REAL TIME SPEECH ENHANCEMENT

Jonas Myhre Schiøtt (s204218) *Malthe Ørberg Pedersen* (s194584) *Viktor Sebastian Petersen* (s204225)

Technical University of Denmark
DTU Compute

ABSTRACT

In this project we have focused on the denoising aspect of a real time speech enhancement model. We continue from the work done by the creators of AudioDec, who have made a fast neural audio codec, to which we have made several experiments in hope of performance improvement. We make slight altercations to their model, such as removing the middle bottleneck and utilizing a dropout approach to further strengthen the reconstruction of a speech signal to which there have been added noise. We make these experiments with the Voice Cloning Toolkit dataset, which have just above 88000 audio files. Our different experiments proved fruitful, as we generally see an increase in performance for every addition we make. We end up with a model that is very similar in performance to that of AudioDec. Our distortion level is higher, but we have a more potent noise reduction, and the removal of their bottleneck resulted in a good improvement in runtime. All in all, we haven't made great improvements to their model in the scope of denoising, despite the many experiments we have carried out.

1. INTRODUCTION

Speech enhancement is the task of improving the speech quality in a signal that might be noisy or otherwise degraded. It is used in many applications such as speech recognition, hearing aids, teleconferencing and audio forensics [2]. One subtask in speech enhancement is that of noise reduction in which we will focus on the use of audio codecs to try and solve this subtask.

Audio codecs are crucial in music streaming, video conferences, digital audio playback and others. Mainly, there are three components a good audio codec must satisfy: 1) good compression, that is able to construct nice representations of the audio signal, 2) low latency, so that encoding and decoding the signal is fast enough to enable communication and 3) good reconstruction, so that the output signal is as close to the input signal as possible. [3]

In this project, the focus will be on achieving a good noise-suppressed reconstruction of noisy audio with a low latency ideally with streaming capabilities. This means that the denoising of audio signals should be of good quality and the latency should be low enough to achieve a near real-time execution. If this is achieved, the model could be used as a preprocessing unit for a speech recognition (SR) model, removing noise in the incoming speech signal. This could potentially allow the SR model to increase its accuracy in noisy environments We will be using the model proposed by AudioDec [3] as a starting point, from which we will try and enhance the denoising aspect of the model. The code for the repo can be found at the following GitHub repository (`https://github.com/s194584/dl-speech-enhancement/tree/main`), in addition to the accompanied Jupyter file, supplied in the assignment hand in.

2. ARCHITECTURE

In this section, we will quickly review the model architecture used by AudioDec, and mention some of the changes we have made to the model. Finally, we go through the loss functions we have used, and their impact on the overall model.

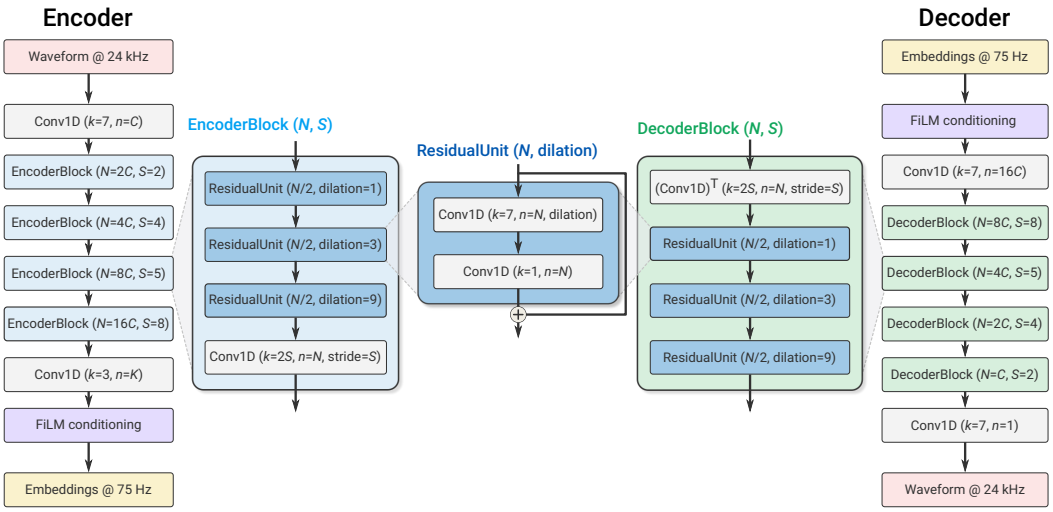

Fig. 1: Encoder and decoder model architecture [1].

2.1. AudioDec

AudioDec came up with a model for audio codecs, that is heavily inspired by the SoundStream model [1]. The SoundStream model is an end-to-end neural audio codec that can compress and decompress audio. It consists of an encoder, projector, quantizer, codebook and decoder in that order, and uses mainly 1-D convolutional layers. The encoder will compress the audio waveform, such that we end with a representation that contains information relevant to prediction. These representations then get projected into a multi-dimensional space and then quantized into codes using residual vector quantization (VQ). This will ensure a constant bit rate in the bottleneck. Finally, the decoder will reconstruct the waveform based on these quantized representations, yielding the reconstruction of the original signal. The model is trained alongside a discriminator, such that we also get access to an adversarial loss.

The largest changes AudioDec made, were to introduce a new training paradigm, using modular architecture and adopt the HiFi-GAN-based generator and multi-period discriminator. The new training paradigm splits the training in two, wherein in the first 200k iterations, the generator is trained, whereas in the next 500k iterations, everything except the decoder is frozen and the discriminator is introduced. This makes the model into a GAN model. This way of training utilizes a modular architecture making it easier to swap out the encoder and/or decoder for specialized tasks, i.e. they use a HiFi-GAN generator in place of the original decoder. This way, they claim to achieve better objective evaluations at the expense of a slower model overall [3].

The HiFi-GAN is a state-of-the-art neural vocoder that uses multi-period discriminators (MPD) to capture the periodic details and multi-scale discriminators (MSD) to capture long-term details. AudioDec altered the HiFi-GAN slightly so that it was more parallel-computational friendly [3].

2.2. Proposed model

As mentioned in the previous section, the goal is for the reconstruction to lack noise and only contain a clear speech signal. For this project, the bit rate limiting bottleneck will be excluded (see Figure 2), as a preliminary exploration showed marginal subjective improvement over the pre-trained AudioDec model for denoising and thus a model that will learn easier was preferred. Furthermore, this change could be beneficial in a few scenarios, where the model should run as fast as possible, i.e. on-device speech recognition. However, keeping the bottleneck would make the model more generally applicable, for example when transferring the data, due to the compression of the storage size.

Like in SoundStream and AudioDec, the model will include a discriminator to generate an adversarial loss for the generator (the auto-encoder). The discriminator is trained to classify original vs decoded audio. Thus the discriminator "leads" the generator to generate audio signals that are indistinguishable from the real audio signal. This is used to promote perceptual quality that cannot necessarily be caught by simple metric losses [3]. This is because it is very difficult to quantize speech audio for a non-human model. To use features like this, the processed audio is compared to real human voices, and thus can be sculpted to sound as close as possible.

2.3. Loss-functions

Throughout the project, five loss functions have been used: (1): L1-loss, (3): feature loss, (2): mel-spectrogram loss, (4): discriminator loss and (5): adversarial loss. Functions 2 through 5 have been used by AudioDec and somewhat by SoundStream. Loss 2, 4 and 5 correspond to the AudioDec equations (5-7) respectively, except the stop gradient operator is not used, as both the encoder and decoder are being trained. The feature loss is also used by AudioDec and is defined in the paper for MelGAN [4]. The five loss functions are defined as follows:

$$\mathcal{L}_{L1} = \mathbb{E}\Big[\, ||x - \mathcal{G}(x)||_1 \,\Big], \tag{1}$$

$$\mathcal{L}_{mel} = \mathbb{E}\Big[\, ||mel(x) - mel(\mathcal{G}(x))||_1 \,\Big], \tag{2}$$

$$\mathcal{L}_{fm} = \mathbb{E}_x\Big[\sum_{i=1}^{T} \frac{1}{N_i} \, ||D_k^{(i)}(x) - D_k^{(i)}(\mathcal{G}(x))||_1\Big], \tag{3}$$

$$\mathcal{L}_{D} = \mathbb{E}_x\Big[\, (1 - D(x))^2 + D(\mathcal{G}(x))^2 \,\Big], \tag{4}$$

$$\mathcal{L}_{adv} = \mathbb{E}_x\Big[\, (1 - D(\mathcal{G}(x)))^2 \,\Big], \tag{5}$$

where x denotes the true signal, $\mathcal{G}(x)$ is the output of the model, $D(x)$ is the output of the discriminator, $D_k^{(i)}$ is the i^{th} layer fea-

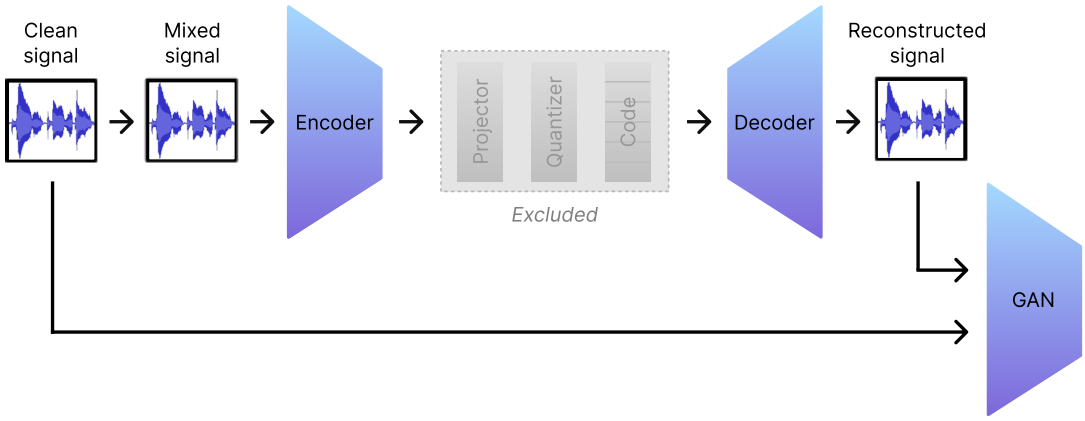

Fig. 2: **Model Architecture advanced from the *Soundstream* model architecture.** The middle Bottle neck featured in the *Soundstream* model, was used for compression, and has been excluded in this model.

ture map of the k^{th} discriminator block, N_i denotes the number of units in each layer and $mel()$ is a function that computes the mel-spectrogram.

The first two losses are ordinary metric losses, which is used to measure the difference of the input and output in some way. The L1-loss simply computes the absolute difference between the two waveforms and takes the mean (MAE). Thus the more equal they are, the less the loss will be. The mel-loss does the same, except it first transforms the waveforms to mel-spectrograms. A mel-spectrogram is like an ordinary spectrogram except it measures the frequency in a logarithmic mel-scale instead of hertz, and it uses decibels instead of amplitude.

The feature matching loss takes an L1 distance between the discriminator feature maps of the real and generated waveforms. It can be seen as a learned similarity metric, as the discriminators learn a feature space that discriminates the real waveform from the generated one [4].

The final two losses regard the adversarial part of the network. The discriminator outputs a value of 1 if it thinks the input is real, and 0 if it thinks the input is fake. Therefore, the discriminator loss is optimized when it correctly guesses the real input as real (ie. the first term in (4)) and the "fake" input as fake (the second term in (4)), meanwhile the generator is optimized when the discriminator guesses the fake as real.

3. EXPERIMENTS

We will first introduce the data we have used for this project, and describe how it was used for training. Afterwards, we will go through the execution of experiments that were conducted to improve the model.

3.1. The dataset and dataloaders

The main part of the project has been using the collection of data from DNS Challenge 4 [5]. This had been downloaded by our supervisor in the past and made available to us in the DTU HPC.

For this project, we have used the Voice Cloning Toolkit (VCTK) dataset, with a main focus on the clean speech samples from `vctk_wav48_silence_trimmed`. This dataset consists of around 44000 utterances recorded in stereo with two different microphones yielding just above 88000 audio files. The utterances are made by 110 different speakers on the same work. The difference between the DNS Challenge 4 version of VCTK and the original is that the silence is trimmed. The noisy dataset consists of 63810 samples. From the two sets of data, mixed signals are produced on the fly and used directly for training the model to denoise. The mixing happens with a random Signal-to-noise ratio (SNR) between 10 and 20 dB. In this way, we do not need additional storage for the mixed audio and can maintain a procedure, that does not require extensive memory.

The clean and noisy datasets will be split into 70/15/15% training, validation and test sets respectively. Furthermore, the PyTorch environment has its random generators seeded manually, such that it is possible to ensure the same splits if the training job should be aborted such that the testing script will use the exact splits to avoid information leakage that could potentially skew the results in favour of the model.

3.2. Training experiments

Early on in the project, we decided that we wanted to create our own training loop, instead of using AudioDec's. That gave us more insight and control as to how we trained the model, and made it easier for us to conduct our experiments. To try and enhance our model, we have experimented with different sample rates, dropout strategies and using the adversarial framework. Below is the timeline of how we added these experiments.

AudioDec (pre-trained): To get up and running with the project, we simply loaded AudioDec's pre-trained models. This would serve as our baseline model that we would want to enhance.

Remove bottleneck: To try and enhance the results, we removed the bottleneck in the model. This would make the model simpler, and we hoped that would lead to better results, as it would be easier for the model to send information through the pipeline. Also, it would make the model faster, which is very important when we want to make it a real-time application.

Metric losses: We then tried to implement our own metric losses. We started by using an L1-loss and mel-loss, as they also used in AudioDec.

Adversary: The next experiment was to add an adversary to the model, in hopes it would help with the high frequencies, as described in [3].

Decrease sample rate: As the training began to take a long time, we wanted to combat this by decreasing the sample rate. This would lead to smaller files, which then would be faster to process. Additionally, the goal is to keep the information on the human sounds. The necessary frequencies range around 2 kHz to 4 kHz and above [6], therefore to keep this information a sample rate of at least double is necessary according to the Nyquist frequency [7]. This led us to settle on a sampling rate of 24kHz.

AudioDec's Mel-loss: As our mel loss resulted in clear horizontal lines when looking at the mel-spectrogram (shown in Figure A.1 in the Appendix), we decided to use AudioDec's mel-loss function instead (see Figure A.2). Later, we discovered that using mel spectrogram loss requires carefully chosen settings to not introduce checkerboard artefacts [4], which was likely the reason AudioDec's implementation worked better.

Gradient clipping: Since the learning curves did seem to converge across epochs, it was still a bit unstable for each batch. To keep this under control, gradient clipping was enabled with low values ranging from 0.1 to 2. The hope was that it would help the model to settle at a lower point than the typical curve and more quickly.

Noise Dropout: When listening to the later models, it was apparent that it was good at filtering away the noise, but unfortunately, it also filtered the speech away. We interpreted this as a case of overfitting on the noise. Thus we implemented a dropout approach, where the noise sample would be overwritten with the clean sample with some probability. Another approach could have been to find a loss function that penalizes this behaviour of chipping away at the clean speech. This could

maybe have been metrics such as SNR and SI-SDR [8] as they take the power of the signals into account, which might expose that clean speech is also silenced. However, due to time constraints, this was not fully explored.

4. RESULTS

We will now look at the results from our experiments. First we will take a look at how the training went, by looking at the training curves. Then we will discuss the evaluation metrics we have used to compare the models. Then we will go over the objective results, subjective results and how the our proposed model and AudioDec's model have performed in a streaming setting.

4.1. Training results

From Figure 3 we see that the learning curves are decreasing and converging, indicating the model is learning something. It has taken about 9 hours to get 20 epochs of training.

Unfortunately, we see in Figure 4 that the model collapses when we use the adversarial framework. This is a problem known to occur in GAN models, but we haven't had the time to implement possible fixes, such as Wasserstein Loss or unrolled GANS [9]. Although we only show one learning curve, know that it is very representing of all other experiments, as they all decrease and converge (with the exception of the GAN-framework).

4.2. Evaluation Metrics

To evaluate the models we have utilized a Deep Noise Suppression Mean Opinion Score (DNSMOS) using the P.835 subjective test methodology [10]. The DNSMOS score is produced by a neural network, that has been trained to predict the Mean Opinion Score (MOS) on 3 categories, Signal quality (SIG), Background quality (BAK) and Overall quality (OVRL), based on data sourced from subjective human ratings. As it predicts MOS, a higher score correlates to better results. Thus a signal score of 5 means the signal is "not distorted", a background score of 5 means the background is "not noticeable" and an overall score of 5 means that the signal overall was "excellent" [11].

To then do the objective evaluation of the different models, they were run to infer on the test set storing their output files, which could then be passed into the DNSMOS network through the accompanying script for local execution. Afterwards, the measures were averaged to attain the results shown in table 1.

	SIG	BAK	OVRL	P808_MOS
L1	2.26	3.76	1.90	2.36
Mel	2.53	3.46	2.13	2.54
Mel GradClip	2.55	3.73	2.19	2.50
24 kHz Mel_Adv	3.10	3.89	2.74	2.76
24 kHz NDR (Our model)	2.95	3.80	2.58	2.70
AudioDec	3.23	3.86	2.87	3.10

Table 1: DNSMOS metrics from the different models. From left to right we see the small improvements we have made to the model.

For a more subjective measure, we have listened to the output of the different models, and given our opinion of how well the models perform. We will also evaluate how the model performs in real time (ie. when streaming) and see if there are any differences here.

4.3. Objective evaluations

In Table 1 we have gathered the results of our experiments. From left to right we have the different stages of our experiments. We see that there is generally an improvement across all metrics as we go to the right. This indicates that our experiments have been somewhat fruitful, in that we have kept enhancing the performance of the

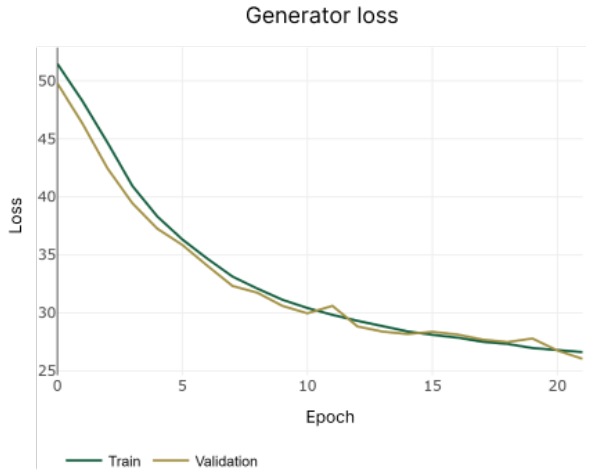

Fig. 3: **Typical Learning Curve.** The above learning curve is typical looking concerning the models in this project.

Fig. 4: **Collapsing Learning Curve** The above learning curve shows the model collapsing, which can occur in GAN models.

model. Although we see an increase, most scores lie between 2 and 3, which translates to "fairly/somewhat distorted speech", "noticeable/somewhat intrusive background noise" and "poor/fair overall speech quality" [11].

4.4. Subjective evaluations

No formal subjective study has been performed, however, 6 random samples (the same as in the Jupyter notebook) have been used to compare AudioDec with our model. Let us start with the characteristics of our model. When you listen to the output audio clips, there is a noticeable "choppy" and "sparkling" and a bit "robot-like" effect on the speech signal, meanwhile, most noise has been filtered away in both the speech sections and quiet sections of the audio clips. Comparing this to AudioDec's model, there are still some "choppy" effects, but the audio sounds more full. This difference is also noticeable in the waveforms from figure 5.

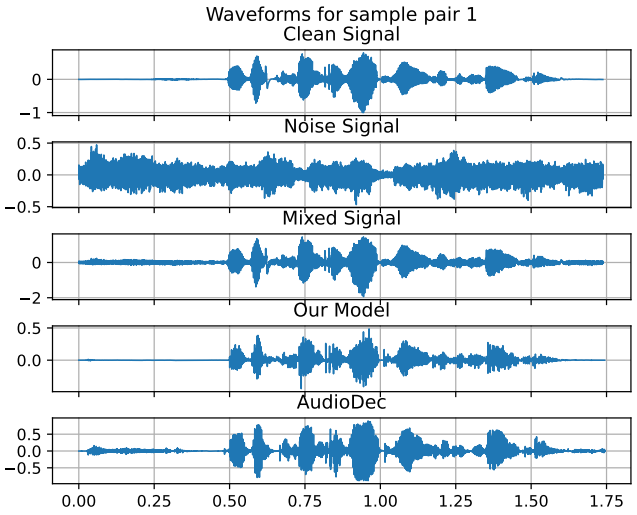

Fig. 5: **Waveform of clean-noise pair.** It is shown, that the reconstruction has removed the noise level from the empty (silent) areas. On the other hand, the waveform of the audio has been suppressed as well.

Our model follows the clean signal pretty well, however, the blobs signifying speech are jagged, resulting in the observed effects. This jagged signal is less noticeable for the waveform for AudioDec. Common for both models is that speech is mostly intelligible for these 6 random samples. In our Jupyter Notebook file, you can hear the performance of our models for yourselves.

4.5. Streaming evaluation

The performance of our proposed model in a streaming framework was measured using a modified version of the `demoStream.py` script from AudioDec [3].

With this script, it is possible to have the model work on an input buffer filled with audio signals coming from a specified input source (i.e. a microphone) and output its reconstructions into an output buffer that will be played in a specified output source (i.e. a speaker). This has been tested briefly with a microphone and speaker in different rooms while a small part of an English story. The test showed that both models can produce a just-below-intelligible output. Our model produces a very monotone voice and is slightly jagged. The AudioDec model contains more features from the human voice, however, the volume seems to oscillate widely, resulting in a very "choppy" sound. We suspect that the human voice features are present both because of the stronger HiFi-GAN vocoder and working at a sample rate of 48kHz allowing information of the higher frequency characteristics. However, it could maybe be possible to achieve something alike by enabling the discriminator in our model in training if more time was available.

Finally taking a look at the run time estimations for a sub-goal of the project, our model is faster. This is also what is to be expected as the removal of the bit rate limiting bottleneck yields a shorter model. The estimations can be seen in table 2.

	Encoding	Decoding
AudioDec v1 (Denoise)	30.55 ± 3.97	43.16 ± 4.62
Our model	21.57 ± 2.35	24.83 ± 2.86

Table 2: **Live demo runtime (ms).** Both models have been run on a laptop* 5 times with a window length of 25 ms. The average runtime is shown.

* CPU AMD Ryzen 7 6800HS Creator Edition, 3.20GHz and 4 threads

5. CONCLUSION

AudioDec [3] has come with a great model that is able to reconstruct sounds in 48 kHz with very high speed, which makes it ideal as an audio codec. When trying to use their model for speech enhancement, the results are fine, but improving on these results should be possible. We have tried to relax their model, by removing the projector, quantizer and codebook, which we hoped would give us better results. Unfortunately, we haven't been able to improve much on AudioDecs denoising model. Our audio signal have a greater distortion on the speaker than AudioDec's model has, but we are able to reduce the noise better than them.

Even with the average results, the model is very fast, and thus we are able to use the model in real-time. This will be very useful if we manage to lower the distortion level in our model without introducing further latency, as we then will have a real-time speech-enhancement model.

6. REFERENCES

[1] Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi, "Soundstream: An end-to-end neural audio codec," 2021.

[2] Linkedin, "What are the current trends and future directions of research and development in speech enhancement?," .

[3] Yi-Chiao Wu, Israel D. Gebru, Dejan Marković, and Alexander Richard, "Audiodec: An open-source streaming high-fidelity neural audio codec," June 2023.

[4] Kundan Kumar, Rithesh Kumar, Thibault de Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre de Brebisson, Yoshua Bengio, and Aaron Courville, "Melgan: Generative adversarial networks for conditional waveform synthesis," 2019.

[5] Harishchandra Dubey, Vishak Gopal, Ross Cutler, Sergiy Matusevych, Sebastian Braun, Emre Sefik Eskimez, Manthan Thakker, Takuya Yoshioka, Hannes Gamper, and Robert Aichner, "Icassp 2022 deep noise suppression challenge," in *ICASSP*, 2022, https://github.com/microsoft/DNS-Challenge/tree/master.

[6] Google Cloud, "Optimize audio files for speech-to-text," `https://cloud.google.com/speech-to-text/docs/optimizing-audio-files-for-speech-to-text`.

[7] Wikipedia contributors, "Nyquist frequency," 2023, [Online; (accessed: 20.12.2023)].

[8] Jonathan Le Roux, Scott Wisdom, Hakan Erdogan, and John R. Hershey, "Sdr - half-baked or well done?," 2018.

[9] Google Machine Learning Education, "Common problems," `https://developers.google.com/machine-learning/gan/problems`.

[10] Chandan KA Reddy, Vishak Gopal, and Ross Cutler, "Dnsmos: A non-intrusive perceptual objective speech quality metric to evaluate noise suppressors," in *ICASSP 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 6493–6497.

[11] ITU-T Recommendation P.835, "Subjective test methodology for evaluating speech communication systems that include noise suppression algorithm," 2003, International Telecommunication Union, Geneva.

A. APPENDIX

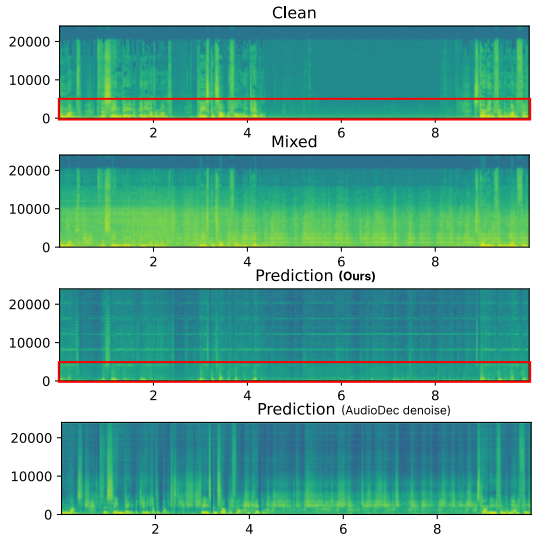

Fig. A.1: Noise interference in Mel Spectogram

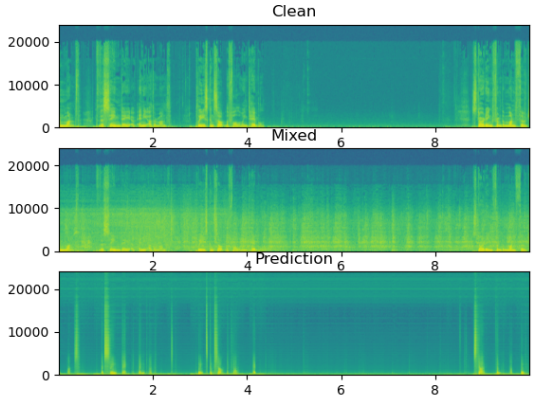

Fig. A.2: Mel Spectogram when using AudioDec's function