# FSDL 2022 강의 자료

## 파트 1: 실습 (Labs)

### Lab 04: 실험 관리 (Experiment Management)

실험 관리의 중요성을 이해하고, TensorBoard의 한계를 넘어 Weights & Biases(W&B)를 활용하여 체계적으로 실험을 추적, 비교, 문서화하는 방법을 배웁니다.

- **핵심 내용**
  - **실험 관리의 중요성**: 정보 손실 방지 및 재현성 확보
  - **TensorBoard 소개 및 한계**: 단일 실험 시각화에는 유용하지만, 다중 실험 비교 및 그룹화가 어려움
  - **Weights & Biases (W&B) 소개**: 우수한 사용자 경험(UX), 주요 프레임워크와의 간편한 연동, 강력한 협업 기능
  - **W&B 핵심 기능**:
    - **Dashboard**: 실험 메타데이터, 하이퍼파라미터, 시스템 자원(CPU/GPU), 모델 구조 등을 자동으로 기록 및 시각화
    - **Artifacts**: 모델 체크포인트, 데이터셋 등 모든 바이너리 파일을 버전 관리하고 계보(Lineage) 추적
    - **Tables**: 모델 입출력 데이터를 표 형태로 시각화하고 간단한 데이터 분석 수행
    - **Reports**: 실험 결과를 체계적으로 문서화하고 공유
  - **W&B 활용**:
    - **하이퍼파라미터 튜닝 (Sweeps)**: 여러 머신에서 분산하여 하이퍼파라미터 탐색을 자동화
    - **API 활용**: 코드를 통해 실험 데이터에 접근하고 관리
- **참고 자료**
  - **채널**: deep learning
  - **비디오 ID**: NEGDJuINE9E
  - **URL**: https://www.youtube.com/watch?v=NEGDJuINE9E

### Lab 05: 트러블슈팅 및 테스트 (Troubleshooting & Testing)

코드의 품질을 높이기 위한 린팅(Linting) 및 테스트 자동화 방법을 배우고, PyTorch 프로파일러를 사용하여 모델 학습 병목 현상을 진단하고 최적화하는 과정을 실습합니다.

- **핵심 내용**
  - **코드 품질 관리**:
    - **Pre-commit**: Git 커밋 전 자동으로 코드 스타일 검사(Black, flake8) 및 정적 분석을 수행
    - **ShellCheck**: 셸 스크립트의 잠재적 버그 검사
  - **ML 시스템 테스트**:
    - **Pytest**: Python 코드의 단위 테스트 및 통합 테스트 프레임워크
    - **Doctest**: 문서 문자열(docstring) 내 예제 코드를 테스트하여 문서와 코드의 일관성 유지
    - **데이터 테스트**: Great Expectations 등을 활용한 데이터 유효성 검증
    - **학습 과정 테스트**: 모델이 작은 데이터 배치에 과적합(overfit)될 수 있는지 확인하는 '기억력 테스트'
  - **성능 트러블슈팅**:
    - **PyTorch Profiler**: 모델 학습 과정의 각 단계(데이터 로딩, 연산 등)에 소요되는 시간을 측정
    - **Trace Viewer**: CPU와 GPU에서 일어나는 모든 연산을 시간 순으로 시각화하여 병목 지점(bottleneck)을 정밀하게 파악

        - **최적화**: GPU 활용률을 높이기 위한 데이터 로더 최적화, 배치 사이즈 조절 등
- **참고 자료**
    - **채널**: deep learning
    - **비디오 ID**: D65SlCSoS-0
    - **URL**: https://www.youtube.com/watch?v=D65SlCSoS-0

## Lab 06: 데이터 어노테이션 (Data Annotation)

원본 데이터(raw data)를 모델 학습에 사용 가능한 형태로 가공하는 방법을 배우고, 오픈소스 어노테이션 툴인 Label Studio를 직접 설치하고 사용하여 데이터 라벨링 작업을 실습합니다.

- **핵심 내용**
    - **데이터셋 구조 이해**: 이미지와 주석(XML 등)이 어떻게 결합되어 학습 데이터로 변환되는지 과정 학습
    - **데이터 합성 (Data Synthesis)**: 기존 어노테이션된 데이터를 활용하여 새로운 학습 데이터를 생성함으로써 데이터 부족 문제 완화
    - **Label Studio 시작하기**:
        - **설치 및 설정**: 웹 서비스로 Label Studio를 실행하고, Ngrok을 통해 외부에서 접근 가능하도록 설정
        - **프로젝트 생성**: 라벨링할 데이터를 불러와 프로젝트 구성
        - **라벨링 인터페이스 설계**: OCR 등 특정 작업에 맞는 라벨링 UI를 템플릿을 기반으로 커스터마이징
    - **데이터 어노테이션 실습**:
        - 직접 데이터를 라벨링하며 발생할 수 있는 모호함(ambiguity)을 파악
        - 명확한 어노테이션 가이드라인의 중요성 이해
- **참고 자료**
    - **채널**: deep learning
    - **비디오 ID**: zoS5Fx2Ou1Y
    - **URL**: https://www.youtube.com/watch?v=zoS5Fx2Ou1Y

## Lab 07: 웹 배포 (Web Deployment)

학습된 PyTorch 모델을 배포 가능한 TorchScript 형식으로 변환하고, Gradio를 사용하여 빠르게 웹 UI를 구축합니다. 더 나아가 모델 서버를 분리하고 AWS Lambda를 활용한 서버리스(Serverless) 배포 아키텍처를 알아봅니다.

- **핵심 내용**
    - **모델 변환**: 학습용 PyTorch 체크포인트를 가볍고 이식성 높은 TorchScript 포맷으로 변환
    - **빠른 프로토타이핑 (Gradio)**:
        - 몇 줄의 코드로 Python 함수를 감싸는 간단한 웹 UI 및 API 자동 생성
        - 노트북 환경 또는 커맨드 라인에서 실행 및 공유 가능
    - **배포 아키텍처**:
        - **Model-in-service**: 웹 서버와 모델이 함께 동작하는 단순한 구조 (Gradio 기본 방식)
        - **Model-as-a-service**: UI(프론트엔드)와 모델(백엔드)을 분리하여 독립적으로 개발 및 확장
    - **서버리스 배포 (AWS Lambda)**:
        - 서버 관리 없이 필요할 때만 코드를 실행하는 서버리스 함수로 모델 API 구축
        - Gradio 앱이 로컬 모델 대신 AWS Lambda에 배포된 모델 API를 호출하도록 변경
    - **컨테이너화 (Docker)**:
        - Docker를 사용하여 의존성 문제를 해결하고, 어떤 환경에서든 동일하게 배포할 수 있는 컨테이너 이미지 생성 과정 이해
- **참고 자료**
    - **채널**: deep learning

    - **비디오 ID**: 2j6rG-4zS6w
    - **URL**: https://www.youtube.com/watch?v=2j6rG-4zS6w

### Lab 08: 모니터링 (Monitoring)

배포된 모델이 실제 환경에서 어떻게 동작하는지 모니터링하는 방법을 배우고, 사용자 피드백과 데이터 분석을 통해 모델의 문제점을 발견하고 개선하는 과정을 실습합니다.

- **핵심 내용**
    - **프로덕션 환경에서의 모니터링 중요성**: 모델의 성능 저하, 예상치 못한 동작 등을 파악
    - **사용자 피드백 수집**:
        - Gradio의 플래그(flagging) 기능을 활용해 간단한 사용자 피드백(예: "부정확함") 수집
    - **ML 모니터링 서비스 (Gantry)**:
        - 프로덕션 환경의 모델 입출력, 사용자 피드백, 시스템 로그 등을 중앙에서 수집 및 분석
        - 로컬 로깅의 한계를 극복하고 체계적인 데이터 관리 가능
    - **데이터 분포 변화 (Distribution Shift) 탐지**:
        - 학습 데이터와 프로덕션 데이터의 분포 차이를 비교하여 모델 성능 저하의 원인 분석
        - Gantry의 프로젝션(Projection) 기능을 활용해 고차원 데이터(이미지, 텍스트)의 특정 속성을 추출하고 모니터링
    - **실패 사례 분석**:
        - 모니터링 결과를 바탕으로 모델이 자주 실패하는 입력(예: 흰 배경에 어두운 글씨, 인쇄체)을 식별
        - 분석 결과를 통해 데이터셋 보강, 모델 구조 변경 등 개선 방향 도출
- **참고 자료**
    - **채널**: deep learning
    - **비디오 ID**: -mKzxSC0r7w
    - **URL**: https://www.youtube.com/watch?v=-mKzxSC0r7w

---

## 파트 2: 이론 강의 (Lectures)

### Lecture 01: 언제 머신러닝을 사용해야 하는가와 강좌 비전

ML 프로젝트를 시작하기 전 가장 먼저 던져야 할 질문인 'ML을 사용해야 하는가?'에 답하고, 성공적인 ML 기반 제품 개발을 위한 전체 라이프사이클과 본 강좌의 목표를 제시합니다.

- **핵심 내용**
    - **ML 기반 제품의 부상**: ML 기술의 대중화와 표준화
    - **ML 프로젝트의 현실**: 학문적 환경('평평한 지구 ML')과 실제 제품 개발('둥근 지구 ML')의 차이점 (배포 후 모니터링 및 개선의 외부 루프)
    - **언제 ML을 사용해야 하는가?**:
        - **준비 상태 확인**: 데이터 수집 인프라, 팀 구성 등
        - **필요성 검토**: 규칙 기반 시스템 등 더 간단한 대안으로 해결할 수 없는 문제인가?
        - **윤리적 고려**: ML 사용이 윤리적으로 타당한가?
    - **성공적인 ML 프로젝트 발굴**:
        - **영향력(Impact)**: 예측 비용을 낮춤으로써 큰 가치를 창출하는 문제
        - **실현 가능성(Feasibility)**: 데이터 확보 용이성, 요구되는 정확도 수준, 문제 자체의 난이도
    - **ML 프로젝트의 유형**:
        - **Software 2.0**: 기존 소프트웨어 기능을 ML로 개선

        - **Human-in-the-loop**: 사람의 작업을 ML이 보조
        - **Autonomous Systems**: 사람의 개입 없이 완전 자동화
    - **ML 프로젝트 라이프사이클**: 계획 -> 데이터 수집 및 라벨링 -> 학습 및 디버깅 -> 배포 및 테스트의 반복적인 과정

### Lecture 02: 개발 인프라 및 도구

성공적인 ML 모델 개발을 위해 필요한 개발 환경, 프레임워크, 컴퓨팅 자원 및 실험 관리 도구 등 전반적인 인프라와 툴링을 다룹니다.

- **핵심 내용**
    - **소프트웨어 엔지니어링 도구**:
        - **IDE**: VS Code 추천 (원격 개발, Git 통합, 디버깅 등)
        - **환경 관리**: Conda와 Pip-tools를 활용한 재현 가능한 Python 환경 구축
    - **딥러닝 프레임워크**:
        - **PyTorch**: 강력한 생태계와 유연성으로 FSDL의 기본 선택
        - **PyTorch Lightning**: 코드 구조화, 분산 학습, 프로파일링 등을 간소화하는 상위 프레임워크
        - **Hugging Face**: 사전 학습된 모델과 데이터셋을 제공하는 핵심 플랫폼
    - **분산 학습**:
        - **Data Parallelism**: 대용량 데이터를 여러 GPU에 나누어 학습 속도 향상 (DDP)
        - **Model Parallelism (Fully Sharded Data Parallel)**: 매우 큰 모델을 여러 GPU에 분산하여 학습 (DeepSpeed, FairScale)
    - **컴퓨팅 자원 (GPU)**:
        - **클라우드 vs. 온프레미스**: 비용, 성능, 확장성 비교
        - **GPU 선택**: 최신 세대(A100 등) GPU를 여러 개 사용하는 것이 시간과 비용 면에서 더 효율적일 수 있음
    - **실험 관리 및 하이퍼파라미터 최적화**:
        - **Weights & Biases**: 실험 추적, 시각화, 공유를 위한 강력한 솔루션
        - **Sweeps**: W&B를 활용한 자동 하이퍼파라미터 탐색

### Lecture 03: 트러블슈팅 및 테스트

버그를 줄이고 개발 속도를 높이기 위한 소프트웨어 테스트 전략과, 데이터, 학습, 모델 등 ML 시스템의 각 구성 요소를 효과적으로 테스트하는 방법을 배웁니다.

- **핵심 내용**
    - **소프트웨어 테스트 철학**:
        - 테스트는 버그를 분류하는 '분류기'와 같으며, 모든 버그를 잡을 수는 없음
        - 100% 코드 커버리지보다 중요한 버그를 잡는 고품질 테스트에 집중
    - **ML 시스템 테스트 전략**:
        - **데이터 테스트**: Great Expectations 등을 활용해 데이터의 기본적인 속성(null 값, 범위 등) 검증
        - **학습 테스트**: 모델이 작은 데이터 배치를 외울(memorize) 수 있는지 확인하여 학습 파이프라인의 건강 상태 점검
        - **모델 테스트**: 특정 입력에 대한 예상 출력을 확인하는 회귀 테스트(regression test) 구축
    - **모델 트러블슈팅 3단계**:
        1. **Make it Run (실행되게 하라)**: 텐서 형태(shape) 오류, 메모리 부족(OOM), 수치적 불안정성(NaN/inf) 문제 해결
        2. **Make it Fast (빠르게 하라)**: 프로파일러를 사용해 병목 지점을 찾고 데이터 로딩, 연산 등을 최적화

3. **Make it Right (정확하게 하라)**: 모델 및 데이터의 규모를 키워(scaling) 성능을 개선

## Lecture 04: 데이터 관리

ML 프로젝트의 근간이 되는 데이터를 효과적으로 저장, 처리, 탐색, 라벨링, 버전 관리하는 다양한 기술과 도구를 알아봅니다.

- **핵심 내용**
  - **데이터 저장**:
    - **파일 시스템 vs. 오브젝트 스토리지 vs. 데이터베이스**: 각 저장 방식의 특징과 용도
    - **데이터베이스**: 구조화된 메타데이터 저장에 필수적. 바이너리 데이터는 오브젝트 스토리지에 저장하고 URL만 관리
  - **데이터 처리 및 워크플로우**:
    - **SQL, Pandas**: 데이터 탐색 및 조작의 핵심 언어
    - **Airflow, Dagster**: 여러 단계로 구성된 데이터 처리 파이프라인을 자동화하고 스케줄링
  - **데이터셋 및 라벨링**:
    - **Hugging Face Datasets**: 수천 개의 공개 데이터셋에 쉽게 접근
    - **자기 지도 학습 (Self-supervised Learning)**: 라벨링 없이 데이터 자체의 구조를 활용해 학습
    - **데이터 증강 (Data Augmentation)**: 기존 데이터에 변형을 가해 학습 데이터 양을 늘림
    - **데이터 라벨링**: 직접 라벨링, 외부 업체 활용, Label Studio와 같은 오픈소스 도구 사용
  - **데이터 버전 관리**:
    - **Git LFS**: 대용량 파일을 Git으로 추적
    - **DVC (Data Version Control)**: 코드, 데이터, 모델의 관계를 모두 추적하는 전문 버전 관리 도구

## Lecture 05: 배포

학습된 모델을 실제 사용자가 접근할 수 있도록 배포하는 다양한 아키텍처와 기술을 배우고, 성능 최적화 및 확장성 문제를 해결하는 방법을 알아봅니다.

- **핵심 내용**
  - **배포의 중요성**: 실제 환경에서 모델의 진정한 성능을 확인하고 개선하기 위한 필수 과정
  - **배포 아키텍처**:
    - **프로토타입 (Gradio, Streamlit)**: 빠른 UI 구축 및 공유
    - **배치 예측 (Batch Prediction)**: 주기적으로 예측 결과를 생성하여 데이터베이스에 저장. 실시간성이 중요하지 않은 경우 적합
    - **온라인 예측 (Online Prediction / Model-as-a-Service)**: 모델을 독립적인 API 서비스로 배포. 가장 일반적이고 유연한 방식
  - **모델 서비스 구축**:
    - **API 설계 (REST API)**: 모델 입출력을 위한 표준 인터페이스
    - **의존성 관리 (Docker)**: 컨테이너를 사용해 재현 가능하고 이식성 높은 배포 환경 구축
    - **서버리스 (AWS Lambda)**: 서버 관리 없이 확장 가능한 모델 API를 쉽게 배포
  - **성능 최적화**:
    - **CPU vs. GPU**: 추론 환경에서는 반드시 GPU가 필요한 것은 아님
    - **최적화 기법**: 모델 경량화(Distillation, Quantization), 캐싱, 배치 처리 등
  - **엣지 배포 (Edge Deployment)**:
    - 네트워크 지연 시간 최소화, 데이터 프라이버시 강화를 위해 사용자 기기(모바일, 브라우저)에서 직접 모델을 실행
    - **프레임워크**: TensorFlow Lite, PyTorch Mobile, ONNX Runtime 등

## Lecture 06: 지속적인 학습 (Continual Learning)

모델 배포 후, 실제 프로덕션 데이터를 활용하여 모델을 지속적으로 모니터링하고 재학습함으로써 성능을 유지하고 개선하는 '지속적인 학습'의 전체 과정을 다룹니다.

- **핵심 내용**
  - **지속적인 학습의 필요성**: 현실 세계의 데이터 분포는 계속 변하므로 모델도 적응해야 함
  - **재학습 전략 (Retraining Strategy)**:
    - **로깅(Logging)**: 어떤 프로덕션 데이터를 기록할 것인가?
    - **큐레이션(Curation)**: 기록된 데이터 중 어떤 것을 라벨링하고 재학습에 사용할 것인가? (Active Learning 등)
    - **재학습 트리거(Trigger)**: 언제 재학습을 시작할 것인가? (주기적, 성능 저하 시)
    - **데이터셋 구성**: 재학습에 사용할 구체적인 데이터는 어떻게 선택할 것인가?
    - **오프라인/온라인 테스트**: 재학습된 모델이 배포하기에 충분히 좋은지 어떻게 검증할 것인가?
  - **모델 모니터링과 관찰 가능성(Observability)**:
    - **핵심 지표**: 사용자 피드백, 모델 성능, 데이터 품질, 분포 변화(drift) 등
    - **모니터링**: 사전에 정의된 '알려진' 문제들을 추적
    - **관찰 가능성**: '알려지지 않은' 문제가 발생했을 때, 시스템 내부를 깊이 파고들어 원인을 분석할 수 있는 능력
  - **초기 전략**: 처음에는 수동으로, 이후에는 주기적 재학습으로 시작하여 점차 자동화 수준을 높이는 것이 현실적

## Lecture 07: 파운데이션 모델 (Foundation Models)

GPT-3, CLIP, DALL-E 2와 같은 초거대 모델, 즉 파운데이션 모델의 핵심 기술인 트랜스포머 아키텍처를 이해하고, 이를 활용한 다양한 애플리케이션과 프롬프트 엔지니어링 기법을 알아봅니다.

- **핵심 내용**
  - **트랜스포머 아키텍처**: 셀프 어텐션(Self-Attention)을 기반으로 한 딥러닝 모델의 혁신
  - **대규모 언어 모델 (LLMs)**:
    - **GPT 시리즈**: 다음 단어를 예측하며 학습, 제로샷/퓨샷 학습 능력 보유
    - **Chinchilla**: 모델 크기와 데이터양의 최적 관계(스케일링 법칙)를 제시
    - **OpenAI API**: gpt-3와 같은 LLM을 API 형태로 사용
  - **프롬프트 엔지니어링**:
    - 모델이 원하는 결과를 내도록 입력(프롬프트)을 설계하는 기술
    - "Let's think step-by-step"과 같은 간단한 문구 추가만으로 성능 향상 가능
    - JSON 형식 지정 등 출력 포맷 제어
  - **다양한 응용**:
    - **코드 생성**: GitHub Copilot, AlphaCode
    - **시맨틱 검색**: 텍스트나 이미지를 벡터로 임베딩하여 의미 기반 검색
  - **멀티모달 모델**:
    - **CLIP**: 이미지와 텍스트를 동일한 벡터 공간에 임베딩
    - **DALL-E 2, Stable Diffusion**: 텍스트 프롬프트로부터 고품질 이미지를 생성하는 확산 모델(Diffusion Model)

## Lecture 08: ML 팀과 프로젝트 관리

성공적인 ML 제품을 만들기 위해 필요한 다양한 직무, 팀 구성, 프로젝트 관리 방법론 및 사용자 중심의 제품 설계 원칙을 다룹니다.

- **핵심 내용**
  - **ML 관련 직무**: ML 엔지니어, ML 연구원, 데이터 과학자, ML 플랫폼 엔지니어, ML 프로덕트 매니저(PM)의 역할과 필요 역량
  - **ML 팀 구성**:
    - 회사의 ML 성숙도에 따라 팀의 구조가 달라짐 (R&D -> 제품 팀 내 임베드 -> 독립 부서 -> ML-First)
  - **ML 프로젝트 관리의 어려움**:
    - 불확실성이 높아 정확한 일정 예측이 어려움
    - 연구와 엔지니어링 문화의 차이
  - **ML 프로젝트 관리 기법**:
    - **확률론적 계획**: 여러 대안을 동시에 탐색하며 리스크 관리
    - **빠른 성공 경험**: 작은 성공을 통해 동기 부여 및 비즈니스 가치 입증
    - **조직 교육**: ML의 특성과 한계를 조직 전체와 공유하여 기대치 조율
  - **ML 제품 설계**:
    - 사용자의 기대를 현실에 맞추는 것이 중요
    - 자동화에 과도하게 의존하지 않고, 사람의 개입(Human-in-the-loop)을 고려
    - 제품 설계 단계부터 사용자 피드백 루프를 구축

## Lecture 09: 윤리 (Ethics)

기술, 특히 머신러닝 시스템을 개발하고 배포할 때 발생하는 다양한 윤리적 문제를 구체적인 사례를 통해 살펴보고, 책임감 있는 개발자가 되기 위한 원칙과 자세를 논의합니다.

- **핵심 내용**
  - **윤리 문제의 접근법**: 추상적 이론보다 구체적인 사례를 통해 문제 이해
  - **기술 윤리의 공통 주제**:
    - **정렬(Alignment) 문제**: 우리가 원하는 것과 시스템이 최적화하는 것(Proxy) 사이의 괴리
    - **이해관계자 간의 상충(Trade-off)**: 누구의 이익을 우선할 것인가?
    - **겸손(Humility)**: 모든 답을 알 수 없음을 인정하는 자세
  - **ML 특유의 윤리적 쟁점**:
    - **공정성(Fairness)**: COMPAS 사례처럼, 모델이 특정 집단에 불리한 예측을 하는 문제. 여러 공정성 지표는 동시에 만족될 수 없음.
    - **책임성(Accountability)**: 모델의 예측 결과에 대해 설명하고 이의를 제기할 수 있는 권리. 딥러닝의 '설명 가능성'은 아직 한계가 명확함.
    - **데이터 소유권(Data Ownership)**: 인터넷에서 수집된 데이터(이미지, 텍스트)를 모델 학습에 사용하는 것의 정당성
    - **개발의 타당성**: "이 시스템을 애초에 만들어야 하는가?"라는 근본적인 질문 (예: 자율 살상 무기)
  - **책임감 있는 개발**:
    - 의료 분야 등 타 분야의 엄격한 윤리 기준에서 배울 점이 많음 (예: 임상시험 절차)
    - Model Cards, Data Cards 등을 통해 모델의 한계와 데이터의 출처를 명확히 문서화
    - 결국 좋은 제품을 만드는 기술 원칙(테스트, 모니터링, 에러 분석 등)이 윤리적인 제품을 만드는 길과 맞닿아 있음.