Automated Music Recommendations Using Similarity Learning

Jamie Burns[1] and Terence L van Zyl[2]

[1] School of Computer Science and Applied Mathematics, University of the Witwatersrand, South Africa `1628649@students.wits.ac.za`
[2] Institute for Intelligent Systems, University of Johannesburg, South Africa `tvanzyl@uj.ac.za`

Abstract. Automatic music recommendation is a machine learning problem in which we try to classify music based on the taste of an individual. Due to a continuous shift towards online music store and streaming services, automatic music recommendation remains an increasingly relevant problem. Many modern music recommendation systems make use of collaborative filtering. However, this method fails when there is a lack of historical data available and is ineffective in recommending unfamiliar or unpopular music. In this paper, we explore the use of a latent factor model for music recommendation. We attempt predicting the latent factors that best describe the sound of a song. We compare the use of traditional music features, Mel-frequency cepstral coefficients (MFCCs) with a Siamese recurrent neural network paired with similarity learning. We compare predictions qualitatively on the Free Music Archive (FMA) dataset. We show that not only does the predicted latent factors produce a sensible recommendation but also the use of deep similarity learning is a well-suited method for music recommendation. Our Siamese recurrent neural network significantly outperforms the traditional approach leading us to motivate for further investigation on larger datasets.

1 Introduction

With the increase in the size and usage of digital music collections, the need for more advanced storage mediums has never been more prevalent. An effective music storage medium has other capabilities beyond efficient song retrieval [27]. One such capability is an automated music recommendation.

Automatic music recommendation is a common machine learning problem that classifies music based on the taste of an individual. These systems allow users to discover music, while also assisting online music stores target audiences for their wares. The variety of music styles and genres, as well as social and geographic factors, that can affect a listeners preference, makes music recommendation a challenging task. When considering individual songs, a variety of items can be recommended. This leads to various approaches including album based, and artist based recommendations. Although these approaches are promising and relatively simple to implement, they also limit artist exposure and prevent users from discovering new music. Thus, these approaches may not always be viable solutions to the problem at hand [20].

Music information retrieval (MIR) plays an important role in music recommendatio n[2]. A common approach to MIR is to make use of textual-metadata (which is used to describe the music item of interest). Due to the continuous growth of digital music collections, these have become difficult to maintain. These challenges lead to a proposal for automated systems for MIR [2].

Currently, there are two dominant architectures for MIR, deep and shallow architectures [9]. Shallow architectures transform a time-variant function, a signal into an instantaneous representation, a feature, to extract descriptive features from audio clips. Architectures are defined as shallow if they rely on a single transform to marginalise the temporal dimension of music audio. Deep architectures consist of multiple stacked shallow architectures, each as their own layer. The use of numerous layers allows us to absorb more specific variations in signals, which is difficult to do directly (using shallow architectures) [9]. These deep architectures can be thought of as latent factor models. This is because they can take a high-dimensional feature representation of an audio clip and transform into an unknown lower-dimensional representation, which can be modelled more accurately [23].

Most shallow architecture approaches make use of MFCCs (Mel-frequency Cepstral coefficients), which are short-time spectral decompositions of an audio signal that represents general audio frequency attributes that are imperative to human hearing [16].

Similarity learning is a machine learning technique that is directly related to regression and classification [11]. The goal of similarity learning is, through the use of some similarity function, to learn from examples how best to represent the similarity or dissimilarity between two objects (in our case, songs) [27]. Due to increased data and the need for advanced information retrieval systems, similarity learning has seen increased interest from the scientific community [29] and is a viable technique for MIR.

The subjective nature of music similarity (i.e. opinions on the similarity between songs are not consistent) makes it difficult to collect large amounts of labelled data [2]. The lack of labelled data limits the performance of supervised learning techniques, which are reliant on a large amount of labelled data to make accurate predictions [14]. This provides a potential advantage to unsupervised learning techniques, such as clustering, that do not need the data to be labelled [14]. Further, classification requires a label per class. As a result to transfer to unseen classes a new model will need to be trained. This leads one to consider similarity learning in which embedding can be transferred to unseen classes without requiring retraining [12]. Similarity learning has seen much attention in music and audio information retrieval and has been used in a variety of different audio classification problems. [17] explored the use of similarity learning to rank the similarity between different YouTube audio clips. Although the research focused on environmental sounds, many of the proposed techniques apply to music audio. [13] proposed the use of similarity learning, along with a deep convolutional neural network for genre, mood and instrument classification. Similarly, [21] performed a study on the use of deep architectures in tandem with similarity learning for artist classification. This research on artist classification was further extended by [4], who focused on the use of similarity learning along with shallow architectures for the same task. [10] used similarity for melody retrieval, this has many applications such as matching hummed or sung melodies to songs in a digital music collection, instrument classification and music genre classification. The above studies reveal several benefits to using similarity learning when compared to classical approaches. This leads us to consider similarity learning as a straightforward extension for content-based music recommendation.

In this paper, we explore the use of a deep architecture, along with similarity learning for MIR and recommendation. We compare our results to a shallow approach, that makes use of MFCCs, for automated music recommendation. As such our contributions are:

- We explore the use of similarity learning for music information retrieval and automated music recommendation.
- We extend on existing work by using triplet loss to the train the *Paralleling Recurrent Convolutional Neural Network(PRCNN)* developed by **(author?)** [6], who use *Categorical Cross-Entropy Loss* in their approach.
- We show that similarity learning can be used to automatically extract descriptive audio features, that can be used for accurate music recommendations.

In terms of its impact, the use of similarity learning for automated music recommendation could lead to more accurate music recommendations allowing users to discover music, that better matches their preferences.

2 Methodology

2.1 Dataset

The FMA dataset [5] is a large music archive that consists of 106,574 songs from 16,341 artists and 14,854 albums, arranged in a hierarchical categorisation of 161 genres. The archive contains a variety of different song data, such as audio clips, precomputed audio features (for each song), Echonest features - which are audio features calculated by the *Echonest Analyzer* [24] - and tracks and artist information. The archive can be split into various sizes (small, medium and large) depending on the size of the experiment. We chose to use the *FMA* dataset over a more popular dataset, such as the *Million Song Data (MSD)* [1], due to its inclusion of audio clips.

For our experiments, similar songs were first grouped into classes based on the following labels: artist, album, genre and sub-genre. Songs which had at least two overlapping labels were placed in the same group. After the initial grouping, manual refinement was done to ensure all class labels were correct (i.e. songs within the same groups sounded similar). This grouping technique differs from grouping by genre, as the majority of classes come from the same genres (like Rock, Hip-Hop and Electronic), but are separated based on the album, artist and sub-genre. Music similarity is inherently subjective and as such, each listener has a unique opinion on which songs sound similar and which do not [2]. This subjective nature makes it difficult to label similar sounding music based solely on the album, artist, sub-genre and genre. The manual grouping was as as a result crucial in the dataset creation process. Each audio clip used was 30 seconds long and we used 80% of the data for training, 10% for validation and 10% for testing.

2.2 Audio features

For the baseline approach, the *Mel-frequency cepstral coefficients (MFCC)* for each song were used. These are calculated by taking the logarithm of the Mel magnitude spectrum and decorrelating the resulting values using a Discrete Cosine Transform [2]. These features were calculated using a window length of 2048 and a hop size of 512. The mean value for the MFCCs was extracted at each frame which resulted in a 1-dimensional 140 features vector for each song. These features were utilised directly to measure the similarity between songs.

For our approach, we used the *Log-scaled Mel-spectograms*. This is achieved by log scaling the Mel-spectrograms from each song. This created a feature vector of shape (640x128). Similarly to the baseline, these feature values were calculated using a window length of 2048 and a hop size of 512 [5]. The Mel-spectrograms were then sent through a deep architecture to extract the most meaningful features for each song.

2.3 Network Architecture

Inspired by the work of [6], a *Paralleling Recurrent Convolutional Neural Network(PRCNN)* was used in our implementation. This specific architecture was preferred due to its ability to preserve the temporal relationships of the original signals in the music, which is not possible using a standard Convolutional Neural Network (CNN). In our implementation we use *triplet loss* to train the *PRCNN*, and utilize the network for feature extraction rather than classification. The network consists of two blocks, the *Convolutional (CNN) block* and the *Bidirectional Gated Recurrent Units Block (BGRU-RNN)*, as seen in Figure 1. The feature vector, described in Section 2.2, is fed through both blocks in parallel.

The CNN block consists of 5 convolutional layers, with 16, 32, 64, 128 and 64 filters respectively. Between each convolutional layer, there are max-pooling layers. The first three max-pooling layers have a pool size of (2x2), and the upper two max-pooling have a pool size of (4x4). A *ReLu activation function* [7] was used for all of the convolutional layers and is described as:

$$R(z_i) = \begin{cases} z_i & z_i > 0 \\ 0 & z_i \leq 0 \end{cases},$$

where z_i is the input value at a given node i. The output from the convolutional block is a 256-dimensional vector.

For the BGRU-RNN block, the first layer is a max-pooling layer with a pool size of (4x2). Following the max-pooling layer is an embedding layer, for further dimensional reduction. The reduced vector is then sent to the bidirectional GRU with 64 units. The output from the BGRU-RNN block is a 128-dimensional vector.

The output from both blocks is then concatenated into a 384-dimensional vector. Finally, the concatenated vector is passed through a dense layer, which uses a *Sigmoid activation function* [19]:

$$S(x) = \frac{1}{1+e^{-x}}.$$

The dense layer returns a 60-dimensional feature vector. The overall architecture is illustrated in Figure 1.

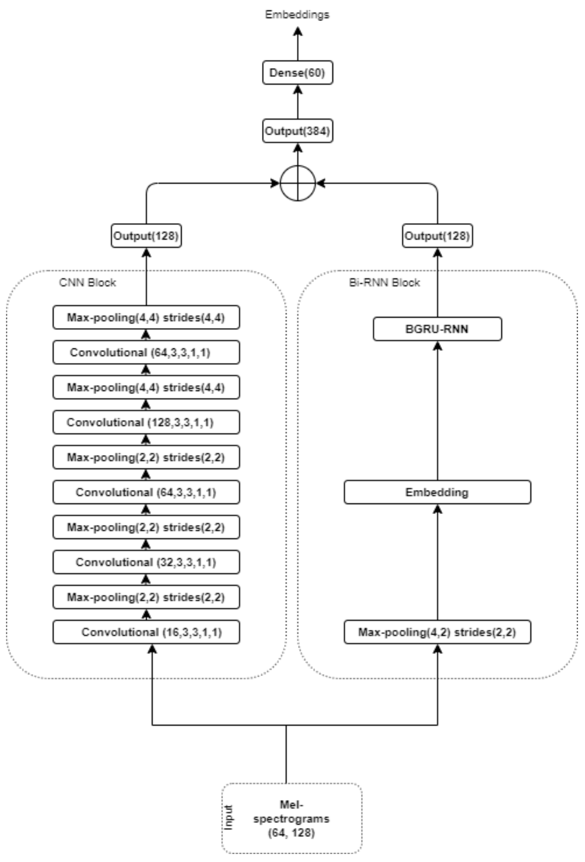
Embeddings
Dense(60)
Output(384)
Output(128)
Output(128)
CNN Block
Max-pooling(4,4) strides(4,4)
Convolutional (64,3,3,1,1)
Max-pooling(4,4) strides(4,4)
Convolutional (128,3,3,1,1)
Max-pooling(2,2) strides(2,2)
Convolutional (64,3,3,1,1)
Max-pooling(2,2) strides(2,2)
Convolutional (32,3,3,1,1)
Max-pooling(2,2) strides(2,2)
Convolutional (16,3,3,1,1)
Bi-RNN Block
BGRU-RNN
Embedding
Max-pooling(4,2) strides(2,2)
Input
Mel-spectrograms (64, 128)

Fig. 1: The network architecture of the PRCNN

The network architecture in Figure 1 was employed as a *Siamese Neural Network* to be trained using *Triplet Loss*. This involves creating 3 identical versions of the same *PRCNN* and updating the weights of all 3 networks with the same values dat each step of the training process (ie. throughout the entire training process the networks and their corresponding weights remain identical). For the prediction of the latent factors (following the training phase), we use only one of the three networks (because the networks are identical the one we choose does not matter).

2.4 Loss Function

The *PRCNN* was trained using two different loss functions, namely, *Triplet Loss* and *Categorical Cross Entropy Loss*. *Triplet loss* is described below as [25]:

$$\text{basic loss} = \Sigma_{i=1}^{N} \left[||f_i^a - f_i^p||^2 - ||f_i^a - f_i^n||^2 + \alpha \right]$$

where f_i^a, f_i^p and f_i^n, respectively represent the latent factor vectors of the anchor, positive and negative samples and $\alpha > 0$ is the margin of similarity (which can be increased to further separate embeddings).

This loss ensures that a given anchor point x_a and a positive point x_p belonging to same class y_a, are closer together than the same anchor point and a negative point x_n belonging to a different class y_n, by at least a margin α [30]. Furthermore any triple sample (anchor, positive, negative) that generated a negative *basic_loss* was set to 0 [25]:

$$\text{loss} = \max\{basic\_loss, 0.0\}$$

To optimise the performance of the triplet loss function a variety of mining strategies were employed, *Batch Random, Batch Semi-Hard* and our own batch strategy, the *custom batch strategy.* The *Batch Random* strategy assigned a random anchor with a random positive sample and a random negative sample. This strategy was used as a baseline to compare the other batch strategies to. The *Batch Semi-Hard* [30] strategy assigned positive and negative samples to a given anchor such that the negative sample was further away from the anchor than the positive sample, but closer, to the anchor, than the positive sample plus the margin α given by:

$$\Sigma_{i=1}^{N}||f_i^a - f_i^p||^2 < \Sigma_{i=1}^{N}||f_i^a - f_i^n||^2 < \Sigma_{i=1}^{N}||f_i^a - f_i^p||^2 + \alpha$$

Finally our batch strategy, the *custom batch* strategy, pairs each anchor with all positive samples and assigns only the hardest negative sample, that is the closest negative sample to the anchor, to each of the anchor, positive pairs to create the triplets.

Before utilizing any of the batch strategies mentioned above, the *PRCNN* was pre-trained for jump start and transfer learning purposes using *Categorical Cross Entropy Loss* [8]:

$$\text{loss} = -\Sigma_{b=1}^{B}\Sigma_{j=1}^{M}(y_j^{(b)} \log(\sigma(z_i^{(b)}))$$

Where M is the number of classes, B is the batch size and y is the binary indicator (0 or 1) for if class label j is the correct class label (as described in Section 2.1) for observation b. Finally $\sigma(z_i)$ is the output of the final layer, which is a *Softmax* activation function and is described as follows:

$$\sigma(z_i) = \frac{e^{z_i}}{\Sigma_{j=1}^{M} e^{z_j}}.$$

Note during this pre-training phase an extra *dense* layer was added to the *PRCNN* (which was later dropped when changing to *Triplet Loss*). This layer was made of 8 units and used a *Softmax* activation function (described in Equation 2.4). This final layer allowed us to use *Categorical Cross-entropy* to pre-train the network.

As a separate experiment, we also trained the *PRCNN* (to convergence) using *Categorical Cross-Entropy Loss* (Described in Equation 2.4). This version of the *PRCNN* was exactly the same as the version used to pre-train the *PRCNN* (for *Triplet Loss*). The final dense layer of the network was popped off, when predicting latent factors, to ensure the dimensions of the latent factor vectors remained consistent for all experiments. This final experiment was used as another baseline to compare the results of the *PRCNN* trained using *Triplet Loss* too.

2.5 Experiments

To test the effectiveness of the described latent factor models, i.e. the output from the *PRCNN* (using all batch strategies) as well as the version trained using *Categorical Cross-Entropy Loss*, and the baseline feature vector (made up of the MFCC features described in Section 2.2), a *KNN algorithm* along with *Cosine similarity* was used to group both the similar latent factors and the similar *MFCC* features. This algorithm was run on all feature/latent factor sets independently.

Given a song, the aim is to recommend N of the most similar sounding songs, by measuring the similarity between each songs respective feature embedding. To measure the similarity between two feature vectors, Cosine similarity [18] was used:

$$\text{similarity} = \frac{A \cdot B}{||A||||B||} = \frac{\Sigma_{i=1}^{n} A_i B_i}{\sqrt{\Sigma_{i=1}^{n} A_i^2}\sqrt{\Sigma_{i=1}^{n} B_i^2}}$$

where A and B represent the feature vectors of two different song's. Cosine similarity was chosen due to its benefits over other metrics such as *Euclidean Distance* and *Manhattan Distance*, for high dimensional vectors, where the magnitude, of the vector, is not important.

2.6 Hyper-parameters

To optimise the hyperparameters of the *PRCNN*, 8-fold cross-validation was used. The technique involves separating a shuffled set of input data and dividing it into 8 equal-sized sets. For each respective run, a different set is used for validation and testing. The final 6 sets for each run are used to perform training on the model. The testing accuracy from the model is averaged across all 6 runs to determine which hyperparameters provided the overall best results. *Early stoppage* was also used to prevent the network from overfitting. Finally, the optimiser function used was *Stochastic Gradient Descent* [22]. We kept the learning rate consistent for all experiments at 0.001.

2.7 Metrics

To measure the performance of our music recommendation system the following metrics are utilised:

Mean Reciprocal Rank (MRR) *Mean Reciprocal Rank (MRR)* is an indication of how high the first correct song recommendation is overall queries, and is described below as:

$$MRR = \frac{1}{|Q|} \Sigma_{i=1}^{|Q|} \frac{1}{rank_i}$$

where $||Q||$ and $rank_i$ respectively denote the number of queries and the position of the first relevant result.

Mean Average Precision (MAP) *Mean Average Precision (MAP)* is a measure of the average recommendation precision over all queries, and is described below as:

$$MAP = \frac{1}{||Q||} \Sigma_{q=1}^{||Q||} AP(q)$$

where $AP(q)$ denotes the average precision of a given query, q.

Precision@k (P@k) *Precision@k (P@K)* indicates how many of the first k recommendations came from the same class as the query, and is described as:

$$Precision@k = \frac{\text{true positives}@k}{(\text{true positives}@k\ ) + (\text{false positives }@k)}$$

Where k has been set to 1, 5 and 9 respectively.

3 Results

The overall *MAP* and *MRR* as well as the *Precision@1*, *Precision@5* and the *Precision@9* are all shown in Table 1. Note the results in Table 1 and Table 2 were retrieved by averaging the results after running each of the experiments, explained in Section 2.5, 30 times.

Table 1: The mean MAP, MRR, Precision@1, Precision@5 and Precision@9 score for each respective feature set. Note that the highest score for each metric is in bold

Features	**MRR**	**MAP**	**P@1**	**P@5**	**P@9**
Baseline Features (MFCC)	0.780 $\pm$ 0.0006	0.664 $\pm$ 0.0010	0.623 $\pm$ 0.0028	0.0.515 $\pm$ 0.0008	0.455 $\pm$ 0.0003
Random Batch Strategy	0.816 $\pm$ 0.0015	0.724 $\pm$ 0.0012	0.679 $\pm$ 0.0047	0.630 $\pm$ 0.0022	0.580 $\pm$ 0.0018
Semi-hard Batch Strategy	**0.900 $\pm$ 0.0003**	**0.845 $\pm$ 0.0002**	**0.827 $\pm$ 0.0013**	**0.750 $\pm$ 0.0017**	**0.687 $\pm$0.0017**
Custom Batch Strategy	0.856 $\pm$ 0.0016	0.773 0.0014	0.750 $\pm$ 0.0041	0.694 $\pm$ 0.0022	0.637 $\pm$ 0.0020
Categorical Crossentropy Loss	0.862 $\pm$ 0.0014	0.785 $\pm$ 0.0014	0.749 $\pm$ 0.0035	0.0.721 $\pm$ 0.002	0.658 $\pm$ 0.0012

The *TSNE* algorithm [15] was used to plot the various feature emebddings in a 2D plane, shown in Figures 2, 3, 4, 5 and 6. Finally, Table 2 shows the *MAP* and *MRR* score for each class individually. Each column refers to one of the feature sets described in Section 2.5, while each row represent each "class" of similar sounding music.

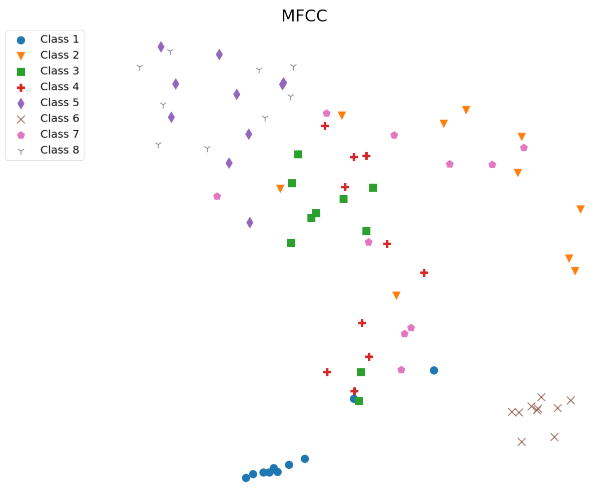

Fig. 2: *TSNE* Plot for *MFCC* features

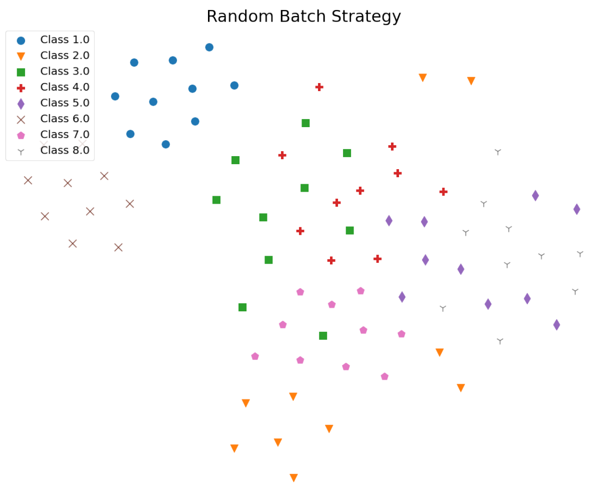

Fig. 3: *TSNE* Plot for *Random batch* features

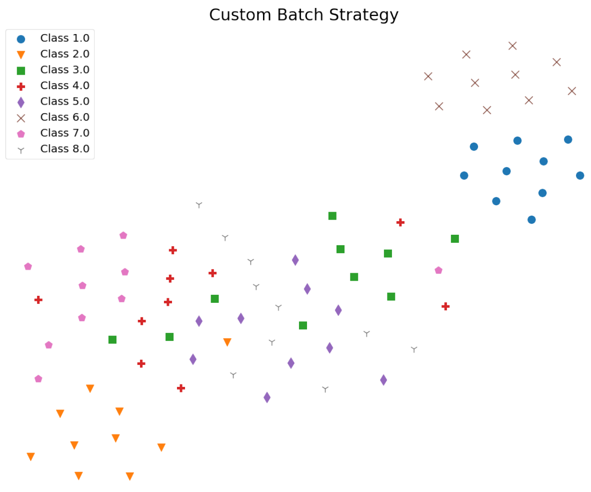

Fig. 4: *TSNE* Plot for *Custom batch* features

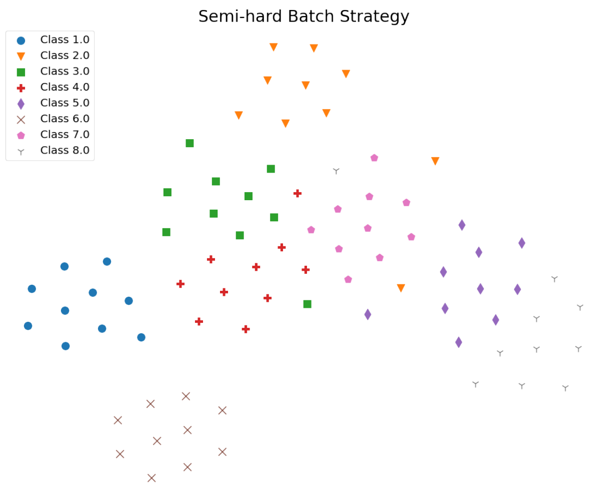

Fig. 5: *TSNE* Plot for *Semi-hard batch* features

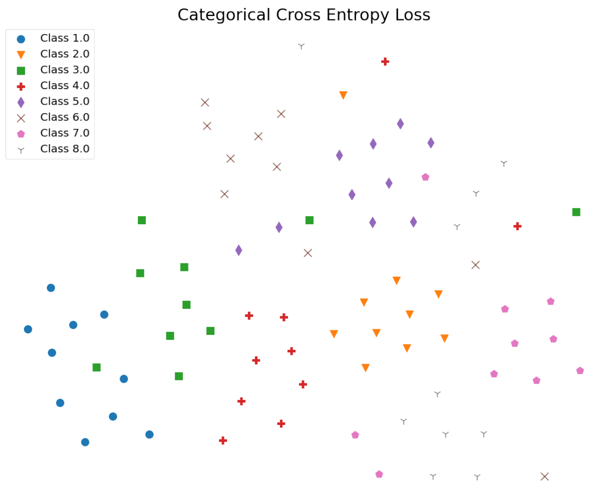

Fig. 6: *TSNE* Plot for *Categorical Crossentropy Loss* features

Table 2: The mean MAP and MRR score of each feature set for each unique music class. Note the highest average of the MAP and MRR scores, for each class, is highlighted in bold (CLL is an abbreviation for Categorical Cross Entropy Loss).

Strategy	**Custom Batch**		**Random Batch**		**Semi-hard Batch**		**Baseline(MFCC)**		**CCL**	
Class	**MAP**	**MRR**	**MAP**	**MRR**	**MAP**	**MRR**	**MAP**	**MRR**	**MAP**	**MRR**
Class 1	**1.0 ± 0.0**	**1.0 ± 0.0**	0.99 ± 0.0	0.99 ± 0.0	**1.0 ± 0.0**	**1.0 ± 0.0**	0.98 ± 0.0	0.99 ± 0.001	**1.0 ± 0.0**	**1.0 ± 0.0**
Class 2	0.68 ± 0.01	0.80 ± 0.009	0.67 ± 0.026	0.77 ± 0.031	**0.83 ± 0.006**	**0.96 ± 0.003**	0.62 ± 0.008	0.75 ± 0.013	0.73 ± 0.009	0.81 ± 0.01
Class 3	0.85 ± 0.007	0.92 ± 0.003	0.70 ± 0.010	0.80 ± 0.010	**0.89 ± 0.003**	**0.93 ± 0.003**	0.57 ± 0.011	0.70 ± 0.007	0.82 ± 0.008	0.9 ± 0.004
Class 4	0.54 ± 0.017	0.64 ± 0.017	0.56 ± 0.007	0.73 ± 0.003	0.59 ± 0.004	0.72 ± 0.010	0.59 ± 0.010	0.69 ± 0.024	**0.62 ± 0.024**	**0.78 ± 0.017**
Class 5	0.83 ± 0.009	0.93 ± 0.009	0.64 ± 0.006	0.78 ± 0.007	**0.93 ± 0.004**	**0.97 ± 0.001**	0.41 ± 0.008	0.64 ± 0.014	0.77 ± 0.15	0.87 ± 0.011
Class 6	0.092 ± 0.0	1.0 ± 0.0	0.97 ± 0.001	0.99 ± 0.0	**1.0 ± 0.0**	**1.0 ± 0.0**	0.93 ± 0.001	0.99 ± 0.0	**1.0 ± 0.0**	**1.0 ± 0.0**
Class 7	0.63 ± 0.010	0.78 ± 0.012	0.58 ± 0.007	0.68 ± 0.018	**0.74 ± 0.007**	**0.86 ± 0.005**	0.53 ± 0.012	0.67 ± 0.01	0.65 ± 0.006	0.75 ± 0.007
Class 8	0.66 ± 0.008	0.77 ± 0.014	0.68 ± 0.009	0.78 ± 0.017	0.62 ± 0.11	0.76 ± 0.020	0.68 ± 0.006	0.8 ± 0.005	**0.69 ± 0.008**	**0.79 ± 0.008**

4 Discussion

4.1 Evaluation

From Table 1 we note that all latent factors extracted using the *PRCNN*, regardless of the chosen batch or training strategy, scored a higher *MAP* and *MRR* score than the MFCC baseline features. The latent factors extracted using the version of the *PRCNN* trained using *catergorical cross-entropy loss* outperformed the latent factors extracted using the *random batch* strategy, while also scoring very similar results (with even a slight advantage) to the latent factors extracted using the *custom batch* strategy. The highest achieved *MRR* and *MAP* scores were 0.900 and 0.845 respectively, and both were achieved using the *Semi-hard Batch Strategy*.

Performance within the different batch strategies was as expected, with the *Random Batch Strategy* performing the worst, followed by the *Custom Batch Strategy* and finally, the best performing tested batch strategy was the *Semi-hard Batch Strategy*. The *Random Batch Strategy* is naive and so was unable to make clear distinctions between classes which shared some degree of similarity. The *Custom Batch Strategy* is slightly more intelligent when creating triplets, and so we see slight improvements over the *Random Batch Strategy*. Finally, significant improvements were seen, over the other two batch strategies, when using the *Semi-hard Batch Strategy*, this can be attributed to the selection of triplets that lead to the greatest degree of separability between different classes of music. It is important to note that we also tested a *Hard Batch Strategy*[30], but using this strategy, the *BGRU-RNN* was unable to converge to a local minimum, and thus almost no distinctions were made between classes.

From Figures 2, 3, 4, 5 and 6 we note that some classes were more separable than others. Certain classes, such as class 1 and class 6, performed well for all features sets, while other classes, such as class 4, performed poorly throughout. Some overlap between classes is to be expected, as certain music features are consistent across all classes, and these features are almost impossible to neglect when measuring the similarity between songs. The results observed in figures 2, 3, 4 and 5 are further supported by Table 2, where we see high *MRR* and *MAP* scores for classes 1 and 6 across all feature sets, while class 4 comparatively performed poorly.

4.2 Related Work

Similar sounding music is, for the most part, subjective, and as such, it is difficult to make a direct comparison between our results, and what is seen in other relevant literature. Although there are certain consistencies within approaches which are important to note.

[6] used a *BGRU-RNN* for *Music Genre Classification* on the GTZAN [28] dataset. Each 30 second audio clip was converted to a Short-time Fourier Transform (STFT) [26] with dimensions 128x513. The network was trained using *Categorical Cross-Entropy Loss* and classification was done in the final layer rather than the feature embedding learning in our approach. This work differs further from our approach, where the *BGRU-RNN* was used rather as a latent factor predictor and *Triplet Loss* was the selected loss function.

[20] proposed the use of content-based music recommendation as a potential substitute for more common *collaborative filtering* methods. They explored 2 approaches to content-based music recommendation, namely, a *Bag-of-Words* representation of *MFCCs* paired with linear regressions and a *Convolutional Neural Network (CNN)*, and compared them with *collaborative filtering methods* from relevant literature. Tests were run on *Million Song Dataset (MSD)* [1], which is significantly larger than our dataset, and so it's hard to make a direct comparison between results. Although, it is important to note that there are certain consistencies, between our results and theirs, with their deep architecture, a non-recurrent *CNN*, outperforming the shallow architecture, *MFCCs*, which is similar to what we saw in our results.

Finally, [3] conducted a comparative study of the benefits of a *Convolutional Recurrent Neural Network (CRNN)* over a *Convolutional Neural Network* for music genre classification. Similarly to [20], they ran tests on the *Million Song Dataset* [1] and so again it is hard to make comparisons between their results and ours. They found that both network architectures have potential benefits for music genre classification. Our chosen network architecture, the *BGRU-RNN*, consists of both a *CNN* component and a *RNN* but connected in parallel, which is in contrast to the *CRNN* in their implementation, where they have them connected in series.

5 Conclusions

In this paper, we proposed the use of similarity learning as a feature learning architecture for music audio. We compared audio features extracted using similarity learning to baseline features, MFCCs, and tested the capabilities of both feature sets in the context of content-based music recommendation. Our results showed that not only is similarity learning a viable approach to audio feature extraction, but, in the context of automated music recommendation, showed significant improvements over the baseline features, the MFCCs and even outperformed the same network trained with a different, more common loss function, namely *categorical cross-entropy loss*. Furthermore, from the tested batch strategies, we found the *Semi-hard batch strategy*, was the most effective for audio feature extraction. Finally, we observed that some classes of music are more differentiable than others, and as such performed better for tested feature extraction methods. In the future, we would like sample features from the entire song rather than just a 30-second clip. We would also like to test various other similarity loss functions, such as angular and pairwise loss.

References

[1] Bertin-Mahieux, T., Ellis, D.P., Whitman, B., Lamere, P.: The million song dataset (2011)

[2] Casey, M.A., Veltkamp, R., Goto, M., Leman, M., Rhodes, C., Slaney, M.: Content-based music information retrieval: Current directions and future challenges. Proceedings of the IEEE **96**(4), 668–696 (2008)

[3] Choi, K., Fazekas, G., Sandler, M., Cho, K.: Convolutional recurrent neural networks for music classification. In: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 2392–2396. IEEE (2017)

[4] Cleveland, J., Cheng, D., Zhou, M., Joachims, T., Turnbull, D.: Content-based music similarity with triplet networks. arXiv preprint arXiv:2008.04938 (2020)

[5] Defferrard, M., Benzi, K., Vandergheynst, P., Bresson, X.: Fma: A dataset for music analysis. arXiv preprint arXiv:1612.01840 (2016)

[6] Feng, L., Liu, S., Yao, J.: Music genre classification with paralleling recurrent convolutional neural network. arXiv preprint arXiv:1712.08370 (2017)

[7] Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics. pp. 315–323 (2011)

[8] Gómez, R.: Understanding categorical cross-entropy loss, binary cross-entropy loss, softmax loss, logistic loss, focal loss and all those confusing names. URL: https://gombru. github. io/2018/05/23/cross_ entropy_loss/(visited on 29/03/2019) (2018)

[9] Humphrey, E.J., Bello, J.P., LeCun, Y.: Moving beyond feature design: Deep architectures and automatic feature learning in music informatics. In: ISMIR. pp. 403–408. Citeseer (2012)

[10] Karsdorp, F., van Kranenburg, P., Manjavacas, E.: Learning similarity metrics for melody retrieval. In: ISMIR. pp. 478–485 (2019)

[11] Kaya, M., Bilge, H.Ş.: Deep metric learning: A survey. Symmetry **11**(9), 1066 (2019)

[12] Kulis, B., et al.: Metric learning: A survey. Foundations and trends in machine learning **5**(4), 287–364 (2012)

[13] Lee, J., Bryan, N.J., Salamon, J., Jin, Z., Nam, J.: Disentangled multidimensional metric learning for music similarity. In: ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 6–10. IEEE (2020)

[14] Lison, P.: An introduction to machine learning. Language Technology Group (LTG) **1**(35) (2015)

[15] Maaten, L.v.d., Hinton, G.: Visualizing data using t-sne. Journal of machine learning research **9**(Nov), 2579–2605 (2008)

[16] Mandel, M.I., Ellis, D.P.: Song-level features and support vector machines for music classification (2005)

[17] Manocha, P., Badlani, R., Kumar, A., Shah, A., Elizalde, B., Raj, B.: Content-based representations of audio using siamese neural networks. In: 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 3136–3140. IEEE (2018)

[18] Nguyen, H.V., Bai, L.: Cosine similarity metric learning for face verification. In: Asian conference on computer vision. pp. 709–720. Springer (2010)

[19] Njikam, A.N.S., Zhao, H.: A novel activation function for multilayer feed-forward neural networks. Applied Intelligence **45**(1), 75–82 (2016)

[20] Van den Oord, A., Dieleman, S., Schrauwen, B.: Deep content-based music recommendation. In: Advances in neural information processing systems. pp. 2643–2651 (2013)

[21] Park, J., Lee, J., Park, J., Ha, J.W., Nam, J.: Representation learning of music using artist labels. arXiv preprint arXiv:1710.06648 (2017)

[22] Robbins, H., Monro, S.: A stochastic approximation method. The annals of mathematical statistics pp. 400–407 (1951)

[23] Sammut, C., Webb, G.: Latent factor models and matrix factorizations (2010)

[24] Schindler, A., Rauber, A.: Capturing the temporal domain in echonest features for improved classification effectiveness. In: International Workshop on Adaptive Multimedia Retrieval. pp. 214–227. Springer (2012)

[25] Schroff, F., Kalenichenko, D., Philbin, J.: Facenet: A unified embedding for face recognition and clustering. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 815–823 (2015)

[26] Sigtia, S., Dixon, S.: Improved music feature learning with deep neural networks. In: 2014 IEEE international conference on acoustics, speech and signal processing (ICASSP). pp. 6959–6963. IEEE (2014)

[27] Slaney, M., Weinberger, K., White, W.: Learning a metric for music similarity. In: International Symposium on Music Information Retrieval (ISMIR). vol. 148 (2008)

[28] Tzanetakis, G., Cook, P.: Gtzan genre collection. Music Analysis, Retrieval and Synthesis for Audio Signals (2002)

[29] Wang, J., Zhang, T., Sebe, N., Shen, H.T., et al.: A survey on learning to hash. IEEE transactions on pattern analysis and machine intelligence **40**(4), 769–790 (2017)

[30] Xuan, H., Stylianou, A., Pless, R.: Improved embeddings with easy positive triplet mining. In: The IEEE Winter Conference on Applications of Computer Vision. pp. 2474–2482 (2020)