Deep learning project

Daniel Biørrith - 201909298

December 2022

1 Introduction

Jay-Z, a world-renowned rapper, has a voice many are familiar with. He is a highly influential musician, with many fans across the globe, hoping to one day to hear his voice in real life; but what if they could sound like him themselves? This is what we seek out to obtain, via a method called voice conversion (VC).
The motivation behind this project is both to be able to create Jay-Z's voice, but also to challenge ourselves in a fun and technical project. Jay-Z has a distinctive and recognizable voice, and creating a voice converter that can accurately reproduce his voice requires the use of advanced deep learning techniques.

Voice Conversion

Voice conversion (VC) aims to convert a speech audio recording from one speaker to another without changing the linguistic content. In voice conversion, many different tasks exist, such as many-to-many, which aims to be able to convert any voice to any voice, or one-to-one, which aims to convert one specific speaker to another specific speaker [1]. Many methods have been applied and even more are emerging, making it an evergrowing field.
The possible use of voice conversion is huge, as it would make tasks that include voices much easier. For example, instead of having hundreds of people to voice different characters in video games, what if one person could voice them all, using a voice converter? It can also be used to augment existing data, in order to provide larger datasets for other tasks such as text-to-speech or voice recognition.
The ultimate goal of a VC is to create a model that is able to do real-time zero-shot any-to-any voice conversion, where the converted voice is indistinguishable from the real voice. However, as that is a task too hard and time-consuming for us mere students, we instead set the scope of this project to create a one-to-one model, which trains on a dataset of parallel training data.

2 Dataset

The dataset used in this project is one we created ourselves. As Jay-Z released multiple of his albums in acapella versions, hours of voice recordings exist, which we chopped up into smaller audio clips of up to five seconds in length, paired with the corresponding text in a .csv file. A text-to-speech model was deployed to said text, specifically the Sneakers 4 Silero model [2]. From this, we had source and target audio, which we resampled to 16000Hz. The final dataset consists of 1282 data points, which was split up into 80 test data points, and 1202 train data points. The reason for no validation split will be expanded upon later. As the dataset is collected from different albums, the quality of audio and general sound/pronunciation might differ. Furthermore, an attempt to remove background vocals and ad-libs have been made, but to completely remove them was not possible. This could be of concern for the final product.
This concluded the creation of the dataset. The size of the dataset was of concern, as from research we found that we would need a large dataset to train on, in order to perform a one-to-one conversion.

3 Data Preprocessing

Standard practice when working with audio in neural networks is providing Mel Spectrograms as inputs to the models, which we also did[3]. In order to obtain the spectrograms, we had to do some data

preprocessing in our dataset class. First, it loads in the data as waveforms using the Librosa library [4]. Next, the waveforms are padded to the same length, ensuring each input and output is of the same size. The data is then normalized, before it's converted to a Mel Spectrogram, from where it's finally normalized again, to ensure each value is in the range of [-1,1]. After all the processing, we would have a Spectrogram in the shape (1x80x428).
We considered doing data augmentation on the input. Many ways of augmenting data exist, such as pitch and time shifting the waveforms, as well as time and frequency masking [5]. However, as we were developing a one-to-one model, we believed it would be a disadvantage to the model, rather than benefit it. We believe this because the it could change the general characteristics of the voice, which is not ideal. Data augmentation would be more useful when parallel data is not present, as methods such as pitch shifting would be able to create paired data from a single input (if one ensures the pitch shift does not affect the data to the point of losing linguistic information, possible measured in word error rate), meaning a parallel dataset doesn't have to be created. Unfortunately, no data augmentation meant that we couldn't increase the size of the dataset.

4 Network Architecture

When developing our network, we sought out the internet for possible existing models we could seek inspiration from or even build upon with transfer learning. However, we had to do it with our intended model in mind. Normally, a voice converter extracts both the linguistic audio features and speaker information from the audio. In a one-to-one model, the target and source speakers are already implicit in the model itself, making it unnecessary to extract features from the speakers. This meant that we would only have to do the feature extraction.
For this, we found SingleVC to draw inspiration. SingleVC is a many-to-one VC, which is implemented as an autoencoder, composed of residual blocks, attention layers, converter blocks, and linear layers. The pre-trained model, however, took an input whose dimensions didn't fit our data. Instead, we tried to implement our own, simpler version, but with little luck. The model converged to output spectrograms with all entries set to 0, and we couldn't figure out the problem[1].
Instead, we chose to tackle the problem as an image-to-image translation problem, as the Spectrograms essentially are images. For this, we implemented a Generative Adversarial Network (GAN)[2]. Several high-performing methods for VC utilizes a GAN model [3]. Our GAN model consists of two networks, a Generator, and a Discriminator, who compete to maximize the other network's loss. Here, the Generator is the one doing the actual image-to-image conversion, whereas the discriminator is fed two images, one real and one made by the generator, and is to guess which is right.
The generator model we created is depicted in figure 1. It is a U-net model, which has an encoder section, a bottleneck, and a decoder section. The encoder serves as the feature extractor, as it convolutes the image size down while increasing the number of channels. From the second to the 7'th layer, batch normalization is applied, while leaky ReLU is applied to the first seven layers, in an attempt to avoid the dying gradients problem[7]. The decoder consists of 6 inverse convolutions to once again scale up the image, where skip connections are applied as shown in the figure. The decoder has dropout in the first three layers to help prevent overfitting. Finally, to keep values of the spectrogram in the range [-1, 1], the hyperbolic tangent function is applied at the very end of the generator. We've played around with the number of layers in the generator, and this is with what we get the best performance.
The design of the discriminator is simply five layers of convolution with leaky ReLU applied to them, as it's a rather simple classification problem. The implementation has not been changed nor played around with much, thus will not be explained in detail.

5 Training

In order to train the network, we had to choose a loss function. As there were two models in the GAN network, two loss functions were chosen. The first one is for the discriminator; as the discriminator is simply right or wrong when predicting one of two targets, Binary Cross Entropy loss was chosen. In theory, when the model converges, the loss value of the discriminator should converge toward 0.5.

[1]The implementation is in the file singleVC_test.ipynb
[2]The implementation takes heavy inspiration from [6]

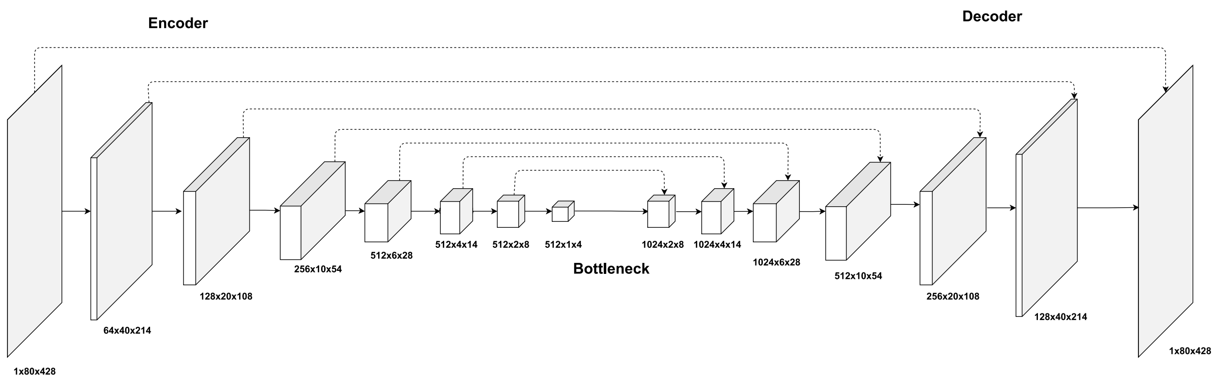


Figure 1: The U-net architecture of the Generator

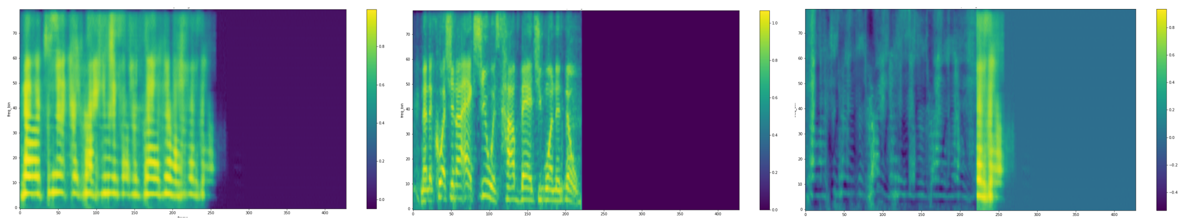
The generator loss consists of two parts. First, L1 loss is applied to find the deviation between the generated picture and the target picture. In addition to that, the generator also depends on the discriminator's loss of the fake images. The loss is calculated as follows:

$$L_{gen} = L_{disc} + \lambda L1$$

Here, λ is a regularization term[8], where a higher value gives the possibility of underfitting, possibly making the model too simple, whereas a lower value increases the possibility of overfitting, but also increases the complexity. After testing values 100, 75, and 50, we set the value equal to 50. When choosing an optimizer we tested two different ones, SGD and adam, and found better results using adam. That is what we used for the final model.
The best learning rate we found were either 0,0001 or 0,0002, with similar results for both. If we went with a higher value, the model would not obtain very good results. Lower values would mean an insignificant change in loss and too long training times.
When training the model, we used small batch sizes, typically 4 or 8, without a significant, measurable difference between the two. Increasing the batch size would only make performance worse, and decreasing would make the model take too long to train.
Important to notice is that we don't test for overfitting. As will be shown later, the results of the model were not satisfying. Thus, instead of implementing an overfitting test, when the performance even on the training data wasn't sufficient, we deemed overfitting the least of our problems. Instead, we printed a target and corresponding generated spectrogram every 10 epochs to visually see and test the improvement (or lack thereof) of the model. However, overfitting of the discriminator could and probably is a problem, which we had to tackle.
Through the training we found that the generator never converges to a value, while the discriminator quickly settles around 0 in loss. This might be caused by the discriminator overfitting and the generator therefore gets stuck trying random outputs, unsuccessfully. To circumvent this, we added some small, Gaussian noise at the start of the discriminator.

6 Results & Discussion

As a spectrogram transformation is a lossy compression, we would need a vocoder to convert a spectrogram back to a waveform. A vocoder typically falls out of the voice conversion scope and is a field of study in and of itself [3]. Because of this, we didn't train a vocoder, and can only obtain useful results by comparing the spectrograms. Figure 2 shows plots of the generated image, target image and difference between the two, respectively.
If you just look at the generated image, it seems completely possible that it's a real spectrogram. However, as we're working with voice conversion, it has to be very close, or identical, to the target image in order to sound anything like we want it to. This is not the case for the obtained output. Generally, the generated images have a more saturated look, with a higher mean value of the whole spectrogram. We imagine, without being able to test it, that this would result in a more noisy sound, as well as a higher-pitched voice than Jay-Z, as the frequencies are higher.
The loss of the generator and the discriminator are shown in figure 3. From the figure it's very clear and easy to see that the discriminator never converges to 0.5 as wanted, but instead oscillates around



(a) Generated spectrogram (b) Target spectrogram (c) Difference

Figure 2: Plot of generated, target, and difference between the two.

1. We either had it converge towards 0, which meant it overfit on the training data, or never got good enough to converge to 0. Sadly, we didn't find the magic spot of balance between the generator and the discriminator, though we tried many different combinations of hyperparameters and network architectures.

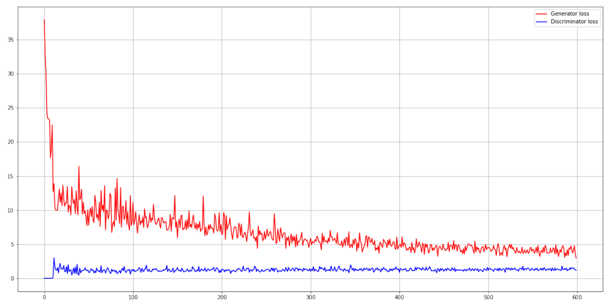


Figure 3: Loss of the discriminator (blue) and the generator (red)

A reason for our model not being of wanted quality might be the dataset. One problem is the size of it, as more datapoints would prevent the discriminator from overfitting. Additionally, because men have lower-frequency voices, they're generally worse for VC, and we even try to convert a woman's voice to a man's voice. Furthermore, as the target data is rapped and not spoken, the pronunciation is not consistent, and the melody makes it harder to predict. Also, occurrence of background/layered vocals might be a contributing factor to the errors of the model.
From a visual testing standpoint, the results obtained with an overfittet discriminator looks more like real spectrograms than with the noised discriminator. However, the generator loss and the L1 loss were lower, even though the spectrogram visually looked worse. In order to test what creates the most accurate audio we would once again need a vocoder, meaning this only leaves speculating.

7 Conclusion & Future Work

Throughout the project, different solutions to voice conversion were explored, and a GAN model was developed for the task. For various reasons, such as a limited and possibly flawed dataset, the model did not obtain the wanted results but did get results that could fool the reader into thinking they were real spectrograms created from recorded audio.
A lot of possible future work exists. More data, or other types, such as normal speech, can be collected to improve upon the dataset. Additionally, a many-to-one model could be developed instead, giving everyone the possibility to convert their voices to Jay-Z. For this to be done, it would be optimal to first train on a large dataset of speakers, and by using data augmentation learn the characteristics of speaking to make a better feature extractor. This, along with speaker embedding[9], could both be an input to a decoder, in order to make a many-to-one or even many-to-many VC model.

Bibliography

[1] “Introduction to deep learning-based voice conversion.” (), [Online]. Available: https://medium.com/@sandipandhar_6564/introduction-to-deep-learning-based-voice-conversion-vc-a-growing-domain-of-speech-synthesis-405ec7fa95b6 (visited on 12/11/2022).

[2] Sneaker4. “Silero models: Pre-trained enterprise-grade stt / tts models and benchmarks.” (), [Online]. Available: https://github.com/snakers4/silero-models (visited on 12/11/2022).

[3] M. Baas. “Comparison and development of practical voice conversion models.” (), [Online]. Available: https://rf5.github.io/assets/docs/skripsie_report_mbaas.pdf (visited on 12/11/2022).

[4] Librosa. “Librosa - audio and music processing in python.” (), [Online]. Available: https://librosa.org/doc/latest/index.html (visited on 12/11/2022).

[5] K. Doshi. “Audio deep learning made simple: Sound classification, step-by-step.” (), [Online]. Available: https://towardsdatascience.com/audio-deep-learning-made-simple-sound-classification-step-by-step-cebc936bbe5 (visited on 12/11/2022).

[6] V. Dey. “Image to image translation in pytorch.” (), [Online]. Available: https://www.codespeedy.com/image-to-image-translation-in-pytorch/ (visited on 12/11/2022).

[7] ML Glosarries. “Activation functions.” (), [Online]. Available: https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html#leakyrelu (visited on 12/11/2022).

[8] Google. “Regularization for simplicity: Lambda.” (), [Online]. Available: https://developers.google.com/machine-learning/crash-course/regularization-for-simplicity/lambda (visited on 12/11/2022).

[9] RF5. “Simple speaker embeddings.” (), [Online]. Available: https://github.com/RF5/simple-speaker-embedding (visited on 12/11/2022).