Homework2: Music Auto Tagging

Multi-Label Classification and Metric Learning Approach

20213013 Jiho Kang

October 25th, 2021

[Algorithm Description]

Network Input [1] reported that Mel-spectrogram can represent audio input representation better than STFTs and MFCCs. In this assignment, Mel-spectrogram is used as input for all network architectures.

Building Block About abbreviations used in table 2,3, and 4, please refer to table 1.

Abbreviation	Building Block
Mel	Mel-spectrogram + Amplitude to dB + Batch Normalization
Conv1d_B	1D Convolution + Batch Normalization + Relu + 1D Max Pooling
Conv1d_F	1D Convolution + Batch Normalization
Conv1d_M	1D Convolution + Batch Normalization + Relu + 1D Max Pooling
ConvY	Vertical Convolution + Batch Normalization + Relu + 2D Max Pooling
ConvH	Horizontal Convolution + Batch Normalization + Relu + 2D Max Pooling
Conv2d	2D Convolution + Batch Normalization + Relu + 2D Max Pooling
MP1d	1D Max Pooling
MP2d	2D Max Pooling
AP1d	1D Average Pooling
AAP1d	1D Adaptive Average Pooling
FC	Fully Connected

Table 1. Abbreviation for layer's building block

Network Architectures for Question 2 Four network architectures were compared; *1D-CNN*, *2D-CNN*, *FCN* (fully convolutional network), and *Musicnn* (pretrained). Refer to table 2 for detail structure of *1D-CNN*, *2D-CNN*, and *FCN*. *Musicnn* was taken from [4]. For more detail, refer to table 3 and figure 1.

Order	Layer & Output size					
	1D-CNN		2D-CNN		FCN	
1	Mel	(B, 1, 96, 188)	Mel	(B, 1, 96, 188)	Mel	(B, 1, 96, 188)
2	Conv1d_B	(B, 32, 64)	Conv2d	(B, 64, 24, 47)	Conv2d	(B, 16, 24, 47)
3	MP1d		MP2d		MP2d	
4	Conv1d_B	(B, 32, 22)	Conv2d	(B, 128, 8, 15)	Conv2d	(B, 32, 8, 15)
5	MP1d		MP2d		MP2d	
6	Conv1d_B	(B, 32, 8)	Conv2d	(B, 128, 2, 5)	Conv2d	(B, 64, 2, 5)
7	MP1d		MP2d		MP2d	
8	AAP1d	(B, 32, 1)	Conv2d	(B, 64, 1, 1)	Conv2d	(B, 128, 1, 1)
9	FC	(B, 50)	MP2d		MP2d	
10			FC	(B, 50)	Conv1d_F	(B, 50)

Table 2. Layer & output size of network architecture for question 2

Order	Network Architectures & Output Size	
	Musicnn	
1	Mel	(B, 1, 96, 188)
2	[ConvY, ConvY, ConvH, ConvH, ConvH]	(B, 204*2+51*3, 188)

3	Conv1d_M	(16, 64, 188)
4	Conv1d_M	(16, 64, 188)
5	Conv1d_M	(16, 64, 188)
6	[MP1d, AP1d]	(16, 753+753, 1)
7	FC+BN+Relu	(16, 200)
8	FC	(B, 50)

Table 3. Layer & output size of Musicnn architecture for question 2

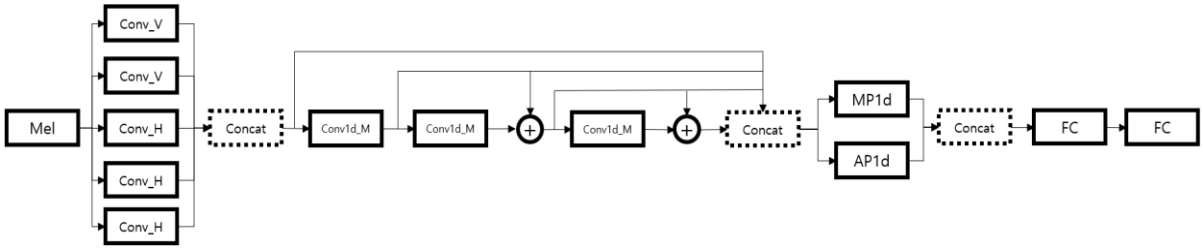

Figure 1. Musicnn architecture overview

Embedding Architectures for Question 4 Three embedding architectures were compared; *Linear*, *CNN*, and *Musicnn* (pretrained). Refer to table 4 for detail structure.

Order	Layer & Output size					
	Linear		CNN		Musicnn	
1	Mel	(B, 1, 96, 188)	Mel	(B, 1, 96, 188)	Mel	(B, 1, 96, 188)
2	FC	(B, 4096)	Conv2d	(B, 64, 24, 47)	[ConvY, ConvY, ConvH, ConvH, ConvH]	(B, 204*2+51*3, 188)
3			MP2d		Conv1d_M	(16, 64, 188)
4			Conv2d	(B, 128, 8, 15)	Conv1d_M	(16, 64, 188)
5			MP2d		Conv1d_M	(16, 64, 188)
6			Conv2d	(B, 128, 2, 5)	[MP1d, AP1d]	(16, 753+753, 1)
7			MP2d		Conv_1d_F	(16, 256)
8			Conv2d	(B, 256, 1, 1)		
9			MP2d			
10			Conv1d_F	(B, 256, 256)		

Table 4. Layer & output size of embedding architecture for question 4

Loss Design for Question 4 Triplet is defined as $t = (x_a, x_p, x_n; y_z)$, where x_a is the anchor sample, x_p is positive anchor, and x_n is negative sample. x_a and x_p are sampled from same label y_z but x_n is sample from negative for label y_z. Accordingly, basic triplet loss is defined as shown in equation (1) where equation (2) is a distance metric. In addition to basic triplet loss, disentangled triplet-based model proposed in [2] and track regularization technique proposed in [3] are adopted for question 4.

$$L_R(t) = \max\{0, \quad D(f(x_a), f(x_n)) - D\left(f(x_a), f(x_p)\right) + \Delta\} \tag{1}$$

$$D\left(f(x_i), f(x_j)\right) = \cos\left(f(x_i), f(x_j)\right) \tag{2}$$

To utilize disentanglement, the similarity notion with respect to class label is formulated as reported in Table 5. For instance, if y_z is *harpsichord*, then the similarity notion of y_z is *instrument*. The masking function is applied to the embedding feature to achieve disentanglement. Importantly, each mask evenly occupies dimensions based on number of similarity notions and dimension of embedding feature. For better understanding, assume that embedding the feature dimension is 256 (=64×4). Then, values of 192 (=64×3) dimensions is

killed by masks and only values of 64 dimensions is remained. Thus, the disentangled triplet loss is formed like equation (3).

Similarity Notion	Class label
Genre (12)	classic, techno, rock, indian, opera, country, metal, new age, dance, pop, electronic, classical
Mood (5)	soft, quiet, loud, ambient, weird
Instrument (31)	harpsichord, sitar, flute, piano, guitar, strings, drums, violin, synth, harp, cello, beat, beats, choral, vocal, vocals, no vocals, male vocal, female vocal, no vocal, solo, female, male, singing, woman, man, choir, voice, male voice, no voice, female voice
Tempo (2)	slow, fast

Table 5. Similarity notion and class label for disentanglement

$$L_D(t) = \max\left\{0,\ D\left(\frac{f(x_a)}{||f(x_a)||} \circ m_s, \frac{f(x_n)}{||f(x_n)||} \circ m_s\right) - \left(\frac{f(x_a)}{||f(x_a)||} \circ m_s, \frac{f(x_p)}{||f(x_p)||} \circ m_s\right) + \Delta\right\} \quad (3)$$

For track regularization, track based triplet is defined as $t' = (x_a, x_{a'}, x_n; y_z)$. x_a and $x_{a'}$ are sampled from same song. Different from disentanglement-based loss, it should be noted that mask function is not applied for track regularization term as shown in equation (4). That is regularization is designed considering for all dimensions of embedding space. Total loss function is equation (5) where λ represents trade-off between semantic similarity and overall track-based similarity [3].

$$L_R(t') = \max\left\{0,\ D\left(\frac{f(x_a)}{||f(x_a)||}, \frac{f(x_n)}{||f(x_n)||}\right) - \left(\frac{f(x_a)}{||f(x_a)||}, \frac{f(x_{a'})}{||f(x_{a'})||}\right) + \Delta\right\} \quad (4)$$

$$L(t, t') = L_D(t) + \lambda L_R(t') \quad (5)$$

[Experiments and Results]

Question 2 Notation *Musicnn-x* in table 6 indicates that the pre-trained parameters are used up to the x^{th} layer, and the layers after that are trained again. The Adam optimizer is used for training. The learning rate to 0.01 is initialized and is reduced by a factor of 2 when the validation loss does not decrease for 30 epochs, up to 4 times, after which early stopping is applied. The results are reported in Table 6.

Model	Model Params	Training Time (s)	ROCAUC
1D-CNN	17,300	196.2	85.3
2D-CNN	299,892	264.6	87.7
FCN	104,184	220.0	87.8
Musicnn-0 (Fully re-trained)	774,250	633.3	88.1
Musicnn-7 (Partially re-trained)	774,250	295.1	89.6
Musicnn-8 (Fully pre-trained)	774,250	No training	89.7

Table 6. Question 2 results

Question 4 To check effectiveness of track regularization and disentanglement, various combinations of loss type were experimented. *Musicnn* embedding model used pre-trained parameters except for last layer. The Adam optimizer is used for training. The learning rate to 0.001 is initialized and is reduced by a factor of 2 when the validation loss does not decrease for 5 epochs, up to 2 times, after which early stopping is applied. The margin Δ for the triplet-

based models is set to 0.4. λ was set to 0.5 for track regularization. The results are reported in Table 7.

Embedding Models	Normalization	Track Regularization	Disentanglement	Training Time (s)	Multi Label Recall			
					R@1	R@2	R@4	R@8
Linear	X	X	X	179.5	42.0	56.0	69.8	81.6
Linear	X	O	X	188.5	45.6	59.7	72.6	82.6
Linear	X	X	O	184.6	40.5	55.4	67.9	79.8
Linear	X	O	O	195.9	45.0	59.5	72.3	82.2
FCN	X	X	X	179.1	44.0	57.8	70.5	81.4
FCN	O	O	X	191.0	45.5	59.6	72.4	82.5
FCN	O	X	O	183.5	43.2	57.6	71.3	82.5
FCN	O	O	O	198.6	45.0	59.5	71.6	82.9
Musicnn	X	X	X	222.4	48.5	62.7	76.0	85.7
Musicnn	O	O	X	229.0	50.0	64.2	76.9	87.1
Musicnn	O	X	O	238.0	50.5	64.9	77.4	86.3
Musicnn	O	O	O	231.8	50.1	64.6	77.4	86.7

Table 7. Question 4 results for training time, multi label recall

[Discussion]

Experiment Result Analysis for Question 2 The originally given settings (epochs: 10, optimizer: Nesterov momentum) of *1D-CNN* and *2D-CNN* were changed to (epochs: 30, optimizer: Adam) and the test was performed. As shown in table 6, the results showed that the performance of *1D-CNN* and *2D-CNN* increased from 64, 77 to 85, 87, respectively. It can be seen that the Adam optimizer worked effectively when the epoch was increased.

When comparing *2D-CNN*, *FCN*, and *Musicnn-0*, it was confirmed that the number of parameters and performance were not proportional. Although the number of parameters of FCN was noticeably smaller than that of *2D-CNN* and *Musicnn-0*, there was no difference in performance. (Refer to table 6)

Musicnn model was pretrained using all of the MagnaTagATune dataset. The interesting thing found here was that the performance deteriorated as the learning of the pretrained *Musicnn* layer with a subset of MagnaTagATune was further carried out. The reason for this is speculated that the model learned a better representation from a larger amount of data. (Refer to table 6)

Experiment Result Analysis for Question 4 To validate effectiveness of implementation for disentanglement and track regularization, an ablation study was conducted. Looking at the case of *Linear* embedding first, the performance was the best when only track regularization was used, and disentanglement did not improve the performance. It was judged that if the masking function was applied in a state where non-linearity could not be expressed, it adversely affected the representation of data. However, in the case of *FCN* and *Musicnn*, performance is improved by using disentanglement. (Refer to table 7)

Similar to the experimental results of Question 2, the pre-trained *Musicnn* had the highest performance. Since it has been trained with a larger amount of data, it seems that Model can have a better representation of data when compared with other models which only were trained using subset of data.

Reference

[1] Keunwoo Choi, George Fazekas, Mark Sandler. Automatic Tagging using Deep Convolutional Neural Networks. *arXiv.* https://arxiv.org/abs/1606.00298, 2016.

[2] Jongpil Lee, Nicholas J. Bryan, Justin Salamon, Zeyu Jin, Juhan Nam. Metric Learning vs Classification for Disentangled Music Representation Learning. *arXiv*. https://arxiv.org/abs/2008.03729, 2020.

[3] Jongpil Lee, Nicholas J. Bryan, Justin Salamon, Zeyu Jin, Juhan Nam. Disentangled Multidimensional Metric Learning for Music Similarity. *IEEE ICASSP,* 2020.

[4] State-of-the-art Music Tagging Models. https://github.com/minzwon/sota-music-tagging-models. *Github*.

Appendix

1. Question 1: Implement a CNN based on a given model specification

```
class Conv_2d(nn.Module):
  def __init__(self, input_channels, output_channels, kernel_size=3, stride=1, padding=1, pooling=2):
    # To do
    #==========================================
    super(Conv_2d, self).__init__()
    self.conv = nn.Conv2d(input_channels, output_channels, kernel_size, 1, padding, bias=True)
    self.bn = nn.BatchNorm2d(output_channels)
    self.relu = nn.ReLU(inplace=True)
    self.mp = nn.MaxPool2d(pooling)
    #==========================================
  def forward(self, x):
    out = self.mp(self.relu(self.bn(self.conv(x))))
    return out
```

```
class CNN2D(nn.Module):
  def __init__(self,
               sample_rate=16000,
               n_fft=512,
               f_min=0.0,
               f_max=8000.0,
               n_mels=96,
               n_class=50):
    super(CNN2D, self).__init__()

    # Spectrogram
    self.spec = torchaudio.transforms.MelSpectrogram(sample_rate=sample_rate,
                                                     n_fft=n_fft,
                                                     f_min=f_min,
                                                     f_max=f_max,
                                                     n_mels=n_mels)
    self.to_db = torchaudio.transforms.AmplitudeToDB()
    self.spec_bn = nn.BatchNorm2d(1)
    # To do
    #==========================================
    self.layer1 = Conv_2d(1, 64, 3, 1, 1, 4)
    self.layer2 = Conv_2d(64, 128, 3, 1, 1, 3)
    self.layer3 = Conv_2d(128, 128, 3, 1, 1, 3)
    self.layer4 = Conv_2d(128, 64, 3, 1, 1, (2,5))
    #==========================================
    self.linear = nn.Linear(64, n_class)

  def forward(self, x):
    x = self.spec(x)
    x = self.to_db(x)
    x = self.spec_bn(x)
    x = self.layer1(x)
    x = self.layer2(x)
    x = self.layer3(x)
    x = self.layer4(x)
    x = x.view(x.size(0), -1)
    x = self.linear(x)
    x = nn.Sigmoid()(x) # for binary cross entropy loss
    return x
```

2. Question 3: Implement the evaluation metric

```
def multilabel_recall(self, sim_matrix, binary_labels, top_k):
  Multilabel_Recall=0
  for i in range(binary_labels.shape[0]):
      indices = (-sim_matrix[i,:]).argsort()[:top_k+1]
      indices = np.delete(indices, np.where(indices==i))
      numerator  = np.sum(np.logical_and(binary_labels[i,:], np.any(binary_labels[indices,:], axis=0)))
      denominator = np.sum(binary_labels[i,:])
      Multilabel_Recall += numerator / denominator
  Multilabel_Recall /= binary_labels.shape[0]
  return Multilabel_Recall
```

3. Disentangled triplet loss

```
class DisentangledTripletLoss(nn.Module):
  def __init__(self, margin):
    super(DisentangledTripletLoss, self).__init__()
    self.margin = margin
    self.relu = nn.ReLU()

  def forward(self, anchor, positive, negative, tag):
    # mask function using similarity_notion
    mask = torch.zeros((anchor.shape[0],anchor.shape[1])).to('cuda')
    num_similarity_notion = 4
    evenly_occupied = int(anchor.shape[1]/num_similarity_notion)
    tag_i=0
    for i in tag:
        tag_idx = TAGS_re.index(i)
        mask_temp = torch.zeros((anchor.shape[1],))
        if tag_idx<=11: # Genre(0-11)
            mask_temp[0:evenly_occupied]=1
        elif tag_idx>=12 and idx<=16: # Mood(12-16)
            mask_temp[evenly_occupied:evenly_occupied*2]=1
        elif tag_idx>=17 and idx<=47: # Instrument(17-47)
            mask_temp[evenly_occupied*2:evenly_occupied*3]=1
        elif tag_idx>=48 and idx<=49: # Tempo
            mask_temp[evenly_occupied*3:evenly_occupied*4]=1
        mask[tag_i,:]=mask_temp
        tag_i+=1

    anchor = anchor / anchor.norm(dim=-1, keepdim=True)
    positive = positive / positive.norm(dim=-1, keepdim=True)
    negative = negative / negative.norm(dim=-1, keepdim=True)

    anchor = anchor*mask
    positive = positive*mask
    negative = negative*mask

    pos_sim = nn.CosineSimilarity(dim=-1)(anchor, positive)
    neg_sim = nn.CosineSimilarity(dim=-1)(anchor, negative)

    losses = self.relu(self.margin - pos_sim + neg_sim)
    return losses.mean()
```

4. Track Regularization

```
import random
class DisentangledTripletDataset(TripletDataset):
  def __getitem__(self, index):
    if self.split in ["TRAIN","VALID"]:
      tag = random.choice(self.tags)
      tag_binary = self.binary[tag]
      positive_tracks = tag_binary[tag_binary != 0]
      negative_tracks = tag_binary[tag_binary == 0]
      anc_id, pos_id = positive_tracks.sample(2).index
      neg_id = negative_tracks.sample(1).index[0]

      anc_waveform = self.item_to_waveform(anc_id)
      reg_waveform = self.item_to_waveform(anc_id) # track regularization
      pos_waveform = self.item_to_waveform(pos_id)
      neg_waveform = self.item_to_waveform(neg_id)
      return anc_waveform, reg_waveform, pos_waveform, neg_waveform, tag

    elif self.split == "TEST":
      item = self.binary.iloc[index]
      id = item.name
      path = os.path.join(self.paths, self.id_to_path[id].replace(".mp3",".npy"))
      waveform = np.load(path)
      chunk_number = waveform.shape[0] // self.input_length
      chunk = np.zeros((chunk_number, self.input_length))
      for idx in range(chunk.shape[0]):
        chunk[idx] = waveform[idx * input_length:(idx+1) * input_length]
      audio = chunk.astype(np.float32)
      label = item.values.astype(np.float32)
      return audio, label
```

```
class TrackRegularizationLoss(nn.Module):
  def __init__(self, margin):
    super(TrackRegularizationLoss, self).__init__()
    self.margin = margin
    self.relu = nn.ReLU()

  def forward(self, anchor, regularizer, negative):
    anchor = anchor / anchor.norm(dim=-1, keepdim=True)
    regularizer = regularizer / regularizer.norm(dim=-1, keepdim=True)
    negative = negative / negative.norm(dim=-1, keepdim=True)

    pos_sim = nn.CosineSimilarity(dim=-1)(anchor, regularizer)
    neg_sim = nn.CosineSimilarity(dim=-1)(anchor, negative)

    losses = self.relu(self.margin - pos_sim + neg_sim)
    return losses.mean()
```

```
def item_to_waveform(self, id):
  path = os.path.join(self.paths, self.id_to_path[id].replace(".mp3", ".npy")) # pre-extr
  waveform = np.load(path)
  random_idx = np.random.randint(low=0, high=int(waveform.shape[0] - self.input_length))
  waveform = waveform[random_idx:random_idx+self.input_length] # extract input
  audio = np.expand_dims(waveform, axis = 0)# 1 x samples
  return audio.astype(np.float32)
```