Homework #2: Music Auto Tagging

The Robotics Program
20195595
Dooyoung Hong

1. Introduction

Music auto-tagging is an important task that can be used in many musical applications such as music search or recommender systems. Conventional methods, such as the classifier through Mel-Frequency Cepstral Coefficients (MFCC), use deterministic feature extractors based on speech domain knowledge and these formulas. The main function of machine learning is configuring the approximator that can classify the feature of data based on neural structures and actual data. Therefore, it can be a more flexible approach compared with the existing methods. Homework #2 aims to perform the auto-tagging for music using machine learning techniques.

2. Algorithm description, Experiments, and results

The base code presents the whole pipeline of deep learning such as data preparation, feature extraction, base model training, and model evaluation. So, this report is wrote based on questions that requires efforts for the improvement of model performance.

First, I implement a 2D CNN architecture following the given architecture.

Layer	Output Size	Details
input	B x 1 x sample_rate * duration,	batch x channel (mono) x samples
mel_spec	B x 1 x 96 x 188	batch x channel (mono) x freq x time
conv + maxpool	B x 64 x 24 x 47	output_channels=64, kernel_size=3, pooling=(4,4)
conv + maxpool	B x 128 x 8 x 15	output_channels=128, kernel_size=3, pooling=(3,3)
conv + maxpool	B x 128 x 2 x 5	output_channels=128, kernel_size=3 pooling=(3,3)
conv + maxpool	B x 64 x 1 x 1	output_channels=64, kernel_size=3 pooling=(2,5)
classifier	B x 50	-

Figure 1. Target 2D CNN structure.

The presented base model just uses 1D CNN. So, 2D CNN model can hope the model performance enhance only adding convolutional channel because modified convolutional filter can include more information each filter. Figure 2 shows the implemented code of 2D CNN. However, I think that the kernel size of layer4 should be 1 because it is not matches the previous output of layer3 and zero padding function not allow only adding the one side padding.

```
CNN2D(
  (spec): MelSpectrogram(
    (spectrogram): Spectrogram()
    (mel_scale): MelScale()
  )
  (to_db): AmplitudeToDB()
  (spec_bn): BatchNorm2d(1, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (layer1): Sequential(
    (0): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=3, stride=3, padding=0, dilation=1, ceil_mode=False)
  )
  (layer3): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=3, stride=3, padding=0, dilation=1, ceil_mode=False)
  )
  (layer4): Sequential(
    (0): Conv2d(128, 64, kernel_size=(1, 1), stride=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=5, padding=0, dilation=1, ceil_mode=False)
  )
  (linear): Linear(in_features=64, out_features=50, bias=True)
)
```

Figure 2. Structure information of 2D CNN.

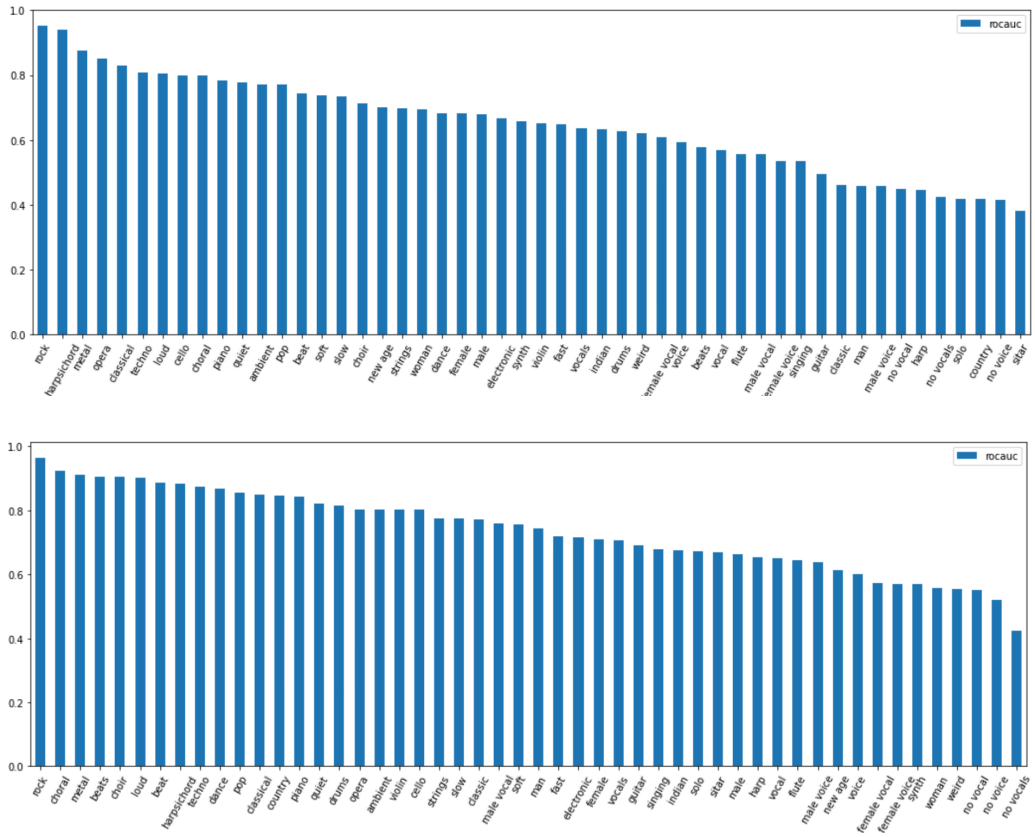

Figure 3. Classification performance of 1D and 2D CNN models.

The implemented 2D CNN model has about 75 % classification accuracy and shows meaningful performance difference than the 65 % accuracy of 1D CNN model presented as a baseline. Figure 3 shows each test result of 1D and 2D CNN models.

```
imp_CNN(
  (spec): MelSpectrogram(
    (spectrogram): Spectrogram()
    (mel_scale): MelScale()
  )
  (to_db): AmplitudeToDB()
  (spec_bn): BatchNorm2d(1, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (layer1): Sequential(
    (0): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (layer3): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (layer4): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=(3, 4), stride=(3, 4), padding=0, dilation=1, ceil_mode=False)
  )
  (layer5): Sequential(
    (0): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=(4, 3), stride=(4, 3), padding=0, dilation=1, ceil_mode=False)
  )
  (dropout): Dropout(p=0.5, inplace=False)
  (linear): Linear(in_features=64, out_features=50, bias=True)
)
```

Figure 4. Improved 2D CNN model.

Second, It is possible to improve the 2D CNN model through the precise hyperparameter setting. I expend the model depth and then adjust the filter, pooling, stride size at each layer so that they can handle input data appropriately. The optimizer used in the model is Adam that is possible to adjust the direction and size of the gradient descent. I also make minor corrections about the learning rate and learning epoch. It contains the dropout rate for the learning speed and overfitting problems. Figure 4 shows the whole specification of the implemented model.

**Question 2 directly offer implementing the residual learning that solves the gradient vanishing problem of the model referring again to the input data. However, this shortcut-based method is implemented in the next question 4, I focus on tuning the hyperparameters of the 2D CNN model in this step.*

The 2D CNN model under the modifications shows a significant performance improvement. It has 86 % of roc accuracy which is about 20 % better than the base model. Figure 5 is the classification performance of modified 2D CNN model and shows a clear difference compared with Figure 3.

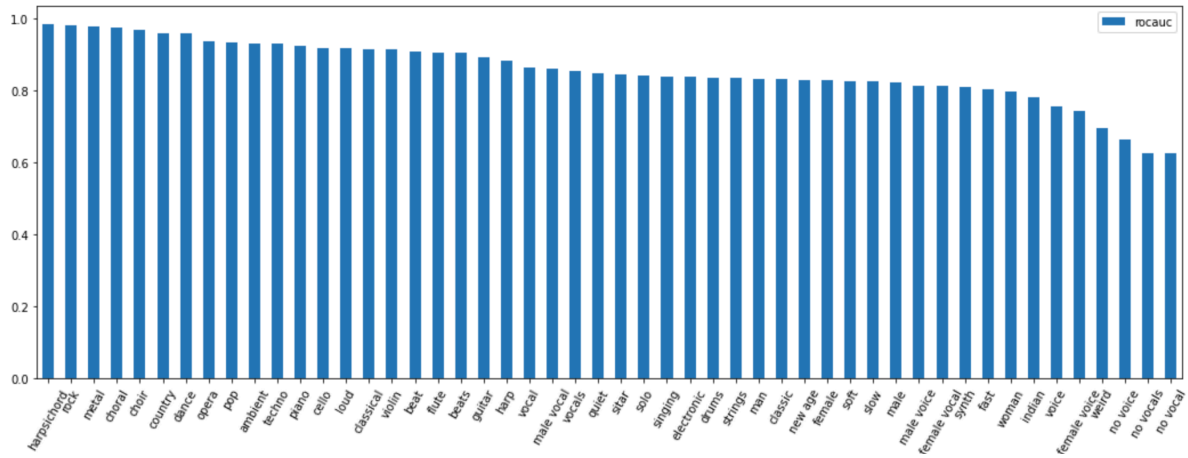

Figure 5. Classification performance of the modified 2D CNN model.

Question 3 is implementing the evaluation metric. Recall@K is the ratio of the recommended label among all relevant labels. Metric learning is a paradigm of representation learning, in which proximity between the representations of items is optimized to correspond to a notion of similarity. However, the data in this HW#2 use multiple labels, so it requires the modified form of the Recall@K function. The definition of modified Recall@K function is below:

$$Recall@K = \frac{1}{N}\Sigma_{q=1}^{N}\frac{n(y^q \cap (\cup_{i=1}^{K} y^i)}{n(y^q)}$$

Figure 6. Recall@k function for multiple labels.

where N is the number of test samples, yq is the ground truth labels of a query, and yi is the ground truth labels of the top K retrieved results. And, $n(\cdot)$ denotes the number of the elements of a set. In this setup, if the set of labels of the top K retrieved results contains all the multiple labels of the query song, the Recall@K is set to 1, otherwise it is set to the correct answer ratio. In this report, Recall@K contains K that is 1, 2, 4, and 8.

So, the report requires the implementing “multilabel_recall” function. I present the multilabel_recall function following above modified Recall@K formula as Figure 7.

```python
def multilabel_recall(self, sim_matrix, binary_labels, top_k):
  # =======================
  ## To-do

  accuracy = []

  for i in range(len(sim_matrix)):
    # 0~ 1338
    # index sort
    sort_i = np.argsort(sim_matrix[i])[::-1][1:top_k+1]

    y_q = binary_labels[i]

    sum_list = []
    for j in sort_i:
        y_i = binary_labels[j]
        sum_list.append(y_i)

    predict_sum = np.array(sum_list).sum(axis=0)
    predict_sum_binary = np.where(predict_sum < 1, predict_sum, 1)
    predict_same_with_label = y_q * predict_sum_binary

    predict = sum(predict_same_with_label)
    label = sum(y_q)
    accuracy.append(predict / label)

  return np.array(accuracy).mean()
  # =======================
```

Figure 7. modified Recall@K function for multiple labels.

The presented linear model as the base line of metric learning includes the embedding layer that has the space about the size of mel spectrogram. Its performance is Figure 8.

```
{'R@1': 0.33860449151711129,
 'R@2': 0.47838488372468957,
 'R@4': 0.6101965186916644,
 'R@8': 0.7357442988510949}
```

Figure 8 Performance of base line about multiple label metric learning.

Finally, Question 4 requires implementing the SOTA algorithms of music auto-tagging for improving the performance. In this step, I implement the three-backbone model following the presented paper lists.

- CRNN : Convolutional Recurrent Neural Networks for Music Classification, Choi et al., 2016.
- Musicnn : End-to-end Learning for Music Audio Tagging at Scale, Pons et al., 2018.
- Short-chunk CNN : Evaluation of CNN-based Automatic Music Tagging Models, Minz et al., 2020.

```
CRNN(
  (spec): MelSpectrogram(
    (spectrogram): Spectrogram()
    (mel_scale): MelScale()
  )
  (to_db): AmplitudeToDB()
  (spec_bn): BatchNorm2d(1, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (layer1): Sequential(
    (0): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=3, stride=3, padding=0, dilation=1, ceil_mode=False)
  )
  (layer3): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=3, stride=3, padding=0, dilation=1, ceil_mode=False)
  )
  (layer4): Sequential(
    (0): Conv2d(128, 64, kernel_size=(1, 1), stride=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=5, padding=0, dilation=1, ceil_mode=False)
  )
  (GRU1): GRU(64, 64, num_layers=2, batch_first=True)
  (GRU2): GRU(64, 64, num_layers=2, batch_first=True)
  (linear): Linear(in_features=64, out_features=50, bias=True)
)
```

Figure 9. Structure of CRNN model.

CRNN is based on the fact that music data is sequential data. It is a method that adds recurrent neural network layers behind CNN layers. The CRNN that is consistent with the characteristic of the data, shows less training loss than the feed-forward networks (2D CNN) during the training. However, the final test result is not significantly different from the 2D CNN (about 85 %), which is due to the overfitting tendency of the RNN model.

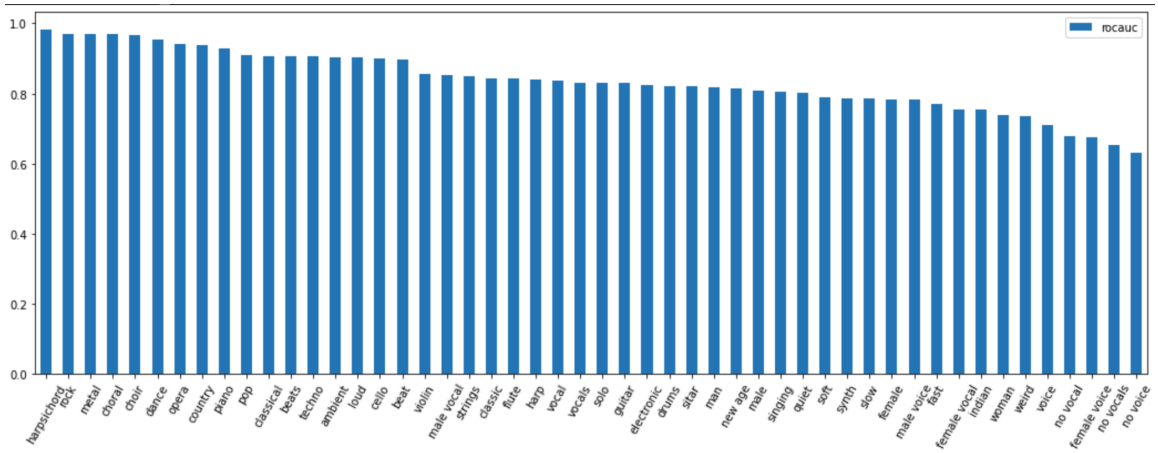

Figure 10. Classification performance of the CRNN model.

Unfortunately, Recall@K evaluation of CRNN are not good, It is not possible to determine whether this is due to the characteristic of the model or a mistake of the metric function (This model is too sensitive to the running rate and also causes frequent CPU limitation in Google Colaboratory).

```
{'R@1': 0.23361910473560962,
 'R@2': 0.37540679385339576,
 'R@4': 0.5348343134750901,
 'R@8': 0.7295996075122289}
```

Figure 11. Recall@K of CRNN model.

```
Musicnn(
  (spec): MelSpectrogram(
    (spectrogram): Spectrogram()
    (mel_scale): MelScale()
  )
  (to_db): AmplitudeToDB()
  (spec_bn): BatchNorm2d(1, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (layers): ModuleList(
    (0): Conv_V(
      (conv): Conv2d(1, 204, kernel_size=(67, 7), stride=(1, 1), padding=(0, 3))
      (bn): BatchNorm2d(204, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU()
    )
    (1): Conv_V(
      (conv): Conv2d(1, 204, kernel_size=(38, 7), stride=(1, 1), padding=(0, 3))
      (bn): BatchNorm2d(204, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU()
    )
    (2): Conv_H(
      (conv): Conv1d(1, 51, kernel_size=(129,), stride=(1,), padding=(64,))
      (bn): BatchNorm1d(51, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU()
    )
    (3): Conv_H(
      (conv): Conv1d(1, 51, kernel_size=(65,), stride=(1,), padding=(32,))
      (bn): BatchNorm1d(51, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU()
    )
    (4): Conv_H(
      (conv): Conv1d(1, 51, kernel_size=(33,), stride=(1,), padding=(16,))
      (bn): BatchNorm1d(51, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU()
    )
  )
  (layer1): Sequential(
    (0): Conv1d(561, 64, kernel_size=(7,), stride=(1,), padding=(3,))
    (1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool1d(kernel_size=1, stride=1, padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv1d(64, 64, kernel_size=(7,), stride=(1,), padding=(3,))
    (1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool1d(kernel_size=1, stride=1, padding=0, dilation=1, ceil_mode=False)
  )
  (layer3): Sequential(
    (0): Conv1d(64, 64, kernel_size=(7,), stride=(1,), padding=(3,))
    (1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool1d(kernel_size=1, stride=1, padding=0, dilation=1, ceil_mode=False)
  )
  (dense1): Linear(in_features=1506, out_features=200, bias=True)
  (bn): BatchNorm1d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU()
  (dropout): Dropout(p=0.5, inplace=False)
  (dense2): Linear(in_features=200, out_features=50, bias=True)
)
```

Figure 12. Structure of Musicnn model.

Musicnn performs the end-to-end learning for music audio tagging at scale. MFCC is a kind of passively generated speech feature aggregation. It may limit the performance of the actual speech system although this information provides meaningful information to the model because it is insufficient to approximate the real speech. End-to-end learning system is an algorithm that tries to supplement the performance of the model by supplementing the approximated speech features through the labeled data on both ends.

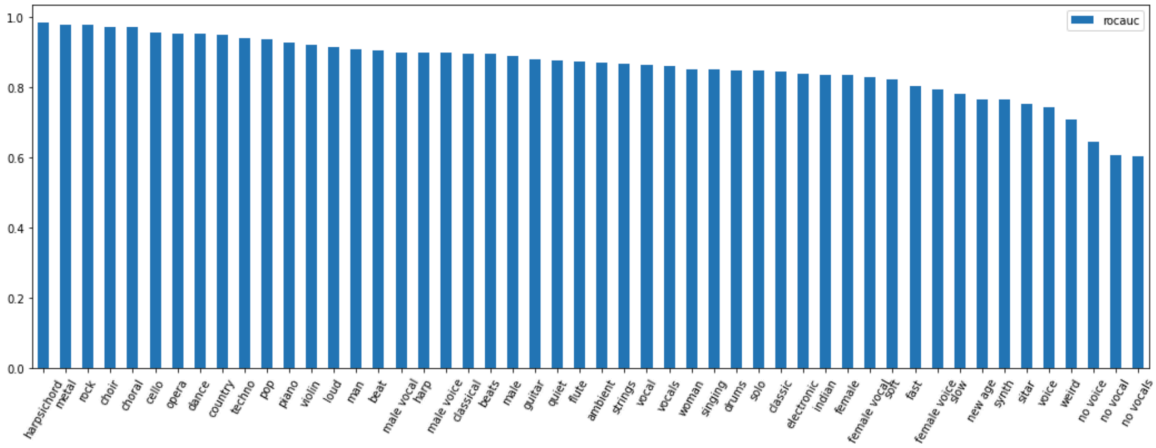

Figure 13. Classification performance of the Musicnn model.

Musicnn shows the 87% of roc accuracy. It improves the classification performance little bit (up to 2 %) than 2D CNN and CRNN models. On the other hands, Recall@K metric of the Musicnn mode shows the quite improved performance. Musicnn only uses short audio excerpts as its inputs during training. So, high classification performance can also be seen as being affected by the size of the entire sample.

```
{'R@1': 0.35445382557033045,
 'R@2': 0.5117413869841054,
 'R@4': 0.6737558288529162,
 'R@8': 0.796695348879815}
```

Figure 14. Recall@K of Musicnn model.

Short-chunkCNN + ResNet takes advantage of chunk level training and residual learning. It obtains accuracy gains from the size of the data and its entire sample as in the case of Musicnn and also boosts the performance of the model through residual connections. Figure 16 is the classification performance of the Short-chunkCNN + ResNet model. This model shows the best performance among all models of this report at the aspect of the roc_accuracy. However, the Recall@K metric does not show high performance although this model shows the highest classification performance. Namely, there are problems in the setting of the metric loss

function or the model requires hyperparameter tuning more. It was no longer possible to proceed due to colab's gpu usage limit.

3. Conclusion

Homework #2 performs the auto-tagging for music using machine learning techniques. It contains from FCN to the model using chunk level training with residual connections. Among them, Short-chunk CNN + Res model shows the highest roc accuracy of xx %. Also, the model of XXX shows the performance of classification at the aspect of multi label evaluation up to xx %. Through this HW, it is possible to look at the methodologies for the evaluation metric including the classification of sound sources and multi-label as a whole.