University of BRISTOL

DEPARTMENT OF COMPUTER SCIENCE

Using Unsupervised Deep Learning for Song Similarity Recommendations

Matthew Stollery

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

Thursday 5th May, 2022

Abstract

This project explores the use of unsupervised neural networks to provide music recommendations based on song similarity. The model is trained in two phases upon a dataset of the 30-second preview clips of over 1 million Spotify tracks. Through the use of Triplet Margin Loss and a choice of three distance functions, recommendations can be provided for an input. Depending on the style of music used as the input, recommendation quality ranges from vaguely within the right genre, up to very good matches which echo the style and vibe of the track provided extremely well. Recommendations work best for ambient music, tracks with long sustained sections, as well as more defined genres such as hip-hop, but fall short on more high tempo music such as drum and bass.

Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Taught Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, this work is my own work. Work done in collaboration with, or with the assistance of others, is indicated as such. I have identified all material in this dissertation which is not my own work through appropriate referencing and acknowledgement. Where I have quoted or otherwise incorporated material which is the work of others, I have included the source in the references. Any views expressed in the dissertation, other than referenced material, are those of the author.

Matthew Stollery, Thursday 5th May, 2022

Contents

List of Figures

List of Tables

Ethics Statement

This project did not require ethical review, as determined by my supervisor, Neill Campbell.

Supporting Technologies

- I used the Python library `PyTorch` to implement the neural network architecture.
- I used the Python library `NumPy` to aid with processing and usage of large arrays
- I used the Python library `scikit-learn` to create t-SNE plots
- I used the Python library `Matplotlib` to aid with the visualisation of plots and figures
- I used the "Spotify 1.2M+ Songs" dataset to aid in the training of the model[19].
- I used the Spotify API, along with the Python implementation `Spotipy`, to gather preview clips of Spotify tracks

Notation and Acronyms

CSV	:	Comma-Separated Values
MSE	:	Mean Squared Error
TML	:	Triplet Margin Loss
AE	:	Autoencoder
LR	:	Learning Rate
GPU	:	Graphics Processing Unit
API	:	Application Programming Interface
CUDA	:	Compute Unified Device Architecture, an API to allow software the use of GPUs
t-SNE	:	T-Distributed Stochastic Neighbor Embedding
NaN	:	Not a Number

Chapter 1

Introduction

Many online music streaming services provide users with music recommendations using a technique called 'collaborate filtering' - a method which combines the similarities between items and similarities between users in order to *"recommend an item to user* **A** *based on the interests of a similar user* **B***"*[22]. While in practice this works well for many users, being part of the algorithm used by music-streaming giant Spotify[15], it relies on information about other users in order to function. This can unintentionally lead to a bias towards popularity, as it is difficult to recommend products with limited historical data. My goal with this project is to build a deep learning model in order to recommend songs objectively and without user data, achieving this by training the model via unsupervised learning upon a dataset of the previews of 1.1 million tracks. I will then evaluate the effectiveness of these recommendations using both subjective metrics (personal option) as well as objective features (Spotify's built-in feature analysis). Figure 1.1 demonstrates how a given input flows through the entire system, first generating an efficient encoding of the spectrogram, then generating an embedding where similar tracks lie nearby. From this, recommendations can be suggested through one of a number of distance functions, before being plotted using t-SNE to visualise the similarity in two dimensions.

A paper by D. Fleder, K. Hosanagar explores the problem of current recommender systems further, stating the "rich-get-richer" effect that recommenders such as collaborative filtering can create [20]. This is supplemented by anecdotal user stories, with one user on Reddit[10] commenting how they received the same track twice within three skips, highlighting not only to the extent that popular songs are prioritised[1], but also that such songs may be suggested so frequently as to be noticeable within short listening sessions. While the latter point does not fall under the scope of this project, it still highlights one of the shortcomings of mainstream music players. In addition, the point about prioritisation of already popular tracks is exactly the goal that my proposition aims to address, with the goal that an objective measurement and the lack of user data removes the incentive to recommend songs more due to popularity.

Further cause for moving away from approaches requiring the use of user data is that of mitigating the *"cold-start problem"*. This problem forms when there is a lack of data about either users or products in a system, in other words *"The cold start problem occurs when the system is unable to form any relation between users and items for which it has insufficient data"*[25]. This problem is part of why collaborative filtering, and services which use it such as Spotify, tend to recommend popular songs lots and underground / unpopular songs less. The low interaction aspect of the cold-start problem occurs for example when new items are added to the catalogue, meaning they have few or no interactions, leading to poor quality or non-existent recommendations. The system that I propose does not use user data or interactions to feed its model, and hence will be unaffected by the cold-start problem.

There is evidence within literature that collaborative filtering approaches - and indeed many existing recommendation systems in general - do not work well for fans of certain genres[24]. This paper by D. Kowald et al. delves into the reasons why such systems tend not to work for "beyond-mainstream music listeners", stating *"recommendation quality is significantly better for users with mainstream taste than for users who prefer beyond-mainstream music across all recommendation approaches"*. This is expanded by defining four non-mainstream subgroups of music in 'folk', 'hardrock', 'ambient' and 'electronica', learning that from these ambient receives the best recommendations while 'hardrock' receives the worst. These sub-groups of beyond-mainstream music are ones I can seek to explore when evaluating my recommendations, and their findings can be used to compare to my own.

[1]the song in question ("Future - Life Is Good ft. Drake") has over 2 billion views on YouTube

My solution, which would aim to provide a more objective measure of similarity driven solely by musical content, has the aim of being able to be used to recommend music to users without the consideration of popularity. This would avoid the “rich-get-richer” effect proposed by D. Fleder and K. Hosanagar[20], and hopes to address the concerns of underground music fans who just wish to find new music to listen to. While this method may not work for everyone, the hope is that for some users it helps them find new music - no consideration of other users in the system means that popular songs are weighted exactly the same as new songs with no plays yet. As a result, this could be used as a tool to help hardcore fans of specific genres to find new music from up-and-coming artists before their peers.

There are a number of challenges to overcome in this process - the first being the dataset. While big companies such as Spotify have their own database of millions of tracks, with easy access to the full length of each tracks, I do not have such access. I settled on the use of the “Spotify 1.2M+ Songs” dataset by R. Figueroa [19], a list of roughly 1.2 million Spotify tracks. The dataset comes in CSV format, containing basic information such as track id, artist and song names, as well as more abstract metrics such as liveliness, acousticness and valence. I opted not to use these latter metrics, primarily due to the fact that I did not know how they were calculated, and could not be sure it was objective - the goal of this project is to create an objective learned measure of similarity, and I felt this was easiest to achieve by using only the musical content as input, with no extra pre-calculated features.

Further challenges lie in that of learning what it means for two segments of music to sound ‘similar’, provided only with raw audio information. This challenge manifests in two ways - the first being how to train a model to learn similarity, with the second being how to quantify how similar whole tracks are using this method. When processing these dataset items, it becomes easier to trim entire tracks down into smaller clips, meaning that each track is comprised of a number of distinct clips. In the final model, this in turn means that each track is represented as a sequence of points in some higher-dimensional space. How does one determine how similar two sequences are when the visualisation of two dimensions is not present? I explore a number of methods with varying degrees of success for this, overall learning that each serves a different purpose and favours different definitions of what it means to be similar.

The high-level objective of this project is to train a model to recognise similar segments of music, and to determine a way to quantify this similarity to provide recommendations. More specifically, the concrete aims are:

1. Develop a network architecture to embed a sequence of audio within a higher-dimensional space, where nearby data points indicate similar sounding audio
2. Train this network upon a dataset of over 1 million Spotify songs
3. Develop a method to quantify similarity over entire sequences
4. Evaluate the effectiveness of these recommendations via the use of both subjective and objective metrics

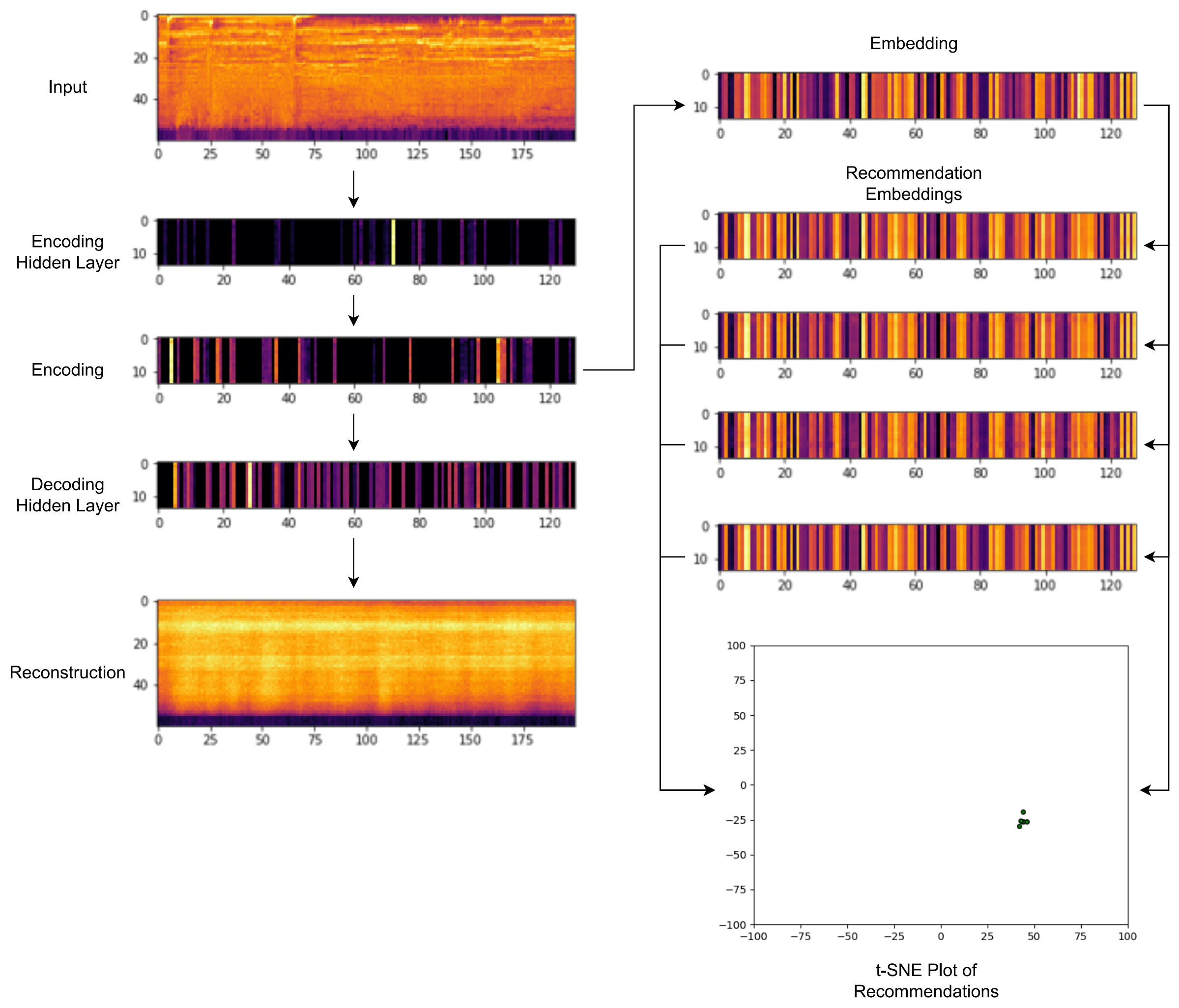


Figure 1.1: Flow of a given input throughout the entire system

Chapter 2

Background

2.1 Technical Background

2.1.1 Deep Learning

What is Deep Learning?

Deep learning is a method of machine learning which aims to mimic the human brain through a combination of data inputs, neurons, weights and biases, which can be combined to build network structures to solve a variety of problems.

A **neuron** or node is a component of layers in a network which can be thought of as acting like a neuron in the brain. The output of a neuron can be described in terms of the following[1]:

- Its input components $\mathbf{x} = x_0, x_1, ...$
- The weights of the connections (or **synapses**) to it $\mathbf{w} = w_0, w_1, ...$, which define how much each connection to the neuron affects the result
- The bias applied to the neuron b, which can be used to rebalance the output value by some amount (e.g. if it results to zero it may correct that to some degree)
- An activation function f which determines whether the outgoing connections should 'fire' based on the weighted sum of the prior components, allowing the modelling of complex non-linear relationships

Combined, the output of a neuron is:

$$y = f((\sum_i w_i x_i) - b)$$

There are two main stages in the training of a network: the **forward pass** (or forward propagation) and the **backward pass** (or backwards propagation). The goal of the forward pass is to calculate and store the intermediate variables of the network as well as the output, going from input to output through each layer in turn. From the calculation of the outputs of all neurons in the network, the output layer indicates either a classification (in the case of supervised learning), a reconstruction or pattern recognition (in unsupervised learning), or an action (in reinforcement learning). Using a **loss function**, the error between the actual and expected output can be calculated, and the next phase can begin.

Backwards propagation, also called backpropagation, is the step where the network corrects the weights and biases previously calculated in order to try and reduce the output of the loss function. This can performed by a method called **gradient descent**, which aims to find the minimum of a given function (in this case the loss function) given the state of the network. Along with the state of the network, this also takes in a parameter called the **learning rate**, which dictates how far the weights and biases are nudged in the direction of the gradient. This can be summed up as:

$$x_{i+1} = x_i - \alpha \nabla f(x_i)$$

[3] where x represents the weight or bias being corrected, α represents the movement amount (or learning rate), and ∇ represents the gradient of the function f at this point. The calculation of this gradient relies

on calculating the derivative, which can be helped by application of the chain rule. Repeated application of these steps results in the model converging to a local minima of the cost function, which indicates a solution in which the model achieves the goal it is set out to solve to some degree.

There are two types of gradient descent - **batch** gradient descent, which takes into account the entire training data when calculating gradients, and **stochastic** gradient descent, which only considers a random sample of data[13].

What are the types of Deep Learning?

There are three main types of learning that can be performed by a neural network - **supervised** learning, **unsupervised** learning, and **reinforcement** learning. Supervised learning is characterised by the inclusion of data labels, with the goal of the network being to learn a set of weights and biases to most accurately match up input items to their corresponding labels, with the goal of generalising to unseen inputs. Unsupervised learning on the other hand is characterised by the lack of labels, often being used to learn patterns and similarities in data. One common way this is used is to learn efficient representations of input items by the inclusion of a bottleneck, through which an item must pass through before being attempted to be reconstructed on the output layer, teaching the network to learn an efficient and accurate representation of the input universe in order to best reconstruct it.

Finally, reinforcement learning contains no analogue to labels, instead learning by receiving signals from the environment in which it operates (positive value indicating a desired outcomes and vice versa), with the outputs of the layer often being mapped to actions an agent (the part that makes decisions based upon said rewards) can perform. This is often used for gaming AI (Artificial Intelligence) systems, as well as for providing personalised recommendations[17], as these contexts allow clear rewards to be provided to the agent.

Why is Unsupervised Learning a good fit for this project?

While one of the mentioned learning types, reinforcement learning, has applications in personalised recommendations, it is not suitable for this project. This is because this is a slow process, and learns recommendations on a user-by-user basis. The goal for this project however is to learn objective measures of similarity, and provide general recommendations on the basis that users are suggested songs that sound similar to an input.

Unsupervised learning also does not require the use of labels, and while it is possible to assign labels to music (based on genre, tempo or a combination of features), the goal of this project is to learn similarity based on content alone, not upon any human-defined labels such as genre. My method will use an **Autoencoder** to learn an efficient representation, before using this representation to learn to separate dissimilar tracks.

What is an Autoencoder?

An autoencoder is a simple network structure, acting somewhat similar to compression, which works by funnelling the input through a bottleneck before attempting to reconstruct it on the output layer. The loss of the network can be easily defined as the difference between the input and the reconstruction defined by a difference function, with common examples including L1 loss ($L = |x - \hat{x}|$) and MSE (Mean Squared Error) ($L = \frac{1}{N}\sum_{i=0}^{N}(x - \hat{x})^2$).

2.1.2 Triplet Margin Loss

There are many common loss functions, with popular examples such as MSE simply measuring the difference between an output and a target. To learn similarity for the use of recommendations however, it is not good enough to just be able to learn similarity between similar tracks - we must also learn *dissimilarity* between different tracks. This requires the use of a loss function that takes three inputs, such as **Triplet Margin Loss** (also known as Triplet Loss or TML). Figure 2.1 visually explains how this works, with the general idea being to learn to make an input (or **anchor**) point closer to a **positive** sample, and further from a **negative** sample. This can be described by the loss function equation:

$$L(a, p, n) = \max(d(a, p) - d(a, n) + m, 0)$$

where m is a margin value to keep negative samples far apart and d is some distance function relating the two samples[30].

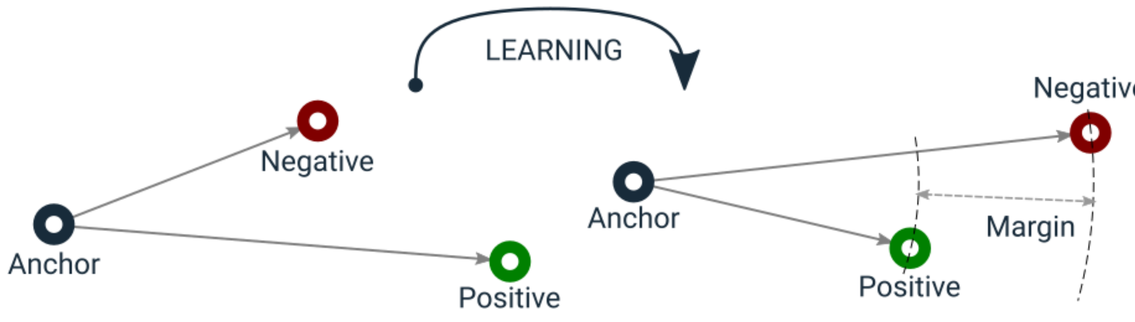


Figure 2.1: An example of how triplet loss learns (source: [30])

This loss function works well for this project, as recommendations should not learn to be clumped together, otherwise all songs could end up being ‘close enough’ to be recommended. As a result, similarity should be learned such that songs that sound different are kept far apart. The problem here is that we do not already know what other songs sound similar - however this can be solved by assigning other segments of the input track as the positive sample, as in general songs flow smoothly and so this prediction should hold. The negative sample can therefore be any other random track, with the hope that the probability of this negative sample being a similar sounding track is near zero.

2.1.3 t-SNE

t-SNE, or t-Distributed Stochastic Neighbor Embedding, is a method of visualising high-dimensional data in a much lower (usually 2 or 3) dimension via a non-linear mapping, which is particularly good at revealing structure. This was first introduced by L. van der Maaten and G. Hinton in their paper “Visualizing Data using t-SNE”[31]. It differs from other dimensionality reduction visualisations by using a *“Student-t distribution rather than a Gaussian to compute the similarity between two points in the low-dimensional space”*, which helps alleviate crowding.

This can be useful in this project for visualising the high-dimensional spaces in which the input tracks are embedded, in order to visually see patterns or similarities between tracks.

2.1.4 Spectrograms

A spectrogram is a method of representing audio in two dimensions - a frequency dimension and a time dimension. This is particularly useful for visualising, and is achieved via the use of the **Fourier transform** - a method of decomposing a signal into it’s individual frequencies and frequency amplitudes. The transform of a function g at a frequency f is given by the following:

$$G(f) = \int_{-\infty}^{\infty} g(t)e^{-2\pi ift}dt$$

A spectrogram can therefore be created by dividing the input into short, equal-length segments, and applying the Fourier transform over the length to acquire the frequency data during that window. This creates the two-dimensional image-like representation of sound described, however has the limitation of by default providing lots of information about high frequencies, which humans find difficult to differentiate between. Humans tend to hear sound non-linearly, so to help the model differentiate frequencies in a similar way we can use what is known as the ‘**Mel**’ scale. This scales the frequency axis by some non-linear transformation, such that the frequency divisions of the spectrogram more accurately mimic how humans perceive sound[21].

2.2 Related Works

A short project by B. Ross and P. Ramakrishnan for their final year project at Stanford titled “song2vec”[28] uses a very similar method to what I plan to do, aiming to classify music obtained from Spotify 30-second preview clips as well as *“metadata about the tracks, albums, and artists”* in the dataset. They use an

Adversarially Learned Interface to compute an embedding, representing the success of an embedding as one that defines a function mapping similar songs to clusters, minimising distances within clusters. They use this to visualise genres and classify tracks based on said genres, determining that their project performs better than traditional embedding methods such as PCA. This project demonstrates the power of using deep learning to measure song similarity, however they do not extend their work to recommend new songs, instead only classifying songs based on genre.

An article by T. Vassallo titled "Calculating Audio Song Similarity Using Siamese Neural Networks"[32] aims to solve a similar problem by utilising SNNs (Siamese Neural Networks), a variant on CNNs (Convolutional Neural Networks) in which two architecturally identical branches are used, such that *"one branch accepts a 'reference' track mel spectrogram as an input while the other accepts a 'difference' track mel spectrogram as an input"*. Vassallo states that mel spectrograms are a good way of representing timbre, and as a result the acoustic characteristics of a track as a whole. The model acts as a feature extractor, allowing a similar method to my proposed project where an input track can be provided, and recommendations returned by calculating the distance to existing feature vectors. This project however shows limited example cases, only including three, yet notes that they succeed in creating a recommendation system that relies solely on audio signal, something I hope to achieve here too.

Another project by J. Cleveland et al. explores the use of content-based music similarity with Triplet networks[16]. This paper cites the same issues described here such as the 'cold start problem', aiming to solve this via the use of Triplet networks (a variant on Siamese networks intended to classify similarity). They perform this embedding by using the artist as a label, *"such that two songs by the same artist are embedded closer to one another than to a third song by a different artist"*. They find that performance is improved when using a random third sample versus using one of a chosen different genre. They use artist and genre labels, but acknowledge that these labels are often too broad to assess similarity.

Z. Noshad, A. Bouyer and M. Noshad[26] approach a similar problem in their paper "Mutual information-based recommender system using autoencoder", wherein they use an autoencoder to *"solve the reliability of the recommendations and updating user ratings based on new ratings"*. They do this in the context of films on a Netflix dataset, achieving good results by using an autoencoder architecture with mutual information to produce a similarity graph.

While not looking to solve a similar problem, Y. Sakashita and M. Aono explore some interesting audio-specific methods of feature extraction in their submission to the 2018 edition of the DCASE audio classification competition[29]. In this, they use both mono and stereo versions of an audio input to compute a number of new spectrograms, such as harmonic and percussive spectrograms, as well as the difference between left and right channel spectrograms. These are combined to be fed into the network, but the idea of harmonic-percussive source separation (HPSS) is interesting. However, I am aiming for this project to be general, so the specific consideration of certain types of features may not be something I explore at this stage.

While not presented as projects in themselves, there are a number of websites offering recommendations that are detached from other major music platforms. Tunebat[4] is one such website, listing features of given songs as well as providing recommendations for "Harmonic Mixing" based on these values. These features appear to be taken from Spotify's feature analysis tool, and while they are provided for all tracks available on the site, recommendations are not provided for less popular tracks.

Songslikex[5] is a similar platform, offering playlists created from similar tracks, however provides no information about how recommendations are determined based on the input. This too falls victim to not offering recommendations for less popular tracks, and this is something I intend to provide with my project.

Chapter 3

Project Execution

3.1 Dataset

3.1.1 Overview

The final dataset chosen for this project was 'Spotify 1.2M+ Songs' by Rodolfo Figueroa [19]. This came in the format of a large CSV file, containing basic information on tracks such as Spotify ID (a unique identifier for each track on Spotify), song title and artists. It also contained more abstract information such as acousticness and energy which, while potentially useful, are pre-computed. As I did not know how these are determined, I opted to ignore this data, however it could potentially be useful for validation or for possible future extensions of this project. As such, the main benefit of this dataset was its large collection of raw track IDs, and as such was useful to me as a collection of as many tracks as possible.

While other datasets such as the 'Million Song Dataset'[8] are available, they all have various drawbacks, with the main one being the lack of any way to obtain the raw audio. Such datsets possess large amounts of feature analysis and metadata for tracks, but lack raw audio for understandable copyright reasons. In addition to this, the aforementioned dataset was the next largest dataset I could find, but was last updated in 2012, meaning it would be rather outdated for today's use.

3.1.2 Acquiring the Data

Once I had chosen the Spotify 1.2M+ Songs dataset as my dataset of choice, the next step became acquiring the data I needed. The plan for this project was to measure similarity using only the song content itself, and without using pre-computed features obtained by third parties. The Spotify API[7] is designed for integration with and access to user playlists, but is also able to quickly look up information about a track given its unique Spotify track ID. Crucially, one of these pieces of information available through the API is a preview URL associated with each track. This URL links to a 30-second long audio preview of a given track, and is exactly what I need to be able to obtain. This is something which would not be possible with generic song datasets, hence the need for a Spotify-specific one.

Using the Python plugin `Spotipy`, a lightweight Python library for the Spotify Web API[9], I was able to write a script to go through Figueroa's dataset, obtain the preview URL of each track, and use a web request to download it locally. These files come in the format of .mp3 files, an audio format that uses lossy compression. This means that each file is relatively small (only 354kb for 30 seconds of audio), but this is due to a low quality 96kbps bit rate. For comparison, if one has a high speed internet connection, Spotify streams music at 160kbps for free users and 320kbps for premium users[12], while high quality premium streaming services such as Tidal offer subscriptions to receive up to 9216kbps[6]. While the latter is excessive, the point is that the preview tracks downloaded are of a much lower quality compared to even what Spotify offers to its own customers. As a result, quality may become a problem in limiting flexibility with this project.

Despite this, the quality is good enough for now, and while the Spotify API imposes a roughly 25,000 requests per 24 hours limit, I was eventually able to obtain 1,087,907 preview tracks. This is noticeably less than 1.2 million, as not all tracks had an associated preview URL - approximately 90.36% of tracks had one. I attempted to figure out whether there was a pattern in which tracks did not have preview URLs in order to account for it in testing, however I couldn't identify such a pattern. Popularity was not a factor, as even my own tracks which receive single-digit plays per month have downloadable previews! Many of the tracks without URLs were older tracks from before Spotify as a service existed, however there are also plenty more tracks from before this time that have URLs. Overall, I cannot tell why some have preview URLs and others do not, however it seems to be a random selection of tracks and should not affect results.

I supplemented this dataset with a few more tracks from popular trending playlists as well as my own personal playlist to allow myself to test the project on my own music more easily, bringing the total up to 1,088,508 audio tracks. This added up to a total of 369GB of raw audio - however it wasn't in a usable form yet.

3.1.3 Processing the Data

While raw audio in a format such as .mp3 is great for listening, it isn't great as an input to a network. Networks typically convert audio to a format called a spectrogram, which acts as a snapshot of the audio. A spectrogram can be thought of as a series of slices of audio at regular timesteps, where the x-axis is the time and the y-axis is the frequency. Each cell in the spectrogram has a value proportional to the amplitude of the frequency band within that given timestep, with high values indicating prominent frequencies and vice-versa. The frequency information can be obtained via a *Fourier Transform*, which is a mathematical formula to allow the decomposition of a signal into its individual frequencies and amplitudes - in other words, it converts from the time domain to the frequency domain. The Fourier transform therefore allows the acquisition of frequency information within a given period of time, forming a slice which can make up the whole image.

For this project, another Python script was created to take each downloaded audio file, compute its log-mel spectrogram, and save it to a file. The parameters of this were the use of a 40ms time window with 50% overlapping windows (giving 50 samples per second), and 60 frequency bands. This lead to each 30 seconds of audio being converted to a 1500x60 image - or so was the intent. It however turns out that not all of the preview clips are exactly 30 seconds long, and so I had to account for this. Any clips longer than 30 seconds can be safely cut down to 30 seconds, and while this loses some information, it's impossible to know if the information lost is in any way more valuable. Only a small percentage of tracks have this issue, so it must be done for consistency. Some tracks however have clips less than 30 seconds long, and unfortunately we have to discard these - there is no easy way to convert these into a format that both stays consistent with the others and also keeps the same information in the spectrogram. Simply appending silence to pad the difference would mean later on that the model learns to make audio similar to silence (which we don't want), and stretching the audio to fill the space would change the track, time-stretching it and altering how it sounds. Changing the parameters for these tracks to simply take a different window size would not be consistent, and it would be bad practice to mix and match parameters like this. Of the tracks in the dataset, only around 0.06% of them had to be removed for this reason, so the loss of data is not significant.

One final issue that appeared in this was the occasional presence of NaN values in the spectrograms, or 'not a number' values. This can occur when dividing by zero, or simply when the audio file is slightly corrupted. The issue with these values is that they overpower any existing calculations, including those in the backpropagation step of learning, meaning that NaN values propagate throughout the network and quickly reduce the network loss to NaN as well. Roughly 0.1% of dataset files had this issue and also had to be discarded. It is worth noting for completeness that many more spectrograms than this had NaN values at the first timestep, due to the fact that many audio files have a short lead-in to the track - this could easily be solved by trimming the first window of the spectrogram.

Figure 3.1 shows a number of examples of generated spectrograms of various types of music. Note how in the ambient and piano tracks, the consistent high low droning frequencies (at y=0) persist throughout, with only occasional flutters of high frequency, while the drum and bass track is much more high energy and full of frequencies, so has a much brighter spectrogram throughout. The hip-hop example shows clearly defined lines where the drums hit, indicating the level of detail present in the spectrograms. Note in all of the examples the relative lack of information at the highest frequencies (bottom of the y-axis) due to the low bitrate of the preview audio.

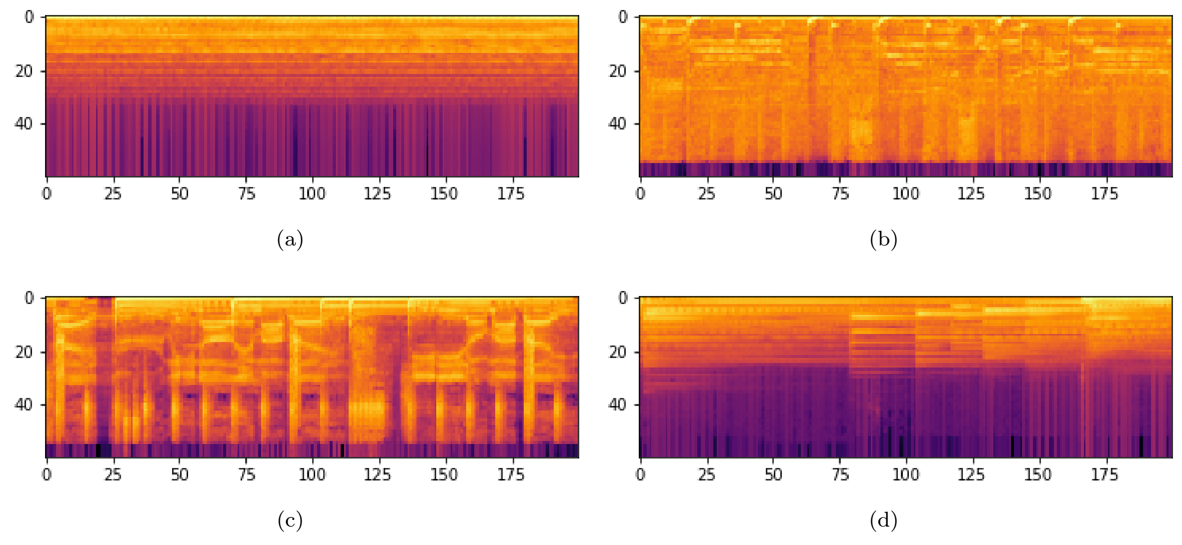


Figure 3.1: Spectrograms for segments of a number of different input tracks in the dataset. (a) An ambient track (b) A drum and bass track (c) A hip-hop track (d) A piano track x axis : time in samples (1 sample = 40ms), y axis : frequency bands (lowest at y=0)

3.2 Model Training

Model training was done in parallel with data collection, as obtaining audio for the entire dataset took over a month. However, with so much data available as time went on, it was still possible to continuously train. It is worth noting though that due to the vast amounts of data available to train with, no overfitting measures were implemented - this could certainly be addressed at a later stage, exploring methods such as early-stopping and audio-specific data augmentation, but as generalisation is usually done to compensate for lack of data, it was not necessary for this project.

Both models are trained using the ‘Adam’ optimiser, which implements stochastic gradient descent, with a learning rate of 10^{-3}

The training of the model was split into two separate stages, which I call phase 1 and phase 2. Phase 1 acts as the feature extraction, while phase 2 is responsible for learning similarity.

3.2.1 Phase 1 - Feature Extraction

Goal

The goal of phase 1 was to take in the spectrograms, and use an autoencoder to learn an efficient representation (**encoding**) in a much smaller space, such that it can recreate the input spectrogram using the features. This therefore learns a set of features that represent each input spectrogram without having to load the entire spectrogram each time. The goal of this is to use these encodings as an input to the next phase, which will learn a new set of features (**embeddings**) that can be used to measure similarity.

Architecture

The first step in creating this model was to create an architecture - for this I decided to use an autoencoder, as it does a number of things I desire for this project. The first is that it is designed for unsupervised learning, in other words we do not supply the model with target labels to map to. As the point of this project is to provide a similarity measure using only the audio, there is no label for the data, and unsupervised learning is perfect for this.

I began to implement this following a guide from A. F. Agarap on Medium[14], and created the network structure shown in figure 3.2. In this diagram, the item below the layer name represents its size - `H` represents the height of the input spectrogram while `W` represents the width. The variable `N` is a parameter I can tune, which represents the size of the bottleneck to use for feature extraction.

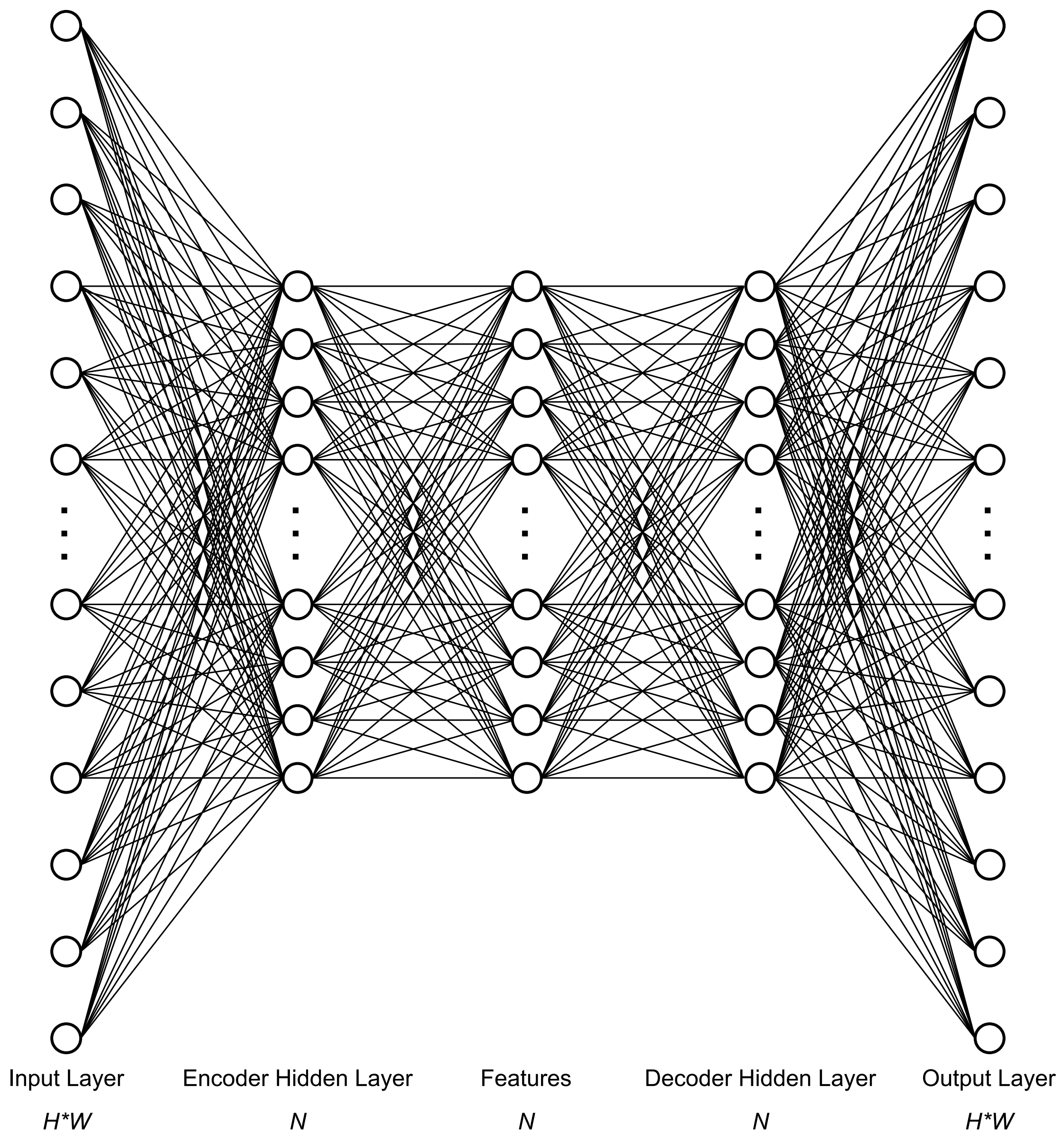


Figure 3.2: Phase 1 Fully Connected Architecture Diagram

All the layers in this model are fully connected, meaning each node on one layer is connected to every node on the next, which allows $H \times W \times N$ weights to be tuned in the hidden layers. The network is set up in such a way that the input spectrogram is received in the input layer, where it is compressed through a hidden layer of size `N` and into an encoding / features layer, where the encoding is learned. This encoding is then fed through a mirrored version of the network before this point, passing through a decoding hidden layer to an output layer of the same size as the input. This allows the input and output to be compared, and the loss and recreation accuracy computed.

Loss

For this first phase, I opted to use the mean-squared error loss function, which computes the error as follows:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

where y_i is the model's prediction (in this case the output) and $\hat{y}_i$ is the ground truth (in this case the input). As such it computes the difference between the input and the output, and can be used as a metric to minimise with training. I can also compute the recreation accuracy in a similar way by using the cosine similarity, which measures how similar two vectors are irrespective of magnitude, only angle. This is the metric I used to rank different encoding sizes, and is calculated as follows:

$$\text{sim}(X,Y) = \cos(\Theta) = \frac{A \cdot B}{||A||||B||}$$

where Θ is the angle between the vectors, $X \cdot Y$ represents the dot product (which itself is defined as element-wise multiplication $a \cdot b = \sum_{i=1}^{n} a_i b_i$, and $||X||$ represents the magnitude of X, or its length.

Performance

Letting the model train with a number of different encoding sizes for 10 epochs, I created table 3.1 to compare the evaluation accuracy (recreation accuracy over the entire dataset) versus the encoding size. The table shows an increase in evaluation accuracy as the encoding size increases, however there are diminishing returns to going from 32 to 128 (although a relatively large improvement in going to 256). Note that these figures were initially obtained early in the development process before the whole dataset was acquired, so while the table seems to suggest that decreasing the layer size from 128 to 64 or even 32 yields very little loss in accuracy, I opted to consider only 64, 128 and 256 when going forward. This is because I assumed that while the model performed well with small layer sizes at this point, it would struggle to generalise when attempting to recreate all 1.1 million spectrograms. It turns out however that the trend remained even on a full dataset, so the use of smaller layer sizes could be something to explore in a later iteration of this project. Larger sizes have the penalty of file size when saving encodings to be visualised, which deterred me from exploring larger layer sizes - additionally, the goal of the autoencoder is to learn an efficient representation, so any tweaks to layer size would aim to increase efficiency by decreasing feature count rather than increasing it.

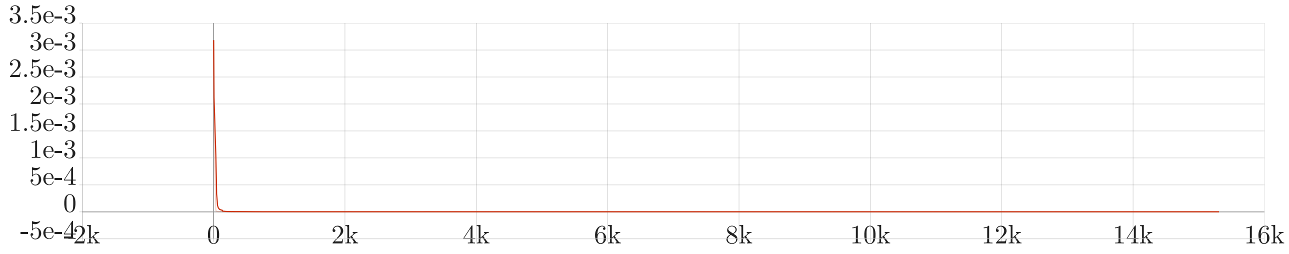
Encoding Size	Evaluation Accuracy
256	95.4%
128	94.4%
64	94.4%
32	94.0%
16	93.0%

Table 3.1: Comparison of Encoding Size (of the hidden layers) against Evaluation Accuracy.

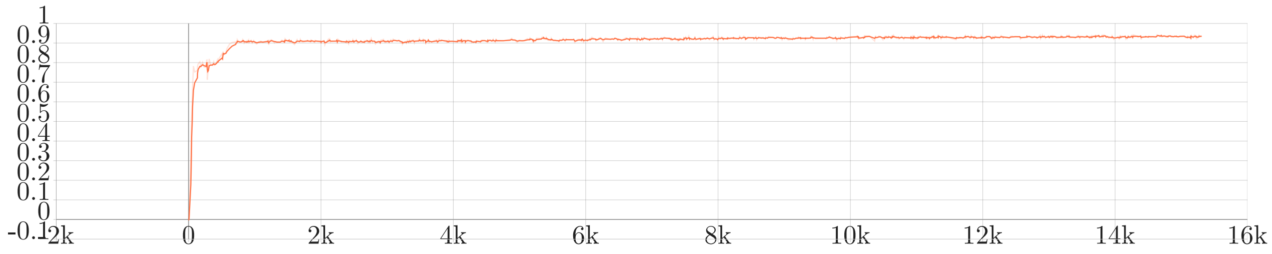
Figure 3.3 shows the loss and accuracy graphs for a full epoch of phase 1 with an encoding size of 128 and batch size of 64 on the full dataset. With these, it is easy to see how quickly the model learns to recreate the input spectrogram with so much data to work with. Despite this, for the final phase 1 model I nonetheless let it run for a full 10 epochs. This was partially due to the speed this phase took to train, and partially due to the fact that at this stage overfitting is not a huge concern. The dataset I have is all the data I could possibly obtain, and there is no need to account of new items outside the dataset at this stage. However, if this project were to be furthered with more data or even integrated into an existing service, overfitting measures could be explored as dataset updates could bring new data which must be generalisable with the model.

Trimming Clips

The input for this phase of the training, and indeed for the next phase, differs slightly from the original spectrograms created in the dataset processing described in section 3.1.3. Since the audio spans 30 seconds, it has a lot of time to change throughout this time, and as a result I decided to trim the original spectrogram into a number of segments, which can be processed separately within the same batch. Initially, this was done by splitting the 30-second sequence into 10 $\times$ 3-second long clips with no overlap. However, as explained in section 3.2.2, I made the move to overlapping clips in phase 2, and for compatibility applied this to phase 1 as well, now instead splitting the sequence into 14 $\times$ 4 second long clips, with an overlap of 2 seconds. The increase in time was due to simple overlaps with 3 second long clips either not lining up well to 30 seconds, or producing too many extra data items, and an increase from 10 to 14 was manageable. There is negligible difference in phase 1 performance when using overlapping clips, but the consistency with phase 2 was welcome.



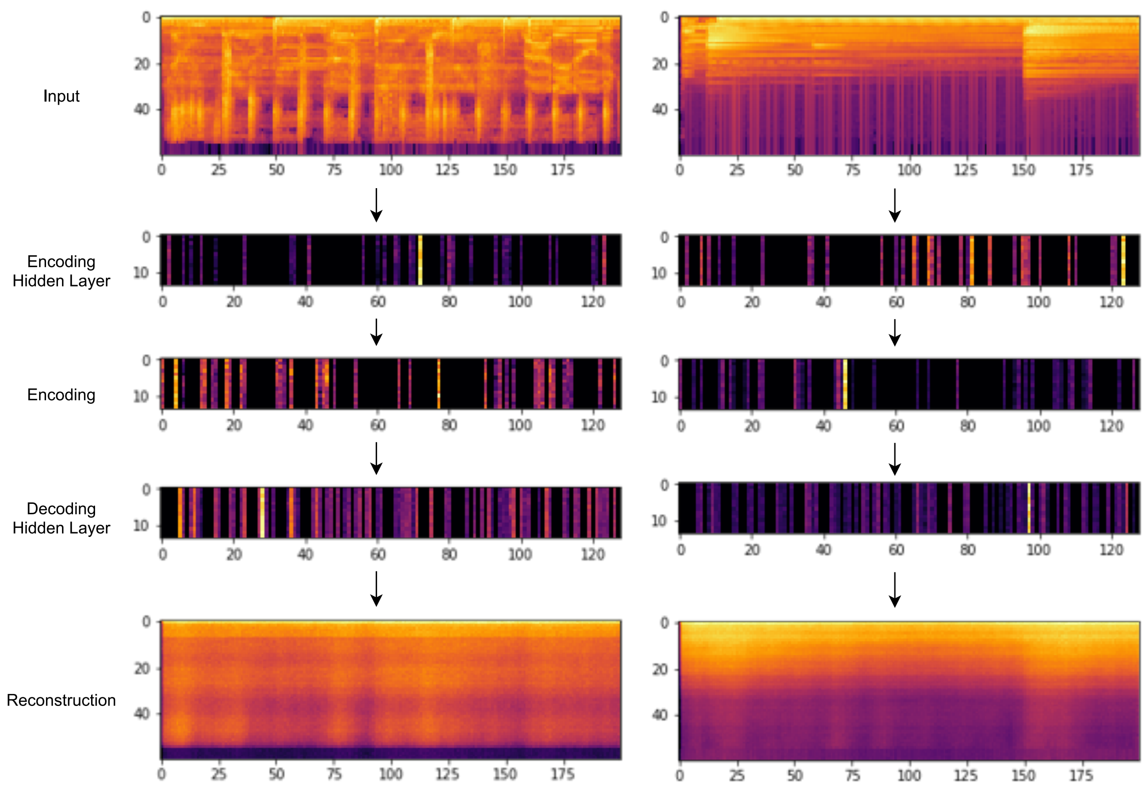
(a)



(b)

Figure 3.3: (a) Loss curve and (b) Accuracy curve for 1 epoch of Phase 1 training at encoding size = 128, batch size = 64
x axis represents step number, y axis represents (a) step loss (b) accuracy

Examples



(a) (b)

Figure 3.4: Diagram of the flow through the model in Phase 1 for
(a) A piano track, Spotify ID '4NRR3UaAWxE3GyAeA4W72M' and
(b) Old Town Road, a hip-hop track, Spotify ID '0F7FA14euOIX8KcbEturGH'

Finally, figure 3.4 shows two examples of how spectrograms change throughout the model, with (a) showing an ambient piano track, and (b) showing a hip-hop track. It is clearly noticeable how the output resembles a smoothed version of the input, however this causes it to lose fine details despite the high evaluation accuracy. Furthermore, observe how at an encoding size of 128 many of the features are unused within a given input. Note that for these figures, brighter colours represent higher values, the spectrograms represent a 4 second long segment of audio, and the encoding and hidden layer images show the node values for each of the 14 segments stacked from top to bottom.

3.2.2 Phase 2 - Similarity

Goal

The second phase of learning had the goal of finding a new set of features to describe the data (now called an **embedding**) which can be used to find similarity - it is trained with the aim of making similar sounding segments of music closer together in this embedding, while distancing it from different sounding segments. This can be used to ultimately provide the recommendations, as tracks that sound similar should fall in nearby regions of space (the dimensionality of which is defined by the size of the embedding), meaning that for a given input this space can be checked for the closest N tracks. This embedding ideally wants to be either the same size or smaller than the encoding found in phase 1, so for testing I used encoding sizes between 64 and 256 and embedding sizes of 64 or 128 to measure loss for a given combination. The results of this, along with the corresponding phase 1 accuracies, are shown in table 3.2.

Encoding Size	Embedding Size	Phase 1 Accuracy	Phase 2 Loss[1]
256	128	95.13%	0.01254
256	64	–	0.01824
128	128	94.82%	0.02066
128	64	–	0.02990
64	64	94.53%	0.03562

Table 3.2: Comparison of Accuracy and Loss with different combinations of Encoding and Embedding size

As can be seen, loss values depend more on the initial encoding size than the actual final embedding size, with 256 → 64 having significantly less loss than 64 → 64. Despite this, I once again went with a compromise and did a majority of the final evaluation using 128 → 128, with it being a middle ground between good performance and not requiring a doubling of the encoding size. Nevertheless, in later iterations it may be useful to experiment with other combinations in order to investigate if they provide better recommendation performance.

Architecture

The two phases of model training have been separated due to the need for a fully trained phase 1 *before* being able to start phase 2. There must exist be a fully fleshed out encoding from spectrograms to features before those features can be used to learn the new embedding. This is done by saving the final model from phase 1, then loading that model into phase 2 using Pytorch's checkpoint system to gather weight values from the trained model. The `encode` function can then be used to obtain only the spectrogram's encoding, which is passed into the model for phase 2. This model by itself is incredibly simple, being just a single fully connected layer going from size N to size M. Here, N is the encoding size from phase 1 and M is the embedding size used for phase 2. The architecture diagram for this is shown in figure 3.5.

While table 3.2 shows loss figures already, at the early stages of development I had not yet found a way to train this part of the model well. My usual loss function, mean squared error, only measures loss between a target and a predictions - however for this stage, we are trying to learn similarity, and as such there is no one target for each sample. Moreover, learning simply to make encodings similar to another similar sounding track (which itself is not defined yet), could potentially lead to many separate items becoming similar, as it does not learn to keep its distance from tracks that sound very different. As such, I chose to switch to using **triplet loss**.

[1]Note that for these values, Phase 2 was trained on a trimmed Dataset of 100k items for quicker training times. Once the best sweet spot was identified, the final model was trained upon the full dataset for use, and the overall loss also decreased.

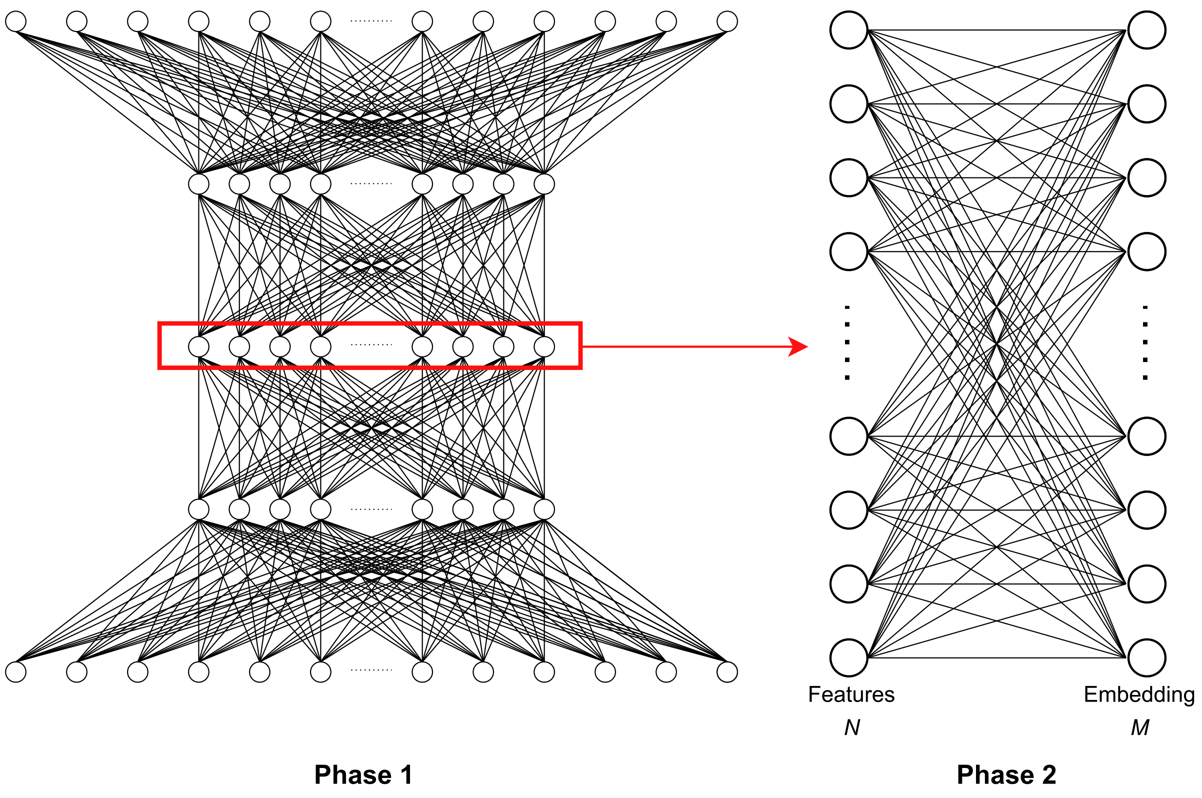


Figure 3.5: Full Architecture Diagram

Triplet Margin Loss

This, also known as triplet margin loss, measures loss using three inputs rather than two - an **anchor** (the learned embedding), a **positive** input (or a matching input, corresponding to an embedding we want to learn to be *similar* to), and a **negative** input (or non-matching, corresponding to an embedding we want to stay separate from).

Inputs

The problem now became determining how to assign these positive and negative inputs - the matching input is meant to be one which is similar, however that is exactly the problem I aim to solve with this project. I however decided that the best approach to this was to assign adjacent segments of the input track as the positive anchor, operating under the assumption that music in general flows smoothly between sections. In reality, tracks sometimes have harsh boundaries where style changes (e.g. going into a chorus), however these cases should be relatively rare - and even if not, it might be useful to consider the verse and chorus of one track to be similar, so that the encoding does not change too drastically over the course of a song.

As for the negative input, this can be assigned as a random other track from the dataset. While it is possible that this randomly selected track does indeed sound similar, with a full dataset of over 1 million unique tracks this should be rare, and usually this randomly selected track will be completely different and should not affect training.

Training

As a result of determining the inputs necessary for for stage, I could begin training. For each batch in the training dataloader, we can assign the original batch to be the anchors and create two new batches, one for the positive inputs and one for the negative inputs. I modified the dataset `__getitem__()` function (originally intended to return the dataset item required for that batch) to not only return the data, but

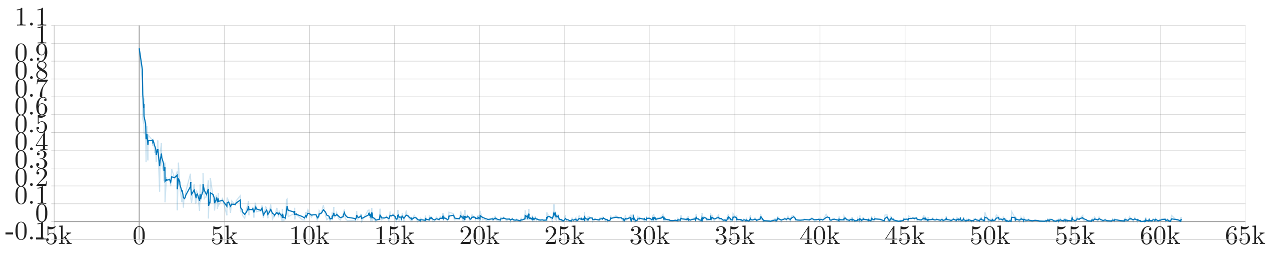


Figure 3.6: Loss curve for 1 epoch of Phase 2 training at encoding size = 128, batch size = 16

also the index of that data item (along with the filename for writing the order of evaluation to a file). Using this index, I could create the positive batch by returning each item of the anchor batch shifted over by some amount of time in the `x` axis, in order to represent a time shift (and so that when trimmed, index 0 of the anchor batch will effectively be compared to index 1 of the same batch, which represents a few seconds later in the same track). The negative batch could be created by returning a collection of random items from the dataset that are not equal to the input index, so as to ensure we never try to separate a track from itself. This itself would be rare, however I learned during evaluation that some tracks have multiple copies in the dataset, so the probability of this occurring in training is not zero, but is however negligible.

These three steps were combined into a new `__gettripletsample__()` function, which is used as follows in the training loop:

```
foreach anchor_batch, index in test_loader:
    initialise pos_batch, neg_batch

    foreach id in index:
        a, p, n = full_dataset.__gettripletsample__(id)
        pos_batch += p
        neg_batch += n

    # prepare batches for training

    anchor_encoding = encode(anchor_batch)
    pos_encoding = encode(pos_batch)
    neg_encoding = encode(neg_batch)

    anchor_embedding = embed(anchor_encoding)
    pos_embedding = embed(pos_encoding)
    neg_embedding = embed(neg_encoding)

    loss = TML(anchor_embedding, pos_embedding, neg_embedding)
    total_loss += loss
```

With this, each batch is initialised as empty, and has each new item appended to it inside the loop over all indexes present in the original batch. Each batch is encoded and then embedded in turn, before performing triplet loss on the outputs. `__gettripletsample__()` returns a triple of the anchor, the positive sample and the negative sample.

Results

The result of this is a model that can be trained to make encodings similar to adjacent segments of the same track, while also making them different to other tracks. Figure 3.6 shows the loss graph for one epoch of this, showing how it learns much more slowly than the phase 1 model, but still quickly learns to separate tracks. There is no equivalent for an accuracy function for phase 2 as there is no target, so training is left for a number of epochs to ensure good results.

To verify that this had indeed learned to classify similarity, I used a t-SNE plot to visualise a number of tracks in this embedding space. A t-SNE plot is a dimensionality reduction of the input (in this case a full list of all embeddings produced by the model), such that higher-dimensional inputs (in this case 128d) can be visualised in a lower (2d) image. This process is non-linear, but does aim to maintain distances between points, making it ideal for investigating the effectiveness of the model. With the model trained on the entire dataset, I visualised the mean position of 893 tracks from my personal playlist, almost all of which fall into the category of electronic music in various different styles. This produced the plot shown in figure 3.7.

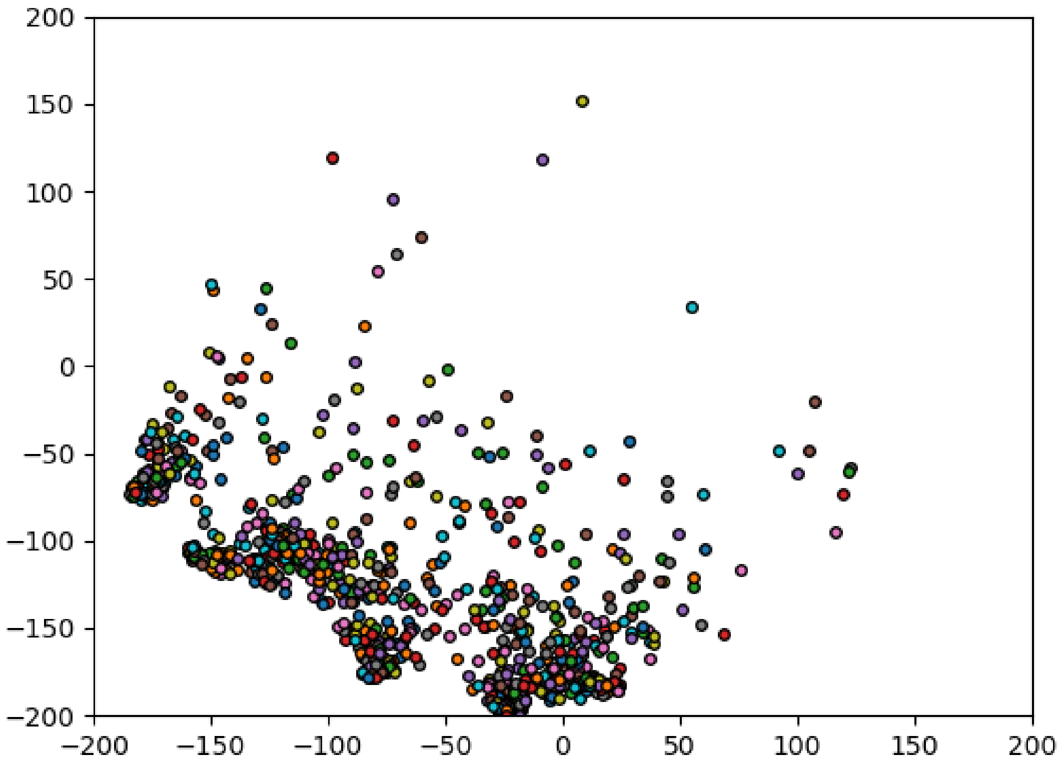


Figure 3.7: t-SNE plot of embeddings for the 893 items in my personal playlist, their mean points plotted in their positions in the entire embedding of all dataset items. Colouring is random

As can be seen in the figure, even with the reduction from 128 dimensions to 2 dimensions, most of the similar style tracks in my playlist fall in similar regions of space, even forming clusters in separate areas, likely corresponding to the exact style and feel of different tracks. Some outliers are visible outside of the main clusters, but this is to be expected either due to the nature of tracks in the playlist, variances in training, or even just the drawbacks of a two-dimensional representation of 128-dimensional space.

Comparing this to a plot of the first 10k tracks processed by the evaluation, shown in figure 3.8[2], it is clear how the inclusion of more varied data fills up the space - even so, there is a clear pattern to the shape formed, not simply filling up the entire space. It is however difficult now to determine similarity for so many tracks, so the best way to analyse the effectiveness of the model is to gather concrete recommendations, which is shown in section 3.3.

Examples

Figures 3.10 and 3.9 show examples of the stages of the model when given spectrograms are passed in, using the same examples as in 3.4. It is clear to see here that while the phase 1 representations were sparse, the phase 2 embedding utilise many more features for a given input.

[2]Note that while the evaluation of the embeddings produced an output for all 1.1 million tracks, due to memory limitations of available hardware it was impossible to visualise the entire space in t-SNE at once. As a result, the t-SNE plot shown is a combination of 10 separate t-SNE plots. Note also that it is impractical to show large numbers of data points, so small subsets are shown for clarity.

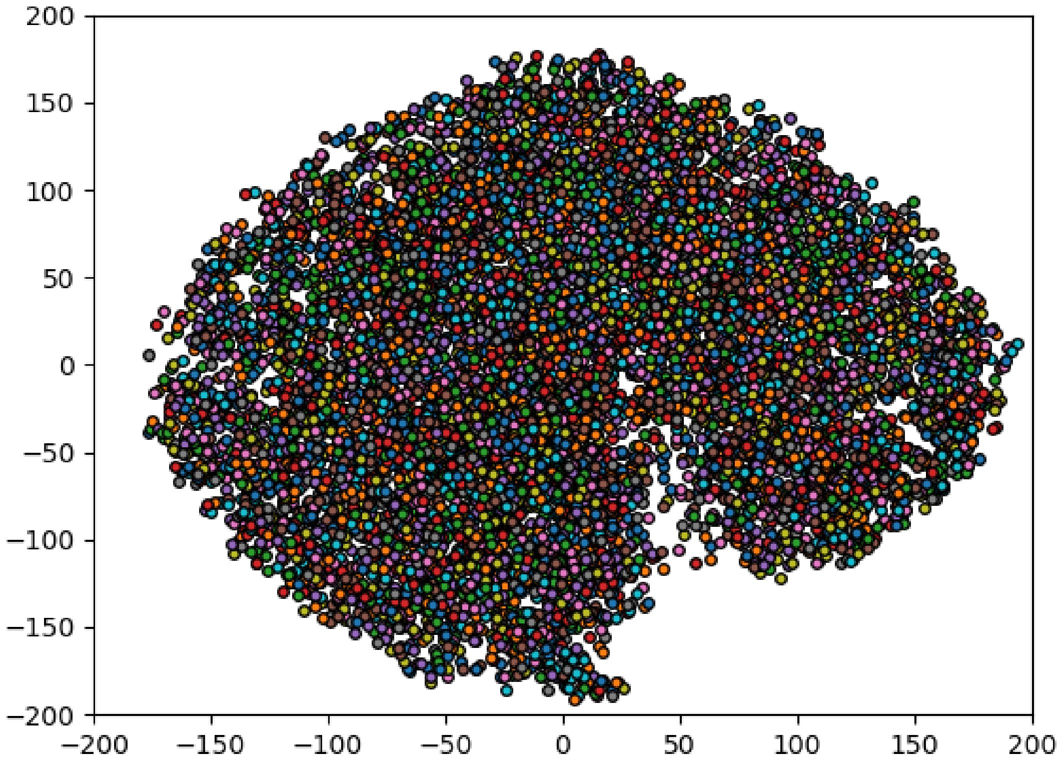


Figure 3.8: t-SNE plot of embeddings for first 10,000 items processed in the evaluation, their mean points plotted in their positions in the entire embedding of all dataset items. Colouring is random

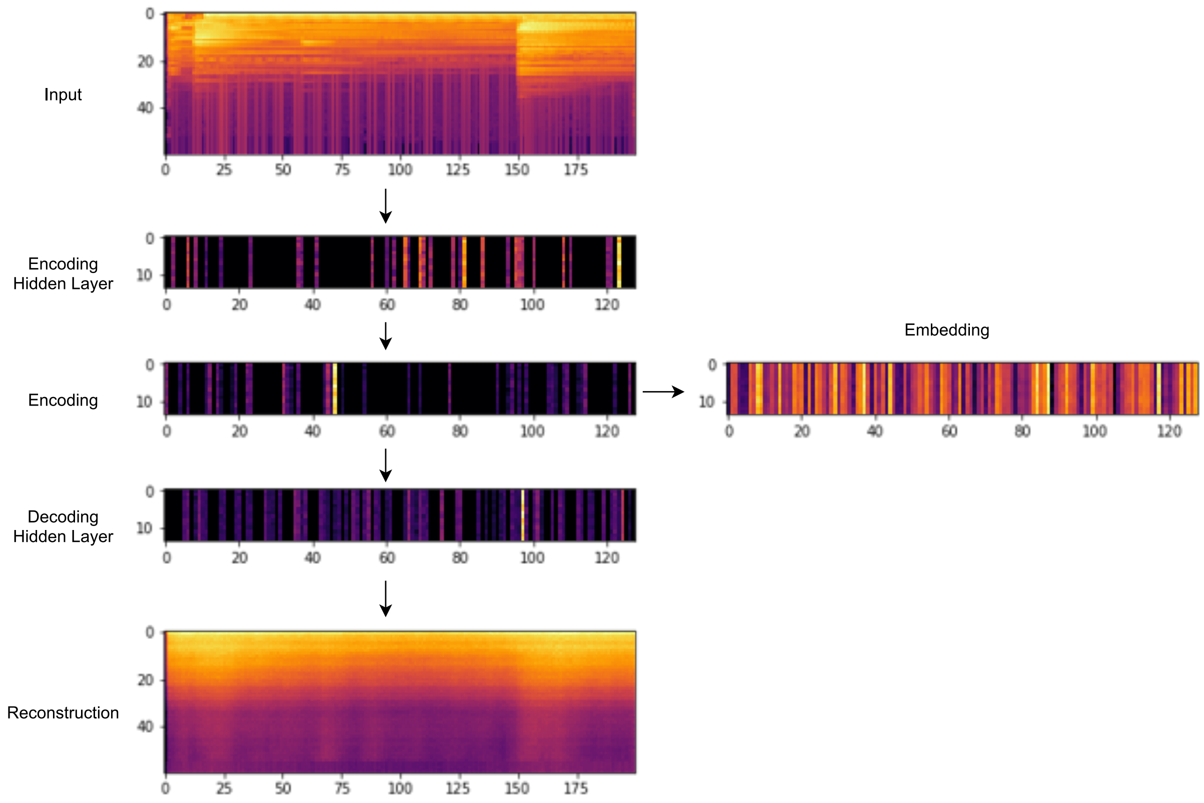


Figure 3.9: Diagram of the flow through the model in Phases 1 and 2 for a given input (A piano track, Spotify ID ‘4NRR3UaAWxE3GyAeA4W72M’)

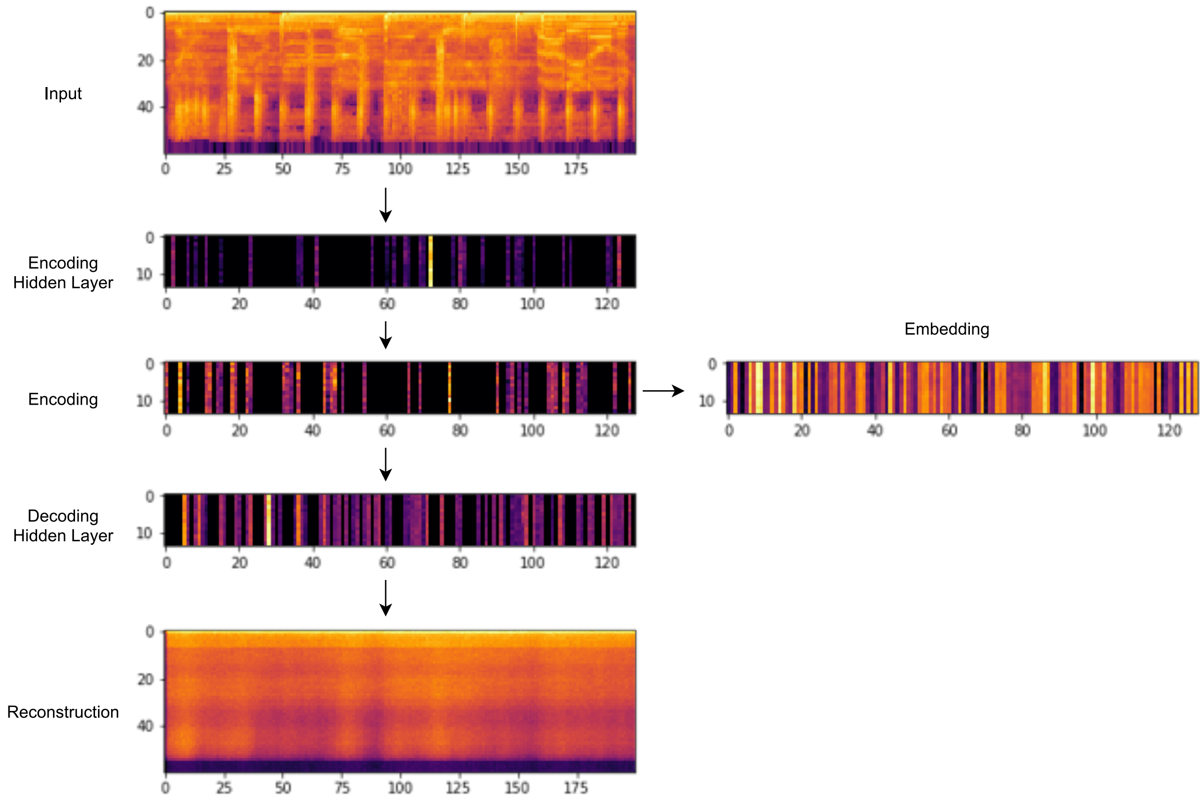


Figure 3.10: Diagram of the flow through the model in Phases 1 and 2 for a given input (Old Town Road, a hip-hop track, Spotify ID ‘0F7FA14euOIX8KcbEturGH’)

3.3 Evaluation

The evaluation stage of this project consisted of actually generating the recommendations with the help of phase 2 of model training. This is as simple as taking in a track, retrieving its embedding from phase 2, and then finding which other embeddings it is closest to. However, this raises a question - what does it mean for two sequences of points to be 'close'?

3.3.1 Distance Functions

When defining the distance between two points, the Euclidean distance is perhaps the most intuitive method to use - it can be thought of as the length of the line segment between the two points, and generalises easily to higher dimensions. In n dimensions, the distance between two points p and q is defined as:

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

However, while this works well for the distance between two individual points, the representation of tracks in the final embedding is a sequence of 14 distinct points. This is because a full 30-second long track is split into 4 second long segments that overlap for 2 seconds each, giving us 14 points. As such, to generate recommendations from the model, we need to define a way to measure how similar two curves (or sequences of lines, sometimes known as 'polylines') are in 128-d space. This in turn gives us a method to say how similar two tracks' embeddings are. I tested three such methods for this, which are outlined below.

1. Simple mean-to-mean Euclidean distance

When armed with a way to measure the distance between points, by far the easiest way to measure the distance between curves is to find the mean of the curve (also referred to as its centre of mass), and simply compute the distance between the mean positions. If we assume that segments of tracks are fairly close together in the embedding (as that is the goal of phase 2 of the model), this mean position would be fairly representative of the points as a whole. However, for tracks which have lone segments that are detached from the main body, or that even have very little pattern in their segments at all, this method is unlikely to work well, as the mean position may be far away from the individual points making up the line.

Calculating a mean in 128 dimensions is simpler than it may seem - it follows the same principle as two-dimensional or even one-dimensional means. if we have a function `mean`, which takes in an array of numbers and returns the sum divided by the number of items (as in it computes the arithmetic mean) as follows:

$$\text{mean}(X) = \frac{\sum_{x \in X} x}{\text{len}(X)}$$

then to calculate the mean of a set of m points in n-dimensional space, we do the following:

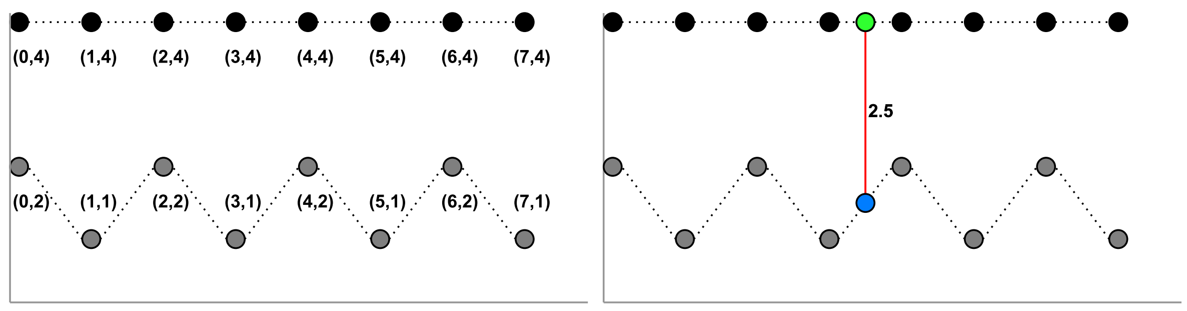
$$p_0 = (p_{0,0}, p_{0,1}, ..., p_{0,n}), ..., p_m = (p_{m,0}, p_{m,1}, ..., p_{m,n})$$

$$p_{\text{mean}} = (M(p,0), M(p,1), ..., M(p,n)) \quad \text{where} \quad M(p,n) = \frac{\sum_{i \in m} p_{i,n}}{n}$$

In other words, we take the arithmetic mean across each coordinate, taking the mean of the numbers describing dimension 0, 1, etc. up to n.

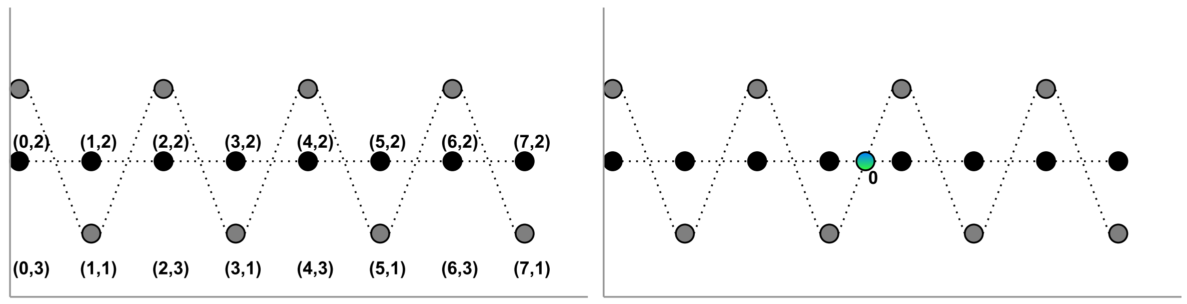
Figure 3.11 demonstrates how this simple method works on two similar curves - one straight line, and one oscillating line transposed below the first. However, consider figure 3.12 which depicts different curves using the same method. The mean-to-mean Euclidean distance determines that the two curves have a distance of zero - in other words, they are the same! However it is clear that while the two curves lie in similar regions of space, they are constructed in different ways, and while they are similar they should not have perfect similarity.

Finally consider a worst-case scenario in 3.13. These two curves have been constructed in such a way that, while their means are the same point, they are always in very different regions of space. This highlights the major shortcoming with this method, in that curves constructed out of points in two distinct regions have a mean very far away from the points themselves.



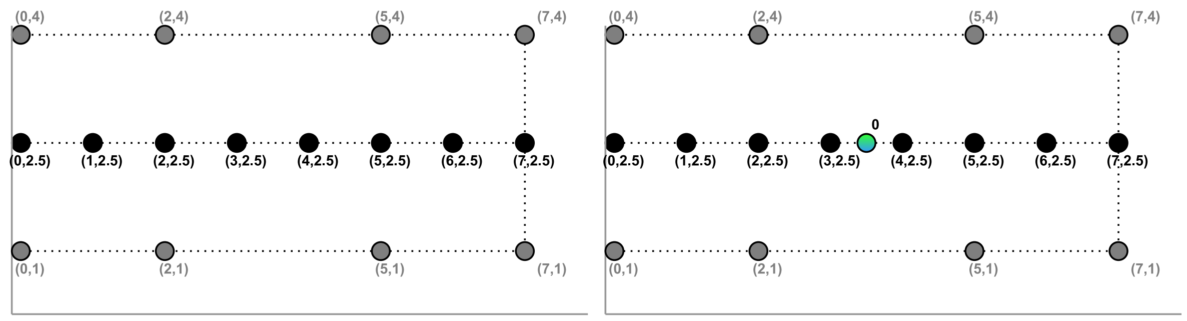
(a) (b)

Figure 3.11: Example of Mean-to-mean Euclidean Distance between two simple curves



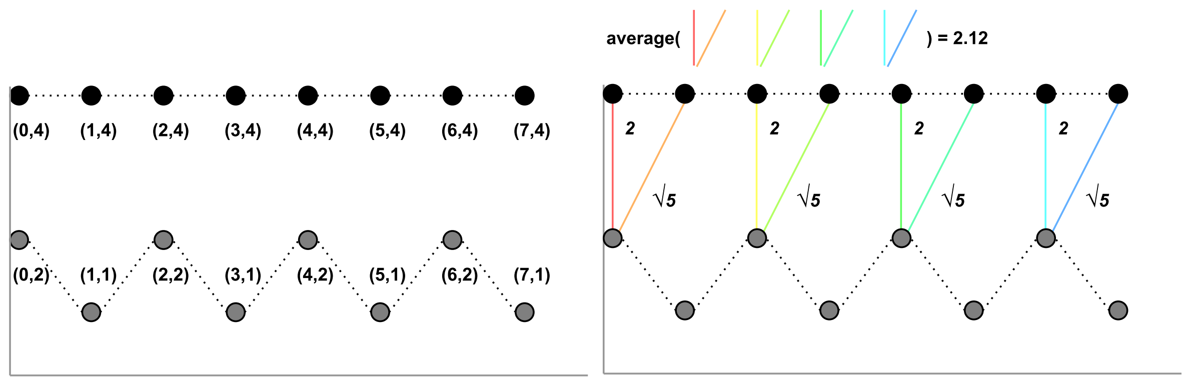
(a) (b)

Figure 3.12: Example of Mean-to-mean Euclidean Distance between two simple curves



(a) (b)

Figure 3.13: Example of Mean-to-mean Euclidean Distance between two simple curves



(a) (b)

Figure 3.14: Example of Minimum Point-to-point Distance between two simple curves

2. Minimum point-to-point distance

The goal of this method is to determine similarity in a way that uses more points than just the means, but that does not rely on ordering - this being so that it could identify tracks with similar segments but that perhaps don't appear in the same order in each track, or to find a track with a repeated segment that is present in an input track. It does this by iterating over each point on the `input` line, finding the distance to the closest point on the `target` line, and then taking the average of these values. This means that order of input matters (it is not necessarily true that $d(p, q) = d(q, p)$), but it should be effective for identifying tracks that the input stays close to, even if some of its points are far away or it only matches up closely with a sub-segment of the target track. The effectiveness of this is discussed in 3.3.2.

Mathematically, this can be expressed as follows:

$$P = [p_0, p_1, ..., p_n] \quad Q = [q_0, q_1, ..., q_n]$$

$$d(P, Q) = \text{average}(\min(D(p_0, q_0), D(p_0, q_1), ..., D(p_0, q_n)), ..., \min(D(p_n, q_0), D(p_n, q_1), ..., D(p_n, q_n))$$

where $D(a, b)$ is the Euclidean distance between the two points, and p_x and q_x are points making up the sequences P and Q. In short, this represents the average of the distances from each point in the sequence to the closest point on the target sequence.

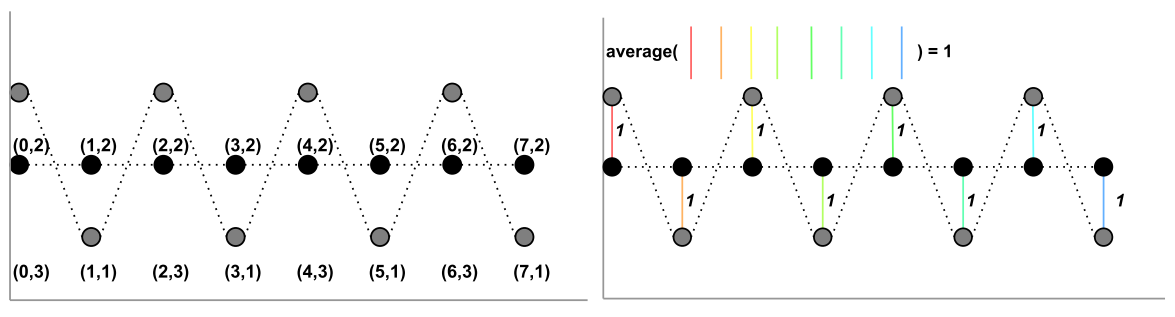
Figure 3.14 shows an example of how this works on the same curves used in 3.11. Notice how this metric defines the target line (grey) to be slightly closer to the input line (black), since there are sections of the target line that lie closer than other sections. The further away points are never considered, which would indicate (in the case of music) that the input line is similar to the closer parts of the target track.

Consider also figure 3.15, which uses the oscillating example given in 3.12. Using just the means of the curves, the distance was calculated to be zero, however with this method the result comes out to be 1. This reflects that, while the curves are indeed similar, even at the closest points they are always separate. As it is not possible to know what exactly is meant by the precise underlying embedding numbers produced by the model, a method that acknowledges the relative closeness of points while still taking into account their definite difference may be useful.

3. Fréchet Distance

While the other two methods are perhaps simple to visualise, neither of them take into account the ordering of the points that make up the line curve. This is where the Fréchet distance comes in - a method which measures similarity by taking into account ordering as well as position. As described by T. Eiter and H. Mannila in their paper "Computing Discrete Fréchet Distance"[18], the measurement can be intuitively described as follows:

> "A man is walking a dog on a leash: the man can move on one curve, the dog on the other; both may vary their speed, but backtracking is not allowed. What is the length of the shortest leash that is sufficient for traversing both curves?"



(a) (b)

Figure 3.15: Example of Minimum Point-to-point Distance between two simple curves

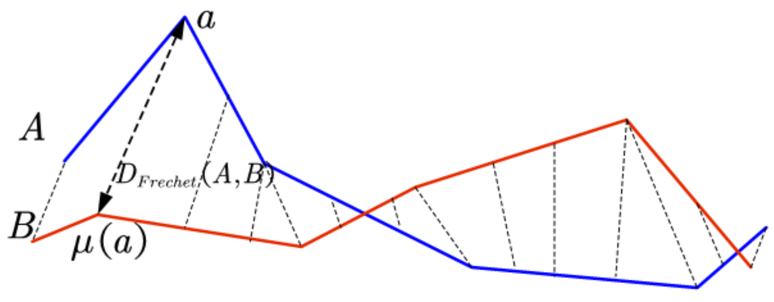
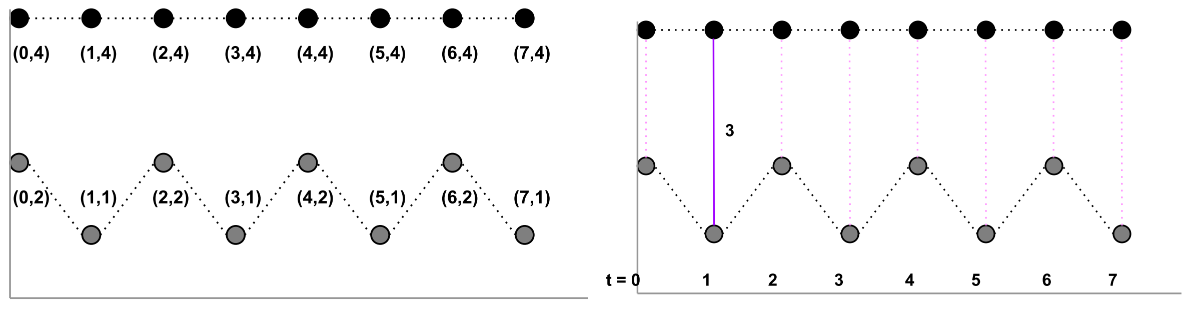


Figure 3.16: Example of how Fréchet Distance is calculated (source: "An Efficient Query Algorithm for Trajectory Similarity Based on Fréchet Distance Threshold" by N. Guo et al.[23])

In other words, it can be thought of as the maximum distance between pairs of points at similar timesteps on each curve. A visual example of what this means is shown in figure 3.16.

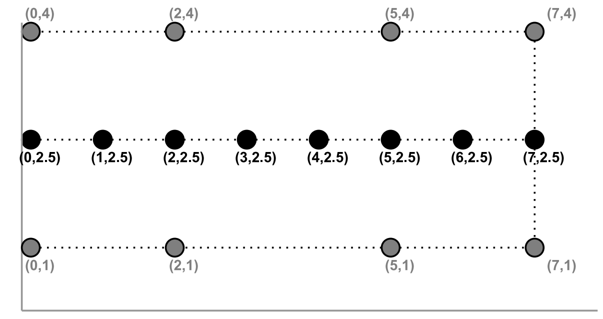
As a result, this distance can be thought of as how similar two songs are when taking ordering, or sequence, into account. For tracks with many distinct sections present in their audio, a low Fréchet distance will indicate that another song has similar sections in a similar ordering. While this may be restrictive for complex tracks, it may end up giving better recommendations in certain cases.

When evaluated on the example introduced in figure 3.11, this gives a worst-case scenario distance of 3, which ends up being the maximum distance between ordered sets of points. Note that if the curves were ordered in reverse, the distance would be much larger. A visual of this is shown in figure 3.17. Note that for Fréchet distance examples, the left-most point is defined as the start point.

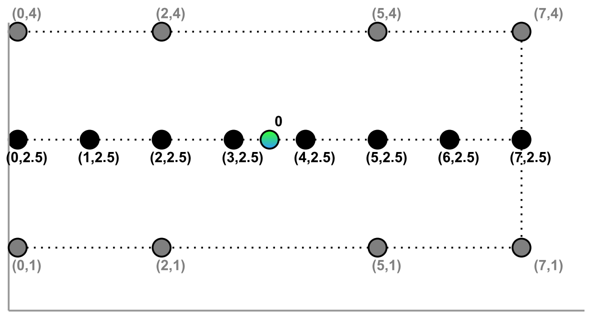


(a) (b)

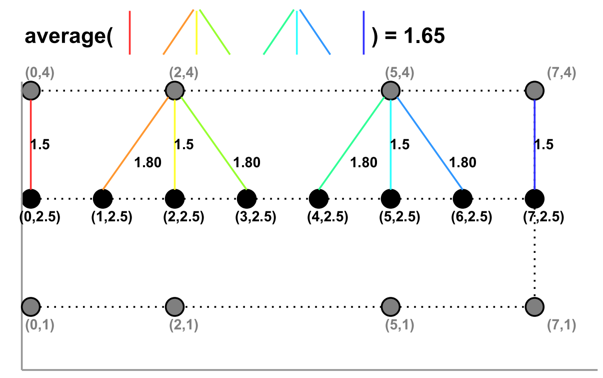
Figure 3.17: Example of Fréchet Distance between two simple curves



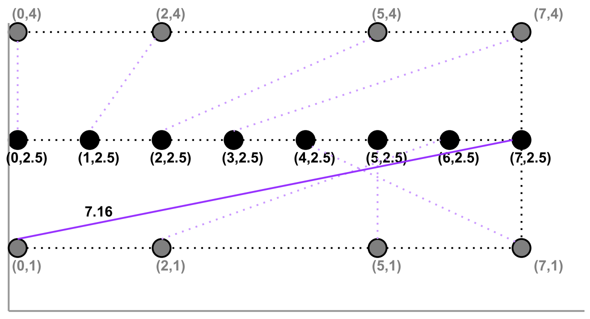
(a)



(b)



(c)



(d)

Figure 3.18: Example of different distance functions on a worst-case scenario curve. (a) Initial (b) Mean-to-mean (c) Minimum point-to-point (d) Fréchet

Figure 3.18 shows a comparison of all four methods on the 'worst-case scenario' curve, where the input curve is straight, and the target curve has two distinct sections on opposite sides of the input. Using the mean-to-mean Euclidean distance, this comes out to be 0, as the two curves have the same mean. Using minimum point-to-point distance, we get a higher value of 1.65, since some parts of the curve are closer than others. However, using the Fréchet distance where ordering matters, we get a much larger distance value of 7.16 for this pair. This is because while the target curve follows a straight line through space, the target curve makes a turn and has its end point close to the input's start point, but far from its own end. This leads to a large maximum distance, indicating that, if these were sequences of music, they would have very different progressions.

NB: It can be difficult to determine which lines are being considered when drawing the Fréchet distance, so some of the intermediate (light purple) lines may be inaccurate - however the longest (dark purple) line is definitely correct.

All visualisations for the three distance functions on three other examples can be found in the appendix A.

3.3.2 Example Recommendations

Below I will demonstrate some examples of recommendations provided by the combination of the trained model and the distance metrics defined above. I will split the analysis of these into two sections - subjective and objective. Subjective analysis will focus on how I personally feel the model did with the recommendations, with there being three success criteria for this - whether the track sounded similar relative to the input, whether I enjoyed the track it recommended, and whether I was already familiar with the track prior to recommendation. The latter is important, as with a dataset size of over 1 million tracks the odds of being recommended by chance a track I am already familiar with are low - so giving the algorithm a track I know and receiving back tracks I know can be considered a success. For the subjective analysis of recommendations, I will be talking about 'style' as well as 'vibe'. These terms are difficult to define objectively, so in this context the 'style' will be the musical structure of the track (similar to the genre, but in some cases more specific - e.g. a slow lo-fi hip-hop track is a different style to a faster, heavier hip-hop track), while ‘Vibe’ refers to the overall sound or mood of the track, for example uplifting, aggressive, etc.

Objective analysis on the other hand will be used to supplement this, looking primarily at the pre-computed audio features (such as energy and acousticness) provided by the dataset and also available through the Spotify API. These features are, as defined by the web API reference docs[2]:

- **Acousticness**, a confidence measure from 0.0 to 1.0 of whether the track is acoustic
- **Danceability**, a measure to describe how suitable a track is for dancing (based on a combination of elements including tempo, rhythm stability, beat strength and overall regularity)
- **Energy**, a perceptual measure representing intensity and activity, contributed to by attributes such as dynamic range, perceived loudness and timbre
- **Instrumentalness**, which predicts whether a track contains vocals (with a value above 0.5 intending to indicate an instrumental)
- **Key**, representing the key of the track as an integer starting from 0 = C, 1 = C# etc.
- **Liveness**, representing the presence of an audience in the recording or a probability the track was performed live
- **Loudness**, the overall loudness of a track in decibels averaged over the entire length (from -60db to 0db)
- **Mode**, indicating the modality of the scale of the track (major = 1, minor = 0)
- **Speechiness**, representing the exclusivity of speech in the recording, with high values representing e.g. podcasts, low values representing music without prominent speech, and medium values representing tracks with layered music and speech such as rap
- **Tempo**, an estimate of the tempo of the track in beats per minute (BPM)
- **Time Signature**, an estimate of the time signature in beats per bar (ranging from 3 = ‘3/4’ to 7 = ‘7/4’)
- **Valence**, a measure of the musical positiveness conveyed by the track, with high values indicating a track sounds more positive (e.g. happy or euphoric), while low values indicate a negative sound (e.g. sad or angry)

From these, the most useful for evaluating recommendations are valence, energy and mode for determining the ‘vibe’ of the track, as well as acousticness, danceability, instrumentalness and tempo for a measure of the ‘style’ of the track. Unfortunately Spotify does not provide track-specific genre information, instead only assigning a genre to artists, which is too broad for this project.

Note that in the online copy of this, clicking on a track ID will redirect to the Spotify page of that track, allowing listening of the music and verification of (or lack of) similarity

Example Recommendation 1: Koven - Gold (Mean-to-Mean Euclidean Distance)

Table 3.3 shows the top 5 most similar tracks to Gold by Koven, according to the mean-to-mean distance metric. This track is electronic (and can be classified further as ‘melodic dubstep’) and features female vocals with a powerful and loud guitar / synth combination in the background, as well as loud rhythmic drums. The Spotify track ID for this is ‘7fKknB0a4fKXCaYl7PHpUO’.

While the first recommendation ended up being a duplicate of the input (due to the input in question having two separate track IDs, both being present in the dataset), the following four recommendations were very good - the best out of any of the tests run (at least out of those within my usual listening sphere). I enjoyed all the suggestions, with three of the four being tracks I was previously familiar with. Two of the four had similar styles (being halftime, melodic dubstep with a vocal layer), while three tracks had similar vibes (being uplifting, bright and having a full sound).

[3]This ended up being a duplicate of the original track, with a different Track ID so it showed up in the recommendations too. The fact the distance was zero however shows that the model works and mapped the same track to the same values.

No.	Track ID	Distance	Similar Vibe?	Similar Style?	Enjoyed?	Prev. Familiar?
1	2sm2f0mbOnfCi1MOLIegdV	0[3]	Duplicate	Duplicate	Duplicate	Duplicate
2	53iWi0aMKYINqGsQxy7y6I	0.39	Yes	No	Yes	No
3	5Kpqr7t4KCI8vrnWrQ4H7B	0.46	Yes	Yes	Yes	Yes
4	2oETRroBvjsgpbKyrRYUZ0	0.60	No	No	Yes	Yes
5	5sPgYCPfIKxXI22NVYNGcT	0.61	Yes	Yes	Yes	Yes

Table 3.3: Table of the top 5 recommendations provided by the mean-to-mean Euclidean distance metric to the input of Gold by Koven

Suggestion 4 was not a similar sounding track by any metric, but was nevertheless one that I enjoyed and that I had heard before. I found this odd, since as mentioned previously, the odds of familiar songs being recommended by chance are very slim. I can't work out why this track was deemed to be similar, however the success of the other three makes this set of recommendations a success.

No.	Valence	Energy	Mode	Acoust.	Dance.	Instrum.	Tempo
Input	0.29	0.779	1	0.0184	0.377	0	150
Rec. No.	Δ Val.	Δ Eng.	Mode	Δ Aco.	Δ Danc.	Δ Inst.	Δ Tempo
2	+0.148	+0.028	0	-0.018	+0.324	+0.028	-22
3	+0.02	+0.013	1	-0.059	+0.097	0	-8
4	-0.273	+0.169	1	-0.018	+0.102	+0.03	-6
5	-0.137	+0.141	1	-0.057	+0.038	0	-5

Table 3.4: Table of difference in audio features of the top 5 recommendations provided by the mean-to-mean Euclidean distance metric to the input of Gold by Koven

Table 3.4 shows the difference between the input features and the audio features of the 4 non-duplicate recommendations. I determined from my subjective analysis that suggestions 2,3 and 5 had similar vibes to the original track, and the feature analysis agrees to come degree - suggestion 3 in particular had the same mode, and was within 0.02 of both the valence and energy values - however suggestions 2 and 5 have different valence values by over 0.13 each, while suggestion 5 itself has a large difference in the energy value. Suggestion 4 was determined to have different vibes as well as style and while the values for valence and energy are indeed the largest difference in the table, it actually has quite similar acousticness, danceability and instrumentalness values. For these sets of values determining style, suggestion 2 has a large difference in danceability, reflected likely by its lower tempo, but the other values remain similar across the board.

Overall, the objective measurements agree to some extent that these songs are similar, however with variances of ± 0.1 being common, it suggests similarity is limited (note that the range for all values except tempo is 0 to 1, so a difference of 0.1 represents 10% of the total range).

Figure 3.19 shows the generated t-SNE plots on the best recommendations. This shows the closeness of the means of the tracks (a), while also showing the lines (b) which help demonstrate why these recommendations would not show up using Fréchet distance (the suggestions from which are shown in `Example Recommendation 2`). This is due to the fact that while the means fall in similar places, there are aspects of the recommended tracks that diverge from the main body, causing there to be a larger maximum distance between the two.

Figure 3.20 shows a comparison of the input spectrograms of the input track and recommendation 2, which was judged to be a very good recommendation. It also shows the corresponding encoding and embedding produced by the model, where it can be seen the similarity between the two despite the noticeable (yet minor) differences in the input spectrograms.

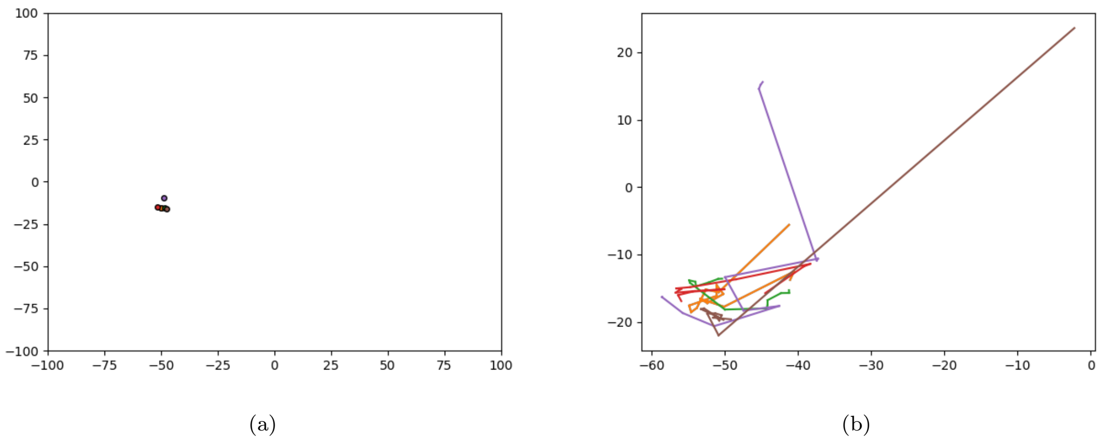


Figure 3.19: TSNE plots for the recommendations of Koven - Gold.
(a) Means and (b) Connected Line Segments (zoomed in)

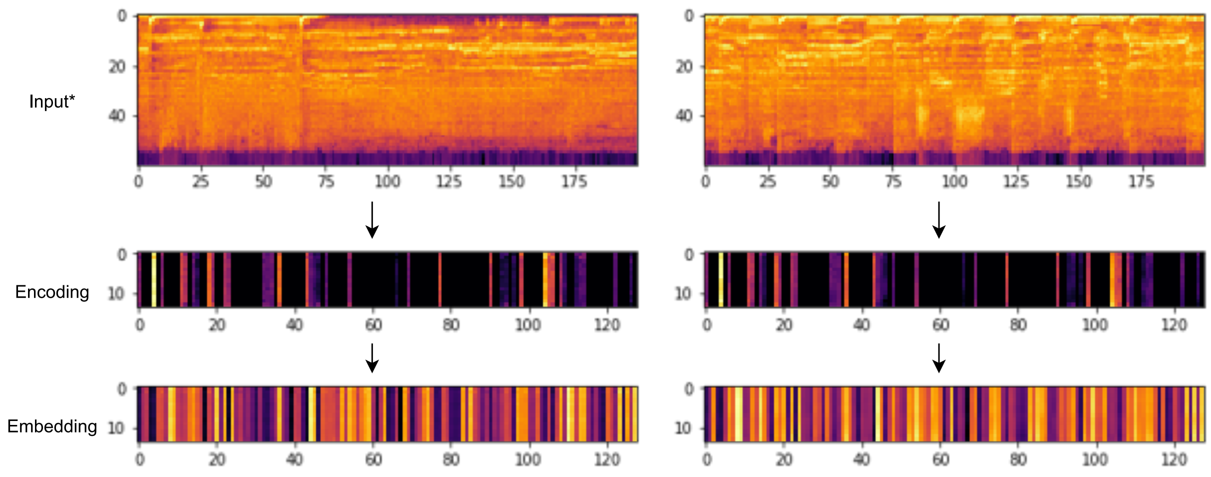


Figure 3.20: Comparison between the input, encoding and embeddings of 'Gold by Koven' and a close recommendation, 'Closer - Daddy's Groove Remix by Tegan and Sara'.
*The input image shown is a 4 second segment of the input spectrogram

Example Recommendation 2: Koven - Gold (Fréchet Distance)

I ran the recommendations on the same track as before, but this time using the Fréchet Distance metric. Table 3.5 shows the result of this:

No.	Track ID	Distance	Similar Style?	Similar Vibe?	Enjoyed?	Prev. Familiar?
1	2sm2f0mbOnfCi1MOLIegdV	0^2	Duplicate	Duplicate	Duplicate	Duplicate
2	3xauOetfCVtLOwPRyYFMNW	1.70	No	No	No	No
3	0ZJyft5ioMEky5zhzvH1nL	1.71	No	No	No	No
4	02hnKrZkgKZkclzMrPwPRw	1.72	Somewhat	Yes	Yes	No
5	0Rnxt0OPm0yinGqOku4Yy1	1.92	No	Yes	Yes	No

Table 3.5: Table of the top 5 recommendations provided by the Fréchet distance metric to the input of Gold by Koven

This set of suggestions was not as successful when using Fréchet distance instead of the mean-to-mean Euclidean distance. The first two non-duplicate suggestions were completely different in both style and vibes, and as a result I did not particularly enjoy them. Suggestion no. 3 in particular was jazz, which is not at all similar to the input! The final two outputs were better however, with both being electronic music and both having bright and full backing synths. Neither were quite the same style as the input, but both were enjoyable.

No.	Valence	Energy	Mode	Acoust.	Dance.	Instrum.	Tempo
Input	0.29	0.779	1	0.0184	0.377	0	150
Rec. No.	Δ Val.	Δ Eng.	Mode	Δ Aco.	Δ Danc.	Δ Inst.	Δ Tempo
2	+0.079	+0.205	1	-0.009	+0.119	0	-20
3	-0.092	+0.183	0	-0.034	+0.324	+0.558	-50
4	+0.179	+0.123	1	+0.068	+0.042	0	-5
5	+0.004	+0.089	0	+0.019	+0.063	+0.022	-50

Table 3.6: Table of difference in audio features of the top 5 recommendations provided by the mean-to-mean Euclidean distance metric to the input of Gold by Koven

Table 3.6 shows the objective analysis of the same tracks, and the style characteristics (the right-most four columns) show how poor the recommendations were - in all but one case they fell at a drastically different tempo, with recommendation 3 having more than a 0.3 difference in three different values (this was the jazz sounding example, so it is clear it did not belong in this recommendation set). The others look slightly better on paper, with them all falling in similar areas in valence, acousticness and danceability, but have relatively large value differences in energy. In all, the objective analysis mostly agreed with my assessment on a number of the recommendations, with suggestions 2 and 3 differing in energy and danceability respectively, while suggestions 4 and 5 were much closer in all, differing by similar amounts to the successful suggestions from example 1.

It is worth noting by this point I realise that mode and tempo are not particularly good metrics for judging differences, but are kept for completeness.

Example Recommendation 3: Lil Nas X - Old Town Road (Mean-to-Mean Euclidean Distance)

For the next example of an input, I decided to use a popular mainstream track in a more popular style, in the chart-topping Old Town Road by Lil Nas X, which falls under the hip-hop / rap genre. I am not as familiar with this genre, however it will be a good test to see how it deals with genres different from the more underground ones I tend to listen to. As such, I will drop the familiarity and enjoyed columns, as I was not familiar with any of the outputs, and am not a fan of the genre enough to provide anything meaningful by commenting on whether I personally enjoyed the songs or not. Note that from here on I will omit duplicates from the table.

No.	Track ID	Distance	Similar Vibe?	Similar Style?
1	7dAYILmG0kJ8NOMe2xrj0k	0.65	No	Yes
2	5XW9c36lVlwwihn4u2oB00	0.68	Somewhat	Somewhat
3	0OZaDXtYLruqpVCzHy6XLn	0.79	Somewhat	Somewhat
4	1kXvMvk1o0S3qkNsalHPgj	0.82	Yes	Yes
5	3kFpse0h5ovW1t2XXCyXhZ	0.86	No	Yes

Table 3.7: Table of the top 5 recommendations provided by the mean-to-mean Euclidean distance metric to the input of Old Town Road by Lil Nas X

It was difficult for me to judge these as I am not familiar particularly with the genres of hip-hop and rap, however I am familiar enough to tell that all the returned tracks fell in that broad category, which was a partial success. The particular style each fell under was harder to determine, as the input track is a country / rap hybrid while all the recommendations were either pure hip-hop or had some different influence. No. 5 fell the closest to the original track in terms of style, being a similar tempo with a similar drum pattern as well as having singing over the top. The rest primarily had more monotone vocals typical of rap. In terms of vibe, suggestion no. 4 came closest, again being slow but with a similar minimal background. As a side note, this fourth suggestion is a Russian rap song, which shows how my program does not take lyrics or language into account, which makes for suggestions that could be more likely to lead users away from their usual area of familiarity, while still retaining the same instrumental sound.

No.	Valence	Energy	Mode	Acoust.	Dance.	Instrum.	Tempo
Input	0.507	0.53	1	0.058	0.607	0	136
Rec. No.	Δ Val.	Δ Eng.	Mode	Δ Aco.	Δ Danc.	Δ Inst.	Δ Tempo
1	+0.094	+0.047	1	-0.021	+0.213	0	+9
2	-0.212	+0.106	0	-0.048	+0.225	0	0
3	-0.332	+0.024	1	-0.051	+0.074	+0.553	+29
4	-0.100	+0.107	1	-0.042	+0.07	+0.109	+24
5	-0.305	+0.052	1	-0.014	+0.189	+0.008	-4

Table 3.8: Table of difference in audio features of the top 5 recommendations provided by the mean-to-mean Euclidean distance metric to the input of Gold by Koven

Table 3.8 shows the objective analysis of the same five suggestions, with the overall takeaway being that these metrics do not properly describe the the data in the same way a subjective listening does. All five suggestions provided by the program were of the same basic genre and, at least to me who doesn't listen to hip-hop much, they all had very similar styles, even if the vibes of each were different. However the features obtained seem to suggest that each has quite different valence and, in some cases, danceability and instrumentalness numbers. In fact, the only suggestion I noted as having a definitely different vibe was suggestion 1, which had the closest valence and energy numbers to the original.

My takeaway from comparing the objective feature analysis numbers and my subjective thoughts is that they seem to be measuring different things, and that it is difficult to find pre-computed features that match up to what the human ear can detect in terms of similarities. If I were to extend this project, it would be useful to research a more comprehensive list of features and how humans tend to perceive them, as well as conducting studies where participants listen to the recommendations of their favourite tracks and give their opinions, rather than me just relying on my own.

Further Recommendation Tests

Below are a short set of extra recommendations I attempted that did not warrant a full track-by-track write-up, but that nonetheless provided interesting results.

Koven - Gold (Minimum point-to-point distance)

Table 3.9 shows the top 5 most similar tracks to Gold by Koven, this time using the minimum point-to-point distance metric. This distance type was implemented last and as such these recommendations were acquired after the other two sets, however these turned out to arguably be the best set for this input - or at least they fared very well. One track, recommendation no. 2, was actually present in table 3.3 for the mean-to-mean Euclidean distance calculation. This indicates that both tracks remain similar throughout and have similar sounds, which can be verified to be the case when listening. In fact, I enjoyed all five best recommendations using this method, having heard all but one before, indicating it successfully kept the recommendations within my sphere of listening.

No.	Track ID	Distance	Similar Vibe?	Similar Style?	Enjoyed?	Prev. Familiar?
1	4XhM8nMNf9x5nC20zGY4bQ	0.66	No	No	Yes	No
2	5Kpqr7t4KCI8vrnWrQ4H7B	0.66	Yes	Yes	Yes	Yes
3	5sPgYCPfIKxXI22NVYNGcT	0.72	Yes	Somewhat	Yes	Yes
4	4mBjcSHnGpmG6n3IQAMCQF	0.73	Somewhat	No	Yes	Yes
5	08ZWBOjJbcXRlXd8pvpJUp[4]	0.78	No	Yes	Yes	Yes

Table 3.9: Table of the top 5 recommendations provided by the mean-to-mean Euclidean distance metric to the input of Gold by Koven

Not all tracks retained the same style and vibe, with no. 1 and 4 being much more bubbly sounding than the input and in different styles, but recommendation no. 2 and 3 got much closer, with no. 2 nailing the sound of the input - which was the one suggested by mean-to-mean as well.

[4]This ID corresponds to a musical podcast, however the track used in the preview clip had ID '5KrQbdqvo-jQAp2wxIO1dtQ'

For this particular input, this distance function worked well - but when looking at recommendations of other tracks not talked in depth about here, it seems to (at least for the closest few), play safe with predictions. In general, I noticed a trend over some other recommendation attempts with different tracks using this method - almost all of the tracks fall within the correct rough genre, occasionally with a similar mood, but rarely getting the style exactly right. All in all, these are not always great recommendations, and for this reason I mostly turned my attention to the other two methods for examples to talk about.

Pop / Drum and Bass : Fox Stevenson - Dreamland
Track ID: 3Vr6lUX3ni1Z6OLvn3YkU6

The input for this track is a poppy drum and bass track, a genre which is characterised by a fast drum pattern and high energy, with this track in particular having quite an upbeat feel to it. However the recommender seemed to struggle with this input, likely due to the high tempo not leading to enough detail being retained from phase 1. While I didn't test it here, it would be interesting to see if this track would receive better recommendations with a higher resolution encoding.

Using mean-to-mean Euclidean distance, the recommendations were poor. The first two non-duplicate recommendations were rock and hardcore in turn - the rock track echoing the input's 'fullness' but being a completely different style and vibe, while the hardcore track picked up on the fast tempo but was otherwise dissimilar. The third recommendation was a strange one, being an old dubstep track by Skrillex - a very different style but a track that I have heard of before and that I enjoy. The final non-duplicate recommendation eventually got it right, being another drum and bass track, albeit a much more high energy one than the input.

Fréchet distance had similar issues, recommending three more rock tracks and an aggressive disco-house hybrid. Minimum point-to-point distance however fared the best, being the only method to keep all five best recommendations within the sphere of electronic music. It featured a house track with a different style but somewhat similar upbeat mood, as well as two more slower but still bright electronic tracks of varying style. Recommendation four was a trap track that once again I am familiar with and enjoy - with the odds of this being a complete accident very low. The fifth suggestion was a drum and bass track, with a very similar style and feel to the input, indicating by far the best recommendation for this particular track.

Overall this example shows clearly that my model struggles with fast tempo tracks, especially since in this case the segment of audio contained a chorus with very little in the way of dynamics, which it seems my model also struggles with. Nevertheless, it was possible to find some similar sounding tracks via the used of minimum point-to-point distance, as well as a selection of other tracks that were different in style but that I am familiar with and enjoy regardless.

Ambient / Acoustic Music : Fire Mist - "Forest in the Morning"
Track ID: 2CnsUd3G5mKpFPXw8fur39

I assumed that ambient music would be easy for the recommender to deal with, as the subdued and often repetitive nature of it would make it easy to identify similar tracks - after all, in the paper by D. Kowald et al.[24], they state that among fans of "beyond-underground" music, recommender systems work best for fans of ambient. However, I did not initially find this to be the case - whether the dataset did not contain much ambient music of the type discussed by Kowald, or whether it just could not find the examples with the training I did, many suggestions it gave when provided with ambient or acoustic music ended up being either classical or in some cases even spoken word. It seemed that, despite the spectrograms being normalised during training, the model picked up on volume more than expected. When using mean-to-mean Euclidean distance, most of the suggestions were quiet classical tracks, with only one other acoustic track (a folk style song), and a spoken word speech recommended too.

Things got much stranger when using the Fréchet distance metric however - despite the input track being mainly acoustic guitars and not much else, the first *three* most similar tracks according to this metric were German opera tracks! These were even all from different composers, eliminating the possibility that it simply mapped well to a particular album or set of opera songs. To add to the strangeness, in the top 10 most 'similar' tracks there was yet another German track, this time a vocal dominated track with fast piano in the background. The remaining recommendations were more opera / classical tracks, however using the Fréchet distance, one or two tracks could be defined as ambient tracks.

My takeaway from this is that ambient and classical fall very close together on the spectrum, and my model has difficulty differentiating between the two.

Lo-Fi / Ambient : Alex Cortiz - "Float on Baby"
Track ID: 1iCBTuVvATjayKMULrY8Jd

In a second attempt to test my program on more ambient music, I tried this track by Alex Cortiz which came from the album "Lo-Fi Explorations". This track had much more loudness to it than the previous test for ambient music, and the recommendations provided echoed this. While these were not perfect, some of the suggested tracks could be classed as lo-fi or ambient, with the top 5 recommendations all having washed out backing ambience, with many either having slow vocals or a rhythmic drum pattern, the latter of which is present in the input track. This test worked a lot better, suggesting that perhaps the magnitude of the sounds in the input song help differentiate better what should be returned.

Ambient : ØYSTEIN SEVÅG - "Motion"
Track ID: 4ONB9uMrnHLnMjUJURfUw8

The final test of the three I conducted on ambient music was on a track which perhaps much more accurately fits the style of pure ambience - the track is a single, evolving texture. The spectrogram for this can be seen in figure 3.21, showing the consistency of the sound throughout. Finally, this fit what I expect many people mean by ambient music, and the recommendations provided mirror this. Three of the five closest tracks contained similar evolving textures, and the other two were pure piano pieces with soft backing ambience and reverb. When provided with pure ambient inputs like this, the model works well and is able to easily identify similar tracks when using mean-to-mean Euclidean distance. When using Fréchet distance, similar results are achieved, with most of the resulting tracks being ambient textures too. However, this method for some reason seemed to return ambient music with a much darker tone to it - perhaps picking up on the rumbling bass note in the input, which Euclidean mean distance did not. In any case, this input was especially easy to generate recommendations for.

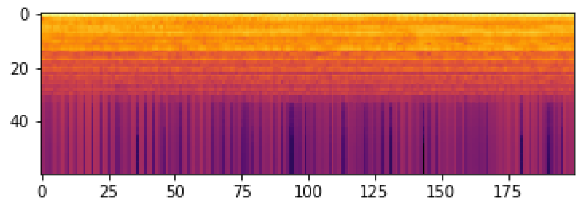


Figure 3.21: Spectrogram for "Motion by ØYSTEIN SEVÅG", an ambient track

Chapter 4

Critical Evaluation

4.1 Summary of Results

Overall, I consider this project as a success. While recommendations were not perfect, and there are plenty of areas to improve, the fact that with such a simple architecture and use of triplet loss there were inputs for which recommendations worked well is impressive. Suggestions were particularly close in terms of musical content when provided with slowly evolving ambient tracks, electronic music with a constant backing texture, or with more mainstream genres like hip-hop. The model however had limited success when presented with faster genres such as drum and bass, with only one of the provided recommendations being in the same style.

The failure cases were however still interesting - even in cases where the recommendations were not of a similar style, in some scenarios these tracks still came from a similar listening sphere. When provided with the drum and bass track mentioned (analysis in 3.3.2), while only one result was of the same genre, two further outputs were songs I was previously familiar with and enjoyed. This shows that despite having difficulty with some types of music, the underlying embedding still in some cases classifies similar types of music together, leading to these different yet enjoyable tracks in recommendations.

4.2 Future Work

There are plenty of opportunities for future extensions of this project driven by various shortcomings and areas to investigate. Perhaps the biggest improvement however could come from conducting a study to gather other people's opinions on the recommendations. This was not possible for this project, as the time taken to acquire the data left little time left for planning surveys. It is however impossible to ignore how the inclusion of a wider variety of listener's opinions would enhance this project greatly.

Beyond this, there are a number of choices I could have made regarding the architecture and building of the project. One such set of shortcoming were limitations of the dataset and the methods used to process it. The first of these is the size of the dataset - while over 1 million tracks is a large number, Spotify themselves claim they have a catalogue of over 82 million tracks[11], meaning that my recommendations can only cover a little beyond 1% of Spotify's library. This means that, while in theory this system could be used to recommend tracks without bias to popularity, it already has no access to a vast majority of Spotify's catalogue, meaning many tracks have no chance at all to be suggested. Ideally, the product would be integrated as part of an entire site [1], however at this early stage that is not possible. More recent tracks are also missing from the dataset, which in a finished product could be provided with regular model updates, however again this is difficult to achieve at this stage.

Furthermore, there is the clear drawback of working only with pre-defined 30-second long preview clips. While having access to any audio at all is great, there is no guarantee these clips are representative of the tracks they come from, especially for long pieces or songs with varying sections throughout. For example, two songs may be found to be similar using this model, but it may be comparing track A's intro to track B's chorus, which could mislead users. If entire tracks were accessible, it would be possible to trim them down into smaller parts to be processed while still treating them as belonging to the same track.

[1]With other services such as SoundCloud being attractive, as they allow any user to upload music, furthering the idea that this project could help users discover new music from undiscovered artists

There is also room for experimentation with different methods of representing the music, both within the model and with the inputs.

Architectural changes represent the region with the most scope for experimentation. While in this project I simply used a basic autoencoder with fully connected layers (due to its surprising effectiveness on paper of recreating spectrograms), there are many more more sophisticated methods for approaching topics like this. One such method is the use of convolutional layers as in a Convolutional Neural Network (CNN). These apply a filter over an input image (of which a spectrogram effectively is), and can be used to more explicitly gather local features versus relying on fully connected layers to figure this out. This would play to the strengths of spectrograms more, and could eliminate the blurry nature of the reconstructions from phase 1.

There is also the possibility of looking into more complex and explicit feature extraction techniques. While my idea was to not influence the model much in what kinds of features it picked up on for reconstruction or similarity, based on results it looks like it would be worth exploring some of these. For example, the idea of harmonic-percussive separation[29] sounds appealing as it applies directly to music and, while it does involve some manual identification of potential features, harmonic and percussive elements are present in almost all music.

There is scope for improvement too within the realm of evaluation and the generating of recommendations. While I have proposed three methods of similarity between embeddings, there are many others to explore. One such example is Dynamic Time Warping, a method used to "dynamically compare time series data when the time indices between comparison data points do not sync up perfectly"[27]. This has applications in areas such as speech recognition, and as such may be useful for working with musical audio too. It is also possible better results could be obtained by weighting different methods and combining them, as it seems the functions proposed have their own utility, and may each be useful in different ways.

Finally, while not necessarily within the scope of what has been talked about in this project, the recommender program itself has its flaws. For starters, it runs extremely slow;y, taking 10 minutes to generate a batch of suggestions with mean-to-mean Euclidean distance, and almost 20 minutes using the other two. This could be improved either by multithreading the track-to-track comparisons, or by pre-computing and storing some (or all) combinations in a lookup table. These would make the program much more appealing to a user, however usability was not the focus of this project.

Finally, one featured I was sadly not able to implement was the ability to input *any* Spotify track to receive recommendations for, even if it was not present in the initial dataset. This is simple in principle, and I did implement it to the best of my ability, however I encountered a strange bug where the acquired preview URL no longer contained a track of length 30 seconds (instead being slightly shorter). I could not figure out why this happened, but it seemed to affect older data too if the preview clips were re-acquired. This may be due to a change in the API, but whatever the reason is it sadly prevented me from implementing this feature effectively. It is however feasible given the architecture I have provided, and given a valid preview clip there is nothing stopping this from being possible, in which case it would surpass the abilities of many websites that offer similar services.

Chapter 5

Conclusion

To summarise, I have created and trained a two phase model to measure song similarity. Phase 1, responsible for creating efficient encodings of input spectrograms, achieves a recreation accuracy of 94-95%, depending on the encoding size used. Phase 2 learns similarity between these encodings via the use of triplet margin loss, and the embedding produced by this phase is the basis for which recommendations can be generated. Through the use of one of three distance measurement functions aimed to measure how close pairs of curves are in 128-dimensional space, recommendations can be generated, which vary in quality depending on the style of music provided and the distance function chosen.

For example, with recommendations provided using mean-to-mean Euclidean distance upon an input of melodic dubstep, the recommendations perform well in a subjective listening test, with three of the four closest non-duplicate matches retaining some degree of similarity to the input (in either vibe or style), while the remaining suggestion is nonetheless enjoyable and a track I was previously familiar with. This shows that while in cases the recommender fails to perfectly recommend similar *sounding* tracks, it does in other cases offer tracks within the vague genre sphere of the original, providing enjoyable songs outside the style.

Use of other distance functions gives inconsistent results however, with the Fréchet distance function excessively prioritising perceived structure over sound. The style of the input track affects things too, with simpler styles such as ambient or more mainstream styles such as hip-hop providing more 'similar' suggestions than more underground genres such as dubstep and drum and bass, however for these inputs some similarities can still be found.

There are a number of possibilities to further this work. One of these could be to experiment further with network architecture - while one of the main benefits of the use of spectrograms as network inputs is the possibility of treating them as images, I used no convolutional layers in the models. This could be explored, in order to modify feature extraction to better preserve local details, or to learn similarity across sequences of encodings rather than simply with adjacent ones.

Another possibility could be to blend the use of the objective measurements discussed in this project (such as the distance functions and phase 2 learning) with pre-computed feature sets obtained by third parties, such as the feature analysis provided by Spotify. While in my analysis of example recommendations, the feature analysis did not tend to agree with my subjective opinions, certain metrics (such as tempo, valence and modality) could be combined in order to generate more informed recommendations. Further distance functions and measures of similarity between curves could be explored in order to better measure the similarity between sequences of embeddings, to find a compromise between pure distance and ordering of points. It could be explored whether certain styles of music are better suited to certain ways of comparing distance, or benefit from a weighted combination of methods.

Finally, a clear improvement to this project would be to conduct studies where participants evaluate recommendations provided by this project both on their preferred music tastes as well as on songs outside their listening habits. This would enable a more diverse range of subjective opinions, with feedback from users with different music tastes to my own. This would enhance the effectiveness of this project greatly, but was not possible here due to time.

To conclude, the technical side of the project has mostly been a success - I have successfully trained a two phase model upon a dataset of 1 million Spotify songs, and used the output to generate recommendations that while inconsistent, are in certain cases both subjectively and objectively similar. The recommendations were not convincing for inputs of faster tempo music, likely hindered by the blurry reconstruction in phase 1, indicating further areas of improvement to explore.

Bibliography

[1] Concepts. https://ml-cheatsheet.readthedocs.io/en/latest/nn_concepts.html#neuron.

[2] Get track's audio features — web api reference: Spotify for developers. https://developer.spotify.com/documentation/web-api/reference/#/operations/get-audio-features.

[3] Gradient descent (article). https://www.khanacademy.org/math/multivariable-calculus/applications-of-multivariable-derivatives/optimizing-multivariable-functions/a/what-is-gradient-descent.

[4] Key amp; bpm database and music finder. https://tunebat.com/.

[5] Songs like x - get the playlist of your dreams based on a song. https://songslikex.com/.

[6] Sound quality — tidal. https://tidal.com/sound-quality.

[7] Web api — spotify for developers. https://developer.spotify.com/documentation/web-api/.

[8] Welcome! million song dataset. http://millionsongdataset.com/.

[9] Welcome to spotipy! https://spotipy.readthedocs.io/en/2.19.0/.

[10] New study finds that music recommendation algorithms don't work so well for fans of hip-hop, but do great for ambient music. https://old.reddit.com/r/hiphopheads/comments/mgddrl/new_study_finds_that_music_recommendation/gst2bw0/, Mar 2021.

[11] About spotify. https://newsroom.spotify.com/company-info/, Apr 2022.

[12] Audio quality - spotify. https://support.spotify.com/us/article/audio-quality/, Apr 2022.

[13] Difference between batch gradient descent and stochastic gradient descent. https://www.geeksforgeeks.org/difference-between-batch-gradient-descent-and-stochastic-gradient-descent/, Feb 2022.

[14] Abien Fred Agarap. Implementing an autoencoder in pytorch. https://medium.com/pytorch/implementing-an-autoencoder-in-pytorch-19baa22647d1, Oct 2020.

[15] Clark Boyd. How spotify recommends your new favorite artist. https://towardsdatascience.com/how-spotify-recommends-your-new-favorite-artist-8c1850512af0, Nov 2019.

[16] Joseph Cleveland, Derek Cheng, Michael Zhou, Thorsten Joachims, and Douglas Turnbull. Content-based music similarity with triplet networks. *CoRR*, abs/2008.04938, 2020.

[17] Kelum Edirisinghe. Reinforcement learning for personalized recommendations, 05 2020.

[18] Thomas Eiter and Heikki Mannila. Computing discrete fréchet distance . 1994.

[19] Rodolfo Figueroa. Spotify 1.2m+ songs. https://www.kaggle.com/datasets/rodolfofigueroa/spotify-12m-songs, Dec 2020.

[20] Daniel Fleder and Kartik Hosanagar. Blockbuster culture's next rise or fall: The impact of recommender systems on sales diversity. *Management Science*, 55(5):697–712, Mar 2009.

[21] Dalya Gartzman. Getting to know the mel spectrogram. https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0, May 2020.

[22] Google. Collaborative filtering. https://developers.google.com/machine-learning/recommendation/collaborative/basics, Feb 2021.

[23] Ning Guo, Mengyu Ma, Wei Xiong, Luo Chen, and Ning Jing. An efficient query algorithm for trajectory similarity based on fréchet distance threshold. *ISPRS International Journal of Geo-Information*, 6:326, 10 2017.

[24] Dominik Kowald, Peter Muellner, Eva Zangerle, Christine Bauer, Markus Schedl, and Elisabeth Lex. Support the underground: Characteristics of beyond-mainstream music listeners. *EPJ Data Science*, 10(1), 2021.

[25] Vijaysinh Lendave. Cold-start problem in recommender systems and its mitigation techniques. https://analyticsindiamag.com/cold-start-problem-in-recommender-systems-and-its-mitigation-techniques/, Sep 2021.

[26] Zahra Noshad, Asgarali Bouyer, and Mohammad Noshad. Mutual information-based recommender system using autoencoder. *Applied Soft Computing*, 109:107547, 2021.

[27] Ricardo Portilla, Ricardo Portilla, Brenner Heintz, and Denny Lee. Understanding dynamic time warping - the databricks blog. https://databricks.com/blog/2019/04/30/understanding-dynamic-time-warping.html, Jan 2022.

[28] Brad Ross and Prasanna Ramakrishnan. song 2 vec : Determining song similarity using deep unsupervised learning. 2017.

[29] Yuma Sakashita. Detection and classification of acoustic scenes and events 2018 challenge acoustic scene classification by ensemble of spectrograms based on adaptive temporal divisions technical report. 2018.

[30] Yusuf Sarigoz. Triplet loss - advanced intro. https://towardsdatascience.com/triplet-loss-advanced-intro-49a07b7d8905, Mar 2022.

[31] Laurens van der Maaten and Geoffrey E. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[32] Thomas Vassallo. Calculating audio song similarity using siamese neural networks. https://towardsdatascience.com/calculating-audio-song-similarity-using-siamese-neural-networks-62730e8f3e3d, Aug 2020.

Appendix A

Distance Function Visualisations

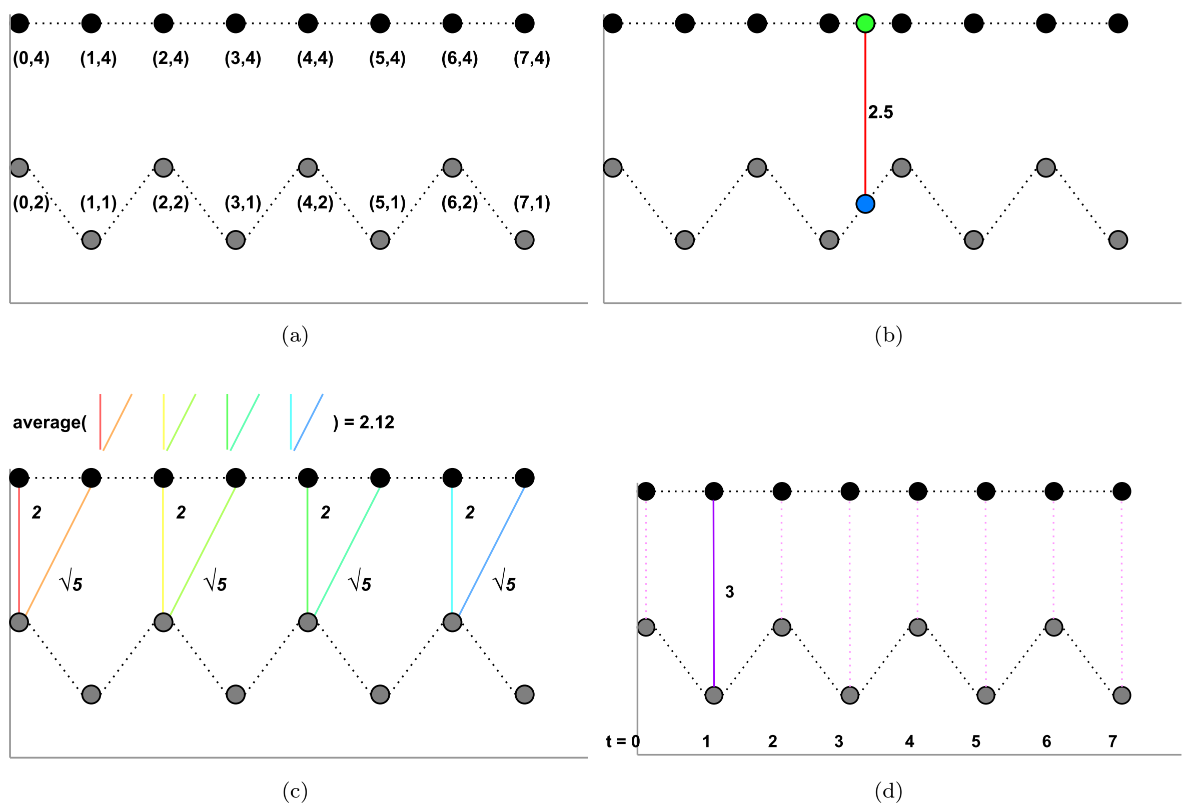


Figure A.1: Example of different distance functions on Example 1.
(a) Initial (b) Mean-to-mean (c) Minimum point-to-point (d) Fréchet

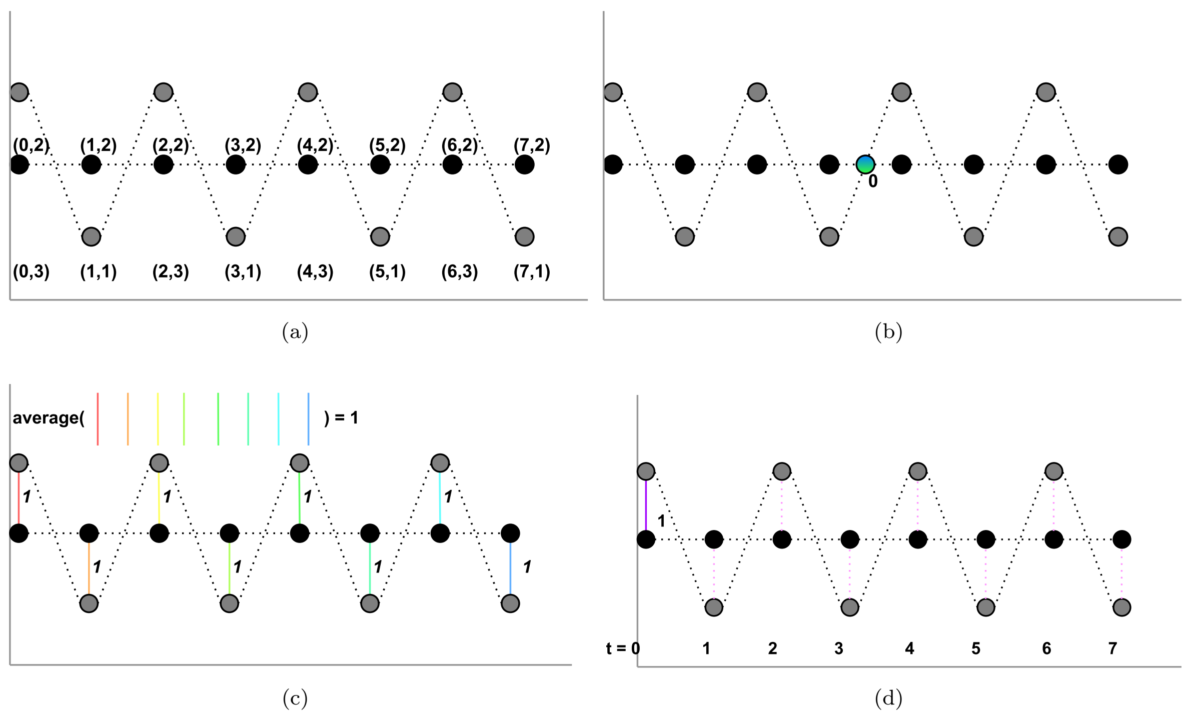


Figure A.2: Example of different distance functions on Example 2.
(a) Initial (b) Mean-to-mean (c) Minimum point-to-point (d) Fréchet

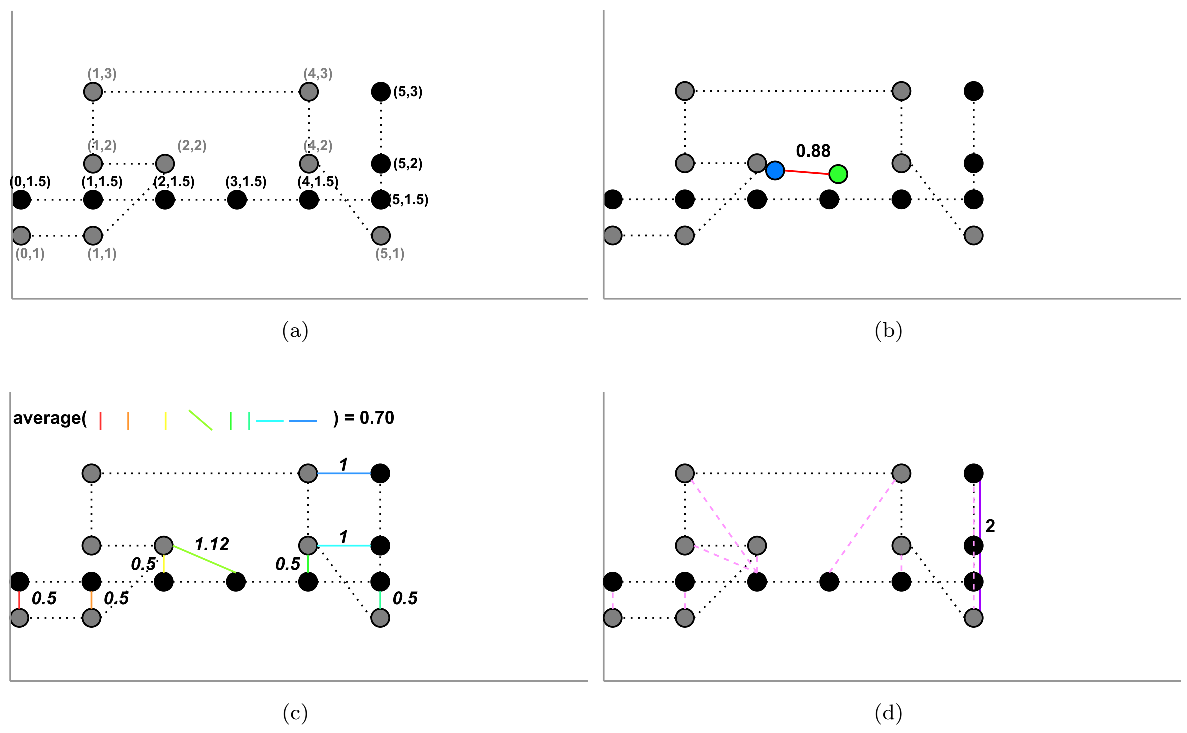


Figure A.3: Example of different distance functions on Example 3.
(a) Initial (b) Mean-to-mean (c) Minimum point-to-point (d) Fréchet

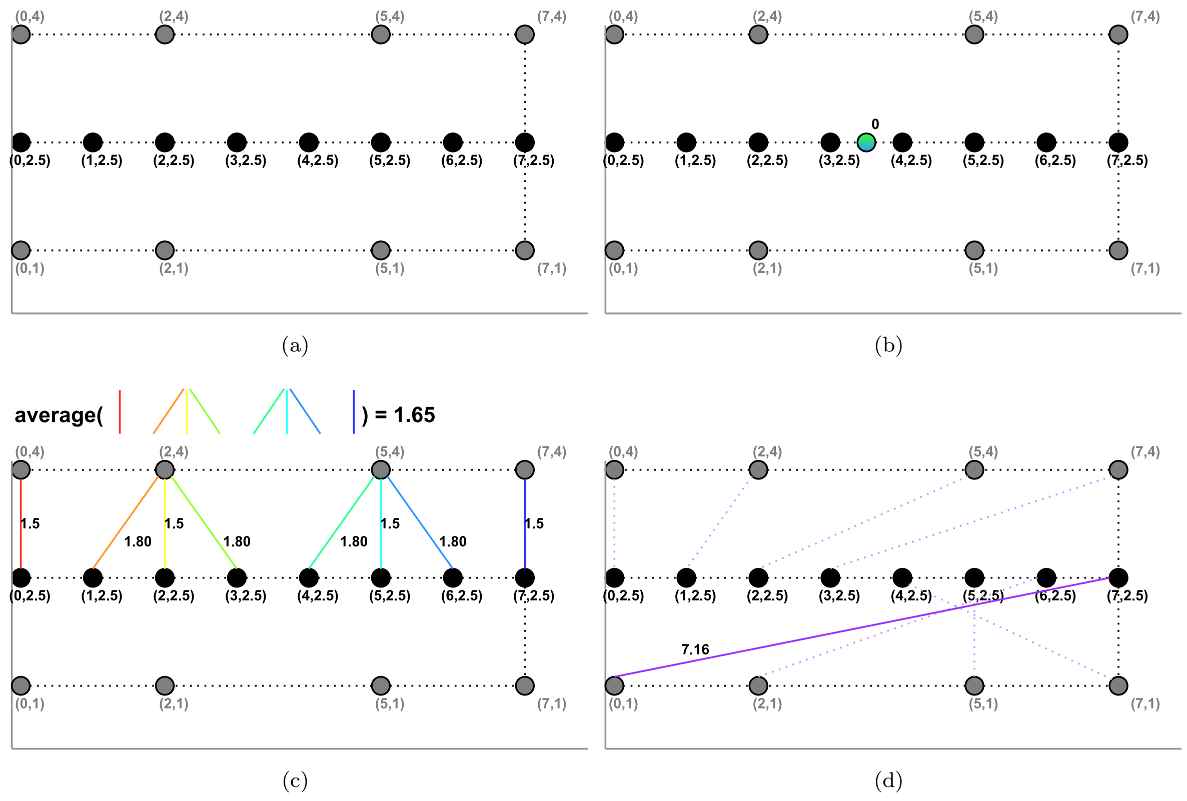


Figure A.4: Example of different distance functions on Example 4.
(a) Initial (b) Mean-to-mean (c) Minimum point-to-point (d) Fréchet