AI를 활용한 WARPAGE 발생 기판의 형태 분류 및 개선안 도출

전자구이통닭

목차 Content

Part 1

서론

1-1
과제의 필요성

1-2
문제정의

1-3
목적 계통도

1-4
제한 조건의 부여

1.1 과제의 필요성

- AI 시대가 본격적으로 도래하면서 전자 제품의 경박단소화가 가속화되고, 이에 따라 PCB 기판의 품질 관리가 더욱 중요해지고 있다. 공정 기술 역시 과거에 비해 더욱 정교하고 높은 수준의 난이도를 요구하게 되었으며, 고품질 PCB의 생산 수율을 높이는 것은 반도체 산업에서 해결해야 할 핵심 과제이다.

- 특히, 첨단 패키징 공정 중 Reflow와 같은 필수적인 제조 과정에서는 PCB 기판에 직접적으로 열이 가해지며, 이로 인해 warpage (기판 휨 현상)가 발생할 수 있습니다. 이러한 warpage는 기판의 평탄도를 저하시키고, 심각한 경우 제품 불량을 초래하여 전체적인 신뢰성을 떨어뜨립니다.

- 따라서, warpage를 사전에 예측하고 이를 효과적으로 개선할 수 있는 방법을 모색하는 것은 PCB 제조 공정에서 필수적이다. 본 프로젝트는 AI 기술을 활용하여 PCB 기판의 warpage를 분석하고, 이를 기반으로 최적의 공정 조건을 도출함으로써 생산 수율을 향상시키는 것이 목표이다.

1.2 문제 정의

- PCB 기판은 Reflow 공정에서 열을 받으면 warpage 현상이 발생한다.

- Warpage의 정도가 산업 기준 허용치를 초과할 경우, 기판이 사용 불가능해지며, 이로 인해 생산 수율이 감소할 위험이 있다.
  이러한 문제를 해결하기 위해 다음과 같은 세 가지 주요 과제로 정의한다.

1. Warpage가 발생한 PCB 기판의 형상을 JEDEC 기준에 따라 효과적으로 분류할 수 있는 방법이 필요하다.
2. 분류된 warpage 발생 기판 이미지를 통해 공정에 적용된 시간 값, 그리고 최대 변위 값을 정확하게 도출할 수 있어야 한다.
3. Warpage를 개선하기 위한 추가적인 공정 조건을 정확하게 예측할 수 있는 모델이 필요하다.

1.3 문제의 목적계통도

“ AI를 활용한 WARPAGE 발생 기판의 형태 분류 및 개선안 도출 ”

정확성

형상 분류 정확도 향상

- CNN 모델을 통한 warpage 형상 분류 R^2 score 85% 이상 달성
- 고해상도 이미지를 통한 기판 형태 분석

예측 모델 정확성 확보

- ClovaOCR API을 활용해 시간, 최대 변위를 도출하여 예측 모델 정확도 개선
- 데이터셋 확장 및 최적화로 예측 성능 향상

효율성

공정 최적화

- Warpage를 최소화하는 최적 Reflow 공정 시간 제시
- 다양한 공정 조건에 대한 시뮬레이션으로 개선안 도출

모델 학습 시간 단축

- 데이터 전처리 최적화로 학습 시간 감소
- M1, M2 모델 간의 효율적인 데이터 파이프라인 구축

안정성

공정 조건의 신뢰성

- 시뮬레이션 기반 공정 조건이 실제 환경에서 일관되게 적용
- 환경 변화에 대한 모델의 견고성 확보

시스템 안전성 확보

- AI 모델의 예측 결과에 따른 공정 안정성 확보
- Warpage 개선을 위한 최적 조건 도출 시 안전 기준 준수하여 안전성 확보

1.4 제한 조건의 부여

본 프로젝트는 국제 표준(JEDEC, IPC 등)을 준수하여 설계 및 제작할 것이다. 이를 통해 모델의 성능, 신뢰성, 호환성을 보장할 수 있다.

- 데이터의 신뢰성: Ansys workbench와 python을 사용하여 시뮬레이션 데이터를 생성하며, 이 데이터의 정확성과 일관성을 전제로 한다.
- 모델의 복잡성: AI 모델은 CNN, OCR, LSTM의 세 가지 유형으로 사용하며, 각각의 모델은 특정 역할을 담당하도록 설계한다.
- 적용 범위: PCB 기판의 크기, 재료(FR4), 공정 조건(온도 및 시간) 등을 일정 범위 내에서 설정하고 분석한다.
- 결과의 해석: Warpage 현상의 개선은 주로 기판의 변형률과 변위 값을 기준으로 평가하며, 이들 값이 IPC 산업 기준 허용 범위 내에 있을 경우 개선된 것으로 간주한다.

Part 2

설계

2-1
관련 이론

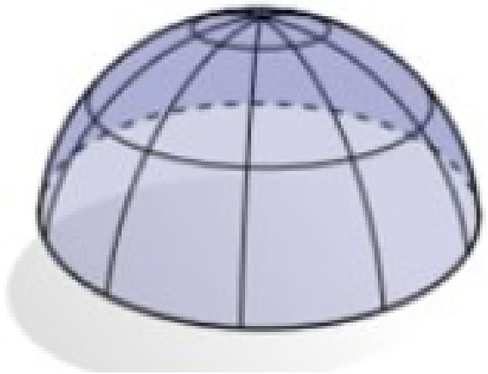
2-2
개념 설계

2-3
상세 설계

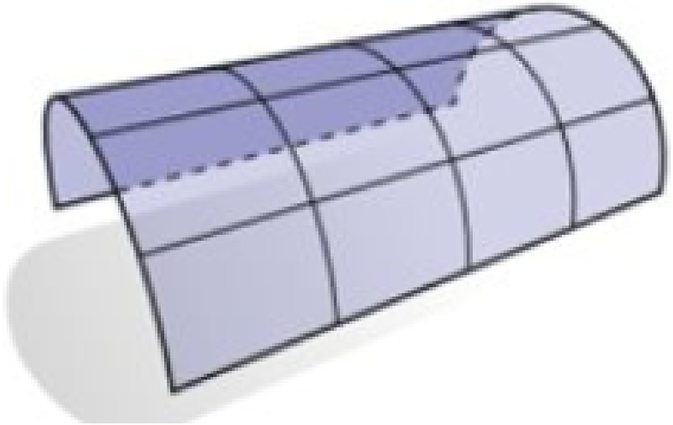
2.1 관련 이론

JEDEC 기준에 따라 PCB 기판의 warpage 형상은 다음과 같이 구분된다.

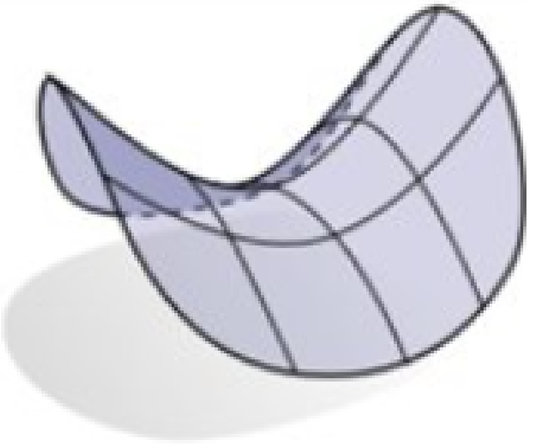
- 구형 (Spherical shape): 두 축 모두 양 혹은 음의 곡률을 가지는 경우
- 원통형 (Cylindrical shape): 하나의 축에서만 곡률을 가지는 경우
- 안장형 (Saddle shape): 한 축과 다른 축의 곡률 부호가 반대인 경우



구형



원통형



안장형

2.1 관련 이론

열팽창 계수 (**C**oefficient of **T**hermal **E**xpansion)
열팽창 계수는 재료가 온도 변화에 따라 얼마나 팽창하거나 수축하는지를 나타내는 물리적 값이다. CTE는 재료의 변형을 예측하는 데 필수적인 요소이고, CTE 값이 클수록 온도 변화에 따른 재료의 변형이 커진다. 이는 PCB 기판의 warpage 현상을 예측하고 개선하는 데 중요한 역할을 한다.

탄성 계수 (Young's Modulus)
탄성 계수는 재료의 강성을 나타내는 값으로, 응력에 대한 변형률의 비율을 의미한다. 이 값이 클수록 재료는 단단하며 변형에 저항하는 힘이 크다. PCB 기판의 warpage 현상을 분석할 때, 탄성계수는 온도 변화에 따른 기판의 변형 정도를 예측하는 데 필수적인 요소이다.

열전도율 및 비열
기판의 열전도율은 warpage 현상에 큰 영향을 미치고, 열전도율이 높을수록 열이 균일하게 퍼져 warpage가 감소할 수 있다. 또한, 비열은 재료가 열을 저장할 수 있는 능력을 나타내며, 이는 기판의 온도 분포와 warpage 형성에 중요한 영향을 미친다.

휨률
IPC 기준 PCB 기판의 warpage 정도를 정량화하기 위해 휨률을 사용하고, 휨률은 일반적으로 다음의 공식을 통해 계산된다.
휨률 (%)=(최대 변위/원래 기판의 길이) x 100%

2.1 관련 이론

AI 모델은 warpage 현상 분석 및 예측에 매우 효과적으로 활용된다.

- **C**onvolutional **N**eural **N**etwork (CNN): 이미지 인식에 강점을 가진 모델로, PCB 기판의 warpage 형태를 분석하고 분류하는 데 사용된다.

- **L**ong **S**hort-**T**erm **M**emory (LSTM): 시계열 데이터를 바탕으로 최적의 공정 조건을 도출하는 데 사용된다. LSTM은 특히 warpage를 줄이기 위한 공정 조건(온도, 시간 등)을 예측하여 개선안을 도출하는 데에 효과적이다.

2.2 개념 설계

본 프로젝트의 개념 설계는 AI 모델을 이용하여 PCB 기판의 warpage를 사전 예측하고, 공정 조건의 최적화 값을 도출하는 것이며, 이를 위해 2단계의 AI 모델을 설계하였다.

- **M1 (CNN)**: PCB 기판 이미지를 입력으로 받아 warpage 형태(원통형, 안장형, 구형)를 분류한다. 기존에 학습된 비지도 학습 모델을 사용하여 warpage 형상을 신속하게 분류할 수 있다.

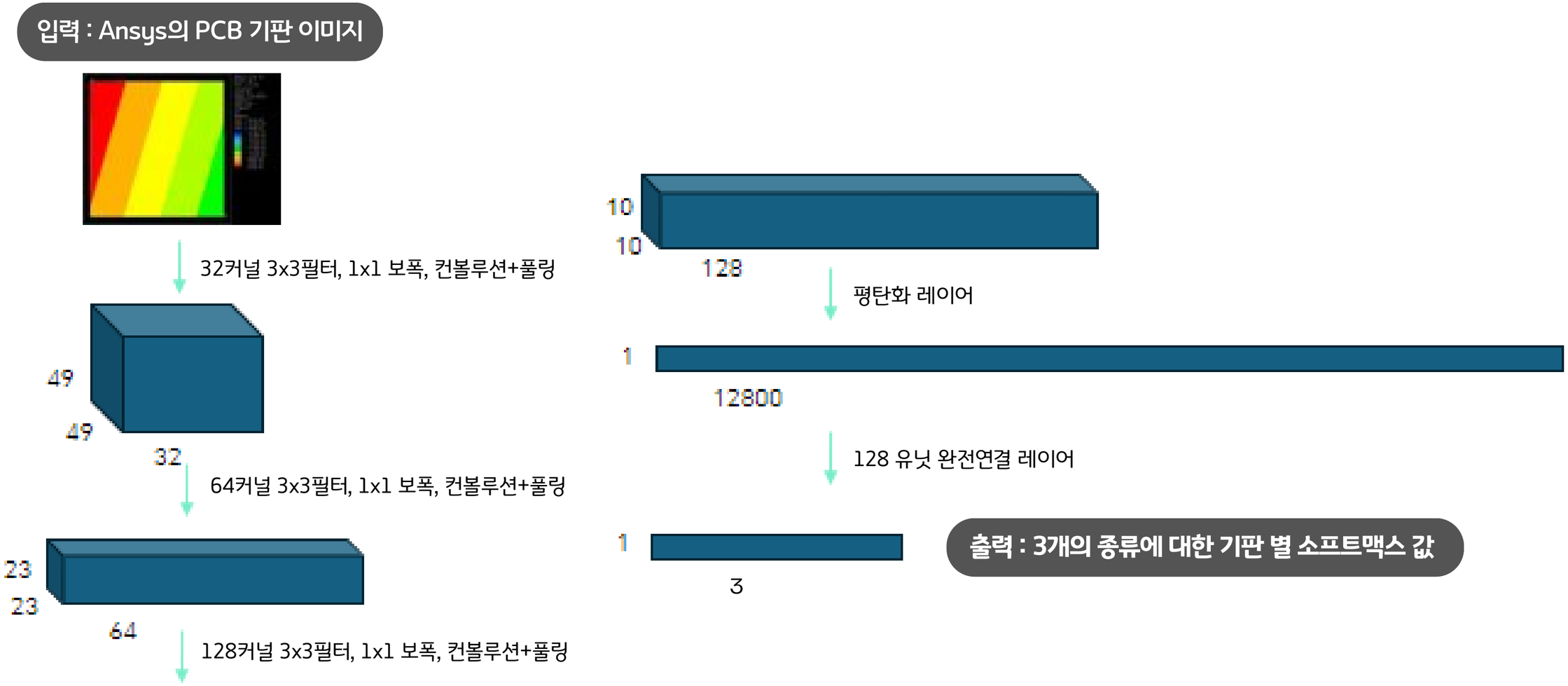
- OCR: CNN으로 분류된 이미지를 바탕으로 시간, 색상 별 변위를 추출한다. 추출된 값은 텍스트 파일로 출력되어 후속 모델(M2)에서 사용된다.

- **M2 (LSTM)**: OCR을 통해 출력한 시간, 변위 데이터를 사용하여 목표 휨률 값을 달성하기 위해 필요한 최대 변위값이 최소가 되는 시간을 예측한다. 이 모델을 통해 최적의 reflow 공정 시간을 도출할 수 있다.

2.3 상세 설계

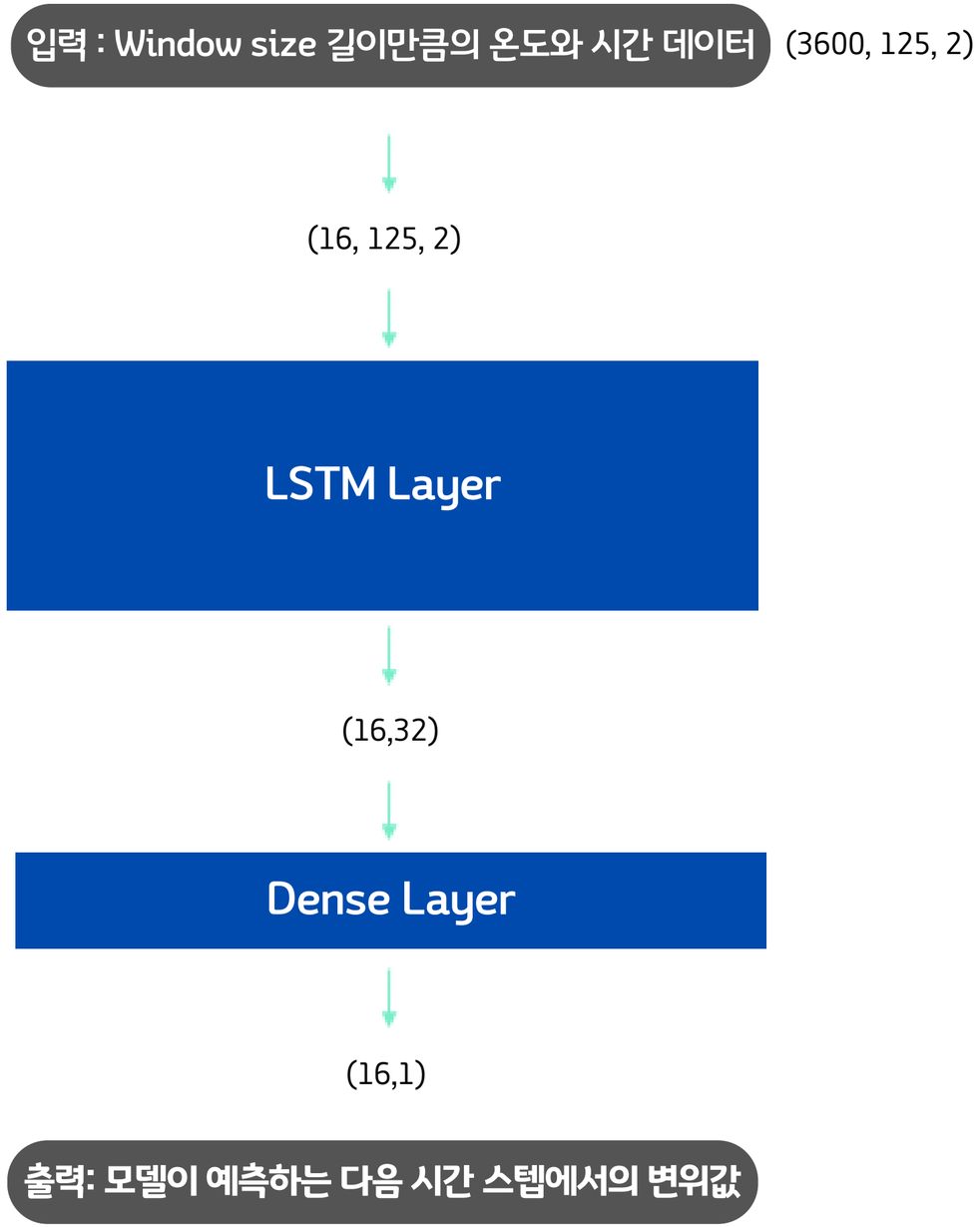
상세 설계 단계에서는 각 AI 모델의 구조와 동작 방식을 구체화하고, 이를 실제 데이터에 적용하여 시뮬레이션을 수행한다.

- M1(CNN)



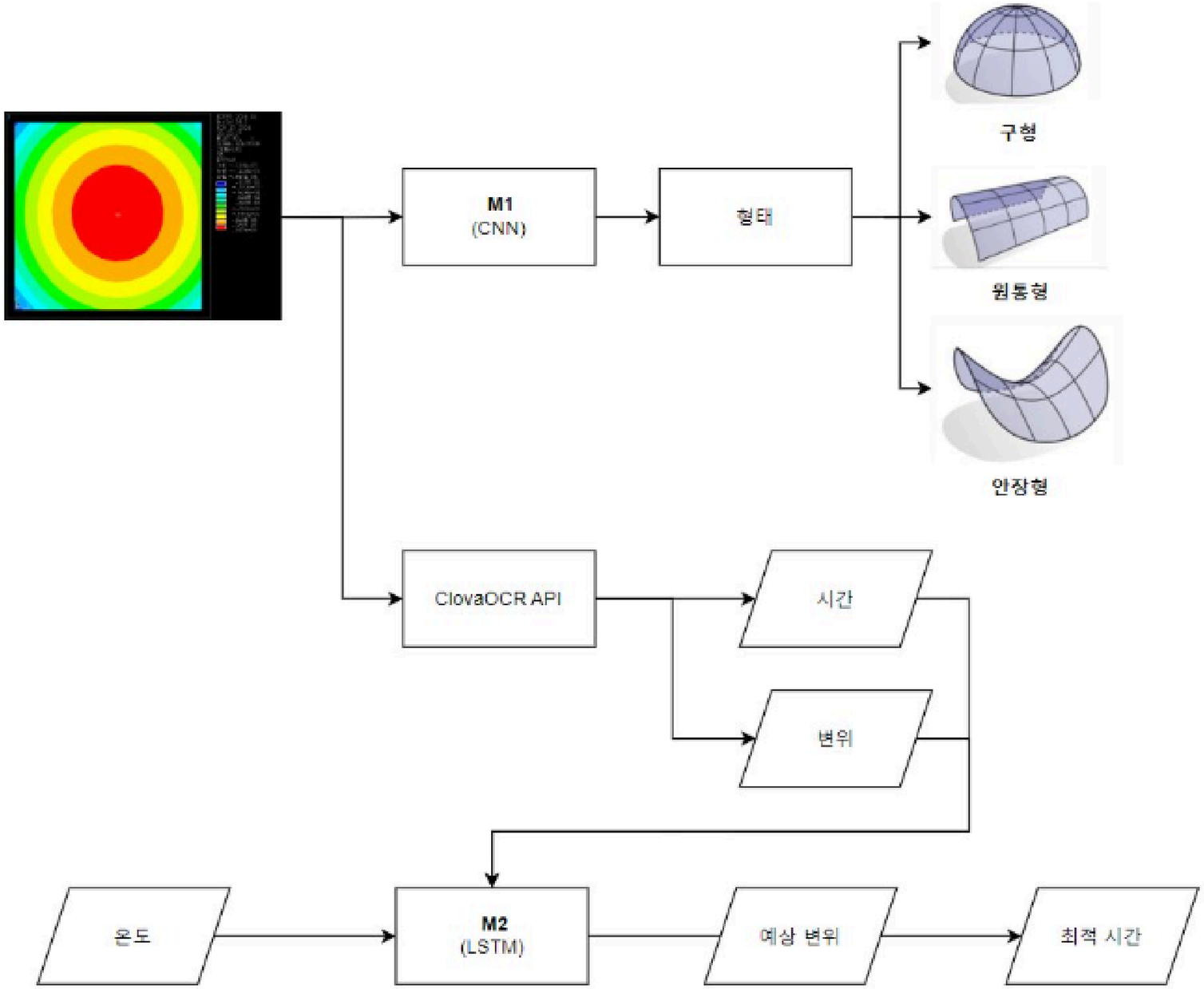
2.3 상세 설계

- M2(LSTM)



2.3 상세 설계

- 전체 과정



Part 4

Dataset 구성

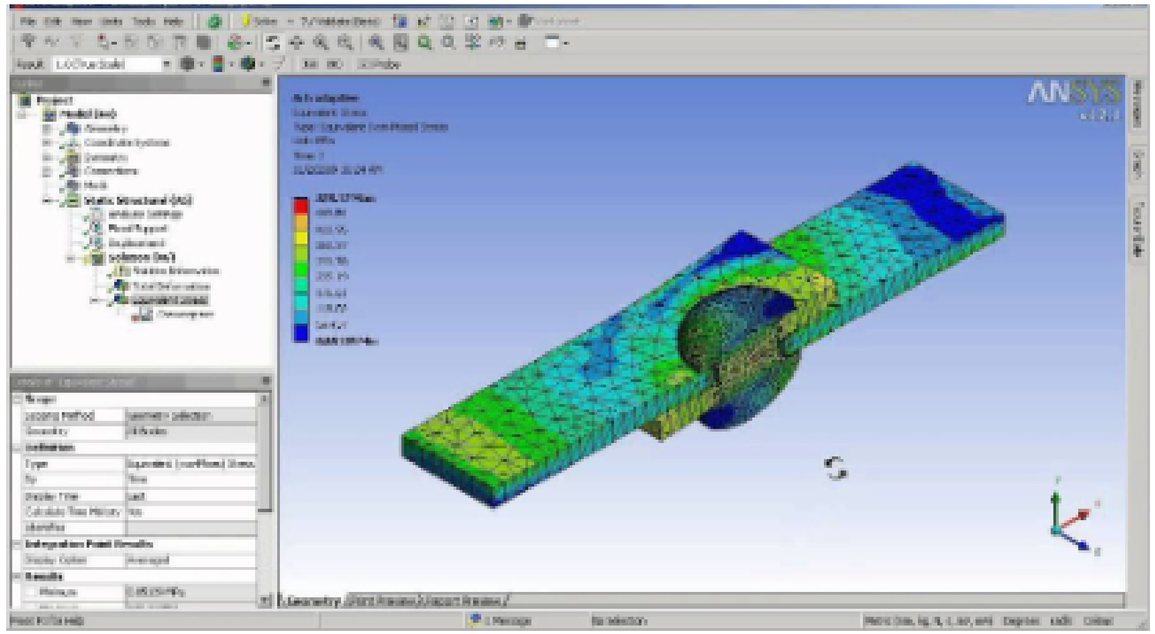
3-1
PCB 기판 dataset

3-2
Test data

3.1 PCB 기판 Dataset

목표로 하는 환경에서 PCB 기판에 열을 가한 후, 이를 시뮬레이션하여 변형률 등의 데이터를 획득하기 위해, Ansys의 MAPDL을 기반으로 한 Python 자동화 스크립트를 통해 데이터셋을 생성

동일한 온도를 대류열을 통해 가했을 때 시간에 따른 기판의 Z축 변위과 그 이미지를 출력



3.1 PCB 기판 Dataset

```
# PCB 크기 설정

board_width = 0.05
board_height = 0.05
board_thickness = 0.002

# PCB 블록 생성
mapdl.block(0, board_width, 0, board_height, 0, board_thickness)

# FR-4 재료 속성 설정
mapdl.mp('EX', 1, 2.04e10)  # Young's Modulus X direction
mapdl.mp('EY', 1, 1.84e10)  # Young's Modulus Y direction
mapdl.mp('EZ', 1, 1.5e10)  # Young's Modulus Z direction

mapdl.mp('PRXY', 1, 0.11)  # Poisson's Ratio XY
mapdl.mp('PRYZ', 1, 0.09)  # Poisson's Ratio YZ
mapdl.mp('PRXZ', 1, 0.14)  # Poisson's Ratio XZ

mapdl.mp('GXY', 1, 9.2e9)  # Shear Modulus XY
mapdl.mp('GYZ', 1, 8.4e9)  # Shear Modulus YZ
mapdl.mp('GXZ', 1, 6.6e9)  # Shear Modulus XZ

mapdl.mp('DENS', 1, 1840)  # Density

mapdl.mp('ALPX', 1, 1.25e-5)  # Thermal Expansion Coefficient X direction
mapdl.mp('ALPY', 1, 1.14e-5)  # Thermal Expansion Coefficient Y direction
mapdl.mp('ALPZ', 1, 8.25e-6)  # Thermal Expansion Coefficient Z direction

mapdl.mp('KXX', 1, 0.38)  # Thermal Conductivity X direction
mapdl.mp('KYY', 1, 0.38)  # Thermal Conductivity Y direction
mapdl.mp('KZZ', 1, 0.36)  # Thermal Conductivity Z direction

# 요소 유형 및 메시 설정
mapdl.et(1, 'SOLID186')
mapdl.vmesh('ALL')
```

[초기 기판 설정]

IPC-2221 표준의 기판 크기 범위에 포함되는 크기인 가로 세로 50mm, 높이2mm로 설정하였고, 재료는 최근 산업계에서 많이 사용되는 재료인 FR-4로 설정하였다.

재료의 물성값은 Ansys workbench에 저장되어있는 materials의 값을 참고하였다. 또한 layer가 나뉘어져 있지 않은 단일 기판을 해석하기 때문에 mesh는 자동 설정으로 두고 해석하였다.

3.1 PCB 기판 Dataset

```
step = 300  # 시간

initial_temp = 25  # 기준 온도
ambient_temp = 300  # 주변 온도
conv_coeff = 100   # 대류 열전달 계수 (W/m^2-K)

mapdl.bf( node: 'ALL',  lab: 'TEMP', initial_temp)  # 초기 온도 설정
# 전체 표면에 대류 열전달 조건 적용
mapdl.sf( nlist: 'ALL',  lab: 'CONV', conv_coeff, ambient_temp)

# SOLUTION 모드로 전환
mapdl.slashsolu()

# 과도 열 해석 설정
mapdl.antype('TRANS')  # 과도 해석
mapdl.trnopt('FULL')  # 전체 과도 해석
mapdl.timint('ON')  # 시간 적분 활성화
mapdl.kbc(0)  # 선형적 시간 증가

# 시간 단계 설정
total_time = step  # 전체 해석 시간
time_steps = 1  # 시간 단계 수 (1초 간격)
mapdl.deltim(1)  # 시간 증분 1초
mapdl.nsubst(total_time/time_steps,  nsbmx: 1,  nsbmn: 1)  # 시간 단계 옵션
```

[시간 및 온도 설정]
초기온도가 25도인 상태의 기판에 300초간 대류열을 가하는 예시이고, 대류 열전달 계수를 100으로 설정한다.

기준 온도(Initial Temperature): 25˚C
일반적인 실온 온도로, 25˚C는 PCB 기판이 공정에 들어가기 전의 상태를 나타낸다.

대류 열전달 계수(Convective Heat Transfer Coefficient): 100 W/m²-K
일반적으로 reflow 오븐의 대류열 전달계수는 50-150 W/m²-K 사이이다. 여기서는 100으로 설정하여 평균적인 reflow 공정 조건을 모사한다.

주변 온도(Ambient Temperature): 300˚C
Reflow 공정의 주요 단계 중 하나는 기판을 높은 온도(보통 200˚C에서 250˚C 사이)로 가열하여 solder paste를 녹이는 것이다. 본 프로젝트는 그보다 조금 높은 300˚C를 설정하여 기판의 변형을 보다 효과적으로 확인하였고, 이는 과도한 열이 가해질 경우 기판이 얼마나 변형될 수 있는지 확인하기 위함이다.

3.1 PCB 기판 Dataset

```
step = 300  # 시간

initial_temp = 25  # 기준 온도
ambient_temp = 300  # 주변 온도
conv_coeff = 100   # 대류 열전달 계수 (W/m^2-K)

mapdl.bf( node: 'ALL',  lab: 'TEMP', initial_temp)  # 초기 온도 설정
# 전체 표면에 대류 열전달 조건 적용
mapdl.sf( nlist: 'ALL',  lab: 'CONV', conv_coeff, ambient_temp)

# SOLUTION 모드로 전환
mapdl.slashsolu()

# 과도 열 해석 설정
mapdl.antype('TRANS')  # 과도 해석
mapdl.trnopt('FULL')  # 전체 과도 해석
mapdl.timint('ON')  # 시간 적분 활성화
mapdl.kbc(0)  # 선형적 시간 증가

# 시간 단계 설정
total_time = step  # 전체 해석 시간
time_steps = 1  # 시간 단계 수 (1초 간격)
mapdl.deltim(1)  # 시간 증분 1초
mapdl.nsubst(total_time/time_steps,  nsbmx: 1,  nsbmn: 1)  # 시간 단계 옵션
```

[해석 및 시간단계 설정]

시간의 흐름에 따른 기판의 변화를 보기 위해 **동적해석** 수행을 진행했고, 시간 **step**을 **1**로 설정하여 1초마다 해석을 수행하도록 설정한다.

전체 해석 시간(Total Time): 300초

일반적으로 reflow 공정은 예열, 리플로우, 냉각의 세 단계를 거친다. 각 단계는 보통 60-90초 정도 소요되며, 총 공정 시간은 약 180-240초가 소요된다. 여기서는 총 300초로 설정하여 reflow 공정 이후에도 기판이 안정화 되는지 확인하려고한다.

시간 단계(Time Steps): 1초 간격으로 300초 설정

시간 단계는 기판이 열에 반응하여 변형되는 과정을 정밀하게 추적하기 위해 설정하였다. 1초 간격으로 설정하여 시간에 따른 기판의 변형을 정확하게 분석할 수 있도록 하였다.

180초부터 300초까지의 예측

M2 모델에서 180초부터 300초까지 예측하는 이유는, 일반적인 reflow 공정에서 가장 중요한 단계인 reflow 단계가 약 180초 정도에 시작되기 때문이다. 그 후 reflow 단계가 끝나면, 300초까지의 데이터는 기판이 안정화 되는 과정을 보여준다. 이러한 설정은 simulation을 실제 공정과 유사하게 하여 공정을 최적화하고, 중요한 시점을 모델링하는 데 사용된다.

3.1 PCB 기판 Dataset

- 데이터 출력

```python
# 모든 노드를 순회하면서 최대 및 최소 UZ 변위 찾기
for i in range(1, 2073):
    uz_node = mapdl.get( par: "uz",  entity: 'node', i,  item1: 'u',  it1num: 'z')
    if uz_node > max_uz:
        max_uz = uz_node
        max_node = i
    if uz_node < min_uz:
        min_uz = uz_node
        min_node = i
# 최대 및 최소 UZ 변위 출력
json_data = {
    'MINIMUM VALUES': {
        'NODE': min_node,
        'VALUE': min_uz,
    },
    'MAXIMUM VALUES': {
        'NODE': max_node,
        'VALUE': max_uz,
    }
}
```

.Json

```python
# Z축 변위를 이미지로 저장
mapdl.plnsol( item: 'U',  comp: 'Z')
mapdl.show('PNG')
temp_files = glob.glob(os.path.join(mapdl.directory, '*.png'))
```

.png

```python
# Z축 변위 출력
strain_data = mapdl.prnsol( item: 'U',  comp: 'Z')  # 변형률 데이터를 가져옴
strain_file = os.path.join(save_directory_txt, f'strain_data_{current_time}.txt')
with open(strain_file, 'w') as file:
    file.write(strain_data)
```

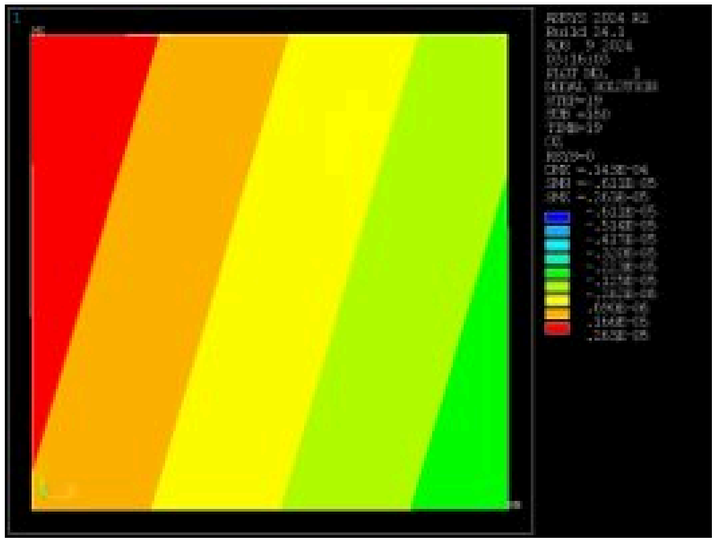
.txt

3.1 PCB 기판 Dataset

- 데이터 출력


```
{
    "MINIMUM VALUES": {
        "NODE": 54,
        "VALUE": -6.1117518e-06
    },
    "MAXIMUM VALUES": {
        "NODE": 614,
        "VALUE": 2.633321 51e-06
    }
}
```


.Json



.png

NODE	UZ
1	-0.14917E-005
2	-0.25353E-005
.	.
.	.
.	.
2071	-0.30287E-005
2072	-0.29484E-005

.txt

Part 4

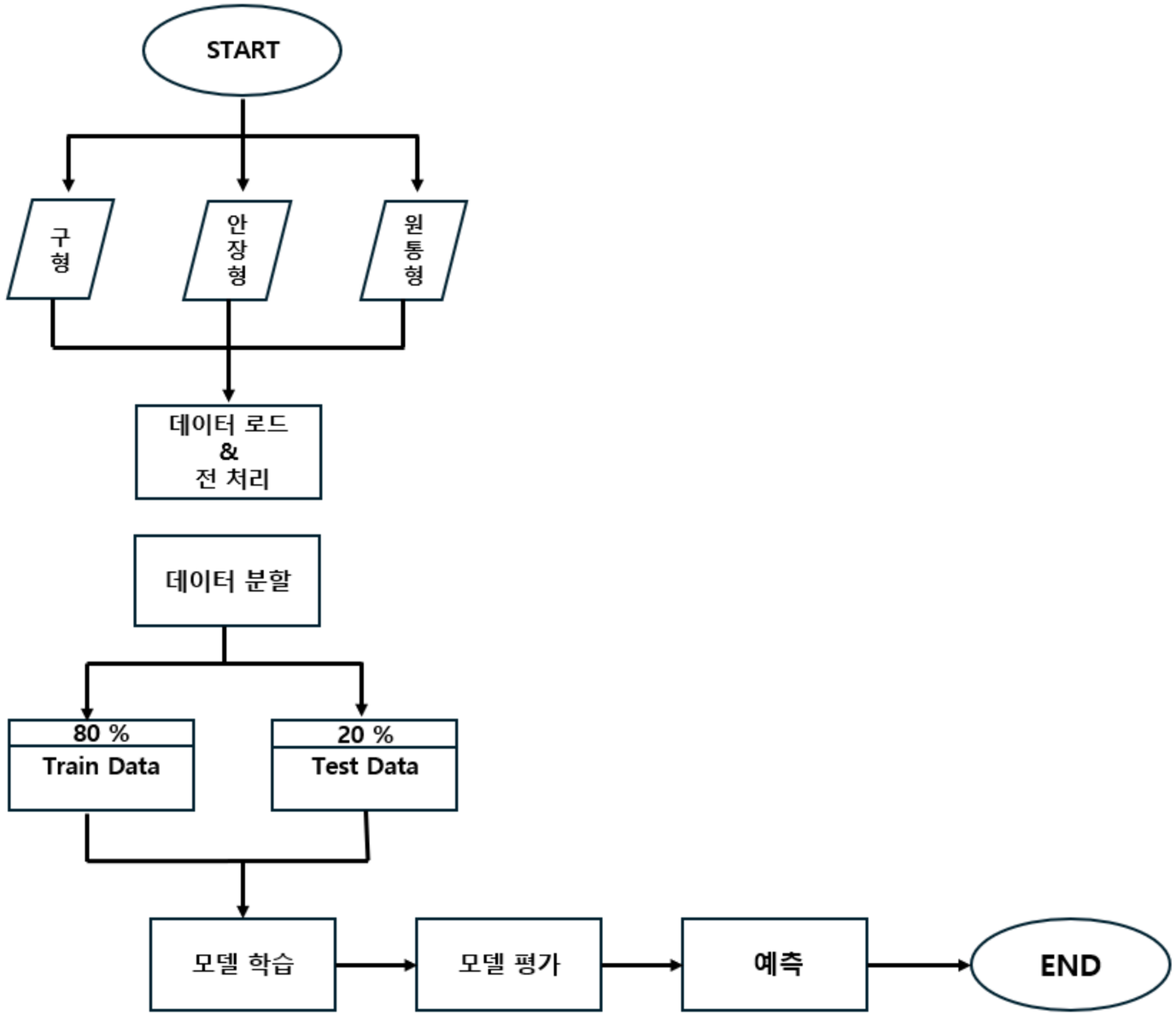
모델

4-1
Model 1

4-2
Model 2

4.1 Model

- CNN


START
구
형
안
장
형
원
통
형
데이터 로드
&
전 처리
데이터 분할
80 %
Train Data
20 %
Test Data
모델 학습
모델 평가
예측
END

4.1 Model

- CNN 모델 구축

```
# 모델 정의
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(img_size, img_size, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.6),
    Dense(3, activation='softmax')  # 출력 레이어의 노드 수를 3으로 변경
])
# 모델 컴파일
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

CNN layer : CNN layer와 Max Pooling layer를 사용하여 시퀀스 데이터를 처리, 활성화 함수로 **Relu** 함수를 사용

출력 layer : 3개의 unit을 가진 Dense Layer를 통해 최종 예측값 출력

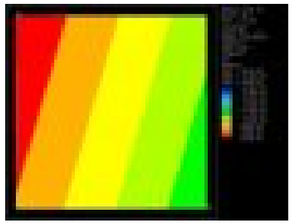
모델 compile : **Adam** 옵티마이저 사용

학습 : **10**번의 epochs

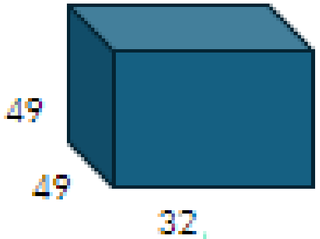
4.1 Model

- CNN 모델 구조

입력 : Ansys의 PCB 기판 이미지

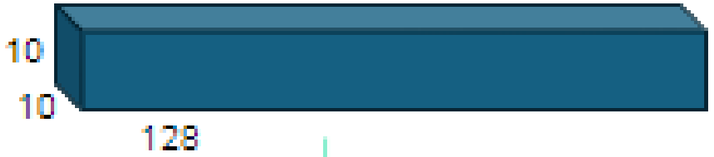


32커널 3x3필터, 1x1 보폭, 컨볼루션+풀링



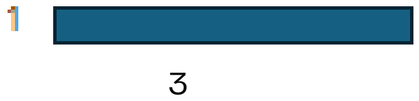
64커널 3x3필터, 1x1 보폭, 컨볼루션+풀링

128커널 3x3필터, 1x1 보폭, 컨볼루션+풀링



평탄화 레이어

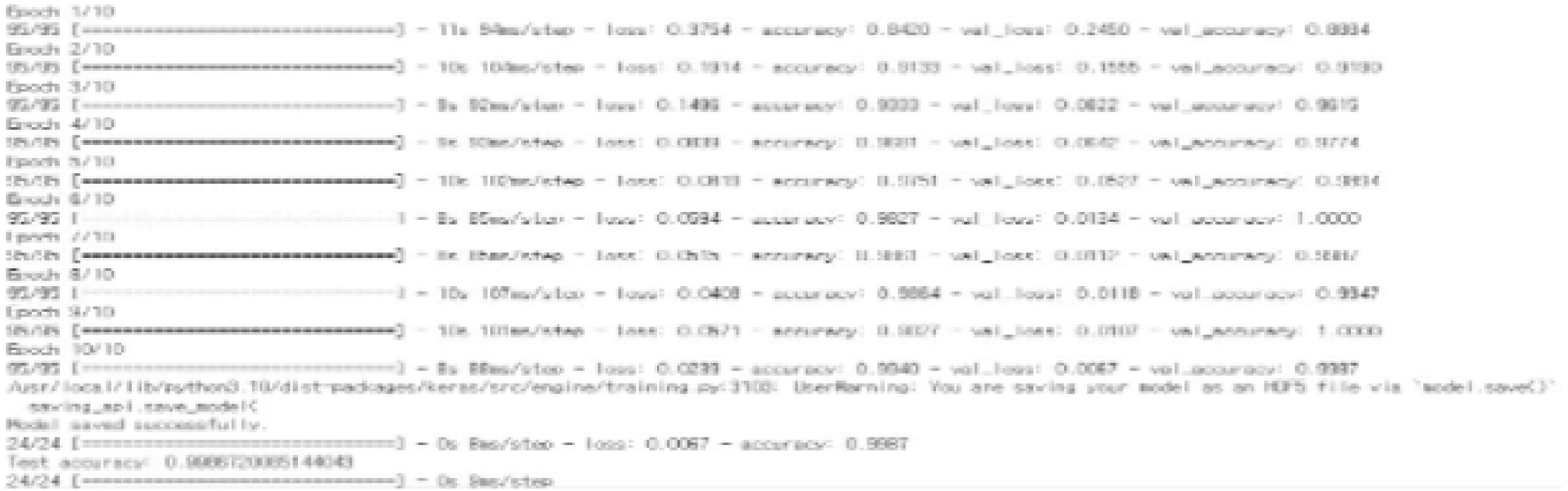
128 유닛 완전연결 레이어



출력 : 3개의 종류에 대한 기판 별 소프트맥스 값

4.1 Model

- CNN 모델 학습 및 성능평가 결과



[학습과정]

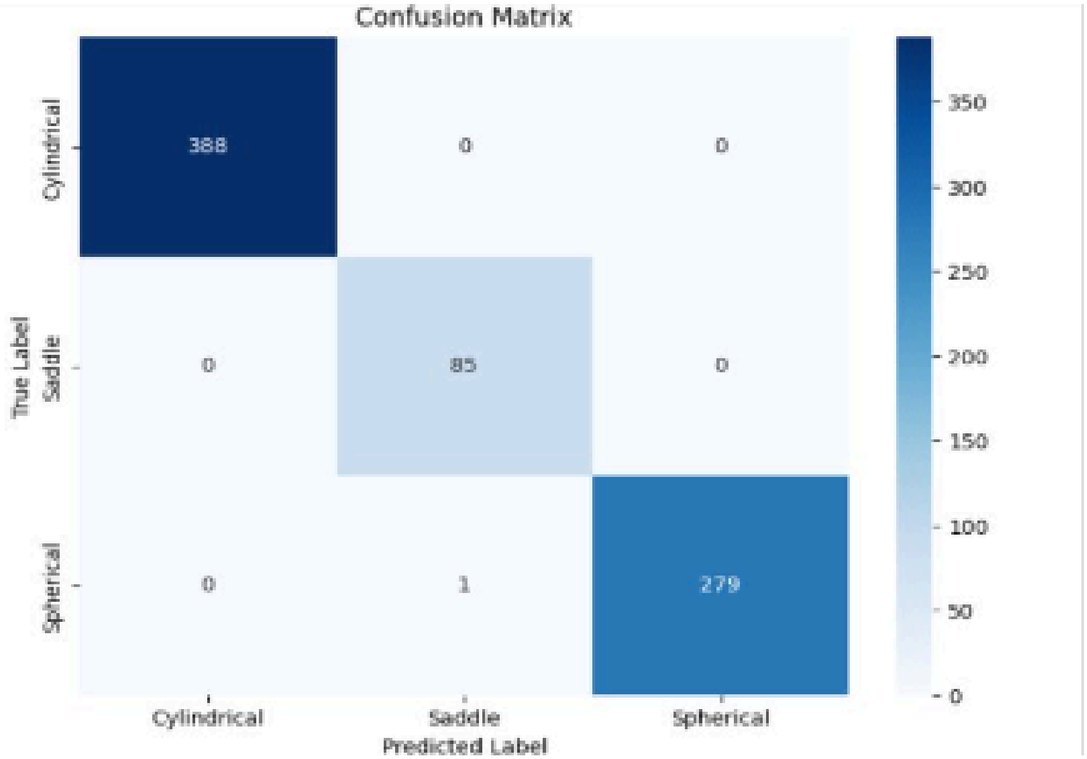
모델은 마지막 epoch에서 **0.0239**로 매우 낮은 loss값을 기록하였으며 이를 통해 모델이 잘 수렴하는 것 확인 가능

[성능지표 분석]

Train Acuracy : 99.4% , Validation Accuracy : 99.87% , Test Accuracy : 99.86%

4.1 Model

- CNN 모델 학습 및 성능평가 결과



(1) Confusion Matrix

Classification Report	Precision	Recall	F1-score	support
Cylindrical	1.00	1.00	1.00	388
Saddle	0.99	1.00	0.99	85
Spherical	1.00	1.00	1.00	280
Accuracy			1.00	280
Macro avg	1.00	1.00	1.00	753
Weighted avg	1.00	1.00	1.00	753

(2) Classification Report

4.1 Model

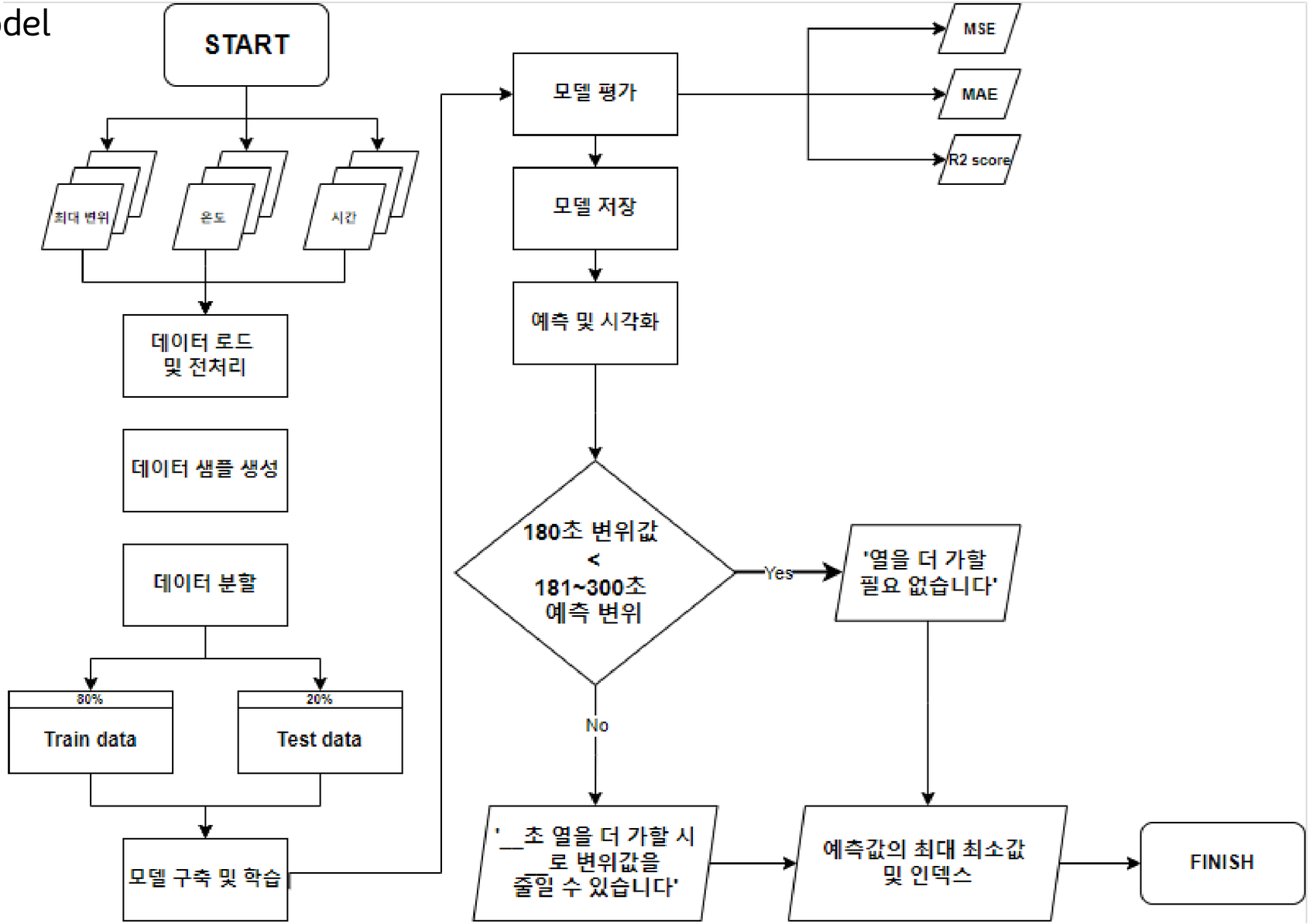
- CNN 모델 예측 결과

파란색 실선은 Train Accuracy와 Train Loss
주황색 점선은 Validation Accuracy와 Validation Loss

CNN 모델이 학습을 통해 Accuracy는 향상되고 Loss는 감소되는 것을 확인

4.2 Mode2

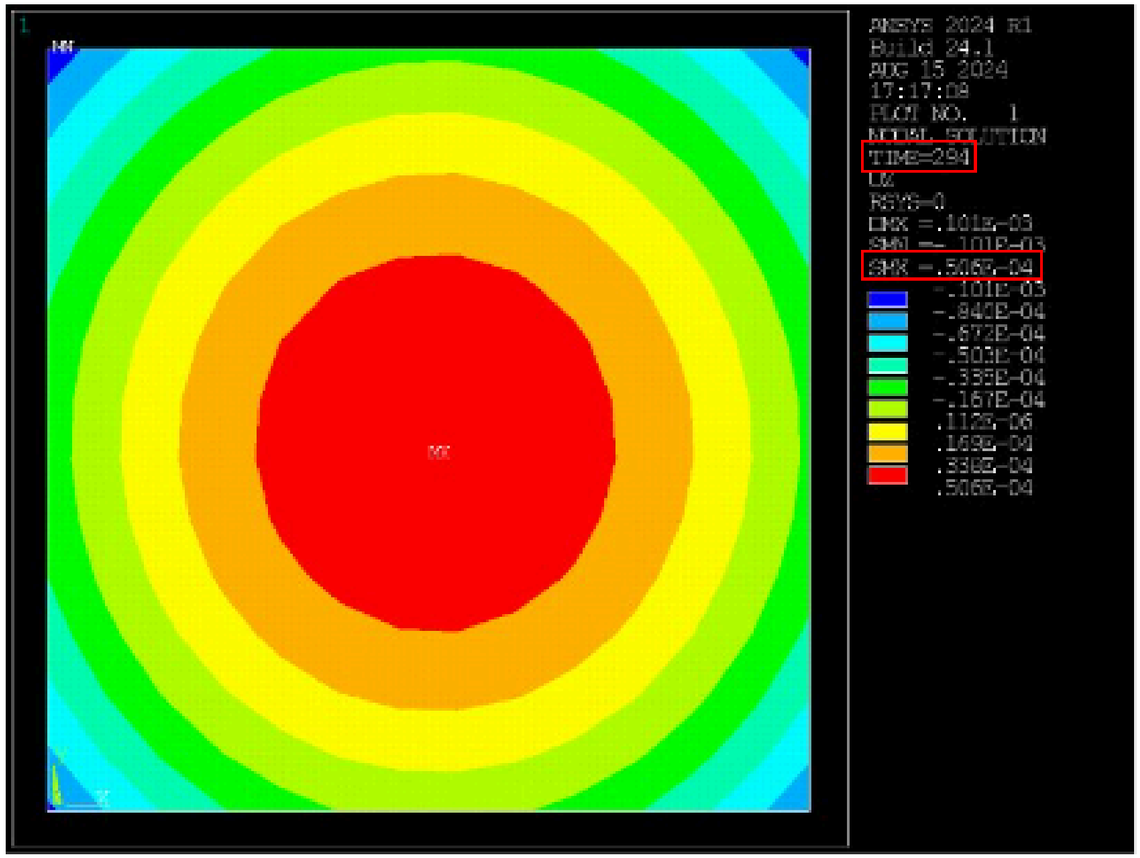
- LSTM model



4.2 Mode2

- LSTM model

  -> ClovaOCR API를 이용하여 이미지에서 추출한 텍스트 파일

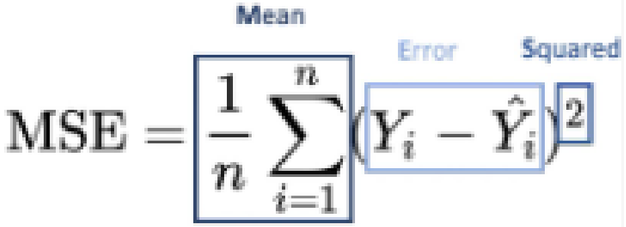



```
[
    "1",
    "ANSYS",
    "2024",
    "R1",
    "MN",
    "Build",
    "24.1",
    "AUG",
    "15",
    "2024",
    "17:17:08",
    "PLOT",
    "NO.",
    "1",
    "NODAL",
    "SOLUTION",
    "TIME=294",
    "UZ",
    "RSYS=0",
    "DMX",
    "=.101E-03",
    "SMN",
    "101E-03",
    "SMX",
    "=.506E-04",
    "101E-03",
    "840E-04",
    "672E-04",
    "503E-04",
    "335E-04",
    "167E-04",
    "112E-06",
    "169E-04",
    "MX",
    "338E-04",
    "506E-04"
]
```

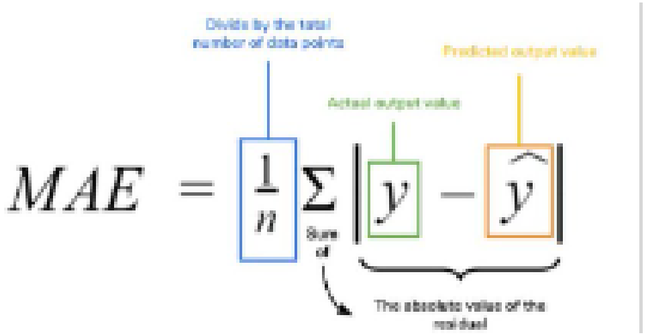

TIME과 기판의 최대 변위값을 json 파일에서 도출

4.2 Mode2

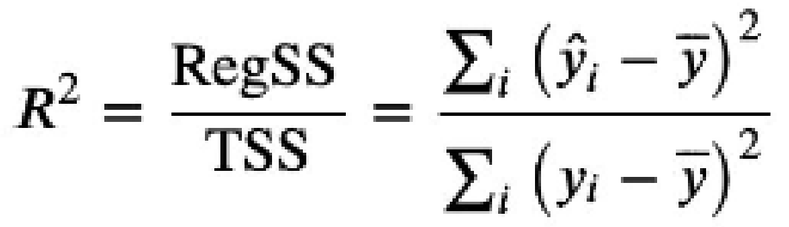
- LSTM model 목표 Spec



MSE



MAE



R^2 score

딥러닝 모델에 일반적으로 사용되는 MSE, MAE, R^2 score를 성능평가 지표로 사용

목표 spec : $R^2 > 0.85$

4.2 Mode2

- 데이터 정제

데이터 로드 및 예측 설정

```
# 상수 정의
DATA_DIR_BASE = 'D:/ansys/new/'  # 데이터 디렉토리 기본 경로
TEMPERATURES = [25, 35, 45, 55, 65, 75, 85, 95, 105, 115,125,135,145,155,165,175,185,195,215,225]  # 온도 리스트
TOTAL_SIZE = 180  # 각 온도에 대한 총 데이터 포인트 수
PREDICTION_RANGE = 300  # 예측할 시간의 범위 (초)
FUTURE_STEPS = PREDICTION_RANGE - TOTAL_SIZE  # 예측할 미래 스텝 수
```

데이터 파일 경로와 온도 리스트를 설정하고, 예측 범위를 정의

데이터 정규화

```
# 데이터 정규화
scaler_data = MinMaxScaler(feature_range=(0, 1))
scaler_temps = MinMaxScaler(feature_range=(0, 1))
scaler_times = MinMaxScaler(feature_range=(0, 1))

data_scaled = scaler_data.fit_transform(data)
temps_scaled = scaler_temps.fit_transform(temperatures)
times_scaled = scaler_times.fit_transform(times)
```

MinMax scaler를 통해 0과 1사이의 값으로 정규화
온도,시간,변위를 동일한 스케일로 맞추어 모델이 특정 변수에 대해 과도하게 민감하게 반응하지 않도록 설정

4.2 Mode2

- LSTM 코드

```
def make_samples(features, labels, window_size):
    X, y = [], []
    for i in range(window_size, len(features)):
        X.append(features[i - window_size:i])  # (온도, 시간)
        y.append(labels[i])  # VALUE
    return np.array(X), np.array(y)

# 데이터 샘플 생성
WINDOW_SIZE = 125
X, y = make_samples(combined_data, data_scaled, WINDOW_SIZE)

# 학습 및 테스트 데이터 분할
train_size = int(0.8 * len(X))
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

데이터 sample 생성 : 지정된 window_size에 따라 시계열 데이터의 일부를 잘라 **입력 sample(x)**와 **label(y)** 생성

Window size 설정 : 시계열 데이터를 처리하기 위해 window_size를 **125**로 설정

데이터 분할 : 전체 데이터를 8:2 비율로 학습 데이터와 테스트 데이터로 분할

4.2 Mode2

- 모델 구축

```
model = Sequential()
model.add(LSTM( units: 32, input_shape=(WINDOW_SIZE, 2), activation='tanh', return_sequences=False))
model.add(Dense(1))

# 모델 컴파일
model.compile(optimizer='adam', loss='mean_squared_error')

# 모델 학습
model.fit(X_train, y_train, epochs=100, batch_size=16)
```

LSTM layer : **32**개의 unit을 가진 하나의 LSTM layer를 사용하여 시퀀스 데이터를 처리, 입력 데이터의 형태는 (**window_size**, 2(온도, 시간)), 활성화 함수로 **tanh** 함수를 사용

출력 layer : 1개의 unit을 가진 dense Layer를 통해 최종 예측값 출력

모델 compile : **Adam** 옵티마이저 사용

학습 : 100번의 epochs동안 **16**의 배치크기로 학습

4.2 Mode2

- 학습 및 성능평가 결과

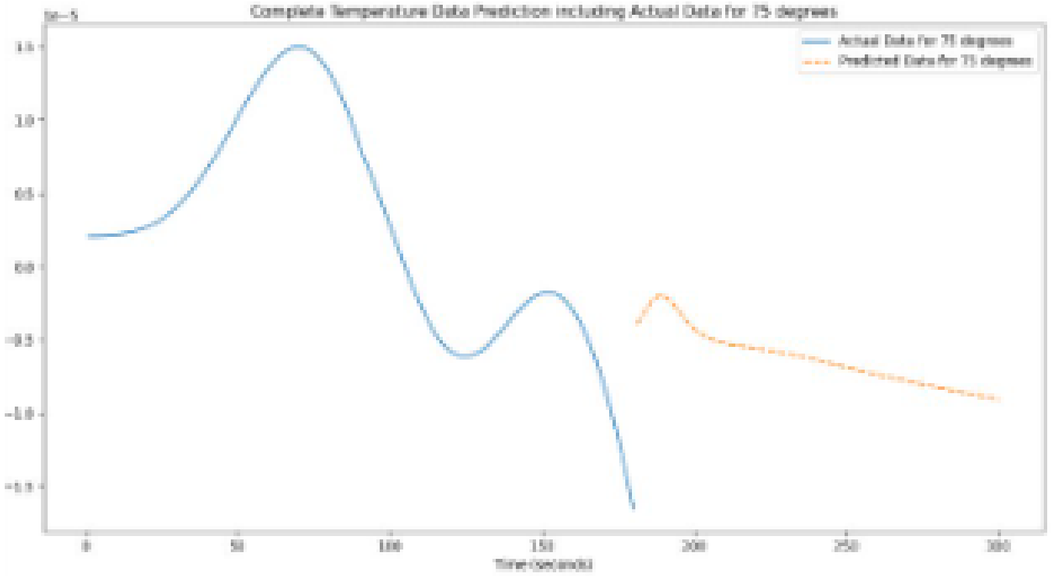
```
Epoch 98/100
174/174 [==============================] - 3s 19ms/step - loss: 0.0026
Epoch 99/100
174/174 [==============================] - 3s 19ms/step - loss: 0.0024
Epoch 100/100
174/174 [==============================] - 3s 19ms/step - loss: 0.0025
22/22 [==============================] - 1s 8ms/step
MSE: 3.9567e-12
MAE: 1.4616e-06
R²: 0.9061
Model saved to D:/ansys/new/Max_model/Max_lstm_model.h5
```

[학습과정]
모델은 마지막 epoch에서 0.0025로 매우 낮은 loss값을 기록하였으며 이를 통해 모델이 잘 수렴함을 확인

[성능지표 분석]
MSE/MAE : 각각 3.9567e-12, 1.4616e-12로 예측의 정확도가 매우 높음
R2 : 결정계수는 0.9061로, 목표치인 0.85 이상을 훨씬 상회하는 것 확인
이는 모델이 90% 이상의 설명력을 가지고 있음을 의미하며 예측 모델이 상당히 신뢰 가능

4.2 Mode2

- 예측 결과



그래프는 75˚C에서의 대류열을 0초부터 300초 가했을 때 기판의 최대 변위값을 실제 데이터와 예측한 데이터를 통해 시각화한 그래프

파란색 실선은 실제 데이터
주황색 점선은 모델이 예측한 데이터

```
75도의 열을 180초 가했을 때 최소 변위값은 -1.646266e-05이며
9초 열을 더 가했을 시 -1.997679e-06로 변위값을 줄일 수 있습니다
75도 열을 189초 가했을 시 -1.997679e-06로 가장 낮은 변위값을 가집니다
75도 열을 300초 가했을 시 -9.031928e-06로 가장 높은 변위값을 가집니다
```

LSTM모델을 통해 시간에 따른 변위값을 성공적으로 예측

Part 5

결론

5-1
아쉬운점

5-2
추후 개선사항

5-3
참고문헌

5.1 아쉬운점

OCR 모델 설계 실패

- OCR 모델을 성공적으로 구현하지 못한 점이 가장 아쉬웠다.

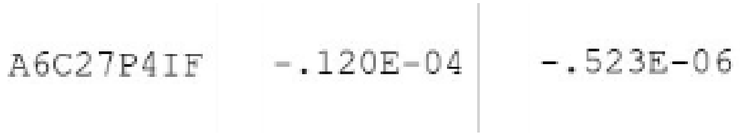
- CRNN, Transformer, EasyOCR 등을 활용해 사전 준비된 이미지 데이터에 사용된 Courier 폰트에 대한 OCR 훈련 모델을 만들고자 했지만, 데이터 처리와 모델 최적화 과정에서 배치(batch) 크기를 맞추지 못하는 문제에 직면했다.

- 이로 인해 충분한 성능을 확보하지 못해 최종 결과물에 포함하지 못했다.

- Courier 폰트에 최적화된 OCR 모델을 만들기 위해, 랜덤한 값으로 구성된 10,000~100,000개의 데이터를 생성했다.

- 이미지에는 소문자 알파벳이 제외되었기 때문에, 대문자 알파벳과 숫자, 그리고 -, . 기호를 추가하여 인식률을 높이고자 했다.

- 밑의 사진은 오른쪽 코드를 실행했을 때 생성되는 데이터의 일부분이다.




```python
import random
from PIL import Image, ImageDraw, ImageFont
import os

1 usage
def create_random_text_image(index, width, height, font_size, image_dir, text_dir):
    # 이미지 생성
    image = Image.new( mode: 'RGB',  size: (width, height), color=(255, 255, 255))
    draw = ImageDraw.Draw(image)
    # Courier 폰트 로드 (시스템에 설치된 Courier 폰트를 사용)
    try:
        font = ImageFont.truetype( font: "cour.ttf", font_size)  # 시스템에 설치된 Courier 폰트를 사용
    except IOError:
        print("Cour font not found")
    # 랜덤 텍스트 생성 (공백 포함하여 9자)
    letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-."
    text = ''.join(random.choice(letters + ' ') for _ in range(9))
    # 텍스트의 너비와 높이 계산
    bbox = draw.textbbox( xy: (0, 0), text, font=font)
    text_width = bbox[2] - bbox[0]
    text_height = bbox[3] - bbox[1]
    # 이미지 중앙에 텍스트 배치
    x = (width - text_width) // 2
    y = (height - text_height) // 2
    draw.text( xy: (x, y), text, fill=(0, 0, 0), font=font)
    # 이미지 저장 디렉토리 확인 및 생성
    if not os.path.exists(image_dir):
        os.makedirs(image_dir)
    # 텍스트 저장 디렉토리 확인 및 생성
    if not os.path.exists(text_dir):
        os.makedirs(text_dir)
    # 파일 이름 설정
    image_filename = os.path.join(image_dir, f'random_text_image_{index}.png')
    text_filename = os.path.join(text_dir, f'random_text_{index}.txt')
    # 이미지 파일 저장
    image.save(image_filename)
    # 텍스트 파일 저장
    with open(text_filename, 'w') as file:
        file.write(text + '\n')
    print(f"Image saved as {image_filename}")
    print(f"Text saved as {text_filename}")
# 이미지 크기, 폰트 크기 설정
width = 121
height = 210
font_size = 20
# 저장할 디렉토리 설정
image_dir = r'D:\courier font\output_images'
text_dir = r'D:\courier font\output_texts'
# 1000개의 랜덤 텍스트 이미지 생성
for i in range(10000):
    create_random_text_image(i + 1, width, height, font_size, image_dir, text_dir)

```

5.1 아쉬운점

1. CRNN을 이용하여 ocr 모델 설계

```python
import os
import cv2
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
import torchvision.transforms as transforms
import numpy as np
from tqdm import tqdm
# 문자 사전
characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-.'
# 디바이스 설정 (GPU가 있으면 cuda, 없으면 cpu)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
# CRNN 모델 정의
4 usages
class CRNN(nn.Module):
    def __init__(self, imgH, nc, nclass, nh):
        super(CRNN, self).__init__()
        self.cnn = nn.Sequential(
            nn.Conv2d(nc,  out_channels: 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d( in_channels: 64,  out_channels: 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d( in_channels: 128,  out_channels: 256, kernel_size=3, stride=1, padding=1),
            nn.ReLU(True),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.rnn_input_size = 256 * (imgH // 8)  # CNN의 마지막 레이어에서 출력되는 feature map의 크기
        self.rnn = nn.LSTM(self.rnn_input_size, nh, bidirectional=True, batch_first=True)
        self.fc = nn.Linear(nh * 2, nclass)

    def forward(self, x):
        conv = self.cnn(x)
        b, c, h, w = conv.size()
        conv = conv.view(b, c * h, w).permute(2, 0, 1)  # (seq_len, batch, input_size) 형태로 변환
        output, _ = self.rnn(conv)  # RNN의 출력을 얻음
        output = self.fc(output)  # Linear 레이어를 통해 최종 출력 얻기
        return output
```

```python
# 데이터셋 클래스 정의
2 usages
class OCRDataset(Dataset):
    def __init__(self, image_dir, text_dir, transform=None):
        self.image_dir = image_dir
        self.text_dir = text_dir
        self.transform = transform
        self.image_paths = [os.path.join(image_dir, f) for f in os.listdir(image_dir) if f.endswith('.png')]
        self.labels = [self._get_label(p) for p in self.image_paths]

    1 usage
    def _get_label(self, img_path):
        base_name = os.path.basename(img_path).replace( __old: '.png',  __new: '.txt')
        label_path = os.path.join(self.text_dir, base_name)
        with open(label_path, 'r') as file:
            return file.readline().strip()

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

    def __getitem__(self, idx):
        img_path = self.image_paths[idx]
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        label = self.labels[idx]

        if self.transform:
            img = self.transform(img)

        return img, label
```

5.1 아쉬운점

1. CRNN을 이용하여 ocr 모델 설계

```
# 데이터 변환
transform = transforms.Compose([
    transforms.ToPILImage(),  # OpenCV에서 읽은 이미지를 PIL 이미지로 변환
    transforms.Resize((32, 128)),  # 이미지 크기 조정 (높이, 너비)
    transforms.ToTensor(),  # 이미지를 텐서로 변환
    transforms.Normalize( mean: (0.5,), std: (0.5,)) # 정규화
])

# 데이터셋 및 데이터로더 설정
train_dataset = OCRDataset(
    image_dir='D:/courier font/output_images',
    text_dir='D:/courier font/output_texts',
    transform=transform
)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, num_workers=4)

val_dataset = OCRDataset(
    image_dir='D:/courier font/output_images',  # 검증 데이터도 동일한 디렉토리에서 가져온다고 가정
    text_dir='D:/courier font/output_texts',
    transform=transform
)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False, num_workers=4)

# CTC 손실 정의
criterion = nn.CTCLoss(blank=0)

# 모델 초기화
nclass = len(characters) + 1  # 문자 클래스 수 + 1 (CTC의 빈 문자(blank)를 위해)
nh = 256  # RNN의 hidden state 크기
model = CRNN(imgH=32, nc=1, nclass=nclass, nh=nh).to(device)

# 옵티마이저 정의
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 문자 사전 매핑
char_to_idx = {char: i + 1 for i, char in enumerate(characters)}
```

```
2 usages
def encode_text(text):
    """문자열을 숫자 인덱스 리스트로 변환"""
    return [char_to_idx.get(char, 0) for char in text]  # 알 수 없는 문자에 대해 0 (빈 문자) 사용


1 usage
def decode_predictions(preds):
    """모델의 예측을 문자열로 변환"""
    pred_labels = []
    for pred in preds:
        pred = pred.detach().cpu().numpy()  # 텐서를 numpy 배열로 변환
        pred = np.argmax(pred, axis=1)  # 가장 높은 확률의 인덱스 선택
        decoded = ''
        prev_char_idx = -1
        for char_idx in pred:
            if char_idx != prev_char_idx:  # 동일한 인덱스가 연속으로 나오지 않도록 필터링
                if char_idx != 0:  # 빈 문자가 아닌 경우
                    decoded += characters[char_idx - 1]
                prev_char_idx = char_idx
        pred_labels.append(decoded)
    return pred_labels
```

5.1 아쉬운점

1. CRNN을 이용하여 ocr 모델 설계

```python
1 usage
def train_and_evaluate(model, train_loader, val_loader, num_epochs, model_path):
    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        for imgs, labels in tqdm(train_loader, desc=f"Epoch {epoch + 1}/{num_epochs}"):
            imgs = imgs.to(device)
            labels = [encode_text(label) for label in labels]
            labels_lengths = torch.tensor([len(label) for label in labels], dtype=torch.long).to(device)

            # Compute input_lengths
            input_lengths = torch.full( size: (imgs.size(0),), imgs.size(3), dtype=torch.long).to(device)

            # Flatten labels for CTCLoss
            concatenated_labels = torch.cat([torch.tensor(label).to(device) for label in labels])

            optimizer.zero_grad()
            preds = model(imgs)
            preds = preds.permute(1, 0, 2)  # (batch, seq_len, nclass) -> (seq_len, batch, nclass)

            try:
                # CTCLoss에 필요한 입력과 정답을 올바르게 설정
                loss = criterion(preds, concatenated_labels, input_lengths, labels_lengths)
            except Exception as e:
                print(f"Error during loss computation: {e}")
                continue

            loss.backward()
            optimizer.step()
            running_loss += loss.item()

        epoch_loss = running_loss / len(train_loader)
        print(f'Epoch {epoch + 1}, Training Loss: {epoch_loss:.4f}')


    # 모델 저장
    torch.save(model.state_dict(), model_path)

    # 모델 평가
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for imgs, labels in val_loader:
            imgs = imgs.to(device)
            labels = [encode_text(label) for label in labels]
            labels_lengths = torch.tensor([len(label) for label in labels], dtype=torch.long).to(device)

            # Compute input_lengths
            input_lengths = torch.full( size: (imgs.size(0),), imgs.size(3), dtype=torch.long).to(device)

            preds = model(imgs)
            preds = preds.permute(1, 0, 2)  # (batch, seq_len, nclass) -> (seq_len, batch, nclass)
            pred_labels = decode_predictions(preds)

            # 레이블 비교
            for pred_label, true_label in zip(pred_labels, labels):
                true_label = ''.join([characters[i - 1] for i in true_label])  # 레이블 복원
                if pred_label == true_label:
                    correct += 1
                total += 1

    print(f'Validation Accuracy: {correct / total:.4f}')
```

5.1 아쉬운점

1. CRNN을 이용하여 ocr 모델 설계

```
193
194  # 모델 초기화 및 학습
195  def main():
196      model_path = 'crnn_model.pth'
197      num_epochs = 2
198      if os.path.exists(model_path):
199          print("저장된 모델을 불러옵니다.")
200          model = CRNN(imgH=32, nc=1, nclass=len(characters) + 1, nh=256).to(device)
201          model.load_state_dict(torch.load(model_path))
202      else:
203          print("새로운 모델을 초기화합니다.")
204          model = CRNN(imgH=32, nc=1, nclass=len(characters) + 1, nh=256).to(device)
205
206      train_and_evaluate(model, train_loader, val_loader, num_epochs, model_path)
207
208
209  if __name__ == '__main__':
210      main()
211
```

5.1 아쉬운점

2. VIT를 이용하여 ocr 모델 설계

```python
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
from PIL import Image
import torchvision.transforms as transforms
from transformers import ViTModel, BertTokenizer, BertForTokenClassification

1 usage
class ImageTextDataset(Dataset):
    def __init__(self, image_folder, text_folder, tokenizer):
        self.image_folder = image_folder
        self.text_folder = text_folder
        self.tokenizer = tokenizer
        self.transform = transforms.Compose([
            transforms.Resize((224, 224)),
            transforms.ToTensor(),
        ])
        self.image_files = [f for f in os.listdir(image_folder) if f.endswith(('.png', '.jpg', '.jpeg'))]

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

    def __getitem__(self, idx):
        image_file = self.image_files[idx]
        image_path = os.path.join(self.image_folder, image_file)
        text_file = image_file.replace( _old: '.png',  _new: '.txt').replace( _old: '.jpg',  _new: '.txt').replace( _old: '.jpeg',  _new: '.txt')
        text_path = os.path.join(self.text_folder, text_file)

        image = Image.open(image_path).convert("RGB")
        image = self.transform(image)

        with open(text_path, 'r') as file:
            text = file.read().strip()

        text_inputs = self.tokenizer(text, return_tensors="pt", padding="max_length", truncation=True, max_length=512)
        labels = text_inputs.input_ids.clone()

        return image, text_inputs.input_ids.squeeze(), text_inputs.attention_mask.squeeze(), labels.squeeze()
```

```python
2 usages
class OCRModel(nn.Module):
    def __init__(self, vit_model, bert_model):
        super(OCRModel, self).__init__()
        self.vit = vit_model
        self.bert = bert_model
        # ViT의 출력 차원을 BERT의 입력 차원에 맞추기 위해 FC 레이어 추가
        self.fc = nn.Linear( in_features: 768,  out_features: 256)  # Change 768 to 256 to match BERT hidden size
        self.output_layer = nn.Linear( in_features: 256, bert_model.config.num_labels)

    def forward(self, pixel_values, input_ids, attention_mask, labels):
        # ViT에서 이미지 특징 추출
        vit_outputs = self.vit(pixel_values=pixel_values)
        image_features = vit_outputs.last_hidden_state[:, 0, :]  # [batch_size, hidden_dim]

        # FC 레이어를 통과하여 BERT와 동일한 차원으로 변환
        image_features = self.fc(image_features)

        # BERT 모델의 텍스트 특징 추출
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
        text_features = outputs.logits

        # 이미지와 텍스트 특징을 결합 (여기서는 단순히 덧셈 예시)
        combined_features = image_features.unsqueeze(1).expand(-1, text_features.size(1), -1) + text_features
        logits = self.output_layer(combined_features)

        return logits
```

5.1 아쉬운점

2. VIT를 이용하여 ocr 모델 설계

```
1 usage
def train_model(model, dataloader, device='cuda'):
    optimizer = optim.Adam(model.parameters(), lr=1e-5)
    model.train()
    model.to(device)

    for epoch in range(15):
        for batch in dataloader:
            images, input_ids, attention_mask, labels = batch
            images = images.to(device)
            input_ids = input_ids.to(device)
            attention_mask = attention_mask.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()

            outputs = model(images, input_ids, attention_mask, labels)
            loss = nn.CrossEntropyLoss()(outputs.view(-1, outputs.size(-1)), labels.view(-1))

            loss.backward()
            optimizer.step()

            print(f"Epoch {epoch+1}, Loss: {loss.item()}")
```

```
def main():
    image_folder = r'D:\courier font\output_images'
    text_folder = r'D:\courier font\output_texts'

    tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
    vit_model = ViTModel.from_pretrained('google/vit-base-patch16-224-in21k')
    bert_model = BertForTokenClassification.from_pretrained( pretrained_model_name_or_path: 'bert-base-uncased', num_labels=256)

    dataset = ImageTextDataset(image_folder, text_folder, tokenizer)
    dataloader = DataLoader(dataset, batch_size=4, shuffle=True)

    model = OCRModel(vit_model, bert_model)

    model_save_dir = r'D:\courier font\models'
    os.makedirs(model_save_dir, exist_ok=True)
    model_save_path = os.path.join(model_save_dir, 'ocr_model.pth')

    = train_model(model, dataloader)

    torch.save(model.state_dict(), model_save_path)
    print(f"Model saved to {model_save_path}")

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

5.1 아쉬운점

3. Easyocr를 이용한 코드

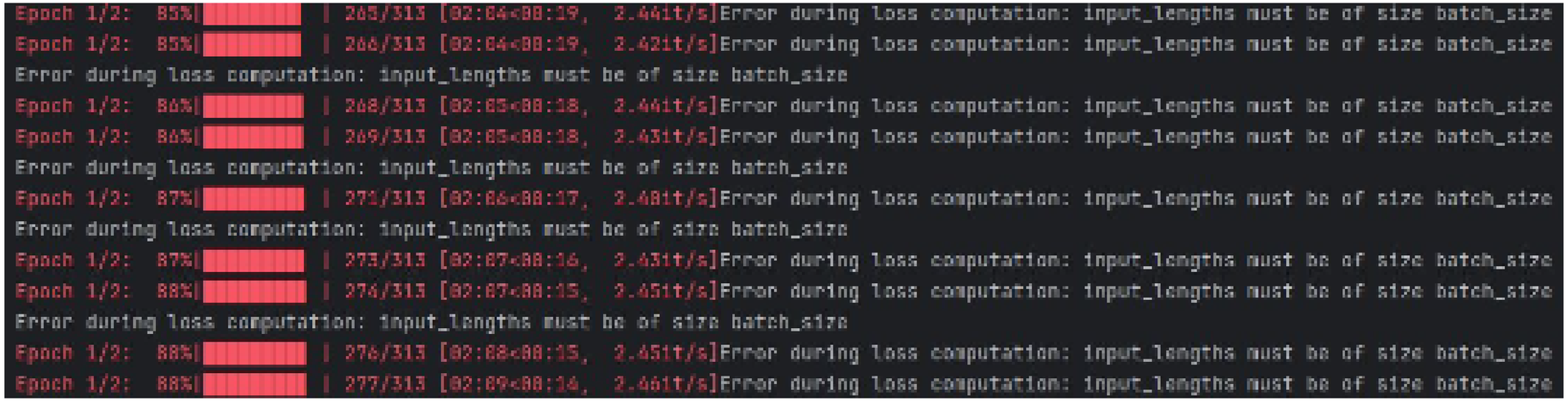
```python
import os
import easyocr
import cv2
import json

os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"

# 이미지 전처리 함수
1 usage
def preprocess_image(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    image = cv2.resize(image, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)
    _, image = cv2.threshold(image, 150, 255, cv2.THRESH_BINARY_INV)
    return image

# EasyOCR 모델 초기화
reader = easyocr.Reader(['ko', 'en'])

# 이미지 파일들이 있는 폴더 경로
folder_path = r'D:\ocr'

# 추출한 텍스트를 저장할 딕셔너리
extracted_texts = {}

# 폴더 내의 모든 PNG 파일 처리
for filename in os.listdir(folder_path):
    if filename.endswith('.png'):
        image_path = os.path.join(folder_path, filename)
        image = preprocess_image(image_path)  # 전처리된 이미지 사용

        # 여러 설정으로 텍스트 읽기
        results1 = reader.readtext(image, contrast_ths=0.7, adjust_contrast=0.5)
        results2 = reader.readtext(image, contrast_ths=0.5, adjust_contrast=0.3)

        # 신뢰도가 더 높은 결과 선택
        final_results = results1 if sum([prob for _, _, prob in results1]) > sum([prob for _, _, prob in results2]) else results2

        # 이미지 파일명을 키로 텍스트 저장
        image_texts = [text for (_, text, _) in final_results]
        extracted_texts[filename] = image_texts

# 추출된 텍스트를 JSON 파일로 저장
output_json_path = os.path.join(folder_path, "extracted_texts.json")
with open(output_json_path, "w", encoding="utf-8") as json_file:
    json.dump(extracted_texts, json_file, indent=4, ensure_ascii=False)

print(f"텍스트 추출 및 저장이 완료되었습니다. JSON 파일 경로: {output_json_path}")
```

5.1 아쉬운점

- CRNN, VIT 중간 결과

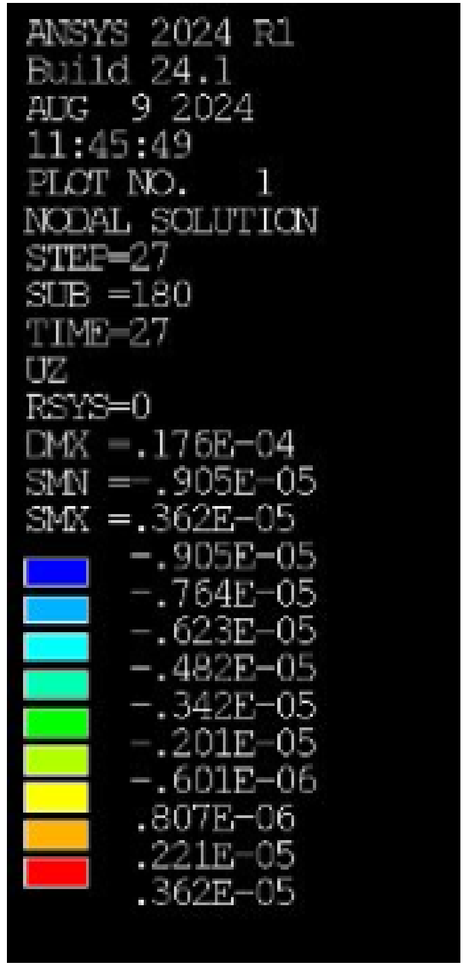
  RCNN과 VIT를 이용하여 ocr 모델을 만드는 과정에서 배치 사이즈 불일치 문제로 코드 상에서 문제가 계속 발생하였다.

  원인을 해결을 하지 못하여 3번째 방법인 easyocr을 이용하여 문제를 해결하려 했으나 이미지 파일에서 텍스트로 변환하는 과정에서 특정 기호를 제대로 인식하지 못하는 문제가 발생했다.

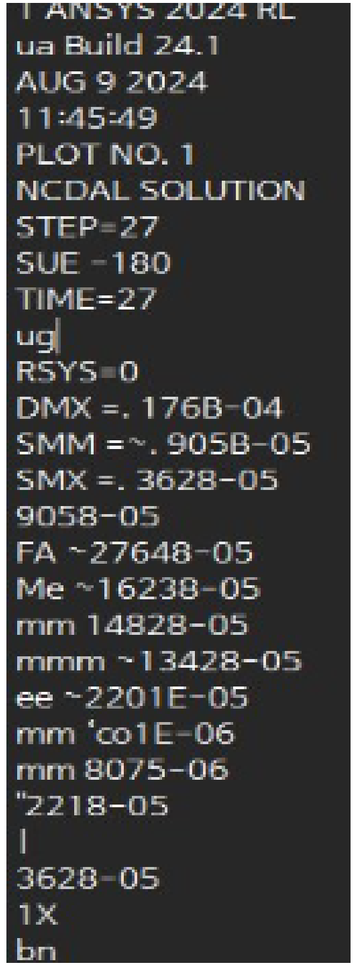


5.1 아쉬운점

- Easy OCR 중간 결과



원본 입력 이미지



출력된 텍스트

기호를 추가하고, easyocr을 사용하였으나, curier font(이미지에 적용된 폰트)에서 성능적인 부분에서 좋지 않아 사용할 수 없었다.

5.1 아쉬운점

- **API 방식의 한계**: LSTM을 사용한 학습 과정에서 Clova OCR API를 활용하여 이미지에서 텍스트 데이터를 추출하려 했으나, API 사용에 따른 비용 문제가 발생하였다. 이로 인해 현실적으로 API 사용이 어려웠으며, 대신 이미지 데이터 생성 과정에서 직접 추출한 텍스트 데이터를 활용하여 LSTM 모델을 학습시키는 방안을 채택하였다.

- **물리적 법칙을 고려하지 못함**: LSTM 모델은 순수하게 데이터에 의존하여 패턴을 학습하기 때문에 이 모델은 실제 물리적 법칙 (ex: 열 전달 방정식, 열팽창 방정식 등)은 고려하지 못하고 데이터를 기반으로만 예측을 수행하였고, 이는 물리적 현실과 일치하지 않는 결과를 도출할 가능성이 존재한다.

- **시간 확장에 따른 정확도 저하 가능성**: 현재 모델은 테스트 데이터를 통해 0.9 이상의 R2-score값을 도출했기 때문에, 300초까지의 예측에서는 큰 문제가 발생하지 않을 것으로 보이지만, 그 이후의 시간에 대한 예측에서는 데이터의 부족과 모델의 한계로 인해 정확도가 낮아질 가능성이 존재한다.

5.2 추후 개선사항

- **텍스트 인식 모델 사용**: 추후 OCR API를 대신할 고성능의 텍스트 인식 모델을 설계하여 사용함으로써 비용 절감과 데이터 추출의 정확성을 동시에 향상시킬 수 있다. 이를 통해 고품질의 데이터 추출하여 전반적인 모델 성능을 더욱 높일 수 있을 것을 기대할 수 있다.

- **데이터 기반 모델과 물리 기반 모델의 결합**: LSTM과 같은 데이터 기반의 모델과 유한 요소 분석(Finite Element Analysis) 등 물리 기반 모델을 결합하여, 물리적 법칙을 반영해 보다 현실적인 예측을 수행할 수 있다. 이러한 접근은 물리적 현상을 더욱 정확하게 반영할 수 있어, 예측의 신뢰성을 높일 수 있다.

- **추가 데이터 확보 및 전이학습**: 추가적인 시뮬레이션 데이터 확보, 데이터 증강 등의 방식을 통해 장기적인 추가 데이터를 확보할 수 있고, 전이학습을 통해 모델을 재학습시켜 모델이 제한된 데이터에서도 장기적인 패턴을 수행하도록 할 수 있다.

5.3 참고문헌 및 참고 사이트

IPC-2221
IPC-6012

Edmund Sek, Chun Hei, et al. "Dynamic warpage simulation of molded PCB under reflow process." Circuit World 49.2 (2023): 202-210.

Kim, Dasol, Mintae Kim, and Wooju Kim. "Wafer edge yield prediction using a combined long short-term memory and feed-forward neural network model for semiconductor manufacturing." IEEE Access 8 (2020): 215125-215132.

황순환, 한성렬, and 이후진. "CAE 와 Decision-tree를 이용한 사출성형 공정개선에 관한 연구." 한국산학기술학회 논문지 22.4 (2021): 580-586.

CLOVAOCR.API : https://www.ncloud.com/product/aiService/ocr

감사합니다

전자구이통닭