Luiss Guido Carli - Libera Università
Internazionale degli Studi Sociali

Department of "Impresa e Management"

Management and Computer Science

Course of Artificial Intelligence and Machine Learning

Voice Cloning with Italian Phonetics

Supervisor:

Prof. Marco Querini

Candidate:

Federico Astolfi
237021

ACADEMIC YEAR 2020/2021

Contents

List of Figures

Abstract

The goal of my project was to implement the Speaker Verification to Multispeaker Text-to-Speech system, for voice cloning, extending it to the Italian language. The models I have created allow to generate an audio from a text, which reproduces what is written with the vocal pitch of a speaker of reference. So the input of the completed system will be an audio of a few seconds of an Italian speaker, who can say any phrase, and a text at will that we want to synthesize. In the paper I examine the 3 models that make up the system, and I analyze my results generated by their implementation with data in Italian.

Chapter 1

Introduction

In recent years, deep learning algorithms have been hugely successful in the world of artificial intelligence, thanks to the development of GPUs, whose speed has significantly increased since 2011. This has been possible mainly thanks to the demand for better performing graphics cards in the world of video games, which have facilitated a much more effective training of convolutional neural network models.
This kind of machine learning model has spread into many fields, for example, the visual recognition of objects or people; voice recognition; the implementation of self-driving cars; being able to recognize road patterns and how to follow them, and much else.
Our goal is to focus on text-to-speech and speech recognition, and to integrate these in order to produce a voice cloning model that can synthesize an audio from a text which reproduces what is written in that text with the vocal characteristics of a chosen speaker.

There are many publications and implementations of deep learning algorithms in the field of voice cloning, but most of them are based on training the model directly on the voice of the speaker that one wants to reproduce. This system can be problematic since, in order to produce this result every time that you want to clone the voice of a new speaker, you have to train the whole model; you also must have a large amount of clean audio files, with little noise and reverberation.
For this reason I was very interested in the solution proposed in the paper ”Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis” [14], which describes a system through the development and integration of three models, the encoder, the synthesizer and the vocoder, in which the synthesizer produces a spectrogram conditioned by the characteristics of the speaker, where these are produced by the encoder, starting from input lasting just a few seconds.

Finally, the vocoder produces a wave-form audio from the spectrogram produced by the synthesizer.
With this solution there is no longer the need to re-train the whole model every time you want to clone the voice of a new speaker, and only a few seconds of the voice are enough to obtain adequate results.
The authors of the paper mentioned above have not publicly released any implementation of their system, however, this was done later by Corentin Jemine [19], who made it open source on github. The implementation follows the processes described in the original paper, with the addition of making it run in real-time, which means to generate the output in a time span which is less than or equal to the duration of the output itself.

My goal is, therefore, with the help of the open source code, to extend the system, structuring it on Italian phonetics, so that one can clone Italian voices and generate Italian speech from an Italian text.

Chapter 2

State Of The Art

Generating natural speech from text, or what is usually called text-to-speech, has been a challenge for quite some time, and over time many solutions and proposals have been devised, which have obtained progressively better results, there were several implementations of this system, among them one of the most interesting is the Diphone Synthesis [3], which consists of the concatenation of diphones, different depending on the phonetics of the language used.

For a long time the technique that was used was to concatenate small units of audio in wave-form, previously recorded, and join them in such a way that they built the desired speech, obviously, as we can imagine, though the results were decent, the naturalness of speech was not optimal.

The evolution occurred with the introduction of the statistical parametric speech synthesis system [4], [5]. This system uses hidden Markov model (HMM), which produces a future sequence of the speech to be generated which is then synthesized.

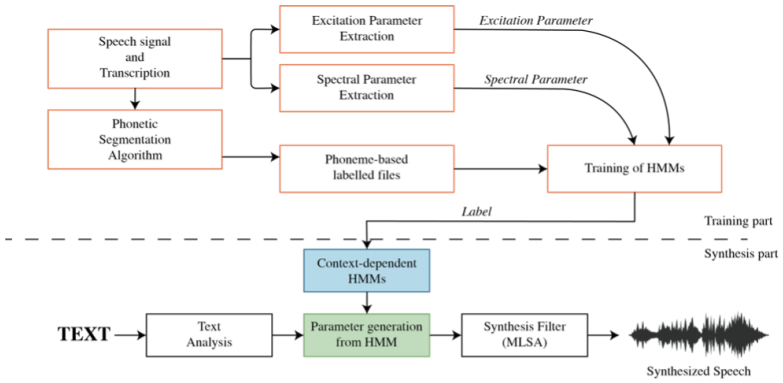

Figure 2.1: The overview of the statistical parametric speech synthe- sis system, based on hidden Markov model

The real revolution in TTS came with the arrival of WaveNet [11] in 2016 and with the subsequent arrival of Tacotron [13] in 2017, models that we will use in the implementation of our project, and that we will analyze in more detail.
WaveNet essentially acts as an acoustic model and as a vocoder, but in order to perform TSS it needs to be conditioned on the characteristics of the language on which we want to build the model; in practice, without the conditioning with the linguistic characteristics, the result has no semantic structure.
Tacotron consists essentially of a dual system, encoder-decoder, that, starting from a sequence of characters, uses the encoder to convert the characters into an internal feature representation, and the decoder to convert the feature representation into frames of the mel-spectrogram, which will then be synthesized into speech by a vocoder.

Moreover, in order to estimate the results of our models we must also introduce the measure Mean Opinion Score, mainly used in order to estimate the quality in the telecommunications, and which typically assigns a value between 1 and 5, where 1 is poor quality and 5 excellent quality. In our case it is particularly useful in order to estimate the naturalness of our output, comparing it to human speech, and also in order to estimate the similarity between two audio.
This technique is the most common technique for evaluating outcomes in the context of the publications we analyzed.

Regarding Speaker Verification to Multispeaker Text-to-Speech (SV2TTS), there are two publications: one by the authors of the framework [14] and one by Corentin Jemine [19], who implemented and made the system proposed in the reference paper public.

2.1 WaveNet

WaveNet [11] is a raw audio generation technique, based on a neural autoregressive generative model.
We know that raw audio waveforms are signals with a very high temporal resolution, of at least 16,000 samples per second. The goal of WaveNet is to predict audio samples, which means that for the generation of an audio duration of 5 seconds, the model must predict about 80,000 samples, which becomes complicated if one considers that the model must be autoregressive.

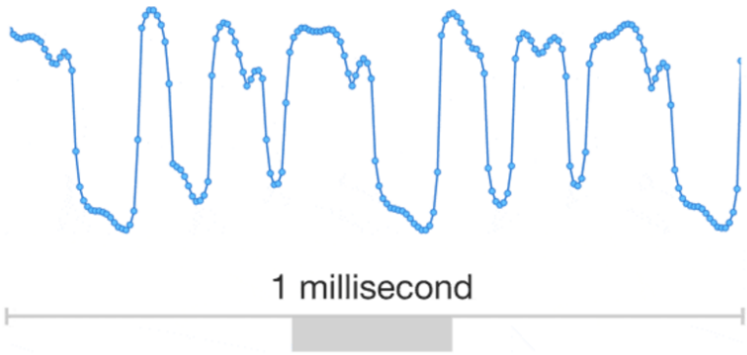

Figure 2.2: Samples of a raw audio wave-form in a 1*ms* time interval (screenshot from [10])

In fact if we consider x as waveform, x is composed of n samples, $x = \{x1, x2, ..., xn\}$, where the joint probability of a waveform is factored as the product of the conditional probabilities of the samples, so each audio sample is conditional on all previously generated samples.

$$p(x) = \prod_{t=1}^{T} p(x_x | x_1, ..., x_{t-1})$$

The conditional probability distribution is modelled by a stack of convolutional layers. Where input and output have the same size, that is the number of samples in the wave-form, each new output is a categorical distribution for the audio sample to be predicted, obtained after being passed through a softmax function, which essentially transforms the output values into the probability of belonging to a given category.

Our input and output have a single dimension, a very good characteristic to implement a Causal Convolution model, which however needs modifications.
In fact, we said that each prediction is conditioned by all the predictions previously made, so the predictions are sequential; after having predicted a sample, this output is inserted into the network as a new input for the next prediction.
The problem of the Causal Convolution model is that to be able to condition the output on all the previous ones it is necessary to have several layers if the input is large.
We call receptive field the number of inputs to which the model refers to condition the next output, the amount of context that a neuron sees in the input to predict its output, considering that for each layer in the Causal Convolution model nodes are conditioned by the last two of the previous layer, as shown in the figure, we can get the receptive field, in the case of 3 layers the receptive field is 5.

In our case, for example, to generate an audio waveform of 1 second we should generate about 16,000 samples, which means that to condition the last output to all previous ones we need 15,998 layers, which is problematic.

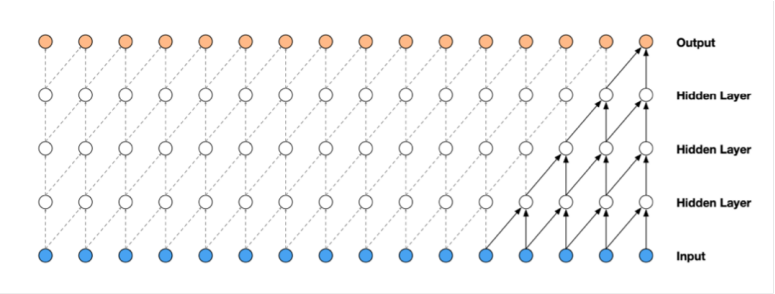

Figure 2.3: Visualization of a stack of causal convolutional layers (from [11])

In order to solve this problem WaveNet implements a Dilated Causal Convolution model, which is very similar to the one described before, but the filter is applied on a larger area, so, essentially, the dilation is the number of nodes that are skipped before taking the one to condition the next layer.
WaveNet duplicates the dilation for each successive layer, so the receptive field grows exponentially as the number of layers increases.

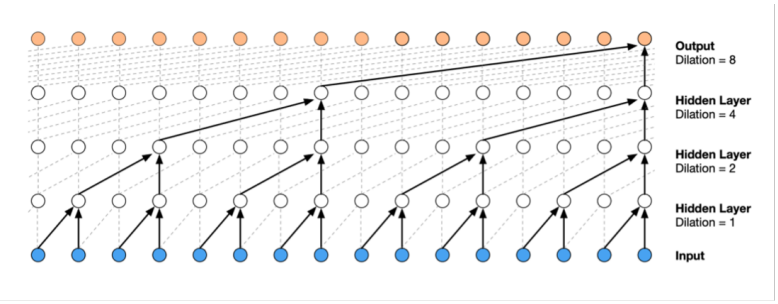

Figure 2.4: Visualization of a stack of dilated causal convolutional layers (from [11])

In this way the necessary number of layers is significantly lower than in the classic model.

2.2 Tacotron

The network is composed of an encoder and a decoder, using a Short Time Fourier Transform (STFT) [6] to predict the Mel Spectrogram [13].
First the encoder is run, which takes in input each character, where each character is

represented as a 512 dimensional vector of character embedding. Already learned, the embeddings is then passed through a 3-layer convolution network, using a 5x1 filter, and each layer exceeded by batch normalization through a reLU activation function, the goal of this convolutional network is to recognize n-grams, that is, phonemes, syllables and words.

The result is then passed into a bi-directional LSTM with 512 units, which produces the encoded features, to be passed to the decoder.

Before arriving at the decoder, however, the encoded features are passed in an attention unit [8] that converts the features in a fixed-length context vector, of size equal to 128, which is the internal feature representation to be sent to the decoder.

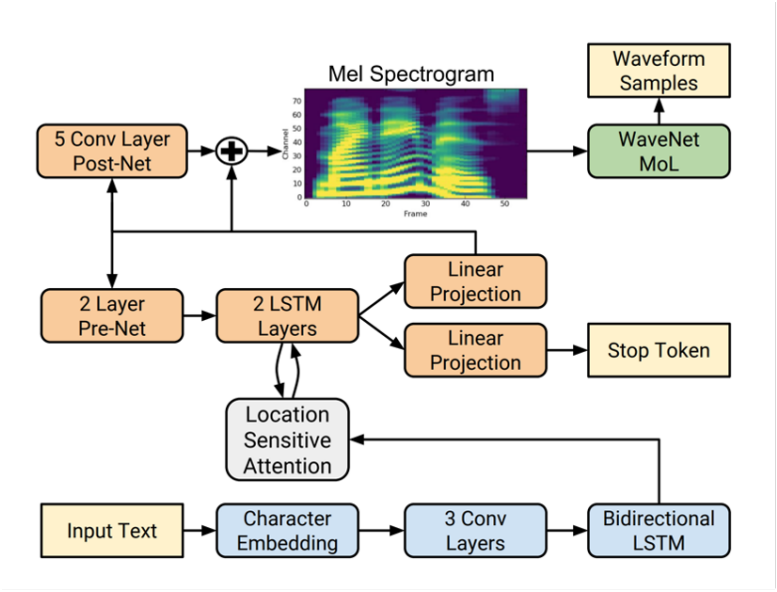

Figure 2.5: Block diagram of the Tacotron 2 system architecture (from [17])

which consists essentially of two fully connected layers, each with 256 neurons and a reLU activation function.

Now the output of the Pre-Net and the internal feature representation are concatenated by 2 uni-directional LSTM layers with 1024 neurons.

The output is then projected in a linear transformation to obtain the spectrogram frame, which is finally enhanced through a Post-Net, composed of 5 convolutional layers.

System	MOS
Parametric	3.492 ± 0.096
Tacotron (Griffin-Lim)	4.001 ± 0.087
Concatenative	4.166 ± 0.091
WaveNet (Linguistic)	4.341 ± 0.051
Ground truth	4.582 ± 0.053
Tacotron 2 (this paper)	$\mathbf{4.526 \pm 0.066}$

Figure 2.6: Mean Opinion Score (MOS) evaluations with 95% confidence intervals computed from the t-distribution for various systems (from [17])

2.3 Generalized End-to-End Loss for Speaker Verification (GE2E)

The generalized End-to-End Loss [18] Function is an evolution of the Tuple-Based End-to-End Loss Function [21].
It is used in speaker verification, and, compared to its predecessor, it manages to reduce the ERR by 10% and simultaneously reduce the training time by 60%.
The purpose is to understand if an utterance belongs to a speaker, based on a series of known utterances of that speaker. This model usually belongs to two categories; the one that interests us is TI-SV, that is text-independent speaker verification.
The application procedure is to divide the utterances in batches each composed of M utterances and N speakers, the features of the utterances are extracted and passed in a LSTM network with an additional linear layer, then the output undergoes an L2 normalization, to produce the embedding vector, which in the case of TI-SV has a projection size of 256.
We denote $e_{j,i}$ as the embedding vector of the *j*th speaker and its *i*th utterance. Then we consider the centroid c_j of speaker *j* as:

$$c_k = \mathbb{E}_m[e_{km}] = \frac{1}{M}\sum_{m=1}^{M} e_{km}$$

Now we can define the similarity matrix, which represents the similarity of each embeddings with all centroids of each speaker, and is defined as scaled cosine similarity,

the greater the value it takes the higher the probability that this utterance belongs to that specific speaker. It was also observed that during training more stable results were obtained by removing the $e_{j,i}$ embedding vector from the centroid, when comparing the $e_{j,i}$ vector to the centroid of its real speaker.

$$c_j^{(-i)} = \frac{1}{M-1} \sum_{m=1, m \neq i}^{m=1} e_{jm},$$

$$S_{ji,k} = \begin{cases} w \cdot \cos\left(e_{ji}, c_j^{(-i)}\right) + b & \text{if } k = j; \\ w \cdot \cos\left(e_{ji}, c_k\right) + b & \text{otherwise.} \end{cases}$$

The result we want to obtain is therefore that the embedding of an utterance is spatially closer to the centroid of its speaker and spatially farther from the centroid of the other speakers.
To do this we use the Softmax function [27] that returns an output equal to 1 if $k = j$ where $k = 1, ..., N$, and returns an output equal to 0 if k is different from j. Accordingly, the loss function on an embedding vector can be defined as:

$$L(e_{ji}) = -S_{ji,j} + \log \sum_{k=1}^{N} \exp(S_{ji,k})$$

This loss function means that we push each embedding vector close to its centroid and pull it away from all other centroids.

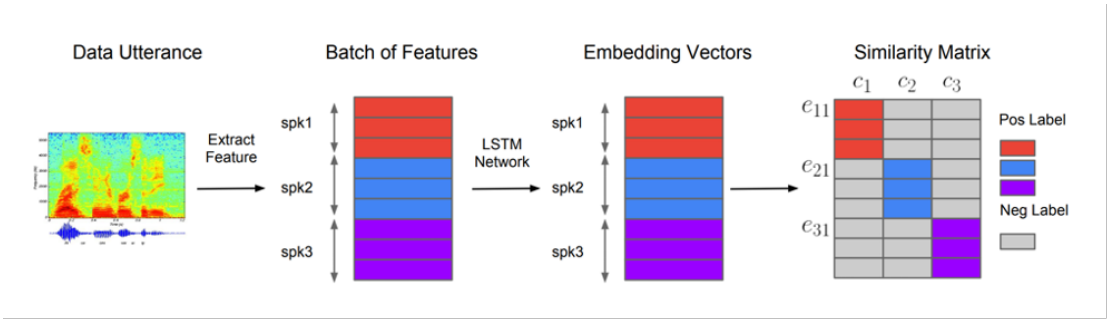

Figure 2.7: GE2E system overview. Different colors indicate utterances/embeddings from different speakers. (from [18])

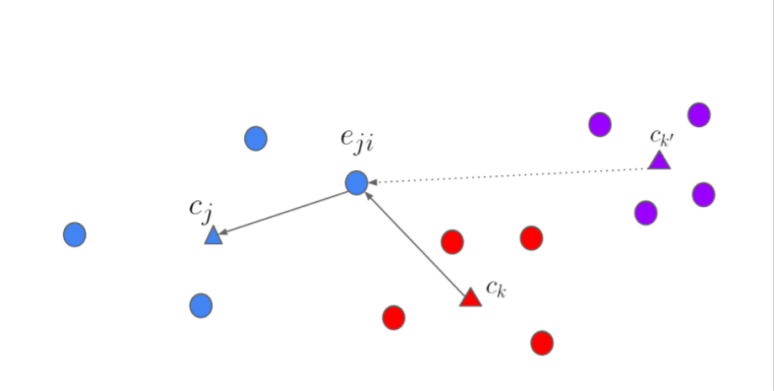

Figure 2.8: GE2E loss pushes the embedding towards the centroid of the true speaker, and away from the centroid of the most similar different speaker. (from [18])

Chapter 3

Speaker Verification to Multispeaker Text-to-Speech

The system we have implemented, which is described in [14], consists mainly of the union of the 3 projects described above: GE2E [18], Tacotron [17] and WaveNet [11], implementing them in a framework that allows zero-shot voice cloning with basically only 5 seconds of reference speech.
The results obtained by the authors of the paper are particularly surprising. Our intent is to reproduce the framework, implementing it with the help of open-source code provided by Corentin Jemine [19], but on the Italian language and phonetics, using datasets of Italian speakers, with a view to obtaining results that are at least comparable to those of the original publication.
The project framework is divided into 3 models: the speaker encoder, the synthesizer and the vocoder.

The speaker encoder produces a vector embedding through a short speech audio of a single speaker, the vector embedding describes the vocal characteristics of the speaker, the model used was structured thanks to what has been described in GE2E [18]. In fact, the purpose of the authors of this paper is that of voice recognition, in that, if the embeddings generated by two utterances are close in latent space, the audio were probably produced by the same speaker.
The embeddings produced by the encoder are perfect to condition the synthesizer to generate a spectrogram that has the same vocal characteristics of the chosen speaker.
The synthesizer instead is based on the model described in Tacotron [17], which generates a spectrogram starting from a text, and is conditioned with the embedding produced by the encoder.

Finally, the vocoder, which follows the model described by WaveNet [11], produces an audio waveform from the spectrogram generated by the synthesizer.

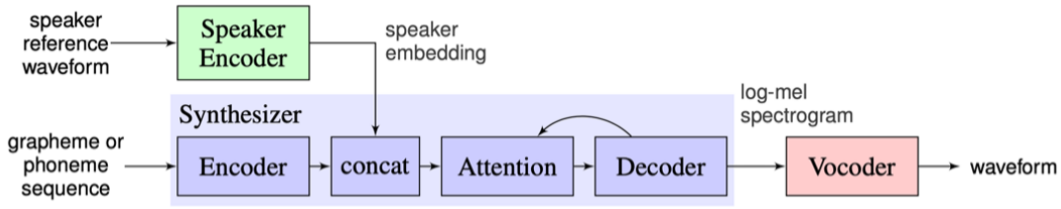

Figure 3.1: System overview. Each of the three components are trained independently. (from [14])

This framework allows one to train the 3 models separately, in a sequential process, which allows each model to be trained with different dataset. It is a useful feature to be able to train the encoder on a dataset which is not too ”clean”, with reverb and background noise, since we imagine that when we use the model, the reference audio of the speaker on which we want to do the voice cloning will not be of very good quality. Precisely, this, however, does not prevent us from training the synthesizer on the other hand on a dataset of clean audio, in order to improve performance in the generation of the spectrogram.
In any case, even if this feature could be useful, in my implementation I will always use all the data I have at my disposal, given the relatively small amount available compared to English language datasets.

3.1 The training process

As previously mentioned, the three models are trained separately, but following a sequential process; in fact, the encoder can be trained individually, while the synthesizer needs the trained encoder to be trained, and the vocoder needs the trained synthesizer, and for this reason we must proceed in this order of training.

The encoder is trained by dividing the dataset into train-set and test-set for each speaker, in order to have for each one both utterances in the train and in the test set, the audio of utterances is then converted into mel-spectrogram and sent to the speaker encoder, which produces the embeddings, which are then passed by GE2E [18] loss to calculate the gradient vector, sent in back propagation to adjust the weights of the model.

Once we have the model of the encoder ready we can start to train the synthesizer,

which receives as input the embedding of an utterance produced by the encoder, and its transcript. The synthesizer as output produces mel-spectrogram, which is compared with the original one by a Loss Function, which returns the gradient vector.

Finally, we can train the vocoder which takes in input the spectrogram predicted by the synthesizer, and predicts the audio waveform, even this is then compared with the original audio by a Loss Function, which allows one to adjust the weights of the model through a gradient vector.
In our implementation we used the same datasets to train the encoder as well as the synthesizer and vocoder, given the limited amount of data available in Italian, although it would have been more effective to use different datasets, as we expect the encoder to have to deal with ”dirtier” audio, whereas clean, good quality audio were perfect for the synthesizer and vocoder to get better results in audio generation.

3.2 Mel-Spectrogram

First of all, I think it is important to define the data we are working with. In fact to train our models we use mainly 3 types of data: audio utterances of speakers in wave forms and in form of mel-spetrograms, and the related transcript in form of simple text.
In this section I want to clarify the ideas about what an audio is in the form of mel-spectrogram [1], [22].

A spectrogram is represented on a Cartesian plane, where the abscissas represents time, while the coordinates represent frequencies, in general the audio we use has a maximum frequency of 16 thousand hertz, so each point of the spectrogram represents a certain frequency at a given time of the audio, and each point takes a value that represents the intensity of that frequency at that time, usually this intensity is represented either with a grey scale or a color scale of different shades.
The problem comes when we consider these intensities of frequency in a linear way. In fact, we realized that the way in which humans perceive audio frequencies is not exactly linear - for example a difference of 200Hz at a low frequency is perceived much more than the same difference at a high frequency, or we perceive as being more similar two high frequencies rather than two low frequencies that differ by the same amount.
So humans perceive frequency logarithmically, and we want a perceptually-relevant amplitude representation and perceptually-relevant frequency representation, and all this is made possible by the mel-spectrogram.
To obtain a mel-spectrogram , we convert all frequencies using the mel scale, proposed in 1937 so that equal distances would sound the same to the ears of a listener.

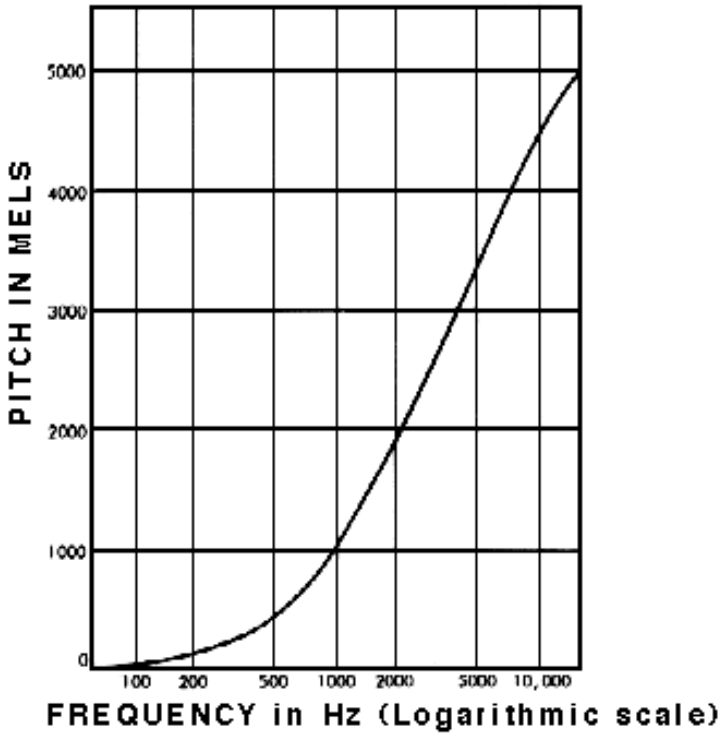

Figure 3.2: The mel scale as a function of frequency (from [1]).

3.3 Lamda Labs – Cloud GPUs

Regarding the training of the models, initially I tried to use my macbook pro, running the training on the CPU, but I immediately realized that it would take too much time to train all three models on my hardware, so much so that a training step of the encoder took about 2 seconds, and in total I trained that model for 250k steps, which on my computer would have taken about 130 consecutive hours.
Considering that the encoder is moreover the ”lightest” model of the three, I immediately decided to move the training to a virtual machine in the cloud, which would provide access to some of the most powerful GPUs for machine learning.
There are several platforms that provide this service, certainly among the most famous are that of google and amazon, but I found it very difficult to get permission to use these platforms, so I decided to rely on a startup called Lamda Labs [23], which has very competitive prices, has available GPUs among the best for ML, and a very user-friendly interface implemented on JupiterLab.

For encoder training I opted for an RTX 6000, while for synthesizer and vocoder training I used an RTX A6000, which is practically twice as fast as the previous one.

Obviously, thanks to the use of this platform I significantly decreased the time required to complete my models.

Chapter 4

The Encoder

As previously stated, the encoder is the first of the three models that we have trained, and, thanks to this, we produce what we call the embedding of the speaker, which is a 256-dimensional vector describing the vocal characteristics of a given speaker.

4.1 LSTM (Long short-term memory) Model

Before describing in detail the model that I implemented to obtain the encoder, it is important to describe what an LSTM model is, as that is the framework we used for our model, and is very efficient in speech recognition [7].

The LSTM model is a successor of the recurrent neural networks (RNN). RNN models have become increasingly popular in the world of machine learning; in fact, they are essentially like a series of classic neural networks, where each neural network passes to the next a ”message” in which it transfers information about its execution, in this way it is as if step by step the model keeps a memory of previous executions.

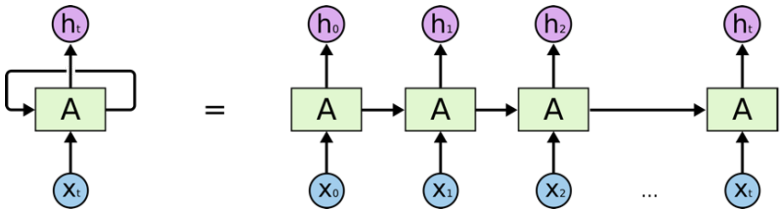

Figure 4.1: An unrolled recurrent neural network. (from [9])

The main problem of RNN models is that they are not able to use long term dependencies, i.e. they have difficulty in using important information that has been passed long

before by the task that it is trying to execute in that particular moment; the greater the distance between the information and the moment in which it is needed, the worse the performance of the model.

This problem is resolved from the LSTM models, in synthesis to each step the message to pass to the successive comes modified, before eliminating the information that the model considers useless and ”emphasizing” those that it considers useful, and subsequently adding the new useful information generated from the present step.
In this way the flow of information that goes through each step is continuously updated so as always to take into account the useful information even in the long term, and the more you move away from a step the more you will keep only the useful information of it, eliminating that which is superfluous.

4.2 The Encoder Structure

To generate my models, I used the same architecture proposed in Corentin Jemine's publication [19].
As mentioned, the framework used for the encoder is a LSTM, with three layers, with 256 hidden sizes per layer.
The input of the model is a mel-spectrogram with 40 channels, which are the frequencies it takes into account, with a window length, the range of frequencies to display, of 5ms, and a step, the distance between the centers of subsequent frames, of 10ms.
In the end the model presents a ReLU layer, which is later L2-normalized to produce a 256-dimensional vector, which is our output, the embedding.

4.3 Data

The biggest problem in training the models was to find quality and substantial datasets; in fact, in the Italian language there are unfortunately few public datasets available. The best I found are 2, the M-AILABS [24] Speech Dataset and the Common Voice Corpus [25]. The first is based on LibriVox a platform of free audiobooks from which the voice clips have been extrapolated to compose the dataset, where each clip is accompanied by the textual transcript; in total about 128h are recorded, all audio are in wav-format and are all at 16000Hz.
While the Common Voice Corpus was formed by Mozilla, with the aim of making datasets of open-source quality available in order to promote innovation in the field

of machine learning, which is also continuously updated to the extent that during the writing of my paper a new version was released that has expanded the previous one by about 100 hours. Fortunately I had not yet trained my models and I was able to take advantage of it. The Italian Common Voice dataset consists of a total of 288 hours, the audio format of the recordings is MP3 that I later converted to wav, and even these are all at 16,000Hz, and also in this dataset each audio is accompanied by the transcript.
To train the encoder I used both datasets together, grouping the audio by speaker.

4.4 The Model

I trained the encoder on the basis of the GE2E [18] process I described earlier, then on the basis of a speech recognition procedure, where the model is able to determine the speaker of an utterance based on the spatial proximity of the generated embedding and the known speaker's embedding.
My goal is to exploit this system to produce meaningful embedding, to be used to condition the synthesizer in reproducing the vocal characteristics of a given speaker.

My model is trained on batches of 10 utterances per speaker and is composed of 64 speakers, where the utterance duration is fixed at 1.6s, which are portions of the original utterance.
When we proceed to use the model, the interference is in real time, the embeddings are produced on a single utterance and not on a batch, and the utterance that we process can have variable length. We can deduce, however, that the production of embeddings at the completion of the model is more efficient using utterances that have the same duration as those used during training, for this reason before passing the utterance to the model we divide it into several segments of the duration of 1.6s, where each segment does not begin at the end of the previous one, but from its middle, in order to ensure greater accuracy in identifying the correct embedding.
Each segment is then passed to the model, and the final output is the average of all embeddings of the individual segments.

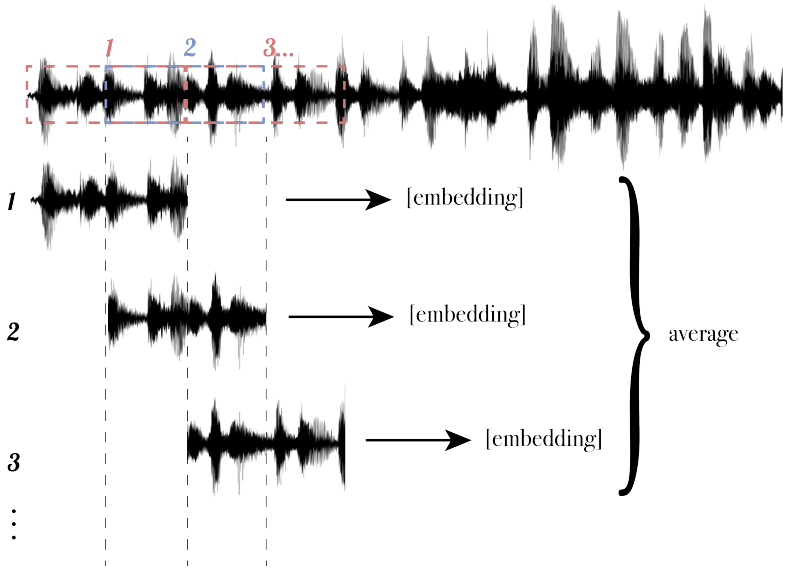

Figure 4.2: The segmentation procedure of the audio wave-form at inference time.

4.5 UMAP

Before talking about my experiment I wanted to devote a few lines to explain the technique I used to display the results of the embeddings produced during the training period, namely Uniform Manifold Approximation and Projection [16] (UMAP), for dimension reduction of the embedding vector.

UMAP is a dimension reduction technique, and our goal is to use this technique to visualize in a two-dimensional space the spatial distance between multiple embeddings, which as we know are points in a 256-dimensional space. What we expect to see is that similar embeddings produced by the same speaker are ”closer” to each other and that the embeddings produced by different speakers are further away.
UMAP allows us to do this; the technique essentially consists of creating a graph in which the embeddings are connected with weighted links, the greater the weight the greater their proximity in two-dimensional space.
In order to create the connections a system of identification of the neighbouring points in an high dimensional space is exploited, to do this a hypersphere is expanded for each point, with variable radius, until it meets another point, as soon as it meets another point a link is created, the smaller was the size of the radius the greater will be the weight of the link. Obviously, the procedure behind this technique is much more complex and is

supported by a strong mathematical theory, but in general this is the approach.

4.6 Preprocess

Before training our model I applied some preprocesses to the audio. Firstly, I eliminated the moments of silence within the utterance with a Voice Activity Detection technique [12], implemented by a python package called webrtcvad, which for each frame assigns 1 if it is a moment of speech and 0 otherwise. Of those segments of audio in which no voice was detected I eliminated only those with a duration greater than 0.2 seconds.
Later, in order to adjust the variation in volume within an audio I normalized it, and then I converted the audio into the mel-spectrogram.

4.7 The Training

After having preprocessed our audio I went to train my model, each training step is processed a batch, I made sure that every 10 steps I updated the information regarding the training process, every 100 steps I produced the projection in a two-dimensional space of 100 embeddings of 10 different speakers, 10 for each speaker, using the UMAP [16] technique, to visualize the progress of my model.
My model started converging after around 200k steps; in total I performed 229k training steps, which took about 18 hours on the RTX 6000 that I rented in the cloud.
I got very satisfactory results, and I was surprised that my model started to converge so early, since as far as I had read in the publications of the other authors who implemented this model in English, the training times were much longer, for example [19] trained the model for 1M steps, while [14] trained their model for 50M steps.

Certainly the first reason for this phenomenon is the difference in the size of the dataset used. Unfortunately, as previously mentioned, there are very few public datasets available in Italian, so it is clear that the small size of the dataset that I used led the model to converge earlier. In addition, I think that another possible reason is related to the language, as I believe it is possible that it is easier to extrapolate the vocal characteristics of a person and distinguish them from those of another with Italian phonetics rather than with English ones.
In particular, the encoder improves as the number of distinct speakers within the dataset increases; during my training I had 1,424 different speakers, while for example [19]

used three English datasets that reached a total of 8,400 speakers, while [14] implemented a private dataset composed of 18,000 speakers. A possible solution that I have not adopted due to lack of time, since it would have required much longer training time, is to use multilingual datasets - in this way you can reach large sizes, but there are not many publications on this possibility and it is not clear whether the use of a mixed dataset produces good performance. In my opinion, for the same size it is certainly better to use a monolingual dataset, but as in my case where the data available is limited I believe that an encoder trained on a mixed dataset of a large size can improve the performance of the entire system.

To view the progress of the training I used a visdom server, a python package that allows you to host a server from your own machine that allows the visualization in real time of some information about the training.
Thanks to the visualization of the trend of the loss function, it was easier to understand when the model was converging.

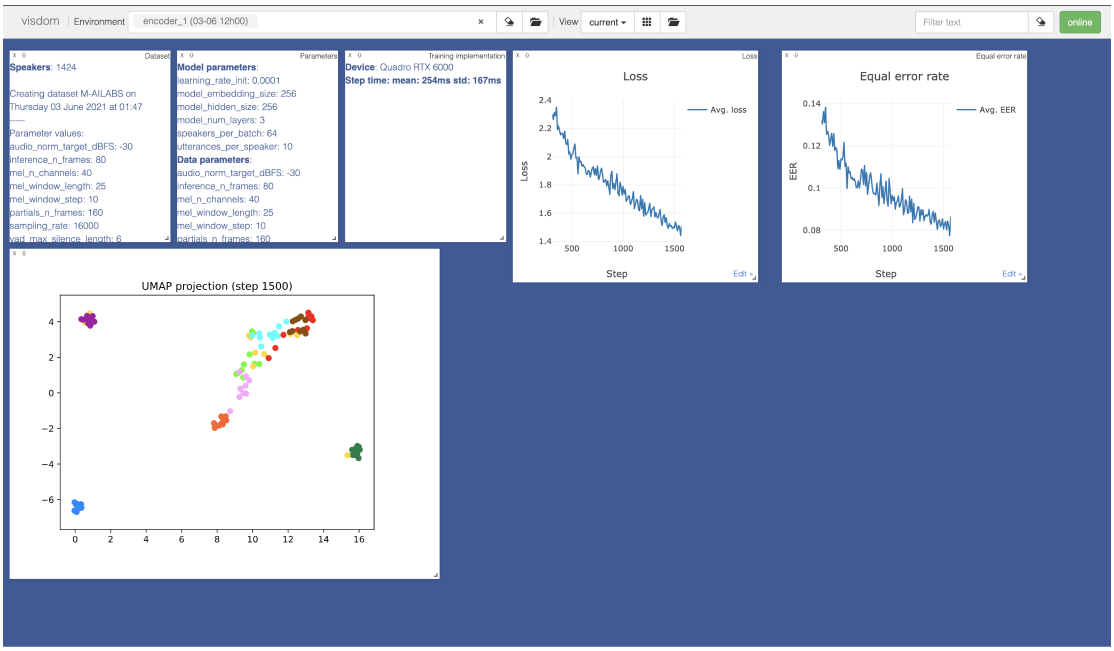

Figure 4.3: Screenshot of the visdom enviroment page, taken after the first 1500 steps.

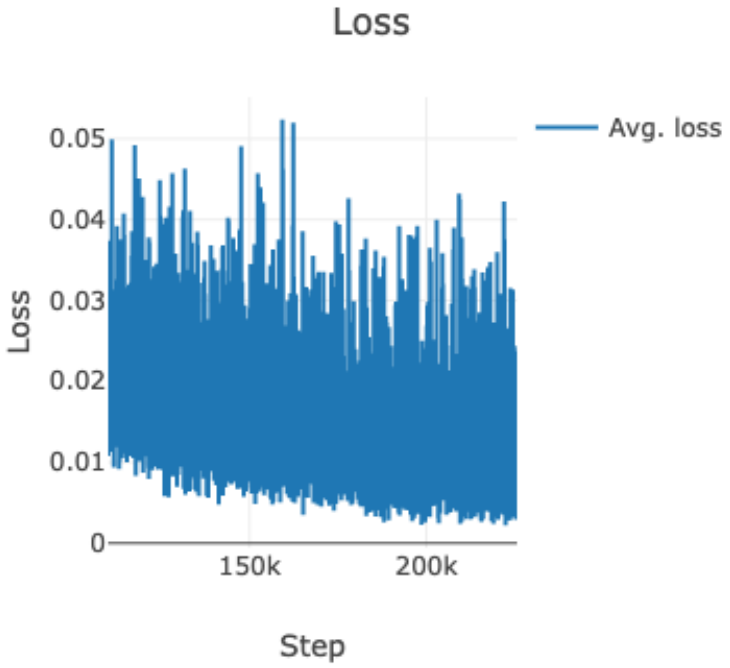

Figure 4.4: The graph of the trend of the loss value, between 120k steps and 220k steps.

The results that can be observed in the UMAP projections are also quite surprising. In fact, after about 100k steps it becomes evident that the separation of the various speakers becomes very efficient, as the utterances of the same speaker, which in the display are indicated with the same colour, have a very small spatial distance, while the utterances of different speakers have a spatial distance which is well-defined and consistent.

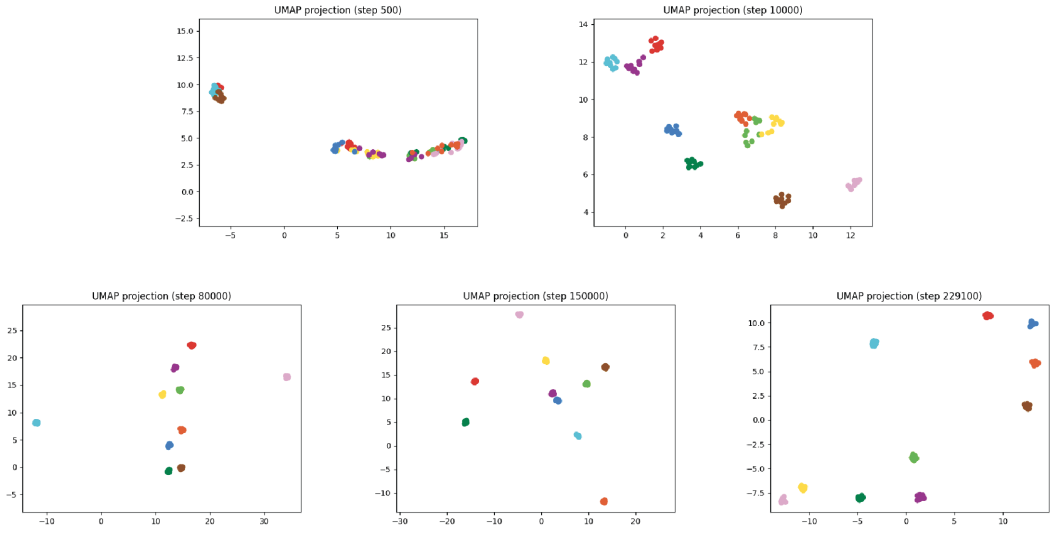

Figure 4.5: My UMAP projections at 500, 10000, 80000, 150000 and 229100 steps

At the end our model reached an Equal Error Rate (ERR) of about 0.002 that, compared with the results of other authors, is surprisingly good, but, as I said, and as we shall see at the end when we have completed all the models, our encoder does not perform so well for our purpose.
The ERR is a measure to evaluate the system performance, the value indicates that the proportion of false acceptances is equal to the proportion of false rejections, lower the ERR higher the accuracy of the model.

Chapter 5

The Synthesizer

The model of the synthesizer that I built is the one described in tacotron [17], which I have already analyzed in section (2.2).
The only difference implemented is the one proposed by the authors of [14], which is described in figure [3.1], which is that, before passing the frames coming out from the synthesizer encoder, they are concatenated with the target speaker embedding produced by our speaker encoder that we have previously trained, and then pass everything to the Location Sensitive Attention unit, which, as we know, produces the internal feature representation to send to the decoder.

Therefore, the inputs of the model are the embedding that produces our encoder, and a string of text.
The output is a mel-spectrogram which reproduces the text, thanks to the conditioning of the embedding taken as input.

5.1 Data

The data that I used to train this model are the same that I used for the encoder. Unfortunately, it is easy to think that this is not a suitable choice, since all the speakers are already ”known” by the encoder, in fact, the best procedure would be to use different datasets to train the two models, and, in particular, it would be even better that the dataset to train the synthesizer are as clean as possible, so with the least amount of background noise and maybe with recordings produced by professional equipment.
As we have already said, however, the data at my disposal are very limited, so I was forced to reuse the same dataset for this model, unlike the other authors who were able

to take advantage of different data of a higher quality.
Moreover, I believe that this is precisely the cause for which, as we will see later, the final result is not optimal, which is therefore an additional reason to consider the use of a multi-language dataset for encoder training.

5.2 Preprocess

I followed the steps proposed by [19], although I did not align the text to the audio, since I did not find a pretrained model with Italian phonetics for the alignment, which would have required to train a new model to do this, and due to a lack of time I decided to skip this procedure; in any case, this could be a future implementation to improve the performance of the model, which is, however, already satisfactory.
Since I did not align the audio with the text I could not split the utterances in the silent moments, as proposed by [19], so I just eliminated the silent moments exactly as done in the encoder, and skipped the utterances that were too short, those lasting less than 1.6 seconds, and those too long, lasting more than 11 seconds; lastly, I converted the audio to mel-spectrogram.
Moreover [19] implemented a noise reduction algorithm, which improves the performance of the produced mel-spectrograms by making them cleaner, to do this it exploits the moments of silence within the utterance and derives the noise, I realized that this procedure does not perform very well without slipping the text in the moments of silence, so I decided to skip this practice too.

Before moving on to the training of the model I produced the embeddings for each utterance, although it is reasonable to think that it is better to derive the speaker embeddings by making the average of the embeddings of all utterances produced by that speaker, when we are going to use our model it will have only one utterance to derive the embedding, so I proceeded with the training of the model with the embeddings of the individual utterances. There are enough utterances lasting about more than 3 seconds to produce embeddings that describe the embeddings of the speaker well.

5.3 The Training

After preprocessing the data, I passed to the training phase. In total I trained the model for 35k steps, on an RTX A6000, which took 1.29s per step, with a batch size of 40, and a reduction factor of 2, which is the number of mel frames synthesized for each

iteration.
During training, every 1,000 steps I would save the model and save on the disk of my machine the audio target of that step and the generated audio, the generated audio corresponds to the mel-spectrogram converted to wav-format using the Griffin-Lim Algorithm [2] which works as vocoder, a mathematical approach to convert a spectrogram into audio. Obviously, the result is not great since we are not using a real vocoder model but it manages to give an idea as to how the synthesizer is behaving.
I also displayed the two mel-spectrograms, the target one and the generated one, and the alignment between the encoder steps and the decoder steps.
In this way I could keep track of how my model was behaving, listening to the generated audio to see if it matched the target one, and comparing mel-spectrograms.

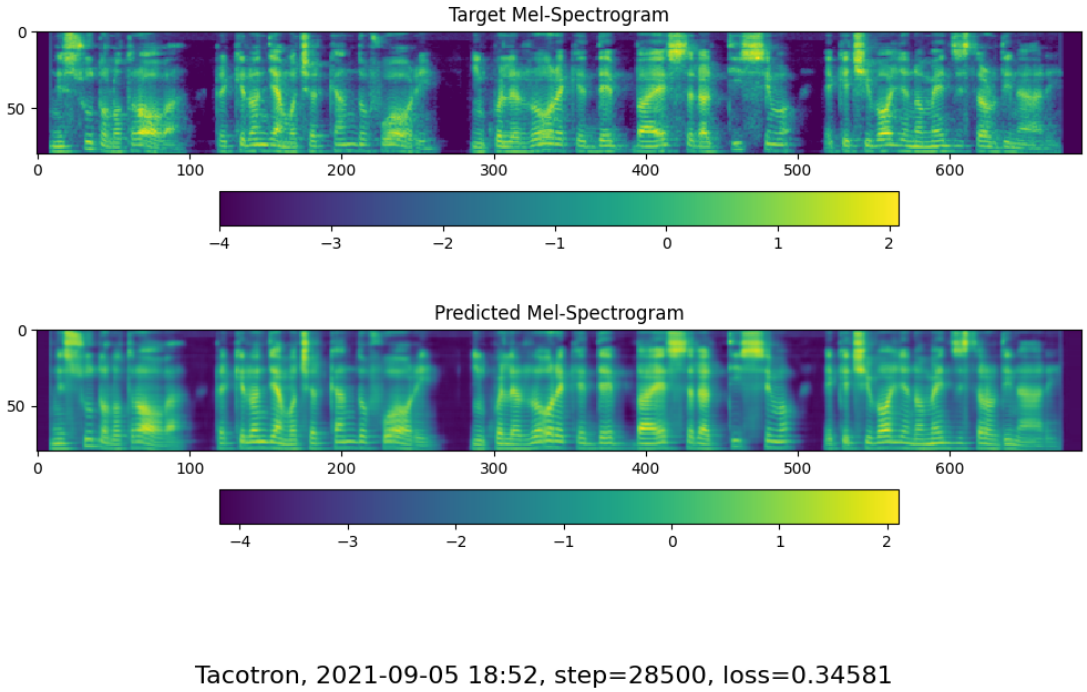

Figure 5.1: the visualization of the two spectrograms, the target one (top) and the one generated by the model (bottom), those in figure are those relative to the 28500 step.

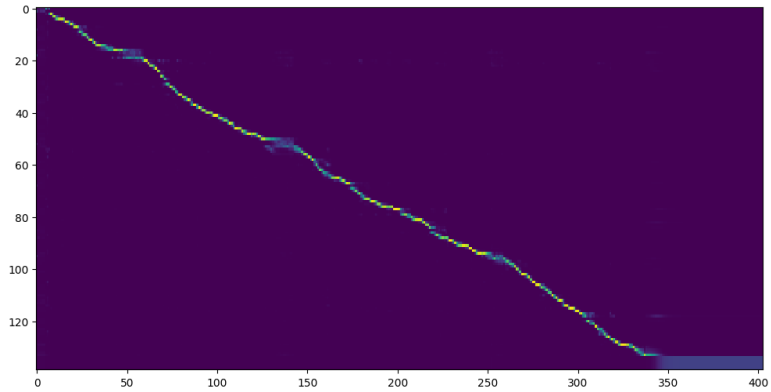

Figure 5.2: The alignment between the encoder and the decoder steps, at 28500 step.

At the end of my training the model reached a value of Loss equal to 0.33. Probably, by continuing to train the model we could have reduced this value to some extent, but the result obtained was quite satisfactory. The model, however, unfortunately, produces a little too much noise, which can be reduced in part by the vocoder, but not entirely - probably, working with the data better during the preprocessing phase can partially solve this problem.

Chapter 6

The Vocoder

The vocoder is the last model of our system. This model is the one that allows to transform the spectrograms generated by the synthesizer in audio wave-form that reproduce a natural and fluid speech as that of a real person.
I used a revisitation of WaveNet [11] namely WaveRNN, proposed in [15].

6.1 The Model

As we have described in the section (2.1) WaveNet [11] is an autoregressive model, which means that every step, before being produced, is conditioned on all the previous steps. This simulates the process of natural generation of the voice very well, but is unfortunately very slow at inference time, especially at high frequencies.
For this reason I decided to use WaveRNN [15] that, in summary, reorganizes the target audio in n-batch, grouping each n-th sample together; in this way we will have n-batch that we can process semi-simultaneously. In fact, the first will work as the classic WaveNet in autoregressive mode, but, as we build the first batch, we can also complete the subsequent ones at the same time, considering that we will have the samples of the first to condition.
In this way the execution time will decrease with an increase in the number of batches in which we decide to divide the audio; obviously, the greater the number of batches, the less natural the output will be.

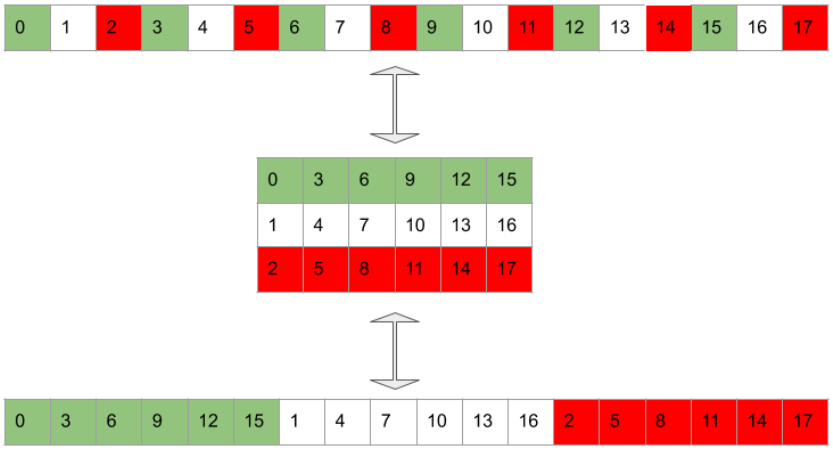

Figure 6.1: An example of the grouping procedure, with a 3 batch (from [20])

In the example in the figure, for instance, we will begin to generate the sample 0, then 3, and then 6 but at the same time also the 1 since we will have available the 0, and so on.

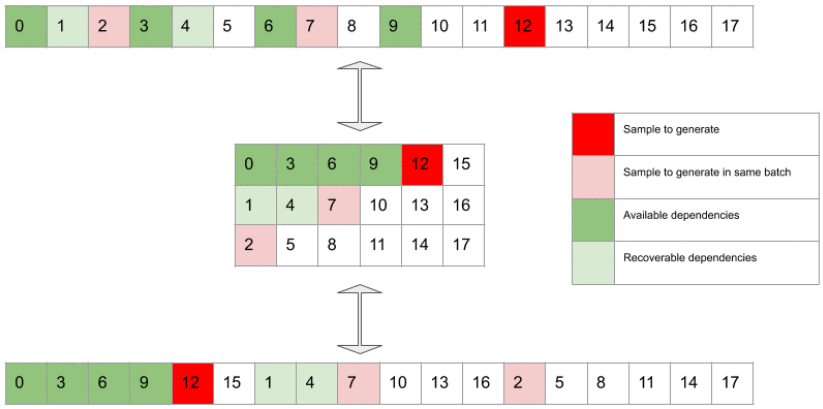

Figure 6.2: In dark green are the available dependencies, i.e. samples used to condition the network when generating a target sample (in dark red). The light green samples are recoverable dependencies relative to the dark red sample. These are samples that could in principle be accessed when generating the dark red sample because they have already been generated at previous steps. However, these are deliberately discarded by design choice. Light red samples are target samples that can be generated in the same batch in which the dark red sample is generated. Please note that samples corresponding to recoverable dependencies relatively to the dark red sample can be available dependencies for the light red target samples. (from [20])

After the upsampling process the whole is passed to two Gated Recurrent Unit (GRU) layers. GRU models are a variation of Recurrent Neural Network, very similar to LSTM models; in fact, they also have the purpose of solving the problem of long term dependencies.

Thanks to this system our model is able to run, for utterances, that are not too long, near real time.

6.2 The Training

The input of the model will be the mel-spectrograms generated by the synthesizer, while the output will be the audio in wave-form and is trained to produce 16khz audio as the synthesizer.
I trained the model for 600k steps, on a RTX A6000, reaching a loss value of 3.64.
The final result of this model is very satisfactory as it succeeds in part to eliminate the disturbance produced by the synthesizer and makes the generated voice quite fluid and natural.
The data used for training are always the same as for the other two models, but in reality even here you could easily implement datasets of other languages. In fact, the model should not be in no way influenced by the language - certainly, with a greater number of data and perhaps selecting only those of better quality one could get better results, but, as in the case of the other models, my time was limited.
Nonetheless, although even here there is room for improvement, all in all it seems to do its job fairly well.

Chapter 7

Streamlit Implementation

Once I finished all three templates, I wanted to find a way to create an easy-to-use interface that would allow their use for my purpose, cloning the voice with Italian phonetics. In order to do this I used a platform called Streamlit [26], an open-source app framework that allows you to create web apps from python scripts, by means of a large number of modular widgets available.
Once you have completed the web app script you can launch it from your terminal to make the web app accessible - your machine will host the server on which the app will run.
Firstly, the app will load the templates from a repository on the machine, then you can select an audio file to pass to the encoder to derive the embedding, which will then be displayed in the app, then you can, within a dedicated box, type the text you want to synthesize. At this point it is the task of the synthesizer and vocoder to generate the final output, and the audio generated will be directly playable through the app.

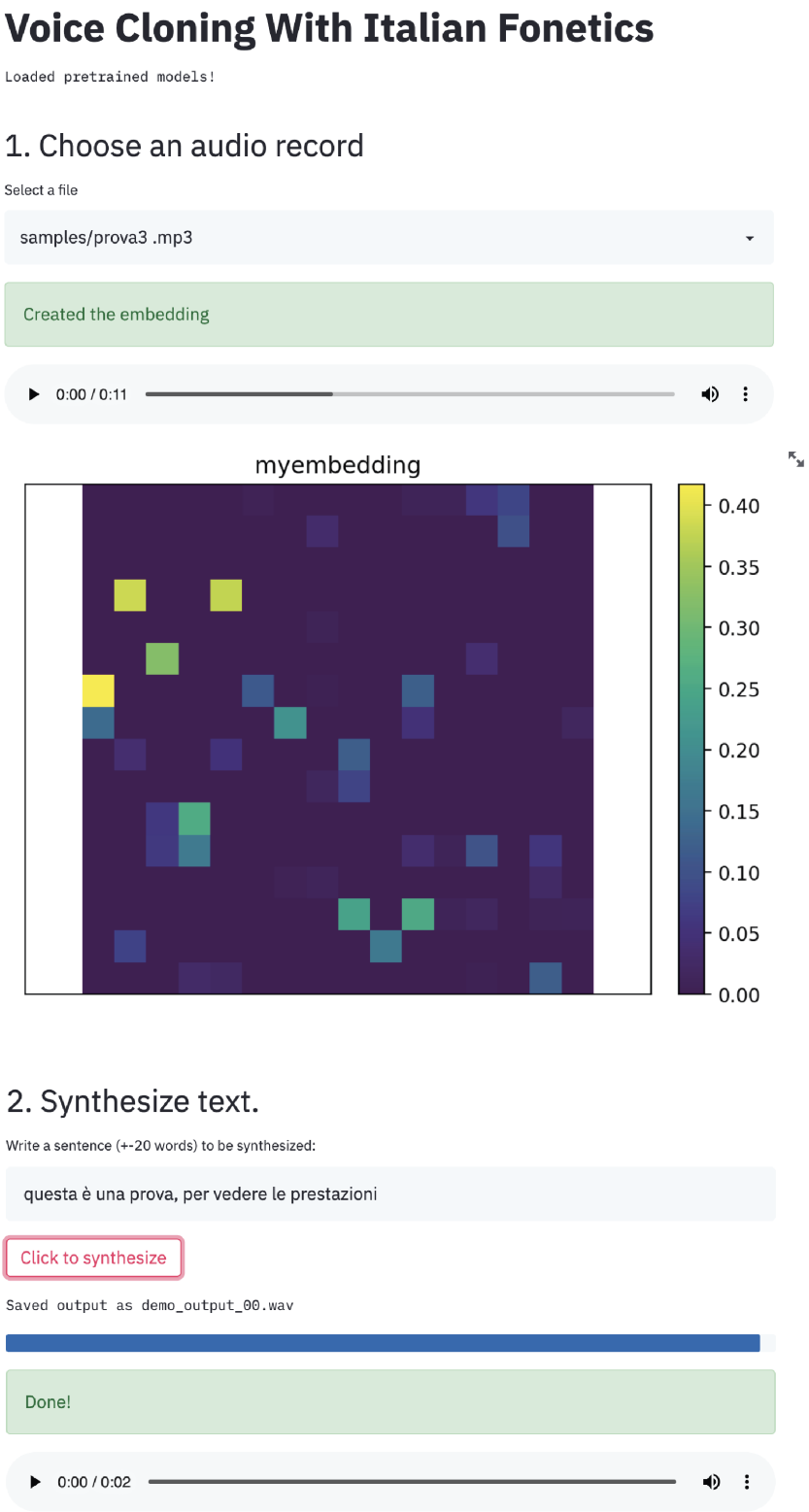

Figure 7.1: Screenshot of my stremlit web-app, after the execution of the models on an test audio and a test text.

Chapter 8

Conclusion

In conclusion I am satisfied with the project, as I was able to obtain satisfactory results considering the limitations in the quality and quantity of data available, and considering the limitations of time.
In any case, as I previously mentioned, we could obtain better results with some additional work. In particular, we could implement a multi-language encoder, in such a way that the model could be trained on larger and better quality datasets. In this way we could also train the encoder and synthesizer on different datasets. Another implementation, which could help to improve the final performance, is to improve the preprocessing of data before training the synthesizer.
These are just some of the improvements that could be made to the project, and I am particularly willing to make them, independently of the writing of this paper, considering the commitment dedicated, and the genuine interest that I have in the project.
I believe that the implementations of this system will be more and more successful over time, for example, in real-time translators, which will be able to reproduce the voices of two interlocutors, or in the world of audio books, where there will be no longer any need for voice actors to read them, or even, for example, for people who have lost their voice and use external systems to communicate, they could give such systems their voice back.
There is a lot of potential, and with time the models we use will be better and better, making it almost impossible for us to distinguish if the voice is real or produced by a computer. This is certainly fascinating, but at the same time it is also worrying, as it raises many ethical and digital security issues.
One thing is for sure: we will not be able to halt progress in the field of artificial intelligence, so it is our job to exploit it in the best and most genuine way, as these tools that

will completely change the way we experience the world.

References

[1] F. Geissler, J. Appleton, and R. Perera, "The development and practice of electronic music," 1975.

[2] D. Griffin and J. S. Lim, "Signal estimation from modified short-time fourier transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, pp. 236–243, 1984.

[3] D. O'Shaughnessy, L. Barbeau, D. Bernardi, and D. Archambault, "Diphone speech synthesis," *Speech Commun.*, vol. 7, pp. 55–65, 1988.

[4] T. Yoshimura, K. Tokuda, T. Masuko, T. Kobayashi, and T. Kitamura, "Simultaneous modeling of spectrum, pitch and duration in hmm-based speech synthesis," in *EUROSPEECH*, 1999.

[5] K. Tokuda, T. Yoshimura, T. Masuko, T. Kobayashi, and T. Kitamura, "Speech parameter generation algorithms for hmm-based speech synthesis," *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.00CH37100)*, vol. 3, 1315–1318 vol.3, 2000.

[6] N. Kehtarnavaz, "Digital signal processing system design," in, N. Kehtarnavaz, Ed., Second Edition. Academic Press, 2008, ch. 7.2 - Short-Time Fourier Transform (STFT).

[7] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *INTERSPEECH*, 2014.

[8] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *CoRR*, vol. abs/1409.0473, 2015.

[9] C. Olah, *Understanding lstm networks*, Aug. 2015. [Online]. Available: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[10] A. van den Oord, *Wavenet: A generative model for raw audio*, Sep. 2016. [Online]. Available: https://deepmind.com/blog/article/wavenet-generative-model-raw-audio.

[11] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," in *SSW*, 2016.

[12] J. Ko, J. Fromm, M. Philipose, I. Tashev, and S. Zarar, "Precision scaling of neural networks for efficient audio processing," *ArXiv*, vol. abs/1712.01340, 2017.

[13] Y. Wang, R. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio, Q. V. Le, Y. Agiomyrgiannakis, R. Clark, and R. Saurous, "Tacotron: Towards end-to-end speech synthesis," in *INTERSPEECH*, 2017.

[14] Y. Jia, Y. Zhang, R. J. Weiss, Q. Wang, J. Shen, F. Ren, Z. Chen, P. Nguyen, R. Pang, I. Lopez-Moreno, and Y. Wu, "Transfer learning from speaker verification to multispeaker text-to-speech synthesis," in *NeurIPS*, 2018.

[15] N. Kalchbrenner, E. Elsen, K. Simonyan, S. Noury, N. Casagrande, E. Lockhart, F. Stimberg, A. van den Oord, S. Dieleman, and K. Kavukcuoglu, "Efficient neural audio synthesis," *ArXiv*, vol. abs/1802.08435, 2018.

[16] L. McInnes and J. Healy, "Umap: Uniform manifold approximation and projection for dimension reduction," *ArXiv*, vol. abs/1802.03426, 2018.

[17] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Skerry-Ryan, R. Saurous, Y. Agiomyrgiannakis, and Y. Wu, "Natural tts synthesis by conditioning wavenet on mel spectrogram predictions," *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4779–4783, 2018.

[18] L. Wan, Q. Wang, A. Papir, and I. Lopez-Moreno, "Generalized end-to-end loss for speaker verification," *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4879–4883, 2018.

[19] C. Jemine, "Automatic multispeaker voice cloning," M.S. thesis, 2019.

[20] T. T. of Papercup, *Wavernn*, Jul. 2020. [Online]. Available: https://medium.com/papercup-ai/subscale-wavernn-f91fc9c7a106.

[21] H. Park, J. Park, and S. W. Lee, "End-to-end trainable self-attentive shallow network for text-independent speaker verification," Aug. 2020.

[22] L. Roberts, *Understanding the mel spectrogram*, Mar. 2020. [Online]. Available: https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram-fca2afa2ce53.

[27] T. Wood, *Softmax function*. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer.

Resources

[23] [Online]. Available: https://lambdalabs.com/service/gpu-cloud.

[24] [Online]. Available: https://www.caito.de/2019/01/the-m-ailabs-speech-dataset/.

[25] [Online]. Available: https://commonvoice.mozilla.org/it/datasets.

[26] [Online]. Available: https://streamlit.io/.