(Fall2021) GCT634 Homework 2

20214591 정윤진

Summary

In this homework, I implemented the given model of question 1 and the given metric of question 3, which are explained in section 1. I implemented and performed related experiments to verify the improvement suggestion for the multi-label classification of question 2 and the metric learning of question 4. I applied data augmentation, multi-channel mel-spectograms and practical searching for hyperparameters as common strategies for the two improved models. Further, musically motivated CNN, residual connection and focal loss function are suggested for the multi-label classification. In metric learning problem, I mainly focused on building a triplet data sampling strategy and suggested classification-based triplet sampling. Through the suggested approaches, the final improved models result in 0.86% roc accuracy and ({'R@1': 0.47634560692813116, 'R@2': 0.6166209464753155, 'R@4': 0.7485045429220182, 'R@8': 0.8455286622762342}) recall for each task. The approaches and detailed methods for the improvement are explained in section 2. Remained discussion and future study are summarized in section 3.

Index

1. Given Model Implements

1.1 2D CNN for Multi-Label Classification (Q1)

As the CNN layer's output size, kernel size and pooling size are given, the question could be simply implemented by using Conv2d module from Pytorch. Within the skeleton Conv_2d class, I put convolutional layers which follow the given output channels and kernel size. At this point, because the kernel size is fixed already, the output size is determined by stride, padding size and pooling layer. However, in max pooling, the stride size is generally as same as the kernel size, as also described in the Pytorch official document of MaxPool2d. So then, after applying the given pooling size as kernel and stride for max pooling layers, it seems that the output sizes between the given layers match exactly. Note that Conv2d and MaxPool2d output sizes can be calculated by a given parameter value, as specified in Figure 1.

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Figure 1. Output Size Equation of Conv2d and MaxPool2d

Since the output sizes already match perfectly with the pooling layers, the size should not be changed by the convolutional layers. Therefore, following the equation again, it can be found that the stride of the convolutional layers becomes 1 and the padding size becomes 1 as well. As the result of the implementation of the given architecture, 0.77% roc accuracy is obtained.

1.2 Evaluation Metric (Q3)

```python
def multilabel_recall(self, sim_matrix, binary_labels, top_k):
    # =======================
    ## To-do
    data_num = len(sim_matrix)
    recall = 0
    for query_idx in range(data_num):
        query = sim_matrix[query_idx]
        sorted_idx = np.argsort(query)[::-1][1:top_k+1]
        y_q = binary_labels[query_idx]
        y_i = np.logical_or.reduce(([binary_labels[pred_idx] for pred_idx in sorted_idx]))
        recall += (np.sum(np.logical_and(y_q, y_i)) / np.sum(y_q))
    recall /= data_num
    return recall
    # =======================
```

Figure 2. Implementation of multilabel recall function

To convert the given equation to rather intuitive code, I exploited the logical functions from Numpy library. First, by using np.argsort function and basic indexing, the top k number of predicted indexes are obtained from the similarity vector of the query sample. This k length of the list, which is denoted as *sorted_idx* in code, is an index list of samples predicted to be most similar to the query sample. With this similar samples list, we can find each sample's binary vectors that represent their tags information and by conducting a logical OR function between them the total predicted tags binary vector can be obtained, which is denoted as y_i. Finally, a logical AND function is applied to find the correct prediction. The rest of the calculation is followed in code. As a result, multi-label recall is obtained as follows as figure 3.

```
{'R@1': 0.3781655237965914,
 'R@2': 0.5136119543886533,
 'R@4': 0.6420729755681207,
 'R@8': 0.7549084410734885}
```

Figure 3. Result of multi-label recall with the given Linear Projection.

2. Model Improvement Approach

2.1 Common Approach

2.1.1 Data Augmentation

To improve model learning, we always should consider overfitting first. There are a lot of methods to prevent overfitting, such as regularization or normalization, and most of them are designed to decrease the bias of the model. These strategies have already been implemented in the given code, by weight decay value and bathNorm2d function.

On the other hand, Data Augmentation is a popular strategy that further expand the space of training data to prevent overfitting problem. By twisting the data slightly, it results in an increasing number of the data points and the area of the learning distribution as well.

The given dataset has a total of 9074 data points, which is an extremely small number compared to the other typical dataset such as *ImageNet*. This can cause overfitting and in fact, it seems that the valid accuracy grows gradually at a certain point of training while training accuracy decreases in the given dataset.

Therefore, I decided to try applying data augmentation, using three popular techniques for

audio augmentation: white noising, shifting frames, and stretching. By simply adding some noise to the data with random values, shifting audio frames in a bit and stretching data with respect to the time axis with 2 different stretch rates, I could obtain augmented new data. You can find the applied augmentations in the submitted notebook. Figure 4 shows augmented waveforms of an example sample.

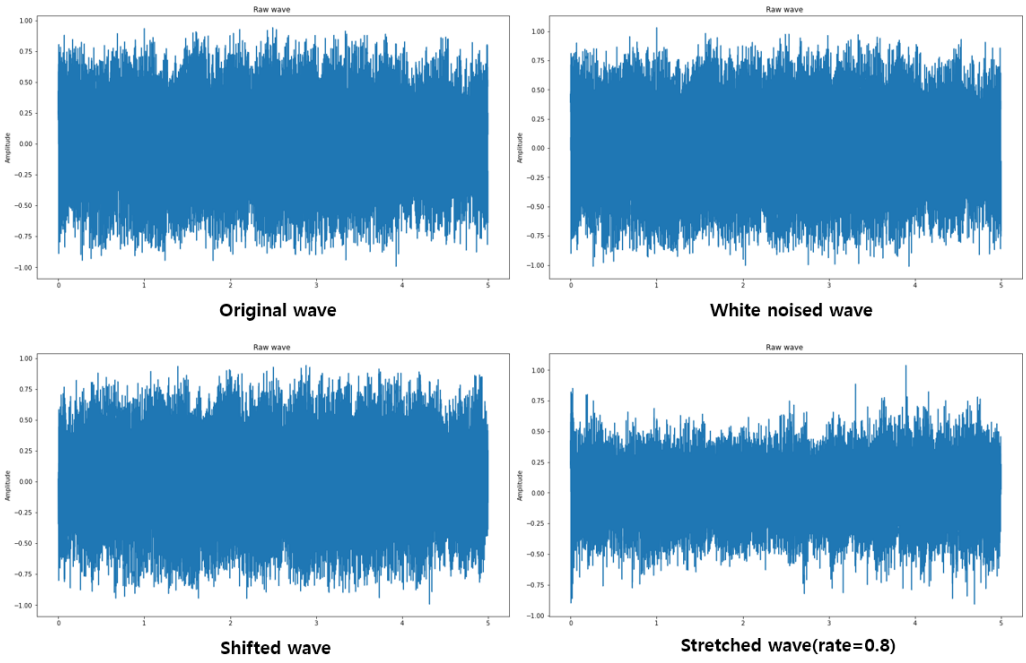


Figure 4. Waveforms of augmented data with 3 different methods.

Samples that were randomly sampled from the original in half are used for each augmentation method. As a result, the total of training data has increased from 6206 to 18618, which is about 3 times of.

I applied these augmented data to the given baseline for the multi-label classification in order to verify the improvement. It was confirmed that the model accuracy increases significantly from 0.64% to 0.71% without changing any of the hyperparameters or model structure. Therefore, I used the augmentation strategy on the following two improved models of question 2 and 4.

2.1.2 Multi-Channel Log Mel-Spectogram

In the given skeleton code, the model uses a mel-spectogram from the input waveform directly. Using a mel-spectogram, which map the frequency onto the mel-scale, is one of the

appropriate ways for capturing audio features as we confirmed in the previous homework. It also performs better than other representations such as MFCCs or Gammatone-Spectrogram in the given classification task. However, using only one mel-spectogram per waveform means watching the waveform with only one perspective as using a pair of hyperparameters such as window size and hop size. Can't we exploit the mel-spectogram more usefully in one waveform? Motivated by this question, I took the idea from image-based CNN models' input channel dimension, which is 3 (R, G, B color values), gives more information than mono channel of mel-spectogram in the same size. The idea is that, by using various hyperparameters, several mel-spectogram with different perspectives can be obtained and are used as one input data, like color dimensions. Therefore, I first created 3 different mel-spectograms with 3 pairs of different window sizes and hop sizes from a waveform, then concatenated them. Because the output size of each is not same, I used an interpolate function to reshape their size equivalent.

While searching to find some proper window and hop size for musical features, I found a paper with a similar approach[1]. Referring to this paper, I applied a proportional pair of window and hop sizes respectively and confirmed it works well. Consequently, for a batch, I reconstructed the (16x3x128x101) size of mel-spectrogram based data as an input.

```
def extract_spectograms(x,
                        sampling_rate=16000,
                        n_fft=512):
    num_channels = 3
    window_sizes = [10, 20, 30]
    hop_sizes = [10, 20, 30]

    specs = []
    for i in range(num_channels):
        window_legth = int(round(window_sizes[i]*sampling_rate/1000))
        hop_length = int (round(hop_sizes[i]*sampling_rate/1000))

        spec = torchaudio.transforms.MelSpectrogram(sample_rate=sampling_rate,
                                                    n_fft=n_fft,
                                                    win_length=window_legth,
                                                    hop_length=hop_length)(x)
        eps = 1e-6 # prevent inf
        spec = torch.log(spec + eps)
        spec = torch.nn.functional.interpolate(spec, size=(128, 101))
        specs.append(spec)

    return torch.cat(specs, dim=1)
```

Figure 5. Implementation of multi-channel log mel-spectogram

The method, which I named as multi-channel log mel-spectogram, is applied to the given

[1] K. Palanisamy, D. Singhania, and A. Yao, "Rethinking cnn models for audio classification," arXiv preprint arXiv:2007.11154, 2020

vanilla CNN2D model from the question 1, except for the final linear layer(layer4)'s max pooling size which is slightly changed because of the output size. With this strategy, the given CNN2D's accuracy increased from 0.76% to 0.78% in 10 epochs. The improvement was not that significant as I expected, however, constructing multi-channel mel-spectogram of various perspectives still makes a positive effect on training so that I decided to apply this strategy for the improvement mission of Question 2 and 4 in common.

2.1.3 Hyperparameter Details

After some practical experiments for optimizers, I found that Adam optimizer fitted with the models. Adam optimizer is kind of mixture method of RMSProp and Momentum, which are related to techniques such as decaying average of squared gradients or adaptive learning rate.

Also, regularization is applied by using weight decay value to prevent overfitting problem and optimal learning rate has found practically.

2.2 Approach for Multi-Label Classification (Q2)

Multi-label classification is basically a classification task. Therefore, how well the model capture features that represent the corresponding labels from input data could be considered as a most important goal. In other words, a model that is able to find unique musical features of each tag is a good model in this problem. One thing to notice is that multi-label classification is different from standard multi-class classification in that one sample can have multiple labels at the same time.

2.2.1 Low-level Musical Feature Extraction Layer

Then how can we make a CNN model capture musical features well? First, let's think about the unique characteristics of music input data. Not like image data, one of the axes of music waveform stands for the time, which is sequential. Therefore, I thought that a small local square kernel size (e.g. 3x3), which does not take into account a time axis, is not appropriate. Then, what is the proper kernel size for the music data? I got a hint from the given reference paper, *musicnnn*.[2]

In *musicnn*, they split their model into two blocks, front-end and back-end layers. The front-end layer consists of 5 CNNs and each has a different role for capturing musical features. Two of

[2] Pons, J. and Serra, X. musicnn: Pre-trained convolutional neural networks for music audio tagging. arXiv preprint arXiv:1909.06654, 2019a.

them are called timbralCNN which have horizontal wide kernel size and the others are TemporalCNN having vertical wide kernel size. The kernel sizes of these CNNs make the model capture low-level musical features in two different ways, timbral and temporal, so that can consider the time axis more cleverly than the standard square kernel. Figure 6 shows the architecture of musical motivated CNN.

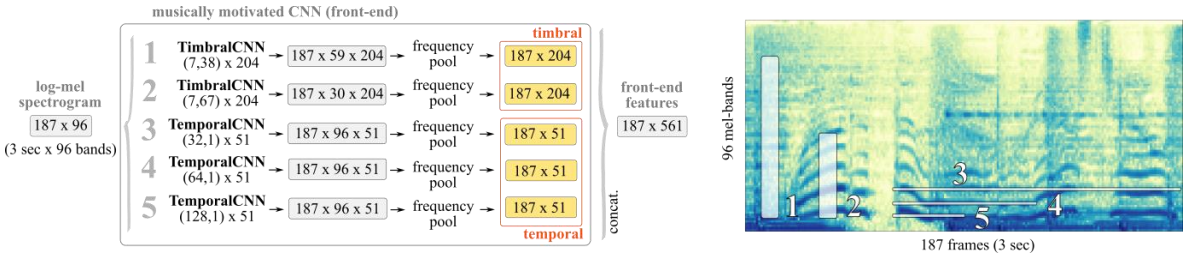


Figure 6. Architecture and result of musically motivated CNN front-end layer of musicnn.

Motivated by the idea of *musicnn*, I modified the front-end layer architecture of it as follows in figure 7. The overall concept is similar but some hyperparameters and pooling processes have changed due to the input and output size.

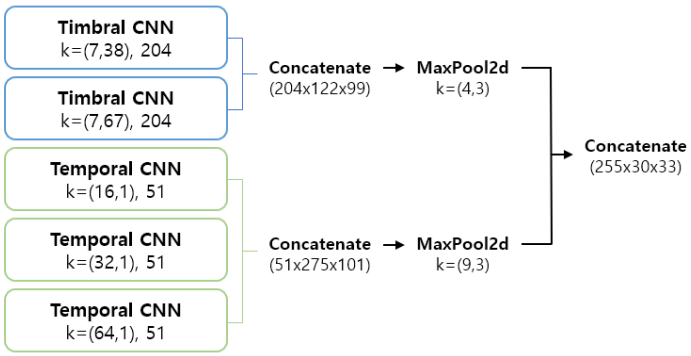


Figure 7. Architecture overview of musical feature CNN, which is the first layer of the final classification model.

I implemented the module as name as 'musical feature CNN' and utilized it for the first block of the improved CNN2D model.

2.2.2 Residual Representation Learning Layer

Since the results obtained from the above musical feature CNN are still low-level features, a process of mapping to a high-level representation is required. I constructed a simple module including 4 residual blocks. One residual block contains 2 convolutional layers and 1 residual connection. The reason I organized a simple model is that a too large model is rather easily overfitted when dealing with small datasets. The experiments using popular pretrained models such as *ResNet* and *EfficientNet* are also performed, however, they did not make a significant performance improvement. Consequently, I decided to use the customized simple residual CNN model as an upper block for the final classification model.

2.2.3 Loss Function Definition

For the loss function, the given method was a binary cross entropy with logit(sigmoid) function. In multi-label classification task, this method is generally used than softmax function because of the difference between multi-class classification and multi-label classification. In multi-class problems, the classes are mutually exclusive, while in multi-label problems each label is somehow related. Therefore, sigmoid BCE is a very suitable method for this multi-label task because it considers simply whether this label corresponds to input data or not.

However, still, I was not sure that BCE with logit is the best loss function for the given problem. Then I found that the given dataset labels are very imbalanced, biased toward some labels, as you can see in Figure 8.

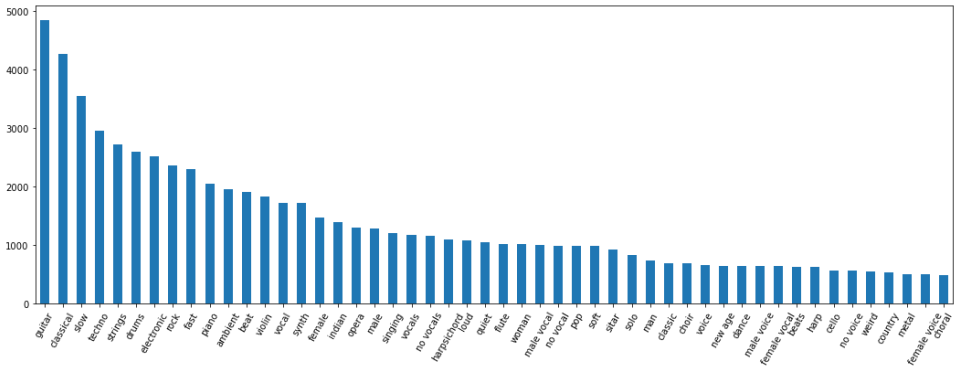


Figure 8. Plot of the labels count. The gap between labels of 'guitar' and labels of 'choral' is about 10 times.

As we already know, imbalanced data can adversely affect training. Furthermore, it further highlights the limitations of BCE, which are 1. Ignoring the impact from correctly predicted

samples 2. The value goes to infinite with an easy negative sample.

Therefore, to address this imbalance, I choose to exploit the loss function by applying Focal Loss idea. Focal Loss[3] which is motivated to improve object detectors in a biased dataset, is a Cross-Entropy Loss that weighs the contribution of each sample to the loss based on the classification error. The idea is that if a sample is classified correctly, the function gives a low contribution and vice versa.

$$\mathrm{FL(p\_\{t\})} = -\alpha_t(1-p_t)^{\gamma}\log\left(p_t\right)$$

Figure 9. Focal Loss Function Definition.

Following the loss definition in figure 9, the equation is the same as the cross-entropy equation but only different with the addition of the gamma squared factor. If a prediction of sample is confident, the gamma squared factor would be smaller and if a prediction is perplexed or misclassified the factor will be getting bigger respectively. Because the multiplication of the gamma factor makes the overall loss value smaller, the adjustment value alpha is also added. Figure 10 shows the comparison of various gamma values of focal loss. Note that when the gamma value is 0, the function is equal to standard binary cross-entropy loss, meaning focal loss could be considered as an improved version of BCE.

In this strategy, it is claimed that the imbalanced data problem is solved by making the loss more focused on the classes which are hard to classify.

[3] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollar; Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2980-2988

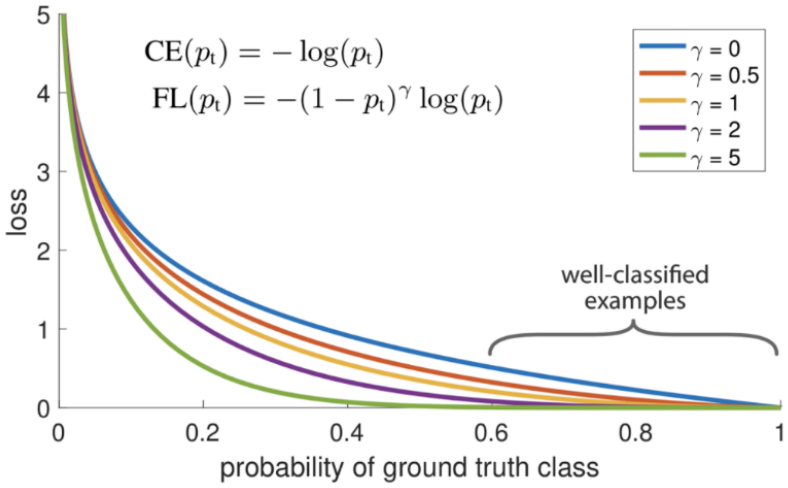
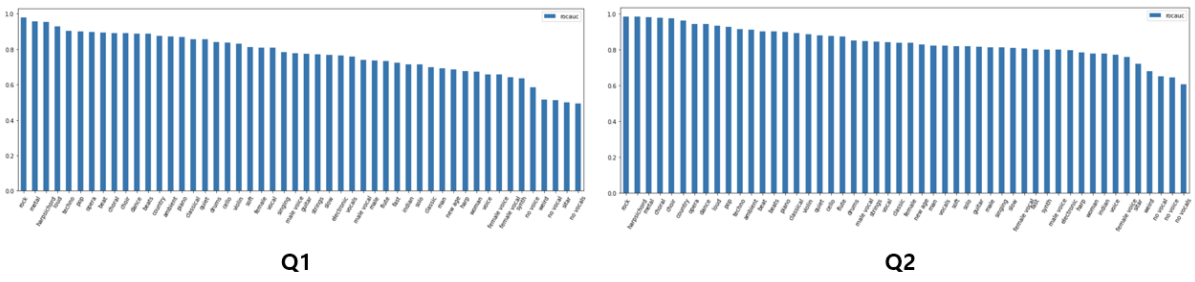


Figure 10. Comparison plot between gamma values of Focal Loss.

Therefore, I simply implemented the function and applied it to the given vanilla baseline CNN model. As a result, it was confirmed that the roc accuracy increased from 0.64% to 0.71%. Through this result, I concluded that Focal Loss is useful with the given dataset and decided to apply it as a loss function for the final improved multi tagging model.

With the suggested approaches, the performance reached up to 0.86%. The best version of the model is saved, so you can load and test it in the submitted notebook. Figure 11 is the comparison of the roc accuracy by tags between Q1 and Q2.



Q1 **Q2**

Figure 11. Comparison of roc accuracy tags between question 1 and question 2.

2.3 Approach for Metric Learning (Q4)

In deep metric learning, triplet sampling and metric definitions are as important as selecting the embedding model well. Because the goal is to keep similar samples close and non-similar samples away in the embedding space, which data should be grouped into triplets is the first factor to be considered. In question 4, I mainly focused on solving this problem.

2.3.1 Triplet Sampling Strategy

Triplet sampling strategy, which is also called Triplet mining, is one of the key factors in metric learning problems using triplets. When I first faced this question, I viewed the problem very simply, so that I came up with the idea of choosing the most non-related data from the anchor as a negative sample and the most related data as a positive sample. To get a relationship between two samples, I came up with the idea of using a multi-label classification model which was trained in question 2. The idea is as follows: After picking a random anchor sample from a dataset, the trained multi-label classification model predicts the labels of the anchor sample. From the result of predicted labels, the tag of max probability is picked as a positive tag and min probability as a negative tag. The pair of random samples per each positive and negative tag becomes triplet data with the anchor.

However, the result of applying the above first strategy shows poor performance, opposite to my expectations. Then I realized that the model was not trained at all due to the max function of loss definition.

$$\mathcal{L} = max(d(a,p) - d(a,n) + margin, 0)$$

Figure 12. Definition of triplet loss. d(·) is the distance between two embeddings.

As you can see in figure 12, if the distance from the negative sample to anchor is far and the distance from the positive one is close, meaning embedding works well, the value of the loss function is zero. Therefore, the model could not be trained at all with the first strategy. After this failure, I found that there is a concept already, called 'Easy triplets, Hard triplets, Semi-hard triplets'.

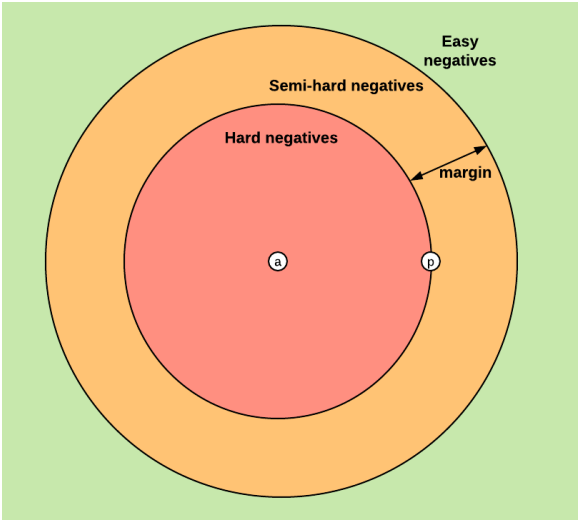


Figure 13. Regions of embedding space for negatives

Figure 13 shows the cases that can occur in triplet mining problems. Easy negatives, which I have made for the first, is not recommended to be included in the training data because it makes loss value zero as I mentioned. They are already embedded well so that there is nothing to be updated a lot. Hard negatives are the situation when the negative sample is embedded closer to the anchor than the positive sample. Thus, this is the most important pair that should be trained. Lastly, semi-hard negatives occur by the margin value. Although the negative sample is farther away from the anchor than the positive one, it cannot exceed the margin distance from the positive embedding so needs to be adjusted a little more.

For this reason, I found out that the model should train to focus on the hard negative and semi-hard negative triplets rather than the easy negative triplets. Then how can hard negative triplets be created? A lot of methods are already suggested such as proxy-based mining or batch-based online mining, however, I tried to build my own strategy using the multi-label tagging model from question 2. So, this time, the triplets are first constructed following the given method, which picks random positive/negative samples by their tags. Then the trained multi-label tagging model predicts the labels of anchor sample and encoded to a binary vector with top k option. By subtracting the ground truth and the predicted labels of anchor, false negative(FN) and false positive(FP) labels can be found, as notated in figure 14. I came up with the idea that these FN and FP labels are related to (semi)hard negatives triplets because the multi-tagging's misclassified labels could have a correlation with the sample somehow. Therefore, I sampled a track from obtained FN labels as a positive sample, and from FP labels as a negative sample.

$$score = t_i - p_i = \begin{cases} FN & (t_i - p_i = 1) \\ TN + TP & (t_i - p_i = 0) \quad i \in (0, n(tag)) \\ FP & (t_i - p_i = -1) \end{cases}$$

Figure 14. Score definition of the triplet sampling strategy, which t is ground truth and p is prediction of given sample. Following the notation, a positive data is sampled from FN and a negative data is sampled from FP.

To confirm my suggestion, I implemented this idea on the Triplet Dataset class and first compared sampled audio files. You can also check the triplet audios on the submitted notebook. Also, I added waveform images of original and newly grouped triplet's waveforms at the end of the document.

Now, in order to verify the classification-based triplet strategy, experiments performed with the given Linear Projection model only changed the number of epochs to 10. As a result, however, the result shows that the accuracy drops when the suggested strategy is applied.

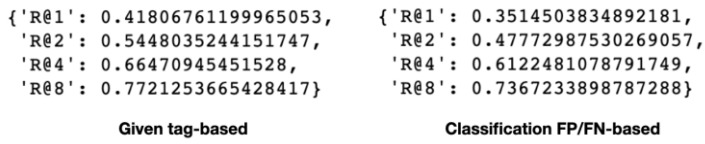


Figure 15. Results of triplet sampling strategies with the vanilla linear projection.

From the results in figure 15, it can be said that the given tag-based random sampling strategy is better than the suggested classification-based sampling strategy. Although the result was not what I expected, I think there is still a lot of potential for exploiting the classification model for metric learning problems.

2.3.2 Metric Distance

The problem of how to set the loss function is also important. Because no matter how good the triplet is constructed, the distance between each triplet sample should be properly calculated to represent their similarity. I simply compared two methods: cosine similarity and pairwise distance. As figure 16 shows, it turns out that there is nothing big difference between the two distance methods.

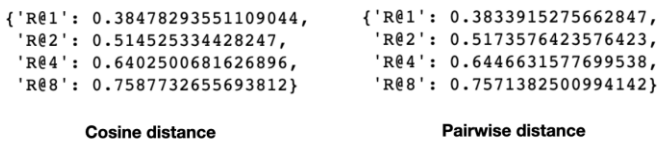


Figure 16. Results of applying cosine distance and pairwise distance.

2.3.2 Backbone Model

Since there is a pre-trained music feature extractor trained in question 2, I decided to use it as an embedding model and performed transfer learning. A linear layer is attached to the end of the music feature extractor mapping the feature vector to the embedding space.

Consequently, I applied the above approaches except for the triplet sampling approach as it could not verify its value. As a result, { 'R@1': 0.47634560692813116, 'R@2': 0.6166209464753155, 'R@4': 0.7485045429220182, 'R@8': 0.8455286622762342 } of accuracy was obtained. The best version of the model is saved, so you can load and test it in the submitted notebook.

3. Discussion & Future Study

In multi-label classification, although I spent time trying to install the pre-trained *musicnn* model, there were many difficulties in my environment, especially due to some conflicting libraries. If someone succeeded to install the pre-trained *musicnn*, I would like to discuss how he/she made it or just solve the conflicting problems together.

In metric learning, I desired to develop the triplet sampling strategy I suggested, which I mainly focused on (and failed). By applying it to other datasets, I wonder that if there is anything I can improve or if it has problems that could be fixed. Further, I plan to study more related topics such as contrastive learning, batch-based sampling, proxy-based sampling so on.

Lastly, while finding the optimal value of the hyperparameters practically, I found that there is a field called hyperparameter optimization including methods such as Grid search, Random search and Bayesian optimization. Since I felt that I have lack intuition about hyperparameters so that I would like to discuss and study hyperparameters details and optimizations.

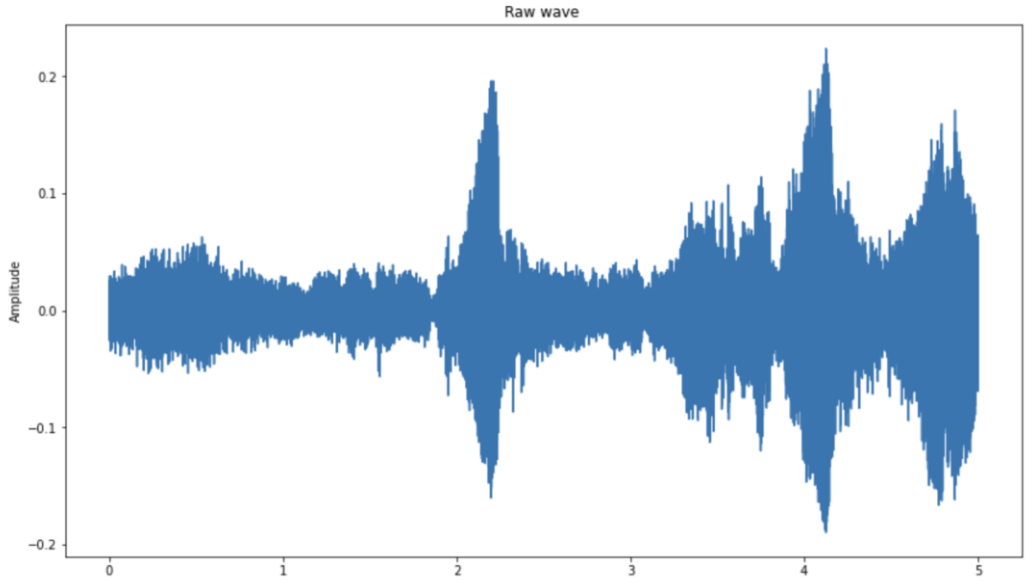
Appendix

Example waveforms of the original triplets and the rearranged triplets by the suggested classification-based sampling strategy.

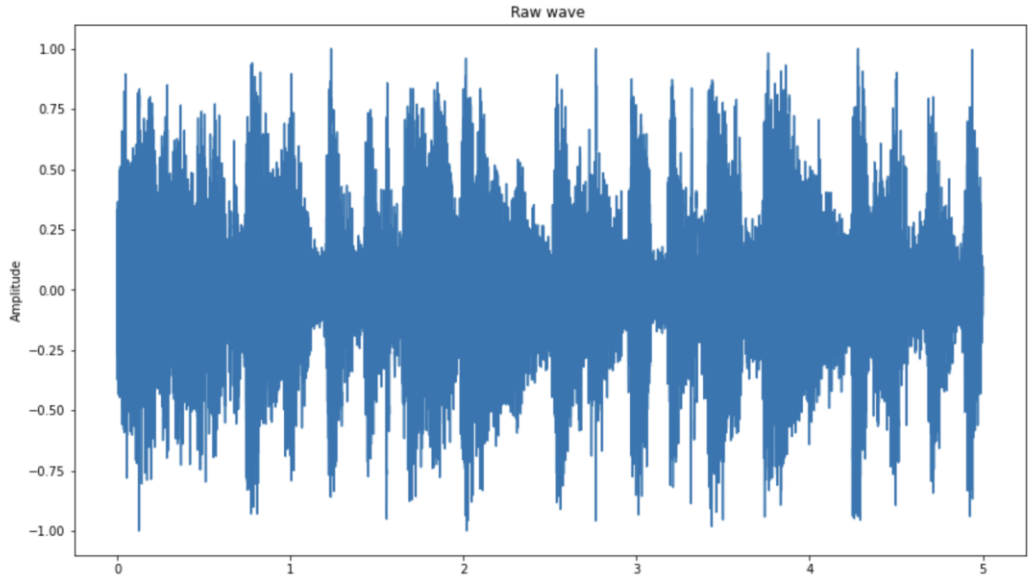
Randomly selected anchor tag = '*female vocal*'

1. Original Triplet

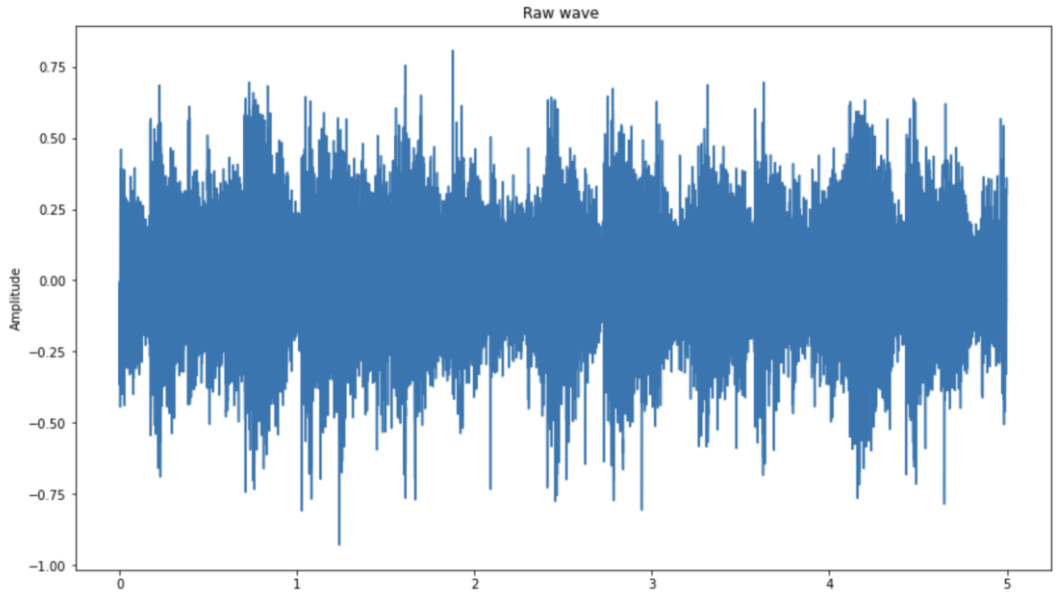
```
1 . Anchor tag:  ['classical', 'slow', 'violin', 'female', 'opera', 'woman', 'female vocal']
```




```
2 . Positive tag:  ['rock', 'woman', 'female vocal', 'female voice']
```

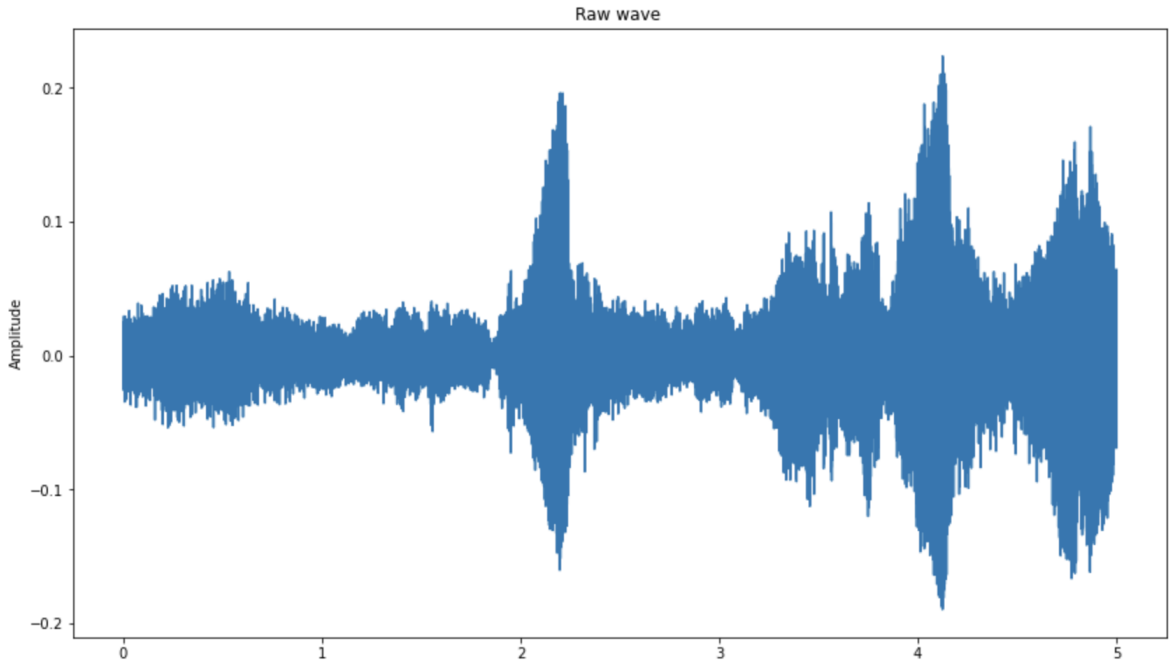


```
3 . Negative tag:  ['techno', 'fast', 'beat', 'synth']
```

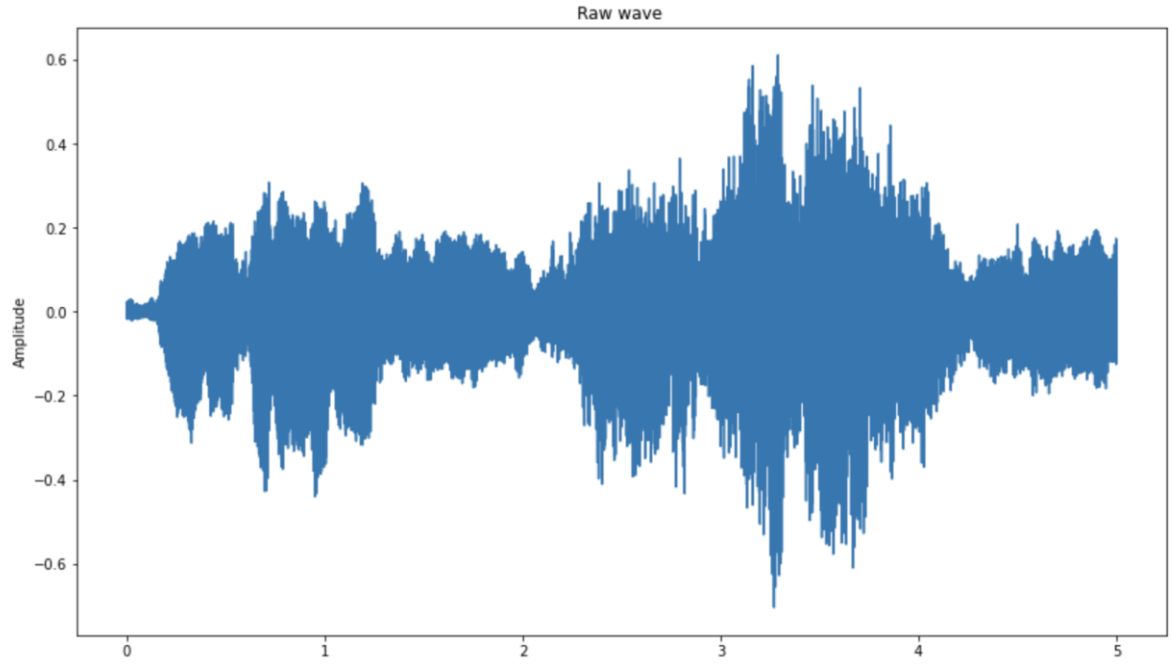


2. Rearranged Triplet

```
fn: tensor([ 2, 12, 17, 40])  fp: tensor([13, 35])
POS Changed to opera 49580
NEG Changed to vocal 27424
1 . Anchor tag:  ['classical', 'slow', 'violin', 'female', 'opera', 'woman', 'female vocal']
```



2 . Positive tag: ['classical', 'strings', 'violin', 'female', 'singing', 'woman', 'female vocal']



3 . Negative tag: ['vocal', 'opera', 'choir', 'choral']