Report for Music Interpolation

Krishna	1001781
Gao Yunyi	1002871
Pengfei	1002949

Abstract

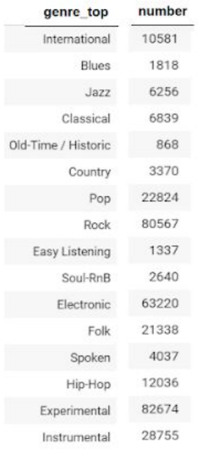
In this report, we aim to interpolate between 2 music excerpts using the extracted feature of mel frequency. Our main tasks are to use autoencoder to get latent vectors of given music excerpts, interpolate between two latent vectors, decode the interpolated latent vector to mel spectrogram and reconstruct to audio. Our experiment is conducted in stages including dataset collection, data preprocessing, experimentation, evaluation and conclusion, which will be presented in the following report.

Dataset and Collection

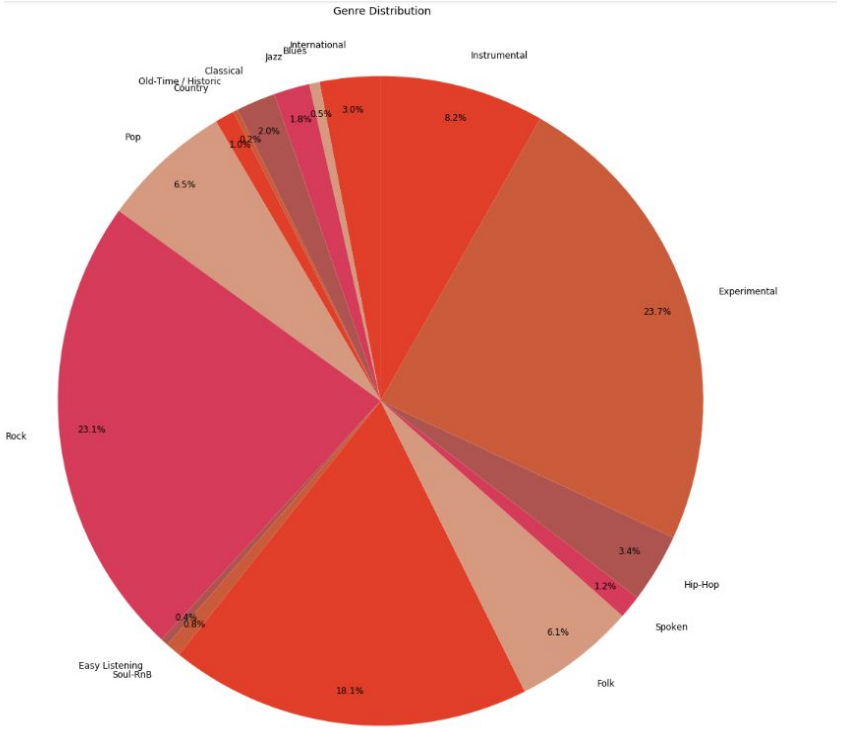
We use the open-source audio dataset called Free Music Archive (FMA). It is an interactive library of high-quality, legal audio downloads directed by WFMU launched in 2009 (https://github.com/mdeff/fma). It contains MP3 files of 77,643 songs, which are 30 seconds each covering across 68 genres and sampled at 44,100Hz, 128kb/s. It contains not only the audio but also precalculated feature of each song as below.

Low-level Musical Features		High-level Musical Features	Other Metadata
Feature	Dimensions	Acousticness Danceability Energy Instrumentalness Liveness Speechiness Tempo Valence	Genre Album Artist
Chroma_Cens Chroma_CQT Chroma_STFT MFCC RMSE Spectral Bandwidth Spectral Centroid Spectral Contrast Spectral Rolloff Tonex ZCR	83 83 83 139 6 6 6 48 6 41 6		

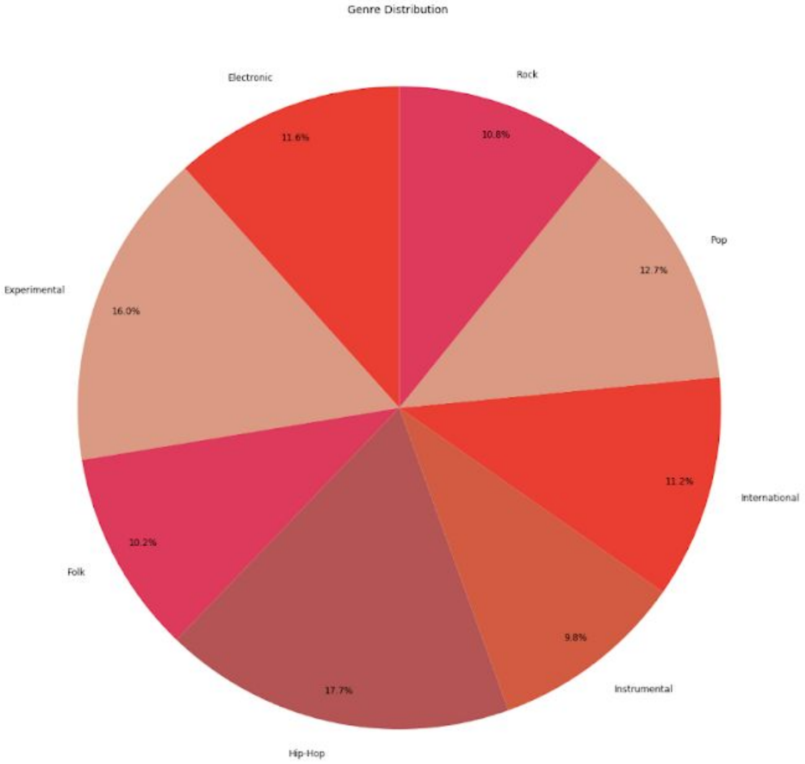
The following is our graphical analysis of the genre distribution over 16 top genres in total. Number is the amount of music tracks under the stated genre.

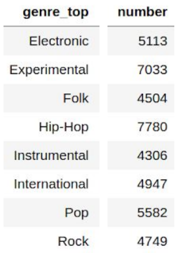


genre_top	number
International	10581
Blues	1818
Jazz	6256
Classical	6839
Old-Time / Historic	868
Country	3370
Pop	22824
Rock	80567
Easy Listening	1337
Soul-RnB	2640
Electronic	63220
Folk	21338
Spoken	4037
Hip-Hop	12036
Experimental	82674
Instrumental	28755



Due to the unbalanced dataset with respect to genre, FMA has already defined a balanced data subset over 8 top genres in its metadata. We have filtered out the balanced subset and below is our graphical visualization of the balanced subset which is used for our future training.



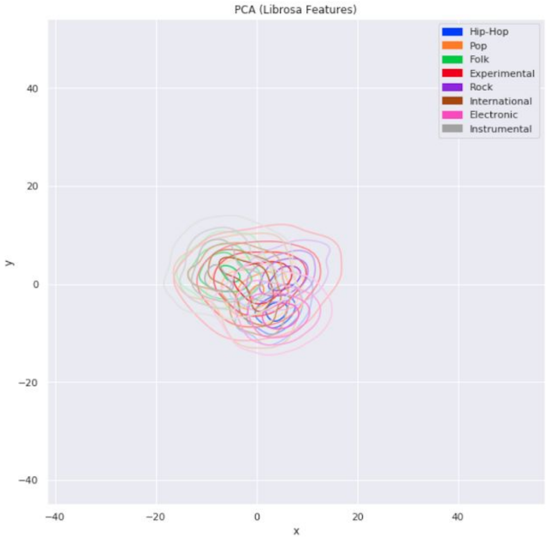


genre_top	number
Electronic	5113
Experimental	7033
Folk	4504
Hip-Hop	7780
Instrumental	4306
International	4947
Pop	5582
Rock	4749

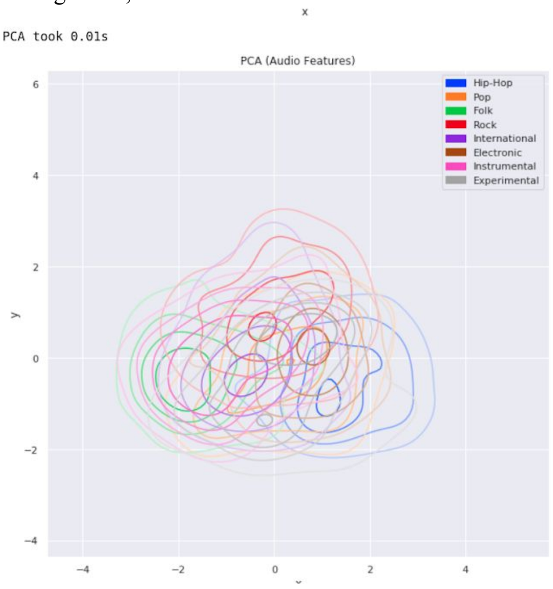
Visualization of low-level musical features

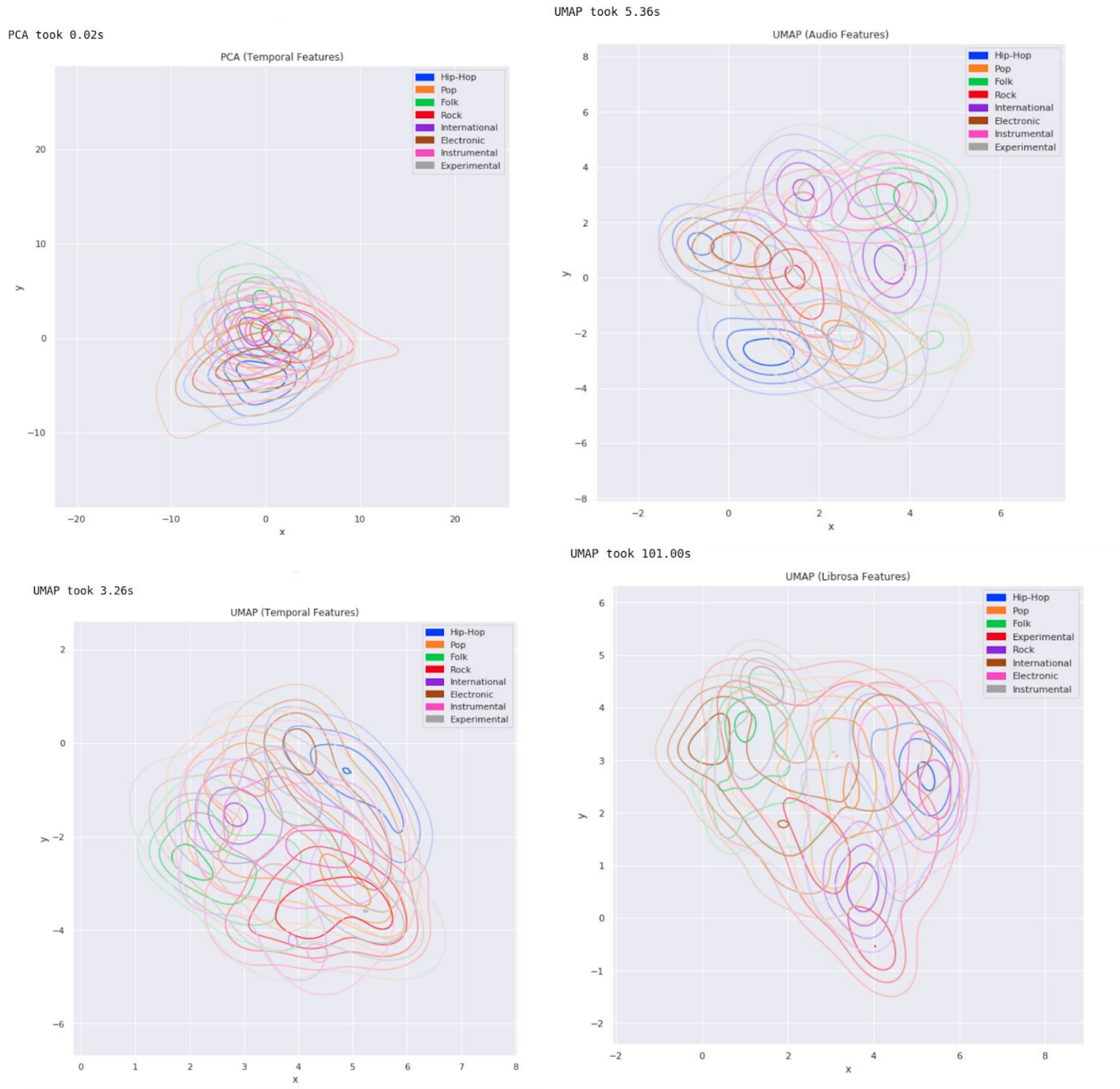
We visualize the balanced dataset distribution over 3 subset of low-level features: librosa feature, temporal feature and audio feature. Three dimensionality reduction methodologies are used including: Tsne, PCA and UMAP.

```
PCA took 0.27s
```



```
PCA took 0.01s
```





```
PCA took 0.02s
```

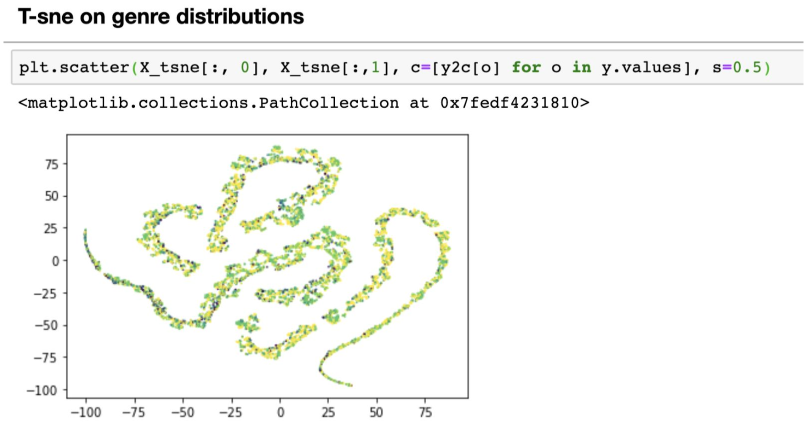
```
UMAP took 5.36s
```

```
UMAP took 3.26s
```

```
UMAP took 101.00s
```

Visualization of high-level musical features

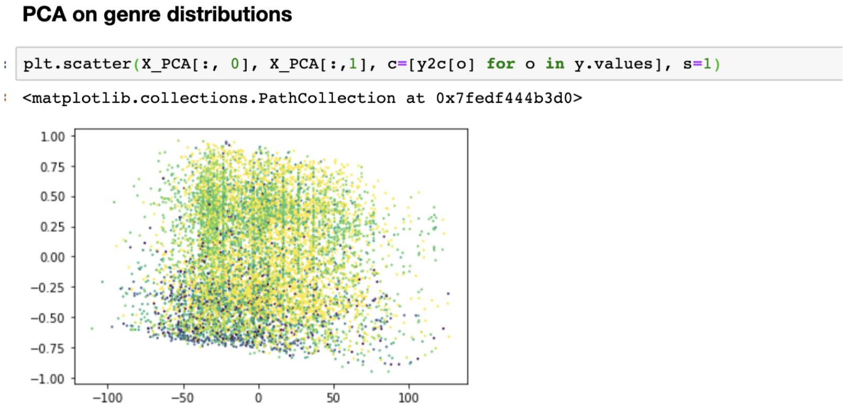
The combination of high level musical features including Acousticness, Danceability, Energy, Instrumentalness, Liveness, Speechiness, Tempo, Valence are visualized with respect to genre using 2 methods: Tsne and PCA, which the latter gives clearer separation between genres.



T-sne on genre distributions

```
plt.scatter(X_tsne[:, 0], X_tsne[:,1], c=[y2c[o] for o in y.values], s=0.5)
```

```
<matplotlib.collections.PathCollection at 0x7fedf4231810>
```



PCA on genre distributions

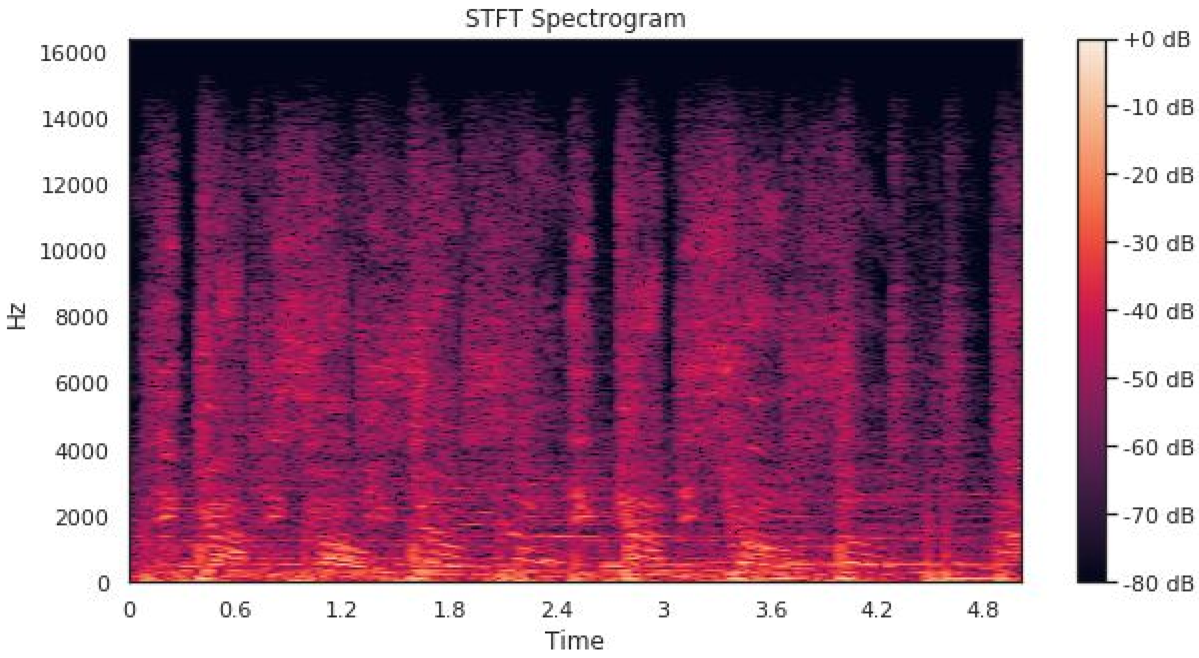
```
plt.scatter(X_PCA[:, 0], X_PCA[:,1], c=[y2c[o] for o in y.values], s=1)
```

```
<matplotlib.collections.PathCollection at 0x7fedf444b3d0>
```

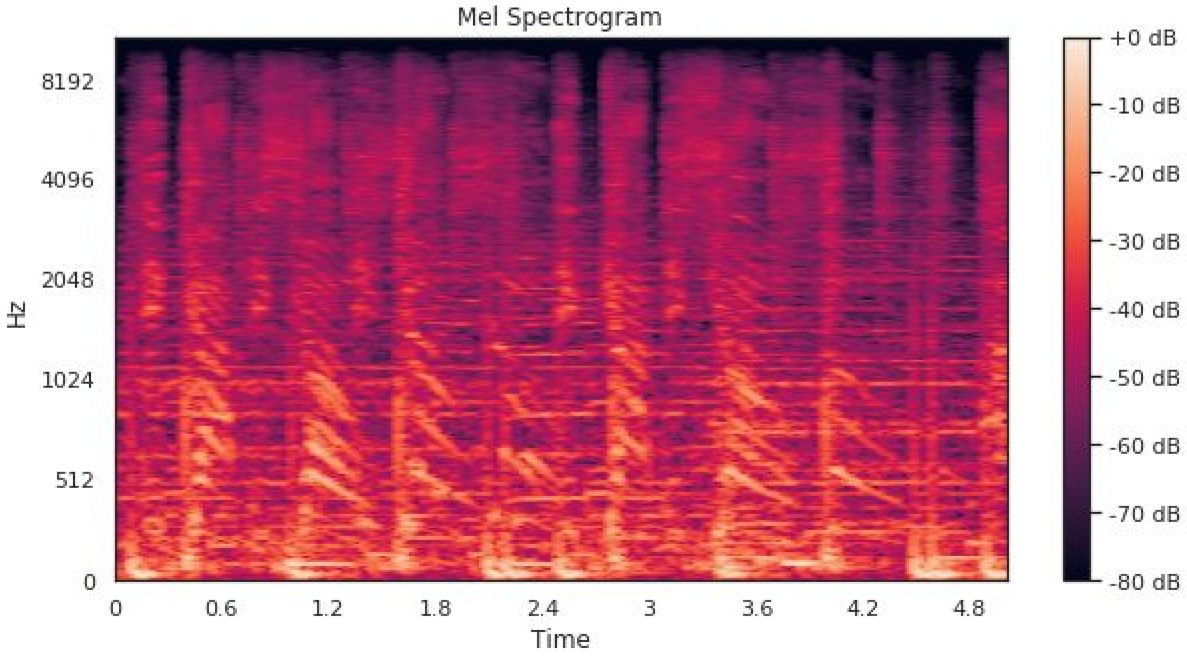
Data Pre-processing

In order to interpolate between audio samples, we need a generative model that's able to capture both the low-level and high-level features of audio clips. Earlier works such as WaveNet aim to do this by directly modelling the individual samples in audio clips. Since audio in its raw form requires hundreds of thousands of samples to represent just a few seconds, and due to the inherently serial nature of audio, modelling it in such a manner means that the learning algorithm would need to learn the relationships between samples that are hundreds of thousands of time-steps apart.

In order to enable our model(s) to capture higher-level information about audio clips, we convert the time-domain audio signal into a time-frequency domain spectrogram. We do this by sequentially applying an FFT over overlapping windows of the time-domain audio signal to obtain what's known as a Short-Time Fourier Transform (STFT).



To better represent the human perception of pitch, we rescale the frequency axis to use the Mel scale. In addition, we take the logarithm of each value in the spectrogram to more closely match how humans perceive loudness.



Since our goal is to be able to generate audio samples (and not just their spectrogram representations), we also need to be able to reconstruct audio clips from Mel spectrograms. Unfortunately, the reconstruction is unavoidably lossy because in deriving a Mel spectrogram we discard some temporal information (FFT Phase) and some frequency information (in converting to the Mel scale).

To convert a Mel spectrogram to audio, we must first recover the removed frequency information by converting the Mel scale to a linear frequency scale. Thereafter, we reconstruct the phase information and apply the inverse FFT function to recover the audio. The Griffin-Lim algorithm is one method of estimating the FFT phase array based on the FFT magnitude array.

The Small subset of the FMA Dataset contains around ~8000 audio clips trimmed to approximately 30 seconds each. After removing ~10 clips that were malformed or unusable, we converted each one to its Mel spectrogram representation. We then split each spectrogram into subsections of 640 frames each, which with our FFT parameters corresponds to 5 seconds of audio. Splitting the spectrograms in this manner and discarding any subsections shorter than the specified 640 frames gives us a clean, consistent dataset. The genre-balanced nature of the small subset of the FMA Dataset we use means that we do not have to perform any rebalancing.

We use \`librosa\` along with \`numpy\` to perform the majority of our data pre-processing. To ensure portability and to keep the disk-footprint of the generated spectrograms small, we store the generated spectrograms as OpenEXR images. OpenEXR is a lossless open image standard that supports 32-bit floating-point values, and is therefore ideal for holding spectrogram data.

When converting an audio clip to a Mel spectrogram, we note a tradeoff between reconstruction quality and reconstruction time. Using \`librosa\`, the spectrograms resulting from the parameters we chose take about 18 seconds to reconstruct (per second of audio) on a high-end consumer CPU.

The parameters we use for spectrogram generation are as follows:

Parameter	**Value**
Audio Sampling Rate	32768 Hz
FFT Window Size	256
STFT Hop Length	4096
Mel Frequency Bins	512

Problem and Algorithm / Model

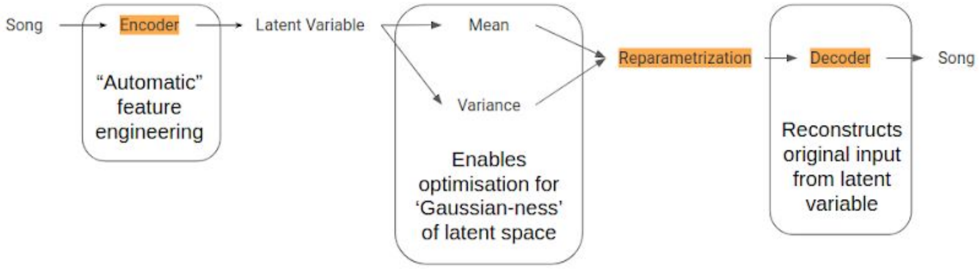
Inspired by MusicVAE (https://magenta.tensorflow.org/music-vae), a machine learning model that let us create palettes for blending and exploring musical scores using MIDI files, we intend to make a model that interpolate between two music excerpts using mel spectrogram, which describes richer content of the audio information.
MusicVAE used one latent space model called autoencoder (AE) which is capable of learning the fundamental characteristics of a training dataset. An autoencoder builds a latent space of a dataset by learning to compress (encode) each example into a latent vector, and then reproduce (decode) the same example from that latent vector. MusicVAE conducts reconstruction, sampling, and smooth interpolation based on this architecture.

Our Approach

Our tasks include:
1) unconditioned audio generation
2) audio interpolation

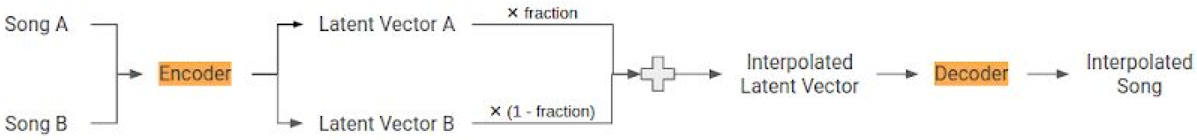
1)Variational Autoencoder(VAE) for latent space interpolation

In order to interpolate music, we need to explore on the variations of our original dataset.The standard autoencoder generates encodings specifically useful for reconstructing its own input using CNN.The loss function is usually either the mean-squared error or cross-entropy between the output and the input, known as the reconstruction loss, which penalizes the network for creating outputs different from the input. While VAE, not making its encoder not output an encoding vector of size n like standard autoencoder, rather, outputting two vectors of size n: a vector of means, μ, and another vector of standard deviations, σ. Hence,decoder learns that not only is a single point in latent space referring to a sample of that class, but all nearby points refer to the same as well. This allows the decoder to not just decode single, specific encodings in the latent space, but ones that slightly vary too.


Song
Encoder
"Automatic" feature engineering
Latent Variable
Mean
Variance
Enables optimisation for 'Gaussian-ness' of latent space
Reparametrization
Decoder
Reconstructs original input from latent variable
Song

Optimised against a combination of
Cross-Entropy (Reconstruction Loss)
and
KL-Divergence ("Gaussian-ness" Loss)

2) Latent Space Interpolation



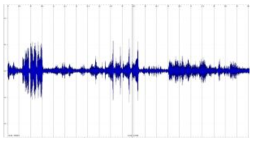
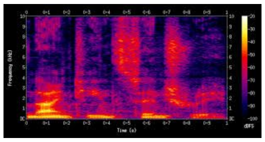
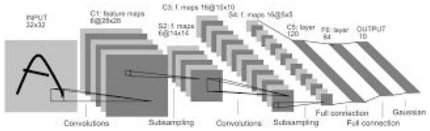
The interpolation of two music excerpts is done by linear algebra of the latent space learned by task(1) of music excerpt A and B and do a linear combination of encoded A and B latent vectors, the constant of "fraction" in the above equation is a scalar determining the magnitude and direction of the interpolation.

The following records our different experiments on building and improving the above 2 tasks.

Experiments

1)Baseline model

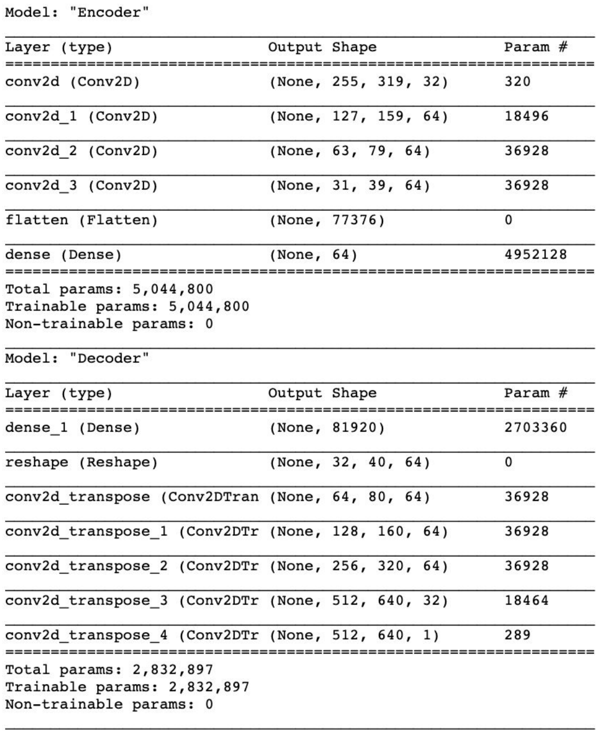
In our first attempt, we used CNN as encoder.


--->

--->


--->Latent Vector

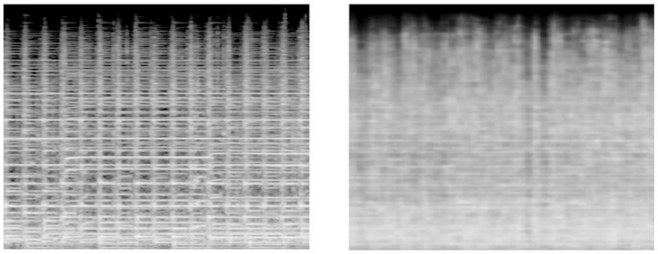
Audio Spectrogram (image) Convolutional Neural Nets(CNN)

Below is the baseline model summary:



```
Model: "Encoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 255, 319, 32)      320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 127, 159, 64)      18496
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 63, 79, 64)        36928
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 31, 39, 64)        36928
_________________________________________________________________
flatten (Flatten)            (None, 77376)             0
_________________________________________________________________
dense (Dense)                (None, 64)                4952128
=================================================================
Total params: 5,044,800
Trainable params: 5,044,800
Non-trainable params: 0
_________________________________________________________________
Model: "Decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 81920)             2703360
_________________________________________________________________
reshape (Reshape)            (None, 32, 40, 64)        0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 64, 80, 64)        36928
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 128, 160, 64)      36928
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 256, 320, 64)      36928
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 512, 640, 32)      18464
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 512, 640, 1)       289
=================================================================
Total params: 2,832,897
Trainable params: 2,832,897
Non-trainable params: 0
_________________________________________________________________
```

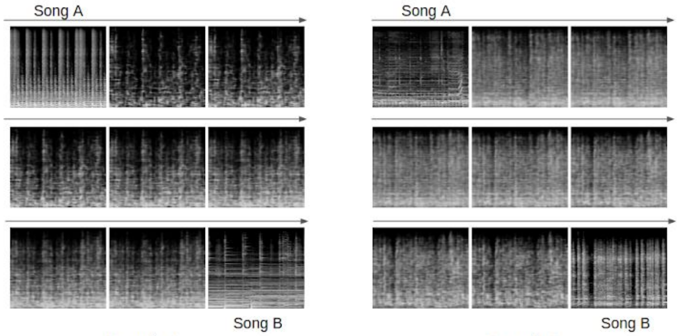
This gives results with one sample of reconstructed spectrogram from its latent vector as below:



Original spectrogram

Reconstructed spectrogram

Below are two samples of interpolated spectrogram from 2 spectrogram reconstructed from baseline model.



Example 1

Example 2

In the baseline model, the reconstruction result is not very similar to the original result.
Through tuning the parameters, we have found out that we are not able to reconstruct on a single model, after debugging, we found out the loss function we have applied before -- tf.nn.sigmoid_cross_entropy_with_logits first maps a logit in sigmoid, which is (0, 1) and then do minization on cross entropy loss. This works well on 0 or 1 digits classification, but however, it does not work well on output values that range bigger than 1. So in the end we have applied the mean squared entropy loss to minimize the loss. After which we are able to overfit on one sample.

2)Conv2D

3) Conv2D inception

The baseline model used CNN. Audio spectrograms are inherently sequential, which means pattern in the lower-left corner of a spectrogram does not mean the same thing as an identical pattern in the top-right corner. CNNs are translation-invariant, whereas spectrograms are not. A better model would take into account the fact that patterns in different locations on the spectrogram have different meanings (pitch, time, etc.) Hence in the following attempts, we used Cov1D to mitigate this problem.

3) Conv1D

In Conv1d, our original input shape is (batch_size, frequency_dimension, time_dimension), we have reshaped the

Evaluation Methodology

Our task of interpolation of two music excerpts can be viewed in 2 main tasks:
(1) unconditioned audio generation (2) audio interpolation

Evaluation Metrics for unconditioned audio generation contains the following questions which are asked in the survey:
Is the output fluent on transitions?
Does the output sound like before reconstructing?
Does the music sound good?

Evaluation Metrics for audio interpolation contains the following questions which are asked in the survey:
Does the output sound like a combination of the two inputs?
Are the features preserved after interpolating?
Are the genre preserved after interpolating?

Genre conservation is tested using genre classifier trained over FMA dataset. Below are our different attempts of genre classifier.

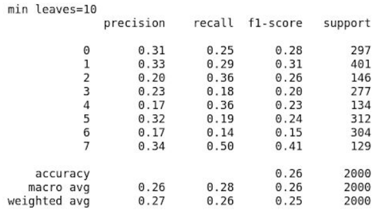
Genre Classifier

Machine Learning approach:

(different parameters are tried given same architecture, only the best result shown below)

1)Decision Tree

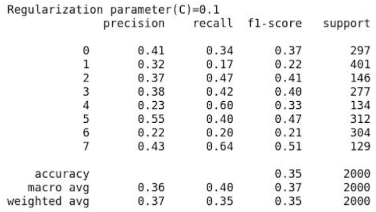
```
leaf_ls=[1,3,5,7,10,25]
```



```
min leaves=10
              precision    recall  f1-score   support

           0       0.31      0.25      0.28       297
           1       0.33      0.29      0.31       401
           2       0.20      0.36      0.26       146
           3       0.23      0.18      0.20       277
           4       0.17      0.36      0.23       134
           5       0.32      0.19      0.24       312
           6       0.17      0.14      0.15       304
           7       0.34      0.50      0.41       129

    accuracy                           0.26      2000
   macro avg       0.26      0.28      0.26      2000
weighted avg       0.27      0.26      0.25      2000
```

2)Logistic Regression

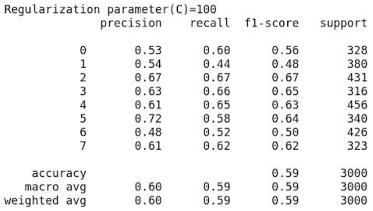
```
C_param_range = [0.001,0.01,0.1,1,10,100]
```



```
Regularization parameter(C)=0.1
              precision    recall  f1-score   support

           0       0.41      0.34      0.37       297
           1       0.32      0.17      0.22       401
           2       0.37      0.47      0.41       146
           3       0.38      0.42      0.40       277
           4       0.23      0.60      0.33       134
           5       0.55      0.40      0.47       312
           6       0.22      0.20      0.21       304
           7       0.43      0.64      0.51       129

    accuracy                           0.35      2000
   macro avg       0.36      0.40      0.37      2000
weighted avg       0.37      0.35      0.35      2000
```

3)SVM

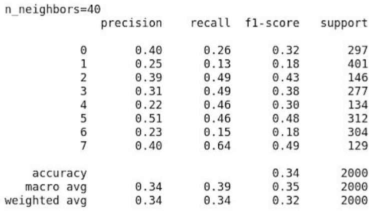
```
C_param_range = [0.001,0.01,0.1,1,10,100,150.200]
```



```
Regularization parameter(C)=100
              precision    recall  f1-score   support

           0       0.53      0.60      0.56       328
           1       0.54      0.44      0.48       380
           2       0.67      0.67      0.67       431
           3       0.63      0.66      0.65       316
           4       0.61      0.65      0.63       456
           5       0.72      0.58      0.64       340
           6       0.48      0.52      0.50       426
           7       0.61      0.62      0.62       323

    accuracy                           0.59      3000
   macro avg       0.60      0.59      0.59      3000
weighted avg       0.60      0.59      0.59      3000
```

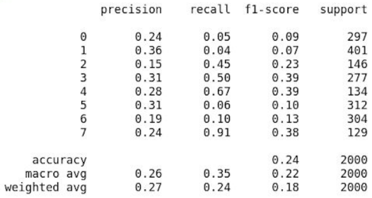
4)KNN

n_neighbours= [5,10,20,40]



```
n_neighbors=40
              precision    recall  f1-score   support

           0       0.40      0.26      0.32       297
           1       0.25      0.13      0.18       401
           2       0.39      0.49      0.43       146
           3       0.31      0.49      0.38       277
           4       0.22      0.46      0.30       134
           5       0.51      0.46      0.48       312
           6       0.23      0.15      0.18       304
           7       0.40      0.64      0.49       129

    accuracy                           0.34      2000
   macro avg       0.34      0.39      0.35      2000
weighted avg       0.34      0.34      0.32      2000
```

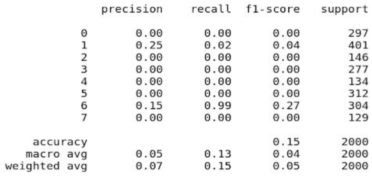
5)Guassian Naive Bayes



```
              precision    recall  f1-score   support

           0       0.24      0.05      0.09       297
           1       0.36      0.04      0.07       401
           2       0.15      0.45      0.23       146
           3       0.31      0.50      0.39       277
           4       0.28      0.67      0.39       134
           5       0.31      0.06      0.10       312
           6       0.19      0.10      0.13       304
           7       0.24      0.91      0.38       129

    accuracy                           0.24      2000
   macro avg       0.26      0.35      0.22      2000
weighted avg       0.27      0.24      0.18      2000
```

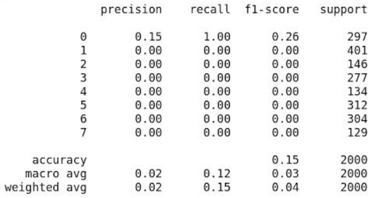
6)GMM

without kmeans mean init



```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       297
           1       0.25      0.02      0.04       401
           2       0.00      0.00      0.00       146
           3       0.00      0.00      0.00       277
           4       0.00      0.00      0.00       134
           5       0.00      0.00      0.00       312
           6       0.15      0.99      0.27       304
           7       0.00      0.00      0.00       129

    accuracy                           0.15      2000
   macro avg       0.05      0.13      0.04      2000
weighted avg       0.07      0.15      0.05      2000
```

with kmeans init (means initialized by one sampler by genre)

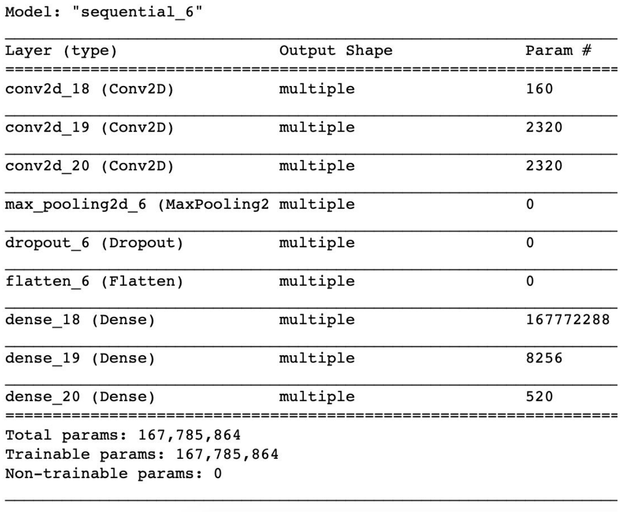


```
              precision    recall  f1-score   support

           0       0.15      1.00      0.26       297
           1       0.00      0.00      0.00       401
           2       0.00      0.00      0.00       146
           3       0.00      0.00      0.00       277
           4       0.00      0.00      0.00       134
           5       0.00      0.00      0.00       312
           6       0.00      0.00      0.00       304
           7       0.00      0.00      0.00       129

    accuracy                           0.15      2000
   macro avg       0.02      0.12      0.03      2000
weighted avg       0.02      0.15      0.04      2000
```

Deep Learning approach:

(Train on 35000 samples, validate on 8320 sample over 43000 samples in balanced dataset)

1)CNN_1:



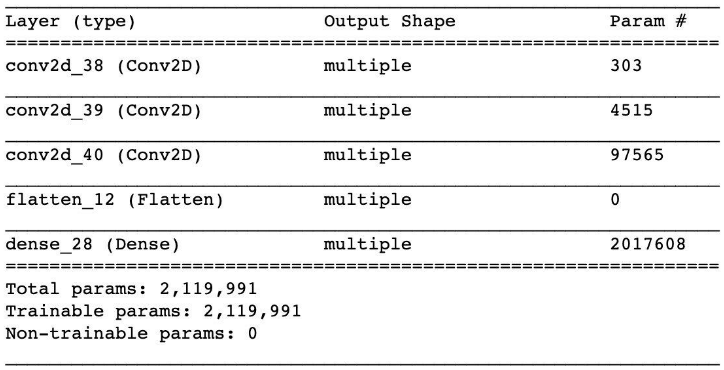
```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           multiple                  160
_________________________________________________________________
conv2d_19 (Conv2D)           multiple                  2320
_________________________________________________________________
conv2d_20 (Conv2D)           multiple                  2320
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 multiple                  0
_________________________________________________________________
dropout_6 (Dropout)          multiple                  0
_________________________________________________________________
flatten_6 (Flatten)          multiple                  0
_________________________________________________________________
dense_18 (Dense)             multiple                  167772288
_________________________________________________________________
dense_19 (Dense)             multiple                  8256
_________________________________________________________________
dense_20 (Dense)             multiple                  520
=================================================================
Total params: 167,785,864
Trainable params: 167,785,864
Non-trainable params: 0
_________________________________________________________________
```

Results:

Epoch Number	Train Accuracy	Validation Accuracy
1	0.4092	0.4099
2	0.6301	0.4128
3	0.7038	0.4012
4	0.8396	0.3960
5	0.9348	0.3952

Over the epochs, training accuracy increases quickly but validation accuracy remains at around 0.4 and drops after 2nd epochs. The molayer will be del is overfitting probably due to overly deep architecture. Fewer layers used in next attempt.

2)CNN_2:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_38 (Conv2D)           multiple                  303
_________________________________________________________________
conv2d_39 (Conv2D)           multiple                  4515
_________________________________________________________________
conv2d_40 (Conv2D)           multiple                  97565
_________________________________________________________________
flatten_12 (Flatten)         multiple                  0
_________________________________________________________________
dense_28 (Dense)             multiple                  2017608
=================================================================
Total params: 2,119,991
Trainable params: 2,119,991
Non-trainable params: 0
_________________________________________________________________
```

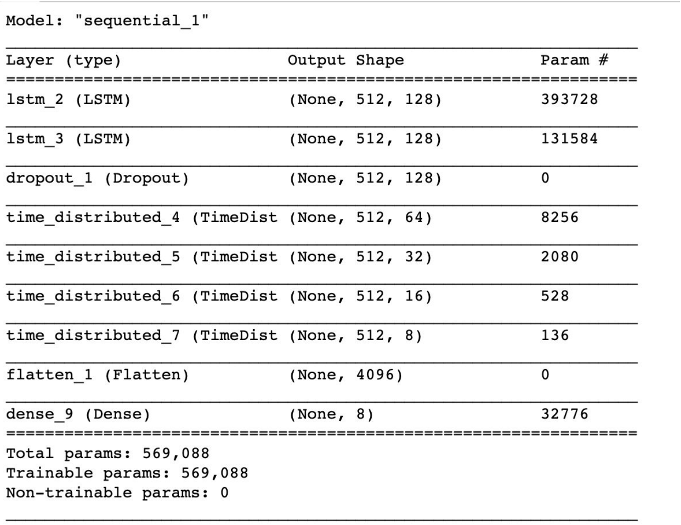
Results:

Epoch Number	Train Accuracy	Validation Accuracy
1	0.2664	0.3132

2	0.3903	0.3463
3	0.4623	0.4303
4	0.5096	0.4547
5	0.5533	0.4480
6	0.5969	0.4561
7	0.6573	0.4301
8	0.7309	0.4104
9	0.8925	0.3919
10	0.9712	0.3849

Over the epochs, training accuracy increases quickly over the epochs but validation accuracy remains at around 0.4 and drops after 6th epochs. Con2D may not work for time-series features and we will attempt on RNN (recurrent neural network), which may fit better for music features.

3)RNN:



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_2 (LSTM)                (None, 512, 128)          393728    
_________________________________________________________________
lstm_3 (LSTM)                (None, 512, 128)          131584    
_________________________________________________________________
dropout_1 (Dropout)          (None, 512, 128)          0         
_________________________________________________________________
time_distributed_4 (TimeDist (None, 512, 64)           8256      
_________________________________________________________________
time_distributed_5 (TimeDist (None, 512, 32)           2080      
_________________________________________________________________
time_distributed_6 (TimeDist (None, 512, 16)           528       
_________________________________________________________________
time_distributed_7 (TimeDist (None, 512, 8)            136       
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0         
_________________________________________________________________
dense_9 (Dense)              (None, 8)                 32776     
=================================================================
Total params: 569,088
Trainable params: 569,088
Non-trainable params: 0
_________________________________________________________________
```

Results:

Epoch Number	Train Accuracy	Validation Accuracy
1	0.3264	0.3681
2	0.4416	0.3927
3	0.4846	0.4028
4	0.5230	0.4106
5	0.5567	0.3972
6	0.5788	0.4038

7	0.6064	0.3950
8	0.6278	0.3679
9	0.6501	0.3791
10	0.6689	0.3668

Over the epochs, training accuracy increases over the epochs but validation accuracy remains at around 0.4 and drops after 4th epochs. Overfit issue also occurs.

Hence, given above attempts, SVM(with regularization parameter C equals to 100 gives the best validation accuracy)

Results and Discussion