Compressão de Áudio com Perdas

JOÃO VITOR ARAKI GONÇALVES *, LUCY MIYUKI MIYAGUSIKU NARITA †, TIAGO LOUREIRO CHAVES ‡

Insituto de Computação, Universidade Estadual de Campinas (UNICAMP)

*Email: *ra176353@students.ic.unicamp.br, †ra182851@students.ic.unicamp.br, ‡ra187690@students.ic.unicamp.br*

***Resumo* – Neste projeto examinou-se a viabilidade do uso de redes neurais Autoencoder como método de comprimir músicas de forma *lossy* (i.e. admitindo perda de informação original).**

O Autoencoder foi treinado com uma coleção de 80 músicas vocais dos gêneros rock e heavy metal, e modelado como uma rede MLP com 5 camadas escondidas, tendo 12348 neurônios nas camadas de entrada e saída, e 4000 na camada central, levando a uma taxa de compressão de 3:1.

Os resultados obtidos mostraram que a rede não poderia ser utilizada como alternativa de compressão pois, além de algoritmos usuais conseguirem taxas muito maiores (p.e. de 7:1 a 10:1 com arquivos .mp3), os áudios reconstruídos pelo Autoencoder apresentam ruído de fundo constante, tornando-os inaceitáveis para uso já que a piora de qualidade é perceptível.

***Palavras-chave* – Autoencoder, *lossy compression*, música**

I. INTRODUÇÃO

Há diversas aplicações de inteligência artificial e aprendizado de máquina no domínio de áudio, como em segurança (p.e. detecção de sons de explosões, tiros de armas, e quebra de vidros [1], [2]), em carros autônomos (p.e. na detecção de eventos sonoros próximos ao veículo), na tradução em tempo real ([3]), ou ainda mesmo na síntese musical ([4], [5]).

Deste modo, decidimos explorar a viabilidade do uso de redes neurais artificiais (ANNs) para a compressão de áudio, mais especificamente, do modelo de rede denominado Autoencoder (AE), devido a resultados promissores apresentados no campo de compressão *lossy* de imagens (p.e. [6], [7]).

Um AE é uma ANN utilizada para o aprendizado de *features* representativas dos dados de entrada (como um sinal de áudio, no caso deste projeto) de maneira não supervisionada. Podemos ver um AE como sendo composto por duas ANNs, um *encoder* e um *decoder* (conforme mostra a Figura 1). As camadas de entrada e saída tem a mesma dimensão, e a camada central é onde temos a representação "codificada" do dado, o *code*, sendo assim a menor camada. Logo, a taxa de compressão obtida, r, é igual à razão entre a dimensão da camada de entrada/saída, n_x, e a dimensão da camada central, n_z: $r = \frac{\text{tamanho original}}{\text{tamanho comprimido}} = \frac{n_x}{n_z}$.

Alguns resultados obtidos com $r \approx 3$ (33% do tamanho original) e $r \approx 4$ (25% do tamanho original) podem ser vistos em: laurelkeys.github.io/sound-engineering e neste link.

Esse trabalho foi inspirado pelo resultado obtido no artigo "*Using TensorFlow Autoencoders with Music*"[8], baseamos parcialmente a nossa arquitetura de rede e métodos de pré processamento nos utilizados pelo autor.

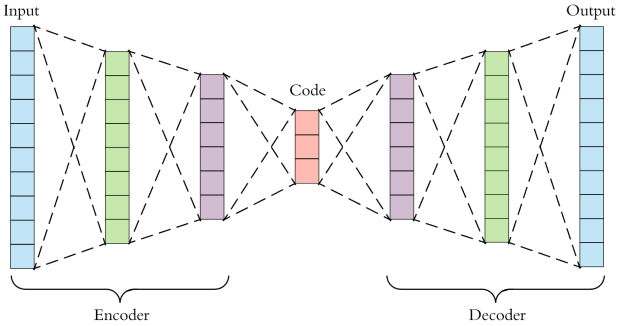

Figura 1. Arquitetura de rede de um Autoencoder

II. PROBLEMA

A. Arquitetura de rede e especificidades

Como queremos avaliar a compressão de áudio, e não a síntese de áudio, optamos por um modelo não generativo, além do fato de que modelos generativos, como GANs e VAEs, notoriamente necessitam de um conjunto muito grande de dados e são mais difíceis e demorados para testar.

Assim, implementamos uma rede neural MLP (i.e. totalmente conectada) com 5 camadas escondidas, sendo duas para o *encoder*, duas para o *decoder*, e uma para *code* (a representação comprimida do áudio efetivamente), com ativação sigmoidal na última camada e ReLU nas outras.

O tamanho das camadas de entrada e saída foi fixado em 12348 neurônios, e os melhores resultados foram obtidos utilizando-se como otimizador uma implementação do algoritmo Adam [9], com taxa de aprendizado de 0.0001, MSE (*Mean Squared Error* [10]) como função de *loss*, e com a seguinte configuração para as camadas escondidas da rede (em ordem): 8400, 5000, 4000, 5000 e 8400 neurônios.

O projeto foi desenvolvido em Python3, com as bibliotecas Keras e TensorFlow para implementar a rede neural, e Pydub e LibROSA para manipular os sinais de áudio. A rede foi treinada utilizando a ferramenta Google Colaboratory (Google Colab), que é um ambiente Jupyter Notebook na nuvem, com placas de vídeo gratuitas.

Para o treinamento da rede foram testados dois datasets distintos, um com 1450 músicas *royalty free* de 2 minutos de duração com gêneros diversos (predominantemente músicas eletrônicas), mas os resultados apresentaram muito ruído, e então foi utilizado outro dataset, com 96 músicas completas (de 3 à 12 minutos de duração) de gêneros semelhantes (rock e heavy metal).

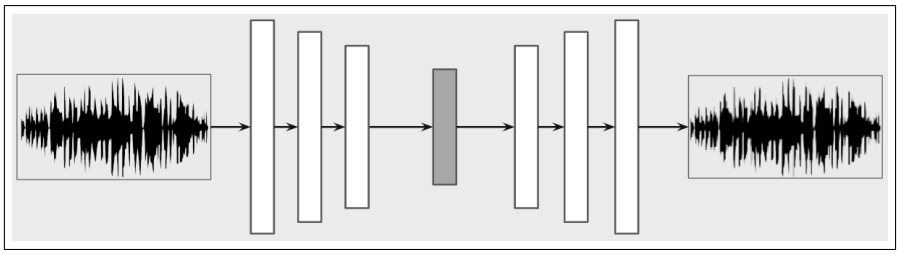

Figura 2. Ao passar pelo Autoencoder, o sinal de áudio é representado com uma menor quantidade de informações (camada cinza), a partir das quais se reconstrói o áudio de saída, de forma a assimilar-se ao áudio original (medindo a diferença pelo erro quadrático médio)

Das 96 músicas, 81 foram utilizadas para treinamento e 15 para validação (escolhidas aleatoriamente). Como as músicas são divididas em segmentos de 0,14s para serem utilizadas como entrada do AE, temos um total de aproximadamente 80 mil *samples* para treinamento. Um conjunto separado, de 10 músicas com estilos diversos (instrumental, pop, e rock com e sem voz), foi utilizado para teste, onde além de avaliar o MSE entre o sinal reconstruído e o original, escutou-se trechos de todas as músicas para ter-se uma avaliação perceptual da qualidade de diferentes resultados.

O pré processamento realizado nos dados, para serem passados como entrada do AE, foi cortar as músicas em segmentos de 6174 samples (0,14 segundos de duração, para o *sample rate* de 44.1 kHz) e normalizar o conteúdo decodificado dos arquivos .wav no intervalo $[0, 1]$, normalizando uma música inteira por vez. Esse intervalo é utilizado devido à função de ativação sigmoidal, aplicada na camada de saída da rede, que produz valores no intervalo $[0, 1]$. Por fim, os dois canais de áudio estéreo foram unidos, totalizando 12348 samples por dado de teste, que é a dimensão das camadas de entrada e saída do AE.

B. Restrições

Uma vez que Autoencoders podem ser vistos como uma técnica de regressão, onde tentamos reconstruir o input original a partir de uma representação de baixa dimensionalidade (a qual também é aprendida durante o treinamento), há uma forte ligação entre os dados utilizados e a representação aprendida. Ou seja, mesmo que um AE seja um ótimo compressor de música clássica, dificilmente ele terá uma boa performance com músicas de diferentes gêneros, como rock ou pop, pois há uma diferença muito grande nos instrumentos utilizados (ou até mesmo na presença de voz), de modo que as *features* "selecionadas"do gênero clássico podem não ser representativas para outros estilos musicais.

Logo, é extremamente improvável que algoritmos clássicos de compressão com perdas, como MP3 e Ogg Vorbis, sejam substituídos por uma rede Autoencoder genérica. Entretanto, para gêneros particulares, a compressão obtida por um AE modelado e treinado de forma específica pode ser muito superior, já que as formas clássicas são generalistas (i.e. a princípio, não agem de forma diferente dependendo do sinal de áudio que será comprimido).

III. Resultados

A Figura 3 mostra a comparação da forma de onda de um trecho original (em vermelho) de 10s da música Nightmare, da banda Avenged Sevenfold, com o áudio gerado pela rede neural (em azul) após 1750 epochs, com compressão de $r = 4$ vezes:

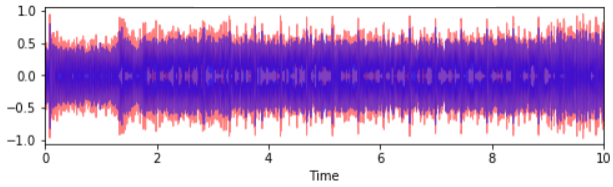

Figura 3. Waveform da música "Nightmare - Avenged Sevenfold"

As Figuras 4 e 5 mostram os espectrogramas em escala Mel da música Equation, de Aphex Twin, que é um exemplo conhecido de esteganografia de áudio [11]. O espectrograma original pode ser visto na Figura 4, e o espectrograma obtido após o sinal passar pelo Autoencoder é dado pela Figura 5:

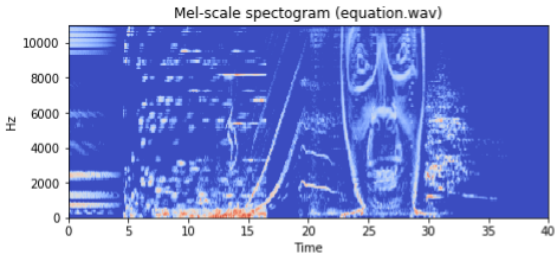

Figura 4. Espectrograma Mel da música "Equation - Aphex Twin" original

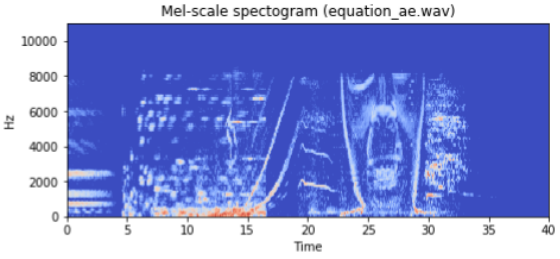

Figura 5. Espectrograma Mel da música "Equation - Aphex Twin" após compressão

		MSE		
		Ours	*DeepMind (v1)*	*DeepMind (v2)*
Sample	*a0.wav*	4.42E+06	8.18E+07	8.25E+07
	a1.wav	5.16E+06	5.93E+07	5.97E+07
	a2.wav	9.20E+05	2.15E+07	2.16E+07
	a3.wav	2.63E+06	2.38E+07	2.40E+07
	a4.wav	3.83E+06	6.26E+07	6.31E+07
	a5.wav	7.95E+05	1.17E+07	1.17E+07
	a6.wav	1.97E+06	2.93E+07	2.93E+07

Tabela I
COMPARAÇÃO ENTRE O ÁUDIO ORIGINAL E RECONSTRUÇÕES COM MSE

A fins comparativos, treinamos também um modelo variacional utilizando a arquitetura VQ-VAE proposta pela *DeepMind* em [12] para a aprendizagem de uma representação discreta dos dados baseada em quantização vetorial (*clustering*), implementada pela biblioteca *Sonnet* ([13]).

O domínio da codificação gerada é dada pelos centróides dos n *clusters* que são encontrados pela quantização vetorial. Para o treinamento desta rede, fizemos um pré processamento dos audios, convertendo-os de *stereo* para *mono* e transformando o sinal para o domínio da frequência utilizando STFT com uma janela de tamanho 6174.

Por conta disso, as camadas de input e output tem tamanho esperado de 3088 *neurons*. A camada de *bottleneck* (o *code*) comprime a entrada para um tamanho 772, isto é, 25% do tamanho original.

Para o *encoder* são utilizadas 2 camadas convolucionais com kernel de tamanho 4 e stride 2; 2 blocos residuais (onde 1 bloco é definido por 4 camadas: ReLU, convolucional kernel tam. 3 stride 1, ReLU, convolucional kernel tam. 1 stride 1) cada um com 12 *hidden units*. Para o decoder são utilizados 2 blocos residuais iguais aos especificados acima e 2 camadas convolucionais transpostas de kernel de tamanho 4 e stride 2. O otimizador utilizado também é o Adam, e a função de erro utilizada pela biblioteca é a soma do MSE normalizado com a função de reconstruction loss apresentada pelo artigo [12].

As versões *v1* e *v2* (veja a Tabela I) dos modelos treinados com a arquitetura VQ-VAE diferem na quantidade de *clusters* utilizados pela quantização vetorial, ou seja, na capacidade de informação armazenada na *layer* de *bottleneck*, sendo a *v1* com 32 clusters e a *v2* com 16 clusters.

Na Tabela I comparamos o áudio reconstruído pela nossa rede com o resultado obtido pela rede VQ-VAE da *DeepMind* [12], utilizando MSE já que esta foi a função de *loss* aplicada no Autoencoder.

IV. ANÁLISE E DISCUSSÃO

Foram testados diferente tipos de pré processamento dos dados, entre eles: padronização com desvio padrão e média, normalização para o intervalo $[0, 1]$, normalização para $[-1, 1]$, transformação para o domínio de frequência com RFFT [14], normalização por música, e normalização por segmento. Porém, a maioria dessas tentativas não produziram bons resultados, a rede não conseguiu aprender *features* relevantes para reconstruir a música, e o áudio resultante era muitas vezes composto apenas de barulhos de estalos, sem lembrar o original.

Experimentou-se um total de 27 configurações diferentes da rede, entre elas, testes com as funções de ativação das camadas ocultas (ReLU, Sigmoid, tanh e ELU), sendo os resultados com Sigmoid e ReLU similares, com a rede com ReLU convergindo mais rápido, mas sem convergência para as outras funções de ativação.

Também foram testados dois otimizadores, o AdaDelta e o Adam, sendo que o Adam na configuração *default* (0.001 *learning rate*) não convergiu, assim foi necessário reduzir a taxa de aprendizado para 0.0001, e o AdaDelta convergiu porém consideravelmente mais devagar. Utilizaram-se duas funções de perda diferentes, a MSE e a *binary cross-entropy*, que não tiveram diferenças perceptíveis em resultado e tempo de execução.

Na Figura 3 vemos que o contorno / envelope da forma de onda gerada é similar ao original, porém há uma diminuição de amplitude (o que é refletido em um som mais baixo).

Analisando os espectrogramas da música Equation (Figura 5), pode-se notar que o output da rede neural não apresenta frequências acima de 8000 Hz. Isso se deve ao fato de que praticamente todas as músicas utilizadas para o treinamento não possuiam frequências dessa grandeza (notas maiores que B_8), fazendo com que a rede aprenda que não é interessante gastar recursos (neurônios e processamento) para tais sinais. Porém, abaixo de 8 kHz vemos claramente a imagem da esteganografia no espectrograma original.

Já pela Tabela I vemos que obtivemos resultados consideravelmente superiores aos da arquitetura VQ-VAE (treinando as duas com o mesmo dataset).

Entretanto, ressaltamos que a principal avaliação da efetividade da rede deve ser feita ouvindo os resultados, e notamos que os áudios de saída do AE lembram claramente os originais, apesar de haver um ruído contínuo de fundo (p.e. original vs gerado). Alguns dos áudios reconstruídos podem ser encontrados neste link.

Um ponto importante a ser mencionado é sobre a função de *loss* utilizada, que foi o MSE. Pela bibliografia psicoacústica sobre percepção e amostragem de sinais sonoros, sabe-se que a percepção humana funciona de maneira bem distinta da forma tradicional de comparação de sinais sonoros computacionalmente como o MSE, ou seja, nossa percepção sonora não é bem modelada tomando apenas o erro quadrático entre sinais de áudio [15], [16].

Assim, é possível que ao compararmos dois sinais reconstruídos com um original utilizando MSE (como na Tabela I), o que apresentar menor erro quadrático médio na verdade seja perceptualmente pior (vide Figura 6).

Por isso, existem métodos que tentam levam em conta fatores psicoacústicos para modelar a percepção humana de sons ao realizar comparações, sendo os mais famosos e utilizados o PESQ [17] e o POLQA [18], que utilizam a escala MOS [19] (onde 5 é o melhor valor possível e 1 é o pior). Entretanto, tais algoritmos não são diferenciáveis, o que impossibilita seu uso como função de perda para ANNs.

Tentou-se comparar os resultados obtidos com o algoritmo PESQ, entretanto, ele só aceita sinais com *sample rate* de 8 kHz e 16 kHz (enquanto utilizamos 44.1 kHz para as músicas), e foi feito para modelar a qualidade de fala (e não de músicas).

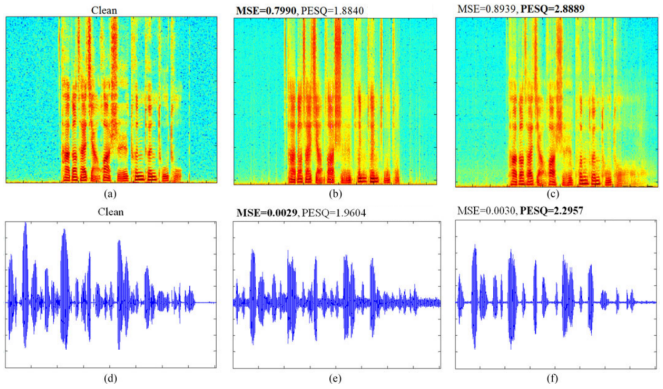

Figura 6. Um menor MSE não garante um melhor desempenho no Mean Opinion Score. A linha superior mostra o caso no domínio da frequência. A linha inferior mostra o caso no domínio do tempo. (Adaptado de [16])

Destacamos ainda, que sem a disponibilidade das placas de vídeo do Google Colab o projeto não teria sido possível, já que as diferentes configurações da rede foram treinadas por mais de 14 dias contínuos ao todo, e esse período seria pelo menos dez vezes maior se treinado num processador ou custaria cerca de US$ 9800,00 numa máquina na nuvem [20].

V. Conclusões

Os resultados não foram bons o suficiente para serem utilizados como um método viável de compressão de áudio, já que é possível facilmente identificar qual é a versão comprimida (devido ao ruído no áudio) e também pelo poder computacional necessário para a compressão e descompressão (a rede toma um total de 2GB de memória RAM, e demora em torno de 1 minuto para comprimir e descomprimir uma música). Além disso, as taxas de compressão obtidas não são tão boas quanto às de outros algoritmos com perda, como MP3 por exemplo (que chega a gerar arquivos com menos de 10% do tamanho original, sem artefatos de som consideráveis).

Porém, ainda assim é possível identificar nitidamente qual música foi comprimida, então mesmo que não seja possível obter um método viável de compressão atualmente, a compressão com ANNs pode se tornar provável com o avanço de hardware e novos algoritmos.

Uma configuração de rede neural que possivelmente obteria melhores resultados seria um LSTM-Autoencoder [21], já que ele levaria em consideração a redundância temporal na música inteira, ao contrário de um *deep* AE com camadas totalmente conectadas, que considera cada um dos segmentos de 0,14s utilizados individualmente.

Outra tentativa interessante para trabalhos futuros seria experimentar Transformers, a nova arquitetura de ANNs apresentada em [22] pelo Google Brain, que mostrou resultados estado-da-arte no processamento de linguagem natural, o que nos leva a crer que seria um bom modelo para o domínio musical (além de ser mais facilmente paralelizável, em relação às redes recorrentes).

Por fim, notamos que outras aplicações na interseção do aprendizado de máquina e da manipulação de áudio já apresentam resultados muito promissores, como na síntese de áudio com a MuseNet [23] da OpenAI e com o MusicVAE [24] do projeto Magenta do Google, ou ainda como a extração de *features* de uma música para serem utilizadas como dado de entrada para outras redes neurais (p.e. de classificação), mostrando a relevância da pesquisa nessa área.

Referências

[1] A. Dufaux, L. Besacier, M. Ansorge, and F. Pellandini, "Automatic sound detection and recognition for noisy environment," 2000. 1

[2] K. Łopatka, J. Kotus, and A. Czyżewski, "Detection, classification and localization of acoustic events in the presence of background noise for acoustic surveillance of hazardous situations," 12 2015. 1

[3] J. Gu, G. Neubig, K. Cho, and V. Li, "Learning to translate in real-time with neural machine translation," 2017. 1

[4] J. Engel, C. Resnick, A. Roberts, S. Dieleman, D. Eck, K. Simonyan, and M. Norouzi, "Neural audio synthesis of musical notes with wavenet autoencoders," 2017. 1

[5] J. Engel, K. Krishna Agrawal, S. Chen, I. Gulrajani, C. Donahue, and A. Roberts, "Gansynth: Adversarial neural audio synthesis," 2019. 1

[6] D. Alexandre, C.-P. Chang, W.-H. Peng, and H.-M. Hang, "An autoencoder-based learned image compressor: Description of challenge proposal by nctu," 2018. 1

[7] L. Theis, W. Shi, A. Cunningham, and F. Huszár, "Lossy image compression with compressive autoencoders," 2017. 1

[8] W. Sherman, "Using tensorflow autoencoders with music," accessado em junho de 2019. [Online]. Available: https://blog.goodaudience.com/using-tensorflow-autoencoders-with-music-f871a76122ba 1

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2015. 1

[10] COUNT BAYESIE (Will Kurt), "A deeper look at mean squared error," accessado em junho de 2019. [Online]. Available: https://www.countbayesie.com/blog/2019/1/30/a-deeper-look-at-mean-squared-error 1

[11] "Steganography — Wikipedia, the free encyclopedia," accessado em junho de 2019. [Online]. Available: https://en.wikipedia.org/wiki/Steganography 2

[12] A. van den Oord, O. Vinyals, and K. Kavukcuoglu, "Neural discrete representation learning," 2017. 3

[13] M. Reynolds, G. Barth-Maron, F. Besse, D. de Las Casas, A. Fidjeland, T. Green, A. Puigdomènech, S. Racanière, J. Rae, and F. Viola, "Open sourcing Sonnet - a new library for constructing neural networks," https://deepmind.com/blog/open-sourcing-sonnet/, 2017. 3

[14] SciPy.org, "NumPy v1.16 Manual — numpy.fft.rfft," accessado em junho de 2019. [Online]. Available: https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.rfft.html 3

[15] S.-W. Fu, C.-F. Liao, and Y. Tsao, "Learning with learned loss function: Speech enhancement with quality-net to improve perceptual evaluation of speech quality," 2019. 3

[16] S.-W. Fu, T.-W. Wang, Y. Tsao, X. lu, and H. Kawai, "End-to-end waveform utterance enhancement for direct evaluation metrics optimization by fully convolutional neural networks," 2018. 3, 4

[17] "Pesq — Wikipedia, the free encyclopedia," accessado em junho de 2019. [Online]. Available: https://en.wikipedia.org/wiki/PESQ 3

[18] "Pesq — Wikipedia, the free encyclopedia," accessado em junho de 2019. [Online]. Available: https://en.wikipedia.org/wiki/POLQA 3

[19] "Mean opinion score — Wikipedia, the free encyclopedia," accessado em junho de 2019. [Online]. Available: https://pt.wikipedia.org/wiki/Mean_Opinion_Score 3

[20] Google Cloud, "Google Compute Engine Pricing," accessado em junho de 2019. [Online]. Available: https://cloud.google.com/compute/pricing 4

[21] H. Sak, A. W. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," 2014. 4

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. 4
[23] C. Payne, "MuseNet," accessado em junho de 2019. [Online]. Available: openai.com/blog/musenet 4
[24] A. Roberts, J. Engel, C. Raffel, C. Hawthorne, and D. Eck, "A hierarchical latent vector model for learning long-term structure in music," 2018. 4

PARTICIPAÇÃO INDIVIDUAL

Abaixo são listadas as principais responsabilidades de cada integrante (sem a pretensão de ser uma listagem exaustiva):

- João Vitor: Processamento dos dados, modelagem inicial da rede, testes de diferentes funções de *loss*, otimizadores, e arquiteturas da rede
- Lucy: Processamento dos dados, avaliação e teste da rede VQ-VAE da *DeepMind*, vídeo do projeto
- Tiago: Processamento dos dados, testes de diferentes arquiteturas da rede, comparação dos resultados obtidos, avaliação de funções perceptuais, apresentação do projeto

O grupo atesta que nenhum membro teve um nível de participação inadequado, e que as atividades desenvolvidas por cada um foram essenciais para a conclusão do projeto como um todo.

Esse trabalho foi uma continuação do projeto desenvolvido na disciplina EE838 (Engenharia de Som), que teve o auxílio de dois outros membros:

- Carlos Avelar: Ajuda com o treino da rede e teste de alguns modelos
- Fábio Ricci: Obtenção do dataset e conversão dos arquivos de .mp3 para .wav

Dado o escopo do problema abordado, os resultados obtidos não seriam possíveis sem o trabalho conjunto e tempo obtido com as duas disciplinas, umas vez que este projeto levou um total de 90 dias para ser executado, sendo mais de 14 dias corridos de treinamento dos diferentes modelos, e resultados razoáveis só foram observados a partir da 24º iteração do modelo testado.