Unconditional Speech Synthesis as Sleeping Aid

Vincent Tonkes, Puja Fadte, Twan Vos

February 2023

Abstract— '*Maarten van Rossem - De Podcast*' is a popular podcast in the Netherlands that, in addition to its high acclaim, is used by many as a sleeping aid. The problem when using it in this manner, however, is that sometimes the topics being discussed are too interesting, such that one stays awake to listen till the end. In this report, we attempt to resolve this problem. We train a generative model on the podcast, such that it can imitate the soothing voice of the main speaker, but will not be able to produce audible words, and instead generates a sort of 'babbling'. To this end, we first need a way to extract fragments in which only the main speaker is talking. We create several temporal classifiers for this, that predict who is talking at a given moment. We find that an approach based on the `ResNet18` convolutional neural network performs best, with a testing accuracy of 98.5%. We use this model to create the aforementioned dataset, and use the `UNAGAN` generative adversarial network for our speech synthesis task. The results we obtained surprised us, as they were of a much higher quality than we expected.

Keywords— Machine Learning, Voice Recognition, Unconditional Speech Synthesis, Podcasts, Maarten van Rossem

CONTENTS

I. Introduction

One of the most popular podcasts in the Netherlands is '*Maarten van Rossem - De Podcast*'[1]. In it, retired history professor Maarten van Rossem (MVR), and producer Tom Jessen (TJ) discuss a wide plethora of subjects, ranging from historic events to current affairs.

Besides the high acclaim, many have commented that listening to this podcast can be helpful for falling asleep, with some even stating that they play it for their newborns at night. MVR his deep, soothing voice, and slow manner of talking are the likely causes of this strange phenomenon. He himself has often expressed that he finds these comments flattering. A good night's sleep, he then says, is invaluable. And if he is able to provide this for free, then that is much better than people having to take sleeping pills.

I (Vincent) can personally attest to this. While I have had sleeping problems my whole life, listening to this podcast has frequently made me fall asleep within 10 to 15 minutes. The only problem, however, is that sometimes the topics being discussed are too interesting, such that I stay awake to hear the end of it. In a sense, then, the content of the podcast stands in the way of the function it has at that moment.

For our semester project, we wish to address this problem. Inspired by Van Oord et al. [1][2], we want to train a generative model on the voice of MVR. The idea then is, that because the training data is presented in limited-sized windows (and, of course, because the model its capacity simply will not allow for anything else) the model will not be aware of any long-term coherence within the input. As such, the audio signal it produces will hopefully possess the acoustic properties of MVR his voice, but be stripped of any contents.

To solve this problem, we first need to extract a large number of fragments containing only MVR talking, from the podcast. Doing this by hand is not feasible, so we need to automate this process. To this end, we will create a temporal classifier that can determine who is speaking at any given moment.

We believe that especially this classification task is a good candidate for a semester project. We think this task will allow us to put our newly obtained Machine Learning knowledge to the test, and that it will provide us with enough opportunities to demonstrate that we are capable of making correct and insightful design decisions. The same will still be true for the generative task, but likely to a lesser degree. While we do understand what we are doing (which will hopefully become apparent in later sections), we must rely more on the hard labor of others here, by using pre-existing architectures instead of designing our own.

The outline of this report will be as follows. Since we are effectively solving two problems, most of the sections to come will also be split into two parts. After providing some background knowledge that is needed to follow along, we will describe our dataset and learning problem. Note, however, that because an important part of our project is about creating a dataset, one will find some parts here that might otherwise be expected in the 'Methods' section. In the 'Methods' section itself, then, we will describe the techniques used to solve the stated problems. Finally, in the 'Results' and 'Discussion' sections we will evaluate our work in a low-level quantitative, and high-level qualitative manner, respectively. Since much of our results will be audiovisual (yes, also visual), we will have to present much of our results in a supplementary blog post, which can be found at https://indooradventurer.github.io/Unconditional-Speech-Synthesis-as-Sleeping-Aid/[3].

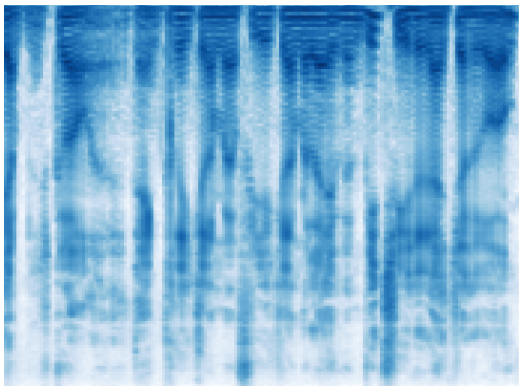

Fig. 1: A Mel-spectrogram representation of MVR talking. Darker shades of blue correspond to greater amplitudes on a decibel scale.

II. Background

Some of the contents that will be discussed in later sections rely on knowledge of spectrographic representations, generative models, and audio vocoders. Since these subjects are not part of the Machine Learning course, they will briefly be discussed below.

2.1. Spectrogram Representations

A discrete-time audio signal $u(t)_{t\in\mathbb{I}}$, is said to exist within the time-domain [2]. Taking the discrete-time Fourier transform (DTFT) over this signal gives an alternative representation, that exists within the frequency domain. While the former representation more easily shows how a signal changes over time, the latter one easily tells us which frequencies a signal is composed of.

A spectrogram representation, then, is the best of both worlds: it shows how the frequencies the signal is composed of change over time. It does this by first dividing the time-domain signal into smaller (overlapping) bins, and taking the DTFT over each of these individually. The found amplitudes are then stored in a 2D image-like array, where columns represent individual time windows, and rows the frequencies. Note that this means that any phase information is discarded. An example is shown in Figure 1.

A special case of such a spectrogram is the 'Mel-spectrogram'. In this representation, frequencies are converted to a logarithmic scale, called the 'Mel-scale', which more accurately depicts how humans perceive frequencies relative to one another [3]. Mel-spectrograms are widely used in Machine Learning. Both as input features for classification tasks, etc., or as the output of generative models,

[1] https://www.maartendepodcast.nl/

[2] See section "Knowing What to Say" on: https://www.deepmind.com/blog/wavenet-a-generative-model-for-raw-audio

[3] Also see our source code on https://github.com/IndoorAdventurer/ML_PodcastProject. Do note that we will make this repository private again after grading.

which can then be converted to an audio signal.

Another special case is the so-called 'Mel-frequency cepstrum' (MFC) [4]. It is computed from the Mel-spectrogram, by first converting all found amplitudes to a logarithmic scale, then taking the discrete cosine transform over each band (column in Figure 1), and again, storing amplitudes in a 2D array. The values in this array are then called MFC-coefficients, or MFCCs for short. MFCCs are known to work well with simple, and linear Machine Learning methods, and in addition, seem particularly well suited for voice recognition/speaker identification tasks.

2.2. Generative Models

Simply put, a generative model is a model that creates something. For audio synthesis, there are two major paradigms being used. The first is the *autoregressive* paradigm [1]. Here we treat the generative task as a time-series prediction problem, and iteratively feed the model back a fixed horizon of its own previous outputs. With this, it can then predict the next output, and then the one after that, etc., such that a complete audio fragment is generated. We will only briefly discuss such a scheme in Section 4.2.1.

The second paradigm is using a *generative adversarial network* (GAN) scheme. Here we use an architecture that is composed of two constituents. The first is called the *generator*. This model learns to sample new points from a distribution that is similar to the one the training data came from. Usually, this is done by making it produce outputs from random noise. The second constituent is the *discriminator*. This model receives either the generator its output as input, or examples from a training set, and has to determine which is which.

The zero-sum game the generator and discriminator are in, where the generator wants to minimize the discriminator its accuracy, while the discriminator itself wants to maximize this, can then lead to an upward spiral, where the generator its outputs become more truthful to the distribution of the dataset over time. Training with such a scheme, however, is very non-trivial and gives rise to many difficulties that prevent convergence. In Section 4.2.2 we will look at a specific GAN model that solves many such problems, which is derived from the `BEGAN` architecture [5].

2.3. Vocoders

Many generative models do not directly produce an audio signal, but instead, produce a spectrographic representation (see Section 2.1). Converting this to sound is called *vocoding*. This process is not straightforward, as much information is missing, such as the phase of individual sinusoidal components. There do exist clever algorithms for this task, such as the Griffin-Lim algorithm [6], but the best approaches seem to involve a neural network and are hence called *neural vocoders*.

III. Dataset and Learning Problem

For this project, we will consider all episodes of the aforementioned podcast, up to November 9th, 2022. In total, this regards 190 hours and 9 minutes of material, divided over 319 episodes. As a first pre-processing step, we convert all episodes into a 16-bit, 1-channel audio format at a sampling rate of 22050 Hz.

To be able to extract a large number of MVR fragments from this 'raw' data, we will first need to create and train a temporal voice recognition classifier. In the next subsection, we will give a formal specification of this learning task, and describe how we create a dataset for this purpose. After that, we explain how we intend to use this classifier to create a dataset for speech syntheses, and give a precise specification of this generative task as well.

3.1. Voice Recognition

At a given moment, in any of the episodes, either MVR is talking, TJ is talking, or neither one is talking. This last case consists of voice messages from listeners, supplementary audio fragments, the intro- and outro-tune, etc. What we need, is a model that can predict which of these three is the case, such that we can use this information to filter out everything except MVR[4].

To put this more formally, let $u_i(t) \in \mathbb{R}$ be the input audio signal (corresponding to podcast episode i), for discrete and finite values $t \in \mathbb{T}_i = \{t_n \in \mathbb{N} \mid t_n \leq N_i\}$, where N_i is the total number of audio samples in episode i. What we then want, is an output signal $y_i(t)_{t\in\mathbb{T}_i} \in \mathbb{L} = \{c_m, c_t, c_n\}$, where c_m, c_t and c_n are the class labels for MVR, TJ and 'Neither', respectively.

Many of the methods presented in the '*Machine Learning Lecture Notes*' by H. Jaeger [7], cannot easily (or at all) be adapted to such a temporal classification task. For that reason, we divide this problem into two subproblems. The first of these is finding a mapping $h : \mathbb{R}^M \to \mathbb{L}$, from fixed-length audio windows $\mathbf{x} \in \mathbb{R}^M$, where $M \ll N_i$ (for any i), to class predictions $y \in \mathbb{L}$, corresponding to the most dominant voice in $\mathbf{x}$. The second subproblem is utilizing this mapping function h to optimally solve the original problem.

To create a dataset, we pick one-second fragments and label these by hand. We do this by first randomly selecting an episode i, with a probability proportional to its length N_i, and then select a starting point from a uniform distribution over the range $[0, N_i - 22050]$ – recall that N_i is the total number of audio samples, at a sampling rate of 22050 Hz.

The complete dataset we construct in this manner contains, in total, 1000 data points. Of these data points, exactly 900 were labeled as MVR. 86 were labeled as TJ, and the remaining 14 as 'Neither'. It is clear, then, that this dataset is fairly unbalanced. Especially for the 'Neither' class, this is problematic, as it is also by far the most heterogeneous one. In Section 4.1 we will pay extensive detail to problems that emerge from this unbalance, and our methods for resolving these.

3.2. Speech Synthesis

With a good temporal classifier available, we can create a dataset for speech synthesis. We first obtain, for every episode i, all longest possible discrete windows $[t_s, t_e]$, such that $y_i(t) = c_m$ for for all $t \in \{t_s, t_s + 1, \ldots, t_e\}$. In other words, all windows where only MVR is talking. Instead of randomly

[4] Note that we do not choose to classify as either 'MVR' or 'Not MVR', since we might want to do interesting things with TJ his voice in the future.

drawing a subset of windows from these, we sort them by their lengths $t_e - t_s$ and pick the top 500 longest ones. The intuition behind this approach is that these longest windows will likely be of the highest quality, as it is especially the long monologues/stories that we find sleep-inducing.

The dataset produced in this manner, as said, consists of 500 audio files. In total, this amounts to 39 hours and 26 minutes of audio, with the longest fragment being 36 minutes, and the shortest 2 minutes and 42 seconds.

With this data, we want to train a model that can produce audio signals with the acoustic properties of MVR his voice, but without any contents or long-term coherence.

To make this more formal: each audio sample $u_i(t_j)$ in our dataset, for concrete time values t_j (and fragments i), can be seen as being obtained from a random variable (RV) $U_{i,j}$. For simplicity, we can pretend that each fragment was drawn from the same distribution. In other words, that for any i and i', the RVs:

$$\bigotimes_{j=1,\dots,N} U_{i,j},$$

and

$$\bigotimes_{j=1,\dots,N} U_{i',j},$$

where N is the minimum of fragment lengths N_i and $N_{i'}$, will have the same distribution, such that we can drop the i.

We can observe that the distribution of any RV U_i will depend in a complicated manner on all its predecessors, $U_{i-1}, U_{i-2}, \dots, U_1$. If, for example, MVR says at the beginning of an episode: "I want to talk about X", then it might be that later he will say: "Oh yes, we should still talk about X". Now, if for some fixed value N, p_{real} describes the distribution of RV:

$$\bigotimes_{j=1,\dots,N} U_j, \tag{1}$$

then what we want is a *sampler* for another distribution $p_{synthetic}$, which is similar to p_{real}, except that any U_j will only be conditionally dependent on a small horizon h of predecessors $U_{j-1}, \dots, U_{j-h}$. In other words, $p_{synthetic}$ should be the distribution we get by 'integrating away' the effects of RVs $U_1, \dots, U_{j-h-1}$, on U_j, for all U_j in Equation 1.

By doing all of this, we effectively filter out the effects the mind has on speech. The mind, after all, will be responsible for a large portion of long-term dependencies within the audio signal, with, for example, its capability of retaining that "we should talk about X", for so long. Short-term dependencies, on the other hand, will be dominated by simple physiology. If, for instance, at some moment we hear an "aah"-sound, then we can expect to hear something similar a few milliseconds later; simply because our lips and tongues can only move so fast. With this, then, we expect and hope that our method will produce a sort of 'mindless babbling', that still contains the soothing properties of MVR his voice.

IV. Methods

In this section, we present our methods for solving the stated problems. In line with the rest of this report, we first focus on the temporal voice classification problem that was described in Section 3.1, and after that treat the speech synthesis problem of Section 3.2.

4.1. Voice Recognition

In this subsection, we will explore a wide selection of methods tried for our voice recognition problem, and conclude with an explanation of our full classification pipeline in Section 4.1.5. Each method we tried had its own idiosyncrasies and required different ways of handling the data-unbalance problem discussed in Section 3.1.

For hyperparameter optimization, we always use a 5-fold cross-validation scheme, and when computationally feasible, perform a grid-search over the hyperparameters, where we simply try every possible combination of sensible values (in an efficient manner where, for example, we do not redo principle component analysis if we are trying different parameters for the classifier that comes after it). To obtain our final model, we retrain on the entire dataset with the found hyperparameters.

During cross-validation, we mainly optimize with respect to accuracy. Due to the unbalance of our dataset, however, this measure will be somewhat deceiving: 90% of our data was labeled as MVR, so any model that learns to always output c_m will already obtain a 90% accuracy. Balanced accuracy, which is defined as the *unweighted* average recall over all classes, will not be a good alternative. The reason for this is that just one correct classification more or less for the 'Neither' class, which is good for just 1.4% of our data, will have a substantial impact on this metric, making it less informative than plain accuracy.

There are, however, two other metrics that we consider, which are precision and sensitivity with respect to the MVR class. Precision tells us what portion of fragments that a model classified as MVR, actually are of MVR talking. This is important for us, as the goal is to create a dataset that consists only of MVR fragments. Sensitivity tells us what portion of data points that we labeled as MVR, were correctly classified by the model. Since we have 190 hours – so more than enough – of raw data, we should find the model with the highest possible precision, under the condition that sensitivity stays above a certain threshold.

All models take spectrographic representation as input. These will always be normalized to the mean and standard deviation of the training set. At inference time, we ensure inputs are first normalized to this same mean and standard deviation as well.

Finally, we use the `Python` programming language for our implementation. We rely on `Numpy` for mathematical operations; `Librosa` for audio operations, such as loading files and creating spectrograms; `Scikit-learn` for Machine Learning techniques and evaluation metrics, and `PyTorch` for the Convolutional Neural Network discussed in Section 4.1.4.

4.1.1. Mixture of Gaussians

For our first approach, we attempt to solve our classification problem of finding a mapping $h : \mathbf{x} \mapsto y$ (see Section 3.1) by somewhat explicitly modeling the joined distribution $p_{X,Y}$. We do this by approximating $p_{X|Y=c_m}$, $p_{X|Y=c_t}$, and $p_{X|Y=c_n}$ by three Mixtures of Gaussians (MoGs), which we then multiply by $P(Y = c_m)$, $P(Y = c_t)$, and $P(Y = c_n)$, respectively, to get $p_{X,Y}$.

Such an approach seems common for voice recognition

[8][5]. In addition, for us, it has an extra benefit that might help solve difficulties that arise from data unbalance: because we divide our problem into three subproblems, one for each class, we are able to regularise for each one separately.

As stated earlier, the 'Neither' class makes up only 1.4% of data, while also being the most diverse. If we approximate its corresponding distribution with a much lower number of Gaussians, say just a single one, then we force it to have a large standard deviation. Geometrically this means it will have a low peak at the mean, but thick tails. Consequently, in areas with many data points corresponding to MVR and TJ, we can expect it to have lower values than the other two MoGs while being higher anywhere else. This means inputs that are unlike anything in the training data will likely be classified as 'Neither'.

For our implementation, we found that MFCCs (See Section 2.1) work better as input features than values from a Mel-spectrogram. Moreover, reducing the number of dimensions by simply lowering the number of MFCCs, is preferred over principal component analysis (PCA) for dimension reduction. Cross-validation showed that approximately 40 MFCCs are optimal.

For some input $\mathbf{x}$, with corresponding MFC-representation $m(\mathbf{x})$, we treat each MFC frame $m_i(\mathbf{x}) \in m(\mathbf{x})$ as an individual data point. At inference time, for each frame $m_i(\mathbf{x})$ we calculate the log-likelihoods of the three MoGs, given $m_i(\mathbf{x})$. After that, we average these values over all frames.

We interpret the resulting average log-likelihoods as log-probability density values $\log(p_{X|Y=c_m}(\mathbf{x}))$, $\log(p_{X|Y=c_t}(\mathbf{x}))$, and $\log(p_{X|Y=c_n}(\mathbf{x}))$. We add to this the corresponding log-priors $\log(P(Y=c_m))$, $\log(P(Y=c_t))$, and $\log(P(Y=c_n))$. To get an output prediction, we then pick the class label that corresponds to the highest found value.

Found parameters We already mentioned that 40 MFCCs were preferred as input. The MVR distribution is best modeled by 23 Gaussians, which is also true for the TJ one. For the 'Neither' class we use a single Gaussian, as cross-validation indeed showed that this is preferred. All Gaussians are constrained to have diagonal covariance matrices. We experimented with a circular constraint for the 'Neither' MoG (i.e. where the covariance matrix can be described as the product of a scalar and the identity matrix) but did not find this beneficial. Finally, we treat priors $P(Y=c_m)$ and $P(Y=c_t)$ as additional free variables, with:

$$P(Y=c_n) = 1 - P(Y=c_m) - P(Y=c_t).$$

The optimal values found here are 0.29, 0.17, and 0.54, respectively. We should note that we optimize with respect to accuracy here, as we found precision w.r.t. MVR sufficiently high for these parameters as well. Also, note that we do not blindly pick the best configuration: we make plots and look at what the top 20 models have in common, to come to a sensible choice.

Lastly, we did a dangerous trick to boost performance. Because the smaller number of data points for the TJ en 'Neither' class was likely still causing problems, we created a second dataset of 1024 clips as follows. We first randomly selected 100 clips at a time, with the procedure described in Section 3.1. Next, we let our MoG-based model classify them, and discarded nearly all clips that were marked as MVR. After that, we labeled the remainder as usual. The result is a dataset that is more balanced but is no longer drawn from the 'real' distribution. In addition, it might reinforce some mistakes. In particular, clips that the model erroneously classified as MVR might often have been discarded before we saw them, such that this case is underrepresented in the new dataset. To alleviate this problem somewhat, we first lowered sensitivity with respect to MVR, by setting $P(Y=c_m)$ to 0.01, while keeping the ratio $P(Y=c_t)/P(Y=c_n)$ constant.

Despite these concerns, we find that using this extra data in the manner that will be described next, greatly improves performance. What we do is, for each of the k cross-validation splits $(train_k, val_k)$ we set:

$$train_k \leftarrow train_k \cup extra,$$

where *extra* is the additional dataset. This assures that we validate with respect to the true distribution, while still being able to take advantage of the extra, more balanced, data. It is important to mention that the hyperparameters described above, were found from this approach instead of the 'vanilla k-fold' one.

We use this same trick for all models discussed next. In the extra dataset, 585 clips were labeled as MVR, 381 as TJ, and 58 as 'Neither'. That MVR is also in the majority here, is because we were still trying to find a good rejection rate for clips classified as MVR along the way.

4.1.2. Ridge Regression

As another approach, we consider linear regression. We did not find literature that suggests this would work well but considered it as a simple baseline to compare other models against. As plain linear regression can be prone to overfitting and numerical issues, we use ridge regression instead. This is a variant of linear regression that adds an L_2 regularising term α to the optimization criterion, where α is a tunable hyperparameter.

Found parameters We find that using 40 MFCCs as input works best. Similar to the previous approach, we briefly considered PCA for dimension reduction, but also here find that this gives no added benefit. Moreover, we do not find that any L_2-regularization is desired, and set α to a value of 10^{-5}.

4.1.3. K Nearest Neighbours

K Nearest Neighbours (KNN) is a simple supervised classification approach which classifies new samples based on its K nearest neighbours from the training data – where K is a selectable hyperparameter. The distance to the neighbours can be computed by several metrics, with the most common being euclidean distance [9, 10]. For our KNN classifier, we consider MFCC features as input, of which a total of 40 seems preferred. In order to train the classifier on these extracted features, two approaches were tried. The first is to take the entire, flattened MFC as input for each data point,

[5] Also see: https://en.wikipedia.org/wiki/Speaker_recognition

and the second is to first take the mean over all frames. We found that this second approach worked best.

Found parameters In contrast to the MoG approach, we find that for KNN it is preferred to use principle component analysis for dimensionality reduction. We found that reducing 40 MFCCs back to 15 compontents is optimal. For K we found that 5 is optimal.

4.1.4. Convolutional Neural Network

For our last classification method, we consider the `ResNet18` Convolutional Neural Network (CNN) [11], of which we alter the first convolutional layer and final linear layer to fit our purpose. This model, which was obtained from `PyTorch` its `TorchVision` library, has approximately 11.6M parameters. While this is one of the smallest commonly used CNNs, for our purposes it is fairly large.

For this model, we found that a Mel-spectrogram input is preferred over an MFC. We take as input the spectrogram of a 0.5-second fragment. Since our dataset contains 1-second fragments, we randomly select a sub-window at training time, which then also serves as a manner to counter overfitting. The output is a 3-dimensional vector, of which we take the class corresponding to the highest value as the predicted output. At validation time, we deterministically classify the [0, 0.5] and [0.5, 1] second windows individually, and average the resulting vectors before taking the maximum class.

After finding good hyper-parameters, we achieved a validation accuracy slightly lower than our MoG-based model, which at this point was our best one. Listening to the wrongly classified examples from the 5-fold cross-validation run, suggested that there was still room for improvement, as we found that many of these examples should not have been that hard to classify. Consequently, we came up with another 'dangerous trick' to boost performance.

For us, the amount of training data is not an issue, seeing that we have 190 hours worth. The issue is human resources for labeling this data, as this process consumes much time. Hence, we automatically created a dataset containing 30.000 1-second clips with our MoG-based model, which we then used for pre-training. For this, we first train a model on this big dataset, where we validate the results with our original 1000 clips. After that, we fine-tune this model in the usual 5-fold cross-validation scheme, where we take $train_k \cup extra$ for training, and val_k for validating, as described in Section 4.1.1. The danger of this method is that during pre-training the model will surely see some misclassified examples. Do note, however, that in this scheme the model will never 'see' examples during training that are later used for validating.

Found parameters We found that the pre-training scheme described above, gives a substantial boost to our validation results, making this CNN the best-performing model of all when using the setup described next.

We optimize with respect to cross-entropy-loss between predicted outputs and one-hot encoded ground truths, with a label-smoothing value of 0.1[6]. We use the `Adam` optimizer [12] as implemented in `PyTorch`, with a learning rate of 10^{-4} during both pre-training and fine-tuning. We lower this value by a factor of 10 after 11 epochs without having found a new minimum loss. We keep all other parameters at their defaults, except for the L_2-regularising `weight_decay` parameter, which we set to `0.01` during pre-training and fine-tuning. Finally, we take an early-stopping approach, where we terminate training after 30 epochs without having found a new minimum loss.

Since we use an early-stopping approach, we cannot simply train a final model on the full dataset as we do for all other methods. Instead, with the five models produced during cross-validation, we create a majority-voting ensemble, which serves as our final model.

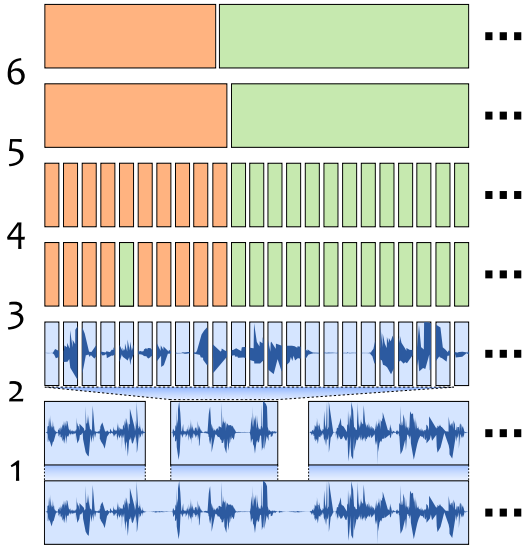

Fig. 2: An illustration of how we go from single-window classification to temporal classification. See Section 4.1.5 for a full explanation of this pipeline.

4.1.5. Segmentation Pipeline

Classifying small windows is one thing; classifying complete podcasts is another. In this subsection, we explain how we use any of the above classifiers to classify an entire episode. We do this in a series of steps, which are also graphically shown in Figure 2. Note that the numbers of the steps correspond to the ones left of the figure.

STEP 1: Given the full episode, we first split it up into non-silent windows. This is because it would not make much sense to classify the silent regions. Instead, as a convention, we label silence as whatever comes before it.

STEP 2: We iterate over all windows and split them up into 0.5-second fragments: a format that all models we implemented can work with.

STEP 3: We let one of our models classify all windows. This gives, for each non-silent window described in step 1, a list of predictions $\mathbf{y} \in \mathbb{L}^N$.

STEP 4: Here we integrate information from neighboring fragments to increase accuracy. For this, we first convert our list of predictions into a one-hot encoded format $\{0,1\}^{3\times N}$. We then convolve each of the three constituent 'channels', with the kernel $(1,4,6,4,1)^T$. This is then converted back to

[6]This prohibits the model from becoming overconfident, which prohibits overfitting

the $\mathbb{L}^N$-format by picking the class corresponding to the highest value at each position. This will ensure that any window labeled c_x that is surrounded by windows labeled c_y, where $c_x \neq c_y$, will be changed to c_y, as we assume such a situation will occur most often due to erroneous classification. Figure 2 shows an example.

We did not choose a simple n-point averager kernel instead, as we felt that fragments that are closer should have more influence than ones that are further away. Also note that all values outside of the range $1 \leq n \leq N$ are assumed to be zero, such that, for example, for $n = 1$ on the MVR-channel, we effectively have $6 \cdot m_1 + 4 \cdot m_2 + 1 \cdot m_3$ as result (where m_n are one-hot values).

STEP 5: We convert our list of class labels into a different format. First, we create a 3-tuple (y, t_s, t_e) for each element in the list, where y is the label (i.e. the element itself), and t_s and t_e are the respective beginning and end of the corresponding 0.5-second fragment. After this, we iteratively merge adjacent tuples if their labels y are the same. We do this by taking t_s from the first, and t_e from the last. Figure 2 illustrates this process more clearly.

STEP 6: Up till this point, classification was done at a temporal resolution of 0.5 seconds. This means that borders, where one person stops talking and another starts, might not be precise enough. Hence, in this step, we refine these.

For each time t_b that can be described as such a found border, we select a 1-second region with t_b right at the center. In this region, we *ipso facto* know that the window [0, 0.5] was classified as some class c_x, and [0.5, 1] as another class c_y, where $c_x \neq c_y$. We add to this, three more classifications, for windows $[i \cdot \frac{1}{4}, i \cdot \frac{1}{4} + 1]$, where $i = 1, 2, 3$. With these three new classifications, plus the two original ones, we count the number n_x of windows that were classified as c_x, and n_y of windows classified as c_y. We then refine the border time t_b with:

$$t_b \leftarrow \frac{b(n_x) + 1 - b(n_y)}{2},$$

where:

$$b(n) := \frac{3}{16} + \frac{1}{8} \cdot n. \tag{2}$$

A derivation of this equation will be given in Appendix A. This will also make clear why the given update rule for t_b is sensible.

STEP 7: One final step, which is not shown in Figure 2, is ignoring everything before the end of the intro-tune, and after the beginning outro-tune. We do this by cross-correlating the intro and outro with the first and last few minutes of every episode and then finding the index of maximum correlation. We only conclude that such an index indeed corresponds to the intro/outro, if the correlation value is above a threshold of 1000, as not all episodes will have an intro/outro.

4.2. Speech Synthesis

In this subsection, we describe how we use our dataset of MVR fragments to create a model that can generate unconditional speech sounds.

4.2.1. Linear Regression

For fun, and as a baseline, we experimented with a simple linear regression-based approach. Here we take a one-second fragment worth of Mel-spectrogram frames as input and attempt to predict the next frame. We train in an iterative manner, by defining this as a neural network with a single linear layer in `PyTorch`, and optimising with respect to the mean squared error. To introduce stochasticity, we always add some random noise to the input – such that this noise is responsible for 1% of variance. For non-linearity, we add squared terms, and finally, take $3 \cdot \tanh(\mathbf{y})$ of the output $\mathbf{y}$, as we noted that otherwise the predictions explode and reach `inf`-values.

As expected, the results were nonsensical. The model immediately finds a stable state from any input of the training set, which consists of a single, constant tone that might have the base frequency of MVR his voice, but does not resemble it in any other manner. Consequently, we will not consider this model further in this report.

4.2.2. Unagan

As a more serious candidate, we consider a type of *generative adversarial network* (GAN, see Section 2.2). Specifically, we use a variant of the `BEGAN`-architecture [5] called `UNAGAN` [13].

Most striking about `BEGAN`, is that the discriminator is defined as an auto-encoder. This is a type of neural network that has to reproduce the input it gets. Key, however, is that it has a bottleneck layer, which forces the model to find a lower-dimensional representation of the input. A common loss function $\mathscr{L}$ for such a model, is the euclidean distance between the input and output:

$$\mathscr{L} := |\mathbf{x} - \mathbf{y}|$$

The question then is: how and why do we use this model as a discriminator? For that, we first give its complete loss function:

$$\mathscr{L}_{discriminator} := \mathscr{L}(D(\mathbf{x})) - k_t \cdot \mathscr{L}(D(G(\mathbf{z}))), \tag{3}$$

where the first term gives the auto-encoding loss with respect to an example from the training data, end the second gives the loss with respect to output from the generator, which gets as input a random noise vector $\mathbf{z}$ (we will discuss k_t momentarily, but for the time being assume it is equal to 1).

If μ_x is the mean of the auto-encoding losses over the training data, and μ_z the mean of the auto-encoding losses over generator outputs, then the loss function in Equation 3 will maximize the distance between these, by making μ_x go to zero, and μ_z to infinity. This distance, $|\mu_x - \mu_z|$, gives a lower bound for the Wasserstein distance, which measures the difference between two distributions [5]. As a consequence, the discriminator will try to make the distributions of the auto-encoding losses for, respectively, the training set examples and generator outputs, be as dissimilar as possible.

The generator, on the other hand, wants to make sure these distributions are similar, as this will likely also make the distribution of its output and that of the training set be similar. Its loss is defined as:

$$\mathscr{L}_{generator} := \mathscr{L}(D(G(\mathbf{z}))).$$

This GAN setup is said to converge much more easily than its predecessors, which used simple classifiers as the discriminator. However, one problem persists: there is no guarantee

that this scheme converges to an equilibrium, where there is a balance between the two terms in Equation 3. To resolve this, the k_t term is introduced. Initially, it is set to zero, such that the discriminator first focuses on its auto-encoding task. During training, it is updated with:

$$k_t \leftarrow k_t + \lambda_k \cdot (\gamma \mathscr{L}(D(\mathbf{x})) - \mathscr{L}(D(G(\mathbf{z})))),$$

where λ_k and γ are hyperparameters.

Finally, to get a global loss that can be used, for example, for early stopping, the authors define:

$$\mathscr{L}_{global} := \mathscr{L}(D(\mathbf{x})) + |\gamma \cdot \mathscr{L}(D(\mathbf{x})) - \mathscr{L}(D(G(\mathbf{z})))|. \quad (4)$$

`UNAGAN` is a variation of `BEGAN` that is specifically designed for unconditional audio generation. It can be used to synthesize anything from piano music to singing voices, and also, of course, to synthesize speech. The most distinguishing architectural change it adds to `BEGAN` is that it uses a lower dimensional noise vector as input for the generator, such the generator its architecture has to be structured in such a way that it upsamples this to reach the desired shape output. For audio synthesis, this is beneficial, as a too-high dimensional noise vector can lead to incoherent, fragmented results.

We run this model on our dataset in the exact configuration described in Liu et al. [13], where we take 10-second windows as input. We do, however, make two modifications that we found beneficial for quality and performance. The first of these regards the used vocoder. Neural vocoders are often trained for specific purposes, such as only for voices, or even for just a single voice. Since Liu et al. considers any application of their model, they use a lightweight vocoder called `MelGan` [14] in their pipeline, which they train on the same dataset as the corresponding `UNAGAN` model. Since we only consider speech synthesis, we can use a higher quality vocoder, namely `Hifi-Gan` [15], which comes as pre-trained for voice synthesis, and does not require further fine-tuning. For this, we change the function that converts the training data into Mel-spectrograms slightly, as one works with base-10 logarithms, and the other with the natural log.

The second improvement regards the creation of a dataset from input `wav`-files. We noticed that the script for this, as an intermediate step, converted all files to a high-loss type of `mp3`-format, while this was not needed. Preliminary experiments suggested that this hurts quality of the model outputs, so we changed this.

Finally, we trained our model for 190 epochs, where each epoch consisted of a random subset of 500 batches, with 5 fragments per batch. For the first 5 epochs we trained on a laptop with an `Nvidia RTX 3060` GPU, after that, we trained till epoch 71 on the free `Google Colab` service, with an `Nvidia Tesla T4` GPU, and for the remainder used the *Peregrine* high-performance computing cluster, belonging to the *University of Groningen* its *Center for Information Technology*, where we had access to a single node with an `Nvidia V100` GPU. In total, training took a bit under 24 hours.

V. Results

In this section, we will present our results for both temporal voice recognition, as well as speech synthesis. Most of our results are best presented in the form of audio and video content. These will be available at `https://indooradventurer.github.io/Unconditional-Speech-Synthesis-as-Sleeping-Aid/`.

5.1. Voice Recognition

Firstly, we will present the results of our different voice recognition approaches. On the aforementioned website, we show a few video clips in which we visualize the output of our classifiers. We do this by presenting a small fragment from the podcast, where we use different colors to indicate model predictions.

Other than that, we present evaluation metrics for all our models in Table 1. We present metrics obtained from doing 10-fold cross-validation (reporting the mean and standard deviation when using the $train_k \cup extra$ scheme discussed in Section 4.1.1), as well as on a separate testing set. For this testing set, we do not consider individual one-second clips, because we found it more appropriate to use a scheme that evaluates the entire pipeline, so also the method discussed in Section 4.1.5. To this end, we randomly sample 32 clips of half a minute each and manually annotate who is speaking at what moment. We do make sure none of these clips are from before the end of the intro, or after the beginning of the outro. We then compare our annotations to the output of the models.

Do note that for `ResNet18` we present the 5-fold validation results, instead of 10-fold ones. We cannot perform 10-fold here, because of the ensemble method discussed in Section 4.1.4.

The results show that `ResNet18` is the best-performing model, both during validation and testing. Furthermore, we see that sensitivity for the MVR class is consistently higher than precision, while we mostly tried to maximize the latter. The reason for this is, again, the unbalance in our data: there simply are not many clips available that the model can mistakenly classify as MVR. At the same time, precision is still high enough to expect a high-quality dataset for speech synthesis. In addition, as mentioned earlier, we will sort MVR fragments by length and then take the top 500. This will likely increase the quality even further, as we find that problems occur most often at times when there is much back-and-forward between MVR and TJ.

It is interesting that our ridge regression model shows a sensitivity of 100% for both validating and testing while scoring relatively low on the other metrics. We believe this shows that this model in particular suffers from data unbalance. The abundance of data points labeled MVR likely steered the weights such that a large portion of the input space would be classified as MVR.

Seeing that `ResNet18` gives the best results, we use this model to create our dataset for speech synthesis.

5.2. Speech Synthesis

The results we obtained for speech synthesis were of surprisingly high quality. We provide multiple fragments on the aforementioned website.

During training, we validated global convergence (Equation 4) on a separate validation set of 200 samples. This value kept decreasing during the 190 epochs we ran. How-

Model	Validation			Testing		
	Accuracy	Precision MVR	Sensitivity MVR	Accuracy	Precision MVR	Sensitivity MVR
ResNet18	99.10%	99.34%	99.89%	98.45%	98.53%	99.75%
Mixture of Gaussians	97.90% ± 1.22	98.89% ± 0.85	99.10% ± 0.98	97.30%	98.12%	98.88%
K Nearest Neighbors	97.70% ± 1.19	97.93% ± 0.91	99.89% ± 0.34	96.49%	97.04%	99.07%
Ridge Regression	95.60% ± 1.74	95.45% ± 1.81	100.00% ± 0.00	95.19%	94.90	100.00%

TABLE 1: EVALUATION METRICS FOR OUR VOICE CLASSIFIERS. WE PRESENT RESULTS FOR BOTH 10-FOLD CROSS-VALIDATION, AS WELL AS ON A SEPARATE TESTING SET.

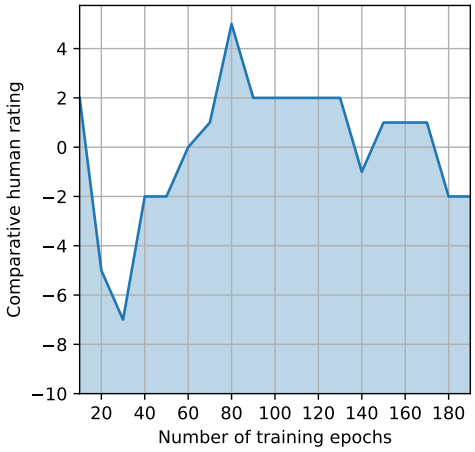

Fig. 3: Unbiased subjective analysis of UNAGAN outputs for different epochs.

ever, we felt that after a certain point, the quality started to get worse. To verify this in an unbiased manner, we created a simple script. In this script, on each trial, we would be presented with the output of two randomly selected versions of our UNAGAN model. We only considered versions that were trained for a number of epochs divisible by 10. Only after judging which output is of higher quality, if any, the script would show us what models were used. Moreover, we would set the random seed to the same value for each comparison, such that the models would receive the same noise vector as input. The result of 80 such comparisons is shown in Figure 3. Here, all models started with a score of 0. After a comparison, the best model would get a score of +1 added, while the worst would get -1.

We see that the best models are indeed found around 110 epochs. Being presented with the selected versions after each comparison suggested to us that especially the ones after 100 and 110 epochs of training gave high-quality results. The figure shows the highest score for the version that was trained for 80 epochs. We believe this is due to chance, as we did not find its results outstanding. On the website, we provide clips for each version, together with the corresponding validation loss.

VI. DISCUSSION

Let us start with the most important question: does this system actually help with sleeping? The answer to this is... inconclusive. I did fall asleep to it twice this week. More often, however, it served as a constant reminder of all the work that still needed to be done for this project, so I quickly turned it off.. I do believe it will work after the deadline has passed since I noticed that something strange happens when listening: because it is a voice, you automatically try to concentrate on what is being said. But this is an impossible task, as there is nothing being said to concentrate on. Doing this for a longer duration results in a bit of an odd state of mind, from which I feel it will be easier to fall asleep. But this is just hand-waving, of course.

We identified one shortcoming of our work. When we were creating the dataset for voice recognition, we only planned on using methods that would work on a per-spectrogram-frame-basis, such as the Mixture of Gaussians approach. With this in mind, we always discarded one-second fragments in which multiple speakers could be heard: we were thinking in a per-fragment way and found that keeping these clips would contaminate one class with some fragments from another. When we started taking the *ResNet18*-approach, however, where we took not single frames, but a 0.5-second fragment as input, it meant that it did not get any examples that could tell it how to deal with these situations, so we should have made a separate dataset for it, perhaps.

For future work, we would like to investigate using text-to-speech models for the same task, but where we use the entire dataset, so also fragments where Tom Jessen is talking. We can then give the output of the temporal classifier as input tokens instead of ones corresponding to spoken sentences, such that we can control who is talking later on. We can then use, for example, a hidden Markov model to generate plausible sequences of input tokens, and compare the results we get in this manner to the ones we would get by training UNAGAN on all data.

To conclude In this report we presented our entire pipeline for processing 'raw data' from '*Maarten van Rossem - De Podcast*', such that we could create from this, a model that produces soothing babbling sounds that might aid in falling asleep. For this, we first had to be able to identify who is talking at any given moment, such that we could extract all fragments where only Maarten van Rossem is talking. We paid extensive detail to this first step, as we felt it was well suited for a semester project in Machine Learning. We found that using ResNet18 for this task gave the best results.

For the second task, of doing the actual synthesis, we were a bit scared at the beginning. During literature search, we found that not many papers mention hardware/time requirements for their methods, and so were not sure if something like this was even possible on the hardware we had access to. Quickly, however, we learned more and got assured that in fact, it was possible. The final results we obtained with UNAGAN really surprised us, as they were much better than we imagined they would be. A nice anecdote to tell about this is the following. A friend asked how it was going with

the project, after which I presented an audio fragment of an hour. While I was playing it, someone else walked in. He said: "Ah, Maarten van Rossem", and started talking about what he thought of him, while sometimes stopping for a few seconds to listen to the fragment. It took him a full minute before he realized it was not really Maarten van Rossem talking. And only after one of us pointed it out to him. I found this a true testament to the power of generative adversarial networks, and to Machine Learning in general.

A. Derivation of Equation 2

In Section 4.1.5 we described an approach for refining the borders (i.e. regions where one person stops taking, and another starts). We did this by looking at a one-second region around the border, where our current estimated time t_b of when the transition occurs, sits right in the middle, at 0.5 seconds.

Our implementation is more general, both with respect to the number of extra windows and with respect to the base interval for classification, which we set to 0.5 seconds in practice. This explanation will also stay general: instead of talking about seconds, we will measure time in 'window units'. Our one-second range then covers two window units, with [0, 1] being the window that was originally classified as c_x, and [1, 2] as c_y, where $c_x \neq c_y$.

Moreover, instead of talking about 3 extra windows, we will talk about $n-1$ extra windows, where in practice we thus had $n = 4$.

With this information covered, we can proceed. We have two original windows, and add $n-1$ intermediate ones: $[i \cdot \frac{1}{n}, i \cdot \frac{1}{n} + 1]$, for $i = 1, 2, \ldots, n-1$. This gives us a total of $n+1$ windows. I.e. for $i = 0, 1, \ldots, n$.

Now, let us assume we have a perfect classifier, which always gives as output whoever speaks the longest in a window. So a window in which MVR talks 51% of the time, and TJ 49%, will always give c_m as output. With such a classifier we should expect a sequence of classifications (for $i = 0, 1, \ldots, n$) of the form: $c_x, c_x, \ldots, c_y, c_y$. In other words, something like $\ldots, c_x, c_y, c_x, \ldots$, or even: $c_x, c_z, c_x, \ldots$ (where $c_z \neq c_x$ and $c_z \neq c_y$) should not occur.

With this, the best new approximation for t_b would be the average of the centers of the two adjacent windows that have a different class: $\ldots, c_x, c_y, \ldots$. Let us first calculate this value for the case where only the first original window was classified as c_x, and the rest as c_y. The center of this first window simply is $\frac{1}{2}$. The center of the second window is:

$$\frac{1}{2} \cdot \frac{1}{n} + \frac{1}{2} \cdot \left(\frac{1}{n} + 1 \right) = \frac{2+n}{2 \cdot n},$$

which gives an average of:

$$\frac{1}{2} \cdot \left(\frac{1}{2} + \frac{2+n}{2 \cdot n} \right) = \frac{1+n}{2 \cdot n}. \tag{5}$$

Similarly, for the case where all except the last are classified as c_x, we find the middle of the last window to be at $\frac{3}{2}$, and of the second to last at:

$$\frac{1}{2} \cdot \frac{n-1}{n} + \frac{1}{2} \cdot \left(\frac{n-1}{n} + 1 \right) = \frac{3 \cdot n - 2}{2 \cdot n},$$

which gives as average:

$$\frac{1}{2} \cdot \left(\frac{3}{2} + \frac{3 \cdot n - 2}{2 \cdot n} \right) = \frac{3 \cdot n - 1}{2 \cdot n}. \tag{6}$$

Now comes the crucial part. When we have a single window classified as c_x (namely the original, first one), t_b is given by Equation 5. If the first n windows are classified as c_x, and only the last one as c_y, t_b would be given by Equation 6. Or in other words, a function $b(\cdot)$ that takes the number of windows classified as c_x as input, and returns the appropriate value t_b as output, would find for $b(1)$ and $b(n)$, the values from Equations 5 and 6, respectively.

It is not hard to see that for intermediate inputs, $b(\cdot)$ would linearly interpolate these values. The linear equation this gives is:

$$b_n(i) = \frac{n+1}{2 \cdot n} + \frac{1 - \frac{1}{n}}{n-1} \cdot (i-1).$$

If we take $n = 4$, and divide the result by 2 to go from 'window units' back to seconds (at a window length of 0.5 seconds), we will find that this gives Equation 2.

With perfect classifiers, this function will give identical values to the described procedure of finding the mean of the center of the two adjacent windows with different classes. With imperfect classifiers, this function is more robust than, for example, finding the first such unequal adjacent pair and taking their centers. However, this function works under the assumption that everything will be labeled either as c_x or c_y. Anything labeled as the third class, c_z, will implicitly be treated as c_y (because the number of windows classified c_y is not given as input and is assumed to be the difference between $n+1$ and the number of windows classified as c_x), which is not desired. To solve this, we work both from left to right and from right to left: We evaluate $b(n_x)$ for the number of windows n_x classified as c_x, and $b(n_y)$ for the number n_y of windows classified as c_y. We then take the average of $b(n_x)$ and $1 - b(n_y)$ to remove the effect of any c_z.

References

[1] A. v. d. 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," *arXiv preprint arXiv:1609.03499*, 2016.

[2] J. H. McClellan, R. W. Schafer, and M. A. Yoder, *DSP first*. Pearson, 2017.

[3] S. S. Stevens, J. E. Volkmann, and E. B. Newman, "A scale for the measurement of the psychological magnitude pitch," *Journal of the Acoustical Society of America*, vol. 8, pp. 185–190, 1937.

[4] M. Sahidullah and G. Saha, "Design, analysis and experimental evaluation of block based transformation in mfcc computation for speaker recognition," *Speech Communication*, vol. 54, no. 4, pp. 543–565, 2012.

[5] D. Berthelot, T. Schumm, and L. Metz, "Began: Boundary equilibrium generative adversarial networks," *arXiv preprint arXiv:1703.10717*, 2017.

[6] N. Perraudin, P. Balazs, and P. L. Søndergaard, "A fast griffin-lim algorithm," in *2013 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 2013, pp. 1–4.

[7] H. Jaeger, *Machine Learning (WMAI010-05)*. University of Groningen, 2023.

[8] D. A. Reynolds, "Automatic speaker recognition using gaussian mixture speaker models," *Lincoln Laboratory Journal*, vol. 8, no. 2, 1995.

[9] P.Mahesha and D.S.Vinod, "An approach for classification of dysfluent and fluent speech using k-nn and svm," *IJCSEA:Vol.2, No.6*, December 2012.

[10] S. M.Kalamani, Dr.S.Valarmathy, "Automatic speech recognition using elm and knn classifiers," *IJIRCCE:Vol. 3, Issue 4*, April 2015.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[13] J.-Y. Liu, Y.-H. Chen, Y.-C. Yeh, and Y.-H. Yang, "Unconditional audio generation with generative adversarial networks and cycle regularization," in *Proceedings of Interspeech 2020*, 2020, pp. 1997–2001.

[14] K. Kumar, R. Kumar, T. de Boissiere, L. Gestin, W. Z. Teoh, J. Sotelo, A. de Brebisson, Y. Bengio, and A. Courville, "Melgan: generative adversarial networks for conditional waveform synthesis," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019, pp. 14 910–14 921.

[15] J. Kong, J. Kim, and J. Bae, "Hifi-gan: generative adversarial networks for efficient and high fidelity speech synthesis," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020, pp. 17 022–17 033.