1장
기본 개념과 도구

1장 기본 개념과 도구

1.1 이미지 처리와 컴퓨터 비전

1.2 필요한 도구들

1.1 이미지 처리와 컴퓨터 비전

1. 이미지 처리와 컴퓨터 비전

- **이미지 처리란?**
  - 기술 발전의 중심에는 끊임없이 진화하고 발전하는 분야인 **이미지 처리**(image processing)가 있음
  - 이 분야는 우리가 주변 세계와 상호 작용하고 이해하는 방식을 근본적으로 변화시켜 전례 없는 방식으로 우리의 삶을 풍요롭게 하는 수많은 애플리케이션을 탄생

1. 이미지 처리와 컴퓨터 비전

- **이미지 처리란?**

  아날로그 이미지 처리

  - 아날로그 이미지 처리(analog image processing)는 사진 촬영 시 필름에 다양한 화학 처리를 하거나 카메라로 촬영한 이미지를 변경 및 편집하기 위해 물리적 필터를 사용하는 등 물리적 수단을 이용해 이미지를 조작하거나 편집하는 것을 말함

  디지털 이미지 처리

  - 디지털 이미지 처리(digital image processing)는 컴퓨터가 디지털 이미지를 처리하는 데 수학적 알고리즘과 계산 기술에 의존
  - 이러한 기술에는 이미지 향상(image upscaling), 이미지 복원(image restoration), 특징 추출(feature extraction) 등이 포함될 수 있음
  - 디지털 형식은 기술의 발전과 함께 기하급수적으로 성장했으며 광범위한 응용 가능성을 보여주어 우리 책에서는 주로 디지털 이미지 처리에 초점을 맞춤

1. 이미지 처리와 컴퓨터 비전

- **이미지 처리란?**
  - 다음은 이미지에서 한 부분의 픽셀 값을 행렬화하여 표현한 것

▼ **그림 1-1 이미지 픽셀**

187	187	187	194	197	173	77	25	19	19
190	187	190	191	158	37	15	14	20	20
187	182	180	127	32	16	13	16	14	12
184	186	172	100	20	13	15	18	13	18
186	190	187	127	18	14	15	14	12	10
189	192	192	148	16	15	11	10	10	9
192	195	181	37	13	10	10	10	10	10
189	194	54	14	11	10	10	10	9	8
189	194	19	16	11	11	10	10	9	9

1. 이미지 처리와 컴퓨터 비전

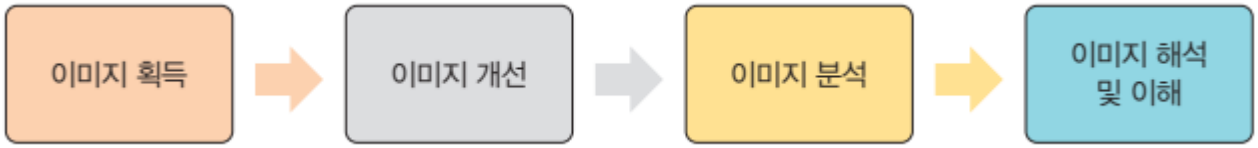
- **이미지 처리란?**

  디지털 이미지의 처리 단계

  - 디지털 이미지를 분석하고 조작하는 과정은 여러 단계로 이루어져 있으며, 각 단계는 특정 목표를 달성하기 위해 설계
  - 이 과정은 특정 알고리즘의 사용을 포함하며, 그 목표는 디지털 이미지의 품질을 향상시키거나 이미지에서 정보를 추출하는 것

1. 이미지 처리와 컴퓨터 비전

▼ 그림 1-3 이미지 처리의 단계



1. 이미지 처리와 컴퓨터 비전

- **이미지 처리란?**
  - 이미지 처리의 단계는 다음과 같음

    1. 이미지 획득

    이미지 획득(image acquisition)은 카메라, 스캐너, 또는 다른 이미지 캡처 장치를 통해

    이루어짐

    획득한 이미지는 디지털 형식으로 변환하여 컴퓨터에서 처리할 수 있도록 함

    이 과정에서는 이미지의 품질을 최대한 보존하면서 디지털 데이터로 변환하는 것이 중요

    이를 위해 다양한 이미지 센서와 변환 알고리즘이 사용

    또한 이미지 획득 과정에서 발생 할 수 있는 여러 문제, 예를 들어 노이즈, 왜곡, 빛의 변화

    등을 처리하는 기술도 중요

1. 이미지 처리와 컴퓨터 비전

- **이미지 처리란?**

  2. 이미지 개선

  이미지 개선(image enhancement) 단계에서는 이미지의 품질을 향상시키는 데 초점을

  맞춤

  이는 노이즈 제거(noise reduction), 명암 조절(contrast adjustment), 색상

  보정(colorcorrection) 등의 작업을 포함

  이 단계의 목표는 이미지를 더욱 명확하고 이해하기 쉽게 만드는 것

  이를 위해 다양한 필터링 기법과 히스토그램 평활화(histogram equalized),

  샤프닝(sharpening) 등의 기술이 사용

  이러한 기술들은 이미지의 노이즈를 줄이고, 세부 사항을 강조하며, 이미지의 전반적인

  품질을 향상시킴

1. 이미지 처리와 컴퓨터 비전

▼ 그림 1-2 이미지 샤프닝 전과 후

1. 이미지 처리와 컴퓨터 비전

- **이미지 처리란?**

  3. 이미지 분석

  이미지 분석(image analysis) 단계에서는 이미지의 유용한 정보를 추출

  이는 특징 추출(feature extraction), 패턴 인식(pattern recognition), 객체 감지(object detection) 등의 작업을 포함

  이 단계의 목표는 이미지에서 의미 있는 데이터를 얻는 것

  이를 위해 에지 검출(edge detection), 코너 검출(corner detection), 텍스처 분석(texture analysis) 등의 기술이 사용

  이러한 기술들은 이미지의 구조와 패턴을 파악하고, 이미지에서 중요한 특징을 식별

  하며, 이 특징을 이용하여 이미지를 분석

1. 이미지 처리와 컴퓨터 비전

- **이미지 처리란?**

  4. 이미지 해석 및 이해

  이미지 해석 및 이해(image interpretation and understanding) 단계에서는 이미지

  분석 단계에서 얻은 데이터를 이용하여 이미지를 해석하고 이해

  이는 이미지 분류(imageclassification), 이미지 검색(image retrieval), 이미지 인식(image recognition) 등의 작업을 포함

  이 단계의 목표는 이미지에서 얻은 데이터를 의미 있는 방식으로 사용하는 것

  이를 위해 패턴 매칭(pattern matching), 머신 러닝(ML, Machine Learning), 딥러닝(DL, DeepLearning) 등의 기술이 사용

  이러한 기술들은 이미지에서 얻은 데이터를 분석하고, 이 데이터를 이용하여 이미지를

  분류하거나, 특정 패턴을 인식하거나, 이미지 내의 객체를 식별하는 등의 작업을 수행

1. 이미지 처리와 컴퓨터 비전

- **이미지 처리란?**

  이미지 처리의 활용 범위

  - 이미지 처리는 여러 영역에서 광범위하게 활용되며 현대 기술의 초석으로써 그 위상을 확고히 하고 있음

  일상 생활에서의 이미지 처리

  - 이미지 처리는 이미지를 개선하고 이상 징후를 감지하는 의료 영상, 매핑 및 감시를 위한 위성 영상, 물체 감지 및 내비게이션을 위한 자율 주행 차량, 심지어 스마트폰 카메라, 소셜 미디어 필터 등 우리의 일상생활에서도 매우 중요하게 사용
  - 시각 장애인의 접근성을 개선하는 데도 크게 기여하는데, 에지 검출 및 분할과 같은 기술로 시각 데이터를 해석하여 오디오 또는 촉각 피드백으로 변환할 수 있는 소프트웨어에 사용

1. 이미지 처리와 컴퓨터 비전

- **이미지 처리란?**

  이미지 처리의 진화

  - 이미지 처리를 매력적인 분야로 만드는 것은 다양한 애플리케이션뿐만이 아님
  - 이미지 처리 분야는 변화하는 기술 환경에 적응하면서 지속적으로 진화
  - 아날로그 처리의 초창기부터 머신 러닝과 인공지능의 현재에 이르기까지 이미지 처리 분야는 크게 확장되고 변화하면서 복잡하고 정교해짐
  - 오늘날에는 딥러닝 기술이 이미지 처리 작업에 적용되어 시각적 세계를 볼 수 있을 뿐만 아니라 이해할 수 있는 시스템이 만들어지고 있음

1. 이미지 처리와 컴퓨터 비전

- **이미지 처리란?**

  이미지 처리의 가치와 가능성

  - 이미지 처리는 강력하고 필수 불가결한 도구로 우리 기술에서 없어서는 안 될 존재가 됨
  - 다양한 기술과 응용 분야, 그리고 끊임없이 진화하는 특성을 통해 흥미롭고 혁신적인 방식으로 우리의 미래를 만들어갈 수 있는 가능성을 지니고 있음

1. 이미지 처리와 컴퓨터 비전

- **컴퓨터 비전이란?**

  이미지 처리와 컴퓨터 비전

  - **컴퓨터 비전은 기계가 시각적 데이터를 이해하고 분석하는 능력을 개발하는 과학 분야**
  - 이미지 처리와 컴퓨터 비전은 공통점이 많지만, 그 목적과 접근 방식에서 중요한 차이점이 있음
  - 두 분야 모두 디지털 이미지를 사용하여 우리가 세상을 이해하는 방식을 확장하고 변형하는 데 초점을 맞추고 있지만, 추구하는 목표와 사용하는 기술은 다름
  - 이미지 처리는 주로 디지털 이미지의 향상, 변형, 복원 등에 중점을 둠
  - 반면 컴퓨터 비전은 이미지 처리에서 생성된 이미지를 분석하고 해석하는 데 초점을 맞춤
  - 컴퓨터 비전은 더 높은 수준의 이해를 필요로 하며, 객체 인식, 패턴 분석, 이미지 분류 등의 작업을 포함
  - 컴퓨터 비전의 목표는 디지털 이미지를 통해 우리가 세상을 인식하고 이해하는 방식을 모방하고, 이미지로부터 의미 있는 정보를 추출하는 것

1. 이미지 처리와 컴퓨터 비전

▼ 그림 1-4 객체 인식 예시

1. 이미지 처리와 컴퓨터 비전

- **컴퓨터 비전이란?**

  컴퓨터 비전의 정의

  - 넓게 정의하면, 컴퓨터 비전은 **시각 기계의 과학 및 기술**
  - 좀 더 구체적으로는 이미지에서 정보를 추출하는 인공 시스템의 이론과 관련이 있음
  - 이미지 데이터는 비디오 시퀀스, 여러 카메라의 뷰 또는 의료용 스캐너의 다차원 데이터 등 다양한 형태를 취할 수 있음
  - 컴퓨터 비전과 인간의 비전은 비슷한 목표를 공유하지만 프로세스에는 큰 차이가 있음
  - 인간은 수년간의 신경 훈련과 개발이 필요한 작업인 물체를 쉽게 인식하고 장면의 깊이를 인식
  - 반면 컴퓨터 비전 시스템은 패턴 인식과 학습 기법에 의존하여 이미지를 해석하는데, 이 과정에는 광범위한 컴퓨팅 리소스와 정교한 알고리즘이 필요함
  - 컴퓨터 비전의 범위를 완전히 이해하려면 관련된 다양한 기술을 살펴보는 것이 필수
  - 이러한 기술은 크게 낮은 수준, 중간 수준, 높은 수준의 비전 작업으로 분류할 수 있음

1. 이미지 처리와 컴퓨터 비전

▼ 그림 1-5 수준에 따른 비전 작업 분류

낮은 수준 비전 작업	중간 수준 비전 작업	높은 수준 비전 작업
• 노이즈 제거 • 대비 향상 • 채도 향상 • 에지 검출	• 이미지 영역 분할 • 이미지 객체로 분할 • 이미지 광학 흐름 추정	• 객체 인식 • 장면 재구성 • 이미지 학습 및 추론

1. 이미지 처리와 컴퓨터 비전

- **컴퓨터 비전이란?**
  - 이러한 기술의 초석은 이미지에서 가장자리, 모서리 또는 텍스처와 같은 고유한 속성을 추출하는 **특징 추출**
  - 이러한 특징은 패턴 인식이나 물체 감지와 같은 더 높은 수준의 작업을 위한 기초가 됨
  - 머신 러닝 알고리즘, 특히 딥러닝은 최적의 특징을 자동으로 학습하고 데이터의 복잡한 패턴을 인식하여 이러한 작업을 혁신적으로 개선
  - 우리 책에서 많이 다룰 특정 유형의 딥러닝 모델인 **합성곱 신경망**(CNN, Convolution NeuralNetwork)은 많은 컴퓨터 비전 작업의 표준으로 부상
  - 이러한 네트워크는 제공된 데이터에서 추출된 특징의 공간적 계층 구조를 자동으로 학습하도록 설계
  - **이미지 분류**(imageclassification), **물체 감지**(object detection), **의미적 분할**(semantic segmentation)과 같은 작업에서 사용되었고, 매우 성공적인 것이 입증

1. 이미지 처리와 컴퓨터 비전

▼ **그림 1-6 의미적 분할 예시**

1. 이미지 처리와 컴퓨터 비전

- **컴퓨터 비전이란?**

  컴퓨터 비전의 활용

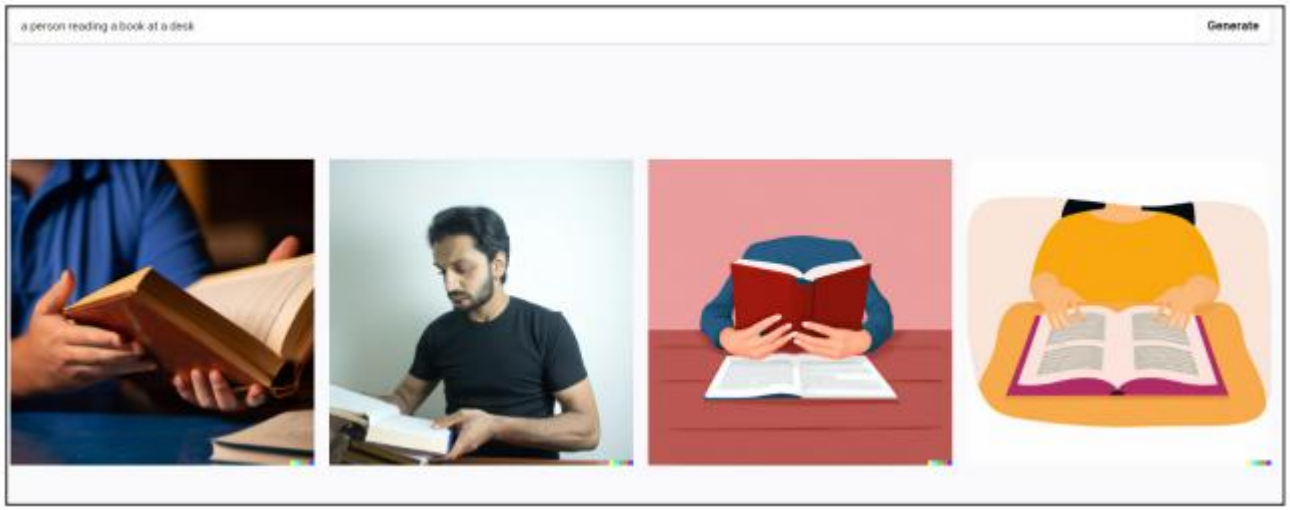
  - 컴퓨터 비전 기술을 이해하는 것도 중요하지만, 컴퓨터 비전 애플리케이션의 범위를 이해하면 이 분야의 범위와 잠재력을 더욱 명확히 알 수 있음
  - 컴퓨터 비전 시스템은 놀랍도록 다양한 산업과 애플리케이션에서 찾아볼 수 있음

1. 이미지 처리와 컴퓨터 비전

▼ **그림 1-7 농업 분야 컴퓨터 비전 활용 예시**

1. 이미지 처리와 컴퓨터 비전

▼ 그림 1-8 DALL·E 2에서 생성한 이미지 (프롬프트: a person reading a book at a



1. 이미지 처리와 컴퓨터 비전

- **이미지 처리와 컴퓨터 비전의 연관성**
  - 두 분야의 관계를 자세히 살펴보기 전에 각 분야가 무엇을 포함하는지 다시 한번 정리해보자

    이미지 처리는 노이즈 제거, 향상 및 분할과 같이 이미지에 적용되는 변환에 중점을 둠

    이미지 품질을 개선하거나 추가 처리를 위해 이미지를 준비하는 것이 목표

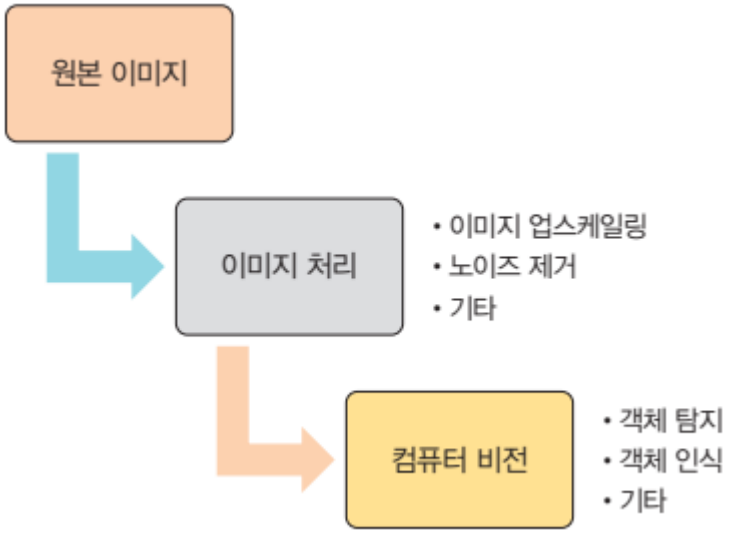
    컴퓨터 비전은 인간의 시각 능력을 모방하여 디지털 이미지나 동영상에서 높은 수준의

    이해를 추출하는 것을 목표로 함

    여기에는 사물을 인식하고, 분류하고, 장면에서 사물의 행동을 해석하는 것도 포함

1. 이미지 처리와 컴퓨터 비전

▼ 그림 1-9 이미지 처리와 컴퓨터 비전의 연결성



1. 이미지 처리와 컴퓨터 비전

- **이미지 처리와 컴퓨터 비전의 연관성**
  - 간단한 컴퓨터 비전 애플리케이션인 얼굴 인식을 예로 들어보자
  - 컴퓨터 비전 알고리즘이 얼굴을 인식하기 전에 입력 이미지는 여러 이미지 처리 단계를 거치는 경우가 많음
  - 여기에는 이미지를 그레이 스케일로 변환(grayscale conversion)하고, 각 픽셀의 값을 정규화(normalization)하고, 존재하는 모든 노이즈를 제거(denoising)하는 작업이 포함될 수 있음
  - 이러한 단계를 거친 후에야 컴퓨터 비전 알고리즘으로 이동하고 이미지에 컴퓨터 비전 알고리즘을 적용하여 이미지에 있는 얼굴을 식별하거나 인식
  - 두 분야의 관계는 단순히 순차적일 뿐만 아니라 상호 보완적인 관계이기도 함
  - 즉, 이미지 처리와 컴퓨터 비전은 목적과 기술은 다르지만 기계가 '볼 수 있도록' 하는 과정과 '봐왔던 것을 재창조'하는 과정에서 서로 보완해주며 밀접하게 얽혀 있는 분야
  - 두 분야는 인공지능 산업 혁명의 최전선에서 그 한계를 확장해가며 계속 발전할 것

2. 필요한 도구들

- **OpenCV**
  - OpenCV는 컴퓨터로 이미지나 영상을 읽고, 이미지의 사이즈 변환이나 회전, 선분 및 도형 그리기, 채널 분리 등의 연산을 처리할 수 있도록 만들어진 오픈 소스 라이브러리
  - 이미지 처리 분야에서 가장 많이 사용
  - 인텔에서 개발을 시작하여 2006년 10월 19일에 버전 1.0이 출시
  - 초기에는 C 언어만 지원하였지만, 버전 2.0부터 C++를 중심적으로 지원하고, 또 이후에 파이썬 라이브러리로도 추가
  - 안면 인식과 지문 인식, 객체 검출, 이상 탐지 등 다양한 이미지 처리에서 사용

2. 필요한 도구들

- **OpenCV**

OpenCV의 강점

- 현재 정말 다양한 언어 및 분야에서 OpenCV를 사용

1. 다양한 이미지 및 비디오 처리

OpenCV는 이미지와 비디오 데이터를 다루는 다양한 기능을 제공

이미지 사이즈 변환, 회전, 필터링, 색상 공간 변환 등 다양한 처리를 간단하게 수행할 수

있음

2. 컴퓨터 비전 알고리즘

컴퓨터 비전 알고리즘을 구현하기 위한 다양한 함수와 클래스를 제공

얼굴 인식, 객체 검출, 이미지 분할, 모션 추적 등의 작업을 쉽게 구현할 수 있음

3. 머신 러닝 통합

머신 러닝 알고리즘을 구현하기 위한 툴도 제공하며, 다양한 머신 러닝 프레임워크와의

통합을 지원

텐서플로, 파이토치 등의 머신 러닝 프레임워크와 연계하여 사용

2. 필요한 도구들

- **OpenCV**

4. 크로스 플랫폼 지원

다양한 플랫폼에서 동작하도록 설계되어 있으며, 윈도우, 리눅스, macOS, 안드로이드,

iOS등에서 사용할 수 있음

5. 오픈 소스 라이선스

아파치 라이선스(Apache License 2.0)를 따르므로 무료로 사용할 수 있고, 소스

코드에 접근하여 필요에 따라 변경할 수 있음

또 OpenCV를 사용한 툴을 상업용으로 배포하는 것 역시 가능

6. 높은 성능

C++로 작성되었기 때문에 빠른 속도와 효율적인 메모리 관리를 지원하며, 병렬 처리와

GPU가속화를 활용하여 높은 성능을 제공

2. 필요한 도구들

- **OpenCV**

  7. 커뮤니티 지원

  활발한 개발자 및 사용자 커뮤니티가 존재하여 지속적으로 개발과 지원이 이루어지고

  있음

  새로운 기능과 업데이트가 지속적으로 이루어지며, 문제를 해결하기 위한 다양한 자료와

  지원이 제공

  8. 다양한 언어 지원

  C++, 파이썬, 자바, C# 등 다양한 언어를 지원하여 개발자들이 자신에게 편한 언어를

  선택하여 사용할 수 있음

  9. 광범위한 응용 분야

  컴퓨터 비전과 이미지 처리 분야뿐만 아니라 로봇, 자율 주행, 의료, 보안, 산업 등 다양한

  분야에서 사용

2. 필요한 도구들

- **OpenCV**

  이미지 입출력

  - OpenCV의 입력과 출력에 대해 살펴보기 전에 먼저 알아야 할 사실은, OpenCV 라이브러리는 구글 코랩이 아닌 로컬 컴퓨터를 기준으로 만들어졌다는 것
  - 즉, 온라인 서버에서 구동하는 구글 코랩에서는 이미지, 영상 출력 기능에 해당하는 기능을 온전히 모두 사용해볼 수 없음
  - 이 때문에 이후 코드에서 이미지를 출력할 때는 코랩에서 동작하는 보완된 코드를 사용하고 있음

2. 필요한 도구들

▼ 그림 1-13 like_lenna.png 이미지

2. 필요한 도구들

- **OpenCV**
  - 이 이미지는 우리가 사용할 like_lenna.png 파일
  - 다음 코드로 이미지 데이터를 가져와서 사용

```
!wget https://raw.githubusercontent.com/Cobslab/imageBible/main/image/like_lenna224.
png -O like_lenna.png
```

  - wget은 리눅스의 터미널의 명령어 중 하나로, 인터넷 주소에 있는 파일을 다운로드할 수 있게 해줌
  - 이 코드를 실행하면 코랩 서버에 바로 사용할 수 있도록 like_lenna.png 파일이 준비

2. 필요한 도구들

- **OpenCV**
  - 이제 파이썬의 OpenCV 라이브러리를 사용하여 해당 이미지 파일을 읽어보자
  - 파이썬에서 OpenCV를 사용할 때는 다음 코드처럼 cv2를 import하여 사용

```
import cv2

image = cv2.imread('like_lenna.png', cv2.IMREAD_GRAYSCALE)
if image is not None:
    print("이미지를 읽어왔습니다.")
else:
    print("이미지를 읽어오지 못했습니다.")
print(f"변수 타입: {type(image)}")
```

```
이미지를 읽어왔습니다.
변수 타입: <class 'numpy.ndarray'>
```

2. 필요한 도구들

- **OpenCV**
  - 출력 결과를 보니 OpenCV에서 해당 이미지 파일을 numpy.ndarray로 불러음
  - 앞에서는 이미지를 픽셀 단위의 숫자들로 구성할 수 있음을 살펴보았음
  - 이러한 숫자들은 그 수가 매우 많기 때문에 효율적으로 저장하고 관리
  - 파이썬의 기본 자료형인 리스트는 매우 편리하지만, 연산할 때 다른 언어에 비해 느리다는 단점이 있음
  - 이미지 전체 픽셀에 대해 연산을 해야 하는 상황처럼 연산량이 많아진다면 기본 자료형인 리스트가 아닌 NumPy의 numpy.ndarray를 사용하면 속도가 빨라지고, 더 적은 메모리 공간을 차지하기 때문에 연산할 때 큰 이점이 있음
  - NumPy 라이브러리는 파이썬 도구이지만, 마치 C 언어에서 처리하는 것과 같은 속도로 각 연산을 처리
  - 추후 데이터를 GPU 연산 장치로 옮길 때도 효율적

2. 필요한 도구들

- **OpenCV**
  - 현재는 조금 단순한 이미지를 먼저 다뤄보기 위해 cv2.IMREAD_GRAYSCALE 인수를 주어 이미지를 흑백으로 받아왔음
  - 이어서 이미지를 출력

```
from google.colab.patches import cv2_imshow
cv2_imshow(image)
```

2. 필요한 도구들

- **OpenCV**
  - 코랩에서는 OpenCV에서 지원하는 이미지 출력 함수 cv2.imshow를 사용할 수 없지만, 비슷하게 사용 가능하도록, cv2_imshow 함수를 만들어 제공
  - 이를 이용하여 더 다양하게 이미지를 변환해볼 것
  - 우리가 읽어온 이미지 image 변수는 어떤 형태일까?

```
print(f"이미지 배열의 형태: {image.shape}")
```

```
이미지 배열의 형태: (224, 224)
```

  - OpenCV는 이미지를 읽어와 각 픽셀에 해당하는 수를 가로 방향 길이, 세로 방향 길이 배열(numpy.ndarray)로 변수에 저장
  - 이로 인해 우리는 이미지의 각 픽셀에 접근해서 수정 혹은 변환하는 것이 가능

2. 필요한 도구들

- **OpenCV**

  이미지 변환

  - OpenCV를 사용해서 해볼 수 있는 다양한 변환 기능이 있음
  - 그중 많이 사용하는 사이즈 변환과 대칭 변환, 회전 변환 기능을 사용

  사이즈 변환

  - cv2.resize 함수를 사용해 이미지의 사이즈를 바꿀 수 있음
  - resize 함수는 다음처럼 사용

```
def resize(src: MatLike, dsize: Size | None, ds: MatLike | None = ..., fax: float
= ..., fey: float = ..., interpolation: int = ...) -> MatLike
```

2. 필요한 도구들

▼ 표 1-1 cv2.resize 함수의 인수 소개

항목	의미
src	입력 이미지 혹은 입력 영상을 입력합니다.
dsize	변환 후 이미지의 형태로, (가로의 길이, 세로의 길이) 형식으로 튜플로 입력합니다.
ds	출력 이미지를 저장할 numpy.ndarray 변수를 입력합니다.
fx, fy	입력 이미지와 출력 이미지의 배율을 의미합니다. 해당 인수를 사용하기 위해서는 dsize 인수에 None 값을 대입합니다.
interpolation	이미지를 확대할 때 비어 있는 픽셀을 채울 규칙을 지정합니다.

2. 필요한 도구들

- **OpenCV**
  - 앞에서 불러왔던 이미지의 image 변수를 사용해 이미지의 사이즈를 바꿔보자

```
image_small = cv2.resize(image,(100,100))
cv2_imshow(image_small)
```

  - (224,224) 사이즈였던 이미지를 (100,100) 사이즈로 수정
  - 이렇게 이미지의 가로, 세로 픽셀의 개수 단위를 정해여 이미지의 사이즈를 바꿀 수 있음

2. 필요한 도구들

- **OpenCV**
  - 이번에는 배율을 지정하는 방식을 사용
  - 원래 image의 사이즈 값을 넣어주었던 dsize 인수에 이미지의 사이즈 대신 None을 대입하고, fx, fy 값을 원하는 배율로 입력

```
image_big = cv2.resize(image, dsize=None, fx=2, fy=2,)
cv2_imshow(image_big)
```

2. 필요한 도구들

- **OpenCV**

  대칭 변환

  - 이미지 대칭 변환은 이미지를 수평, 수직 또는 두 축 모두에 대해 반전시키는 작업을 의미
  - OpenCV에서는 cv2.flip() 함수를 사용하여 이미지를 대칭, 변환할 수 있음
  - flip 함수는 image 변수와 대칭으로 돌릴 축을 0, 1 등으로 명시하여 변환을 수행
  - 0으로 명시하면 다음처럼 수평축으로 반전

2. 필요한 도구들

- **OpenCV**

```
image_fliped = cv2.flip(image, 0)
cv2_imshow(image_fliped)
```

2. 필요한 도구들

- **OpenCV**
  - 1로 명시하면 다음처럼 세로축으로 반전

```
image_fliped = cv2.flip(image, 1)
cv2_imshow(image_fliped)
```

2. 필요한 도구들

- **OpenCV**

  회전 변환

  - 이미지 회전 변환은 이미지를 주어진 각도만큼 회전시키는 작업을 말함
  - OpenCV에서는 cv2.getRotationMatrix2D 함수를 사용하여 회전 변환 행렬을 생성하고, cv2.warpAffine 함수를 사용하여 실제로 이미지를 회전시킴
  - cv2.getRotationMatrix2D 함수는 center 인수로 어떤 점을 기준으로 회전시킬지를, angle 인수로 얼마나 회전시킬지를, scale 인수로 결과 이미지의 배율을 어떻게 할지를 지정하여 다음처럼 해당 변환 행렬을 반환

```
def getRotationMatrix2D(center: Point2f, angle: float, scale: float) -> MatLike
```

2. 필요한 도구들

- **OpenCV**
  - cv2.warpAffine 함수는 앞에서 만들어진 변환 행렬과 이미지를 받아 다음처럼 실제 회전 변환을 수행

```
height, width = image.shape
matrix = cv2.getRotationMatrix2D((width/2, height/2), 90, 1)
result = cv2.warpAffine(image, matrix, (width, height))
cv2_imshow(result)
```

2. 필요한 도구들

- **OpenCV**
  - 다음처럼 원하는 각도로 이미지를 회전시킬 수도 있음

```
matrix = cv2.getRotationMatrix2D((width/2, height/2), 30, 1)
result = cv2.warpAffine(image, matrix, (width, height), borderValue=200)
cv2_imshow(result)
```

2. 필요한 도구들

- **OpenCV**

  자르기

  - 이미지 자르기 기능은 파이썬 배열에서의 슬라이싱 기능을 사용
  - 다음 코드는 이미지의 왼쪽 상단을 기준으로 픽셀 단위 가로 100, 세로 100만큼의 위치를 잘라줌
  - 컴퓨터에서의 배열 특성에 따라 image[:100,:100]에서 콤마(,) 앞쪽의 :100은 세로 방향의 사이즈, 뒤쪽의 :100은 가로 방향의 사이즈를 나타냄

```
cv2_imshow(image[:100, :100])
```

2. 필요한 도구들

- **OpenCV**
  - 다음 코드는 이미지의 왼쪽 상단을 기준으로 가로, 세로 모두 51번째 픽셀부터 150번째 픽셀까지의 위치를 잘라줌

```
cv2_imshow(image[50:150, 50:150])
```

2. 필요한 도구들

- **OpenCV**
  - numpy.ndarray의 슬라이싱은 원본 객체의 값을 그대로 참조
  - 즉, 할당되어 있는 픽셀의 숫자 값을 바꿔주면 원본 자체의 픽셀 값이 함께 변함
  - 다음 코드를 통해 자른 이미지에 다른 값을 할당시키면 원본 사진 자체가 변한 것을 확인할 수 있음

2. 필요한 도구들

- **OpenCV**

```
croped_image = image[50:150, 50:150]
croped_image[:] = 200
cv2_imshow(image)
```

2. 필요한 도구들

- **OpenCV**
  - 이렇게 이미지 일부를 잘라서 변화를 주고 싶지만, 원본 이미지에 영향을 미치고 싶지 않을 때는 자른 객체에 대한 깊은 복사를 하여 사용
  - NumPy의 numpy.ndarray는 copy 메서드로 깊은 복사 기능을 제공

2. 필요한 도구들

- **OpenCV**

```
image = cv2.imread('like_lenna.png', cv2.IMREAD_GRAYSCALE)
croped_image = image[50:150, 50:150].copy()
croped_image[:] = 200
cv2_imshow(image)
```

2. 필요한 도구들

- **OpenCV**
  - 이와 같이 copy 메서드를 사용하여 할당된 cropped_image 변수에는 다른 값을 할당해도 원본 이미지의 값이 변하지 않음

2. 필요한 도구들

- **OpenCV**

  도형 그리기

  - OpenCV는 읽어온 이미지 위에 원하는 도형을 그릴 수 있는 기능을 제공
  - 이 기능을 사용해 이미지 위에 객체를 구분하는 박스를 그리거나, 글씨를 쓰는 일 등의 작업을 할 수 있음
  - 자주 사용하는 기능과 함수는 다음과 같음
    - 선 그리기: cv2.line
    - 원 그리기: cv2.circle
    - 직사각형 그리기: cv2.rectangle
    - 타원 그리기: cv2.ellipse
    - 다각형 그리기: cv2.polylines, cv2.fillPoly

2. 필요한 도구들

- **OpenCV**

  선 그리기: cv2.line

  - cv2.line 함수는 다음처럼 사용

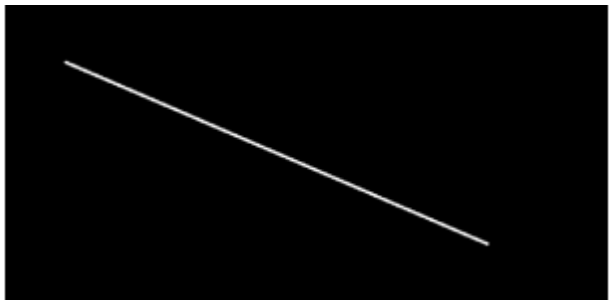
```
def line(img: MatLike, pt1: Point, pt2: Point, color: Scalar, thickness: int = ...,
lineType: int = ..., shift: int = ...) -> MatLike
```

2. 필요한 도구들

- **OpenCV**
  - 선을 그리는 cv2.line 함수는 이미지와 두 개의 점을 받아서 이어줌
  - color, thickness 등의 옵션으로 선을 다양하게 그려볼 수 있음
  - 다음 코드는 색상의 픽셀 값으로 255를 설정해 흰색 선을 그림

```
space = np.zeros((500, 1000), dtype=np.uint8)
line_color = 255
space = cv2.line(space, (100, 100), (800, 400), line_color, 3, 1)

cv2_imshow(space)
```



  - 위와 같이 평면에서 두 개 점의 좌표를 받아 선을 그려줌

2. 필요한 도구들

- **OpenCV**

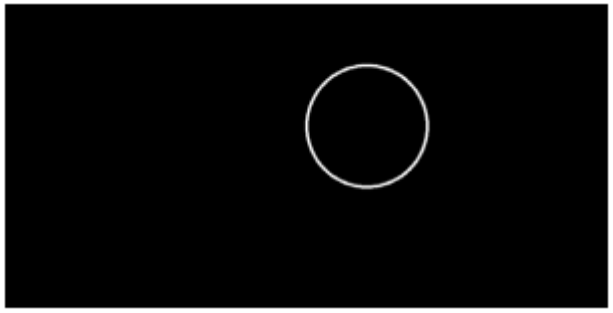
  원 그리기: cv2.circle

  - cv2.circle 함수는 원의 중심 좌표와 반지름 값을 받아 원을 그림

```
def circle(img: MatLike, center: Point, radius: int, color: Scalar, thickness: int
= ..., lineType: int = ..., shift: int = ...) -> MatLike
```

```
space = np.zeros((500, 1000), dtype=np.uint8)
color = 255
space = cv2.circle(space, (600, 200), 100, color, 4, 1)

cv2_imshow(space)
```



2. 필요한 도구들

- **OpenCV**

  직사각형 그리기: cv2.rectangle

  - 직사각형을 그리는 기능은 객체 탐지에서 특정 대상이 있을 때, 상자 표시로 해당 대상을 나타내는 데 사용하는 기능
  - 다음 코드와 같이 이미지와 두 개의 좌표를 받아 이미지 위에 직사각형을 그려줌
  - pt1, pt2는 각각 사각형의 왼쪽 위와 오른쪽 아래의 꼭짓점 좌표를 의미

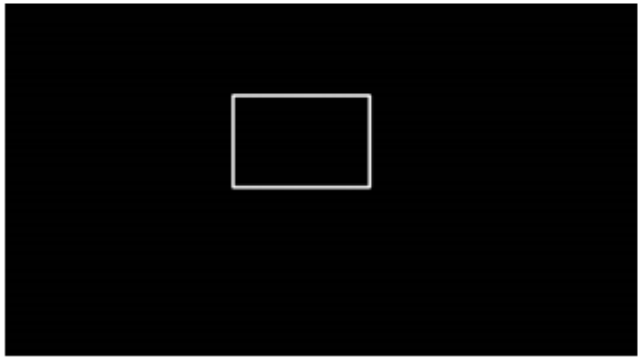
```
def rectangle(img: MatLike, pt1: Point, pt2: Point, color: Scalar, thickness: int
= ..., lineType: int = ..., shift: int = ...) -> MatLike
```

2. 필요한 도구들

- **OpenCV**
  - 다음 코드를 실행하여 확인해보자

```
space = np.zeros((768, 1388), dtype=np.uint8)
line_color = 255
space = cv2.rectangle(space, (500, 200), (800, 400), line_color, 5, 1)

cv2_imshow(space)
```



2. 필요한 도구들

- **OpenCV**

  타원 그리기: cv2.ellipse

  - 이미지와 타원의 중심 및 축, 축의 각도, 타원을 그릴 각도(시작점과 끝점)를 받아, 이미지 위에 원하는 타원을 그림

```
def ellipse(img: MatLike, center: Point, axes: Size, angle: float, startAngle:
float, endAngle: float, color: Scalar, thickness: int = ..., lineType: int = ...,
shift: int = ...) -> MatLike
```

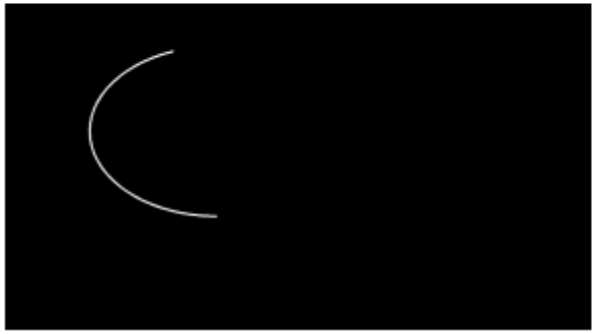
  - center에는 타원 중심의 좌표, axes에는 축의 좌표, angle에는 타원축의 각도, startAngle에는 타원을 그리기 시작할 각도 값, endAngle에는 타원 그리기를 끝낼 각도, color에는 색상(숫자로 기입), thickness에는 선의 두께를 입력
  - lineType과 shift는 어떤 타원을 그릴 것인지와 직결되는 인수가 아니므로 생략

2. 필요한 도구들

- **OpenCV**
  - 다음 코드를 실행하여 확인해보자

```
space = np.zeros((768, 1388), dtype=np.uint8)
line_color = 255
space = cv2.ellipse(space, (500, 300), (300, 200), 0, 90, 250, line_color, 4)

cv2_imshow(space)
```



2. 필요한 도구들

- **OpenCV**

  다각형 그리기: cv2.polylines, cv2.fillPoly

  - 두 가지 함수로 다각형을 그릴 수 있음
  - cv2.polylines 기능은 여러 점을 잇는 선을 그려서 다각형을 그릴 수 있게 지원하고, cv2.fillPoly 기능은 색칠된 면을 갖는 다각형을 그림

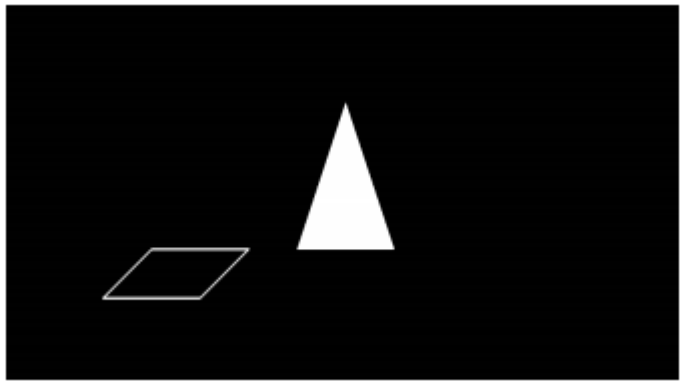
```
def polylines(img: MatLike, pts: Sequence[MatLike], isClosed: bool, color: Scalar,
thickness: int = ..., lineType: int = ..., shift: int = ...) -> MatLike


def fillPoly(img: MatLike, pts: Sequence[MatLike], color: Scalar, lineType: int =
..., shift: int = ..., offset: Point = ...) -> MatLike
```

2. 필요한 도구들

- **OpenCV**
  - 다음 코드는 두 가지 방법으로 다각형을 그림

```
space = np.zeros((768, 1388), dtype=np.uint8)
color = 255
obj1 = np.array([[300, 500], [500, 500], [400, 600], [200, 600]])
obj2 = np.array([[600, 500], [800, 500], [700, 200]])
space = cv2.polylines(space, [obj1], True, color, 3)
space = cv2.fillPoly(space, [obj2], color, 1)
cv2_imshow(space)
```



2. 필요한 도구들

- **OpenCV**
  - 이 코드에서 cv2.polylines 함수는 [[300, 500], [500, 500], [400, 600], [200, 600]] 위치의 평행사변형
  - cv2.fillPoly 함수는 [[600, 500], [800, 500], [700, 200]] 위치의 삼각형을 그려줌

2. 필요한 도구들

- 텐서플로

▼ 그림 1-14 텐서플로

TensorFlow

2. 필요한 도구들

- **텐서플로**
  - 텐서플로(TensorFlow)는 구글 브레인 팀에서 개발되어 2015년에 공개된 오픈 소스 머신 러닝 라이브러리
  - 공개 이후로 텐서플로는 머신 러닝 및 딥러닝 분야에서 중요한 역할을 하고 있으며, 연구자와 개발자들 사이에서 널리 사용되고 있음
  - 텐서플로는 이름에서 알 수 있듯이 텐서(다차원 배열)를 기반으로 하는 연산을 수행하며, 텐서플로를 사용하면 복잡한 머신 러닝 모델과 알고리즘을 쉽게 구현할 수 있음
  - 텐서플로는 데이터 플로 그래프를 사용하여 연산을 표현이 그래프는 노드와 에지로 구성되어 있으며, 노드는 연산, 에지는 노드 사이를 이동하는 데이터(텐서)를 나타냄
  - 데이터 플로 그래프를 사용해서 텐서플로는 병렬 처리와 지연 실행을 쉽게 수행

2. 필요한 도구들

- **텐서플로**

  편의성

  - 텐서플로의 특징 중 하나는 편의성에 있음
  - 고수준 API 지원, 텐서플로 개발진들이 미리 개발해둔 사전 빌드된 층 및 모델 제공, 즉시 실행 모드 등이 특징

  고수준 API 지원

  - 텐서플로의 가장 사용자 친화적인 기능 중 하나는 고수준 API인 케라스(Keras)가 있다는 것

▼ **그림 1-15 케라스**

2. 필요한 도구들

- **텐서플로**
  - 케라스 API는 간단하고 일반적인 사용 사례에 최적화되어 있어 거의 모든 표준 모델을 정의하고 구성할 수 있는 명확하고 간결한 방법을 제공
  - 간단한 선형 회귀부터 복잡한 심층 신경망까지, 단 몇 줄의 코드만으로 모델을 설정하고 컴파일할 수 있음
  - 케라스는 모델 아키텍처를 자유롭게 정의할 수 있는 기능도 제공
  - 예를 들어 순차적 모델을 사용하면 각 층에 정확히 하나의 입력 텐서와 하나의 출력 텐서가 있는 층을 쉽게 스택처럼 차곡차곡 쌓아 올릴 수 있음
  - 이는 정보의 흐름이 입력에서 출력으로 한 방향으로만 이루어지는 모델에 유용

2. 필요한 도구들

- **텐서플로**
  - 이와는 대조적으로 Model 클래스 API를 사용하면 더 복잡한 모델을 만들 수도 있음
  - 이 클래스는 층 그래프를 정의하는 데 사용
  - 다중 출력 모델, 방향성 비순환 그래프 또는 공유 층이 있는 모델(데이터 흐름이 엄격하게 선형적이지 않은 상황)을 만들 때 유용
  - 모델의 복잡성에 관계없이 케라스는 직관적이고 사용자 친화적인 모델 생성 방식을 제공

2. 필요한 도구들

- **텐서플로**

  이식성 및 호환성

  - 플랫폼 간 이식성과 호환성은 텐서플로의 가장 큰 특징 중 두 가지로, 다양한 환경과 기기에서 작업하는 개발자를 위한 매우 다재다능한 도구
  - 텐서플로는 하나의 플랫폼이나 특정 카테고리의 디바이스만을 위한 도구가 아니라 다양한 운영 환경에서 머신 러닝 애플리케이션을 위한 범용 솔루션이 될 수 있었음
  - 소프트웨어의 맥락에서 이식성은 동일한 소프트웨어를 다른 환경에서도 사용할 수 있는 사용성을 의미
  - 이는 텐서플로와 같이 광범위한 애플리케이션에 사용할 수 있도록 설계된 도구의 중요한 특성
  - 텐서플로 모델의 높은 이식성 덕분에 개발자는 한 환경에서 머신 러닝 모델을 작업한 후 테스트, 배포 또는 추가 개발을 위해 다른 환경으로 쉽게 전환할 수 있음
  - 이는 유연성과 적응성이 핵심인 머신 러닝 분야에서 특히 중요

2. 필요한 도구들

- **텐서플로**
  - 이식성은 개발자가 특정 플랫폼이나 기기에 종속되는 것을 방지
  - 다양한 하드웨어(CPU, GPU, TPU), 소프트웨어 플랫폼(윈도우, macOS, 리눅스) 등 다양한 환경에서 실행할 수 있으므로 개발자는 단일 운영 체제에 국한되지 않음
  - 텐서플로를 사용하는 개발자, 연구자 및 조직 커뮤니티는 머신 러닝 모델의 기능이나 효율성을 저하시키지 않으면서도 원하는 운영 환경을 선택할 수 있음
  - 개인용 macOS에서 작업하든, 윈도우 워크스테이션에서 작업하든, 리눅스 서버에서 작업하든 동일한 효율로 텐서플로 모델을 개발, 훈련 및 배포할 수 있음
  - 이러한 수준의 플랫폼 독립성은 서로 다른 시스템에서 작업하는 팀 간의 협업을 가능하게 하고 플랫폼별 종속성으로 인한 문제를 제거

2. 필요한 도구들

- **텐서플로**
  - 텐서플로 이식성의 또 다른 핵심 측면은 모바일 및 에지 디바이스와의 호환성
  - 텐서플로에서 제공하는 도구 세트인 텐서플로 라이트(TensorFlow Lite)를 사용하면 모바일 및 에지 장치에서 실행할 수 있는 형식으로 텐서플로 모델을 변환할 수 있음
  - 이를 통해 온디바이스 머신 러닝, 실시간 처리 및 IoT 애플리케이션과 같은 완전히 새로운 텐서플로 모델 애플리케이션의 문이 열림
  - 모바일 및 에지 디바이스에서 머신 러닝 모델을 실행할 수 있는 기능은 상당한 영향을 미칠 수 있음
  - 예를 들어 실시간으로 작동해야 하는 머신 러닝 모델은 온디바이스 추론을 통해 서버 기반 또는 클라우드 기반 솔루션에 비해 지연 시간을 크게 줄일 수 있음
  - 디바이스에서 모델을 실행하면 데이터를 서버로 전송할 필요가 없으므로 사용자 개인정보 보호가 향상될 수 있음

2. 필요한 도구들

- **텐서플로**
  - 텐서플로는 로컬 환경에만 국한되지 않음
  - 구글 클라우드 플랫폼(Google Cloud Platform, GCP), 아마존 웹 서비스(Amazon Web Services, AWS), 마이크로소프트 애저(Microsoft Azure)와 같은 클라우드 플랫폼과 원활하게 통합되도록 설계
  - 클라우드 호환성 덕분에 개발자는 이러한 플랫폼에서 사용할 수 있는 방대한 계산 리소스를 활용하여 로컬 머신에서는 불가능했던 복잡한 대규모 모델을 학습할 수 있음
  - 텐서플로의 이식성과 플랫폼 간 호환성은 머신 러닝을 위한 매우 유연하고 보편적으로 적용할 수 있는 도구
  - 개인용 컴퓨터에서 모바일 기기, 단일 머신에서 방대한 클라우드 기반 인프라에 이르기까지 다양한 플랫폼과 기기에서 모델을 개발하고 배포할 수 있는 능력은 텐서플로에게 상당한 우위를 제공
  - 이러한 기능 덕분에 텐서플로는 개발자에게 실용적인 선택이 될 뿐만 아니라 다양한 애플리케이션에서 머신 러닝의 접근성과 다용도성, 영향력을 높일 수 있음

2. 필요한 도구들

- **텐서플로**

  확장성

  - 텐서플로의 확장성은 다른 머신 러닝 라이브러리와 차별화되는 포인트 중 하나이며 머신 러닝 분야에서 인기가 높은 주요 이유
  - 확장성이란 시스템, 네트워크 또는 프로세스가 증가하는 작업량을 유능한 방식으로 처리할 수 있는 능력 또는 이러한 증가를 수용하기 위해 확장할 수 있는 잠재력을 말함
  - 텐서플로는 모델 개발과 배포 모두를 위한 확장 가능한 솔루션을 제공하므로 다양한 규모의 머신 러닝 애플리케이션에 이상적인 선택

2. 필요한 도구들

- **텐서플로**
  - 머신 러닝 프로젝트는 제한된 양의 데이터와 간단한 모델로 소규모로 시작하는 경우가 많음
  - 프로젝트가 발전함에 따라 더 큰 데이터 세트를 처리하고, 더 복잡한 모델을 만들고, 더 많은 컴퓨팅 리소스를 필요로 할 수 있음
  - 증가하는 수요를 충족하기 위해 확장할 수 있는 능력은 모든 머신 러닝 프레임워크의 중요한 기능
  - 이를 통해 소규모 실험에서 대규모 배포로 좀 더 원활하게 전환할 수 있음
  - 필요에 따라 확장할 수 있는 이러한 기능은 프로젝트 초기 단계에 투자한 시간과 리소스를 낭비하지 않고 프로젝트의 수명 주기 내내 동일한 도구와 기술을 사용할 수 있도록 보장

2. 필요한 도구들

- **텐서플로**

  1. CPU, GPU 및 TPU 확장성

  코드 변경 없이, CPU(중앙 처리 장치), GPU(그래픽 처리 장치), 심지어 머신 러닝

  워크로드를 가속화하는 데 특별히 사용할 수 있는 구글의 맞춤형 개발 애플리케이션별

  집적 회로(ASIC)인 TPU(텐서 처리 장치)에서도 실행할 수 있음

  GPU와 TPU는 일반적인 CPU보다 초당 훨씬 더 많은 계산을 수행할 수 있으므로 머신

  러닝에서 흔히 사용되는 행렬 및 벡터 연산에 훨씬 더 효율적

  GPU와 TPU에서 모델을 훈련할 수 있기 때문에 텐서플로는 CPU만 사용할 때보다 훨씬

  더 큰 데이터 세트와 더 복잡한 모델을 처리할 수 있음

2. 필요한 도구들

- **텐서플로**

  2. 분산 컴퓨팅

  다양한 유형의 하드웨어에서 실행할 수 있을 뿐만 아니라 텐서플로는 분산 컴퓨팅도 지원

  이를 통해 개발자는 여러 머신에서 동시에 모델을 훈련할 수 있음

  이는 매우 큰 데이터 세트나 특히 복잡한 모델을 처리하는 데 중요한 기능

  텐서플로의 분산 전략 API는 분산 컴퓨팅의 많은 세부 사항을 추상화하여 모델 학습을

  더 쉽게 확장할 수 있게 해줌

  여러 GPU에서 동기식 트레이닝을 위한 tf.distribute.MirroredStrategy, TPU에서

  트레이닝을 위한 tf.distribute.experimental.TPUStrategy 등 데이터 및 계산을

  분산하기 위한 다양한 전략을 지원

2. 필요한 도구들

- **텐서플로**

  3. 대규모 모델 배포

  모델이 학습되면 텐서플로는 대규모로 모델을 배포할 수 있는 도구도 제공

  텐서플로 서빙(TensorFlow Serving)은 프로덕션 환경을 위해 설계된 머신러닝 모델을

  위한 유연한 고성능 서빙 시스템

  텐서플로 모델과 바로 통합할 수 있지만 다른 유형의 모델을 제공하도록 확장할 수 있음

  특히 여러 모델을 서비스하고, 모델 버전 관리를 수행하고, 다양한 유형의 하드웨어에서

  모델을 서비스할 수 있음

  매우 유연하고 확장 가능하며 대규모로 모델을 제공할 수 있도록 설계되어 프로덕션급

  제품으로 전환이 가능

2. 필요한 도구들

- **텐서플로**
  - 텐서플로의 핵심적인 기능인 확장성은 모든 규모와 다양한 복잡성을 지닌 머신 러닝 작업을 처리할 수 있음
  - 단일 CPU에서 간단한 모델을 실행하든 분산된 GPU 또는 TPU 네트워크에서 복잡한 모델을 훈련하든, 텐서플로는 작업 규모를 확장하는 데 필요한 도구와 유연성을 제공
  - 이러한 확장성은 텐서플로의 휴대성 및 광범위한 기능과 결합되어 텐서플로를 머신 러닝을 위한 강력한 도구로 만들어줌

2. 필요한 도구들

- **텐서플로**

  유연성

  - 텐서플로의 방대한 기능들 중 다른 한 핵심은 내재된 유연성
  - 이러한 유연성 덕분에 간단한 선형 회귀부터 복잡한 신경망 아키텍처에 이르기까지 다양한 작업에 고유하게 적용
  - 모델 아키텍처 설계를 넘어 데이터 전처리부터 배포까지 모든 수준의 개발에 적용
  - 지속적으로 진화하는 머신 러닝 영역에서 획일성은 더 이상 통하지 않음
  - 문제의 복잡성과 가변성으로 인해 상황에 따라 고수준 API가 아닌, 저수준 API(low level API), 사용자 정의 층들도 변경할 수 있는 도구와 프레임워크가 필요해졌음
  - 바로 이 지점에서 텐서플로의 유연성이 중요한 특성으로 부각

2. 필요한 도구들

- **텐서플로**

  1. 사용자 정의 층

  텐서플로는 기본적으로 많은 표준 층을 제공하지만, 특정 문제에 따라 고유한 층이 필요한 경우가 있을 수 있음

  텐서플로의 저수준 API는 번거로움 없이 이러한 층을 제작할 수 있는 기능을 제공

  2. 사용자 정의 손실 함수

  상황에 따라 표준 손실 함수를 사용해 모든 문제를 효과적으로 해결할 수 있는 것은 아님

  맞춤형 손실 함수를 생성할 수 있는 도구를 제공함으로써 텐서플로는 자유도 높은 모델 설계를 할 수 있음

2. 필요한 도구들

- **텐서플로**

  3. 새로운 최적화 전략 구현

  경사 하강법 변형 함수들이 머신 러닝 환경을 지배하고 있지만, 특정 시나리오에서는

  혁신적인 최적화 방법이 필요할 수 있음

  텐서플로는 이러한 방법을 원활하게 통합할 수 있도록 지원

  4. tf.data를 사용한 데이터 파이프라인

  머신 러닝에서는 데이터가 가장 중요

  텐서플로의 tf.data API는 효율적인 데이터 파이프라인을 구축할 수 있는 강력한

  유틸리티 세트를 제공

  정형 데이터, 비정형 데이터, 심지어 시계열 및 NLP를 위한 시퀀스까지, tf.data는 모든

  데이터를 처리할 수 있을 만큼 다용도로 사용할 수 있음

2. 필요한 도구들

- **텐서플로**
  - 초기 버전부터 지금까지, 텐서플로는 지속적으로 발전해왔으며, AI 발전에 있어 보조를 하기도, 주력으로 사용되기도 하였음
  - 사용자 친화적인 인터페이스나 심층적인 사용자 지정 옵션을 통해 텐서플로는 AI의 대중화를 위해 노력하고 있으며, 누구나 어디서나 머신 러닝의 힘을 활용할 수 있도록 보장