Audio Classification with Pytorch

11/10/23

Mel

- 원본 음성 파동을 STFT한 결과에 Mel 변환 수행
- 저주파 소리의 변화에 민감하고 고주파 소리의 변화에는 둔감한 인간 인지 기준을 반영
- STFT, 원본 음성 파동보다 저차원의 정보(위상 정보 소실)
- Mel bin은 보통 80, 128로 설정

MFCC

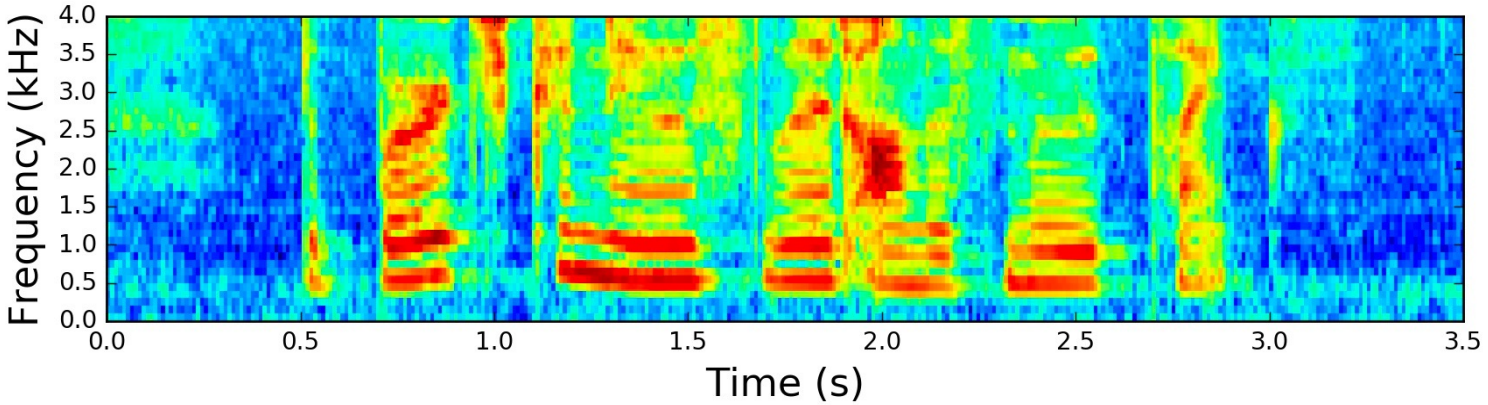
- Mel Spectrogram을 Inverse FFT 수행한 것
- Mel Spectrogram보다 저차원의 정보로, Cepstral Coefficients 의 개수가 보통 13, 26, 39.

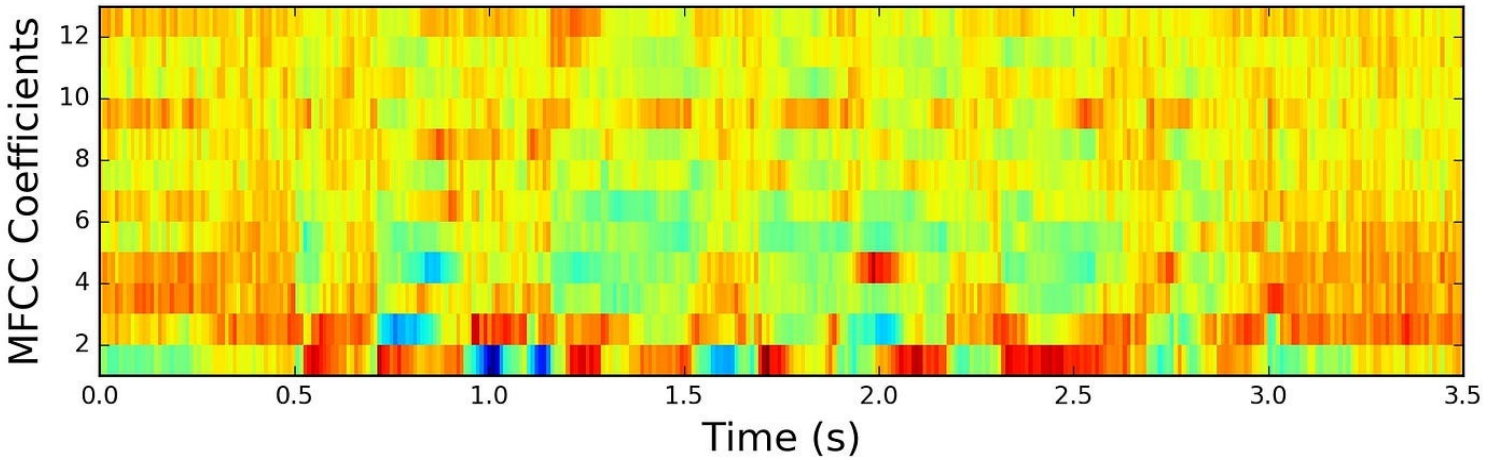
전처리 개선사항

- 라벨 별 데이터 용량이 현저히 차이가 나기 때문에 이를 데이터 크기에 비례해 샘플링하면 오버피팅이 될 수 있음 →Equal probability sampling을 수행해서 오버피팅을 막으려고 함.

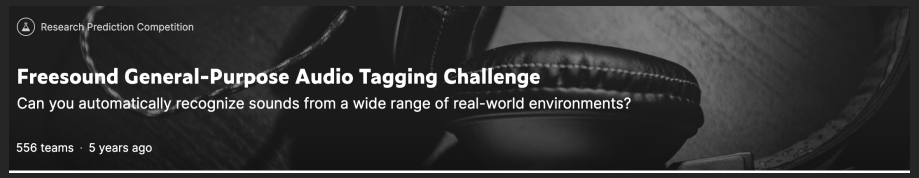
- **Mel Spectrogram 셋업**
- Sr = 16000
- N_mels = 80 #feature_size
- Augmentation = False
- Shape = [41, 80] #[sequence_len, N_mels]

Mel VS MFCC


Frequency (kHz)
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Time (s)


MFCC Coefficients
12
10
8
6
4
2
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Time (s)

데이터, 모델 개발 목표 셋업



- 14개의 악기의 소리 데이터에 대한 데이터셋

- 0.5초의 음성을 듣고 오디오 레이블을 분류할 수 있는 모델 개발하고자 함

-

전처리 셋업

- **Mel Spectrogram 셋업**

Sr = 44100

N_mels = 128 #feature_size

Augmentation = False

Shape = [56789, 44, 128] #[sample_size, sequence_len, N_mels]

- **MFCC 셋업**

Sr = 12000

Numcep = 13

Nfilt = 26 #mel spectrogram의 n_mels와 같음.

Window_size = 25ms

Shape = [56789, 13, 49] #[sample_size, #cepstral coefs, sequence_len]

학습 셋업

Optimizer = Adam

Lr = 0.01

Epochs = 50

Batch_size = 16

Activation = ReLU

Train_resnet_mel이 좀 이상하다..

실험 목표

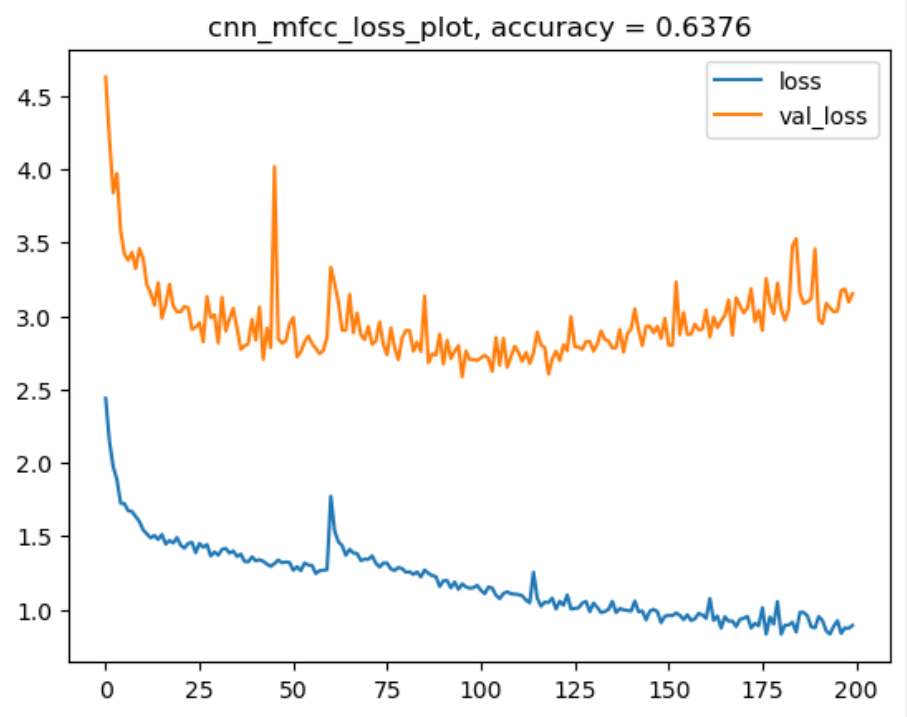
- 파이토치의 사용법 익히기
- Mel Spectrogram과 MFCC 중 무엇이 더 성능이 좋은지 확인하기
- CNN, LSTM, Resnet, NewCNN 중 어떤 모델 구조가 가장 고성능일지 확인하기

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 16, 11, 47]             160
       BatchNorm2d-2          [-1, 16, 11, 47]              32
              ReLU-3          [-1, 16, 11, 47]               0
              ReLU-4          [-1, 16, 11, 47]               0
            Conv2d-5           [-1, 32, 9, 45]           4,640
       BatchNorm2d-6           [-1, 32, 9, 45]              64
              ReLU-7           [-1, 32, 9, 45]               0
              ReLU-8           [-1, 32, 9, 45]               0
           Flatten-9                [-1, 12960]               0
           Linear-10                  [-1, 512]       6,636,032
             ReLU-11                  [-1, 512]               0
             ReLU-12                  [-1, 512]               0
           Linear-13                   [-1, 64]          32,832
             ReLU-14                   [-1, 64]               0
             ReLU-15                   [-1, 64]               0
           Linear-16                   [-1, 14]             910
          Softmax-17                   [-1, 14]               0
================================================================
Total params: 6,674,670
Trainable params: 6,674,670
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.76
Params size (MB): 25.46
Estimated Total Size (MB): 26.22
----------------------------------------------------------------
```

CNN 모델 구조

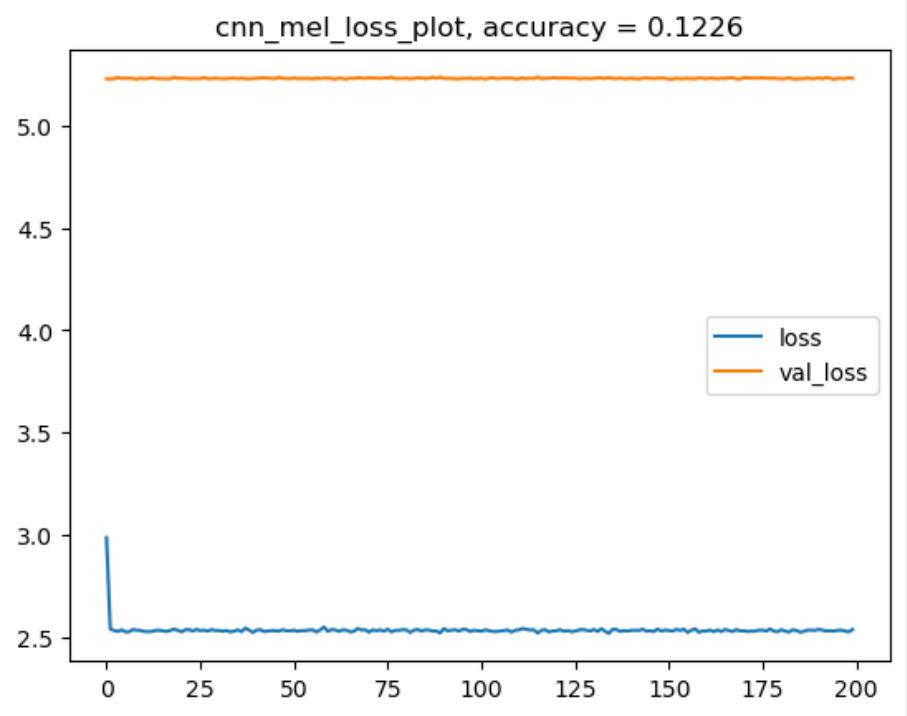
- Dropout 대신 BatchNormalization 사용
- Hidden dimension = (16, 32)
- Pooling을 적용하지 않음

CNN MFCC

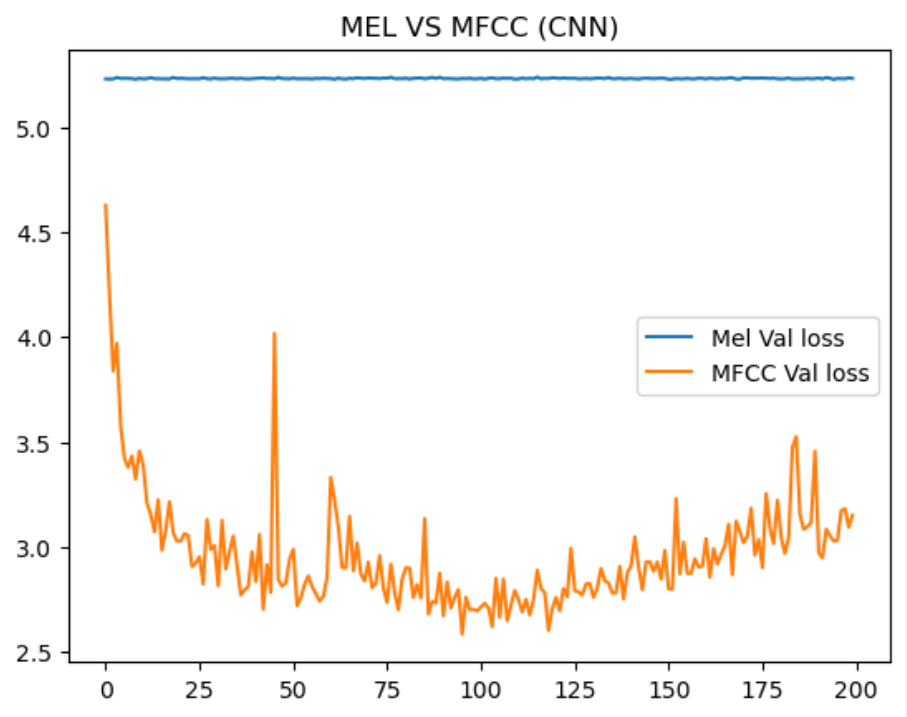

cnn_mfcc_loss_plot, accuracy = 0.6376
loss
val_loss
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
25
50
75
100
125
150
175
200

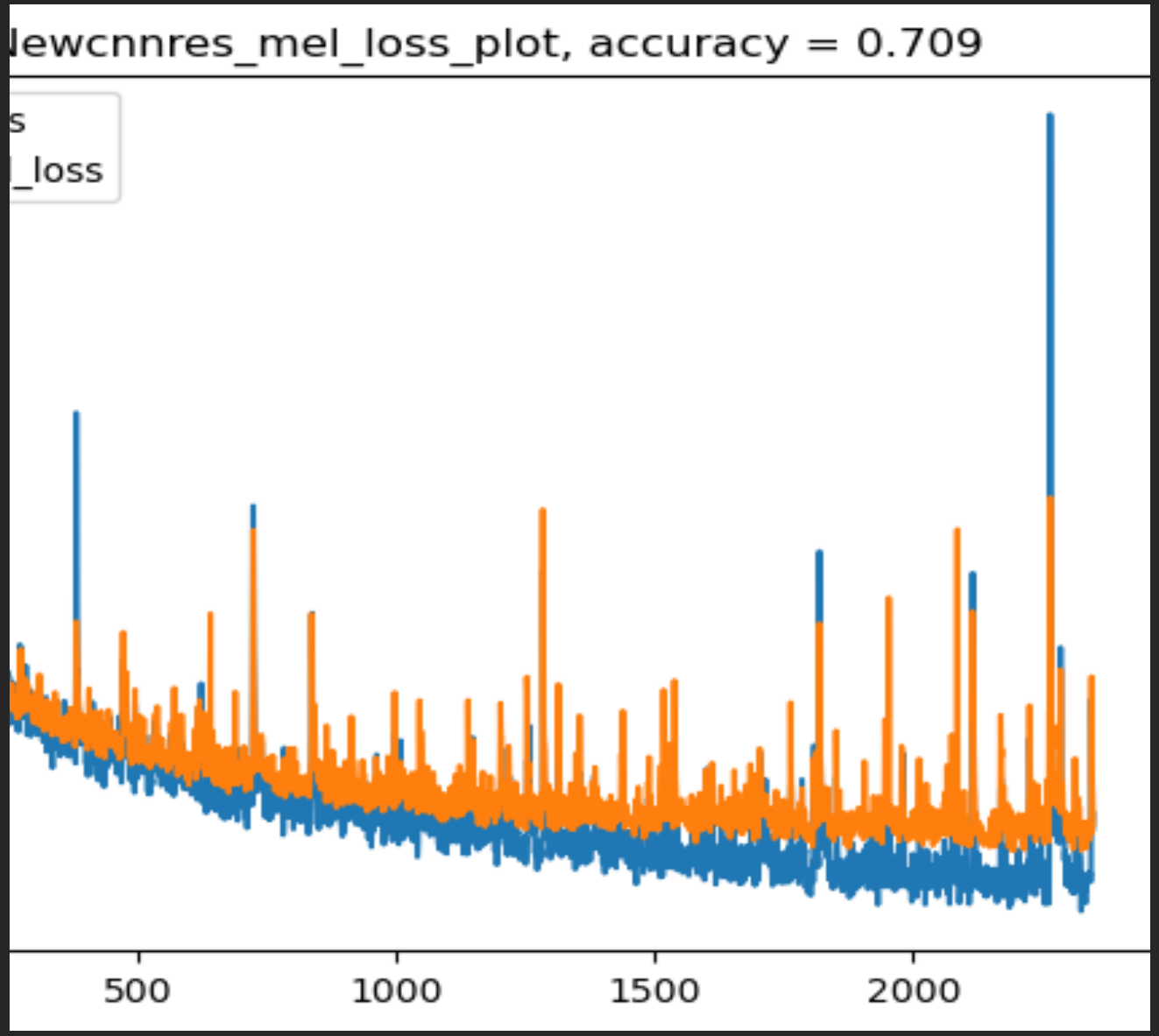
CNN melspectrogram

- 모델의 구조가 데이터 분류에 효과적이지 않음



CNN Mel VS MFCC





변경 후 CNN 모델

모델 구조: [Conv2d(kernelsize = (3, 3)), LeakyRelu] X 5

HyperParameter: Dim_h = 64

Flatten 방식: adaptive pooling

변경사항 요약: 모델이 피처 인코딩과 라벨 디코딩으로 이루어져 있으면 피처 인코딩(CNN) 부분에서 층을 더 깊게 쌓아 Expression Power를 키우고 디코딩(Flatten, Decoding) 부분에서 dimension을 낮춰가며 정보를 압축하는 방식으로 linear를 쌓음.

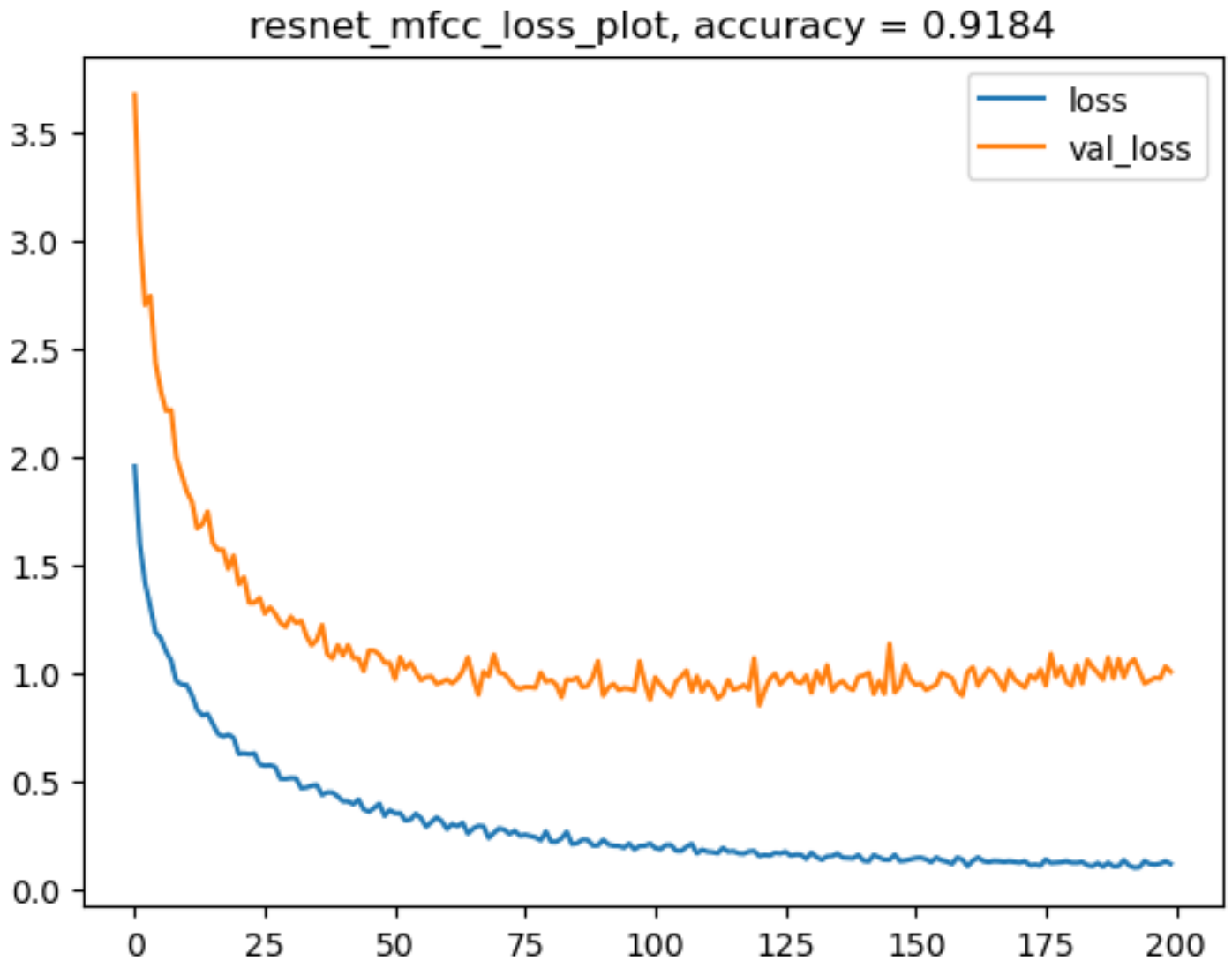
Resnet 모델 구조

```
]         BasicBlock-34             [-1, 128, 2, 7]               0
            Conv2d-35             [-1, 256, 1, 4]         294,912
       BatchNorm2d-36             [-1, 256, 1, 4]             512
              ReLU-37             [-1, 256, 1, 4]               0
            Conv2d-38             [-1, 256, 1, 4]         589,824
       BatchNorm2d-39             [-1, 256, 1, 4]             512
            Conv2d-40             [-1, 256, 1, 4]          32,768
       BatchNorm2d-41             [-1, 256, 1, 4]             512
              ReLU-42             [-1, 256, 1, 4]               0
        BasicBlock-43             [-1, 256, 1, 4]               0
            Conv2d-44             [-1, 256, 1, 4]         589,824
       BatchNorm2d-45             [-1, 256, 1, 4]             512
              ReLU-46             [-1, 256, 1, 4]               0
            Conv2d-47             [-1, 256, 1, 4]         589,824
       BatchNorm2d-48             [-1, 256, 1, 4]             512
              ReLU-49             [-1, 256, 1, 4]               0
        BasicBlock-50             [-1, 256, 1, 4]               0
            Conv2d-51             [-1, 512, 1, 2]       1,179,648
       BatchNorm2d-52             [-1, 512, 1, 2]           1,024
              ReLU-53             [-1, 512, 1, 2]               0
            Conv2d-54             [-1, 512, 1, 2]       2,359,296
       BatchNorm2d-55             [-1, 512, 1, 2]           1,024
            Conv2d-56             [-1, 512, 1, 2]         131,072
       BatchNorm2d-57             [-1, 512, 1, 2]           1,024
              ReLU-58             [-1, 512, 1, 2]               0
        BasicBlock-59             [-1, 512, 1, 2]               0
            Conv2d-60             [-1, 512, 1, 2]       2,359,296
       BatchNorm2d-61             [-1, 512, 1, 2]           1,024
              ReLU-62             [-1, 512, 1, 2]               0
            Conv2d-63             [-1, 512, 1, 2]       2,359,296
       BatchNorm2d-64             [-1, 512, 1, 2]           1,024
              ReLU-65             [-1, 512, 1, 2]               0
        BasicBlock-66             [-1, 512, 1, 2]               0
AdaptiveAvgPool2d-67             [-1, 512, 1, 1]               0
            Linear-68                   [-1, 14]           7,182
================================================================
Total params: 11,177,422
Trainable params: 11,177,422
Non-trainable params: 0
```

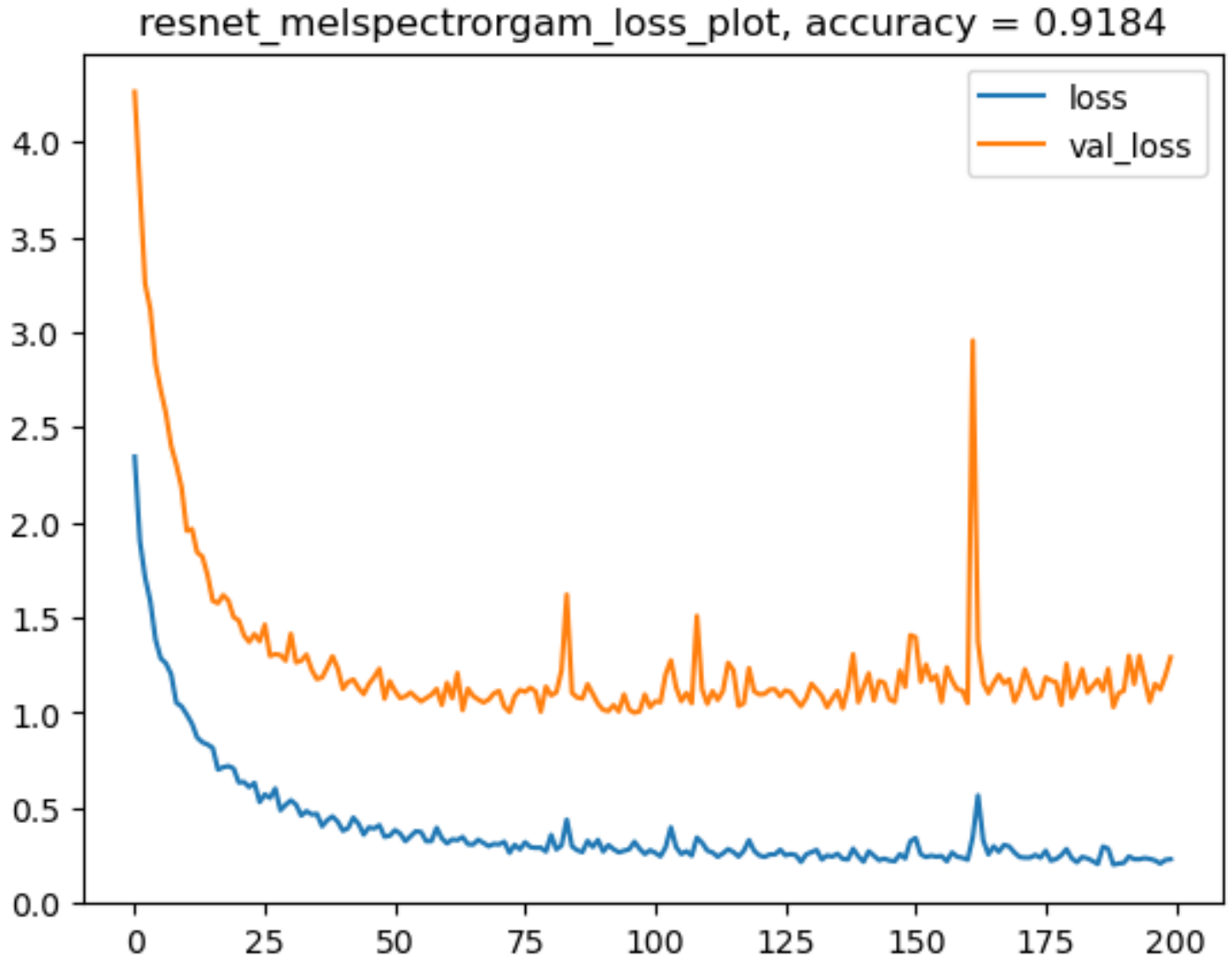
기존 Resnet64의 형태를 차용하되
Residual을 projection 연결하는 것이 아닌
Identity connection으로 연결하도록 함.

Hidden_size = 64

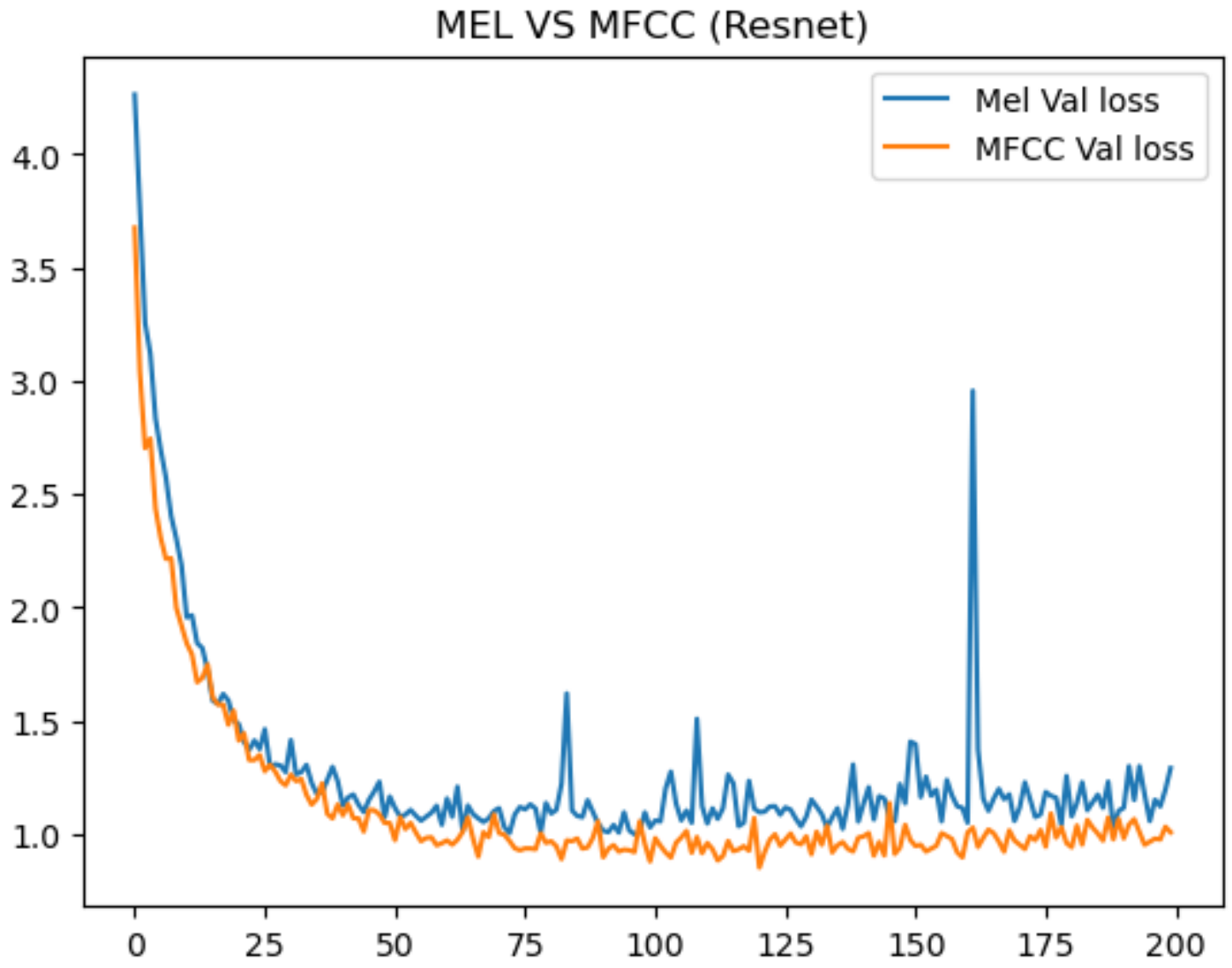
Resnet MFCC


resnet_mfcc_loss_plot, accuracy = 0.9184
loss
val_loss
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
25
50
75
100
125
150
175
200

Resnet
Melspectrogram



Resnet Mel VS MFCC

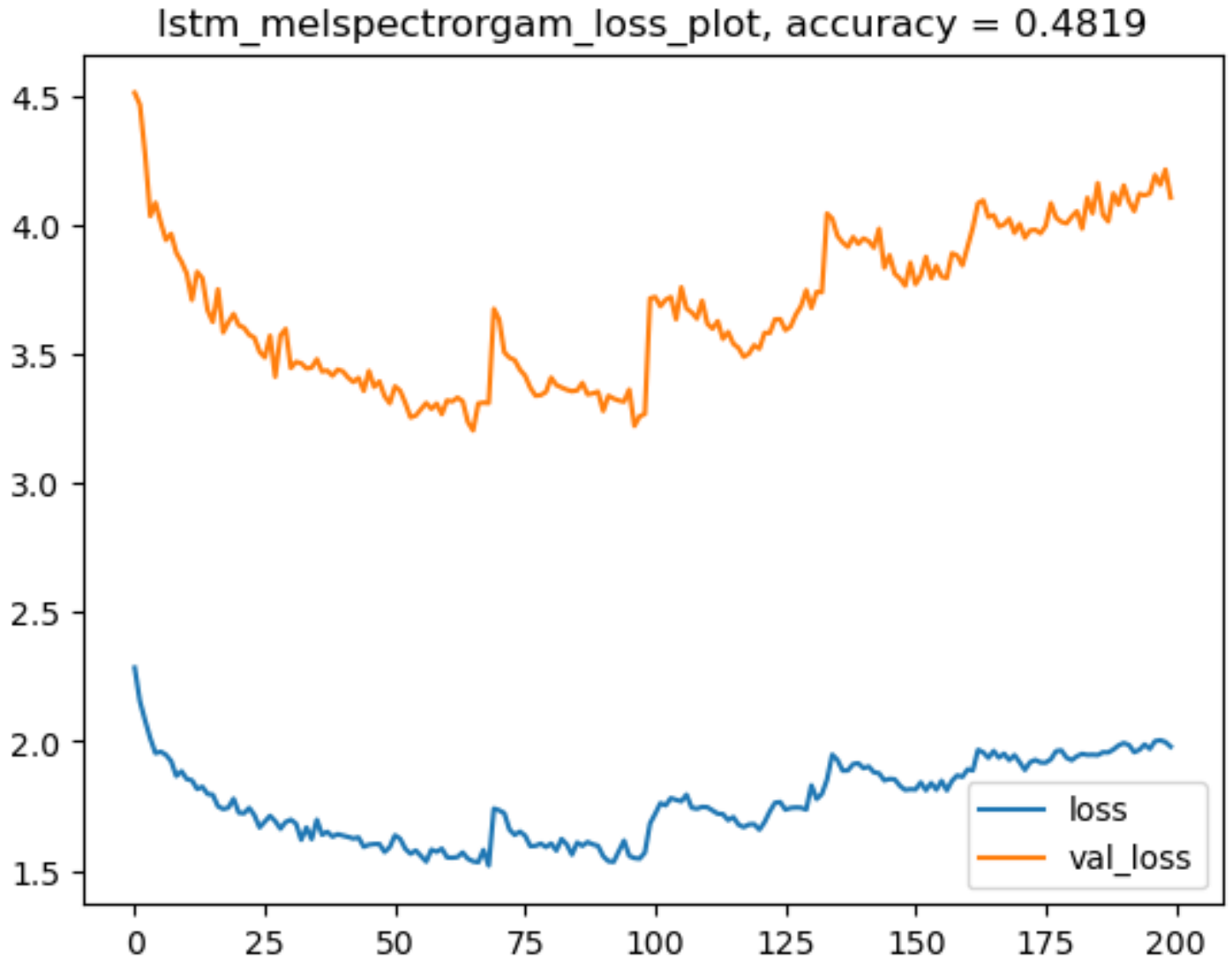


```
DeepLSTM(
  (act): ReLU()
  (flat): Flatten(start_dim=1, end_dim=-1)
  (lstm): LSTM(13, 16, num_layers=3, batch_first=True)
  (bn): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (linear_1): Linear(in_features=48, out_features=128, bias=True)
  (linear_2): Linear(in_features=128, out_features=64, bias=True)
  (linear_3): Linear(in_features=64, out_features=14, bias=True)
)
```

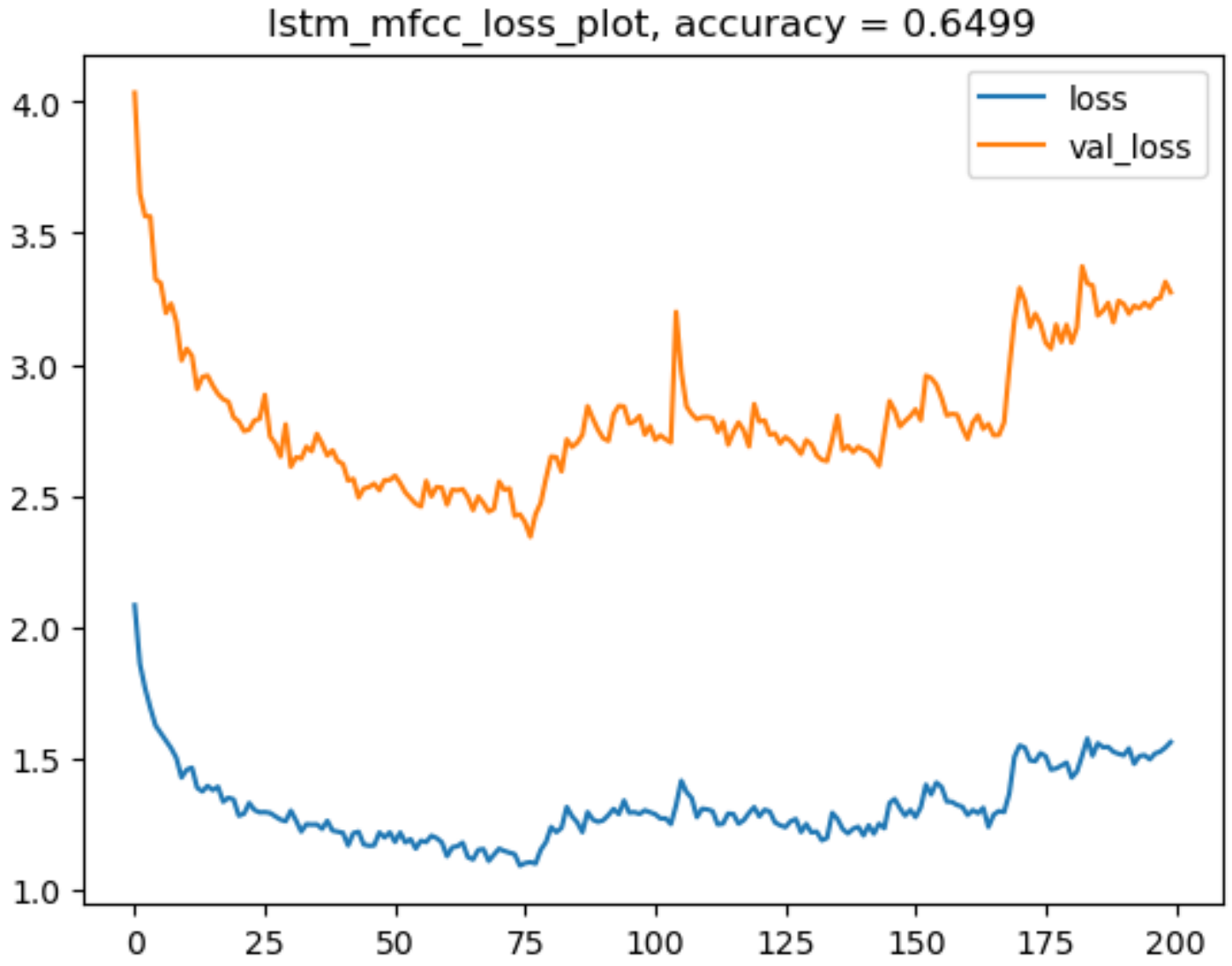
LSTM 모델 구조

- 단층 LSTM을 사용.
- 시간순서상 마지막 레이어들의 hidden state을 flatten한 뒤 linear layer로 연결해 예측하게끔 함
- Hidden_size = 16
- N_layer = 3

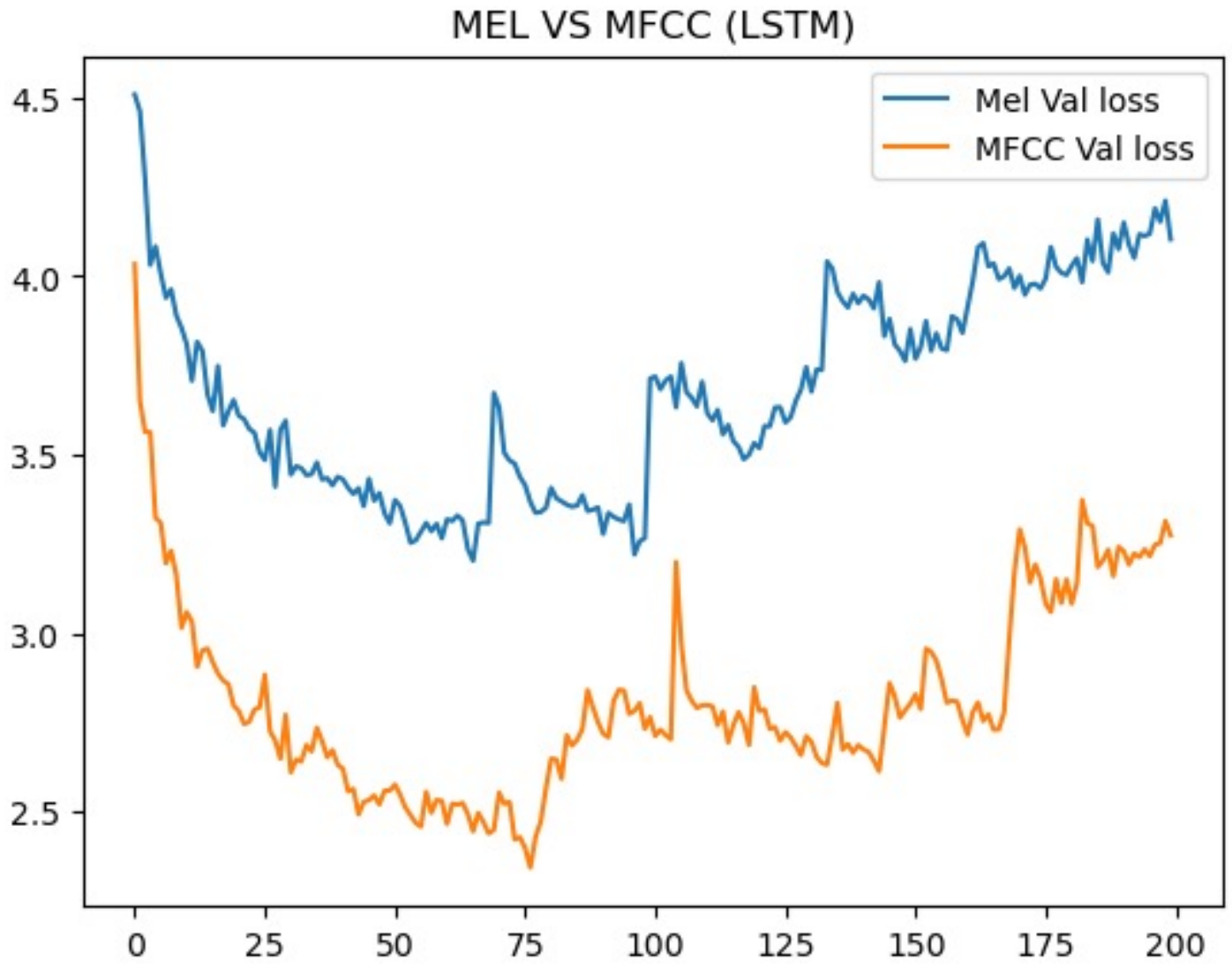
LSTM
MelSpectrogram



LSTM MFCC


lstm_mfcc_loss_plot, accuracy = 0.6499
loss
val_loss
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
25
50
75
100
125
150
175
200

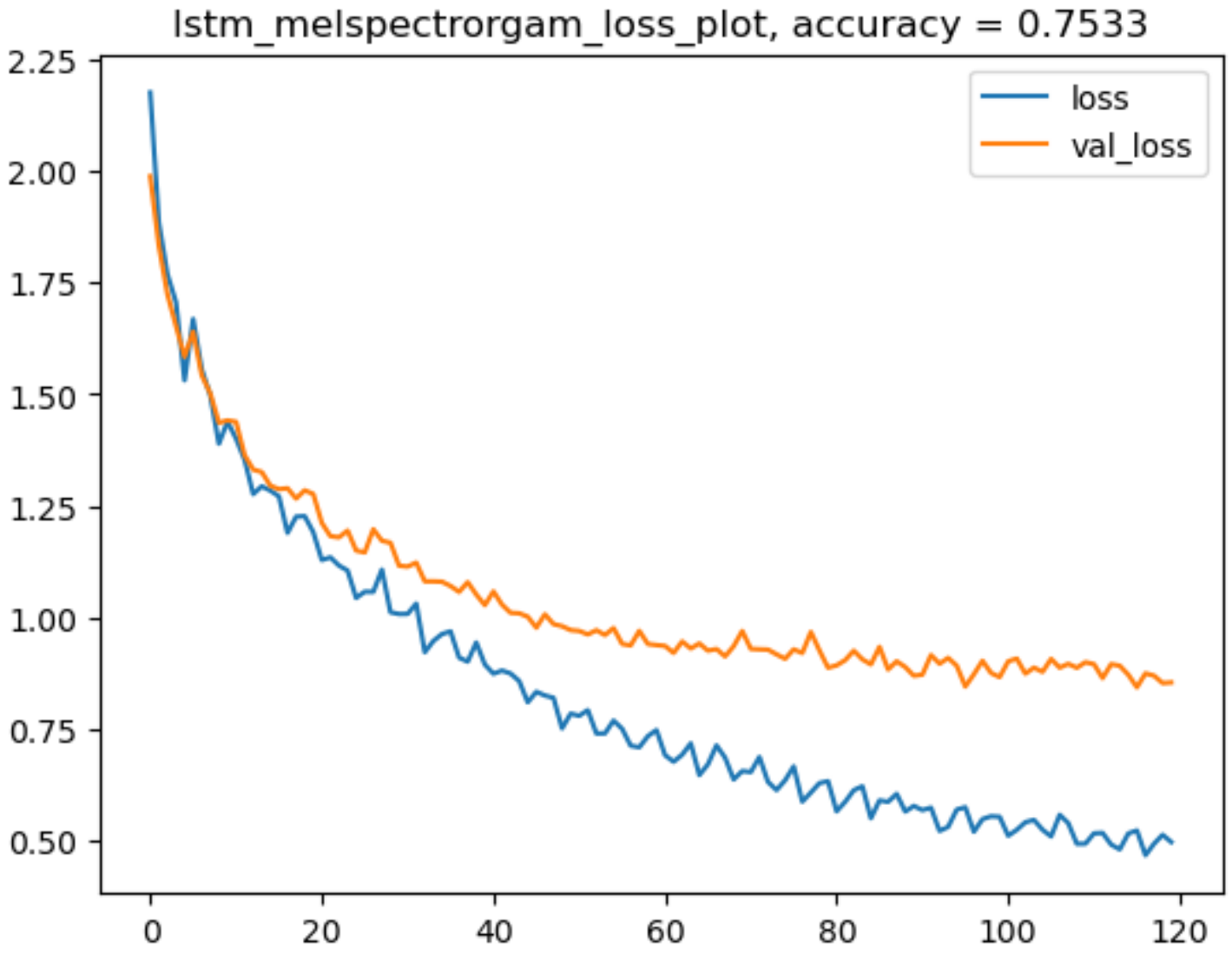
LSTM Mel VS MFCC



변경 후 LSTM 모델

Expression Power를 키우기 위해 hidden dimension을 16에서 128로 키움.

Num_layer를 4에서 8로 변경



결론 및 제언

CNN의 성능이 LSTM보다 낮은 것, 특히 melspectrogram에서 그 현상이 두드러지는 것은 CNN이 Local feature extraction에 특화되어 있는 반면 LSTM은 비교적 Global feature extraction을 잘하기 때문임.

오디오 데이터는 샘플링 과정에서 redundancy가 크고, 주파수 측면에서 기본음의 패턴이 정수배로 반복되는 패턴을 가지기 때문에 time-wise, frequency-wise correlation이 굉장히 큰 편.

Local feature뿐만 아니라 Global Feature를 추출할 수 있는 게 굉장히 중요해 보임.

CNN과 LSTM을 동시에 쓰는 방향으로 Classification을 진행 계획.

(참고) Variable length input 처리에 대하여

- CNN이나 LSTM, Resnet등 Feature representation을 하는 데까지는 input sequence의 length가 달라져도 큰 차이가 없으나 flatten 후 분류를 위한 linear layers에 연결시킬 때 노드의 개수가 달라져 동일한 방식의 학습을 할 수 없게 된다. 이럴 때 사용할 수 있는 방식에는 Adaptive Pooling이 있다.
- Adaptive Pooling을 이용하면 Flatten 레이어의 입력 차원을 일정하게 유지할 수 있음.