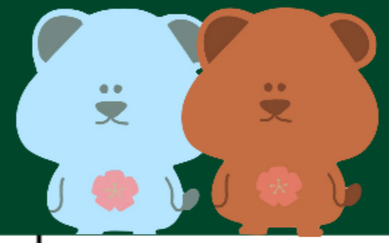


Euron 5th team project:

Hmm2Song

고급심화 세션 – 박지연 강효은 차수빈

EWHA EURON

0. INTRO

Hmm..2Song?

- **갑자기 떠오르는 노래, 어떻게 찾을까?**
- 허밍으로 노래를 찾아주는 구글 hum to search에서 영감을 받아, 허밍 뿐만 아니라 장르, 가수 성별 등 Hmm.. 하고 떠오르는 요소들로 노래를 찾아주는 모델을 구현해보자

EWHA EURON

Contents

01 **데이터 수집**

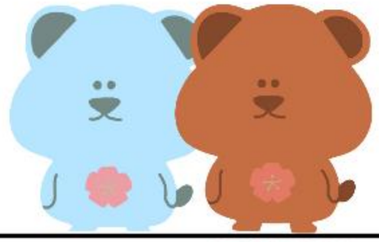
Melon data crawling / download song

02 **데이터셋 구축 / 데이터 전처리**

make full input / make split input / make split hum / npy dataset

03 **모델링 / 실험**

CNN layer + Triplet loss / cosine similarity / pairwise similarity

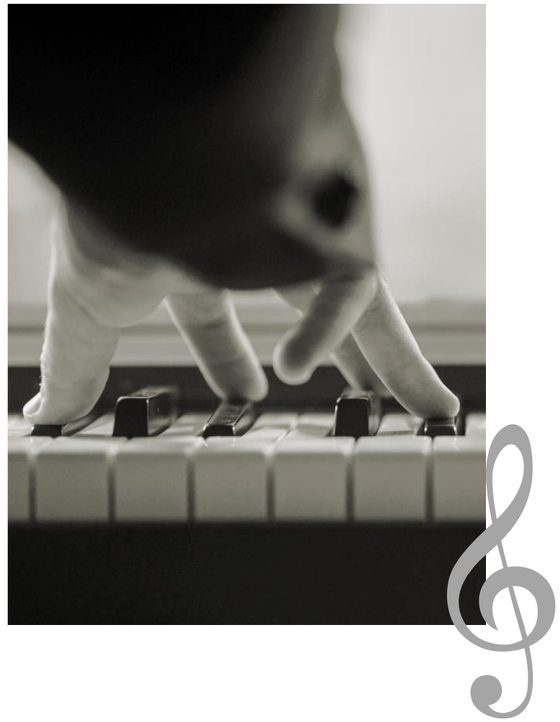


EWHA EURON

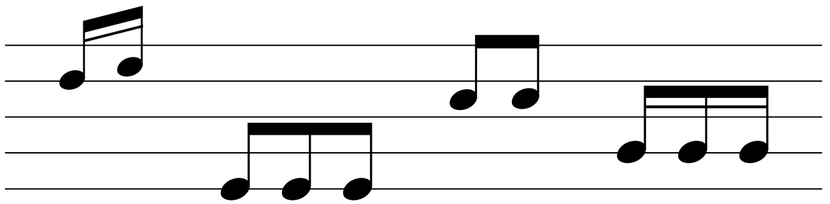
01

데이터 수집

Melon data crawling / download song



데이터 수집



RnB & Soul	Indie	Folk & Blues	Rock & Metal
Trot	Hiphop	Dance	Ballad

2023년 12월 기준, 각 장르별 TOP100 (Dance/Ballad는 TOP150) 을 수집 대상으로 함

이후 각 장르 별 중복을 제거하여 총 821곡의 데이터 수집

데이터 수집

Contents of info.csv

멜론 데이터 크롤링을 통해 수집 대상 곡에 대한 정보를 추출 + 불가피한 경우 수동 입력

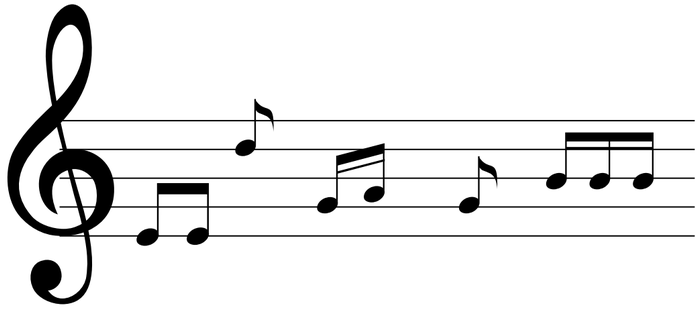
Title	곡 제목
Singer	가수
Lyric	가사
genre	곡 장르 (중복인 경우 두 장르 모두 포함)
path	이후 데이터셋 구축 과정에서 Full Input 데이터의 path를 기록
gender	가수의 성별을 기록 (수동 작성) – 여성 / 남성 / 혼성

데이터 수집

Contents of info.csv

멜론 데이터 크롤링을 통해 수집 대상 곡에 대한 정보를 추출 + 불가피한 경우 수동 입력

	A	B	C	D	E	F	G
	id	Title	Singer	Lyric	genre	path	gender
0	0	To. X	태연 (TAEYEON	처음 본 널 기억해 We sk	RnB_Soul	/content/drive/My	여성
1	1	Either Way	IVE (아이브)	누가 내 말투가 재수없대	RnB_Soul	/content/drive/My	여성
2	2	DIE 4 YOU	DEAN	I don't wanna go Till eve	RnB_Soul	/content/drive/My	남성
3	3	Must Have Love	SG 워너비, 브라	함께 있단 이유로 행복했	RnB_Soul	/content/drive/My	혼성
4	4	Still With You	정국	날 스치는 그대의 옅은 그	RnB_Soul	/content/drive/My	남성
5	5	Yes or No	정국	This ain't another love s	RnB_Soul	/content/drive/My	남성
6	6	미워 (Ego)	Crush	유난이라고 하지 마 밤길	RnB_Soul	/content/drive/My	남성
7	7	Merry PLLIstmas	PLAVE	Starry night 오늘을 준비	RnB_Soul	/content/drive/My	남성
8	8	정이라고 하자 (F	BIG Naughty (서	Back to the day 갓 10대	RnB_Soul	/content/drive/My	남성
9	9	빛이 나는 너에거	던 (DAWN)	그때 널 만나지 않았더라	RnB_Soul	/content/drive/My	남성
10	10	오늘도 빛나는 너	마크툽 (MAKTUB	별빛이 내린 밤 그 풍경 ᄉ	RnB_Soul, Indie	/content/drive/My	남성
11	11	괜찮아도 괜찮아	디오 (D.O.)	숱하게 스쳐간 감정들에	RnB_Soul	/content/drive/My	남성
12	12	밤 (Night)	dori	영화 같던 이 밤 사랑했던	RnB_Soul	/content/drive/My	남성
13	13	사람 Pt.2 (feat. C	Agust D	So time is yet now Right	RnB_Soul	/content/drive/My	남성
14	14	벌써 크리스마스	케이윌, 소유 (SC	첫눈이 내려와 눈을 감았	RnB_Soul	/content/drive/My	혼성



02

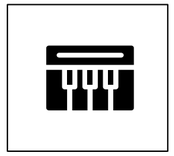
데이터셋 구축
데이터 전처리

make full input / make split input / make split hum / npy dataset

EWHA EURON

데이터셋 구축 / 데이터 전처리

데이터셋 구축 Flow

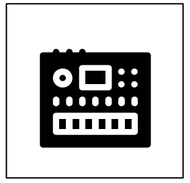


Full Input

* Spleeter 모듈을 통해 voice/ mr 분리
* Voice 기준으로 소리가 없는 부분에 대해 mr로 채워 full input 생성

Split Input

* Full input을 10초 단위로 분할 (window size = 8sec)



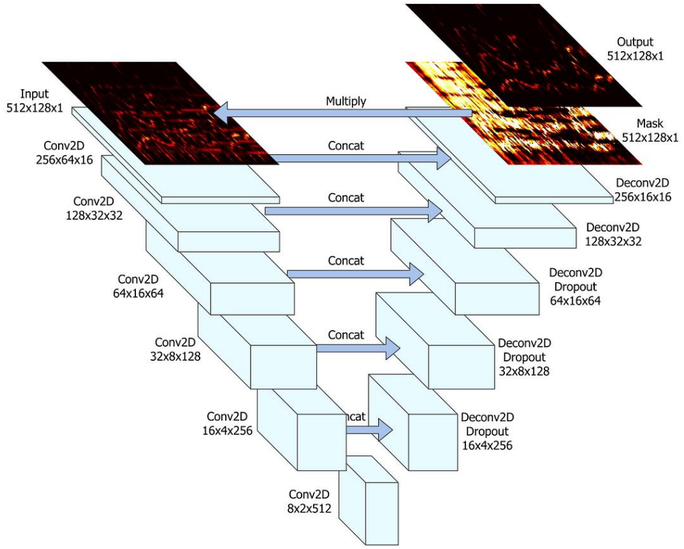
Split hum

* Split input에 대해 5가지 방법으로 변형 /증강하여 허밍데이터 생성

데이터셋 구축

Full Input

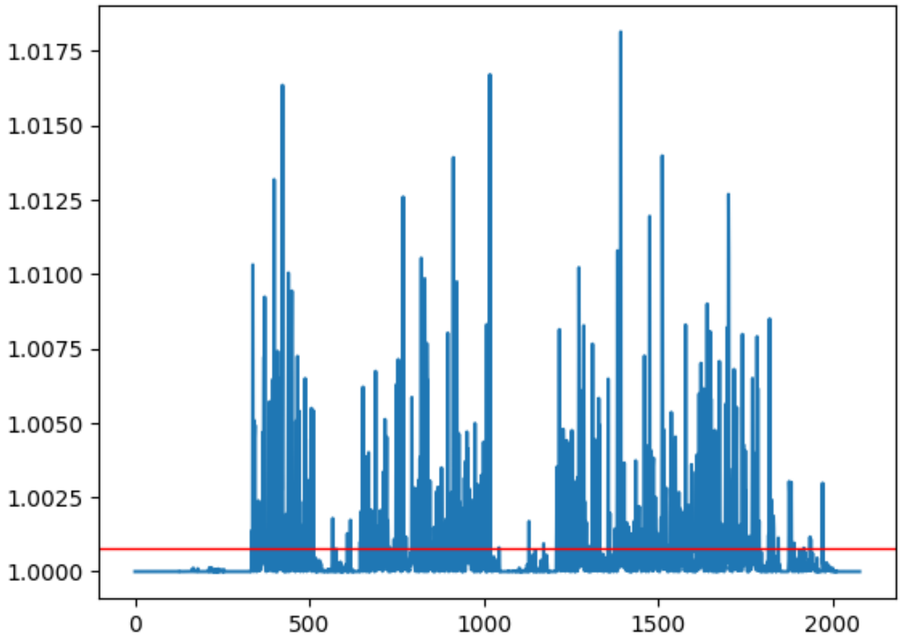
* Spleeter 모듈을 통해 voice/ mr 분리



- Skip-connection이 포함된 Encoder-Decoder 기반 CNN U-Net architecture
- Encoder : vocal, instrument에 대한 representation 학습
- Decoder : component masking 처리하여 waveform 생성

Full Input

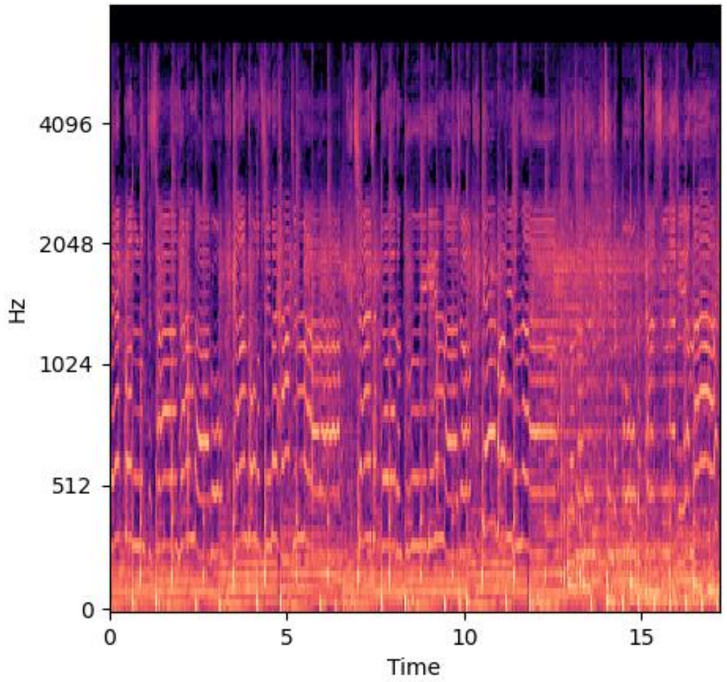
* Voice 기준으로 소리가 없는 부분에 대해 mr로 채워 full input 생성



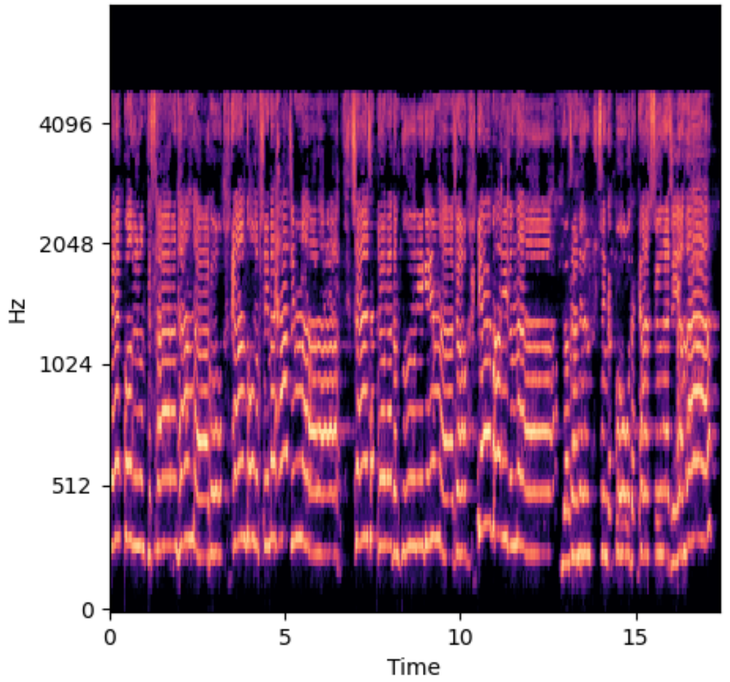
- 분리된 voice 음원의 진폭에 대해 절댓값을 씌워 그래프 생성
- 개별 곡의 절댓값 진폭 mean 값을 임계점으로 설정
- 임계점 이하의 시간들은 voice가 없는 구간으로 판단하여 해당 구간에는 mr을 삽입

Full Input

* WHY? : mel-spectrogram을 그려보았을 때, 음원의 특징을 더 확실히 보여준다



Voice 분리 작업 전



Voice 분리 작업 후

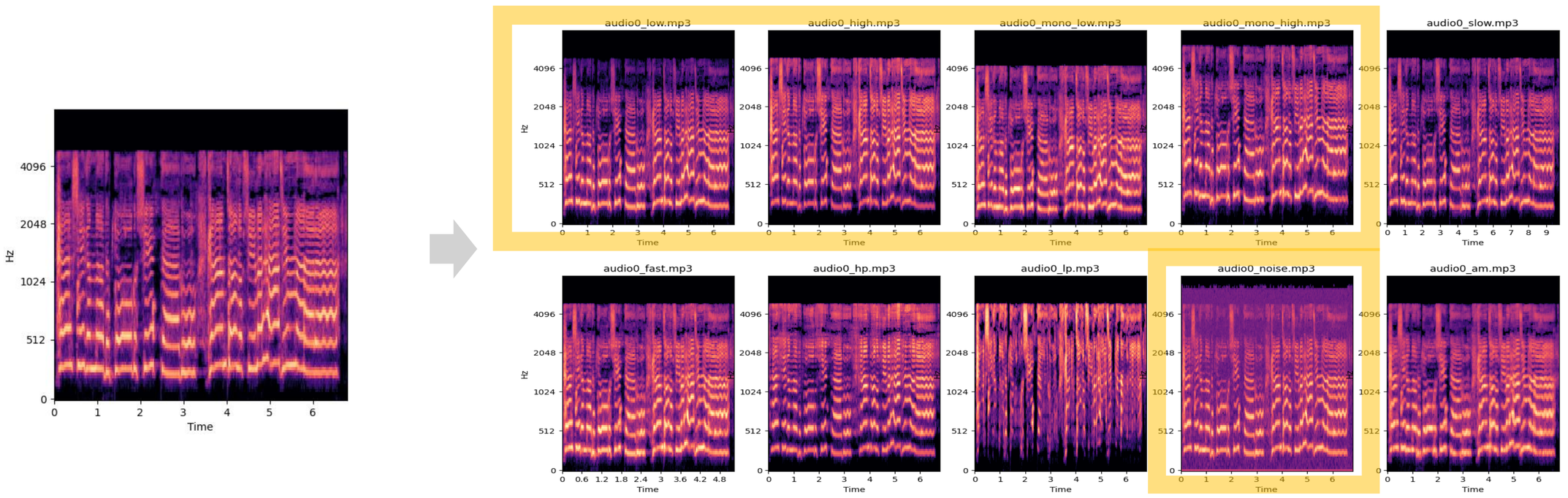
Split Input

* Full input을 10초 단위로 분할 (window size = 8sec)

데이터셋 구축

Split Hum

* Split input에 대해 5가지 방법으로 변형 / 증강하여 허밍데이터 생성

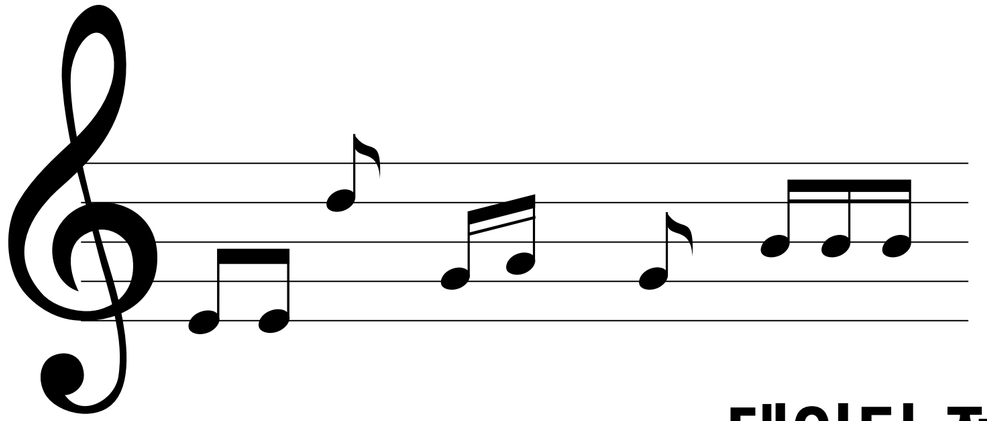


변형 후 mel-spectrogram을 그려보았을 때 음원 특성이 유지되면서도 형태가 변화한 증강 type을 채택

Split Hum

* Split input에 대해 5가지 방법으로 변형 / 증강하여 허밍데이터 생성

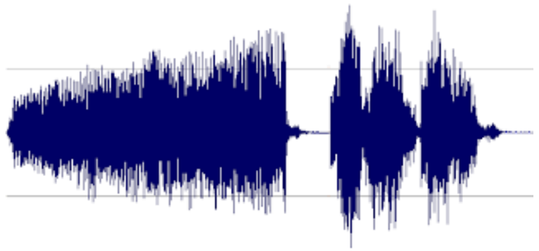
[title]_fh	High filter 적용
[title]_fl	Low filter 적용
[title]_mh	Monotone high 변환
[title]_ml	Monotone low 변환
[title]_n	Nosie 추가



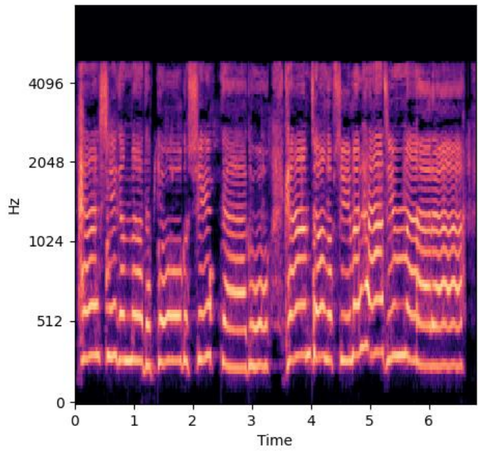
데이터 전처리 작업

데이터 전처리

Split Input



Wave form



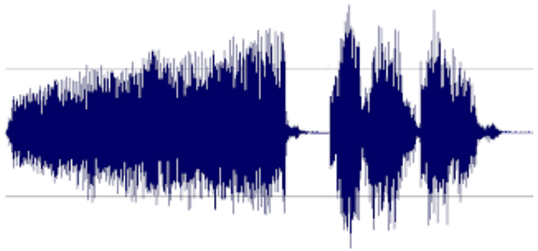

```
array([[3.837e-06, 1.451e-06, ..., 8.352e-14, 1.296e-11],
       [2.213e-05, 7.866e-06, ..., 8.532e-14, 1.329e-11],
       ...,
       [1.115e-05, 5.192e-06, ..., 3.675e-08, 2.470e-08],
       [6.473e-07, 4.402e-07, ..., 1.794e-08, 2.908e-08]],
      dtype=float32)
```

Mel vector

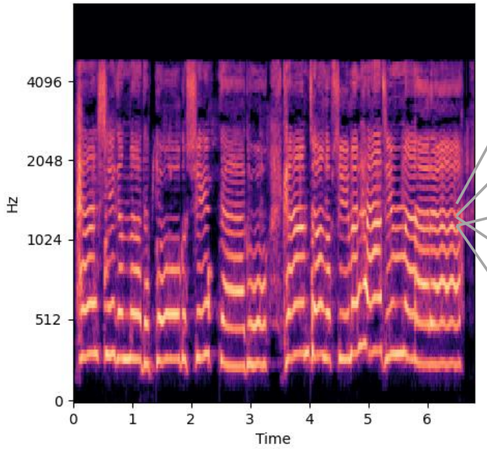
Wave form 형태의 split input을 mel vector로 변환하여 원곡 데이터로 사용 (npy 형태)

데이터 전처리

Split Hum



Wave form



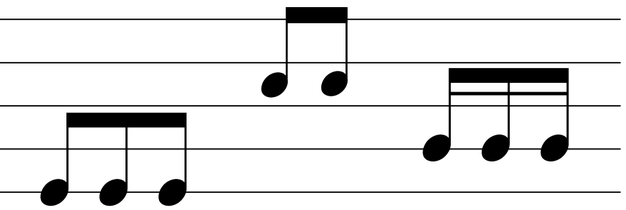
Mel vector

- High filter .npy
- Low filter .npy
- Monotone high .npy
- Monotone low .npy
- Noise .npy

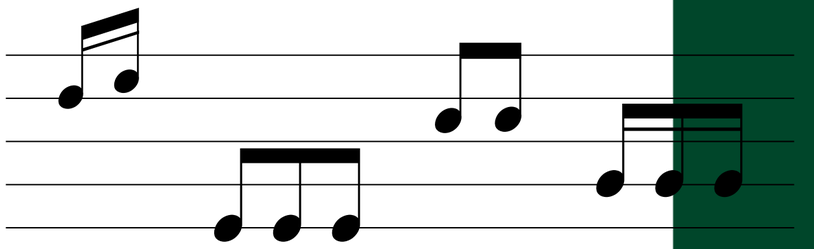
Wave form 형태의 split hum을 mel vector로 변환하여 허밍 데이터로 사용 (npy 형태)

이때, 각 변환 형태 별로 하나의 npy 파일로 묶어 학습 시 빠르게 load할 수 있도록 함

03



모델링 & 실험

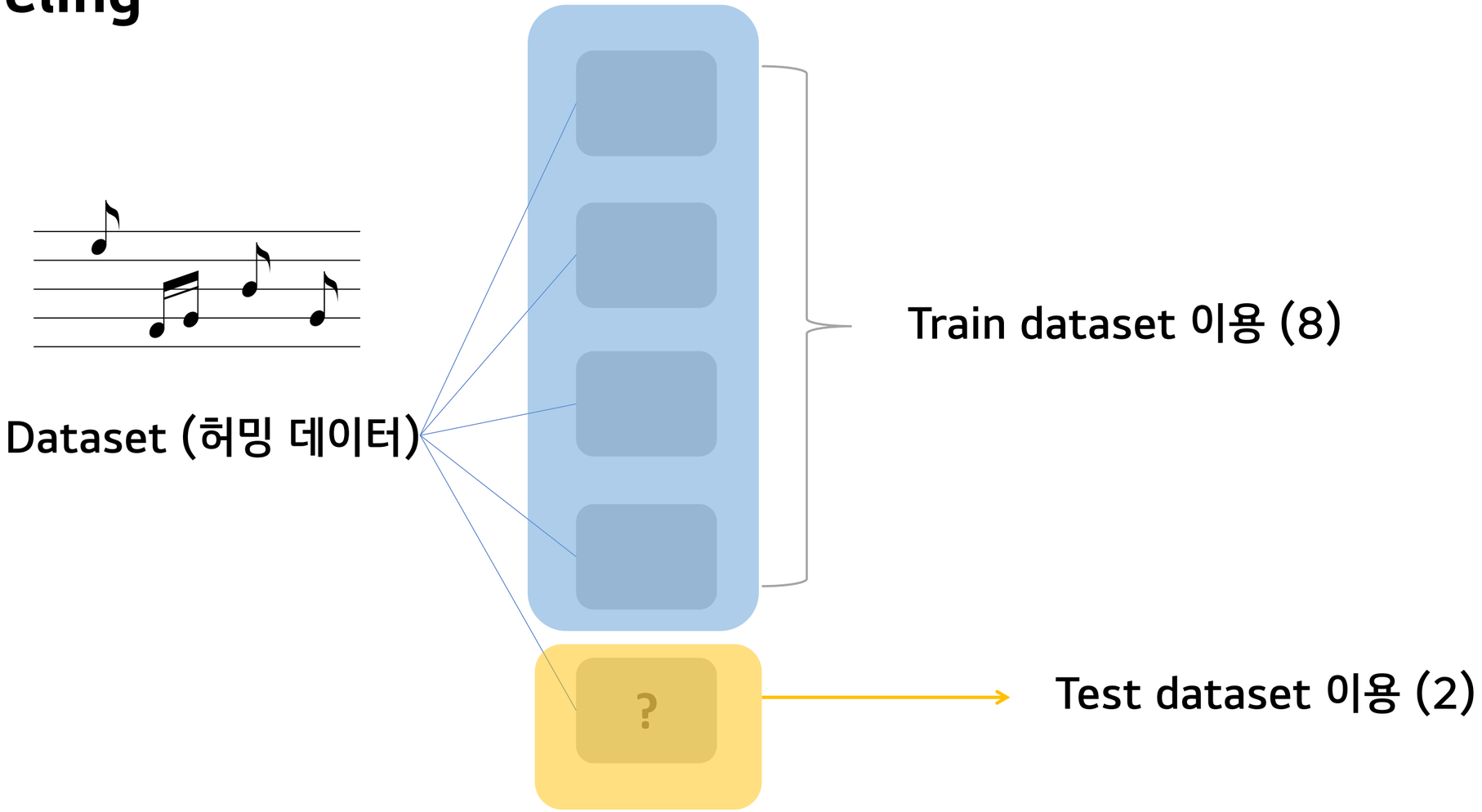


CNN layer + Triplet loss / cosine similarity / pairwise similarity

EWHA EURON

모델링_data split

* Modeling



모델링_Train

* Modeling

```python
class ConvSubnet(nn.Module):
  def __init__(self, embedding_dims = 128):
    super(ConvSubnet, self).__init__()
    ## Convolutional Layer
    self.conv1 = nn.Conv2d(1, 64, kernel_size = 3, padding = 1)
    self.conv2 = nn.Conv2d(64, 128, kernel_size = 3, padding = 1)
    self.conv3 = nn.Conv2d(128, 256, kernel_size = 3, padding = 1)
    ## MaxPooling Layer
    self.pool = nn.MaxPool2d(kernel_size = 2, stride = 2)
    ## Fully Connected Layer
    # maxpooling을 3번 거치면서 height(128 => 64 -> 32 -> 16), width(431 => 215, 107, 53)
    self.fc = nn.Linear(848, embedding_dims)  # 16 * 53 = 848

  def forward(self, x):
    ## 데이터 차원 조정 -> [batch, channel, height, width]
    x = x.unsqueeze(1)
    ## Convolutional Layer를 통과시켜 특징을 추출하고 활성화 함수를 적용
    x = self.pool(F.relu(self.conv1(x)))
    x = self.pool(F.relu(self.conv2(x)))
    x = self.pool(F.relu(self.conv3(x)))
    ## 데이터를 1차원으로 펼침
    x = x.view(-1, 848)
    ## FC Layer를 통과시켜 임베딩 생성
    x = self.fc(x)
    return x
```

ConvSubnet

Convolution layer를 통해
Mel vector를 embedding

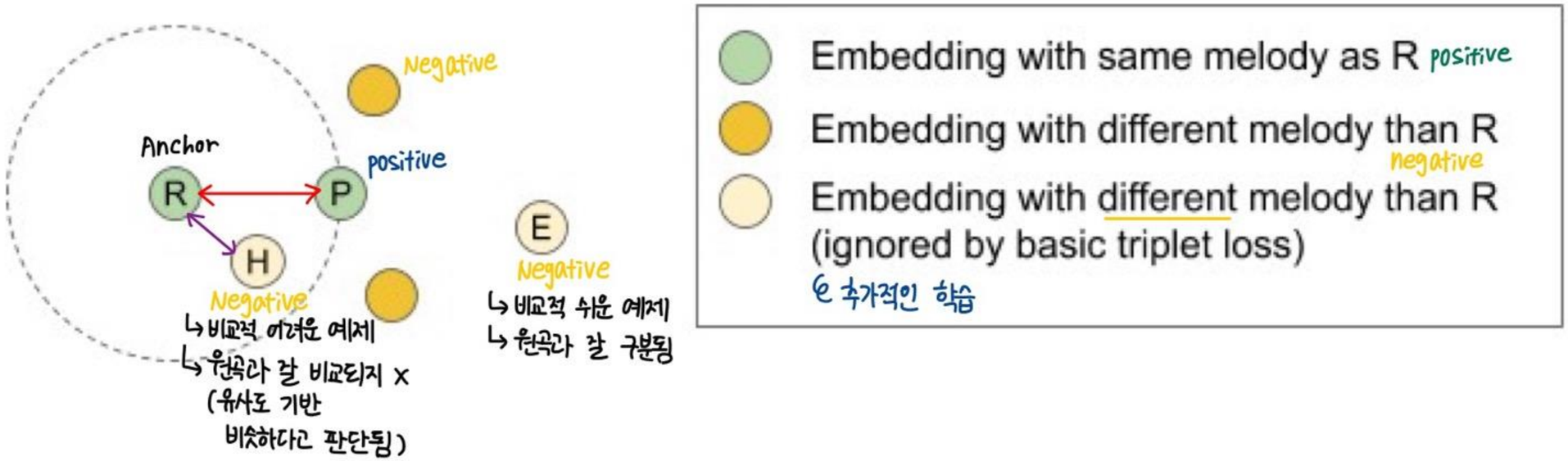
```python
# 모델을 학습 모드로 설정
model.train()

for epoch in range(num_epochs):
  print(f'========== Epoch {epoch + 1} ==========')
  for step, (anchor, positive, negative) in enumerate(trainloader, 1):
    optimizer.zero_grad()

    anchor, positive, negative = anchor.to(device), positive.to(device), negative.to(device)
    embedded_anchor, embedded_positive, embedded_negative = model(anchor), model(positive), model(negative)

    loss = triplet_loss(embedded_anchor, embedded_positive, embedded_negative)
    loss.backward()
    optimizer.step()

    loss_curve.append(loss.item())

    if step % log_steps == 0:
      print(f'Epoch {epoch + 1}, Step {step}, Loss: {loss.item()}')

  # 에폭이 끝날 때마다 모델 저장
  if (epoch + 1) % checkpoint == 0:
      torch.save(model.state_dict(), os.path.join(model_dir, f'model_weights_{epoch + 1}.pth'))

  print(f'Epoch {epoch + 1}, Loss: {loss.item()}')
  print()
```

TripletNetwork

학습 시 Embedding된 vector를
Tripletloss를 이용하여
원곡과 허밍 데이터 간의 거리를 좁힘

* Modeling

Tripletloss?



사용자의 흥얼거림(Anchor)과 원곡(Positive) 사이의 거리를 줄이고,
사용자의 흥얼거림과 다른 노래(Negative) 사이의 거리를 늘리는 방식으로 triplet loss를 적용

* Modeling

Tripletloss?

: 유클리디안 거리를 이용하여 거리 계산

$$\mathcal{L}(A, P, N) = \max(\| \mathrm{f}(A) - \mathrm{f}(P) \|_2 - \| \mathrm{f}(A) - \mathrm{f}(N) \|_2 + \alpha, 0)$$

L2 Norm

- A: anchor point(기준, 사용자의 흥얼거림)
- P: positive input(동일한 클래스, 원곡)
- N: negative input(다른 클래스, 다른 노래)
- alpha: margin(positive와 negative를 분리할 때 그 기준이 되는 거리)
- f: embedding

모델링_Inference / 실험

* Inference test

```python
def infer_with_filtering(model, test_mel, music_emb, info_df, user_genre, user_gender):
    with torch.no_grad():
        # 테스트 데이터에 대한 mel 벡터를 모델에 전달하여 임베딩 얻기
        test_embedding = model(test_mel.unsqueeze(0).to(device))

        # 필터링을 위한 데이터프레임 생성
        filtered_info_df = info_df[(info_df['genre'] == user_genre) & (info_df['gender'] == user_gender)]

        if filtered_info_df.empty:  # 필터링된 데이터가 없을 경우
            return None

        # 필터링된 데이터프레임의 id와 song_no를 일치시켜서 필터링된 음악 정보를 가져옴
        filtered_music_df = music_emb[music_emb['song_no'].isin(filtered_info_df['id'])].reset_index(drop=True)

        # 모든 원곡의 mel 벡터에 대해 임베딩을 계산하고 거리 측정
        distances = []
        for _, row in filtered_music_df.iterrows():
            #org_embedding = model(load_spectrogram(row['org']).unsqueeze(0).to(device))
            org_embedding = model(row['load_emb'].unsqueeze(0).to(device))
            dist = distance.cosine(test_embedding.cpu().detach().numpy().reshape(-1), org_embedding.cpu().detach().
            distances.append(dist)

        if not distances:  # 필터링된 데이터가 없을 경우
            return None

        # 거리가 가장 가까운 원곡의 정보 얻기
        closest_idx = distances.index(min(distances))
        closest_song_no = filtered_music_df.loc[closest_idx, 'song_no']
        first_song_info = info_df.loc[info_df['id'] == closest_song_no].iloc[0]
        print("== 첫번째로 유사한 원곡 정보 ==")
        print("곡 번호:", first_song_info['id'])
        print("제목:", first_song_info['Title'])
        print("장르:", first_song_info['genre'])
        print("성별:", first_song_info['gender'])
        print("가사:", first_song_info['Lyric'])
        print("")
```

- Info.csv에서 수집한 장르정보 & 성별정보를 활용하여 일차적으로 유사도 계산 대상을 축소
- DB에 대규모의 곡들이 저장되어 있더라도 <u>빠른 검색을 보장</u>

모델링_Inference / 실험

* Inference test

```python
def infer_with_filtering(model, test_mel, music_emb, info_df, user_genre, user_gender):
    with torch.no_grad():
        # 테스트 데이터에 대한 mel 벡터를 모델에 전달하여 임베딩 얻기
        test_embedding = model(test_mel.unsqueeze(0).to(device))

        # 필터링을 위한 데이터프레임 생성
        filtered_info_df = info_df[(info_df['genre'] == user_genre) & (info_df['gender'] == user_gender)]

        if filtered_info_df.empty:  # 필터링된 데이터가 없을 경우
            return None

        # 필터링된 데이터프레임의 id와 song_no를 일치시켜서 필터링된 음악 정보를 가져옴
        filtered_music_df = music_emb[music_emb['song_no'].isin(filtered_info_df['id'])].reset_index(drop=True)

        # 모든 원곡의 mel 벡터에 대해 임베딩을 계산하고 거리 측정
        distances = []
        for _, row in filtered_music_df.iterrows():
            #org_embedding = model(load_spectrogram(row['org']).unsqueeze(0).to(device))
            org_embedding = model(row['load_emb'].unsqueeze(0).to(device))
            dist = distance.cosine(test_embedding.cpu().detach().numpy().reshape(-1), org_embedding.cpu().detach().
            distances.append(dist)

        if not distances:  # 필터링된 데이터가 없을 경우
            return None

        # 거리가 가장 가까운 원곡의 정보 얻기
        closest_idx = distances.index(min(distances))
        closest_song_no = filtered_music_df.loc[closest_idx, 'song_no']
        first_song_info = info_df.loc[info_df['id'] == closest_song_no].iloc[0]
        print("== 첫번째로 유사한 원곡 정보 ==")
        print("곡 번호:", first_song_info['id'])
        print("제목:", first_song_info['Title'])
        print("장르:", first_song_info['genre'])
        print("성별:", first_song_info['gender'])
        print("가사:", first_song_info['Lyric'])
        print("")
```

Cosine similarity

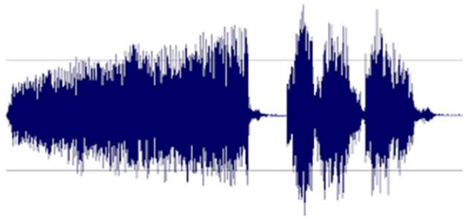
Test dataset 결과 7.3%의 낮은 정확도

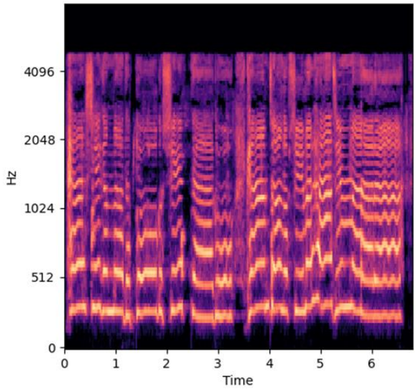
Pairwise similarity

Test dataset 결과 99.5%의 높은 정확도

모델링_Inference / 실험

* Inference_실제 허밍





Create embedding vector

& Pairwise similarity

모델의 정확도에 비해 실제 허밍데이터에 대해서 잘 추론하지는 못했음

본 프로젝트에서 생성 및 학습시킨 허밍 데이터와 실제 우리의 허밍이 완전히 유사하지는 않기 때문

모델 학습이 성공적으로 이뤄졌다는 점에서 이후 허밍 데이터셋을 보완한다면 좋은 결과가 나올 수 있다는 시사점을 지님

모델링_Inference / 실험

* Inference_실제 허밍

```
[43] test_hum = '/content/drive/MyDrive/Hmm2Song/차수빈/test_voice/Ditto/Ditto_13.mp3
     test_mel = test_spectrogram(test_hum)

     user_genre = input("장르를 입력하세요: ")
     user_gender = input("성별을 입력하세요: ")
```

뉴진스 –Ditto에 대한 커버 음악 허밍 음원

```
장르를 입력하세요: Dance
성별을 입력하세요: 여성
```

장르 / 가수 성별 입력

```
## 추론 결과 얻기
inferred_info = infer_with_filtering(model, test_mel, music_emb, song_info, user_genre, user_gender)

if inferred_info.any():
    print("== 가장 유사한 원곡 정보 ==")
    print("곡 번호:", inferred_info['id'])
    print("제목:", inferred_info['Title'])
    print("가수:", inferred_info['Singer'])
    print("장르:", inferred_info['genre'])
    print("가수 성별:", inferred_info['gender'])
    print("가사:", inferred_info['Lyric'])
else:
    print("입력한 장르와 성별에 해당하는 음악이 없습니다.")
```

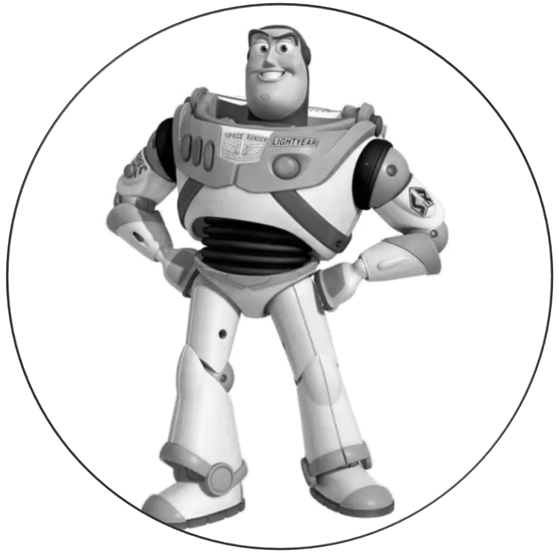
성공적으로 추론

```
== 가장 유사한 원곡 정보 ==
곡 번호: 548
제목: Ditto
가수: NewJeans
장르: Dance
가수 성별: 여성
가사: Woo woo woo woo ooh Woo woo woo woo Stay in the middle Like you a little Don’ t want no riddle 말해줘 say it back Oh say it ditto 아침은 너무 멀어 So say it ditto 훌쩍 커버렸어 함께한 기억처럼
```

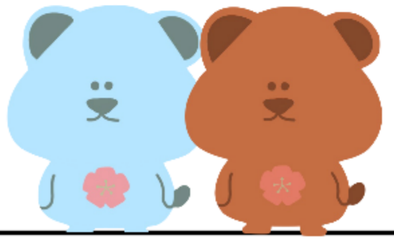
OUR TEAM MEMBER

휴먼기계바이오공학부
박지연

통계학과
차수빈



통계학과
강효은



EWHA EURON

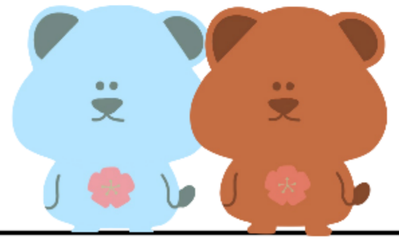
THANK YOU

Notion

https://military-soybean-f73.notion.site/Euron-5th-team-project-09ecd3c2ba2b492293d8ec398dc740a1?pvs=4

GitHub

https://github.com/hyooz-Kang/Hmm2Song



EWHA EURON