ALL THAT K-POP X MORE THAN K-POP

weAone

SHOW ME WHAT YOU GOT

LIST

우린 모두 케이팝으로 연결되어 있어

1. 나의 최애가 내 최애곡을?

2. 내 최애가 춘 춤의 안무가?!

3. 내 무드를 알려줘!

4. 내 곡과 어울리는 곡?!

주제 선정 배경

우리는 K-pop을 '듣는다'
좀 더 즐겨볼 수 없을까?
무대영상, 교차편집, 숨듣명,,,
케이팝러에게는 아쉽다,,
그렇다고? 매번 콘서트를 갈 수 없는 노릇!

우린 '딥러닝'을 통해 k-pop을 즐겨보고자 한다.

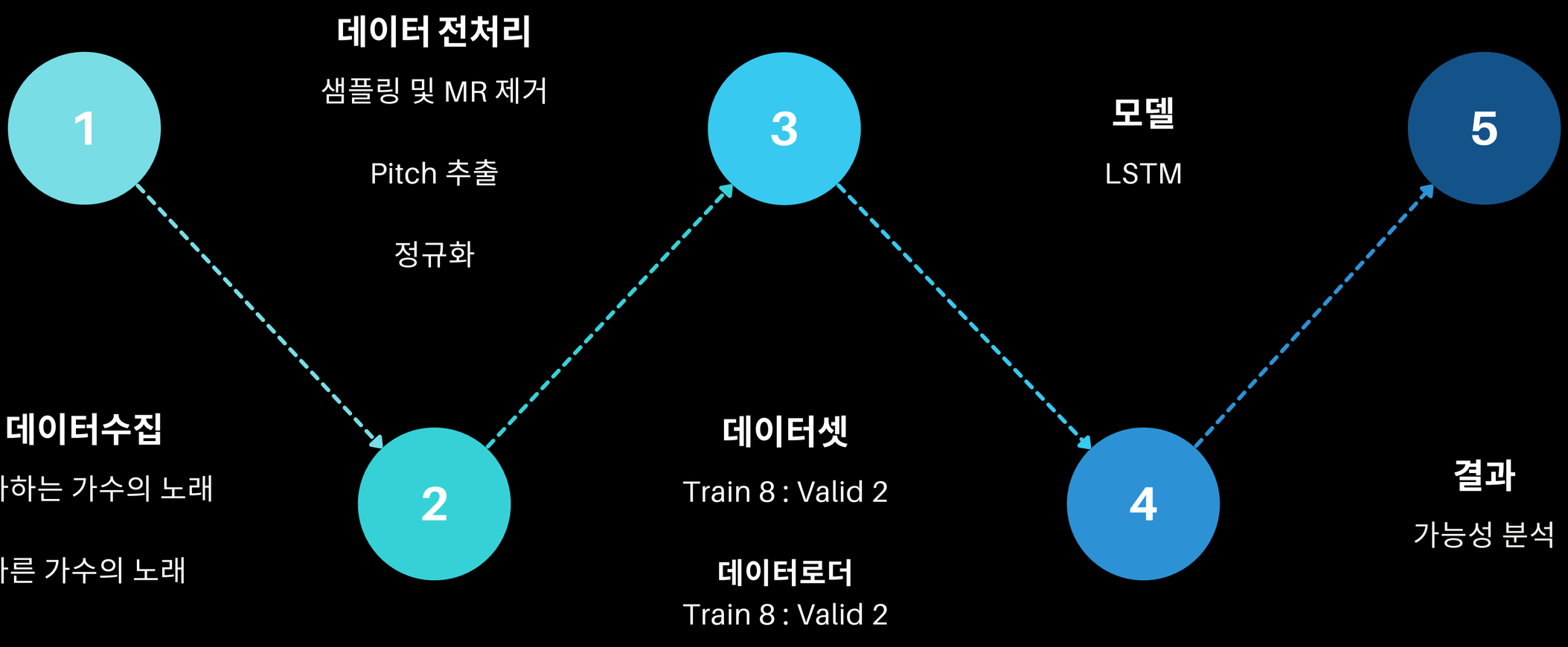
WHITEBOARD PAGE

나의 최애가
나의 최애곡을
불러줄 수 있을까?

발표자 : 이승민

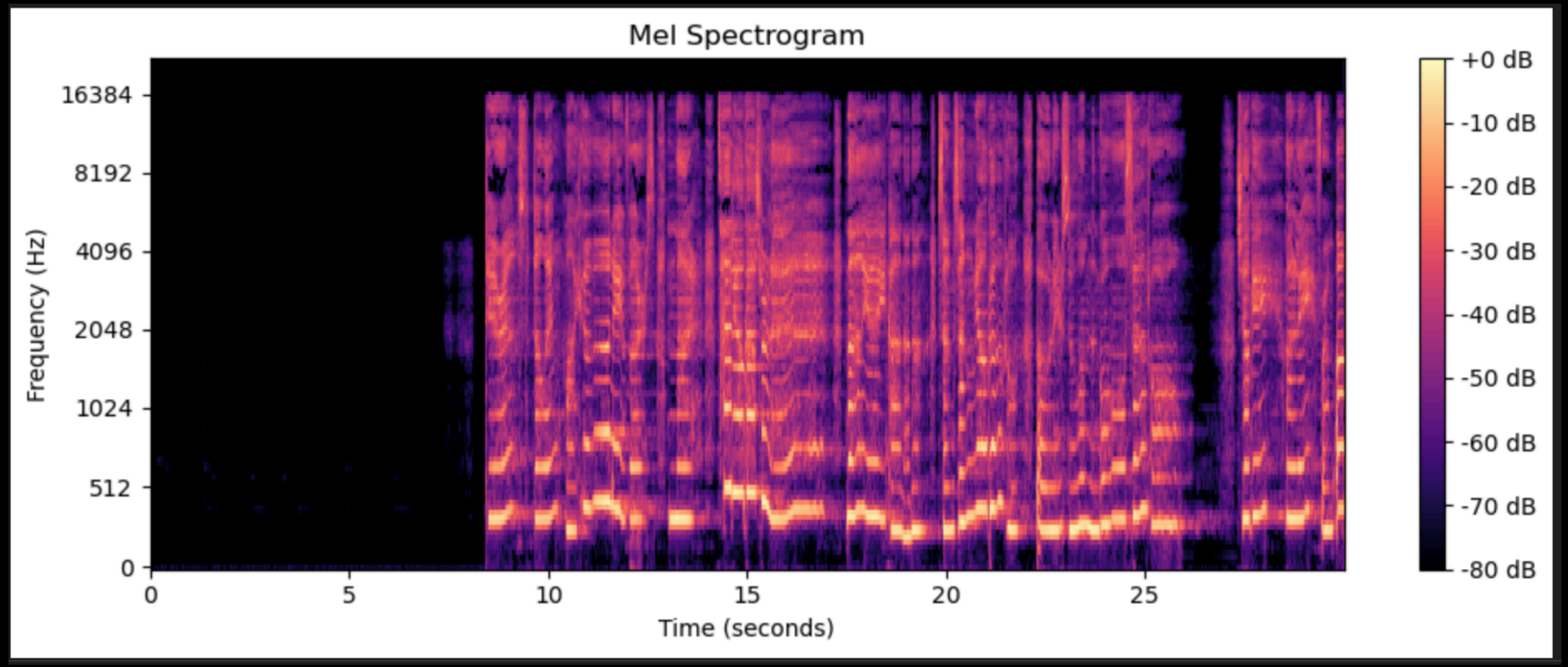
소주제 선정 배경 : 가수들의 커버

Work Flow


1
데이터수집
좋아하는 가수의 노래
다른 가수의 노래
2
데이터 전처리
샘플링 및 MR 제거
Pitch 추출
정규화
3
데이터셋
Train 8 : Valid 2
데이터로더
Train 8 : Valid 2
4
모델
LSTM
5
결과
가능성 분석

모델 방향성 고민

CNN vs RNN(LSTM) v



RNN(LSTM) 선정 이유 : 확장성 고려

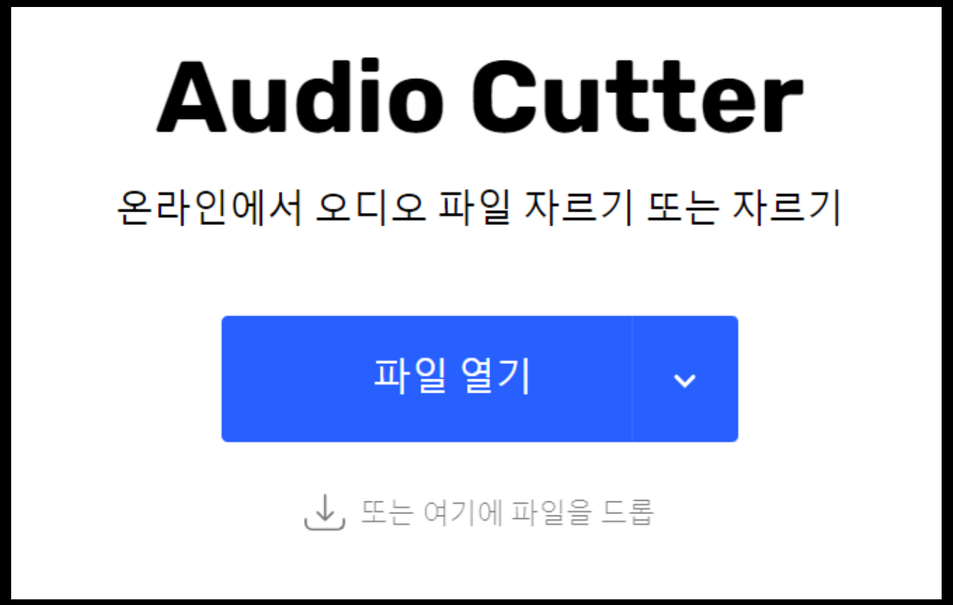
-> 고음 지속 능력에 대한 판단 모델 등 음악적 능력을 판단하는 모델 구축 시
위 사진과 같이 다양한 필터를 통해 특징을 추출해내는 CNN보다
시간적 의존성을 학습하고 장기 의존성 문제를 해결하는 RNN이 유리함

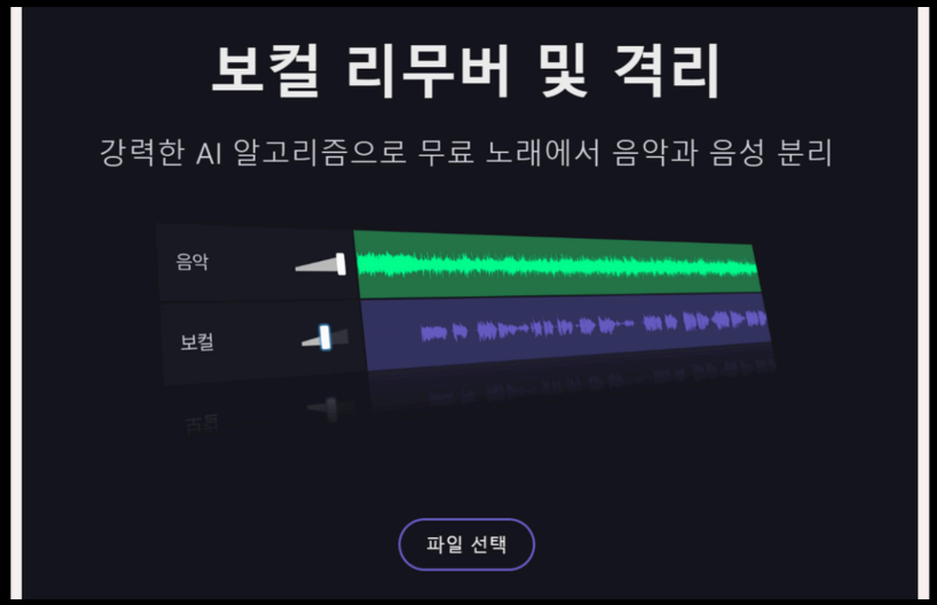
데이터 수집

1. 좋아하는 가수 선정
2. 가수의 노래 10곡 선정
3. 최애곡 (다른 가수) 선정

데이터 전처리

- 샘플링 및 MR 제거





데이터 전처리

- Pitch 추출

librosa 모듈 사용

python package for music and audio analysis.

데이터 전처리

- 정규화

피치 데이터의 스케일이 2000을 넘어가서
Loss값이 크게 잡힐 수 있으므로 스케일링 진행

랜덤으로 추출하였기에 정규분포를 따르지 않고, 최저-최고음의
피치를 뽑기에 이상치가 없으므로 Min-Max 스케일링 진행

데이터셋, 데이터로더 생성

```
# 데이터셋 생성
train_dataset = VocalRangeDataset(torch.tensor(train_features, dtype=torch.float32), torch.tensor(train_targets, dtype=torch.float32))
val_dataset = VocalRangeDataset(torch.tensor(val_features, dtype=torch.float32), torch.tensor(val_targets, dtype=torch.float32))
```

✓ 0.0s

```
# 데이터로더 생성
train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=4, shuffle=False)
```

✓ 0.0s

모델 생성 및 학습

```python
import torch.nn as nn

class VocalRangeModel(nn.Module):
    def __init__(self):
        super(VocalRangeModel, self).__init__()
        self.lstm = nn.LSTM(input_size=2, hidden_size=50, num_layers=2, batch_first=True)
        self.fc = nn.Linear(50, 2)  # 최종 출력은 최저음과 최고음

    def forward(self, x):
        # LSTM 레이어
        out, (hidden, cell) = self.lstm(x)
        # 마지막 시간 단계의 출력을 사용
        out = self.fc(out[:, -1, :])
        return out
```

```python
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)


# 손실을 저장할 리스트 초기화
train_losses = []
val_losses = []

# 학습 루프
EPOCHS = 50

for epoch in range(EPOCHS):  # 에포크 수 설정
    model.train()
    total_train_loss = 0
    num_batches = 0

    for data, targets in dataloader:
        optimizer.zero_grad()
        # 입력 데이터 차원 조정: 배치 크기 x 시퀀스 길이 x 특성 수
        outputs = model(data.unsqueeze(1))
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()
        total_train_loss += loss.item()
        num_batches += 1

    avg_train_loss = total_train_loss / num_batches
    train_losses.append(avg_train_loss)  # 훈련 손실 저장


    model.eval()
    val_loss = 0
    total_val_loss = 0
    num_val_batches = 0

    with torch.no_grad():
        for data, targets in val_loader:
            outputs = model(data.unsqueeze(1))
            val_loss += criterion(outputs, targets).item()
            total_val_loss += val_loss
            num_val_batches += 1

        avg_val_loss = total_val_loss / num_val_batches
        val_losses.append(avg_val_loss)  # 검증 손실 저장

    print(f'Epoch {epoch+1: >3},    Training Loss = {avg_train_loss:.8f},    Validation Loss = {avg_val_loss:.8f}')
```

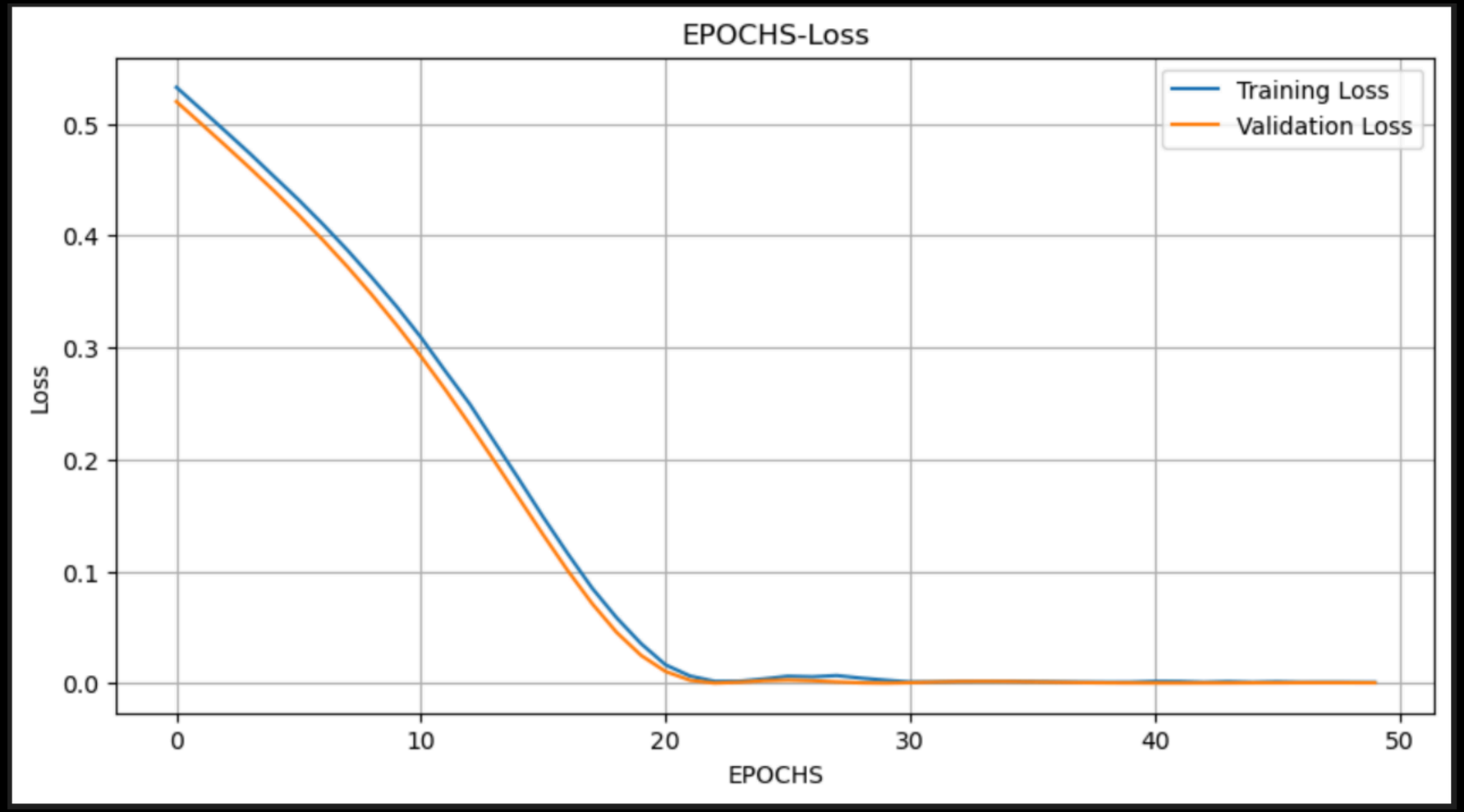
모델 생성 및 학습

- 훈련 결과

```
Epoch   1,    Training Loss = 0.53249474,    Validation Loss = 0.51969087
Epoch   2,    Training Loss = 0.51294770,    Validation Loss = 0.50012767
Epoch   3,    Training Loss = 0.49325164,    Validation Loss = 0.48044139
Epoch   4,    Training Loss = 0.47335348,    Validation Loss = 0.46037501
Epoch   5,    Training Loss = 0.45254300,    Validation Loss = 0.43969887
Epoch   6,    Training Loss = 0.43181786,    Validation Loss = 0.41819364
Epoch   7,    Training Loss = 0.40984244,    Validation Loss = 0.39565051
Epoch   8,    Training Loss = 0.38676200,    Validation Loss = 0.37189224
Epoch   9,    Training Loss = 0.36224674,    Validation Loss = 0.34676957
Epoch  10,    Training Loss = 0.33661284,    Validation Loss = 0.32014042
Epoch  11,    Training Loss = 0.30893979,    Validation Loss = 0.29193056
Epoch  12,    Training Loss = 0.27862709,    Validation Loss = 0.26212585
Epoch  13,    Training Loss = 0.24920354,    Validation Loss = 0.23085225
Epoch  14,    Training Loss = 0.21569584,    Validation Loss = 0.19844040
Epoch  15,    Training Loss = 0.18234638,    Validation Loss = 0.16538629
Epoch  16,    Training Loss = 0.14829959,    Validation Loss = 0.13242280
Epoch  17,    Training Loss = 0.11574154,    Validation Loss = 0.10054309
Epoch  18,    Training Loss = 0.08484210,    Validation Loss = 0.07103611
Epoch  19,    Training Loss = 0.05841110,    Validation Loss = 0.04532708
Epoch  20,    Training Loss = 0.03526518,    Validation Loss = 0.02475761
Epoch  21,    Training Loss = 0.01633572,    Validation Loss = 0.01040072
Epoch  22,    Training Loss = 0.00634489,    Validation Loss = 0.00254892
Epoch  23,    Training Loss = 0.00160593,    Validation Loss = 0.00004415
Epoch  24,    Training Loss = 0.00161453,    Validation Loss = 0.00075567
Epoch  25,    Training Loss = 0.00368224,    Validation Loss = 0.00219904
...
Epoch  47,    Training Loss = 0.00077180,    Validation Loss = 0.00039522
Epoch  48,    Training Loss = 0.00083761,    Validation Loss = 0.00042096
Epoch  49,    Training Loss = 0.00078232,    Validation Loss = 0.00036697
Epoch  50,    Training Loss = 0.00071977,    Validation Loss = 0.00029858
```

모델 생성 및 학습

- 시각화


EPOCHS-Loss
Training Loss
Validation Loss
Loss
0.0
0.1
0.2
0.3
0.4
0.5
0
10
20
30
40
50
EPOCHS

모델 생성 및 학습

- 역스케일링 후 음역대 추출

```python
# 역스케일링 함수
def unscale_pitch(scaled_pitch, min_pitch, max_pitch):
    # min-max 스케일링한 값을 다시 역스케일링
    return scaled_pitch * (max_pitch - min_pitch) + min_pitch
```

```python
# 평균 계산 및 역스케일링
final_min_freq = np.mean(final_predicted_min_freqs)
final_max_freq = np.mean(final_predicted_max_freqs)

# 역스케일링 적용
final_min_freq = unscale_pitch(final_min_freq, min_pitch, max_pitch)
final_max_freq = unscale_pitch(final_max_freq, min_pitch, max_pitch)

# 학습 완료 후 최종 예측된 음역대를 노트로 변환
final_min_note = freq_to_note(final_min_freq)
final_max_note = freq_to_note(final_max_freq)

# 최종 예측된 음역대 출력
print(f"Final Predicted Vocal Range: {final_min_note} to {final_max_note}")
```

✓ 0.0s

```
Final Predicted Vocal Range: C♯2 to A6
```

결과

내가 좋아하는 노래,
불러줄 수 있어?

```
# 부를 수 있는 확률 계산
probability = calculate_singing_probability((artist_min_note, artist_max_note), (song_min_note, song_max_note))
print(f"당신의 최애가 최애곡을 부를 수 있는 확률! : {probability:.2f}%")
```

✓ 0.0s

```
당신의 최애가 최애곡을 부를 수 있는 확률! : 0.93%
```

결과

- 다른 샘플로 테스트

```
Epoch   1,    Training Loss = 0.42125759,    Validation Loss = 0.41288865
Epoch   2,    Training Loss = 0.40998022,    Validation Loss = 0.40192482
Epoch   3,    Training Loss = 0.39893518,    Validation Loss = 0.39112866
Epoch   4,    Training Loss = 0.38807084,    Validation Loss = 0.38043654
Epoch   5,    Training Loss = 0.37724714,    Validation Loss = 0.36977351
Epoch   6,    Training Loss = 0.36646114,    Validation Loss = 0.35906652
Epoch   7,    Training Loss = 0.35560516,    Validation Loss = 0.34824914
Epoch   8,    Training Loss = 0.34462258,    Validation Loss = 0.33726230
Epoch   9,    Training Loss = 0.33342491,    Validation Loss = 0.32605183
Epoch  10,    Training Loss = 0.32199001,    Validation Loss = 0.31456518
Epoch  11,    Training Loss = 0.31025116,    Validation Loss = 0.30275315
Epoch  12,    Training Loss = 0.29812145,    Validation Loss = 0.29057014
Epoch  13,    Training Loss = 0.28563394,    Validation Loss = 0.27797085
Epoch  14,    Training Loss = 0.27271423,    Validation Loss = 0.26491749
Epoch  15,    Training Loss = 0.25928850,    Validation Loss = 0.25138161
Epoch  16,    Training Loss = 0.24538092,    Validation Loss = 0.23734275
Epoch  17,    Training Loss = 0.23089329,    Validation Loss = 0.22279617
Epoch  18,    Training Loss = 0.21589912,    Validation Loss = 0.20774868
Epoch  19,    Training Loss = 0.20041274,    Validation Loss = 0.19222689
Epoch  20,    Training Loss = 0.18436478,    Validation Loss = 0.17628539
Epoch  21,    Training Loss = 0.16797398,    Validation Loss = 0.15999523
Epoch  22,    Training Loss = 0.15123786,    Validation Loss = 0.14346419
Epoch  23,    Training Loss = 0.13434166,    Validation Loss = 0.12682879
Epoch  24,    Training Loss = 0.11747998,    Validation Loss = 0.11025427
Epoch  25,    Training Loss = 0.10068030,    Validation Loss = 0.09394236
...
Epoch  47,    Training Loss = 0.00040049,    Validation Loss = 0.00066387
Epoch  48,    Training Loss = 0.00039495,    Validation Loss = 0.00099749
Epoch  49,    Training Loss = 0.00044244,    Validation Loss = 0.00126730
Epoch  50,    Training Loss = 0.00057142,    Validation Loss = 0.00146798
```

결과

- 다른 샘플로 테스트

```
# 최종 예측된 음역대 출력
print(f"Final Predicted Vocal Range: {final_min_note} to {final_max_note}")
```

✓ 0.0s

```
Final Predicted Vocal Range: C2 to A♯5
```

```
# 부를 수 있는 확률 계산
probability = calculate_singing_probability((artist_min_note, artist_max_note), (song_min_note, song_max_note))
print(f"일반인이 최애곡을 부를 수 있는 확률! : {probability:.2f}%")
```

✓ 0.0s

```
일반인이 최애곡을 부를 수 있는 확률! : 0.77%
```

결론

- 음역대를 추출하려면 역스케일링을
  통한 주파수 회복이 중요하다.

- 음역대만으로 접근을 한다면
높은 정확도를 뽑아낼 수 있다.

- 다양한 요소를 고려할 시 시계열 처리로 접근
  해야하므로 LSTM과 RNU로 접근해야한다.

WHITEBOARD PAGE

안무 영상으로 안무가 분류하기

라치카, 바다, 배윤정, 위뎀보이즈, 최영준

발표자 : 이시명


BreakNew


SUDDEN ATTACK


1st Look

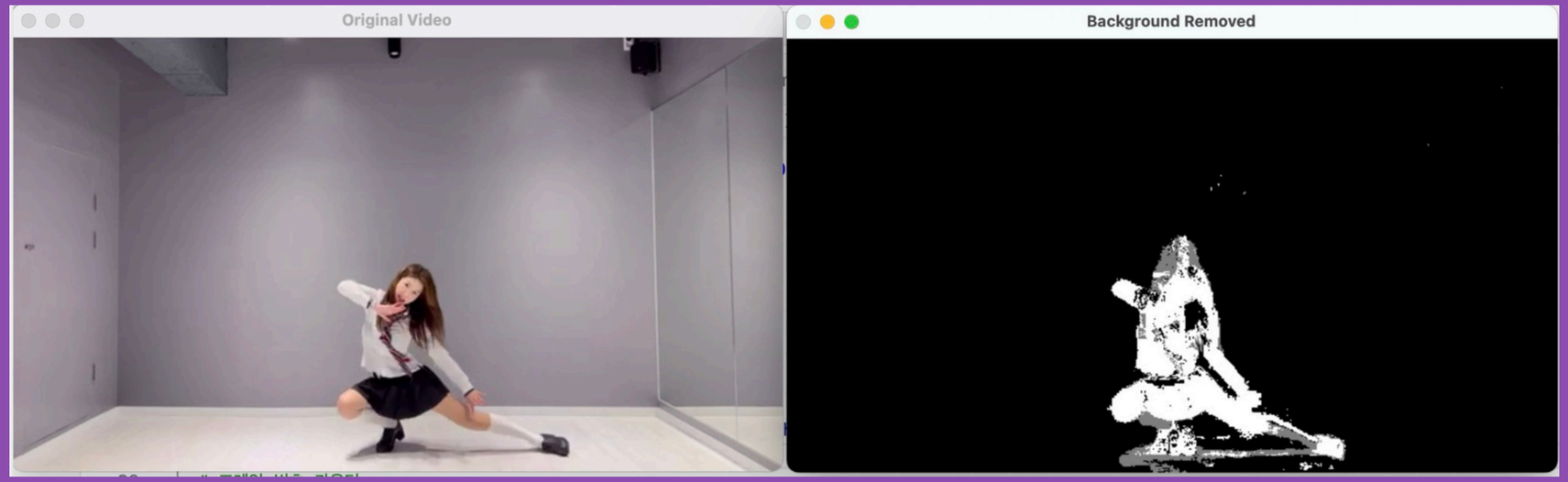
영상 데이터셋 구성

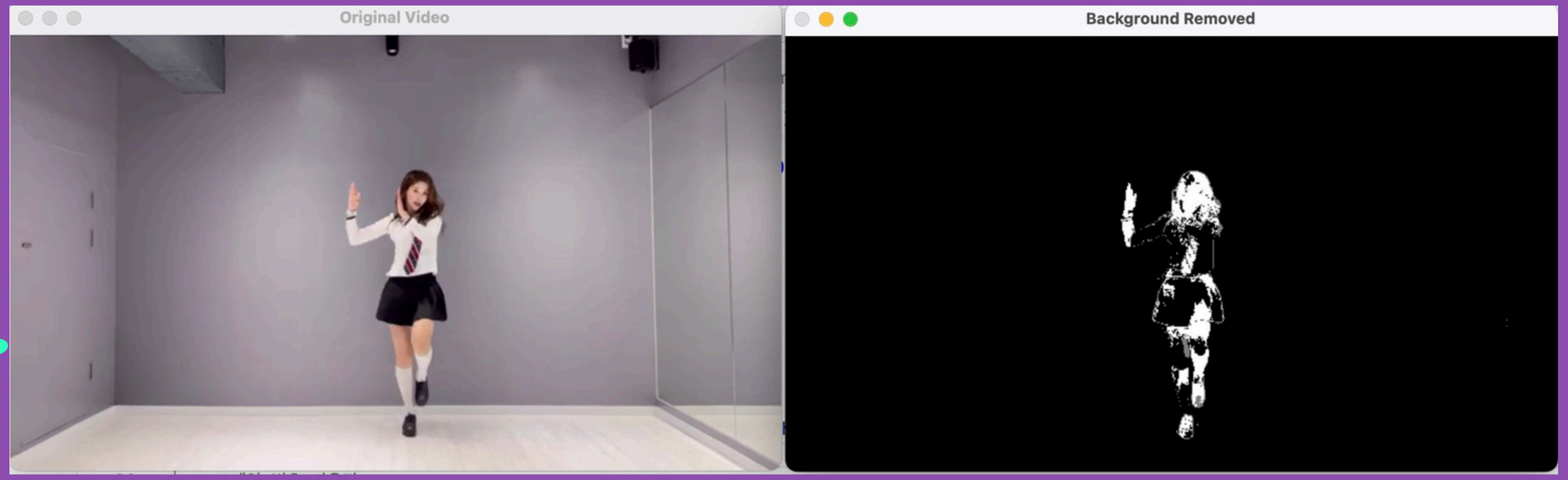
안무팀 별 최소 50개 영상 준비

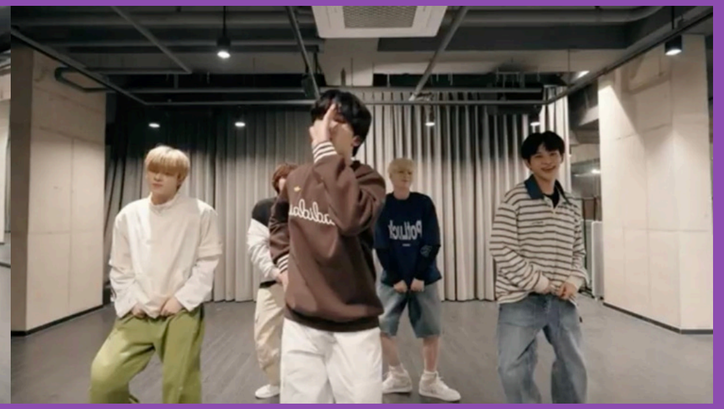
30개 영상으로 데이터셋 구성

20:5:5 train/valid/test 구성

OPEN CV를 이용한 배경 제거



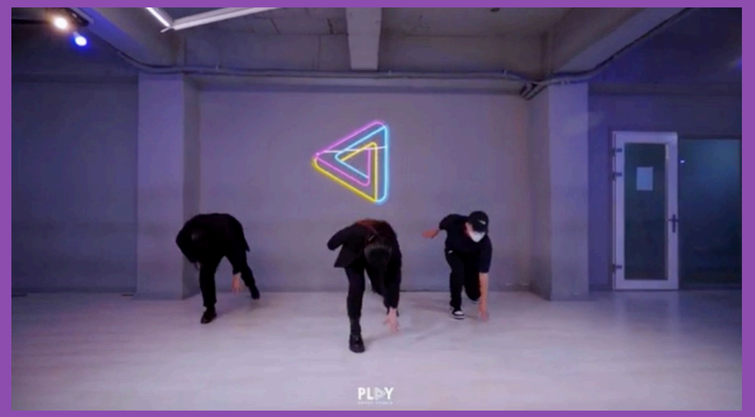




카메라 무빙 없을 것

최소한의 인원만

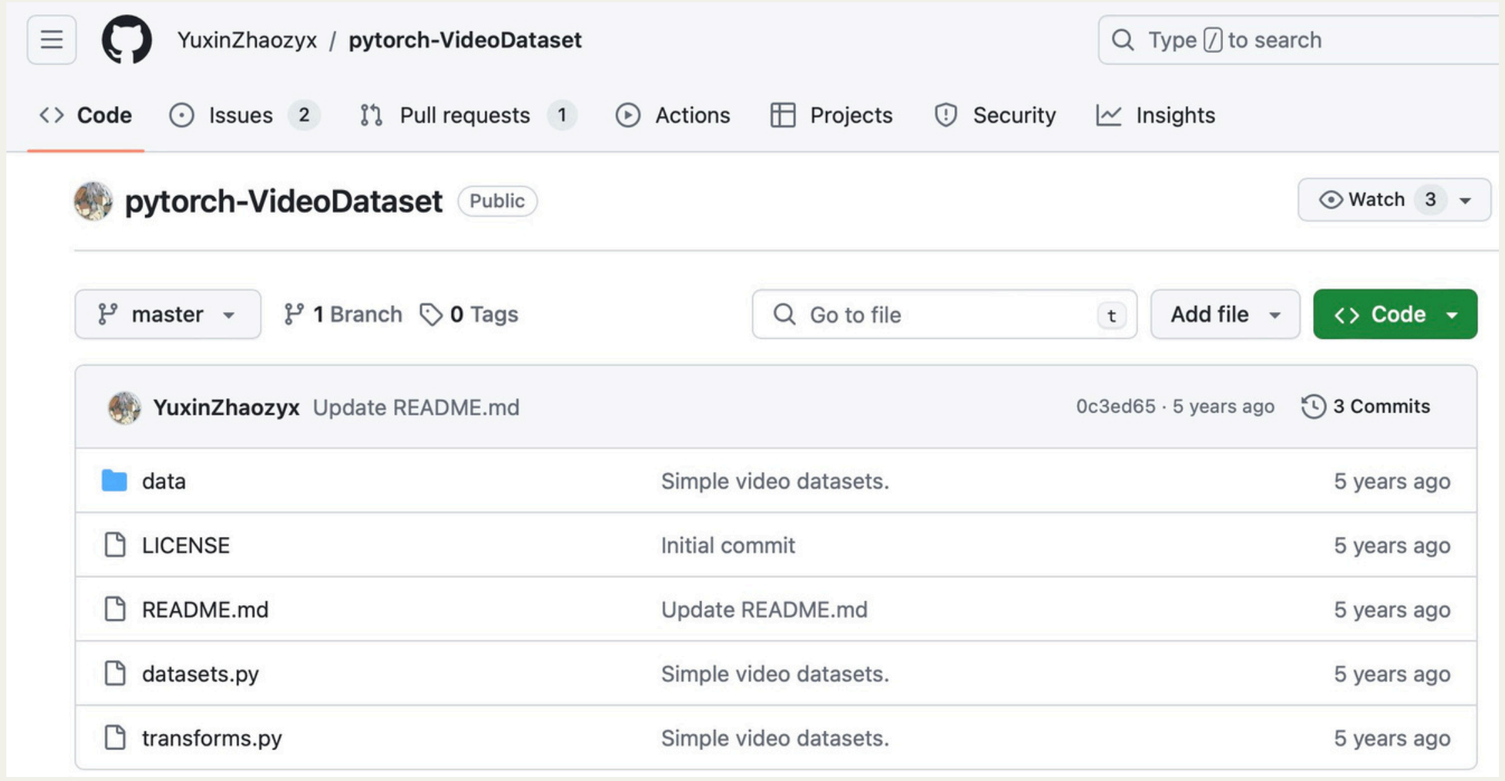
거울에 반사되는 모습이 없거나
있더라도 고정적일 것



학원 로고 등이 없을 것

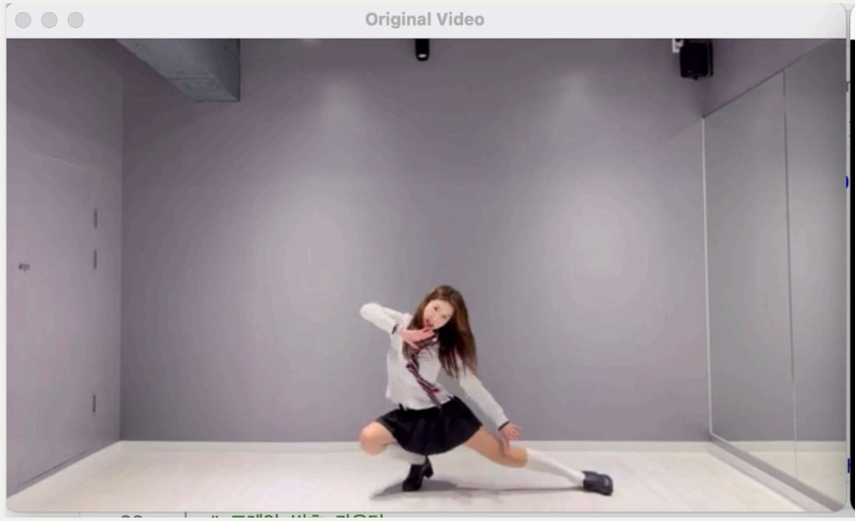
개인 안무 커버 거울모드 영상 위주로 선정

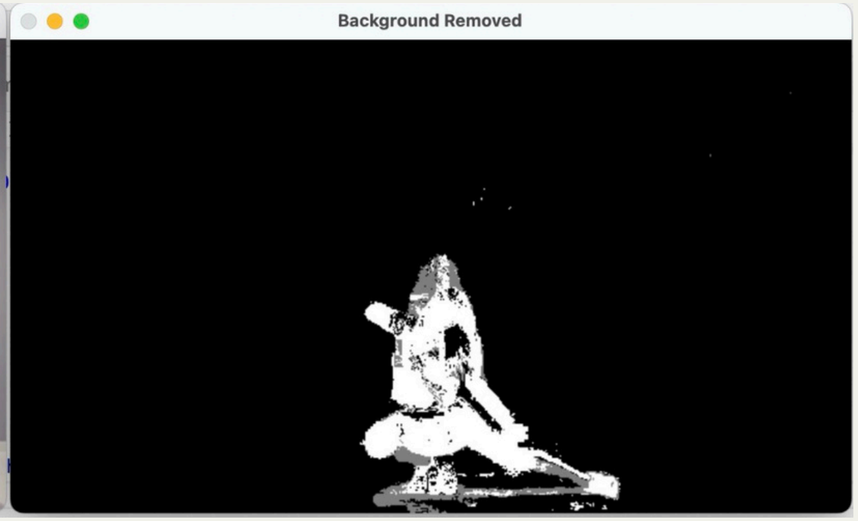
VIDEO TO TENSOR



하루를 투자했으나 5차원 데이터 이해가 안 돼서 실패

영상 처리과정





원본 영상

흑백 영상

흑백 프레임

CrossEntropyLoss, AdamW

```python
class CNN(nn.Module):
    def __init__(self, num_classes):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)  # Output size: (32, 36, 64)

        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)  # Output size: (64, 18, 32)

        self.fc1 = nn.Linear(64 * 16 * 8, 128)  #After pool2: torch.Size([9, 64, 16, 8]) 64 * 16* 8 꼴 하기
        self.relu3 = nn.ReLU()
        self.fc2 = nn.Linear(128, num_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu1(x)
        x = self.pool1(x)
        # print("After pool1:", x.shape)  # Debug: print the shape


        x = self.conv2(x)
        x = self.relu2(x)
        x = self.pool2(x)
        # print("After pool2:", x.shape)  # Debug: print the shape

        x = x.view(x.size(0), -1)
        # print("Before FC:", x.shape)  # Debug: print the shape

        x = self.fc1(x)
        x = self.relu3(x)
        x = self.fc2(x)
        return x
```

CNN

```python
class CRNN(nn.Module):
    def __init__(self, num_classes):
        super(CRNN, self).__init__()
        self.num_classes = num_classes
        # CNN Layers
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

        # LSTM Layers
        self.lstm = nn.LSTM(input_size=64 * 16 * 8, hidden_size=256, num_layers=1, batch_first=True)
        self.fc = nn.Linear(256, num_classes)

    def forward(self, x):
        batch_size, seq_len, C, H, W = x.shape
        c_in = x.view(batch_size * seq_len, C, H, W)  # Combine batch and seq_len for CNN processing
        x = self.conv1(c_in)
        x = self.relu1(x)
        x = self.pool1(x)
        x = self.conv2(x)
        x = self.relu2(x)
        x = self.pool2(x)

        # Prepare for LSTM
        x = x.view(batch_size, seq_len, -1)  # Combine batch and seq_len for LSTM processing
        lstm_out, (h_n, c_n) = self.lstm(x)
        # We use the last hidden state to classify
        x = self.fc(lstm_out[:, -1, :])
        return x
```

CRNN

모델별 결과

CNN

```
Epoch [20/30], Train Loss: 0.1587, Train Accuracy: 0.9441, Valid Loss: 0.1602, Valid Accuracy: 0.9434
Epoch [21/30], Train Loss: 0.1575, Train Accuracy: 0.9445, Valid Loss: 0.1554, Valid Accuracy: 0.9453
Epoch [22/30], Train Loss: 0.1567, Train Accuracy: 0.9450, Valid Loss: 0.1545, Valid Accuracy: 0.9468
Epoch [23/30], Train Loss: 0.1558, Train Accuracy: 0.9453, Valid Loss: 0.1634, Valid Accuracy: 0.9432
Epoch [24/30], Train Loss: 0.1544, Train Accuracy: 0.9457, Valid Loss: 0.1667, Valid Accuracy: 0.9430
Epoch [25/30], Train Loss: 0.1542, Train Accuracy: 0.9457, Valid Loss: 0.1696, Valid Accuracy: 0.9404
Epoch [26/30], Train Loss: 0.1544, Train Accuracy: 0.9456, Valid Loss: 0.1663, Valid Accuracy: 0.9418
Epoch [27/30], Train Loss: 0.1529, Train Accuracy: 0.9462, Valid Loss: 0.1613, Valid Accuracy: 0.9433
Epoch [28/30], Train Loss: 0.1532, Train Accuracy: 0.9464, Valid Loss: 0.1442, Valid Accuracy: 0.9508
```

Test Loss: 1.6101
Test Accuracy: 0.2265

CRNN

```
Epoch [1/20], Train Loss: 0.6385, Train Accuracy: 0.7537, Valid Loss: 0.3680, Valid Accuracy: 0.8667
Epoch [2/20], Train Loss: 0.2778, Train Accuracy: 0.9006, Valid Loss: 0.2569, Valid Accuracy: 0.9089
Epoch [3/20], Train Loss: 0.2086, Train Accuracy: 0.9264, Valid Loss: 0.1988, Valid Accuracy: 0.9307
Epoch [4/20], Train Loss: 0.1804, Train Accuracy: 0.9365, Valid Loss: 0.1790, Valid Accuracy: 0.9376
Epoch [5/20], Train Loss: 0.1661, Train Accuracy: 0.9420, Valid Loss: 0.1703, Valid Accuracy: 0.9416
Epoch [6/20], Train Loss: 0.1541, Train Accuracy: 0.9462, Valid Loss: 0.1562, Valid Accuracy: 0.9459
Epoch [7/20], Train Loss: 0.1471, Train Accuracy: 0.9487, Valid Loss: 0.1581, Valid Accuracy: 0.9440
Epoch [8/20], Train Loss: 0.1413, Train Accuracy: 0.9512, Valid Loss: 0.1592, Valid Accuracy: 0.9431
Epoch [9/20], Train Loss: 0.1360, Train Accuracy: 0.9532, Valid Loss: 0.1457, Valid Accuracy: 0.9502
Epoch [10/20], Train Loss: 0.1320, Train Accuracy: 0.9543, Valid Loss: 0.1441, Valid Accuracy: 0.9502
```

Test Loss: 0.1548
Test Accuracy: 0.9460

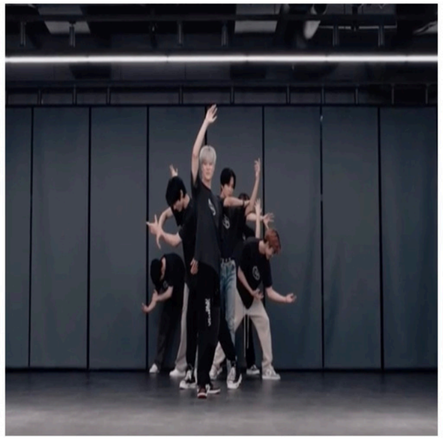
과대적합? 그러나 가능성은 있다

인원 수가 많으면 위뎀보이즈 or 최영준 ?

그러나 해당 댄서의 시그니처 포즈라고 예상되는 것들을 넣으면 예측률이 높다

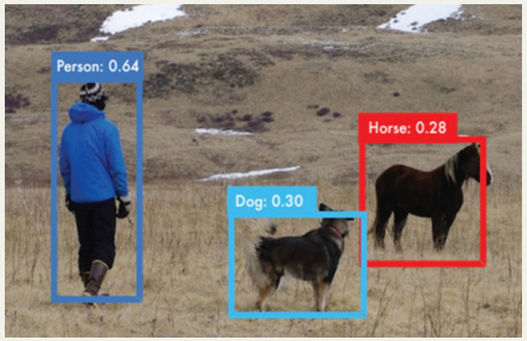
Prediction: 배윤정 안무가입니다.

Prediction: 바다 안무가입니다.



Prediction: 최영준 안무가입니다.

발전시킬 점



YuxinZhaozyx / pytorch-VideoDataset

Code Issues 2 Pull requests 1

pytorch-VideoDataset Public

YOLO를 써서 남은 배경정보도 모두 없애버린다면?

VideoToTensor여야
전체적인 움직임에 대한 이해가 생기지 않을까?

안무가보다는 장르적인 특성을 인식하는 게 아닐까?

WHITEBOARD PAGE

사진을 통한 무드 분석과 노래 추천

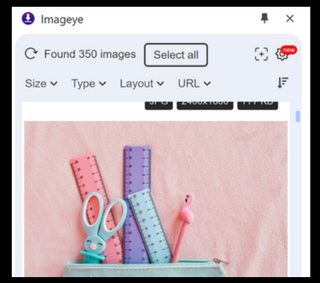
발표자 : 임소영

자료 준비

크롬에서 키워드 검색
(ex. angry mood)

▼

Imageye 이용하여 다운로드



▼

다운로드 파일 중 jpg가 아닌 파일 형식과 크기가 너무 작거나 글씨로 채워진 사진들은 delete

전처리

1) 사진크기 (32,32)로 resize
2) 텐서화
3) Normalize
   (0.485, 0.456, 0.406),
   (0.229, 0.224, 0.225)

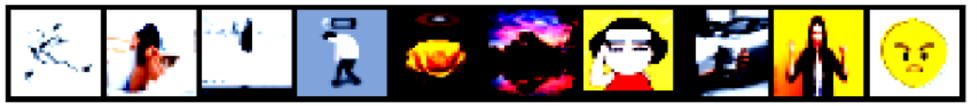
Dataset

- Train dataset = 0.7
- valid dataset = 0.1
- test dataset = 0.2

Dataloader

- batchsize = 10
- 1개의 배치 안에 있는 이미지 확인

```python
def show_batch(dl):
    """Plot images grid of single batch"""
    for (images, labels) in dl:
        fig,ax = plt.subplots(figsize = (16,12))
        ax.set_xticks([])
        ax.set_yticks([])
        ax.imshow(make_grid(images,nrow=16).permute(1,2,0))
        break

show_batch(train_dl)
```



model class 생성

mood_model(

(conv1): Conv2d(3, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

(conv2): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

(pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

(fc1): Linear(in_features=1024, out_features=64, bias=True)

(fc2): Linear(in_features=64, out_features=32, bias=True)

(fc3): Linear(in_features=32, out_features=8, bias=True)

)

변수 지정

- device 설정
- model 변수 지정
- 손실함수 : CrossEntropyLoss
- optimizer : Adam (lr = 0.01)
- scheduler : MultiStepLR

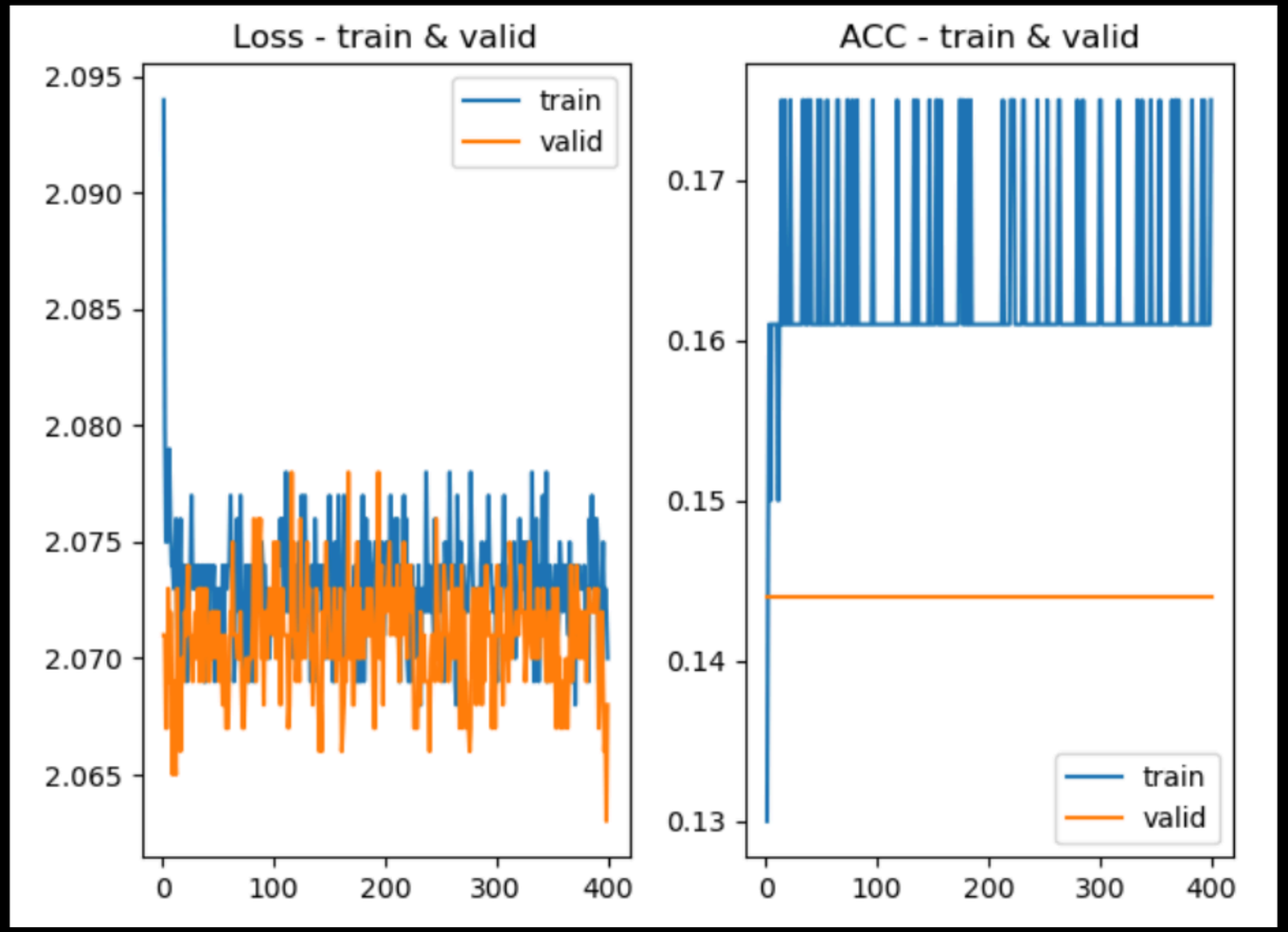
학습, 검증, 예측 함수 생성

400 epoch

[valid]

loss = 2

acc = 0.14



predict

predict loss : 2.074 predict acc : 0.156

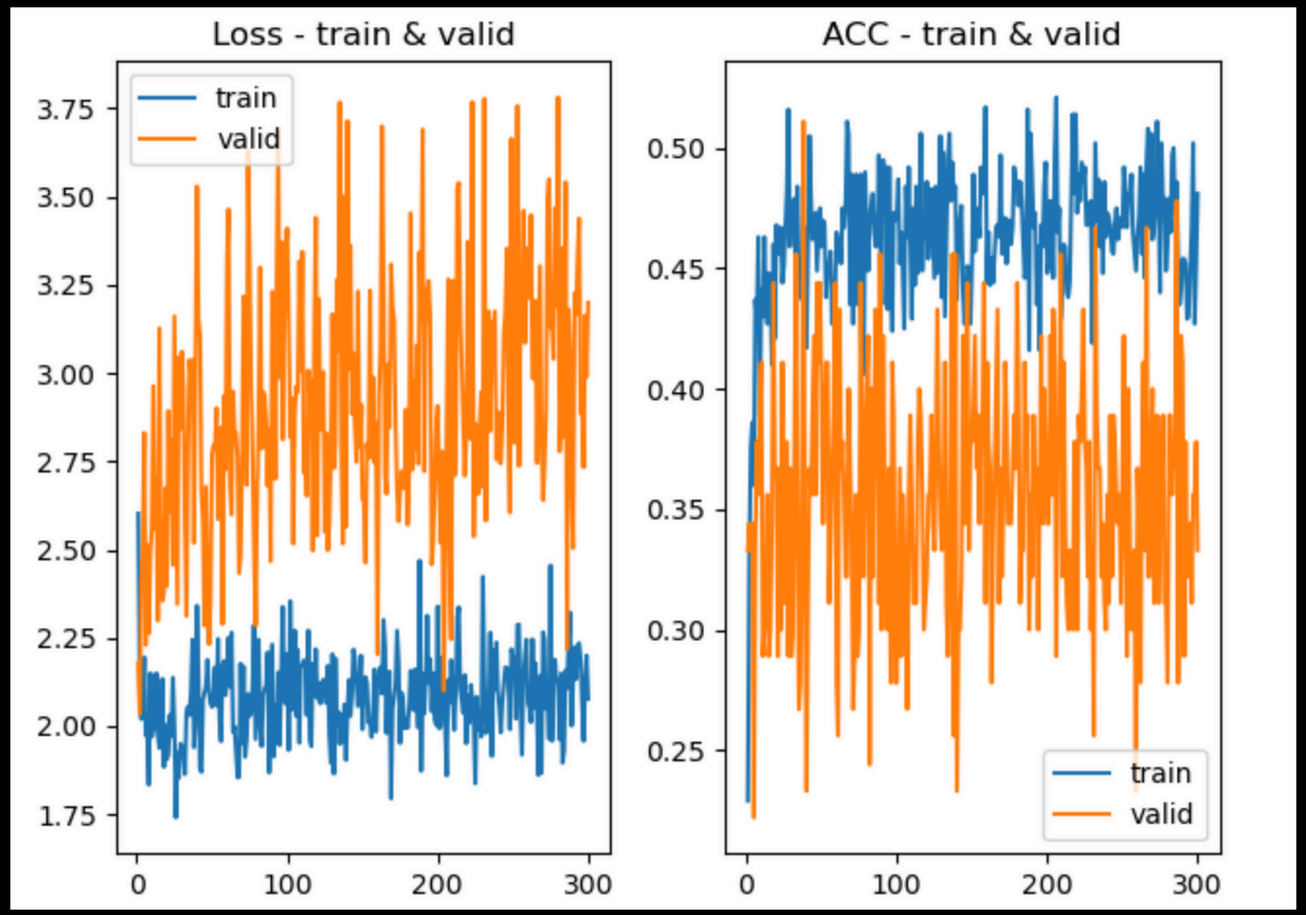
갑자기, 달콤했던 resnet의
기억이 생각났다.

*resnet18*으로 만들어서 가동시켜 보자!

🏃=3 🏃=3 🏃=3

RESNET18 (가중치 설정 O) 결과

300 epoch



[기존 CNN - valid]

loss = 2

acc = 0.14

[ResNet - valid]

loss = 2.9

acc = 0.35

accuracy가 향상되었다.

ResNet18의 predict

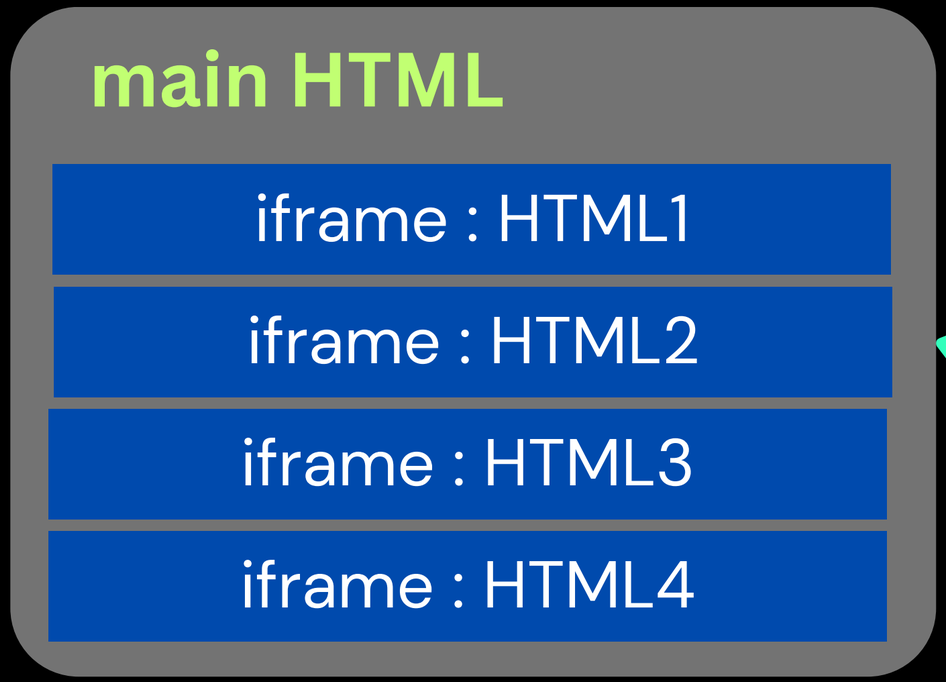
[CNN의 predict]
predict loss : 2.074
predict acc : 0.156

[ResNet의 predict]
predict loss : 3.274
predict acc : 0.3

predict의 accuracy도 향상되었다.

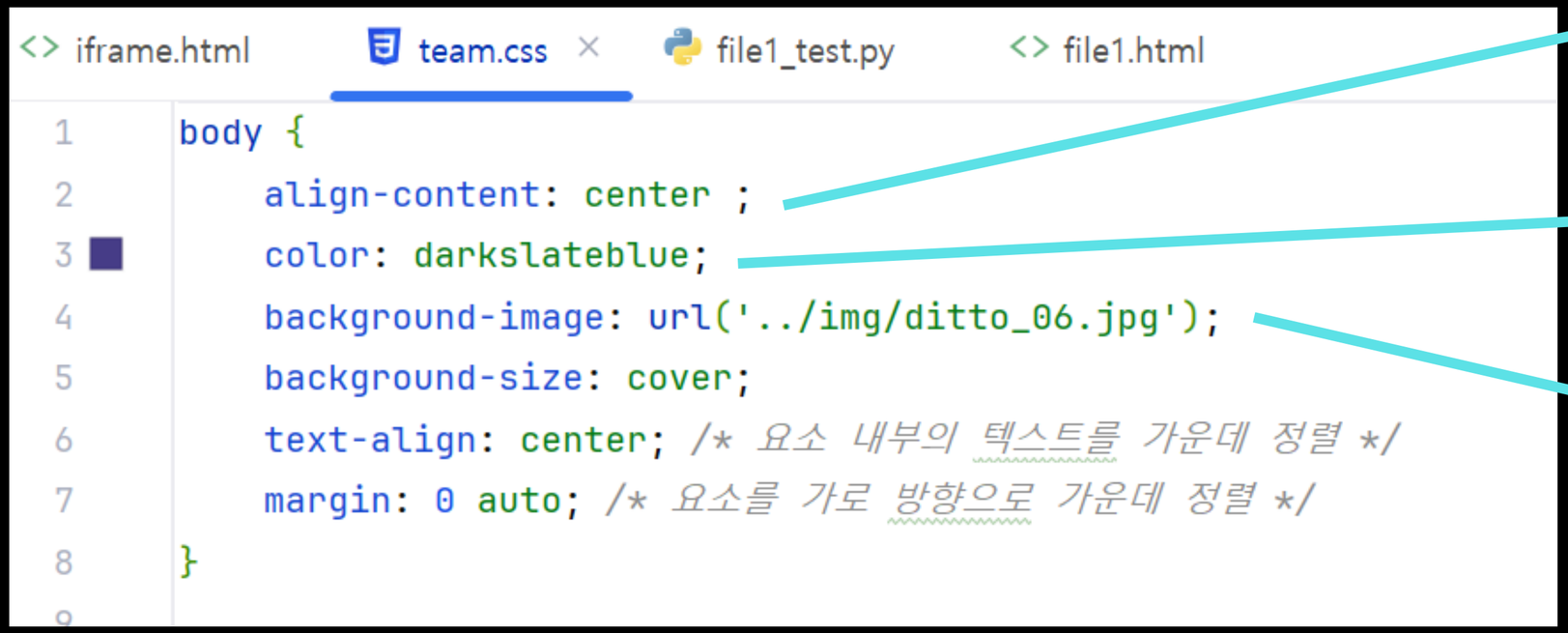
모델 저장
HTML 생성

배경음악




```
<body>
    <audio controls autoplay = "autoplay">
      <source src="../img/goodbye_summer.mp3" type="audio/mpeg">
    </audio>
    <br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>
    <br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>
    <br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>

    <H1>학창시절의 당신은 무슨 음악을 들었나요? </H1>
    <H4><strong>team : WeAreOne</strong></H4>
    <hr color = 'red' size = '3'><br>

    <H3>그때 들었던 음악을 좀 더 알고싶지 않나요?</H3><br>
    <H3>그래서 준비했습니다</H3><br><br><br>
    <H3>📌 나의 mood를 알려줘!</H3>
    <H3>📌 알고리즘아. 내가 원하는 곡의 추천곡을 알려줘</H3>
    <H3>📌 내 최애가 내가 좋아하는 노래를 부를 수 있을까</H3>
    <H3>📌 나의 가수가 추는 춤을 만든 안무가</H3><br><br><br>

    <hr color = 'red' size = '3'><br>

<!-- 첫 번째 HTML 파일을 로드하는 iframe -->
<iframe src="file1.html" width="100%" height="700px" frameborder="3"></iframe>

<!-- 두 번째 HTML 파일을 로드하는 iframe -->
<iframe src="file2.html" width="100%" height="700px" frameborder="3"></iframe>
```

main html에 적용된 CSS



중앙정렬

폰트색 설정

배경 사진 지정

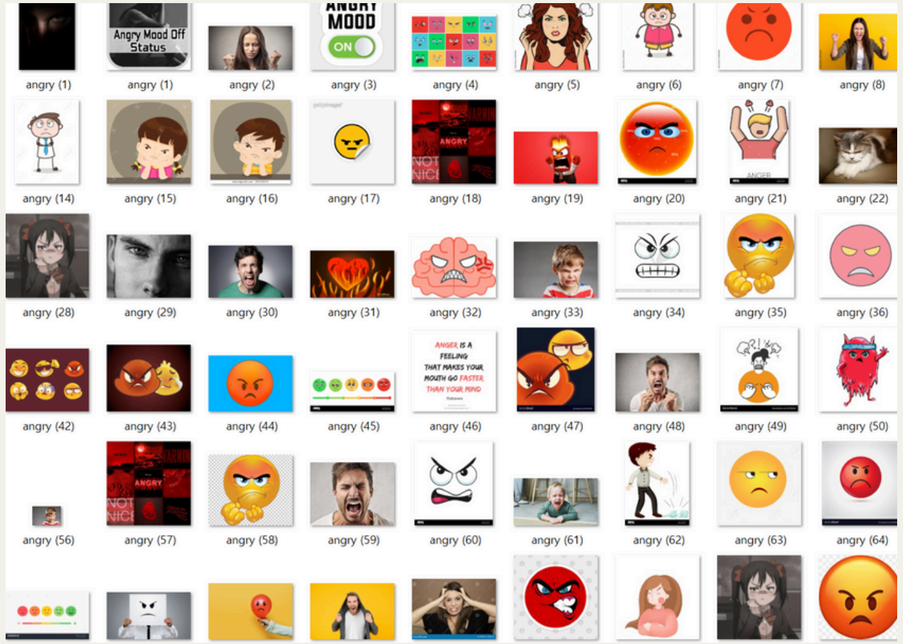
html1 (무드 모델 시연할 창)

```
<h1>임소영</h1>
<h3>당신의 무드를 알려드릴게요.</h3>
<h3>당신의 사진을 넣어주세요! </h3>

<form method = 'post' action = "/cgi-bin/file1_test.py" enctype = 'multipart/form-data'>
    <input type = "file" name="img_file" accept = "image/png, image/jpeg"><br><br>
    <br>
    <input type="image" src="../img/save_button.png" alt="Submit"style="width: 100px; height: 100px;"><br><br><br>
</form>
```

사진을 제출할 form 생성

제출 버튼에 이미지 설정

왜 성능이 그렇게 되었을까?



Garbage In , Garbage Out

각 무드당 100개, 총 800개

좀 더 발전시켜보자면?

- 엎었던 open CV를 활용하여 HTML에서 셀카를 찍는 기능 재도전
- 셀카를 찍고 본인의 표정의 mood를 분석하여 노래 추천하기

~~사실 저의 최종 목표는 이거였어요.. ㅜ~~

WHITEBOARD PAGE

장르 분석 및 유사도 추천

발표자 : 이윤서

순서 댄스 / 트로트 / 발라드 다중분류

1. CNN

2. CNN+RNN

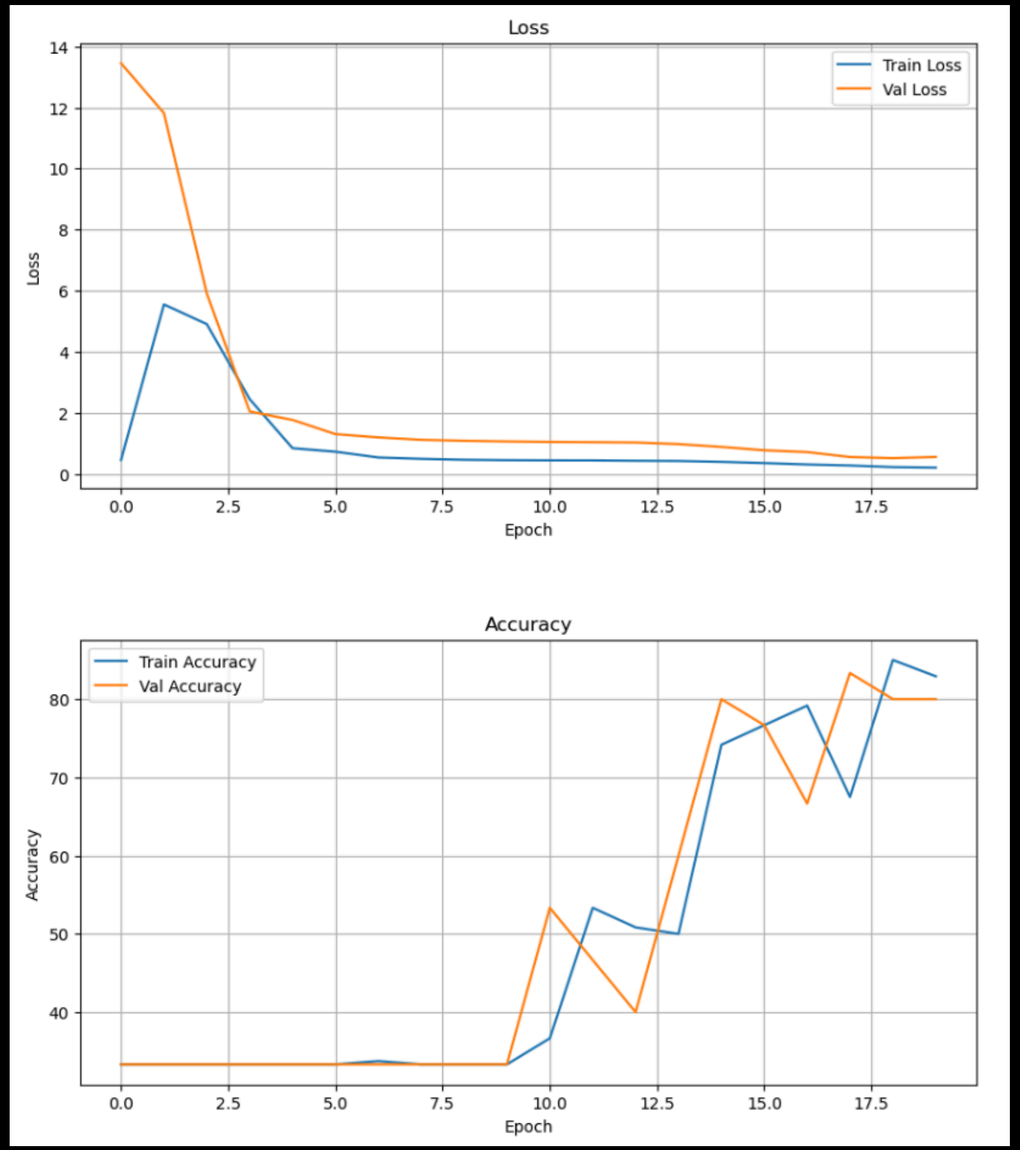
3. 머신러닝

4. 노래추천

CNN

모델 구조

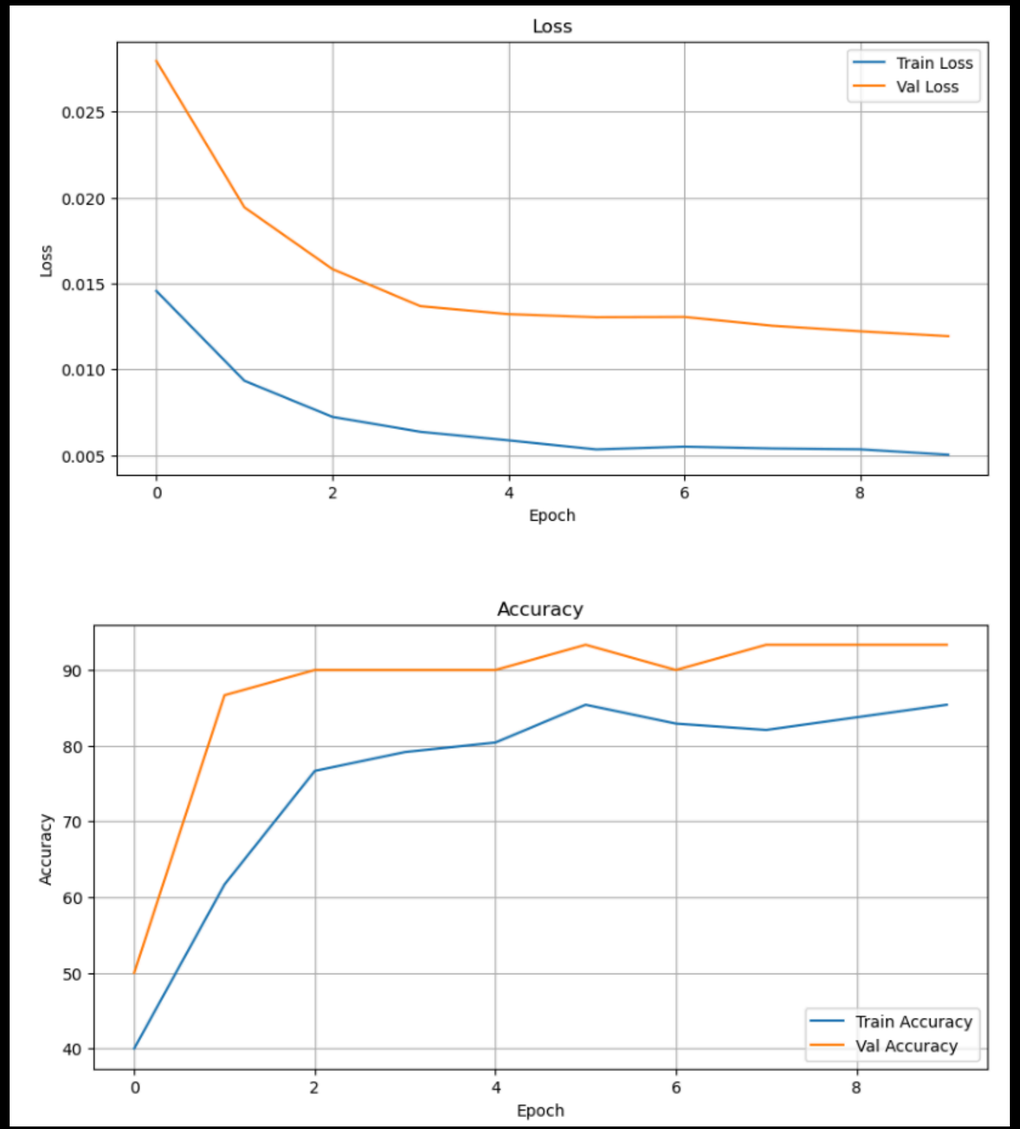
```
MultiClassCNN(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc1): Linear(in_features=640000, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=3, bias=True)
)
```



CNN

VGG16 모델 구조

```
genreClassifier(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
...
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4, bias=True)
  )
)
```

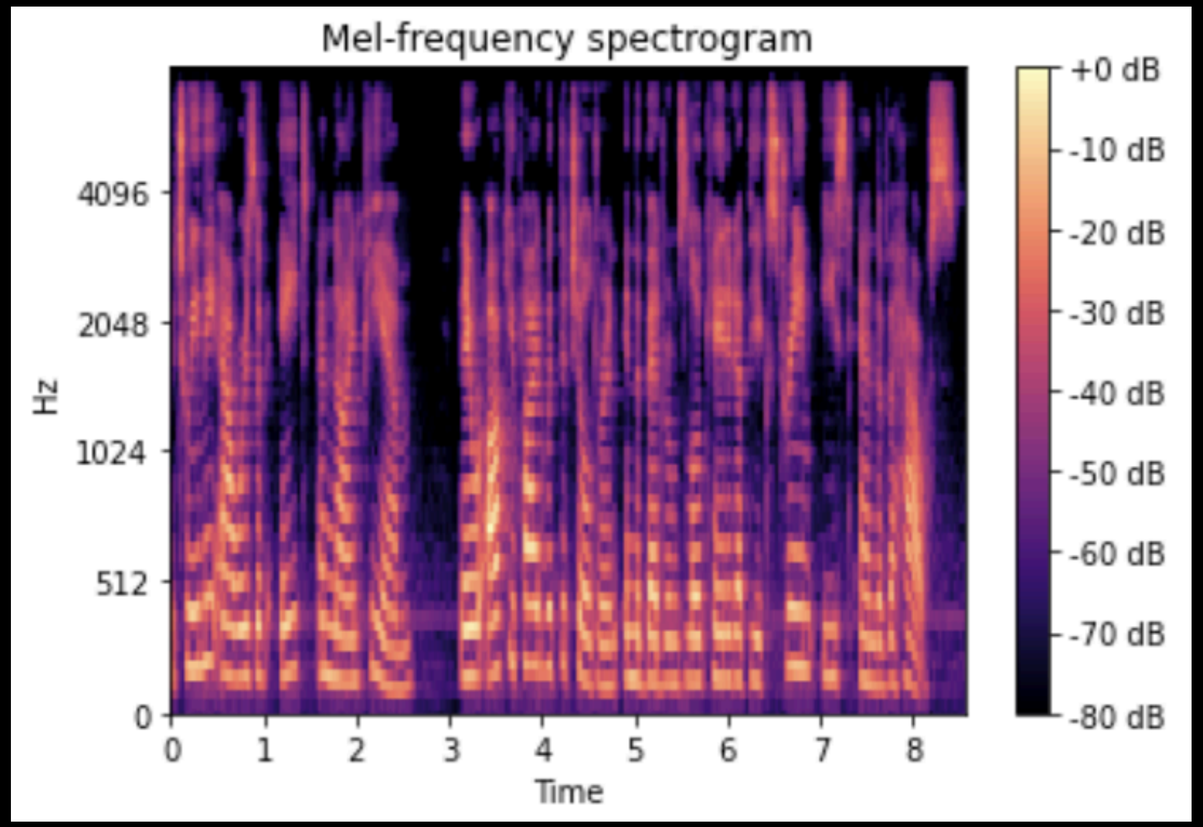


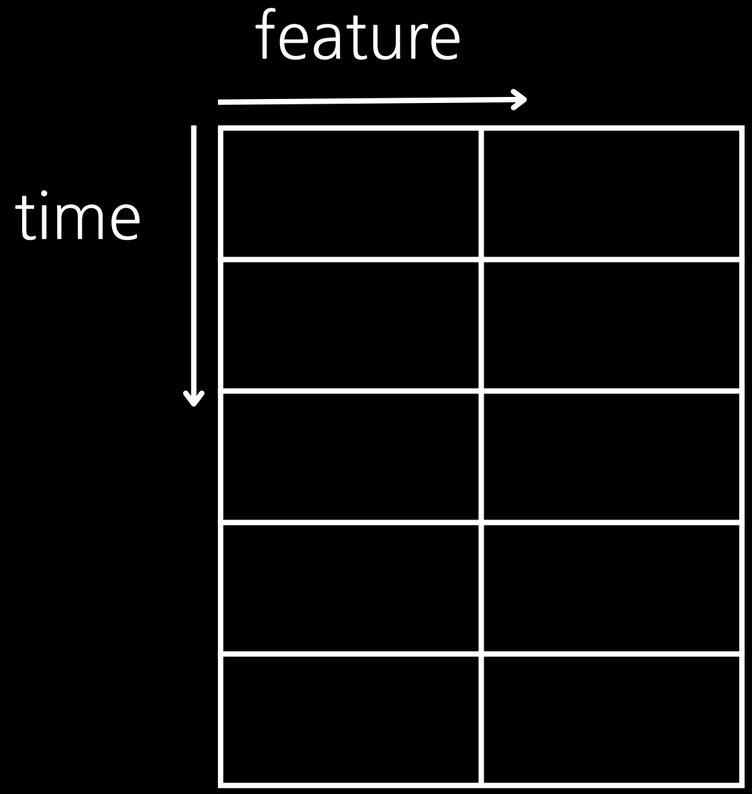
CNN

결과

	Test Accuracy
계층3 CNN	0.800
VGG16	0.930

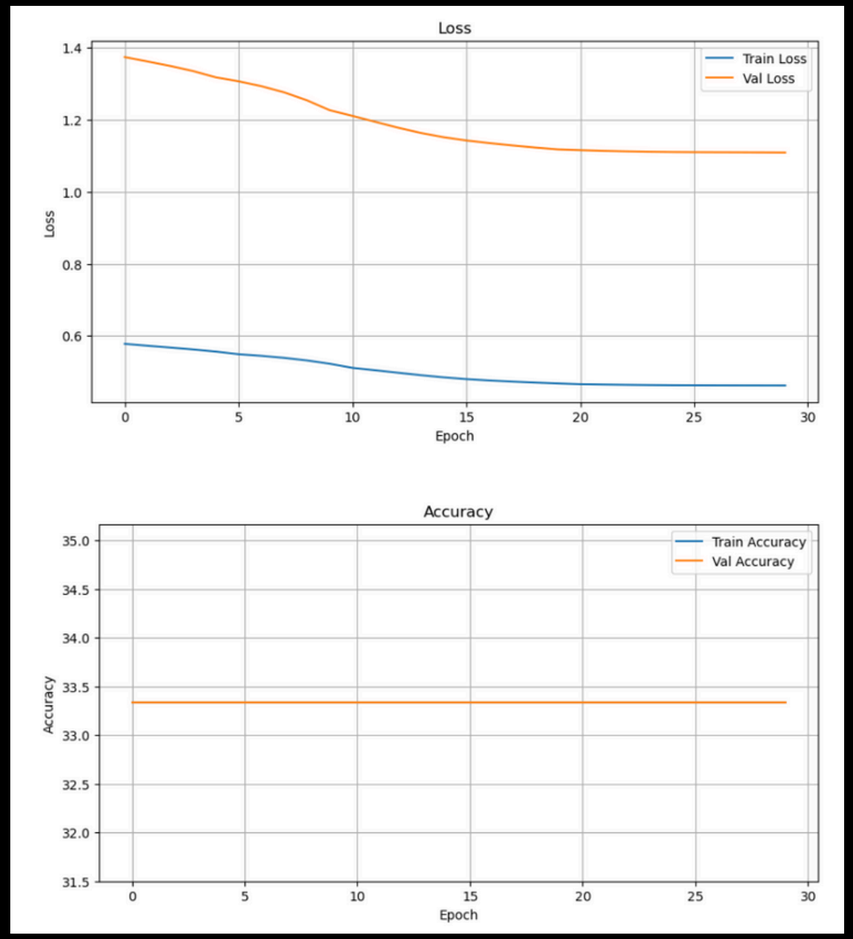
CNN+RNN





시간 별로 주파수 범위의 평균과 분산을 구하여 전치

CNN+RNN



피쳐가 2개니까 그런 거 아닐까?

결과 : 0.33 -> 하나로 밀고 있다

CNN+RNN

STFT(Short Time) 기반 특성 뽑기

```python
# Spectral Centroids 계산
spectral_centroids = librosa.feature.spectral_centroid(y=y, sr=sr)[0]

# Spectral Rolloff 계산
spectral_rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)[0]

# Chromagram 계산
chromagram = librosa.feature.chroma_stft(y=y, sr=sr, hop_length=512)

# Zero-Crossing Rate (제로 크로싱? 비율)
zero_crossing_rate = librosa.feature.zero_crossing_rate(y)[0]

# Root Mean Square (RMS) Energy (평균 제곱근 에너지)
rms_energy = librosa.feature.rms(y=y)[0]

# Spectral Bandwidth (주파수 대역폭)
spectral_bandwidth = librosa.feature.spectral_bandwidth(y=y, sr=sr)[0]

# Spectral Contrast (주파수 대비)
spectral_contrast = librosa.feature.spectral_contrast(y=y, sr=sr)

# Spectral Flatness (주파수 평평도)
spectral_flatness = librosa.feature.spectral_flatness(y=y)

# BPM 계산
tempo , _ = librosa.beat.beat_track(y=y,sr=sr)

# 하모니(음색) -> 코러스 좀... / 충격파(둥둥)
harmonic, percussive = librosa.effects.hpss(y=y)
```

```
(1378,)
(1378,)
(12, 1378)
(1378,)
(1378,)
(1378,)
(7, 1378)
(1, 1378)
(705184,)
(1296,)
(1296,)
(12, 1296)
(1296,)
```

```
(1296,)
(1296,)
(7, 1296)
(1, 1296)
(663330,)
(1292,)
(1292,)
(12, 1292)
(1292,)
(1292,)
(1292,)
(7, 1292)
(1, 1292)
```

피쳐 길이가 다 다름

-> 실패!

chromagram

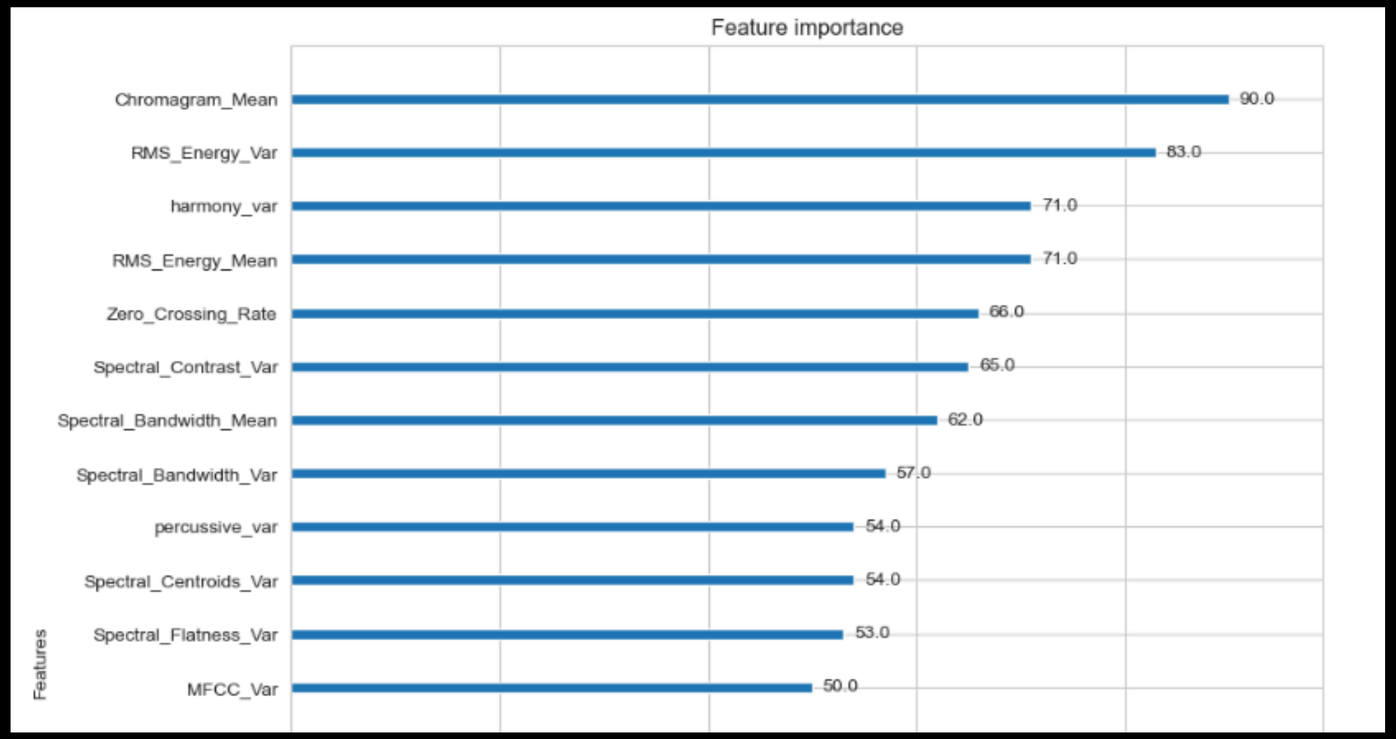
ML

뽑은 feature 22개로 머신러닝 돌리기

```
data = {
    # 파일 경로
    'Filename': os.path.join(wav_dir, wav_file),
    # 특성
    'MFCC_Mean': [np.mean(mfccs)],
    'MFCC_Var': [np.var(mfccs)],
    'Chromagram_Mean': [np.mean(chromagram)],
    'Chromagram_Var': [np.var(chromagram)],
    'Spectral_Centroids_Mean': [np.mean(spectral_centroids)],
    'Spectral_Centroids_Var': [np.var(spectral_centroids)],
    'Spectral_Rolloff_Mean': [np.mean(spectral_rolloff)],
    'Spectral_Rolloff_Var': [np.var(spectral_rolloff)],
    'Zero_Crossing_Rate': [np.mean(zero_crossing_rate)],
    'RMS_Energy_Mean': [np.mean(rms_energy)],
    'RMS_Energy_Var': [np.var(rms_energy)],
    'Spectral_Bandwidth_Mean': [np.mean(spectral_bandwidth)],
    'Spectral_Bandwidth_Var': [np.var(spectral_bandwidth)],
    'Spectral_Contrast_Mean': [np.mean(spectral_contrast)],
    'Spectral_Contrast_Var': [np.var(spectral_contrast)],
    'Spectral_Flatness_Mean': [np.mean(spectral_flatness)],
    'Spectral_Flatness_Var': [np.var(spectral_flatness)],
    'Tempo': [tempo],
    "harmony_mean" : [harmonic.mean()],
    "harmony_var" : [harmonic.var()],
    "precussive_mean" : [percussive.mean()],
    "percussive_var" : [percussive.var()],
    # 정답!
    'label': genre_name
}
```

1.chromagram(템포에 대한 정보X)

2. RMSEnergy(템포에 대한 정보O)



ML

결과

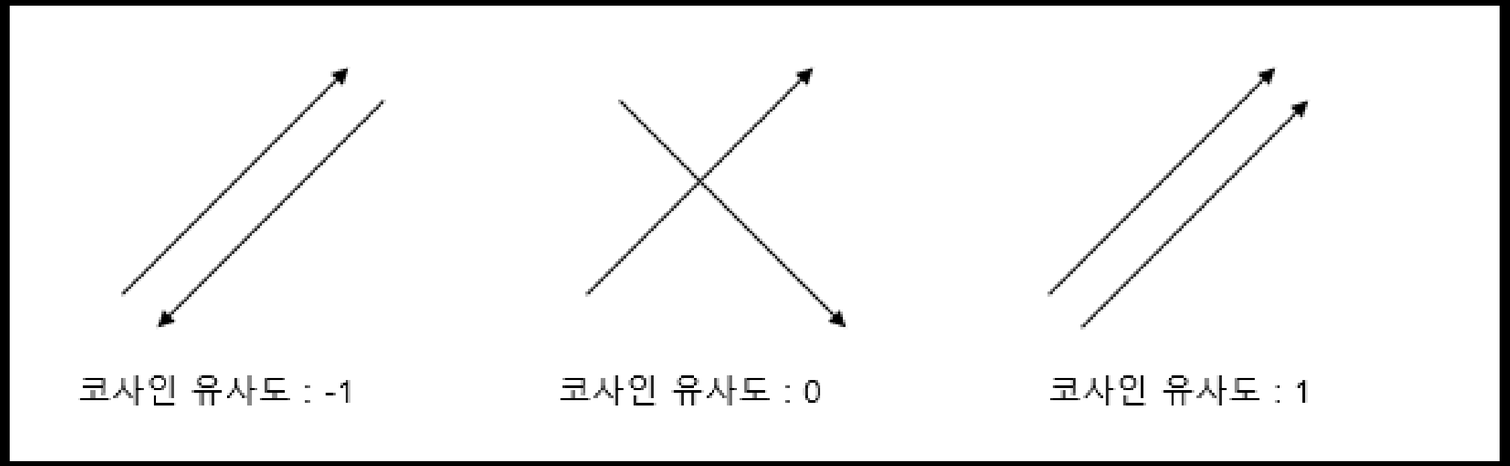
	Accuracy	Precision	Recall	F1
랜덤포레스트	0.900	0.902	0.903	0.900
XGBoost	0.900	0.908	0.903	0.903

모델 최종 결과

	Test Accuracy
VGG16	0.930
XGBoost	0.900
CNN+RNN	0.333

노래 추천

코사인 유사도



두 벡터가 가리키는 방향이 얼마나 **유사한지** 계산

노래 추천

결과

```
find_similar_songs('balad5.wav', sim_c
```

✓ 0.0s

내 음악 : balad5.wav
유사한 음악

Filename	
./sounds/ballade/balad20.wav	0.857262
./sounds/ballade/balad26.wav	0.793949
./sounds/ballade/balad12.wav	0.790618
./sounds/ballade/balad24.wav	0.789220
./sounds/trot/trot31.wav	0.788101

```
find_similar_songs('dance5.wav', sim_
```

✓ 0.0s

내 음악 : dance5.wav
유사한 음악

Filename	
./sounds/dance/dance19.wav	0.862671
./sounds/dance/dance70.wav	0.830613
./sounds/dance/dance66.wav	0.827733
./sounds/dance/dance69.wav	0.821662
./sounds/dance/dance1.wav	0.780364

```
find_similar_songs('trot5.wav', sim_c
```

✓ 0.0s

내 음악 : trot5.wav
유사한 음악

Filename	
./sounds/trot/trot18.wav	0.918261
./sounds/trot/trot20.wav	0.779123
./sounds/trot/trot24.wav	0.717591
./sounds/trot/trot16.wav	0.713232
./sounds/trot/trot46.wav	0.680030

새로운 음악을 넣으면 기존 음악들과의 유사도를 측정하여 음악을 추천해줌

HTML demonstration