2025년 온디바이스 AI 3일 집중 실습 교재 - 확장판

초보자부터 중급자까지 따라하는 최신 AI 기술 완벽 가이드

🎯 이 교재의 목표

안녕하세요! 이 교재는 AI에 관심이 있지만 어디서부터 시작해야 할지 막막한 분들을 위해 만들어졌습니다. 3일 동안 여러분은 최신 AI 기술을 직접 만져보고, 실제로 작동하는 애플리케이션을 만들어볼 것입니다.

우리가 함께 만들 것들:

- 📸 사진을 보고 설명해주는 AI
- 🎥 실시간으로 물체를 찾아내는 시스템
- 🗣️ 음성으로 대화하는 AI 비서

준비물:

- 노트북 (NVIDIA GPU가 있으면 좋지만, 없어도 괜찮습니다)
- Python 기초 지식 (변수, 함수, 반복문 정도만 알면 충분합니다)
- 호기심과 열정!

📚 목차

🚀 Day 1: 멀티모달 AI - 보고 이해하는 인공지능

1장: 개발 환경 구축 - 초보자도 걱정 없어요!

1.1 왜 환경 구축이 중요한가요?

AI 개발을 요리에 비유해보겠습니다. 맛있는 요리를 하려면 좋은 주방과 도구가 필요하듯이, AI 개발에도 적절한 개발 환경이 필요합니다. 잘 구축된 환경은 여러분의 개발 속도를 10배 이상 빠르게 만들어줍니다!

1.2 내 컴퓨터는 어떤 타입일까?

먼저 여러분의 컴퓨터가 어떤 종류인지 확인해봅시다:

```
# 내 시스템 정보 확인하기
import platform
import subprocess
import sys

def check_system_info():
    """시스템 정보를 친절하게 알려주는 함수"""

    print("🖥️ 시스템 정보 확인 중...")
    print("=" * 50)

    # 운영체제 확인
    os_info = platform.system()
    os_version = platform.version()
    print(f"📌 운영체제: {os_info} {os_version}")

    # Python 버전 확인
    python_version = sys.version.split()[0]
    print(f"🐍 Python 버전: {python_version}")

    # CPU 정보
    processor = platform.processor()
    print(f"💻 프로세서: {processor}")

    # GPU 확인 (NVIDIA)
    try:
        nvidia_smi = subprocess.check_output(['nvidia-smi', '--query-
gpu=name', '--format=csv,noheader'],
                                             encoding='utf-8').strip()
        print(f"🎮 NVIDIA GPU: {nvidia_smi}")
    except:
        print("🎮 NVIDIA GPU: 감지되지 않음")

    # 메모리 확인
    try:
        import psutil
        memory = psutil.virtual_memory()
        print(f"💾 메모리: {memory.total / (1024**3):.1f}GB 중
```

```
    {memory.available / (1024**3):.1f}GB 사용 가능")
    except:
        print("💾 메모리: psutil을 설치하면 확인 가능합니다")

    print("=" * 50)

    # 추천사항 제공
    if 'NVIDIA' in str(processor) or 'nvidia_smi' in locals():
        print("✅ GPU가 감지되었습니다! 빠른 AI 학습이 가능합니다.")
    else:
        print("💡 GPU가 없어도 괜찮습니다. CPU로도 충분히 실습 가능합니다!")

    if float(python_version[:3]) < 3.8:
        print("⚠️ Python 3.8 이상으로 업그레이드를 권장합니다.")
    else:
        print("✅ Python 버전이 적절합니다.")

# 시스템 정보 확인 실행
check_system_info()
```

1.3 Conda 설치하기 - 가상 환경의 마법

Conda란? Conda는 Python 패키지들을 관리해주는 도구입니다. 마치 스마트폰의 앱스토어처럼, 필요한 라이브러리를 쉽게 설치하고 관리할 수 있게 해줍니다.

왜 가상 환경을 사용하나요? 여러 프로젝트를 진행하다 보면 각 프로젝트마다 필요한 라이브러리 버전이 다를 수 있습니다. 가상 환경은 각 프로젝트를 위한 독립된 공간을 만들어줍니다.

```
# Miniconda 설치 (가벼운 버전의 Conda)
# Windows 사용자는 다운로드 후 설치:
# https://docs.conda.io/en/latest/miniconda.html

# Mac/Linux 사용자:
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-$(uname -
s)-$(uname -m).sh
bash Miniconda3-latest-*.sh

# 설치 확인
conda --version
```

1.4 우리의 첫 번째 AI 환경 만들기

이제 본격적으로 AI 개발을 위한 환경을 만들어봅시다:

```
# 1. 새로운 가상 환경 생성
# 'ondevice-ai'는 우리가 만들 환경의 이름입니다
conda create -n ondevice-ai python=3.11 -y

# 2. 환경 활성화
```

```
# 이 명령어를 실행하면 프롬프트가 (ondevice-ai)로 바뀝니다
conda activate ondevice-ai

# 3. 환경이 잘 활성화되었는지 확인
python --version  # Python 3.11.x가 나와야 합니다
```

1.5 필수 라이브러리 설치 - AI의 도구상자

이제 AI 개발에 필요한 도구들을 설치해봅시다. 각 도구가 무엇을 하는지 설명드리겠습니다:

```
# requirements.txt 파일 생성
requirements_content = """
# 핵심 AI 프레임워크
torch==2.5.0          # 딥러닝의 심장! 모든 AI 모델의 기반
torchvision==0.20.0   # 이미지 처리를 위한 도구 모음
transformers==4.46.0  # 최신 AI 모델들을 쉽게 사용할 수 있게 해주는 라이브러리

# 모델 최적화
accelerate==1.0.0     # GPU를 효율적으로 사용하게 해줍니다
bitsandbytes==0.44.0  # 모델을 가볍게 만들어주는 마법의 도구

# 이미지 처리
opencv-python==4.9.0.80  # 컴퓨터 비전의 스위스 군용 칼
pillow==10.2.0           # 이미지 파일을 다루는 기본 도구
supervision==0.22.0      # 객체 탐지 결과를 예쁘게 시각화

# 오디오 처리
openai-whisper==20231117  # OpenAI의 음성 인식 모델
gtts==2.5.0               # 텍스트를 음성으로 변환
pygame==2.5.2             # 소리를 재생하는 도구
sounddevice==0.4.6        # 마이크로 녹음하기

# 유틸리티
numpy==1.26.4       # 숫자 계산의 기본
scipy==1.12.0       # 과학 계산 도구
pandas==2.2.0       # 데이터 분석의 필수품
matplotlib==3.8.3   # 그래프 그리기
tqdm==4.66.2        # 진행 상황을 보여주는 프로그레스 바

# 웹 인터페이스
gradio==4.16.0      # 웹 UI를 쉽게 만들어주는 도구
fastapi==0.109.0    # API 서버 구축
uvicorn==0.27.0     # FastAPI 서버 실행
"""

# 파일 저장
with open('requirements.txt', 'w') as f:
    f.write(requirements_content)

print("requirements.txt 파일이 생성되었습니다!")
```

이제 설치해봅시다:

```
# NVIDIA GPU가 있는 경우
pip install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cu121
pip install -r requirements.txt

# Mac (Apple Silicon) 사용자
pip install torch torchvision torchaudio
pip install -r requirements.txt

# GPU가 없는 경우 (CPU만 사용)
pip install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cpu
pip install -r requirements.txt
```

1.6 설치 확인 - 모든 것이 잘 작동하는지 테스트

```
# 설치 확인 스크립트
def test_installation():
    """설치가 제대로 되었는지 확인하는 친절한 함수"""

    print("🔍 설치 확인을 시작합니다...\n")

    # 1. PyTorch 확인
    try:
        import torch
        print(f"✅ PyTorch {torch.__version__} 설치 완료!")

        # GPU 사용 가능 여부
        if torch.cuda.is_available():
            gpu_name = torch.cuda.get_device_name(0)
            print(f"   🎮 GPU 사용 가능: {gpu_name}")
            print(f"   💾 GPU 메모리:
{torch.cuda.get_device_properties(0).total_memory / 1024**3:.1f}GB")
        elif torch.backends.mps.is_available():
            print("   🍎 Apple Silicon GPU 사용 가능!")
        else:
            print("   💻 CPU 모드로 실행됩니다 (GPU보다 느리지만 실습에는 문제없습니
다)")
    except ImportError:
        print("❌ PyTorch 설치 실패")

    # 2. Transformers 확인
    try:
        import transformers
        print(f"\n✅ Transformers {transformers.__version__} 설치 완료!")
    except ImportError:
        print("\n❌ Transformers 설치 실패")
```

```
    # 3. OpenCV 확인
    try:
        import cv2
        print(f"\n✅ OpenCV {cv2.__version__} 설치 완료!")
    except ImportError:
        print("\n❌ OpenCV 설치 실패")

    # 4. 간단한 AI 모델 테스트
    print("\n🧪 간단한 AI 테스트 실행 중...")
    try:
        import torch
        import torch.nn as nn

        # 아주 작은 신경망 만들기
        class TinyNet(nn.Module):
            def __init__(self):
                super().__init__()
                self.layer = nn.Linear(10, 1)

            def forward(self, x):
                return self.layer(x)

        # 모델 생성 및 테스트
        model = TinyNet()
        test_input = torch.randn(1, 10)
        output = model(test_input)

        print("✅ AI 모델 테스트 성공!")
        print(f"   입력 크기: {test_input.shape}")
        print(f"   출력 값: {output.item():.4f}")

    except Exception as e:
        print(f"❌ AI 모델 테스트 실패: {e}")

    print("\n🎉 환경 설정이 완료되었습니다! 이제 AI 개발을 시작할 준비가 되었습니다.")

# 테스트 실행
test_installation()
```

2장: 멀티모달 AI란 무엇인가? - 인간처럼 보고 듣고 이해하는 AI

2.1 멀티모달 AI의 탄생 배경

여러분, 우리 인간은 어떻게 세상을 이해할까요? 우리는 눈으로 보고, 귀로 듣고, 손으로 만지며 세상을 종합적으로 이해합니다. 기존의 AI는 텍스트만 이해하거나, 이미지만 인식하는 등 한 가지 감각만 가지고 있었습니다. 하지만 이제는 다릅니다!

멀티모달 AI의 역사:

- 2021년: OpenAI의 CLIP이 이미지와 텍스트를 함께 이해하기 시작
- 2022년: Flamingo, BLIP 등이 등장하며 시각적 대화가 가능해짐
- 2023년: GPT-4V가 출시되며 멀티모달 AI가 대중화

- 2024년: 더 작고 빠른 모델들이 등장 (LLaVA, CogVLM)
- 2025년 현재: 스마트폰에서도 실행 가능한 초경량 모델 시대!

2.2 2025년 최신 멀티모달 모델 완전 정복

🦙 Llama 4 시리즈 (Meta)

배경 스토리: Meta(구 Facebook)는 오픈소스 AI의 선구자입니다. Mark Zuckerberg는 "AI는 모두에게 열려있어야 한다"는 철학으로 Llama 시리즈를 무료로 공개했습니다.

모델 라인업:

1. **Llama 4 Scout (17B)**
   - 용도: 일반 노트북에서도 실행 가능한 경량 모델
   - 특징: 16K 토큰 컨텍스트, 50개 언어 지원
   - 장점: 빠른 속도, 적은 메모리 사용
   - 단점: 복잡한 추론에는 한계
2. **Llama 4 Maverick (45B)**
   - 용도: 전문가 수준의 이미지 분석
   - 특징: 의료 영상, 위성 사진 분석 가능
   - 장점: 높은 정확도, 전문 분야 특화
   - 단점: 고사양 GPU 필요
3. **Llama 4 Titan (175B)**
   - 용도: 연구소나 기업용 최고 성능 모델
   - 특징: GPT-4V를 능가하는 성능
   - 장점: 최고의 성능
   - 단점: 일반 사용자는 사용 불가

💎 Gemma 3 시리즈 (Google)

배경 스토리: Google DeepMind는 "작지만 강력한" 모델을 목표로 Gemma를 개발했습니다. 이름은 라틴어로 "보석"을 의미합니다.

모델 특징:

```
# Gemma 모델들의 특징을 시각화
import matplotlib.pyplot as plt
import numpy as np

models = ['Gemma 3\nFeather\n(4B)', 'Gemma 3\nSwift\n(12B)', 'Gemma
3\nHawk\n(27B)']
memory_usage = [4, 12, 27]  # GB
performance = [70, 85, 95]  # 상대적 성능
mobile_friendly = [100, 60, 20]  # 모바일 적합도

x = np.arange(len(models))
```

```
width = 0.25

fig, ax = plt.subplots(figsize=(10, 6))
bars1 = ax.bar(x - width, memory_usage, width, label='메모리 사용량 (GB)',
color='skyblue')
bars2 = ax.bar(x, performance, width, label='성능 점수', color='lightgreen')
bars3 = ax.bar(x + width, mobile_friendly, width, label='모바일 적합도',
color='coral')

ax.set_xlabel('모델', fontsize=12)
ax.set_ylabel('점수', fontsize=12)
ax.set_title('Gemma 3 시리즈 비교', fontsize=14, fontweight='bold')
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()

# 막대 위에 값 표시
for bars in [bars1, bars2, bars3]:
    for bar in bars:
        height = bar.get_height()
        ax.annotate(f'{height}',
                    xy=(bar.get_x() + bar.get_width() / 2, height),
                    xytext=(0, 3),
                    textcoords="offset points",
                    ha='center', va='bottom')

plt.tight_layout()
plt.show()
```

🌟 MiniCPM-V 2.6 (OpenBMB)

배경 스토리: 중국 칭화대학교의 OpenBMB 팀이 개발한 이 모델은 "작은 거인"이라 불립니다. 8B 파라미터로 GPT-4V의 성능에 근접했습니다.

핵심 특징:

- 초고해상도 이미지 지원 (1344×1344)
- 180개 프레임의 비디오 이해
- 모바일 기기에서 실행 가능
- 한국어 성능 우수

🎯 InternVL 2.5 (Shanghai AI Lab)

배경 스토리: 상하이 AI 연구소가 "동서양의 지혜를 결합"한다는 목표로 개발했습니다.

독특한 기능:

- 다중 이미지 비교 분석
- 차트와 그래프 이해
- OCR 기능 내장
- 수식 인식 가능

🧠 Phi-4 Vision (Microsoft)

배경 스토리: Microsoft는 "작지만 똑똑한" AI를 만드는 데 집중했습니다. Phi는 그리스 문자 φ(파이)에서 따왔습니다.

특별한 점:

- 14B 파라미터로 초경량
- 수학, 코딩에 특화
- 추론 능력 탁월
- Azure 클라우드 최적화

2.3 멀티모달 AI의 작동 원리 - 쉽게 이해하기

멀티모달 AI가 어떻게 작동하는지 단계별로 알아봅시다:

```
# 멀티모달 AI의 작동 과정을 시각화
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from matplotlib.patches import FancyBboxPatch, ConnectionPatch
import numpy as np

fig, ax = plt.subplots(figsize=(12, 8))

# 배경 색상
ax.set_facecolor('#f0f0f0')

# 1. 입력 단계
input_img = FancyBboxPatch((0.5, 6), 2, 1.5,
                           boxstyle="round,pad=0.1",
                           facecolor='lightblue',
                           edgecolor='darkblue',
                           linewidth=2)
ax.add_patch(input_img)
ax.text(1.5, 6.75, '이미지\n입력', ha='center', va='center', fontsize=12,
fontweight='bold')

input_text = FancyBboxPatch((0.5, 4), 2, 1.5,
                            boxstyle="round,pad=0.1",
                            facecolor='lightgreen',
                            edgecolor='darkgreen',
                            linewidth=2)
ax.add_patch(input_text)
ax.text(1.5, 4.75, '텍스트\n질문', ha='center', va='center', fontsize=12,
fontweight='bold')

# 2. 인코딩 단계
img_encoder = FancyBboxPatch((3.5, 6), 2.5, 1.5,
                             boxstyle="round,pad=0.1",
                             facecolor='#FFE5B4',
                             edgecolor='#FF8C00',
                             linewidth=2)
ax.add_patch(img_encoder)
```

```
ax.text(4.75, 6.75, '이미지\n인코더', ha='center', va='center', fontsize=12,
fontweight='bold')

text_encoder = FancyBboxPatch((3.5, 4), 2.5, 1.5,
                              boxstyle="round,pad=0.1",
                              facecolor='#FFE5B4',
                              edgecolor='#FF8C00',
                              linewidth=2)
ax.add_patch(text_encoder)
ax.text(4.75, 4.75, '텍스트\n인코더', ha='center', va='center', fontsize=12,
fontweight='bold')

# 3. 융합 단계
fusion = FancyBboxPatch((7, 5), 2.5, 2,
                        boxstyle="round,pad=0.1",
                        facecolor='#E6E6FA',
                        edgecolor='#9370DB',
                        linewidth=3)
ax.add_patch(fusion)
ax.text(8.25, 6, '멀티모달\n융합', ha='center', va='center', fontsize=12,
fontweight='bold')

# 4. 출력 단계
output = FancyBboxPatch((10.5, 5), 2.5, 2,
                        boxstyle="round,pad=0.1",
                        facecolor='#FFB6C1',
                        edgecolor='#DC143C',
                        linewidth=2)
ax.add_patch(output)
ax.text(11.75, 6, '답변\n생성', ha='center', va='center', fontsize=12,
fontweight='bold')

# 화살표 추가
arrows = [
    ((2.5, 6.75), (3.5, 6.75)),  # 이미지 입력 → 인코더
    ((2.5, 4.75), (3.5, 4.75)),  # 텍스트 입력 → 인코더
    ((6, 6.75), (7, 6)),         # 이미지 인코더 → 융합
    ((6, 4.75), (7, 5)),         # 텍스트 인코더 → 융합
    ((9.5, 6), (10.5, 6))        # 융합 → 출력
]

for start, end in arrows:
    arrow = ConnectionPatch(start, end, "data", "data",
                            arrowstyle="->", shrinkA=5, shrinkB=5,
                            mutation_scale=20, fc="black", linewidth=2)
    ax.add_artist(arrow)

# 제목과 설명
ax.text(7, 8, '멀티모달 AI 작동 원리', fontsize=16, fontweight='bold',
ha='center')
ax.text(7, 2, '이미지와 텍스트를 함께 이해하여 종합적인 답변을 생성합니다',
        fontsize=12, ha='center', style='italic')

ax.set_xlim(0, 14)
```

```
ax.set_ylim(1, 9)
ax.axis('off')

plt.tight_layout()
plt.show()
```

2.4 실제로 어디에 사용될까요?

멀티모달 AI의 실제 활용 사례를 살펴봅시다:

1. **의료 분야**
    - X-ray, MRI 영상 분석
    - 의사의 질문에 따른 영상 해석
    - 진단 보조 시스템
2. **교육 분야**
    - 수학 문제 사진을 찍으면 풀이 과정 설명
    - 과학 실험 영상 분석
    - 언어 학습 도우미
3. **일상 생활**
    - 요리 사진으로 레시피 찾기
    - 옷 사진으로 코디 추천
    - 여행 사진으로 장소 정보 제공

3장: 다양한 최신 모델 실습 - 직접 만져보는 AI

3.1 첫 번째 모델: BLIP-2로 시작하기

BLIP-2는 Salesforce가 만든 모델로, 초보자가 시작하기에 완벽합니다. 왜냐하면 설치가 쉽고, 성능도 좋기 때문입니다!

```
# BLIP-2 기본 사용법 - 초보자를 위한 상세 설명 포함
from transformers import Blip2Processor, Blip2ForConditionalGeneration
import torch
from PIL import Image
import requests
import matplotlib.pyplot as plt

class SimpleImageAnalyzer:
    """초보자를 위한 간단한 이미지 분석기"""

    def __init__(self):
        """모델을 준비하는 초기화 함수"""
        print("🚀 이미지 분석 AI를 준비하고 있습니다...")

        # 모델 이름 - Hugging Face에서 가져옵니다
        model_name = "Salesforce/blip2-opt-2.7b"
```

```
        # 프로세서: 이미지와 텍스트를 모델이 이해할 수 있는 형태로 변환
        self.processor = Blip2Processor.from_pretrained(model_name)

        # 모델: 실제 AI의 두뇌 역할
        # GPU가 있으면 빠르게, 없으면 CPU로 실행
        if torch.cuda.is_available():
            print("🎮 GPU를 사용합니다 - 빠른 처리가 가능합니다!")
            self.model = Blip2ForConditionalGeneration.from_pretrained(
                model_name,
                torch_dtype=torch.float16,  # 메모리 절약을 위해 16비트 사용
                device_map="auto"  # 자동으로 최적의 장치 선택
            )
        else:
            print("💻 CPU를 사용합니다 - 조금 느리지만 충분합니다!")
            self.model = Blip2ForConditionalGeneration.from_pretrained(
                model_name,
                torch_dtype=torch.float32  # CPU는 32비트 사용
            )

        print("✅ AI가 준비되었습니다!\n")

    def analyze_image(self, image_path, question="이 이미지를 설명해주세요"):
        """이미지를 분석하고 질문에 답하는 함수"""

        # 1. 이미지 불러오기
        print(f"📸 이미지를 불러오고 있습니다: {image_path}")

        if image_path.startswith('http'):
            # 인터넷에서 이미지 다운로드
            image = Image.open(requests.get(image_path, stream=True).raw)
        else:
            # 컴퓨터에서 이미지 열기
            image = Image.open(image_path)

        # RGB로 변환 (일부 이미지는 RGBA나 흑백일 수 있음)
        image = image.convert('RGB')

        # 2. 이미지와 질문을 모델이 이해할 수 있는 형태로 변환
        inputs = self.processor(image, question, return_tensors="pt")

        # GPU를 사용한다면 데이터도 GPU로 이동
        if torch.cuda.is_available():
            inputs = {k: v.cuda() for k, v in inputs.items()}

        # 3. AI에게 물어보기
        print("🤔 AI가 이미지를 분석하고 있습니다...")

        with torch.no_grad():  # 학습이 아닌 추론 모드
            generated_ids = self.model.generate(
                **inputs,
                max_new_tokens=100,  # 최대 100개의 단어로 답변
                temperature=0.7,     # 창의성 정도 (0~1, 높을수록 창의적)
                do_sample=True,      # 확률적 샘플링 사용
```

```
                top_p=0.95          # 상위 95%의 확률을 가진 단어만 고려
            )

        # 4. AI의 답변을 사람이 읽을 수 있는 텍스트로 변환
        answer = self.processor.decode(generated_ids[0],
skip_special_tokens=True)

        # 5. 결과 표시
        self.show_result(image, question, answer)

        return answer

    def show_result(self, image, question, answer):
        """결과를 예쁘게 보여주는 함수"""
        plt.figure(figsize=(10, 6))

        # 이미지 표시
        plt.subplot(1, 2, 1)
        plt.imshow(image)
        plt.axis('off')
        plt.title('분석한 이미지', fontsize=14, fontweight='bold')

        # 질문과 답변 표시
        plt.subplot(1, 2, 2)
        plt.text(0.1, 0.7, f"질문: {question}", fontsize=12,
                wrap=True, fontweight='bold')
        plt.text(0.1, 0.3, f"AI 답변: {answer}", fontsize=11,
                wrap=True, style='italic')
        plt.axis('off')
        plt.title('AI의 분석 결과', fontsize=14, fontweight='bold')

        plt.tight_layout()
        plt.show()

# 실제로 사용해보기!
analyzer = SimpleImageAnalyzer()

# 다양한 예제 이미지로 테스트
test_images = [
    {
        'url': 'https://images.unsplash.com/photo-1514888286974-
6c03e2ca1dba',
        'questions': [
            "이 이미지를 설명해주세요",
            "어떤 동물이 보이나요?",
            "동물의 표정은 어떤가요?",
            "배경은 어떤 색인가요?"
        ]
    },
    {
        'url': 'https://images.unsplash.com/photo-1506619216599-
9d16d0903dfd',
        'questions': [
            "이곳은 어디인가요?",
```

```
            "날씨는 어떤가요?",
            "몇 명의 사람이 보이나요?",
            "어떤 활동을 하고 있나요?"
        ]
    }
]

# 각 이미지에 대해 여러 질문하기
for img_data in test_images:
    print(f"\n{'='*50}")
    print(f"새로운 이미지 분석 시작!")
    print(f"{'='*50}\n")

    for question in img_data['questions']:
        answer = analyzer.analyze_image(img_data['url'], question)
        print(f"\n💬 Q: {question}")
        print(f"🤖 A: {answer}")
        print("-" * 30)
```

3.2 MiniCPM-V 2.6 실습 - 긴 비디오도 이해하는 AI

이제 더 고급 모델인 MiniCPM-V를 사용해봅시다. 이 모델은 특히 긴 비디오를 이해하는 데 탁월합니다!

```
# MiniCPM-V 2.6 실습
# 주의: 이 모델은 더 많은 메모리가 필요합니다

from transformers import AutoModel, AutoTokenizer
import torch
import numpy as np
from PIL import Image
from decord import VideoReader, cpu
import matplotlib.pyplot as plt

class VideoUnderstandingAI:
    """비디오를 이해하는 똑똑한 AI"""

    def __init__(self):
        """MiniCPM-V 2.6 모델 초기화"""
        print("🎬 비디오 이해 AI를 준비하고 있습니다...")

        model_name = "openbmb/MiniCPM-V-2_6"

        # 토크나이저: 텍스트를 토큰으로 변환
        self.tokenizer = AutoTokenizer.from_pretrained(
            model_name,
            trust_remote_code=True  # 커스텀 코드 실행 허용
        )

        # 모델 로드
        self.model = AutoModel.from_pretrained(
            model_name,
```

```
            trust_remote_code=True,
            torch_dtype=torch.bfloat16 if torch.cuda.is_available() else
torch.float32
        )

        # 디바이스 설정
        self.device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
        self.model = self.model.to(self.device)

        print(f"✅ 비디오 AI 준비 완료! (디바이스: {self.device})")

    def extract_frames(self, video_path, num_frames=8):
        """비디오에서 균등하게 프레임을 추출하는 함수"""
        print(f"🎞️ 비디오에서 {num_frames}개의 프레임을 추출합니다...")

        # 비디오 읽기
        vr = VideoReader(video_path, ctx=cpu(0))
        total_frames = len(vr)

        # 균등한 간격으로 프레임 선택
        indices = np.linspace(0, total_frames-1, num_frames, dtype=int)

        frames = []
        for idx in indices:
            frame = vr[idx].asnumpy()
            frame_pil = Image.fromarray(frame)
            frames.append(frame_pil)

        return frames, indices, total_frames

    def analyze_video(self, video_path, question="이 비디오에서 무슨 일이 일어나고
있나요?"):
        """비디오를 분석하고 질문에 답하는 함수"""

        # 1. 프레임 추출
        frames, indices, total_frames = self.extract_frames(video_path)

        # 2. 프레임 시각화
        self.visualize_frames(frames, indices, total_frames)

        # 3. 모델에 입력
        print(f"\n🤔 AI가 비디오를 분석하고 있습니다...")
        print(f"질문: {question}")

        # 입력 준비
        msgs = [
            {
                'role': 'user',
                'content': [
                    *[{'type': 'image', 'image': frame} for frame in
frames],
                    {'type': 'text', 'text': question}
                ]
```

```
                }
            ]

            # 추론
            with torch.no_grad():
                answer = self.model.chat(
                    msgs=msgs,
                    tokenizer=self.tokenizer,
                    max_new_tokens=200,
                    temperature=0.7
                )

            print(f"\n🤖 AI의 답변: {answer}")

            return answer, frames

        def visualize_frames(self, frames, indices, total_frames):
            """추출된 프레임을 시각화하는 함수"""
            fig, axes = plt.subplots(2, 4, figsize=(12, 6))
            axes = axes.ravel()

            for i, (frame, idx) in enumerate(zip(frames, indices)):
                axes[i].imshow(frame)
                axes[i].set_title(f'프레임 {idx}/{total_frames}')
                axes[i].axis('off')

            plt.suptitle('비디오에서 추출한 프레임들', fontsize=14,
    fontweight='bold')
            plt.tight_layout()
            plt.show()

    # 사용 예시 (실제 비디오 파일이 필요합니다)
    """
    video_ai = VideoUnderstandingAI()

    # 비디오 분석 예제
    video_questions = [
        "이 비디오에서 무슨 일이 일어나고 있나요?",
        "주요 등장인물은 누구인가요?",
        "어떤 감정이 느껴지나요?",
        "이 비디오의 주제는 무엇인가요?"
    ]

    # 여러 질문으로 분석
    for question in video_questions:
        answer, frames = video_ai.analyze_video("sample_video.mp4", question)
    """
```

3.3 InternVL 2.5 실습 - 차트와 문서를 이해하는 AI

InternVL은 특히 차트, 그래프, 문서를 잘 이해합니다. 학생들에게 아주 유용한 모델이죠!

```
# InternVL 2.5 - 문서와 차트 분석 전문가
import torch
from transformers import AutoModel, AutoTokenizer
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

class DocumentAnalyzer:
    """문서, 차트, 그래프를 분석하는 AI"""

    def __init__(self):
        print("📊 문서 분석 AI를 준비하고 있습니다...")

        # InternVL 모델 로드
        model_name = "OpenGVLab/InternVL2-8B"

        self.tokenizer = AutoTokenizer.from_pretrained(
            model_name,
            trust_remote_code=True
        )

        self.model = AutoModel.from_pretrained(
            model_name,
            trust_remote_code=True,
            torch_dtype=torch.bfloat16 if torch.cuda.is_available() else
torch.float32
        )

        self.device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
        self.model = self.model.to(self.device).eval()

        print("✅ 문서 분석 AI 준비 완료!")

    def create_sample_chart(self):
        """테스트용 차트를 생성하는 함수"""
        # 샘플 데이터 생성
        months = ['1월', '2월', '3월', '4월', '5월', '6월']
        product_a = [45, 52, 48, 58, 63, 67]
        product_b = [38, 41, 43, 46, 51, 55]
        product_c = [30, 35, 33, 38, 42, 48]

        # 차트 생성
        plt.figure(figsize=(10, 6))

        x = np.arange(len(months))
        width = 0.25

        plt.bar(x - width, product_a, width, label='제품 A',
color='#FF6B6B')
        plt.bar(x, product_b, width, label='제품 B', color='#4ECDC4')
        plt.bar(x + width, product_c, width, label='제품 C',
color='#45B7D1')
```

```
        plt.xlabel('월', fontsize=12)
        plt.ylabel('판매량 (천 개)', fontsize=12)
        plt.title('2025년 상반기 제품별 판매 추이', fontsize=14,
fontweight='bold')
        plt.xticks(x, months)
        plt.legend()
        plt.grid(axis='y', alpha=0.3)

        # 차트를 이미지로 저장
        plt.savefig('sample_chart.png', dpi=150, bbox_inches='tight')
        plt.show()

        return 'sample_chart.png'

    def analyze_chart(self, image_path, questions=None):
        """차트나 문서를 분석하는 함수"""

        if questions is None:
            questions = [
                "이 차트가 보여주는 주요 정보는 무엇인가요?",
                "가장 높은 성과를 보인 제품은 무엇인가요?",
                "전체적인 트렌드는 어떤가요?",
                "이 데이터에서 주목할 만한 점은 무엇인가요?"
            ]

        # 이미지 로드
        image = Image.open(image_path)

        print(f"\n📈 차트 분석을 시작합니다...")

        results = []
        for question in questions:
            print(f"\n❓ 질문: {question}")

            # 모델에 입력
            response = self.model.chat(
                self.tokenizer,
                pixel_values=image,
                question=question,
                max_new_tokens=200,
                do_sample=True,
                temperature=0.7
            )

            print(f"🤖 답변: {response}")
            results.append({'question': question, 'answer': response})

        return results

    def analyze_math_problem(self, image_path):
        """수학 문제를 분석하고 풀이하는 함수"""

        prompts = [
```

```
            "이 수학 문제를 단계별로 풀어주세요.",
            "사용된 공식이나 개념을 설명해주세요.",
            "답을 확인하고 검증해주세요."
        ]

        print("\n🔢 수학 문제 분석을 시작합니다...")

        image = Image.open(image_path)

        for prompt in prompts:
            print(f"\n📝 {prompt}")

            response = self.model.chat(
                self.tokenizer,
                pixel_values=image,
                question=prompt,
                max_new_tokens=300,
                do_sample=False  # 수학은 정확성이 중요하므로 샘플링 비활성화
            )

            print(f"💡 {response}")

# 실습 예시
analyzer = DocumentAnalyzer()

# 1. 샘플 차트 생성 및 분석
chart_path = analyzer.create_sample_chart()
results = analyzer.analyze_chart(chart_path)

# 2. 커스텀 질문으로 분석
custom_questions = [
    "어느 제품의 성장률이 가장 높나요?",
    "3월에 판매량이 감소한 제품이 있나요?",
    "이 데이터를 바탕으로 어떤 비즈니스 결정을 내릴 수 있을까요?"
]
analyzer.analyze_chart(chart_path, custom_questions)
```

3.4 Phi-4 Vision 실습 - 코딩과 수학의 달인

Microsoft의 Phi-4는 특히 코딩과 수학 문제를 잘 해결합니다!

```
# Phi-4 Vision - 코딩과 수학 전문가
from transformers import AutoModelForCausalLM, AutoProcessor
import torch
from PIL import Image, ImageDraw, ImageFont
import matplotlib.pyplot as plt

class CodingMathAssistant:
    """코딩과 수학을 도와주는 AI 조교"""

    def __init__(self):
```

```
        print("👨‍💻 코딩/수학 AI 조교를 준비하고 있습니다...")

        model_id = "microsoft/phi-4-vision-preview"

        # 프로세서와 모델 로드
        self.processor = AutoProcessor.from_pretrained(model_id)
        self.model = AutoModelForCausalLM.from_pretrained(
            model_id,
            torch_dtype=torch.float16 if torch.cuda.is_available() else
torch.float32,
            device_map="auto"
        )

        print("✅ AI 조교 준비 완료!")

    def create_code_snippet_image(self, code, language="python"):
        """코드를 이미지로 변환하는 함수"""

        # 이미지 생성
        img = Image.new('RGB', (800, 600), color='#1e1e1e')
        draw = ImageDraw.Draw(img)

        # 간단한 코드 하이라이팅 (실제로는 pygments 등 사용)
        lines = code.strip().split('\n')
        y_offset = 30

        # 타이틀
        draw.text((20, 10), f"{language.upper()} CODE", fill='#ffffff')

        # 코드 라인
        for i, line in enumerate(lines):
            # 라인 번호
            draw.text((20, y_offset), f"{i+1:3d}", fill='#858585')

            # 코드 내용 (간단한 색상 구분)
            x_offset = 60
            if line.strip().startswith('#'):
                color = '#608b4e'  # 주석
            elif any(keyword in line for keyword in ['def', 'class',
'import', 'for', 'if', 'while']):
                color = '#c586c0'  # 키워드
            else:
                color = '#d4d4d4'  # 일반 코드

            draw.text((x_offset, y_offset), line, fill=color)
            y_offset += 25

        return img

    def analyze_code(self, code_text, question="이 코드를 설명해주세요"):
        """코드를 분석하고 설명하는 함수"""

        # 코드를 이미지로 변환
        code_image = self.create_code_snippet_image(code_text)
```

```
        # 분석 수행
        print(f"\n💻 코드 분석 중...")
        print(f"질문: {question}")

        inputs = self.processor(
            text=question,
            images=code_image,
            return_tensors="pt"
        )

        with torch.no_grad():
            outputs = self.model.generate(
                **inputs,
                max_new_tokens=300,
                temperature=0.2,  # 코드 설명은 정확해야 하므로 낮은 temperature
                do_sample=True
            )

        response = self.processor.decode(outputs[0],
skip_special_tokens=True)

        # 결과 시각화
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

        # 코드 이미지
        ax1.imshow(code_image)
        ax1.set_title('분석할 코드', fontweight='bold')
        ax1.axis('off')

        # AI 설명
        ax2.text(0.05, 0.95, "AI의 코드 분석:", fontweight='bold',
                transform=ax2.transAxes, verticalalignment='top')
        ax2.text(0.05, 0.85, response, transform=ax2.transAxes,
                verticalalignment='top', wrap=True, fontsize=10)
        ax2.axis('off')

        plt.tight_layout()
        plt.show()

        return response

    def solve_math_step_by_step(self, problem_text):
        """수학 문제를 단계별로 해결하는 함수"""

        prompts = [
            f"문제: {problem_text}\n\n이 문제를 단계별로 풀어주세요.",
            "각 단계에서 사용한 공식을 설명해주세요.",
            "답을 검증해주세요."
        ]

        print(f"\n🔢 수학 문제: {problem_text}")
        print("=" * 50)
```

```
        full_solution = []

        for i, prompt in enumerate(prompts):
            print(f"\n단계 {i+1}:")

            inputs = self.processor(
                text=prompt,
                return_tensors="pt"
            )

            with torch.no_grad():
                outputs = self.model.generate(
                    **inputs,
                    max_new_tokens=200,
                    temperature=0.1,
                    do_sample=True
                )

            response = self.processor.decode(outputs[0],
skip_special_tokens=True)
            print(response)
            full_solution.append(response)

        return full_solution

# 실습 예제
assistant = CodingMathAssistant()

# 1. 코드 분석 예제
sample_code = """
def fibonacci(n):
    # 피보나치 수열의 n번째 항을 계산
    if n <= 1:
        return n

    # 메모이제이션을 위한 리스트
    fib = [0] * (n + 1)
    fib[0] = 0
    fib[1] = 1

    # 동적 프로그래밍으로 계산
    for i in range(2, n + 1):
        fib[i] = fib[i-1] + fib[i-2]

    return fib[n]

# 테스트
result = fibonacci(10)
print(f"피보나치 수열의 10번째 항: {result}")
"""

# 다양한 질문으로 코드 분석
code_questions = [
    "이 코드를 설명해주세요",
```

```
    "시간 복잡도는 어떻게 되나요?",
    "이 코드를 개선할 방법이 있나요?",
    "재귀적 방법과 비교하면 어떤 장점이 있나요?"
]

for question in code_questions:
    assistant.analyze_code(sample_code, question)

# 2. 수학 문제 해결 예제
math_problems = [
    "x^2 - 5x + 6 = 0의 해를 구하세요.",
    "반지름이 7cm인 원의 넓이를 구하세요.",
    "1부터 100까지의 자연수의 합을 구하세요."
]

for problem in math_problems:
    solution = assistant.solve_math_step_by_step(problem)
```

4장: 실전 프로젝트 - 나만의 이미지 설명 AI 만들기

이제 배운 내용을 종합하여 실제로 사용할 수 있는 애플리케이션을 만들어봅시다!

4.1 프로젝트 개요: 스마트 사진 일기

우리가 만들 애플리케이션은 다음과 같은 기능을 가집니다:

- 📸 사진을 찍거나 업로드
- 💬 AI가 사진을 분석하여 일기 작성
- 🎨 감정과 분위기 분석
- 📅 날짜별로 저장 및 관리

4.2 전체 시스템 구조

```
# 시스템 아키텍처 시각화
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from matplotlib.patches import FancyBboxPatch, Circle, Arrow
import numpy as np

fig, ax = plt.subplots(figsize=(14, 10))
ax.set_xlim(0, 14)
ax.set_ylim(0, 10)
ax.set_facecolor('#f5f5f5')

# 컴포넌트 정의
components = [
    # (x, y, width, height, label, color)
    (1, 7, 3, 1.5, "카메라/갤러리\n인터페이스", '#FF6B6B'),
    (5, 7, 3, 1.5, "이미지 전처리\n모듈", '#FFA500'),
    (9, 7, 3, 1.5, "멀티모달 AI\n분석 엔진", '#4ECDC4'),
```

```
    (1, 4, 3, 1.5, "감정 분석\n모듈", '#9B59B6'),
    (5, 4, 3, 1.5, "일기 생성\n모듈", '#3498DB'),
    (9, 4, 3, 1.5, "데이터 저장\n시스템", '#2ECC71'),
    (5, 1, 3, 1.5, "사용자\n인터페이스", '#E74C3C')
]

# 컴포넌트 그리기
for x, y, w, h, label, color in components:
    box = FancyBboxPatch((x, y), w, h,
                         boxstyle="round,pad=0.1",
                         facecolor=color,
                         edgecolor='black',
                         linewidth=2,
                         alpha=0.8)
    ax.add_patch(box)
    ax.text(x + w/2, y + h/2, label,
           ha='center', va='center',
           fontsize=11, fontweight='bold',
           color='white')

# 연결선 그리기
connections = [
    ((2.5, 7), (5, 7.75)),      # 카메라 → 전처리
    ((8, 7.75), (9, 7.75)),     # 전처리 → AI
    ((10.5, 7), (10.5, 5.5)),   # AI → 저장
    ((9, 7.75), (4, 5.5)),      # AI → 감정분석
    ((2.5, 4), (5, 4.75)),      # 감정분석 → 일기생성
    ((8, 4.75), (9, 4.75)),     # 일기생성 → 저장
    ((6.5, 4), (6.5, 2.5)),     # 일기생성 → UI
    ((9, 4.75), (8, 2.5)),      # 저장 → UI
]

for start, end in connections:
    ax.annotate('', xy=end, xytext=start,
                arrowprops=dict(arrowstyle='->', lw=2,
                                color='#34495E', alpha=0.7))

# 제목
ax.text(7, 9, '스마트 사진 일기 시스템 구조',
       fontsize=16, fontweight='bold', ha='center')

ax.axis('off')
plt.tight_layout()
plt.show()
```

4.3 핵심 모듈 구현

```
# smart_photo_diary.py - 스마트 사진 일기 애플리케이션

import torch
from transformers import Blip2Processor, Blip2ForConditionalGeneration
```

```
from transformers import pipeline
from PIL import Image
import cv2
import numpy as np
from datetime import datetime
import json
import os
from typing import Dict, List, Optional
import gradio as gr

class SmartPhotoDiary:
    """AI 기반 스마트 사진 일기 시스템"""

    def __init__(self):
        """시스템 초기화"""
        print("📔 스마트 사진 일기 시스템을 초기화하고 있습니다...")

        # 1. 이미지 분석 모델
        self.init_image_analyzer()

        # 2. 감정 분석 모델
        self.init_emotion_analyzer()

        # 3. 데이터 저장 경로
        self.diary_path = "photo_diary"
        os.makedirs(self.diary_path, exist_ok=True)

        print("✅ 시스템 초기화 완료!")

    def init_image_analyzer(self):
        """이미지 분석 모델 초기화"""
        print("  🖼️ 이미지 분석 모델 로딩 중...")

        model_name = "Salesforce/blip2-opt-2.7b"
        self.image_processor = Blip2Processor.from_pretrained(model_name)

        # GPU/CPU 자동 선택
        device = "cuda" if torch.cuda.is_available() else "cpu"
        dtype = torch.float16 if device == "cuda" else torch.float32

        self.image_model = Blip2ForConditionalGeneration.from_pretrained(
            model_name,
            torch_dtype=dtype,
            device_map="auto" if device == "cuda" else None
        )

        if device == "cpu":
            self.image_model = self.image_model.to(device)

    def init_emotion_analyzer(self):
        """감정 분석 모델 초기화"""
        print("  😊 감정 분석 모델 로딩 중...")

        # 다국어 감정 분석 파이프라인
```

```
        self.emotion_analyzer = pipeline(
            "text-classification",
            model="j-hartmann/emotion-english-distilroberta-base",
            device=0 if torch.cuda.is_available() else -1
        )

        # 감정 이모지 매핑
        self.emotion_emojis = {
            'joy': '😊',
            'sadness': '😢',
            'anger': '😠',
            'fear': '😨',
            'surprise': '😲',
            'disgust': '🤢',
            'neutral': '😐'
        }

    def analyze_image(self, image: Image.Image) -> Dict:
        """이미지를 종합적으로 분석"""

        results = {}

        # 1. 장면 설명
        scene_prompt = "Describe this image in detail:"
        scene_inputs = self.image_processor(image, scene_prompt,
return_tensors="pt")

        if torch.cuda.is_available():
            scene_inputs = {k: v.cuda() for k, v in scene_inputs.items()}

        with torch.no_grad():
            scene_ids = self.image_model.generate(**scene_inputs,
max_new_tokens=100)

        scene_description = self.image_processor.decode(scene_ids[0],
skip_special_tokens=True)
        results['scene'] = scene_description

        # 2. 주요 객체 찾기
        objects_prompt = "What are the main objects in this image?"
        objects_inputs = self.image_processor(image, objects_prompt,
return_tensors="pt")

        if torch.cuda.is_available():
            objects_inputs = {k: v.cuda() for k, v in
objects_inputs.items()}

        with torch.no_grad():
            objects_ids = self.image_model.generate(**objects_inputs,
max_new_tokens=50)

        objects_description = self.image_processor.decode(objects_ids[0],
skip_special_tokens=True)
        results['objects'] = objects_description
```

```
        # 3. 분위기 파악
        mood_prompt = "What is the mood or atmosphere of this image?"
        mood_inputs = self.image_processor(image, mood_prompt,
return_tensors="pt")

        if torch.cuda.is_available():
            mood_inputs = {k: v.cuda() for k, v in mood_inputs.items()}

        with torch.no_grad():
            mood_ids = self.image_model.generate(**mood_inputs,
max_new_tokens=50)

        mood_description = self.image_processor.decode(mood_ids[0],
skip_special_tokens=True)
        results['mood'] = mood_description

        # 4. 색상 분석
        results['colors'] = self.analyze_colors(image)

        return results

    def analyze_colors(self, image: Image.Image) -> Dict:
        """이미지의 주요 색상 분석"""

        # PIL 이미지를 numpy 배열로 변환
        img_array = np.array(image)

        # 색상 히스토그램 계산
        if len(img_array.shape) == 3:
            # RGB 이미지
            avg_color = img_array.mean(axis=(0, 1))

            # 주요 색상 판단
            r, g, b = avg_color

            if r > g and r > b:
                dominant = "빨간색 계열"
            elif g > r and g > b:
                dominant = "녹색 계열"
            elif b > r and b > g:
                dominant = "파란색 계열"
            else:
                dominant = "중성 색상"

            # 밝기 판단
            brightness = avg_color.mean()
            if brightness > 200:
                brightness_desc = "밝은"
            elif brightness > 100:
                brightness_desc = "보통 밝기의"
            else:
                brightness_desc = "어두운"
```

```
            return {
                'dominant': dominant,
                'brightness': brightness_desc,
                'avg_rgb': avg_color.tolist()
            }
        else:
            return {
                'dominant': "흑백",
                'brightness': "흑백 이미지",
                'avg_rgb': [0, 0, 0]
            }

    def generate_diary_entry(self, image_analysis: Dict, user_context: str
= "") -> Dict:
        """분석 결과를 바탕으로 일기 항목 생성"""

        # 현재 시간
        now = datetime.now()

        # 일기 내용 생성
        diary_text = f"""
오늘 {now.strftime('%Y년 %m월 %d일 %H시 %M분')}의 순간을 기록합니다.

📸 장면 묘사:
{image_analysis['scene']}

🎨 발견한 것들:
{image_analysis['objects']}

💫 분위기:
{image_analysis['mood']}

🎨 색감:
오늘의 사진은 {image_analysis['colors']['brightness']}
{image_analysis['colors']['dominant']}이
인상적이었습니다.
"""

        # 사용자 컨텍스트 추가
        if user_context:
            diary_text += f"\n✍️ 나의 메모:\n{user_context}\n"

        # 감정 분석
        emotion_results = self.emotion_analyzer(diary_text)
        primary_emotion = emotion_results[0]

        diary_text +=
f"\n{self.emotion_emojis.get(primary_emotion['label'], '😊 ')} " \
                     f"오늘의 감정: {primary_emotion['label']} " \
                     f"({primary_emotion['score']*100:.1f}%)"

        return {
            'date': now.isoformat(),
            'text': diary_text,
```

```
            'analysis': image_analysis,
            'emotion': primary_emotion,
            'user_context': user_context
        }

    def save_diary_entry(self, image: Image.Image, diary_entry: Dict) ->
str:
        """일기 항목을 저장"""

        # 날짜별 폴더 생성
        date_str = datetime.now().strftime('%Y%m%d')
        day_folder = os.path.join(self.diary_path, date_str)
        os.makedirs(day_folder, exist_ok=True)

        # 타임스탬프로 파일명 생성
        timestamp = datetime.now().strftime('%H%M%S')

        # 이미지 저장
        image_path = os.path.join(day_folder, f"{timestamp}_photo.jpg")
        image.save(image_path)

        # 일기 데이터 저장
        diary_path = os.path.join(day_folder, f"{timestamp}_diary.json")
        with open(diary_path, 'w', encoding='utf-8') as f:
            json.dump(diary_entry, f, ensure_ascii=False, indent=2)

        # 텍스트 버전도 저장
        text_path = os.path.join(day_folder, f"{timestamp}_diary.txt")
        with open(text_path, 'w', encoding='utf-8') as f:
            f.write(diary_entry['text'])

        return day_folder

    def create_gradio_interface(self):
        """Gradio 웹 인터페이스 생성"""

        def process_image(image, user_memo):
            """이미지 처리 및 일기 생성"""

            if image is None:
                return "이미지를 업로드해주세요!", None

            # PIL Image로 변환
            if isinstance(image, np.ndarray):
                image = Image.fromarray(image)

            # 이미지 분석
            print("🔍 이미지를 분석하고 있습니다...")
            analysis = self.analyze_image(image)

            # 일기 생성
            print("✍️ 일기를 작성하고 있습니다...")
            diary_entry = self.generate_diary_entry(analysis, user_memo)
```

```
            # 저장
            print("💾 일기를 저장하고 있습니다...")
            save_path = self.save_diary_entry(image, diary_entry)

            # 결과 이미지 생성 (일기 내용을 이미지에 오버레이)
            result_image = self.create_diary_image(image, diary_entry)

            return diary_entry['text'], result_image

        # Gradio 인터페이스
        with gr.Blocks(title="스마트 사진 일기", theme=gr.themes.Soft()) as
demo:
            gr.Markdown("""
            # 📔 스마트 사진 일기

            사진을 업로드하면 AI가 자동으로 일기를 작성해드립니다!
            """)

            with gr.Row():
                with gr.Column():
                    image_input = gr.Image(
                        label="사진 업로드",
                        type="numpy"
                    )
                    memo_input = gr.Textbox(
                        label="메모 (선택사항)",
                        placeholder="오늘의 특별한 순간이나 느낌을 적어주세요...",
                        lines=3
                    )
                    submit_btn = gr.Button("일기 작성하기", variant="primary")

                with gr.Column():
                    diary_output = gr.Textbox(
                        label="AI가 작성한 일기",
                        lines=15
                    )
                    result_image = gr.Image(
                        label="일기가 포함된 이미지"
                    )

            # 예시 추가
            gr.Examples(
                examples=[
                    ["오늘 공원에서 산책했어요"],
                    ["맛있는 음식을 먹었습니다"],
                    ["친구들과 즐거운 시간"],
                    ["혼자만의 조용한 시간"]
                ],
                inputs=memo_input
            )

            submit_btn.click(
                fn=process_image,
                inputs=[image_input, memo_input],
```

```
                outputs=[diary_output, result_image]
            )

        return demo

    def create_diary_image(self, original_image: Image.Image, diary_entry:
Dict) -> Image.Image:
        """일기 내용을 이미지에 오버레이"""

        # 원본 이미지 복사
        img = original_image.copy()

        # numpy 배열로 변환
        img_array = np.array(img)

        # 반투명 오버레이 추가
        overlay = img_array.copy()
        cv2.rectangle(overlay, (0, 0), (img_array.shape[1], 150), (0, 0,
0), -1)
        img_array = cv2.addWeighted(img_array, 0.7, overlay, 0.3, 0)

        # 텍스트 추가 (OpenCV는 한글 지원이 제한적이므로 영문으로)
        date_str = datetime.now().strftime('%Y-%m-%d %H:%M')
        emotion = diary_entry['emotion']['label']

        cv2.putText(img_array, f"Photo Diary - {date_str}",
                    (20, 40), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255),
2)
        cv2.putText(img_array, f"Emotion: {emotion}",
                    (20, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255,
255), 2)
        cv2.putText(img_array, f"Mood: {diary_entry['analysis']['mood']
[:50]}...",
                    (20, 120), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255,
255), 1)

        return Image.fromarray(img_array)

# 애플리케이션 실행
def run_smart_photo_diary():
    """스마트 사진 일기 앱 실행"""

    diary = SmartPhotoDiary()
    interface = diary.create_gradio_interface()

    print("\n🚀 스마트 사진 일기가 실행됩니다!")
    print("웹 브라우저에서 자동으로 열립니다...")

    interface.launch(
        server_name="0.0.0.0",
        server_port=7860,
        share=True  # 공유 링크 생성
    )
```

```
# 실행
if __name__ == "__main__":
    run_smart_photo_diary()
```

4.4 고급 기능 추가

이제 더 흥미로운 기능들을 추가해봅시다!

```
# advanced_features.py - 고급 기능 모듈

import calendar
from collections import defaultdict
import plotly.graph_objects as go
import plotly.express as px
from wordcloud import WordCloud
import matplotlib.pyplot as plt

class DiaryAnalytics:
    """일기 분석 및 시각화 기능"""

    def __init__(self, diary_path="photo_diary"):
        self.diary_path = diary_path
        self.entries = self.load_all_entries()

    def load_all_entries(self) -> List[Dict]:
        """모든 일기 항목 로드"""
        entries = []

        for date_folder in os.listdir(self.diary_path):
            folder_path = os.path.join(self.diary_path, date_folder)
            if os.path.isdir(folder_path):
                for file in os.listdir(folder_path):
                    if file.endswith('_diary.json'):
                        with open(os.path.join(folder_path, file), 'r',
encoding='utf-8') as f:
                            entry = json.load(f)
                            entries.append(entry)

        return sorted(entries, key=lambda x: x['date'])

    def emotion_timeline(self):
        """감정 변화 타임라인 생성"""

        dates = []
        emotions = []
        scores = []

        for entry in self.entries:
            date = datetime.fromisoformat(entry['date'])
            dates.append(date)
            emotions.append(entry['emotion']['label'])
```

```
            scores.append(entry['emotion']['score'])

        # Plotly로 인터랙티브 차트 생성
        fig = go.Figure()

        # 감정별 색상
        emotion_colors = {
            'joy': '#FFD93D',
            'sadness': '#6BBAEC',
            'anger': '#FF6B6B',
            'fear': '#A8E6CF',
            'surprise': '#FFB6C1',
            'disgust': '#C3AED6',
            'neutral': '#D3D3D3'
        }

        for emotion in emotion_colors:
            emotion_dates = [d for d, e in zip(dates, emotions) if e ==
emotion]
            emotion_scores = [s for e, s in zip(emotions, scores) if e ==
emotion]

            fig.add_trace(go.Scatter(
                x=emotion_dates,
                y=emotion_scores,
                mode='markers+lines',
                name=emotion,
                marker=dict(
                    size=10,
                    color=emotion_colors[emotion]
                ),
                line=dict(
                    color=emotion_colors[emotion],
                    width=2
                )
            ))

        fig.update_layout(
            title="감정 변화 타임라인",
            xaxis_title="날짜",
            yaxis_title="감정 강도",
            hovermode='x unified',
            template='plotly_white'
        )

        return fig

    def monthly_mood_calendar(self, year=None, month=None):
        """월별 감정 캘린더 생성"""

        if year is None:
            year = datetime.now().year
        if month is None:
            month = datetime.now().month
```

```
        # 해당 월의 일기들 필터링
        month_entries = defaultdict(list)

        for entry in self.entries:
            entry_date = datetime.fromisoformat(entry['date'])
            if entry_date.year == year and entry_date.month == month:
                day = entry_date.day
                month_entries[day].append(entry['emotion']['label'])

        # 캘린더 생성
        fig, ax = plt.subplots(figsize=(12, 8))

        # 월의 첫날과 마지막날
        first_day = datetime(year, month, 1)
        last_day = datetime(year, month, calendar.monthrange(year, month)
[1])

        # 캘린더 그리드
        cal = calendar.monthcalendar(year, month)

        # 감정 이모지
        emotion_emojis = {
            'joy': '😊',
            'sadness': '😢',
            'anger': '😠',
            'fear': '😨',
            'surprise': '😲',
            'disgust': '🤢',
            'neutral': '😐'
        }

        # 캘린더 그리기
        for week_num, week in enumerate(cal):
            for day_num, day in enumerate(week):
                if day == 0:
                    continue

                x = day_num
                y = len(cal) - week_num - 1

                # 날짜 표시
                ax.text(x, y + 0.4, str(day),
                        ha='center', va='center', fontsize=12)

                # 감정 표시
                if day in month_entries:
                    emotions = month_entries[day]
                    # 가장 빈번한 감정
                    most_common = max(set(emotions), key=emotions.count)
                    emoji = emotion_emojis.get(most_common, '😊')

                    ax.text(x, y - 0.2, emoji,
                            ha='center', va='center', fontsize=20)
```

```
        # 스타일링
        ax.set_xlim(-0.5, 6.5)
        ax.set_ylim(-0.5, len(cal) - 0.5)
        ax.set_xticks(range(7))
        ax.set_xticklabels(['월', '화', '수', '목', '금', '토', '일'])
        ax.set_yticks([])

        # 격자 추가
        for i in range(8):
            ax.axvline(i - 0.5, color='gray', alpha=0.3)
        for i in range(len(cal) + 1):
            ax.axhline(i - 0.5, color='gray', alpha=0.3)

        ax.set_title(f'{year}년 {month}월 감정 캘린더',
                     fontsize=16, fontweight='bold', pad=20)

        return fig

    def word_cloud_from_diaries(self):
        """일기 내용으로 워드 클라우드 생성"""

        # 모든 일기 텍스트 수집
        all_text = ' '.join([entry['text'] for entry in self.entries])

        # 한국어 처리를 위한 설정
        # (실제로는 konlpy 등을 사용하여 형태소 분석 필요)

        # 워드 클라우드 생성
        wordcloud = WordCloud(
            width=800,
            height=400,
            background_color='white',
            colormap='viridis',
            font_path='NanumGothic.ttf'  # 한글 폰트 경로
        ).generate(all_text)

        # 시각화
        plt.figure(figsize=(12, 6))
        plt.imshow(wordcloud, interpolation='bilinear')
        plt.axis('off')
        plt.title('내 일기의 주요 단어들', fontsize=16, fontweight='bold')

        return plt.gcf()

    def photo_collage(self, max_photos=9):
        """최근 사진들로 콜라주 생성"""

        photos = []

        # 최근 사진들 수집
        for entry in self.entries[-max_photos:]:
            date = datetime.fromisoformat(entry['date'])
            date_str = date.strftime('%Y%m%d')
```

```
            time_str = date.strftime('%H%M%S')

            photo_path = os.path.join(
                self.diary_path,
                date_str,
                f"{time_str}_photo.jpg"
            )

            if os.path.exists(photo_path):
                img = Image.open(photo_path)
                photos.append(img)

        if not photos:
            return None

        # 콜라주 크기 계산
        n = len(photos)
        rows = int(np.sqrt(n))
        cols = int(np.ceil(n / rows))

        # 각 사진 크기
        photo_size = (300, 300)

        # 콜라주 생성
        collage = Image.new('RGB', (cols * photo_size[0], rows *
photo_size[1]))

        for i, photo in enumerate(photos):
            # 사진 크기 조정
            photo = photo.resize(photo_size, Image.Resampling.LANCZOS)

            # 위치 계산
            row = i // cols
            col = i % cols

            # 콜라주에 추가
            collage.paste(photo, (col * photo_size[0], row *
photo_size[1]))

        return collage

# 분석 대시보드 생성
def create_analytics_dashboard(diary_path="photo_diary"):
    """분석 대시보드 Gradio 인터페이스"""

    analytics = DiaryAnalytics(diary_path)

    with gr.Blocks(title="일기 분석 대시보드", theme=gr.themes.Soft()) as
dashboard:
        gr.Markdown("""
        # 📊 나의 사진 일기 분석

        AI가 분석한 나의 감정 변화와 일기 패턴을 확인해보세요!
        """)
```

```
        with gr.Tab("감정 타임라인"):
            timeline_plot = gr.Plot(
                label="시간에 따른 감정 변화"
            )

            def update_timeline():
                return analytics.emotion_timeline()

            timeline_btn = gr.Button("타임라인 업데이트")
            timeline_btn.click(fn=update_timeline, outputs=timeline_plot)

        with gr.Tab("월별 감정 캘린더"):
            with gr.Row():
                year_input = gr.Number(
                    label="년도",
                    value=datetime.now().year,
                    precision=0
                )
                month_input = gr.Number(
                    label="월",
                    value=datetime.now().month,
                    minimum=1,
                    maximum=12,
                    precision=0
                )

            calendar_plot = gr.Plot(label="감정 캘린더")

            def update_calendar(year, month):
                return analytics.monthly_mood_calendar(int(year),
int(month))

            calendar_btn = gr.Button("캘린더 생성")
            calendar_btn.click(
                fn=update_calendar,
                inputs=[year_input, month_input],
                outputs=calendar_plot
            )

        with gr.Tab("워드 클라우드"):
            wordcloud_plot = gr.Plot(label="일기 주요 단어")

            def generate_wordcloud():
                return analytics.word_cloud_from_diaries()

            wordcloud_btn = gr.Button("워드 클라우드 생성")
            wordcloud_btn.click(fn=generate_wordcloud,
outputs=wordcloud_plot)

        with gr.Tab("사진 콜라주"):
            collage_image = gr.Image(label="최근 사진 모음")

            def create_collage():
```

```
                return analytics.photo_collage()

            collage_btn = gr.Button("콜라주 만들기")
            collage_btn.click(fn=create_collage, outputs=collage_image)

    return dashboard
```

4.5 프로젝트 완성 및 실행

```
# main.py - 전체 애플리케이션 실행

def run_complete_photo_diary_system():
    """완전한 스마트 사진 일기 시스템 실행"""

    print("""
    ╔═══════════════════════════════════════════╗
    ║     📔 스마트 사진 일기 시스템 📔       ║
    ║                                           ║
    ║  AI가 당신의 일상을 특별하게 기록합니다  ║
    ╚═══════════════════════════════════════════╝
    """)

    # 메뉴 선택
    print("\n무엇을 하시겠습니까?")
    print("1. 새로운 일기 작성")
    print("2. 일기 분석 대시보드")
    print("3. 전체 시스템 실행")

    choice = input("\n선택 (1-3): ")

    if choice == "1":
        # 일기 작성 모드
        diary = SmartPhotoDiary()
        interface = diary.create_gradio_interface()
        interface.launch(share=True)

    elif choice == "2":
        # 분석 대시보드
        dashboard = create_analytics_dashboard()
        dashboard.launch(share=True)

    elif choice == "3":
        # 전체 시스템 (탭으로 구성)
        with gr.Blocks(title="스마트 사진 일기 - 전체 시스템") as app:
            with gr.Tab("📝 일기 작성"):
                diary = SmartPhotoDiary()
                diary_interface = diary.create_gradio_interface()

            with gr.Tab("📊 분석 대시보드"):
                analytics_interface = create_analytics_dashboard()
```

```
        app.launch(share=True)

    else:
        print("잘못된 선택입니다.")

if __name__ == "__main__":
    run_complete_photo_diary_system()
```

Day 1 마무리 및 과제

오늘 배운 내용 정리

1. **개발 환경 구축**
    - Python 가상 환경 설정
    - 필수 라이브러리 설치
    - GPU/CPU 환경 확인
2. **멀티모달 AI 이론**
    - 멀티모달 AI의 개념과 발전 과정
    - 최신 모델들의 특징과 장단점
    - 실제 활용 사례
3. **다양한 모델 실습**
    - BLIP-2: 기본적인 이미지 이해
    - MiniCPM-V: 비디오 분석
    - InternVL: 문서와 차트 이해
    - Phi-4: 코드와 수학 문제 해결
4. **실전 프로젝트**
    - 스마트 사진 일기 시스템 구축
    - 감정 분석 기능 추가
    - 데이터 시각화 및 분석

실습 과제

1. **기본 과제**
    - 자신의 사진 10장으로 일기 작성해보기
    - 다양한 질문으로 이미지 분석 테스트
    - 감정 캘린더 확인하기
2. **도전 과제**
    - 새로운 멀티모달 모델 추가하기
    - 음성 메모 기능 추가
    - 일기를 PDF로 내보내기 기능

3. **창의 과제**
    - 가족 앨범 정리 AI 만들기
    - 여행 사진 자동 분류기
    - 요리 레시피 추천 시스템

내일 예고

내일은 더욱 흥미진진한 **컴퓨터 비전**의 세계로 들어갑니다!

- YOLO로 실시간 객체 탐지
- 사람의 행동을 인식하는 AI
- 나만의 스마트 보안 시스템 만들기

오늘 하루 수고하셨습니다! 🎉

🎯 Day 2: 컴퓨터 비전 - 실시간으로 세상을 인식하기

5장: YOLO의 모든 것 - 역사부터 최신 버전까지

5.1 YOLO의 탄생 스토리

2015년, 워싱턴 대학의 Joseph Redmon이라는 박사과정 학생이 있었습니다. 그는 기존의 객체 탐지 방법이 너무 느리다고 생각했습니다. "왜 이미지를 여러 번 봐야 하지? 사람은 한 번만 봐도 모든 걸 알아채는데!"

이런 아이디어에서 YOLO(You Only Look Once)가 탄생했습니다.

5.2 YOLO의 진화 과정

```
# YOLO 진화 타임라인 시각화
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from matplotlib.patches import FancyBboxPatch
import numpy as np

fig, ax = plt.subplots(figsize=(14, 8))

# YOLO 버전 정보
yolo_versions = [
    {'version': 'YOLOv1', 'year': 2015, 'fps': 45, 'mAP': 63.4, 'creator':
'Joseph Redmon',
     'feature': '실시간 객체 탐지의 시작'},
    {'version': 'YOLOv2', 'year': 2016, 'fps': 67, 'mAP': 76.8, 'creator':
'Joseph Redmon',
     'feature': 'Anchor Box 도입'},
    {'version': 'YOLOv3', 'year': 2018, 'fps': 65, 'mAP': 82.0, 'creator':
'Joseph Redmon',
     'feature': '다중 스케일 예측'},
    {'version': 'YOLOv4', 'year': 2020, 'fps': 65, 'mAP': 85.4, 'creator':
'Alexey Bochkovskiy',
```

```
     'feature': 'Bag of Freebies'},
    {'version': 'YOLOv5', 'year': 2020, 'fps': 140, 'mAP': 84.0,
'creator': 'Ultralytics',
     'feature': 'PyTorch 구현'},
    {'version': 'YOLOv6', 'year': 2022, 'fps': 150, 'mAP': 85.5,
'creator': 'Meituan',
     'feature': '산업 최적화'},
    {'version': 'YOLOv7', 'year': 2022, 'fps': 155, 'mAP': 86.7,
'creator': 'WongKinYiu',
     'feature': 'E-ELAN 구조'},
    {'version': 'YOLOv8', 'year': 2023, 'fps': 160, 'mAP': 87.0,
'creator': 'Ultralytics',
     'feature': 'Anchor-free'},
    {'version': 'YOLOv9', 'year': 2024, 'fps': 170, 'mAP': 88.0,
'creator': 'Community',
     'feature': 'PGI & GELAN'},
    {'version': 'YOLOv10', 'year': 2024, 'fps': 175, 'mAP': 88.5,
'creator': 'THU',
     'feature': 'NMS-free'},
    {'version': 'YOLO11', 'year': 2024, 'fps': 180, 'mAP': 89.0,
'creator': 'Ultralytics',
     'feature': 'C3k2 블록'},
    {'version': 'YOLOv12', 'year': 2025, 'fps': 200, 'mAP': 90.5,
'creator': 'OpenAI',
     'feature': 'ViT 하이브리드'}
]

# 년도별 위치 계산
years = [v['year'] for v in yolo_versions]
positions = np.linspace(1, 10, len(yolo_versions))

# 타임라인 그리기
for i, (version, pos) in enumerate(zip(yolo_versions, positions)):
    # 박스 그리기
    if i < 3:
        color = '#FF6B6B'  # Redmon 시대
    elif i < 8:
        color = '#4ECDC4'  # 커뮤니티 시대
    else:
        color = '#45B7D1'  # 최신 시대

    box = FancyBboxPatch((pos-0.4, 2), 0.8, 2,
                         boxstyle="round,pad=0.1",
                         facecolor=color,
                         edgecolor='black',
                         linewidth=2,
                         alpha=0.8)
    ax.add_patch(box)

    # 버전 정보
    ax.text(pos, 3.5, version['version'], ha='center', fontweight='bold',
fontsize=10)
    ax.text(pos, 3.2, f"{version['year']}", ha='center', fontsize=8)
    ax.text(pos, 2.8, f"FPS: {version['fps']}", ha='center', fontsize=8)
```

```
    ax.text(pos, 2.5, f"mAP: {version['mAP']}%", ha='center', fontsize=8)

    # 특징
    ax.text(pos, 1.5, version['feature'], ha='center', fontsize=8,
            wrap=True, style='italic')

    # 제작자
    ax.text(pos, 0.5, version['creator'], ha='center', fontsize=7,
            color='gray')

# 화살표 그리기
for i in range(len(positions)-1):
    ax.arrow(positions[i]+0.4, 3, positions[i+1]-positions[i]-0.8, 0,
            head_width=0.2, head_length=0.1, fc='gray', ec='gray',
alpha=0.5)

# 스타일링
ax.set_xlim(0, 11)
ax.set_ylim(0, 5)
ax.set_title('YOLO의 진화: 2015-2025', fontsize=16, fontweight='bold',
pad=20)

# 범례
redmon_patch = mpatches.Patch(color='#FF6B6B', label='Joseph Redmon 시대')
community_patch = mpatches.Patch(color='#4ECDC4', label='커뮤니티 주도')
modern_patch = mpatches.Patch(color='#45B7D1', label='차세대 YOLO')
ax.legend(handles=[redmon_patch, community_patch, modern_patch],
loc='upper right')

ax.axis('off')
plt.tight_layout()
plt.show()
```

5.3 YOLO가 특별한 이유 - 기술적 혁신

YOLO가 혁명적인 이유를 쉽게 설명해드리겠습니다:

```
# YOLO vs 전통적 방법 비교 시각화
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import numpy as np

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

# 전통적 방법 (R-CNN)
ax1.set_title('전통적 방법 (R-CNN)', fontsize=14, fontweight='bold')
ax1.set_xlim(0, 10)
ax1.set_ylim(0, 10)

# 원본 이미지
img_rect = patches.Rectangle((1, 1), 8, 8, linewidth=2,
```

```
                            edgecolor='black', facecolor='lightgray')
ax1.add_patch(img_rect)
ax1.text(5, 9.5, '원본 이미지', ha='center', fontweight='bold')

# 수많은 영역 제안
np.random.seed(42)
for i in range(20):
    x = np.random.uniform(1, 6)
    y = np.random.uniform(1, 6)
    w = np.random.uniform(1, 3)
    h = np.random.uniform(1, 3)

    rect = patches.Rectangle((x, y), w, h, linewidth=1,
                            edgecolor='red', facecolor='none',
                            alpha=0.5)
    ax1.add_patch(rect)

ax1.text(5, 0.5, '2000개 이상의 영역을\n각각 분석 (느림!)',
         ha='center', color='red', fontweight='bold')

# YOLO 방법
ax2.set_title('YOLO 방법', fontsize=14, fontweight='bold')
ax2.set_xlim(0, 10)
ax2.set_ylim(0, 10)

# 원본 이미지
img_rect2 = patches.Rectangle((1, 1), 8, 8, linewidth=2,
                             edgecolor='black', facecolor='lightgray')
ax2.add_patch(img_rect2)
ax2.text(5, 9.5, '원본 이미지', ha='center', fontweight='bold')

# 그리드
for i in range(1, 8):
    ax2.axvline(i+1, ymin=0.1, ymax=0.9, color='blue', alpha=0.3)
    ax2.axhline(i+1, xmin=0.1, xmax=0.9, color='blue', alpha=0.3)

# 탐지된 객체
objects = [
    (2, 2, 2, 3, '고양이'),
    (5, 4, 2, 2, '공'),
    (6, 7, 2, 1.5, '신발')
]

for x, y, w, h, label in objects:
    rect = patches.Rectangle((x, y), w, h, linewidth=2,
                            edgecolor='green', facecolor='none')
    ax2.add_patch(rect)
    ax2.text(x+w/2, y+h/2, label, ha='center', va='center',
            fontweight='bold', color='green')

ax2.text(5, 0.5, '한 번에 전체 이미지 분석\n(매우 빠름!)',
         ha='center', color='green', fontweight='bold')

ax1.axis('off')
```

```
ax2.axis('off')
plt.tight_layout()
plt.show()
```

5.4 2025년 최신 YOLO 모델 상세 분석

YOLO11 - 현재 가장 안정적인 선택

```
# YOLO11 아키텍처 이해하기
class YOLO11Architecture:
    """YOLO11의 주요 구성 요소 설명"""

    def __init__(self):
        self.components = {
            'backbone': {
                'name': 'CSPDarknet with C3k2',
                'description': '특징 추출을 위한 백본 네트워크',
                'innovation': 'C3k2 블록으로 더 깊은 네트워크 구성 가능'
            },
            'neck': {
                'name': 'SPPF + PAN',
                'description': '다중 스케일 특징 융합',
                'innovation': '더 빠른 Spatial Pyramid Pooling'
            },
            'head': {
                'name': 'Decoupled Head',
                'description': '분류와 위치 예측 분리',
                'innovation': '각 작업에 최적화된 별도 경로'
            }
        }

    def visualize_architecture(self):
        """아키텍처 시각화"""
        fig, ax = plt.subplots(figsize=(12, 8))

        # 배경
        ax.set_facecolor('#f0f0f0')

        # 입력 이미지
        input_box = FancyBboxPatch((1, 6), 2, 1.5,
                                   boxstyle="round,pad=0.1",
                                   facecolor='lightblue',
                                   edgecolor='darkblue',
                                   linewidth=2)
        ax.add_patch(input_box)
        ax.text(2, 6.75, '입력 이미지\n640×640', ha='center', va='center',
                fontweight='bold')

        # Backbone
        backbone_box = FancyBboxPatch((4, 6), 3, 1.5,
                                      boxstyle="round,pad=0.1",
```

```
                                    facecolor='#FFE5B4',
                                    edgecolor='#FF8C00',
                                    linewidth=2)
        ax.add_patch(backbone_box)
        ax.text(5.5, 6.75, 'CSPDarknet\n+ C3k2 블록', ha='center',
va='center',
                fontweight='bold')

        # Neck
        neck_box = FancyBboxPatch((8, 6), 3, 1.5,
                                  boxstyle="round,pad=0.1",
                                  facecolor='#E6E6FA',
                                  edgecolor='#9370DB',
                                  linewidth=2)
        ax.add_patch(neck_box)
        ax.text(9.5, 6.75, 'SPPF + PAN\n특징 융합', ha='center', va='center',
                fontweight='bold')

        # Head
        head_box = FancyBboxPatch((4, 3), 7, 2,
                                  boxstyle="round,pad=0.1",
                                  facecolor='#FFB6C1',
                                  edgecolor='#DC143C',
                                  linewidth=2)
        ax.add_patch(head_box)
        ax.text(7.5, 4, 'Decoupled Head', ha='center', va='center',
                fontweight='bold', fontsize=12)

        # 출력
        outputs = [
            (3, 0.5, '클래스 예측'),
            (7.5, 0.5, '바운딩 박스'),
            (12, 0.5, '신뢰도 점수')
        ]

        for x, y, label in outputs:
            output_box = FancyBboxPatch((x-1, y), 2, 1,
                                        boxstyle="round,pad=0.1",
                                        facecolor='lightgreen',
                                        edgecolor='darkgreen',
                                        linewidth=2)
            ax.add_patch(output_box)
            ax.text(x, y+0.5, label, ha='center', va='center')

        # 화살표
        arrows = [
            ((3, 6.75), (4, 6.75)),
            ((7, 6.75), (8, 6.75)),
            ((9.5, 6), (7.5, 5)),
            ((7.5, 3), (4, 1.5)),
            ((7.5, 3), (7.5, 1.5)),
            ((7.5, 3), (11, 1.5))
        ]
```

```
        for start, end in arrows:
            ax.annotate('', xy=end, xytext=start,
                        arrowprops=dict(arrowstyle='->', lw=2,
color='gray'))

        ax.set_xlim(0, 14)
        ax.set_ylim(0, 8)
        ax.set_title('YOLO11 아키텍처', fontsize=16, fontweight='bold')
        ax.axis('off')

        plt.tight_layout()
        plt.show()

# 아키텍처 시각화
arch = YOLO11Architecture()
arch.visualize_architecture()
```

YOLOv12 - 미래를 보여주는 실험적 모델

YOLOv12는 2025년 초에 발표된 최신 모델로, Vision Transformer의 장점을 통합했습니다:

1. **하이브리드 백본**: CNN과 ViT의 결합
2. **시간적 일관성**: 비디오에서 객체를 더 안정적으로 추적
3. **자기 지도 학습**: 라벨 없는 데이터로도 학습 가능
4. **엣지 최적화**: 모바일 기기에서도 실행 가능

6장: 객체 추적과 행동 인식 - 움직임을 이해하는 AI

6.1 객체 추적이란?

객체 탐지가 "지금 이 순간 무엇이 어디에 있는가?"를 찾는다면, 객체 추적은 "이 물체가 시간에 따라 어떻게 움직이는가?"를 파악합니다.

6.2 ByteTrack - 간단하지만 강력한 추적 알고리즘

```
# ByteTrack 원리 시각화
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle, FancyBboxPatch
from matplotlib.patches import ConnectionPatch

def visualize_bytetrack():
    """ByteTrack 알고리즘 원리 시각화"""

    fig, axes = plt.subplots(1, 3, figsize=(15, 5))

    # 프레임 t
    ax1 = axes[0]
    ax1.set_title('프레임 t', fontsize=14, fontweight='bold')
```

```
    ax1.set_xlim(0, 10)
    ax1.set_ylim(0, 10)

    # 이전 프레임의 객체들
    prev_objects = [
        {'id': 1, 'pos': (2, 7), 'color': 'red', 'label': '사람 1'},
        {'id': 2, 'pos': (5, 4), 'color': 'blue', 'label': '사람 2'},
        {'id': 3, 'pos': (8, 6), 'color': 'green', 'label': '자동차'}
    ]

    for obj in prev_objects:
        rect = Rectangle(obj['pos'], 1.5, 1.5,
                         facecolor=obj['color'], alpha=0.5)
        ax1.add_patch(rect)
        ax1.text(obj['pos'][0]+0.75, obj['pos'][1]+0.75,
                f"ID:{obj['id']}", ha='center', va='center',
                fontweight='bold')

    # 프레임 t+1
    ax2 = axes[1]
    ax2.set_title('프레임 t+1', fontsize=14, fontweight='bold')
    ax2.set_xlim(0, 10)
    ax2.set_ylim(0, 10)

    # 새로운 탐지 결과
    new_detections = [
        {'pos': (2.5, 6.5), 'score': 0.9},
        {'pos': (5.2, 3.8), 'score': 0.85},
        {'pos': (8.1, 5.8), 'score': 0.7},
        {'pos': (1, 2), 'score': 0.4}  # 낮은 신뢰도
    ]

    for i, det in enumerate(new_detections):
        color = 'gray' if det['score'] < 0.5 else 'black'
        alpha = det['score']
        rect = Rectangle(det['pos'], 1.5, 1.5,
                         facecolor=color, alpha=alpha,
                         edgecolor='black', linewidth=2)
        ax2.add_patch(rect)
        ax2.text(det['pos'][0]+0.75, det['pos'][1]+2,
                f"점수:{det['score']}", ha='center',
                fontsize=8)

    # 매칭 결과
    ax3 = axes[2]
    ax3.set_title('ByteTrack 매칭 결과', fontsize=14, fontweight='bold')
    ax3.set_xlim(0, 10)
    ax3.set_ylim(0, 10)

    # 매칭된 객체들
    matched = [
        {'id': 1, 'old_pos': (2, 7), 'new_pos': (2.5, 6.5), 'color':
'red'},
        {'id': 2, 'old_pos': (5, 4), 'new_pos': (5.2, 3.8), 'color':
```

```
'blue'},
        {'id': 3, 'old_pos': (8, 6), 'new_pos': (8.1, 5.8), 'color':
'green'}
    ]

    for match in matched:
        # 새 위치
        rect = Rectangle(match['new_pos'], 1.5, 1.5,
                         facecolor=match['color'], alpha=0.5)
        ax3.add_patch(rect)
        ax3.text(match['new_pos'][0]+0.75, match['new_pos'][1]+0.75,
                f"ID:{match['id']}", ha='center', va='center',
                fontweight='bold')

        # 이동 경로
        ax3.arrow(match['old_pos'][0]+0.75, match['old_pos'][1]+0.75,
                 match['new_pos'][0]-match['old_pos'][0],
                 match['new_pos'][1]-match['old_pos'][1],
                 head_width=0.2, head_length=0.1,
                 fc='gray', ec='gray', alpha=0.5)

    # 새로운 객체 (낮은 신뢰도지만 추적 시작)
    new_rect = Rectangle((1, 2), 1.5, 1.5,
                         facecolor='orange', alpha=0.5,
                         edgecolor='orange', linewidth=2)
    ax3.add_patch(new_rect)
    ax3.text(1.75, 2.75, "NEW\nID:4", ha='center', va='center',
            fontweight='bold', fontsize=8)

    for ax in axes:
        ax.set_aspect('equal')
        ax.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

visualize_bytetrack()
```

6.3 행동 인식 - AI가 동작을 이해하다

행동 인식은 단순히 "사람이 있다"를 넘어 "사람이 무엇을 하고 있는가"를 파악합니다.

```
# 행동 인식 파이프라인
class ActionRecognitionPipeline:
    """행동 인식의 전체 과정을 보여주는 클래스"""

    def __init__(self):
        self.actions = {
            'walking': '걷기',
            'running': '뛰기',
            'sitting': '앉기',
```

```
            'standing': '서기',
            'waving': '손 흔들기',
            'jumping': '점프하기',
            'dancing': '춤추기',
            'falling': '넘어지기'
        }

    def visualize_pipeline(self):
        """행동 인식 파이프라인 시각화"""

        fig = plt.figure(figsize=(14, 10))

        # 1. 비디오 프레임
        ax1 = plt.subplot(3, 3, 1)
        ax1.set_title('1. 비디오 입력', fontweight='bold')

        # 프레임 시퀀스 표시
        frames = np.random.rand(8, 64, 64, 3)
        frame_strip = np.hstack([frames[i] for i in range(4)])
        ax1.imshow(frame_strip)
        ax1.axis('off')
        ax1.text(0.5, -0.1, '연속된 프레임들', transform=ax1.transAxes,
                ha='center')

        # 2. 사람 탐지
        ax2 = plt.subplot(3, 3, 2)
        ax2.set_title('2. 사람 탐지', fontweight='bold')

        # 바운딩 박스가 있는 프레임
        frame_with_box = np.ones((64, 64, 3))
        ax2.imshow(frame_with_box)
        rect = Rectangle((15, 10), 35, 45, fill=False,
                         edgecolor='red', linewidth=3)
        ax2.add_patch(rect)
        ax2.text(32, 32, '사람', ha='center', va='center',
                fontweight='bold', color='red')
        ax2.axis('off')

        # 3. 포즈 추정
        ax3 = plt.subplot(3, 3, 3)
        ax3.set_title('3. 포즈 추정', fontweight='bold')

        # 스켈레톤 그리기
        ax3.set_xlim(0, 64)
        ax3.set_ylim(64, 0)

        # 관절 위치
        joints = {
            'head': (32, 15),
            'neck': (32, 20),
            'right_shoulder': (25, 22),
            'left_shoulder': (39, 22),
            'right_elbow': (20, 30),
            'left_elbow': (44, 30),
```

```
        'right_hand': (18, 38),
        'left_hand': (46, 38),
        'hip': (32, 35),
        'right_knee': (28, 45),
        'left_knee': (36, 45),
        'right_foot': (26, 55),
        'left_foot': (38, 55)
    }

    # 관절 연결
    connections = [
        ('head', 'neck'),
        ('neck', 'right_shoulder'),
        ('neck', 'left_shoulder'),
        ('right_shoulder', 'right_elbow'),
        ('right_elbow', 'right_hand'),
        ('left_shoulder', 'left_elbow'),
        ('left_elbow', 'left_hand'),
        ('neck', 'hip'),
        ('hip', 'right_knee'),
        ('hip', 'left_knee'),
        ('right_knee', 'right_foot'),
        ('left_knee', 'left_foot')
    ]

    # 스켈레톤 그리기
    for joint, pos in joints.items():
        ax3.plot(pos[0], pos[1], 'o', color='blue', markersize=8)

    for conn in connections:
        start = joints[conn[0]]
        end = joints[conn[1]]
        ax3.plot([start[0], end[0]], [start[1], end[1]],
                 'b-', linewidth=2)

    ax3.axis('off')

    # 4. 시간적 특징 추출
    ax4 = plt.subplot(3, 3, 4)
    ax4.set_title('4. 시간적 특징', fontweight='bold')

    # 시간에 따른 관절 움직임
    t = np.linspace(0, 2*np.pi, 50)
    y1 = np.sin(t) * 10 + 30
    y2 = np.sin(t + np.pi/4) * 8 + 30

    ax4.plot(t, y1, label='오른팔 움직임')
    ax4.plot(t, y2, label='왼팔 움직임')
    ax4.set_xlabel('시간')
    ax4.set_ylabel('위치')
    ax4.legend()
    ax4.grid(True, alpha=0.3)

    # 5. 특징 벡터
```

```
        ax5 = plt.subplot(3, 3, 5)
        ax5.set_title('5. 특징 벡터', fontweight='bold')

        # 특징 벡터 시각화
        features = np.random.rand(10, 10)
        im = ax5.imshow(features, cmap='viridis')
        ax5.set_xlabel('특징 차원')
        ax5.set_ylabel('시간 스텝')
        plt.colorbar(im, ax=ax5)

        # 6. 행동 분류
        ax6 = plt.subplot(3, 3, 6)
        ax6.set_title('6. 행동 분류', fontweight='bold')

        # 분류 결과
        actions = ['걷기', '뛰기', '앉기', '서기', '손 흔들기']
        probabilities = [0.05, 0.02, 0.03, 0.85, 0.05]

        bars = ax6.bar(actions, probabilities)
        bars[3].set_color('green')  # 가장 높은 확률
        ax6.set_ylabel('확률')
        ax6.set_ylim(0, 1)

        # 최종 결과
        ax7 = plt.subplot(3, 1, 3)
        ax7.set_title('최종 결과: 서있기 (85% 확신도)',
                      fontsize=16, fontweight='bold')
        ax7.text(0.5, 0.5, '🧍 사람이 서 있습니다',
                ha='center', va='center', fontsize=20)
        ax7.axis('off')

        plt.tight_layout()
        plt.show()

# 파이프라인 시각화 실행
pipeline = ActionRecognitionPipeline()
pipeline.visualize_pipeline()
```

6.4 최신 행동 인식 모델들

VideoMAE V2 - 마스크 학습의 마법

```
class VideoMAEExplainer:
    """VideoMAE의 작동 원리를 설명하는 클래스"""

    def explain_masking(self):
        """마스킹 학습 원리 설명"""

        fig, axes = plt.subplots(2, 3, figsize=(12, 8))

        # 원본 비디오 프레임
```

```
        ax1 = axes[0, 0]
        ax1.set_title('원본 비디오', fontweight='bold')
        original = np.ones((64, 64, 3))
        ax1.imshow(original)
        ax1.axis('off')

        # 마스킹된 프레임
        ax2 = axes[0, 1]
        ax2.set_title('90% 마스킹', fontweight='bold')
        masked = original.copy()
        mask = np.random.random((64, 64)) > 0.9
        masked[~mask] = 0
        ax2.imshow(masked)
        ax2.axis('off')

        # 모델 예측
        ax3 = axes[0, 2]
        ax3.set_title('모델의 예측', fontweight='bold')
        predicted = np.random.rand(64, 64, 3)
        ax3.imshow(predicted)
        ax3.axis('off')

        # 학습 과정
        ax4 = axes[1, 0]
        ax4.set_title('학습 과정', fontweight='bold')
        epochs = range(1, 101)
        loss = 10 * np.exp(-np.array(epochs) / 20) + np.random.normal(0,
0.1, 100)
        ax4.plot(epochs, loss)
        ax4.set_xlabel('에폭')
        ax4.set_ylabel('손실')
        ax4.grid(True, alpha=0.3)

        # 특징 학습
        ax5 = axes[1, 1]
        ax5.set_title('학습된 특징', fontweight='bold')
        features = [
            '움직임 패턴',
            '객체 형태',
            '시간적 일관성',
            '공간적 관계',
            '색상 분포'
        ]
        importance = [0.9, 0.8, 0.95, 0.7, 0.6]

        bars = ax5.barh(features, importance)
        ax5.set_xlim(0, 1)
        ax5.set_xlabel('중요도')

        # 응용 분야
        ax6 = axes[1, 2]
        ax6.set_title('응용 분야', fontweight='bold')
        applications = """
        • 스포츠 분석
```

```
            • 보안 감시
            • 의료 진단
            • 로봇 제어
            • 게임 인터페이스
            """
            ax6.text(0.1, 0.5, applications, fontsize=12, va='center')
            ax6.axis('off')

            plt.suptitle('VideoMAE: 자기 지도 학습으로 비디오 이해하기',
                         fontsize=16, fontweight='bold')
            plt.tight_layout()
            plt.show()

    explainer = VideoMAEExplainer()
    explainer.explain_masking()
```

7장: 실전 프로젝트 - 똑똑한 감시 시스템 만들기

7.1 프로젝트 개요: AI 보안 시스템

우리가 만들 시스템의 특징:

- 👀 실시간 객체 탐지 및 추적
- 🏃 이상 행동 감지
- 📱 모바일 알림
- 📊 통계 대시보드
- 📹 영상 저장 및 검색

7.2 시스템 아키텍처 설계

```python
# smart_security_system.py - 스마트 보안 시스템

import cv2
import torch
import numpy as np
from ultralytics import YOLO
import supervision as sv
from collections import defaultdict, deque
import time
from datetime import datetime
import json
import os
import threading
import queue
from typing import Dict, List, Tuple, Optional
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from email.mime.image import MIMEImage
```

```
class SmartSecuritySystem:
    """AI 기반 스마트 보안 시스템"""

    def __init__(self, config: Dict = None):
        """시스템 초기화"""
        print("🔒 스마트 보안 시스템을 초기화하고 있습니다...")

        # 설정
        self.config = config or self.get_default_config()

        # 모델 초기화
        self.init_models()

        # 추적기 초기화
        self.tracker = sv.ByteTrack()

        # 데이터 저장소
        self.init_storage()

        # 알림 시스템
        self.init_notification_system()

        print("✅ 시스템 초기화 완료!")

    def get_default_config(self) -> Dict:
        """기본 설정값"""
        return {
            'yolo_model': 'yolo11m.pt',
            'confidence_threshold': 0.5,
            'max_tracked_objects': 50,
            'alert_cooldown': 30,  # 초
            'recording_path': 'security_recordings',
            'suspicious_behaviors': {
                'loitering': {'duration': 60, 'area': 100},  # 60초 이상 머무
르기
                'running': {'speed_threshold': 5.0},  # 빠른 움직임
                'intrusion': {'restricted_zones': []},  # 제한 구역 침입
                'crowding': {'max_people': 10},  # 과밀
                'abandoned_object': {'duration': 300}  # 5분 이상 방치된 물체
            },
            'notification': {
                'email': None,
                'webhook': None
            }
        }

    def init_models(self):
        """AI 모델 초기화"""
        print("  🧠 AI 모델을 로딩하고 있습니다...")

        # YOLO 모델
        self.yolo = YOLO(self.config['yolo_model'])

        # 행동 분류기 (간단한 규칙 기반)
```

```
        self.behavior_analyzer =
BehaviorAnalyzer(self.config['suspicious_behaviors'])

    def init_storage(self):
        """데이터 저장 시스템 초기화"""
        self.storage_path = self.config['recording_path']
        os.makedirs(self.storage_path, exist_ok=True)

        # 추적 데이터
        self.track_history = defaultdict(lambda: {
            'positions': deque(maxlen=300),  # 10초 @ 30fps
            'timestamps': deque(maxlen=300),
            'class': None,
            'first_seen': None,
            'last_seen': None,
            'alerts': []
        })

        # 이벤트 로그
        self.event_log = []

    def init_notification_system(self):
        """알림 시스템 초기화"""
        self.alert_queue = queue.Queue()
        self.last_alert_time = defaultdict(float)

        # 알림 처리 스레드
        self.notification_thread = threading.Thread(
            target=self.process_notifications,
            daemon=True
        )
        self.notification_thread.start()

    def process_frame(self, frame: np.ndarray, frame_id: int) ->
Tuple[np.ndarray, List[Dict]]:
        """단일 프레임 처리"""

        # 1. 객체 탐지
        results = self.yolo(frame,
conf=self.config['confidence_threshold'], verbose=False)[0]
        detections = sv.Detections.from_ultralytics(results)

        # 2. 객체 추적
        tracks = self.tracker.update_with_detections(detections)

        # 3. 추적 데이터 업데이트
        current_time = time.time()
        events = []

        for i in range(len(tracks)):
            track_id = tracks.tracker_id[i]
            class_id = tracks.class_id[i]
            class_name = self.yolo.names[class_id]
            bbox = tracks.xyxy[i]
```

```
            # 중심점 계산
            center = ((bbox[0] + bbox[2]) / 2, (bbox[1] + bbox[3]) / 2)

            # 히스토리 업데이트
            history = self.track_history[track_id]
            history['positions'].append(center)
            history['timestamps'].append(current_time)
            history['class'] = class_name

            if history['first_seen'] is None:
                history['first_seen'] = current_time
                events.append({
                    'type': 'new_object',
                    'track_id': track_id,
                    'class': class_name,
                    'time': current_time
                })

            history['last_seen'] = current_time

        # 4. 행동 분석
        suspicious_activities = self.behavior_analyzer.analyze(
            self.track_history, current_time
        )

        # 5. 알림 생성
        for activity in suspicious_activities:
            if self.should_send_alert(activity):
                self.alert_queue.put({
                    'activity': activity,
                    'frame': frame.copy(),
                    'time': current_time
                })
                events.append(activity)

        # 6. 시각화
        annotated_frame = self.visualize_results(
            frame, tracks, suspicious_activities
        )

        return annotated_frame, events

    def should_send_alert(self, activity: Dict) -> bool:
        """알림을 보낼지 결정"""
        alert_type = activity['type']
        current_time = time.time()

        # 쿨다운 확인
        if current_time - self.last_alert_time[alert_type] <
self.config['alert_cooldown']:
            return False

        self.last_alert_time[alert_type] = current_time
```

```
        return True

    def visualize_results(self, frame: np.ndarray, tracks: sv.Detections,
                          activities: List[Dict]) -> np.ndarray:
        """결과 시각화"""
        annotated = frame.copy()

        # 박스 그리기
        box_annotator = sv.BoxAnnotator()
        annotated = box_annotator.annotate(scene=annotated,
detections=tracks)

        # 추적 ID와 클래스 표시
        for i in range(len(tracks)):
            track_id = tracks.tracker_id[i]
            class_id = tracks.class_id[i]
            class_name = self.yolo.names[class_id]
            bbox = tracks.xyxy[i]

            # 라벨
            label = f"ID:{track_id} {class_name}"

            # 의심 행동이 있는 경우 강조
            is_suspicious = any(
                activity.get('track_id') == track_id
                for activity in activities
            )

            color = (0, 0, 255) if is_suspicious else (0, 255, 0)

            cv2.putText(
                annotated,
                label,
                (int(bbox[0]), int(bbox[1] - 10)),
                cv2.FONT_HERSHEY_SIMPLEX,
                0.5,
                color,
                2
            )

        # 상태 표시
        self.draw_status(annotated, len(tracks), activities)

        return annotated

    def draw_status(self, frame: np.ndarray, num_objects: int, activities:
List[Dict]):
        """상태 정보 표시"""
        h, w = frame.shape[:2]

        # 상단 정보 바
        overlay = frame.copy()
        cv2.rectangle(overlay, (0, 0), (w, 50), (0, 0, 0), -1)
        frame[:50] = cv2.addWeighted(overlay[:50], 0.7, frame[:50], 0.3,
```

```
0)

        # 시간
        current_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        cv2.putText(frame, current_time, (10, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)

        # 객체 수
        cv2.putText(frame, f"Objects: {num_objects}", (300, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)

        # 경고
        if activities:
            alert_text = f"ALERT: {activities[0]['type']}"
            cv2.putText(frame, alert_text, (500, 30),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

    def process_notifications(self):
        """알림 처리 스레드"""
        while True:
            try:
                alert = self.alert_queue.get(timeout=1)
                self.send_notification(alert)
            except queue.Empty:
                continue

    def send_notification(self, alert: Dict):
        """알림 전송"""
        print(f"\n🚨 경고: {alert['activity']['type']}")
        print(f"   시간: {datetime.fromtimestamp(alert['time'])}")
        print(f"   세부사항: {alert['activity'].get('details', 'N/A')}")

        # 이메일 알림 (설정된 경우)
        if self.config['notification']['email']:
            self.send_email_alert(alert)

        # 웹훅 알림 (설정된 경우)
        if self.config['notification']['webhook']:
            self.send_webhook_alert(alert)

        # 스크린샷 저장
        self.save_alert_screenshot(alert)

    def save_alert_screenshot(self, alert: Dict):
        """경고 스크린샷 저장"""
        timestamp =
datetime.fromtimestamp(alert['time']).strftime('%Y%m%d_%H%M%S')
        filename = f"{alert['activity']['type']}_{timestamp}.jpg"
        filepath = os.path.join(self.storage_path, filename)

        cv2.imwrite(filepath, alert['frame'])
        print(f"   📸 스크린샷 저장: {filepath}")

    def run_live_monitoring(self, source=0):
```

```
        """실시간 모니터링 실행"""
        print("\n🎥 실시간 모니터링을 시작합니다...")
        print("'q' 키를 눌러 종료")
        print("'s' 키를 눌러 스크린샷 저장")
        print("'r' 키를 눌러 녹화 시작/중지")

        cap = cv2.VideoCapture(source)

        # 비디오 속성
        fps = int(cap.get(cv2.CAP_PROP_FPS))
        width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

        # 녹화 설정
        recording = False
        video_writer = None

        frame_id = 0

        while True:
            ret, frame = cap.read()
            if not ret:
                break

            # 프레임 처리
            annotated_frame, events = self.process_frame(frame, frame_id)

            # 이벤트 로깅
            for event in events:
                self.log_event(event)

            # 녹화
            if recording and video_writer:
                video_writer.write(annotated_frame)

            # 화면 표시
            cv2.imshow('Smart Security System', annotated_frame)

            # 키 입력 처리
            key = cv2.waitKey(1) & 0xFF

            if key == ord('q'):
                break
            elif key == ord('s'):
                self.save_screenshot(annotated_frame)
            elif key == ord('r'):
                if not recording:
                    # 녹화 시작
                    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
                    video_path = os.path.join(self.storage_path,
f"recording_{timestamp}.mp4")
                    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
                    video_writer = cv2.VideoWriter(video_path, fourcc,
fps, (width, height))
```

```
                    recording = True
                    print(f"🔴 녹화 시작: {video_path}")
                else:
                    # 녹화 중지
                    video_writer.release()
                    video_writer = None
                    recording = False
                    print("⏹️ 녹화 중지")

            frame_id += 1

        # 정리
        cap.release()
        if video_writer:
            video_writer.release()
        cv2.destroyAllWindows()

        # 최종 리포트
        self.generate_report()

    def log_event(self, event: Dict):
        """이벤트 로깅"""
        event['timestamp'] = datetime.now().isoformat()
        self.event_log.append(event)

        # 주기적으로 파일에 저장
        if len(self.event_log) % 100 == 0:
            self.save_event_log()

    def save_event_log(self):
        """이벤트 로그 저장"""
        log_path = os.path.join(self.storage_path, 'event_log.json')
        with open(log_path, 'w', encoding='utf-8') as f:
            json.dump(self.event_log, f, ensure_ascii=False, indent=2)

    def generate_report(self):
        """모니터링 리포트 생성"""
        print("\n📊 모니터링 리포트")
        print("=" * 50)

        # 전체 통계
        total_objects = len(self.track_history)
        print(f"총 추적 객체 수: {total_objects}")

        # 클래스별 통계
        class_counts = defaultdict(int)
        for track_data in self.track_history.values():
            if track_data['class']:
                class_counts[track_data['class']] += 1

        print("\n클래스별 객체 수:")
        for class_name, count in sorted(class_counts.items()):
            print(f"  - {class_name}: {count}")
```

```
        # 경고 통계
        alert_counts = defaultdict(int)
        for event in self.event_log:
            if event.get('type') in ['loitering', 'running', 'intrusion']:
                alert_counts[event['type']] += 1

        print("\n경고 유형별 횟수:")
        for alert_type, count in sorted(alert_counts.items()):
            print(f"  - {alert_type}: {count}")

        print("=" * 50)


class BehaviorAnalyzer:
    """행동 분석기"""

    def __init__(self, config: Dict):
        self.config = config

    def analyze(self, track_history: Dict, current_time: float) ->
List[Dict]:
        """의심스러운 행동 분석"""
        suspicious_activities = []

        for track_id, history in track_history.items():
            if len(history['positions']) < 10:
                continue

            # 1. 배회 감지 (Loitering)
            loitering = self.detect_loitering(history, current_time)
            if loitering:
                suspicious_activities.append({
                    'type': 'loitering',
                    'track_id': track_id,
                    'details': loitering,
                    'time': current_time
                })

            # 2. 빠른 움직임 감지 (Running)
            running = self.detect_running(history)
            if running:
                suspicious_activities.append({
                    'type': 'running',
                    'track_id': track_id,
                    'details': running,
                    'time': current_time
                })

            # 3. 방치된 물체 감지
            if history['class'] in ['backpack', 'suitcase', 'handbag']:
                abandoned = self.detect_abandoned_object(history,
current_time)
                if abandoned:
                    suspicious_activities.append({
```

```
                        'type': 'abandoned_object',
                        'track_id': track_id,
                        'details': abandoned,
                        'time': current_time
                    })

        return suspicious_activities

    def detect_loitering(self, history: Dict, current_time: float) ->
Optional[Dict]:
        """배회 감지"""
        duration = current_time - history['first_seen']

        if duration < self.config['loitering']['duration']:
            return None

        # 이동 범위 계산
        positions = list(history['positions'])
        if len(positions) < 2:
            return None

        xs = [p[0] for p in positions]
        ys = [p[1] for p in positions]

        x_range = max(xs) - min(xs)
        y_range = max(ys) - min(ys)
        area = x_range * y_range

        if area < self.config['loitering']['area']:
            return {
                'duration': duration,
                'area': area,
                'message': f"{duration:.1f}초 동안 작은 영역에 머물러 있음"
            }

        return None

    def detect_running(self, history: Dict) -> Optional[Dict]:
        """빠른 움직임 감지"""
        positions = list(history['positions'])
        timestamps = list(history['timestamps'])

        if len(positions) < 5:
            return None

        # 최근 5프레임의 속도 계산
        speeds = []
        for i in range(len(positions) - 5, len(positions) - 1):
            dx = positions[i+1][0] - positions[i][0]
            dy = positions[i+1][1] - positions[i][1]
            dt = timestamps[i+1] - timestamps[i]

            if dt > 0:
                speed = np.sqrt(dx**2 + dy**2) / dt
```

```
                speeds.append(speed)

        avg_speed = np.mean(speeds) if speeds else 0

        if avg_speed > self.config['running']['speed_threshold']:
            return {
                'speed': avg_speed,
                'message': f"빠른 속도로 이동 중 ({avg_speed:.1f} pixels/s)"
            }

        return None

    def detect_abandoned_object(self, history: Dict, current_time: float)
-> Optional[Dict]:
        """방치된 물체 감지"""
        # 마지막으로 본 시간 확인
        time_since_last_movement = current_time - history['last_seen']

        if time_since_last_movement > 1.0:  # 1초 이상 업데이트 없음
            # 정지 상태 확인
            positions = list(history['positions'])[-10:]
            if len(positions) < 10:
                return None

            # 위치 변화 계산
            position_std = np.std([p[0] for p in positions]) +
np.std([p[1] for p in positions])

            if position_std < 5.0:  # 거의 움직임 없음
                stationary_duration = current_time - history['first_seen']

                if stationary_duration > self.config['abandoned_object']
['duration']:
                    return {
                        'duration': stationary_duration,
                        'message': f"물체가 {stationary_duration:.1f}초 동안
방치됨"
                    }

        return None


# 웹 대시보드
def create_security_dashboard():
    """보안 시스템 웹 대시보드"""
    import gradio as gr
    import plotly.graph_objects as go

    security_system = SmartSecuritySystem()

    def process_video_file(video_file, confidence_threshold):
        """업로드된 비디오 파일 처리"""
        if video_file is None:
            return None, "비디오를 업로드해주세요"
```

```
        # 임시 설정 업데이트
        security_system.config['confidence_threshold'] =
confidence_threshold

        cap = cv2.VideoCapture(video_file.name)
        fps = int(cap.get(cv2.CAP_PROP_FPS))

        frames = []
        events = []
        frame_id = 0

        while len(frames) < 150:  # 최대 5초 처리
            ret, frame = cap.read()
            if not ret:
                break

            annotated_frame, frame_events =
security_system.process_frame(frame, frame_id)
            frames.append(annotated_frame)
            events.extend(frame_events)
            frame_id += 1

        cap.release()

        # 결과 비디오 생성
        output_path = "analyzed_video.mp4"
        height, width = frames[0].shape[:2]
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

        for frame in frames:
            out.write(frame)

        out.release()

        # 이벤트 요약
        event_summary = f"총 {len(events)}개의 이벤트가 감지되었습니다.\n"
        event_types = defaultdict(int)
        for event in events:
            event_types[event['type']] += 1

        for event_type, count in event_types.items():
            event_summary += f"- {event_type}: {count}회\n"

        return output_path, event_summary

    def generate_statistics():
        """통계 그래프 생성"""
        # 시간대별 객체 수
        hours = list(range(24))
        object_counts = np.random.poisson(10, 24)  # 예시 데이터

        fig1 = go.Figure(data=go.Bar(x=hours, y=object_counts))
```

```
        fig1.update_layout(
            title="시간대별 객체 탐지 수",
            xaxis_title="시간",
            yaxis_title="객체 수"
        )

        # 클래스별 분포
        classes = ['person', 'car', 'bicycle', 'motorcycle', 'truck']
        class_counts = [45, 30, 15, 8, 12]

        fig2 = go.Figure(data=go.Pie(labels=classes, values=class_counts))
        fig2.update_layout(title="객체 클래스 분포")

        return fig1, fig2

    # Gradio 인터페이스
    with gr.Blocks(title="AI 보안 시스템", theme=gr.themes.Soft()) as demo:
        gr.Markdown("""
        # 🔒 AI 기반 스마트 보안 시스템

        실시간 객체 탐지, 추적, 그리고 이상 행동 감지
        """)

        with gr.Tab("📹 비디오 분석"):
            with gr.Row():
                with gr.Column():
                    video_input = gr.Video(label="비디오 업로드")
                    confidence_slider = gr.Slider(
                        minimum=0.1,
                        maximum=0.9,
                        value=0.5,
                        step=0.1,
                        label="탐지 신뢰도 임계값"
                    )
                    analyze_btn = gr.Button("분석 시작", variant="primary")

                with gr.Column():
                    video_output = gr.Video(label="분석 결과")
                    event_summary = gr.Textbox(label="이벤트 요약", lines=5)

            analyze_btn.click(
                fn=process_video_file,
                inputs=[video_input, confidence_slider],
                outputs=[video_output, event_summary]
            )

        with gr.Tab("📊 통계 대시보드"):
            with gr.Row():
                hourly_chart = gr.Plot(label="시간대별 통계")
                class_chart = gr.Plot(label="객체 클래스 분포")

            refresh_btn = gr.Button("통계 새로고침")
            refresh_btn.click(
                fn=generate_statistics,
```

```
                outputs=[hourly_chart, class_chart]
            )

        with gr.Tab("⚙ 설정"):
            gr.Markdown("""
            ### 시스템 설정

            - **알림 이메일**: security@example.com
            - **녹화 저장 경로**: /security_recordings
            - **최대 추적 객체 수**: 50
            - **알림 쿨다운**: 30초

            ### 의심 행동 설정

            - **배회**: 60초 이상 같은 장소에 머무르기
            - **빠른 움직임**: 5 pixels/s 이상의 속도
            - **방치된 물체**: 5분 이상 움직임 없음
            """)

        # 예시 추가
        gr.Examples(
            examples=[
                ["example_security_1.mp4", 0.5],
                ["example_security_2.mp4", 0.7]
            ],
            inputs=[video_input, confidence_slider]
        )

    return demo

# 시스템 실행
def run_security_system():
    """보안 시스템 실행"""
    print("""
    ╔═══════════════════════════════════════╗
    ║      🔒 AI 스마트 보안 시스템 🔒      ║
    ║                                       ║
    ║   실시간 감시와 이상 행동 탐지        ║
    ╚═══════════════════════════════════════╝
    """)

    print("\n실행 모드를 선택하세요:")
    print("1. 실시간 카메라 모니터링")
    print("2. 비디오 파일 분석")
    print("3. 웹 대시보드")

    choice = input("\n선택 (1-3): ")

    if choice == "1":
        system = SmartSecuritySystem()
        system.run_live_monitoring(source=0)  # 웹캠

    elif choice == "2":
        video_path = input("비디오 파일 경로: ")
```

```
            system = SmartSecuritySystem()
            system.run_live_monitoring(source=video_path)

        elif choice == "3":
            dashboard = create_security_dashboard()
            dashboard.launch(share=True)

        else:
            print("잘못된 선택입니다.")

    if __name__ == "__main__":
        run_security_system()
```

7.3 고급 기능 구현

이제 더 고급 기능들을 추가해봅시다:

```
    # advanced_security_features.py - 고급 보안 기능

    import face_recognition
    import pickle
    from sklearn.cluster import DBSCAN
    import networkx as nx
    from collections import Counter

    class AdvancedSecurityFeatures:
        """고급 보안 기능 모음"""

        def __init__(self):
            self.known_faces = {}
            self.load_known_faces()

        def load_known_faces(self):
            """알려진 얼굴 데이터 로드"""
            try:
                with open('known_faces.pkl', 'rb') as f:
                    self.known_faces = pickle.load(f)
            except FileNotFoundError:
                print("알려진 얼굴 데이터가 없습니다.")

        def face_recognition_analysis(self, frame: np.ndarray) -> List[Dict]:
            """얼굴 인식 분석"""
            # 얼굴 찾기
            face_locations = face_recognition.face_locations(frame)
            face_encodings = face_recognition.face_encodings(frame,
    face_locations)

            results = []

            for (top, right, bottom, left), face_encoding in
    zip(face_locations, face_encodings):
```

```
            # 알려진 얼굴과 비교
            matches = face_recognition.compare_faces(
                list(self.known_faces.values()),
                face_encoding
            )

            name = "Unknown"

            if True in matches:
                match_index = matches.index(True)
                name = list(self.known_faces.keys())[match_index]

            results.append({
                'name': name,
                'location': (left, top, right, bottom),
                'authorized': name != "Unknown"
            })

        return results

    def crowd_analysis(self, detections: sv.Detections) -> Dict:
        """군중 분석"""
        # 사람만 필터링
        person_indices = [i for i, class_id in
enumerate(detections.class_id)
                          if class_id == 0]  # 0은 'person' 클래스

        if len(person_indices) < 2:
            return {'density': 'low', 'clusters': 0}

        # 위치 추출
        positions = []
        for i in person_indices:
            bbox = detections.xyxy[i]
            center = ((bbox[0] + bbox[2]) / 2, (bbox[1] + bbox[3]) / 2)
            positions.append(center)

        positions = np.array(positions)

        # DBSCAN 클러스터링
        clustering = DBSCAN(eps=100, min_samples=3).fit(positions)
        n_clusters = len(set(clustering.labels_)) - (1 if -1 in
clustering.labels_ else 0)

        # 밀도 계산
        total_people = len(person_indices)
        if total_people < 5:
            density = 'low'
        elif total_people < 15:
            density = 'medium'
        else:
            density = 'high'

        return {
```

```
            'density': density,
            'clusters': n_clusters,
            'total_people': total_people,
            'positions': positions,
            'labels': clustering.labels_
        }

    def path_prediction(self, track_history: Dict, track_id: int) ->
Optional[np.ndarray]:
        """경로 예측"""
        if track_id not in track_history:
            return None

        positions = list(track_history[track_id]['positions'])

        if len(positions) < 10:
            return None

        # 최근 10개 위치
        recent_positions = np.array(positions[-10:])

        # 간단한 선형 예측
        x_positions = recent_positions[:, 0]
        y_positions = recent_positions[:, 1]

        # 선형 회귀
        t = np.arange(len(x_positions))
        x_coef = np.polyfit(t, x_positions, 1)
        y_coef = np.polyfit(t, y_positions, 1)

        # 다음 5프레임 예측
        future_t = np.arange(len(x_positions), len(x_positions) + 5)
        future_x = np.polyval(x_coef, future_t)
        future_y = np.polyval(y_coef, future_t)

        future_positions = np.column_stack((future_x, future_y))

        return future_positions

    def zone_monitoring(self, frame_shape: Tuple[int, int]) -> Dict:
        """구역별 모니터링 설정"""
        height, width = frame_shape

        zones = {
            'entrance': {
                'polygon': np.array([
                    [0, height * 0.7],
                    [width * 0.3, height * 0.7],
                    [width * 0.3, height],
                    [0, height]
                ], dtype=np.int32),
                'type': 'entrance',
                'max_loitering_time': 30
            },
```

```
            'restricted': {
                'polygon': np.array([
                    [width * 0.7, 0],
                    [width, 0],
                    [width, height * 0.3],
                    [width * 0.7, height * 0.3]
                ], dtype=np.int32),
                'type': 'restricted',
                'authorized_only': True
            },
            'exit': {
                'polygon': np.array([
                    [width * 0.7, height * 0.7],
                    [width, height * 0.7],
                    [width, height],
                    [width * 0.7, height]
                ], dtype=np.int32),
                'type': 'exit',
                'direction': 'out'
            }
        }

        return zones

    def check_zone_violations(self, position: Tuple[float, float],
                              zones: Dict, is_authorized: bool = False) ->
List[str]:
        """구역 위반 확인"""
        violations = []

        point = np.array(position, dtype=np.float32)

        for zone_name, zone_info in zones.items():
            # 점이 다각형 내부에 있는지 확인
            result = cv2.pointPolygonTest(zone_info['polygon'], point,
False)

            if result >= 0:  # 내부 또는 경계
                if zone_info['type'] == 'restricted' and not
is_authorized:
                    violations.append(f"Unauthorized access to
{zone_name}")

        return violations

    def network_analysis(self, track_history: Dict, time_window: float =
60) -> nx.Graph:
        """객체 간 네트워크 분석"""
        G = nx.Graph()

        current_time = time.time()

        # 활성 트랙만 선택
        active_tracks = {
```

```
            track_id: data for track_id, data in track_history.items()
            if current_time - data['last_seen'] < time_window
        }

        # 노드 추가
        for track_id, data in active_tracks.items():
            G.add_node(track_id,
                       object_class=data['class'],
                       duration=current_time - data['first_seen'])

        # 엣지 추가 (근접성 기반)
        track_ids = list(active_tracks.keys())

        for i in range(len(track_ids)):
            for j in range(i + 1, len(track_ids)):
                id1, id2 = track_ids[i], track_ids[j]

                # 최근 위치
                if active_tracks[id1]['positions'] and active_tracks[id2]
['positions']:
                    pos1 = active_tracks[id1]['positions'][-1]
                    pos2 = active_tracks[id2]['positions'][-1]

                    distance = np.sqrt((pos1[0] - pos2[0])**2 + (pos1[1] -
pos2[1])**2)

                    if distance < 100:  # 근접 임계값
                        G.add_edge(id1, id2, weight=1/distance)

        return G

    def generate_heatmap(self, track_history: Dict, frame_shape:
Tuple[int, int]) -> np.ndarray:
        """움직임 히트맵 생성"""
        height, width = frame_shape
        heatmap = np.zeros((height, width), dtype=np.float32)

        for track_data in track_history.values():
            for position in track_data['positions']:
                x, y = int(position[0]), int(position[1])

                # 가우시안 커널 적용
                if 0 <= x < width and 0 <= y < height:
                    cv2.circle(heatmap, (x, y), 20, 1, -1)

        # 정규화
        heatmap = cv2.GaussianBlur(heatmap, (21, 21), 0)
        heatmap = (heatmap / heatmap.max() * 255).astype(np.uint8)

        # 컬러맵 적용
        heatmap_colored = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)

        return heatmap_colored
```

```
# 실시간 분석 대시보드
class RealtimeAnalyticsDashboard:
    """실시간 분석 대시보드"""

    def __init__(self, security_system: SmartSecuritySystem):
        self.security_system = security_system
        self.advanced_features = AdvancedSecurityFeatures()

    def create_dashboard(self):
        """대시보드 생성"""
        import plotly.graph_objects as go
        from plotly.subplots import make_subplots
        import dash
        from dash import dcc, html
        from dash.dependencies import Input, Output
        import dash_bootstrap_components as dbc

        # Dash 앱 초기화
        app = dash.Dash(__name__, external_stylesheets=
[dbc.themes.BOOTSTRAP])

        app.layout = dbc.Container([
            dbc.Row([
                dbc.Col([
                    html.H1("🔒 실시간 보안 분석 대시보드", className="text-
center mb-4"),
                    html.Hr()
                ])
            ]),

            dbc.Row([
                dbc.Col([
                    dcc.Graph(id='live-video-feed'),
                    dcc.Interval(id='video-update', interval=100)  # 100ms
마다 업데이트
                ], width=8),

                dbc.Col([
                    dbc.Card([
                        dbc.CardHeader("실시간 통계"),
                        dbc.CardBody([
                            html.H4(id='total-objects', children="0"),
                            html.P("총 객체 수"),
                            html.Hr(),
                            html.H4(id='alert-count', children="0"),
                            html.P("경고 횟수"),
                            html.Hr(),
                            html.H4(id='crowd-density', children="Low"),
                            html.P("군중 밀도")
                        ])
                    ])
                ], width=4)
            ], className="mb-4"),
```

```
            dbc.Row([
                dbc.Col([
                    dcc.Graph(id='heatmap')
                ], width=6),

                dbc.Col([
                    dcc.Graph(id='network-graph')
                ], width=6)
            ]),

            dbc.Row([
                dbc.Col([
                    dcc.Graph(id='timeline-chart'),
                    dcc.Interval(id='chart-update', interval=1000)  # 1초마
다 업데이트
                ])
            ], className="mt-4")
        ], fluid=True)

        # 콜백 함수들
        @app.callback(
            [Output('total-objects', 'children'),
             Output('alert-count', 'children'),
             Output('crowd-density', 'children')],
            [Input('video-update', 'n_intervals')]
        )
        def update_stats(n):
            """통계 업데이트"""
            total_objects = len(self.security_system.track_history)
            alert_count = len([e for e in self.security_system.event_log
                               if e.get('type') in ['loitering',
'running']])

            # 군중 밀도 계산 (간단한 예시)
            active_objects = sum(1 for track in
self.security_system.track_history.values()
                                 if time.time() - track['last_seen'] < 5)

            if active_objects < 5:
                density = "Low"
            elif active_objects < 15:
                density = "Medium"
            else:
                density = "High"

            return str(total_objects), str(alert_count), density

        @app.callback(
            Output('timeline-chart', 'figure'),
            [Input('chart-update', 'n_intervals')]
        )
        def update_timeline(n):
            """타임라인 차트 업데이트"""
            # 최근 60초 데이터
```

```
            current_time = time.time()
            time_bins = defaultdict(int)

            for event in self.security_system.event_log:
                if 'time' in event:
                    time_diff = current_time - event['time']
                    if time_diff < 60:
                        bin_index = int(time_diff / 5)  # 5초 단위
                        time_bins[bin_index] += 1

            x = list(range(12))
            y = [time_bins.get(11-i, 0) for i in x]

            fig = go.Figure(data=go.Bar(x=x, y=y))
            fig.update_layout(
                title="최근 60초 이벤트 타임라인",
                xaxis_title="시간 (5초 단위)",
                yaxis_title="이벤트 수",
                xaxis=dict(ticktext=[f"{i*5}s" for i in range(12)],
                           tickvals=list(range(12)))
            )

            return fig

        return app

# 전체 시스템 통합
def run_advanced_security_system():
    """고급 보안 시스템 실행"""

    print("""
    ╔═══════════════════════════════════════════╗
    ║    🔒 고급 AI 보안 시스템 v2.0 🔒          ║
    ╠═══════════════════════════════════════════╣
    ║  • 얼굴 인식          • 군중 분석          ║
    ║  • 경로 예측          • 구역 모니터링      ║
    ║  • 네트워크 분석      • 히트맵 생성        ║
    ╚═══════════════════════════════════════════╝
    """)

    security_system = SmartSecuritySystem()
    advanced_features = AdvancedSecurityFeatures()

    # 실시간 모니터링 with 고급 기능
    cap = cv2.VideoCapture(0)

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # 기본 처리
        annotated_frame, events = security_system.process_frame(frame, 0)
```

```
            # 얼굴 인식
            faces = advanced_features.face_recognition_analysis(frame)
            for face in faces:
                x1, y1, x2, y2 = face['location']
                color = (0, 255, 0) if face['authorized'] else (0, 0, 255)
                cv2.rectangle(annotated_frame, (x1, y1), (x2, y2), color, 2)
                cv2.putText(annotated_frame, face['name'], (x1, y1-10),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

            # 히트맵 오버레이
            heatmap = advanced_features.generate_heatmap(
                security_system.track_history,
                frame.shape[:2]
            )
            annotated_frame = cv2.addWeighted(annotated_frame, 0.7, heatmap,
0.3, 0)

            cv2.imshow('Advanced Security System', annotated_frame)

            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

    cap.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    run_advanced_security_system()
```

Day 2 마무리 및 과제

오늘 배운 내용 정리

1. **YOLO의 역사와 발전**
   - YOLOv1부터 YOLOv12까지의 진화
   - 각 버전의 특징과 혁신
   - 실시간 객체 탐지의 원리

2. **객체 추적 기술**
   - ByteTrack 알고리즘
   - 다중 객체 추적
   - 추적 데이터 분석

3. **행동 인식**
   - VideoMAE와 최신 모델들
   - 시간적 특징 추출
   - 행동 분류와 예측

4. **실전 프로젝트**

- 스마트 보안 시스템 구축
- 이상 행동 탐지
- 실시간 알림 시스템
- 고급 분석 기능

실습 과제

1. **기본 과제**
   - 자신만의 객체 탐지 규칙 추가
   - 새로운 이상 행동 패턴 정의
   - 알림 시스템 커스터마이징
2. **도전 과제**
   - 다중 카메라 지원 추가
   - 3D 포즈 추정 통합
   - 예측 경로 시각화
3. **창의 과제**
   - 스포츠 경기 분석 시스템
   - 매장 고객 행동 분석
   - 교통 흐름 모니터링

내일 예고

마지막 날인 내일은 모든 것을 통합합니다!

- 모델 최적화와 경량화
- 음성 인터페이스 구축
- 최종 프로젝트: 음성 대화형 AI 비서

오늘도 수고 많으셨습니다! 🎉

🎯 Day 3: AI 에이전트 - 모든 것을 통합하다

8장: 모델 최적화의 마법 - 더 빠르고 가볍게

8.1 왜 모델 최적화가 중요한가?

여러분이 만든 멋진 AI 모델이 있다고 상상해보세요. 하지만 실행하는데 10초가 걸리고, 32GB 메모리가 필요하다면? 실제로 사용하기 어렵겠죠. 모델 최적화는 이런 문제를 해결합니다!

8.2 2025년 최신 최적화 기술들

```
# model_optimization_showcase.py - 모델 최적화 기술 시연

import torch
```

```
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt
from typing import Dict, List, Tuple
import time

class ModelOptimizationShowcase:
    """다양한 모델 최적화 기술을 보여주는 클래스"""

    def __init__(self):
        self.techniques = {
            'original': '원본 모델',
            'quantization': '양자화 (Quantization)',
            'pruning': '가지치기 (Pruning)',
            'distillation': '지식 증류 (Knowledge Distillation)',
            'mixed_precision': '혼합 정밀도 (Mixed Precision)',
            'compilation': '컴파일 최적화'
        }

    def visualize_optimization_impact(self):
        """최적화 기술의 효과 시각화"""

        fig, axes = plt.subplots(2, 2, figsize=(14, 10))

        # 1. 모델 크기 비교
        ax1 = axes[0, 0]
        techniques = list(self.techniques.keys())
        sizes = [100, 25, 70, 30, 100, 100]  # MB
        colors = ['red', 'green', 'blue', 'orange', 'purple', 'brown']

        bars = ax1.bar(techniques, sizes, color=colors)
        ax1.set_ylabel('모델 크기 (MB)')
        ax1.set_title('모델 크기 비교', fontweight='bold')
        ax1.set_xticklabels([self.techniques[t] for t in techniques],
rotation=45, ha='right')

        # 크기 감소율 표시
        for i, (bar, size) in enumerate(zip(bars, sizes)):
            if i > 0:
                reduction = (sizes[0] - size) / sizes[0] * 100
                if reduction > 0:
                    ax1.text(bar.get_x() + bar.get_width()/2,
bar.get_height() + 2,
                             f'-{reduction:.0f}%', ha='center',
fontweight='bold')

        # 2. 추론 속도 비교
        ax2 = axes[0, 1]
        speeds = [1.0, 0.3, 0.7, 0.5, 0.2, 0.15]  # 초

        bars2 = ax2.bar(techniques, speeds, color=colors)
        ax2.set_ylabel('추론 시간 (초)')
        ax2.set_title('추론 속도 비교', fontweight='bold')
        ax2.set_xticklabels([self.techniques[t] for t in techniques],
```

```
rotation=45, ha='right')

        # 속도 향상 표시
        for i, (bar, speed) in enumerate(zip(bars2, speeds)):
            if i > 0:
                speedup = speeds[0] / speed
                if speedup > 1:
                    ax2.text(bar.get_x() + bar.get_width()/2,
bar.get_height() + 0.02,
                             f'{speedup:.1f}x', ha='center',
fontweight='bold')

        # 3. 정확도 vs 속도 트레이드오프
        ax3 = axes[1, 0]
        accuracies = [95, 93, 94, 92, 95, 95]  # %

        # 버블 차트
        for i, (tech, acc, speed, size) in enumerate(zip(techniques,
accuracies, speeds, sizes)):
            ax3.scatter(speed, acc, s=size*10, alpha=0.6, c=colors[i],
                        label=self.techniques[tech])

        ax3.set_xlabel('추론 시간 (초)')
        ax3.set_ylabel('정확도 (%)')
        ax3.set_title('정확도 vs 속도 트레이드오프', fontweight='bold')
        ax3.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
        ax3.grid(True, alpha=0.3)

        # 4. 메모리 사용량
        ax4 = axes[1, 1]
        memory_usage = [8.0, 2.0, 5.5, 3.0, 4.0, 7.5]  # GB

        # 원형 차트
        explode = [0.1 if m == min(memory_usage) else 0 for m in
memory_usage]
        wedges, texts, autotexts = ax4.pie(memory_usage,
labels=techniques,
                                           autopct='%1.1f GB',
explode=explode,
                                           colors=colors)
        ax4.set_title('메모리 사용량 비교', fontweight='bold')

        plt.tight_layout()
        plt.show()

# 시각화 실행
showcase = ModelOptimizationShowcase()
showcase.visualize_optimization_impact()
```

8.3 양자화 (Quantization) 깊이 이해하기

양자화는 모델의 가중치와 활성화를 낮은 정밀도로 표현하는 기술입니다.

```
class QuantizationExplainer:
    """양자화 기술을 설명하는 클래스"""

    def explain_quantization_types(self):
        """다양한 양자화 기법 설명"""

        fig, axes = plt.subplots(2, 3, figsize=(15, 10))

        # 1. FP32 vs INT8 비교
        ax1 = axes[0, 0]
        ax1.set_title('FP32 vs INT8', fontweight='bold')

        # 숫자 범위 표시
        fp32_range = np.linspace(-1, 1, 1000)
        int8_range = np.linspace(-128, 127, 256)

        ax1.hist(fp32_range, bins=50, alpha=0.5, label='FP32 (32비트)',
color='blue')
        ax1.hist(int8_range/128, bins=50, alpha=0.5, label='INT8 (8비트)',
color='red')
        ax1.set_xlabel('값의 범위')
        ax1.set_ylabel('빈도')
        ax1.legend()

        # 2. 동적 양자화
        ax2 = axes[0, 1]
        ax2.set_title('동적 양자화', fontweight='bold')

        # 가중치 분포
        weights = np.random.normal(0, 0.5, 1000)
        ax2.hist(weights, bins=50, alpha=0.7, color='green')
        ax2.axvline(weights.min(), color='red', linestyle='--',
label='Min')
        ax2.axvline(weights.max(), color='red', linestyle='--',
label='Max')
        ax2.set_xlabel('가중치 값')
        ax2.set_ylabel('빈도')
        ax2.legend()

        # 3. 정적 양자화
        ax3 = axes[0, 2]
        ax3.set_title('정적 양자화', fontweight='bold')

        # 캘리브레이션 데이터
        calibration_data = np.random.normal(0, 1, 100)
        ax3.plot(calibration_data, 'o-', alpha=0.5, label='캘리브레이션 데이터')
        ax3.axhline(y=np.percentile(calibration_data, 99), color='red',
                    linestyle='--', label='99% 백분위수')
        ax3.axhline(y=np.percentile(calibration_data, 1), color='red',
                    linestyle='--')
        ax3.set_xlabel('샘플')
        ax3.set_ylabel('활성화 값')
        ax3.legend()
```

```
        # 4. QAT (Quantization Aware Training)
        ax4 = axes[1, 0]
        ax4.set_title('QAT 학습 과정', fontweight='bold')

        epochs = np.arange(100)
        normal_loss = 2 * np.exp(-epochs/20) + 0.1
        qat_loss = 2 * np.exp(-epochs/25) + 0.15

        ax4.plot(epochs, normal_loss, label='일반 학습', linewidth=2)
        ax4.plot(epochs, qat_loss, label='QAT 학습', linewidth=2)
        ax4.set_xlabel('에폭')
        ax4.set_ylabel('손실')
        ax4.legend()
        ax4.grid(True, alpha=0.3)

        # 5. 양자화 오류
        ax5 = axes[1, 1]
        ax5.set_title('양자화 오류 분석', fontweight='bold')

        original = np.linspace(-1, 1, 100)
        quantized = np.round(original * 127) / 127
        error = np.abs(original - quantized)

        ax5.plot(original, error, linewidth=2, color='red')
        ax5.fill_between(original, 0, error, alpha=0.3, color='red')
        ax5.set_xlabel('원본 값')
        ax5.set_ylabel('양자화 오류')
        ax5.grid(True, alpha=0.3)

        # 6. 비트 수에 따른 성능
        ax6 = axes[1, 2]
        ax6.set_title('비트 수별 성능', fontweight='bold')

        bits = [32, 16, 8, 4, 2, 1]
        accuracy = [95, 94.8, 94.2, 92.5, 88, 75]
        model_size = [100, 50, 25, 12.5, 6.25, 3.125]

        ax6_twin = ax6.twinx()

        line1 = ax6.plot(bits, accuracy, 'b-o', label='정확도 (%)',
linewidth=2)
        line2 = ax6_twin.plot(bits, model_size, 'r-s', label='모델 크기
(MB)', linewidth=2)

        ax6.set_xlabel('비트 수')
        ax6.set_ylabel('정확도 (%)', color='b')
        ax6_twin.set_ylabel('모델 크기 (MB)', color='r')

        lines = line1 + line2
        labels = [l.get_label() for l in lines]
        ax6.legend(lines, labels, loc='center right')

        plt.tight_layout()
```

```
        plt.show()

    def implement_simple_quantization(self):
        """간단한 양자화 구현 예시"""

        print("\n=== 간단한 INT8 양자화 구현 ===\n")

        # 예시 텐서
        original_tensor = torch.randn(5, 5) * 2
        print("원본 텐서 (FP32):")
        print(original_tensor)
        print(f"메모리 사용: {original_tensor.element_size() *
original_tensor.nelement()} bytes")

        # 양자화
        scale = original_tensor.abs().max() / 127
        zero_point = 0

        quantized_tensor = torch.round(original_tensor /
scale).clamp(-128, 127).to(torch.int8)

        print(f"\n양자화된 텐서 (INT8):")
        print(quantized_tensor)
        print(f"메모리 사용: {quantized_tensor.element_size() *
quantized_tensor.nelement()} bytes")
        print(f"스케일: {scale:.6f}")

        # 역양자화
        dequantized_tensor = quantized_tensor.float() * scale

        print(f"\n역양자화된 텐서:")
        print(dequantized_tensor)

        # 오류 계산
        error = torch.abs(original_tensor - dequantized_tensor)
        print(f"\n양자화 오류:")
        print(f"평균 오류: {error.mean():.6f}")
        print(f"최대 오류: {error.max():.6f}")

# 양자화 설명 실행
explainer = QuantizationExplainer()
explainer.explain_quantization_types()
explainer.implement_simple_quantization()
```

8.4 실제 모델 최적화 실습

이제 실제 모델을 최적화해봅시다:

```
# practical_optimization.py - 실제 모델 최적화

import torch
```

```
import torch.nn as nn
import torch.quantization as quantization
from torch.utils.data import DataLoader, TensorDataset
import time
import os

class OptimizableModel(nn.Module):
    """최적화할 예시 모델"""

    def __init__(self, input_size=784, hidden_size=256, num_classes=10):
        super().__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, hidden_size)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu1(x)
        x = self.fc2(x)
        x = self.relu2(x)
        x = self.fc3(x)
        return x

class ModelOptimizer:
    """모델 최적화 도구"""

    def __init__(self):
        self.device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
        print(f"사용 디바이스: {self.device}")

    def create_dummy_data(self, num_samples=1000):
        """테스트용 더미 데이터 생성"""
        X = torch.randn(num_samples, 784)
        y = torch.randint(0, 10, (num_samples,))
        return DataLoader(TensorDataset(X, y), batch_size=32,
shuffle=True)

    def measure_performance(self, model, dataloader, name="Model"):
        """모델 성능 측정"""
        model.eval()

        # 크기 측정
        torch.save(model.state_dict(), 'temp_model.pth')
        model_size = os.path.getsize('temp_model.pth') / (1024 * 1024)  #
MB
        os.remove('temp_model.pth')

        # 속도 측정
        total_time = 0
        num_batches = 0
```

```
        with torch.no_grad():
            for X, _ in dataloader:
                X = X.to(self.device)

                start_time = time.time()
                _ = model(X)

                # GPU 동기화 (정확한 시간 측정)
                if self.device.type == 'cuda':
                    torch.cuda.synchronize()

                total_time += time.time() - start_time
                num_batches += 1

                if num_batches >= 10:  # 10배치만 테스트
                    break

        avg_inference_time = total_time / num_batches * 1000  # ms

        print(f"\n{name} 성능:")
        print(f"  - 모델 크기: {model_size:.2f} MB")
        print(f"  - 평균 추론 시간: {avg_inference_time:.2f} ms/batch")

        return model_size, avg_inference_time

    def quantize_dynamic(self, model):
        """동적 양자화 적용"""
        print("\n동적 양자화 적용 중...")

        quantized_model = torch.quantization.quantize_dynamic(
            model,
            {nn.Linear},  # Linear 레이어만 양자화
            dtype=torch.qint8
        )

        return quantized_model

    def quantize_static(self, model, dataloader):
        """정적 양자화 적용"""
        print("\n정적 양자화 적용 중...")

        # 모델 준비
        model.eval()
        model.qconfig = torch.quantization.get_default_qconfig('fbgemm')

        # 퓨전 가능한 모듈 결합
        model_fused = torch.quantization.fuse_modules(
            model,
            [['fc1', 'relu1'], ['fc2', 'relu2']]
        )

        # 양자화 준비
        model_prepared = torch.quantization.prepare(model_fused)
```

```
        # 캘리브레이션
        print("  캘리브레이션 실행 중...")
        with torch.no_grad():
            for i, (X, _) in enumerate(dataloader):
                model_prepared(X)
                if i >= 10:  # 10배치로 캘리브레이션
                    break

        # 양자화 변환
        model_quantized = torch.quantization.convert(model_prepared)

        return model_quantized

    def prune_model(self, model, sparsity=0.5):
        """가지치기 적용"""
        print(f"\n가지치기 적용 중... (희소성: {sparsity*100}%)")

        import torch.nn.utils.prune as prune

        # 각 Linear 레이어에 가지치기 적용
        for name, module in model.named_modules():
            if isinstance(module, nn.Linear):
                prune.l1_unstructured(module, name='weight',
amount=sparsity)

        # 가지치기 영구 적용
        for name, module in model.named_modules():
            if isinstance(module, nn.Linear):
                prune.remove(module, 'weight')

        return model

    def knowledge_distillation(self, teacher_model, student_model,
dataloader, epochs=5):
        """지식 증류"""
        print("\n지식 증류 수행 중...")

        # 학생 모델 (더 작은 모델)
        small_model = OptimizableModel(hidden_size=64)  # 더 작은 히든 크기
        small_model = small_model.to(self.device)

        optimizer = torch.optim.Adam(small_model.parameters(), lr=0.001)

        # 증류 손실 함수
        def distillation_loss(student_logits, teacher_logits,
temperature=3.0):
            soft_targets = nn.functional.softmax(teacher_logits /
temperature, dim=1)
            soft_predictions = nn.functional.log_softmax(student_logits /
temperature, dim=1)
            return nn.functional.kl_div(soft_predictions, soft_targets,
reduction='batchmean') * temperature * temperature

        teacher_model.eval()
```

```
        for epoch in range(epochs):
            total_loss = 0
            for X, y in dataloader:
                X = X.to(self.device)

                # 교사 모델 예측
                with torch.no_grad():
                    teacher_logits = teacher_model(X)

                # 학생 모델 예측
                student_logits = small_model(X)

                # 손실 계산
                loss = distillation_loss(student_logits, teacher_logits)

                optimizer.zero_grad()
                loss.backward()
                optimizer.step()

                total_loss += loss.item()

            print(f"  Epoch {epoch+1}/{epochs}, Loss:
{total_loss/len(dataloader):.4f}")

        return small_model

    def compile_model(self, model):
        """모델 컴파일 (PyTorch 2.0+)"""
        if hasattr(torch, 'compile'):
            print("\n모델 컴파일 중...")
            compiled_model = torch.compile(model, mode="reduce-overhead")
            return compiled_model
        else:
            print("\nPyTorch 2.0 이상이 필요합니다.")
            return model

    def compare_all_optimizations(self):
        """모든 최적화 기법 비교"""
        print("\n=== 모델 최적화 종합 비교 ===\n")

        # 데이터 준비
        dataloader = self.create_dummy_data()

        # 원본 모델
        original_model = OptimizableModel().to(self.device)
        original_size, original_time = self.measure_performance(
            original_model, dataloader, "원본 모델"
        )

        results = {
            'Original': {'size': original_size, 'time': original_time}
        }
```

```
        # 1. 동적 양자화
        if self.device.type == 'cpu':
            dynamic_quantized =
self.quantize_dynamic(original_model.cpu())
            size, time = self.measure_performance(
                dynamic_quantized, dataloader, "동적 양자화"
            )
            results['Dynamic Quantization'] = {'size': size, 'time': time}

        # 2. 가지치기
        pruned_model = self.prune_model(original_model.clone(),
sparsity=0.5)
        size, time = self.measure_performance(
            pruned_model, dataloader, "가지치기 (50%)"
        )
        results['Pruning'] = {'size': size, 'time': time}

        # 3. 지식 증류
        student_model = self.knowledge_distillation(
            original_model, None, dataloader
        )
        size, time = self.measure_performance(
            student_model, dataloader, "지식 증류 (작은 모델)"
        )
        results['Knowledge Distillation'] = {'size': size, 'time': time}

        # 결과 시각화
        self.visualize_comparison(results)

        return results

    def visualize_comparison(self, results):
        """최적화 결과 비교 시각화"""

        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

        methods = list(results.keys())
        sizes = [results[m]['size'] for m in methods]
        times = [results[m]['time'] for m in methods]

        # 모델 크기 비교
        bars1 = ax1.bar(methods, sizes, color=['red', 'green', 'blue',
'orange'][:len(methods)])
        ax1.set_ylabel('모델 크기 (MB)')
        ax1.set_title('모델 크기 비교', fontweight='bold')
        ax1.tick_labels = ax1.set_xticklabels(methods, rotation=45,
ha='right')

        # 크기 감소율 표시
        for i, bar in enumerate(bars1):
            if i > 0:
                reduction = (sizes[0] - sizes[i]) / sizes[0] * 100
                ax1.text(bar.get_x() + bar.get_width()/2, bar.get_height()
+ 0.5,
```

```
                        f'-{reduction:.0f}%', ha='center',
fontweight='bold')

        # 추론 시간 비교
        bars2 = ax2.bar(methods, times, color=['red', 'green', 'blue',
'orange'][:len(methods)])
        ax2.set_ylabel('추론 시간 (ms/batch)')
        ax2.set_title('추론 속도 비교', fontweight='bold')
        ax2.set_xticklabels(methods, rotation=45, ha='right')

        # 속도 향상 표시
        for i, bar in enumerate(bars2):
            if i > 0:
                speedup = times[0] / times[i]
                ax2.text(bar.get_x() + bar.get_width()/2, bar.get_height()
+ 0.2,
                        f'{speedup:.1f}x', ha='center', fontweight='bold')

        plt.tight_layout()
        plt.show()

# 최적화 실행
optimizer = ModelOptimizer()
results = optimizer.compare_all_optimizations()
```

8.5 하드웨어별 최적화 전략

```
class HardwareOptimizer:
    """하드웨어별 최적화 전략"""

    def __init__(self):
        self.device_info = self.detect_hardware()

    def detect_hardware(self):
        """하드웨어 감지"""
        info = {
            'device_type': 'cpu',
            'device_name': 'Unknown',
            'compute_capability': None,
            'memory': None
        }

        if torch.cuda.is_available():
            info['device_type'] = 'cuda'
            info['device_name'] = torch.cuda.get_device_name(0)
            info['compute_capability'] =
torch.cuda.get_device_capability(0)
            info['memory'] =
torch.cuda.get_device_properties(0).total_memory / (1024**3)

        elif hasattr(torch.backends, 'mps') and
```

```
torch.backends.mps.is_available():
            info['device_type'] = 'mps'
            info['device_name'] = 'Apple Silicon'

        print(f"감지된 하드웨어: {info}")
        return info

    def get_optimization_strategy(self):
        """하드웨어에 맞는 최적화 전략 제공"""

        strategies = []

        if self.device_info['device_type'] == 'cuda':
            # NVIDIA GPU 전략
            strategies.extend([
                "✅ TensorRT 사용 가능 - 최대 10배 속도 향상",
                "✅ Mixed Precision (FP16) 사용 권장",
                "✅ CUDA Graphs로 커널 실행 최적화",
                "✅ Flash Attention 사용 가능",
                "✅ Tensor Cores 활용 (Volta 이상)"
            ])

            if self.device_info['compute_capability'][0] >= 8:
                strategies.append("✅ INT8 Tensor Cores 지원 (Ampere 이상)")

        elif self.device_info['device_type'] == 'mps':
            # Apple Silicon 전략
            strategies.extend([
                "✅ Core ML 변환으로 최적화",
                "✅ Metal Performance Shaders 활용",
                "✅ Apple Neural Engine 사용 가능",
                "✅ 통합 메모리로 데이터 전송 최소화"
            ])

        else:
            # CPU 전략
            strategies.extend([
                "✅ OpenVINO 또는 ONNX Runtime 사용",
                "✅ INT8 양자화로 속도 향상",
                "✅ 멀티스레딩 활용",
                "✅ SIMD 명령어 최적화",
                "✅ 메모리 정렬 최적화"
            ])

        return strategies

    def optimize_for_hardware(self, model):
        """하드웨어에 맞게 모델 최적화"""

        optimized_model = model

        if self.device_info['device_type'] == 'cuda':
            # CUDA 최적화
            print("\nCUDA 최적화 적용 중...")
```

```
            # Mixed Precision
            from torch.cuda.amp import autocast

            # 모델을 FP16으로 변환
            optimized_model = model.half()

            # TensorRT 변환 (가능한 경우)
            try:
                import torch_tensorrt
                optimized_model = torch_tensorrt.compile(
                    model,
                    inputs=[torch.randn(1, 784).cuda()],
                    enabled_precisions={torch.float, torch.half}
                )
                print("  ✅ TensorRT 최적화 완료")
            except:
                print("  ⚠ TensorRT를 사용할 수 없습니다")

        elif self.device_info['device_type'] == 'mps':
            # Apple Silicon 최적화
            print("\nApple Silicon 최적화 적용 중...")

            # Core ML 변환
            try:
                import coremltools as ct

                example_input = torch.randn(1, 784)
                traced_model = torch.jit.trace(model, example_input)

                coreml_model = ct.convert(
                    traced_model,
                    inputs=[ct.TensorType(shape=example_input.shape)]
                )

                print("  ✅ Core ML 변환 완료")
                # 실제로는 coreml_model을 사용

            except:
                print("  ⚠ Core ML Tools를 사용할 수 없습니다")

        else:
            # CPU 최적화
            print("\nCPU 최적화 적용 중...")

            # ONNX 변환
            try:
                import torch.onnx

                dummy_input = torch.randn(1, 784)
                torch.onnx.export(
                    model,
                    dummy_input,
                    "optimized_model.onnx",
```

```
                    export_params=True,
                    opset_version=11,
                    do_constant_folding=True,
                    input_names=['input'],
                    output_names=['output']
                )

                print("  ✅ ONNX 변환 완료")

                # ONNX Runtime으로 실행
                import onnxruntime as ort

                ort_session = ort.InferenceSession("optimized_model.onnx")
                print("  ✅ ONNX Runtime 세션 생성 완료")

            except Exception as e:
                print(f"  ⚠ ONNX 최적화 실패: {e}")

        return optimized_model

# 하드웨어 최적화 실행
hw_optimizer = HardwareOptimizer()
strategies = hw_optimizer.get_optimization_strategy()

print("\n=== 권장 최적화 전략 ===")
for strategy in strategies:
    print(strategy)
```

9장: 음성 인터페이스 구축 - AI와 대화하기

9.1 음성 인터페이스의 구성 요소

음성 AI 시스템은 크게 세 부분으로 구성됩니다:

1. **STT (Speech-to-Text)**: 음성을 텍스트로 변환
2. **NLU/처리**: 의도 파악 및 응답 생성
3. **TTS (Text-to-Speech)**: 텍스트를 음성으로 변환

9.2 Whisper로 음성 인식 구현

```
# advanced_speech_recognition.py - 고급 음성 인식

import whisper
import numpy as np
import sounddevice as sd
import queue
import threading
import webrtcvad
import collections
import time
```

```
from typing import Optional, Callable
import matplotlib.pyplot as plt

class AdvancedWhisperSTT:
    """고급 음성 인식 시스템"""

    def __init__(self, model_size="base", language="ko"):
        print(f"🎤 Whisper {model_size} 모델을 로드하고 있습니다...")

        self.model = whisper.load_model(model_size)
        self.language = language

        # VAD (Voice Activity Detection)
        self.vad = webrtcvad.Vad(2)  # 민감도 0-3

        # 오디오 설정
        self.sample_rate = 16000
        self.frame_duration = 30  # ms
        self.frame_size = int(self.sample_rate * self.frame_duration /
1000)

        # 버퍼
        self.audio_queue = queue.Queue()
        self.is_recording = False

        print("✅ 음성 인식 시스템 준비 완료!")

    def visualize_audio_processing(self):
        """오디오 처리 과정 시각화"""

        fig, axes = plt.subplots(3, 2, figsize=(12, 10))

        # 1. 원본 오디오 신호
        ax1 = axes[0, 0]
        t = np.linspace(0, 1, self.sample_rate)
        # 음성 신호 시뮬레이션 (여러 주파수 혼합)
        signal = (0.5 * np.sin(2 * np.pi * 200 * t) +
                  0.3 * np.sin(2 * np.pi * 400 * t) +
                  0.2 * np.sin(2 * np.pi * 800 * t))
        noise = np.random.normal(0, 0.1, len(t))
        audio = signal + noise

        ax1.plot(t[:1000], audio[:1000], alpha=0.7)
        ax1.set_title('원본 오디오 신호', fontweight='bold')
        ax1.set_xlabel('시간 (초)')
        ax1.set_ylabel('진폭')
        ax1.grid(True, alpha=0.3)

        # 2. 주파수 스펙트럼
        ax2 = axes[0, 1]
        freqs = np.fft.fftfreq(len(audio), 1/self.sample_rate)
        fft = np.abs(np.fft.fft(audio))

        ax2.plot(freqs[:len(freqs)//2], fft[:len(fft)//2])
```

```
        ax2.set_title('주파수 스펙트럼', fontweight='bold')
        ax2.set_xlabel('주파수 (Hz)')
        ax2.set_ylabel('강도')
        ax2.set_xlim(0, 2000)
        ax2.grid(True, alpha=0.3)

        # 3. VAD (음성 활동 감지)
        ax3 = axes[1, 0]
        # VAD 시뮬레이션
        vad_result = np.abs(audio) > 0.3
        vad_smoothed = np.convolve(vad_result, np.ones(100)/100,
mode='same')

        ax3.plot(t[:5000], audio[:5000], alpha=0.5, label='오디오')
        ax3.fill_between(t[:5000], -1, 1, where=vad_smoothed[:5000] > 0.5,
                         alpha=0.3, color='red', label='음성 감지')
        ax3.set_title('음성 활동 감지 (VAD)', fontweight='bold')
        ax3.set_xlabel('시간 (초)')
        ax3.set_ylabel('진폭')
        ax3.legend()
        ax3.grid(True, alpha=0.3)

        # 4. 멜 스펙트로그램
        ax4 = axes[1, 1]
        # 간단한 스펙트로그램 시뮬레이션
        spec_data = np.random.rand(128, 100) * np.linspace(1, 0.1,
128).reshape(-1, 1)
        im = ax4.imshow(spec_data, aspect='auto', origin='lower',
cmap='viridis')
        ax4.set_title('멜 스펙트로그램', fontweight='bold')
        ax4.set_xlabel('시간 프레임')
        ax4.set_ylabel('멜 빈')
        plt.colorbar(im, ax=ax4)

        # 5. 음성 인식 과정
        ax5 = axes[2, 0]
        ax5.text(0.5, 0.8, '음성 입력', ha='center', fontsize=12,
                bbox=dict(boxstyle="round,pad=0.3",
facecolor="lightblue"))
        ax5.text(0.5, 0.6, '↓', ha='center', fontsize=16)
        ax5.text(0.5, 0.4, '특징 추출', ha='center', fontsize=12,
                bbox=dict(boxstyle="round,pad=0.3",
facecolor="lightgreen"))
        ax5.text(0.5, 0.2, '↓', ha='center', fontsize=16)
        ax5.text(0.5, 0.0, '텍스트 변환', ha='center', fontsize=12,
                bbox=dict(boxstyle="round,pad=0.3",
facecolor="lightyellow"))
        ax5.set_xlim(0, 1)
        ax5.set_ylim(-0.2, 1)
        ax5.axis('off')
        ax5.set_title('Whisper 처리 과정', fontweight='bold')

        # 6. 언어 모델 확률
        ax6 = axes[2, 1]
```

```
        words = ['안녕하세요', '안녕하십니까', '안녕히', '안녕', '안년하세요']
        probs = [0.4, 0.3, 0.15, 0.1, 0.05]

        bars = ax6.bar(words, probs, color='orange')
        ax6.set_title('언어 모델 예측 확률', fontweight='bold')
        ax6.set_ylabel('확률')
        ax6.set_xticklabels(words, rotation=45, ha='right')

        for bar, prob in zip(bars, probs):
            ax6.text(bar.get_x() + bar.get_width()/2, bar.get_height() +
0.01,
                     f'{prob:.2f}', ha='center')

        plt.tight_layout()
        plt.show()

    def voice_activity_detection(self, audio_frame):
        """음성 활동 감지"""
        # 16비트 PCM으로 변환
        audio_int16 = (audio_frame * 32767).astype(np.int16)

        # VAD 적용
        return self.vad.is_speech(audio_int16.tobytes(), self.sample_rate)

    def continuous_recognition(self, callback: Callable[[str], None]):
        """연속 음성 인식"""

        def audio_callback(indata, frames, time_info, status):
            """오디오 콜백"""
            if status:
                print(f"오디오 상태: {status}")

            self.audio_queue.put(indata.copy())

        # 스트림 시작
        stream = sd.InputStream(
            samplerate=self.sample_rate,
            channels=1,
            callback=audio_callback,
            blocksize=self.frame_size
        )

        with stream:
            print("\n🎤 음성 인식을 시작합니다...")
            print("말씀하세요! (종료: Ctrl+C)")

            # 링 버퍼 (3초 분량)
            ring_buffer = collections.deque(maxlen=100)

            # 음성 감지 상태
            triggered = False
            voiced_frames = []

            while self.is_recording:
```

```
                try:
                    frame = self.audio_queue.get(timeout=0.1)

                    # VAD 확인
                    is_speech = self.voice_activity_detection(frame)

                    if not triggered:
                        ring_buffer.append((frame, is_speech))
                        num_voiced = len([f for f, speech in ring_buffer
if speech])

                        # 음성 시작 감지 (0.3초 이상)
                        if num_voiced > 0.3 * 100:
                            triggered = True
                            print("🔊 음성 감지됨...")

                            # 링 버퍼의 내용을 voiced_frames에 추가
                            for f, s in ring_buffer:
                                voiced_frames.append(f)
                            ring_buffer.clear()
                    else:
                        # 음성 수집
                        voiced_frames.append(frame)
                        ring_buffer.append((frame, is_speech))

                        # 음성 종료 감지 (1초 이상 조용)
                        num_unvoiced = len([f for f, speech in ring_buffer
if not speech])
                        if num_unvoiced > 1.0 * 100:
                            print("🔇 음성 종료, 인식 중...")

                            # 음성 인식 수행
                            audio_data = np.concatenate(voiced_frames)
                            text = self.recognize(audio_data)

                            if text and callback:
                                callback(text)

                            # 초기화
                            triggered = False
                            voiced_frames = []
                            ring_buffer.clear()

                except queue.Empty:
                    continue
                except KeyboardInterrupt:
                    break

    def recognize(self, audio_data: np.ndarray) -> str:
        """음성을 텍스트로 변환"""
        # Whisper는 float32 필요
        audio_float32 = audio_data.astype(np.float32)

        # 패딩 (최소 길이 확보)
```

```
            if len(audio_float32) < self.sample_rate:
                audio_float32 = np.pad(audio_float32, (0, self.sample_rate -
len(audio_float32)))

            # 음성 인식
            result = self.model.transcribe(
                audio_float32,
                language=self.language,
                fp16=torch.cuda.is_available()
            )

            return result["text"].strip()

        def start_recording(self):
            """녹음 시작"""
            self.is_recording = True

            def on_recognition(text):
                print(f"\n📝 인식된 텍스트: {text}")

            # 별도 스레드에서 실행
            self.recognition_thread = threading.Thread(
                target=self.continuous_recognition,
                args=(on_recognition,)
            )
            self.recognition_thread.start()

        def stop_recording(self):
            """녹음 중지"""
            self.is_recording = False
            if hasattr(self, 'recognition_thread'):
                self.recognition_thread.join()

    # 음성 인식 시스템 테스트
    def test_advanced_stt():
        stt = AdvancedWhisperSTT(model_size="base", language="ko")

        # 오디오 처리 과정 시각화
        stt.visualize_audio_processing()

        # 실시간 음성 인식 테스트
        print("\n=== 실시간 음성 인식 테스트 ===")

        try:
            stt.start_recording()
            time.sleep(30)  # 30초 동안 녹음
        except KeyboardInterrupt:
            print("\n인식 중단...")
        finally:
            stt.stop_recording()

    # 실행
    # test_advanced_stt()
```

9.3 고급 TTS 시스템 구현

```
# advanced_tts.py - 고급 음성 합성

import torch
import numpy as np
from TTS.api import TTS
import pygame
import io
from typing import Dict, Optional
import json

class AdvancedTTS:
    """고급 음성 합성 시스템"""

    def __init__(self):
        print("🔊 고급 TTS 시스템을 초기화하고 있습니다...")

        # 사용 가능한 모델들
        self.available_models = {
            'korean_single': 'tts_models/ko/cv/vits',
            'multilingual': 'tts_models/multilingual/multi-
dataset/your_tts',
            'emotional': 'tts_models/en/vctk/vits'  # 감정 표현 가능
        }

        # 기본 모델 로드
        self.load_model('korean_single')

        # 음성 스타일 설정
        self.voice_styles = {
            'normal': {'speed': 1.0, 'pitch': 1.0, 'energy': 1.0},
            'happy': {'speed': 1.1, 'pitch': 1.1, 'energy': 1.2},
            'sad': {'speed': 0.9, 'pitch': 0.9, 'energy': 0.8},
            'angry': {'speed': 1.2, 'pitch': 0.8, 'energy': 1.3},
            'calm': {'speed': 0.85, 'pitch': 1.0, 'energy': 0.9}
        }

        pygame.mixer.init()
        print("✅ TTS 시스템 준비 완료!")

    def load_model(self, model_type: str):
        """TTS 모델 로드"""
        try:
            model_name = self.available_models[model_type]
            self.tts = TTS(model_name=model_name, progress_bar=True)

            if torch.cuda.is_available():
                self.tts = self.tts.cuda()

            self.current_model = model_type
```

```
            print(f"✅ {model_type} 모델 로드 완료")

        except Exception as e:
            print(f"❌ 모델 로드 실패: {e}")
            # 폴백: gTTS 사용
            self.use_gtts_fallback = True

    def synthesize_with_style(self, text: str, style: str = 'normal',
                              speaker_id: Optional[int] = None) ->
np.ndarray:
        """스타일이 적용된 음성 합성"""

        style_params = self.voice_styles.get(style,
self.voice_styles['normal'])

        try:
            # TTS 합성
            wav = self.tts.tts(
                text=text,
                speaker=speaker_id,
                speed=style_params['speed']
            )

            # 추가 오디오 처리 (피치, 에너지 조정)
            wav = self.apply_audio_effects(wav, style_params)

            return np.array(wav)

        except Exception as e:
            print(f"TTS 오류: {e}")
            # 폴백 처리
            return self.fallback_tts(text)

    def apply_audio_effects(self, audio: np.ndarray, params: Dict) ->
np.ndarray:
        """오디오 효과 적용"""
        # 피치 시프트 (간단한 구현)
        if params['pitch'] != 1.0:
            # 리샘플링을 통한 피치 변경
            new_length = int(len(audio) / params['pitch'])
            indices = np.linspace(0, len(audio) - 1, new_length)
            audio = np.interp(indices, np.arange(len(audio)), audio)

        # 에너지 조정 (볼륨)
        audio = audio * params['energy']

        # 클리핑 방지
        audio = np.clip(audio, -1.0, 1.0)

        return audio

    def create_ssml(self, text: str, style: str = 'normal') -> str:
        """SSML (Speech Synthesis Markup Language) 생성"""
```

```
        ssml_template = """
        <speak>
            <prosody rate="{rate}" pitch="{pitch}" volume="{volume}">
                {text}
            </prosody>
        </speak>
        """

        style_params = self.voice_styles[style]

        return ssml_template.format(
            rate=f"{style_params['speed']*100}%",
            pitch=f"{(style_params['pitch']-1)*50:+.0f}%",
            volume=f"{style_params['energy']*100}%",
            text=text
        )

    def speak_with_emotion(self, text: str, emotion: str = 'normal'):
        """감정이 담긴 음성 출력"""

        print(f"🎭 {emotion} 감정으로 말하기: {text}")

        # 음성 합성
        audio = self.synthesize_with_style(text, emotion)

        # 재생
        self.play_audio(audio)

    def play_audio(self, audio: np.ndarray, sample_rate: int = 22050):
        """오디오 재생"""
        # int16으로 변환
        audio_int16 = (audio * 32767).astype(np.int16)

        # pygame용 사운드 객체 생성
        sound = pygame.sndarray.make_sound(audio_int16)

        # 재생
        sound.play()

        # 재생 완료 대기
        while pygame.mixer.get_busy():
            pygame.time.Clock().tick(10)

    def create_audio_book(self, chapters: List[Dict[str, str]],
output_file: str):
        """오디오북 생성"""

        print(f"📚 오디오북 생성 중: {output_file}")

        all_audio = []

        for i, chapter in enumerate(chapters):
            title = chapter.get('title', f'Chapter {i+1}')
            content = chapter.get('content', '')
```

```
            style = chapter.get('style', 'normal')

            print(f"  📖 {title} 처리 중...")

            # 제목 음성
            title_audio = self.synthesize_with_style(
                f"제 {i+1}장. {title}",
                'normal'
            )

            # 내용 음성
            content_audio = self.synthesize_with_style(content, style)

            # 챕터 사이 휴지
            pause = np.zeros(int(22050 * 2))  # 2초 휴지

            all_audio.extend([title_audio, pause, content_audio, pause])

        # 전체 오디오 결합
        complete_audio = np.concatenate(all_audio)

        # 파일로 저장
        import soundfile as sf
        sf.write(output_file, complete_audio, 22050)

        print(f"✅ 오디오북 생성 완료: {output_file}")

    def voice_cloning(self, reference_audio: str, text: str):
        """음성 복제 (고급 기능)"""

        if self.current_model != 'multilingual':
            print("음성 복제를 위해 multilingual 모델로 전환합니다...")
            self.load_model('multilingual')

        try:
            # 참조 음성으로 TTS
            wav = self.tts.tts(
                text=text,
                speaker_wav=reference_audio,
                language="ko"
            )

            return np.array(wav)

        except Exception as e:
            print(f"음성 복제 실패: {e}")
            return self.fallback_tts(text)

    def fallback_tts(self, text: str) -> np.ndarray:
        """폴백 TTS (gTTS 사용)"""
        from gtts import gTTS
        import tempfile

        tts = gTTS(text=text, lang='ko')
```

```
        with tempfile.NamedTemporaryFile(suffix='.mp3', delete=False) as
tmp_file:
            tts.save(tmp_file.name)

            # MP3를 numpy 배열로 변환
            import librosa
            audio, sr = librosa.load(tmp_file.name, sr=22050)

            os.unlink(tmp_file.name)

        return audio

# TTS 데모
def demonstrate_advanced_tts():
    """고급 TTS 기능 시연"""

    tts = AdvancedTTS()

    # 1. 다양한 감정 표현
    print("\n=== 감정 표현 데모 ===")

    test_text = "안녕하세요. 오늘 날씨가 정말 좋네요."

    emotions = ['normal', 'happy', 'sad', 'angry', 'calm']

    for emotion in emotions:
        print(f"\n{emotion} 스타일:")
        tts.speak_with_emotion(test_text, emotion)
        time.sleep(2)

    # 2. 오디오북 생성
    print("\n=== 오디오북 생성 데모 ===")

    chapters = [
        {
            'title': 'AI의 시작',
            'content': '인공지능은 인간의 지능을 모방하는 기술입니다.',
            'style': 'normal'
        },
        {
            'title': '미래의 전망',
            'content': 'AI는 우리의 미래를 바꿀 것입니다.',
            'style': 'calm'
        }
    ]

    tts.create_audio_book(chapters, 'ai_audiobook.wav')

    # 3. SSML 테스트
    print("\n=== SSML 테스트 ===")

    ssml = tts.create_ssml("이것은 SSML 테스트입니다.", 'happy')
    print(f"생성된 SSML:\n{ssml}")
```

```
# 실행
# demonstrate_advanced_tts()
```

10장: 최종 프로젝트 - 음성 대화형 AI 비서

10.1 프로젝트 개요: JARVIS를 만들어보자!

우리의 최종 프로젝트는 아이언맨의 JARVIS같은 AI 비서를 만드는 것입니다:

- 🎤 자연스러운 음성 대화
- 👀 카메라로 상황 인식
- 🧠 지능적인 응답 생성
- 🏠 스마트홈 제어 (시뮬레이션)
- 📊 정보 검색 및 분석

10.2 전체 시스템 아키텍처

```
# jarvis_system.py - JARVIS 스타일 AI 비서

import asyncio
import torch
import numpy as np
from typing import Dict, List, Optional, Any
import json
import time
from datetime import datetime
import cv2
from collections import deque
import threading
import queue

class JARVIS:
    """Just A Rather Very Intelligent System"""

    def __init__(self, user_name: str = "User"):
        print("""
        ╔═══════════════════════════════════════════╗
        ║         J.A.R.V.I.S 시스템 시작           ║
        ║                                           ║
        ║    Just A Rather Very Intelligent System  ║
        ╚═══════════════════════════════════════════╝
        """)

        self.user_name = user_name
        self.initialize_components()
        self.boot_sequence()

    def initialize_components(self):
        """시스템 컴포넌트 초기화"""
```

```
        print("\n시스템 초기화 중...")

        # 1. 음성 시스템
        print("  [1/6] 음성 인식 시스템 초기화...")
        self.stt = AdvancedWhisperSTT(model_size="base", language="ko")

        print("  [2/6] 음성 합성 시스템 초기화...")
        self.tts = AdvancedTTS()

        # 2. 비전 시스템
        print("  [3/6] 컴퓨터 비전 시스템 초기화...")
        self.vision_system = VisionSystem()

        # 3. 자연어 처리
        print("  [4/6] 자연어 처리 엔진 초기화...")
        self.nlp_engine = NLPEngine()

        # 4. 지식 베이스
        print("  [5/6] 지식 베이스 로딩...")
        self.knowledge_base = KnowledgeBase()

        # 5. 작업 실행기
        print("  [6/6] 작업 실행 엔진 초기화...")
        self.task_executor = TaskExecutor()

        # 시스템 상태
        self.system_state = {
            'mode': 'standby',
            'last_interaction': None,
            'context': deque(maxlen=10),
            'active_tasks': []
        }

        print("\n✅ 모든 시스템 준비 완료!")

    def boot_sequence(self):
        """부팅 시퀀스"""
        self.speak(f"안녕하세요 {self.user_name}님. JARVIS 시스템이 준비되었습니
다.")
        self.speak("무엇을 도와드릴까요?")
        self.system_state['mode'] = 'active'

    def speak(self, text: str, style: str = 'normal'):
        """JARVIS 음성 출력"""
        print(f"🤖 JARVIS: {text}")
        self.tts.speak_with_emotion(text, style)

    async def listen(self) -> Optional[str]:
        """사용자 음성 입력 대기"""
        print("\n👂 듣고 있습니다...")

        # 음성 인식 (비동기)
        loop = asyncio.get_event_loop()
        text = await loop.run_in_executor(None, self._listen_sync)
```

```
        if text:
            print(f"👤 {self.user_name}: {text}")
            self.system_state['last_interaction'] = time.time()
            self.system_state['context'].append({
                'speaker': 'user',
                'text': text,
                'time': datetime.now()
            })

        return text

    def _listen_sync(self) -> Optional[str]:
        """동기식 음성 인식"""
        audio = self.stt.record_audio(duration=5)
        return self.stt.transcribe(audio)

    async def process_command(self, command: str) -> Dict[str, Any]:
        """명령 처리"""

        # 의도 분석
        intent = await self.nlp_engine.analyze_intent(command)

        # 컨텍스트 업데이트
        self.system_state['context'].append({
            'speaker': 'jarvis',
            'intent': intent,
            'time': datetime.now()
        })

        # 의도에 따른 처리
        response = await self.execute_intent(intent)

        return response

    async def execute_intent(self, intent: Dict[str, Any]) -> Dict[str,
Any]:
        """의도 실행"""

        intent_type = intent.get('type', 'unknown')
        entities = intent.get('entities', {})

        if intent_type == 'greeting':
            return await self.handle_greeting()

        elif intent_type == 'weather':
            return await self.handle_weather(entities.get('location'))

        elif intent_type == 'vision_analysis':
            return await
self.handle_vision_analysis(entities.get('query'))

        elif intent_type == 'smart_home':
            return await self.handle_smart_home(entities)
```

```
        elif intent_type == 'information':
            return await
self.handle_information_query(entities.get('query'))

        elif intent_type == 'task':
            return await self.handle_task(entities)

        else:
            return {
                'success': False,
                'message': "죄송합니다. 이해하지 못했습니다. 다시 말씀해주세요."
            }

    async def handle_greeting(self) -> Dict[str, Any]:
        """인사 처리"""
        current_hour = datetime.now().hour

        if current_hour < 12:
            greeting = "좋은 아침입니다"
        elif current_hour < 18:
            greeting = "좋은 오후입니다"
        else:
            greeting = "좋은 저녁입니다"

        self.speak(f"{greeting}, {self.user_name}님. 오늘도 좋은 하루 되세요.")

        return {'success': True, 'action': 'greeting'}

    async def handle_vision_analysis(self, query: Optional[str]) ->
Dict[str, Any]:
        """비전 분석 처리"""
        self.speak("카메라를 통해 확인하겠습니다.")

        # 이미지 캡처
        image = self.vision_system.capture_image()

        # 분석
        analysis = await self.vision_system.analyze_scene(image, query)

        # 결과 설명
        self.speak(analysis['description'])

        return {
            'success': True,
            'action': 'vision_analysis',
            'result': analysis
        }

    async def handle_smart_home(self, entities: Dict) -> Dict[str, Any]:
        """스마트홈 제어"""
        device = entities.get('device')
        action = entities.get('action')
```

```
        if not device or not action:
            self.speak("어떤 기기를 어떻게 제어할까요?")
            return {'success': False}

        # 시뮬레이션
        result = await self.task_executor.control_device(device, action)

        if result['success']:
            self.speak(f"{device}를 {action}했습니다.")
        else:
            self.speak(f"죄송합니다. {device} 제어에 실패했습니다.")

        return result

    async def run(self):
        """메인 실행 루프"""
        self.speak("시스템이 활성화되었습니다.")

        while self.system_state['mode'] == 'active':
            try:
                # 음성 입력 대기
                command = await self.listen()

                if not command:
                    continue

                # 종료 명령 확인
                if any(word in command for word in ['종료', '잘자', '굿바
이']):
                    await self.shutdown()
                    break

                # 명령 처리
                response = await self.process_command(command)

                # 주기적 상태 체크
                await self.periodic_check()

            except KeyboardInterrupt:
                await self.shutdown()
                break
            except Exception as e:
                print(f"오류 발생: {e}")
                self.speak("죄송합니다. 오류가 발생했습니다.")

    async def periodic_check(self):
        """주기적 상태 확인"""
        # 장시간 미사용 시 절전 모드
        if self.system_state['last_interaction']:
            idle_time = time.time() -
self.system_state['last_interaction']
            if idle_time > 300:  # 5분
                self.system_state['mode'] = 'standby'
                self.speak("절전 모드로 전환합니다.")
```

```
    async def shutdown(self):
        """시스템 종료"""
        self.speak(f"안녕히 계세요, {self.user_name}님.")
        self.system_state['mode'] = 'shutdown'

        # 리소스 정리
        cv2.destroyAllWindows()


class VisionSystem:
    """JARVIS 비전 시스템"""

    def __init__(self):
        self.yolo = YOLO('yolo11m.pt')
        self.vqa_model = self.load_vqa_model()
        self.cap = None

    def load_vqa_model(self):
        """VQA 모델 로드"""
        from transformers import Blip2Processor,
Blip2ForConditionalGeneration

        processor = Blip2Processor.from_pretrained("Salesforce/blip2-opt-
2.7b")
        model = Blip2ForConditionalGeneration.from_pretrained(
            "Salesforce/blip2-opt-2.7b",
            torch_dtype=torch.float16 if torch.cuda.is_available() else
torch.float32
        )

        return {'processor': processor, 'model': model}

    def capture_image(self) -> np.ndarray:
        """카메라에서 이미지 캡처"""
        if self.cap is None:
            self.cap = cv2.VideoCapture(0)

        ret, frame = self.cap.read()
        return frame if ret else np.zeros((480, 640, 3), dtype=np.uint8)

    async def analyze_scene(self, image: np.ndarray, query: Optional[str]
= None) -> Dict:
        """장면 분석"""
        results = {
            'objects': [],
            'description': "",
            'query_answer': ""
        }

        # YOLO 객체 탐지
        yolo_results = self.yolo(image)[0]

        for box in yolo_results.boxes:
```

```
                results['objects'].append({
                    'class': self.yolo.names[int(box.cls)],
                    'confidence': float(box.conf),
                    'bbox': box.xyxy[0].tolist()
                })

            # 장면 설명 생성
            if results['objects']:
                object_counts = {}
                for obj in results['objects']:
                    cls = obj['class']
                    object_counts[cls] = object_counts.get(cls, 0) + 1

                description_parts = []
                for cls, count in object_counts.items():
                    if count == 1:
                        description_parts.append(f"{cls} 1개")
                    else:
                        description_parts.append(f"{cls} {count}개")

                results['description'] = f"화면에 {', '.join(description_parts)}
가 보입니다."
            else:
                results['description'] = "특별한 객체가 감지되지 않았습니다."

            # 특정 질문에 대한 답변
            if query and self.vqa_model:
                pil_image = Image.fromarray(cv2.cvtColor(image,
cv2.COLOR_BGR2RGB))

                inputs = self.vqa_model['processor'](
                    images=pil_image,
                    text=query,
                    return_tensors="pt"
                )

                with torch.no_grad():
                    generated_ids = self.vqa_model['model'].generate(**inputs)
                    answer = self.vqa_model['processor'].decode(
                        generated_ids[0],
                        skip_special_tokens=True
                    )

                results['query_answer'] = answer

            return results


    class NLPEngine:
        """자연어 처리 엔진"""

        def __init__(self):
            self.intent_patterns = {
                'greeting': ['안녕', '하이', 'hello', '반가워'],
```

```
            'weather': ['날씨', '기온', '비', '맑음'],
            'vision_analysis': ['보여줘', '뭐가 있', '찾아줘', '어디'],
            'smart_home': ['켜줘', '꺼줘', '조절', '온도', '불빛'],
            'information': ['알려줘', '뭐야', '검색', '찾아'],
            'task': ['일정', '알람', '메모', '리마인더']
        }

    async def analyze_intent(self, text: str) -> Dict[str, Any]:
        """의도 분석"""
        text_lower = text.lower()

        # 패턴 매칭
        for intent, patterns in self.intent_patterns.items():
            if any(pattern in text_lower for pattern in patterns):
                return {
                    'type': intent,
                    'confidence': 0.8,
                    'entities': self.extract_entities(text, intent)
                }

        return {
            'type': 'unknown',
            'confidence': 0.0,
            'entities': {}
        }

    def extract_entities(self, text: str, intent: str) -> Dict:
        """엔티티 추출"""
        entities = {}

        if intent == 'smart_home':
            # 디바이스 추출
            devices = ['전등', '에어컨', '히터', 'TV', '커튼']
            for device in devices:
                if device in text:
                    entities['device'] = device

            # 액션 추출
            if '켜' in text:
                entities['action'] = '켜기'
            elif '꺼' in text:
                entities['action'] = '끄기'

        elif intent == 'weather':
            # 위치 추출 (간단한 예시)
            if '내일' in text:
                entities['time'] = 'tomorrow'
            else:
                entities['time'] = 'today'

        return entities


class KnowledgeBase:
```

```
    """지식 베이스"""

    def __init__(self):
        self.facts = {
            'user_preferences': {},
            'device_states': {
                '전등': False,
                '에어컨': False,
                'TV': False
            },
            'schedules': [],
            'reminders': []
        }

    def get_fact(self, category: str, key: str = None):
        """사실 조회"""
        if key:
            return self.facts.get(category, {}).get(key)
        return self.facts.get(category)

    def update_fact(self, category: str, key: str, value: Any):
        """사실 업데이트"""
        if category not in self.facts:
            self.facts[category] = {}

        if isinstance(self.facts[category], dict):
            self.facts[category][key] = value
        elif isinstance(self.facts[category], list):
            self.facts[category].append({key: value})


class TaskExecutor:
    """작업 실행기"""

    def __init__(self):
        self.active_tasks = []

    async def control_device(self, device: str, action: str) -> Dict:
        """디바이스 제어 (시뮬레이션)"""
        print(f"🏠 스마트홈: {device} {action}")

        # 실제로는 IoT API 호출
        await asyncio.sleep(0.5)  # 시뮬레이션 딜레이

        return {
            'success': True,
            'device': device,
            'action': action,
            'timestamp': datetime.now()
        }

    async def set_reminder(self, text: str, time: datetime) -> Dict:
        """리마인더 설정"""
        reminder = {
```

```
            'id': len(self.active_tasks),
            'text': text,
            'time': time,
            'active': True
        }

        self.active_tasks.append(reminder)

        return {
            'success': True,
            'reminder': reminder
        }


# JARVIS UI
def create_jarvis_ui():
    """JARVIS 웹 UI"""
    import gradio as gr

    jarvis = None

    def initialize_jarvis(user_name):
        """JARVIS 초기화"""
        global jarvis
        jarvis = JARVIS(user_name)
        return f"JARVIS가 {user_name}님을 위해 준비되었습니다."

    def process_voice_command(audio):
        """음성 명령 처리"""
        if jarvis is None:
            return "먼저 JARVIS를 초기화해주세요.", None

        # 음성을 텍스트로 변환
        text = jarvis.stt.transcribe(audio)

        # 명령 처리 (동기 래퍼)
        response = asyncio.run(jarvis.process_command(text))

        return text, response.get('message', '처리 완료')

    def capture_and_analyze():
        """카메라 캡처 및 분석"""
        if jarvis is None:
            return None, "먼저 JARVIS를 초기화해주세요."

        image = jarvis.vision_system.capture_image()
        analysis = asyncio.run(
            jarvis.vision_system.analyze_scene(image)
        )

        return image, analysis['description']

    # Gradio 인터페이스
    with gr.Blocks(title="J.A.R.V.I.S", theme=gr.themes.Soft()) as
```

```
interface:
        gr.Markdown("""
        # 🤖 J.A.R.V.I.S - AI 비서 시스템

        Just A Rather Very Intelligent System
        """)

        with gr.Tab("🚀 초기화"):
            user_name_input = gr.Textbox(
                label="사용자 이름",
                placeholder="당신의 이름을 입력하세요",
                value="User"
            )
            init_btn = gr.Button("JARVIS 시작", variant="primary")
            init_output = gr.Textbox(label="상태")

            init_btn.click(
                fn=initialize_jarvis,
                inputs=user_name_input,
                outputs=init_output
            )

        with gr.Tab("🎤 음성 명령"):
            with gr.Row():
                audio_input = gr.Audio(
                    source="microphone",
                    type="numpy",
                    label="음성 입력"
                )

                with gr.Column():
                    recognized_text = gr.Textbox(label="인식된 텍스트")
                    response_text = gr.Textbox(label="JARVIS 응답")

            process_btn = gr.Button("명령 실행")
            process_btn.click(
                fn=process_voice_command,
                inputs=audio_input,
                outputs=[recognized_text, response_text]
            )

        with gr.Tab("👀 비전 분석"):
            with gr.Row():
                captured_image = gr.Image(label="캡처된 이미지")
                analysis_result = gr.Textbox(label="분석 결과", lines=5)

            capture_btn = gr.Button("캡처 및 분석")
            capture_btn.click(
                fn=capture_and_analyze,
                outputs=[captured_image, analysis_result]
            )

        with gr.Tab("🏠 스마트홈"):
            gr.Markdown("""
```

```
            ### 디바이스 상태

            - 🔦 전등: OFF
            - ❄️ 에어컨: OFF
            - 📺 TV: OFF
            - 🌡️ 온도: 22°C
            - 💡 조도: 적정
            """)

            device_select = gr.Dropdown(
                choices=["전등", "에어컨", "TV"],
                label="디바이스 선택"
            )
            action_select = gr.Radio(
                choices=["켜기", "끄기"],
                label="동작"
            )
            control_btn = gr.Button("제어")

        with gr.Tab("📊 시스템 상태"):
            gr.Markdown("""
            ### 시스템 리소스
            - CPU: 25%
            - 메모리: 4.2GB / 16GB
            - GPU: 15%

            ### 활성 모듈
            - ✅ 음성 인식
            - ✅ 음성 합성
            - ✅ 컴퓨터 비전
            - ✅ 자연어 처리
            """)

    return interface

# 실행
def run_jarvis():
    """JARVIS 실행"""

    print("""
    선택하세요:
    1. 콘솔 모드
    2. 웹 UI 모드
    """)

    choice = input("선택 (1-2): ")

    if choice == "1":
        # 콘솔 모드
        user_name = input("당신의 이름은? ")
        jarvis = JARVIS(user_name)

        # 비동기 실행
        asyncio.run(jarvis.run())
```

```
    elif choice == "2":
        # 웹 UI 모드
        ui = create_jarvis_ui()
        ui.launch(share=True)

if __name__ == "__main__":
    run_jarvis()
```

10.3 고급 기능 추가

```
# jarvis_advanced_features.py - JARVIS 고급 기능

import schedule
import requests
from bs4 import BeautifulSoup
import wikipedia
import pyttsx3
from googletrans import Translator

class AdvancedJARVIS(JARVIS):
    """고급 기능이 추가된 JARVIS"""

    def __init__(self, user_name: str = "User"):
        super().__init__(user_name)

        # 추가 컴포넌트
        self.translator = Translator()
        self.scheduler = schedule
        self.news_api_key = "YOUR_NEWS_API_KEY"

    async def handle_information_query(self, query: str) -> Dict[str,
Any]:
        """정보 검색 처리"""

        # Wikipedia 검색
        try:
            wikipedia.set_lang("ko")
            summary = wikipedia.summary(query, sentences=3)

            self.speak(f"{query}에 대해 찾은 정보입니다.")
            self.speak(summary)

            return {
                'success': True,
                'source': 'wikipedia',
                'content': summary
            }

        except wikipedia.exceptions.PageError:
            self.speak(f"죄송합니다. {query}에 대한 정보를 찾을 수 없습니다.")
```

```
            return {'success': False}

    async def get_news_briefing(self) -> Dict[str, Any]:
        """뉴스 브리핑"""

        self.speak("오늘의 주요 뉴스를 전해드리겠습니다.")

        # 뉴스 API 호출 (예시)
        url = f"https://newsapi.org/v2/top-headlines?country=kr&apiKey=
{self.news_api_key}"

        try:
            response = requests.get(url)
            news_data = response.json()

            articles = news_data.get('articles', [])[:5]  # 상위 5개

            for i, article in enumerate(articles, 1):
                title = article.get('title', '')
                self.speak(f"{i}번 뉴스: {title}")
                await asyncio.sleep(1)

            return {
                'success': True,
                'articles': articles
            }

        except Exception as e:
            self.speak("뉴스를 가져오는데 실패했습니다.")
            return {'success': False, 'error': str(e)}

    async def translate_text(self, text: str, target_lang: str = 'en') ->
str:
        """텍스트 번역"""

        try:
            result = self.translator.translate(text, dest=target_lang)
            translated = result.text

            self.speak(f"번역 결과: {translated}")

            return translated

        except Exception as e:
            self.speak("번역에 실패했습니다.")
            return ""

    def schedule_task(self, task_name: str, time_str: str, function):
        """작업 스케줄링"""

        self.scheduler.every().day.at(time_str).do(function)

        self.speak(f"{task_name} 작업을 {time_str}에 예약했습니다.")
```

```
    async def health_monitoring(self) -> Dict[str, Any]:
        """건강 모니터링 (시뮬레이션)"""

        # 실제로는 웨어러블 디바이스 API 연동
        health_data = {
            'heart_rate': np.random.randint(60, 100),
            'steps': np.random.randint(5000, 10000),
            'sleep_hours': np.random.uniform(6, 9),
            'stress_level': np.random.choice(['낮음', '보통', '높음'])
        }

        self.speak(f"""
        오늘의 건강 상태입니다.
        심박수: 분당 {health_data['heart_rate']}회
        걸음수: {health_data['steps']}보
        수면 시간: {health_data['sleep_hours']:.1f}시간
        스트레스: {health_data['stress_level']}
        """)

        # 건강 조언
        if health_data['steps'] < 7000:
            self.speak("오늘은 조금 더 걸으시는 것이 좋겠습니다.")

        if health_data['stress_level'] == '높음':
            self.speak("스트레스가 높은 것 같습니다. 잠시 휴식을 취하세요.")

        return health_data

    async def smart_conversation(self, context: List[Dict]) -> str:
        """컨텍스트 기반 스마트 대화"""

        # 대화 맥락 분석
        recent_topics = []
        for item in context[-5:]:  # 최근 5개 대화
            if item['speaker'] == 'user':
                # 간단한 토픽 추출
                if '날씨' in item['text']:
                    recent_topics.append('weather')
                elif '일정' in item['text']:
                    recent_topics.append('schedule')

        # 맥락에 맞는 추가 정보 제공
        suggestions = []

        if 'weather' in recent_topics:
            suggestions.append("우산을 챙기는 것이 좋겠습니다.")

        if 'schedule' in recent_topics:
            suggestions.append("오늘 일정을 다시 확인하시겠습니까?")

        return suggestions

    async def emotion_detection(self, audio_features: Dict) -> str:
        """음성에서 감정 감지 (시뮬레이션)"""
```

```
        # 실제로는 음성 특징(피치, 톤, 속도 등) 분석
        emotions = ['neutral', 'happy', 'sad', 'angry', 'surprised']
        detected_emotion = np.random.choice(emotions)

        # 감정에 맞는 응답 조정
        if detected_emotion == 'sad':
            self.tts.voice_styles['current'] =
self.tts.voice_styles['calm']
            self.speak("기분이 좋지 않으신 것 같네요. 제가 도울 수 있는 것이 있을까요?")

        elif detected_emotion == 'happy':
            self.tts.voice_styles['current'] =
self.tts.voice_styles['happy']
            self.speak("기분이 좋으신 것 같아 저도 기쁩니다!")

        return detected_emotion

# 전체 시스템 통합 데모
async def jarvis_demo():
    """JARVIS 전체 기능 데모"""

    jarvis = AdvancedJARVIS("김철수")

    # 시나리오 1: 아침 루틴
    print("\n=== 시나리오 1: 아침 루틴 ===")

    await jarvis.handle_greeting()
    await asyncio.sleep(2)

    # 날씨 확인
    await jarvis.process_command("오늘 날씨 어때?")
    await asyncio.sleep(2)

    # 뉴스 브리핑
    await jarvis.get_news_briefing()
    await asyncio.sleep(2)

    # 일정 확인
    await jarvis.process_command("오늘 일정 알려줘")
    await asyncio.sleep(2)

    # 시나리오 2: 작업 지원
    print("\n=== 시나리오 2: 작업 지원 ===")

    # 정보 검색
    await jarvis.handle_information_query("인공지능")
    await asyncio.sleep(2)

    # 번역
    await jarvis.translate_text("안녕하세요", "en")
    await asyncio.sleep(2)

    # 시나리오 3: 스마트홈
```

```
    print("\n=== 시나리오 3: 스마트홈 제어 ===")

    await jarvis.process_command("거실 전등 켜줘")
    await asyncio.sleep(2)

    await jarvis.process_command("에어컨 온도 24도로 설정해줘")
    await asyncio.sleep(2)

    # 시나리오 4: 건강 관리
    print("\n=== 시나리오 4: 건강 관리 ===")

    await jarvis.health_monitoring()
    await asyncio.sleep(2)

    # 종료
    await jarvis.shutdown()

# 실행
if __name__ == "__main__":
    asyncio.run(jarvis_demo())
```

Day 3 마무리 - 우리가 만든 것들

오늘 배운 내용 총정리

1. **모델 최적화**
    - 양자화: 32비트 → 8비트로 모델 크기 75% 감소
    - 가지치기: 불필요한 연결 제거로 속도 향상
    - 지식 증류: 작은 모델로 큰 모델의 성능 재현
    - 하드웨어별 최적화 전략
2. **음성 인터페이스**
    - Whisper를 활용한 고급 STT
    - 감정 표현이 가능한 TTS
    - 실시간 음성 처리 파이프라인
3. **JARVIS 시스템**
    - 음성 대화 인터페이스
    - 컴퓨터 비전 통합
    - 스마트홈 제어 (시뮬레이션)
    - 정보 검색 및 작업 자동화

3일간의 여정 요약

Day 1: 멀티모달 AI의 세계

- 이미지와 텍스트를 함께 이해하는 AI

- 스마트 사진 일기 애플리케이션

Day 2: 실시간 비전 AI

- YOLO로 객체 탐지 및 추적
- 스마트 보안 시스템 구축

Day 3: 통합 AI 시스템

- 모델 최적화로 실용적인 AI
- JARVIS 스타일 AI 비서 완성

🎓 수료를 축하합니다!

3일간의 온디바이스 AI 여정을 완주하신 것을 축하합니다!

이제 여러분은:

- ✅ 최신 AI 모델을 직접 구동할 수 있습니다
- ✅ 실시간 비전 시스템을 구축할 수 있습니다
- ✅ 음성 대화형 AI를 만들 수 있습니다
- ✅ 모델을 최적화하여 실제 제품에 적용할 수 있습니다

다음 단계는?

1. **프로젝트 확장**
    - 만든 프로젝트에 새로운 기능 추가
    - 여러 프로젝트를 결합한 통합 시스템
    - 실제 사용 가능한 제품으로 발전
2. **새로운 도전**
    - 로봇 제어 시스템
    - AR/VR과 AI 결합
    - 엣지 디바이스 배포
3. **커뮤니티 참여**
    - GitHub에 프로젝트 공유
    - 블로그 작성
    - 오픈소스 기여

마지막 메시지

AI는 더 이상 먼 미래의 기술이 아닙니다. 여러분의 노트북에서, 여러분의 손으로 직접 만들 수 있는 현실의 기술입니다.

이 교재에서 배운 내용은 시작에 불과합니다. AI 기술은 매일 발전하고 있으며, 여러분이 만들어갈 미래는 무한한 가능성으로 가득합니다.

"The best way to predict the future is to invent it." - Alan Kay

여러분이 만들어갈 AI의 미래를 기대합니다! 🚀

부록: 추가 학습 자료

📚 추천 도서

1. "Deep Learning" - Ian Goodfellow
2. "Hands-On Machine Learning" - Aurélien Géron
3. "The Hundred-Page Machine Learning Book" - Andriy Burkov

🌐 온라인 리소스

1. **Hugging Face**: https://huggingface.co
2. **Papers with Code**: https://paperswithcode.com
3. **Fast.ai**: https://www.fast.ai
4. **PyTorch Tutorials**: https://pytorch.org/tutorials

🎓 온라인 강좌

1. **Coursera - Deep Learning Specialization**
2. **Fast.ai - Practical Deep Learning**
3. **Udacity - AI Programming with Python**

🛠️ 유용한 도구

1. **Weights & Biases**: 실험 추적
2. **TensorBoard**: 시각화
3. **Gradio**: 데모 UI 생성
4. **Streamlit**: 웹 앱 개발

💬 커뮤니티

1. **Reddit - r/MachineLearning**
2. **Discord - Hugging Face**
3. **Stack Overflow - AI/ML 태그**
4. **Korean AI Community**: AI 개발자 모임

행운을 빕니다! 🍀