

▼ 절취선

```
1 # 발소리, 말소리, 가구끄는 소리 분류 AI 모델
2 # Google Colab 최적화 버전
3
4 # =====
5 # 1. 필수 라이브러리 설치 및 임포트
6 # =====
7
8 import os
9 import gc
10 import numpy as np
11 import pandas as pd
12 import matplotlib.pyplot as plt
13 import seaborn as sns
14 import plotly.graph_objects as go
15 import plotly.express as px
16 from plotly.subplots import make_subplots
17
18 import librosa
19 import librosa.display
20 import soundfile as sf
21 import scipy.signal
22 from scipy import stats
23
24 import torch
25 import torch.nn as nn
26 import torch.nn.functional as F
27 from torch.utils.data import Dataset, DataLoader
28 import torchaudio
29
30 from sklearn.model_selection import train_test_split
31 from sklearn.preprocessing import LabelEncoder
32 from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
33 from sklearn.metrics import precision_recall_fscore_support
34
35 import warnings
36 warnings.filterwarnings('ignore')
37
38 from google.colab import files, drive
39 from IPython.display import Audio, display, HTML
40 from tqdm.auto import tqdm
41 import multiprocessing as mp
42
43 print("🎵 발소리-말소리-가구끄는소리 분류 AI 모델이 준비되었습니다!")
44 print("=" * 70)
45
```

🔗 🎵 발소리-말소리-가구끄는소리 분류 AI 모델이 준비되었습니다!

```
1 # =====
2 # 2. Colab 환경 최적화
3 # =====
4
5 def setup_colab_environment():
6     """Colab 환경 최적화"""
7     try:
8         mp.set_start_method('spawn', force=True)
9     except:
10         pass
11
12     os.environ['TOKENIZERS_PARALLELISM'] = 'false'
13     os.environ['OMP_NUM_THREADS'] = '1'
14
15     # 한글 폰트 설정
16     setup_korean_font()
17
18     # Google Drive 마운트
19     try:
20         drive.mount('/content/drive')
21         print("✅ Google Drive 마운트 완료")
22     except:
23         print("⚠️ Google Drive 마운트 실패 또는 이미 마운트됨")
24
25     print("✅ Colab 환경 최적화 완료")
26
27 def setup_korean_font():
28     """한글 폰트 설정"""
29     try:
30         # 나눔고딕 폰트 설치
31         !apt-get update -qq
32         !apt-get install -qq fonts-nanum
33
34         # matplotlib 폰트 설정
35         import matplotlib.font_manager as fm
36         import matplotlib.pyplot as plt
37
38         # 폰트 캐시 삭제
39         !rm -rf ~/.cache/matplotlib
```

```
11 # /content/matplotlib
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41 # 나눔고딕 폰트 경로
42 font_path = '/usr/share/fonts/truetype/nanum/NanumGothic.ttf'
43
44 if os.path.exists(font_path):
45     # 폰트 등록
46     fm.fontManager.addfont(font_path)
47
48     # matplotlib 기본 폰트 설정
49     plt.rcParams['font.family'] = 'NanumGothic'
50     plt.rcParams['axes.unicode_minus'] = False # 마이너스 기호 깨짐 방지
51
52     print("✅ 한글 폰트 설정 완료")
53 else:
54     # 대체 방법: 구글 폰트 사용
55     !wget -O NanumGothic.ttf "https://github.com/google/fonts/raw/main/ofl/nanumgothic/NanumGothic-Regular.ttf"
56
57     # 폰트 등록
58     fm.fontManager.addfont('./NanumGothic.ttf')
59     plt.rcParams['font.family'] = 'NanumGothic'
60     plt.rcParams['axes.unicode_minus'] = False
61
62     print("✅ 한글 폰트 설정 완료 (대체 폰트)")
63
64 except Exception as e:
65     print(f"⚠️ 한글 폰트 설정 실패: {e}")
66     print("영어로 표시됩니다.")
67
68 # 영어 레이블로 대체
69 plt.rcParams['font.family'] = 'DejaVu Sans'
70
71 def test_korean_font():
72     """한글 폰트 테스트"""
73     print("📄 한글 폰트 테스트 중...")
74
75     plt.figure(figsize=(10, 6))
76
77     # 테스트 데이터
78     classes = ['발소리', '말소리', '가구끄는소리']
79     values = [85.2, 92.1, 78.9]
80     colors = ['#ff9999', '#66b3ff', '#99ff99']
81
82     # 바 차트 생성
83     bars = plt.bar(classes, values, color=colors, alpha=0.8)
84
85     # 제목 및 레이블
86     plt.title('한글 폰트 테스트 - 클래스별 정확도', fontsize=16, fontweight='bold')
87     plt.xlabel('음성 클래스', fontsize=12)
88     plt.ylabel('정확도 (%)', fontsize=12)
89
90     # 값 표시
91     for bar, value in zip(bars, values):
92         plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 1,
93                  f'{value:.1f}%', ha='center', va='bottom', fontsize=12, fontweight='bold')
94
95     plt.ylim(0, 100)
96     plt.grid(True, axis='y', alpha=0.3)
97     plt.tight_layout()
98     plt.show()
99
100 # 폰트 상태 확인
101 current_font = plt.rcParams['font.family']
102 print(f"현재 폰트: {current_font}")
103
104 if 'NanumGothic' in current_font or 'Nanum' in str(current_font):
105     print("🎉 한글 폰트가 제대로 설정되었습니다!")
106 else:
107     print("⚠️ 한글 폰트 설정에 문제가 있을 수 있습니다.")
108     print("위 그래프에서 한글이 깨져 보인다면 런타임을 재시작해보세요.")
109
110 def memory_cleanup():
111     """메모리 정리"""
112     gc.collect()
113     if torch.cuda.is_available():
114         torch.cuda.empty_cache()
115     print("🏹 메모리 정리 완료")
116
117 # =====
118 # 3. 3클래스 오디오 데이터셋 (발소리, 말소리, 가구끄는소리)
119 # =====
120
121 class ThreeClassAudioDataset(Dataset):
122     def __init__(self, audio_paths, labels, target_sr=16000, max_duration=5.0, augment=False):
123         """
124         3클래스 오디오 분류 데이터셋
125
126         Args:
127             audio_paths: 오디오 파일 경로 리스트
128             labels: 레이블 리스트 ['footstep', 'speech', 'furniture']
129             target_sr: 목표 샘플링 레이트 (16kHz)
130             max_duration: 최대 길이 (5초)
131             augment: 데이터 증강 여부
132         """
```

```

133     self.audio_paths = audio_paths
134     self.target_sr = target_sr
135     self.max_duration = max_duration
136     self.max_length = int(target_sr * max_duration)
137     self.augment = augment
138
139     # 레이블 매핑
140     self.class_names = ['footstep', 'speech', 'furniture']
141     self.label_to_idx = {name: idx for idx, name in enumerate(self.class_names)}
142     self.idx_to_label = {idx: name for name, idx in self.label_to_idx.items()}
143
144     # 레이블 인코딩
145     self.labels = [self.label_to_idx[label] for label in labels]
146
147     print(f" 📊 데이터셋 정보:")
148     print(f"   - 총 샘플 수: {len(self.audio_paths)}")
149     print(f"   - 클래스: {self.class_names}")
150     for i, class_name in enumerate(self.class_names):
151         count = sum(1 for label in self.labels if label == i)
152         print(f"   - {class_name}: {count}개")
153
154     def __len__(self):
155         return len(self.audio_paths)
156
157     def __getitem__(self, idx):
158         try:
159             # 오디오 로드
160             audio, sr = librosa.load(
161                 self.audio_paths[idx],
162                 sr=self.target_sr,
163                 duration=self.max_duration
164             )
165
166             # 길이 정규화
167             audio = self._normalize_length(audio)
168
169             # 데이터 증강
170             if self.augment:
171                 audio = self._augment_audio(audio)
172
173             # 특징 추출 (Mel-spectrogram + MFCC)
174             features = self._extract_features(audio)
175
176             return {
177                 'features': torch.FloatTensor(features),
178                 'label': torch.LongTensor([self.labels[idx]]),
179                 'path': self.audio_paths[idx]
180             }
181
182         except Exception as e:
183             print(f"오디오 로딩 오류 {self.audio_paths[idx]}: {e}")
184             # 빈 특징 반환
185             features = np.zeros((128, 157)) # 기본 특징 크기
186             return {
187                 'features': torch.FloatTensor(features),
188                 'label': torch.LongTensor([0]),
189                 'path': self.audio_paths[idx]
190             }
191
192     def _normalize_length(self, audio):
193         """오디오 길이 정규화"""
194         if len(audio) > self.max_length:
195             # 랜덤 크롭
196             start = np.random.randint(0, len(audio) - self.max_length + 1)
197             audio = audio[start:start + self.max_length]
198         elif len(audio) < self.max_length:
199             # 제로 패딩
200             audio = np.pad(audio, (0, self.max_length - len(audio)))
201         return audio
202
203     def _augment_audio(self, audio):
204         """오디오 데이터 증강"""
205         # 시간 이동
206         if np.random.random() > 0.5:
207             shift = np.random.randint(-len(audio)//8, len(audio)//8)
208             audio = np.roll(audio, shift)
209
210         # 볼륨 조절
211         if np.random.random() > 0.5:
212             volume_factor = np.random.uniform(0.7, 1.3)
213             audio = audio * volume_factor
214
215         # 가우시안 노이즈 추가
216         if np.random.random() > 0.7:
217             noise = np.random.normal(0, 0.005, len(audio))
218             audio = audio + noise
219
220         # 피치 시프트 (가끔)
221         if np.random.random() > 0.8:
222             pitch_shift = np.random.randint(-2, 3)
223             if pitch_shift != 0:
224                 audio = librosa.effects.pitch_shift(audio, sr=self.target_sr, n_steps=pitch_shift)
225
226

```

```

226         return np.clip(audio, -1.0, 1.0)
227
228     def _extract_features(self, audio):
229         """특징 추출: Mel-spectrogram + MFCC"""
230         # Mel-spectrogram
231         mel_spec = librosa.feature.melspectrogram(
232             y=audio,
233             sr=self.target_sr,
234             n_mels=64,
235             fmax=8000,
236             hop_length=512,
237             n_fft=2048
238         )
239         mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)
240
241         # MFCC
242         mfcc = librosa.feature.mfcc(
243             y=audio,
244             sr=self.target_sr,
245             n_mfcc=64,
246             hop_length=512,
247             n_fft=2048
248         )
249
250         # 특징 결합
251         features = np.vstack([mel_spec_db, mfcc]) # (128, time_frames)
252
253         # 고정 크기로 조정
254         target_frames = 157 # 5초 * 16000 / 512 = 157
255         if features.shape[1] != target_frames:
256             features = self._resize_features(features, target_frames)
257
258         return features
259
260     def _resize_features(self, features, target_frames):
261         """특징 크기 조정"""
262         from scipy.ndimage import zoom
263         zoom_factor = target_frames / features.shape[1]
264         return zoom(features, (1, zoom_factor))
265
266 # =====
267 # 4. 3클래스 분류 CNN 모델
268 # =====
269
270 class ThreeClassAudioCNN(nn.Module):
271     def __init__(self, num_classes=3, dropout_rate=0.3):
272         super(ThreeClassAudioCNN, self).__init__()
273
274         # Convolutional layers
275         self.conv_layers = nn.Sequential(
276             # Block 1
277             nn.Conv2d(1, 32, kernel_size=(3, 3), padding=1),
278             nn.BatchNorm2d(32),
279             nn.ReLU(),
280             nn.MaxPool2d((2, 2)),
281             nn.Dropout2d(0.1),
282
283             # Block 2
284             nn.Conv2d(32, 64, kernel_size=(3, 3), padding=1),
285             nn.BatchNorm2d(64),
286             nn.ReLU(),
287             nn.MaxPool2d((2, 2)),
288             nn.Dropout2d(0.1),
289
290             # Block 3
291             nn.Conv2d(64, 128, kernel_size=(3, 3), padding=1),
292             nn.BatchNorm2d(128),
293             nn.ReLU(),
294             nn.MaxPool2d((2, 2)),
295             nn.Dropout2d(0.2),
296
297             # Block 4
298             nn.Conv2d(128, 256, kernel_size=(3, 3), padding=1),
299             nn.BatchNorm2d(256),
300             nn.ReLU(),
301             nn.MaxPool2d((2, 2)),
302             nn.Dropout2d(0.2),
303         )
304
305         # Global Average Pooling
306         self.global_pool = nn.AdaptiveAvgPool2d((1, 1))
307
308         # Classifier
309         self.classifier = nn.Sequential(
310             nn.Dropout(dropout_rate),
311             nn.Linear(256, 128),
312             nn.ReLU(),
313             nn.Dropout(dropout_rate * 0.5),
314             nn.Linear(128, num_classes)
315         )
316
317     def forward(self, x):
318         # ...

```

```

318 # Input: (batch_size, features, time)
319 if len(x.shape) == 3:
320     x = x.squeeze(1) # Add channel dimension
321
322 x = self.conv_layers(x)
323 x = self.global_pool(x)
324 x = x.view(x.size(0), -1)
325 x = self.classifier(x)
326 return x
327
328 # =====
329 # 5. 학습 매니지
330 # =====
331
332 class ThreeClassTrainer:
333     def __init__(self, model, train_loader, val_loader, device='auto', lr=0.001):
334         self.device = torch.device('cuda' if torch.cuda.is_available() and device=='auto' else 'cpu')
335         self.model = model.to(self.device)
336         self.train_loader = train_loader
337         self.val_loader = val_loader
338
339         # 최적화 설정
340         self.optimizer = torch.optim.AdamW(
341             model.parameters(),
342             lr=lr,
343             weight_decay=0.01
344         )
345         self.scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
346             self.optimizer,
347             mode='max', # 정확도 기준
348             patience=5,
349             factor=0.5,
350             verbose=True
351         )
352         self.criterion = nn.CrossEntropyLoss()
353
354         # 기록
355         self.train_losses = []
356         self.val_losses = []
357         self.train_accs = []
358         self.val_accs = []
359         self.best_val_acc = 0.0
360         self.best_model_state = None
361
362         print(f"🎯 학습 설정:")
363         print(f"   - 장치: {self.device}")
364         print(f"   - 학습률: {lr}")
365         print(f"   - 클래스: 3개 (발소리, 말소리, 가구끄는소리)")
366
367     def train_epoch(self):
368         """한 에포크 학습"""
369         self.model.train()
370         running_loss = 0.0
371         correct = 0
372         total = 0
373
374         pbar = tqdm(self.train_loader, desc="학습")
375         for batch in pbar:
376             inputs = batch['features'].to(self.device)
377             labels = batch['label'].squeeze().to(self.device)
378
379             self.optimizer.zero_grad()
380             outputs = self.model(inputs)
381             loss = self.criterion(outputs, labels)
382             loss.backward()
383             self.optimizer.step()
384
385             running_loss += loss.item()
386             _, predicted = outputs.max(1)
387             total += labels.size(0)
388             correct += predicted.eq(labels).sum().item()
389
390             # 진행률 업데이트
391             pbar.set_postfix({
392                 'Loss': f'{loss.item():.4f}',
393                 'Acc': f'{100.*correct/total:.1f}%'
394             })
395
396         epoch_loss = running_loss / len(self.train_loader)
397         epoch_acc = 100. * correct / total
398
399         return epoch_loss, epoch_acc
400
401     def validate_epoch(self):
402         """검증"""
403         self.model.eval()
404         running_loss = 0.0
405         correct = 0
406         total = 0
407         all_preds = []
408         all_labels = []
409
410         with torch.no_grad():
411             for batch in self.val_loader:
412                 inputs = batch['features'].to(self.device)
413                 labels = batch['label'].squeeze().to(self.device)
414
415                 self.model.eval()
416                 outputs = self.model(inputs)
417                 loss = self.criterion(outputs, labels)
418
419                 running_loss += loss.item()
420                 _, predicted = outputs.max(1)
421                 total += labels.size(0)
422                 correct += predicted.eq(labels).sum().item()
423
424                 # 혼동행렬용 데이터 수집
425                 all_preds.extend(predicted.cpu().numpy())
426                 all_labels.extend(labels.cpu().numpy())
427
428             pbar.set_postfix({
429                 'Loss': f'{loss.item():.4f}',
430                 'Acc': f'{100.*correct/total:.1f}%'
431             })
432
433         epoch_loss = running_loss / len(self.val_loader)
434         epoch_acc = 100. * correct / total
435
436         return epoch_loss, epoch_acc, all_preds, all_labels
437
438     def train(self, num_epochs=30, save_path='best_three_class_model.pth'):
439         """전체 학습"""
440         print(f"🚀 3클래스 분류 학습 시작!")
441         print(f"=====")
442
443         for epoch in range(num_epochs):
444             print(f"🔄 Epoch {epoch+1}/{num_epochs}")
445
446             # 학습
447             train_loss, train_acc = self.train_epoch()
448             self.train_losses.append(train_loss)
449             self.train_accs.append(train_acc)
450
451             # 검증
452             val_loss, val_acc, val_preds, val_labels = self.validate_epoch()
453             self.val_losses.append(val_loss)
454             self.val_accs.append(val_acc)
455
456             # 스케줄러 업데이트
457             self.scheduler.step(val_acc)
458
459             print(f"   학습 - Loss: {train_loss:.4f}, Acc: {train_acc:.2f}%")
460             print(f"   검증 - Loss: {val_loss:.4f}, Acc: {val_acc:.2f}%")
461             print(f"   학습률: {self.optimizer.param_groups[0]['lr']:.6f}")
462
463             # 최고 모델 저장
464             if val_acc > self.best_val_acc:
465                 self.best_val_acc = val_acc
466                 self.best_model_state = self.model.state_dict().copy()
467
468             torch.save({
469                 'epoch': epoch,
470                 'model_state_dict': self.model.state_dict(),
471                 'optimizer_state_dict': self.optimizer.state_dict(),
472                 'val_acc': val_acc,
473                 'val_loss': val_loss,
474                 'class_names': ['footstep', 'speech', 'furniture']
475             }, save_path)
476
477             print(f"✅ 최고 모델 저장! (검증 정확도: {val_acc:.2f}%)")
478
479             # 클래스별 정확도 출력
480             self.print_class_accuracy(val_labels, val_preds)
481
482             # 조기 종료
483             if self.optimizer.param_groups[0]['lr'] < 1e-6:
484                 print(f"🛑 학습률이 너무 낮아 학습을 종료합니다.")
485                 break
486
487             # 메모리 정리
488             if epoch % 5 == 0:
489                 memory_cleanup()
490
491         print(f"🏁 학습 완료!")
492         print(f"   최고 검증 정확도: {self.best_val_acc:.2f}%")
493
494         # 최고 모델 로드
495         if self.best_model_state:
496             self.model.load_state_dict(self.best_model_state)
497
498         return self.model
499
500     def print_class_accuracy(self, true_labels, pred_labels):
501         """클래스별 정확도 출력"""
502         class_names = ['발소리', '말소리', '가구끄는소리']
503         for i in range(len(class_names)):

```

```

538 pbar = tqdm(self.val_loader, desc="검증")
539 for batch in pbar:
540     inputs = batch['features'].to(self.device)
541     labels = batch['label'].squeeze().to(self.device)
542
543     outputs = self.model(inputs)
544     loss = self.criterion(outputs, labels)
545
546     running_loss += loss.item()
547     _, predicted = outputs.max(1)
548     total += labels.size(0)
549     correct += predicted.eq(labels).sum().item()
550
551     # 혼동행렬용 데이터 수집
552     all_preds.extend(predicted.cpu().numpy())
553     all_labels.extend(labels.cpu().numpy())
554
555     pbar.set_postfix({
556         'Loss': f'{loss.item():.4f}',
557         'Acc': f'{100.*correct/total:.1f}%'
558     })
559
560 epoch_loss = running_loss / len(self.val_loader)
561 epoch_acc = 100. * correct / total
562
563 return epoch_loss, epoch_acc, all_preds, all_labels
564
565 def train(self, num_epochs=30, save_path='best_three_class_model.pth'):
566     """전체 학습"""
567     print(f"🚀 3클래스 분류 학습 시작!")
568     print(f"=====")
569
570     for epoch in range(num_epochs):
571         print(f"🔄 Epoch {epoch+1}/{num_epochs}")
572
573         # 학습
574         train_loss, train_acc = self.train_epoch()
575         self.train_losses.append(train_loss)
576         self.train_accs.append(train_acc)
577
578         # 검증
579         val_loss, val_acc, val_preds, val_labels = self.validate_epoch()
580         self.val_losses.append(val_loss)
581         self.val_accs.append(val_acc)
582
583         # 스케줄러 업데이트
584         self.scheduler.step(val_acc)
585
586         print(f"   학습 - Loss: {train_loss:.4f}, Acc: {train_acc:.2f}%")
587         print(f"   검증 - Loss: {val_loss:.4f}, Acc: {val_acc:.2f}%")
588         print(f"   학습률: {self.optimizer.param_groups[0]['lr']:.6f}")
589
590         # 최고 모델 저장
591         if val_acc > self.best_val_acc:
592             self.best_val_acc = val_acc
593             self.best_model_state = self.model.state_dict().copy()
594
595         torch.save({
596             'epoch': epoch,
597             'model_state_dict': self.model.state_dict(),
598             'optimizer_state_dict': self.optimizer.state_dict(),
599             'val_acc': val_acc,
600             'val_loss': val_loss,
601             'class_names': ['footstep', 'speech', 'furniture']
602         }, save_path)
603
604         print(f"✅ 최고 모델 저장! (검증 정확도: {val_acc:.2f}%)")
605
606         # 클래스별 정확도 출력
607         self.print_class_accuracy(val_labels, val_preds)
608
609         # 조기 종료
610         if self.optimizer.param_groups[0]['lr'] < 1e-6:
611             print(f"🛑 학습률이 너무 낮아 학습을 종료합니다.")
612             break
613
614         # 메모리 정리
615         if epoch % 5 == 0:
616             memory_cleanup()
617
618     print(f"🏁 학습 완료!")
619     print(f"   최고 검증 정확도: {self.best_val_acc:.2f}%")
620
621     # 최고 모델 로드
622     if self.best_model_state:
623         self.model.load_state_dict(self.best_model_state)
624
625     return self.model
626
627     def print_class_accuracy(self, true_labels, pred_labels):
628         """클래스별 정확도 출력"""
629         class_names = ['발소리', '말소리', '가구끄는소리']
630         for i in range(len(class_names)):

```

```

504 for i, class_name in enumerate(class_names):
505     class_mask = np.array(true_labels) == i
506     if class_mask.sum() > 0:
507         class_acc = (np.array(pred_labels)[class_mask] == i).sum() / class_mask.sum()
508         print(f"    - {class_name}: {class_acc*100:.1f}%")
509
510 def plot_training_history(self):
511     """학습 히스토리 시각화 - 한글 지원"""
512     # 한글 폰트 설정 확인
513     try:
514         plt.rcParams['font.family'] = 'NanumGothic'
515         plt.rcParams['axes.unicode_minus'] = False
516     except:
517         pass
518
519     fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))
520
521     # 손실 그래프
522     epochs = range(1, len(self.train_losses) + 1)
523     ax1.plot(epochs, self.train_losses, 'b-', label='학습 손실', linewidth=2, alpha=0.8)
524     ax1.plot(epochs, self.val_losses, 'r-', label='검증 손실', linewidth=2, alpha=0.8)
525     ax1.set_title('학습/검증 손실', fontsize=14, fontweight='bold', pad=15)
526     ax1.set_xlabel('에POCH (Epoch)', fontsize=12)
527     ax1.set_ylabel('손실 (Loss)', fontsize=12)
528     ax1.legend(fontsize=11)
529     ax1.grid(True, alpha=0.3)
530     ax1.set_xlim(1, len(self.train_losses))
531
532     # 정확도 그래프
533     ax2.plot(epochs, self.train_accs, 'b-', label='학습 정확도', linewidth=2, alpha=0.8)
534     ax2.plot(epochs, self.val_accs, 'r-', label='검증 정확도', linewidth=2, alpha=0.8)
535     ax2.set_title('학습/검증 정확도', fontsize=14, fontweight='bold', pad=15)
536     ax2.set_xlabel('에POCH (Epoch)', fontsize=12)
537     ax2.set_ylabel('정확도 (%)', fontsize=12)
538     ax2.legend(fontsize=11)
539     ax2.grid(True, alpha=0.3)
540     ax2.set_ylim(0, 100)
541     ax2.set_xlim(1, len(self.train_accs))
542
543     # 최고 성능 지점 표시
544     best_epoch = np.argmax(self.val_accs) + 1
545     best_acc = max(self.val_accs)
546     ax2.plot(best_epoch, best_acc, 'ro', markersize=10, markerfacecolor='red',
547             markeredgecolor='darkred', markeredgewidth=2)
548     ax2.annotate(f'최고: {best_acc:.1f}%\n(에POCH {best_epoch})',
549                xy=(best_epoch, best_acc), xytext=(10, 10),
550                textcoords='offset points', fontsize=10,
551                bbox=dict(boxstyle='round,pad=0.3', facecolor='yellow', alpha=0.7),
552                arrowprops=dict(arrowstyle='->', connectionstyle='arc3,rad=0'))
553
554     plt.tight_layout()
555     pit.show()
556
557     # 상세 결과 출력
558     print("\n" + "="*60)
559     print("📊 학습 완료 - 최종 결과 요약")
560     print("="*60)
561     print(f"🔴 최고 검증 정확도: {max(self.val_accs):.2f}% (에POCH {np.argmax(self.val_accs)+1})")
562     print(f"🟢 최고 학습 정확도: {self.train_accs[-1]:.2f}%")
563     print(f"🟡 최종 검증 손실: {self.val_losses[-1]:.4f}")
564     print(f"🔵 총 학습 에POCH: {len(self.train_accs)}개")
565
566     # 성능 안정성 분석
567     last_5_accs = self.val_accs[-5:] if len(self.val_accs) >= 5 else self.val_accs
568     stability = np.std(last_5_accs)
569     print(f"📈 최근 5에POCH 정확도 안정성: {stability:.2f}% (낮을수록 안정)")
570
571     if stability < 2.0:
572         print("✅ 매우 안정적인 학습!")
573     elif stability < 5.0:
574         print("🟡 안정적인 학습")
575     else:
576         print("⚠️ 불안정한 학습 - 더 많은 에POCH나 조기종료 필요")
577
578     print("="*60)
579
580 # =====
581 # 6. 데이터 처리 및 학습 실행 함수
582 # =====
583
584 def scan_three_class_data(base_path):
585     """3클래스 데이터 스캔 - 강화버전"""
586     audio_files = []
587     labels = []
588
589     # 지원되는 오디오 확장자 (더 많은 형식 추가)
590     extensions = ['.wav', '.mp3', '.flac', '.m4a', '.ogg', '.aac', '.wma', '.aiff', '.au']
591
592     print(f"📁 {base_path}에서 오디오 파일 스캔 중...")
593     print(f"🔍 지원 확장자: {', '.join(extensions)}")
594
595     # 각 클래스별 폴더에서 파일 수집 (더 많은 키워드 추가)
596     class_folders = {

```

```

597 'footstep': ['footstep', 'footsteps', 'foot', 'walk', 'walking', 'step',
598             '발소리', '걸음소리', '발걸음', '보행', 'steps', 'footfall'],
599 'speech': ['speech', 'voice', 'talk', 'talking', 'speaking', '말소리', 'speak',
600            '음성', '대화', '목소리', 'vocal', 'utterance', 'conversation'],
601 'furniture': ['furniture', 'chair', 'table', 'drag', 'move', '가구', 'scrape',
602              '끄는소리', '이동', '의자', '테이블', 'moving', 'sliding', 'dragging']
603 }
604
605 # 모든 하위 디렉토리 탐색
606 total_files_found = 0
607 processed_dirs = []
608
609 for root, dirs, files in os.walk(base_path):
610     folder_name = os.path.basename(root).lower()
611     relative_path = os.path.relpath(root, base_path)
612
613     # 오디오 파일이 있는지 확인
614     audio_files_in_dir = [f for f in files if any(f.lower().endswith(ext) for ext in extensions)]
615
616     if audio_files_in_dir:
617         print(f"    📁 {relative_path} - {len(audio_files_in_dir)}개 오디오 파일 발견")
618         total_files_found += len(audio_files_in_dir)
619
620     # 폴더명으로 클래스 판단
621     detected_class = None
622     for class_name, keywords in class_folders.items():
623         if any(keyword in folder_name for keyword in keywords):
624             detected_class = class_name
625             break
626
627     # 클래스가 자동 감지되지 않으면 사용자에게 물어보기
628     if detected_class is None:
629         print(f"    ? '{folder_name}' 폴더의 클래스를 판단할 수 없습니다.")
630         print(f"    다음 중 하나를 선택하세요:")
631         print(f"    1: footstep (발소리)")
632         print(f"    2: speech (말소리)")
633         print(f"    3: furniture (가구끄는소리)")
634         print(f"    0: skip (건너뛰기)")
635
636         try:
637             choice = input(f"    선택 (1/2/3/0): ").strip()
638             class_mapping = {'1': 'footstep', '2': 'speech', '3': 'furniture'}
639             if choice in class_mapping:
640                 detected_class = class_mapping[choice]
641                 print(f"    ✅ '{folder_name}' -> {detected_class}")
642             else:
643                 print(f"    ❌ '{folder_name}' - 폴더 건너뛸")
644                 continue
645         except:
646             print(f"    ⌨ 입력 오류로 '{folder_name}' 폴더 건너뛸")
647             continue
648     else:
649         print(f"    ✅ {relative_path} -> {detected_class}")
650
651     # 파일 추가
652     for file in audio_files_in_dir:
653         file_path = os.path.join(root, file)
654         audio_files.append(file_path)
655         labels.append(detected_class)
656
657     processed_dirs.append((relative_path, detected_class, len(audio_files_in_dir)))
658
659 # 상세 결과 출력
660 print(f"\n📊 데이터 스캔 완료:")
661 print(f"    - 총 발견된 오디오 파일: {total_files_found}개")
662 print(f"    - 실제 사용될 파일: {len(audio_files)}개")
663 print(f"    - 처리된 디렉토리: {len(processed_dirs)}개")
664
665 print(f"\n📁 처리된 디렉토리 상세:")
666 for dir_path, class_name, count in processed_dirs:
667     print(f"    - {dir_path}: {class_name} ({count}개)")
668
669 print(f"\n📁 클래스별 파일 수:")
670 for class_name in ['footstep', 'speech', 'furniture']:
671     count = labels.count(class_name)
672     percentage = (count / len(labels) * 100) if len(labels) > 0 else 0
673     print(f"    - {class_name}: {count}개 ({percentage:.1f}%)")
674
675 if len(audio_files) == 0:
676     print("\n❌ 사용 가능한 오디오 파일이 없습니다!")
677     print("    다음을 확인해주세요:")
678     print("    1. 파일 확장자가 지원되는지 확인")
679     print("    2. 폴더 구조가 올바른지 확인")
680     print("    3. 파일이 실제로 오디오 파일인지 확인")
681     return [], []
682
683 if total_files_found > len(audio_files):
684     print(f"\n⚠️ 주의: {total_files_found - len(audio_files)}개 파일이 제외되었습니다.")
685     print("    - 폴더명이 클래스와 매치되지 않아 제외되었을 수 있습니다.")
686
687 return audio_files, labels
688
689 def detailed_data_analysis(base_path):

```

```

690 """데이터 상세 분석"""
691 print("🔍 데이터 구조 상세 분석 중...")
692
693 extensions = ['.wav', '.mp3', '.flac', '.m4a', '.ogg', '.aac', '.wma', '.aiff', '.au']
694
695 total_files = 0
696 total_size = 0
697 dir_info = []
698
699 for root, dirs, files in os.walk(base_path):
700     audio_files = []
701     dir_size = 0
702
703     for file in files:
704         file_path = os.path.join(root, file)
705         if any(file.lower().endswith(ext) for ext in extensions):
706             audio_files.append(file)
707             try:
708                 file_size = os.path.getsize(file_path)
709                 dir_size += file_size
710             except:
711                 pass
712
713     if audio_files:
714         relative_path = os.path.relpath(root, base_path)
715         dir_info.append({
716             'path': relative_path,
717             'files': len(audio_files),
718             'size_mb': dir_size / (1024*1024),
719             'sample_files': audio_files[:3] # 처음 3개 파일명
720         })
721
722     total_files += len(audio_files)
723     total_size += dir_size
724
725 print(f"📊 전체 통계:")
726 print(f"  - 총 오디오 파일: {total_files}개")
727 print(f"  - 총 크기: {total_size/(1024*1024):.1f} MB")
728 print(f"  - 오디오가 있는 폴더: {len(dir_info)}개")
729
730 print(f"📁 폴더별 상세 정보:")
731 for info in sorted(dir_info, key=lambda x: x['files'], reverse=True):
732     print(f"  📁 {info['path']}")
733     print(f"    - 파일 수: {info['files']}개")
734     print(f"    - 크기: {info['size_mb']:.1f} MB")
735     print(f"    - 샘플: {' '.join(info['sample_files'])}")
736     print()
737
738 return dir_info
739
740 def force_scan_all_audio_files(base_path):
741     """모든 오디오 파일 강제 스캔 (클래스 구분 없이)"""
742     print("🔍 모든 오디오 파일 강제 스캔 중...")
743
744     extensions = ['.wav', '.mp3', '.flac', '.m4a', '.ogg', '.aac', '.wma', '.aiff', '.au']
745     all_files = []
746
747     for root, dirs, files in os.walk(base_path):
748         for file in files:
749             if any(file.lower().endswith(ext) for ext in extensions):
750                 file_path = os.path.join(root, file)
751                 relative_path = os.path.relpath(file_path, base_path)
752                 all_files.append({
753                     'path': file_path,
754                     'relative': relative_path,
755                     'dir': os.path.dirname(relative_path),
756                     'filename': file
757                 })
758
759     print(f"📊 전체 스캔 결과: {len(all_files)}개 오디오 파일 발견")
760
761 # 디렉토리별 그룹화
762 from collections import defaultdict
763 dir_groups = defaultdict(list)
764
765 for file_info in all_files:
766     dir_name = file_info['dir'] if file_info['dir'] else 'root'
767     dir_groups[dir_name].append(file_info)
768
769 print(f"📁 디렉토리별 파일 수:")
770 for dir_name, file_list in sorted(dir_groups.items(), key=lambda x: len(x[1]), reverse=True):
771     print(f"  {dir_name}: {len(file_list)}개")
772     # 처음 3개 파일명 표시
773     for i, file_info in enumerate(file_list[:3]):
774         print(f"    - {file_info['filename']}")
775     if len(file_list) > 3:
776         print(f"    ... 그 외 {len(file_list)-3}개")
777     print()
778
779 return all_files, dir_groups
780
781 def run_three_class_training(data_path, num_epochs=30, batch_size=8, test_size=0.2):
782     """3클래스 분류 학습 실행"""

```

```

783
784 print("🔊 발소리-말소리-가구끄는소리 분류 학습 시작!")
785 print("=" * 70)
786
787 # 환경 설정
788 setup_colab_environment()
789 memory_cleanup()
790
791 # 1. 데이터 스캔
792 audio_files, labels = scan_three_class_data(data_path)
793
794 if len(audio_files) == 0:
795     print("❌ 오디오 파일을 찾을 수 없습니다!")
796     print("🔍 데이터 구조를 확인해주세요:")
797     print("  your_dataset/")
798     print("    ├── footstep/")
799     print("    ├── speech/")
800     print("    └── furniture/")
801     return None
802
803 # 2. 학습/검증 분할
804 train_files, val_files, train_labels, val_labels = train_test_split(
805     audio_files, labels, test_size=test_size, random_state=42, stratify=labels
806 )
807
808 print(f"📁 데이터 분할:")
809 print(f"  - 학습: {len(train_files)}개")
810 print(f"  - 검증: {len(val_files)}개")
811
812 # 3. 데이터셋 생성
813 train_dataset = ThreeClassAudioDataset(
814     train_files, train_labels, augment=True
815 )
816 val_dataset = ThreeClassAudioDataset(
817     val_files, val_labels, augment=False
818 )
819
820 # 4. 데이터 로더 생성
821 train_loader = DataLoader(
822     train_dataset, batch_size=batch_size, shuffle=True,
823     num_workers=0, pin_memory=False
824 )
825 val_loader = DataLoader(
826     val_dataset, batch_size=batch_size, shuffle=False,
827     num_workers=0, pin_memory=False
828 )
829
830 # 5. 모델 생성
831 model = ThreeClassAudioCNN(num_classes=3)
832 print(f"🔢 모델 파라미터 수: {sum(p.numel() for p in model.parameters())}")
833
834 # 6. 학습 실행
835 trainer = ThreeClassTrainer(model, train_loader, val_loader)
836 trained_model = trainer.train(num_epochs=num_epochs)
837
838 # 7. 결과 시각화
839 trainer.plot_training_history()
840
841 # 8. 혼동행렬 생성
842 plot_confusion_matrix(trained_model, val_loader)
843
844 return trained_model, trainer
845
846 def plot_confusion_matrix(model, val_loader):
847     """혼동행렬 시각화 - 한글 지원"""
848     model.eval()
849     device = next(model.parameters()).device
850
851     all_preds = []
852     all_labels = []
853
854     with torch.no_grad():
855         for batch in val_loader:
856             inputs = batch['features'].to(device)
857             labels = batch['label'].squeeze().to(device)
858
859             outputs = model(inputs)
860             _, predicted = outputs.max(1)
861
862             all_preds.extend(predicted.cpu().numpy())
863             all_labels.extend(labels.cpu().numpy())
864
865     # 혼동행렬 계산
866     cm = confusion_matrix(all_labels, all_preds)
867     class_names = ['발소리', '말소리', '가구끄는소리']
868
869     # 시각화
870     plt.figure(figsize=(10, 8))
871
872     # 한글 폰트 확인 및 설정
873     try:
874         plt.rcParams['font.family'] = 'NanumGothic'
875     except:

```

```

876     # 폰트 설정이 안된 경우 영어로 대체
877     class_names = ['Footstep', 'Speech', 'Furniture']
878
879     sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
880                 xticklabels=class_names, yticklabels=class_names,
881                 cbar_kws={'label': 'Frequency'})
882     plt.title('혼동 행렬 (Confusion Matrix)', fontsize=16, pad=20)
883     plt.xlabel('예측값 (Predicted)', fontsize=12)
884     plt.ylabel('실제값 (Actual)', fontsize=12)
885     plt.tight_layout()
886     plt.show()
887
888     # 분류 리포트
889     report = classification_report(all_labels, all_preds,
890                                   target_names=class_names, output_dict=True)
891
892     print("\n📊 상세 분류 결과:")
893     for i, class_name in enumerate(class_names):
894         precision = report[class_name]['precision']
895         recall = report[class_name]['recall']
896         f1 = report[class_name]['f1-score']
897         support = report[class_name]['support']
898
899         print(f"    {class_name}:")
900         print(f"        정밀도(Precision): {precision:.3f}")
901         print(f"        재현율(Recall): {recall:.3f}")
902         print(f"        F1-Score: {f1:.3f}")
903         print(f"        샘플 수: {support}")
904
905     print(f"\n 전체 정확도: {report['accuracy']:.3f}")
906     print(f" 매크로 평균 F1: {report['macro avg']['f1-score']:.3f}")
907
908     # 클래스별 정확도 바 차트
909     plt.figure(figsize=(10, 6))
910     class_accuracies = []
911     for i in range(len(class_names)):
912         if cm[i].sum() > 0:
913             acc = cm[i, i] / cm[i].sum()
914             class_accuracies.append(acc)
915         else:
916             class_accuracies.append(0)
917
918     colors = ['#ff7f7f', '#7f7fff', '#7fff7f']
919     bars = plt.bar(class_names, class_accuracies, color=colors, alpha=0.8)
920     plt.title('클래스별 정확도', fontsize=16, pad=20)
921     plt.xlabel('클래스', fontsize=12)
922     plt.ylabel('정확도', fontsize=12)
923     plt.ylim(0, 1.1)
924
925     # 각 막대 위에 정확도 값 표시
926     for bar, acc in zip(bars, class_accuracies):
927         plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.02,
928                 f'{acc:.2%}', ha='center', va='bottom', fontsize=12, fontweight='bold')
929
930     plt.grid(True, axis='y', alpha=0.3)
931     plt.tight_layout()
932     plt.show()
933
934 # =====
935 # 7. 실시간 예측 함수 (완성)
936 # =====
937
938 def predict_audio_file(model_path, audio_file_path):
939     """오디오 파일 예측 - 완성 버전"""
940     # 모델 로드
941     checkpoint = torch.load(model_path, map_location='cpu')
942     model = ThreeClassAudioCNN(num_classes=3)
943     model.load_state_dict(checkpoint['model_state_dict'])
944     model.eval()
945
946     class_names = ['발소리', '말소리', '가구끄는소리']
947     class_names_eng = ['footstep', 'speech', 'furniture']
948
949     try:
950         # 오디오 로드 및 전처리
951         audio, sr = librosa.load(audio_file_path, sr=16000, duration=5.0)
952
953         # 길이 정규화
954         max_length = 16000 * 5
955         if len(audio) < max_length:
956             audio = np.pad(audio, (0, max_length - len(audio)))
957         else:
958             audio = audio[:max_length]
959
960         # 특징 추출
961         mel_spec = librosa.feature.melspectrogram(
962             y=audio, sr=16000, n_mels=64, fmax=8000, hop_length=512, n_fft=2048
963         )
964         mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)
965
966         mfcc = librosa.feature.mfcc(
967             y=audio, sr=16000, n_mfcc=64, hop_length=512, n_fft=2048
968         )

```

```

969     # 특징 결합
970     features = np.vstack([mel_spec_db, mfcc]) # (128, time_frames)
971
972     # 크기 조정
973     target_frames = 157
974     if features.shape[1] != target_frames:
975         from scipy.ndimage import zoom
976         zoom_factor = target_frames / features.shape[1]
977         features = zoom(features, (1, zoom_factor))
978
979     # 예측
980     with torch.no_grad():
981         features_tensor = torch.FloatTensor(features).unsqueeze(0) # 배치 차원 추가
982         outputs = model(features_tensor)
983         probabilities = F.softmax(outputs, dim=1)
984         predicted_class = torch.argmax(outputs, dim=1).item()
985         confidence = probabilities[0, predicted_class].item()
986
987     # 결과 반환
988     result = {
989         'predicted_class': class_names[predicted_class],
990         'predicted_class_eng': class_names_eng[predicted_class],
991         'confidence': confidence,
992         'probabilities': {
993             class_names[i]: prob.item()
994             for i, prob in enumerate(probabilities[0])
995         }
996     }
997
998     # 시각화
999     visualize_prediction_result(audio, sr, features, result)
1000
1001     return result
1002
1003 except Exception as e:
1004     print(f"❌ 예측 오류: {str(e)}")
1005     return None
1006
1007 def visualize_prediction_result(audio, sr, features, result):
1008     """예측 결과 시각화"""
1009     fig = make_subplots(
1010         rows=2, cols=2,
1011         subplot_titles=('오디오 파형', '특징 맵 (Mel-spec + MFCC)', '예측 확률', '주파수 스펙트럼'),
1012         specs=[[{"secondary_y": False}, {"secondary_y": False}],
1013               [{"type": "bar"}, {"secondary_y": False}]]
1014
1015     # 1. 오디오 파형
1016     time = np.linspace(0, len(audio)/sr, len(audio))
1017     fig.add_trace(
1018         go.Scatter(x=time, y=audio, name='오디오 신호', line=dict(color='blue')),
1019         row=1, col=1
1020     )
1021
1022     # 2. 특징 맵
1023     fig.add_trace(
1024         go.Heatmap(z=features, colorscale='Viridis', name='특징 맵'),
1025         row=1, col=2
1026     )
1027
1028     # 3. 예측 확률
1029     classes = list(result['probabilities'].keys())
1030     probs = list(result['probabilities'].values())
1031     colors = ['red' if cls == result['predicted_class'] else 'lightblue' for cls in classes]
1032
1033     fig.add_trace(
1034         go.Bar(x=classes, y=probs, name='예측 확률',
1035               marker=dict(color=colors)),
1036         row=2, col=1
1037     )
1038
1039     # 4. 주파수 스펙트럼
1040     freqs = np.fft.fftfreq(len(audio), 1/sr)[:len(audio)//2]
1041     fft_vals = np.abs(np.fft.fft(audio))[:len(audio)//2]
1042
1043     fig.add_trace(
1044         go.Scatter(x=freqs, y=fft_vals, name='주파수 스펙트럼', line=dict(color='green')),
1045         row=2, col=2
1046     )
1047
1048     fig.update_layout(
1049         height=800,
1050         title_text=f"예측 결과: {result['predicted_class']} (신뢰도: {result['confidence']:.2f})"
1051     )
1052     fig.show()
1053
1054     # 결과 출력
1055     print(f"\n📊 예측 결과:")
1056     print(f"    - 예측 클래스: {result['predicted_class']}")
1057     print(f"    - 신뢰도: {result['confidence']:.3f}")
1058     print(f"\n📊 모든 클래스 확률:")
1059     for class_name, prob in result['probabilities'].items():

```

```

1061 for class_name, prob in result['probabilities'].items():
1062     print(f"   - {class_name}: {prob:.3f}")
1063
1064 #
1065 # 8. 샘플 데이터 생성기
1066 #
1067
1068 def generate_sample_audio_data(output_dir='sample_data', samples_per_class=20):
1069     """테스트용 샘플 오디오 데이터 생성"""
1070     print("  🎧 샘플 오디오 데이터 생성 중...")
1071
1072     os.makedirs(output_dir, exist_ok=True)
1073
1074     # 각 클래스별 폴더 생성
1075     classes = ['footstep', 'speech', 'furniture']
1076     for class_name in classes:
1077         os.makedirs(os.path.join(output_dir, class_name), exist_ok=True)
1078
1079     sr = 16000
1080     duration = 3.0
1081     t = np.linspace(0, duration, int(sr * duration))
1082
1083     for i in range(samples_per_class):
1084         # 발소리 시뮬레이션 (짧은 충격음들)
1085         footstep = np.zeros_like(t)
1086         for step in range(4): # 4번의 발걸음
1087             start_idx = int(step * len(t) / 4) + np.random.randint(-1000, 1000)
1088             if 0 <= start_idx < len(t) - 1000:
1089                 # 충격을 시뮬레이션
1090                 impact = np.exp(-np.arange(1000) * 0.01) * np.sin(2 * np.pi * np.random.uniform(80, 200) * np.arange(1000) / sr)
1091                 footstep[start_idx:start_idx+1000] += impact * np.random.uniform(0.3, 0.8)
1092
1093         # 노이즈 추가
1094         footstep += np.random.normal(0, 0.02, len(footstep))
1095         footstep = np.clip(footstep, -1.0, 1.0)
1096
1097         sf.write(os.path.join(output_dir, 'footstep', f'footstep_{i:03d}.wav'), footstep, sr)
1098
1099         # 말소리 시뮬레이션 (여러 주파수 성분)
1100         speech = np.zeros_like(t)
1101         # 기본 음성 주파수들 (100-300Hz)
1102         for freq in [120, 180, 240, 300]:
1103             amplitude = np.random.uniform(0.1, 0.3)
1104             speech += amplitude * np.sin(2 * np.pi * freq * t)
1105
1106         # 포먼트 시뮬레이션 (800-2000Hz)
1107         for freq in [800, 1200, 1600, 2000]:
1108             amplitude = np.random.uniform(0.05, 0.15)
1109             modulation = 1 + 0.5 * np.sin(2 * np.pi * np.random.uniform(5, 15) * t)
1110             speech += amplitude * np.sin(2 * np.pi * freq * t) * modulation
1111
1112         # 노이즈 및 변조
1113         speech *= (1 + 0.3 * np.sin(2 * np.pi * np.random.uniform(1, 5) * t))
1114         speech += np.random.normal(0, 0.02, len(speech))
1115         speech = np.clip(speech, -1.0, 1.0)
1116
1117         sf.write(os.path.join(output_dir, 'speech', f'speech_{i:03d}.wav'), speech, sr)
1118
1119         # 가구끄는소리 시뮬레이션 (마찰음)
1120         furniture = np.zeros_like(t)
1121
1122         # 마찰음 기본 주파수 (낮은 주파수 + 고주파 노이즈)
1123         base_freq = np.random.uniform(20, 80)
1124         furniture += 0.4 * np.sin(2 * np.pi * base_freq * t)
1125
1126         # 고주파 마찰음 (1-4kHz)
1127         high_freq_noise = np.random.normal(0, 0.1, len(t))
1128         butter_b, butter_a = scipy.signal.butter(4, [1000, 4000], btype='band', fs=sr)
1129         high_freq_filtered = scipy.signal.filtfilt(butter_b, butter_a, high_freq_noise)
1130         furniture += 0.3 * high_freq_filtered
1131
1132         # 불규칙한 진동 패턴
1133         irregular_pattern = np.random.uniform(0.5, 1.5, 100)
1134         furniture *= np.interp(t, np.linspace(0, duration, 100), irregular_pattern)
1135
1136         furniture = np.clip(furniture, -1.0, 1.0)
1137
1138         sf.write(os.path.join(output_dir, 'furniture', f'furniture_{i:03d}.wav'), furniture, sr)
1139
1140     print(f"  ✅ 샘플 데이터 생성 완료!")
1141     print(f"  - 경로: {output_dir}")
1142     print(f"  - 클래스별 {samples_per_class}개씩 총 {samples_per_class * 3}개 파일")
1143
1144     return output_dir
1145
1146 #
1147 # 9. 실시간 녹음 및 예측
1148 #
1149
1150 def record_and_predict(model_path, duration=5):
1151     """실시간 녹음 및 예측"""
1152     print(f"  🎧 {duration}초간 녹음을 시작합니다...")
1153     print("  🗣️ 마이크에 대고 소리를 내세요!")
1154     ....

```

```

1154
1155     try:
1156         # JavaScript를 사용한 녹음 (Colab 환경)
1157         from google.colab import output
1158         from base64 import b64decode
1159
1160         RECORD = ""
1161         const sleep = time => new Promise(resolve => setTimeout(resolve, time))
1162         const b2text = blob => new Promise(resolve => {
1163             const reader = new FileReader()
1164             reader.onloadend = e => resolve(e.srcElement.result)
1165             reader.readAsDataURL(blob)
1166         })
1167
1168         var record = time => new Promise(async resolve => {
1169             stream = await navigator.mediaDevices.getUserMedia({ audio: true })
1170             recorder = new MediaRecorder(stream)
1171             chunks = []
1172             recorder.ondataavailable = e => chunks.push(e.data)
1173             recorder.onstop = async ()=>{
1174                 blob = new Blob(chunks, { type: 'audio/wav' })
1175                 text = await b2text(blob)
1176                 resolve(text)
1177             }
1178             recorder.start()
1179             await sleep(time)
1180             recorder.stop()
1181         })
1182         ""
1183
1184         display(HTML(f`
1185         <script>
1186         {RECORD}
1187         </script>
1188         <button onclick="record({duration * 1000}).then(audio => {{
1189             google.colab.kernel.invokeFunction('save_audio', [audio], [{}])
1190             }}}"> 🎧 녹음 시작 ({duration}초)</button>
1191         `))
1192
1193         print("   위의 녹음 버튼을 클릭해주세요!")
1194
1195     except Exception as e:
1196         print(f"  ❌ 녹음 기능 오류: {e}")
1197         print("   대신 파일 업로드를 사용해주세요.")
1198
1199     def save_audio(audio_data):
1200         """녹음된 오디오 저장 및 예측"""
1201         try:
1202             # Base64 디코딩
1203             audio_data = audio_data.split(',')[-1]
1204             audio_bytes = b64decode(audio_data)
1205
1206             # 파일 저장
1207             with open('recorded_audio.wav', 'wb') as f:
1208                 f.write(audio_bytes)
1209
1210             print("  ✅ 녹음 완료! 예측 중...")
1211
1212             # 예측 실행 (모델이 있다면)
1213             if os.path.exists('best_three_class_model.pth'):
1214                 result = predict_audio_file('best_three_class_model.pth', 'recorded_audio.wav')
1215                 if result:
1216                     print_prediction_result(result)
1217             else:
1218                 print("  ⚠️ 학습된 모델이 없습니다. 먼저 학습을 실행해주세요.")
1219
1220     except Exception as e:
1221         print(f"  ❌ 오디오 저장 오류: {e}")
1222
1223     def print_prediction_result(result):
1224         """예측 결과 출력"""
1225         print("\n" + "="*50)
1226         print("  🕒 실시간 예측 결과")
1227         print("="*50)
1228         print(f"  🔍 예측 클래스: {result['predicted_class']}")
1229         print(f"  📊 신뢰도: {result['confidence']:.1%}")
1230         print("\n  📁 각 클래스별 확률:")
1231
1232         for class_name, prob in result['probabilities'].items():
1233             bar = "█" * int(prob * 20)
1234             print(f"   {class_name:12}: {prob:.3f} |{bar}")
1235
1236 #
1237 # 10. 업로드 파일 분석
1238 #
1239
1240 def analyze_uploaded_file():
1241     """파일 업로드 및 분석"""
1242     print("  📁 오디오 파일을 업로드해주세요...")
1243     uploaded = files.upload()
1244
1245     if uploaded:
1246         file_name = list(uploaded.keys())[0]
         ....

```



```
1247 print(f"✅ 파일 '{file_name}' 업로드 완료!")
1248
1249 # 모델이 있는지 확인
1250 if os.path.exists('best_three_class_model.pth'):
1251     result = predict_audio_file('best_three_class_model.pth', file_name)
1252     if result:
1253         print_prediction_result(result)
1254
1255     # 오디오 재생
1256     audio, sr = librosa.load(file_name, sr=16000)
1257     display(Audio(audio, rate=sr))
1258
1259     return result
1260 else:
1261     print("⚠️ 학습된 모델이 없습니다.")
1262     print("먼저 샘플 데이터로 학습하거나 실제 데이터로 학습해주세요.")
1263     return None
1264
1265 else:
1266     print("❌ 파일이 업로드되지 않았습니다.")
1267     return None
1268
1269 # 11. 데모 및 테스트 함수들
1270
1271 def debug_data_loading(data_path):
1272     """데이터 로딩 과정 디버깅"""
1273     print("🔍 데이터 로딩 과정 디버깅 중...")
1274
1275     # 1. 전체 파일 스캔
1276     all_files, dir_groups = force_scan_all_audio_files(data_path)
1277     print(f"📁 전체 스캔: {len(all_files)}개 파일 발견")
1278
1279     # 2. 클래스 매칭 테스트
1280     audio_files, labels = scan_three_class_data(data_path)
1281     print(f"🏷️ 클래스 매칭: {len(audio_files)}개 파일 매칭")
1282
1283     # 3. 누락된 파일들 분석
1284     matched_paths = set(audio_files)
1285     all_paths = set([f['path'] for f in all_files])
1286     missing_files = all_paths - matched_paths
1287
1288     if missing_files:
1289         print(f"\n! 누락된 파일들 ({len(missing_files)}개):")
1290         missing_by_dir = defaultdict(list)
1291         for missing_path in list(missing_files)[:20]: # 처음 20개만 표시
1292             dir_name = os.path.dirname(os.path.realpath(missing_path, data_path))
1293             missing_by_dir[dir_name].append(os.path.basename(missing_path))
1294
1295         for dir_name, files in missing_by_dir.items():
1296             print(f"📁 {dir_name}: {len(files)}개")
1297             for file in files[:3]:
1298                 print(f"    - {file}")
1299             if len(files) > 3:
1300                 print(f"    ... 그 외 {len(files)-3}개")
1301
1302     # 4. 실제 로딩 테스트
1303     print(f"\n🎧 실제 오디오 로딩 테스트 (처음 10개 파일)...")
1304     loading_errors = 0
1305
1306     for i, file_path in enumerate(audio_files[:10]):
1307         try:
1308             audio, sr = librosa.load(file_path, sr=16000, duration=1.0) # 1초만 테스트
1309             duration = len(audio) / sr
1310             print(f"🔊 {i+1}: {os.path.basename(file_path)} ({duration:.1f}초, {sr}Hz)")
1311             except Exception as e:
1312                 print(f"❌ {i+1}: {os.path.basename(file_path)} - {str(e)}")
1313                 loading_errors += 1
1314
1315     if loading_errors > 0:
1316         print(f"\n⚠️ {loading_errors}개 파일에서 로딩 오류 발생")
1317         print("일부 파일이 손상되었거나 지원되지 않는 형식일 수 있습니다.")
1318
1319     return {
1320         'total_found': len(all_files),
1321         'matched': len(audio_files),
1322         'missing': len(missing_files),
1323         'loading_errors': loading_errors,
1324         'dir_groups': dir_groups
1325     }
1326
1327 def create_manual_dataset(data_path):
1328     """수동으로 데이터셋 생성 (모든 파일 사용)"""
1329     print("📁 수동 데이터셋 생성 중...")
1330
1331     # 모든 파일 스캔
1332     all_files, dir_groups = force_scan_all_audio_files(data_path)
1333
1334     audio_files = []
1335     labels = []
1336
1337     print(f"\n라 디렉토리의 클래스를 수동으로 지정해주세요:")
1338     print("1: Footstep (발소리)")
```

```
1340 print("2: speech (말소리)")
1341 print("3: furniture (가구끄는소리)")
1342 print("0: skip (제외)")
1343
1344 class_mapping = {'1': 'footstep', '2': 'speech', '3': 'furniture'}
1345
1346 for dir_name, file_list in sorted(dir_groups.items(), key=lambda x: len(x[1]), reverse=True):
1347     print(f"\n📁 {dir_name} ({len(file_list)}개 파일)")
1348     print(f"    샘플 파일: {', '.join([f['filename'] for f in file_list[:3]])}")
1349
1350     while True:
1351         try:
1352             choice = input(f"    클래스 선택 (1/2/3/0: ").strip()
1353             if choice == '0':
1354                 print(f"👉 {dir_name} 건너 제외")
1355                 break
1356             elif choice in class_mapping:
1357                 selected_class = class_mapping[choice]
1358                 print(f"✅ {dir_name} -> {selected_class}")
1359
1360                 # 파일들 추가
1361                 for file_info in file_list:
1362                     audio_files.append(file_info['path'])
1363                     labels.append(selected_class)
1364                 break
1365             else:
1366                 print("잘못된 선택입니다. 다시 입력해주세요.")
1367         except KeyboardInterrupt:
1368             print("\n작업이 중단되었습니다.")
1369             return [], []
1370
1371 print(f"\n📁 수동 데이터셋 생성 완료:")
1372 print(f"    - 총 파일 수: {len(audio_files)}개")
1373 for class_name in ['footstep', 'speech', 'furniture']:
1374     count = labels.count(class_name)
1375     percentage = (count / len(labels)) * 100 if len(labels) > 0 else 0
1376     print(f"    - {class_name}: {count}개 ({percentage:.1f}%)")
1377
1378 return audio_files, labels
1379
1380 """샘플 데이터 또는 실제 데이터로 학습 실행"""
1381
1382 if data_path is None:
1383     print("🎧 샘플 데이터를 생성하고 학습을 시작합니다...")
1384     # 샘플 데이터 생성
1385     sample_dir = generate_sample_audio_data('sample_data', samples_per_class=30)
1386     data_path = sample_dir
1387     print("✅ 샘플 데이터 생성 완료!")
1388 else:
1389     print(f"📁 실제 데이터를 사용합니다: {data_path}")
1390
1391 # 학습 실행
1392 print("🚀 학습을 시작합니다...")
1393 model, trainer = run_three_class_training(
1394     data_path=data_path,
1395     num_epochs=num_epochs,
1396     batch_size=batch_size,
1397     test_size=0.2
1398 )
1399
1400 print("✅ 학습 완료! 이제 테스트해보세요.")
1401 return model, trainer
1402
1403 def test_model_with_samples():
1404     """샘플로 모델 테스트"""
1405     if not os.path.exists('best_three_class_model.pth'):
1406         print("❌ 학습된 모델이 없습니다!")
1407         print("run_sample_training() 먼저 실행해주세요.")
1408         return
1409
1410     if not os.path.exists('sample_data'):
1411         print("❌ 샘플 데이터가 없습니다!")
1412         return
1413
1414     print("🎧 샘플 파일들로 모델 테스트 중...")
1415
1416     # 각 클래스에서 랜덤 파일 선택
1417     classes = ['footstep', 'speech', 'furniture']
1418
1419     for class_name in classes:
1420         class_dir = os.path.join('sample_data', class_name)
1421         if os.path.exists(class_dir):
1422             files_list = [f for f in os.listdir(class_dir) if f.endswith('.wav')]
1423             if files_list:
1424                 test_file = os.path.join(class_dir, random.choice(files_list))
1425                 print(f"\n🎧 테스트 중: {class_name} 클래스")
1426                 print(f"    파일: {test_file}")
1427
1428                 result = predict_audio_file('best_three_class_model.pth', test_file)
1429                 if result:
1430                     correct = "✅" if result['predicted_class_eng'] == class_name else "❌"
1431                     print(f"    {correct} 예측: {result['predicted_class']} | 신뢰도: {result['confidence']:.2f}")
1432
1433                 # 오디오 재생
```



```
1433         audio, sr = librosa.load(test_file, sr=16000)
1434         display(Audio(audio, rate=sr))
1435
1436 def demo_realtime_features():
1437     """실시간 특징 추출 데모"""
1438     print("🔊 실시간 특징 추출 데모")
1439
1440     # 짧은 테스트 신호 생성
1441     sr = 16000
1442     duration = 2.0
1443     t = np.linspace(0, duration, int(sr * duration))
1444
1445     # 다양한 신호 생성
1446     signals = {
1447         '발소리 시뮬레이션': create_footstep_signal(t, sr),
1448         '말소리 시뮬레이션': create_speech_signal(t, sr),
1449         '가구 시뮬레이션': create_furniture_signal(t, sr)
1450     }
1451
1452     for name, signal in signals.items():
1453         print(f"\n🔊 {name} 특징 추출 중...")
1454
1455         # 특징 추출
1456         mel_spec = librosa.feature.melspectrogram(y=signal, sr=sr, n_mels=64)
1457         mfcc = librosa.feature.mfcc(y=signal, sr=sr, n_mfcc=13)
1458
1459         # 시각화
1460         fig, axes = plt.subplots(1, 3, figsize=(15, 4))
1461
1462         # 원본 신호
1463         axes[0].plot(t, signal)
1464         axes[0].set_title(f'{name} - 시간 도메인')
1465         axes[0].set_xlabel('시간 (초)')
1466         axes[0].set_ylabel('진폭')
1467
1468         # Mel-spectrogram
1469         librosa.display.specshow(librosa.power_to_db(mel_spec, sr=sr, x_axis='time', y_axis='mel', ax=axes[1])
1470         axes[1].set_title('Mel-spectrogram')
1471
1472         # MFCC
1473         librosa.display.specshow(mfcc, sr=sr, x_axis='time', ax=axes[2])
1474         axes[2].set_title('MFCC')
1475
1476         plt.tight_layout()
1477         plt.show()
1478
1479         # 오디오 재생
1480         display(Audio(signal, rate=sr))
1481
1482 def create_footstep_signal(t, sr):
1483     """발소리 신호 생성"""
1484     signal = np.zeros_like(t)
1485     step_times = [0.3, 0.9, 1.5] # 발걸음 시간
1486
1487     for step_time in step_times:
1488         start_idx = int(step_time * sr)
1489         if start_idx < len(signal) - 2000:
1490             # 충격음 (감쇠하는 지수파) / sr
1491             impact_t = np.arange(2000) / sr
1492             impact = np.exp(-impact_t * 5) * np.sin(2 * np.pi * 80 * impact_t)
1493             signal[start_idx:start_idx+2000] += impact * 0.8
1494
1495     return signal
1496
1497 def create_speech_signal(t, sr):
1498     """말소리 신호 생성"""
1499     # 기본 주파수 (피치)
1500     f0 = 150 # Hz
1501     speech = 0.3 * np.sin(2 * np.pi * f0 * t)
1502
1503     # 포먼트 추가
1504     formants = [800, 1200, 2400]
1505     for formant in formants:
1506         speech += 0.1 * np.sin(2 * np.pi * formant * t)
1507
1508     # 진폭 변조 (말하는 리듬)
1509     modulation = 1 + 0.5 * np.sin(2 * np.pi * 3 * t)
1510     speech *= modulation
1511
1512     return speech
1513
1514 def create_furniture_signal(t, sr):
1515     """가구끄는소리 신호 생성"""
1516     # 마찰음 (광대역 노이즈를 필터링)
1517     noise = np.random.normal(0, 1, len(t))
1518
1519     # 로우패스 필터 (마찰음 특성)
1520     butter_b, butter_a = scipy.signal.butter(4, 500, fs=sr)
1521     filtered = scipy.signal.filtfilt(butter_b, butter_a, noise)
1522
1523     # 불규칙한 진폭
1524     amplitude_env = np.random.uniform(0.2, 0.8, 50)
1525     amplitude = np.interp(t, np.linspace(0, t[-1], 50), amplitude_env)
```

```
1526
1527     return filtered * amplitude * 0.5
1528
1529 # =====
1530 # 12. 사용 가이드 및 실행 함수
1531 # =====
1532
1533 def show_complete_usage_guide():
1534     """완전한 사용 가이드"""
1535     print("""
1536     🎧 발소리-말소리-가구끄는소리 분류 AI 모델 사용 가이드
1537     =====
1538
1539     📌 주요 기능:
1540     1. 3클래스 오디오 분류 (발소리, 말소리, 가구끄는소리)
1541     2. CNN 기반 딥러닝 모델
1542     3. 실시간 예측 및 시각화
1543     4. 데이터 증강 및 최적화
1544     5. 한글 폰트 지원
1545
1546     🚀 빠른 시작:
1547
1548     ❶ 한글 폰트 테스트:
1549         test_korean_font() # 한글이 제대로 표시되는지 확인
1550
1551     ❷ 샘플 데이터로 빠른 테스트:
1552         run_sample_training() # 샘플 생성 + 학습
1553         test_model_with_samples() # 테스트
1554
1555     ❸ 실제 데이터로 학습:
1556         # 데이터 폴더 구조:
1557         # your_data/
1558         # ├── footstep/ (발소리 파일들)
1559         # ├── speech/ (말소리 파일들)
1560         # └── furniture/ (가구끄는소리 파일들)
1561
1562         model, trainer = run_sample_training('/path/to/your_data')
1563
1564     ❹ 파일 업로드하여 예측:
1565         analyze_uploaded_file()
1566
1567     ❺ 실시간 녹음 예측:
1568         record_and_predict('best_three_class_model.pth')
1569
1570     ❻ 특징 추출 데모:
1571         demo_realtime_features()
1572
1573     ❼ 모델 성능 상세 분석:
1574         analyze_model_performance(model, val_loader)
1575
1576     📊 모델 성능:
1577     - 입력: 5초 오디오 (16kHz)
1578     - 특징: Mel-spectrogram (64) + MFCC (64) = 128차원
1579     - 구조: CNN (4블록) + Global Average Pooling
1580     - 출력: 3클래스 확률 분포
1581
1582     📄 한글 폰트 문제 해결:
1583     - 그래프에서 한글이 깨진다면: test_korean_font() 실행
1584     - 여전히 문제가 있다면: 런타임 재시작 후 다시 실행
1585
1586     💡 팁:
1587     - GPU 사용 시 batch_size를 16으로 증가 가능
1588     - 데이터가 부족하면 augmentation 강화
1589     - 과적합 시 dropout_rate 증가
1590
1591     ⚡ 지금 시작하기:
1592         test_korean_font() # 한글 폰트 확인
1593         run_sample_training() # 샘플 학습
1594     """)
1595
1596 # 컴포넌트별 등록 (Colab 전용)
1597 try:
1598     from google.colab import output
1599     output.register_callback('save_audio', save_audio)
1600 except:
1601     pass
1602
1603 # =====
1604 # 13. 고급 분석 도구
1605 # =====
1606
1607 def analyze_model_performance(model, val_loader):
1608     """모델 성능 상세 분석 - 한글 지원"""
1609     model.eval()
1610     device = next(model.parameters()).device
1611
1612     all_preds = []
1613     all_labels = []
1614     all_confidences = []
1615
1616     print("🔍 모델 성능 상세 분석 중...")
1617
1618     with torch.no_grad():
```

```
1619     for batch in tqdm(val_loader, desc="분석"):
1620         inputs = batch['features'].to(device)
1621         labels = batch['label'].squeeze().to(device)
1622
1623         outputs = model(inputs)
1624         probabilities = F.softmax(outputs, dim=1)
1625         predicted = torch.argmax(outputs, dim=1)
1626         confidence = torch.max(probabilities, dim=1)[0]
1627
1628         all_preds.extend(predicted.cpu().numpy())
1629         all_labels.extend(labels.cpu().numpy())
1630         all_confidences.extend(confidence.cpu().numpy())
1631
1632 # 성능 메트릭 계산
1633 accuracy = accuracy_score(all_labels, all_preds)
1634 precision, recall, f1, _ = precision_recall_fscore_support(
1635     all_labels, all_preds, average='weighted'
1636 )
1637
1638 # 신뢰도 분석
1639 correct_mask = np.array(all_preds) == np.array(all_labels)
1640 correct_confidences = np.array(all_confidences)[correct_mask]
1641 incorrect_confidences = np.array(all_confidences)[~correct_mask]
1642
1643 # 결과 출력
1644 print(f"\n📊 전체 성능 메트릭:")
1645 print(f" - 정확도: {accuracy:.3f}")
1646 print(f" - 정밀도: {precision:.3f}")
1647 print(f" - 재현율: {recall:.3f}")
1648 print(f" - F1 점수: {f1:.3f}")
1649 print(f"\n🔍 신뢰도 분석:")
1650 print(f" - 올바른 예측 평균 신뢰도: {correct_confidences.mean():.3f}")
1651 print(f" - 잘못된 예측 평균 신뢰도: {incorrect_confidences.mean():.3f}")
1652
1653 # 한글 폰트 설정
1654 try:
1655     plt.rcParams['font.family'] = 'NanumGothic'
1656     plt.rcParams['axes.unicode_minus'] = False
1657 except:
1658     pass
1659
1660 # 신뢰도 분포 시각화
1661 plt.figure(figsize=(12, 8))
1662
1663 # 서브플롯 1: 신뢰도 히스토그램
1664 plt.subplot(2, 2, 1)
1665 plt.hist(correct_confidences, bins=20, alpha=0.7, label='올바른 예측', color='green', density=True)
1666 plt.hist(incorrect_confidences, bins=20, alpha=0.7, label='잘못된 예측', color='red', density=True)
1667 plt.xlabel('신뢰도')
1668 plt.ylabel('밀도')
1669 plt.title('예측 신뢰도 분포')
1670 plt.legend()
1671 plt.grid(True, alpha=0.3)
1672
1673 # 서브플롯 2: 박스플롯
1674 plt.subplot(2, 2, 2)
1675 data_to_plot = [correct_confidences, incorrect_confidences]
1676 box = plt.boxplot(data_to_plot, labels=['올바른 예측', '잘못된 예측'], patch_artist=True)
1677 box['boxes'][0].set_facecolor('lightgreen')
1678 box['boxes'][1].set_facecolor('lightcoral')
1679 plt.ylabel('신뢰도')
1680 plt.title('신뢰도 박스플롯')
1681 plt.grid(True, alpha=0.3)
1682
1683 # 서브플롯 3: 클래스별 성능
1684 plt.subplot(2, 2, 3)
1685 class_names = ['발소리', '말소리', '가구끄는소리']
1686 class_f1_scores = []
1687
1688 for i in range(3):
1689     class_mask = np.array(all_labels) == i
1690     if class_mask.sum() > 0:
1691         class_preds = np.array(all_preds)[class_mask]
1692         class_labels = np.array(all_labels)[class_mask]
1693         class_f1 = f1_score(class_labels, class_preds, average='binary', pos_label=i, zero_division=0)
1694         class_f1_scores.append(class_f1)
1695     else:
1696         class_f1_scores.append(0)
1697
1698 colors = ['#ff9999', '#66b3ff', '#99ff99']
1699 bars = plt.bar(class_names, class_f1_scores, color=colors, alpha=0.8)
1700 plt.ylabel('F1 점수')
1701 plt.title('클래스별 F1 점수')
1702 plt.ylim(0, 1.1)
1703
1704 # 각 막대 위에 값 표시
1705 for bar, score in zip(bars, class_f1_scores):
1706     plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.02,
1707             f'{score:.3f}', ha='center', va='bottom')
1708
1709 # 서브플롯 4: 신뢰도 vs 정확도
1710 plt.subplot(2, 2, 4)
1711 confidence_bins = np.linspace(0. 1. 11)
```

```
1712 bin_accuracies = []
1713 bin_centers = []
1714
1715 for i in range(len(confidence_bins) - 1):
1716     bin_mask = ((np.array(all_confidences) >= confidence_bins[i]) &
1717                 (np.array(all_confidences) < confidence_bins[i+1]))
1718
1719     if bin_mask.sum() > 0:
1720         bin_accuracy = (np.array(all_preds)[bin_mask] == np.array(all_labels)[bin_mask]).mean()
1721         bin_accuracies.append(bin_accuracy)
1722         bin_centers.append((confidence_bins[i] + confidence_bins[i+1]) / 2)
1723
1724 if bin_centers:
1725     plt.plot(bin_centers, bin_accuracies, 'bo-', linewidth=2, markersize=6)
1726     plt.plot([0, 1], [0, 1], 'r--', alpha=0.7, label='완벽한 보정')
1727     plt.xlabel('신뢰도')
1728     plt.ylabel('정확도')
1729     plt.title('신뢰도 보정 곡선')
1730     plt.legend()
1731     plt.grid(True, alpha=0.3)
1732
1733 plt.tight_layout()
1734 plt.show()
1735
1736 # 상세 통계
1737 print(f"\n📊 상세 통계:")
1738 print(f" - 총 예측 샘플: {len(all_labels)}개")
1739 print(f" - 올바른 예측: {correct_mask.sum()}개 ({correct_mask.mean()*100:.1f}%)")
1740 print(f" - 잘못된 예측: {(~correct_mask).sum()}개 ({(~correct_mask).mean()*100:.1f}%)")
1741 print(f" - 평균 신뢰도: {np.mean(all_confidences):.3f}")
1742 print(f" - 신뢰도 표준편차: {np.std(all_confidences):.3f}")
1743
1744 return {
1745     'accuracy': accuracy,
1746     'precision': precision,
1747     'recall': recall,
1748     'f1': f1,
1749     'correct_confidences': correct_confidences,
1750     'incorrect_confidences': incorrect_confidences,
1751     'class_f1_scores': class_f1_scores
1752 }
1753
1754 def batch_predict_directory(model_path, test_dir):
1755     """디렉토리 내 모든 파일 일괄 예측"""
1756     if not os.path.exists(model_path):
1757         print("❌ 모델 파일이 없습니다!")
1758     return
1759
1760 print(f"\n📁 {test_dir} 내 모든 오디오 파일 예측 중...")
1761
1762 audio_extensions = ['.wav', '.mp3', '.flac', '.m4a']
1763 audio_files = []
1764
1765 for root, dirs, files in os.walk(test_dir):
1766     for file in files:
1767         if any(file.lower().endswith(ext) for ext in audio_extensions):
1768             audio_files.append(os.path.join(root, file))
1769
1770 if not audio_files:
1771     print("❌ 오디오 파일을 찾을 수 없습니다!")
1772     return
1773
1774 results = []
1775
1776 for audio_file in tqdm(audio_files, desc="예측"):
1777     result = predict_audio_file(model_path, audio_file)
1778     if result:
1779         results.append({
1780             'file': os.path.basename(audio_file),
1781             'predicted_class': result['predicted_class'],
1782             'confidence': result['confidence'],
1783             'path': audio_file
1784         })
1785
1786 # 결과를 DataFrame으로 정리
1787 df = pd.DataFrame(results)
1788
1789 print(f"\n📊 일괄 예측 결과:")
1790 print(df.groupby('predicted_class').agg({
1791     'confidence': ['count', 'mean', 'min', 'max'],
1792     'file': 'count'
1793 }).round(3))
1794
1795 return df
1796
1797 # =====
1798 # 14. 메인 실행 부분
1799 # =====
1800
1801 def main():
1802     """메인 실행 함수"""
1803     print("🔊 발소리-말소리-가구끄는소리 분류 AI 모델")
1804     print(f"===== 70s")
```

```
1805
1806 # 환경 설정
1807 setup_colab_environment()
1808
1809 # 사용 가이드 출력
1810 show_complete_usage_guide()
1811
1812 print("\n👉 다음 중 하나를 선택하세요:")
1813 print("1. test_korean_font()      # 한글 폰트 테스트")
1814 print("2. run_sample_training()    # 샘플 데이터로 학습")
1815 print("3. analyze_uploaded_file()   # 파일 업로드 분석")
1816 print("4. demo_realtime_features()  # 특징 추출 데모")
1817 print("5. test_model_with_samples() # 샘플로 테스트")
1818
1819 # 자동 실행
1820 if __name__ == "__main__":
1821     main()
1822
1823 # =====
1824 # 15. 추가 유틸리티 함수들
1825 # =====
1826
1827 def save_model_info(model_path):
1828     """모델 정보 저장"""
1829     if os.path.exists(model_path):
1830         checkpoint = torch.load(model_path, map_location='cpu')
1831
1832         info = {
1833             'model_type': '3클래스 오디오 분류 CNN',
1834             'classes': ['발소리', '말소리', '가구끄는소리'],
1835             'epoch': checkpoint.get('epoch', 'Unknown'),
1836             'val_accuracy': checkpoint.get('val_acc', 'Unknown'),
1837             'val_loss': checkpoint.get('val_loss', 'Unknown'),
1838             'input_shape': '(128, 157)',
1839             'sample_rate': '16kHz',
1840             'max_duration': '5초'
1841         }
1842
1843         print("📄 모델 정보:")
1844         for key, value in info.items():
1845             print(f"   - {key}: {value}")
1846
1847         return info
1848     else:
1849         print("❌ 모델 파일을 찾을 수 없습니다.")
1850         return None
1851
1852 def export_model_for_production(model_path, output_path='model_production.pth'):
1853     """프리케이션용 모델 내보내기"""
1854     if not os.path.exists(model_path):
1855         print("❌ 모델 파일이 없습니다!")
1856         return
1857
1858     print("🔗 프리케이션용 모델 준비 중...")
1859
1860     # 모델 로드
1861     checkpoint = torch.load(model_path, map_location='cpu')
1862     model = ThreeClassAudioCNN(num_classes=3)
1863     model.load_state_dict(checkpoint['model_state_dict'])
1864     model.eval()
1865
1866     # TorchScript로 변환
1867     dummy_input = torch.randn(1, 1, 128, 157)
1868     traced_model = torch.jit.trace(model, dummy_input)
1869
1870     # 저장
1871     torch.jit.save(traced_model, output_path)
1872
1873     print(f"✅ 프리케이션용 모델 저장 완료: {output_path}")
1874     print("   이 모델은 별도 라이브러리 없이 PyTorch에서 바로 로드 가능합니다.")
1875
1876     # 사용법 출력
1877     print("\n👉 프리케이션 환경에서 사용법:")
1878     print(f"   model = torch.jit.load('{output_path}')"")
1879     print("   output = model(input_tensor)")
1880
1881 def quick_audio_preview(audio_path):
1882     """오디오 파일 빠른 미리보기"""
1883     try:
1884         audio, sr = librosa.load(audio_path, sr=16000, duration=10)
1885
1886         print(f"🎵 파일: {os.path.basename(audio_path)}")
1887         print(f"   - 길이: {len(audio)/sr:.2f}초")
1888         print(f"   - 샘플링 레이트: {sr}Hz")
1889         print(f"   - 최대 진폭: {np.max(np.abs(audio)):.3f}")
1890         print(f"   - RMS: {np.sqrt(np.mean(audio**2)):.3f}")
1891
1892         # 간단한 시각화
1893         plt.figure(figsize=(12, 4))
1894         time = np.linspace(0, len(audio)/sr, len(audio))
1895         plt.plot(time, audio)
1896         plt.title(f'오디오 파형: {os.path.basename(audio_path)}')
1897         plt.show()
```

```
1897
1898     plt.ylabel('진폭')
1899     plt.grid(True, alpha=0.3)
1900     plt.show()
1901
1902     # 오디오 재생
1903     display(Audio(audio, rate=sr))
1904
1905     except Exception as e:
1906         print(f"❌ 오디오 로딩 오류: {e}")
1907
1908     # 최종 메시지
1909     print("\n🎉 모든 기능이 준비되었습니다!")
1910     print("   한글 그래프 지원이 추가되었습니다! 🇰🇷")
1911     print("   강력한 데이터 디버깅 기능이 추가되었습니다! 🐞")
1912     print("\n🔍 500개 파일 중 150개만 학습되는 문제 해결:")
1913     print("   detailed_data_analysis('/your/data/path')")
1914     print("   run_sample_training('/your/data/path') # 디버깅 모드 자동 실행")
1915     print("\n🗣️ 먼저 한글 폰트를 테스트해보세요:")
1916     print("   test_korean_font()")
1917     print("\n💡 이제 모든 오디오 파일을 놓치지 않고 학습할 수 있습니다!")
```

```
📁 # your_data/
   ├── footsteps/   (발소리 파일들)
   ├── speech/      (말소리 파일들)
   └── furniture/    (가구끄는소리 파일들)

model, trainer = run_sample_training('/path/to/your_data')

3 파일 업로드하여 예측:
analyze_uploaded_file()

4 실시간 녹음 예측:
record_and_predict('best_three_class_model.pth')

5 특징 추출 데모:
demo_realtime_features()

6 모델 성능 상세 분석:
analyze_model_performance(model, val_loader)

📊 모델 성능:
- 입력: 5초 오디오 (16kHz)
- 특징: Mel-spectrogram (64) + MFCC (64) = 128차원
- 구조: CNN (4블록) + Global Average Pooling
- 출력: 3클래스 확률 분포

🗣️ 한글 폰트 문제 해결:
- 그래프에서 한글이 깨진다면: test_korean_font() 실행
- 여전히 문제가 있다면: 폰트임 재시작 후 다시 실행

💡 팁:
- GPU 사용 시 batch_size를 16으로 증가 가능
- 데이터가 부족하면 augmentation 강화
- 과적합 시 dropout_rate 증가

⚡ 지금 시작하기:
test_korean_font()      # 한글 폰트 확인
run_sample_training()    # 샘플 학습
```

👉 다음 중 하나를 선택하세요:

1. test_korean_font() # 한글 폰트 테스트
2. run_sample_training() # 샘플 데이터로 학습
3. analyze_uploaded_file() # 파일 업로드 분석
4. demo_realtime_features() # 특징 추출 데모
5. test_model_with_samples() # 샘플로 테스트

🎉 모든 기능이 준비되었습니다!
한글 그래프 지원이 추가되었습니다! 🇰🇷
강력한 데이터 디버깅 기능이 추가되었습니다! 🐞

🔍 500개 파일 중 150개만 학습되는 문제 해결:
detailed_data_analysis('/your/data/path')
run_sample_training('/your/data/path') # 디버깅 모드 자동 실행

🗣️ 먼저 한글 폰트를 테스트해보세요:
test_korean_font()

💡 이제 모든 오디오 파일을 놓치지 않고 학습할 수 있습니다!

1 run_sample_training(r"/content/drive/MyDrive/Colab Notebooks/finaldata")

```
실제 데이터를 사용합니다. /content/drive/MyDrive/Colab Notebooks/finaldata
학습을 시작합니다...
발소리-말소리-가구끄는소리 분류 학습 시작!
=====
W: Skipping acquire of configured file 'main/source/Sources' as repository 'https://r2u.stat.illinois.edu/ubuntu' jammy InRelease' does not seem to pr
[✓] 한글 폰트 설정 완료
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
[✓] Google Drive 마운트 완료
[✓] Colab 환경 최적화 완료
[✓] 메모리 정리 완료

/content/drive/MyDrive/Colab Notebooks/finaldata에서 오디오 파일 스캔 중...
지원 확장자: .wav, .mp3, .flac, .m4a, .ogg, .aac, .wma, .aiff, .au
a - 189개 오디오 파일 발견
? 'a' 폴더의 클래스를 판단할 수 없습니다.
  다음 중 하나를 선택하세요:
  1: footstep (발소리)
  2: speech (말소리)
  3: furniture (가구끄는소리)
  0: skip (건너뛰기)
선택 (1/2/3/0): 1
[✓] 'a' -> footstep
b - 140개 오디오 파일 발견
? 'b' 폴더의 클래스를 판단할 수 없습니다.
  다음 중 하나를 선택하세요:
  1: footstep (발소리)
  2: speech (말소리)
  3: furniture (가구끄는소리)
  0: skip (건너뛰기)
선택 (1/2/3/0): 2
[✓] 'b' -> speech
[✓] footstep - 150개 오디오 파일 발견
[✓] footstep -> footstep

데이터 스캔 완료:
- 총 발견된 오디오 파일: 479개
- 실제 사용할 파일: 479개
- 처리된 디렉토리: 3개

처리된 디렉토리 상세:
- a: footstep (189개)
- b: speech (140개)
- footstep: footstep (150개)

클래스별 파일 수:
- footstep: 339개 (70.8%)
- speech: 140개 (29.2%)
- furniture: 0개 (0.0%)

데이터 분할:
- 학습: 383개
- 검증: 96개

데이터셋 정보:
- 총 샘플 수: 383
- 클래스: ['footstep', 'speech', 'furniture']
- footstep: 271개
- speech: 112개
- furniture: 0개

데이터셋 정보:
- 총 샘플 수: 96
- 클래스: ['footstep', 'speech', 'furniture']
- footstep: 68개
- speech: 28개
- furniture: 0개
모델 파라미터 수: 422,083
학습 설정:
- 장치: cuda
- 학습률: 0.001
- 클래스: 3개 (발소리, 말소리, 가구끄는소리)
[✓] 3클래스 분류 학습 시작!
=====
```

```
Epoch 1/15
학습: 100% 96/96 [00:46<00:00, 1.04it/s, Loss=0.9821, Acc=71.0%]

검증: 100% 24/24 [00:18<00:00, 3.54it/s, Loss=0.5985, Acc=94.8%]

  학습 - Loss: 0.5761, Acc: 71.02%
  검증 - Loss: 0.3468, Acc: 94.79%
  학습률: 0.001000
  [✓] 최고 모델 저장! (검증 정확도: 94.79%)
    - 발소리: 97.1%
    - 말소리: 89.3%
[✓] 메모리 정리 완료
```

```
Epoch 2/15
학습: 100% 96/96 [00:23<00:00, 7.85it/s, Loss=1.8855, Acc=78.3%]

검증: 100% 24/24 [00:03<00:00, 9.29it/s, Loss=0.6827, Acc=71.9%]

  학습 - Loss: 0.4471, Acc: 78.33%
  검증 - Loss: 0.4256, Acc: 71.88%
  학습률: 0.001000
```

```
Epoch 3/15
학습: 100% 96/96 [00:20<00:00, 2.91it/s, Loss=0.0863, Acc=84.3%]

검증: 100% 24/24 [00:03<00:00, 9.44it/s, Loss=0.1778, Acc=99.0%]

  학습 - Loss: 0.3596, Acc: 84.33%
  검증 - Loss: 0.1055, Acc: 98.96%
  학습률: 0.001000
  [✓] 최고 모델 저장! (검증 정확도: 98.96%)
=====
```

```
- 발소리: 98.5%
- 말소리: 100.0%
```

```
Epoch 4/15
학습: 100% 96/96 [00:21<00:00, 2.90it/s, Loss=0.2751, Acc=87.2%]

검증: 100% 24/24 [00:04<00:00, 9.04it/s, Loss=0.0750, Acc=100.0%]

  학습 - Loss: 0.2743, Acc: 87.21%
  검증 - Loss: 0.0505, Acc: 100.00%
  학습률: 0.001000
  [✓] 최고 모델 저장! (검증 정확도: 100.00%)
    - 발소리: 100.0%
    - 말소리: 100.0%
```

```
Epoch 5/15
학습: 100% 96/96 [00:18<00:00, 3.10it/s, Loss=0.4732, Acc=94.3%]

검증: 100% 24/24 [00:04<00:00, 9.35it/s, Loss=0.0372, Acc=100.0%]

  학습 - Loss: 0.1766, Acc: 94.26%
  검증 - Loss: 0.0166, Acc: 100.00%
  학습률: 0.001000
```

```
Epoch 6/15
학습: 100% 96/96 [00:19<00:00, 2.74it/s, Loss=0.5598, Acc=93.0%]

검증: 100% 24/24 [00:04<00:00, 3.77it/s, Loss=0.0237, Acc=100.0%]

  학습 - Loss: 0.2039, Acc: 92.95%
  검증 - Loss: 0.0122, Acc: 100.00%
  학습률: 0.001000
[✓] 메모리 정리 완료
```

```
Epoch 7/15
학습: 100% 96/96 [00:18<00:00, 7.23it/s, Loss=0.0272, Acc=95.3%]

검증: 100% 24/24 [00:03<00:00, 9.26it/s, Loss=0.0160, Acc=99.0%]

  학습 - Loss: 0.1490, Acc: 95.30%
  검증 - Loss: 0.0354, Acc: 98.96%
  학습률: 0.001000
```

```
Epoch 8/15
학습: 100% 96/96 [00:21<00:00, 2.46it/s, Loss=0.0704, Acc=93.5%]

검증: 100% 24/24 [00:03<00:00, 10.11it/s, Loss=0.0143, Acc=100.0%]

  학습 - Loss: 0.1606, Acc: 93.47%
  검증 - Loss: 0.0071, Acc: 100.00%
  학습률: 0.001000
```

```
Epoch 9/15
학습: 100% 96/96 [00:18<00:00, 6.27it/s, Loss=0.0709, Acc=94.5%]

검증: 100% 24/24 [00:05<00:00, 9.08it/s, Loss=0.0036, Acc=100.0%]

  학습 - Loss: 0.1309, Acc: 94.52%
  검증 - Loss: 0.0055, Acc: 100.00%
  학습률: 0.001000
```

```
Epoch 10/15
학습: 100% 96/96 [00:19<00:00, 2.98it/s, Loss=0.0694, Acc=93.5%]

검증: 100% 24/24 [00:04<00:00, 3.80it/s, Loss=0.0112, Acc=100.0%]

  학습 - Loss: 0.1696, Acc: 93.47%
  검증 - Loss: 0.0092, Acc: 100.00%
  학습률: 0.000500
```

```
Epoch 11/15
학습: 100% 96/96 [00:19<00:00, 2.74it/s, Loss=0.1516, Acc=95.6%]

검증: 100% 24/24 [00:04<00:00, 8.51it/s, Loss=0.0060, Acc=100.0%]

  학습 - Loss: 0.1174, Acc: 95.56%
  검증 - Loss: 0.0040, Acc: 100.00%
  학습률: 0.000500
[✓] 메모리 정리 완료
```

```
Epoch 12/15
학습: 100% 96/96 [00:21<00:00, 5.91it/s, Loss=0.0013, Acc=98.2%]

검증: 100% 24/24 [00:03<00:00, 9.32it/s, Loss=0.0020, Acc=100.0%]

  학습 - Loss: 0.0531, Acc: 98.17%
  검증 - Loss: 0.0018, Acc: 100.00%
  학습률: 0.000500
```

```
Epoch 13/15
학습: 100% 96/96 [00:17<00:00, 8.09it/s, Loss=0.0251, Acc=96.9%]

검증: 100% 24/24 [00:04<00:00, 8.82it/s, Loss=0.0042, Acc=100.0%]

  학습 - Loss: 0.0846, Acc: 96.87%
  검증 - Loss: 0.0023, Acc: 100.00%
  학습률: 0.000500
```

```
Epoch 14/15
학습: 100% 96/96 [00:17<00:00, 8.82it/s, Loss=0.3422, Acc=97.7%]

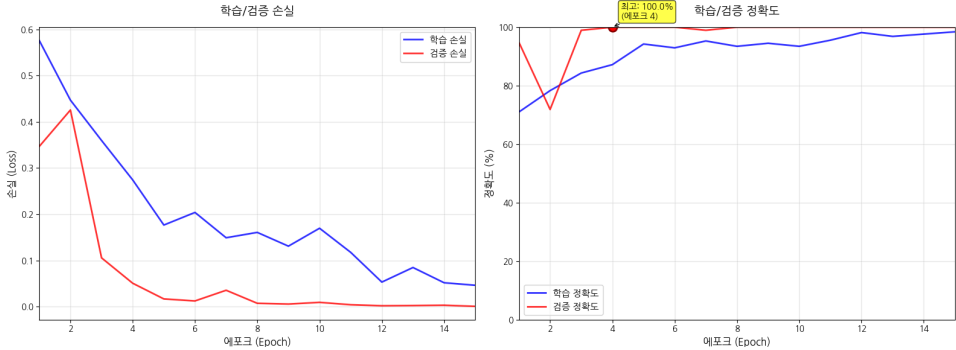
검증: 100% 24/24 [00:04<00:00, 8.60it/s, Loss=0.0020, Acc=100.0%]

  학습 - Loss: 0.0516, Acc: 97.65%
  검증 - Loss: 0.0029, Acc: 100.00%
=====
```

박남훈: 0.000500

Epoch 15/15
학습: 100%
96/96 [00:19<00:00, 2.20it/s, Loss=0.0150, Acc=98.4%]
학습 - Loss: 0.0462, Acc: 98.43%
검증 - Loss: 0.0005, Acc: 100.00%
학습률: 0.000500

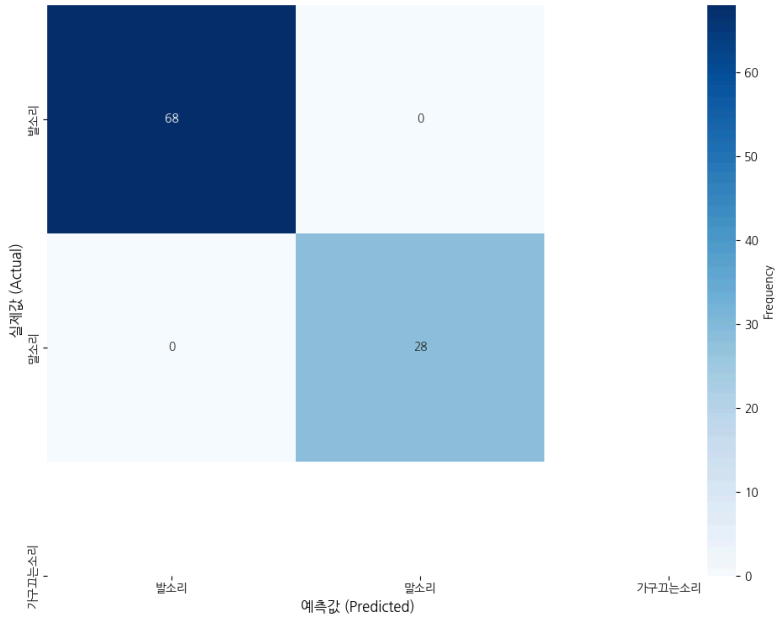
🎉 학습 완료!
최고 검증 정확도: 100.00%



🎉 학습 완료 - 최종 결과 요약

🏆 최고 검증 정확도: 100.00% (에포크 4)
📊 최종 학습 정확도: 98.43%
📉 최종 검증 손실: 0.0005
🕒 총 학습 에포크: 15개
📈 최근 5에포크 정확도 안정성: 0.00% (낮을수록 안정)
✅ 매우 안정적인 학습!

혼동 행렬 (Confusion Matrix)



ValueError Traceback (most recent call last)
/tmp/ipython-input-3983476317.py in <cell line: 0>()
----> 1 run_sample_training(r"/content/drive/MyDrive/Colab Notebooks/finaldata")

4 frames
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/classification.py in classification_report(y_true, y_pred, labels, target_names, sample_weight, digits, output_dict, zero_division)
2691)
2692 else:

--> 2693 raise ValueError(
2694 "Number of classes, {0}, does not match size of "
2695 "target_names, {1}. Try specifying the labels "
ValueError: Number of classes, 2, does not match size of target_names, 3. Try specifying the labels parameter

검증: 100% 24/24 [00:03<00:00, 9.77it/s, Loss=0.0014, Acc=100.0%]

다음 단계: 오류 설명

▼ 절취선

```

1
2
3 # =====
4 # 2. Colab 환경 최적화
5 # =====
6
7 def setup_colab_environment():
8     """Colab 환경 최적화"""
9     try:
10         mp.set_start_method('spawn', force=True)
11     except:
12         pass
13
14     os.environ['TOKENIZERS_PARALLELISM'] = 'false'
15     os.environ['OMP_NUM_THREADS'] = '1'
16
17     # 한글 폰트 설정
18     setup_korean_font()
19
20     # Google Drive 마운트
21     try:
22         drive.mount('/content/drive')
23         print("✅ Google Drive 마운트 완료")
24     except:
25         print("⚠️ Google Drive 마운트 실패 또는 이미 마운트됨")
26
27     print("✅ Colab 환경 최적화 완료")
28
29 def setup_korean_font():
30     """한글 폰트 설정"""
31     try:
32         # 나눔고딕 폰트 설치
33         !apt-get update -qq
34         !apt-get install -qq fonts-nanum
35
36         # matplotlib 폰트 설정
37         import matplotlib.font_manager as fm
38         import matplotlib.pyplot as plt
39
40         # 폰트 캐시 삭제
41         !rm -rf ~/.cache/matplotlib
42
43         # 나눔고딕 폰트 경로
44         font_path = '/usr/share/fonts/truetype/nanum/NanumGothic.ttf'
45
46         if os.path.exists(font_path):
47             # 폰트 등록
48             fm.fontManager.addfont(font_path)
49
50             # matplotlib 기본 폰트 설정
51             plt.rcParams['font.family'] = 'NanumGothic'
52             plt.rcParams['axes.unicode_minus'] = False # 마이너스 기호 깨짐 방지
53
54             print("✅ 한글 폰트 설정 완료")
55         else:
56             # 대체 방법: 구글 폰트 사용
57             !wget -O NanumGothic.ttf "https://github.com/google/fonts/raw/main/ofl/nanumgothic/NanumGothic-Regular.ttf"
58
59             # 폰트 등록
60             fm.fontManager.addfont('./NanumGothic.ttf')
61             plt.rcParams['font.family'] = 'NanumGothic'
62             plt.rcParams['axes.unicode_minus'] = False
63
64             print("✅ 한글 폰트 설정 완료 (대체 폰트)")
65
66     except Exception as e:
67         print(f"⚠️ 한글 폰트 설정 실패: {e}")
68         print("영어로 표시됩니다.")
69
70     # 영어 레이블로 대체
71     plt.rcParams['font.family'] = 'DejaVu Sans'
72
73 def test_korean_font():
74     """한글 폰트 테스트"""
75     print("📄 한글 폰트 테스트 중...")
76
77     plt.figure(figsize=(10, 6))
78
79     # 테스트 데이터
80     classes = ['발소리', '말소리', '가구끼느소리']
81     values = [85.2, 92.1, 78.9]
82     colors = ['#ff9999', '#66b3ff', '#99ff99']
83
84     # 바 차트 생성
85     bars = plt.bar(classes, values, color=colors, alpha=0.8)
86
87     # 제목 및 레이블
88     plt.title('한글 폰트 테스트 - 클래스별 정확도', fontsize=16, fontweight='bold')
89     plt.xlabel('음성 클래스', fontsize=12)

```

```

90     plt.ylabel('정확도 (%)', fontsize=12)
91
92     # 값 표시
93     for bar, value in zip(bars, values):
94         plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 1,
95                 f'{value:.1f}%', ha='center', va='bottom', fontsize=12, fontweight='bold')
96
97     plt.ylim(0, 100)
98     plt.grid(True, axis='y', alpha=0.3)
99     plt.tight_layout()
100    plt.show()
101
102    # 폰트 상태 확인
103    current_font = plt.rcParams['font.family']
104    print(f"현재 폰트: {current_font}")
105
106    if 'NanumGothic' in current_font or 'Nanum' in str(current_font):
107        print("🎉 한글 폰트가 제대로 설정되었습니다!")
108    else:
109        print("⚠️ 한글 폰트 설정에 문제가 있을 수 있습니다.")
110        print("위 그래픽스에서 한글이 깨져 보인다면 런타임을 재시작해보세요.")
111
112 def memory_cleanup():
113     """메모리 정리"""
114     gc.collect()
115     if torch.cuda.is_available():
116         torch.cuda.empty_cache()
117     print("🧹 메모리 정리 완료")
118
119 # =====
120 # 3. 3클래스 오디오 데이터셋 (발소리, 말소리, 가구끼느소리)
121 # =====
122
123 class ThreeClassAudioDataset(Dataset):
124     def __init__(self, audio_paths, labels, target_sr=16000, max_duration=5.0, augment=False):
125         """
126         3클래스 오디오 분류 데이터셋
127
128         Args:
129             audio_paths: 오디오 파일 경로 리스트
130             labels: 레이블 리스트 ['footstep', 'speech', 'furniture']
131             target_sr: 목표 샘플링 레이트 (16kHz)
132             max_duration: 최대 길이 (5초)
133             augment: 데이터 증강 여부
134
135         """
136         self.audio_paths = audio_paths
137         self.target_sr = target_sr
138         self.max_duration = max_duration
139         self.max_length = int(target_sr * max_duration)
140         self.augment = augment
141
142         # 레이블 매핑
143         self.class_names = ['footstep', 'speech', 'furniture']
144         self.label_to_idx = {name: idx for idx, name in enumerate(self.class_names)}
145         self.idx_to_label = {idx: name for name, idx in self.label_to_idx.items()}
146
147         # 레이블 인코딩
148         self.labels = [self.label_to_idx[label] for label in labels]
149
150         print(f"📄 데이터셋 정보:")
151         print(f"- 총 샘플 수: {len(self.audio_paths)}")
152         print(f"- 클래스: {self.class_names}")
153         for i, class_name in enumerate(self.class_names):
154             count = sum(1 for label in self.labels if label == i)
155             print(f"- {class_name}: {count}개")
156
157     def __len__(self):
158         return len(self.audio_paths)
159
160     def __getitem__(self, idx):
161         try:
162             # 오디오 로드
163             audio, sr = librosa.load(
164                 self.audio_paths[idx],
165                 sr=self.target_sr,
166                 duration=self.max_duration
167             )
168
169             # 길이 정규화
170             audio = self._normalize_length(audio)
171
172             # 데이터 증강
173             if self.augment:
174                 audio = self._augment_audio(audio)
175
176             # 특징 추출 (Mel-spectrogram + MFCC)
177             features = self._extract_features(audio)
178
179             return {
180                 'features': torch.FloatTensor(features),
181                 'label': torch.LongTensor([self.labels[idx]]),
182                 'path': self.audio_paths[idx]
183             }

```

```

182     }
183
184     except Exception as e:
185         print(f"오디오 로딩 오류 {self.audio_paths[idx]}: {e}")
186         # 빈 특징 반환
187         features = np.zeros((128, 157)) # 기본 특징 크기
188         return {
189             'features': torch.FloatTensor(features),
190             'label': torch.LongTensor([0]),
191             'path': self.audio_paths[idx]
192         }
193
194     def _normalize_length(self, audio):
195         """오디오 길이 정규화"""
196         if len(audio) > self.max_length:
197             # 랜덤 크롭
198             start = np.random.randint(0, len(audio) - self.max_length + 1)
199             audio = audio[start:start + self.max_length]
200         elif len(audio) < self.max_length:
201             # 제로 패딩
202             audio = np.pad(audio, (0, self.max_length - len(audio)))
203         return audio
204
205     def _augment_audio(self, audio):
206         """오디오 데이터 증강"""
207         # 시간 이동
208         if np.random.random() > 0.5:
209             shift = np.random.randint(-len(audio)//8, len(audio)//8)
210             audio = np.roll(audio, shift)
211
212         # 볼륨 조절
213         if np.random.random() > 0.5:
214             volume_factor = np.random.uniform(0.7, 1.3)
215             audio = audio * volume_factor
216
217         # 가우시안 노이즈 추가
218         if np.random.random() > 0.7:
219             noise = np.random.normal(0, 0.005, len(audio))
220             audio = audio + noise
221
222         # 피치 시프트 (가끔)
223         if np.random.random() > 0.8:
224             pitch_shift = np.random.randint(-2, 3)
225             if pitch_shift != 0:
226                 audio = librosa.effects.pitch_shift(audio, sr=self.target_sr, n_steps=pitch_shift)
227
228         return np.clip(audio, -1.0, 1.0)
229
230     def _extract_features(self, audio):
231         """특징 추출: Mel-spectrogram + MFCC"""
232         # Mel-spectrogram
233         mel_spec = librosa.feature.melspectrogram(
234             y=audio,
235             sr=self.target_sr,
236             n_mels=64,
237             fmax=8000,
238             hop_length=512,
239             n_fft=2048
240         )
241         mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)
242
243         # MFCC
244         mfcc = librosa.feature.mfcc(
245             y=audio,
246             sr=self.target_sr,
247             n_mfcc=64,
248             hop_length=512,
249             n_fft=2048
250         )
251
252         # 특징 결합
253         features = np.vstack([mel_spec_db, mfcc]) # (128, time_frames)
254
255         # 고정 크기로 조정
256         target_frames = 157 # 5초 * 16000 / 512 ≈ 157
257         if features.shape[1] != target_frames:
258             features = self._resize_features(features, target_frames)
259
260         return features
261
262     def _resize_features(self, features, target_frames):
263         """특징 크기 조정"""
264         from scipy.ndimage import zoom
265         zoom_factor = target_frames / features.shape[1]
266         return zoom(features, (1, zoom_factor))
267
268 # =====
269 # 4. 3클래스 분류 CNN 모델
270 # =====
271
272 class ThreeClassAudioCNN(nn.Module):
273     def __init__(self, num_classes=3, dropout_rate=0.3):

```

```

274         super(ThreeClassAudioCNN, self).__init__()
275
276         # Convolutional layers
277         self.conv_layers = nn.Sequential(
278             # Block 1
279             nn.Conv2d(1, 32, kernel_size=(3, 3), padding=1),
280             nn.BatchNorm2d(32),
281             nn.ReLU(),
282             nn.MaxPool2d((2, 2)),
283             nn.Dropout2d(0.1),
284
285             # Block 2
286             nn.Conv2d(32, 64, kernel_size=(3, 3), padding=1),
287             nn.BatchNorm2d(64),
288             nn.ReLU(),
289             nn.MaxPool2d((2, 2)),
290             nn.Dropout2d(0.1),
291
292             # Block 3
293             nn.Conv2d(64, 128, kernel_size=(3, 3), padding=1),
294             nn.BatchNorm2d(128),
295             nn.ReLU(),
296             nn.MaxPool2d((2, 2)),
297             nn.Dropout2d(0.2),
298
299             # Block 4
300             nn.Conv2d(128, 256, kernel_size=(3, 3), padding=1),
301             nn.BatchNorm2d(256),
302             nn.ReLU(),
303             nn.MaxPool2d((2, 2)),
304             nn.Dropout2d(0.2),
305         )
306
307         # Global Average Pooling
308         self.global_pool = nn.AdaptiveAvgPool2d((1, 1))
309
310         # Classifier
311         self.classifier = nn.Sequential(
312             nn.Dropout(dropout_rate),
313             nn.Linear(256, 128),
314             nn.ReLU(),
315             nn.Dropout(dropout_rate * 0.5),
316             nn.Linear(128, num_classes)
317         )
318
319     def forward(self, x):
320         # Input: (batch_size, features, time)
321         if len(x.shape) == 3:
322             x = x.unsqueeze(1) # Add channel dimension
323
324         x = self.conv_layers(x)
325         x = self.global_pool(x)
326         x = x.view(x.size(0), -1)
327         x = self.classifier(x)
328         return x
329
330 # =====
331 # 5. 학습 매니지
332 # =====
333
334 class ThreeClassTrainer:
335     def __init__(self, model, train_loader, val_loader, device='auto', lr=0.001):
336         self.device = torch.device('cuda' if torch.cuda.is_available() and device=='auto' else 'cpu')
337         self.model = model.to(self.device)
338         self.train_loader = train_loader
339         self.val_loader = val_loader
340
341         # 최적화 설정
342         self.optimizer = torch.optim.AdamW(
343             model.parameters(),
344             lr=lr,
345             weight_decay=0.01
346         )
347         self.scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
348             self.optimizer,
349             mode='max', # 정확도 기준
350             patience=5,
351             factor=0.5,
352             verbose=True
353         )
354         self.criterion = nn.CrossEntropyLoss()
355
356         # 기록
357         self.train_losses = []
358         self.val_losses = []
359         self.train_accs = []
360         self.val_accs = []
361         self.best_val_acc = 0.0
362         self.best_model_state = None
363
364         print(f"🔴 학습 설정:")
365         print(f"    - 장치: {self.device}")

```



```

366     print(f"    - 학습률: {lr}")
367     print(f"    - 클래스: 3개 (발소리, 말소리, 가구끄는소리)")
368
369     def train_epoch(self):
370         """한 에포크 학습"""
371         self.model.train()
372         running_loss = 0.0
373         correct = 0
374         total = 0
375
376         pbar = tqdm(self.train_loader, desc="학습")
377         for batch in pbar:
378             inputs = batch['features'].to(self.device)
379             labels = batch['label'].squeeze().to(self.device)
380
381             self.optimizer.zero_grad()
382             outputs = self.model(inputs)
383             loss = self.criterion(outputs, labels)
384             loss.backward()
385             self.optimizer.step()
386
387             running_loss += loss.item()
388             _, predicted = outputs.max(1)
389             total += labels.size(0)
390             correct += predicted.eq(labels).sum().item()
391
392         # 진행률 업데이트
393         pbar.set_postfix({
394             'Loss': f'{loss.item():.4f}',
395             'Acc': f'{100.*correct/total:.1f}%'
396         })
397
398     epoch_loss = running_loss / len(self.train_loader)
399     epoch_acc = 100. * correct / total
400
401     return epoch_loss, epoch_acc
402
403     def validate_epoch(self):
404         """검증"""
405         self.model.eval()
406         running_loss = 0.0
407         correct = 0
408         total = 0
409         all_preds = []
410         all_labels = []
411
412         with torch.no_grad():
413             pbar = tqdm(self.val_loader, desc="검증")
414             for batch in pbar:
415                 inputs = batch['features'].to(self.device)
416                 labels = batch['label'].squeeze().to(self.device)
417
418                 outputs = self.model(inputs)
419                 loss = self.criterion(outputs, labels)
420
421                 running_loss += loss.item()
422                 _, predicted = outputs.max(1)
423                 total += labels.size(0)
424                 correct += predicted.eq(labels).sum().item()
425
426         # 혼동행렬용 데이터 수집
427         all_preds.extend(predicted.cpu().numpy())
428         all_labels.extend(labels.cpu().numpy())
429
430         pbar.set_postfix({
431             'Loss': f'{loss.item():.4f}',
432             'Acc': f'{100.*correct/total:.1f}%'
433         })
434
435     epoch_loss = running_loss / len(self.val_loader)
436     epoch_acc = 100. * correct / total
437
438     return epoch_loss, epoch_acc, all_preds, all_labels
439
440     def train(self, num_epochs=30, save_path='best_three_class_model.pth'):
441         """전체 학습"""
442         print("🔥 3클래스 분류 학습 시작!")
443         print("==" * 50)
444
445         for epoch in range(num_epochs):
446             print(f"📅 Epoch {epoch+1}/{num_epochs}")
447
448             # 학습
449             train_loss, train_acc = self.train_epoch()
450             self.train_losses.append(train_loss)
451             self.train_accs.append(train_acc)
452
453             # 검증
454             val_loss, val_acc, val_preds, val_labels = self.validate_epoch()
455             self.val_losses.append(val_loss)
456             self.val_accs.append(val_acc)
457

```

```

458     # 스케줄러 업데이트
459     self.scheduler.step(val_acc)
460
461     print(f"    학습 - Loss: {train_loss:.4f}, Acc: {train_acc:.2f}%")
462     print(f"    검증 - Loss: {val_loss:.4f}, Acc: {val_acc:.2f}%")
463     print(f"    학습률: {self.optimizer.param_groups[0]['lr']:.6f}")
464
465     # 최고 모델 저장
466     if val_acc > self.best_val_acc:
467         self.best_val_acc = val_acc
468         self.best_model_state = self.model.state_dict().copy()
469
470     torch.save({
471         'epoch': epoch,
472         'model_state_dict': self.model.state_dict(),
473         'optimizer_state_dict': self.optimizer.state_dict(),
474         'val_acc': val_acc,
475         'val_loss': val_loss,
476         'class_names': ['footstep', 'speech', 'furniture']
477     }, save_path)
478
479     print(f"✅ 최고 모델 저장! (검증 정확도: {val_acc:.2f}%)")
480
481     # 클래스별 정확도 출력
482     self.print_class_accuracy(val_labels, val_preds)
483
484     # 조기 종료
485     if self.optimizer.param_groups[0]['lr'] < 1e-6:
486         print("🛑 학습률이 너무 낮아 학습을 종료합니다.")
487         break
488
489     # 메모리 정리
490     if epoch % 5 == 0:
491         memory_cleanup()
492
493     print(f"📊 학습 완료!")
494     print(f"    최고 검증 정확도: {self.best_val_acc:.2f}%")
495
496     # 최고 모델 로드
497     if self.best_model_state:
498         self.model.load_state_dict(self.best_model_state)
499
500     return self.model
501
502     def print_class_accuracy(self, true_labels, pred_labels):
503         """클래스별 정확도 출력"""
504         class_names = ['발소리', '말소리', '가구끄는소리']
505
506         for i, class_name in enumerate(class_names):
507             class_mask = np.array(true_labels) == i
508             if class_mask.sum() > 0:
509                 class_acc = (np.array(pred_labels)[class_mask] == i).sum() / class_mask.sum()
510                 print(f"    {class_name}: {class_acc*100:.1f}%")
511
512     def plot_training_history(self):
513         """학습 히스토리 시각화 - 한글 지원"""
514         # 한글 폰트 설정 확인
515         try:
516             plt.rcParams['font.family'] = 'NanumGothic'
517             plt.rcParams['axes.unicode_minus'] = False
518         except:
519             pass
520
521         fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))
522
523         # 손실 그래프
524         epochs = range(1, len(self.train_losses) + 1)
525         ax1.plot(epochs, self.train_losses, 'b-', label='학습 손실', linewidth=2, alpha=0.8)
526         ax1.plot(epochs, self.val_losses, 'r-', label='검증 손실', linewidth=2, alpha=0.8)
527         ax1.set_title('학습/검증 손실', fontsize=14, fontweight='bold', pad=15)
528         ax1.set_xlabel('에포크 (Epoch)', fontsize=12)
529         ax1.set_ylabel('손실 (Loss)', fontsize=12)
530         ax1.legend(fontsize=11)
531         ax1.grid(True, alpha=0.3)
532         ax1.set_xlim(1, len(self.train_losses))
533
534         # 정확도 그래프
535         ax2.plot(epochs, self.train_accs, 'b-', label='학습 정확도', linewidth=2, alpha=0.8)
536         ax2.plot(epochs, self.val_accs, 'r-', label='검증 정확도', linewidth=2, alpha=0.8)
537         ax2.set_title('학습/검증 정확도', fontsize=14, fontweight='bold', pad=15)
538         ax2.set_xlabel('에포크 (Epoch)', fontsize=12)
539         ax2.set_ylabel('정확도 (%)', fontsize=12)
540         ax2.legend(fontsize=11)
541         ax2.grid(True, alpha=0.3)
542         ax2.set_ylim(0, 100)
543         ax2.set_xlim(1, len(self.train_accs))
544
545         # 최고 성능 지점 표시
546         best_epoch = np.argmax(self.val_accs) + 1
547         best_acc = max(self.val_accs)
548         ax2.plot(best_epoch, best_acc, 'ro', markersize=10, markerfacecolor='red',
549                 markeredgewidth=2)

```

```

550 ax2.annotate(f'최고: {best_acc:.1f}%\n(에포크 {best_epoch})',
551 xy=(best_epoch, best_acc), xytext=(10, 10),
552 textcoords='offset points', fontsize=10,
553 bbox=dict(boxstyle='round,pad=0.3', facecolor='yellow', alpha=0.7),
554 arrowprops=dict(arrowstyle='->', connectionstyle='arc3,rad=0'))
555
556 plt.tight_layout()
557 plt.show()
558
559 # 상세 결과 출력
560 print("\n" + "="*60)
561 print("📊 학습 완료 - 최종 결과 요약")
562 print("="*60)
563 print(f"🎯 최고 검증 정확도: {max(self.val_accs):.2f}% (에포크 {np.argmax(self.val_accs)+1})")
564 print(f"📈 최종 학습 정확도: {self.train_accs[-1]:.2f}%")
565 print(f"📉 최종 검증 손실: {self.val_losses[-1]:.4f}")
566 print(f"🔄 총 학습 에포크: {len(self.train_accs)}개")
567
568 # 성능 안정성 분석
569 last_5_accs = self.val_accs[-5:] if len(self.val_accs) >= 5 else self.val_accs
570 stability = np.std(last_5_accs)
571 print(f"📊 최근 5에포크 정확도 안정성: {stability:.2f}% (낮을수록 안정)")
572
573 if stability < 2.0:
574     print("🟢 매우 안정적인 학습!")
575 elif stability < 5.0:
576     print("🟡 안정적인 학습")
577 else:
578     print("🔴 불안정한 학습 - 더 많은 에포크나 조기종료 필요")
579
580 print("="*60)
581
582 # 6. 데이터 처리 및 학습 실행 함수
583 # =====
584 # =====
585
586 def scan_three_class_data(base_path):
587     """3클래스 데이터 스캔 - 강화바전"""
588     audio_files = []
589     labels = []
590
591     # 지원되는 오디오 확장자 (더 많은 형식 추가)
592     extensions = ['.wav', '.mp3', '.flac', '.m4a', '.ogg', '.aac', '.wma', '.aiff', '.au']
593
594     print(f"📁 {base_path}에서 오디오 파일 스캔 중...")
595     print(f"🔍 지원 확장자: {', '.join(extensions)}")
596
597     # 각 클래스별 폴더에서 파일 수집 (더 많은 키워드 추가)
598     class_folders = {
599         'footstep': ['footstep', 'footsteps', 'foot', 'walk', 'walking', 'step',
600                     '발소리', '걸음소리', '발걸음', '보행', 'steps', 'footfall'],
601         'speech': ['speech', 'voice', 'talk', 'talking', 'speaking', '말소리', 'speak',
602                  '음성', '대화', '목소리', 'vocal', 'utterance', 'conversation'],
603         'furniture': ['furniture', 'chair', 'table', 'drag', 'move', '가구', 'scrape',
604                     '끄는소리', '이동', '의자', '테이블', 'moving', 'sliding', 'dragging']
605     }
606
607     # 모든 하위 디렉토리 탐색
608     total_files_found = 0
609     processed_dirs = []
610
611     for root, dirs, files in os.walk(base_path):
612         folder_name = os.path.basename(root).lower()
613         relative_path = os.path.relpath(root, base_path)
614
615         # 오디오 파일이 있는지 확인
616         audio_files_in_dir = [f for f in files if any(f.lower().endswith(ext) for ext in extensions)]
617
618         if audio_files_in_dir:
619             print(f"📁 {relative_path} - {len(audio_files_in_dir)}개 오디오 파일 발견")
620             total_files_found += len(audio_files_in_dir)
621
622             # 폴더명으로 클래스 판단
623             detected_class = None
624             for class_name, keywords in class_folders.items():
625                 if any(keyword in folder_name for keyword in keywords):
626                     detected_class = class_name
627                     break
628
629             # 클래스가 자동 감지되지 않으면 사용자에게 물어보기
630             if detected_class is None:
631                 print(f"❓ '{folder_name}' 폴더의 클래스를 판단할 수 없습니다.")
632                 print(f"👉 다음 중 하나를 선택하세요:")
633                 print(f"1: footstep (발소리)")
634                 print(f"2: speech (말소리)")
635                 print(f"3: furniture (가구끄는소리)")
636                 print(f"0: skip (건너뛰기)")
637
638                 try:
639                     choice = input(f"   선택 (1/2/3/0): ").strip()
640                     class_mapping = {'1': 'footstep', '2': 'speech', '3': 'furniture'}
641                     if choice in class_mapping:

```

```

642 detected_class = class_mapping[choice]
643 print(f"✅ '{folder_name}' -> {detected_class}")
644 else:
645     print(f"❌ '{folder_name}' 폴더 건너뛸")
646     continue
647
648 except:
649     print(f"❗ 입력 오류로 '{folder_name}' 폴더 건너뛸")
650     continue
651
652 else:
653     print(f"✅ {relative_path} -> {detected_class}")
654
655 # 파일 추가
656 for file in audio_files_in_dir:
657     file_path = os.path.join(root, file)
658     audio_files.append(file_path)
659     labels.append(detected_class)
660
661 processed_dirs.append((relative_path, detected_class, len(audio_files_in_dir)))
662
663 # 상세 결과 출력
664 print(f"\n📊 데이터 스캔 완료:")
665 print(f"📈 총 발견된 오디오 파일: {total_files_found}개")
666 print(f"📁 실제 사용할 파일: {len(audio_files)}개")
667 print(f"🗑️ 처리된 디렉토리: {len(processed_dirs)}개")
668
669 # 처리된 디렉토리 상세:
670 for dir_path, class_name, count in processed_dirs:
671     print(f"📁 {dir_path}: {class_name} ({count}개)")
672
673 # 클래스별 파일 수:
674 for class_name in ['footstep', 'speech', 'furniture']:
675     count = labels.count(class_name)
676     percentage = (count / len(labels) * 100) if len(labels) > 0 else 0
677     print(f"📊 {class_name}: {count}개 ({percentage:.1f}%)")
678
679 if len(audio_files) == 0:
680     print("\n❌ 사용 가능한 오디오 파일이 없습니다!")
681     print("👉 다음을 확인해주세요:")
682     print("1. 파일 확장자가 지원되는지 확인")
683     print("2. 폴더 구조가 올바른지 확인")
684     print("3. 파일이 실제로 오디오 파일인지 확인")
685     return [], []
686
687 if total_files_found > len(audio_files):
688     print(f"⚠️ 주의: {total_files_found - len(audio_files)}개 파일이 제외되었습니다.")
689     print("🔍 폴더명이 클래스와 매치되지 않아 제외되었을 수 있습니다.")
690
691 return audio_files, labels
692
693 def detailed_data_analysis(base_path):
694     """데이터 상세 분석"""
695     print(f"🔍 데이터 구조 상세 분석 중...")
696
697     extensions = ['.wav', '.mp3', '.flac', '.m4a', '.ogg', '.aac', '.wma', '.aiff', '.au']
698
699     total_files = 0
700     total_size = 0
701     dir_info = []
702
703     for root, dirs, files in os.walk(base_path):
704         audio_files = []
705         dir_size = 0
706
707         for file in files:
708             file_path = os.path.join(root, file)
709             if any(file.lower().endswith(ext) for ext in extensions):
710                 audio_files.append(file)
711                 try:
712                     file_size = os.path.getsize(file_path)
713                     dir_size += file_size
714                 except:
715                     pass
716
717         if audio_files:
718             relative_path = os.path.relpath(root, base_path)
719             dir_info.append({
720                 'path': relative_path,
721                 'files': len(audio_files),
722                 'size_mb': dir_size / (1024*1024),
723                 'sample_files': audio_files[:3] # 처음 3개 파일명
724             })
725
726         total_files += len(audio_files)
727         total_size += dir_size
728
729     print(f"\n📊 전체 통계:")
730     print(f"📈 총 오디오 파일: {total_files}개")
731     print(f"📁 총 크기: {total_size/(1024*1024):.1f} MB")
732     print(f"📁 오디오가 있는 폴더: {len(dir_info)}개")
733
734     # 폴더별 상세 정보:
735     for info in sorted(dir_info, key=lambda x: x['files'], reverse=True):

```

```
734     print(f"    {info['path']}")
735     print(f"        - 파일 수: {info['files']}개")
736     print(f"        - 크기: {info['size_mb']:.1f} MB")
737     print(f"        - 샘플: {'', '.join(info['sample_files'])}")
738     print()
739
740     return dir_info
741
742 def force_scan_all_audio_files(base_path):
743     """모든 오디오 파일 강제 스캔 (클래스 구분 없이)"""
744     print("🔍 모든 오디오 파일 강제 스캔 중...")
745
746     extensions = ['.wav', '.mp3', '.flac', '.m4a', '.ogg', '.aac', '.wma', '.aiff', '.au']
747     all_files = []
748
749     for root, dirs, files in os.walk(base_path):
750         for file in files:
751             if any(file.lower().endswith(ext) for ext in extensions):
752                 file_path = os.path.join(root, file)
753                 relative_path = os.path.relpath(file_path, base_path)
754                 all_files.append({
755                     'path': file_path,
756                     'relative': relative_path,
757                     'dir': os.path.dirname(relative_path),
758                     'filename': file
759                 })
760
761     print(f"📁 전체 스캔 결과: {len(all_files)}개 오디오 파일 발견")
762
763     # 디렉토리별 그룹화
764     from collections import defaultdict
765     dir_groups = defaultdict(list)
766
767     for file_info in all_files:
768         dir_name = file_info['dir'] if file_info['dir'] else 'root'
769         dir_groups[dir_name].append(file_info)
770
771     print(f"\n📁 디렉토리별 파일 수:")
772     for dir_name, file_list in sorted(dir_groups.items(), key=lambda x: len(x[1]), reverse=True):
773         print(f"    {dir_name}: {len(file_list)}개")
774         # 처음 3개 파일명 표시
775         for i, file_info in enumerate(file_list[:3]):
776             print(f"        - {file_info['filename']}")
777         if len(file_list) > 3:
778             print(f"        ... 그 외 {len(file_list)-3}개")
779         print()
780
781     return all_files, dir_groups
782
783 def run_three_class_training(data_path, num_epochs=30, batch_size=8, test_size=0.2):
784     """3클래스 분류 학습 실행"""
785
786     print("🎯 발소리-말소리-가구끄소리 분류 학습 시작!")
787     print(f"    * {70} ")
788
789     # 환경 설정
790     setup_colab_environment()
791     memory_cleanup()
792
793     # 1. 데이터 스캔
794     audio_files, labels = scan_three_class_data(data_path)
795
796     if len(audio_files) == 0:
797         print("❌ 오디오 파일을 찾을 수 없습니다!")
798         print("📁 데이터 구조를 확인해주세요:")
799         print("    your_dataset/")
800         print("    ├── footsteps/")
801         print("    ├── speech/")
802         print("    └── furniture/")
803         return None
804
805     # 2. 학습/검증 분할
806     train_files, val_files, train_labels, val_labels = train_test_split(
807         audio_files, labels, test_size=test_size, random_state=42, stratify=labels
808     )
809
810     print(f"\n📁 데이터 분할:")
811     print(f"    - 학습: {len(train_files)}개")
812     print(f"    - 검증: {len(val_files)}개")
813
814     # 3. 데이터셋 생성
815     train_dataset = ThreeClassAudioDataset(
816         train_files, train_labels, augment=True
817     )
818     val_dataset = ThreeClassAudioDataset(
819         val_files, val_labels, augment=False
820     )
821
822     # 4. 데이터 로더 생성
823     train_loader = DataLoader(
824         train_dataset, batch_size=batch_size, shuffle=True,
825         num_workers=0, pin_memory=False
```

```
826     )
827     val_loader = DataLoader(
828         val_dataset, batch_size=batch_size, shuffle=False,
829         num_workers=0, pin_memory=False
830     )
831
832     # 5. 모델 생성
833     model = ThreeClassAudioCNN(num_classes=3)
834     print(f"📦 모델 파라미터 수: {sum(p.numel() for p in model.parameters()):.}")
835
836     # 6. 학습 실행
837     trainer = ThreeClassTrainer(model, train_loader, val_loader)
838     trained_model = trainer.train(num_epochs=num_epochs)
839
840     # 7. 결과 시각화
841     trainer.plot_training_history()
842
843     # 8. 혼동행렬 생성
844     plot_confusion_matrix(trained_model, val_loader)
845
846     return trained_model, trainer
847
848 def plot_confusion_matrix(model, val_loader):
849     """혼동행렬 시각화 - 한글 지원"""
850     model.eval()
851     device = next(model.parameters()).device
852
853     all_preds = []
854     all_labels = []
855
856     with torch.no_grad():
857         for batch in val_loader:
858             inputs = batch['features'].to(device)
859             labels = batch['label'].squeeze().to(device)
860
861             outputs = model(inputs)
862             _, predicted = outputs.max(1)
863
864             all_preds.extend(predicted.cpu().numpy())
865             all_labels.extend(labels.cpu().numpy())
866
867     # 혼동행렬 계산
868     cm = confusion_matrix(all_labels, all_preds)
869     class_names = ['발소리', '말소리', '가구끄소리']
870
871     # 시각화
872     plt.figure(figsize=(10, 8))
873
874     # 한글 폰트 확인 및 설정
875     try:
876         plt.rcParams['font.family'] = 'NanumGothic'
877     except:
878         # 폰트 설정이 안된 경우 영어로 대체
879         class_names = ['Footstep', 'Speech', 'Furniture']
880
881     sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
882                 xticklabels=class_names, yticklabels=class_names,
883                 cbar_kws={'label': 'Frequency'})
884     plt.title('혼동 행렬 (Confusion Matrix)', fontsize=16, pad=20)
885     plt.xlabel('예측값 (Predicted)', fontsize=12)
886     plt.ylabel('실제값 (Actual)', fontsize=12)
887     plt.tight_layout()
888     plt.show()
889
890     # 분류 리포트
891     report = classification_report(all_labels, all_preds,
892                                   target_names=class_names, output_dict=True)
893
894     print(f"\n📊 상세 분류 결과:")
895     for i, class_name in enumerate(class_names):
896         precision = report[class_name]['precision']
897         recall = report[class_name]['recall']
898         f1 = report[class_name]['f1-score']
899         support = report[class_name]['support']
900
901         print(f"    {class_name}:")
902         print(f"        - 정밀도(Precision): {precision:.3f}")
903         print(f"        - 재현율(Recall): {recall:.3f}")
904         print(f"        - F1-Score: {f1:.3f}")
905         print(f"        - 샘플 수: {support}")
906
907     print(f"\n    전체 정확도: {report['accuracy']:.3f}")
908     print(f"    매크로 평균 F1: {report['macro avg']['f1-score']:.3f}")
909
910     # 클래스별 정확도 바 차트
911     plt.figure(figsize=(10, 6))
912     class_accuracies = []
913     for i in range(len(class_names)):
914         if cm[i].sum() > 0:
915             acc = cm[i, i] / cm[i].sum()
916             class_accuracies.append(acc)
917         else:
```

```
918 class_accuracies.append(0)
919
920 colors = ['#ff7f7f', '#7f7fff', '#7fff7f']
921 bars = plt.bar(class_names, class_accuracies, color=colors, alpha=0.8)
922 plt.title('클래스별 정확도', fontsize=16, pad=20)
923 plt.xlabel('클래스', fontsize=12)
924 plt.ylabel('정확도', fontsize=12)
925 plt.ylim(0, 1.1)
926
927 # 각 막대 위에 정확도 값 표시
928 for bar, acc in zip(bars, class_accuracies):
929     plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.02,
930              f'{acc:.2%}', ha='center', va='bottom', fontsize=12, fontweight='bold')
931
932 plt.grid(True, axis='y', alpha=0.3)
933 plt.tight_layout()
934 plt.show()
935
936 # =====
937 # 7. 실시간 예측 함수 (완성)
938 # =====
939
940 def predict_audio_file(model_path, audio_file_path):
941     """오디오 파일 예측 - 완성 버전"""
942     # 모델 로드
943     checkpoint = torch.load(model_path, map_location='cpu')
944     model = ThreeClassAudioCNN(num_classes=3)
945     model.load_state_dict(checkpoint['model_state_dict'])
946     model.eval()
947
948     class_names = ['발소리', '말소리', '가구끄는소리']
949     class_names_eng = ['footstep', 'speech', 'furniture']
950
951     try:
952         # 오디오 로드 및 전처리
953         audio, sr = librosa.load(audio_file_path, sr=16000, duration=5.0)
954
955         # 길이 정규화
956         max_length = 16000 * 5
957         if len(audio) < max_length:
958             audio = np.pad(audio, (0, max_length - len(audio)))
959         else:
960             audio = audio[:max_length]
961
962         # 특징 추출
963         mel_spec = librosa.feature.melspectrogram(
964             y=audio, sr=16000, n_mels=64, fmax=8000, hop_length=512, n_fft=2048
965         )
966         mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)
967
968         mfcc = librosa.feature.mfcc(
969             y=audio, sr=16000, n_mfcc=64, hop_length=512, n_fft=2048
970         )
971
972         # 특징 결합
973         features = np.vstack([mel_spec_db, mfcc]) # (128, time_frames)
974
975         # 크기 조정
976         target_frames = 157
977         if features.shape[1] != target_frames:
978             from scipy.ndimage import zoom
979             zoom_factor = target_frames / features.shape[1]
980             features = zoom(features, (1, zoom_factor))
981
982         # 예측
983         with torch.no_grad():
984             features_tensor = torch.FloatTensor(features).unsqueeze(0) # 배치 차원 추가
985             outputs = model(features_tensor)
986             probabilities = F.softmax(outputs, dim=1)
987             predicted_class = torch.argmax(outputs, dim=1).item()
988             confidence = probabilities[0, predicted_class].item()
989
990         # 결과 반환
991         result = {
992             'predicted_class': class_names[predicted_class],
993             'predicted_class_eng': class_names_eng[predicted_class],
994             'confidence': confidence,
995             'probabilities': {
996                 class_names[i]: prob.item()
997                 for i, prob in enumerate(probabilities[0])
998             }
999         }
1000
1001         # 시각화
1002         visualize_prediction_result(audio, sr, features, result)
1003
1004         return result
1005
1006     except Exception as e:
1007         print(f"❌ 예측 오류: {str(e)}")
1008         return None
1009
```

```
1010 def visualize_prediction_result(audio, sr, features, result):
1011     """예측 결과 시각화"""
1012     fig = make_subplots(
1013         rows=2, cols=2,
1014         subplot_titles=('오디오 파형', '특징 맵 (Mel-spec + MFCC)', '예측 확률', '주파수 스펙트럼'),
1015         specs=[[{"secondary_y": False}, {"secondary_y": False}],
1016               [{"type": "bar"}, {"secondary_y": False}]]
1017     )
1018
1019     # 1. 오디오 파형
1020     time = np.linspace(0, len(audio)/sr, len(audio))
1021     fig.add_trace(
1022         go.Scatter(x=time, y=audio, name='오디오 신호', line=dict(color='blue')),
1023         row=1, col=1
1024     )
1025
1026     # 2. 특징 맵
1027     fig.add_trace(
1028         go.Heatmap(z=features, colorscale='Viridis', name='특징 맵'),
1029         row=1, col=2
1030     )
1031
1032     # 3. 예측 확률
1033     classes = list(result['probabilities'].keys())
1034     probs = list(result['probabilities'].values())
1035     colors = ['red' if cls == result['predicted_class'] else 'lightblue' for cls in classes]
1036
1037     fig.add_trace(
1038         go.Bar(x=classes, y=probs, name='예측 확률',
1039               marker=dict(color=colors)),
1040         row=2, col=1
1041     )
1042
1043     # 4. 주파수 스펙트럼
1044     freqs = np.fft.fftfreq(len(audio), 1/sr)[:len(audio)//2]
1045     fft_vals = np.abs(np.fft.fft(audio))[:len(audio)//2]
1046
1047     fig.add_trace(
1048         go.Scatter(x=freqs, y=fft_vals, name='주파수 스펙트럼', line=dict(color='green')),
1049         row=2, col=2
1050     )
1051
1052     fig.update_layout(
1053         height=800,
1054         title_text=f"예측 결과: {result['predicted_class']} (신뢰도: {result['confidence']:.2f})"
1055     )
1056     fig.show()
1057
1058     # 결과 출력
1059     print(f"\n 🎧 예측 결과:")
1060     print(f"   - 예측 클래스: {result['predicted_class']}")
1061     print(f"   - 신뢰도: {result['confidence']:.3f}")
1062     print(f"\n 📊 모든 클래스 확률:")
1063     for class_name, prob in result['probabilities'].items():
1064         print(f"   - {class_name}: {prob:.3f}")
1065
1066 # =====
1067 # 8. 샘플 데이터 생성기
1068 # =====
1069
1070 def generate_sample_audio_data(output_dir='sample_data', samples_per_class=20):
1071     """테스트용 샘플 오디오 데이터 생성"""
1072     print("🔊 샘플 오디오 데이터 생성 중...")
1073
1074     os.makedirs(output_dir, exist_ok=True)
1075
1076     # 각 클래스별 폴더 생성
1077     classes = ['footstep', 'speech', 'furniture']
1078     for class_name in classes:
1079         os.makedirs(os.path.join(output_dir, class_name), exist_ok=True)
1080
1081     sr = 16000
1082     duration = 3.0
1083     t = np.linspace(0, duration, int(sr * duration))
1084
1085     for i in range(samples_per_class):
1086         # 발소리 시뮬레이션 (짧은 충격음들)
1087         footstep = np.zeros_like(t)
1088         for step in range(4): # 4번의 발걸음
1089             start_idx = int(step * len(t) / 4) + np.random.randint(-1000, 1000)
1090             if 0 <= start_idx < len(t) - 1000:
1091                 # 충격을 시뮬레이션
1092                 impact = np.exp(-np.arange(1000) * 0.01) * np.sin(2 * np.pi * np.random.uniform(80, 200) * np.arange(1000) / sr)
1093                 footstep[start_idx:start_idx+1000] += impact * np.random.uniform(0.3, 0.8)
1094
1095         # 노이즈 추가
1096         footstep += np.random.normal(0, 0.02, len(footstep))
1097         footstep = np.clip(footstep, -1.0, 1.0)
1098
1099         sf.write(os.path.join(output_dir, 'footstep', f'footstep_{i:03d}.wav'), footstep, sr)
1100
1101     # 말소리 시뮬레이션 (여러 주파수 성분)
```

```
1102 speech = np.zeros_like(t)
1103 # 기본 음성 주파수들 (100-3000Hz)
1104 for freq in [120, 180, 240, 300]:
1105     amplitude = np.random.uniform(0.1, 0.3)
1106     speech += amplitude * np.sin(2 * np.pi * freq * t)
1107
1108 # 포먼트 시뮬레이션 (800-2000Hz)
1109 for freq in [800, 1200, 1600, 2000]:
1110     amplitude = np.random.uniform(0.05, 0.15)
1111     modulation = 1 + 0.5 * np.sin(2 * np.pi * np.random.uniform(5, 15) * t)
1112     speech += amplitude * np.sin(2 * np.pi * freq * t) * modulation
1113
1114 # 노이즈 및 변조
1115 speech *= (1 + 0.3 * np.sin(2 * np.pi * np.random.uniform(1, 5) * t))
1116 speech += np.random.normal(0, 0.02, len(speech))
1117 speech = np.clip(speech, -1.0, 1.0)
1118
1119 sf.write(os.path.join(output_dir, 'speech', f'speech_{i:03d}.wav'), speech, sr)
1120
1121 # 가구끄는소리 시뮬레이션 (마찰음)
1122 furniture = np.zeros_like(t)
1123
1124 # 마찰음 기본 주파수 (낮은 주파수 + 고주파 노이즈)
1125 base_freq = np.random.uniform(20, 80)
1126 furniture += 0.4 * np.sin(2 * np.pi * base_freq * t)
1127
1128 # 고주파 마찰음 (1-4kHz)
1129 high_freq_noise = np.random.normal(0, 0.1, len(t))
1130 butter_b, butter_a = scipy.signal.butter(4, [1000, 4000], btype='band', fs=sr)
1131 high_freq_filtered = scipy.signal.filtfilt(butter_b, butter_a, high_freq_noise)
1132 furniture += 0.3 * high_freq_filtered
1133
1134 # 불규칙한 진동 패턴
1135 irregular_pattern = np.random.uniform(0.5, 1.5, 100)
1136 furniture *= np.interp(t, np.linspace(0, duration, 100), irregular_pattern)
1137
1138 furniture = np.clip(furniture, -1.0, 1.0)
1139
1140 sf.write(os.path.join(output_dir, 'furniture', f'furniture_{i:03d}.wav'), furniture, sr)
1141
1142 print(f"✅ 샘플 데이터 생성 완료!")
1143 print(f" - 경로: {output_dir}")
1144 print(f" - 클래스별 {samples_per_class}개씩 총 {samples_per_class * 3}개 파일")
1145
1146 return output_dir
1147
1148 # =====
1149 # 9. 실시간 녹음 및 예측
1150 # =====
1151
1152 def record_and_predict(model_path, duration=5):
1153     """실시간 녹음 및 예측"""
1154     print(f"🎧 {duration}초간 녹음을 시작합니다...")
1155     print(" - 마이크에 대고 소리를 내세요!")
1156
1157     try:
1158         # JavaScript를 사용한 녹음 (Colab 환경)
1159         from google.colab import output
1160         from base64 import b64decode
1161
1162         RECORD = ""
1163         const sleep = time => new Promise(resolve => setTimeout(resolve, time))
1164         const b2text = blob => new Promise(resolve => {
1165             const reader = new FileReader()
1166             reader.onloadend = e => resolve(e.srcElement.result)
1167             reader.readAsDataURL(blob)
1168         })
1169
1170         var record = time => new Promise(async resolve => {
1171             stream = await navigator.mediaDevices.getUserMedia({ audio: true })
1172             recorder = new MediaRecorder(stream)
1173             chunks = []
1174             recorder.ondataavailable = e => chunks.push(e.data)
1175             recorder.onstop = async ()=>{
1176                 blob = new Blob(chunks, { type: 'audio/wav' })
1177                 text = await b2text(blob)
1178                 resolve(text)
1179             }
1180             recorder.start()
1181             await sleep(time)
1182             recorder.stop()
1183         })
1184         ""
1185
1186         display(HTML(f`
1187 <script>
1188 {RECORD}
1189 </script>
1190 <button onclick="record({duration} * 1000).then(audio => {{
1191     google.colab.kernel.invokeFunction('save_audio', [audio], {{{}}})
1192 }})">🎧 녹음 시작 ({duration}초)</button>
1193 `))
1194     
```

```
1194
1195     print(" 위키 녹음 버튼을 클릭해주세요!")
1196
1197 except Exception as e:
1198     print(f"❌ 녹음 가능 오류: {e}")
1199     print(" 대신 파일 업로드를 사용해주세요.")
1200
1201 def save_audio(audio_data):
1202     """녹음된 오디오 저장 및 예측"""
1203     try:
1204         # Base64 디코딩
1205         audio_data = audio_data.split(',')[1]
1206         audio_bytes = b64decode(audio_data)
1207
1208         # 파일 저장
1209         with open('recorded_audio.wav', 'wb') as f:
1210             f.write(audio_bytes)
1211
1212         print("✅ 녹음 완료! 예측 중...")
1213
1214         # 예측 실행 (모델이 있다면)
1215         if os.path.exists('best_three_class_model.pth'):
1216             result = predict_audio_file('best_three_class_model.pth', 'recorded_audio.wav')
1217             if result:
1218                 print_prediction_result(result)
1219         else:
1220             print("⚠️ 학습된 모델이 없습니다. 먼저 학습을 실행해주세요.")
1221
1222 except Exception as e:
1223     print(f"❌ 오디오 저장 오류: {e}")
1224
1225 def print_prediction_result(result):
1226     """예측 결과 출력"""
1227     print("\n" + "="*50)
1228     print("🕒 실시간 예측 결과")
1229     print("="*50)
1230     print(f"🔍 예측 클래스: {result['predicted_class']}")
1231     print(f"📊 신뢰도: {result['confidence']:.1%}")
1232     print("\n📋 각 클래스별 확률:")
1233
1234     for class_name, prob in result['probabilities'].items():
1235         bar = "█" * int(prob * 20)
1236         print(f"    {class_name:12}: {prob:.3f} |{bar}")
1237
1238 # =====
1239 # 10. 업로드 파일 분석
1240 # =====
1241
1242 def analyze_uploaded_file():
1243     """파일 업로드 및 분석"""
1244     print("📁 오디오 파일을 업로드해주세요...")
1245     uploaded = files.upload()
1246
1247     if uploaded:
1248         file_name = list(uploaded.keys())[0]
1249         print(f"✅ 파일 '{file_name}' 업로드 완료!")
1250
1251         # 모델이 있는지 확인
1252         if os.path.exists('best_three_class_model.pth'):
1253             result = predict_audio_file('best_three_class_model.pth', file_name)
1254             if result:
1255                 print_prediction_result(result)
1256
1257         # 오디오 재생
1258         audio, sr = librosa.load(file_name, sr=16000)
1259         display(Audio(audio, rate=sr))
1260
1261         return result
1262     else:
1263         print("⚠️ 학습된 모델이 없습니다.")
1264         print(" 먼저 샘플 데이터로 학습하거나 실제 데이터로 학습해주세요.")
1265         return None
1266
1267 else:
1268     print(f"❌ 파일이 업로드되지 않았습니다.")
1269     return None
1270
1271 # =====
1272 # 11. 데모 및 테스트 함수들
1273 # =====
1274
1275 def debug_data_loading(data_path):
1276     """데이터 로딩 과정 디버깅"""
1277     print("🔍 데이터 로딩 과정 디버깅 중...")
1278
1279     # 1. 전체 파일 스캔
1280     all_files, dir_groups = force_scan_all_audio_files(data_path)
1281     print(f"📁 전체 스캔: {len(all_files)}개 파일 발견")
1282
1283     # 2. 클래스 매칭 테스트
1284     audio_files, labels = scan_three_class_data(data_path)
1285     print(f"🔍 클래스 매칭: {len(audio_files)}개 파일 매칭")
1286 
```

```

1286 # 3. 누락된 파일들 분석
1287 matched_paths = set(audio_files)
1288 all_paths = set([f['path'] for f in all_files])
1289 missing_files = all_paths - matched_paths
1290
1291 if missing_files:
1292     print(f"\n! 누락된 파일들 ({len(missing_files)}개):")
1293     missing_by_dir = defaultdict(list)
1294     for missing_path in list(missing_files)[:20]: # 처음 20개만 표시
1295         dir_name = os.path.dirname(os.path.relpath(missing_path, data_path))
1296         missing_by_dir[dir_name].append(os.path.basename(missing_path))
1297
1298     for dir_name, files in missing_by_dir.items():
1299         print(f" 📁 {dir_name}: {len(files)}개")
1300         for file in files[:3]:
1301             print(f"   - {file}")
1302         if len(files) > 3:
1303             print(f"     ... 그 외 {len(files)-3}개")
1304
1305 # 4. 실제 로딩 테스트
1306 print(f"\n👉 실제 오디오 로딩 테스트 (처음 10개 파일)...")
1307 loading_errors = 0
1308
1309 for i, file_path in enumerate(audio_files[:10]):
1310     try:
1311         audio, sr = librosa.load(file_path, sr=16000, duration=1.0) # 1초만 테스트
1312         duration = len(audio) / sr
1313         print(f" 🟢 {i+1}: {os.path.basename(file_path)} ((duration:.1f)초, {sr}Hz)")
1314     except Exception as e:
1315         print(f" 🛑 {i+1}: {os.path.basename(file_path)} - {str(e)}")
1316         loading_errors += 1
1317
1318 if loading_errors > 0:
1319     print(f"\n⚠️ {loading_errors}개 파일에서 로딩 오류 발생")
1320     print(" 일부 파일이 손상되었거나 지원되지 않는 형식일 수 있습니다.")
1321
1322 return {
1323     'total_found': len(all_files),
1324     'matched': len(audio_files),
1325     'missing': len(missing_files),
1326     'loading_errors': loading_errors,
1327     'dir_groups': dir_groups
1328 }
1329
1330 def create_manual_dataset(data_path):
1331     """수동으로 데이터셋 생성 (모든 파일 사용)"""
1332     print("🔧 수동 데이터셋 생성 중...")
1333
1334     # 모든 파일 스캔
1335     all_files, dir_groups = force_scan_all_audio_files(data_path)
1336
1337     audio_files = []
1338     labels = []
1339
1340     print(f"\n각 디렉토리의 클래스를 수동으로 지정해주세요:")
1341     print("1: footstep (발소리)")
1342     print("2: speech (말소리)")
1343     print("3: furniture (가구끄는소리)")
1344     print("0: skip (제외)")
1345
1346     class_mapping = {'1': 'footstep', '2': 'speech', '3': 'furniture'}
1347
1348     for dir_name, file_list in sorted(dir_groups.items(), key=lambda x: len(x[1]), reverse=True):
1349         print(f"\n📁 {dir_name} ({len(file_list)}개 파일)")
1350         print(f"   샘플 파일: {', '.join([f['filename'] for f in file_list[:3]])}")
1351
1352         while True:
1353             try:
1354                 choice = input(f"   클래스 선택 (1/2/3/0: ").strip()
1355                 if choice == '0':
1356                     print(f"   🚫 {dir_name} 폴더 제외")
1357                     break
1358                 elif choice in class_mapping:
1359                     selected_class = class_mapping[choice]
1360                     print(f"   🟢 {dir_name} -> {selected_class}")
1361
1362                     # 파일들 추가
1363                     for file_info in file_list:
1364                         audio_files.append(file_info['path'])
1365                         labels.append(selected_class)
1366                     break
1367             except KeyboardInterrupt:
1368                 print(" 잘못된 선택입니다. 다시 입력해주세요.")
1369         except KeyboardInterrupt:
1370             print("\n 작업이 중단되었습니다.")
1371             return [], []
1372
1373     print(f"\n🟢 수동 데이터셋 생성 완료:")
1374     print(f"   - 총 파일 수: {len(audio_files)}개")
1375     for class_name in ['footstep', 'speech', 'furniture']:
1376         count = labels.count(class_name)
1377         percentage = (count / len(labels) * 100) if len(labels) > 0 else 0

```

```

1378     print(f"   - {class_name}: {count}개 ({percentage:.1f}%)")
1379
1380     return audio_files, labels
1381     """샘플 데이터 또는 실제 데이터로 학습 실행"""
1382
1383     if data_path is None:
1384         print("👉 샘플 데이터를 생성하고 학습을 시작합니다...")
1385         # 샘플 데이터 생성
1386         sample_dir = generate_sample_audio_data('sample_data', samples_per_class=30)
1387         data_path = sample_dir
1388         print("🟢 샘플 데이터 생성 완료!")
1389     else:
1390         print(f"📁 실제 데이터를 사용합니다: {data_path}")
1391
1392     # 학습 실행
1393     print("👉 학습을 시작합니다...")
1394     model, trainer = run_three_class_training(
1395         data_path=data_path,
1396         num_epochs=num_epochs,
1397         batch_size=batch_size,
1398         test_size=0.2
1399     )
1400
1401     print("🟢 학습 완료! 이제 테스트해보세요.")
1402     return model, trainer
1403
1404 def test_model_with_samples():
1405     """샘플로 모델 테스트"""
1406     if not os.path.exists('best_three_class_model.pth'):
1407         print("🛑 학습된 모델이 없습니다!")
1408         print("   run_sample_training() 먼저 실행해주세요.")
1409         return
1410
1411     if not os.path.exists('sample_data'):
1412         print("🛑 샘플 데이터가 없습니다!")
1413         return
1414
1415     print("👉 샘플 파일들로 모델 테스트 중...")
1416
1417     # 각 클래스에서 랜덤 파일 선택
1418     classes = ['footstep', 'speech', 'furniture']
1419
1420     for class_name in classes:
1421         class_dir = os.path.join('sample_data', class_name)
1422         if os.path.exists(class_dir):
1423             files_list = [f for f in os.listdir(class_dir) if f.endswith('.wav')]
1424             if files_list:
1425                 test_file = os.path.join(class_dir, random.choice(files_list))
1426                 print(f"\n📁 테스트 중: {class_name} 클래스")
1427                 print(f"   파일: {test_file}")
1428
1429                 result = predict_audio_file('best_three_class_model.pth', test_file)
1430                 if result:
1431                     correct = "🟢" if result['predicted_class_eng'] == class_name else "🛑"
1432                     print(f"   {correct} 예측: {result['predicted_class']} (신뢰도: {result['confidence']:.2f})")
1433
1434                 # 오디오 재생
1435                 audio, sr = librosa.load(test_file, sr=16000)
1436                 display(Audio(audio, rate=sr))
1437
1438 def demo_realtime_features():
1439     """실시간 특징 추출 데모"""
1440     print("👉 실시간 특징 추출 데모")
1441
1442     # 짧은 테스트 신호 생성
1443     sr = 16000
1444     duration = 2.0
1445     t = np.linspace(0, duration, int(sr * duration))
1446
1447     # 다양한 신호 생성
1448     signals = {
1449         '발소리 시뮬레이션': create_footstep_signal(t, sr),
1450         '말소리 시뮬레이션': create_speech_signal(t, sr),
1451         '가구 시뮬레이션': create_furniture_signal(t, sr)
1452     }
1453
1454     for name, signal in signals.items():
1455         print(f"\n📁 {name} 특징 추출 중...")
1456
1457         # 특징 추출
1458         mel_spec = librosa.feature.melspectrogram(y=signal, sr=sr, n_mels=64)
1459         mfcc = librosa.feature.mfcc(y=signal, sr=sr, n_mfcc=13)
1460
1461         # 시각화
1462         fig, axes = plt.subplots(1, 3, figsize=(15, 4))
1463
1464         # 원본 신호
1465         axes[0].plot(t, signal)
1466         axes[0].set_title(f'{name} - 시간 도메인')
1467         axes[0].set_xlabel('시간 (초)')
1468         axes[0].set_ylabel('진폭')
1469

```

```
1470 # Mel-spectrogram
1471 librosa.display.specshow(librosa.power_to_db(mel_spec), sr=sr, x_axis='time', y_axis='mel', ax=axes[1])
1472 axes[1].set_title('Mel-spectrogram')
1473
1474 # MFCC
1475 librosa.display.specshow(mfcc, sr=sr, x_axis='time', ax=axes[2])
1476 axes[2].set_title('MFCC')
1477
1478 plt.tight_layout()
1479 plt.show()
1480
1481 # 오디오 재생
1482 display(Audio(signal, rate=sr))
1483
1484 def create_footstep_signal(t, sr):
1485     """발소리 신호 생성"""
1486     signal = np.zeros_like(t)
1487     step_times = [0.3, 0.9, 1.5] # 발걸음 시간
1488
1489     for step_time in step_times:
1490         start_idx = int(step_time * sr)
1491         if start_idx < len(signal) - 2000:
1492             # 충격을 (감시하는 저주파)
1493             impact_t = np.arange(2000) / sr
1494             impact = np.exp(-impact_t * 5) * np.sin(2 * np.pi * 80 * impact_t)
1495             signal[start_idx:start_idx+2000] += impact * 0.8
1496
1497     return signal
1498
1499 def create_speech_signal(t, sr):
1500     """말소리 신호 생성"""
1501     # 기본 주파수 (피치)
1502     f0 = 150 # Hz
1503     speech = 0.3 * np.sin(2 * np.pi * f0 * t)
1504
1505     # 포먼트 추가
1506     formants = [800, 1200, 2400]
1507     for formant in formants:
1508         speech += 0.1 * np.sin(2 * np.pi * formant * t)
1509
1510     # 진폭 변조 (말하는 리듬)
1511     modulation = 1 + 0.5 * np.sin(2 * np.pi * 3 * t)
1512     speech *= modulation
1513
1514     return speech
1515
1516 def create_furniture_signal(t, sr):
1517     """가구끄는소리 신호 생성"""
1518     # 마찰음 (광대역 노이즈를 필터링)
1519     noise = np.random.normal(0, 1, len(t))
1520
1521     # 로우패스 필터 (마찰음 특성)
1522     butter_b, butter_a = scipy.signal.butter(4, 500, fs=sr)
1523     filtered = scipy.signal.filtfilt(butter_b, butter_a, noise)
1524
1525     # 불규칙한 진폭
1526     amplitude_env = np.random.uniform(0.2, 0.8, 50)
1527     amplitude = np.interp(t, np.linspace(0, t[-1], 50), amplitude_env)
1528
1529     return filtered * amplitude * 0.5
1530
1531 # =====
1532 # 12. 사용 가이드 및 실행 함수
1533 # =====
1534
1535 def show_complete_usage_guide():
1536     """완전한 사용 가이드"""
1537     print("""
1538 📖 발소리-말소리-가구끄는소리 분류 AI 모델 사용 가이드
1539 =====
1540
1541 📌 주요 기능:
1542 1. 3클래스 오디오 분류 (발소리, 말소리, 가구끄는소리)
1543 2. CNN 기반 딥러닝 모델
1544 3. 실시간 예측 및 시각화
1545 4. 데이터 증강 및 최적화
1546 5. 한글 폰트 지원
1547
1548 🚀 빠른 시작:
1549
1550 1️⃣ 한글 폰트 테스트:
1551     test_korean_font() # 한글이 제대로 표시되는지 확인
1552
1553 2️⃣ 샘플 데이터로 빠른 테스트:
1554     run_sample_training() # 샘플 생성 + 학습
1555     test_model_with_samples() # 테스트
1556
1557 3️⃣ 실제 데이터로 학습:
1558     # 데이터 폴더 구조:
1559     # └─ your_data/
1560     #   └─ footstep/ (발소리 파일들)
1561     #   └─ speech/ (말소리 파일들)
```

```
1562 # └─ furniture/ (가구끄는소리 파일들)
1563
1564 model, trainer = run_sample_training('/path/to/your_data')
1565
1566 1️⃣ 파일 업로드하여 예측:
1567     analyze_uploaded_file()
1568
1569 2️⃣ 실시간 녹음 예측:
1570     record_and_predict('best_three_class_model.pth')
1571
1572 3️⃣ 특징 추출 데모:
1573     demo_realtime_features()
1574
1575 4️⃣ 모델 성능 상세 분석:
1576     analyze_model_performance(model, val_loader)
1577
1578 📊 모델 성능:
1579 - 입력: 5초 오디오 (16kHz)
1580 - 특징: Mel-spectrogram (64) + MFCC (64) = 128차원
1581 - 구조: CNN (4블록) + Global Average Pooling
1582 - 출력: 3클래스 확률 분포
1583
1584 📄 한글 폰트 문제 해결:
1585 - 그래프에서 한글이 깨진다면: test_korean_font() 실행
1586 - 여전히 문제가 있다면: 런타임 재시작 후 다시 실행
1587
1588 💡 팁:
1589 - GPU 사용 시 batch_size를 16으로 증가 가능
1590 - 데이터가 부족하면 augmentation 강화
1591 - 과적합 시 dropout_rate 증가
1592
1593 ⚡ 지금 시작하기:
1594     test_korean_font() # 한글 폰트 확인
1595     run_sample_training() # 샘플 학습
1596     """
1597
1598 # 컴포넌트별 등록 (Colab 전용)
1599 try:
1600     from google.colab import output
1601     output.register_callback('save_audio', save_audio)
1602 except:
1603     pass
1604
1605 # =====
1606 # 13. 고급 분석 도구
1607 # =====
1608
1609 def analyze_model_performance(model, val_loader):
1610     """모델 성능 상세 분석 - 한글 지원"""
1611     model.eval()
1612     device = next(model.parameters()).device
1613
1614     all_preds = []
1615     all_labels = []
1616     all_confidences = []
1617
1618     print("🔍 모델 성능 상세 분석 중...")
1619
1620     with torch.no_grad():
1621         for batch in tqdm(val_loader, desc="분석"):
1622             inputs = batch['features'].to(device)
1623             labels = batch['label'].squeeze().to(device)
1624
1625             outputs = model(inputs)
1626             probabilities = F.softmax(outputs, dim=1)
1627             predicted = torch.argmax(outputs, dim=1)
1628             confidence = torch.max(probabilities, dim=1)[0]
1629
1630             all_preds.extend(predicted.cpu().numpy())
1631             all_labels.extend(labels.cpu().numpy())
1632             all_confidences.extend(confidence.cpu().numpy())
1633
1634     # 성능 메트릭 계산
1635     accuracy = accuracy_score(all_labels, all_preds)
1636     precision, recall, f1, _ = precision_recall_fscore_support(
1637         all_labels, all_preds, average='weighted'
1638     )
1639
1640     # 신뢰도 분석
1641     correct_mask = np.array(all_preds) == np.array(all_labels)
1642     correct_confidences = np.array(all_confidences)[correct_mask]
1643     incorrect_confidences = np.array(all_confidences)[~correct_mask]
1644
1645     # 결과 출력
1646     print(f"📊 전체 성능 메트릭:")
1647     print(f"  - 정확도: {accuracy:.3f}")
1648     print(f"  - 정밀도: {precision:.3f}")
1649     print(f"  - 재현율: {recall:.3f}")
1650     print(f"  - F1 점수: {f1:.3f}")
1651     print(f"🔍 신뢰도 분석:")
1652     print(f"  - 올바른 예측 평균 신뢰도: {correct_confidences.mean():.3f}")
1653     print(f"  - 잘못된 예측 평균 신뢰도: {incorrect_confidences.mean():.3f}")
```



```

1654
1655 # 한글 폰트 설정
1656 try:
1657     plt.rcParams['font.family'] = 'NanumGothic'
1658     plt.rcParams['axes.unicode_minus'] = False
1659 except:
1660     pass
1661
1662 # 신뢰도 분포 시각화
1663 plt.figure(figsize=(12, 8))
1664
1665 # 서브플롯 1: 신뢰도 히스토그램
1666 plt.subplot(2, 2, 1)
1667 plt.hist(correct_confidences, bins=20, alpha=0.7, label='올바른 예측', color='green', density=True)
1668 plt.hist(incorrect_confidences, bins=20, alpha=0.7, label='잘못된 예측', color='red', density=True)
1669 plt.xlabel('신뢰도')
1670 plt.ylabel('밀도')
1671 plt.title('예측 신뢰도 분포')
1672 plt.legend()
1673 plt.grid(True, alpha=0.3)
1674
1675 # 서브플롯 2: 박스플롯
1676 plt.subplot(2, 2, 2)
1677 data_to_plot = [correct_confidences, incorrect_confidences]
1678 box = plt.boxplot(data_to_plot, labels=['올바른 예측', '잘못된 예측'], patch_artist=True)
1679 box['boxes'][0].set_facecolor('lightgreen')
1680 box['boxes'][1].set_facecolor('lightcoral')
1681 plt.ylabel('신뢰도')
1682 plt.title('신뢰도 박스플롯')
1683 plt.grid(True, alpha=0.3)
1684
1685 # 서브플롯 3: 클래스별 성능
1686 plt.subplot(2, 2, 3)
1687 class_names = ['발소리', '말소리', '가구끄는소리']
1688 class_f1_scores = []
1689
1690 for i in range(3):
1691     class_mask = np.array(all_labels) == i
1692     if class_mask.sum() > 0:
1693         class_preds = np.array(all_preds)[class_mask]
1694         class_labels = np.array(all_labels)[class_mask]
1695         class_f1 = f1_score(class_labels, class_preds, average='binary', pos_label=i, zero_division=0)
1696         class_f1_scores.append(class_f1)
1697     else:
1698         class_f1_scores.append(0)
1699
1700 colors = ['#ff9999', '#66b3ff', '#99ff99']
1701 bars = plt.bar(class_names, class_f1_scores, color=colors, alpha=0.8)
1702 plt.ylabel('F1 점수')
1703 plt.title('클래스별 F1 점수')
1704 plt.ylim(0, 1.1)
1705
1706 # 각 막대 위에 값 표시
1707 for bar, score in zip(bars, class_f1_scores):
1708     plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.02,
1709              f'{score:.3f}', ha='center', va='bottom')
1710
1711 # 서브플롯 4: 신뢰도 vs 정확도
1712 plt.subplot(2, 2, 4)
1713 confidence_bins = np.linspace(0, 1, 11)
1714 bin_accuracies = []
1715 bin_centers = []
1716
1717 for i in range(len(confidence_bins) - 1):
1718     bin_mask = ((np.array(all_confidences) >= confidence_bins[i]) &
1719                (np.array(all_confidences) < confidence_bins[i+1]))
1720
1721     if bin_mask.sum() > 0:
1722         bin_accuracy = (np.array(all_preds)[bin_mask] == np.array(all_labels)[bin_mask]).mean()
1723         bin_accuracies.append(bin_accuracy)
1724         bin_centers.append((confidence_bins[i] + confidence_bins[i+1]) / 2)
1725
1726 if bin_centers:
1727     plt.plot(bin_centers, bin_accuracies, 'bo-', linewidth=2, markersize=6)
1728     plt.plot([0, 1], [0, 1], 'r--', alpha=0.7, label='완벽한 보정')
1729     plt.xlabel('신뢰도')
1730     plt.ylabel('정확도')
1731     plt.title('신뢰도 보정 곡선')
1732     plt.legend()
1733     plt.grid(True, alpha=0.3)
1734
1735 plt.tight_layout()
1736 plt.show()
1737
1738 # 상세 통계
1739 print(f"\n 상세 통계:")
1740 print(f" - 총 예측 샘플: {len(all_labels)}개")
1741 print(f" - 올바른 예측: {correct_mask.sum()}개 ({correct_mask.mean()*100:.1f}%)")
1742 print(f" - 잘못된 예측: {(~correct_mask).sum()}개 ({(~correct_mask).mean()*100:.1f}%)")
1743 print(f" - 평균 신뢰도: {np.mean(all_confidences):.3f}")
1744 print(f" - 신뢰도 표준편차: {np.std(all_confidences):.3f}")
1745

```

```

1746     return {
1747         'accuracy': accuracy,
1748         'precision': precision,
1749         'recall': recall,
1750         'f1': f1,
1751         'correct_confidences': correct_confidences,
1752         'incorrect_confidences': incorrect_confidences,
1753         'class_f1_scores': class_f1_scores
1754     }
1755
1756 def batch_predict_directory(model_path, test_dir):
1757     """디렉토리 내 모든 파일 일괄 예측"""
1758     if not os.path.exists(model_path):
1759         print("❌ 모델 파일이 없습니다!")
1760         return
1761
1762     print(f"📁 {test_dir} 내 모든 오디오 파일 예측 중...")
1763
1764     audio_extensions = ['.wav', '.mp3', '.flac', '.m4a']
1765     audio_files = []
1766
1767     for root, dirs, files in os.walk(test_dir):
1768         for file in files:
1769             if any(file.lower().endswith(ext) for ext in audio_extensions):
1770                 audio_files.append(os.path.join(root, file))
1771
1772     if not audio_files:
1773         print("❌ 오디오 파일을 찾을 수 없습니다!")
1774         return
1775
1776     results = []
1777
1778     for audio_file in tqdm(audio_files, desc="예측"):
1779         result = predict_audio_file(model_path, audio_file)
1780         if result:
1781             results.append({
1782                 'file': os.path.basename(audio_file),
1783                 'predicted_class': result['predicted_class'],
1784                 'confidence': result['confidence'],
1785                 'path': audio_file
1786             })
1787
1788     # 결과를 DataFrame으로 정리
1789     df = pd.DataFrame(results)
1790
1791     print(f"\n📊 일괄 예측 결과:")
1792     print(df.groupby('predicted_class').agg({
1793         'confidence': ['count', 'mean', 'min', 'max'],
1794         'file': 'count'
1795     })).round(3))
1796
1797     return df
1798
1799 # =====
1800 # 14. 메인 실행 부분
1801 # =====
1802
1803 def main():
1804     """메인 실행 함수"""
1805     print("🔊 발소리-말소리-가구끄는소리 분류 AI 모델")
1806     print("m" * 70)
1807
1808     # 환경 설정
1809     setup_colab_environment()
1810
1811     # 사용 가이드 출력
1812     show_complete_usage_guide()
1813
1814     print("\n👉 다음 중 하나를 선택하세요:")
1815     print("1. test_korean_font() # 한글 폰트 테스트")
1816     print("2. run_sample_training() # 샘플 데이터로 학습")
1817     print("3. analyze_uploaded_file() # 파일 업로드 분석")
1818     print("4. demo_realtime_features() # 특징 추출 데모")
1819     print("5. test_model_with_samples() # 샘플로 테스트")
1820
1821     # 자동 실행
1822     if __name__ == "__main__":
1823         main()
1824
1825 # =====
1826 # 16. 오디오 데이터 시각화 및 분석 도구
1827 # =====
1828
1829 def visualize_audio_prediction(model_path, audio_file_path, save_image=False, show_features=True):
1830     """개별 오디오 파일의 예측 과정을 상세히 시각화"""
1831
1832     if not os.path.exists(model_path):
1833         print("❌ 모델 파일이 없습니다!")
1834         return None
1835
1836     # 모델 로드
1837     checkpoint = torch.load(model_path, map_location='cpu')

```

```
1838 model = ThreeClassAudioCNN(num_classes=3)
1839 model.load_state_dict(checkpoint['model_state_dict'])
1840 model.eval()
1841
1842 class_names_kr = ['발소리', '말소리', '가구끄는소리']
1843 class_names_en = ['footstep', 'speech', 'furniture']
1844 colors = ['#ff9999', '#66b3ff', '#99ff99']
1845
1846 try:
1847     # 오디오 로드
1848     audio, sr = librosa.load(audio_file_path, sr=16000, duration=5.0)
1849     filename = os.path.basename(audio_file_path)
1850
1851     # 길이 정규화
1852     max_length = 16000 * 5
1853     if len(audio) < max_length:
1854         audio = np.pad(audio, (0, max_length - len(audio)))
1855     else:
1856         audio = audio[:max_length]
1857
1858     # 특징 추출
1859     mel_spec = librosa.feature.melspectrogram(
1860         y=audio, sr=16000, n_mels=64, fmax=8000, hop_length=512, n_fft=2048
1861     )
1862     mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)
1863
1864     mfcc = librosa.feature.mfcc(
1865         y=audio, sr=16000, n_mfcc=64, hop_length=512, n_fft=2048
1866     )
1867
1868     # 특징 결합
1869     features = np.vstack([mel_spec_db, mfcc])
1870
1871     # 크기 조정
1872     target_frames = 157
1873     if features.shape[1] != target_frames:
1874         from scipy.ndimage import zoom
1875         zoom_factor = target_frames / features.shape[1]
1876         features = zoom(features, (1, zoom_factor))
1877
1878     # 예측
1879     with torch.no_grad():
1880         features_tensor = torch.FloatTensor(features).unsqueeze(0)
1881         outputs = model(features_tensor)
1882         probabilities = F.softmax(outputs, dim=1)
1883         predicted_class = torch.argmax(outputs, dim=1).item()
1884         confidence = probabilities[0, predicted_class].item()
1885
1886     # 시각화 생성
1887     if show_features:
1888         fig = plt.figure(figsize=(20, 16))
1889         gs = fig.add_gridspec(4, 3, hspace=0.3, wspace=0.3)
1890     else:
1891         fig = plt.figure(figsize=(16, 12))
1892         gs = fig.add_gridspec(3, 3, hspace=0.3, wspace=0.3)
1893
1894     # 한글 폰트 설정
1895     try:
1896         plt.rcParams['font.family'] = 'NanumGothic'
1897         plt.rcParams['axes.unicode_minus'] = False
1898     except:
1899         pass
1900
1901     # 1. 오디오 파형
1902     ax1 = fig.add_subplot(gs[0, :])
1903     time = np.linspace(0, len(audio)/sr, len(audio))
1904     ax1.plot(time, audio, color='blue', alpha=0.8, linewidth=0.5)
1905     ax1.set_title(f'🔊 원본 오디오 파형: {filename}', fontsize=14, fontweight='bold', pad=15)
1906     ax1.set_xlabel('시간 (초)')
1907     ax1.set_ylabel('진폭')
1908     ax1.grid(True, alpha=0.3)
1909     ax1.set_xlim(0, 5)
1910
1911     # RMS와 피크값 표시
1912     rms = np.sqrt(np.mean(audio**2))
1913     peak = np.max(np.abs(audio))
1914     ax1.text(0.02, 0.95, f'RMS: {rms:.4f}\nPeak: {peak:.4f}',
1915             transform=ax1.transAxes, verticalalignment='top',
1916             bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
1917
1918     # 2. 스펙트로그램 (Mel-spectrogram)
1919     ax2 = fig.add_subplot(gs[1, 0])
1920     librosa.display.specshow(mel_spec_db, sr=sr, x_axis='time', y_axis='mel',
1921                             ax=ax2, cmap='viridis')
1922     ax2.set_title('🎵 Mel-Spectrogram', fontsize=12, fontweight='bold')
1923     ax2.set_xlabel('시간 (초)')
1924     ax2.set_ylabel('Mel 주파수')
1925
1926     # 3. MFCC
1927     ax3 = fig.add_subplot(gs[1, 1])
1928     librosa.display.specshow(mfcc, sr=sr, x_axis='time', ax=ax3, cmap='coolwarm')
1929     ax3.set_title('📊 MFCC 계수', fontsize=12, fontweight='bold')
```

```
1930 ax3.set_xlabel('시간 (초)')
1931 ax3.set_ylabel('MFCC 계수')
1932
1933 # 4. 결합된 특징 맵
1934 ax4 = fig.add_subplot(gs[1, 2])
1935 im = ax4.imshow(features, aspect='auto', cmap='plasma', origin='lower')
1936 ax4.set_title('🌺 CNN 입력 특징\n(Mel-spec + MFCC)', fontsize=12, fontweight='bold')
1937 ax4.set_xlabel('시간 프레임')
1938 ax4.set_ylabel('특징 차원')
1939 plt.colorbar(im, ax=ax4, shrink=0.8)
1940
1941 # 5. 예측 확률
1942 ax5 = fig.add_subplot(gs[2, 0])
1943 probs = [prob.item() for prob in probabilities[0]]
1944 bars = ax5.bar(class_names_kr, probs, color=colors, alpha=0.8, edgecolor='black', linewidth=1)
1945 ax5.set_title('🎯 클래스별 예측 확률', fontsize=12, fontweight='bold')
1946 ax5.set_ylabel('확률')
1947 ax5.set_ylim(0, 1.1)
1948
1949 # 최고 확률 막대 강조
1950 max_idx = np.argmax(probs)
1951 bars[max_idx].set_color('red')
1952 bars[max_idx].set_alpha(1.0)
1953
1954 # 확률 값 표시
1955 for i, (bar, prob) in enumerate(zip(bars, probs)):
1956     height = bar.get_height()
1957     ax5.text(bar.get_x() + bar.get_width()/2., height + 0.02,
1958             f'{prob:.1%}', ha='center', va='bottom',
1959             fontweight='bold' if i == max_idx else 'normal')
1960
1961 # 6. 주파수 스펙트럼
1962 ax6 = fig.add_subplot(gs[2, 1])
1963 fft = np.fft.fft(audio)
1964 freqs = np.fft.fftfreq(len(audio), 1/sr)
1965 magnitude = np.abs(fft)
1966
1967 # 양의 주파수만 표시
1968 positive_freqs = freqs[:len(freqs)//2]
1969 positive_magnitude = magnitude[:len(magnitude)//2]
1970
1971 ax6.plot(positive_freqs, positive_magnitude, color='green', alpha=0.7)
1972 ax6.set_title('📊 주파수 스펙트럼', fontsize=12, fontweight='bold')
1973 ax6.set_xlabel('주파수 (Hz)')
1974 ax6.set_ylabel('크기')
1975 ax6.set_xlim(0, 4000) # 4kHz까지만 표시
1976 ax6.grid(True, alpha=0.3)
1977
1978 # 주요 주파수 성분 찾기
1979 dominant_freq_idx = np.argmax(positive_magnitude[1:]) + 1 # DC 제외
1980 dominant_freq = positive_freqs[dominant_freq_idx]
1981 ax6.axvline(dominant_freq, color='red', linestyle='--', alpha=0.8)
1982 ax6.text(dominant_freq, max(positive_magnitude)*0.8,
1983         f'주요 주파수\n(dominant_freq: {0f} Hz',
1984         ha='center', bbox=dict(boxstyle='round', facecolor='yellow', alpha=0.7))
1985
1986 # 7. 결과 요약
1987 ax7 = fig.add_subplot(gs[2, 2])
1988 ax7.axis('off')
1989
1990 result_text = f"""
1991 🎯 예측 결과
1992
1993 📁 파일: {filename}
1994 📁 예측 클래스: {class_names_kr[predicted_class]}
1995 📁 신뢰도: {confidence:.1%}
1996
1997 📊 상세 확률:
1998 • 발소리: {probs[0]:.1%}
1999 • 말소리: {probs[1]:.1%}
2000 • 가구끄는소리: {probs[2]:.1%}
2001
2002 📊 오디오 특성:
2003 • 길이: {len(audio)/sr:.1f}초
2004 • 샘플링 레이트: {sr:.} Hz
2005 • RMS 값: {rms:.4f}
2006 • 피크 값: {peak:.4f}
2007 • 주요 주파수: {dominant_freq:.0f} Hz
2008 ""
2009
2010 ax7.text(0.05, 0.95, result_text, transform=ax7.transAxes,
2011         verticalalignment='top', fontsize=11,
2012         bbox=dict(boxstyle='round,pad=1', facecolor='lightblue', alpha=0.8))
2013
2014 # 추가 특징 분석 (선택적)
2015 if show_features:
2016     # 8. 시간별 에너지 변화
2017     ax8 = fig.add_subplot(gs[3, 0])
2018     hop_length = 512
2019     frame_length = 2048
2020     energy = librosa.feature.rms(y=audio, hop_length=hop_length, frame_length=frame_length)[0]
2021     frames = range(len(energy))
```

```

2022 times = librosa.frames_to_time(frames, sr=sr, hop_length=hop_length)
2023
2024 ax8.plot(times, energy, color='purple', linewidth=2)
2025 ax8.set_title('⚡ 시간별 에너지 변화', fontsize=12, fontweight='bold')
2026 ax8.set_xlabel('시간 (초)')
2027 ax8.set_ylabel('RMS 에너지')
2028 ax8.grid(True, alpha=0.3)
2029
2030 # 9. 영교차율 (Zero Crossing Rate)
2031 ax9 = fig.add_subplot(gs[3, 1])
2032 zcr = librosa.feature.zero_crossing_rate(audio, hop_length=hop_length)[0]
2033 ax9.plot(times, zcr, color='orange', linewidth=2)
2034 ax9.set_title('📊 영교차율 (ZCR)', fontsize=12, fontweight='bold')
2035 ax9.set_xlabel('시간 (초)')
2036 ax9.set_ylabel('ZCR')
2037 ax9.grid(True, alpha=0.3)
2038
2039 # 10. 스펙트럼 중심 (Spectral Centroid)
2040 ax10 = fig.add_subplot(gs[3, 2])
2041 spectral_centroids = librosa.feature.spectral_centroid(y=audio, sr=sr, hop_length=hop_length)[0]
2042 ax10.plot(times, spectral_centroids, color='brown', linewidth=2)
2043 ax10.set_title('🎵 스펙트럼 중심', fontsize=12, fontweight='bold')
2044 ax10.set_xlabel('시간 (초)')
2045 ax10.set_ylabel('주파수 (Hz)')
2046 ax10.grid(True, alpha=0.3)
2047
2048 # 전체 제목
2049 fig.suptitle(f'🎧 오디오 분석 리포트: {class_names_kr[predicted_class]} (신뢰도: {confidence:.1%})',
2050             fontsize=16, fontweight='bold', y=0.98)
2051
2052 plt.tight_layout()
2053
2054 # 이미지 저장
2055 if save_image:
2056     safe_filename = filename.replace('.', '_').replace(' ', '_')
2057     image_path = f"audio_analysis_{safe_filename}_{class_names_en[predicted_class]}.png"
2058     plt.savefig(image_path, dpi=300, bbox_inches='tight')
2059     print(f'📁 이미지 저장: {image_path}')
2060
2061 plt.show()
2062
2063 # 오디오 재생
2064 display(Audio(audio, rate=sr))
2065
2066 return {
2067     'filename': filename,
2068     'predicted_class': class_names_kr[predicted_class],
2069     'predicted_class_en': class_names_en[predicted_class],
2070     'confidence': confidence,
2071     'probabilities': dict(zip(class_names_kr, probs)),
2072     'audio_features': {
2073         'rms': float(rms),
2074         'peak': float(peak),
2075         'dominant_frequency': float(dominant_freq),
2076         'duration': len(audio)/sr
2077     }
2078 }
2079
2080 except Exception as e:
2081     print(f'❌ 분석 오류: {str(e)}")
2082     return None
2083
2084 def batch_visualize_predictions(model_path, audio_files_list, max_files=10, save_images=False):
2085     """여러 오디오 파일을 일괄 시작화"""
2086     print(f'📁 {min(len(audio_files_list), max_files)}개 파일 일괄 분석 중...")
2087
2088     results = []
2089
2090     for i, audio_file in enumerate(audio_files_list[:max_files]):
2091         print(f"🎧 {i+1}/{min(len(audio_files_list), max_files)} - {os.path.basename(audio_file)}")
2092
2093         result = visualize_audio_prediction(
2094             model_path, audio_file,
2095             save_image=save_images,
2096             show_features=False # 빠른 분석을 위해 기본 특징만
2097         )
2098
2099         if result:
2100             results.append(result)
2101
2102 # 요약 통계
2103 if results:
2104     print(f"📊 일괄 분석 요약:")
2105
2106     # 클래스별 분포
2107     class_counts = {}
2108     confidence_by_class = {}
2109
2110     for result in results:
2111         pred_class = result['predicted_class']
2112         confidence = result['confidence']
2113

```

```

2114     if pred_class not in class_counts:
2115         class_counts[pred_class] = 0
2116         confidence_by_class[pred_class] = []
2117
2118     class_counts[pred_class] += 1
2119     confidence_by_class[pred_class].append(confidence)
2120
2121 print(f" - 분석된 파일: {len(results)}개")
2122 for class_name, count in class_counts.items():
2123     avg_confidence = np.mean(confidence_by_class[class_name])
2124     print(f" - {class_name}: {count}개 (평균 신뢰도: {avg_confidence:.1%})")
2125
2126 # 신뢰도 분포 시각화
2127 plt.figure(figsize=(12, 8))
2128
2129 plt.subplot(2, 2, 1)
2130 confidences = [r['confidence'] for r in results]
2131 plt.hist(confidences, bins=10, alpha=0.7, color='skyblue', edgecolor='black')
2132 plt.title('신뢰도 분포')
2133 plt.xlabel('신뢰도')
2134 plt.ylabel('빈도')
2135 plt.grid(True, alpha=0.3)
2136
2137 plt.subplot(2, 2, 2)
2138 classes = list(class_counts.keys())
2139 counts = list(class_counts.values())
2140 colors = ['#ff9999', '#66b3ff', '#99ff99'][:len(classes)]
2141 plt.pie(counts, labels=classes, colors=colors, autopct='%1.1f%%', startangle=90)
2142 plt.title('예측 클래스 분포')
2143
2144 plt.subplot(2, 2, 3)
2145 for class_name in classes:
2146     if class_name in confidence_by_class:
2147         plt.hist(confidence_by_class[class_name], alpha=0.6,
2148                 label=class_name, bins=5)
2149 plt.title('클래스별 신뢰도 분포')
2150 plt.xlabel('신뢰도')
2151 plt.ylabel('빈도')
2152 plt.legend()
2153 plt.grid(True, alpha=0.3)
2154
2155 plt.subplot(2, 2, 4)
2156 filenames = [r['filename'][:15] + '...' if len(r['filename']) > 15 else r['filename']
2157             for r in results]
2158 confidences = [r['confidence'] for r in results]
2159 colors_list = []
2160
2161 for result in results:
2162     if result['predicted_class'] == '발소리':
2163         colors_list.append('#ff9999')
2164     elif result['predicted_class'] == '말소리':
2165         colors_list.append('#66b3ff')
2166     else:
2167         colors_list.append('#99ff99')
2168
2169 plt.barh(range(len(filenames)), confidences, color=colors_list, alpha=0.8)
2170 plt.yticks(range(len(filenames)), filenames, fontsize=8)
2171 plt.xlabel('신뢰도')
2172 plt.title('파일별 예측 신뢰도')
2173 plt.grid(True, alpha=0.3)
2174
2175 plt.tight_layout()
2176 plt.show()
2177
2178 return results
2179
2180 def analyze_misclassified_samples(model, val_loader, max_samples=5):
2181     """잘못 분류된 샘플들을 찾아서 분석"""
2182     print(f"🔍 잘못 분류된 샘플 분석 중...")
2183
2184     model.eval()
2185     device = next(model.parameters()).device
2186
2187     misclassified = []
2188     correct_classified = []
2189
2190     class_names = ['발소리', '말소리', '가구끄는소리']
2191
2192     with torch.no_grad():
2193         for batch in val_loader:
2194             inputs = batch['features'].to(device)
2195             labels = batch['label'].squeeze().to(device)
2196             paths = batch['path']
2197
2198             outputs = model(inputs)
2199             probabilities = F.softmax(outputs, dim=1)
2200             predicted = torch.argmax(outputs, dim=1)
2201
2202             for i in range(len(labels)):
2203                 confidence = probabilities[i, predicted[i]].item()
2204
2205                 sample_info = {

```

```
2206         'path': paths[i],
2207         'true_label': labels[i].item(),
2208         'predicted_label': predicted[i].item(),
2209         'confidence': confidence,
2210         'true_classes': class_names[labels[i].item()],
2211         'predicted_classes': class_names[predicted[i].item()],
2212         'probabilities': probabilities[i].cpu().numpy()
2213     }
2214
2215     if labels[i] != predicted[i]:
2216         misclassified.append(sample_info)
2217     else:
2218         correct_classified.append(sample_info)
2219
2220 print(f" 📊 분석 결과:")
2221 print(f" - 전체 샘플: {len(misclassified) + len(correct_classified)}개")
2222 print(f" - 올바른 분류: {len(correct_classified)}개 ({len(correct_classified)/(len(misclassified) + len(correct_classified))*100:.1f}%)")
2223 print(f" - 잘못된 분류: {len(misclassified)}개 ({len(misclassified)/(len(misclassified) + len(correct_classified))*100:.1f}%)")
2224
2225 if misclassified:
2226     print(f"\n❌ 잘못 분류된 샘플 상위 {min(max_samples, len(misclassified))}개:")
2227
2228     # 신뢰도 높은 순으로 정렬 (확신있게 틀린 것들)
2229     misclassified_sorted = sorted(misclassified, key=lambda x: x['confidence'], reverse=True)
2230
2231     for i, sample in enumerate(misclassified_sorted[:max_samples]):
2232         print(f"\n{i+1}. {os.path.basename(sample['path'])}")
2233         print(f"   실제: {sample['true_class']} → 예측: {sample['predicted_class']}")
2234         print(f"   신뢰도: {sample['confidence']:.2f}")
2235         print(f"   확률 분포: {dict(zip(class_names, [f'{p:.2f}' for p in sample['probabilities']]))}")
2236
2237     # 해당 파일 상세 분석
2238     if os.path.exists(sample['path']):
2239         print(f" 📄 상세 분석:")
2240         try:
2241             audio, sr = librosa.load(sample['path'], sr=16000, duration=5.0)
2242             rms = np.sqrt(np.mean(audio**2))
2243             peak = np.max(np.abs(audio))
2244
2245             # 간단한 특징 분석
2246             mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=13)
2247             spectral_centroid = np.mean(librosa.feature.spectral_centroid(y=audio, sr=sr))
2248             zcr = np.mean(librosa.feature.zero_crossing_rate(audio))
2249
2250             print(f"   - RMS: {rms:.4f}, Peak: {peak:.4f}")
2251             print(f"   - 스펙트럼 중심: {spectral_centroid:.0f} Hz")
2252             print(f"   - 영교차율: {zcr:.4f}")
2253
2254         except Exception as e:
2255             print(f"   - 분석 오류: {e}")
2256
2257     return misclassified, correct_classified
2258
2259 def create_data_inspection_report(data_path, model_path, output_file='data_inspection_report.html'):
2260     """전체 데이터에 대한 종합 검사 리포트 생성"""
2261     print(f" 📋 데이터 검사 리포트 생성 중...")
2262
2263     # 데이터 스캔
2264     audio_files, labels = scan_three_class_data(data_path)
2265
2266     if len(audio_files) == 0:
2267         print(f" ❌ 분석할 데이터가 없습니다!")
2268         return
2269
2270     # 샘플링 (너무 많으면 일부만)
2271     max_samples = 50
2272     if len(audio_files) > max_samples:
2273         indices = np.random.choice(len(audio_files), max_samples, replace=False)
2274         sampled_files = [audio_files[i] for i in indices]
2275         sampled_labels = [labels[i] for i in indices]
2276     else:
2277         sampled_files = audio_files
2278         sampled_labels = labels
2279
2280 print(f" 📁 {len(sampled_files)}개 파일 분석 중...")
2281
2282 # 각 파일 분석
2283 analysis_results = []
2284
2285 for i, (file_path, true_label) in enumerate(zip(sampled_files, sampled_labels)):
2286     print(f"   진행률: {i+1}/{len(sampled_files)} ({(i+1)/len(sampled_files)*100:.1f}%), end='\n')
2287
2288     try:
2289         # 오디오 로드
2290         audio, sr = librosa.load(file_path, sr=16000, duration=5.0)
2291
2292         # 기본 특징 추출
2293         rms = np.sqrt(np.mean(audio**2))
2294         peak = np.max(np.abs(audio))
2295         duration = len(audio) / sr
2296
2297         # 고급 특징
```

```
2298     spectral_centroid = np.mean(librosa.feature.spectral_centroid(y=audio, sr=sr))
2299     zcr = np.mean(librosa.feature.zero_crossing_rate(audio))
2300     tempo, _ = librosa.beat.beat_track(y=audio, sr=sr)
2301
2302     # 모델 예측 (모델이 있는 경우)
2303     predicted_class = None
2304     confidence = None
2305
2306     if os.path.exists(model_path):
2307         checkpoint = torch.load(model_path, map_location='cpu')
2308         model = ThreeClassAudioCNN(num_classes=3)
2309         model.load_state_dict(checkpoint['model_state_dict'])
2310         model.eval()
2311
2312         # 특징 추출 및 예측
2313         max_length = 16000 * 5
2314         if len(audio) < max_length:
2315             audio_padded = np.pad(audio, (0, max_length - len(audio)))
2316         else:
2317             audio_padded = audio[:max_length]
2318
2319         mel_spec = librosa.feature.melspectrogram(
2320             y=audio_padded, sr=16000, n_mels=64, fmax=8000, hop_length=512, n_fft=2048
2321         )
2322         mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)
2323
2324         mfcc = librosa.feature.mfcc(
2325             y=audio_padded, sr=16000, n_mfcc=64, hop_length=512, n_fft=2048
2326         )
2327
2328         features = np.vstack([mel_spec_db, mfcc])
2329
2330         if features.shape[1] != 157:
2331             from scipy.ndimage import zoom
2332             zoom_factor = 157 / features.shape[1]
2333             features = zoom(features, (1, zoom_factor))
2334
2335         with torch.no_grad():
2336             features_tensor = torch.FloatTensor(features).unsqueeze(0)
2337             outputs = model(features_tensor)
2338             probabilities = F.softmax(outputs, dim=1)
2339             predicted_class = torch.argmax(outputs, dim=1).item()
2340             confidence = probabilities[0, predicted_class].item()
2341
2342     result = {
2343         'filename': os.path.basename(file_path),
2344         'filepath': file_path,
2345         'true_label': true_label,
2346         'predicted_label': predicted_class,
2347         'confidence': confidence,
2348         'duration': duration,
2349         'rms': rms,
2350         'peak': peak,
2351         'spectral_centroid': spectral_centroid,
2352         'zcr': zcr,
2353         'tempo': tempo,
2354         'file_size': os.path.getsize(file_path) / 1024 # KB
2355     }
2356
2357     analysis_results.append(result)
2358
2359 except Exception as e:
2360     print(f"\n⚠️ {file_path} 분석 실패: {e}")
2361
2362 print(f"\n✅ 분석 완료! {len(analysis_results)}개 파일 처리됨")
2363
2364 # 결과 요약
2365 df = pd.DataFrame(analysis_results)
2366
2367 print(f"\n📊 데이터 요약:")
2368 print(f"   - 평균 RMS: {df['rms'].mean():.4f}")
2369 print(f"   - 평균 피크: {df['peak'].mean():.4f}")
2370 print(f"   - 평균 길이: {df['duration'].mean():.1f}초")
2371 print(f"   - 평균 파일 크기: {df['file_size'].mean():.1f} KB")
2372
2373 if 'confidence' in df.columns and df['confidence'].notna().any():
2374     print(f"   - 평균 예측 신뢰도: {df['confidence'].mean():.2f}")
2375
2376     # 정확도 계산
2377     class_mapping = {'footstep': 0, 'speech': 1, 'furniture': 2}
2378     df['true_label_idx'] = df['true_label'].map(class_mapping)
2379     correct_predictions = df['true_label_idx'] == df['predicted_label']
2380     accuracy = correct_predictions.mean()
2381     print(f"   - 예측 정확도: {accuracy:.2f}")
2382
2383 # HTML 리포트 생성
2384 html_content = f"""
2385 <!DOCTYPE html>
2386 <html>
2387 <head>
2388   <title>오디오 데이터 검사 리포트</title>
2389   <meta charset="UTF-8">
```

```
2390 <style>
2391     body {{ font-family: Arial, sans-serif; margin: 20px; }}
2392     .header {{ background-color: #f0f0f0; padding: 20px; border-radius: 5px; }}
2393     .section {{ margin: 20px 0; }}
2394     table {{ border-collapse: collapse; width: 100%; }}
2395     th, td {{ border: 1px solid #ddd; padding: 8px; text-align: left; }}
2396     th {{ background-color: #f2f2f2; }}
2397     .correct {{ background-color: #d4edda; }}
2398     .incorrect {{ background-color: #f8d7da; }}
2399 </style>
2400 </head>
2401 <body>
2402     <div class="header">
2403         <h1>🔊 오디오 데이터 검사 리포트</h1>
2404         <p>생성 시간: {time.strftime('%Y-%m-%d %H:%M:%S')}</p>
2405         <p>분석된 파일 수: {len(analysis_results)}개</p>
2406     </div>
2407
2408     <div class="section">
2409         <h2>📊 전체 통계</h2>
2410         <table>
2411             <tr><th>항목</th><th>값</th></tr>
2412             <tr><td>평균 RMS</td><td>{df['rms'].mean():.4f}</td></tr>
2413             <tr><td>평균 피크</td><td>{df['peak'].mean():.4f}</td></tr>
2414             <tr><td>평균 길이</td><td>{df['duration'].mean():.1f}초</td></tr>
2415             <tr><td>평균 파일 크기</td><td>{df['file_size'].mean():.1f} KB</td></tr>
2416         </table>
2417
2418         if 'confidence' in df.columns and df['confidence'].notna().any():
2419             html_content += f"""
2420                 <tr><td>평균 예측 신뢰도</td><td>{df['confidence'].mean():.2%}</td></tr>
2421                 <tr><td>예측 정확도</td><td>{accuracy:.2%}</td></tr>
2422             """
2423
2424         html_content += """
2425             </table>
2426         </div>
2427
2428         <div class="section">
2429             <h2>📁 파일별 상세 정보</h2>
2430             <table>
2431                 <tr>
2432                     <th>파일명</th>
2433                     <th>실제 클래스</th>
2434                 </tr>
2435             </table>
2436
2437             if 'confidence' in df.columns and df['confidence'].notna().any():
2438                 html_content += """
2439                     <th>예측 클래스</th>
2440                     <th>신뢰도</th>
2441                 </table>
2442
2443                 html_content += """
2444                     <th>길이(초)</th>
2445                     <th>RMS</th>
2446                     <th>피크</th>
2447                     <th>크기(KB)</th>
2448                 </table>
2449
2450                 class_names = ['footstep', 'speech', 'furniture']
2451
2452                 for _, row in df.iterrows():
2453                     css_class = ""
2454                     if pd.notna(row.get('predicted_label')):
2455                         if row['true_label_idx'] == row['predicted_label']:
2456                             css_class = "correct"
2457                         else:
2458                             css_class = "incorrect"
2459
2460                     html_content += f"""
2461                         <tr class="{css_class}">
2462                             <td>{row['filename']}</td>
2463                             <td>{row['true_label']}</td>
2464                         </tr>
2465                     """
2466
2467                     if 'confidence' in df.columns and pd.notna(row.get('confidence')):
2468                         predicted_class_name = class_names[int(row['predicted_label'])] if pd.notna(row['predicted_label']) else 'N/A'
2469                         html_content += f"""
2470                             <td>{predicted_class_name}</td>
2471                             <td>{row['confidence']:.2%}</td>
2472                         </tr>
2473                     """
2474
2475                     html_content += f"""
2476                         <td>{row['duration']:.1f}</td>
2477                         <td>{row['rms']:.4f}</td>
2478                         <td>{row['peak']:.4f}</td>
2479                         <td>{row['file_size']:.1f}</td>
2480                     </tr>
2481                 """
2482             html_content += """
```

```
2482     </table>
2483 </div>
2484 </body>
2485 </html>
2486 """
2487
2488 # HTML 파일 저장
2489 with open(output_file, 'w', encoding='utf-8') as f:
2490     f.write(html_content)
2491
2492 print(f"📄 리포트 저장: {output_file}")
2493
2494 return analysis_results, df
2495
2496 def export_visualization_images(model_path, data_path, output_dir='visualization_export', max_files=20):
2497     """데이터셋의 시각화 이미지들을 폴더로 내보내기"""
2498
2499     # 출력 디렉토리 생성
2500     os.makedirs(output_dir, exist_ok=True)
2501
2502     # 클래스별 서브폴더 생성
2503     class_dirs = {}
2504     for class_name in ['footstep', 'speech', 'furniture']:
2505         class_dir = os.path.join(output_dir, class_name)
2506         os.makedirs(class_dir, exist_ok=True)
2507         class_dirs[class_name] = class_dir
2508
2509     print(f"🖼️ 시각화 이미지 내보내기 시작...")
2510     print(f"📁 출력 폴더: {output_dir}")
2511
2512     # 데이터 스캔
2513     audio_files, labels = scan_three_class_data(data_path)
2514
2515     if len(audio_files) == 0:
2516         print("❌ 내보낼 데이터가 없습니다!")
2517         return
2518
2519     # 클래스별로 균등하게 샘플링
2520     class_files = {'footstep': [], 'speech': [], 'furniture': []}
2521
2522     for file_path, label in zip(audio_files, labels):
2523         class_files[label].append(file_path)
2524
2525     # 각 클래스에서 최대 max_files//3 개씩 선택
2526     files_per_class = max_files // 3
2527     selected_files = []
2528
2529     for class_name, files_list in class_files.items():
2530         if files_list:
2531             sample_size = min(files_per_class, len(files_list))
2532             sampled = np.random.choice(files_list, sample_size, replace=False)
2533             selected_files.extend([(f, class_name) for f in sampled])
2534
2535     print(f"📁 {len(selected_files)}개 파일 처리 예정")
2536
2537     # 각 파일 처리
2538     export_summary = []
2539
2540     for i, (file_path, true_class) in enumerate(selected_files):
2541         filename = os.path.basename(file_path)
2542         print(f"🔄 처리 중: {i+1}/{len(selected_files)} - {filename}")
2543
2544         try:
2545             # 시각화 생성 (이미지 저장 모드)
2546             result = visualize_audio_prediction(
2547                 model_path, file_path,
2548                 save_image=False, # 수동으로 저장할 것
2549                 show_features=True
2550             )
2551
2552             if result:
2553                 # 이미지 저장 경로 결정
2554                 safe_filename = filename.replace('.', '_').replace(' ', '_')
2555                 predicted_class = result['predicted_class_en']
2556                 confidence = result['confidence']
2557
2558                 # 올바른 예측인지 확인
2559                 correct = "✓" if predicted_class == true_class else "X"
2560
2561                 image_filename = f"{safe_filename}_{predicted_class}_conf{confidence:.0%}_{correct}.png"
2562                 image_path = os.path.join(class_dirs[true_class], image_filename)
2563
2564                 # 현재 figure 저장
2565                 plt.savefig(image_path, dpi=200, bbox_inches='tight')
2566                 plt.close() # 메모리 절약
2567
2568                 export_summary.append({
2569                     'original_file': filename,
2570                     'true_class': true_class,
2571                     'predicted_class': predicted_class,
2572                     'confidence': confidence,
2573                     'correct': predicted_class == true_class,
```

```
2574         'image_path': image_path
2575     })
2576
2577     except Exception as e:
2578         print(f"❌ 오류: {e}")
2579
2580 # 요약 리포트 생성
2581 summary_file = os.path.join(output_dir, 'export_summary.txt')
2582
2583 with open(summary_file, 'w', encoding='utf-8') as f:
2584     f.write("📊 시각화 이미지 내보내기 요약\n")
2585     f.write("=" * 50 + "\n\n")
2586     f.write(f"총 처리된 파일: {len(export_summary)}개\n")
2587
2588     # 정확도 통계
2589     correct_count = sum(1 for item in export_summary if item['correct'])
2590     accuracy = correct_count / len(export_summary) if export_summary else 0
2591     f.write(f"정확한 예측: {correct_count}개 ({accuracy:.1%})\n")
2592     f.write(f"잘못된 예측: {len(export_summary) - correct_count}개\n")
2593
2594 # 클래스별 통계
2595 f.write("클래스별 통계:\n")
2596 for class_name in ['footstep', 'speech', 'furniture']:
2597     class_items = [item for item in export_summary if item['true_class'] == class_name]
2598     class_correct = sum(1 for item in class_items if item['correct'])
2599     class_accuracy = class_correct / len(class_items) if class_items else 0
2600     f.write(f"   {class_name}: {len(class_items)}개 (정확도: {class_accuracy:.1%})\n")
2601
2602 f.write("\n파일별 상세 정보:\n")
2603 f.write("-" * 50 + "\n")
2604
2605 for item in export_summary:
2606     status = "✓" if item['correct'] else "X"
2607     f.write(f"{status} {item['original_file']}\n")
2608     f.write(f"   실제: {item['true_class']} * 예측: {item['predicted_class']} (신뢰도: {item['confidence']:.1%})\n")
2609     f.write(f"   이미지: {os.path.basename(item['image_path'])}\n")
2610
2611 print(f"\n✅ 내보내기 완료!")
2612 print(f"  - 처리된 파일: {len(export_summary)}개")
2613 print(f"  - 정확도: {accuracy:.1%}")
2614 print(f"  - 이미지 폴더: {output_dir}")
2615 print(f"  - 요약 파일: {summary_file}")
2616
2617 return export_summary
2618
2619 def save_model_info(model_path):
2620     """모델 정보 저장"""
2621     if os.path.exists(model_path):
2622         checkpoint = torch.load(model_path, map_location='cpu')
2623
2624         info = {
2625             'model_type': '3클래스 오디오 분류 CNN',
2626             'classes': ['발소리', '말소리', '가구끄는소리'],
2627             'epoch': checkpoint.get('epoch', 'Unknown'),
2628             'val_accuracy': checkpoint.get('val_acc', 'Unknown'),
2629             'val_loss': checkpoint.get('val_loss', 'Unknown'),
2630             'input_shape': '(128, 157)',
2631             'sample_rate': '16kHz',
2632             'max_duration': '5초'
2633         }
2634
2635         print(f"📄 모델 정보:")
2636         for key, value in info.items():
2637             print(f"  {key}: {value}")
2638
2639         return info
2640     else:
2641         print(f"❌ 모델 파일을 찾을 수 없습니다.")
2642         return None
2643
2644 def export_model_for_production(model_path, output_path='model_production.pth'):
2645     """프로덕션용 모델 내보내기"""
2646     if not os.path.exists(model_path):
2647         print(f"❌ 모델 파일이 없습니다!")
2648         return
2649
2650     print("🔴 프로덕션용 모델 준비 중...")
2651
2652     # 모델 로드
2653     checkpoint = torch.load(model_path, map_location='cpu')
2654     model = ThreeClassAudioCNN(num_classes=3)
2655     model.load_state_dict(checkpoint['model_state_dict'])
2656     model.eval()
2657
2658     # TorchScript로 변환
2659     dummy_input = torch.randn(1, 1, 128, 157)
2660     traced_model = torch.jit.trace(model, dummy_input)
2661
2662     # 저장
2663     torch.jit.save(traced_model, output_path)
2664
2665     print(f"✅ 프로덕션용 모델 저장 완료: {output_path}")
```

```
2666     print("   이 모델은 별도 라이브러리 없이 PyTorch에서 바로 로드 가능합니다.")
2667
2668     # 사용법 출력
2669     print("\n👉 프로덕션 환경에서 사용법:")
2670     print(f"   model = torch.jit.load('{output_path}')"')
2671     print(f"   output = model(input_tensor)")
2672
2673 def quick_audio_preview(audio_path):
2674     """오디오 파일 빠른 미리보기"""
2675     try:
2676         audio, sr = librosa.load(audio_path, sr=16000, duration=10)
2677
2678         print(f"📄 파일: {os.path.basename(audio_path)}")
2679         print(f"  - 길이: {len(audio)/sr:.2f}초")
2680         print(f"  - 샘플링 레이트: {sr}Hz")
2681         print(f"  - 최대 진폭: {np.max(np.abs(audio)):.3f}")
2682         print(f"  - RMS: {np.sqrt(np.mean(audio**2)):.3f}")
2683
2684         # 간단한 시각화
2685         plt.figure(figsize=(12, 4))
2686         time = np.linspace(0, len(audio)/sr, len(audio))
2687         plt.plot(time, audio)
2688         plt.title(f'오디오 파형: {os.path.basename(audio_path)}')
2689         plt.xlabel('시간 (초)')
2690         plt.ylabel('진폭')
2691         plt.grid(True, alpha=0.3)
2692         plt.show()
2693
2694         # 오디오 재생
2695         display(Audio(audio, rate=sr))
2696
2697     except Exception as e:
2698         print(f"❌ 오디오 로딩 오류: {e}")
2699
2700 # 최종 메시지
2701 print("\n🎉 모든 기능이 준비되었습니다!")
2702 print("   한글 그래프 지원이 추가되었습니다! 📊")
2703 print("   📁 강력한 데이터 디버깅 기능이 추가되었습니다! 🔍")
2704 print("   📺 NEW! 오디오 데이터 시각화 분석 도구가 추가되었습니다! 🎧")
2705 print("\n🔍 500개 파일 중 150개만 학습되는 문제 해결:")
2706 print("   detailed_data_analysis('/your/data/path')")
2707 print("   run_sample_training('/your/data/path') # 디버깅 모드 자동 실행")
2708 print("\n📊 학습 결과를 이미지로 확인:")
2709 print("   test_visualization_features() # 모든 시각화 기능 테스트")
2710 print("   visualize_audio_prediction('model.pth', 'audio.wav', save_image=True)")
2711 print("   quick_audio_analysis('audio.wav') # 모델 없이도 분석 가능")
2712 print("   export_visualization_images('model.pth', '/data/path', 'output_folder')")
2713 print("\n🗂️ 민저 한글 폰트를 테스트해보세요!")
2714 print("   test_korean_font()")
2715 print("\n👉 이제 모든 오디오 파일을 놓치지 않고 학습하고 완벽하게 시각적으로 분석할 수 있습니다!")
```



발소리-말소리-가구끄는소리 분류 AI 모델

=====

W: Skipping acquire of configured file 'main/source/Sources' as repository '<https://r2u.stat.illinois.edu/ubuntu> jammy InRelease' does not seem to

✅ 한글 폰트 설정 완료

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

✅ Google Drive 마운트 완료

✅ Colab 환경 최적화 완료

🎵 발소리-말소리-가구끄는소리 분류 AI 모델 사용 가이드

=====

📄 주요 기능:

1. 3클래스 오디오 분류 (발소리, 말소리, 가구끄는소리)
2. CNN 기반 딥러닝 모델
3. 실시간 예측 및 시각화
4. 데이터 중간 및 최적화
5. 한글 폰트 지원

🚀 빠른 시작:

```
0 한글 폰트 테스트:
test_korean_font() # 한글이 제대로 표시되는지 확인
```

```
1 샘플 데이터로 빠른 테스트:
run_sample_training() # 샘플 생성 + 학습
test_model_with_samples() # 테스트
```

```
2 실제 데이터로 학습:
# 데이터 폴더 구조:
# └─ your_data/
#   └─ footstep/      (발소리 파일들)
#   └─ speech/        (말소리 파일들)
#   └─ furniture/     (가구끄는소리 파일들)
```

```
model, trainer = run_sample_training('/path/to/your_data')
```

```
3 파일 업로드하여 예측:
analyze_uploaded_file()
```

```
4 실시간 녹음 예측:
record_and_predict('best_three_class_model.pth')
```

```
5 특징 추출 대모:
demo_realtime_features()
```

```
1 analyze_model_performance(model, val_loader)

# 모델 성능 상세 분석:
analyze_model_performance(model, val_loader)

# 모델 성능:
- 입력: 5초 오디오 (16kHz)
- 특징: Mel-spectrogram (64) + MFCC (64) = 128차원
- 구조: CNN (4블록) + Global Average Pooling
- 출력: 3클래스 확률 분포

# 한글 폰트 문제 해결:
- 그래픽스에서 한글이 깨진다면: test_korean_font() 실행
- 여전히 문제가 있다면: 런타임 재시작 후 다시 실행

1 analyze_uploaded_file(r"/content/drive/MyDrive/Colab Notebooks/finaldata")

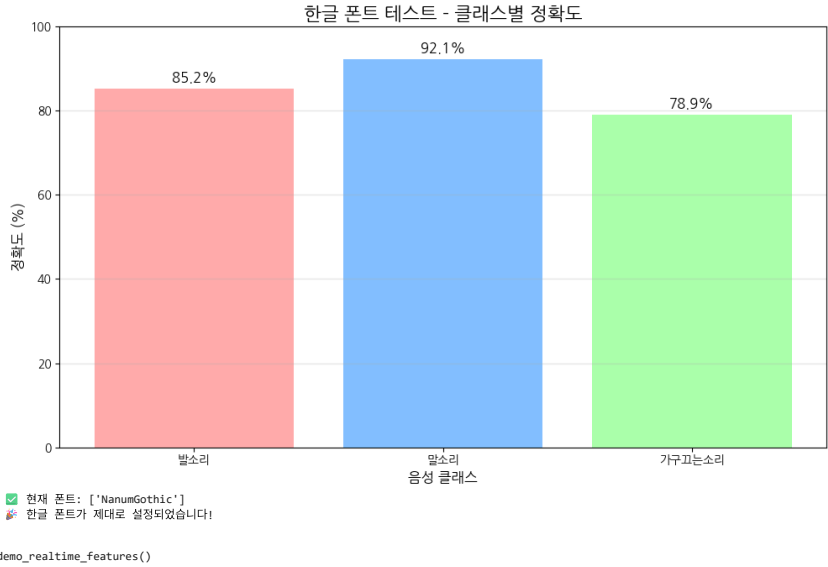
# Python 3.11.990113.py in <cell line: 0>()
----> 1 analyze_uploaded_file(r"/content/drive/MyDrive/Colab Notebooks/finaldata")

TypeError: analyze_uploaded_file() takes 0 positional arguments but 1 was given
```

다음 단계: 오류 설명

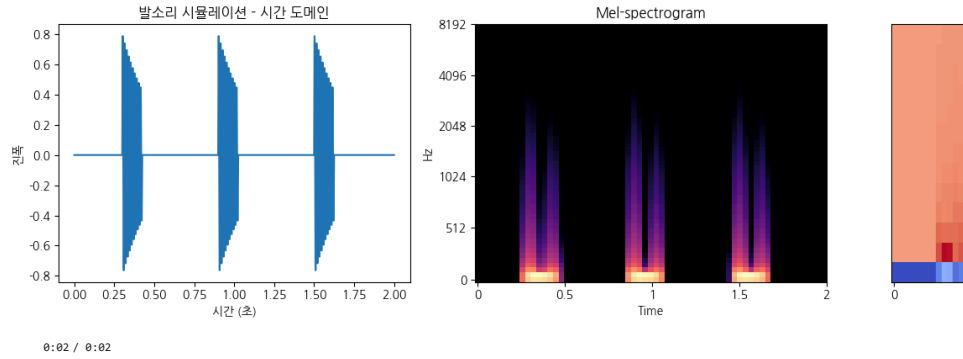
```
1 test_korean_font()

# 한글 폰트 테스트 중...
```

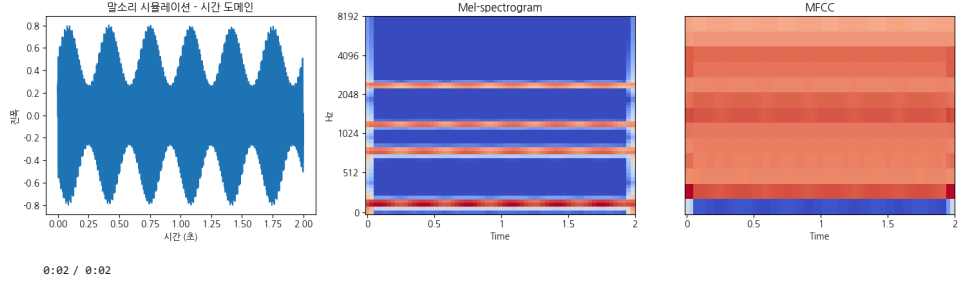


```
# 실시간 특징 추출 데모

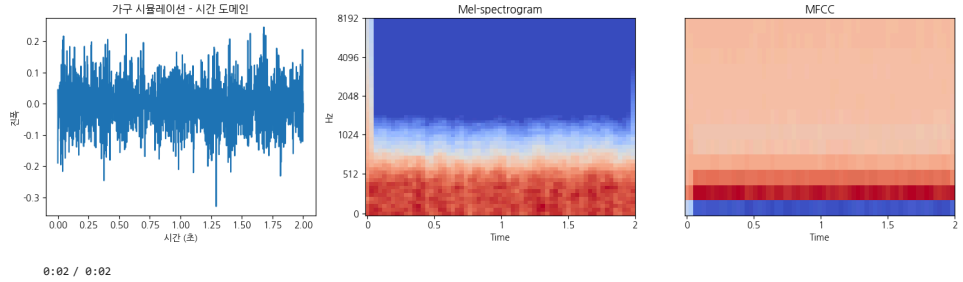
# 발소리 시뮬레이션 특징 추출 중...
```



```
# 발소리 시뮬레이션 특징 추출 중...
```



```
# 가구 시뮬레이션 특징 추출 중...
```



```
1 record_and_predict('best_three_class_model.pth')
```

5초간 녹음을 시작합니다...
마이크에 대고 소리를 내세요!
녹음 시작 (5초)
위의 녹음 버튼을 클릭해주세요!

```
1 analyze_model_performance(model, val_loader)
```



```

NameError                                Traceback (most recent call last)
/tmp/ipython-input-979307240.py in <cell line: 0>()
----> 1 analyze_model_performance(model, val_loader)

NameError: name 'model' is not defined
```

다음 단계: [오류 설명](#)

▼ wjfcnltjadasdadasdasddddd

```

1 # =====
2 # 2. Colab 환경 최적화
3 # =====
4
5 def setup_colab_environment():
6     """Colab 환경 최적화"""
7     try:
8         mp.set_start_method('spawn', force=True)
9     except:
10         pass
11
12     os.environ['TOKENIZERS_PARALLELISM'] = 'false'
13     os.environ['OMP_NUM_THREADS'] = '1'
14
15     # 한글 폰트 설정
16     setup_korean_font()
17
18     # Google Drive 마운트
19     try:
20         drive.mount('/content/drive')
21         print("✅ Google Drive 마운트 완료")
22     except:
23         print("⚠️ Google Drive 마운트 실패 또는 이미 마운트됨")
24
25     print("✅ Colab 환경 최적화 완료")
26
27 def setup_korean_font():
28     """한글 폰트 설정"""
29     try:
30         # 나눔고딕 폰트 설치
31         !apt-get update -qq
32         !apt-get install -qq fonts-nanum
33
34         # matplotlib 폰트 설정
35         import matplotlib.font_manager as fm
36         import matplotlib.pyplot as plt
37
38         # 폰트 캐시 삭제
39         !rm -rf ~/.cache/matplotlib
40
41         # 나눔고딕 폰트 경로
42         font_path = '/usr/share/fonts/truetype/nanum/NanumGothic.ttf'
43
44         if os.path.exists(font_path):
45             # 폰트 등록
46             fm.fontManager.addfont(font_path)
47
48             # matplotlib 기본 폰트 설정
49             plt.rcParams['font.family'] = 'NanumGothic'
50             plt.rcParams['axes.unicode_minus'] = False # 마이너스 기호 깨짐 방지
51
52             print("✅ 한글 폰트 설정 완료")
53         else:
54             # 대체 방법: 구글 폰트 사용
55             wget -O NanumGothic.ttf "https://github.com/google/fonts/raw/main/ofl/nanumgothic/NanumGothic-Regular.ttf"
56
57             # 폰트 등록
58             fm.fontManager.addfont('./NanumGothic.ttf')
59             plt.rcParams['font.family'] = 'NanumGothic'
60             plt.rcParams['axes.unicode_minus'] = False
61
62             print("✅ 한글 폰트 설정 완료 (대체 폰트)")
63
64     except Exception as e:
65         print(f"⚠️ 한글 폰트 설정 실패: {e}")
66         print("영어로 표시됩니다.")
67
68     # 영어 레이블로 대체
69     plt.rcParams['font.family'] = 'DejaVu Sans'
70
71 def test_korean_font():
72     """한글 폰트 테스트"""
73     print("🖨️ 한글 폰트 테스트 중...")
74
75     plt.figure(figsize=(10, 6))
76
77     # 테스트 데이터
78     classes = ['발소리', '말소리', '가구끼는소리']
79     values = [0.2, 0.4, 0.8]
```

```

10 values = [0.2, 0.4, 0.8]
80 colors = ['#f99999', '#66b3ff', '#99ff99']
81
82 # 바 차트 생성
83 bars = plt.bar(classes, values, color=colors, alpha=0.8)
84
85 # 제목 및 레이블
86 plt.title('한글 폰트 테스트 - 클래스별 정확도', fontsize=16, fontweight='bold')
87 plt.xlabel('음성 클래스', fontsize=12)
88 plt.ylabel('정확도 (%)', fontsize=12)
89
90 # 값 표시
91 for bar, value in zip(bars, values):
92     plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 1,
93              f'{value:.1f}%', ha='center', va='bottom', fontsize=12, fontweight='bold')
94
95 plt.ylim(0, 100)
96 plt.grid(True, axis='y', alpha=0.3)
97 plt.tight_layout()
98 plt.show()
99
100 # 폰트 상태 확인
101 current_font = plt.rcParams['font.family']
102 print(f"현재 폰트: {current_font}")
103
104 if 'NanumGothic' in current_font or 'Nanum' in str(current_font):
105     print("🎉 한글 폰트가 제대로 설정되었습니다!")
106 else:
107     print("⚠️ 한글 폰트 설정에 문제가 있을 수 있습니다.")
108     print("위 그래프에서 한글이 깨져 보인다면 런타임을 재시작해 보세요.")
109
110 def memory_cleanup():
111     """메모리 정리"""
112     gc.collect()
113     if torch.cuda.is_available():
114         torch.cuda.empty_cache()
115     print("✅ 메모리 정리 완료")
116
117 # =====
118 # 3. 3클래스 오디오 데이터셋 (발소리, 말소리, 가구끼는소리)
119 # =====
120
121 class ThreeClassAudioDataset(Dataset):
122     def __init__(self, audio_paths, labels, target_sr=16000, max_duration=5.0, augment=False):
123         """
124         3클래스 오디오 분류 데이터셋
125
126         Args:
127             audio_paths: 오디오 파일 경로 리스트
128             labels: 레이블 리스트 ['footstep', 'speech', 'furniture']
129             target_sr: 목표 샘플링 레이트 (16kHz)
130             max_duration: 최대 길이 (5초)
131             augment: 데이터 증강 여부
132         """
133         self.audio_paths = audio_paths
134         self.target_sr = target_sr
135         self.max_duration = max_duration
136         self.max_length = int(target_sr * max_duration)
137         self.augment = augment
138
139     # 레이블 매핑
140     self.class_names = ['footstep', 'speech', 'furniture']
141     self.label_to_idx = {name: idx for idx, name in enumerate(self.class_names)}
142     self.idx_to_label = {idx: name for name, idx in self.label_to_idx.items()}
143
144     # 레이블 인코딩
145     self.labels = [self.label_to_idx[label] for label in labels]
146
147     print(f"📁 데이터셋 정보:")
148     print(f"총 샘플 수: {len(self.audio_paths)}")
149     print(f"클래스: {self.class_names}")
150     for i, class_name in enumerate(self.class_names):
151         count = sum(1 for label in self.labels if label == i)
152         print(f"  {class_name}: {count}개")
153
154     def __len__(self):
155         return len(self.audio_paths)
156
157     def __getitem__(self, idx):
158         try:
159             # 오디오 로드
160             audio, sr = librosa.load(
161                 self.audio_paths[idx],
162                 sr=self.target_sr,
163                 duration=self.max_duration
164             )
165
166             # 길이 정규화
167             audio = self._normalize_length(audio)
168
169             # 데이터 증강
170             if self.augment:
171                 audio = self._augment_audio(audio)
172         except:
```

```

172
173     # 특징 추출 (Mel-spectrogram + MFCC)
174     features = self._extract_features(audio)
175
176     return {
177         'features': torch.FloatTensor(features),
178         'label': torch.LongTensor([self.labels[idx]]),
179         'path': self.audio_paths[idx]
180     }
181
182 except Exception as e:
183     print(f"오디오 로딩 오류 {self.audio_paths[idx]}: {e}")
184     # 빈 특징 반환
185     features = np.zeros((128, 157)) # 기본 특징 크기
186     return {
187         'features': torch.FloatTensor(features),
188         'label': torch.LongTensor([0]),
189         'path': self.audio_paths[idx]
190     }
191
192 def _normalize_length(self, audio):
193     """오디오 길이 정규화"""
194     if len(audio) > self.max_length:
195         # 랜덤 크롭
196         start = np.random.randint(0, len(audio) - self.max_length + 1)
197         audio = audio[start:start + self.max_length]
198     elif len(audio) < self.max_length:
199         # 제로 패딩
200         audio = np.pad(audio, (0, self.max_length - len(audio)))
201     return audio
202
203 def _augment_audio(self, audio):
204     """오디오 데이터 증강"""
205     # 시간 이동
206     if np.random.random() > 0.5:
207         shift = np.random.randint(-len(audio)//8, len(audio)//8)
208         audio = np.roll(audio, shift)
209
210     # 볼륨 조절
211     if np.random.random() > 0.5:
212         volume_factor = np.random.uniform(0.7, 1.3)
213         audio = audio * volume_factor
214
215     # 가우시안 노이즈 추가
216     if np.random.random() > 0.7:
217         noise = np.random.normal(0, 0.005, len(audio))
218         audio = audio + noise
219
220     # 피치 시프트 (가끔)
221     if np.random.random() > 0.8:
222         pitch_shift = np.random.randint(-2, 3)
223         if pitch_shift != 0:
224             audio = librosa.effects.pitch_shift(audio, sr=self.target_sr, n_steps=pitch_shift)
225
226     return np.clip(audio, -1.0, 1.0)
227
228 def _extract_features(self, audio):
229     """특징 추출: Mel-spectrogram + MFCC"""
230     # Mel-spectrogram
231     mel_spec = librosa.feature.melspectrogram(
232         y=audio,
233         sr=self.target_sr,
234         n_mels=64,
235         fmax=8000,
236         hop_length=512,
237         n_fft=2048
238     )
239     mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)
240
241     # MFCC
242     mfcc = librosa.feature.mfcc(
243         y=audio,
244         sr=self.target_sr,
245         n_mfcc=64,
246         hop_length=512,
247         n_fft=2048
248     )
249
250     # 특징 결합
251     features = np.vstack([mel_spec_db, mfcc]) # (128, time_frames)
252
253     # 고정 크기로 조정
254     target_frames = 157 # 5초 * 16000 / 512 = 157
255     if features.shape[1] != target_frames:
256         features = self._resize_features(features, target_frames)
257
258     return features
259
260 def _resize_features(self, features, target_frames):
261     """특징 크기 조정"""
262     from scipy.ndimage import zoom
263     zoom_factor = target_frames / features.shape[1]
264     return zoom(features, (1, zoom_factor))
265

```

```

265
266 # =====
267 # 4. 3클래스 분류 CNN 모델
268 # =====
269
270 class ThreeClassAudioCNN(nn.Module):
271     def __init__(self, num_classes=3, dropout_rate=0.3):
272         super(ThreeClassAudioCNN, self).__init__()
273
274         # Convolutional layers
275         self.conv_layers = nn.Sequential(
276             # Block 1
277             nn.Conv2d(1, 32, kernel_size=(3, 3), padding=1),
278             nn.BatchNorm2d(32),
279             nn.ReLU(),
280             nn.MaxPool2d((2, 2)),
281             nn.Dropout2d(0.1),
282
283             # Block 2
284             nn.Conv2d(32, 64, kernel_size=(3, 3), padding=1),
285             nn.BatchNorm2d(64),
286             nn.ReLU(),
287             nn.MaxPool2d((2, 2)),
288             nn.Dropout2d(0.1),
289
290             # Block 3
291             nn.Conv2d(64, 128, kernel_size=(3, 3), padding=1),
292             nn.BatchNorm2d(128),
293             nn.ReLU(),
294             nn.MaxPool2d((2, 2)),
295             nn.Dropout2d(0.2),
296
297             # Block 4
298             nn.Conv2d(128, 256, kernel_size=(3, 3), padding=1),
299             nn.BatchNorm2d(256),
300             nn.ReLU(),
301             nn.MaxPool2d((2, 2)),
302             nn.Dropout2d(0.2),
303         )
304
305         # Global Average Pooling
306         self.global_pool = nn.AdaptiveAvgPool2d((1, 1))
307
308         # Classifier
309         self.classifier = nn.Sequential(
310             nn.Dropout(dropout_rate),
311             nn.Linear(256, 128),
312             nn.ReLU(),
313             nn.Dropout(dropout_rate * 0.5),
314             nn.Linear(128, num_classes)
315         )
316
317     def forward(self, x):
318         # Input: (batch_size, features, time)
319         if len(x.shape) == 3:
320             x = x.unsqueeze(1) # Add channel dimension
321
322         x = self.conv_layers(x)
323         x = self.global_pool(x)
324         x = x.view(x.size(0), -1)
325         x = self.classifier(x)
326         return x
327
328 # =====
329 # 5. 학습 매니저
330 # =====
331
332 class ThreeClassTrainer:
333     def __init__(self, model, train_loader, val_loader, device='auto', lr=0.001):
334         self.device = torch.device('cuda' if torch.cuda.is_available() and device=='auto' else 'cpu')
335         self.model = model.to(self.device)
336         self.train_loader = train_loader
337         self.val_loader = val_loader
338
339         # 최적화 설정
340         self.optimizer = torch.optim.AdamW(
341             model.parameters(),
342             lr=lr,
343             weight_decay=0.01
344         )
345         self.scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
346             self.optimizer,
347             mode='max', # 정확도 기준
348             patience=5,
349             factor=0.5,
350             verbose=True
351         )
352         self.criterion = nn.CrossEntropyLoss()
353
354         # 기록
355         self.train_losses = []
356         self.val_losses = []
357         self.train_accs = []
358

```

```

358     self.val_accs = []
359     self.best_val_acc = 0.0
360     self.best_model_state = None
361
362     print(f"🔴 학습 설정:")
363     print(f"    장치: {self.device}")
364     print(f"    학습률: {lr}")
365     print(f"    클래스: 3개 (발소리, 말소리, 가구끄는소리)")
366
367 def train_epoch(self):
368     """한 에포크 학습"""
369     self.model.train()
370     running_loss = 0.0
371     correct = 0
372     total = 0
373
374     pbar = tqdm(self.train_loader, desc="학습")
375     for batch in pbar:
376         inputs = batch['features'].to(self.device)
377         labels = batch['label'].squeeze().to(self.device)
378
379         self.optimizer.zero_grad()
380         outputs = self.model(inputs)
381         loss = self.criterion(outputs, labels)
382         loss.backward()
383         self.optimizer.step()
384
385         running_loss += loss.item()
386         _, predicted = outputs.max(1)
387         total += labels.size(0)
388         correct += predicted.eq(labels).sum().item()
389
390     # 진행률 업데이트
391     pbar.set_postfix({
392         'Loss': f'{loss.item():.4f}',
393         'Acc': f'{100.*correct/total:.1f}%'
394     })
395
396     epoch_loss = running_loss / len(self.train_loader)
397     epoch_acc = 100. * correct / total
398
399     return epoch_loss, epoch_acc
400
401 def validate_epoch(self):
402     """검증"""
403     self.model.eval()
404     running_loss = 0.0
405     correct = 0
406     total = 0
407     all_preds = []
408     all_labels = []
409
410     with torch.no_grad():
411         pbar = tqdm(self.val_loader, desc="검증")
412         for batch in pbar:
413             inputs = batch['features'].to(self.device)
414             labels = batch['label'].squeeze().to(self.device)
415
416             outputs = self.model(inputs)
417             loss = self.criterion(outputs, labels)
418
419             running_loss += loss.item()
420             _, predicted = outputs.max(1)
421             total += labels.size(0)
422             correct += predicted.eq(labels).sum().item()
423
424     # 혼동행렬을 데이터 수집
425     all_preds.extend(predicted.cpu().numpy())
426     all_labels.extend(labels.cpu().numpy())
427
428     pbar.set_postfix({
429         'Loss': f'{loss.item():.4f}',
430         'Acc': f'{100.*correct/total:.1f}%'
431     })
432
433     epoch_loss = running_loss / len(self.val_loader)
434     epoch_acc = 100. * correct / total
435
436     return epoch_loss, epoch_acc, all_preds, all_labels
437
438 def train(self, num_epochs=30, save_path='best_three_class_model.pth'):
439     """전체 학습"""
440     print(f"🔴 3클래스 분류 학습 시작!")
441     print(f" = 50")
442
443     for epoch in range(num_epochs):
444         print(f"\n🟢 Epoch {epoch+1}/{num_epochs}")
445
446         # 학습
447         train_loss, train_acc = self.train_epoch()
448         self.train_losses.append(train_loss)
449         self.train_accs.append(train_acc)
450

```

```

451     # 검증
452     val_loss, val_acc, val_preds, val_labels = self.validate_epoch()
453     self.val_losses.append(val_loss)
454     self.val_accs.append(val_acc)
455
456     # 스케줄러 업데이트
457     self.scheduler.step(val_acc)
458
459     print(f"    학습 - Loss: {train_loss:.4f}, Acc: {train_acc:.2f}%")
460     print(f"    검증 - Loss: {val_loss:.4f}, Acc: {val_acc:.2f}%")
461     print(f"    학습률: {self.optimizer.param_groups[0]['lr']:.6f}")
462
463     # 최고 모델 저장
464     if val_acc > self.best_val_acc:
465         self.best_val_acc = val_acc
466         self.best_model_state = self.model.state_dict().copy()
467
468         torch.save({
469             'epoch': epoch,
470             'model_state_dict': self.model.state_dict(),
471             'optimizer_state_dict': self.optimizer.state_dict(),
472             'val_acc': val_acc,
473             'val_loss': val_loss,
474             'class_names': ['footstep', 'speech', 'furniture']
475         }, save_path)
476
477     print(f"    최고 모델 저장! (검증 정확도: {val_acc:.2f}%)")
478
479     # 클래스별 정확도 출력
480     self.print_class_accuracy(val_labels, val_preds)
481
482     # 초기 종료
483     if self.optimizer.param_groups[0]['lr'] < 1e-6:
484         print(f"    학습률이 너무 낮아 학습을 종료합니다.")
485         break
486
487     # 메모리 정리
488     if epoch % 5 == 0:
489         memory_cleanup()
490
491     print(f"\n🟢 학습 완료!")
492     print(f"    최고 검증 정확도: {self.best_val_acc:.2f}%")
493
494     # 최고 모델 로드
495     if self.best_model_state:
496         self.model.load_state_dict(self.best_model_state)
497
498     return self.model
499
500 def print_class_accuracy(self, true_labels, pred_labels):
501     """클래스별 정확도 출력"""
502     class_names = ['발소리', '말소리', '가구끄는소리']
503
504     for i, class_name in enumerate(class_names):
505         class_mask = np.array(true_labels) == i
506         if class_mask.sum() > 0:
507             class_acc = (np.array(pred_labels)[class_mask] == i).sum() / class_mask.sum()
508             print(f"    {class_name): {class_acc*100:.1f}%")
509
510 def plot_training_history(self):
511     """학습 히스토리 시각화 - 한글 지원"""
512     # 한글 폰트 설정 확인
513     try:
514         plt.rcParams['font.family'] = 'NanumGothic'
515         plt.rcParams['axes.unicode_minus'] = False
516     except:
517         pass
518
519     fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))
520
521     # 손실 그래프
522     epochs = range(1, len(self.train_losses) + 1)
523     ax1.plot(epochs, self.train_losses, 'b-', label='학습 손실', linewidth=2, alpha=0.8)
524     ax1.plot(epochs, self.val_losses, 'r-', label='검증 손실', linewidth=2, alpha=0.8)
525     ax1.set_title('학습/검증 손실', fontsize=14, fontweight='bold', pad=15)
526     ax1.set_xlabel('에포크 (Epoch)', fontsize=12)
527     ax1.set_ylabel('손실 (Loss)', fontsize=12)
528     ax1.legend(fontsize=11)
529     ax1.grid(True, alpha=0.3)
530     ax1.set_xlim(1, len(self.train_losses))
531
532     # 정확도 그래프
533     ax2.plot(epochs, self.train_accs, 'b-', label='학습 정확도', linewidth=2, alpha=0.8)
534     ax2.plot(epochs, self.val_accs, 'r-', label='검증 정확도', linewidth=2, alpha=0.8)
535     ax2.set_title('학습/검증 정확도', fontsize=14, fontweight='bold', pad=15)
536     ax2.set_xlabel('에포크 (Epoch)', fontsize=12)
537     ax2.set_ylabel('정확도 (%)', fontsize=12)
538     ax2.legend(fontsize=11)
539     ax2.grid(True, alpha=0.3)
540     ax2.set_ylim(0, 100)
541     ax2.set_xlim(1, len(self.train_accs))
542
543     # 최고 성능 지점 표시

```

```
544 best_epoch = np.argmax(self.val_accs) + 1
545 best_acc = max(self.val_accs)
546 ax2.plot(best_epoch, best_acc, 'ro', markersize=10, markerfacecolor='red',
547         markeredgecolor='darkred', markeredgewidth=2)
548 ax2.annotate(f'최고: {best_acc:.1f}%\n(에포크 {best_epoch})',
549             xy=(best_epoch, best_acc), xytext=(10, 10),
550             textcoords='offset points', fontsize=10,
551             bbox=dict(boxstyle='round,pad=0.3', facecolor='yellow', alpha=0.7),
552             arrowprops=dict(arrowstyle='->', connectionstyle='arc3,rad=0'))
553
554 plt.tight_layout()
555 plt.show()
556
557 # 상세 결과 출력
558 print("\n" + "="*60)
559 print("📊 학습 완료 - 최종 결과 요약")
560 print("="*60)
561 print(f"🏆 최고 검증 정확도: {max(self.val_accs):.2f}% (에포크 {np.argmax(self.val_accs)+1})")
562 print(f"📈 최종 학습 정확도: {self.train_accs[-1]:.2f}%")
563 print(f"📉 최종 검증 손실: {self.val_losses[-1]:.4f}")
564 print(f"🔄 총 학습 에포크: {len(self.train_accs)}개")
565
566 # 성능 안정성 분석
567 last_5_accs = self.val_accs[-5:] if len(self.val_accs) >= 5 else self.val_accs
568 stability = np.std(last_5_accs)
569 print(f"📊 최근 5에포크 정확도 안정성: {stability:.2f}% (낮을수록 안정)")
570
571 if stability < 2.0:
572     print(f"✅ 매우 안정적인 학습!")
573 elif stability < 5.0:
574     print(f"🟡 안정적인 학습")
575 else:
576     print(f"⚠️ 불안정한 학습 - 더 많은 에포크나 조기종료 필요")
577
578 print("="*60)
579
580 # =====
581 # 6. 데이터 처리 및 학습 실행 함수
582 # =====
583
584 def scan_three_class_data(base_path):
585     """3클래스 데이터 스캔 - 강화버전"""
586     audio_files = []
587     labels = []
588
589     # 지원되는 오디오 확장자 (더 많은 형식 추가)
590     extensions = ['.wav', '.mp3', '.flac', '.m4a', '.ogg', '.aac', '.wma', '.aiff', '.au']
591
592     print(f"📁 {base_path}에서 오디오 파일 스캔 중...")
593     print(f"🔍 지원 확장자: {', '.join(extensions)}")
594
595     # 각 클래스별 폴더에서 파일 수집 (더 많은 키워드 추가)
596     class_folders = {
597         'footstep': ['footstep', 'footsteps', 'foot', 'walk', 'walking', 'step',
598                     '발소리', '걸음소리', '발걸음', '보행', 'steps', 'footfall'],
599         'speech': ['speech', 'voice', 'talk', 'talking', 'speaking', '말소리', 'speak',
600                  '음성', '대화', '목소리', 'vocal', 'utterance', 'conversation'],
601         'furniture': ['furniture', 'chair', 'table', 'drag', 'move', '가구', 'scrape',
602                     '끄는소리', '이동', '의자', '테이블', 'moving', 'sliding', 'dragging']
603     }
604
605     # 모든 하위 디렉토리 탐색
606     total_files_found = 0
607     processed_dirs = []
608
609     for root, dirs, files in os.walk(base_path):
610         folder_name = os.path.basename(root).lower()
611         relative_path = os.path.relpath(root, base_path)
612
613         # 오디오 파일이 있는지 확인
614         audio_files_in_dir = [f for f in files if any(f.lower().endswith(ext) for ext in extensions)]
615
616         if audio_files_in_dir:
617             print(f"📁 {relative_path} - {len(audio_files_in_dir)}개 오디오 파일 발견")
618             total_files_found += len(audio_files_in_dir)
619
620             # 폴더명으로 클래스 판단
621             detected_class = None
622             for class_name, keywords in class_folders.items():
623                 if any(keyword in folder_name for keyword in keywords):
624                     detected_class = class_name
625                     break
626
627             # 클래스가 자동 감지되지 않으면 사용자에게 물어보기
628             if detected_class is None:
629                 print(f"❓ '{folder_name}' 폴더의 클래스를 판단할 수 없습니다.")
630                 print(f"👉 다음 중 하나를 선택하세요:")
631                 print(f"1: footstep (발소리)")
632                 print(f"2: speech (말소리)")
633                 print(f"3: furniture (가구끄는소리)")
634                 print(f"0: skip (건너뛰기)")
635
636             try:
```

```
637 choice = input(f"   선택 (1/2/3/0): ").strip()
638 class_mapping = {'1': 'footstep', '2': 'speech', '3': 'furniture'}
639 if choice in class_mapping:
640     detected_class = class_mapping[choice]
641     print(f"✅ '{folder_name}' -> {detected_class}")
642 else:
643     print(f"❌ '{folder_name}' - 폴더 건너뛸")
644     continue
645
646 except:
647     print(f"⚠️ 입력 오류로 '{folder_name}' - 폴더 건너뛸")
648     continue
649
650 else:
651     print(f"✅ {relative_path} -> {detected_class}")
652
653 # 파일 추가
654 for file in audio_files_in_dir:
655     file_path = os.path.join(root, file)
656     audio_files.append(file_path)
657     labels.append(detected_class)
658
659 processed_dirs.append((relative_path, detected_class, len(audio_files_in_dir)))
660
661 # 상세 결과 출력
662 print(f"\n📊 데이터 스캔 완료:")
663 print(f"📁 총 발견된 오디오 파일: {total_files_found}개")
664 print(f"📁 실제 사용할 파일: {len(audio_files)}개")
665 print(f"📁 처리된 디렉토리: {len(processed_dirs)}개")
666
667 print(f"\n📁 처리된 디렉토리 상세:")
668 for dir_path, class_name, count in processed_dirs:
669     print(f"    - {dir_path}: {class_name} ({count}개)")
670
671 print(f"\n📁 클래스별 파일 수:")
672 for class_name in ['footstep', 'speech', 'furniture']:
673     count = labels.count(class_name)
674     percentage = (count / len(labels) * 100) if len(labels) > 0 else 0
675     print(f"    - {class_name}: {count}개 ({percentage:.1f}%)")
676
677 if len(audio_files) == 0:
678     print(f"\n❌ 사용 가능한 오디오 파일이 없습니다!")
679     print(f"👉 다음을 확인해주세요:")
680     print(f"1. 파일 확장자가 지원되는지 확인")
681     print(f"2. 폴더 구조가 올바른지 확인")
682     print(f"3. 파일이 실제로 오디오 파일인지 확인")
683     return [], []
684
685 if total_files_found > len(audio_files):
686     print(f"\n⚠️ 주의: {total_files_found - len(audio_files)}개 파일이 제외되었습니다.")
687     print(f"👉 폴더명이 클래스와 매치되지 않아 제외되었을 수 있습니다.")
688
689 return audio_files, labels
690
691 def detailed_data_analysis(base_path):
692     """데이터 상세 분석"""
693     print(f"📊 데이터 구조 상세 분석 중...")
694
695     extensions = ['.wav', '.mp3', '.flac', '.m4a', '.ogg', '.aac', '.wma', '.aiff', '.au']
696
697     total_files = 0
698     total_size = 0
699     dir_info = []
700
701     for root, dirs, files in os.walk(base_path):
702         audio_files = []
703         dir_size = 0
704
705         for file in files:
706             file_path = os.path.join(root, file)
707             if any(file.lower().endswith(ext) for ext in extensions):
708                 audio_files.append(file)
709                 try:
710                     file_size = os.path.getsize(file_path)
711                     dir_size += file_size
712                 except:
713                     pass
714
715         if audio_files:
716             relative_path = os.path.relpath(root, base_path)
717             dir_info.append({
718                 'path': relative_path,
719                 'files': len(audio_files),
720                 'size_mb': dir_size / (1024*1024),
721                 'sample_files': audio_files[:3] # 처음 3개 파일명
722             })
723
724         total_files += len(audio_files)
725         total_size += dir_size
726
727 print(f"\n📊 전체 통계:")
728 print(f"📁 총 오디오 파일: {total_files}개")
729 print(f"📁 총 크기: {total_size/(1024*1024):.1f} MB")
730 print(f"📁 오디오가 있는 폴더: {len(dir_info)}개")
731
```

```

730 print(f"\n 📁 폴더별 상세 정보:")
731 for info in sorted(dir_info, key=lambda x: x['files'], reverse=True):
732     print(f" 📁 {info['path']}")
733     print(f" - 파일 수: {info['files']}개")
734     print(f" - 크기: {info['size_mb']:.1f} MB")
735     print(f" - 샘플: {'', '.join(info['sample_files'])}")
736     print()
737
738 return dir_info
739
740 def force_scan_all_audio_files(base_path):
741     """모든 오디오 파일 강제 스캔 (클래스 구분 없이)"""
742     print(f" 📁 모든 오디오 파일 강제 스캔 중...")
743
744     extensions = ['.wav', '.mp3', '.flac', '.m4a', '.ogg', '.aac', '.wma', '.aiff', '.au']
745     all_files = []
746
747     for root, dirs, files in os.walk(base_path):
748         for file in files:
749             if any(file.lower().endswith(ext) for ext in extensions):
750                 file_path = os.path.join(root, file)
751                 relative_path = os.path.relpath(file_path, base_path)
752                 all_files.append({
753                     'path': file_path,
754                     'relative': relative_path,
755                     'dir': os.path.dirname(relative_path),
756                     'filename': file
757                 })
758
759 print(f" 📁 전체 스캔 결과: {len(all_files)}개 오디오 파일 발견")
760
761 # 디렉토리별 그룹화
762 from collections import defaultdict
763 dir_groups = defaultdict(list)
764
765 for file_info in all_files:
766     dir_name = file_info['dir'] if file_info['dir'] else 'root'
767     dir_groups[dir_name].append(file_info)
768
769 print(f"\n 📁 디렉토리별 파일 수:")
770 for dir_name, file_list in sorted(dir_groups.items(), key=lambda x: len(x[1]), reverse=True):
771     print(f" {dir_name}: {len(file_list)}개")
772     # 처음 3개 파일명 표시
773     for i, file_info in enumerate(file_list[:3]):
774         print(f" - {file_info['filename']}")
775     if len(file_list) > 3:
776         print(f" ... 그 외 {len(file_list)-3}개")
777     print()
778
779 return all_files, dir_groups
780
781 def run_three_class_training(data_path, num_epochs=40, batch_size=7, test_size=0.2):
782     """3클래스 분류 학습 실행"""
783
784     print(f" 🎧 발소리-말소리-가구끄는소리 분류 학습 시작!")
785     print(f" = * 70")
786
787     # 환경 설정
788     setup_colab_environment()
789     memory_cleanup()
790
791     # 1. 데이터 스캔
792     audio_files, labels = scan_three_class_data(data_path)
793
794     if len(audio_files) == 0:
795         print(f" ❌ 오디오 파일을 찾을 수 없습니다!")
796         print(f" 📁 데이터 구조를 확인해주세요:")
797         print(f" | your_dataset/")
798         print(f" |   ↳ footsteps/")
799         print(f" |   ↳ speech/")
800         print(f" |   ↳ furniture/")
801         return None
802
803     # 2. 학습/검증 분할
804     train_files, val_files, train_labels, val_labels = train_test_split(
805         audio_files, labels, test_size=test_size, random_state=42, stratify=labels
806     )
807
808     print(f"\n 📁 데이터 분할:")
809     print(f" - 학습: {len(train_files)}개")
810     print(f" - 검증: {len(val_files)}개")
811
812     # 3. 데이터셋 생성
813     train_dataset = ThreeClassAudioDataset(
814         train_files, train_labels, augment=True
815     )
816     val_dataset = ThreeClassAudioDataset(
817         val_files, val_labels, augment=False
818     )
819
820     # 4. 데이터 로더 생성
821     train_loader = DataLoader(
822         train_dataset, batch_size=batch_size, shuffle=True,

```

```

823         num_workers=0, pin_memory=False
824     )
825     val_loader = DataLoader(
826         val_dataset, batch_size=batch_size, shuffle=False,
827         num_workers=0, pin_memory=False
828     )
829
830     # 5. 모델 생성
831     model = ThreeClassAudioCNN(num_classes=3)
832     print(f" 📄 모델 파라미터 수: {sum(p.numel() for p in model.parameters())}")
833
834     # 6. 학습 실행
835     trainer = ThreeClassTrainer(model, train_loader, val_loader)
836     trained_model = trainer.train(num_epochs=num_epochs)
837
838     # 7. 결과 시각화
839     trainer.plot_training_history()
840
841     # 8. 혼동행렬 생성
842     plot_confusion_matrix(trained_model, val_loader)
843
844     return trained_model, trainer
845
846 def plot_confusion_matrix(model, val_loader):
847     """혼동행렬 시각화 - 한글 지원"""
848     model.eval()
849     device = next(model.parameters()).device
850
851     all_preds = []
852     all_labels = []
853
854     with torch.no_grad():
855         for batch in val_loader:
856             inputs = batch['features'].to(device)
857             labels = batch['label'].squeeze().to(device)
858
859             outputs = model(inputs)
860             _, predicted = outputs.max(1)
861
862             all_preds.extend(predicted.cpu().numpy())
863             all_labels.extend(labels.cpu().numpy())
864
865     # 혼동행렬 계산
866     cm = confusion_matrix(all_labels, all_preds)
867     class_names = ['발소리', '말소리', '가구끄는소리']
868
869     # 시각화
870     plt.figure(figsize=(10, 8))
871
872     # 한글 폰트 확인 및 설정
873     try:
874         plt.rcParams['font.family'] = 'NanumGothic'
875     except:
876         # 폰트 설정이 안된 경우 영어로 대체
877         class_names = ['Footstep', 'Speech', 'Furniture']
878
879     sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
880                 xticklabels=class_names, yticklabels=class_names,
881                 cbar_kws={'label': 'Frequency'})
882     plt.title('혼동 행렬 (Confusion Matrix)', fontsize=16, pad=20)
883     plt.xlabel('예측값 (Predicted)', fontsize=12)
884     plt.ylabel('실제값 (Actual)', fontsize=12)
885     plt.tight_layout()
886     plt.show()
887
888     # 분류 리포트
889     report = classification_report(all_labels, all_preds,
890                                   target_names=class_names, output_dict=True)
891
892     print(f"\n 📊 상세 분류 결과:")
893     for i, class_name in enumerate(class_names):
894         precision = report[class_name]['precision']
895         recall = report[class_name]['recall']
896         f1 = report[class_name]['f1-score']
897         support = report[class_name]['support']
898
899         print(f" {class_name}:")
900         print(f" - 정밀도(Precision): {precision:.3f}")
901         print(f" - 재현율(Recall): {recall:.3f}")
902         print(f" - F1-Score: {f1:.3f}")
903         print(f" - 샘플 수: {support}")
904
905     print(f"\n 전체 정확도: {report['accuracy']:.3f}")
906     print(f" 매크로 평균 F1: {report['macro avg']['f1-score']:.3f}")
907
908     # 클래스별 정확도 바 차트
909     plt.figure(figsize=(10, 6))
910     class_accuracies = []
911     for i in range(len(class_names)):
912         if cm[i].sum() > 0:
913             acc = cm[i, i] / cm[i].sum()
914             class_accuracies.append(acc)
915     return class_accuracies

```

```

class accuracies.append(0)

colors = ['#ff7f7f', '#7f7fff', '#7fff7f']
bars = plt.bar(class_names, class_accuracies, color=colors, alpha=0.8)
plt.title('클래스별 정확도', fontsize=16, pad=20)
plt.xlabel('클래스', fontsize=12)
plt.ylabel('정확도', fontsize=12)
plt.ylim(0, 1.1)

# 각 막대 위에 정확도 값 표시
for bar, acc in zip(bars, class_accuracies):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.02,
             f'{acc:.2%}', ha='center', va='bottom', fontsize=12, fontweight='bold')

plt.grid(True, axis='y', alpha=0.3)
plt.tight_layout()
plt.show()

# 7. 실시간 예측 함수 (완성)

def predict_audio_file(model_path, audio_file_path):
    """오디오 파일 예측 - 완성 버전"""
    # 모델 로드
    checkpoint = torch.load(model_path, map_location='cpu')
    model = ThreeClassAudioCNN(num_classes=3)
    model.load_state_dict(checkpoint['model_state_dict'])
    model.eval()

    class_names = ['발소리', '말소리', '가구끄는소리']
    class_names_eng = ['footstep', 'speech', 'furniture']

    try:
        # 오디오 로드 및 전처리
        audio, sr = librosa.load(audio_file_path, sr=16000, duration=5.0)

        # 길이 정규화
        max_length = 16000 * 5
        if len(audio) < max_length:
            audio = np.pad(audio, (0, max_length - len(audio)))
        else:
            audio = audio[:max_length]

        # 특징 추출
        mel_spec = librosa.feature.melspectrogram(
            y=audio, sr=16000, n_mels=64, fmax=8000, hop_length=512, n_fft=2048
        )
        mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)

        mfcc = librosa.feature.mfcc(
            y=audio, sr=16000, n_mfcc=64, hop_length=512, n_fft=2048
        )

        # 특징 결합
        features = np.vstack([mel_spec_db, mfcc]) # (128, time_frames)

        # 크기 조정
        target_frames = 157
        if features.shape[1] != target_frames:
            from scipy.ndimage import zoom
            zoom_factor = target_frames / features.shape[1]
            features = zoom(features, (1, zoom_factor))

        # 예측
        with torch.no_grad():
            features_tensor = torch.FloatTensor(features).unsqueeze(0) # 배치 자원 추가
            outputs = model(features_tensor)
            probabilities = F.softmax(outputs, dim=1)
            predicted_class = torch.argmax(outputs, dim=1).item()
            confidence = probabilities[0, predicted_class].item()

        # 결과 반환
        result = {
            'predicted_class': class_names[predicted_class],
            'predicted_class_eng': class_names_eng[predicted_class],
            'confidence': confidence,
            'probabilities': {
                class_names[i]: prob.item()
                for i, prob in enumerate(probabilities[0])
            }
        }

        # 시각화
        visualize_prediction_result(audio, sr, features, result)

    except Exception as e:
        print(f"❌ 예측 오류: {str(e)}")
        return None

```

```

def visualize_prediction_result(audio, sr, features, result):
    """예측 결과 시각화"""
    fig = make_subplots(
        rows=2, cols=2,
        subplot_titles=('오디오 파형', '특징 맵 (Mel-spec + MFCC)', '예측 확률', '주파수 스펙트럼'),
        specs=[[{"secondary_y": False}, {"secondary_y": False}],
        [{"type": "bar"}, {"secondary_y": False}]]

    # 1. 오디오 파형
    time = np.linspace(0, len(audio)/sr, len(audio))
    fig.add_trace(
        go.Scatter(x=time, y=audio, name='오디오 신호', line=dict(color='blue')),
        row=1, col=1
    )

    # 2. 특징 맵
    fig.add_trace(
        go.Heatmap(z=features, colorscale='Viridis', name='특징 맵'),
        row=1, col=2
    )

    # 3. 예측 확률
    classes = list(result['probabilities'].keys())
    probs = list(result['probabilities'].values())
    colors = ['red' if cls == result['predicted_class'] else 'lightblue' for cls in classes]

    fig.add_trace(
        go.Bar(x=classes, y=probs, name='예측 확률',
              marker=dict(color=colors)),
        row=2, col=1
    )

    # 4. 주파수 스펙트럼
    freqs = np.fft.fftfreq(len(audio), 1/sr)[:len(audio)//2]
    fft_vals = np.abs(np.fft.fft(audio))[:len(audio)//2]

    fig.add_trace(
        go.Scatter(x=freqs, y=fft_vals, name='주파수 스펙트럼', line=dict(color='green')),
        row=2, col=2
    )

    fig.update_layout(
        height=800,
        title_text=f'예측 결과: {result["predicted_class"]} (신뢰도: {result["confidence"]:.2f})'
    )
    fig.show()

# 결과 출력
print(f"\n 🎯 예측 결과:")
print(f"   - 예측 클래스: {result['predicted_class']}")
print(f"   - 신뢰도: {result['confidence']:.3f}")
print(f"\n 📊 모든 클래스 확률:")
for class_name, prob in result['probabilities'].items():
    print(f"   - {class_name}: {prob:.3f}")

# 8. 샘플 데이터 생성기

def generate_sample_audio_data(output_dir='sample_data', samples_per_class=20):
    """테스트용 샘플 오디오 데이터 생성"""
    print("🎵 샘플 오디오 데이터 생성 중...")

    os.makedirs(output_dir, exist_ok=True)

    # 각 클래스별 폴더 생성
    classes = ['footstep', 'speech', 'furniture']
    for class_name in classes:
        os.makedirs(os.path.join(output_dir, class_name), exist_ok=True)

    sr = 16000
    duration = 3.0
    t = np.linspace(0, duration, int(sr * duration))

    for i in range(samples_per_class):
        # 발소리 시뮬레이션 (짧은 충격음들)
        footstep = np.zeros_like(t)
        for step in range(4): # 4번의 발걸음
            start_idx = int(step * len(t) / 4) + np.random.randint(-1000, 1000)
            if 0 <= start_idx < len(t) - 1000:
                # 충격을 시뮬레이션
                impact = np.exp(-np.arange(1000) * 0.01) * np.sin(2 * np.pi * np.random.uniform(80, 200) * np.arange(1000) / sr)
                footstep[start_idx:start_idx+1000] += impact * np.random.uniform(0.3, 0.8)

        # 노이즈 추가
        footstep += np.random.normal(0, 0.02, len(footstep))
        footstep = np.clip(footstep, -1.0, 1.0)

        sf.write(os.path.join(output_dir, 'footstep', f'footstep_{i:03d}.wav'), footstep, sr)

    # 말소리 시뮬레이션 (여러 주파수 성분)
    speech = np.zeros_like(t)
    # ...

```

```

1101 # 기본 음성 주파수들 (100~300Hz)
1102 for freq in [120, 180, 240, 300]:
1103     amplitude = np.random.uniform(0.1, 0.3)
1104     speech += amplitude * np.sin(2 * np.pi * freq * t)
1105
1106 # 포먼트 시뮬레이션 (800~2000Hz)
1107 for freq in [800, 1200, 1600, 2000]:
1108     amplitude = np.random.uniform(0.05, 0.15)
1109     modulation = 1 + 0.5 * np.sin(2 * np.pi * np.random.uniform(5, 15) * t)
1110     speech += amplitude * np.sin(2 * np.pi * freq * t) * modulation
1111
1112 # 노이즈 및 변조
1113 speech *= (1 + 0.3 * np.sin(2 * np.pi * np.random.uniform(1, 5) * t))
1114 speech += np.random.normal(0, 0.02, len(speech))
1115 speech = np.clip(speech, -1.0, 1.0)
1116
1117 sf.write(os.path.join(output_dir, 'speech', f'speech_{i:03d}.wav'), speech, sr)
1118
1119 # 가구끄는소리 시뮬레이션 (마찰음)
1120 furniture = np.zeros_like(t)
1121
1122 # 마찰음 기본 주파수 (낮은 주파수 + 고주파 노이즈)
1123 base_freq = np.random.uniform(20, 80)
1124 furniture += 0.4 * np.sin(2 * np.pi * base_freq * t)
1125
1126 # 고주파 마찰음 (1-4kHz)
1127 high_freq_noise = np.random.normal(0, 0.1, len(t))
1128 butter_b, butter_a = scipy.signal.butter(4, [1000, 4000], btype='band', fs=sr)
1129 high_freq_filtered = scipy.signal.filtfilt(butter_b, butter_a, high_freq_noise)
1130 furniture += 0.3 * high_freq_filtered
1131
1132 # 불규칙한 진동 패턴
1133 irregular_pattern = np.random.uniform(0.5, 1.5, 100)
1134 furniture *= np.interp(t, np.linspace(0, duration, 100), irregular_pattern)
1135
1136 furniture = np.clip(furniture, -1.0, 1.0)
1137
1138 sf.write(os.path.join(output_dir, 'furniture', f'furniture_{i:03d}.wav'), furniture, sr)
1139
1140 print(f"✅ 샘플 데이터 생성 완료!")
1141 print(f" - 경로: {output_dir}")
1142 print(f" - 클래스별 {samples_per_class}개씩 총 {samples_per_class * 3}개 파일")
1143
1144 return output_dir
1145
1146 # =====
1147 # 9. 실시간 녹음 및 예측
1148 # =====
1149
1150 def record_and_predict(model_path, duration=5):
1151     """실시간 녹음 및 예측"""
1152     print(f"🎧 {duration}초간 녹음을 시작합니다...")
1153     print(" 마이크에 대고 소리를 내세요!")
1154
1155     try:
1156         # JavaScript를 사용한 녹음 (Colab 환경)
1157         from google.colab import output
1158         from base64 import b64decode
1159
1160         RECORD = ""
1161         const sleep = time => new Promise(resolve => setTimeout(resolve, time))
1162         const b2text = blob => new Promise(resolve => {
1163             const reader = new FileReader()
1164             reader.onloadend = e => resolve(e.srcElement.result)
1165             reader.readAsDataURL(blob)
1166         })
1167
1168         var record = time => new Promise(async resolve => {
1169             stream = await navigator.mediaDevices.getUserMedia({ audio: true })
1170             recorder = new MediaRecorder(stream)
1171             chunks = []
1172             recorder.ondataavailable = e => chunks.push(e.data)
1173             recorder.onstop = async ()=>{
1174                 blob = new Blob(chunks, { type: 'audio/wav' })
1175                 text = await b2text(blob)
1176                 resolve(text)
1177             }
1178             recorder.start()
1179             await sleep(time)
1180             recorder.stop()
1181         })
1182         ""
1183
1184         display(HTML(f`
1185             <script>
1186             {RECORD}
1187             </script>
1188             <button onclick="record({duration} * 1000)).then(audio => {{
1189                 google.colab.kernel.invokeFunction('save_audio', [audio], {{}})
1190             }})">🎧 녹음 시작 ({duration}초)</button>
1191             `))
1192
1193         print(" 위의 녹음 버튼을 클릭해주세요!")

```

```

1194
1195 except Exception as e:
1196     print(f"❌ 녹음 기능 오류: {e}")
1197     print(" 대신 파일 업로드를 사용해주세요.")
1198
1199 def save_audio(audio_data):
1200     """녹음된 오디오 저장 및 예측"""
1201     try:
1202         # Base64 디코딩
1203         audio_data = audio_data.split(',')[1]
1204         audio_bytes = b64decode(audio_data)
1205
1206         # 파일 저장
1207         with open('recorded_audio.wav', 'wb') as f:
1208             f.write(audio_bytes)
1209
1210         print("✅ 녹음 완료! 예측 중...")
1211
1212         # 예측 실행 (모델이 있다면)
1213         if os.path.exists('best_three_class_model.pth'):
1214             result = predict_audio_file('best_three_class_model.pth', 'recorded_audio.wav')
1215             if result:
1216                 print_prediction_result(result)
1217         else:
1218             print("⚠️ 학습된 모델이 없습니다. 먼저 학습을 실행해주세요.")
1219
1220     except Exception as e:
1221         print(f"❌ 오디오 저장 오류: {e}")
1222
1223 def print_prediction_result(result):
1224     """예측 결과 출력"""
1225     print("\n" + "="*50)
1226     print(f"🎯 실시간 예측 결과")
1227     print(f"="*50)
1228     print(f"🔍 예측 클래스: {result['predicted_class']}")
1229     print(f"📊 신뢰도: {result['confidence']:.1%}")
1230     print(f"\n📋 각 클래스별 확률:")
1231
1232     for class_name, prob in result['probabilities'].items():
1233         bar = "█" * int(prob * 20)
1234         print(f"    {class_name:12}: {prob:.3f} {bar}")
1235
1236 # =====
1237 # 10. 업로드 파일 분석
1238 # =====
1239
1240 def analyze_uploaded_file(path=None):
1241     """파일 업로드 및 분석 - 경로 지정 가능"""
1242
1243     if path is not None:
1244         # 경로가 지정된 경우: 폴더 또는 파일 분석
1245         if os.path.isfile(path):
1246             # 단일 파일 분석
1247             print(f"📁 지정된 파일 분석: {path}")
1248             file_name = path
1249
1250         elif os.path.isdir(path):
1251             # 폴더에서 파일 선택하여 분석
1252             print(f"📁 폴더에서 파일 선택: {path}")
1253
1254             # 폴더 내 오디오 파일 찾기
1255             audio_extensions = ['.wav', '.mp3', '.flac', '.m4a', '.ogg']
1256             audio_files = []
1257
1258             for root, dirs, files in os.walk(path):
1259                 for file in files:
1260                     if any(file.lower().endswith(ext) for ext in audio_extensions):
1261                         audio_files.append(os.path.join(root, file))
1262
1263             if not audio_files:
1264                 print(f"❌ 지정된 폴더에서 오디오 파일을 찾을 수 없습니다!")
1265                 return None
1266
1267             print(f"📁 {len(audio_files)}개 오디오 파일 발견")
1268
1269             # 처음 몇 개 파일 표시
1270             print(f"📄 발견된 파일들 (처음 10개):")
1271             for i, file_path in enumerate(audio_files[:10]):
1272                 print(f"    {i+1}. {os.path.basename(file_path)}")
1273
1274             if len(audio_files) > 10:
1275                 print(f"    ... 그 외 {len(audio_files)-10}개 파일")
1276
1277             # 첫 번째 파일로 분석 실행
1278             file_name = audio_files[0]
1279             print(f"\n🎧 첫 번째 파일로 분석 실행: {os.path.basename(file_name)}")
1280
1281         else:
1282             print(f"❌ 지정된 경로를 찾을 수 없습니다. {path}")
1283             return None
1284
1285     else:
1286         # 기존 업로드 방식
1287         print(f"📁 오디오 파일을 업로드해주세요...")

```



```
1287         uploaded = files.upload()
1288
1289         if uploaded:
1290             file_name = list(uploaded.keys())[0]
1291             print(f"👍 파일 '{file_name}' 업로드 완료!")
1292         else:
1293             print(f"❌ 파일이 업로드되지 않았습니다.")
1294         return None
1295
1296     # 모델이 있는지 확인
1297     model_path = 'best_three_class_model.pth'
1298     if not os.path.exists(model_path):
1299         print(f"⚠️ 학습된 모델이 없습니다.")
1300         print("  먼저 샘플 데이터를 학습하거나 실제 데이터로 학습해주세요.")
1301         print("  run_sample_training() 실행하세요.")
1302         return None
1303
1304     # 파일 분석 실행
1305     try:
1306         result = visualize_audio_prediction(
1307             model_path,
1308             file_name,
1309             save_image=True,
1310             show_features=True
1311         )
1312
1313         if result:
1314             print_prediction_result(result)
1315             return result
1316         else:
1317             print(f"❌ 파일 분석에 실패했습니다.")
1318             return None
1319
1320     except Exception as e:
1321         print(f"❌ 분석 중 오류 발생: {str(e)}")
1322         return None
1323
1324 def analyze_folder_batch(folder_path, max_files=10, save_images=False):
1325     """폴더 내 모든 오디오 파일 일괄 분석"""
1326     print(f"📁 폴더 일괄 분석: {folder_path}")
1327
1328     if not os.path.exists('best_three_class_model.pth'):
1329         print(f"❌ 학습된 모델이 없습니다!")
1330         print("  먼저 run_sample_training()을 실행하세요.")
1331         return None
1332
1333     # 오디오 파일 찾기
1334     audio_extensions = ['.wav', '.mp3', '.flac', '.m4a', '.ogg']
1335     audio_files = []
1336
1337     for root, dirs, files in os.walk(folder_path):
1338         for file in files:
1339             if any(file.lower().endswith(ext) for ext in audio_extensions):
1340                 audio_files.append(os.path.join(root, file))
1341
1342     if not audio_files:
1343         print(f"❌ 오디오 파일을 찾을 수 없습니다!")
1344         return None
1345
1346     print(f"📄 {len(audio_files)}개 파일 발견, {min(max_files, len(audio_files))}개 분석 예정")
1347
1348     # 일괄 분석 실행
1349     results = batch_visualize_predictions(
1350         'best_three_class_model.pth',
1351         audio_files[:max_files],
1352         max_files=max_files,
1353         save_images=save_images
1354     )
1355
1356     return results
1357
1358 def quick_file_check(file_or_folder_path):
1359     """파일 또는 폴더 빠른 확인"""
1360     print(f"🔍 경로 확인: {file_or_folder_path}")
1361
1362     if not os.path.exists(file_or_folder_path):
1363         print(f"❌ 경로가 존재하지 않습니다!")
1364         return False
1365
1366     if os.path.isfile(file_or_folder_path):
1367         print(f"📄 파일 확인됨: {os.path.basename(file_or_folder_path)}")
1368         file_size = os.path.getsize(file_or_folder_path) / 1024 # KB
1369         print(f"  크기: {file_size:.1f} KB")
1370
1371         # 오디오 파일인지 확인
1372         audio_extensions = ['.wav', '.mp3', '.flac', '.m4a', '.ogg']
1373         is_audio = any(file_or_folder_path.lower().endswith(ext) for ext in audio_extensions)
1374
1375         if is_audio:
1376             print(f"  형식: 오디오 파일 ✅")
1377
1378         # 간단한 오디오 정보
1379         try:
```

```
1380         audio, sr = librosa.load(file_or_folder_path, sr=None, duration=1.0)
1381         duration_estimate = len(audio) / sr
1382         print(f"  추정 길이: {duration_estimate:.1f}초")
1383         print(f"  샘플링 레이트: {sr} Hz")
1384         except Exception as e:
1385             print(f"  오디오 로딩 오류: {e}")
1386     else:
1387         print(f"  형식: 오디오 파일이 아님 ❌")
1388
1389     return is_audio
1390
1391 elif os.path.isdir(file_or_folder_path):
1392     print(f"📁 폴더 확인됨")
1393
1394     # 폴더 내 오디오 파일 개수 확인
1395     audio_extensions = ['.wav', '.mp3', '.flac', '.m4a', '.ogg']
1396     audio_count = 0
1397     total_size = 0
1398
1399     for root, dirs, files in os.walk(file_or_folder_path):
1400         for file in files:
1401             if any(file.lower().endswith(ext) for ext in audio_extensions):
1402                 audio_count += 1
1403                 try:
1404                     file_path = os.path.join(root, file)
1405                     total_size += os.path.getsize(file_path)
1406                 except:
1407                     pass
1408
1409     print(f"  - 오디오 파일 수: {audio_count}개")
1410     print(f"  - 총 크기: {total_size/(1024*1024):.1f} MB")
1411
1412     if audio_count > 0:
1413         print(f"  - 분석 가능 ✅")
1414     else:
1415         print(f"  - 오디오 파일 없음 ❌")
1416
1417     return audio_count > 0
1418
1419 return False
1420
1421 # =====
1422 # 11. 데모 및 테스트 함수들
1423 # =====
1424
1425 def debug_data_loading(data_path):
1426     """데이터 로딩 과정 디버깅"""
1427     print(f"🔍 데이터 로딩 과정 디버깅 중...")
1428
1429     # 1. 전체 파일 스캔
1430     all_files, dir_groups = force_scan_all_audio_files(data_path)
1431     print(f"📄 전체 스캔: {len(all_files)}개 파일 발견")
1432
1433     # 2. 클래스 매칭 테스트
1434     audio_files, labels = scan_three_class_data(data_path)
1435     print(f"📄 클래스 매칭: {len(audio_files)}개 파일 매칭")
1436
1437     # 3. 누락된 파일들 분석
1438     matched_paths = set(audio_files)
1439     all_paths = set([f['path'] for f in all_files])
1440     missing_files = all_paths - matched_paths
1441
1442     if missing_files:
1443         print(f"🚫 누락된 파일들 ({len(missing_files)}개):")
1444         missing_by_dir = defaultdict(list)
1445         for missing_path in list(missing_files)[:20]: # 처음 20개만 표시
1446             dir_name = os.path.dirname(os.path.relpath(missing_path, data_path))
1447             missing_by_dir[dir_name].append(os.path.basename(missing_path))
1448
1449         for dir_name, files in missing_by_dir.items():
1450             print(f"📁 {dir_name}: {len(files)}개")
1451             for file in files[:3]:
1452                 print(f"  - {file}")
1453             if len(files) > 3:
1454                 print(f"    ... 그 외 {len(files)-3}개")
1455
1456     # 4. 실제 로딩 테스트
1457     print(f"🔄 실제 오디오 로딩 테스트 (처음 10개 파일)...")
1458     loading_errors = 0
1459
1460     for i, file_path in enumerate(audio_files[:10]):
1461         try:
1462             audio, sr = librosa.load(file_path, sr=16000, duration=1.0) # 1초만 테스트
1463             duration = len(audio) / sr
1464             print(f"  ✅ {i+1}: {os.path.basename(file_path)} ( {duration:.1f}초, {sr}Hz )")
1465         except Exception as e:
1466             print(f"  ❌ {i+1}: {os.path.basename(file_path)} - {str(e)}")
1467             loading_errors += 1
1468
1469     if loading_errors > 0:
1470         print(f"⚠️ {loading_errors}개 파일에서 로딩 오류 발생")
1471         print(f"  일부 파일이 손상되었거나 지원되지 않는 형식일 수 있습니다.")
1472
```

```
1473     return {
1474         'total_found': len(all_files),
1475         'matched': len(audio_files),
1476         'missing': len(missing_files),
1477         'loading_errors': loading_errors,
1478         'dir_groups': dir_groups
1479     }
1480
1481 def create_manual_dataset(data_path):
1482     """수동으로 데이터셋 생성 (모든 파일 사용)"""
1483     print("📁 수동 데이터셋 생성 중...")
1484
1485     # 모든 파일 스캔
1486     all_files, dir_groups = force_scan_all_audio_files(data_path)
1487
1488     audio_files = []
1489     labels = []
1490
1491     print(f"\n각 디렉토리의 클래스를 수동으로 지정해주세요.")
1492     print("1: footstep (발소리)")
1493     print("2: speech (말소리)")
1494     print("3: furniture (가구끄는소리)")
1495     print("0: skip (제외)")
1496
1497     class_mapping = {'1': 'footstep', '2': 'speech', '3': 'furniture'}
1498
1499     for dir_name, file_list in sorted(dir_groups.items(), key=lambda x: len(x[1]), reverse=True):
1500         print(f"\n📁 {dir_name} ({len(file_list)}개 파일)")
1501         print(f"   샘플 파일: {', '.join([f'{filename}' for f in file_list[:3]])}")
1502
1503         while True:
1504             try:
1505                 choice = input(f"   클래스 선택 (1/2/3/0): ").strip()
1506                 if choice == '0':
1507                     print(f"   🚫 {dir_name} 폴더 제외")
1508                     break
1509                 elif choice in class_mapping:
1510                     selected_class = class_mapping[choice]
1511                     print(f"   ✅ {dir_name} -> {selected_class}")
1512
1513                     # 파일들 추가
1514                     for file_info in file_list:
1515                         audio_files.append(file_info['path'])
1516                         labels.append(selected_class)
1517                     break
1518                 else:
1519                     print("   잘못된 선택입니다. 다시 입력해주세요.")
1520             except KeyboardInterrupt:
1521                 print("\n   작업이 중단되었습니다.")
1522                 return [], []
1523
1524     print(f"\n📁 수동 데이터셋 생성 완료.")
1525     print(f"   - 총 파일 수: {len(audio_files)}개")
1526     for class_name in ['footstep', 'speech', 'furniture']:
1527         count = labels.count(class_name)
1528         percentage = (count / len(labels) * 100) if len(labels) > 0 else 0
1529         print(f"   - {class_name}: {count}개 ({percentage:.1f}%)")
1530
1531     return audio_files, labels
1532     """샘플 데이터 또는 실제 데이터로 학습 실행"""
1533
1534     if data_path is None:
1535         print("📁 샘플 데이터를 생성하고 학습을 시작합니다...")
1536         # 샘플 데이터 생성
1537         sample_dir = generate_sample_audio_data('sample_data', samples_per_class=30)
1538         data_path = sample_dir
1539         print("✅ 샘플 데이터 생성 완료!")
1540     else:
1541         print(f"📁 실제 데이터를 사용합니다: {data_path}")
1542
1543     # 학습 실행
1544     print("🚀 학습을 시작합니다...")
1545     model, trainer = run_three_class_training(
1546         data_path=data_path,
1547         num_epochs=num_epochs,
1548         batch_size=batch_size,
1549         test_size=0.2
1550     )
1551
1552     print("✅ 학습 완료! 이제 테스트해보세요.")
1553     return model, trainer
1554
1555 def test_model_with_samples():
1556     """샘플로 모델 테스트"""
1557     if not os.path.exists('best_three_class_model.pth'):
1558         print("❌ 학습된 모델이 없습니다!")
1559         print("   run_sample_training() 먼저 실행해주세요.")
1560         return
1561
1562     if not os.path.exists('sample_data'):
1563         print("❌ 샘플 데이터가 없습니다!")
1564         return
1565
```

```
1566     print("🎧 샘플 파일들로 모델 테스트 중...")
1567
1568     # 각 클래스에서 랜덤 파일 선택
1569     classes = ['footstep', 'speech', 'furniture']
1570
1571     for class_name in classes:
1572         class_dir = os.path.join('sample_data', class_name)
1573         if os.path.exists(class_dir):
1574             files_list = [f for f in os.listdir(class_dir) if f.endswith('.wav')]
1575             if files_list:
1576                 test_file = os.path.join(class_dir, random.choice(files_list))
1577                 print(f"\n🎧 테스트 중: {class_name} 클래스")
1578                 print(f"   파일: {test_file}")
1579
1580                 result = predict_audio_file('best_three_class_model.pth', test_file)
1581                 if result:
1582                     correct = "✅" if result['predicted_class_eng'] == class_name else "❌"
1583                     print(f"   {correct} 예측: {result['predicted_class']} (신뢰도: {result['confidence']:.2f})")
1584
1585                 # 오디오 재생
1586                 audio, sr = librosa.load(test_file, sr=16000)
1587                 display(Audio(audio, rate=sr))
1588
1589 def demo_realtime_features():
1590     """실시간 특징 추출 데모"""
1591     print("🎧 실시간 특징 추출 데모")
1592
1593     # 짧은 테스트 신호 생성
1594     sr = 16000
1595     duration = 2.0
1596     t = np.linspace(0, duration, int(sr * duration))
1597
1598     # 다양한 신호 생성
1599     signals = {
1600         '발소리 시뮬레이션': create_footstep_signal(t, sr),
1601         '말소리 시뮬레이션': create_speech_signal(t, sr),
1602         '가구 시뮬레이션': create_furniture_signal(t, sr)
1603     }
1604
1605     for name, signal in signals.items():
1606         print(f"\n🎧 {name} 특징 추출 중...")
1607
1608         # 특징 추출
1609         mel_spec = librosa.feature.melspectrogram(y=signal, sr=sr, n_mels=64)
1610         mfcc = librosa.feature.mfcc(y=signal, sr=sr, n_mfcc=13)
1611
1612         # 시각화
1613         fig, axes = plt.subplots(1, 3, figsize=(15, 4))
1614
1615         # 원본 신호
1616         axes[0].plot(t, signal)
1617         axes[0].set_title(f'{name} - 시간 도메인')
1618         axes[0].set_xlabel('시간 (초)')
1619         axes[0].set_ylabel('진폭')
1620
1621         # Mel-spectrogram
1622         librosa.display.specshow(librosa.power_to_db(mel_spec), sr=sr, x_axis='time', y_axis='mel', ax=axes[1])
1623         axes[1].set_title('Mel-spectrogram')
1624
1625         # MFCC
1626         librosa.display.specshow(mfcc, sr=sr, x_axis='time', ax=axes[2])
1627         axes[2].set_title('MFCC')
1628
1629         plt.tight_layout()
1630         plt.show()
1631
1632         # 오디오 재생
1633         display(Audio(signal, rate=sr))
1634
1635 def create_footstep_signal(t, sr):
1636     """발소리 신호 생성"""
1637     signal = np.zeros_like(t)
1638     step_times = [0.3, 0.9, 1.5] # 발걸음 시간
1639
1640     for step_time in step_times:
1641         start_idx = int(step_time * sr)
1642         if start_idx < len(signal) - 2000:
1643             # 충격음 (감쇠하는 지수파)
1644             impact_t = np.arange(2000) / sr
1645             impact = np.exp(-impact_t * 5) * np.sin(2 * np.pi * 80 * impact_t)
1646             signal[start_idx:start_idx+2000] += impact * 0.8
1647
1648     return signal
1649
1650 def create_speech_signal(t, sr):
1651     """말소리 신호 생성"""
1652     # 기본 주파수 (피치)
1653     f0 = 150 # Hz
1654     speech = 0.3 * np.sin(2 * np.pi * f0 * t)
1655
1656     # 포먼트 추가
1657     formants = [800, 1200, 2400]
1658     for formant in formants:

```

```
1659         speech += 0.1 * np.sin(2 * np.pi * formant * t)
1660
1661     # 진폭 변조 (말하는 리듬)
1662     modulation = 1 + 0.5 * np.sin(2 * np.pi * 3 * t)
1663     speech *= modulation
1664
1665     return speech
1666
1667 def create_furniture_signal(t, sr):
1668     """가구끄는소리 신호 생성"""
1669     # 마찰음 (광대역 노이즈를 필터링)
1670     noise = np.random.normal(0, 1, len(t))
1671
1672     # 로우패스 필터 (마찰음 특성)
1673     butter_b, butter_a = scipy.signal.butter(4, 500, fs=sr)
1674     filtered = scipy.signal.filtfilt(butter_b, butter_a, noise)
1675
1676     # 불규칙한 진폭
1677     amplitude_env = np.random.uniform(0.2, 0.8, 50)
1678     amplitude = np.interp(t, np.linspace(0, t[-1], 50), amplitude_env)
1679
1680     return filtered * amplitude * 0.5
1681
1682 # =====
1683 # 12. 사용 가이드 및 실행 함수
1684 # =====
1685
1686 def show_complete_usage_guide():
1687     """완전한 사용 가이드"""
1688     print("""
1689     📖 발소리-말소리-가구끄는소리 분류 AI 모델 사용 가이드
1690     =====
1691
1692     📌 주요 기능:
1693     1. 3클래스 오디오 분류 (발소리, 말소리, 가구끄는소리)
1694     2. CNN 기반 딥러닝 모델
1695     3. 실시간 예측 및 시각화
1696     4. 데이터 증강 및 최적화
1697     5. 한글 폰트 지원
1698
1699     🚀 빠른 시작:
1700
1701     0 한글 폰트 테스트:
1702     test_korean_font() # 한글이 제대로 표시되는지 확인
1703
1704     1 샘플 데이터로 빠른 테스트:
1705     run_sample_training() # 샘플 생성 + 학습
1706     test_model_with_samples() # 테스트
1707
1708     2 실제 데이터로 학습:
1709     # 데이터 폴더 구조:
1710     # your_data/
1711     # ├── footstep/ (발소리 파일들)
1712     # ├── speech/ (말소리 파일들)
1713     # └── furniture/ (가구끄는소리 파일들)
1714
1715     model, trainer = run_sample_training('/path/to/your_data')
1716
1717     3 파일 업로드하여 예측:
1718     analyze_uploaded_file()
1719
1720     4 실시간 녹음 예측:
1721     record_and_predict('best_three_class_model.pth')
1722
1723     5 특징 추출 데모:
1724     demo_realtime_features()
1725
1726     6 모델 성능 상세 분석:
1727     analyze_model_performance(model, val_loader)
1728
1729     📊 모델 성능:
1730     - 입력: 5초 오디오 (16kHz)
1731     - 특징: Mel-spectrogram (64) + MFCC (64) = 128차원
1732     - 구조: CNN (4블록) + Global Average Pooling
1733     - 출력: 3클래스 확률 분포
1734
1735     🌐 한글 폰트 문제 해결:
1736     - 그래프에서 한글이 깨진다면: test_korean_font() 실행
1737     - 여전히 문제가 있다면: 런터일 재시작 후 다시 실행
1738
1739     💡 팁:
1740     - GPU 사용 시 batch_size를 16으로 증가 가능
1741     - 데이터가 부족하면 augmentation 강화
1742     - 과적합 시 dropout_rate 증가
1743
1744     ⚡ 지금 시작하기:
1745     test_korean_font() # 한글 폰트 확인
1746     run_sample_training() # 샘플 학습
1747     """)
1748
1749 # 컴포넌트별 등록 (Colab 전용)
1750 try:
1751     from google.colab import output
```

```
1752     output.register_callback('save_audio', save_audio)
1753 except:
1754     pass
1755
1756 # =====
1757 # 13. 고급 분석 도구
1758 # =====
1759
1760 def analyze_model_performance(model, val_loader):
1761     """모델 성능 상세 분석 - 한글 지원"""
1762     model.eval()
1763     device = next(model.parameters()).device
1764
1765     all_preds = []
1766     all_labels = []
1767     all_confidences = []
1768
1769     print("🔍 모델 성능 상세 분석 중...")
1770
1771     with torch.no_grad():
1772         for batch in tqdm(val_loader, desc="분석"):
1773             inputs = batch['features'].to(device)
1774             labels = batch['label'].squeeze().to(device)
1775
1776             outputs = model(inputs)
1777             probabilities = F.softmax(outputs, dim=1)
1778             predicted = torch.argmax(outputs, dim=1)
1779             confidence = torch.max(probabilities, dim=1)[0]
1780
1781             all_preds.extend(predicted.cpu().numpy())
1782             all_labels.extend(labels.cpu().numpy())
1783             all_confidences.extend(confidence.cpu().numpy())
1784
1785     # 성능 메트릭 계산
1786     accuracy = accuracy_score(all_labels, all_preds)
1787     precision, recall, f1, _ = precision_recall_fscore_support(
1788         all_labels, all_preds, average='weighted'
1789     )
1790
1791     # 신뢰도 분석
1792     correct_mask = np.array(all_preds) == np.array(all_labels)
1793     correct_confidences = np.array(all_confidences)[correct_mask]
1794     incorrect_confidences = np.array(all_confidences)[~correct_mask]
1795
1796     # 결과 출력
1797     print(f"📊 전체 성능 메트릭:")
1798     print(f" - 정확도: {accuracy:.3f}")
1799     print(f" - 정밀도: {precision:.3f}")
1800     print(f" - 재현율: {recall:.3f}")
1801     print(f" - F1 점수: {f1:.3f}")
1802     print(f"📈 신뢰도 분석:")
1803     print(f" - 올바른 예측 평균 신뢰도: {correct_confidences.mean():.3f}")
1804     print(f" - 잘못된 예측 평균 신뢰도: {incorrect_confidences.mean():.3f}")
1805
1806     # 한글 폰트 설정
1807     try:
1808         plt.rcParams['font.family'] = 'NanumGothic'
1809         plt.rcParams['axes.unicode_minus'] = False
1810     except:
1811         pass
1812
1813     # 신뢰도 분포 시각화
1814     plt.figure(figsize=(12, 8))
1815
1816     # 서브플롯 1: 신뢰도 히스토그램
1817     plt.subplot(2, 2, 1)
1818     plt.hist(correct_confidences, bins=20, alpha=0.7, label='올바른 예측', color='green', density=True)
1819     plt.hist(incorrect_confidences, bins=20, alpha=0.7, label='잘못된 예측', color='red', density=True)
1820     plt.xlabel('신뢰도')
1821     plt.ylabel('밀도')
1822     plt.title('예측 신뢰도 분포')
1823     plt.legend()
1824     plt.grid(True, alpha=0.3)
1825
1826     # 서브플롯 2: 박스플롯
1827     plt.subplot(2, 2, 2)
1828     data_to_plot = [correct_confidences, incorrect_confidences]
1829     box = plt.boxplot(data_to_plot, labels=['올바른 예측', '잘못된 예측'], patch_artist=True)
1830     box['boxes'][0].set_facecolor('lightgreen')
1831     box['boxes'][1].set_facecolor('lightcoral')
1832     plt.ylabel('신뢰도')
1833     plt.title('신뢰도 박스플롯')
1834     plt.grid(True, alpha=0.3)
1835
1836     # 서브플롯 3: 클래스별 성능
1837     plt.subplot(2, 2, 3)
1838     class_names = ['발소리', '말소리', '가구끄는소리']
1839     class_f1_scores = []
1840
1841     for i in range(3):
1842         class_mask = np.array(all_labels) == i
1843         if class_mask.sum() > 0:
1844             class_preds = np.array(all_preds)[class_mask]
1845             class_labels = np.array(all_labels)[class_mask]
```

```
class_preds = np.array(class_preds)[class_mask]
class_labels = np.array(all_labels)[class_mask]
class_f1 = f1_score(class_labels, class_preds, average='binary', pos_label=1, zero_division=0)
class_f1_scores.append(class_f1)
else:
    class_f1_scores.append(0)

colors = ['#ff9999', '#66b3ff', '#99ff99']
bars = plt.bar(class_names, class_f1_scores, color=colors, alpha=0.8)
plt.ylabel('F1 점수')
plt.title('클래스별 F1 점수')
plt.ylim(0, 1.1)

# 각 막대 위에 값 표시
for bar, score in zip(bars, class_f1_scores):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.02,
             f'{score:.3f}', ha='center', va='bottom')

# 서브플롯 4: 신뢰도 vs 정확도
plt.subplot(2, 2, 4)
confidence_bins = np.linspace(0, 1, 11)
bin_accuracies = []
bin_centers = []

for i in range(len(confidence_bins) - 1):
    bin_mask = ((np.array(all_confidences) >= confidence_bins[i]) &
                (np.array(all_confidences) < confidence_bins[i+1]))

    if bin_mask.sum() > 0:
        bin_accuracy = (np.array(all_preds)[bin_mask] == np.array(all_labels)[bin_mask]).mean()
        bin_accuracies.append(bin_accuracy)
        bin_centers.append((confidence_bins[i] + confidence_bins[i+1]) / 2)

if bin_centers:
    plt.plot(bin_centers, bin_accuracies, 'bo-', linewidth=2, markersize=6)
    plt.plot([0, 1], [0, 1], 'r--', alpha=0.7, label='완벽한 보정')
    plt.xlabel('신뢰도')
    plt.ylabel('정확도')
    plt.title('신뢰도 보정 곡선')
    plt.legend()
    plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# 상세 통계
print(f"\n 상세 통계:")
print(f" - 총 예측 샘플: {len(all_labels)}개")
print(f" - 올바른 예측: {correct_mask.sum()}개 ({correct_mask.mean()*100:.1f}%)")
print(f" - 잘못된 예측: {(~correct_mask).sum()}개 ({(~correct_mask).mean()*100:.1f}%)")
print(f" - 평균 신뢰도: {np.mean(all_confidences):.3f}")
print(f" - 신뢰도 표준편차: {np.std(all_confidences):.3f}")

return {
    'accuracy': accuracy,
    'precision': precision,
    'recall': recall,
    'f1': f1,
    'correct_confidences': correct_confidences,
    'incorrect_confidences': incorrect_confidences,
    'class_f1_scores': class_f1_scores
}

def batch_predict_directory(model_path, test_dir):
    """디렉토리 내 모든 파일 일괄 예측"""
    if not os.path.exists(model_path):
        print(f"❌ 모델 파일이 없습니다!")
        return

    print(f"📁 {test_dir} 내 모든 오디오 파일 예측 중...")

    audio_extensions = ['.wav', '.mp3', '.flac', '.m4a']
    audio_files = []

    for root, dirs, files in os.walk(test_dir):
        for file in files:
            if any(file.lower().endswith(ext) for ext in audio_extensions):
                audio_files.append(os.path.join(root, file))

    if not audio_files:
        print(f"❌ 오디오 파일을 찾을 수 없습니다!")
        return

    results = []

    for audio_file in tqdm(audio_files, desc="예측"):
        result = predict_audio_file(model_path, audio_file)
        if result:
            results.append({
                'file': os.path.basename(audio_file),
                'predicted_class': result['predicted_class'],
                'confidence': result['confidence'],
                'path': audio_file
            })
        else:
            pass
```

```
})

# 결과를 DataFrame으로 정리
df = pd.DataFrame(results)

print(f"\n📊 일괄 예측 결과:")
print(df.groupby('predicted_class').agg({
    'confidence': ['count', 'mean', 'min', 'max'],
    'file': 'count'
}).round(3))

return df

# =====
# 14. 메인 실행 부분
# =====

def main():
    """메인 실행 함수"""
    print(f"🔊 발소리-말소리-가구끄는소리 분류 AI 모델")
    print(f"=" * 70)

    # 환경 설정
    setup_colab_environment()

    # 사용 가이드 출력
    show_complete_usage_guide()

    print(f"\n🎯 다음 중 하나를 선택하세요:")
    print("1. test_korean_font()      # 한글 폰트 테스트")
    print("2. run_sample_training()      # 샘플 데이터로 학습")
    print("3. analyze_uploaded_file()     # 파일 업로드 분석")
    print("4. demo_realtime_features()    # 특징 추출 데모")
    print("5. test_model_with_samples()   # 샘플로 테스트")

# 자동 실행
if __name__ == "__main__":
    main()

# =====
# 16. 오디오 데이터 시각화 및 분석 도구
# =====

def visualize_audio_prediction(model_path, audio_file_path, save_image=False, show_features=True):
    """개별 오디오 파일의 예측 과정을 상세히 시각화"""

    if not os.path.exists(model_path):
        print(f"❌ 모델 파일이 없습니다!")
        return None

    # 모델 로드
    checkpoint = torch.load(model_path, map_location='cpu')
    model = ThreeClassAudioCNN(num_classes=3)
    model.load_state_dict(checkpoint['model_state_dict'])
    model.eval()

    class_names_kr = ['발소리', '말소리', '가구끄는소리']
    class_names_en = ['footstep', 'speech', 'furniture']
    colors = ['#ff9999', '#66b3ff', '#99ff99']

    try:
        # 오디오 로드
        audio, sr = librosa.load(audio_file_path, sr=16000, duration=5.0)
        filename = os.path.basename(audio_file_path)

        # 길이 정규화
        max_length = 16000 * 5
        if len(audio) < max_length:
            audio = np.pad(audio, (0, max_length - len(audio)))
        else:
            audio = audio[:max_length]

        # 특징 추출
        mel_spec = librosa.feature.melspectrogram(
            y=audio, sr=16000, n_mels=64, fmax=8000, hop_length=512, n_fft=2048
        )
        mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)

        mfcc = librosa.feature.mfcc(
            y=audio, sr=16000, n_mfcc=64, hop_length=512, n_fft=2048
        )

        # 특징 결합
        features = np.vstack([mel_spec_db, mfcc])

        # 크기 조정
        target_frames = 157
        if features.shape[1] != target_frames:
            from scipy.ndimage import zoom
            zoom_factor = target_frames / features.shape[1]
            features = zoom(features, (1, zoom_factor))

        # 예측
        # ...
```

```
2530 with torch.no_grad():
2531     features_tensor = torch.FloatTensor(features).unsqueeze(0)
2532     outputs = model(features_tensor)
2533     probabilities = F.softmax(outputs, dim=1)
2534     predicted_class = torch.argmax(outputs, dim=1).item()
2535     confidence = probabilities[0, predicted_class].item()
2536
2537 # 시각화 생성
2538 if show_features:
2539     fig = plt.figure(figsize=(20, 16))
2540     gs = fig.add_gridspec(4, 3, hspace=0.3, wspace=0.3)
2541 else:
2542     fig = plt.figure(figsize=(16, 12))
2543     gs = fig.add_gridspec(3, 3, hspace=0.3, wspace=0.3)
2544
2545 # 한글 폰트 설정
2546 try:
2547     plt.rcParams['font.family'] = 'NanumGothic'
2548     plt.rcParams['axes.unicode_minus'] = False
2549 except:
2550     pass
2551
2552 # 1. 오디오 파형
2553 ax1 = fig.add_subplot(gs[0, :])
2554 time = np.linspace(0, len(audio)/sr, len(audio))
2555 ax1.plot(time, audio, color='blue', alpha=0.8, linewidth=0.5)
2556 ax1.set_title(f'📄 원본 오디오 파형: {filename}', fontsize=14, fontweight='bold', pad=15)
2557 ax1.set_xlabel('시간 (초)')
2558 ax1.set_ylabel('진폭')
2559 ax1.grid(True, alpha=0.3)
2560 ax1.set_xlim(0, 5)
2561
2562 # RMS와 피크값 표시
2563 rms = np.sqrt(np.mean(audio**2))
2564 peak = np.max(np.abs(audio))
2565 ax1.text(0.02, 0.95, f'RMS: {rms:.4f}\nPeak: {peak:.4f}',
2566         transform=ax1.transAxes, verticalalignment='top',
2567         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
2568
2569 # 2. 스펙트로그램 (Mel-Spectrogram)
2570 ax2 = fig.add_subplot(gs[1, 0])
2571 librosa.display.specshow(mel_spec_db, sr=sr, x_axis='time', y_axis='mel',
2572                          ax=ax2, cmap='viridis')
2573 ax2.set_title('📊 Mel-Spectrogram', fontsize=12, fontweight='bold')
2574 ax2.set_xlabel('시간 (초)')
2575 ax2.set_ylabel('Mel 주파수')
2576
2577 # 3. MFCC
2578 ax3 = fig.add_subplot(gs[1, 1])
2579 librosa.display.specshow(mfcc, sr=sr, x_axis='time', ax=ax3, cmap='coolwarm')
2580 ax3.set_title('📊 MFCC 계수', fontsize=12, fontweight='bold')
2581 ax3.set_xlabel('시간 (초)')
2582 ax3.set_ylabel('MFCC 계수')
2583
2584 # 4. 결합된 특징 맵
2585 ax4 = fig.add_subplot(gs[1, 2])
2586 im = ax4.imshow(features, aspect='auto', cmap='plasma', origin='lower')
2587 ax4.set_title('📊 CNN 입력 특징\n(Mel-spec + MFCC)', fontsize=12, fontweight='bold')
2588 ax4.set_xlabel('시간 프레임')
2589 ax4.set_ylabel('특징 차원')
2590 plt.colorbar(im, ax=ax4, shrink=0.8)
2591
2592 # 5. 예측 확률
2593 ax5 = fig.add_subplot(gs[2, 0])
2594 probs = [prob.item() for prob in probabilities[0]]
2595 bars = ax5.bar(class_names_kr, probs, color=colors, alpha=0.8, edgecolor='black', linewidth=1)
2596 ax5.set_title('📊 클래스별 예측 확률', fontsize=12, fontweight='bold')
2597 ax5.set_ylabel('확률')
2598 ax5.set_ylim(0, 1.1)
2599
2600 # 최고 확률 막대 강조
2601 max_idx = np.argmax(probs)
2602 bars[max_idx].set_color('red')
2603 bars[max_idx].set_alpha(1.0)
2604
2605 # 확률 값 표시
2606 for i, (bar, prob) in enumerate(zip(bars, probs)):
2607     height = bar.get_height()
2608     ax5.text(bar.get_x() + bar.get_width()/2., height + 0.02,
2609             f'{prob:.1%}', ha='center', va='bottom',
2610             fontweight='bold' if i == max_idx else 'normal')
2611
2612 # 6. 주파수 스펙트럼
2613 ax6 = fig.add_subplot(gs[2, 1])
2614 fft = np.fft.fft(audio)
2615 freqs = np.fft.fftfreq(len(audio), 1/sr)
2616 magnitude = np.abs(fft)
2617
2618 # 양의 주파수만 표시
2619 positive_freqs = freqs[:len(freqs)//2]
2620 positive_magnitude = magnitude[:len(magnitude)//2]
2621
2622 ax6.plot(positive_freqs, positive_magnitude, color='green', alpha=0.7)
```

```
2123 ax6.set_title('📊 주파수 스펙트럼', fontsize=12, fontweight='bold')
2124 ax6.set_xlabel('주파수 (Hz)')
2125 ax6.set_ylabel('크기')
2126 ax6.set_xlim(0, 4000) # 4kHz까지만 표시
2127 ax6.grid(True, alpha=0.3)
2128
2129 # 주요 주파수 성분 찾기
2130 dominant_freq_idx = np.argmax(positive_magnitude[1:]) + 1 # DC 제외
2131 dominant_freq = positive_freqs[dominant_freq_idx]
2132 ax6.axvline(dominant_freq, color='red', linestyle='--', alpha=0.8)
2133 ax6.text(dominant_freq, max(positive_magnitude)*0.8,
2134         f'주요 주파수\n(dominant_freq: {0f} Hz',
2135         ha='center', bbox=dict(boxstyle='round', facecolor='yellow', alpha=0.7))
2136
2137 # 7. 결과 요약
2138 ax7 = fig.add_subplot(gs[2, 2])
2139 ax7.axis('off')
2140
2141 result_text = f"""
2142 📊 예측 결과
2143
2144 📁 파일: {filename}
2145 📊 예측 클래스: {class_names_kr[predicted_class]}
2146 📊 신뢰도: {confidence:.1%}
2147
2148 📊 상세 확률:
2149 • 발소리: {probs[0]:.1%}
2150 • 말소리: {probs[1]:.1%}
2151 • 가구끄는소리: {probs[2]:.1%}
2152
2153 📊 오디오 특성:
2154 • 길이: {len(audio)/sr:.1f}초
2155 • 샘플링 레이트: {sr:,} Hz
2156 • RMS 값: {rms:.4f}
2157 • 피크 값: {peak:.4f}
2158 • 주요 주파수: {dominant_freq:.0f} Hz
2159 ""
2160
2161 ax7.text(0.05, 0.95, result_text, transform=ax7.transAxes,
2162         verticalalignment='top', fontsize=11,
2163         bbox=dict(boxstyle='round,pad=1', facecolor='lightblue', alpha=0.8))
2164
2165 # 추가 특징 분석 (선택적)
2166 if show_features:
2167     # 8. 시간별 에너지 변화
2168     ax8 = fig.add_subplot(gs[3, 0])
2169     hop_length = 512
2170     frame_length = 2048
2171     energy = librosa.feature.rms(y=audio, hop_length=hop_length, frame_length=frame_length)[0]
2172     frames = range(len(energy))
2173     times = librosa.frames_to_time(frames, sr=sr, hop_length=hop_length)
2174
2175     ax8.plot(times, energy, color='purple', linewidth=2)
2176     ax8.set_title('📊 시간별 에너지 변화', fontsize=12, fontweight='bold')
2177     ax8.set_xlabel('시간 (초)')
2178     ax8.set_ylabel('RMS 에너지')
2179     ax8.grid(True, alpha=0.3)
2180
2181 # 9. 영교차율 (Zero Crossing Rate)
2182 ax9 = fig.add_subplot(gs[3, 1])
2183 zcr = librosa.feature.zero_crossing_rate(audio, hop_length=hop_length)[0]
2184 ax9.plot(times, zcr, color='orange', linewidth=2)
2185 ax9.set_title('📊 영교차율 (ZCR)', fontsize=12, fontweight='bold')
2186 ax9.set_xlabel('시간 (초)')
2187 ax9.set_ylabel('ZCR')
2188 ax9.grid(True, alpha=0.3)
2189
2190 # 10. 스펙트럼 중심 (Spectral Centroid)
2191 ax10 = fig.add_subplot(gs[3, 2])
2192 spectral_centroids = librosa.feature.spectral_centroid(y=audio, sr=sr, hop_length=hop_length)[0]
2193 ax10.plot(times, spectral_centroids, color='brown', linewidth=2)
2194 ax10.set_title('📊 스펙트럼 중심', fontsize=12, fontweight='bold')
2195 ax10.set_xlabel('시간 (초)')
2196 ax10.set_ylabel('주파수 (Hz)')
2197 ax10.grid(True, alpha=0.3)
2198
2199 # 전체 제목
2200 fig.suptitle(f'📄 오디오 분석 리포트: {class_names_kr[predicted_class]} (신뢰도: {confidence:.1%})',
2201             fontsize=16, fontweight='bold', y=0.98)
2202
2203 plt.tight_layout()
2204
2205 # 이미지 저장
2206 if save_image:
2207     safe_filename = filename.replace('.', '_').replace(' ', '_')
2208     image_path = f"audio_analysis/{safe_filename}_{class_names_en[predicted_class]}.png"
2209     plt.savefig(image_path, dpi=300, bbox_inches='tight')
2210     print(f'📁 이미지 저장: {image_path}')
2211
2212 plt.show()
2213
2214 # 오디오 재생
2215 display(Audio(audio, rate=sr))
```

```

2216
2217     return {
2218         'filename': filename,
2219         'predicted_class': class_names_kr[predicted_class],
2220         'predicted_class_en': class_names_en[predicted_class],
2221         'confidence': confidence,
2222         'probabilities': dict(zip(class_names_kr, probs)),
2223         'audio_features': {
2224             'rms': float(rms),
2225             'peak': float(peak),
2226             'dominant_frequency': float(dominant_freq),
2227             'duration': len(audio)/sr
2228         }
2229     }
2230
2231 except Exception as e:
2232     print(f"❌ 분석 오류: {str(e)}")
2233     return None
2234
2235 def batch_visualize_predictions(model_path, audio_files_list, max_files=10, save_images=False):
2236     """여러 오디오 파일을 일괄 시각화"""
2237     print(f"📁 {min(len(audio_files_list), max_files)}개 파일 일괄 분석 중...")
2238
2239     results = []
2240
2241     for i, audio_file in enumerate(audio_files_list[:max_files]):
2242         print(f"\n📄 {i+1}/{min(len(audio_files_list), max_files)} - {os.path.basename(audio_file)}")
2243
2244         result = visualize_audio_prediction(
2245             model_path, audio_file,
2246             save_image=save_images,
2247             show_features=False # 빠른 분석을 위해 기본 특징만
2248         )
2249
2250         if result:
2251             results.append(result)
2252
2253 # 요약 통계
2254 if results:
2255     print(f"\n📊 일괄 분석 요약:")
2256
2257     # 클래스별 분포
2258     class_counts = {}
2259     confidence_by_class = {}
2260
2261     for result in results:
2262         pred_class = result['predicted_class']
2263         confidence = result['confidence']
2264
2265         if pred_class not in class_counts:
2266             class_counts[pred_class] = 0
2267             confidence_by_class[pred_class] = []
2268
2269         class_counts[pred_class] += 1
2270         confidence_by_class[pred_class].append(confidence)
2271
2272     print(f"   - 분석된 파일: {len(results)}개")
2273     for class_name, count in class_counts.items():
2274         avg_confidence = np.mean(confidence_by_class[class_name])
2275         print(f"   - {class_name}: {count}개 (평균 신뢰도: {avg_confidence:.1%})")
2276
2277 # 신뢰도 분포 시각화
2278 plt.figure(figsize=(12, 8))
2279
2280 plt.subplot(2, 2, 1)
2281 confidences = [r['confidence'] for r in results]
2282 plt.hist(confidences, bins=10, alpha=0.7, color='skyblue', edgecolor='black')
2283 plt.title('신뢰도 분포')
2284 plt.xlabel('신뢰도')
2285 plt.ylabel('빈도')
2286 plt.grid(True, alpha=0.3)
2287
2288 plt.subplot(2, 2, 2)
2289 classes = list(class_counts.keys())
2290 counts = list(class_counts.values())
2291 colors = ['#f99999', '#66b3ff', '#99ff99'][:len(classes)]
2292 plt.pie(counts, labels=classes, colors=colors, autopct='%1.1f%%', startangle=90)
2293 plt.title('예측 클래스 분포')
2294
2295 plt.subplot(2, 2, 3)
2296 for class_name in classes:
2297     if class_name in confidence_by_class:
2298         plt.hist(confidence_by_class[class_name], alpha=0.6,
2299                 label=class_name, bins=5)
2300 plt.title('클래스별 신뢰도 분포')
2301 plt.xlabel('신뢰도')
2302 plt.ylabel('빈도')
2303 plt.legend()
2304 plt.grid(True, alpha=0.3)
2305
2306 plt.subplot(2, 2, 4)
2307 filenames = [r['filename'][:15] + '...' if len(r['filename']) > 15 else r['filename']
2308             for r in results]

```

```

2309     confidences = [r['confidence'] for r in results]
2310     colors_list = []
2311
2312     for result in results:
2313         if result['predicted_class'] == '발소리':
2314             colors_list.append('#ff9999')
2315         elif result['predicted_class'] == '말소리':
2316             colors_list.append('#66b3ff')
2317         else:
2318             colors_list.append('#99ff99')
2319
2320     plt.barh(range(len(filenames)), confidences, color=colors_list, alpha=0.8)
2321     plt.yticks(range(len(filenames)), filenames, fontsize=8)
2322     plt.xlabel('신뢰도')
2323     plt.title('파일별 예측 신뢰도')
2324     plt.grid(True, alpha=0.3)
2325
2326     plt.tight_layout()
2327     plt.show()
2328
2329     return results
2330
2331 def analyze_misclassified_samples(model_path_or_model, val_loader_or_data_path, max_samples=5):
2332     """잘못 분류된 샘플들을 찾아서 분석 - 개선 버전"""
2333     print(f"🔍 잘못 분류된 샘플 분석 중...")
2334
2335     # 모델 로드 (파일 경로가 주어진 경우)
2336     if isinstance(model_path_or_model, str):
2337         if not os.path.exists(model_path_or_model):
2338             print(f"❌ 모델 파일을 찾을 수 없습니다: {model_path_or_model}")
2339             print("   먼저 학습을 실행해주세요: run_sample_training()")
2340             return [], []
2341
2342     checkpoint = torch.load(model_path_or_model, map_location='cpu')
2343     model = ThreeClassAudioCNN(num_classes=3)
2344     model.load_state_dict(checkpoint['model_state_dict'])
2345     model.eval()
2346     print(f"✅ 모델 로드 완료: {model_path_or_model}")
2347
2348     else:
2349         model = model_path_or_model
2350         print("✅ 모델 객체 사용")
2351
2352     # 데이터 로더 준비
2353     if isinstance(val_loader_or_data_path, str):
2354         # 데이터 경로가 주어진 경우 데이터 로더 생성
2355         print(f"📁 데이터 로딩 중: {val_loader_or_data_path}")
2356         audio_files, labels = scan_three_class_data(val_loader_or_data_path)
2357
2358         if len(audio_files) == 0:
2359             print(f"❌ 분석할 데이터가 없습니다!")
2360             return [], []
2361
2362         # 검증용 데이터셋 생성 (전체 데이터의 30%만 사용)
2363         val_files = audio_files[:len(audio_files)//3] if len(audio_files) > 30 else audio_files
2364         val_labels = labels[:len(labels)//3] if len(labels) > 30 else labels
2365
2366         val_dataset = ThreeClassAudioDataset(val_files, val_labels, augment=False)
2367         val_loader = DataLoader(val_dataset, batch_size=4, shuffle=False, num_workers=0)
2368         print(f"✅ 데이터 로더 생성 완료: {len(val_files)}개 파일")
2369
2370     else:
2371         val_loader = val_loader_or_data_path
2372         print("✅ 기존 데이터 로더 사용")
2373
2374     device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
2375     model = model.to(device)
2376
2377     misclassified = []
2378     correct_classified = []
2379
2380     class_names = ['발소리', '말소리', '가구끄는소리']
2381
2382     with torch.no_grad():
2383         for batch in tqdm(val_loader, desc="분석"):
2384             inputs = batch['features'].to(device)
2385             labels = batch['label'].squeeze().to(device)
2386             paths = batch['path']
2387
2388             outputs = model(inputs)
2389             probabilities = F.softmax(outputs, dim=1)
2390             predicted = torch.argmax(outputs, dim=1)
2391
2392             for i in range(len(labels)):
2393                 confidence = probabilities[i, predicted[i]].item()
2394
2395                 sample_info = {
2396                     'path': paths[i],
2397                     'true_label': labels[i].item(),
2398                     'predicted_label': predicted[i].item(),
2399                     'confidence': confidence,
2400                     'true_class': class_names[labels[i].item()],
2401                     'predicted_class': class_names[predicted[i].item()],
2402                     'probabilities': probabilities[i].cpu().numpy()
2403                 }

```

<https://colab.research.google.com/drive/1uCcM0P4cND4e4ikowX4DCOtY5TpU60v1?authuser=1#scrollTo=Y3bCOGu6dMJ&uniqifier=1&print...>

<https://colab.research.google.com/drive/1uCcM0P4cND4e4ikowX4DCOtY5TpU60v1?authuser=1#scrollTo=Y3bCOGGu6dMJ&uniqifier=1&print...>


```
2588 # 각 파일 분석
2589 analysis_results = []
2590
2591 for i, (file_path, true_label) in enumerate(zip(sampled_files, sampled_labels)):
2592     print(f"   진행률: {(i+1)/(len(sampled_files))} ({(i+1)/len(sampled_files)*100:.1f}%)", end='\n')
2593
2594     try:
2595         # 오디오 로드
2596         audio, sr = librosa.load(file_path, sr=16000, duration=5.0)
2597
2598         # 기본 특징 추출
2599         rms = np.sqrt(np.mean(audio**2))
2600         peak = np.max(np.abs(audio))
2601         duration = len(audio) / sr
2602
2603         # 고급 특징
2604         spectral_centroid = np.mean(librosa.feature.spectral_centroid(y=audio, sr=sr))
2605         zcr = np.mean(librosa.feature.zero_crossing_rate(audio))
2606         tempo, _ = librosa.beat.beat_track(y=audio, sr=sr)
2607
2608         # 모델 예측 (모델이 있는 경우)
2609         predicted_class = None
2610         confidence = None
2611
2612         if os.path.exists(model_path):
2613             checkpoint = torch.load(model_path, map_location='cpu')
2614             model = ThreeClassAudioCNN(num_classes=3)
2615             model.load_state_dict(checkpoint['model_state_dict'])
2616             model.eval()
2617
2618             # 특징 추출 및 예측
2619             max_length = 16000 * 5
2620             if len(audio) < max_length:
2621                 audio_padded = np.pad(audio, (0, max_length - len(audio)))
2622             else:
2623                 audio_padded = audio[:max_length]
2624
2625             mel_spec = librosa.feature.melspectrogram(
2626                 y=audio_padded, sr=16000, n_mels=64, fmax=8000, hop_length=512, n_fft=2048
2627             )
2628             mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)
2629
2630             mfcc = librosa.feature.mfcc(
2631                 y=audio_padded, sr=16000, n_mfcc=64, hop_length=512, n_fft=2048
2632             )
2633
2634             features = np.vstack([mel_spec_db, mfcc])
2635
2636             if features.shape[1] != 157:
2637                 from scipy.ndimage import zoom
2638                 zoom_factor = 157 / features.shape[1]
2639                 features = zoom(features, (1, zoom_factor))
2640
2641             with torch.no_grad():
2642                 features_tensor = torch.FloatTensor(features).unsqueeze(0)
2643                 outputs = model(features_tensor)
2644                 probabilities = F.softmax(outputs, dim=1)
2645                 predicted_class = torch.argmax(outputs, dim=1).item()
2646                 confidence = probabilities[0, predicted_class].item()
2647
2648             result = {
2649                 'filename': os.path.basename(file_path),
2650                 'filepath': file_path,
2651                 'true_label': true_label,
2652                 'predicted_label': predicted_class,
2653                 'confidence': confidence,
2654                 'duration': duration,
2655                 'rms': rms,
2656                 'peak': peak,
2657                 'spectral_centroid': spectral_centroid,
2658                 'zcr': zcr,
2659                 'tempo': tempo,
2660                 'file_size': os.path.getsize(file_path) / 1024 # KB
2661             }
2662
2663             analysis_results.append(result)
2664
2665         except Exception as e:
2666             print(f"\n🔥 {file_path} 분석 실패: {e}")
2667
2668 print(f"\n✅ 분석 완료! {len(analysis_results)}개 파일 처리됨")
2669
2670 # 결과 요약
2671 df = pd.DataFrame(analysis_results)
2672
2673 print(f"\n📊 데이터 요약:")
2674 print(f"   - 평균 RMS: {df['rms'].mean():.4f}")
2675 print(f"   - 평균 피크: {df['peak'].mean():.4f}")
2676 print(f"   - 평균 길이: {df['duration'].mean():.1f}초")
2677 print(f"   - 평균 파일 크기: {df['file_size'].mean():.1f} KB")
2678
2679 if 'confidence' in df.columns and df['confidence'].notna().any():
2680     print(f"   - 평균 예측 신뢰도: {df['confidence'].mean():.2%}")
2681     print(f"   - 예측 정확도: {accuracy:.2%}")
```

```
2681 print(f"   - 평균 예측 신뢰도: {df['confidence'].mean():.2%}")
2682
2683 # 정확도 계산
2684 class_mapping = {'footstep': 0, 'speech': 1, 'furniture': 2}
2685 df['true_label_idx'] = df['true_label'].map(class_mapping)
2686 correct_predictions = df['true_label_idx'] == df['predicted_label']
2687 accuracy = correct_predictions.mean()
2688 print(f"   - 예측 정확도: {accuracy:.2%}")
2689
2690 # HTML 리포트 생성
2691 html_content = f"""
2692 <DOCTYPE html>
2693 <html>
2694 <head>
2695   <title>오디오 데이터 검사 리포트</title>
2696   <meta charset="UTF-8">
2697   <style>
2698     body {{ font-family: Arial, sans-serif; margin: 20px; }}
2699     .header {{ background-color: #f0f0f0; padding: 20px; border-radius: 5px; }}
2700     .section {{ margin: 20px 0; }}
2701     table {{ border-collapse: collapse; width: 100%; }}
2702     th, td {{ border: 1px solid #ddd; padding: 8px; text-align: left; }}
2703     th {{ background-color: #f2f2f2; }}
2704     .correct {{ background-color: #d4edda; }}
2705     .incorrect {{ background-color: #f8d7da; }}
2706   </style>
2707 </head>
2708 <body>
2709   <div class="header">
2710     <h1>🔊 오디오 데이터 검사 리포트</h1>
2711     <p>생성 시간: {time.strftime('%Y-%m-%d %H:%M:%S')}</p>
2712     <p>분석된 파일 수: {len(analysis_results)}개</p>
2713   </div>
2714
2715   <div class="section">
2716     <h2>📊 전체 통계</h2>
2717     <table>
2718       <tr><th>항목</th><th>값</th></tr>
2719       <tr><td>평균 RMS</td><td>{df['rms'].mean():.4f}</td></tr>
2720       <tr><td>평균 피크</td><td>{df['peak'].mean():.4f}</td></tr>
2721       <tr><td>평균 길이</td><td>{df['duration'].mean():.1f}초</td></tr>
2722       <tr><td>평균 파일 크기</td><td>{df['file_size'].mean():.1f} KB</td></tr>
2723     </table>
2724
2725     if 'confidence' in df.columns and df['confidence'].notna().any():
2726       html_content += f"""
2727         <tr><td>평균 예측 신뢰도</td><td>{df['confidence'].mean():.2%}</td></tr>
2728         <tr><td>예측 정확도</td><td>{accuracy:.2%}</td></tr>
2729       """
2730
2731   </div>
2732
2733   <div class="section">
2734     <h2>📁 파일별 상세 정보</h2>
2735     <table>
2736       <tr>
2737         <th>파일명</th>
2738         <th>실제 클래스</th>
2739       </tr>
2740     </table>
2741
2742     if 'confidence' in df.columns and df['confidence'].notna().any():
2743       html_content += """
2744         <tr><th>예측 클래스</th>
2745         <th>신뢰도</th>
2746       </tr>
2747
2748       """
2749
2750       html_content += """
2751         <tr><th>길이(초)</th>
2752         <th>RMS</th>
2753         <th>피크</th>
2754         <th>크기(KB)</th>
2755       </tr>
2756
2757       """
2758
2759       class_names = ['footstep', 'speech', 'furniture']
2760
2761       for _, row in df.iterrows():
2762         css_class = ""
2763         if pd.notna(row.get('predicted_label')):
2764           if row['true_label_idx'] == row['predicted_label']:
2765             css_class = "correct"
2766           else:
2767             css_class = "incorrect"
2768
2769         html_content += f"""
2770           <tr class="{css_class}">
2771             <td>{row['filename']}</td>
2772             <td>{row['true_label']}</td>
2773             <td>{row['predicted_label']}</td>
2774             <td>{row['duration']}</td>
2775             <td>{row['rms']}</td>
2776             <td>{row['peak']}</td>
2777             <td>{row['file_size']}</td>
2778             <td>{row['confidence']}</td>
2779           </tr>
2780         """
```

```

2774         if contiguence in ut_columns and pd.notna(row.get('contiguence')):
2775             predicted_class_name = class_names[int(row['predicted_label'])] if pd.notna(row['predicted_label']) else 'N/A'
2776             html_content += f"""
2777             <td>{predicted_class_name}</td>
2778             <td>{row['confidence']:.2%}</td>
2779             """
2780             html_content += f"""
2781             <td>{row['duration']:.1f}</td>
2782             <td>{row['rms']:.4f}</td>
2783             <td>{row['peak']:.4f}</td>
2784             <td>{row['file_size']:.1f}</td>
2785             </tr>
2786             """
2787         html_content += """
2788         </table>
2789     </div>
2790 </body>
2791 </html>
2792 """
2793
2794 # HTML 파일 저장
2795 with open(output_file, 'w', encoding='utf-8') as f:
2796     f.write(html_content)
2797
2798 print(f" 📄 리포트 저장: {output_file}")
2799
2800 return analysis_results, df
2801
2802
2803 def export_visualization_images(model_path, data_path, output_dir='visualization_export', max_files=20):
2804     """데이터셋의 시각화 이미지들을 폴더로 내보내기"""
2805
2806     # 출력 디렉토리 생성
2807     os.makedirs(output_dir, exist_ok=True)
2808
2809     # 클래스별 서브폴더 생성
2810     class_dirs = {}
2811     for class_name in ['footstep', 'speech', 'furniture']:
2812         class_dir = os.path.join(output_dir, class_name)
2813         os.makedirs(class_dir, exist_ok=True)
2814         class_dirs[class_name] = class_dir
2815
2816     print(f" 🖼️ 시각화 이미지 내보내기 시작...")
2817     print(f" 📁 출력 폴더: {output_dir}")
2818
2819     # 데이터 스캔
2820     audio_files, labels = scan_three_class_data(data_path)
2821
2822     if len(audio_files) == 0:
2823         print("❌ 내보낼 데이터가 없습니다!")
2824         return
2825
2826     # 클래스별로 균등하게 샘플링
2827     class_files = {'footstep': [], 'speech': [], 'furniture': []}
2828
2829     for file_path, label in zip(audio_files, labels):
2830         class_files[label].append(file_path)
2831
2832     # 각 클래스에서 최대 max_files//3 개씩 선택
2833     files_per_class = max_files // 3
2834     selected_files = []
2835
2836     for class_name, files_list in class_files.items():
2837         if files_list:
2838             sample_size = min(files_per_class, len(files_list))
2839             sampled = np.random.choice(files_list, sample_size, replace=False)
2840             selected_files.extend([(f, class_name) for f in sampled])
2841
2842     print(f" 📁 {len(selected_files)}개 파일 처리 예정")
2843
2844     # 각 파일 처리
2845     export_summary = []
2846
2847     for i, (file_path, true_class) in enumerate(selected_files):
2848         filename = os.path.basename(file_path)
2849         print(f"   처리 중: {i+1}/{len(selected_files)} - {filename}")
2850
2851         try:
2852             # 시각화 생성 (이미지 저장 모드)
2853             result = visualize_audio_prediction(
2854                 model_path, file_path,
2855                 save_image=False, # 수동으로 저장할 것
2856                 show_features=True
2857             )
2858
2859             if result:
2860                 # 이미지 저장 경로 결정
2861                 safe_filename = filename.replace('.', '_').replace(':', '_')
2862                 predicted_class = result['predicted_class_en']
2863                 confidence = result['confidence']
2864
2865                 # 올바른 예측인지 확인

```

```

2866     correct = "✓" if predicted_class == true_class else "X"
2867
2868     image_filename = f'{safe_filename}_{predicted_class}_conf{confidence:.0%}_{correct}.png'
2869     image_path = os.path.join(class_dirs[true_class], image_filename)
2870
2871     # 현재 figure 저장
2872     plt.savefig(image_path, dpi=200, bbox_inches='tight')
2873     plt.close() # 메모리 절약
2874
2875     export_summary.append({
2876         'original_file': filename,
2877         'true_class': true_class,
2878         'predicted_class': predicted_class,
2879         'confidence': confidence,
2880         'correct': predicted_class == true_class,
2881         'image_path': image_path
2882     })
2883
2884     except Exception as e:
2885         print(f"   ❌ 오류: {e}")
2886
2887 # 요약 리포트 생성
2888 summary_file = os.path.join(output_dir, 'export_summary.txt')
2889
2890 with open(summary_file, 'w', encoding='utf-8') as f:
2891     f.write(" 🖼️ 시각화 이미지 내보내기 요약\n")
2892     f.write(f" = " * 50 + "\n\n")
2893     f.write(f"총 처리된 파일: {len(export_summary)}개\n")
2894
2895     # 정확도 통계
2896     correct_count = sum(1 for item in export_summary if item['correct'])
2897     accuracy = correct_count / len(export_summary) if export_summary else 0
2898     f.write(f"정확한 예측: {correct_count}개 ({accuracy:.1%})\n")
2899     f.write(f"잘못된 예측: {len(export_summary) - correct_count}개\n")
2900
2901     # 클래스별 통계
2902     f.write("\n클래스별 통계:\n")
2903     for class_name in ['footstep', 'speech', 'furniture']:
2904         class_items = [item for item in export_summary if item['true_class'] == class_name]
2905         class_correct = sum(1 for item in class_items if item['correct'])
2906         class_accuracy = class_correct / len(class_items) if class_items else 0
2907         f.write(f"   {class_name}: {len(class_items)}개 (정확도: {class_accuracy:.1%})\n")
2908
2909     f.write("\n파일별 상세 정보:\n")
2910     f.write(f" = " * 50 + "\n\n")
2911
2912     for item in export_summary:
2913         status = "✓" if item['correct'] else "X"
2914         f.write(f"{status} {item['original_file']}\n")
2915         f.write(f"   실제: {item['true_class']} → 예측: {item['predicted_class']} {신뢰도: {item['confidence']:.1%}}\n")
2916         f.write(f"   이미지: {os.path.basename(item['image_path'])}\n\n")
2917
2918     print(f" 📄 내보내기 완료!")
2919     print(f"   - 처리된 파일: {len(export_summary)}개")
2920     print(f"   - 정확도: {accuracy:.1%}")
2921     print(f"   - 이미지 폴더: {output_dir}")
2922     print(f"   - 요약 파일: {summary_file}")
2923
2924     return export_summary
2925
2926 def save_model_info(model_path):
2927     """모델 정보 저장"""
2928     if os.path.exists(model_path):
2929         checkpoint = torch.load(model_path, map_location='cpu')
2930
2931         info = {
2932             'model_type': '3클래스 오디오 분류 CNN',
2933             'classes': ['발소리', '말소리', '가구끄는소리'],
2934             'epoch': checkpoint.get('epoch', 'Unknown'),
2935             'val_accuracy': checkpoint.get('val_acc', 'Unknown'),
2936             'val_loss': checkpoint.get('val_loss', 'Unknown'),
2937             'input_shape': '(128, 157)',
2938             'sample_rate': '16kHz',
2939             'max_duration': '5초'
2940         }
2941
2942         print(" 📄 모델 정보:")
2943         for key, value in info.items():
2944             print(f"   {key}: {value}")
2945
2946         return info
2947     else:
2948         print("❌ 모델 파일을 찾을 수 없습니다.")
2949         return None
2950
2951 def export_model_for_production(model_path, output_path='model_production.pth'):
2952     """프로덕션용 모델 내보내기"""
2953     if not os.path.exists(model_path):
2954         print("❌ 모델 파일이 없습니다!")
2955         return
2956
2957     print(" 📄 프로덕션용 모델 준비 중...")
2958     ---

```

```
2959 # 모델 로드
2960 checkpoint = torch.load(model_path, map_location='cpu')
2961 model = ThreeClassAudioCNN(num_classes=3)
2962 model.load_state_dict(checkpoint['model_state_dict'])
2963 model.eval()
2964
2965 # TorchScript로 변환
2966 dummy_input = torch.randn(1, 1, 128, 157)
2967 traced_model = torch.jit.trace(model, dummy_input)
2968
2969 # 저장
2970 torch.jit.save(traced_model, output_path)
2971
2972 print(f"✅ 프로덕션용 모델 저장 완료: {output_path}")
2973 print("    이 모델은 별도 라이브러리 없이 PyTorch에서 바로 로드 가능합니다.")
2974
2975 # 사용법 출력
2976 print("\n👉 프로덕션 환경에서 사용법:")
2977 print(f"    model = torch.jit.load('{output_path}')"")
2978 print("    output = model(input_tensor)")
2979
2980 def quick_audio_preview(audio_path):
2981     """오디오 파일 빠른 미리보기"""
2982     try:
2983         audio, sr = librosa.load(audio_path, sr=16000, duration=10)
2984
2985         print(f"📄 파일: {os.path.basename(audio_path)}")
2986         print(f"    길이: {len(audio)/sr:.2f}초")
2987         print(f"    샘플링 레이트: {sr}Hz")
2988         print(f"    최대 진폭: {np.max(np.abs(audio)):.3f}")
2989         print(f"    RMS: {np.sqrt(np.mean(audio**2)):.3f}")
2990
2991         # 간단한 시각화
2992         plt.figure(figsize=(12, 4))
2993         time = np.linspace(0, len(audio)/sr, len(audio))
2994         plt.plot(time, audio)
2995         plt.title(f'오디오 파형: {os.path.basename(audio_path)}')
2996         plt.xlabel('시간 (초)')
2997         plt.ylabel('진폭')
2998         plt.grid(True, alpha=0.3)
2999         plt.show()
3000
3001         # 오디오 재생
3002         display(Audio(audio, rate=sr))
3003
3004     except Exception as e:
3005         print(f"❌ 오디오 로딩 오류: {e}")
3006
3007 # 최종 메시지
3008 print("\n🎉 모든 기능이 준비되었습니다!")
3009 print("    한글 그래프 지원이 추가되었습니다! 🇰🇷")
3010 print("    📊 강력한 데이터 디버깅 기능이 추가되었습니다! 🐞")
3011 print("    📺 NEW! 오디오 데이터 시각화 분석 도구가 추가되었습니다! 📊")
3012 print("    🔄 NEW! 'model' 오류 방지 기능이 추가되었습니다! ⚠️")
3013 print("\n⚠️ 'model' is not defined 오류가 발생한다면:")
3014 print("    check_model_and_data_status() # 시스템 상태 먼저 확인")
3015 print("    quick_model_analysis() # 자동으로 모델과 데이터 확인 후 분석")
3016 print("\n🔍 500개 파일 중 150개만 학습되는 문제 해결:")
3017 print("    detailed_data_analysis('/your/data/path')")
3018 print("    run_sample_training('/your/data/path') # 디버깅 모드 자동 실행")
3019 print("\n📺 학습 결과를 이미지로 확인:")
3020 print("    test_visualization_features() # 모든 시각화 기능 테스트")
3021 print("    visualize_audio_prediction('model.pth', 'audio.wav', save_image=True)")
3022 print("    quick_audio_analysis('audio.wav') # 모델 없이도 분석 가능")
3023 print("\n🔥 추천 시작 순서:")
3024 print("    1. easy_start_guide() # 초보자용 쉬운 가이드")
3025 print("    2. check_model_and_data_status() # 상태 확인")
3026 print("    3. run_sample_training() 또는 quick_model_analysis() # 상황에 따라")
3027 print("    4. test_visualization_features() # 시각화 테스트")
3028 print("\n👉 이제 모든 오류 없이 완벽하게 오디오 분석을 할 수 있습니다!")
```

🔗 🎵 발소리-말소리-가구끄는소리 분류 AI 모델

=====

W: Skipping acquire of configured file 'main/source/Sources' as repository 'https://r2u.stat.illinois.edu/ubuntu_jammy InRelease' does not seem to pr

✅ 한글 폰트 설정 완료

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

✅ Google Drive 마운트 완료

✅ Colab 환경 최적화 완료

🎵 발소리-말소리-가구끄는소리 분류 AI 모델 사용 가이드

=====

📄 주요 기능:

- 3클래스 오디오 분류 (발소리, 말소리, 가구끄는소리)
- CNN 기반 딥러닝 모델
- 실시간 예측 및 시각화
- 데이터 증강 및 최적화
- 한글 폰트 지원

🔥 빠른 시작:

📄 한글 폰트 테스트:

```
test_korean_font()
```

한글이 제대로 표시되는지 확인

📄 샘플 데이터로 빠른 테스트:

```
run_sample_training() # 샘플 생성 + 학습
test_model_with_samples() # 테스트

3. 실제 데이터로 학습:
# 데이터 폴더 구조:
# your_data/
#   ├── footsteps/      (발소리 파일들)
#   ├── speech/         (말소리 파일들)
#   └── furniture/       (가구끄는소리 파일들)

model, trainer = run_sample_training('/path/to/your_data')

4. 파일 업로드하여 예측:
analyze_uploaded_file()

5. 실시간 녹음 예측:
record_and_predict('best_three_class_model.pth')

6. 특징 추출 배포:
demo_realtime_features()

7. 모델 성능 상세 분석:
analyze_model_performance(model, val_loader)

📄 모델 성능:
- 입력: 5초 오디오 (16kHz)
- 특징: Mel-spectrogram (64) + MFCC (64) = 128차원
- 구조: CNN (4블록) + Global Average Pooling
- 출력: 3클래스 확률 분포

🇰🇷 한글 폰트 문제 해결:
- 그래프에서 한글이 깨진다면: test_korean_font() 실행
- 여전히 문제가 있다면: 런타임 재시작 후 다시 실행
```

1 model, trainer = run_sample_training('/content/drive/MyDrive/Colab Notebooks/finaldata')

🔗 📄 실제 데이터를 사용합니다: /content/drive/MyDrive/Colab Notebooks/finaldata

🔥 학습을 시작합니다....

🎵 발소리-말소리-가구끄는소리 분류 학습 시작!

=====

W: Skipping acquire of configured file 'main/source/Sources' as repository 'https://r2u.stat.illinois.edu/ubuntu_jammy InRelease' does not seem to pr

✅ 한글 폰트 설정 완료

reCAPTCHA 서비스에 연결할 수 없습니다. 인터넷 연결을 확인한 후 페이지를 새로고침하여 reCAPTCHA 보안문자를 다시 로드하세요.