LEARNING TO RANK MUSIC TRACKS USING TRIPLET LOSS

Laure Prétet, Gael Richard, Geoffroy Peeters

LEARNING TO RANK MUSIC TRACKS USING TRIPLET LOSS

Laure Prétet[⋆†] *Gaël Richard*[⋆] *Geoffroy Peeters*[⋆]

[†] Creaminal, Paris, France
[⋆] LTCI, Télécom Paris, Institut Polytechnique de Paris, France

ABSTRACT

Most music streaming services rely on automatic recommendation algorithms to exploit their large music catalogs. These algorithms aim at retrieving a ranked list of music tracks based on their similarity with a target music track. In this work, we propose a method for direct recommendation based on the audio content without explicitly tagging the music tracks. To that aim, we propose several strategies to perform triplet mining from ranked lists. We train a Convolutional Neural Network to learn the similarity via triplet loss. These different strategies are compared and validated on a large-scale experiment against an auto-tagging based approach. The results obtained highlight the efficiency of our system, especially when associated with an Auto-pooling layer.

***Index Terms*—** audio music similarity, deep learning, triplet loss, triplet mining.

1. INTRODUCTION

Many domains, such as music streaming services, make use of large music catalogs. To organize these tracks, it is necessary to provide efficient retrieval mechanisms. While browsing by tags (genre, mood, instrumentation) can be efficient for small-scale catalogs, it does not provide an efficient retrieval mechanism at scale. This is why most music streaming services rely on music recommendation. For this purpose, an algorithm is used to retrieve a ranked list of music tracks based on their similarity with a target music track. Provided a similarity metric is defined, the music ranking problem reduces to a *music similarity problem*. This task has been the subject of many publications (see [1] for an overview) and can be tackled in different ways.

In *Collaborative filtering recommendation*, two music tracks can be considered similar if they are listened to by the same audience [2] [3]. Obviously, if no one has ever listen to a music track (for example a new track in the catalog), it can not be recommended. This is known as the cold start problem [4]. However, when applicable, collaborative filtering has proved to provide very good results for recommendation.

In *Tag-based recommendation*, a tag-based similarity measure can be designed to compare music tracks based on their respective tags. Tags can be manually annotated (such as in Pandora [5]), crowd-sourced (such as in Last.fm), or automatically inferred from the audio content (auto-tagging case [6]).

In *Direct recommendation*, it is possible to compute directly a distance between two music tracks based on their audio similarity. For example, in one of the pioneer works [7], track MFCCs were represented by a Gaussian Mixture Model and a Kullback-Leibler divergence was used to compare two tracks. Since these methods use a costly pairwise comparison, they cannot scale to large catalogs.

In this work, we propose a method for *direct recommendation* based on the audio content. To this aim, we suppose we have access to a large music catalog professionally annotated with tags. We also assume we are given a function $\mathcal{S}$ which allows to compute a similarity score between two sets of tags. For a given target track, $\mathcal{S}$ allows us to retrieve similar tracks based on their tags. Our goal is to reproduce the similarity ranking given by $\mathcal{S}$ *without explicitly tagging the tracks*. We denote our approximation by $\hat{\mathcal{S}}$. For a given target track, $\hat{\mathcal{S}}$ allows us to retrieve similar tracks based on audio directly. This allows to skip the long and expensive manual tagging step. In this work, we tackle this task using deep learning. We train a Convolutional Neural Network (CNN) with triplet loss to learn a projection of the audio signal such that the proximity between two projected music tracks accounts for their similarity $\mathcal{S}$. We compare this similarity to the one obtained by a CNN trained to estimate automatically the related tags which are then used in $\mathcal{S}$.

Paper contributions. The three main contributions of our work are the following. First, we propose several strategies to perform triplet mining from ranked lists. This allows to apply a triplet loss to the relative music similarity problem. Second, we compare and validate these different strategies on a large-scale experiment against the auto-tagging based approach. Third, we demonstrate the efficiency of the recently proposed Auto-pooling layer [8] for a music task.

Paper organization. In section 2, we review works related to ours. In section 3, we describe our proposed method to mine triplets from ranked lists. In section 4, we evaluate the proposed approach, along with an auto-tagger baseline, on a music retrieval task. In section 5, we draw the conclusions of our results.

2. RELATED WORK

2.1. Music auto-tagging

In Music Information Retrieval (MIR), we call *music auto-tagging* the task of predicting tags directly from audio signals. Tags are keywords that describe a music track in terms of genre, mood, instrumentation or any other high-level attributes.

Traditional approaches for music auto-tagging rely on the extraction of handcrafted features to feed a classifier (linear or non-linear) [9] [10]. Recently, deep learning approaches have allowed to learn the features directly from the data (waveforms or spectrograms), leading to improved performances [11] [12] [13] [6]. Some of these systems have proven their capacity to learn useful information from audio [11] [14]. Therefore, we take inspiration from those for our CNN architecture, but we use this CNN to learn music similarity instead of tags.

2.2. Learning to rank

The task of retrieving items in a collection and to sort them by relevance arises in a variety of domains. In early works, Weston [**?**]

and Usunier [15] proposed loss functions capable of optimizing the top of ranked lists in item recommendation and information retrieval contexts. Another common approach is to learn a similarity notion from classes. In this case, it is assumed that the similarity of items within a same class should be higher than the similarity of items from different classes. As a result, the model's recommendations are associated to a binary relevance score: the recommended item does or does not belong to the expected class. Such problems have been studied by for example by Weinberger and Saul [16] and Hoffer and Ailon [17], who employed CNNs with innovative loss functions to perform higher-accuracy image classification. Today, a widely used loss for similarity learning is the *triplet loss*, as introduced by [16] and used by Schroff and Philbin [18] for face recognition. In their work, they use the triplet loss to force the CNN to learn a projection of the image data such that the projections of images of the same person will be pulled together, and the ones from different persons will be pushed apart. The distance employed to compare the projections of the data is usually the Euclidean distance, squared Euclidean distance, or the cosine distance [19].

In our problem (learning a similarity from ranked lists), there are however no classes, nor binary relevance labels for each query-document pair. Such a problem has already been addressed outside the music case. For example, Mcfee and Lanckriet [20] propose to use a listwise loss function to learn text recommendation from ranked lists. Wang et al. [21] propose to use the triplet loss to learn a ranking of similar images. Our proposal takes inspiration from the work of [21] but for the music case.

2.3. Music similarity

The literature on music similarity is vast (see Wolff and Weyde [22] for an overview). For example, Slaney et al. [23] propose a set of linear transforms to embed tracks into an Euclidean metric space and evaluate them on a nearest neighbor task for album, artist and blog recognition. Tag-based approaches to metric learning include a method by Weston et. al. [24] to project both audio features and music tags into a shared embedding space. Wolf and Weyde [22] insist on modeling *relative* music similarity (rankings) rather than absolute similarity ratings, in order to avoid consistency issues due to subjective user ratings. Following this idea, Lu et al. [25] employ a CNN with an improved triplet loss to predict relative music similarity. However [25] do not propose any mining strategies from ranked lists[1]. Our work takes inspiration from [25] but proposes a mining strategy from ranked lists.

3. PROPOSED METHOD

3.1. Problem definition

Let $D = \{t_1, \dots, t_N\}$ be a set of N tracks annotated with a taxonomy of m tags. The problem addressed in this study is the following: given a query track t, compute a ranked list of tracks from the dataset D ordered by descending similarity to t. Let $\mathcal{S}$ be an oracle similarity function that, given two sets of tag likelihoods $t_1 \in [0,1]^m$ and $t_2 \in [0,1]^m$, returns a similarity score $\mathcal{S}(t_1, t_2) \in \mathbb{R}_+$.

For any $t \in D$, let $R^{\mathcal{S}}(t) = [r_1(t), \dots, r_{N-1}(t)]$ be the ordered list of the other tracks of D ranked by decreasing similarity according to the function $\mathcal{S}$. This is the **ground truth**, against which our system will be evaluated. For any $t \in D$, let $R^{\hat{\mathcal{S}}}(t) = [\hat{r}_1(t), \dots, \hat{r}_{N-1}(t)]$ be the estimated list of recommended tracks made by the system for the target track t.

In this paper, we formulate the music ranking problem as a nearest neighbor search problem in a d-dimensional Euclidean space. A model is trained to define a specific *embedding space* (a projection of the data) in which the Euclidean distance allows to retrieve tracks with the same ranking as with the oracle similarity function $\mathcal{S}$.

3.2. Mining triplets from ranked lists

We denote by $f(t) \in \mathbb{R}^d$ the embedding of the track t. f is obtained by training a CNN using a triplet loss.

This loss takes as input a triplet of tracks that consists of an anchor *a*, a positive example *p* and a negative example *n*. The CNN outputs the embedding vectors of those, *f(a), f(p)* and *f(n)* respectively. The triplet is created such that the positive is more similar to the anchor than the negative according to the ground truth $\mathcal{S}$. The triplet loss then compares the squared Euclidean distances between the three embedding vectors and ensures that the same condition is respected in the embedding space:

$$\mathcal{L}(a,p,n) = \max(||f(a) - f(p)||_2^2 - ||f(a) - f(n)||_2^2 + \alpha, 0).$$

In this expression, α is a margin parameter that enforces a minimal distance between the positive and negative pairs.

To train such a system, it is necessary to prepare the data in the form of triplets (a,p,n). Let t be a target track and $R^{\mathcal{S}}(t) = [r_1(t), \dots, r_{N-1}(t)]$ the associated reference ranking. We define training triplets by using the track t as the anchor and by mining a positive and a negative element from $R^{\mathcal{S}}(t)$. A triplet is considered *valid* if the index of the positive element is lower than the one of the negative element: $(a,p,n) = (t, r_i(t), r_j(t)) \quad \forall i < j$.

In practice, for large datasets, it is infeasible to use all valid triplets for training, because their number grows cubically with N. Additionally, all triplets may not be useful for training. Figure 1 (top) shows an illustration of the average similarity scores $[\mathcal{S}(t, r_1(t)), \dots, \mathcal{S}(t, r_{N-1}(t))]$ for each track $t \in D$. The curve shows that a few first tracks in the ranking are very similar to their target, while most tracks in the dataset are actually irrelevant: their similarity score is lower than 50%. Therefore, after a certain rank in $R^{\mathcal{S}}(t)$, mining positive samples does not make sense.

Given these observations, we limit the set of possible positives per anchor to the N_p first elements of the reference ranking: $p \in [r_1(t), \dots, r_{N_p}(t)]$. We also limit the overall number of negatives per anchor-positive pair to N_n. Then, to select the N_n negative examples for a given anchor-positive pair $(t, r_i(t))$, three different strategies are tested (see Figure 1, bottom):

- **Neighbors:** The negatives are the N_n elements in $R^{\mathcal{S}}(t)$ that come directly after the positive: $n \in [r_{i+1}(t), \dots, r_{i+1+N_n}(t)]$.
- **Random uniform:** The negatives are sampled uniformly among the full $R^{\mathcal{S}}(t)$ list after the positive: $n \in \{r_j(t)\}_{j>i}$ s.t. $P(n) \sim \mathcal{U}(1/(N-i+1))$.
- **Distance-based** (inspired by [27] and [21]): The negatives are sampled among the full $R^{\mathcal{S}}(t)$ list after the positive with a probability that is proportional to its similarity with the anchor: $n \in \{r_j(t)\}_{j>i}$ s.t. $P(n) \propto \mathcal{S}(t,n)$.

This way, in all three variants, the set of triplets can be preselected offline and we do not mine triplets during training. The resulting total number of triplets is $N \times N_p \times N_n$.

3.3. Auto-pooling

In usual CNNs applied to audio signals, the time dimension of the audio signal is progressively discarded by a succession of max-

[1] This is because these two last studies train their model using the similarity triplet annotations provided in the MagnaTagATune dataset [26], which has only partial similarity information, annotated by non-expert users.

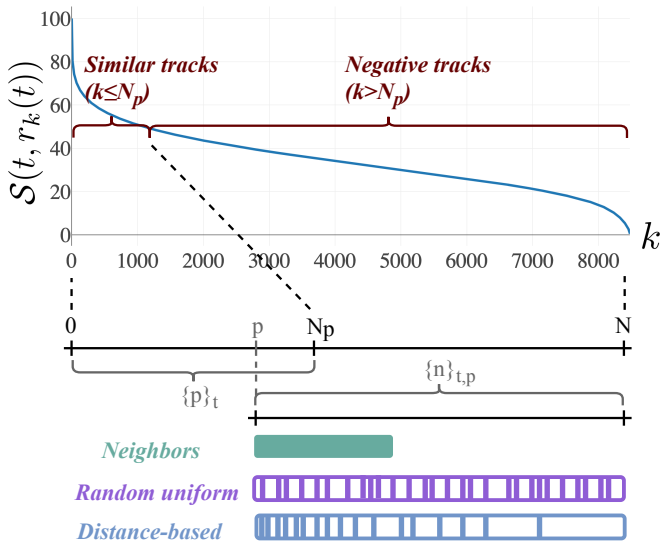

Fig. 1: Illustration of the three proposed triplet mining strategies. Top: Similarity scores in a typical similarity ranking (in percentage), as a function of the rank in the list $R^{\mathcal{S}}(t)$. Bottom: Three strategies to mine negative samples for a given anchor-positive pair (t, p).

pooling layers. This implies a strong assumption related to how information over time is processed: we only keep the maximum activation over successive time frames. Other choices have been made in the past such as the combination of Mean, Max and L2 Pooling [28]. A very elegant formulation has been proposed by McFee et al. [8] with the Auto-Pooling layer, which allows to interpolate between several pooling operators (such as min-, max-, and average-pooling) via a learned parameter. Auto-pooling has provided very good results for an audio event detection task. To our knowledge, it has not been used for music-related tasks. We do this here for the task of music similarity.

3.4. System architectures

In the following, the input representation used for all our architectures is the Constant-Q transform (CQT). For each track, we compute a CQT of 96 bins (12 bins/octave) with f_{min}=32.70 Hz, and a hop size of 1024 at 44.1 kHz. We then convert it to power amplitude and log-scale the magnitudes. The input of our CNN is a patch of 512 CQT frames (11.88s). This duration was chosen as a good compromise between memory efficiency and sufficient musical context. Since the annotations of our dataset are at the track level (see Section 4.1), we randomly select several of these (96×512) patches to represent a given track and assume that the annotations apply to each patch.

At test time, each track is represented by 8 randomly selected patches. When the network is used for auto-tagging, we pass each of them through our trained network to obtain the tag probability vector. We then simply use the average vector over the 8 tag probability vectors. When the network is used for embedding, we compute the average embedding vector over the 8 embedding vectors.

All models have been implemented using Keras with an Adam optimizer, a batch size of 42 patches and early stopping.

Baseline system: Auto-tagger (AT): The similarity function $\mathcal{S}$, used for ranking, relies on tag likelihood annotations. A naive approach is therefore to automatically estimate these tag likelihoods from the audio and then apply $\mathcal{S}$ directly to the estimated likelihood vectors. Auto-tagging is a multi-label classification problem (output activations are sigmoids, loss defined as the sum of binary cross-entropies). Preliminary experiments have shown that VGG-like architectures [6] were more suited to our dataset than musically-motivated architectures [13]. Thus, we reproduce the *FCN-5* architecture proposed by Choi et al. [6]. We adapt it to the shape of our inputs (96×512) and outputs (m tags). While the ground-truth annotations are likelihoods in $[0, 1]^m$, we train the system with binarized outputs $\{0, 1\}^m$. This system is referred to as *AT Baseline* in the rest of the paper. We give its architecture in Table 1, column 1 and provide the details (dropout, activations) in our code. We train it with a learning rate of 10^{-4}.

AT Baseline	TL	TL Autopool
CQT *(input: F=96,T=512,C=1)*		
Conv2D (3,3)×128		Conv2D (3,3)×64
MP (2,4) *(F=48,T=128,C=128)*		MP (2,2) *(F=48,T=256,C=64)*
Conv2D (3,3)×256		Conv2D (3,3)×128
MP (2,4) *(F=24,T=32,C=256)*		MP (2,2) *(F=24,T=128,C=128)*
Conv2D (3,3)×512		Conv2D (3,3)×256
MP (2,4) *(F=12,T=8,C=512)*		MP (2,2) *(F=12,T=64,C=256)*
Conv2D (3,3)×1024		Conv2D (3,3)×512
MP (3,3) *(F=4,T=2,C=1024)*		MP (2,2) *(F=6,T=32,C=512)*
Conv2D (3,3)×2048		Conv2D (3,3)×1024
MP (4,2) *(F=1,T=1,C=2048)*		MP (6,1) *(F=1,T=32,C=1024)*
		FC (*d*) *(F=1,T=32,C=d)*
FC (*m*)	FC (*d*)	Autopool (1,32) (*d*)

Table 1: Details of the three architectures used. In italic are the output sizes of the Max-Pooling (MP) or Fully-Connected (FC) layers.

Triplet loss system (TL): Our objective here is to estimate directly an embedding such that applying the Euclidean distance between the embeddings of two tracks t_1, t_2 mimics $\mathcal{S}(t_1, t_2)$. The network we use to compute the embedding is similar to the *AT Baseline* one, but it differs in the output layer. The last fully-connected layer has now d units (the dimension of the embedding space) instead of the m units, and has linear activations instead of m sigmoid activations (see Table 1, column 2). After a short grid search in a pilot experiment, we set d to 128. The embeddings are L2-normalized to the unit sphere. The margin parameter α of the triplet loss is set to 0.5 and the learning rate to 10^{-6}. Each mini-batch contains 42 triplets of patches. A given mini-batch represents one anchor track, one positive track and 42 negative tracks. Patches from these tracks are then randomly selected.

In the rest of the paper we denote this network as *TL* . We test it with the three sampling strategies presented in 3.2: *Neighbors, Random uniform* and *Distance-based* with N_p=15 and N_n=250.

Triplet loss system with Auto-pooling (TL Autopool): In the *AT Baseline* and *TL* networks, the time dimension is progressively removed by a succession of max-pooling layers. We test here the use of the Auto-Pooling layer proposed by McFee et al. [8]. Two main adaptations were necessary to use Auto-pooling in our setup. First, the max-pooling sizes of the *TL* network need to be adapted to carry some temporal information until the last layer. Second, the number of filters needs to be divided by two due to GPU memory constraints. This network is refered to as *TL Autopool* in the rest of the paper (see Table 1, column 3). As for the *TL* network, we train it to output

embeddings using the triplet loss. In the following, TL Autopool will only be tested with the Distance-based mining strategy.

4. EVALUATION OF THE PROPOSED METHOD

4.1. Dataset

To test our proposal, we need a dataset for which tags have been annotated at the track level and a similarity metric has been designed. Such datasets exist in streaming services such as Pandora. In our case, we use an extract of $N = 14,246$ tracks from the catalog of Creaminal, a music supervision company. This dataset is private and cannot be shared but the proposals made here are not specific to this dataset and can be applied to other ones.

Each track has a duration comprised between 45s and 5 minutes, and is sampled at 44100 Hz. The taxonomy used for this dataset is made of $m = 488$ tags, organized in 5 categories: Genre (e.g. Blues, Reggae, Electro-funk, Japanese Pop), Recording (e.g. Acoustic, Saturated, Guitar bass), Mood (e.g. Epic, Dancing, Nostalgic), Movement (e.g. Acceleration, Repetitive), and Lyrics (e.g. Death, Freedom, Nature). Each track was professionally annotated with a number of tags comprised between 5 and 35, the average number of tags per track being 16.8. The dataset features a majority of Pop tracks, along with Electro, Rock, Country and Movie soundtracks. The function $\mathcal{S}$ is also specific to this dataset and relies on a non-linear combination of weighted tags. For our experiments, we split the dataset into a training, validation and test sets (60%, 20% and 20% respectively)[2]. The distribution of tags is approximately the same in training and test.

4.2. Evaluation metrics

To evaluate our systems, we used each track of the test set as a query and ask the systems to rank all the other tracks of the test set by decreasing similarity with the query.

Let $R_k^{\mathcal{S}}(t) = [r_1(t), \ldots, r_r(k)]$ be the list $R^{\mathcal{S}}(t)$, truncated at rank k. Without loss of generality, we consider here that the *relevant* tracks to recommend for a given test query t are the five first tracks of the ranking: $R_5^{\mathcal{S}}(t)$. Thus, for each target track t, we ask our system to retrieve the 5 relevant ground truth recommendations $R_5^{\mathcal{S}}(t)$ among its k estimated recommendation $R_k^{\hat{\mathcal{S}}}(t)$. Here we set k to 20. We then use four of the evaluation metrics proposed by [1]:

- *Mean Average Precision (MAP)*: evaluates if the relevant tracks appear in high position in $R_k^{\hat{\mathcal{S}}}(t)$;
- *Recall@k*: indicates which proportion of the relevant tracks appear in $R_k^{\hat{\mathcal{S}}}(t)$;
- *Reciprocal rank (RR)*: is the inverse of the rank of the first relevant track in $R_k^{\hat{\mathcal{S}}}(t)$. Since we consider only the top k, the reciprocal rank is set to 0 if the rank is higher than k;
- *Normalized Discounted Cumulative Gain (nDCG)*: this metric allows to have a *relevance scale* instead of binary relevance judgments (e.g., recommending $r_1(t)$ will produce a higher score than recommending $r_5(t)$ at the same rank in $R_k^{\hat{\mathcal{S}}}(t)$).

4.3. Results and discussion

In Table 2, we compare the systems AT Baseline, TL (with the three mining strategies *Neighbors*, *Random uniform* and *Distance-based*) and TL Autopool. We indicate for each the average and confidence interval at 95% of the four metrics (expressed as percentages).

[2] The same artist cannot be in both the training and test set. However it can appear both as a query and recommendations at test time; which may bias the results.

Model	MAP@20	Recall@20	RR@20	nDCG@20
AT Baseline	4.50 ± 0.34	12.57 ± 0.66	15.62 ± 1.07	11.30 ± 0.69
TL Neighbors	5.58 ± 0.41	12.73 ± 0.67	19.18 ± 1.23	13.41 ± 0.80
TL Random uniform	5.39 ± 0.38	15.01 ± 0.70	17.86 ± 1.12	13.50 ± 0.76
TL Distance-based	**5.98 ± 0.40**	**15.79 ± 0.73**	**19.89 ± 1.19**	**14.41 ± 0.78**
TL Autopool	**7.99 ± 0.51**	**17.74 ± 0.79**	**24.68 ± 1.34**	**17.95 ± 0.92**

Table 2: Comparison of the results of the AT Baseline vs TL systems (with various sampling strategies) vs TL Autopool. Higher is better.

We first observe that the AT baseline is outperformed by all TL systems on all metrics. This shows that in our case, learning the ranking directly is more efficient than learning the tags and applying $\mathcal{S}$ to their estimates. For information, the AT Baseline system (which replicates [6] architecture) achieves a mean-over-tag AUC of 0.79 on its auto-tagging task.

We then see that among the various mining strategies of the TL systems, the Distance-based negative sampling performs best on all metrics. It should be noted that the Distance-based negative sampling was initially proposed by [27] which uses the learned embeddings to compute the distance online. In our case, the distance can be calculated offline since we use the ground truth $\mathcal{S}$ instead of the embeddings for the distance computation. The Neighbors and Random uniform sampling strategies have similar performances, but the first has a better recall while the latter has a better reciprocal rank.

The last row of Table 2 indicates the results of the TL Autopool system. We observe a boost in performances due to the added flexibility of Auto-pooling. This system is able to retrieve one of the top 5 most relevant tracks with almost a probability of 1/5 (Recall@20 = 17.74). It should be noted that in our case, neither the query nor the reference tracks have been seen during training. The first relevant track is on average at rank 5.6 (inverse of RR@20=24.68), among approximately 2,900 test tracks. This makes our system a promising approach to efficient music retrieval in large datasets. Additionally, an informal listening test reveals that some of the recommended tracks, although judged "irrelevant" by our evaluation system, actually share important characteristics with the target.

Reproducibility: Although we cannot distribute our private dataset and its oracle similarity function, to allow reproducibility of our work we provide the architecture and experimental code [3].

5. CONCLUSION AND PERSPECTIVES

We propose here a method to learn a similarity ranking using a triplet loss network and a dataset of reference rankings. We show that using the triplet loss to learn the ranking gives better results than learning the tags used by the ground truth similarity function. This result is consistent across all our metrics. Finally, we show that Auto-pooling, proposed in the framework of audio event detection, also allows improvement in the case of music similarity.

Future works will focus on improving the training efficiency by mining on the fly useful triplets [19] [29] [30].

[3] https://gitlab.com/creaminal/publications/icassp2020-learning-to-rank-music-tracks

6. REFERENCES

[1] Julián Urbano, *Evaluation in Audio Music Similarity*, Ph.D. thesis, Universidad Carlos III de Madrid, 2013.

[2] Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen, "Deep content-based music recommendation," in *Proc. of NIPS*, Lake Tahoe, NV, USA, 2013.

[3] Hung-Chen Chen and Arbee LP Chen, "A music recommendation system based on music data grouping and user interests," in *Proc of ACM CIKM*, Seville , Spain, 2001, pp. 231–238.

[4] Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David M Pennock, "Methods and metrics for cold-start recommendations," in *Proc. of SIGIR*, Tampere, Finland, 2002.

[5] Stephanie Clifford, "Pandora's Long Strange Trip," in *https://www.inc.com/magazine/20071001/pandoras-long-strange-trip.html*. 2007, Inc.com.

[6] Keunwoo Choi, George Fazekas, and Mark Sandler, "Automatic tagging using deep convolutional neural networks," in *Proc. of ISMIR*, Suzhou, China, 2016.

[7] Jean-Julien Aucouturier and Francois Pachet, "Finding Songs That Sound The Same," in *Proc. of IEEE Benelux Workshop on Model based Processing and Coding of Audio*, Leuven, Belgium, 2002.

[8] Brian McFee, Justin Salamon, and Juan Pablo Bello, "Adaptive pooling operators for weakly labeled sound event detection," *IEEE/ACM Transactions on Audio, Speech and Language Processing*, vol. 26, no. 11, pp. 2180–2193, 2018.

[9] Douglas Eck, Paul Lamere, Thierry Bertin-Mahieux, and Stephen Green, "Automatic Generation of Social Tags for Music Recommendation," in *Proc. of NIPS*, Vancouver, BC, Canada., 2008.

[10] Yandre M.G. Costa, Luiz S. Oliveira, and Carlos N. Silla, "An evaluation of Convolutional Neural Networks for music classification using spectrograms," *Applied Soft Computing Journal*, vol. 52, pp. 28–38, 2017.

[11] Sander Dieleman and Benjamin Schrauwen, "End-to-end learning for music audio," in *Proc. of ICASSP*, Florence, Italy, 2014.

[12] Jordi Pons, Oriol Nieto, Matthew Prockup, Erik M. Schmidt, Andreas F. Ehmann, and Xavier Serra, "End-to-end learning for music audio tagging at scale," in *Proc. of ISMIR*, Paris, France, 2018.

[13] Jordi Pons, Olga Slizovskaia, Rong Gong, Emilia Gómez, and Xavier Serra, "Timbre analysis of music audio signals with convolutional neural networks," in *Proc. of EUSIPCO*, Kos, Greece, 2017.

[14] Keunwoo Choi, George Fazekas, and Mark Sandler, "Explaining Deep Convolutional Neural Networks on Music Classification," *arXiv preprint arXiv:1607.02444*, 2016.

[15] Nicolas Usunier, David Buffoni, and Patrick Gallinari, "Ranking with ordered weighted pairwise classification.," in *Proc. of ICML*, Montreal, Canada, 2009.

[16] Kilian Q Weinberger and Lawrence K Saul, "Distance metric learning for large margin nearest neighbor classification," *Journal of Machine Learning Research*, vol. 10, no. Feb, pp. 207–244, 2009.

[17] Elad Hoffer and Nir Ailon, "Deep metric learning using triplet network," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2015, vol. 9370, pp. 84–92, Springer Verlag.

[18] Florian Schroff and James Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering," in *Proc of CVPR*, Boston, MA, USA, 2015.

[19] Alexander Hermans, Lucas Beyer, and Bastian Leibe, "In Defense of the Triplet Loss for Person Re-Identification," *arXiv preprint arXiv:1703.07737*, 2017.

[20] Brian Mcfee and Gert Lanckriet, "Metric Learning to Rank," in *Proc of ICML*, Haifa, Israel, 2010.

[21] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu, "Learning Fine-grained Image Similarity with Deep Ranking," in *Proc of CVPR*, Columbus, OH, USA, 2014.

[22] Daniel Wolff and Tillman Weyde, "Learning music similarity from relative user ratings," *Information Retrieval*, vol. 17, no. 2, pp. 109–136, 2014.

[23] Malcolm Slaney, Kilian Weinberger, and William White, "Learning a Metric for Music Similarity," in *Proc. of ISMIR*, Philadelphia, PA, USA, 2008.

[24] Jason Weston, Samy Bengio, and Philippe Hamel Google, "Large-Scale Music Annotation and Retrieval : Learning to Rank in Joint Semantic Spaces," *arXiv preprint*, 2011.

[25] Rui Lu, Kailun Wu, Zhiyao Duan, and Changshui Zhang, "Deep ranking: Triplet MatchNet for music metric learning," in *Proc. of ICASSP*, New Orleans, LA, USA, 2017.

[26] Edith Law, Kris West, Michael Mandel, and Mert J Bay Stephen Downie, "Evaluation of Algorithms Using Games: The Case of Music Tagging," in *Proc. of ISMIR*, Kobe, Japan, 2009.

[27] R. Manmatha, Chao Yuan Wu, Alexander J. Smola, and Philipp Krahenbuhl, "Sampling Matters in Deep Embedding Learning," in *Proc. of ICCV*, Venice, Italy, 2017.

[28] Sander Dieleman, "Recommending music on Spotify with deep learning," 2014.

[29] Anastasiya Mishchuk, Dmytro Mishkin, Filip Radenovic, and Jiri Matas, "Working hard to know your neighbor's margins: Local descriptor learning loss," in *Proc. of NIPS*, Long Beach, CA, USA, 2017.

[30] Guillaume Doras and Geoffroy Peeters, "Cover Detection Using Dominant Melody Embeddings," in *Proc. of ISMIR*, Delft, The Netherlands, 2019.