) as an Artificial Neural Network (ANN) that has some kind of specialization for being able to pick-out or detect patterns from the input and make sense of them. This ability to pick up patterns is what makes CNN so useful for analyzing images or sounds, such as image classification or single word recognition. In this section we will explain what differentiates a Convolutional Neural Network from a simple Multi-Layer Perceptron (MLP).

First a CNN has hidden layers called convolutional layers and these layers are precisely what enables these networks to specialize in patterns detection. Although a CNN may also have, as often happens, other types of layers, these convolutional layers are its backbone. Like any other type of layer, also a convolutional layer receives an input, transform it in some way, and output the transformed input to the next layer. In a convolutional layer this transformation is called a convolutional operation.

[14] LeCun, Y., Bengio, Y. & Hinton, G. (2015). "Deep learning" *Nature* ***521,*** *436–444*

In order to understand this proces we can imagine as if each layer utilizes some *filters,* which are what allow the network to detect hidden patterns. Specifically, when we talk about "patterns" we are referring to any characteristic of the input, such as all the edges or geometric shapes of an image. Each filter specializes randomly, during the training of the network, in the recognition of characteristics of the input that it learns to recognize as recurring within the set of inputs that is provided to it during the training and therefore learning process. A filter could therefore begin to "specialize" in the recognition of the edges of an image while another in the recognition of angles, or circles, etc.

This type of simple and more or less geometric filters are the those that are created at the beginning of the network, the deeper a network becomes, the more these filters can become sophisticated and learn to recognize increasingly complex patterns. in subsequent layers, therefore, instead of edges or simple shapes, our filters can specialize in recognizing more abstract and specific shapes or objects, such as the fundamental features of a face in a facial recognition system or the tones that make up a word in a speech recognition system.

When we create and add a convolutional layer to our network, we must therefore first specify how many filters we want the convolutional layer to have and the size of these filters. Each filter can be thought of as a relatively small matrix for which we decide the number of rows and columns. The values that initially fill our matrix are initialized randomly. When a convolutional layer receives an input, the filter will slide over it until it has covered all the input in its dimensionality. This sliding process is referred to as *convolving*. While the filter passes over each value in which the input has been decomposed, it multiplies the internal values of the matrix that composes it with the values on which it is convolving. The matrix of dot products made by the sliding of the filter on the originals input values matrix will be the final output of the convolutional layer. This final output will summarize the feature that that filter has extracted, and that’s why we call the output of a convolutional layer a *feature map*.

3.4.1 Max Pooling

As we said before, a CNN is not only built with convolutional layers. Instead, usually, after each convolutional layer we find a *Pooling layer.* We have seen that the convolutional layers in a convolutional neural network summarizes the presence of features in the input. One problem with output feature maps is that they are sensitive to the location of features in the input. One approach to address this sensitivity is to sample feature maps. This has the effect of making the resulting subsampled feature maps more robust to changes in the position of the feature in the image, referred to by the technical phrase "local translation invariance". Pool layers provide an approach to down-sampling feature maps by summarizing the presence of features in feature map patches. Two common pooling methods are the medium pool and the maximum pool which summarize the average presence of a feature and the most activated presence of a feature, respectively (5).

3.4.2 Dense layer / Fully connected layer

The last type of layer that we usually found at the end of a typical CNN, before the output layer, is the *Dense layer.* This layer is a simple fully connected layer, where each neuron is connected to all the other neurons of the previous layer. Basically, convolutional layers help extract certain features from the input, while the fully connected layer is (better) able to generalize from these features in the output space. Then, we move from the least flexible to the most flexible level type, reducing the dimensionality of the output so that we are able to better converge in a result.

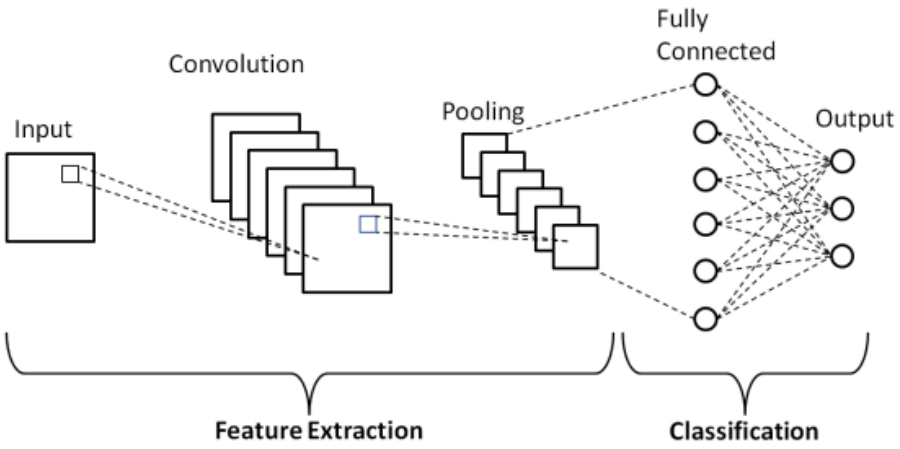


Figure 3.3: The structure of a Convolutional Neural Network

3.5 Recurrent neural network

Humans don't start thinking from scratch every second. Instead, we are able to understand complex information thanks to our ability to put together information obtained at different moments in time. This is because our thoughts have persistence. Neural networks we have analyzed so far are referred to as *feedforward neural* networks, where activations flow only in one direction, from the input layer to the output layer. A recurrent neural network has a similar functioning as a feedforward neural network, except that it also has connections that point backwards. The simplest possible RNN is composed of a neuron that receives input, produces an output, and sends that output to itself. This process is described in Figure 3.4. The idea behind RNNs is to use *sequential information.* In a traditional neural network, such as those seen in the previous sections, we assume that all inputs (and outputs) are independent of each other. But this technique performs very badly when you go to analyze inputs that require a sequential approach, whether it is essential to consider the information of the previous outputs to analyze the current inputs. If we want to predict the next word in a sentence, or, as in our case, being able to operate a speech to text translation on a continuous speech, we must also consider all the previous words in order to make sense of the speech and improve the reliability of predictions or of the transcripts. RNNs are called recurring because they perform the same task for each element of a sequence, with the output depending on previous calculations. In order to do so, RNN make use of a special *memory*, that captures information about what has been calculated so far (6).

"Whenever there is a sequence of data and that temporal dynamics that connects the data is more important than the spatial content of each individual frame."

Lex Fridman (MIT)

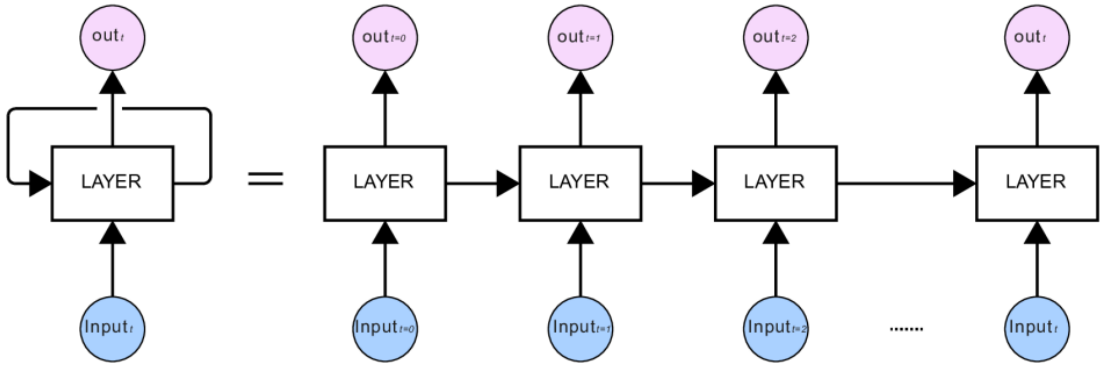


Figure 3.4: The structure of a recurrent neural network

3.6 Long short-term memory neural networks (LSTM)

Due to the transformations that data undergoes when passing through an RNN, some information is lost at each time phase. After a while, in fact, the state of the RNN practically no longer contains any trace of the first inputs. We can consider RNNs as short memory networks. To address this problem, various types of long-term memory cells have been introduced. Those new cells have been proved so effective that base cells are no longer used anymore. Long Short-Term Memory (LSTM) cell was proposed in 1997 by Sepp Hochreiter and Jürgen Schmidhuber[15] and gradually improved over the years by several researchers. The key idea here is that the network can learn what to store in the long-term state, what to throw away, and what to read from it.

[15] Hochreiter, S. & Schmidhuber, Jü. (1997). "Long short-term memory". Neural computation, pp 1735--1780.

3.7 Regularization

When building neural networks or machine learning algorithms it is important to consider the problem of *overfitting*. Overfitting is when the model begins to learn features that are too specific to the training set. Basically, the model not only learns the general rules that lead from the input to the output, but also more rules, that perhaps describe the training set, but which are not necessarily valid at a general level. This process leads to a decrease in the training error but also to an increase in the evaluation error. As a result, our model will perform worse on unknown data due to these specific rules that the model has learned from the training set. If overfitting occurs when our model is too suitable for the training set, the opposite phenomenon is called underfitting, i.e. when the model learns too general rules. We can find an illustration of the phenomena mentioned above in Figure 3.5. In the next subsections we will describe some techniques to overcome the problem of overfitting.

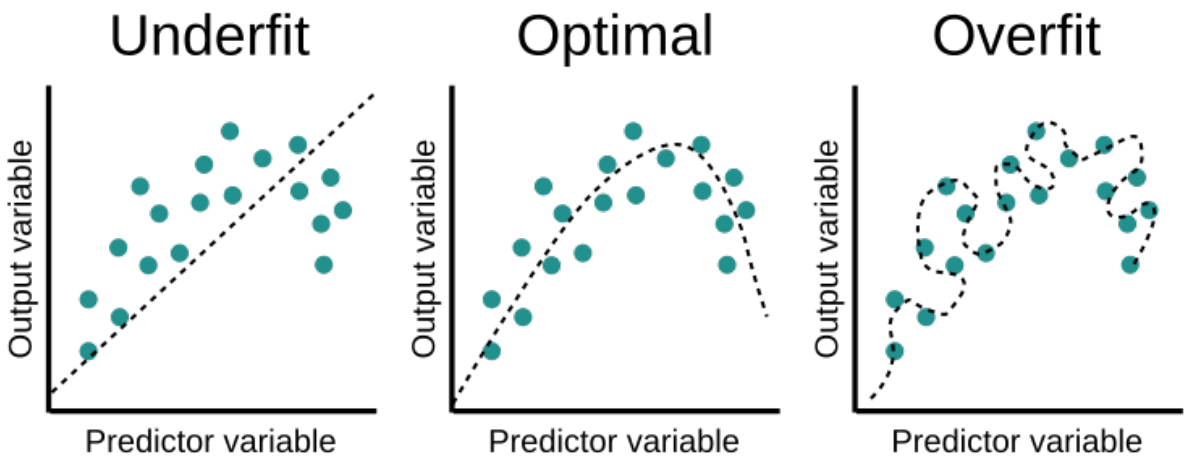


Figure 3.5: Underfitting, Optimal and Overfitting

3.7.1 Dropout

Dropout is one of the most used and efficient regularization techniques to eliminate as much as possible the overfitting effect during the training of a neural network[16].
During training a certain number of layer outputs are randomly ignored or "dropped out". This serves to reduce the number of weights in the network that focus on few strong features of the training set, in order to generalize and prevent units from co-adapting too much. On the basis of several benchmarks it has been shown that the introduction of dropout can give a great improvement on complex neural networks trained on a small training set[17]. When using dropout, it is necessary to decide the *dropout probability*, which expresses the probability that each node will be excluded during training.

3.7.2 Weight decay

Weight decay, or L_2 regularization, is a regularization technique applied to the weights of a neural network. We minimize a loss function that compromises both the primary loss function and a penalty on the Weight Norm. The weight decay method, so, is simply an addition to the loss function of the network and can be described through the following equation:

$$L_{new}(w) = L_{original}(w) + \lambda w^T w$$

where λ is a value determining the strength of the penalty and $L(w)$ is our chosen loss function. If we have a very small λ value the weight decay will not help to regularize the network. On the other hand, if λ is too large our error function will diminish, and our network will just aim to keep the weight of the network at 0. This effect can be seen if figure 3.6.

[16] Nitish Srivastava and Geoffrey et al. Hinton. (2014). "Dropout: A simple way to prevent neural networks from overfitting". *Journal of Machine Learning Research, 15(1):1929–1958.*

[17] Nitish Srivastava and Geoffrey et al. Hinton. (2014). "Dropout: A simple way to prevent neural networks from overfitting". *Journal of Machine Learning Research, 15(1):1929–1958.*

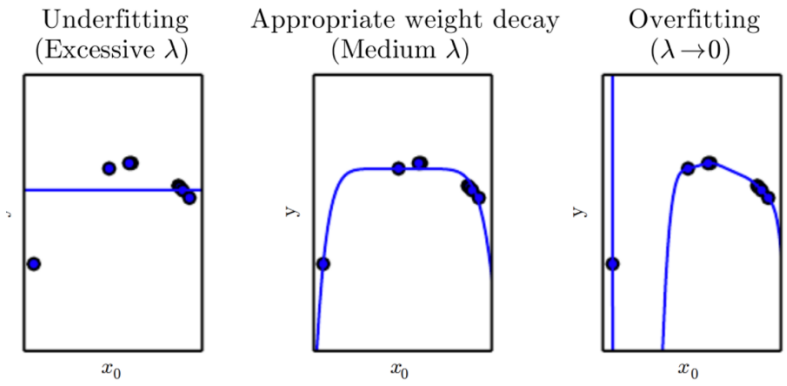


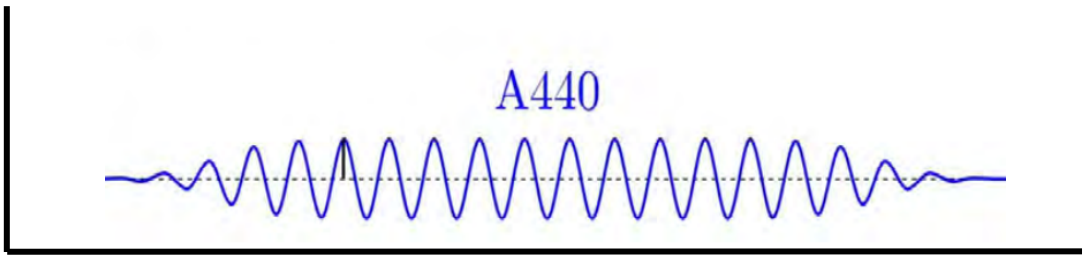
Figure 3.6: Underfitting, Optimal weight decay and overfitting with different lambdas

3.8 Fourier transform

After having explained the functioning of a neural network and the main characteristics and methods of deep learning, in this last section of the chapter we briefly discuss one of the most important preprocessing techniques that an audio file must undergo before being passed through a neural network, a fundamental step when creating a speech recognition system, which is the subject of this thesis.
Indeed, what is passed to a neural network starting from an audio file in an STT system are the so-called MFCCs[18], important parameters extracted from an audio file that accurately represent it. By learning to recognize and analyze these MFCCs, a neural network become able to recognize specific words from an audio, based on these parameters. In order to extract MFCCs from the audio file a Fourier transform is used. A Fourier transform is about decomposing, or extracting, frequencies from sound. To understand the concept, we can reference to the notes. Consider for example the note "A", we know that if we would measure the air pressure near the microphone or the instrument that is emitting the sound as a function of time, the function would oscillate up and down around its equilibrium as in the wave shown in figure 3.7, making 440 oscillation per second. A lower-pitched note, like a D, has the same structure, just fewer beats per second, as shown in figure 3.8. When both notes are played together, the pressure difference would be the sum of the pressures exerted on the air by the sound of the two notes if they were played individually, as shown in figure 3.9.

[18]Logan, Beth (2000). "Mel frequency cepstral coefficients for music modeling." *Ismir. Vol. 270.*

Pressure (440 oscillation per second)



Time

Figure 3.7: "A" note, A440, 440 oscillation per second (beats per second)

Pressure (294 oscillation per second)

Time

Figure 3.8: "D" note, D294, 294 oscillation per second (beats per second)

Figure 3.9: the resulting sum of note "A" plus note "D"

When a microphone records sound, it is simply picking up air pressure at many different points in time. In the case of the two notes played together he would only "see" the final sum, indeed, de graph shown in figure 3.10. A Fourier transform is what it is used to the decompose the signal formed by the “sum” of many different frequencies and decompose it in the pure frequencies that make it up. The fast Fourier transform (FFT), which is an algorithm that computes the discrete Fourier transform of a sequence, is the common implementation of Fourier transform. In our specific case, FFT will be used to extract MFCCs, the fundamental audio feature we mentioned in the beginning of the paragraph, from the audio that we want to convert into text form, in order to perform the STT translation.

4

Research

4.1 Speech Recognition using Recurrent Neural Networks

This section discusses the work by Aditya Amerkar, Gaurav Deshmukh, Parikshit Awasarmol and Piyush Dave on "Speech Recognition using Recurrent Neural Networks"[19].
I found this paper interesting since it deals with the same task I did and described in section 5 of this thesis, that is speech recognition, but using recurrent neural networks instead of a CNN. Thanks to this work, we can understand the differences of the two types of networks and their performance.
The main difference between the two models is that a convolutional neural network is not capable of processing sequential information. This is because the network processes every input taking into account only the present state of each neuron, which is not modified by previous inputs. A recurrent neural network instead has this capability. Each neuron has a sort of “memory”, that allows them to analyze and process each input taking into account some of the information contained in the previous processed input. This mechanism is essential for the processing of sequential information, since in most cases these have some connection between them that must be taken into account. If a convolutional neural network does a remarkable job for processes such as speech recognition for single and few words, for more complex speech recognition projects, which want to process more words or even complex sentences, the use of recurrent neural networks is essential.

[19] Aditya Amerkar, Gaurav Deshmukh, Parikshit Awasarmol and Piyush Dave (2018). "Speech Recognition using Recurrent Neural Networks".

4.2 Speech-to-Speech translation using deep learning

Here we present and discuss the work done by Fredrik Bredmar in 2017 at the Computing Science division of the University of Gothenburg:
"Speech-to-speech translation using deep learning"[20].
Although the focus of this thesis is the study of speech-to-text methods, we found Bredmar's work very interesting as it goes beyond just the translation from "voice" to "text", but deals with a complete speech-to-speech translation, capable of translating the input voice into another language without ever going through a text form.
Most of the similar projects, indeed, which carry out a linguistic translation task, such as Google Translate, obtain a Speech-to-speech translation through three distinct but communicating steps. Firstly, through a speech-to-text model, the voice is translated into a written text, secondly the written text is translated into the new language and finally the new text is, through a speech synthesis model (speech synthesizer or text-to-speech model), translated again into an audio format. The problem of this three steps method is that the "language translation" part of the model is done within the text domain. Although these three components have, in recent years, become increasingly efficient and precise, thanks to the advances made in the field of neural networks and deep learning, the translation from speech to text totally eliminates the voice characteristics of the speaker, as emotion, pitch or accent, which indeed are not reported in the subsequent text-to-speech translation. Bredmar's work aims to carry out a speech-to-speech translation by eliminating the intermediate step of text-to-text translation, building an LSTM neural network capable of receiving as input an audio file of a voice in a specific language and returning as an output another audio file containing the translation of the input audio into a new language, maintaining all the vocal characteristics of the latter.
"(..) we investigate the performance of a speech-to-speech translation system
that persevere voice characteristics throughout the translation. The long short-term memory (LSTM) network will be the core of the system."

[20] Frederik Bredmar (2017). "Speech-to-speech translation using deep learning". Department of Computer Science and Engineering, University of Gothenburg.

4.2.1 Dataset and model

A private dataset was created for model training. Bredmar proposes to build a dataset that has many sentences of varying lengths and different tones, that simulate as much as possible common conversations, recorded by a variety of different people. Since the model is designed for a translation from English to French, each sentence had to be available in both languages. Dubbed films were selected to accomplish the task, given the thriving French dubbing. During dubbing in addition, the voice characteristics should be preserved. The final dataset consisted of 16 movies, with each movie generating approximately 700 sentences ranging in length from 3 to 50 seconds and thus a total of 11000 sentences.
The long short-term memory network used is instead composed of two layers with 800 neurons each. For the construction of the latter was used the Google *Tensorflow* library, described in Chapter 2.2.

4.2.2 Results and experiments

The first experiment performed by the author of the paper is on network performance, executed by changing the size of the network by adding more neurons in both the two layers. As a result, we get a clear improvement in performance when the dimension of the network is increased, as can be seen in Figure 4.1, described by the loss function that decreases significantly as the size of the network increases. However, the decrease in the loss function is however small with respect to the increase of neurons in the network, it is clearly evidenced how greater networks results in lower training and evaluating error. For the second experiment the effects on the loss function due to the increase of the training set are analyzed. The resulting plot can be seen in Figure 4.2. From this last test it can be noticed that the increase of the training set results in a notable decrease of the loss function, but also that the decrease of the training error is more rapid in the early increments of the training size, suggesting that for even more precise model an enormous dataset must be used.

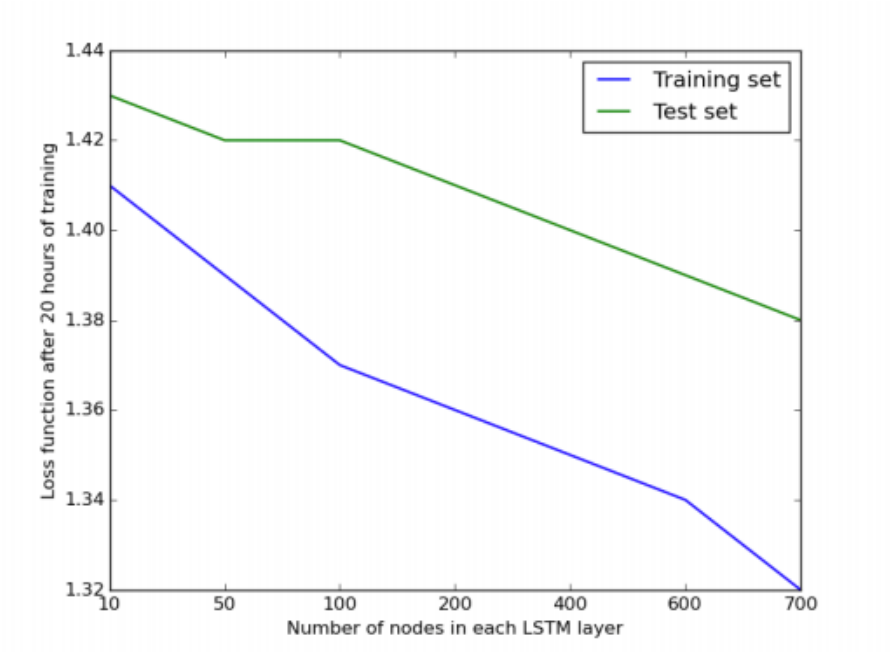


Figure 4.1: Graph showing how the training set loss and evaluation function change depending on the size of the trained network.

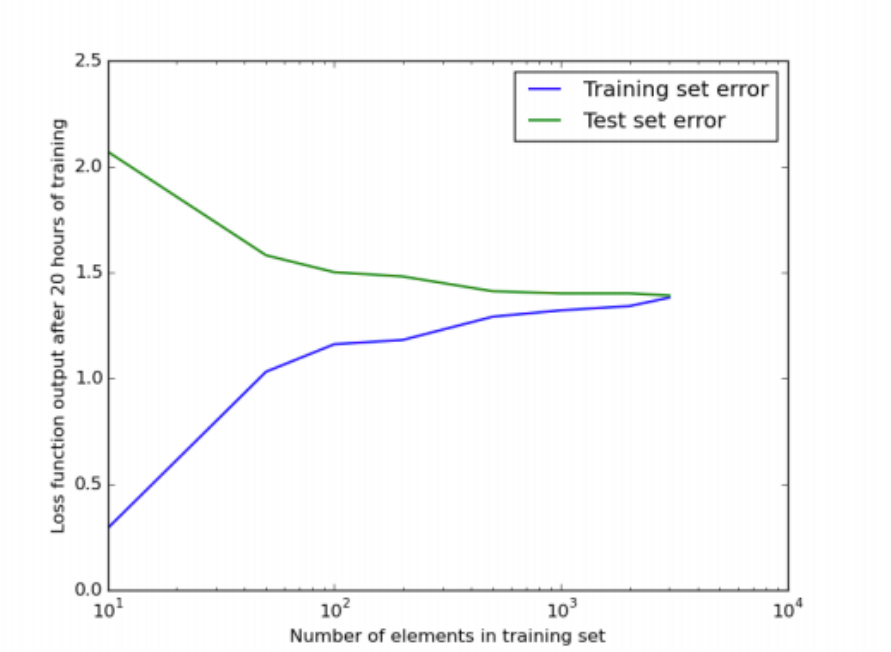


Figure 4.2: Graph showing how the training set loss and evaluation function change depending on the size of the training set.

4.2.3 Conclusions on the results

The result of this work can be considered very interesting as it shows that LSTM network architecture is enough complex for this kind of sequences, based on training and test error. The problem reported by the author of the work is that the network is still not large enough to manage well a larger or complex dataset, that is the network is underfitting the dataset. However, the positive result of the experiment remains interesting, demonstrating that, with the appropriate instrumentation capable of handling a greater complexity of the network architecture, speech-to-speech translation without the need for speech-to-text or text-to-speech models is possible and able to partially maintain the speech characteristics of the input source.

5

Keyword Spotting System

For this thesis a Speech Recognition projects is described. The project deals with the basics of Speech to Text, resulting in the construction of a Convolutional Neural Network capable of recognizing ten simple words and transcribing them into text starting from an audio file. In this chapter the complete construction of the model and the experimentation made on it is described in detail.

5.1 Keyword Spotting System

The project considered deals with the basics of Speech Recognition techniques via CNN. While maintaining a low overall complexity, the project results in a great way to understand how STT techniques work in broad terms. The system is able to identify a limited number of Keyword by analyzing an input audio file and then classifying it into one of the possible words set on which the network was trained.
In practice, such a model can be used to make certain systems capable of understanding simple voice commands, such as "Stop", "Go", "Left" or "Right".
It consists of three different .py files, respectively:
Prepare_dataset.py, Train.py and Keyword_spotting_service.py.

5.2 Dataset

First of all let's focus on the data that will be fed to the Keyword Spotting System.
A Google-supplied dataset called "Speech Command Dataset" was used to train the model in question. The dataset is composed by 65,000 one-second long utterances of 30 short words, by thousands of different people, contributed by members of the public through the AIY website (7). For the first part of this project, ten of the thirty available words where considered, in particular the words: "Down", "Go", "Left", "No", "Of", "On", "Right", "Stop", "Up", "Yes".
After having built and analyzed the model trained on these ten words, the dataset will be expanded to all the thirty words in order to study the differences that this "expansion" implies on the effectiveness and accuracy of the model.

5.2.1 Preparing the Dataset

The first file (Prepare_dataset.py) deals with the creation of a JSON file, starting from the available dataset, which will then be used by the network for training.
For preparing the dataset to be fed to the network, we need to extract some audio features from each of the audio samples that we have in our dataset, and then store those features on the JSON file.
In particular, the audio features that are going to be extracted are the so called "MFCCs" (Mel Frequency Cepstral Co-efficents), which have been one the most important features used for speech recognition in the last few decades [18]. Their success relates to their ability to reproduce the speech amplitude spectrum in a compact form (Logan, Beth 2000).
This process of "extraction" can be seen as taking different "snapshots" at each time segment of every audio file and for each snapshot extracting a number of MFCCs coefficients. Since sound is a wave it is not possible indeed to extract any features by taking a single sample from it, for this reasons MFCCs are computed over a window. In order to do so a Python library called "Librosa" is used. To compute MFCCs Librosa uses fast Fourier transform (FFT) which require the exact length of the window in number of samples, the length of each of the sample that compose the window, the number of MFCCs that we want to extract from each sample and the length of the FFT. We will set the number of samples to "22050", since it is computed that it is the exact number of segment that compose a one second worth of sound (which is the length of

all the samples audio file that we are using for training), the “hop length”, that is how “big” each sample should be in number of frame, to “512”, the number of MFCCs to be extracted to 13 and the length of the windows of the Fast Fourier transform to “2048”. Besides the length of the window in number of sample (22050), all the other parameters are set to those value by default.
We can now create the “prepare_dataset” function, that will take as input the dataset and the above explained parameters and generate as output the json file, containing all the audio features from each audio samples, that will be fed to the Convolutional Neural Network.

5.3 Creation and training of the CNN

For this section we are going to design and build the architecture of a convolutional neural network, compile it, train it and evaluate it. All of this is done in the second file that is the train.py.
We start by importing the dataset from the json file and by splitting it into two different partitions, one used for training that will be the 90% of the dataset and one for testing that will get the remaining 1% of the dataset. A third partition with validation purposes will be also created and it will get the 10% of the training partition. Now that we have all of our data, we can focus on the model itself. For doing so we need to use “Keras”, an open source library built on top of TensorFlow (one of the most used python libraries for machine learning) that provides a Python interface for Artificial Neural Networks. The network that we are going to build will be composed by three convolutional layers followed by a dense layer and a Softmax classifier, that is used to output from the network the probability that the input corresponds to each one of the ten words.
Each one of the layers is created by specifying its number of filters, its kernel (the size of each filter), its input shape (just for the first layer), its activation function and the kernel regularizer that is used to avoid the model to overfit the training set.
The creation of each new convolutional layer is followed by a Normalization layer, that is basically a transformation that maintains the mean output close to 0 and the output standard deviation close to 1, and a down-sampling layer, accomplished by the Max Pooling process, which is a sample-based discretization process with the objective of down-sampling an input representation through reducing its dimensionality.

For each convolutional layer the activation function that is utilized is “relu”, which has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance (8).
As general parameters for the network, learning rate, number of epochs and batch size were set respectively to 0.0001, 40 and 32.
The learning rate is a parameter related to the optimization algorithm of the network which determines the step size at each iteration while moving towards a minimum of the loss function. Smaller learning rate implies more precision of the networks at the expenses of time to train of the model. Since here the model will train quit fast we can use a small learning rate to increase the performance of the network.
The number of epochs tell us instead how many times the network will pass through the whole training dataset. The batch size instead tells us the number of samples that will be propagated through the networks. After the three convolutional layers we need to flatten the network output from three dimension to a one dimension array in order to it to be passed to the dense layer. A dense layer is a layer where each unit or neuron is connected to each neuron in the next layer. After the dense layer we have the dropout layer. Dropout is a regularization technique, which aims to reduce the complexity of the model with the goal to prevent overfitting. Using “dropout", you randomly deactivate certain units (neurons) in a layer with a certain probability p from a Bernoulli distribution (typically 50%, but this yet another hyperparameter to be tuned). So, if you set half of the activations of a layer to zero, the neural network won’t be able to rely on particular activations in a given feed-forward pass during training, and so, again, prevent overfitting.
As stated before, our output layer will be a Softmax classifier, which will output the probability that our input audio contains each one of the 10 words. The word with the higher probability will be selected and given as the output of the model.
Appendix A shows the code used to build this convolutional neural network.
The training process was executed on a 2018 Macbook Pro 13, packed with a Quad-Core i5 2,3 GHz, 8 GB 2133 MHz DDR3 RAM and an Intel Iris Plus Graphics 655 with 1536 MB of memory.
After the model has completed its training, I got a test error (loss) equal to 0.2278 and a test accuracy equal to 0.9514.
In Figure 5.1 and figure 5.2 it is possible to respectively see the resulting plot of training and validation loss and accuracy over the 40 epochs training.

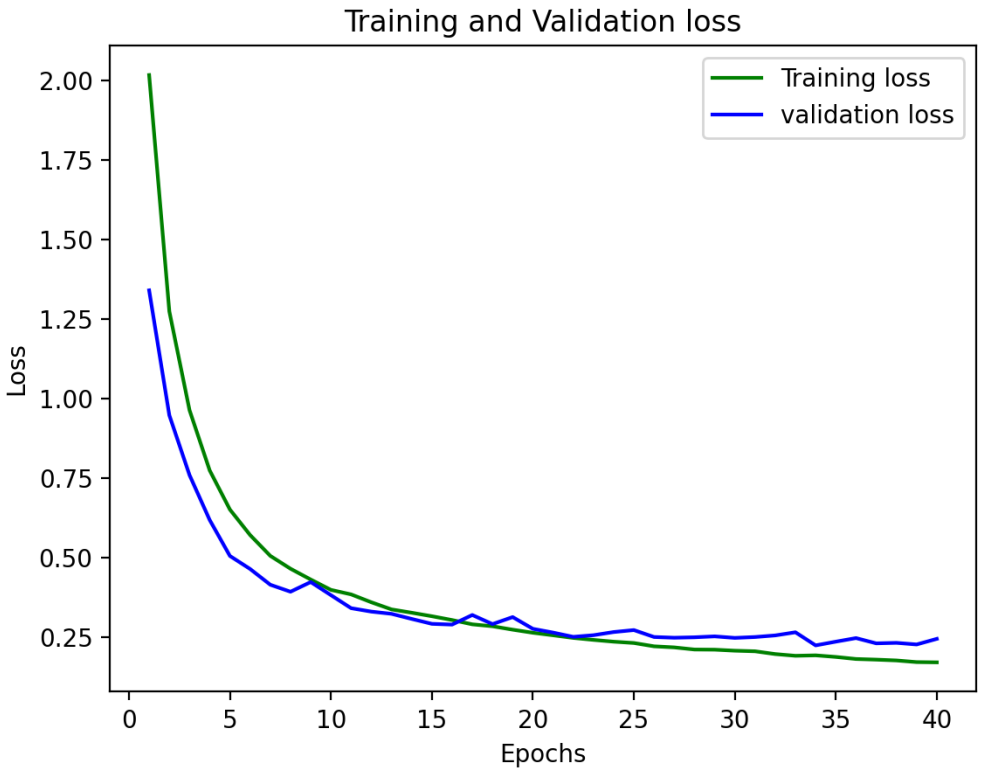


Figure 5.1: Loss function decreasing over the 40 epochs training

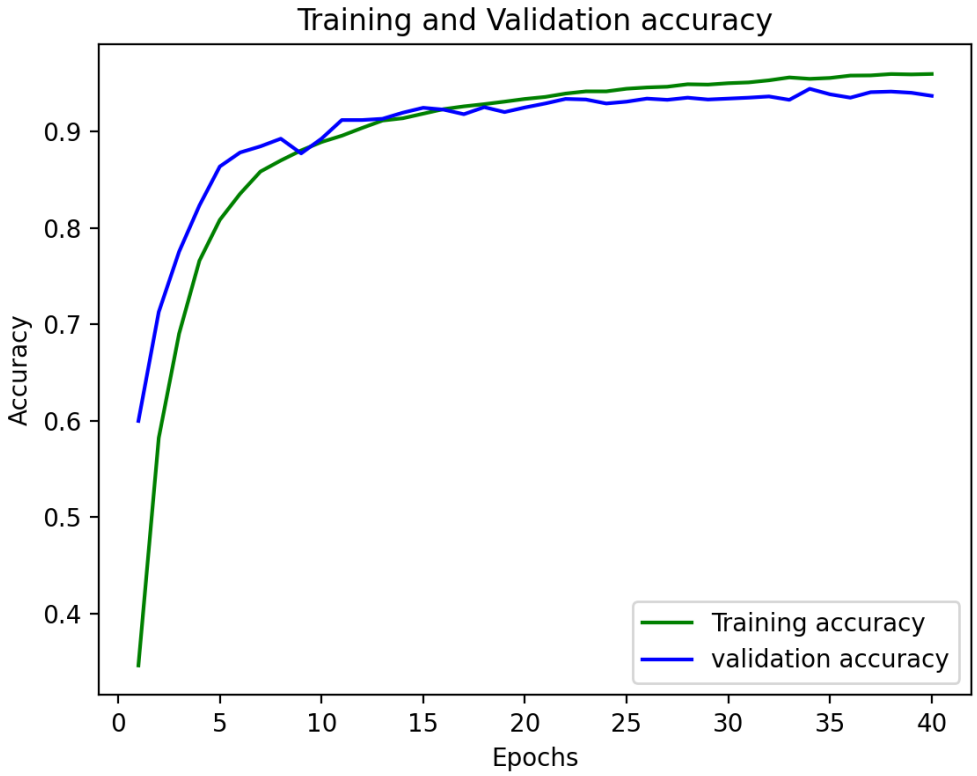


Figure 5.2: Accuracy of the network increasing over the 40 epochs training

5.4 Experiments

In this section the experiments carried out will be discussed. Several models have been created by varying the hyperparameters in order to optimize the model in the most appropriate way.

5.4.1 Removing Dropout

The first change that I decided to apply to model in order to observe its effects was removing the dropout layer. Although this choice is obviously counterintuitive since the dropout greatly improves the performance of the network, I found it interesting to remove it in order to clearly measure how much this increase in performance due to its presence was marked. I created a new training section and a new model called "model.h6", this time trained without the dropout layer. Removing the dropout layer should mean increasing the overfitting of the model. What I expected to see was an increase in the train set accuracy and a decrease in the test set accuracy, caused by the model adapting too much on the training set. After having trained the model what I got was a train set accuracy of 0.9732 and a test set accuracy of 0.9320. The same results are visible form the loss function, producing a test error of 0.2547 while for the training set I got just 0.1363. As expected, I lost 2% on accuracy and gained 3% on the error for the test set compared with the first model, while I gained some accuracy and decreased the error on the training set, thank to overfitting.

5.4.2 Increasing the number of predictable words

As a second experiment in order to test the performance of the network I decided to increase the number of words that the speech recognizer was able to handle from 10 to 35. In order to do so I downloaded the updated version of the Speech Commands Dataset from Google which contains 35 words, three times and a half more than the previous version. From this experiment I expected my accuracy to decrease and my test error to increase since the CNN is not very large and so shouldn't be much capable of working with large dataset. After having changed the dataset from which the model was trained, I started the training process again. This time the training process took around 36 minutes to be completed and as results I got a test accuracy of 0.9008 (90%) and a test error of 0.3824. As expected, the accuracy of the network decreased from

95% to 90% and the test error increase from 22% to 38%. This demonstrate how this type of network is not capable of scaling well when the dataset size increase considerably. Still, the network was capable of recognizing the majority of the word that I tested, the old ones as well as the new ones.

5.4.3 Changing the activation function

As last experiment I decided to change the activation function. Activation functions are used in neural network in order for them to process non-linear function and so, more complex information. The activation function is precisely what decides if a specific neuron should activate depending on the input. The first activation function used was the so called "sigmoid activation function", used in the early 1990s. This type of activation function has mainly two problems in it. The first was that it was non-zero centered and the second was that it causes the gradient to "vanish". After the sigmoid the most used function was the "Tanh activation function" which solved the problem of zero centrality but still caused the gradient to vanish in some cases. The tanh function was used for CNN until 2010, when it was nearly totally substituted by the "ReLu activation function". ReLu is still at the moment the most used activation function in the world of convolutional neural network, it solves the problem of the zero centrality and also the problem of the vanishing gradient. Still, one problem that ReLu could have is that it can cause the "death" of some neurons. When a big gradient is in fact backward propagated through a ReLu neuron a weight update could be generated in way that that neuron will never be activated again by any input. To solve this problem an experimental version ReLu has been developed and it's called "Leaky ReLu". This modified ReLu activation function seems to stop the "dying neurons phenomena". I decided to implement this feature in the network in order to increase its reliability. Leaky ReLu is not automatically included in the Keras package and must be imported separately. After having imported it, I update the activation function of all the convolutional layers and started a new training process. At the end of the training, I got the same results in terms of test and training error and accuracy but the network should now be more stable and optimized.

6

Conclusions

As conclusions, it can be said that the world of neural networks in the field of voice modeling and analysis is constantly expanding and producing new and interesting results every year. For a STT or Speech Recognition model built using a neural network, the ideal would be the use of a recurrent neural network, a very large dataset and enormous computational power, three characteristics that make it a difficult goal to pursue in academic terms. Nonetheless, however, by analyzing these complicated models at a theoretical level, it is clear that this type of application can result in excellent results if achieved with adequate resources. A simple Speech Recognition program, on the other hand, which deals with the understanding of a few simple words, can be implemented instead through a simpler network, obtaining very good results. From the experimentation and research carried out during the drafting of this thesis it is however clear the enormous potential of these network structures, which over time, with new algorithms and new increasingly powerful processors, is becoming one of the most varied and practical practices in the world of information technology. The revolution brought about by neural networks radically changes our conception of programming, from something that, based on pre-established rules, is able to produce results, to something that can compose these rules autonomously by observing the results. This type of application greatly expands the horizons of what can be achieved through a computer, making us touch a future in which artificial intelligence could be the master. As in the context of speech recognition, the use of neural networks can be applied to almost all practices that somehow have an IT process behind them, greatly enhancing all programming areas that would require extreme complexity in their design. The study of these fascinating models has further increased my interest in this subject, which I will certainly carry on as my studies progress.

Creating an artificial being has been man's dream since the dawn of science. Not just at the beginning of the modern era, when our ancestors stunned the world with the first thinking machines: primitive monsters playing chess! How far we have come ... The artificial being is a reality, a perfect simulacrum, articulated in the limbs, articulated in the language and not lacking in human reactions.

Allen Hobby
A.I Artificial Intelligence (2001)

Appendix A

```python
def build_model(input_shape, learning_rate, error="sparse_categorical_crossentropy"):

    #build the network (tensorflow and keras)
    model = keras.Sequential()

    # conv layer 1
    model.add(keras.layers.Conv2D(64, (3, 3), activation=keras.layers.LeakyReLU(alpha=0.01),
                                  input_shape=input_shape, kernel_regularizer=keras.regularizers.l2(0.001)))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.MaxPool2D((3,3), strides=(2,2), padding="same"))

    # conv layer 2
    model.add(keras.layers.Conv2D(32, (3, 3), activation=keras.layers.LeakyReLU(alpha=0.01),
                                  kernel_regularizer=keras.regularizers.l2(0.001)))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.MaxPool2D((3,3), strides=(2, 2), padding="same"))

    # conv layer 3
    model.add(keras.layers.Conv2D(32, (2, 2), activation=keras.layers.LeakyReLU(alpha=0.01),
                                  kernel_regularizer=keras.regularizers.l2(0.001)))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.MaxPool2D((2, 2), strides=(2, 2), padding="same"))

    #flatten the output feed it into dense layer
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(64, activation=keras.layers.LeakyReLU(alpha=0.01)))
    model.add(keras.layers.Dropout(0.3))

    #sfotmax classifier
    model.add(keras.layers.Dense(NUM_KEYWORDS, activation="softmax")) # [0.1, 0.7, 0.2]

    #compile the model
    optimiser = keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimiser, loss=error, metrics=["accuracy"])
```

Appendix A: The Python code used to build the convolutional neural network used in the Keywords spotting system explained in chapter 5.3

Bibliography

Amerkar, A., Deshmukh, G., Awasarmol, P., Dave, P. (2018).
"Speech Recognition using Recurrent Neural Networks".

Bohouta, Gamal & Këpuska, Veton (2017).
"Comparing Speech Recognition Systems (Microsoft API, Google API And CMU Sphinx)".
Journal of Engineering Research and Application. 2248-9622. 20-24.

Carnegie Mellon University. (2019).
"CMUSphinx Tutorial for Developers".

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams (1985).
"Learning internal representations by error propagation".
Technical report, DTIC Document.

Filippidou F, Moussiades L. (2020)
"A Benchmarking of IBM, Google and Wit Automatic Speech Recognition Systems, Artificial Intelligence Applications and Innovations".

Frederik Bredmar (2017).
"Speech-to-speech translation using deep learning".
Department of Computer Science and Engineering, University of Gothenburg.

Hochreiter, S. & Schmidhuber, Jü. (1997).
"Long short-term memory".
Neural computation, pp 1735--1780.

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012),
"Imagenet classification with deep convolutional neural networks, Advances in neural Information processing systems*". (p./pp. 1097--1105)*.

LeCun, Y., Bengio, Y. & Hinton, G. (2015).
"Deep learning" *Nature **521,** 436–444*

Logan, Beth (2000)
"Mel frequency cepstral coefficients for music modeling." *Ismir. Vol. 270.*

Maureen Caudill (1989).
"Neural nets primer".
Part vi. AI Expert, 4(2):61–67, 1989.

Minsky, M., Papert, S. (1969).
"Perceptrons: An Introduction to Computational Geometry".
Cambridge, MA, USA: MIT Press.

Nitish Srivastava and Geoffrey et al. Hinton. (2014).
"Dropout: A simple way to prevent neural networks from overfitting".
Journal of Machine Learning Research, 15(1):1929–1958.

Peter Lyman and Hal R. (2003).
"*How much information?* coordinator: Kirsten Swearingen, School of Information Management and Systems at the University of California at Berkeley.

Rumelhart, D., Hinton, G. & Williams, R. (1986).
"Learning representations by back-propagating errors".
Nature 323, 533–536.

Tachibana, H., Uenoyama, K. and Aihara, S. (2018).
"Efficiently Trainable Text-to-Speech System Based on Deep Convolutional Networks with Guided Attention".
IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 4784-4788.

Wang SC. (2003).
"Artificial Neural Network. In: Interdisciplinary Computing in Java Programming".
The Springer International Series in Engineering and Computer Science, vol 743. Springer, Boston, MA

Warren S. McCulloch and Walter Pitts. (1943).
"A logical calculus of the ideas immanent in nervous activity".
The Bulletin of Mathematical Biophysics, 5(4):115–133.

Young, S. J., P. C. Woodland, and W. J. Byrne (1993).
"HTK: Hidden Markov Model Toolkit V1. 5".
Cambridge Univ. Eng. Dept. Speech Group and Entropic Research Lab. Inc., Washington DC.

Yves Chauvin, David E. Rumelhart (1995).
"*Backpropagation: Theory, Architectures, and Applications"*.

Web References

(1) Valerio Velardo (2020).
Deep Learning Audio Application from Design to Development.

(2) Allison Linn (2016).
Historic Achievement: Microsoft researchers reach human parity in conversational speech recognition.

(3) Tensorflow Website (2021).
An end-to-end open source machine learning platform.

(4) The New York Times Archives (1958).
July 13 1958, (Section E, Page 9).

(5) Jason Brownlee (2019).
A Gentle Introduction to Pooling Layers for Convolutional Neural Networks.
Deep Learning for Computer Vision.

(6) Purnasai Gudikandula (2019).
Recurrent Neural Networks and LSTM explained.

(7) Pete Warden (2017).
Google Speech Command Dataset
Software Engineer, Google Brain Team.

(8) Jason Brownlee (2019).
A Gentle Introduction to the Rectified Linear Unit (ReLU)
Deep Learning Performance.