Building a framework for self-supervised audio representation learning with Contrastive Predictive Coding

Janosch Bajorath, Mathis Pink, Minseok Kang

March 2021

Abstract

Contrastive predictive coding (CPC) is a framework for unsupervised representation learning. It tries to learn a useful representation of audio features by predicting a future time point within a data sequence. It makes use of negative sampling, i.e. the prediction of a model should be similar to the actual future data but different from dissimilar data. In this project, we re-implement the CPC framework by making use of various sub-architectures. To evaluate trained CPC models we use the extracted features to classify music genres. Additionally, we analyze extracted features directly by TSNE dimensionality reduction.

1 Introduction

The goal of unsupervised representation learning is to learn without supervision a mapping from input data space to a feature space which can be used for other tasks such as classification. Common representation learning models are Autoencoder and Variational Autoencoder. In Contrastive Predictive Coding (CPC) [4], such a mapping is acquired by learning to predict a future time point of a sequence by representing it in latent space instead of reconstructing the model input directly. This assumes features that can predict future data with high precision are useful features. The authors hypothesize the reason for it is that predicted data are conditionally dependent on general features that can be used in many tasks.

CPC is inspired by representation learning models for natural language processing like Word2Vec models, in that it incorporates ideas from noise contrastive estimation (NCE) in its loss function. This means that it's not enough for the predicted latent embeddings of the model to be as similar as possible to the actual future embeddings. At the same time, it should be as different as possible from random negative samples that come from another sequence as well.

In this project, we re-implement a CPC model and explore the usefulness of learned features in the context of music genre classification. We first build a framework that allows us to train CPC models with different components. We then train different CPC models on music data. After that, learned music feature vectors are extracted from the trained CPC model. To examine the learned embeddings, we explore visualizations using a dimension reduction. Lastly we use the extracted features to train a linear classification model. Our main contribution however is that we built a framework that allows to train CPC models on audio data of any kind.

2 Methods

Our experiments consist of three stages. In the first stage, a CPC model is trained to learn useful features from raw audio data. We vary the sub-architectures of CPC and explore three different architectures. Second, we deploy the CPC model to extract useful features. Third, a classifier is trained by making use of extracted features to classify music genres. In this section we first describe the data that we use. We then explain the idea behind CPC and the design choices we make for the specific task of music representation learning.

2.1 Data used for training and evaluation

In this section, we describe data set that we use for the project.

2.1.1 CPC model training

To train the CPC models, we use 8000 audio clips from music given by the Free Music Archive (FMA) data set [2]. Each wav file has a duration of 30 seconds and a sample rate of 22050 Hz. To train the 1D convolutional encoder models, we downsample the raw audio to 4420 Hz in the training data pipeline. For the 2D convolutional Encoder models, we use the librosa library to transform the raw audio into Mel-spectrograms [3] with the frequency divided into 64 Mel bins and a temporal resolution of 480 time-steps per second. The mel spectra are made to achieve square encoder input images (64x64), based on our default CPC training parameters.

2.1.2 Classification model training

To train the classification models, we do not use embeddings obtained from the CPC model that were used for training. Instead we obtain embeddings from CPC for unseen data, namely the GTZAN dataset that has been widely used for music information retrieval [6]. GTZAN consists of 1000 audio clips of 30 seconds duration, labeled into 10 genres (making it 100 wav files per genre).

2.1.3 Classification model evaluation

Since GTZAN samples multiple audio clips from the same songs and there is a lack of metadata that could be used to counter possible inflation on test set performance [5], we decided not to use GTZAN to test the trained classification models for generalization. Instead we create a third dataset for each genre, using 30 second samples from 10 different YouTube containing albums or a compilation of music that belongs to a genre. We provide a document with links and corresponding descriptions on GitHub.

In both our self-created YouTube dataset (used for testing the classifier) and GTZAN (used for training the classifier) there are 10 classes: rock, blues, metal, jazz, classical, country, pop, disco, hiphop and reggae.

Importantly, for all genres except country, music from a single musician/band are used to sample 100 wav files of 30 second duration. For country, a corresponding compilation of country songs was used.

2.2 CPC

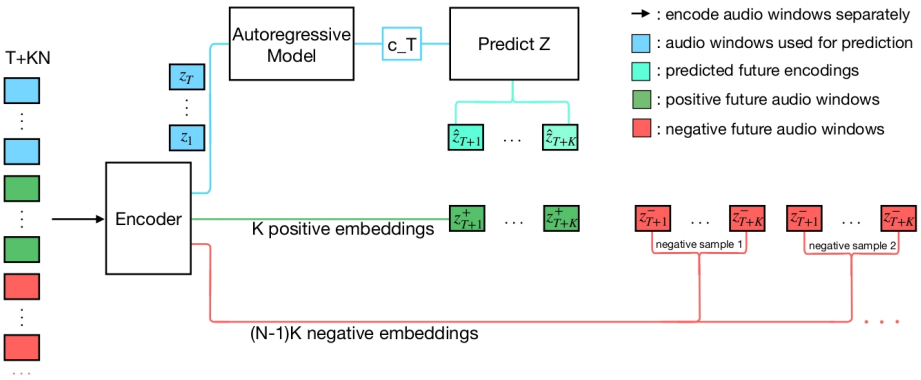


Figure 1: Contrastive Predictive Coding General Architecture

The T audio windows (colored blue in our graphics) are first encoded by an encoder to obtain T embeddings z_1 up to z_T. These embedding vectors are used as sequential input to an autoregressive model that outputs a single vector c_T. This context embedding is what is supposed to contain information about the input sequence that is useful to make predictions about the embeddings of k next audio windows. In other words, the mutual information between c_T and the encoded future time-step z_{T+k} should be maximized. To obtain predictions, c_T must be projected to the same space as z_i (single encoded audio windows). The goal is to have k predictions ($\hat{z}_{T+1}$..., $\hat{z}_{T+k}$) that are most similar to the embeddings of the actual future audio windows (z_{T+1}..., z_{T+k}) and at the same time most dissimilar to embeddings of random audio windows coming from other audio files (called negative samples and represented in red in our graphics). The InfoNCE objective function that is used in CPC expresses this rationale and is

shown by [4] to maximize a lower bound of mutual information between the context embedding and the encoded future audio windows.

2.2.1 Data pipeline

We make the assumption that 4 seconds and only lower frequencies (allowing for the use of downsampling from 22050 Hz to 4410 Hz) are necessary to be able to learn features relevant to music genre classification. This allowed us to build small CPC models that can be trained with fewer than 3GB of VRAM.

Figure 2 illustrates how a single data sample is composed. We take an audio file of length 30s and split it into T+k audio segments of which the k segments are the positive sample the model should predict. We then take N-1 other k segments from other audio files as the negative samples (which should be different from our prediction in the latent space).

Implementation: For raw audio waveform, we first load a number of audio files that matches the batch size. This allows us to only preprocess and load batch size audio files instead of batch size x N files, significantly speeding up the data generation process during training. We split each audio file into T+k segments with T = 27 and k = 3, meaning we take 3.6 seconds of audio to predict the next 400 ms. The choice of parameters were based on the original paper. We then concatenate this with N-1 negative samples of 3 segments, where N is equal to our batch size, which is set to 8. This means that in a batch, each loaded audio is used once as a positive sample and 7 times a 400ms segment is sampled from it to act as a negative sample for another file.

Implementation: To train a CPC model on 2D spectrograms rather than on 1D raw audio, we first convert the audio to corresponding Mel spectrograms. The durations and numbers of segments that we use for raw audio are also used here. To generate the Mel spectrograms we apply a sampling window of 2048 (audio range used to compute the fast fourier transform) and extracted 64 Mel bands (bins of frequencies). To ensure that the encoder input has a square shape of 64 bands x 64 time-steps, the hop length (displacement between sampling window centers for the Fourier transform) is automatically computed based on the full audio length (4s) for the CPC model and the sampling rate of the audio file.

2.2.2 Encoder

Not every detail of x_t is important in order to predict future data, which is why it is helpful to not predict the noisy x_t but rather an encoded version of it, z_t. By mapping the data to a lower dimensional space, only important high-level features remain and low-level noise is removed [4].

Any type of encoder can be used here. The original paper [4] use the CPC audio model to do phoneme and speaker classification, a simple 1-D strided convolution architecture was used, which is also what we do. Audio data is one-dimensional data with graded change in its value along the time dimension, so using convolution filters to exploit this locality is well-justified. In addition

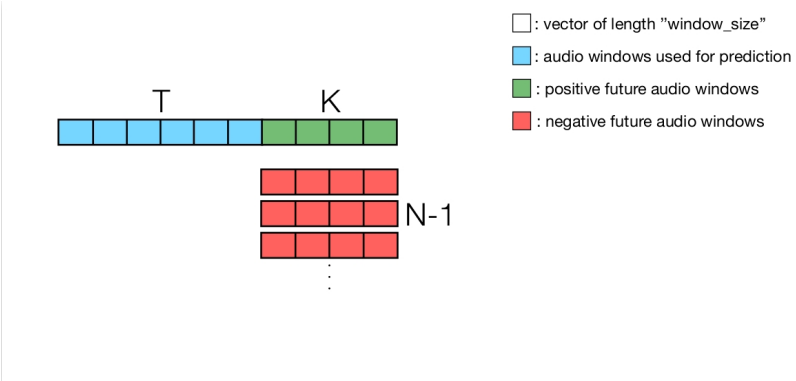


Figure 2: To train a CPC model on raw audio or spectrograms, one audio clip is needed that is split into T audio windows plus k future audio windows that come after T. Additionally a set of N-1 negative/false versions of the k future audio windows are needed. These are typically sampled from other files.

to the convolutional encoder, we probed another common type of encoder for music data by transforming raw audio data into mel-spectrograms as explained in subsubsection 2.1.1. The generated images are then (horizontally split into T+k patches) encoded using a 2-d CNN. Note that parameters of both networks are chosen such that the total number of parameters are similar.

Implementation: The 1D-convolution encoder consists of multiple 1d convolution layers with batch normalization and leaky ReLU non-linearity. We choose leaky ReLU to deal with vanishing gradients. After the convolutional layers, a flatten layer is followed by two stacks of dropout with a rate of 0.1 and fully connected layers. Table 1 denotes parameters used for the architecture. The complete encoder has 5,654,784 parameters.

dim. output	stride sizes	kernel sizes	num. filters	activation
256	[5,4,2,2,2]	[10,8,4,4,4]	[512,512,512,512,512]	leaky ReLU

Table 1: Default attributes used for constructing the 1D-convolution encoder.

Implementation: The 2D-convolution encoder for Mel spectrograms consists of several convolution blocks, each comprising a 2D convolution layer, followed by 2D spatial Dropout and a non-linearity (leaky ReLU). After the convolution blocks the feature maps are flattened, followed by one densely connected block with leaky ReLU nonlinearity and Dropout added. An output dense layer with leaky ReLU outputs the embedding (z_t). Hyperparameters used for the specific layers are shown in Table 2. The entire encoder comprises 5,672,832 weights.

dim. output	stride sizes	kernel sizes	num. filters	activation
256	[2,2,2,2]	[3,3,3,3]	[32,64,256,512]	leaky ReLU

Table 2: Default attributes used for constructing the 2D-convolution encoder.

2.2.3 Autoregressive model

Similar to how it's possible to use any encoder model in CPC, any type of auto-regressive model that can predict a next time-step from a sequence can be used. By default the authors of the original paper used a Gated Recurrent Unit RNN [4] that receives as an input a defined window of a continuous sequence and returns only the last hidden state. In an alternative CPC model, we used the encoder part of a transformer [8] as the auto-regressive model. Since the original encoder of the transformer model provides element-wise context vectors, we use additional fully connected layers to combine them into a single context vector. Also here, number of parameters in two types of model were kept similar.

Implementation: The GRU auto-regressive model has c_{dim}= 512 as its output dimension.

Implementation: The Attention-based auto-regressive model implementation is a modified version of **tensorflow tutorial**. The embedding layer is removed as we embed the raw audio windows with the encoder. At the end, stacks of dropout and fully connected layers are added on top to combine context of individual inputs to a single vector. Table 3 denotes parameters used for the architecture with naming conventions following the convention used in the tutorial. The model has in total 1,374,508 weights, similar to the GRU.

num. enc. layers	num. heads	z_dim	dff
2	2	256	100
num. dense units	activation	max. pos. encoding	drop rate
[100, 512]	leaky ReLU	T+K	0.1

Table 3: Default attributes used for constructing the attention-based auto-regressive model.

2.2.4 Prediction

The context vector can be thought of as a compact representation of all data until the current time point $x_{\leq T}$. Therefore, we try to predict the future data given the context by modelling $p(x_{T+k}|c_T)$ [4]. However, we again have a problem that x_{T+k} is given in a high-dimensional space. x_{T+k} not only contains actual features but also low-level noise. This means that a lot of our capacity of our prediction model would be wasted on fitting such noise. Therefore, the

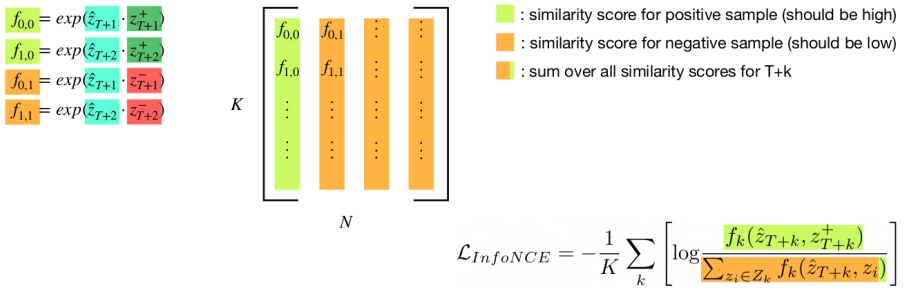


Figure 3: Illustration of how the loss is computed. The model returns a matrix with similarity scores for all possible combinations of K and N. The loss would be minimal if the similarity score between predictions and positive samples are maximized and vice versa for negative samples.

authors of CPC opt for a compact representation which compresses x_{T+k} and c_T into a single vector. Since c_T however is given in a space that is of a different dimensionality from z_{T+k}, we need to project it. [4] use k linear projection layers defined by weight matrices W_k for each time step to predict into the future. For each time step, another W_k is learned to transform c_T into a prediction.

2.2.5 Similarity measure and InfoNCE Loss

Given a predicted representation of a future audio segment and the actual encoded future audio window, we want to test how similar they are. We also want to measure how similar the prediction is to a number of random unrelated encoded audio windows. [4] use the exponential of the dot product between predicted embedding and target embedding as the chosen similarity measure which we also adopt. However as the authors state, any other similarity measure can in principle be used. The similarity quantity is called an f-score. [4] demonstrate that simultaneously predicting multiple steps into the future gives better representations compared to k = 1.

$$f_k(z_{T+k}, \hat{z}_{T+k}) = \exp({z_{T+k}}^{\mathrm{T}} W_k c_T) = \exp({z_{T+k}}^{\mathrm{T}} \hat{z}_{T+k}) \tag{1}$$

Implementation: Figure 3 illustrates how the loss is computed. Since computing InfoNCE requires computing the similarity function NK times, it is implemented in a parallel fashion, resulting in a matrix of shape (K, N) in which the first column represents the f-scores for the positive sample and all other columns represent f-scores for negative samples.

2.2.6 Training

To train CPC we use the Adam optimizer with a learning rate of 1e-5 with batches of 8, one positive sample and 7 negative samples.

2.3 Analysis of learned features

As training of CPC model takes a long time, we save the extracted features using learned CPC in separate files. The parameter num_em_samples_per_data decides how often we take a cut of length $4sec$ from a single data to generate multiple embeddings for each audio file. This effectively has an effect similar to data augmentation because we are taking gradually shifted inputs with increased numbers.

To directly analyze the quality of features found by CPC, we visualize the high-dimensional features found by CPC in a 2D-space. The dimension is reduced by using t-distributed stochastic neighbor embedding (t-SNE) [7]. In short, t-SNE finds a subspace in which the distance relationship in the original space is maximally preserved. We take random 5000 feature data from train and test data each and map them to a subspace. The projected data is then visualized in two different ways. First, for each train and test data, 10 different genres are marked in different colors. We expect each genre to form a distinct cluster if learned features are useful for classification. Second, for each genre, we mark train and test data in two different colors. if clusters formed by train and test data are far apart from each other, this means that the CPC model cannot generalize over different music within the same genre. We thus use these figures to qualitatively evaluate over-fitting of CPC model.

2.3.1 Classifier architecture

We use a linear classifier with a softmax output layer with a dropout layer in-between input and output. Like [1] we do not opt for a complicated classifier architecture for two reasons. First, if learned features from the CPC model prove to be useful for music classification, those features should be sufficient to classify music without complicated feature expansions inside the classifier. Second, by using more complicated classifier, we run into a risk of over-fitting. However, we assumed that c_{dim}=512 is too large, meaning that even a simple linear model could over-fit to noise. Therefore, we use dimensionality-reduced features as input to the network. To this end, an auto-encoder is trained on whole embeddings. Afterwards, the dimension of features are first reduced by using auto-encoders, which are then fed into the classifier.

Implementation: Our auto-encoder has multiple stacks of fully-connected layers and dropout layers. Table 4 denotes parameter values used for the architecture.

num. dense units	dropout rate	activation	reduced dim
[256, 256, 128, 64, 32, 64, 128, 256]	0.1	leaky ReLU	32

Table 4: Default attributes used for constructing the auto-encoder.

2.3.2 Training and testing classifier

To train the models we defined default parameters, see Table 5. In order to quantify the classification performance, we use the typical supervised learning scheme for classification. The categorical cross entropy for the test data should be maximized as we are using a softmax classification layer. The difference between loss of train and test data can be used to evaluate how much the classifier over-fits.

epochs	learning rate	batch size	embeddings train	embeddings test
10	0.0001	32	90,000	5,000

Table 5: Default attributes used for training the autoencoder and classifier.

Additionally, we analyze the classifier by counting how often each label would be classified as another classes. This results in a matrix in which the row stands for different labels and the column stands for different predictions. We normalize the count such that the sum of the matrix equals 1. The sum over the main diagonal would then be the accuracy of the classifier. The sum over rows (sweep from left to right) would quantify how often each class was entailed in data and should ideally be uniformly distributed. The sum over columns (sweep from bottom to top) would quantify how often each class was predicted and tells us whether the model is biased to prefer certain class.

3 Results

3.1 Loss of CPC

Figure 4 illustrates the loss of CPC model for three different architectures. Other than that the loss for the 2D-convolutionl encoder is already at the start, we couldn't observe any significant difference between architectures.

3.2 Loss and accuracy of classifier

Figure 5 illustrates that no significant difference in test loss and accuracy can be found between three different architectures with the 2D-Conv-GRU performing worst and the 1D-Conv-AttentionNet giving the best results with a validation accuracy of roughly 28%. Randomly initializing the best CPC network and

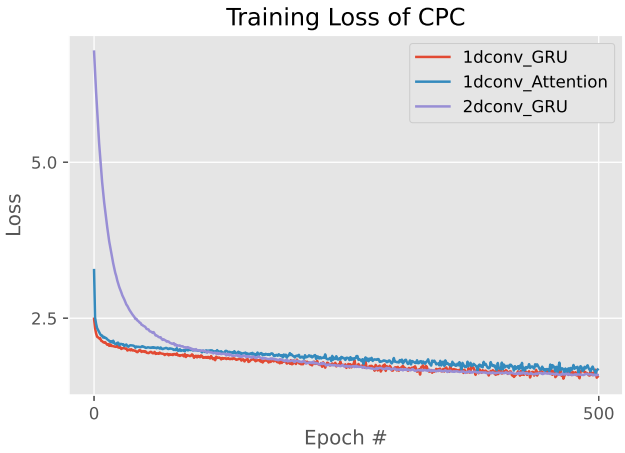


Figure 4: Loss of CPC for three architecture.

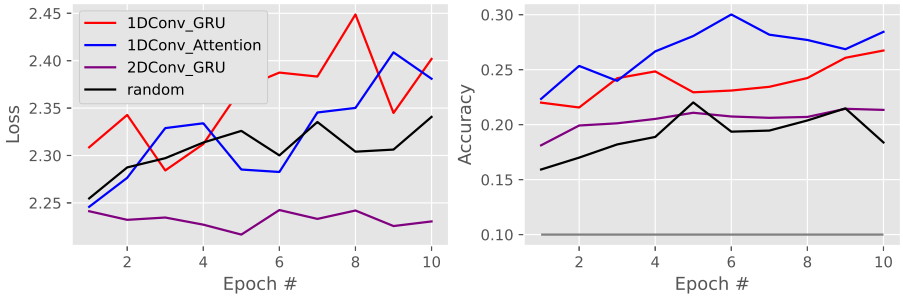


Figure 5: Test loss and accuracy for three different architectures and random initialized CPC. The baseline accuracy is random performance (=0.1).

directly obtaining embeddings to train the classifier still lead to an accuracy of 20%. Refer to subsection A.1 for further results showing loss and accuracy for individual architectures and for autoencoders.

3.3 Confusion matrix and tSNE analysis

We analyze confusion matrices and tSNE plots separately for each model. We only present some of the results here. Full plots for all genres and train/test split can be found in subsection A.2. Figure 6 depicts the confusion matrix for the embeddings of the 1Dconv GRU model. The model learned to predict metal and jazz music relatively good with value of 0.054 and 0.056 where value of 0.1 would be perfect performance given uniform distributed data. Otherwise, hiphop music was often classified as reggae. Figure 10 illustrates learned features projected onto 2D space. The train and test clusters of metal and jazz music

do not form closer clusters as we would expect. However, we can see that the train cluster of hiphop music finds itself in a similar space as the test cluster of reggae music, which can explain the high mis-classification of hiphop music as reggae music.

Next, we analyze the result of the 1DConv Attention model (Figure 7). We note that the classification accuracy for jazz and hiphop is relatively high with values around 0.059 and 0.052 respectively. Reggae music in the YouTube dataset is often misclassified as pop. Figure 11 shows t-SNE analysis of these four genres. Jazz music form relatively tight clusters that are close for both train and test data. However, hiphop music clusters are rather far away from each other. We also couldn't see any signature of association between reggae and pop music.

Finally, we present the result for 2Dconv GRU model. Figure 8 shows that the model has learned to classify classical music well with frequency about 0.05. At the same time, the model almost never predicts any music to be rock or pop. Figure 12 however doesn't provide a good explanation for why classical music is classified so well as we don't see any specific clusters formed. Same tendency can be found in pop music as well, which might explain why pop music is almost never predicted due to its small specificity.

We also trained a classifier with features that come from a untrained CPC model. Figure 9 shows that the classifier predicts almost all genres of music are most often as jazz, which presumably depends on dominant weight bias towards jazz unit at initialization. We note that jazz music is most often classified as classical music. Figure 13 shows projected embeddings for these two genres. We do not recognize any particular structures for these two genres. However, we note that embeddings for all genres seems to form two distinct clusters, which again must depend on initialization.

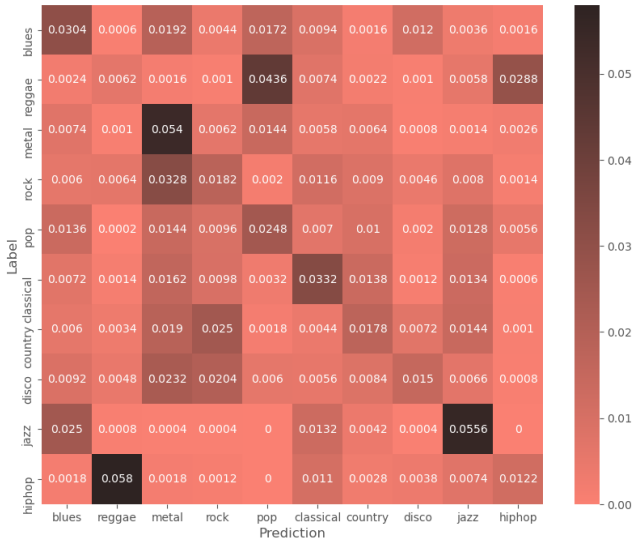


Figure 6: Confusion matrix of 1Dconv GRU model.

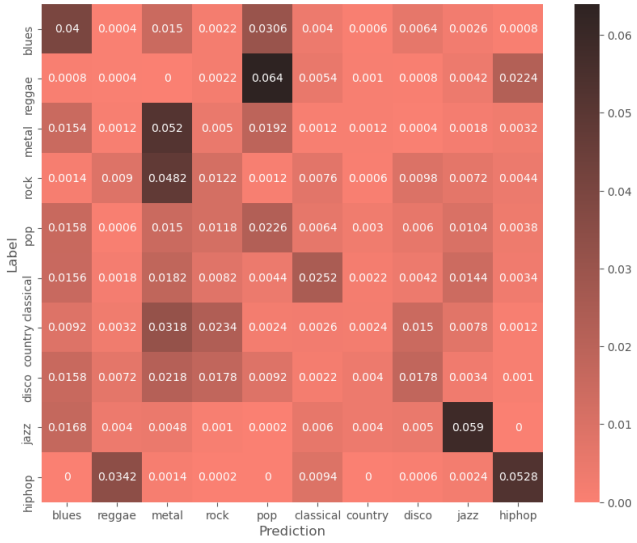


Figure 7: Confusion matrix of 1Dconv Attention model.

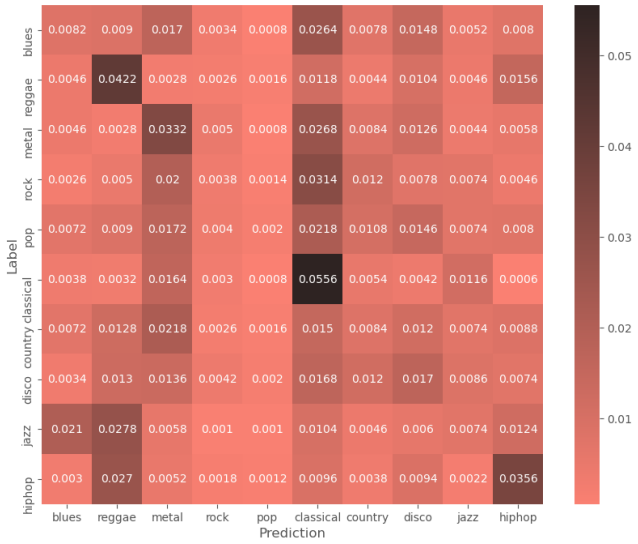


Figure 8: Confusion matrix of 2Dconv GRU model.

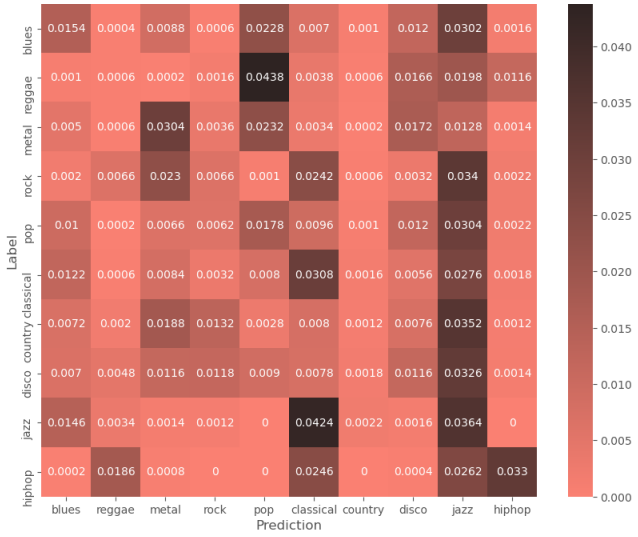


Figure 9: Confusion matrix of untrained CPC model.

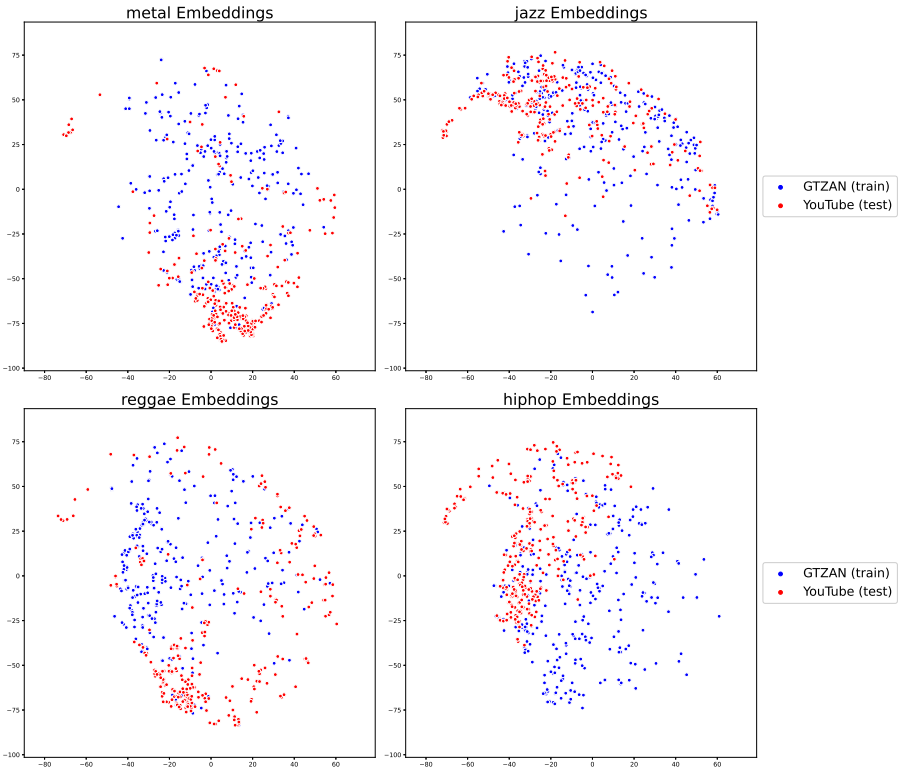


Figure 10: Illustration of learned features by 1dConv GRU model for four different genres.

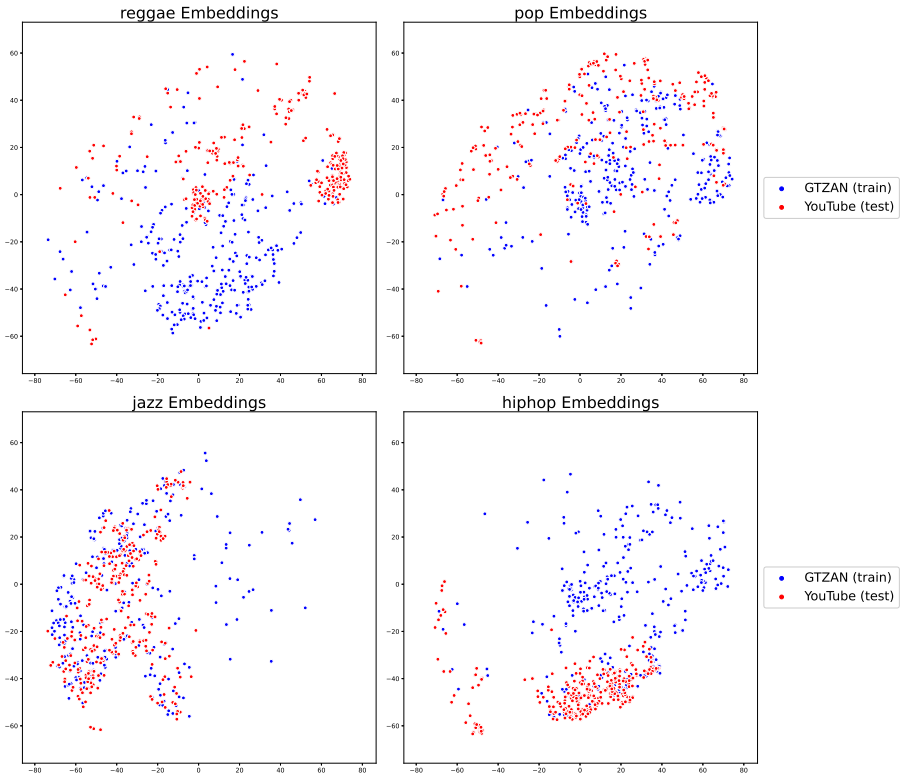


Figure 11: Illustration of learned features by 1dConv Attention model for four different genres.

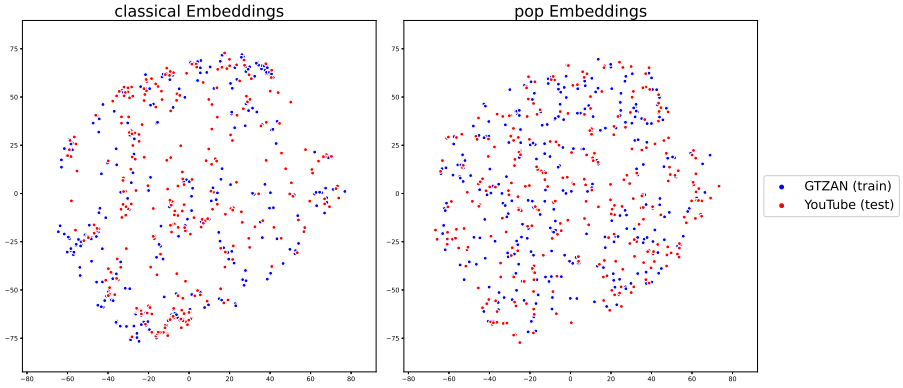


Figure 12: Illustration of learned features by 2dConv GRU model for four different genres.

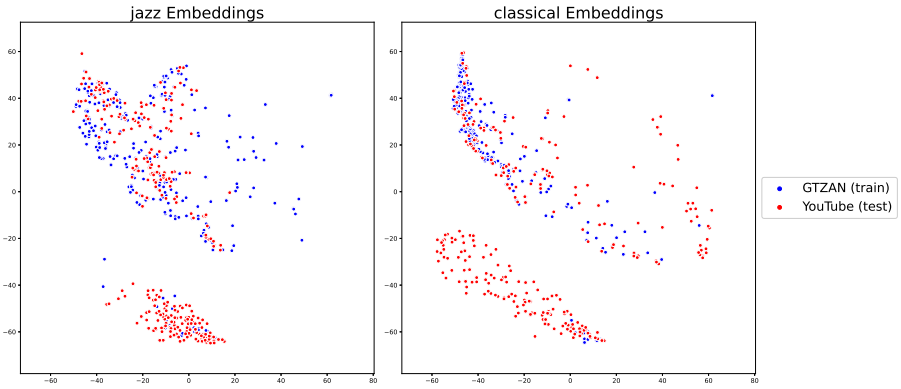


Figure 13: Illustration of learned features by 2dConv GRU model for four different genres.

4 Discussions

Before we discuss the result of our project, we would like to criticize few aspects of the original paper. We cite from chapter 3.1. of the paper:

"... on the 16KHz PCM audio waveform. We use five convolutional layers with strides [5, 4, 2, 2, 2], filter-sizes [10, 8, 4, 4, 4] and 512 hidden units with ReLU activations. ...so that there is a feature vector for every 10ms of speech, ..."

This means that a single window has a size of 160. However, following the given stride and kernel sizes, the output at the end would have a negative output size. Additionally, they didn't explain the number of filters used for each layer.

Second problem with the paper is that the paper is ambiguous about how the InfoNCE loss is computed in practice. Equation 2 reprints the loss defined in the paper. Since f_k is defined for every future time step k, one needs to combine k different f_k to get a scalar loss value. However, the loss defined in the paper only describes how averaging over different samples are done and not over different future time steps. We assumed that it is natural to take weigh future time steps equally. But one could imagine weighted average of different future time steps with more weights for future time steps can be beneficial, which is why we leave it as an option in our loss function to linearly weight the immediate future higher than the far future.

$$\mathcal{L}_{InfoNCE} = -\mathbf{E}_X \left[\log \frac{f_k(\hat{x}_{t+k}, c_t)}{\sum_{x_j \in X} f_k(x_j, c_t)} \right] \tag{2}$$

Since our classifier is trained on features from data which was not used to train the CPC model, we cannot distinguish the cause of over-fitting. In order to do so, we propose the following training scheme. Dataset A is used to train a CPC model. Afterwards, two distinct classifiers with the same architecture are

trained once using dataset A and once using dataset B. Finally, the classifier trained with dataset B is tested using dataset C. $\mathcal{L}_B - \mathcal{L}_A$ then quantifies how much the CPC model over-fits to the dataset A indirectly as the CPC model was trained on dataset A. $\mathcal{L}_C - \mathcal{L}_B$ quantifies the over-fitting of classifier as in standard supervised learning scheme. We couldn't implement such a training-test scheme as our dataset A (fma data in our case) had a different number of classes then dataset B (GTZAN data).

The fact that audio from our YouTube dataset for one genre largely comes from the same artist (except for country) might explain why there appear to be more clusters in the corresponding t-SNE dimension reduced scatter plots.

5 Conclusion

While we did not obtain any outstanding results with the data and model architectures used, we do provide a framework which can be used to train custom CPC models on any folder of audio files and perform further experiments. We do want to stress the importance of data quality and choice of data generation hyperparameters. Listening to tracks from FMA lead us to suspect that better data might lead to better results. We do provide example instructions on how we would suggest to use our framework to train a differently customized CPC model on speech recognition in a tutorial document on GitHub.

References

[1] Kyunghyun Cho et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: 1406.1078 [cs.CL].

[2] Michaël Defferrard et al. "FMA: A Dataset for Music Analysis". In: *18th International Society for Music Information Retrieval Conference (ISMIR)*. 2017. arXiv: 1612.01840. URL: https://arxiv.org/abs/1612.01840.

[3] Mark Sandler Keunwoo Choi George Fazekas. *Automatic tagging using deep convolutional neural networks*. arXiv: 1606.00298 [cs.SD].

[4] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. *Representation Learning with Contrastive Predictive Coding*. 2019. arXiv: 1807.03748 [cs.LG].

[5] Bob L. Sturm. "The State of the Art Ten Years After a State of the Art: Future Research in Music Information Retrieval". In: *Journal of New Music Research* 43.2 (Apr. 2014), pp. 147–172. ISSN: 1744-5027. DOI: 10.1080/09298215.2014.894533. URL: http://dx.doi.org/10.1080/09298215.2014.894533.

[6] G. Tzanetakis and P. Cook. "Musical genre classification of audio signals". In: *IEEE Transactions on Speech and Audio Processing* 10.5 (2002), pp. 293–302. DOI: 10.1109/TSA.2002.800560.

[7] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.11 (2008).

[8] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

A Appendix

A.1 Classifier loss / accuracy

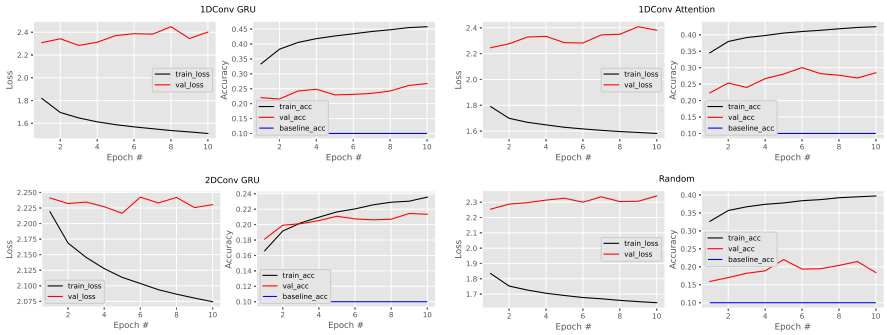


Figure 14: Train and test loss / accuracy for three different architectures. Architectures are 1DConv GRU, 1DConv Attention-based, 2DConv GRU and random CPC from left to right then from top to bottom.

A.2 tSNE analysis

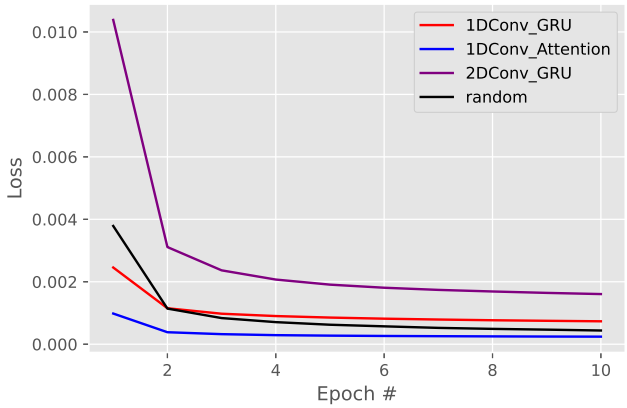
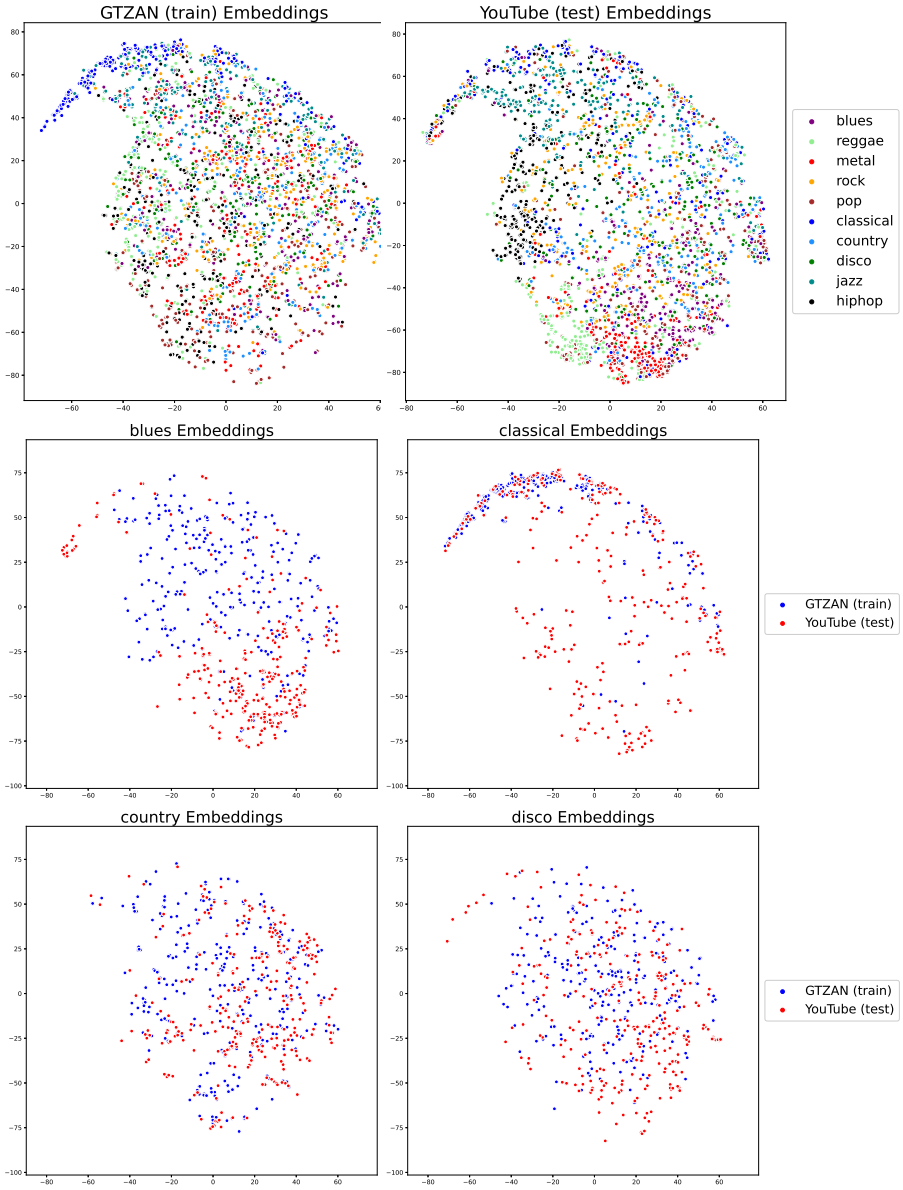


Figure 15: Train loss for autoencoders used for dimensionality reduction.


GTZAN (train) Embeddings
YouTube (test) Embeddings
blues
reggae
metal
rock
pop
classical
country
disco
jazz
hiphop
blues Embeddings
classical Embeddings
GTZAN (train)
YouTube (test)
country Embeddings
disco Embeddings
GTZAN (train)
YouTube (test)

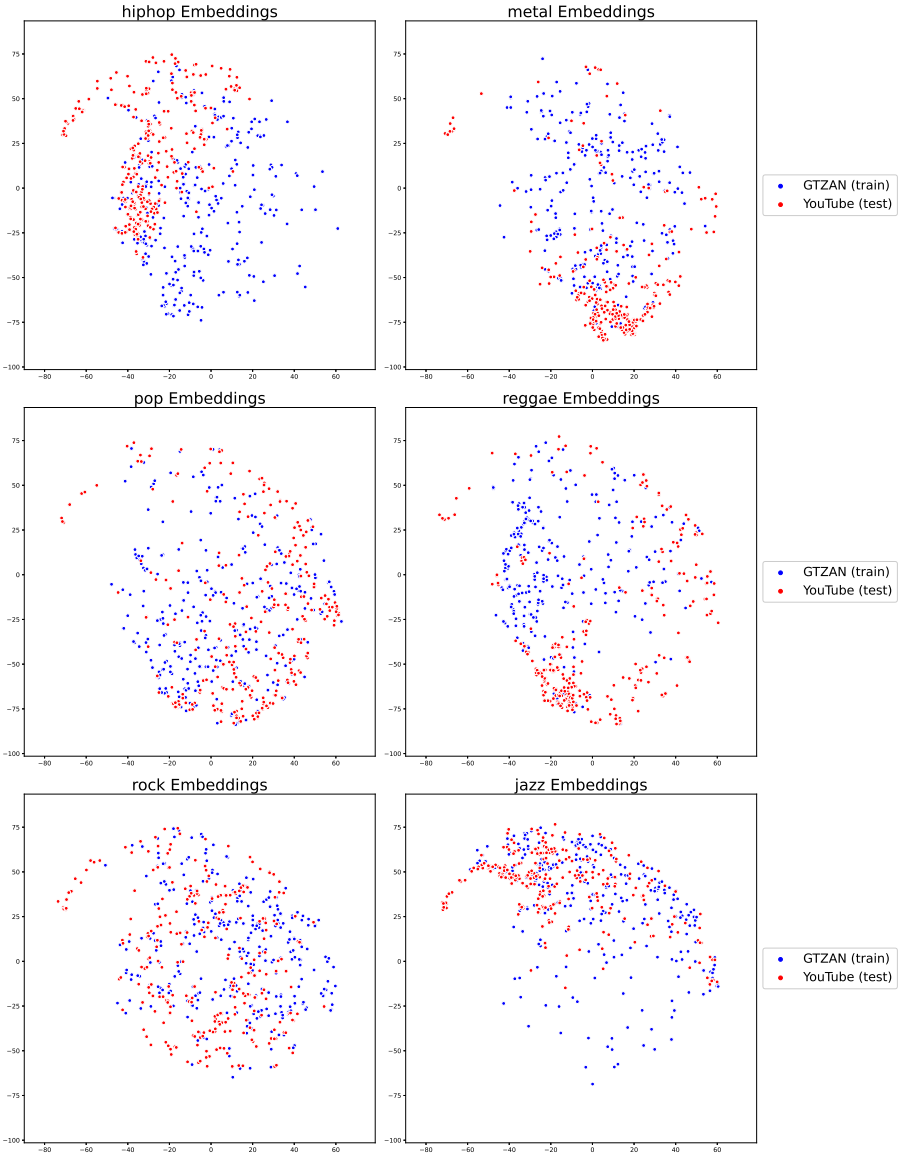
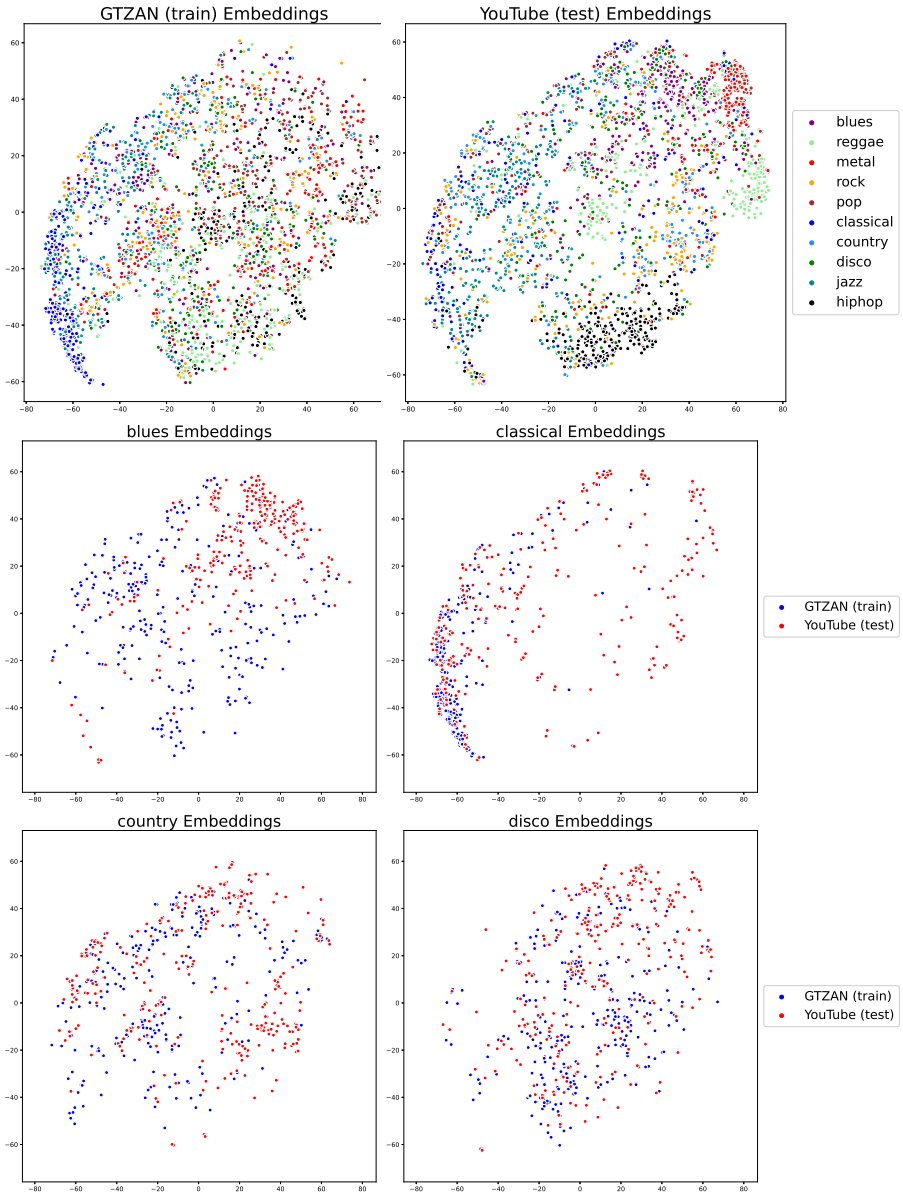


Figure 17: Illustration of learned features by 1dConv GRU model for train/test split and 10 different genres.


GTZAN (train) Embeddings
YouTube (test) Embeddings
blues
reggae
metal
rock
pop
classical
country
disco
jazz
hiphop
blues Embeddings
classical Embeddings
GTZAN (train)
YouTube (test)
country Embeddings
disco Embeddings
GTZAN (train)
YouTube (test)

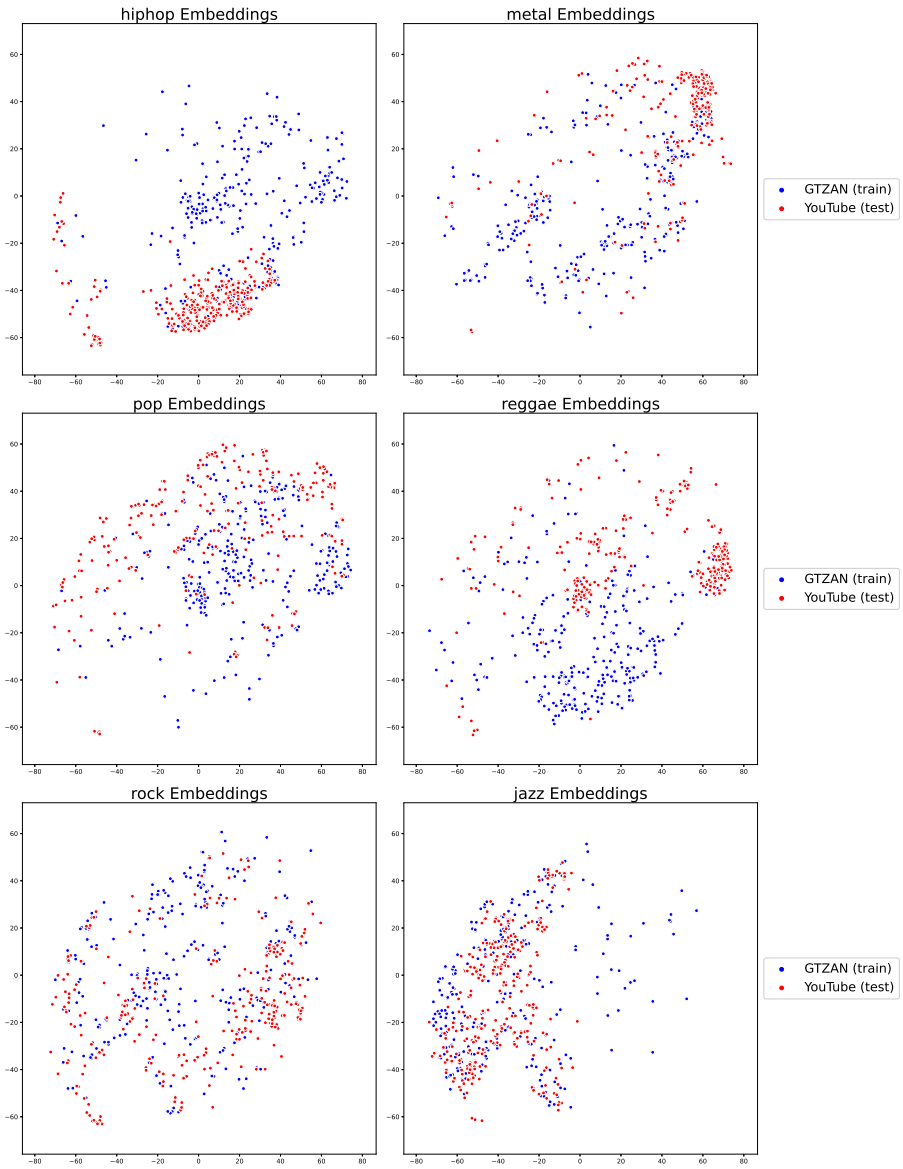
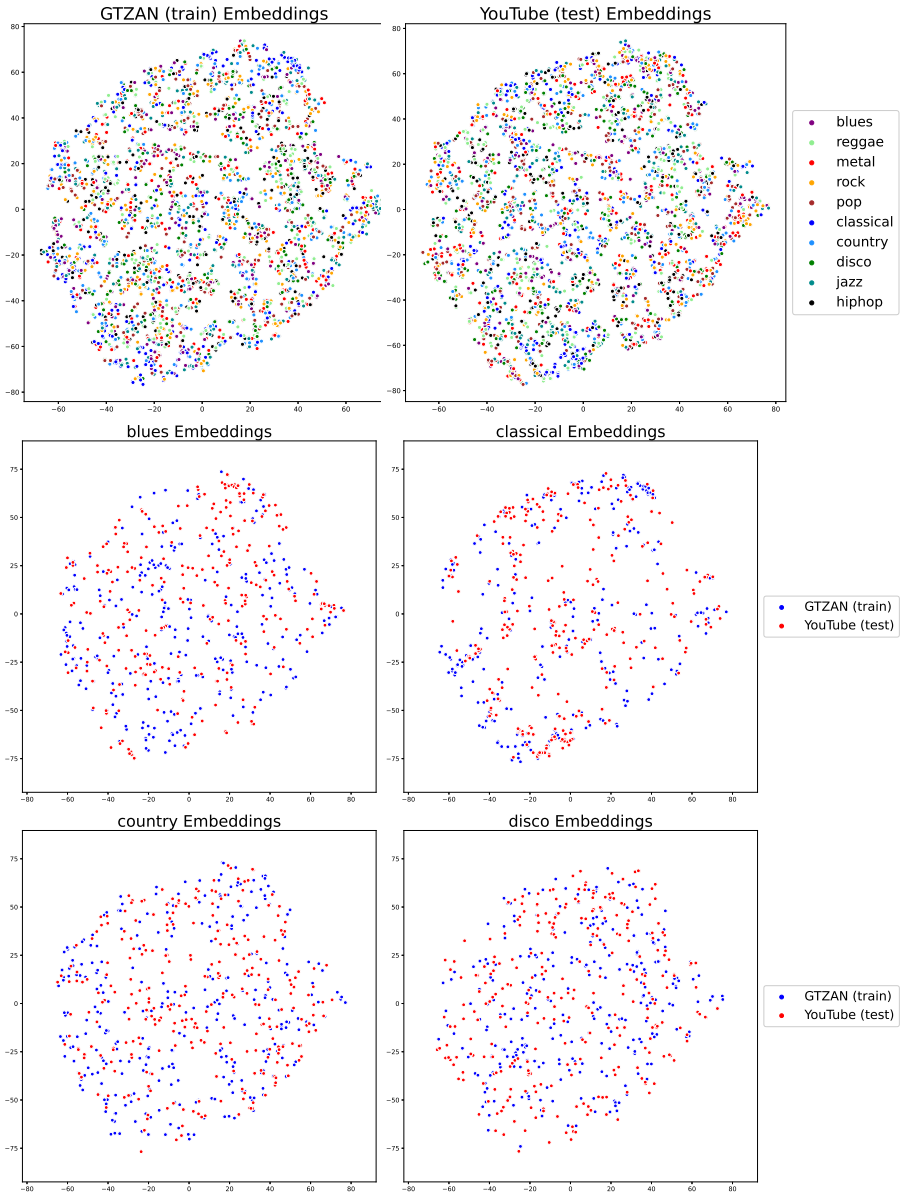


Figure 19: Illustration of learned features by 1dConv Attention model for train/test split and 10 different genres.


GTZAN (train) Embeddings
YouTube (test) Embeddings
blues
reggae
metal
rock
pop
classical
country
disco
jazz
hiphop
blues Embeddings
classical Embeddings
GTZAN (train)
YouTube (test)
country Embeddings
disco Embeddings
GTZAN (train)
YouTube (test)

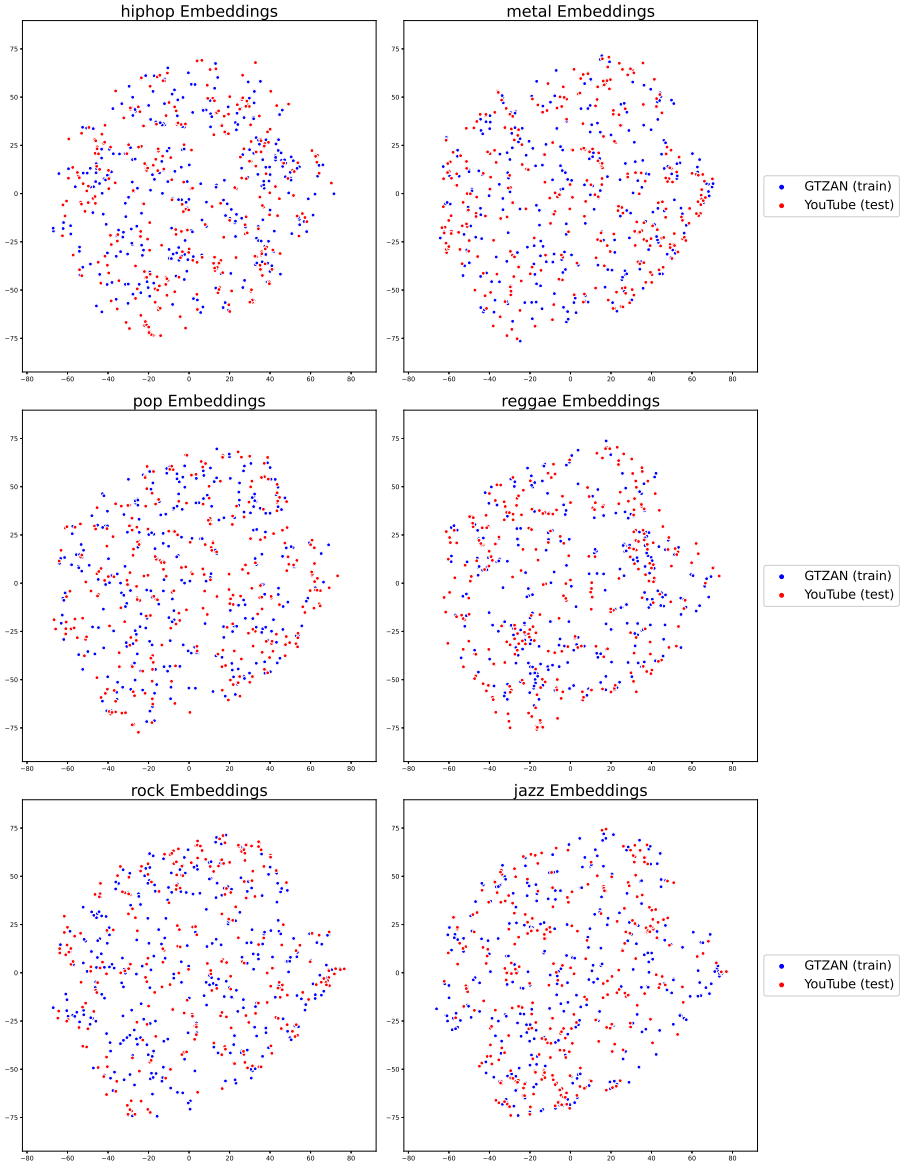


Figure 21: Illustration of learned features by 2dConv GRU model for train/test split and 10 different genres.