Towards Neural Audio Compression

Dimitrije Antić, Jovan Cicvarić, Zafir Stojanovski

Abstract

Learning useful latent representations without supervision is a key step for efficiently and effectively compressing data. In this report, we present a neural speech codec for compressing the phonetic content captured by the Mel-Frequency Cepstral Coefficients (MFCCs) of the original audio waveform. Our model, which is in large extend based on the Vector Quantised-Variational AutoEncoder (VQ-VAE) architecture, is simple, yet powerful enough to reconstruct both the overall composition, as well as the relevant highlights of these coefficients. Furthermore, we provide an extensive empirical analysis on the compression capabilities exhibited by our model on the CSTR Voice Cloning Tool-Kit (VCTK) dataset.

1 Introduction

In order to efficiently transmit and store speech signals, audio codecs create a minimally redundant representation of the input signal which is then decoded at the receiver as accurately as possible. These audio codecs compress speech signals by eliminating redundant and unnecessary information, with their design often leveraging extensive domain expertise to keep compression rates high, while keeping artifacts at a minimum [1]. VQ-VAE models are auto-encoders where latent vectors are quantized using a learned vector quantization scheme. These discrete representations have been shown to have a good inductive bias for speech, and perform well on unsupervised acoustic unit discovery tasks [2]. Leveraging these models, it is possible to fully learn the codecs, thus allowing for learning based compression. Several such works [3] [5] [10] have already demonstrated very strong results, with the state-of-the-art being [2], which mainly relies on a deep *convolutional* encoder, a *VQ-VAE* bottleneck, and a *WaveNet* decoder.

In this report, we give a short overview of the current state of the literature, as well as provide a simplified framework for neural speech compression. Considering that there is no current official open-sourced implementation of the WaveNet decoder used in [2], coupled with the immense computational requirements for training this model, we have settled on mimicking the idea with a rather simplified setup. Even though we have initially explored all feasible ideas obtained during the literature review, for the purpose of our work we have settled on using a **deconvolutional** decoder, instead of the typically used WaveNet. Naturally, since our work was carried out on our personal computers, we were limited in the quantity of experiments and methods that we could explore or modify. Nevertheless, our work introduces a novelty compared to the existing literature by also providing an elaborate empirical analysis on the compression capabilities of our model.

This report is structured in the following way: in section 2 we provide an overview of the dataset, as well as the data representation used for training and evaluating our model; in section 3 we shortly summarize the essential ingredients of the VQ-VAE model, along with the task-specific components that we incorporate into it; finally, in section 4 we present our findings on the model's performance for the problem of compressing speech data. Upon completion, the code will be available at: https://github.com/antic11d/neural-compression.

2 Data

All of our experiments were carried on the English multi-speaker corpus from the **CSTR Voice Cloning Tool-Kit (VCTK)** [11]. The dataset consists of speech data uttered by 110 English speakers with various accents. Each speaker reads out roughly 400 sentences which are selected from various newspapers, the Rainbow Passage, and an elicitation paragraph used for the Speech Accent Archive. Although this dataset is typically used for applications related to speech recognition and text-to-speech translation, we will nonetheless rely on it for the task of speech compression.

2.1 Preprocessing

Initially, we trim the original audio signals to one second long utterances. Then, we further trim the silence from the start and the end of each audio waveform. Next, we perform **Mel-Frequency Cepstral Coefficient (MFCC)** extraction from the trimmed audio in order to obtain a fairly robust representation for the model inputs [10] [5] [1]. Finally, after performing the train-validation split, we further normalize the inputs.

2.2 Mel-Frequency Cepstral Coefficients (MFCC)

Mel-Frequency Cepstral Coefficients (MFCCs) [9] are coefficients that make up a **Mel-Frequency Cepstrum** (MFC). In particular, they are derived from a type of Cepstral representation of the audio clip. The main difference between the Cepstrum and the Mel-Frequency Cepstrum (MFC) is that in the MFC, the frequency bands are equally spaced on the mel scale. This spacing has been shown to be a better approximation to the human auditory system's response compared to the linearly-spaced frequency bands used in the normal spectrum. Moreover, the mel scale relates the perceived frequency (or pitch) of a pure tone to its actual measured frequency. Since humans are much better at discerning small changes in the pitch at low frequencies than they are at high frequencies, this frequency warping can allow for a better representation of the sound waveform. In practice, MFCCs are commonly used features for the task of speech recognition because they tend to preserve the *phonetic content* of sentences. Typically, they are obtained in the following five-step procedure:

1. Apply the Fourier transform to a specified signal window
2. Map the powers of the spectrum onto the mel scale
3. Take the logs of the powers at each of the mel frequencies
4. Perform the discrete cosine transform onto the list of mel log powers (as if it were a signal)
5. Obtain the resulting mel coefficients as the amplitudes of the resulting spectrum

In order to reduce overfitting, for each MFCC we further concatenate the *delta features*, as well as the *delta-delta features* (i.e. the delta features of the calculated delta features). Most commonly, these two quantities are known as the *differential* and *acceleration* coefficients.

Despite the fact that MFCCs tend to be a very useful representation for many speech-related problems, a downside of using these features for the task of audio compression is that in fact, we lose the essential pitch information. In turn, this prohibits a valid reconstruction of the audio for a given MFCC. Even though there are several existing methods for reconstructing the audio from these coefficients, typically these approaches either require training a separate model for this task [7], or rely on a further manual investigation of the derived features [6]. Nevertheless, given the high robustness in preserving the phonetic content of the original audio, we have decided to continue exploring the compression capabilities of the models that operate on these coefficients.

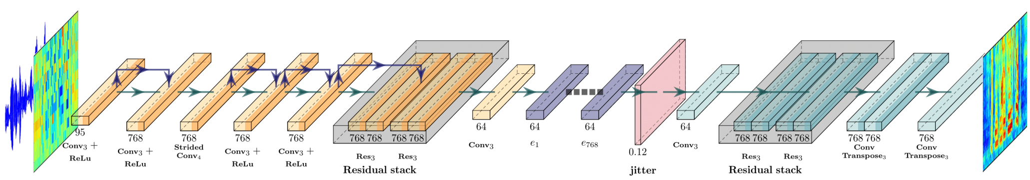

Figure 1: **The model architecture**. © Image credit: Charly Lamothe, github.com/swasun

3 Method

For the purpose of our task, we modify the method proposed in [2] by exchanging the WaveNet decoder with a deconvolutional one. Subsection 3.1 provides a general overview of the VQ-VAE model, whereas subsection 3.2 briefly describes the relevant building blocks of our architecture.

3.1 Vector Quantised-Variational AutoEncoder (VQ-VAE)

In general, Variational Auto-Encoders (VAEs) consist of three main components: an **encoder** network that parametrizes a posterior distribution $q(z|x)$ of the latent variable z given input data x, a **prior** over the latent variable $p(z)$, and a **decoder** that parametrizes a distribution over the input data $p(x|z)$. Typically, the posterior and the prior are assumed be *Gaussian* distributed with a diagonal covariance matrix, thus allowing to utilize the Gaussian reparametrization trick [8].

In contrast to the traditional VAE, the VQ-VAE [10] consists of a prior and a posterior which are **categorical**. In this case, the samples that are being drawn actually index an **embedding table** (also known as a *codebook*; see Appendix, Fig. 3).

Let $e \in \mathbb{R}^{K \times D}$, where K is the size of the discrete latent space, and D is dimensionality of the latent vectors. Then, the resulting codebook can be represented by $e_i, i \in 1, 2, ..., K$. In that case, given an input x, the encoder produces an output $z_e(x)$. Since this is a continuous variable, we **quantize** it by finding its nearest neighbor in the embedding space:

$$q(z = k|x) = \begin{cases} 1 & \text{for } k = \text{argmin }_j ||z_e(x) - e_j||_2 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Therefore, the input to the decoder is:

$$z_q(x) = e_k, \text{ where } k = \underset{j}{\text{argmin}} \; ||z_e(x) - e_j||_2 \tag{2}$$

Since the argmin operator in Eq. 2 is *non-differentiable*, in order to allow for end-to-end learning, we perform an approximation by simply copying the gradients from the decoder input $z_q(x)$ to the encoder output $z_e(x)$. In other words, given that in the forward computation we pass $z_q(x)$ to the decoder, in the backprop step we directly pass the gradient $\nabla_z \mathcal{L}$ unaltered back to the encoder.

The total *loss* $\mathcal{L}$ used to train the VQ-VAE consists of three main components:

$$\mathcal{L} = \log p(x|z_q(x)) + ||sg[z_e(x)] - e||_2^2 + \beta ||z_e(x) - sg[e]||_2^2 \tag{3}$$

where $sg[x]$ denotes the stop-gradient operation. The first term is called the **reconstruction loss**, and is used to optimize the encoder and the decoder. The second term is known as the **codebook alignment loss**, and is used to get the embedding vectors e_i as close as possible to the encoder outputs $z_e(x)$. The last term is the **codebook commitment** loss, which makes sure that the encoder commits to the learned embedding.

3.2 Architecture

As we have mentioned before, the architecture we use consists of a convolutional encoder, a VQ-VAE bottleneck, and a deconvolutional decoder (see Fig. 1). Furthermore, we employ *skip connections* and *residual blocks* [4] in both the encoder and the decoder. These components can allow for a finer gradient flow, which results in better learning. As recommended in [2], we also experiment with using *jitter* layers, which are a dropout inspired time-jitter regularization. As a result, during training, each latent vector can replace either one or both of its neighbors. Similarly to dropout, this prevents the model from relying on consistency across groups of tokens.

4 Results

In this section, we provide an empirical analysis for the compression capabilities of the previously discussed model. We mainly explore how altering the usage of **jitter** and the **number of embeddings** in the codebook can have an effect on the model performance. In the following, we denote D as the number of latent neurons, and K as the number of embedding vectors.

Parameters		Baseline	Entropy	Bitrate	DTW
jitter	**# embeddings**	**(train set)**	**(train set)**	**(test set)**	**(test set)**
yes	32	120.00	102.70	161.53	1089.19
	44	131.02	115.04	242.93	1067.61
	64	144.00	127.82	388.66	1049.52
	128	168.00	112.09	916.89	1043.95
	256	192.00	169.76	2073.81	1041.65
no	32	120.00	113.89	161.20	886.89
	44	131.02	123.14	241.89	875.96
	64	144.00	132.69	385.88	855.83
	128	168.00	156.15	901.88	804.61
	256	192.00	171.61	2344.65	782.71

Table 1: **Evaluation** of the model for several hyperparameter choices on various metrics

After training each model for roughly 40 epochs, we select a checkpoint corresponding to the lowest loss. Then, using a 1000 draws from the training set, we compute the sum of the **entropies** for each latent dimension ($-\sum_{d=1}^{D}\sum_{k=1}^{K} p_{dk} \cdot \log_2 p_{dk}$) and compare it to the **baseline** measure ($D \cdot \log_2 K$) for sanity check.

We can easily notice from Table 1 that in each case, the calculated entropy is in fact smaller than the baseline measure, due to the non-uniformity of the embedding distribution for each latent dimension (see Appendix, Fig. 4). Moreover, using a 1000 draws from the test set, we compute the **bitrate** ($-\sum_{k=1}^{K} \log_2 p_k$) in order to evaluate how well the model performs on previously unseen data.

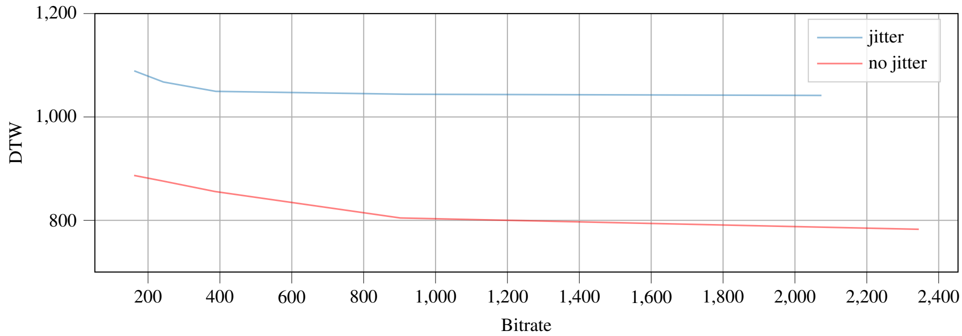

Figure 2: **Rate-distortion curve** for various sizes of the embedding table (see Table 1)

Last but not least, using the Dynamic Time Warping metric, we calculate the **rate-distortion** curve for various model parameters (see Fig. 2). In time series analysis, **Dynamic Time Warping (DTW)** is an algorithm for measuring the similarity between two temporal sequences, which may vary in speed (see Appendix, Algorithm 1). This is a commonly used metric in the domain of speech recognition, since it can cope with different speaking paces.

For the purpose of our application, we use DTW to measure the distances between the original and reconstructed MFCCs (see Appendix, Fig. 5). From the above plot, one can easily notice that using jitter as a regularizer hurts model performance, since altering the embeddings corresponds in less accurate reconstructions. Moreover, as we increase the model capacity by expanding the size of the codebook, we notice an improvement in the reconstruction performance.

On a final note, despite the fact that we were limited in our experiments due to computational requirements, we strongly believe that our findings provide valuable insights in utilizing various architecture choices for the purpose of neural audio compression.

References

[1] J. Casebeer, V. Vale, U. Isik, J.-M. Valin, R. Giri, and A. Krishnaswamy. Enhancing into the codec: Noise robust speech coding with vector-quantized autoencoders. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 711–715, 2021.

[2] J. Chorowski, R. J. Weiss, S. Bengio, and A. van den Oord. Unsupervised speech representation learning using wavenet autoencoders. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27(12):2041–2053, Dec 2019.

[3] C. Gârbacea, A. van den Oord, Y. Li, F. S. C. Lim, A. Luebs, O. Vinyals, and T. C. Walters. Low bit-rate speech coding with VQ-VAE and a wavenet decoder. *CoRR*, abs/1910.06464, 2019.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[5] F. S. C. Lim, W. Bastiaan Kleijn, M. Chinen, and J. Skoglund. Robust low rate speech coding based on cloned networks and wavenet. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6769–6773, 2020.

[6] B. Milner and X. Shao. Clean speech reconstruction from mfcc vectors and fundamental frequency using an integrated front-end. *Speech Communication*, 48:697–715, 06 2006.

[7] W. Ping, K. Peng, and J. Chen. Clarinet: Parallel wave generation in end-to-end text-to-speech. *CoRR*, abs/1807.07281, 2018.

[8] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models, 2014.

[9] M. Sahidullah and G. Saha. Design, analysis and experimental evaluation of block based transformation in mfcc computation for speaker recognition. *Speech Communication*, 54(4):543–565, 2012.

[10] A. van den Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. *CoRR*, abs/1711.00937, 2017.

[11] C. Veaux, J. Yamagishi, and K. MacDonald. Cstr vctk corpus: English multi-speaker corpus for cstr voice cloning toolkit. 2017.

5 Appendix

5.1 Vector Quantised-Variational AutoEncoder (VQ-VAE)

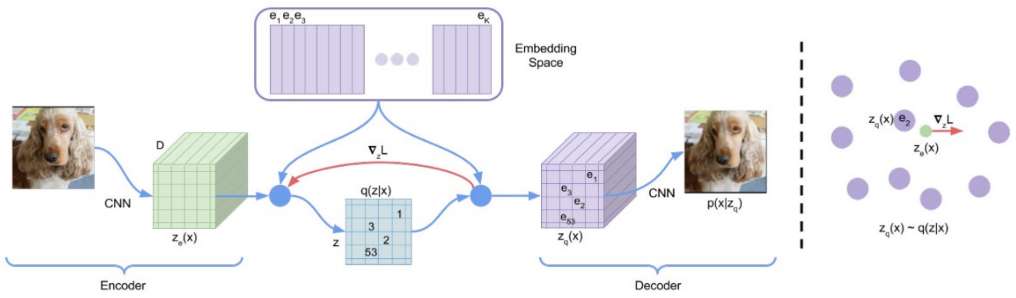

Figure 3: Left: The VQ-VAE model architecture. Right: Visualization of the embedding space. © Image credit: van den Oord et al., 2017

5.2 Embedding distribution

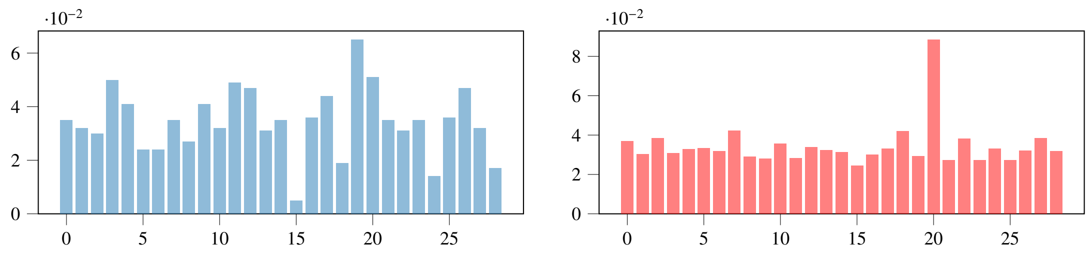

Figure 4: **Embedding distribution** on the train set (left) vs. the test set (right)

5.3 Dynamic Time Warping

This example illustrates the implementation of the dynamic time warping algorithm when the two sequences s and t are strings of discrete symbols. For two symbols x and y, $d(x, y)$ is a distance between the symbols, e.g. $d(x, y) = |x - y|$.

Algorithm 1 Dynamic Time Warping (DTW)

procedure DTW_DISTANCE(s: array $[1 \ldots n]$, t: array $[1 \ldots m]$)
    $DTW \leftarrow$ array$[0 \ldots n, 0 \ldots m]$

    for $i = 0, \ldots, n$ **do**
        for $j = 0, \ldots, m$ **do**
            $DTW[i, j] \leftarrow \infty$
    $DTW[0, 0] \leftarrow 0$

    for $i = 0, \ldots, n$ **do**
        for $j = 0, \ldots, m$ **do**
            $cost \leftarrow d(s[i], t[j])$
            $DTW[i, j] \leftarrow cost + \min(DTW[i-1, j], DTW[i, j-1], DTW[i-1, j-1])$

    return $DTW[n, m]$

5.4 Reconstruction and model performance

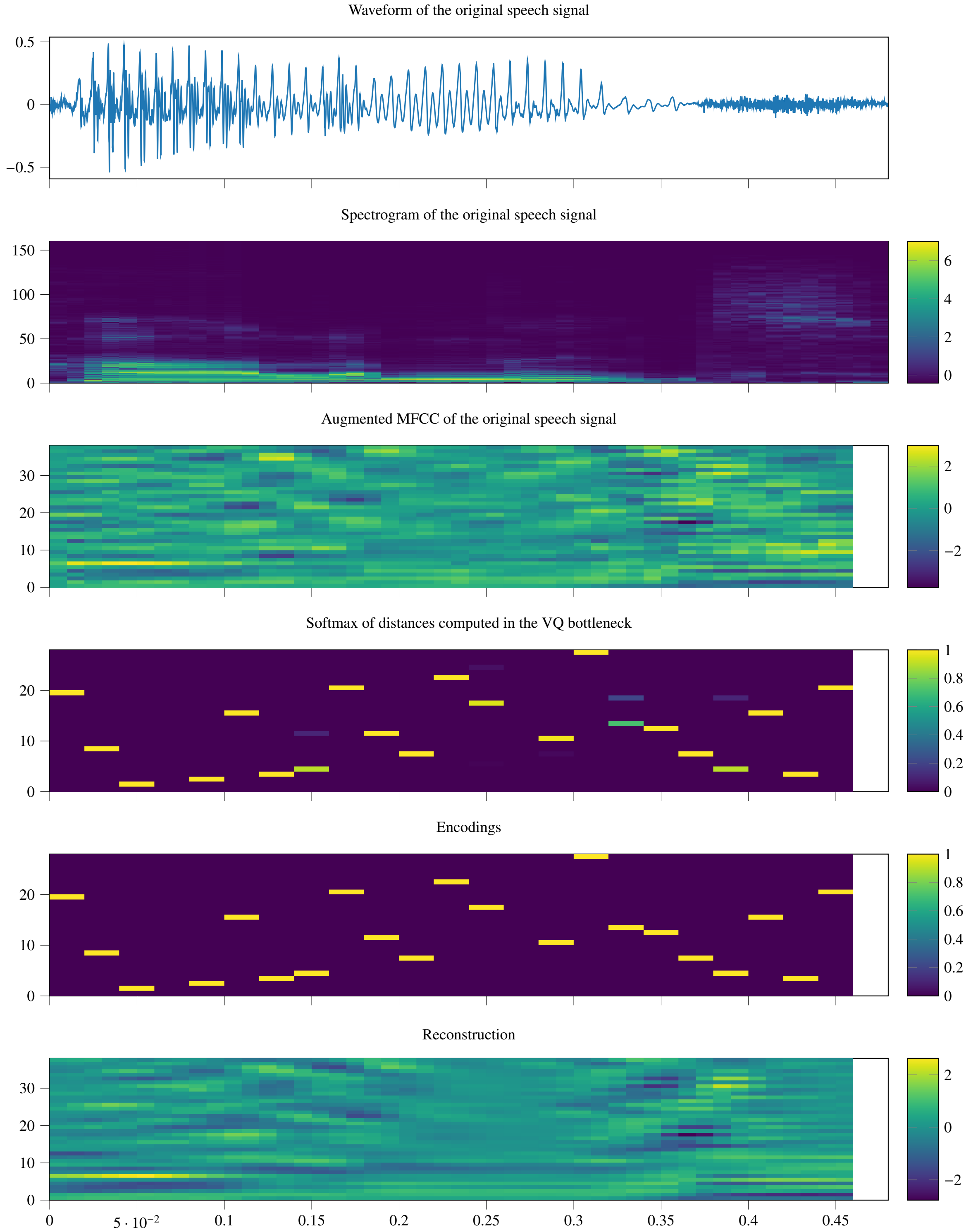

Figure 5: **Evaluation comparison plot**. Notice how the reconstruction captures both the overall structure, as well as the important highlights of the (original) augmented MFCC.