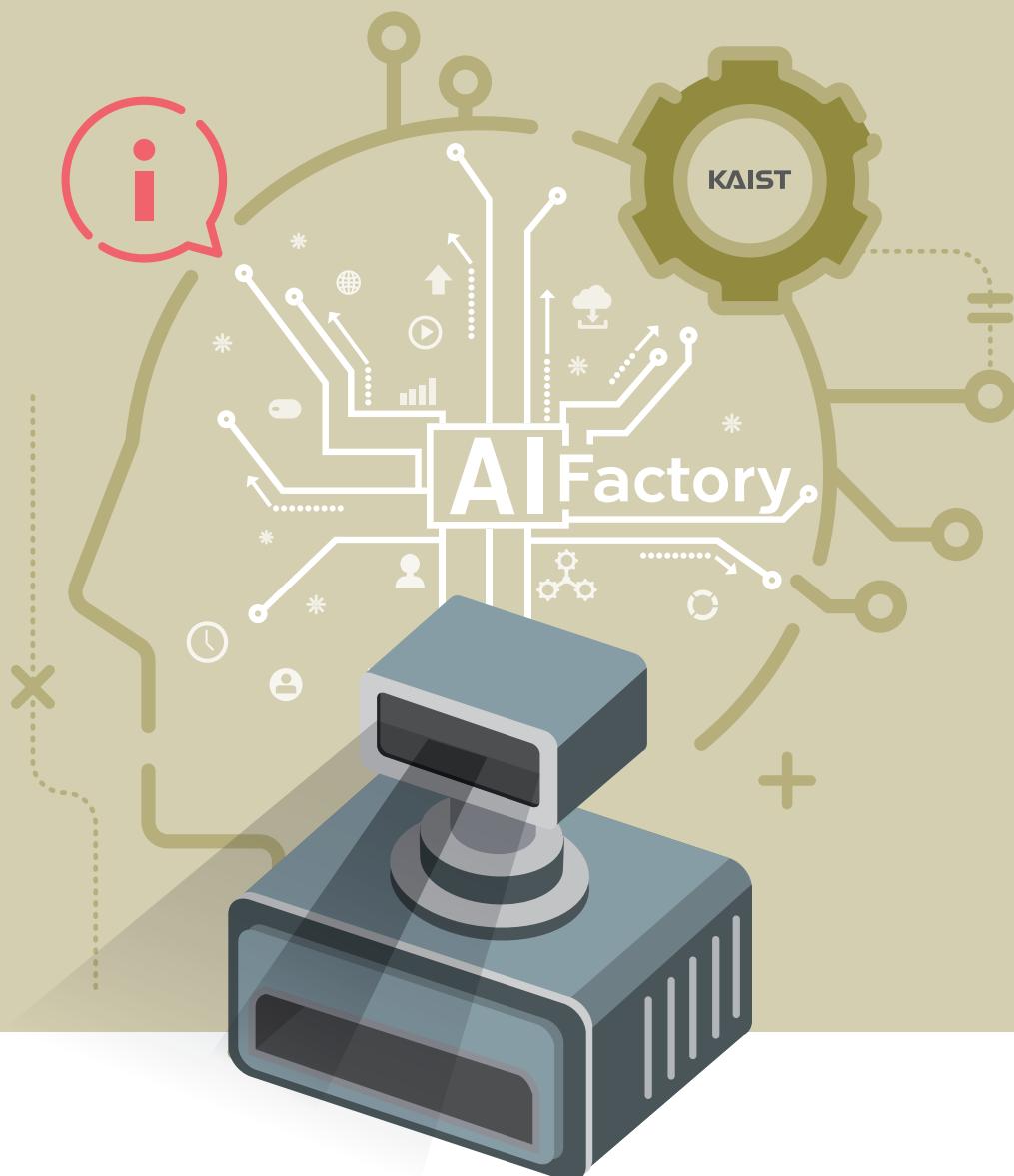
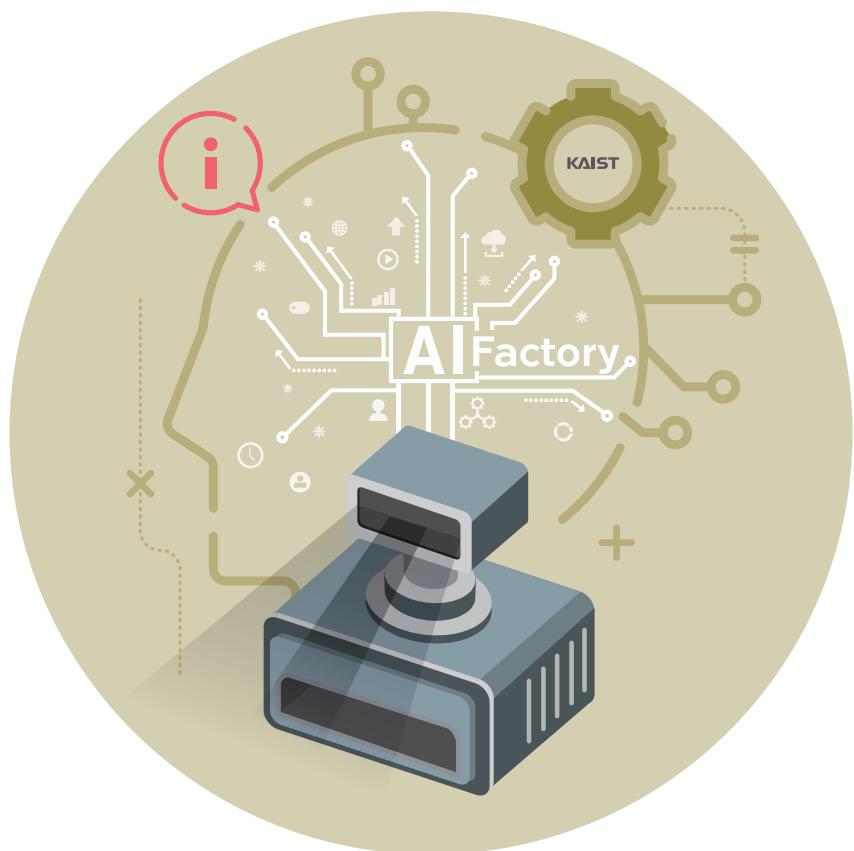


「제조 현장용 OCR학습 AI 데이터셋」 분석실습 가이드북

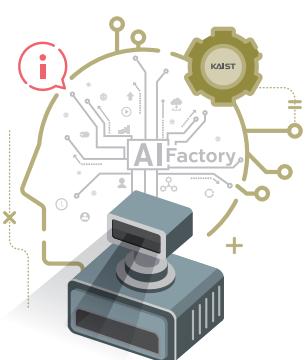


「제조 현장용 OCR학습 AI 데이터셋」 분석실습 가이드북



Contents

1	분석요약	04
2	분석 실습	
	1. 분석 개요	05
	1.1 분석 배경	05
	1) 공정(설비) 개요	
	2) 분석배경(현장이슈·pain point) 소개	
	1.2 분석 목표	06
	1) 분석목표 설정	
	2) 시사점(implication) 요약기술	
	2. 분석실습	06
	2.1 제조데이터 소개	06
	1) 데이터 수집 방법	
	2) 데이터 유형/구조	
	2.2 분석 모델 소개	07
	1) AI 모델의 목적	
	2) 딥 뉴럴 네트워크 구조	
	3) 학습 방법	
	4) 알고리즘 구축 절차	
	2.3 분석 체험	10
	1) 필요 SW, 패키지 설치 방법 및 절차 가이드	
	2) 실습 방법	
	3) 학습 데이터	
	4) 분석 단계별 Process	
	[단계 ①] 라이브러리 불러오기	
	[단계 ②] 데이터 로더 생성 및 데이터 확인	
	[단계 ③] 데이터 시각화	
	[단계 ④] 모델 학습	
	[단계 ⑤] 결과 분석 및 해석	
	3. 유사 타 현장의 「제조 현장용 OCR학습 AI 데이터셋」 분석 적용	23
부 록	분석환경 구축을 위한 설치 가이드	24



「제조 현장용 OCR학습 AI 데이터셋」분석실습 가이드북

- 필요 SW : Anaconda, Cuda, Python 3.8
- 필요 패키지 : PyTorch 1.6.0, torchvision 0.7.0, Pillow 7.2.0, matplotlib 3.3.1
- 분석 환경 : [CPU] –Intel(R) Core(TM) i7-7700K, [GPU] GeForce GTX 1070, [RAM] 32GB
- 필요 데이터 : 첨부된 ZIP 파일의 XML파일 및 JPG 파일을 참조
- 주 관 기관 : 한국과학기술원(KAIST)
- 수 행 기관 : 한국과학기술원(KAIST) 산업및시스템공학과 문일철교수



1 분석요약

No	구 분	내 용
1	분석 목적 (현장 이슈, 목적)	<ul style="list-style-type: none">- 제관 과정의 무게 충진 공정으로 Count-Weight 생산품의 핵심 품질을 좌우하는 공정이다.충진량이 정확해야 하며 이를 이미지로 기록하고 있다.지속적인 관리를 위하여 이미지에서 충진중량을 광학 문자 인식 (Optical Character Recognition)으로 DB에 자동기록할 예정이다.
2	데이터셋 형태 및 수집방법	<ul style="list-style-type: none">- 분석에 사용된 변수 : Image 데이터 및 그에 수반된 Tagging XML- 데이터 수집 방법 : 클라우드 기반 RPMS로 현장 이미지 확보- 데이터셋 파일 확장자 : jpg(이미지), xml(라벨 정보)
3	데이터 개수 데이터셋 총량	<ul style="list-style-type: none">- 데이터 개수 : 27,237개- 데이터셋 총량 : 76.1 MB
4	분석적용 알고리즘	알고리즘 합성곱신경망(CNN)
		<ul style="list-style-type: none">- CNN 알고리즘은 데이터의 특징을 복수의 필터로 추출하고 강화하여 입력 데이터를 분류하는 알고리즘으로 주로 이미지 및 영상 분류, 문자인식 등의 분야에서 높은 성능을 보인다.그 중에서도 깊은 망에 대해서도 최적화 가능한 ResNet(Residual Neural Network)은 Skip connection을 이용한 Residual learning 모델이다. 깊은 뉴럴 네트워크의 학습을 위하여 레이어와 레이어를 건너뛰어 연결 시켜주는 레지듀얼 연결을 추가하여 효율적인 결과를 얻을 수 있다.
5	분석결과 및 시사점	<ul style="list-style-type: none">- 제조 현장의 OCR은 일반적인 길거리나 문서의 OCR과는 다른 형태의 노이즈가 부여되며, 이를 일반 OCR 기계학습 모듈을 통해 판독하기 어렵다. 중소 제조 현장적용을 위한 OCR을 개발할 경우의 판독 성공률은 99% 이상을 달성할 수 있었다.

2 분석 실습

1. 분석 개요

1.1 분석 배경

1) 공정(설비) 개요

- 제시된 분석 사례는 제관 카운터웨이트(Counter-Weight)의 충진과정에 대한 내용이다. 카운터웨이트(Counter-Weight)는 쉽게 말해 무게추로써, 규정된 무게대로 무게추의 내용물이 채워져야지만 품질 문제가 없다. 이를 위하여, 충진공정은 저울을 활용하여 충진 과정을 제어하는데, 저울에 표기된 무게를 OCR로 파악할 필요성이 있다.



[그림 1] 카운터웨이트(Counter-Weight)의 장착 위치 및 형태

- 해당 공정은 철가루를 무게추에 채우는 과정으로 현장의 오염도가 매우 심하여 노이즈가 심한 상태이다. 그러므로 일반적인 OCR 모듈로는 쉽게 인식되지 않는다.

2) 분석배경(현장이슈·pain point) 소개

- 무게추의 무게 계측은 카운터웨이트(Counter-Weight)의 직접적인 품질 문제가 된다. 제시된 공정은 경북 경주 소재 한 제조 공장의 카운터웨이트(Counter-Weight) 작업 과정이며, 카운터웨이트(Counter-Weight)의 품질문제 보장을 위하여, 자동화된 무게 입력 과정을 구현하고자 데이터를 수집하였다.

1.2 분석 목표

1) 분석목표 설정

- 제안된 분석의 목표는 제관 카운터웨이트(Counter-Weight)의 충진과정에서 계측되는 무게를 저울의 LED 숫자표기를 OCR하여 디지털로 변환하는 과정을 목표로 한다.

2) 시사점(implication) 요약기술

- 현재 충청도 소재의 한 공장에는 충진공정에서 충진이 얼마나 되었는지 자동적으로 기록할 수 있는 AI 기법의 적용이 필요한 상황이다. 이에 AI기반 OCR모형을 구축하고 AI로 현재 계측된 생산품의 중량을 확인할 수 있는 분석을 수행하고자 한다. 본 분석은 품질관리 문제로 열악한 중소기업에 AI를 적용하여 품질 데이터 획득에 기여했다는 점에서 시사점이 크다고 판단된다.

2. 분석실습

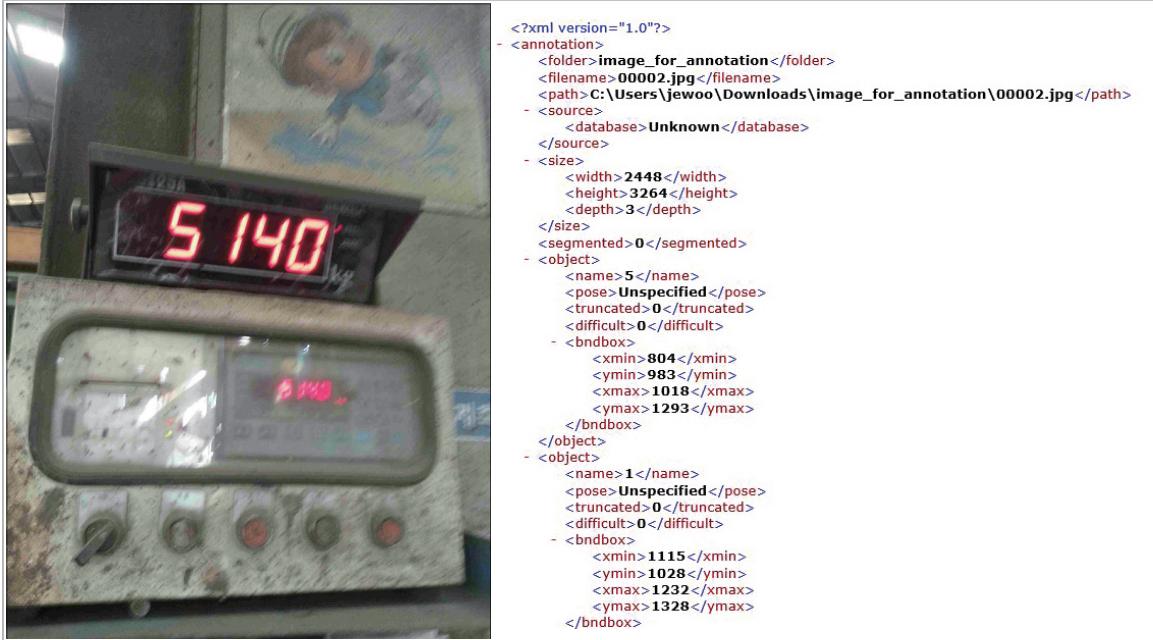
2.1 제조데이터 소개

1) 데이터 수집 방법

- 제조 분야 : 제관 카운터웨이트(Counter-Weight)
- 제조 공정명 : 충진공정
- 수집장비 : 무게 계량 저울의 계측 표기를 스마트폰으로 촬영후 RPMS를 통해 업로드
- 수집 기간(주기) : 모든 생산품에 대해서 수집, 1년간 수집결과

2) 데이터 유형/구조

- 데이터셋 구조 : 원본(3,894매의 JPG 이미지)로 부터 추출된 27,237개의 JPG 이미지 데이터
- [그림2]의 XML 파일은 원본이미지로부터 이미지의 크기, bndbox에 검출된 숫자에 대응되는 위치와 숫자의 라벨을 포함하고 있다.



[그림 2] (좌) 이미지 데이터의 사례, (우) Bounding Box가 포함된 Digit 판정 Annotation

2.2 분석 모델 소개

1) AI 모델의 목적

- 딥 뉴럴 네트워크를 이용한 숫자 분류 모델
- 입력으로 숫자가 있는 이미지 데이터를 받아 0 부터 9 사이의 10 개의 숫자 중 하나로 분류한다.
- 이미지 처리에 적합한 것으로 알려진 컨벌루션 뉴럴 네트워크¹(convolutional neural network, CNN)을 사용하였다.

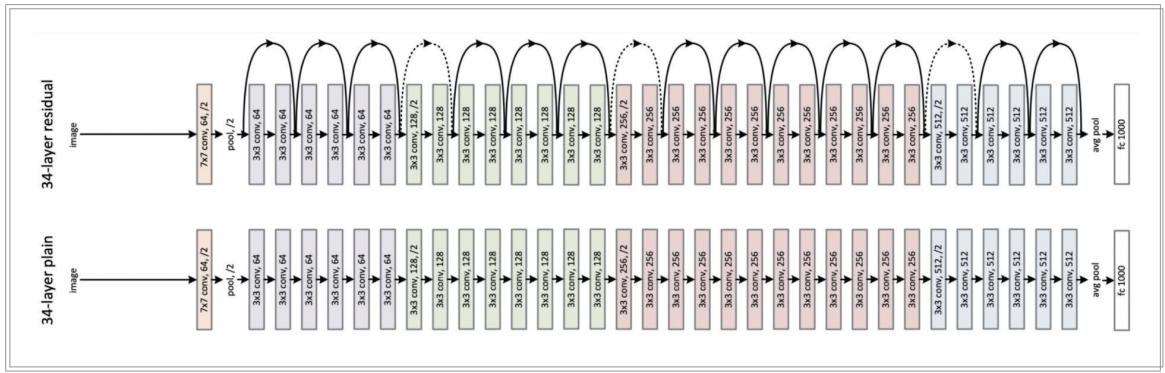
2) 딥 뉴럴 네트워크 구조

- 역전파²(Backpropagation)로 학습되는 뉴럴 네트워크의 특성상 레이어가 많은 뉴럴 네트워크의 경우 출력으로부터 얻은 구배(gradients)가 입력에 가까운 층의 레이어까지 잘 전파되지 않아 학습에 어려움이 있다.
- 깊은 뉴럴 네트워크의 학습을 위하여 레이어와 레이어를 건너 뛰어 연결 시켜주는 레지 듀얼 연결(residual connection)을 추가한 Residual Neural Network (ResNet) 구조가 효율적으로 알려져 있다.

1) Convolutional neural network (CNN)는 뉴럴 네트워크 구조의 한 종류로 이미지 처리 또는 자연어 처리에 주로 사용된다. CNN의 가장 큰 특징은 위치 정보를 보존할 수 있다는 것으로 특정 크기의 필터가 서로 다른 위치에 적용되어 다른 위치에 대해서도 같은 연산이 적용된다.

2) 역전파라고 불리며 학습 과정에서 라벨 정보를 통해 계산된 손실함수를 최소화할 때 미분값을 손실함수부터 입력 지점까지 역전파하여 계산 하는 방법이다.

- 본 분석에서도 이미지 처리를 위해 ResNet 구조를 사용하였다.
- ResNet의 구조



[그림 3] ResNet의 구조. 일반적인 뉴럴 네트워크와 달리 연속된 레이어를 뛰어 넘어 연결시켜주는
레지듀얼 연결(residual connection)이 존재

- ResNet은 크게 입력 이미지를 처리하는 컨벌루션 레이어³(convolutional layer), 중간 레지듀얼 연결(residual connection)으로 이루어진 서로 다른 크기의 피쳐(feature)를 처리하는 여러 레이어로 이루어진 블록⁴, 마지막으로 예측에 사용되는 완전 연결 레이어⁵(fully connected layer)로 크게 3가지로 나누어 생각할 수 있다.
- 총 블록의 수가 3개인 ResNet의 경우 레지듀얼 연결(residual connection)으로 처리되는 각각의 블록의 레이어 수에 따라 총 ResNet의 레이어 수가 결정되며 각 블록의 레이어 수가 n개일 때 총 레이어의 수는 $(3n+2)$ 개가 된다.
- 본 분석에서는 레이어 수를 8, 14, 20개로 나누어 사용하였으며, 일반적으로 레이어의 수가 많을 때 성능이 좋아지지만 실제 적용을 할 때 사용되는 계산량이 더 요구된다는 상충 관계가 있다.

3) 학습 방법

- 뉴럴 네트워크의 데이터를 가장 잘 설명할 수 있는 최적 파라미터는 수학적으로 바로 계산을 할 수 없기 때문에 손실 함수를 줄이는 방향으로 경사 하강법⁶(gradient descent)을 사용하여 뉴럴 네트워크를 학습하였다.
- 경사 하강법(gradient descent)은 손실 함수의 기울기를 이용하여 손실 함수를 줄이는 방향으로 파라미터를 학습하는 방법이며 전역적인 최적해를 구할 수 있음을 보장되지 않는다.

3) Convolutional operation이 일어나는 레이어를 의미한다.

4) 일반적인 CNN 구조에서 같은 크기의 레이어를 처리하는 연산을 반복하여 쌓은 레이어를 묶어 블록이라 부른다.

5) CNN과 달리 모든 뉴런 간의 연결이 되어 있는 레이어이다.

6) 특정 지점에서 미분을 통해 손실 함수를 최소화하는 방향으로 파라미터를 업데이트 하는 방법이다.

- 경사 하강법(gradient descent)을 실행할 때 전체 학습 데이터를 사용하는 것은 배치 경사 하강법(batch gradient descent)라고 하며 많은 계산량을 요구한다는 단점이 있다.
- 따라서 본 분석에서는 일부 데이터(mini-batch)를 사용하여 경사 하강법(gradient descent)을 수행하는 확률적 경사 하강법(stochastic gradient descent) 알고리즘을 사용하여 뉴럴 네트워크를 학습하였다.
- 구배(gradient) 값을 사용하여 뉴럴 네트워크의 파라미터를 업데이트를 해줄지의 정도를 결정하는 것 또한 중요하다.
- 뉴럴 네트워크의 파라미터 업데이트의 정도를 결정하는 파라미터를 학습률(learning rate)이라고 하며 일반적으로 $0.1 \sim 0.0001$ 사이의 값을 사용한다.
- 미분 값을 사용하여 어떻게 뉴럴 네트워크의 파라미터를 업데이트를 할지 결정하는 옵티마이저(optimizer)의 설계 또한 학습에 영향을 많이 미치며, 이때까지 학습에 사용된 구배(gradient)의 경향성을 사용하는 모멘텀(momentum)을 사용하는 것이 효과적이라고 알려져 있다.
- 아담 옵티마이저(adam optimizer)는 학습이 진행되며, 학습경향에 따라 구배(gradient)의 크기를 줄여 안정적인 학습을 유도한다.
- 본 분석에서는 모멘텀(momentum)과 확률적 경사 하강법(stochastic gradient descent) 자체를 그대로 사용하는 옵티마이저(optimizer)와 아담 옵티마이저(Adam optimizer)를 적용하였다.
- 학습이 진행됨에 따라 학습률(learning rate)을 감소시켜줄 때 일반적으로 학습이 안정적으로 되며 학습률(learning rate)을 감소시켜주는 학습 스케줄을 지수함수, 코사인(cosine) 함수 등을 이용하여 모델링 가능하다.
- 본 분석에서는 다양한 학습률(learning rate), 옵티마이저(optimizer), 학습 스케줄을 적용하여 최적 학습 방법을 실험적으로 찾는 것을 목표로 한다.

4) 알고리즘 구축 절차

- 학습에 사용할 뉴럴 네트워크의 구조 설정한다. (예시: ResNet8, ResNet14, ResNet20)
- 입력으로 넣을 이미지의 크기 설정한다.
- 컴퓨터의 CPU, GPU 및 결과 분석을 통해 한 번의 반복(iteration)에 사용될 이미지의 수인 배치(batch)의 크기 설정한다.
- 학습 데이터를 몇 번 반복하여 뉴럴 네트워크를 학습할지 결정하는 에포크(epoch)를 설정한다.

- 최적화에 사용할 옵티마이저(optimizer)를 설정한다.
- 구배(gradient) 값을 뉴럴 네트워크의 파라미터를 얼마나 업데이트 해줄지 결정하는 학습률(learning rate) 및 학습 스케줄 설정한다.
- 학습에 사용되는 파라미터를 설정하여 뉴럴 네트워크의 학습을 진행하며 검증(validation)을 통해 최적 파라미터를 찾아 최적의 뉴럴 네트워크 구축한다.

2.3 분석 체험

1) 필요 SW, 패키지 설치 방법 및 절차 가이드

- python: anaconda 설치를 통해 여러 라이브러리를 함께 다운로드 가능하다. (설치 링크: <https://www.anaconda.com/products/individual>)
- 그래픽 카드 드라이버 다운로드: nvidia 드라이버 사이트에 접속하여 그래픽 카드에 맞는 CUDA 설치할 수 있다. (nvidia 드라이버 다운로드 링크: <https://www.nvidia.com/Download/index.aspx?lang=kr>)
- Pytorch 설치: pytorch 공식 사이트에서 OS, CUDA 버전에 맞추어 커멘드를 받아 설치하며 터미널 창에 run this command를 복사하여 실행할 수 있다. (pytorch 공식 사이트: <https://pytorch.org/>)
- 자세한 설치 방법은 부록 참고



[그림 4] OS, CUDA 버전에 맞추어 클릭하면 그에 알맞은 커멘드가 생성되며 이를 복사하여 pytorch 설치 가능하다.
CUDA 버전을 모르는 경우 터미널 창에 nvcc --version을 입력하여 확인 가능하다.

2) 실습 방법

- 실습 코드와 학습 데이터를 다운로드하여 해당 디렉토리로 이동한다.
- 해당 디렉토리에서 터미널을 열고 학습에 사용할 파라미터를 설정하여 학습 진행 가능하다.

- main.py 파일을 알맞은 설정에 따라 아래와 같은 커멘드를 터미널 창에서 python으로 실행하여 학습 진행 가능하다.

```
python main.py --model resnet8 --data_dir PATH_TO_DATA --size 64
--batch_size 128 --epochs 100 --optimizer sgd --lr 0.1 --scheduler cosine
```

- 각 파라미터는 “--파라미터명”과 “파라미터”로 띄어쓰기를 통해 설정할 수 있다.
- main.py 코드의 main 함수 안에서 입력된 파라미터를 파싱하는 코드가 구현되어 있다.

```
def main():
    ...
    ...
    argparse는 터미널 창에서 입력을 파싱할 수 있게 도와줌
    예를 들어 main 함수에서 설정하는 변수인 model을 직접 코드를 수정하지 않고 터미널에서 코드 실행시에
    python main.py --model XXX 형태로 실행하면 model을 XXX로 결정할 수 있음

    변수 종류
    model: 뉴럴 네트워크 구조를 의미
        group1 - resnet8, resnet14, resnet20, resnet32, resnet44, resnet56, resnet110, resnet8x4, resnet32x4
        group2 - resnet18, resnet34, resnet50, resnet101, resnet152
        group2의 parameter가 일반적으로 많고 성능이 좋지만 학습에 시간이 오래걸림, 숫자는 layer 수를 의미하며 layer
        수가 많아질수록 일반적으로 성능이 좋아짐
    data_dir: digit_data 폴더가 있는 경로
    size: 입력 이미지의 크기, 크기가 클수록 성능이 좋아질 수 있지만, 연산량이 커지기 때문에 적절한 크기 세팅 필요
        (default: 64)
    batch_size: 한 번의 iteration에 사용되는 data instance의 수
    epochs: dataset 전체를 몇 번 사용하여 학습할 지 결정하는 요소로 1 epoch에 전체 dataset의 모든 instance를 1회 사용
    optimizer: 뉴럴 네트워크의 parameter를 학습시키는 방법을 결정 (sgd, adam)
    lr: 학습을 처음 시작할 때 결정하는 learning rate의 값. 한 번의 iteration에서 parameter를 얼마나 update 시킬지
        결정하는 값 클수록 한 번에 크게 parameter를 update 하지만, learning rate가 너무 큰 경우 학습이 잘 안될 수 있음
    scheduler: learning rate를 조절하는 스케줄을 결정 (cosine, poly, exponential)
    weight_decay: 모델이 training data에 과적합되는 것을 막기 위해 정규화시키는 정도를 결정

    터미널 창 사용 예시
    python main.py --model resnet8 --data_dir ../../digit_data --size 64 --batch_size 128 --epochs 100
    --optimizer adam --lr 0.001 --scheduler cosine --weight_decay 5e-4
    위와 같이 띄어쓰기로 argument 구분
    ...
    ...
    parser = argparse.ArgumentParser()
    parser.add_argument('--model', default='resnet8', help='network architecture, resnet is only implemented')
    parser.add_argument('--data_dir', default='../../digit_data', type=str, help='digit data directory')
    parser.add_argument('--size', default=64, type=int, help='size of resized image')
    parser.add_argument('--batch_size', default=128, type=int, help='batch size')

    parser.add_argument('--epochs', default=100, type=int, help='number of training epochs')
    parser.add_argument('--optimizer', default='adam', type=str, help='optimizer')
    parser.add_argument('--lr', default=0.001, type=float, help='learning rate')
    parser.add_argument('--scheduler', default='cosine', type=str, help='learning rate scheduler, cosine, poly ...')
    parser.add_argument('--weight_decay', default=5e-4, type=float, help='weight decay for l2 regularization')

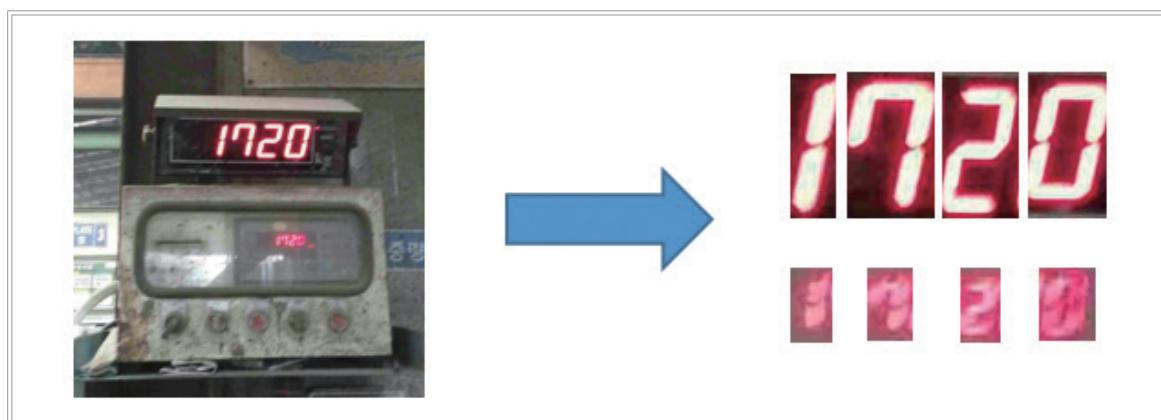
    args = parser.parse_args() # argument를 parsing
```

[그림 5] main.py

- model은 뉴럴 네트워크의 구조로 resnet8, resnet14, resnet20 등을 사용 가능하며, data_dir은 데이터를 저장한 디렉토리로 설정이 필요하다. size는 뉴럴 네트워크의 입력으로 넣을 이미지의 크기 조정을 위한 가로, 세로의 길이이며 본 분석에서는 모두 64로 통일하여 사용하였다. batch_size는 한 번의 반복(iteration)에 사용할 이미지의 수를 의미한다. epochs는 총 학습 데이터를 반복하여서 볼 횟수를 의미한다. optimizer는 확률적 경사 하강법(stochastic gradient descent)을 사용할 경우 sgd, 아담 옵티마이저(Adam optimizer)를 사용할 경우 adam으로 설정 가능하다. lr은 학습률(learning rate)을 의미한다. scheduler는 학습 스케줄 방법을 의미하며, poly, cosine, exponential으로 설정 가능하다.

3) 학습 데이터

- 공장에서 계측된 숫자 원본데이터 (3,894매/JPG 파일)을 사람이 각각의 숫자와 위치와 종류를 라벨링하여 XML 파일로 표현하였고, XML을 바탕으로 추출한 이미지를 digit_data로 저장하였다.



[그림 6] 학습에 사용된 숫자의 예시이며 원쪽의 원본 이미지에서 각 숫자의 위치를 라벨링하여 총 8개의 숫자 이미지를 생성

- 원본 이미지에서 얻은 숫자의 이미지는 각각 어떤 종류인지 라벨링이 되어 있다. (예시. 그림 6의 우측의 숫자의 경우 좌측상단부터 우측하단까지 각각 1, 7, 2, 0, 1, 7, 2, 0으로 라벨링이 되어 있다.)
- 본 가이드북에서는 추출된 digit_data인 27,237개의 이미지를 학습에 사용하였다.

4) 분석 단계별 Process



[그림 7] Flow Chart

- 부록의 classification.ipynb 참고
- Anaconda prompt 창에서 jupyter notebook을 실행하여 첨부된 코드 파일 내의 노트북 파일을 통해 전반적인 학습 프로세스를 볼 수 있다.

[단계 ①] 라이브러리 불러오기

①-1. 필요 라이브러리 불러오기

```
import os
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import torch
import torchvision
import torch.nn as nn
import torch.optim as optim
from torchvision import transforms
from torch.utils.data.dataloader import DataLoader
import models # models 폴더 내의 코드를 import
```

[그림 8] 패키지 import 예시

- 필요 라이브러리를 불러온다. 이때 models 폴더 내에 구현된 모델 클래스 또한 함께 불러온다.

[단계 ②] 데이터 로더 생성 및 데이터 확인

②-1. 데이터 로더 생성

```
class DigitData:
    def __init__(self, path, size=64, split='train'):
        self.path = path
        self.size = (size, size)

        # training set과 validation set 구분
        if split == 'train':
            self.image_files = open(os.path.join(path, 'train_data.txt'), 'r').read().splitlines()
        else:
            self.image_files = open(os.path.join(path, 'valid_data.txt'), 'r').read().splitlines()

        # 전체 데이터셋의 RGB 평균과 표준편차
        mean = [0.80048384, 0.44734452, 0.50106468]
        std = [0.22327253, 0.29523788, 0.24583565]
        self.transform = transforms.Compose([transforms.Resize(self.size), transforms.ToTensor(),
                                             transforms.Normalize(mean=mean, std=std)])

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

    def __getitem__(self, idx):
        path = os.path.join(self.path, self.image_files[idx])
        img = Image.open(path).convert('RGB')
        img = self.transform(img)
        target = int(self.image_files[idx].split('/')[0])
        return img, target

path = '../../../../../digit_data' # digit_data가 있는 디렉토리
size = 64 # 이미지의 조정된 크기
batch_size = 128 # 한 번의 iteration에 사용할 instance의 수

train_data = DigitData(path, size, 'train')
valid_data = DigitData(path, size, 'valid')
train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
valid_loader = DataLoader(valid_data, batch_size=batch_size, shuffle=True)
```

[그림 9] Data loader 생성

- digit_data를 pytorch에서 실행하는 형태로 처리하여 모델 학습을 하기 위하여 pytorch의 데이터 로더(data loader)를 생성한다.

②-2. 데이터 확인

```
# 데이터의 수
print('# instances of train: {}'.format(len(train_data)))
print('# instances of validation: {}'.format(len(valid_data)))

# instances of train: 21789
# instances of validation: 5448
```

[그림 10] : 데이터 수 확인

- 데이터의 로딩이 잘 되었는지 로딩된 데이터의 수를 통해 확인할 수 있다.

[단계 ③] 데이터 시각화

③-1. 데이터 시각화

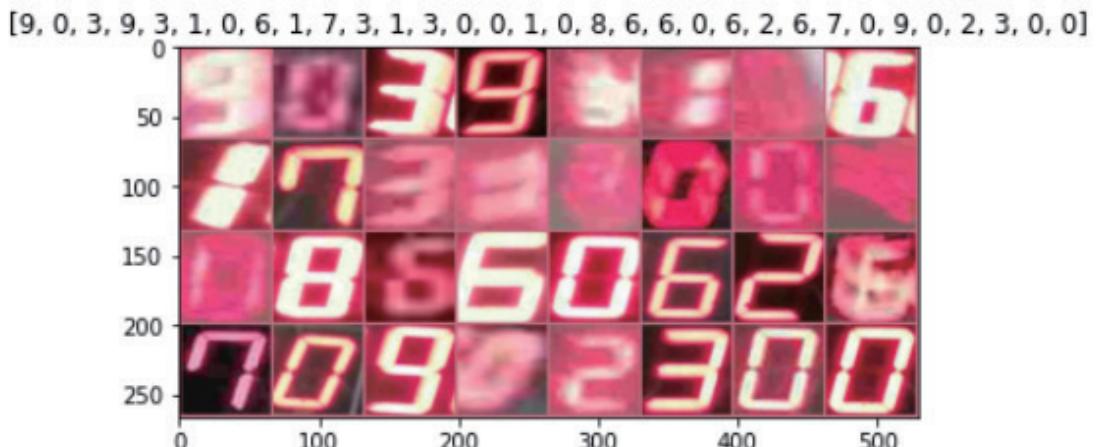
```
def imshow(inp, title=None):
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.80048384, 0.44734452, 0.50106468])
    std = np.array([0.22327253, 0.29523788, 0.24583565])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)
    if title is not None:
        plt.title(title)
    plt.pause(0.001)

inputs, classes = next(iter(train_loader))
out = torchvision.utils.make_grid(inputs[:32])

imshow(out, title=[i.item() for i in classes[:32]])
```

[그림 11] 데이터 시각화를 위한 코드

- 그림 11의 코드는 학습 데이터의 시각화를 위한 함수 코드이다.



[그림 12] 학습 데이터의 예시

- 학습 데이터의 시각화 결과이다.

[단계 ④] 모델 학습

④-1. 파라미터 세팅 및 성능 측정을 위한 함수 구현

```
model = 'resnet20' # resnet 구조 사용 가능, resnet8, resnet14, ... or resnet18, resnet34, resnet50 ...
optimizer = 'sgd' # sgd or adam
scheduler = 'cosine' # cosine, poly, exponential
learning_rate = 0.1 # learning rate
weight_decay = 0.0005 # l2 regularization weight
epochs = 100 # dataset 전체를 학습에 사용하는 횟수

model = models.__dict__[model](num_classes=10)
criterion = nn.CrossEntropyLoss() # loss 설정

if optimizer == 'sgd':
    optimizer = optim.SGD(model.parameters(), lr=learning_rate, momentum=0.9, weight_decay=weight_decay)
elif optimizer == 'adam':
    optimizer = optim.Adam(model.parameters(), lr=learning_rate, weight_decay=weight_decay)

if scheduler == 'cosine':
    scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, epochs)
elif scheduler == 'poly':
    scheduler = optim.lr_scheduler.LambdaLR(optimizer, lambda x: (learning_rate * (1.0 - x / epochs) ** 0.9))
elif scheduler == 'exponential':
    scheduler = optim.lr_scheduler.ExponentialLR(optimizer, 0.99)

is_cuda = torch.cuda.is_available() # gpu 환경이 구축된 경우 gpu를 사용하여 학습 진행
if is_cuda:
    model.cuda()
```

[그림 13] 학습 세팅 예시 코드

- 학습 세팅을 위한 코드로 main.py에서 변수(argument)를 분석(parsing)하는 것을 직접 지정한 것이다. 아래의 코드는 main.py 코드로 학습 실행 시에는 터미널 창에서 지정되는 파라미터 등을 직접 작성한 예시이다.
- 아래의 코드는 평가를 위한 객체로 초기화(reset) 매소드(method)를 통해 각 속성(attribute)을 초기화하고, update 매소드(method)를 통해 각 속성(attribute)을 학습 또는 테스트 시에 업데이트한다.

```
class AvgMeter(object):
    # 성능 측정을 위한 객체
    def __init__(self):
        self.reset()

    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count
```

[그림 14] 성능 측정을 위한 클래스

- 아래의 함수를 통해 정확도를 측정 가능하다.

```

def cal_acc(outputs, targets):
    # 정답률 확인을 위한
    _, pred = torch.max(outputs.data, 1)
    correct = pred.eq(targets).sum().item()
    acc = correct / targets.size(0)
    return acc

```

[그림 15] 정확도 측정을 위한 함수

④-2. 모델 학습 및 검증

- 아래의 함수는 학습을 위한 것으로 데이터셋 전체를 한 번 사용하는 1 에포크(epoch)의 학습이 이루어진다.

```

def train(model, data_loader, criterion, optimizer, cuda):
    """
    모델의 학습을 위한 함수
    model: 학습에 사용되는 뉴럴 네트워크 모델
    data_loader: training data를 불러오는 객체
    criterion: loss function으로 cross entropy 사용
    optimizer: loss function에 따라 model의 parameter를 업데이트
    cuda: GPU 사용 여부
    """

    model.train() # model을 training setting으로 교체

    # loss, accuracy 기록 (AvgMeter class 참고)
    train_loss = AvgMeter()
    train_acc = AvgMeter()
    for imgs, targets in data_loader:
        if cuda: # GPU를 사용하는 경우 data_loader의 출력을 CPU에서 GPU로 옮겨줌
            imgs, targets = imgs.cuda(), targets.cuda()
        outputs = model(imgs) # model 예측
        loss = criterion(outputs, targets) # loss 계산

        # model parameter update
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # loss, accuracy 기록
        train_loss.update(loss.item(), targets.size(0))
        train_acc.update(cal_acc(outputs, targets), targets.size(0))
    return train_loss.avg, train_acc.avg

```

[그림 16] 에포크(epoch) 학습 과정을 위한 함수

- 아래의 함수는 테스트를 위한 것으로 검증(validation) 데이터에 대해 한 정확도를 계산한다.

```

def valid(model, data_loader, cuda):
    ...
    모델의 평가를 위한 함수
    model: 평가에 사용되는 뉴럴 네트워크 모델
    cuda: GPU 사용 여부
    ...
    model.eval() # model을 validation을 위한 상태로 변경

    valid_acc = AvgMeter() # accuracy 기록 (AvgMeter class 참고)

    for imgs, targets in data_loader:
        if cuda: # GPU를 사용하는 경우 data_loader의 출력을 CPU에서 GPU로 옮겨줄
            imgs, targets = imgs.cuda(), targets.cuda()
        with torch.no_grad(): # validation 시 gradient 계산 불필요
            outputs = model(imgs) # model 예측
        valid_acc.update(cal_acc(outputs, targets), targets.size(0)) # accuracy 기록
    return valid_acc.avg

```

[그림 17] 검증(validation) 데이터에 대한 정확도를 측정하기 위한 함수

- 지정된 세팅에 따라 학습 진행 및 학습 진행 상황을 출력(print)하여 현재 상황에 대한 표기를 해주는 코드이다.

```

# 학습 진행
train_loss_list = []
train_acc_list = []
valid_acc_list = []
for epoch in range(epochs):
    train_loss, train_acc = train(model, train_loader, criterion, optimizer, is_cuda)
    scheduler.step()
    valid_acc = valid(model, valid_loader, is_cuda)
    train_loss_list.append(train_loss)
    train_acc_list.append(train_acc)
    valid_acc_list.append(valid_acc)
    print('Epoch: {0:>3d} | Train loss: {1:0.4f} | Train Acc: {2:0.4f} | Valid Acc: {3:0.4f}'.
          format(epoch + 1, train_loss, train_acc, valid_acc))

```

[그림 18] 학습 진행 상황을 표기하기 위한 코드

- 아래 그림 19는 잘못 분류된 이미지가 무엇인지 확인하기 위한 코드이다.

```

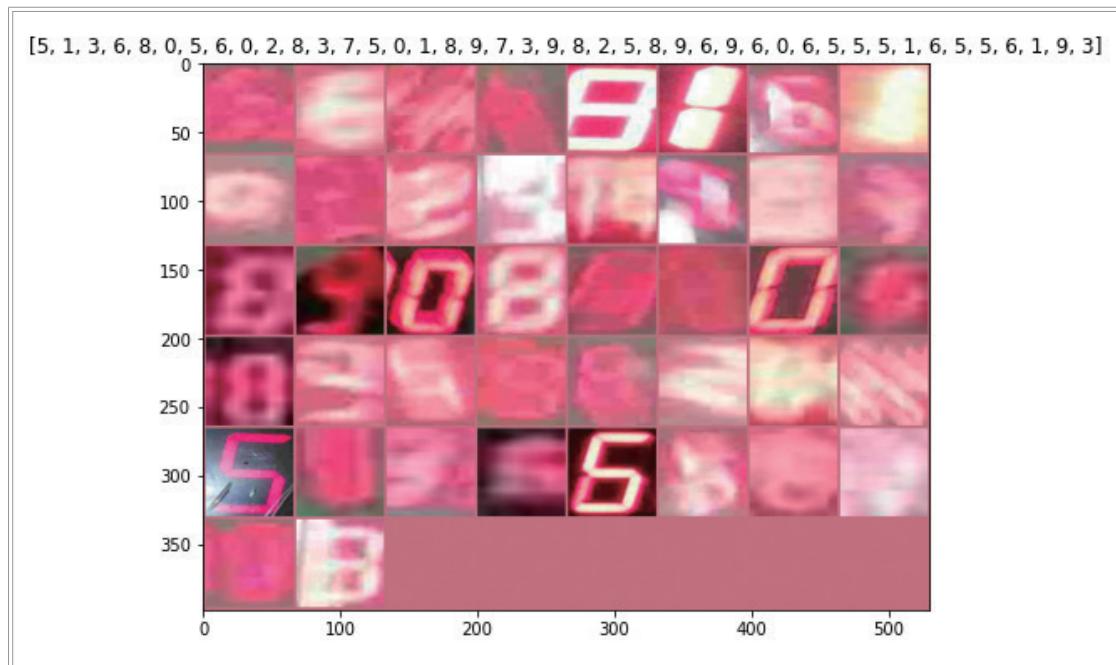
error_imgs = []
error_classes = []
for imgs, targets in valid_loader:
    if is_cuda:
        imgs, targets = imgs.cuda(), targets.cuda()
    with torch.no_grad():
        _, pred = torch.max(model(imgs), 1)
    for idx, (p, t) in enumerate(zip(pred, targets)):
        if not p == t:
            error_imgs.append(imgs[idx])
            error_classes.append(t)

out = torchvision.utils.make_grid(torch.stack(error_imgs, 0))
plt.figure(figsize = (10, 6))
imshow(out.cpu(), title=[i.item() for i in error_classes])

```

[그림 19] 오류 분석을 위한 코드

- 아래의 그림 20은 학습된 모델이 검증 데이터(validation data) 중 잘못 분류한 데이터의 예시이다.

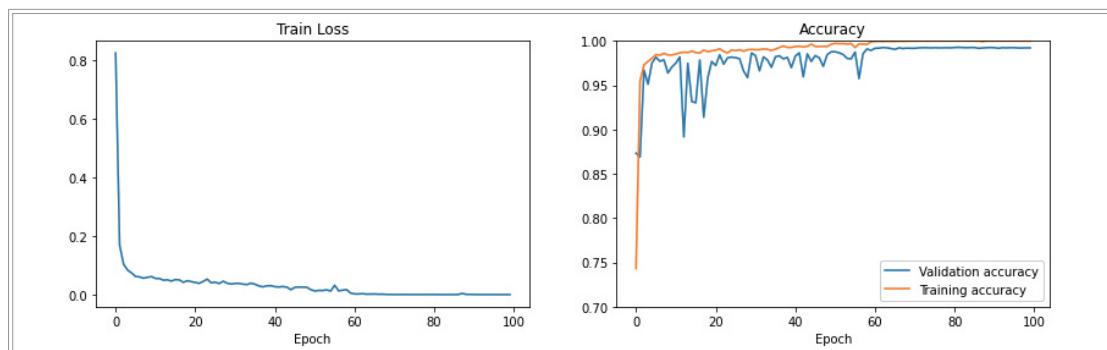


- 아래의 그림 21은 에포크(epoch)에 따른 손실 함수(train loss)의 감소와 학습, 검증 데이터(validation data)에 대한 정확도 변화를 시각화하기 위한 코드와 예시이다.

```
plt.rcParams["figure.figsize"] = (14,4)
ax1 = plt.subplot(121)
ax1.set_title('Train Loss')
ax1.plot(train_loss_list)
ax1.set_xlabel('Epoch')

ax2 = plt.subplot(122)
ax2.set_ylim([0.7,1])
ax2.set_title('Accuracy')
ax2.set_xlabel('Epoch')

ax2.plot(valid_acc_list, label='Validation accuracy')
ax2.plot(train_acc_list, label='Training accuracy')
plt.legend()
plt.show()
```

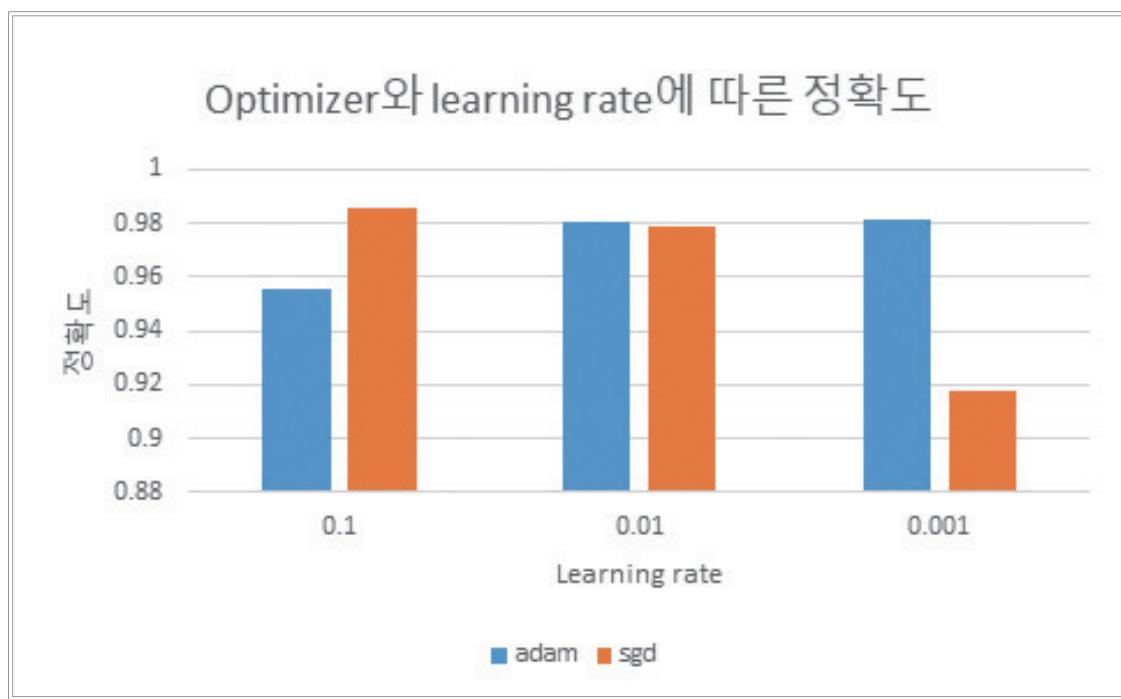


[그림 21] 학습에 따른 손실 함수와 정확도의 변화로, 결과에 따르면 에포크(epoch)가 증가함에 따라 손실함수는 감소하고, 정확도는 높아지는 것을 확인 할 수 있다.

[단계 ⑤] 결과 분석 및 해석

⑤-1. optimizer와 학습률(learning rate)에 따른 성능 분석

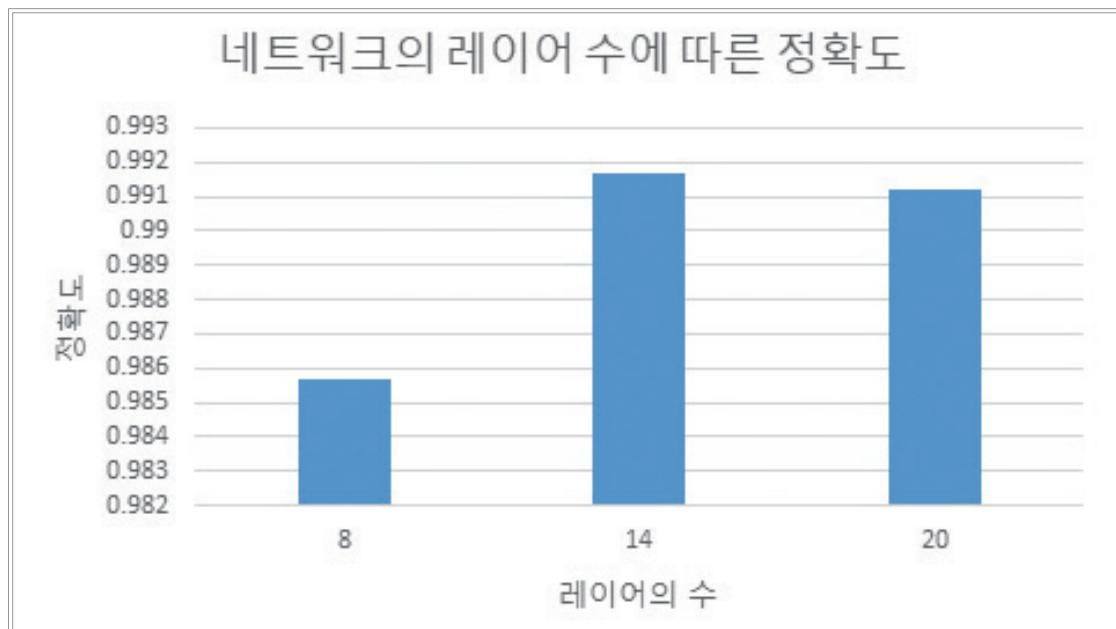
- optimizer와 학습률(learning rate) 외의 학습에 사용되는 파라미터는 고정하여 최적 옵티마이저(optimizer)와 학습률(learning rate)을 찾기 위한 실험을 진행하였다.
- 실험 결과 아담 옵티마이저(adam optimizer)보다 확률적 경사 하강법(stochastic gradient descent)을 사용한 옵티마이저(optimizer)가 더 적합함을 알 수 있었으며, 확률적 경사 하강법(stochastic gradient descent)을 사용한 경우 학습률(learning rate)을 0.1으로 둘 때 성능이 더 좋음을 확인 가능하였다.



[그림 24] x축은 학습률(learning rate)을 의미하며 y축은 검증(validation) 데이터에 대한 정확도를 의미함.
확률적 경사 하강법(stochastic gradient descent)을 학습률(learning rate) 0.1로 두고 학습하였을 때 최적임을 알 수 있음

⑤-2. 네트워크 크기에 따른 성능 분석

- 네트워크의 크기가 증가함에 따라 성능 향상의 정도를 확인하기 위하여 실험을 진행하였다.
- 아래의 그림 23을 통해 레이어의 수가 8개일 때보다 14개일 때 성능 향상이 큰 것을 확인할 수 있지만, 레이어의 수가 20개일 때의 성능이 오히려 14개일 때보다 더 나쁜 것을 확인할 수 있었다.

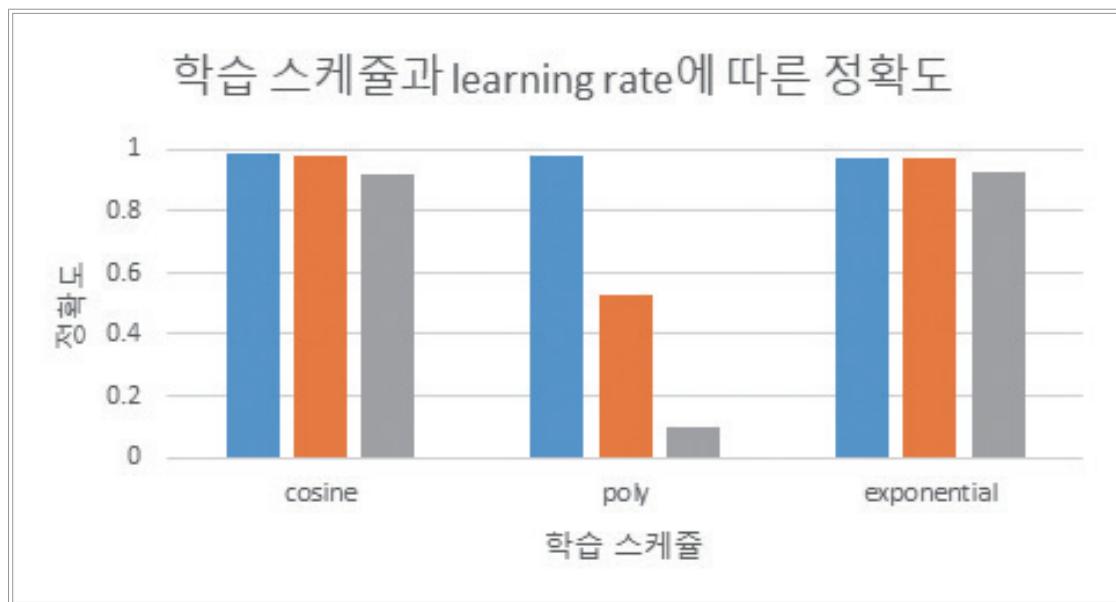


[그림 25] 레이어의 수에 따른 검증(validation) 데이터에 대한 정확도

- 하지만 그 차이가 매우 작기 때문에 ($0.05\% p$ 차이) 레이어의 개수가 14개가 최적이라기보다 레이어의 개수를 14개 이상으로 늘리는 것이 성능 향상에 도움이 되지 않을 것이라 해석할 수 있다.

⑤-3. 학습 스케줄에 따른 성능 분석

- 학습률(learning rate)을 조절하는 학습 스케줄 또한 성능에 영향을 미치는 정도가 크며, 이를 최적화할 필요성이 있다.
- 아래의 그림 24를 봤을 때 학습 스케줄에 따른 성능 차이가 있음을 볼 수 있으며, 학습률(learning rate)에 따라 성능 차이가 나는 것을 확인할 수 있다.

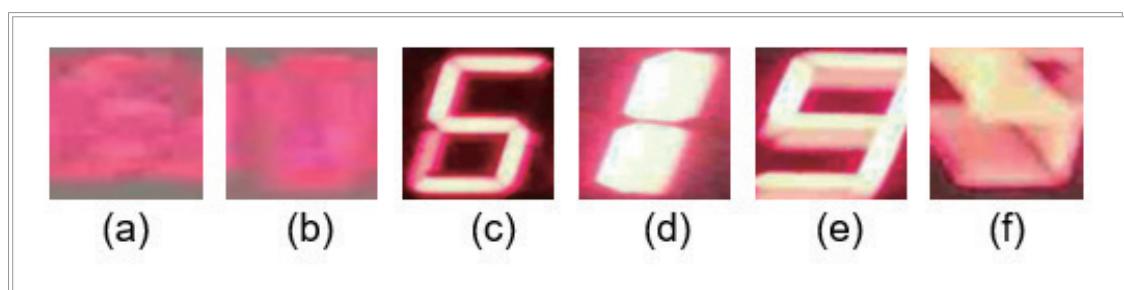


[그림 26] 학습 스케줄과 학습률(learning rate)에 따른 정확도

- 특히 poly 학습 스케줄의 경우 학습률(learning rate)의 크기가 성능에 큰 영향을 줌을 확인 가능하다.
- 위의 결과를 통해 cosine 학습 스케줄이 최적 학습률(learning rate) 조절 스케줄임을 확인할 수 있다.

⑤-4. 실험 결과 및 분석

- 레이어가 20개인 ResNet 구조와 확률적 경사 하강법(stochastic gradient descent) 알고리즘을 모멘텀(momentum)을 주어 사용한 옵티마이저(optimizer) 그리고 학습률(learning rate)을 0.1으로 코사인(cosine) 학습 스케줄링으로 학습 진행하였다.
- 배치(batch)의 크기는 128, 에포크(epoch) 수는 100으로 입력 이미지의 크기는 64x64로 두고 학습하였다.
- 검증(validation) 데이터에서 학습한 네트워크가 제대로 인식하지 못한 숫자의 경우 그림 25와 같이 실제 알아보기 힘든 경우가 많았다.



[그림 27] 인식에 실패한 숫자의 예시. (a),(b)의 경우 육안으로 인식이 매우 힘들 정도로 숫 흔들림이 심하여 인식이 어렵다. (c)의 경우 실제 숫자는 6이지만 희미한 영역으로 인해 5로 잘못 인식되었다. (d)의 경우 숫자는 10이지만 흔들림으로 인해 0으로 인식되었다. (e)의 경우 오른쪽 가장 자리가 잘려 실제 숫자는 9이지만 5로 인식되었다. (f)의 경우 흔들림으로 인해 5를 3으로 인식되었다.

- 20개의 레이어로 이루어진 ResNet20 구조에서 검증(validation) 데이터를 99% 이상의 정확도로 분류할 수 있음을 확인할 수 있었다.

3. 유사 태현장의 「제조 현장용 OCR학습 AI 데이터셋」 분석 적용

3.1 본 분석이 적용 가능한 제조현장 소개

- 많은 현장에서 센서와 클라우드와의 연계가 쉽지 않은 상황이다. 특히 레가시 센서가 많은 상황이고, 이것이 숫자 등으로 현장에서 표출되는 상황이라면, OCR의 활용성은 매우 높을 수 있다. 즉, 레가시 센서가 많은 제조현장에 본 분석은 적용 가능하다.

3.2 본 「제조 현장용 OCR학습 AI 데이터셋」 분석을 원용하여 타 제조현장 적용 시, 주요 고려사항

- OCR의 표출 폰트, 색상, 위치, 뒤틀림에 따라 학습의 난이도가 달라질 수 있으며, 입력되는 이미지의 흔들림, 각도, 조도 등에 따라서도 학습 성능이 달라질 수 있다. 각 현장에 맞는 OCR 커스터마이즈가 필요한 사항이다.

1. 소프트웨어 설치 방법

◎ Python

- Anaconda 설치를 통한 Python 설치한다.
- 링크를 통해 Anaconda 설치 사이트 접속한다. (<https://www.anaconda.com/products/individual#windows>)
- 컴퓨터 사양에 맞는 프로그램 설치한다. (내PC – 속성 – 시스템 종류를 통해 운영 체제 버트 수에 맞는 프로그램을 설치한다.)



[그림 1] Anaconda 설치 웹사이트

- 설치 후 아래와 같이 Anaconda prompt를 통해 python 실행 가능하다.

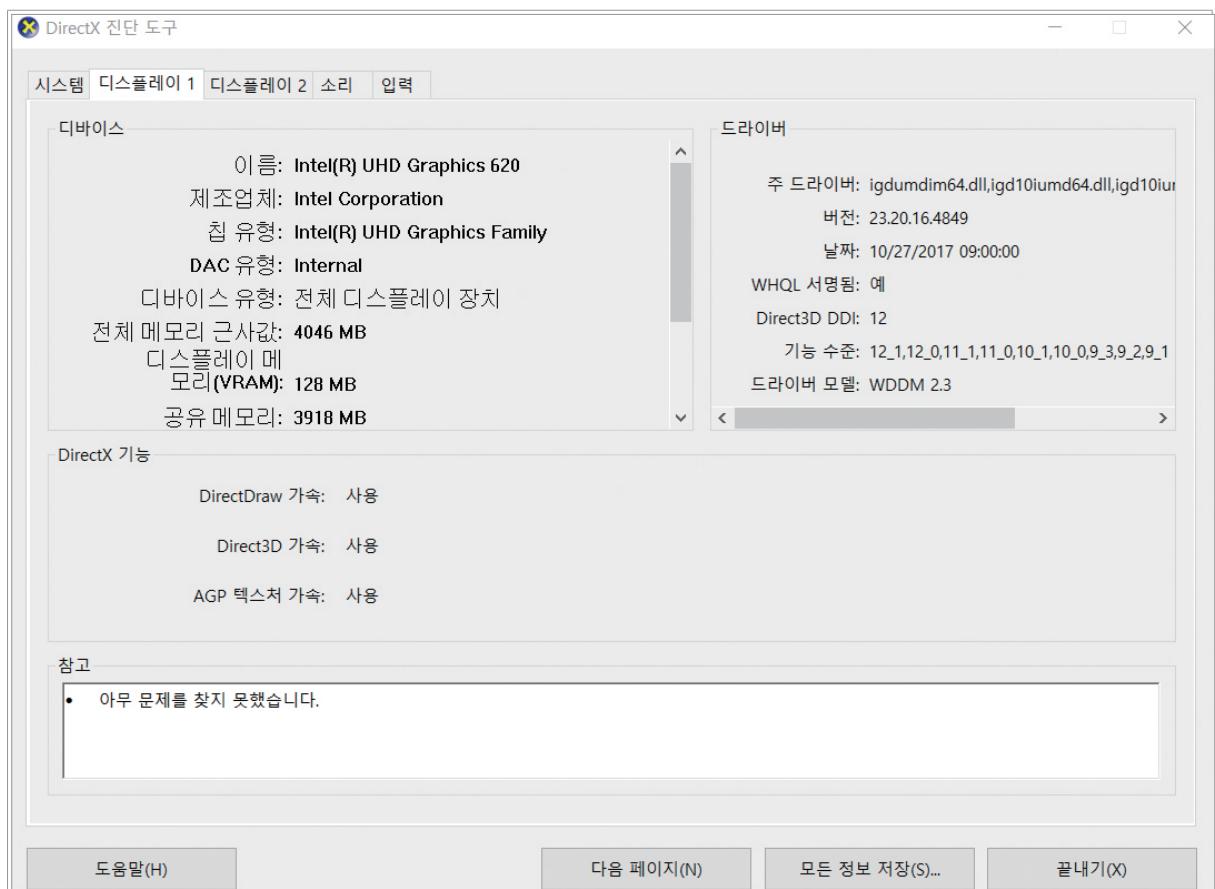
The screenshot shows the Anaconda Prompt window titled 'Anaconda Prompt - python'. The terminal output shows the Python 3.6.2 interpreter starting at '(base) C:\Users\WAIILab-gram1>python'. It displays the standard Python help message and ends with '>>> S'.

[그림 2] python 실행 예시

- 새로운 패키지의 경우 Anaconda prompt 창에서 “pip install xxxx” 형태로 설치 가능하다.
- requirements.txt 형태로 설치 패키지가 제공되는 경우에 “pip install -r requirements.txt”로 패키지 설치 가능하다.

● CUDA

- GPU 사용을 위해 NVIDIA에서 제공하는 프로그램의 설치가 필요하다.
- 그래픽 카드의 확인: dxdiag 명령어를 통해 그래픽 카드 확인 가능하다.



[그림 3] 그래픽 카드 검색 예시

- 그래픽 카드 드라이버 다운로드 링크를 통해 현재 그래픽 카드에 맞는 드라이버를 다운로드하여 설치할 수 있다. (링크: <https://www.nvidia.co.kr/Download/index.aspx?lang=kr>)
- 아래의 그림과 같이 그래픽 카드 드라이버를 제품 유형과 컴퓨터의 운영 체제에 알맞게 기입하여 드라이버 다운로드 및 버전을 확인할 수 있다.



[그림 4] NVIDIA 드라이버 다운로드에 사용할 그래픽 카드 정보를 입력하여 드라이버 버전을 확인하고 다운로드 할 수 있다.

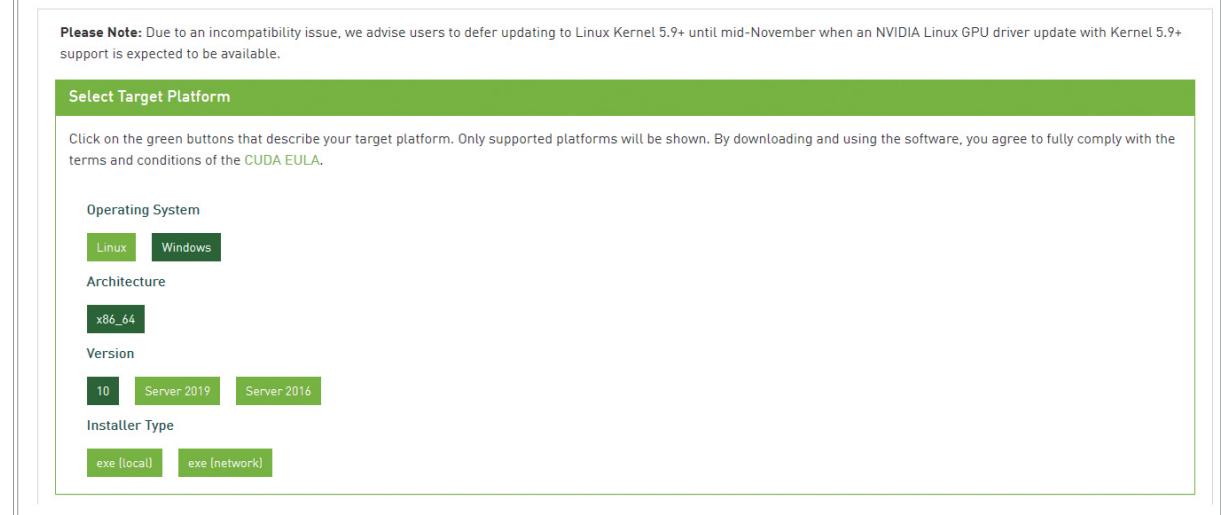
- 아래의 표를 참고하여 그래픽 카드 드라이버 버전에 알맞은 CUDA 설치한다.

Table 1. CUDA Toolkit and Compatible Driver Versions	
CUDA Toolkit	Linux x86_64 Driver Version
CUDA 11.1 (11.1.0)	>= 450.80.02
CUDA 11.0 (11.0.3)	>= 450.36.06
CUDA 10.2 (10.2.89)	>= 440.33
CUDA 10.1 (10.1.105)	>= 418.39
CUDA 10.0 (10.0.130)	>= 410.48
CUDA 9.2 (9.2.88)	>= 396.26
CUDA 9.1 (9.1.85)	>= 390.46
CUDA 9.0 (9.0.76)	>= 384.81
CUDA 8.0 (8.0.61 GA2)	>= 375.26
CUDA 8.0 (8.0.44)	>= 367.48
CUDA 7.5 (7.5.16)	>= 352.31
CUDA 7.0 (7.0.28)	>= 346.46

[그림 5] 드라이버 버전에 따른 CUDA 버전

- CUDA 설치 링크: <https://developer.nvidia.com/cuda-toolkit-archive>
- 운영체제에 알맞게 선택하여 Installer Type을 local로 지정하여 설치한다.

CUDA Toolkit 11.1.0

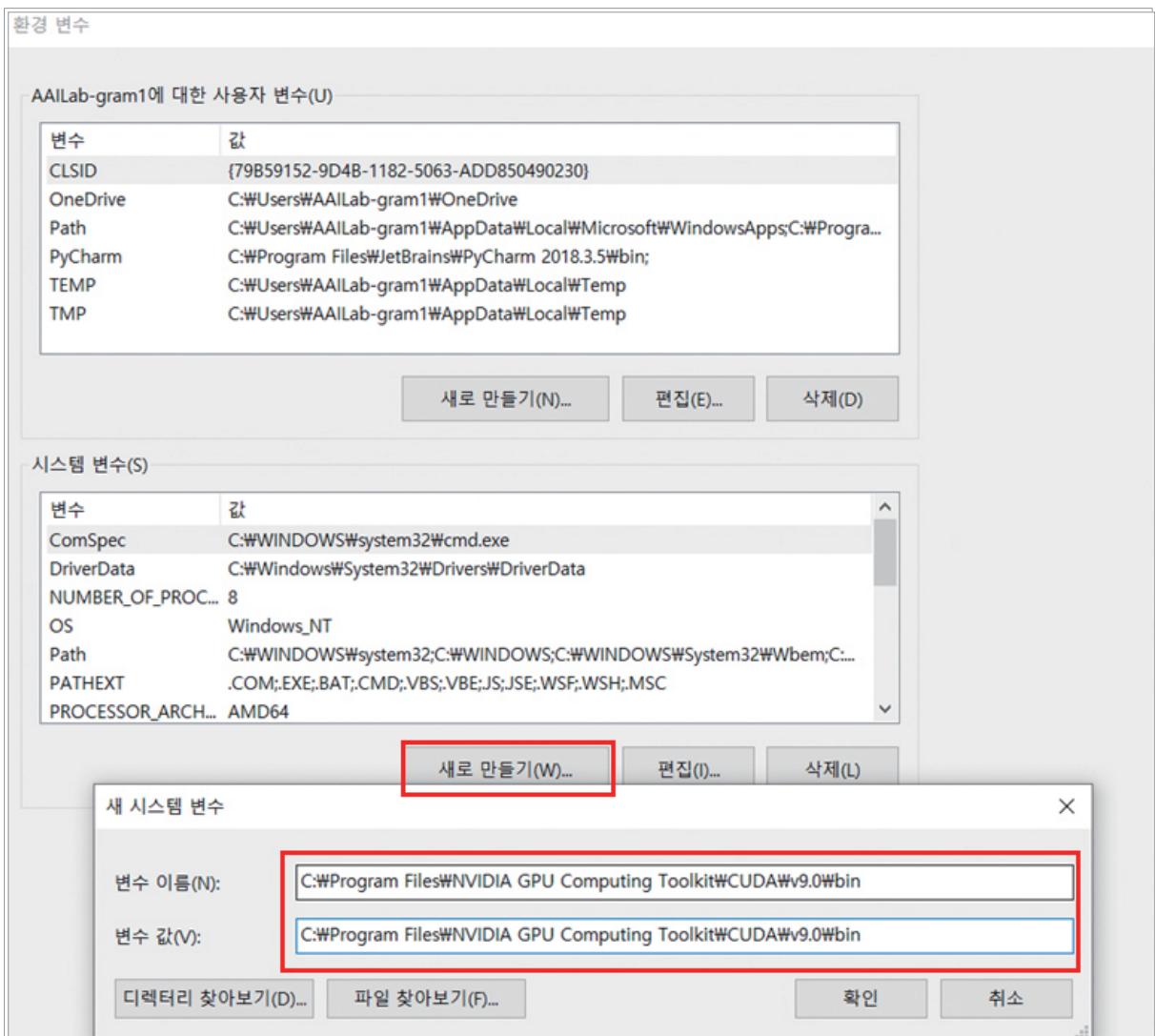


[그림 6] CUDA 설치 웹 사이트 예시

- 환경 변수 등록을 위하여 CUDA 설치 후 C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA 경로의 생성 여부 확인한다.
- 설치가 되었다면 버전이 9.0인 경우 아래의 세 가지 경로를 환경 변수로 등록한다.

```
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.0\bin  
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.0\include  
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.0\lib
```

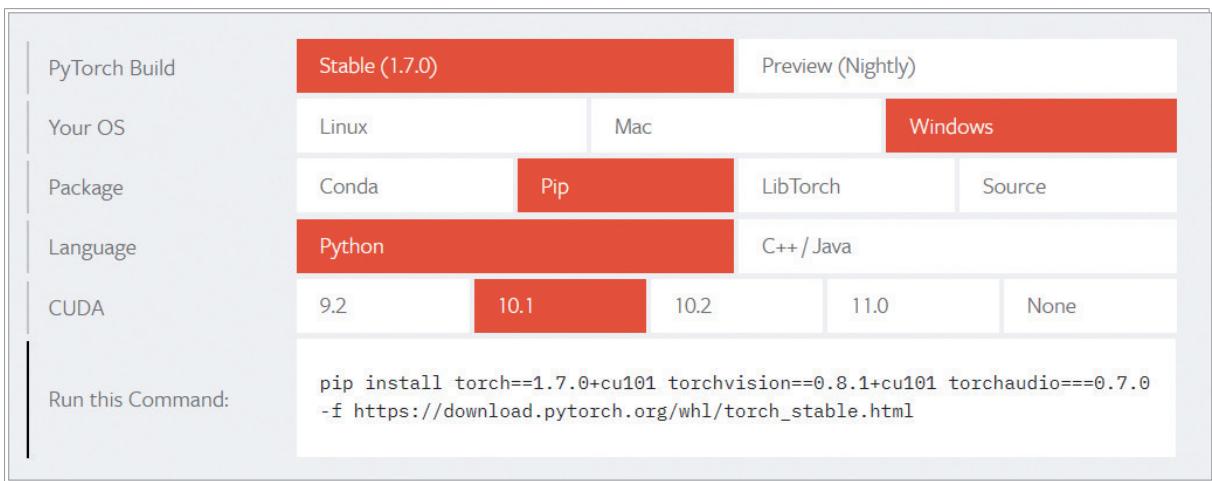
- 제어판 > 시스템 > 고급 > 환경변수를 통해 시스템 변수를 추가한다.



[그림 7] 시스템 변수 추가 예시

◎ Pytorch

- Pytorch 공식 홈페이지를 통해 설치한다. (링크: <https://pytorch.org/>)
- 공식 홈페이지 접속 후 스크롤을 내리면 설치를 위한 command를 알 수 있다. Stable 버전을 선택 후 운영체제(OS), Package는 Pip로 클릭, Langauge는 Python으로 클릭, 그리고 알맞은 CUDA 버전을 클릭하여 생성되는 command를 복사한다.

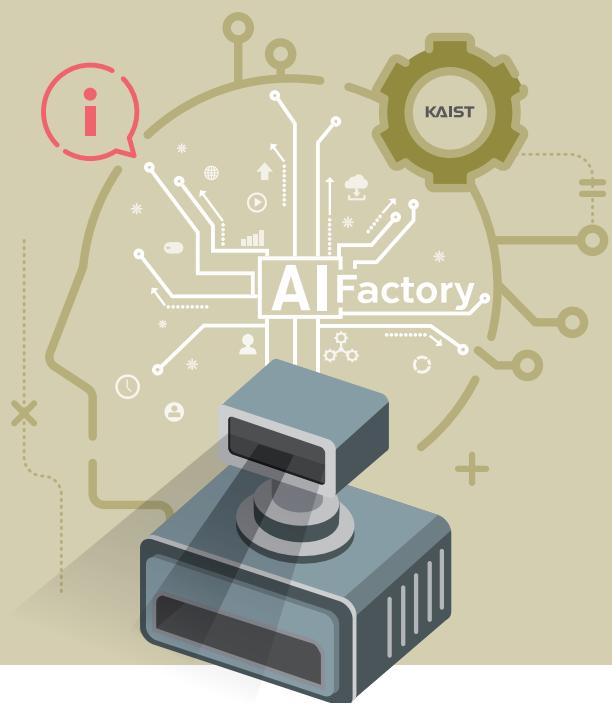


[그림 8] pytorch 설치 웹 사이트 예시

- Anaconda prompt를 실행 후 복사한 command 입력 후 pytorch를 설치한다.
- Anaconda prompt에서 Python 실행 후 “import torch”를 통해 pytorch 설치 여부를 확인할 수 있다.

『제조 현장용 OCR 학습 AI 데이터셋』

분석실습 가이드북



중소벤처기업부



스마트제조혁신추진단



34141 대전광역시 유성구 대학로 291 한국과학기술원(KAIST)
T. (042)350-2114 F. (042)350-2210(2220)