```
from google.colab import drive
import pandas as pd
import json


# 구글 드라이브 마운트
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
SAVE_PATH='/content/drive/MyDrive/프로젝트/2025_과외_프로젝트1/data'
```

```
data1 = pd.read_pickle(SAVE_PATH+'/음성데이터전처리_1.pkl')
data2 = pd.read_pickle(SAVE_PATH+'/음성데이터전처리_2.pkl')
data3 = pd.read_pickle(SAVE_PATH+'/음성데이터전처리_3.pkl')
data4 = pd.read_pickle(SAVE_PATH+'/음성데이터전처리_4.pkl')

```

```
data=pd.concat([data1,data2])
```

```
data=pd.concat([data,data3])
```

```
data=pd.concat([data,data4])
```

```
# 이제 바로 사용 가능!
print(f"✅ MFCC 타입: {type(data['mfcc'].iloc[0])}")
print(f"✅ MFCC shape: {data['mfcc'].iloc[0].shape}")
```

```
✅ MFCC 타입: <class 'numpy.ndarray'>
✅ MFCC shape: (30, 5935)
```

```
data.isnull().sum()
```

	0
emotion	0
start	0
end	0
text	0
mp3_files	0
label	0
mfcc	0
id	0

dtype: int64

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2876 entries, 0 to 715
Data columns (total 8 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   emotion    2876 non-null   object
 1   start      2876 non-null   object
 2   end        2876 non-null   object
 3   text       2876 non-null   object
 4   mp3_files  2876 non-null   object
 5   label      2876 non-null   object
 6   mfcc       2876 non-null   object
 7   id         2876 non-null   object
dtypes: object(8)
memory usage: 202.2+ KB
```

```
data=data[['text','mfcc','label']]
```

```
data
```

	text	mfcc	label
0	네. 건강해요. 무릎이 까졌어요. 한 일주일 전이요. 아빠랑 같이 살 때부터요. 아...	[[-287.63055, -167.36935, -126.68648, -130.883...	슬픔
1	네. 일주일 전에 머리를 많이 맞았는데 계속 아파요. 아빠한테요. 이마가 찢어져서 ...	[[-280.85858, -154.98929, -73.88041, -53.0979,...	슬픔
2	건강해요. 아니요. 제가 좋아하는 물고기 영상 볼 때요. 제가 물고기를 엄청 좋아해...	[[-293.23975, -226.5664, -236.06046, -279.0968...	슬픔
3	네. 다쳤어요. 손바닥이랑 종아리랑 얼굴이랑 등이요. 병원은 안 갔는데 멍들고 피...	[[-323.36456, -284.55374, -242.94849, -172.724...	불안
4	허벅지랑 입이 아파요. 새엄마가 거짓말한다고 때렸어요. 허벅지는 피랑 멍이 들고 ...	[[-293.35422, -204.26611, -198.02936, -221.593...	불안
...	...	...	...
711	배가 아파요. 모르겠어요. 배 전체가 아픈 것 같아요. 한 달 넘은 것 같아요. 일...	[[-282.7017, -228.5789, -254.6171, -274.73004,...	슬픔
712	아픈 곳 없어요. 없어요. 친구들이랑 놀 때 즐거워요. 그냥 친구들 즐거워요. 화났...	[[-224.39046, -155.57378, -164.24448, -241.605...	슬픔
713	모르겠어요. 네. 없는데요. 없는데요. 애들하고 보드게임 할 때요. 머리를 써서 게...	[[-355.52365, -284.78622, -282.87534, -286.180...	분노
714	네. 아픈 곳 없이 건강해요. 없어요. 사진 찍고 노는 게 즐거워요. 친구들 사진도...	[[-302.62863, -213.26546, -196.8219, -176.1394...	기쁨
715	아픈 곳 없어요. 그런 적도 없어요. 저는 소설 읽을 때가 제일 즐거워요. 그냥 내...	[[-202.66762, -112.82149, -89.933014, -110.204...	슬픔

2876 rows × 3 columns

다음 단계: `data 변수로 코드 생성` `New interactive sheet`

```
for i in data['mfcc']:
    print(i.shape)
    break
```

```
(30, 5935)
```

```
data['mfcc'][0].shape
```

```
(4,)
```

PHQ-9

```
direct_core = [
    "자살 생각이 든다", "죽고싶다", "죽는게 낫다",
    "자해 생각이 든다", "자해할 생각이다"
]
```

```
core_list = [
    "우울하다", "희망이 없다", "즐겁지 않다",
    "행동이 느려졌다", "말이 느려졌다",
    "일상에 집중을 못한다", "실패했다",
    "가족을 실망 시켰다", "기운이 없다", "피곤하다"
]

```

```
indirect_list = [
    "흥미가 없다", "흥미가 떨어지다",
    "식욕이 줄다", "입맛이 없다", "많이 먹다",
    "잠들기 어렵다", "잠을 너무 많이 잔다",
    "가만히 있질 못한다", "안절부절하다", "잘못하고있다"
]
```

```
phq9_symptoms={'direct_core':direct_core,'core_list':core_list,'indirect_list':indirect_list }
```

```
phq9_keys = list(phq9_symptoms.keys())
```

```
phq9_keys
```

```
['direct_core', 'core_list', 'indirect_list']
```

키워드 점수 가중치

- phq-9 스코어 제외

형태소 분석

```
- phq9_symptoms 단어들과 정제된 텍스트 데이터를 각각 형태소 분석해준다
```

```
# #Mecab 설치 (Colab)
# !pip install konlpy
# !pip install mecab-python3
# !bash <(curl -s https://raw.githubusercontent.com/konlpy/konlpy/master/scripts/mecab.sh)
```

```
# from konlpy.tag import Mecab

# # Mecab 객체 생성
# mecab = Mecab()
```

```
# pos_phq9={}

# for i in phq9_keys:
#     #print(phq9_symptoms[i])

#     steam_list=[]

#     for j in phq9_symptoms[i]:
#         #print(j)
#         pos=mecab.pos(j)
#         #print(pos)

#         steam=[]
#         for word, tag in pos:

#             #print(word, tag)

#             if tag.startswith('VV') or tag.startswith('VA'): # VV(동사), VA(형용사)의 경우 어간만 사용
#                 steam.append(word)
#             elif tag.startswith('NN'):  # NNG(일반명사), NNP(고유명사) 등 명사의 경우 단어 자체를 사용
#                 steam.append(word)

#             #print(steam)

#         steam_list.append(' '.join(steam))
#         #print(steam_list)

#     pos_phq9[i]=steam_list


# print(pos_phq9)
```

```
# pos_phq9
```

```
# def pos(i):

#     steam_list=[]

#     pos=mecab.pos(i)
#     #print(pos)

#     steam=[]
#     for word, tag in pos:

#         #print(word, tag)

#         if tag.startswith('VV') or tag.startswith('VA'): # VV(동사), VA(형용사)의 경우 어간만 사용
#             steam.append(word)
#         elif tag.startswith('NN'):  # NNG(일반명사), NNP(고유명사) 등 명사의 경우 단어 자체를 사용
#             steam.append(word)

#         #print(steam)

#     return ' '.join(steam)

```

```
# data['text_pos']=data['text'].apply(lambda x: pos(x))
```

```
# data
```

```
# data['label'].value_counts()
```

phq9_score 생성

```
# phq9_keys
```

```
# import numpy as np

# def create_phq9_vector(text):
#     phq9_secore=0

#     for i,v in enumerate(phq9_keys):

#         if v =='direct_core':

#             for j in pos_phq9[v]:

#                 if j in text:
#                     phq9_secore+=3.0


#         elif v =='core_list':

#             for j in pos_phq9[v]:

#                 if j in text:
#                     phq9_secore+=2.0


#         else:

#             for j in pos_phq9[v]:

#                 if j in text:
#                     phq9_secore+=1.0

#     return  phq9_secore

```

```
# data['phq9_score']=data['text_pos'].apply(lambda x: create_phq9_vector(x))
```

```
# data
```

의미 유사도 가중치

텍스트 데이터 벡터화

```
# max_length 정하기

data['text_length']=data['text'].apply(lambda x: len(x))
```

```
batch_size=32


max_length=int(data['text_length'].quantile(0.90))
print(f'90인 지점:{max_length}')
print(f"최대 텍스트 길이: {data['text_length'].max()}")
print(f"평균 텍스트 길이: {data['text_length'].mean()}")


```

```
90인 지점:1479
최대 텍스트 길이: 3405
평균 텍스트 길이: 1000.46140472879
```

```
import torch
import torch.nn.functional as F
from transformers import AutoTokenizer, AutoModel
from tqdm import tqdm
import numpy as np
from tqdm import tqdm


device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

MODEL_NAME = "klue/roberta-base"
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
klue = AutoModel.from_pretrained(MODEL_NAME).to(device)
```

```
Using device: cuda
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100%                                                 375/375 [00:00<00:00, 44.1kB/s]
vocab.txt:         248k/? [00:00<00:00, 17.1MB/s]
tokenizer.json:         752k/? [00:00<00:00, 46.2MB/s]
special_tokens_map.json: 100%                                                 173/173 [00:00<00:00, 19.2kB/s]
config.json: 100%                                                 546/546 [00:00<00:00, 49.6kB/s]
model.safetensors: 100%                                                 443M/443M [00:06<00:00, 74.5MB/s]
Some weights of RobertaModel were not initialized from the model checkpoint at klue/roberta-base and are newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
def encode_texts(texts, max_length, batch_size):

    if hasattr(texts, 'tolist'):
        texts = texts.tolist()

    with torch.no_grad():

        text_vecs=[]
        for i in tqdm(range(0, len(texts), batch_size)):

            batch=texts[i:i+batch_size]

            token=tokenizer(batch,
                    padding=True,  # 짧은 문장은 패딩 추가
                    truncation=True,  # 긴 문장은 잘라냄
                    max_length=max_length,
                    return_tensors="pt"  # 파이토치 텐서로 반환
                )


            input_ids=token['input_ids'].to(device)
            attention_mask=token['attention_mask'].to(device)

            output=klue.forward(input_ids, attention_mask )
            last_hidden_state=output.last_hidden_state


            mask=attention_mask.unsqueeze(-1).float() # 패딩 제외하기
            mean=(last_hidden_state*mask).sum(dim=1)/mask.sum(dim=1).clamp(min=1e-9) # 평균 계산
            mean=F.normalize(mean, p=2, dim=1) # 단위 벡터로 만들기

            text_vecs.append(mean.detach().cpu())

        text_vecs=torch.cat(text_vecs, dim=0) # 리스트=> 텐서화


    return text_vecs
```

```
print(f"모델 최대 길이: {klue.config.max_position_embeddings}") , print(f"현재 max_length: {max_length}")
```

```
모델 최대 길이: 514
현재 max_length: 1479
(None, None)
```

phq-9 단어 벡터화

```
phq9_words=[]

for i in phq9_symptoms:
    for v in phq9_symptoms[i]:
        phq9_words.append(v)
```

```
phq9_emb=encode_texts(phq9_words, len(phq9_words), batch_size)
```

```
100%|████████| 1/1 [00:00<00:00,  1.40it/s]
```

```
if max_length > 512:
    max_length=512
else:
    pass
```

```
text_emb = encode_texts(data['text'], max_length, batch_size)
```

```
100%|████████| 90/90 [01:29<00:00,  1.01it/s]
```

내적

```
cos_sim=(text_emb @ phq9_emb.T).numpy()

max_sim=cos_sim.max(axis=1)
```

```
data['similarity']=max_sim
```

```
data.head()
```

	text	mfcc	label	text_length	similarity
0	네. 건강해요. 무릎이 까졌어요. 한 일주일 전이요. 아빠랑 같이 살 때부터요. 아...	[[-287.63055, -167.36935, -126.68648, -130.883...	슬픔	826	0.511867
1	네. 일주일 전에 머리를 많이 맞았는데 계속 아파요. 아빠한테요. 이마가 찢어져서 ...	[[-280.85858, -154.98929, -73.88041, -53.0979,...	슬픔	781	0.509609
2	건강해요. 아니요. 제가 좋아하는 물고기 영상 볼 때요. 제가 물고기를 엄청 좋아해...	[[-293.23975, -226.5664, -236.06046, -279.0968...	슬픔	1145	0.494879
3	네. 다쳤어요. 손바닥이랑 종아리랑 얼굴이랑 등이요. 병원은 안 갔는데 멍들고 피...	[[-323.36456, -284.55374, -242.94849, -172.724...	불안	899	0.472573
4	허벅지랑 입이 아파요. 새엄마가 거짓말한다고 때렸어요. 허벅지는 피랑 멍이 들고 ...	[[-293.35422, -204.26611, -198.02936, -221.593...	불안	1514	0.495485

다음 단계: data 변수로 코드 생성 | New interactive sheet

최종 점수 계산

- total_all = {(np.array(kw_scores_all) * W_KW) + (max_sim_all * W_SEM)} * TARGET_CATS

- W_KW = 1.0 # 키워드 점수 가중치 자살
- W_SEM = 0.6 # 의미 유사도 가중치
- TARGET_CATS = ["학업 및 진로", "학교폭력/따돌림"] CATEGORY_BOOST = 1.5

```
# W_KW=1.0
# W_SEM= 0.6
```

```
#concat_data['TARGET_CATS']=concat_data['상황키워드'].apply(lambda x: 1.5 if x in ["학업 및 진로", "학교폭력/따돌림"] else 1.0)
```

```
# data['phq9_total_score']=((data['phq9_score']*W_KW)+(data['similarity']*W_SEM))
```

```
# data.columns
```

```
# data.head()
```

```
data=data[['text','similarity','mfcc','label']]
```

```
data['label'].value_counts()
```

label	count
슬픔	1450
불안	659
기쁨	561
분노	94
당황	89
상처	23

dtype: int64

```
data.head()
```

	text	similarity	mfcc	label
0	네. 건강해요. 무릎이 까졌어요. 한 일주일 전이요. 아빠랑 같이 살 때부터요. 아...	0.511867	[[-287.63055, -167.36935, -126.68648, -130.883...	슬픔
1	네. 일주일 전에 머리를 많이 맞았는데 계속 아파요. 아빠한테요. 이마가 찢어져서 ...	0.509609	[[-280.85858, -154.98929, -73.88041, -53.0979,...	슬픔
2	건강해요. 아니요. 제가 좋아하는 물고기 영상 볼 때요. 제가 물고기를 엄청 좋아해...	0.494879	[[-293.23975, -226.5664, -236.06046, -279.0968...	슬픔
3	네. 다쳤어요. 손바닥이랑 종아리랑 얼굴이랑 등이요. 병원은 안 갔는데 멍들고 피...	0.472573	[[-323.36456, -284.55374, -242.94849, -172.724...	불안
4	허벅지랑 입이 아파요. 새엄마가 거짓말한다고 때렸어요. 허벅지는 피랑 멍이 들고 ...	0.495485	[[-293.35422, -204.26611, -198.02936, -221.593...	불안

다음 단계: data 변수로 코드 생성 | New interactive sheet

mfcc(max length) 패딩

```
# 라이브러리 임포트
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader
from tqdm import tqdm
import matplotlib.pyplot as plt
```

```
## mfcc 최대 길이
time_steps = [arr.shape[1] for arr in data['mfcc']]
max_time = max(time_steps)
percentile_90 = int(np.percentile(time_steps, 90))
mean_time = np.mean(time_steps)

print(f"최대 길이: {max_time}")
print(f"90% 분위수: {percentile_90}")
print(f"평균 길이: {mean_time:.1f}")

# 90% 분위수 사용
MAX_TIME_STEPS = percentile_90

```

```
최대 길이: 18000
90% 분위수: 8144
평균 길이: 5648.4
```

```
# 2. 패딩 함수
def pad_mfcc(mfcc_array, max_length):

    current_length = mfcc_array.shape[1]

    if current_length < max_length:
        # Zero padding
        pad_width = ((0, 0), (0, max_length - current_length))
        return np.pad(mfcc_array, pad_width, mode='constant', constant_values=0)
    else:
        # Truncate
        return mfcc_array[:, :max_length]
```

```
data['mfcc_padded'] = data['mfcc'].apply(lambda x: pad_mfcc(x, MAX_TIME_STEPS))
```

모델링

multimodal

```
from transformers import AutoTokenizer
import numpy as np
```

```
data['text']
```

	text
0	네. 건강해요. 무릎이 까졌어요. 한 일주일 전이요. 아빠랑 같이 살 때부터요. 아...
1	네. 일주일 전에 머리를 많이 맞았는데 계속 아파요. 아빠한테요. 이마가 찢어져서 ...
2	건강해요. 아니요. 제가 좋아하는 물고기 영상 볼 때요. 제가 물고기를 엄청 좋아해...
3	네. 다쳤어요. 손바닥이랑 종아리랑 얼굴이랑 등이요. 병원은 안 갔는데 멍들고 피...
4	허벅지랑 입이 아파요. 새엄마가 거짓말한다고 때렸어요. 허벅지는 피랑 멍이 들고 ...
...	...
711	배가 아파요. 모르겠어요. 배 전체가 아픈 것 같아요. 한 달 넘은 것 같아요. 일...
712	아픈 곳 없어요. 없어요. 친구들이랑 놀 때 즐거워요. 그냥 친구들 즐거워요. 화났...
713	모르겠어요. 네. 없는데요. 없는데요. 애들하고 보드게임 할 때요. 머리를 써서 게...
714	네. 아픈 곳 없이 건강해요. 없어요. 사진 찍고 노는 게 즐거워요. 친구들 사진도...
715	아픈 곳 없어요. 그런 적도 없어요. 저는 소설 읽을 때가 제일 즐거워요. 그냥 내...

2876 rows × 1 columns

dtype: object

```
# Klue 토크나이저 로드
tokenizer = AutoTokenizer.from_pretrained('klue/roberta-base')
```

```
all_text_token_length=[]
```

```
for text in data['text']:
    encoding=tokenizer(text, add_special_tokens=True)
    all_text_token_length.append(len(encoding['input_ids']))

print(f'총 {len(all_text_token_length)}개의 문장에 대한 토큰 길이 계산 완료')
print(f'예시 앞 5개 문장길이 {all_text_token_length[:5]}')
```

```
Token indices sequence length is longer than the specified maximum sequence length for this model (605 > 512). Running this sequence through the model will result in indexing errors
총 2876개의 문장에 대한 토큰 길이 계산 완료
예시 앞 5개 문장길이 [417, 417, 605, 474, 821]
```

```
print('모든 문장에서')
print(f'토큰 길이 최댓값은 {max(all_text_token_length)} 이다.')
print(f'토큰 길이 최솟값은 {min(all_text_token_length)} 이다.')

print(f'토큰 길이 평균은 {np.mean(all_text_token_length)} 이다.')
print(f'토큰 길이 표준편차는 {np.std(all_text_token_length)} 이다.')


print("-" * 30)
print(f"90% 지점의 길이: {np.percentile(all_text_token_length, 90)}")
print(f"95% 지점의 길이: {np.percentile(all_text_token_length, 95)}")
print(f"99% 지점의 길이: {np.percentile(all_text_token_length, 99)}")
```

```
모든 문장에서
토큰 길이 최댓값은 1793 이다.
토큰 길이 최솟값은 200 이다.
토큰 길이 평균은 528.8689151599443 이다.
토큰 길이 표준편차는 184.70196794135694 이다.
------------------------------
90% 지점의 길이: 784.0
95% 지점의 길이: 890.25
99% 지점의 길이: 1120.0
```

```
text_max_length=890.25
```

```
if text_max_length>512:
    text_max_length=512
```

```
text_max_length
```

```
512
```

데이터로드 함수

```
from transformers import AutoModel, AutoTokenizer
from sklearn.preprocessing import LabelEncoder
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
import random
import copy
from torch.utils.data import Dataset, DataLoader, WeightedRandomSampler

batch_size=32
```

```
def multimodal_data_load(text, max_length):

    tokenizer = AutoTokenizer.from_pretrained('klue/roberta-base')

    label_encoder = LabelEncoder()
    label_encoded = label_encoder.fit_transform(text['label'])
    text = text.copy()
    text['label_encoded'] = label_encoded

    label_count = text['label'].nunique()

    print(f"클래스 수: {label_count}")
    for idx, label_name in enumerate(label_encoder.classes_):
        print(f"  {idx}: {label_name}")


    train_idx, test_idx = train_test_split(
        np.arange(len(text)),
        test_size=0.3,
        stratify=text['label_encoded'],
        random_state=42
    )

    train_data = text.iloc[train_idx].reset_index(drop=True)
    test_data = text.iloc[test_idx].reset_index(drop=True)









    # ============================================================
    # 🔥 [핵심] WeightedRandomSampler 설정
    # 데이터프레임을 늘리지 않고, 불균형한 데이터를 균형 잡히게 뽑아줍니다.
    # ============================================================

    # 1. 클래스별 샘플 수 계산
    class_counts = train_data['label_encoded'].value_counts().sort_index()

    # 2. 클래스별 가중치 계산 (개수가 적을수록 가중치가 높음)
    # weight = 1 / count
    class_weights = 1.0 / class_counts
    class_weights = torch.tensor(class_weights.values, dtype=torch.float)

    # 3. 각 샘플(Row)마다 가중치 부여
    # 예: 우울증 데이터는 가중치 10, 일반 데이터는 가중치 1
    sample_weights = [class_weights[label] for label in train_data['label_encoded']]
    sample_weights = torch.tensor(sample_weights, dtype=torch.float)

    # 4. 샘플러 생성
    # num_samples를 지정하여 전체 데이터 크기를 조절할 수 있음 (여기선 원본 크기 유지하거나 늘릴 수 있음)
    # 보통 불균형 해소를 위해 충분히 뽑기 위해 len(train_data) 그대로 사용하거나 늘림.
    sampler = WeightedRandomSampler(
        weights=sample_weights,
        num_samples=len(train_data), # 혹은 len(train_data) * 2 등으로 늘려도 RAM 안 터짐
        replacement=True
    )
    # ============================================================









    # Dataset 생성
    train_dataset = TextDataset(
        texts=train_data['text'],
        phq9_features=train_data[['similarity']],
        mfcc=train_data['mfcc_padded'],
        labels=train_data['label_encoded'],
        tokenizer=tokenizer,
        max_length=max_length,
        augment=True
    )

    test_dataset = TextDataset(
        texts=test_data['text'],
        phq9_features=test_data[['similarity']],
        mfcc=test_data['mfcc_padded'],
        labels=test_data['label_encoded'],
        tokenizer=tokenizer,
        max_length=max_length,
        augment=False
    )
```

```

    # DataLoader
    train_loader = DataLoader(train_dataset, batch_size=batch_size,sampler=sampler, shuffle=False)
    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

    print(f"✅ 훈련 데이터: {len(train_dataset)}개")

    print(f"✅ 테스트 데이터: {len(test_dataset)}개")

    return train_loader, test_loader, label_count
```

```
class TextDataset(Dataset):
    def __init__(self, texts, phq9_features, mfcc, labels, tokenizer, max_length, augment=False):
        self.texts = texts.reset_index(drop=True)  # 확실하게 리셋
        self.labels = labels.reset_index(drop=True) # 확실하게 리셋
        self.tokenizer=tokenizer
        self.max_length=int(max_length)
        self.phq9_features=phq9_features.reset_index(drop=True)
        self.mfcc = mfcc.reset_index(drop=True)

        self.augment = augment #증강처리

    def __len__(self):
        return len(self.texts)



    # --- 내부 증강 함수 ---
    def augment_mfcc(self, mfcc_data):
        aug_mfcc = copy.deepcopy(mfcc_data)
        noise = np.random.normal(0, 0.02, aug_mfcc.shape)
        return aug_mfcc + noise

    def augment_text(self, text):
        words = text.split()
        if len(words) <= 1: return text
        if random.random() > 0.5:
            idx1, idx2 = random.sample(range(len(words)), 2)
            words[idx1], words[idx2] = words[idx2], words[idx1]
        else:
            if len(words) > 2:
                del words[random.randint(0, len(words)-1)]
        return ' '.join(words)
    # ----------------------



    def __getitem__(self, idx):
        text = str(self.texts.iloc[idx])
        phq9_features=self.phq9_features.iloc[idx].values.astype(np.float32)
        label = int(self.labels.iloc[idx])

        mfcc_data = self.mfcc.iloc[idx]


        if self.augment:
          text = self.augment_text(text)
          mfcc_data = self.augment_mfcc(mfcc_data)




        mfcc_data = np.expand_dims(mfcc_data, axis=0)


        encoding=self.tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=self.max_length,
            padding='max_length',
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt'
             )
        return (
                encoding['input_ids'].flatten(),
                encoding['attention_mask'].flatten(),
                torch.tensor(phq9_features, dtype=torch.float32),
                torch.tensor(mfcc_data, dtype=torch.float32),
                torch.tensor(label, dtype=torch.long)
        )
```

모델 함수

```
from transformers import AutoModel,AutoConfig
import torch.nn as nn
from tqdm import tqdm

```

```
class LSTMAudioClassifier(nn.Module):
    def __init__(self, num_class, input_size=30, hidden_size=128, num_layers=2, dropout=0.3, target_length=500):
        super(LSTMAudioClassifier, self).__init__()

        self.target_length = target_length

        self.lstm = nn.LSTM(
            input_size=input_size,
            hidden_size=hidden_size,
            num_layers=num_layers,
            batch_first=True,
            dropout=dropout,
            bidirectional=True
        )
        self.dropout = nn.Dropout(dropout)
        self.classifier = nn.Linear(hidden_size*2, num_class)

    def forward(self, mfcc):
        # mfcc shape: (batch, 1, 30, 8144)
        x = mfcc.squeeze(1)      # (batch, 30, 8144)


        # Adaptive Average Pooling으로 시퀀스 길이 축소
        x = F.adaptive_avg_pool1d(x, output_size=self.target_length)
        x = x.permute(0, 2, 1)

        lstm_out, _ = self.lstm(x)
        last_output = lstm_out[:, -1, :]

        x = self.dropout(last_output)
        y_pred = self.classifier(x)

        return y_pred
```

```
class klueClassifier(nn.Module):
    def __init__(self, num_class, num_layers_to_train=3, dropout=0.3):
        super(klueClassifier,self).__init__()

        self.klue=AutoModel.from_pretrained('klue/roberta-base')
        config= AutoConfig.from_pretrained('klue/roberta-base')
        hidden_size=config.hidden_size

        # ========================================
        # 부분 파인튜닝
        total_layers = 12  # klue는 12개 레이어
        layers_to_freeze = total_layers - num_layers_to_train

        for param in self.klue.embeddings.parameters():
            param.requires_grad = False

        # 2) 처음 N개 레이어 freeze
        for i in range(layers_to_freeze):
```

```
            for param in self.klue.encoder.layer[i].parameters():
                param.requires_grad = False
        # ========================================


        self.phq9_feature_layer=nn.Linear(1,16)

        self.dropout=nn.Dropout(dropout)
        self.classifier= nn.Linear(hidden_size+16, num_class)




    def forward(self, input_ids, attention_mask, phq9_features):

        output=self.klue(input_ids=input_ids, attention_mask=attention_mask)
        cls_token=output.pooler_output


        phq9=self.phq9_feature_layer(phq9_features)
        phq9=torch.relu(phq9)
        combined =torch.cat([cls_token, phq9], dim=1)


        x=self.dropout(combined)
        y_pred=self.classifier(x)


        return y_pred
```

```
class MultiModalClassifier(nn.Module):
    def __init__(self, num_class, text_num_layers_to_train=3, audio_input_size=30, audio_hidden_size=128, audio_target_length=500):
        super(MultiModalClassifier, self).__init__()

        # 텍스트 + PHQ9 모델
        self.text_model = klueClassifier(
            num_class=num_class,
            num_layers_to_train=text_num_layers_to_train
        )

        # 음성(MFCC) 모델
        self.audio_model = LSTMAudioClassifier(
            num_class=num_class,
            input_size=audio_input_size,
            hidden_size=audio_hidden_size,
            target_length=audio_target_length
        )

    def forward(self, input_ids, attention_mask, phq9_features, mfcc):
        text_y = self.text_model(input_ids, attention_mask, phq9_features)
        audio_y = self.audio_model(mfcc)

        final_y = (text_y + audio_y) / 2.0

        return final_y
```

```

def multimodal_train(train_loader, model, criterion, optimizer, device, epochs=6):

    history = {'Epoch': [], 'loss': []}

    for i in range(epochs):
        model.train()
        total_loss=0

        for x, attention_mask, phq9, mfcc, y in tqdm(train_loader):

            x= x.to(device)
            attention_mask=attention_mask.to(device)
            phq9=phq9.to(device)
            mfcc=mfcc.to(device)
            y= y.to(device)

            y_pred=model(x, attention_mask, phq9, mfcc)
            loss=criterion(y_pred, y)

            optimizer.zero_grad()
            loss.backward()

            optimizer.step()

            total_loss+=loss.item()


        history['loss'].append(total_loss / len(train_loader))
        history['Epoch'].append(i)

        if i % 2 == 0:
            print (f'Epoch {i}: Loss={total_loss/len(train_loader)}')

    return history

```

모델평가 함수

```
from sklearn.metrics import(
    accuracy_score,
    f1_score,
    precision_score,
    recall_score,
    classification_report,
    confusion_matrix
)
import numpy as np


```

```
def multimodal_eval(test_loader, model, depression_classes, device=device, label_names=None ):
    model.eval()

    all_preds=[]
    all_labels=[]

    with torch.no_grad():
        for x, attention_mask, phq9, mfcc, y in test_loader:  # ✅ 3개!
            x = x.to(device)
            attention_mask = attention_mask.to(device)
            phq9=phq9.to(device)
            mfcc=mfcc.to(device)

            y_pred = model(x, attention_mask, phq9, mfcc)  # ✅ attention_mask 포함
            preds = torch.argmax(y_pred, dim=1)

            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(y.cpu().numpy())



    accuracy = accuracy_score(all_labels, all_preds)
    f1_macro = f1_score(all_labels, all_preds, average='macro')
    f1_weighted = f1_score(all_labels, all_preds, average='weighted')
    cm = confusion_matrix(all_labels, all_preds)

    print(f"\n✅ Accuracy: {accuracy:.4f}")
    print(f"✅ Macro F1-Score: {f1_macro:.4f}")
    print(f"✅ Weighted F1-Score: {f1_weighted:.4f}")
    print(f"\n🔢 Confusion Matrix:")
    print(cm)



    # ========================================
    # 3. 우울 신호 집중 분석
    # ========================================
```

```
    print("₩n" + "="*60)
    print(f"🎯 우울 신호 집중 분석 ({depression_classes})")
    print("="*60)

    # 이진 분류로 변환
    all_labels_np = np.array(all_labels)
    all_preds_np = np.array(all_preds)

    y_true_binary = np.isin(all_labels_np, depression_classes).astype(int)
    y_pred_binary = np.isin(all_preds_np, depression_classes).astype(int)

    # Precision & Recall
    precision = precision_score(y_true_binary, y_pred_binary, zero_division=0)
    recall = recall_score(y_true_binary, y_pred_binary, zero_division=0)
    f1 = f1_score(y_true_binary, y_pred_binary, zero_division=0)



    # 실제 우울 신호인데 기타로 예측
    false_negatives = np.sum((y_true_binary == 1) & (y_pred_binary == 0))
    # True Positive
    true_positives = np.sum((y_true_binary == 1) & (y_pred_binary == 1))
    # 전체 우울 신호
    total_depression = np.sum(y_true_binary == 1)



    print(f"₩n{'='*60}")
    print("🎯 우울 신호 집중 분석 (불안 + 상처 + 슬픔)")
    print(f"{'='*60}")
    print(f"✅ Precision: {precision:.4f}")
    print(f"✅ Recall: {recall:.4f}")
    print(f"✅ F1-Score: {f1:.4f}")
    print(f"₩n⚠️ False Negative: {false_negatives}건 / {total_depression}건")
    print(f"✅ True Positive: {true_positives}건 / {total_depression}건")


    # Classification Report
    if label_names: # = ['기쁨', '당황', '분노', '불안', '상처', '슬픔']
        print(f"₩n📋 Classification Report:")
        print(classification_report(all_labels, all_preds, target_names=label_names))



    return {
        'accuracy': accuracy,
        'f1_macro': f1_macro,
        'f1_weighted': f1_weighted,
        'confusion_matrix': cm,
        'depression_precision': precision,
        'depression_recall': recall,
        'depression_f1': f1,
        'false_negatives': false_negatives,
        'true_positives': true_positives,
        'total_depression': total_depression,
    }

```

데이터 입력

```
total_result=[]
```

```
results={'task':'six_label_all_text_phq9_multimodal',
         'text': 'all',
         'label':'six',
         'model':'multimodal'
                 }
```

```
max_length=text_max_length

batch_size=8 # 메모리 문제
epochs=6
```

```
from torch.utils.data import Dataset, DataLoader, WeightedRandomSampler

train_loader, test_loader, label_count=multimodal_data_load(data, max_length)
```

```
클래스 수: 6
  0: 기쁨
  1: 당황
  2: 분노
  3: 불안
  4: 상처
  5: 슬픔
✅ 훈련 데이터: 2013개
✅ 테스트 데이터: 863개
```

```
import torch.nn.functional as F

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

MultiModal_model = MultiModalClassifier(
    num_class=label_count,
    text_num_layers_to_train=3,
    audio_input_size=30,
    audio_hidden_size=128,
    audio_target_length=200
).to(device)

MultiModal_criterion = nn.CrossEntropyLoss()
MultiModal_optimizer = torch.optim.AdamW(
    MultiModal_model.parameters(),
    lr=1e-5,
    weight_decay=0.01
)

print(f"✅ 모델 생성 완료!")
```

```
Some weights of RobertaModel were not initialized from the model checkpoint at klue/roberta-base and are newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
✅ 모델 생성 완료!
```

```
multimodal_train(train_loader, MultiModal_model, MultiModal_criterion, MultiModal_optimizer, device, epochs=6)
```

```
100%|██████████| 252/252 [01:51<00:00,  2.25it/s]
Epoch 0: Loss=1.7649880581431918
100%|██████████| 252/252 [01:50<00:00,  2.27it/s]
100%|██████████| 252/252 [01:50<00:00,  2.27it/s]
Epoch 2: Loss=0.8907697631253136
100%|██████████| 252/252 [01:50<00:00,  2.27it/s]
100%|██████████| 252/252 [01:51<00:00,  2.27it/s]
Epoch 4: Loss=0.5823732987046242
100%|██████████| 252/252 [01:50<00:00,  2.28it/s]
{'Epoch': [0, 1, 2, 3, 4, 5],
 'loss': [1.7649880581431918,
  1.2980793125572658,
  0.8907697631253136,
  0.6852923324183812,
  0.5823732987046242,
  0.503316495153639 1]}
```

```
torch.save(MultiModal_model.state_dict(), SAVE_PATH + 'phase3_six_label_all_text_phq9_multimodal.pt')
print("모델 저장 완료")
```

```
모델 저장 완료
```

```
six_label_depression_classes = [3, 4, 5]
four_label_depression_classes = [1]
```

```
result=multimodal_eval(test_loader, MultiModal_model, six_label_depression_classes)
```

```
✅ Accuracy: 0.6466
```

```
✅ Macro F1-Score: 0.4734
✅ Weighted F1-Score: 0.6442

🟦 Confusion Matrix:
[[158   2   4   1   0   3]
 [ 14   6   2   2   0   3]
 [  4   4  15   3   0   2]
 [ 12   1   2 162   0  21]
 [  1   1   1   1   1   2]
 [ 42  15  36 124   2 216]]

============================================================
🎯 우울 신호 집중 분석 ([3, 4, 5])
============================================================

============================================================
🎯 우울 신호 집중 분석 (불안 + 상처 + 슬픔)
============================================================
✅ Precision: 0.9742
✅ Recall: 0.8266
✅ F1-Score: 0.8943

⚠️ False Negative: 111건 / 640건
✅ True Positive: 529건 / 640건
```

```
results.update(result)
```

```
total_result.append(results)
```

```
import gc
del MultiModal_model
del MultiModal_optimizer
torch.cuda.empty_cache()
gc.collect()
```

```
4769
```

Fine_Tuning

데이터로드 함수

```
max_length
```

```
512
```

```
from transformers import AutoModel, AutoTokenizer
from sklearn.preprocessing import LabelEncoder
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split


batch_size=32
```

```
# data_phq9=data[['text','label','phq9_score','similarity','phq9_total_score']]

data_phq9 = data[['text', 'label', 'similarity']]
```

```

class TextDataset(Dataset):
    def __init__(self, texts, phq9_features, labels, tokenizer, max_length, augment=False):
        self.texts = texts.reset_index(drop=True)  # 확실하게 리셋
        self.labels = labels.reset_index(drop=True) # 확실하게 리셋
        self.tokenizer=tokenizer
        self.max_length=int(max_length)
        self.phq9_features=phq9_features.reset_index(drop=True)
        self.augment = augment

    def __len__(self):
        return len(self.texts)



    # --- 내부 텍스트 증강 함수 ---
    def augment_text(self, text):
        words = text.split()
        if len(words) <= 1: return text

        # 50% 확률로 순서 섞기, 50% 확률로 단어 삭제
        if random.random() > 0.5:
            idx1, idx2 = random.sample(range(len(words)), 2)
            words[idx1], words[idx2] = words[idx2], words[idx1]
        else:
            if len(words) > 2:
                del words[random.randint(0, len(words)-1)]
        return ' '.join(words)
    # ----------------------------



    def __getitem__(self, idx):
        text = str(self.texts.iloc[idx])

        # ✅ [핵심] Train 모드일 때만 텍스트 실시간 증강
        if self.augment:
          text = self.augment_text(text)



        phq9_features=self.phq9_features.iloc[idx].values.astype(np.float32)
        label = int(self.labels.iloc[idx])

        encoding=self.tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=self.max_length,
            padding='max_length',
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt'
             )
        return (
                encoding['input_ids'].flatten(),
                encoding['attention_mask'].flatten(),
                torch.tensor(phq9_features, dtype=torch.float32),
                torch.tensor(label, dtype=torch.long)
        )
```

```
def FineTuning_data_load(text, max_length):

    tokenizer = AutoTokenizer.from_pretrained('klue/roberta-base')

    label_encoder = LabelEncoder()
    label_encoded = label_encoder.fit_transform(text['label'])
    text = text.copy()
    text['label_encoded'] = label_encoded

    label_count = text['label'].nunique()

    print(f"클래스 수: {label_count}")
    for idx, label_name in enumerate(label_encoder.classes_):
        print(f"  {idx}: {label_name}")


    train_idx, test_idx = train_test_split(
        np.arange(len(text)),
        test_size=0.3,
        stratify=text['label_encoded'],
        random_state=42
    )

    train_data = text.iloc[train_idx].reset_index(drop=True)
    test_data = text.iloc[test_idx].reset_index(drop=True)


```

```

    print("⚠️ 메모리 절약을 위해 WeightedRandomSampler(실시간 밸런싱)를 사용합니다.")

    # ==========================================================
    # 🔥 [핵심] WeightedRandomSampler 설정 (데이터 복사 없이 밸런싱)
    # ==========================================================

    # 1. 클래스별 샘플 수 계산
    class_counts = train_data['label_encoded'].value_counts().sort_index()

    # 2. 클래스별 가중치 계산 (개수가 적을수록 가중치 높음)
    class_weights = 1.0 / class_counts
    class_weights = torch.tensor(class_weights.values, dtype=torch.float)

    # 3. 각 샘플마다 가중치 부여
    sample_weights = [class_weights[label] for label in train_data['label_encoded']]
    sample_weights = torch.tensor(sample_weights, dtype=torch.float)

    # 4. 샘플러 생성 (데이터를 늘리고 싶으면 num_samples를 len(train_data) * 2 등으로 설정)
    sampler = WeightedRandomSampler(
        weights=sample_weights,
        num_samples=len(train_data), # 원본 데이터 개수 유지 (원하면 늘려도 됨)
        replacement=True
    )
    # ==========================================================




    # Dataset 생성
    train_dataset = TextDataset(
        texts=train_data['text'],
        phq9_features=train_data[['similarity']],
        labels=train_data['label_encoded'],
        tokenizer=tokenizer,
        max_length=max_length,
        augment=True
    )

    test_dataset = TextDataset(
        texts=test_data['text'],
        phq9_features=test_data[['similarity']],
        labels=test_data['label_encoded'],
        tokenizer=tokenizer,
        max_length=max_length,
        augment=False
    )

    # DataLoader
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=False, sampler=sampler)
    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

    print(f"✅ 훈련 데이터: {len(train_dataset)}개")
    print(f"✅ 테스트 데이터: {len(test_dataset)}개")

    return train_loader, test_loader, label_count
```

모델 함수

```
from transformers import AutoModel,AutoConfig
import torch.nn as nn
from tqdm import tqdm

```

```
class klueClassifier(nn.Module):
    def __init__(self, num_class, num_layers_to_train=3, dropout=0.3):
        super(klueClassifier,self).__init__()

        self.klue=AutoModel.from_pretrained('klue/roberta-base')
        config= AutoConfig.from_pretrained('klue/roberta-base')
        hidden_size=config.hidden_size

        # ==========================================
        # 부분 파인튜닝
        total_layers = 12  # klue는 12개 레이어
        layers_to_freeze = total_layers - num_layers_to_train

        for param in self.klue.embeddings.parameters():
            param.requires_grad = False

        # 2) 처음 N개 레이어 freeze
        for i in range(layers_to_freeze):
            for param in self.klue.encoder.layer[i].parameters():
                param.requires_grad = False
        # ==========================================


        self.phq9_feature_layer=nn.Linear(1,16)

        self.dropout=nn.Dropout(dropout)
        self.classifier= nn.Linear(hidden_size+16, num_class)




    def forward(self, input_ids, attention_mask, phq9_features):

        output=self.klue(input_ids=input_ids, attention_mask=attention_mask)
        cls_token=output.pooler_output


        phq9=self.phq9_feature_layer(phq9_features)
        phq9=torch.relu(phq9)
        combined =torch.cat([cls_token, phq9], dim=1)


        x=self.dropout(combined)
        y_pred=self.classifier(x)


        return y_pred


```

```

def FineTuning_train(train_loader, model, criterion, optimizer, device=device, epochs=3):

    history = {'Epoch': [], 'loss': []}

    for i in range(epochs):
        model.train()
        total_loss=0

        for x, attention_mask, phq9, y in tqdm(train_loader):

            x= x.to(device)
            attention_mask=attention_mask.to(device)
            phq9=phq9.to(device)
            y= y.to(device)

            y_pred=model(x, attention_mask, phq9)
            loss=criterion(y_pred, y)

            optimizer.zero_grad()
            loss.backward()

            optimizer.step()

            total_loss+=loss.item()


        history['loss'].append(total_loss / len(train_loader))
        history['Epoch'].append(i)

```

```
        if i % 2 == 0:
            print (f'Epoch {i}: Loss={total_loss/len(train_loader)}')

    return history

```

모델평가 함수

```
from sklearn.metrics import(
    accuracy_score,
    f1_score,
    precision_score,
    recall_score,
    classification_report,
    confusion_matrix
)
import numpy as np


```

```
def FineTuning_eval(test_loader, model, depression_classes, device=device, label_names=None ):
    model.eval()

    all_preds=[]
    all_labels=[]

    with torch.no_grad():
        for x, attention_mask, phq9, y in test_loader:  # ✅ 3개!
            x = x.to(device)
            attention_mask = attention_mask.to(device)
            phq9=phq9.to(device)

            y_pred = model(x, attention_mask, phq9)  # ✅ attention_mask 포함
            preds = torch.argmax(y_pred, dim=1)

            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(y.cpu().numpy())



    accuracy = accuracy_score(all_labels, all_preds)
    f1_macro = f1_score(all_labels, all_preds, average='macro')
    f1_weighted = f1_score(all_labels, all_preds, average='weighted')
    cm = confusion_matrix(all_labels, all_preds)

    print(f"\n✅ Accuracy: {accuracy:.4f}")
    print(f"✅ Macro F1-Score: {f1_macro:.4f}")
    print(f"✅ Weighted F1-Score: {f1_weighted:.4f}")
    print(f"\n🔢 Confusion Matrix:")
    print(cm)



    # ========================================
    # 3. 우울 신호 집중 분석
    # ========================================


    print("\n" + "="*60)
    print(f"🎯 우울 신호 집중 분석 ({depression_classes})")
    print("="*60)


    # 이진 분류로 변환
    all_labels_np = np.array(all_labels)
    all_preds_np = np.array(all_preds)

    y_true_binary = np.isin(all_labels_np, depression_classes).astype(int)
    y_pred_binary = np.isin(all_preds_np, depression_classes).astype(int)

    # Precision & Recall
    precision = precision_score(y_true_binary, y_pred_binary, zero_division=0)
    recall = recall_score(y_true_binary, y_pred_binary, zero_division=0)
    f1 = f1_score(y_true_binary, y_pred_binary, zero_division=0)



    # 실제 우울 신호인데 기타로 예측
    false_negatives = np.sum((y_true_binary == 1) & (y_pred_binary == 0))
    # True Positive
    true_positives = np.sum((y_true_binary == 1) & (y_pred_binary == 1))
    # 전체 우울 신호
    total_depression = np.sum(y_true_binary == 1)



    print(f"\n{'='*60}")
    print("🎯 우울 신호 집중 분석 (불안 + 상처 + 슬픔)")
    print(f"{'='*60}")
    print(f"✅ Precision: {precision:.4f}")
    print(f"✅ Recall: {recall:.4f}")
    print(f"✅ F1-Score: {f1:.4f}")
    print(f"\n⚠️ False Negative: {false_negatives}건 / {total_depression}건")
    print(f"✅ True Positive: {true_positives}건 / {total_depression}건")


    # Classification Report
    if label_names: # = ['기쁨', '당황', '분노', '불안', '상처', '슬픔']
        print(f"\n📋 Classification Report:")
        print(classification_report(all_labels, all_preds, target_names=label_names))



    return {
        'accuracy': accuracy,
        'f1_macro': f1_macro,
        'f1_weighted': f1_weighted,
        'confusion_matrix': cm,
        'depression_precision': precision,
        'depression_recall': recall,
        'depression_f1': f1,
        'false_negatives': false_negatives,
        'true_positives': true_positives,
        'total_depression': total_depression,
    }

```

데이터 입력

```
phq9_total_result=[]



results={'task':'six_label_all_text_phq9',
         'text': 'all',
         'label':'six',
         'model':'finetuning',
                  }
```

```
batch_size=8
epoch=6
```

```
six_label_depression_classes = [3, 4, 5]
four_label_depression_classes = [1]
```

```
train_loader, test_loader, label_count=FineTuning_data_load(data_phq9, max_length)
```

```
클래스 수: 6
  0: 기쁨
  1: 당황
  2: 분노
  3: 불안
  4: 상처
  5: 슬픔
```

```
⚠️ 메모리 절약을 위해 WeightedRandomSampler(실시간 밸런싱)를 사용합니다.
✅ 훈련 데이터: 2013개
✅ 테스트 데이터: 863개
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
finetuning_model = klueClassifier(num_class=label_count, num_layers_to_train=3, dropout=0.3).to(device)


FineTuning_criterion=nn.CrossEntropyLoss()
FineTuning_optimizer = torch.optim.AdamW([
    {'params': finetuning_model.klue.parameters(), 'lr': 2e-5},      # KLUE는 작게
    {'params': finetuning_model.classifier.parameters(), 'lr': 1e-4}  # 분류기는 크게
], weight_decay=0.01)
```

```
Some weights of RobertaModel were not initialized from the model checkpoint at klue/roberta-base and are newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
FineTuning_train(train_loader, finetuning_model, FineTuning_criterion, FineTuning_optimizer, device, epochs=6)
```

```
100%|████████| 252/252 [01:34<00:00,  2.68it/s]
Epoch 0: Loss=1.4960583046315208
100%|████████| 252/252 [01:33<00:00,  2.69it/s]
100%|████████| 252/252 [01:33<00:00,  2.69it/s]
Epoch 2: Loss=0.5967151057270784
100%|████████| 252/252 [01:33<00:00,  2.69it/s]
100%|████████| 252/252 [01:33<00:00,  2.69it/s]
Epoch 4: Loss=0.3523830319680864 6
100%|████████| 252/252 [01:33<00:00,  2.69it/s]
{'Epoch': [0, 1, 2, 3, 4, 5],
 'loss': [1.4960583046315208,
  0.7804750747505634,
  0.5967151057270784,
  0.4289772048563001 4,
  0.3523830319680864 6,
  0.3034725650582315 ]}
```

```
torch.save(finetuning_model.state_dict(), SAVE_PATH + 'phase3_six_label_all_text_phq9.pt')
print("모델 저장 완료")
```

```
모델 저장 완료
```

```
six_label_depression_classes = [3, 4, 5]
four_label_depression_classes = [1]
```

```
result=FineTuning_eval(test_loader, finetuning_model, six_label_depression_classes, device=device)
```

```
✅ Accuracy: 0.7115
✅ Macro F1-Score: 0.5136
✅ Weighted F1-Score: 0.7115

🔢 Confusion Matrix:
[[145   6   1   3   0  13]
 [ 12   8   1   0   0   6]
 [  5   1  11   0   1  10]
 [  9   1   2 142   1  43]
 [  1   1   0   1   1   3]
 [ 24  12  13  78   1 307]]

============================================================
🎯 우울 신호 집중 분석 ([3, 4, 5])
============================================================

============================================================
🎯 우울 신호 집중 분석 (불안 + 상처 + 슬픔)
============================================================
✅ Precision: 0.9459
✅ Recall: 0.9016
✅ F1-Score: 0.9232

⚠️ False Negative: 63건 / 640건
✅ True Positive: 577건 / 640건
```

```
results.update(result)
```

```
total_result.append(results)
```

```
total_result
```

```
[{'task': 'six_label_all_text_phq9_multimodal',
  'text': 'all',
  'label': 'six',
  'model': 'multimodal',
  'accuracy': 0.6465816917728853,
  'f1_macro': 0.473413940088802,
  'f1_weighted': 0.6442434842235258,
  'confusion_matrix': array([[158,   2,   4,   1,   0,   3],
         [ 14,   6,   2,   2,   0,   3],
         [  4,   4,  15,   3,   0,   2],
         [ 12,   1,   2, 162,   0,  21],
         [  1,   1,   1,   1,   1,   2],
         [ 42,  15,  36, 124,   2, 216]]),
  'depression_precision': 0.9742173112338858,
  'depression_recall': 0.8265625,
  'depression_f1': 0.8943364327979713,
  'false_negatives': np.int64(111),
  'true_positives': np.int64(529),
  'total_depression': np.int64(640)},
 {'task': 'six_label_all_text_phq9',
  'text': 'all',
  'label': 'six',
  'model': 'finetuning',
  'accuracy': 0.7114716106604867,
  'f1_macro': 0.5136014461405877,
  'f1_weighted': 0.7114714905480 1,
  'confusion_matrix': array([[145,   6,   1,   3,   0,  13],
         [ 12,   8,   1,   0,   0,   6],
         [  5,   1,  11,   0,   1,  10],
         [  9,   1,   2, 142,   1,  43],
         [  1,   1,   0,   1,   1,   3],
         [ 24,  12,  13,  78,   1, 307]]),
  'depression_precision': 0.9459016393442623,
  'depression_recall': 0.9015625,
  'depression_f1': 0.9232,
  'false_negatives': np.int64(63),
  'true_positives': np.int64(577),
  'total_depression': np.int64(640)}]
```

```
import gc
del finetuning_model
del FineTuning_optimizer
torch.cuda.empty_cache()
gc.collect()
```

```
4763
```

비교 및 시각화

```
total_result=pd.DataFrame(total_result)
```

```
total_result.to_csv('total_result_phase3')
```

```
total_result
```

	task	text	label	model	accuracy	f1_macro	f1_weighted	confusion_matrix	depression_precision	depression_recall	depression_f1	false_negatives	true_positives	total_depression
0	six_label_all_text_phq9_multimodal	all	six	multimodal	0.646582	0.473414	0.644243	[[158, 2, 4, 1, 0, 3], [14, 6, 2, 2, 0, 3], [4...	0.974217	0.826562	0.894336	111	529	640
1	six_label_all_text_phq9	all	six	finetuning	0.711472	0.513601	0.711471	[[145, 6, 1, 3, 0, 13], [12, 8, 1, 0, 0, 6], [...	0.945902	0.901563	0.923200	63	577	640

다음 단계: total_result 변수로 코드 생성 | New interactive sheet

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
figure, axs = plt.subplots(3,1,figsize=(10,20))

metrics=['accuracy', 'f1_macro', 'depression_precision', 'depression_recall','depression_f1']

for i,v in enumerate(metrics):
    #axs[i].bar(total_result['task'], total_result[v])
    sns.barplot(data=total_result, x='task', y=v, hue='model', ax= axs[i])
    axs[i].set_title(v, fontsize=18)
    axs[i].set_ylabel('Score')
    axs[i].grid(axis='y', alpha=0.3)
    axs[i].set_ylim(bottom=total_result[v].min() * 0.99)

plt.tight_layout(h_pad=5.0)

plt.show()
```