지능화 캡스톤 프로젝트

프로젝트 #1 결과 발표

(CNN을 이용한 Wafermap 불량 분류)

2022. 06. 15

충북대학교 산업인공지능학과

[20-8조] 고정재, 유용주

INDUSTRIAL AI RESEARCH CENTER

수행방법 및 기여도

❖ 수행방법

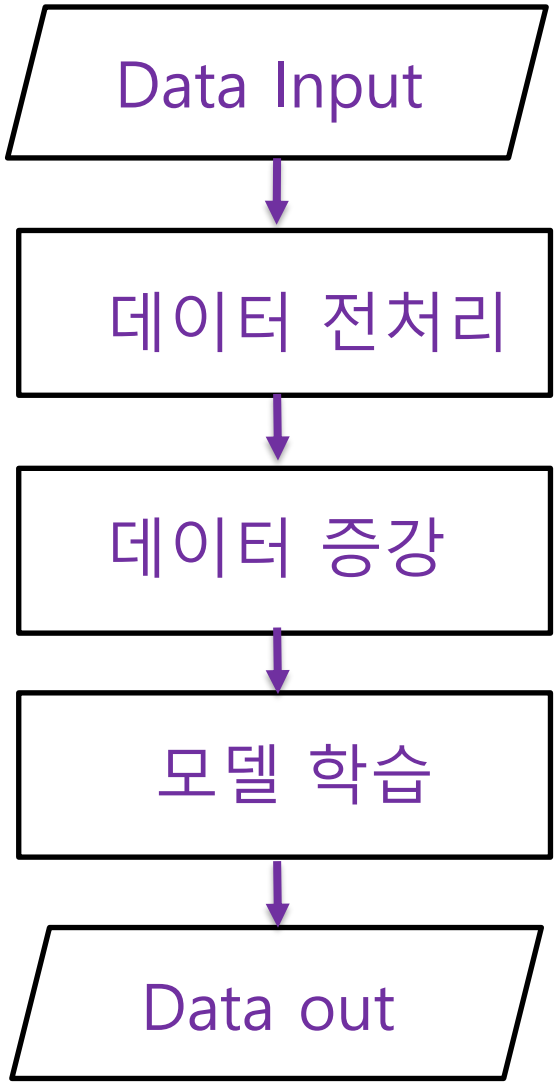
- 같은 회사에 재직중이나 팀이 달라 일과 시간 이후에 프로젝트 수행.
- 회사 업무가 전공분야가 아니라, CNN에 대해 공부를 하면서 진행 함.
- 코딩 능력이 부족해 캐글이나 github에 있는 자료를 참고하여 실습을 한다고 생각하며 수행.

❖ 업무분장 및 기여도

이름	비중	수행내용	비고
유용주	50%	• CNN에 대한 공부 자료 수집 • 발표 자료 작성 • 발표	
고정재	50%	• 데이터 전처리 • 데이터 증량 • 발표	

데이터셋

❖ 진행 흐름도



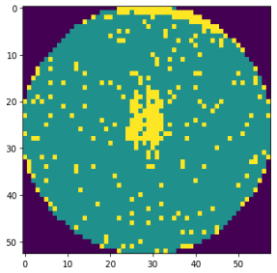
데이터셋

1. Dataset

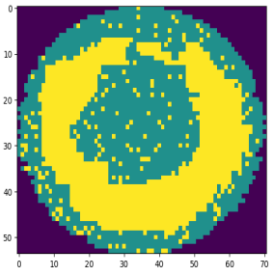
- Kaggle에 있는 wm811k-wafer-map을 활용
- Total 811,457개의 Data 구성 → Label(x) = 638,507, Label(o) = 172,950
  → With pattern : 25,519
  → Non pattern : 147,431

2. Dataset 분석

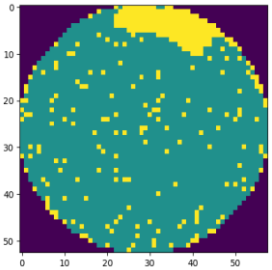
- 9 types (8 defect types + 1 non defect type)
- 결함 종류 : Center, Donut, Edge-Loc, Edge-Ring, Loc, Random, Scratch, Near-full, none.



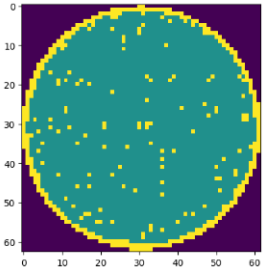
center



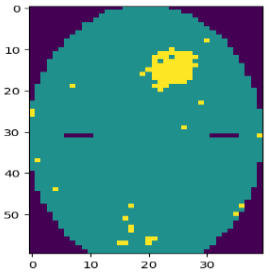
donut



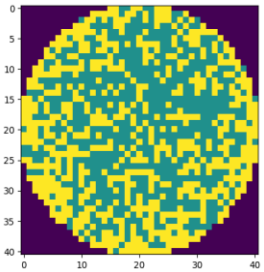
Edge-local



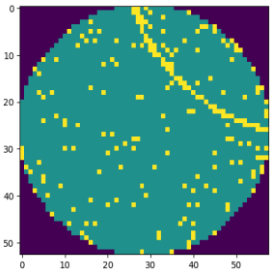
Edge-ring



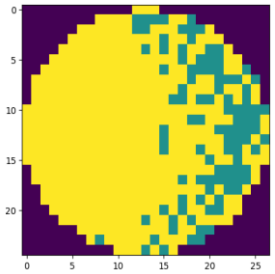
local



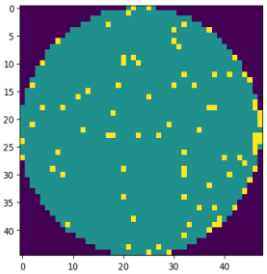
random



scratch

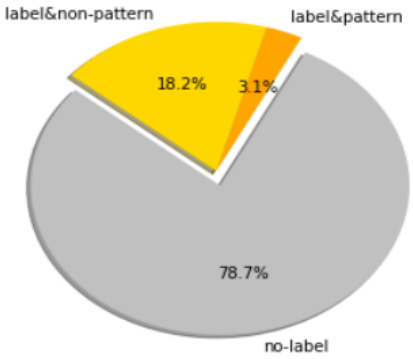


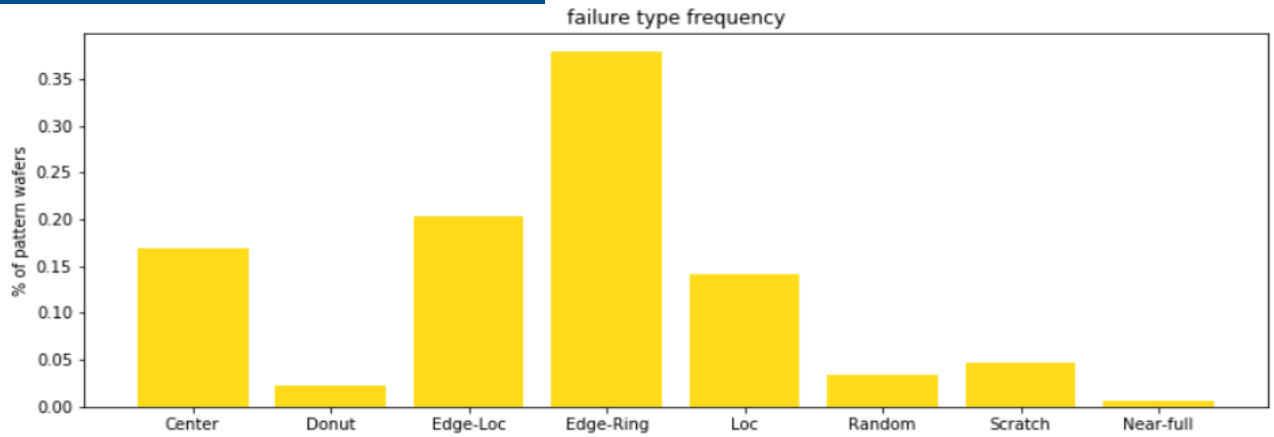
Near-full



none

데이터셋





※ 데이터의 불균형으로 overfitting 발생 가능 존재하여 데이터 augmentation 필요

3. 결함 원인

- **center : 불량 패턴이 웨이퍼 정 가운데에 위치한 경우**
  →RF(Radio Frequency) 출력의 이상으로 인해 웨이퍼 외부 속도에서 발생하는 액압의 이상
- **dount : 불량 패턴이 웨이퍼 가장자리에 위치한 경우**
  →플라즈마로 인한 산화물 손상은 전류 또는 전압 스트레스로 인한
- **edge-local : 불량 패턴이 웨이퍼 가장자리에 위치하며, 볼록하며(Convex) 두께 있는 덩어리 형태를 가질 경우**
- **edge-ring : 불량 패턴이 안이 고리 형태 혹은 고리의 일부의 형태를 가질 경우**
  → 연마압력, 연마헤드 및 테이블 속도, 웨이퍼의 처리시간, 웨이퍼 표면에서 일어나는 화학반응, 슬러리의 유체역학적 상태, 패드의 특성
- **local : 불량 패턴이 볼록하며(Convex) 두께 있는 덩어리 형태를 가질 경우**
  →슬릿 밸브 누출, 로봇 핸드오프 중 이상 또는 펌프 이상
- **random : 웨이퍼 맵 상에 군집을 이룬 불량 칩이 near-full보다 적고 none보다 많을 경우**
  →오염된 파이프, 샤워 헤드의 이상 또는 제어 웨이퍼의 이상으로 인해 발생합니다.
- **scratch : 불량 패턴이 얇은 선의 형태로 위치한 경우**
  →주로 로봇 핸드오프 중 이상 또는 웨이퍼 충격으로 인해 발생
- **near-full : 웨이퍼 맵 상에 군집을 이룬 불량 칩이 대부분 일 경우**
- **none : 웨이퍼 맵 상에 군집을 이룬 불량 칩이 없을 경우**

데이터셋(Convolutional Autoencoder)

- Data 확대의 필요성
  → Data 불균형 문제가 존재하여 해결하기 위해 Data 확대 진행
  → wafermap은 이미지 데이터라 Convolutional Autoencoder를 이용 함.

```
1 # Encoder
2 input_shape = (26, 26, 3)
3 input_tensor = Input(input_shape)
4 encode = layers.Conv2D(64, (3,3), padding='same', activation='relu')(input_tensor)
5
6 latent_vector = layers.MaxPool2D()(encode)
7
8 # Decoder
9 decode_layer_1 = layers.Conv2DTranspose(64, (3,3), padding='same', activation='relu')
10 decode_layer_2 = layers.UpSampling2D()
11 output_tensor = layers.Conv2DTranspose(3, (3,3), padding='same', activation='sigmoid')
12
13 # connect decoder layers
14 decode = decode_layer_1(latent_vector)
15 decode = decode_layer_2(decode)
16
17 ae = models.Model(input_tensor, output_tensor(decode))
18 ae.compile(optimizer = 'Adam',
19              loss = 'mse',
20            )
```

```
[23]  1 ae.summary()

Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 26, 26, 3)]       0

 conv2d (Conv2D)             (None, 26, 26, 64)        1792

 max_pooling2d (MaxPooling2D  (None, 13, 13, 64)       0
 )

 conv2d_transpose (Conv2DTra  (None, 13, 13, 64)       36928
 nspose)

 up_sampling2d (UpSampling2D  (None, 26, 26, 64)       0
 )

 conv2d_transpose_1 (Conv2DT  (None, 26, 26, 3)        1731
 ranspose)

=================================================================
Total params: 40,451
Trainable params: 40,451
Non-trainable params: 0
```

Autoencoder란 ?

- 차원이 적은 데이터가 있어도 다시 복원할 수 있도록 특징을 찾아내는 것이 목표.
- 영상 의학 분야 등 아직 데이터 수가 충분 하지 않은 분야에서 사용.
- 부족한 학습 데이터 수를 효과적으로 늘려주는 특징.

데이터셋(Convolutional Autoencoder)

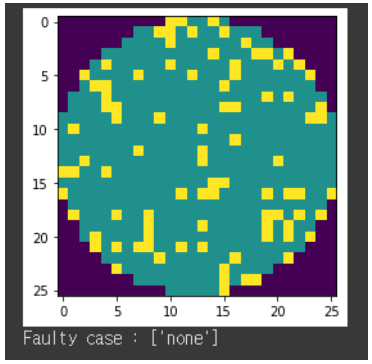
- Autoencoder model 레이어의 일부로 인코더 모델, 디코더 모델을 제작 하였고, 이는 노이즈 추가 하기 위함.
- 인코딩된 결함 웨이퍼 벡터에 노이즈를 추가 함.

```
# start train
ae.fit(new_x, new_x,
       batch_size=batch_size,
       epochs=epoch,
       verbose=2)
```

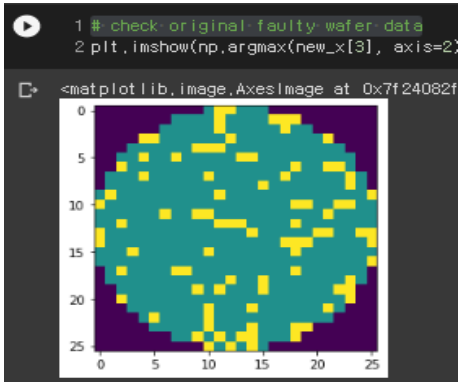
```
Epoch 1/15
15/15 - 33s - loss: 0.1641 - 33s/epoch - 2s/step
Epoch 2/15
15/15 - 32s - loss: 0.0997 - 32s/epoch - 2s/step
Epoch 3/15
15/15 - 32s - loss: 0.0842 - 32s/epoch - 2s/step
Epoch 4/15
15/15 - 31s - loss: 0.0742 - 31s/epoch - 2s/step
Epoch 5/15
15/15 - 33s - loss: 0.0654 - 33s/epoch - 2s/step
Epoch 6/15
15/15 - 32s - loss: 0.0590 - 32s/epoch - 2s/step
Epoch 7/15
15/15 - 32s - loss: 0.0536 - 32s/epoch - 2s/step
Epoch 8/15
15/15 - 32s - loss: 0.0490 - 32s/epoch - 2s/step
Epoch 9/15
15/15 - 32s - loss: 0.0446 - 32s/epoch - 2s/step
Epoch 10/15
15/15 - 32s - loss: 0.0401 - 32s/epoch - 2s/step
Epoch 11/15
15/15 - 32s - loss: 0.0358 - 32s/epoch - 2s/step
Epoch 12/15
15/15 - 32s - loss: 0.0320 - 32s/epoch - 2s/step
Epoch 13/15
15/15 - 32s - loss: 0.0288 - 32s/epoch - 2s/step
Epoch 14/15
15/15 - 32s - loss: 0.0260 - 32s/epoch - 2s/step
Epoch 15/15
15/15 - 32s - loss: 0.0238 - 32s/epoch - 2s/step
<keras.callbacks.History at 0x7f24078c8190>
```

데이터셋(Augmentation)

- Data 전처리를 하여 Class 불균형 해결

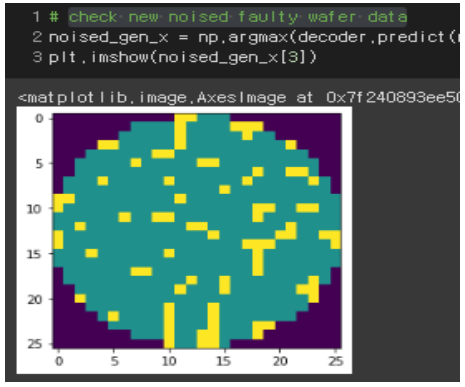


Wafermap의 크기가 다양하여 26 X 26 resizing



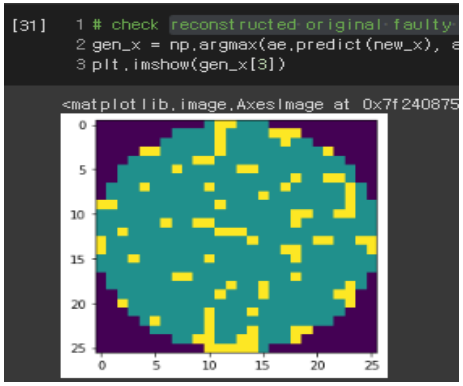
<원본 결함 웨이퍼 데이터>

+



<신규 노이즈 웨이퍼 데이터>

=



<재구성된 웨이퍼 데이터>

```
Center : 90
Donut : 1
Edge-Loc : 296
Edge-Ring : 31
Loc : 297
Near-full : 16
Random : 74
Scratch : 72
none : 13489
```

기존 Dataset

이러한 과정을 거쳐 웨이퍼의 불량 타입별 총 갯수를 2000개로 데이터 증량하여 불균형 해소함.

```
Center : 2160
Donut : 2002
Edge-Loc : 2368
Edge-Ring : 2046
Loc : 2376
Near-full : 2032
Random : 2146
Scratch : 2088
none : 2489
```

변경 Dataset

교차 검증

Simple 2D CNN Model 웨이퍼 데이터가 이미지이기 때문에, CNN 모델을 분류에 사용한다.

모델 생성 함수를 정의하고, Scikit-Learn교차 검증을 통해 모델을 검증

```python
def create_model():
    input_shape = (26, 26, 3)
    input_tensor = Input(input_shape)

    conv_1 = layers.Conv2D(16, (3,3), activation='relu', padding='same')(input_tensor)
    conv_2 = layers.Conv2D(64, (3,3), activation='relu', padding='same')(conv_1)
    conv_3 = layers.Conv2D(128, (3,3), activation='relu', padding='same')(conv_2)

    flat = layers.Flatten()(conv_3)

    dense_1 = layers.Dense(512, activation='relu')(flat)
    dense_2 = layers.Dense(128, activation='relu')(dense_1)
    output_tensor = layers.Dense(9, activation='softmax')(dense_2)

    model = models.Model(input_tensor, output_tensor)
    model.compile(optimizer='Adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

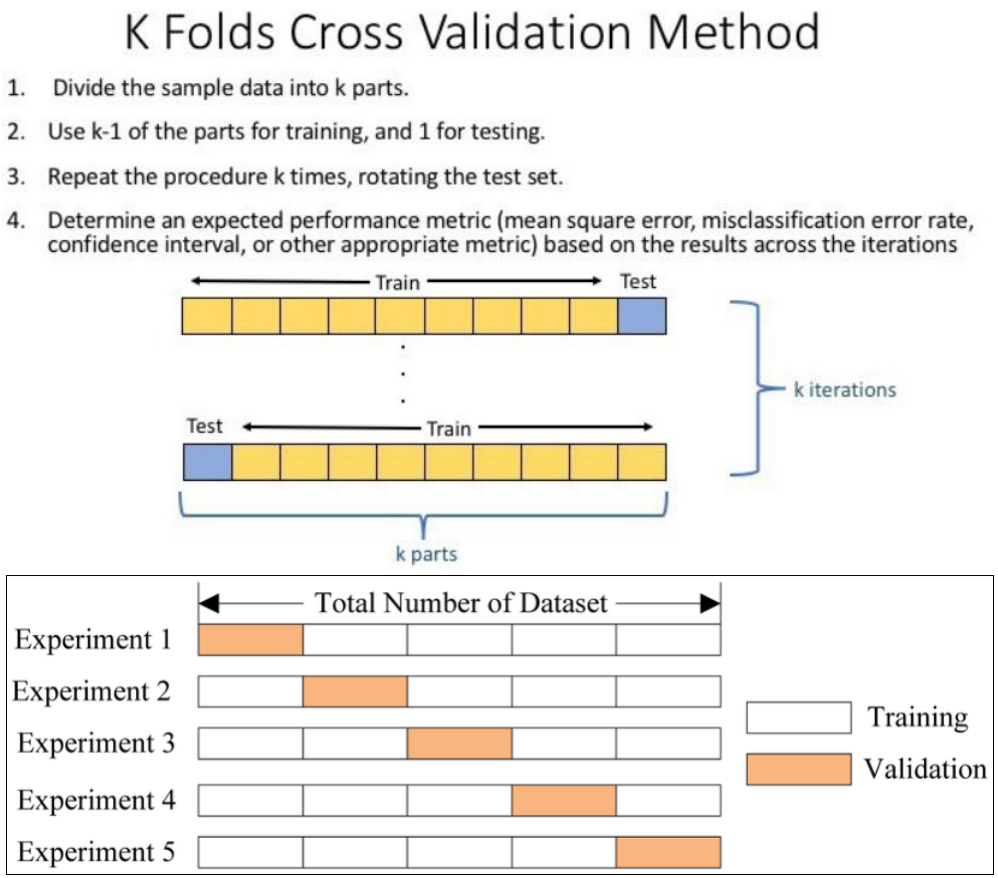
```python
# Make keras model to sklearn classifier.
model = KerasClassifier(build_fn=create_model, epochs=15, batch_size=1024, verbose=2)
# 3-Fold Crossvalidation
kfold = KFold(n_splits=3, shuffle=True, random_state=2019)
results = cross_val_score(model, x_train, y_train, cv=kfold)
# Check 3-fold model's mean accuracy
print('Simple CNN Cross validation score : {:.4f}'.format(np.mean(results)))
```

모델 생성 함수 정의

K Fold Cross검증을 사용하여 CNN을 검증

K Fold는 K개의 Fold를 만들어서 진행하는 교차 검증이다

사용 이유 – 총 데이터 개수가 적은 데이터 셋에 대하여 정확도를 향상 시킬 수 있음.

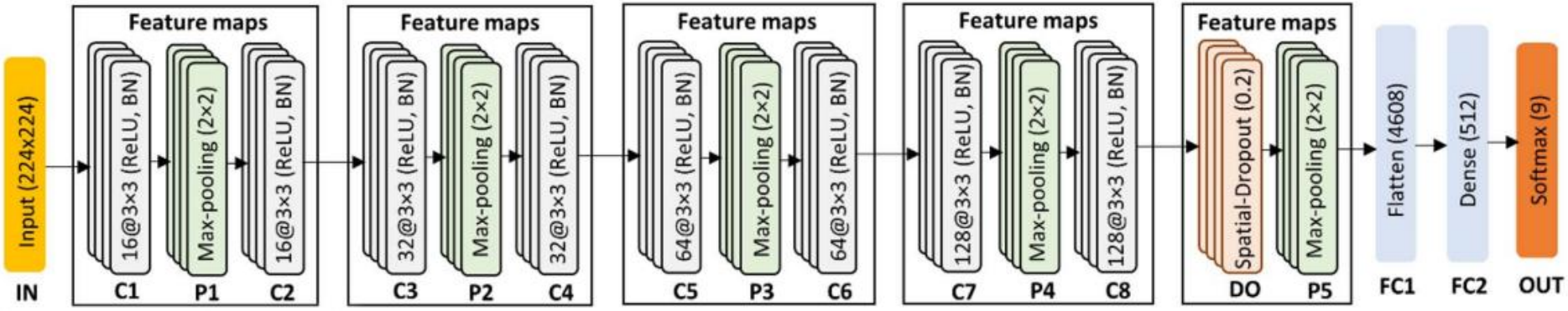


K Folds Cross Validation Method

K를 3으로 지정하고K Folds Score = 98%

CNN 구조

CNN 구조



*Note: IN denotes input layer; C convolutional layer; P pooling layer; DO dropout layer; FC fully connected layer; OUT output layer; and BN batch normalization

TABLE I
PROPOSED DEEP CNN MODEL PARAMETERS

Layer	Type	Feature Maps	Output Size	Filter size	Padding	Activation
IN	Input	3 (RGB)	224×224	–	–	–
C1	Convolution1	16	222×222	3×3	No	ReLU
P1	Max Pooling1	16	111×111	2×2	No	–
C2	Convolution2	16	111×111	3×3	Yes	ReLU
C3	Convolution3	32	111×111	3×3	Yes	ReLU
P2	Max Pooling2	32	55×55	2×2	No	–
C4	Convolution4	32	55×55	3×3	Yes	ReLU
C5	Convolution5	64	55×55	3×3	Yes	ReLU
P3	Max Pooling3	64	27×27	2×2	No	–
C6	Convolution6	64	27×27	3×3	Yes	ReLU
C7	Convolution7	128	27×27	3×3	Yes	ReLU
P4	Max Pooling4	128	13×13	2×2	No	–
C8	Convolution8	128	13×13	3×3	Yes	ReLU
P5	Max Pooling5	128	6×6	2×2	No	–
FC1	Fully-Connected1	1	4608	–	–	ReLU
FC2	Fully Connected2	1	512	–	–	ReLU
OUT	Output	1	9	–	–	Softmax

과적합을 방지하기 위한 규제화(regulation)

- *Batch Normalization(정규화)*
- *Spatial Dropout = 0.2*

CNN 구조

주요 코드 및 실행 결과

- 활성화 함수 : relu, optimizer : Adam, loss함수 : categorical_crossentropy

```
def create_model():
    input_shape = (26, 26, 3)
    input_tensor = Input(input_shape)

    conv_1 = layers.Conv2D(16, (3,3), activation='relu', padding='same')(input_tensor)
    conv_2 = layers.Conv2D(64, (3,3), activation='relu', padding='same')(conv_1)
    conv_3 = layers.Conv2D(128, (3,3), activation='relu', padding='same')(conv_2)

    flat = layers.Flatten()(conv_3)

    dense_1 = layers.Dense(512, activation='relu')(flat)
    dense_2 = layers.Dense(128, activation='relu')(dense_1)
    output_tensor = layers.Dense(9, activation='softmax')(dense_2)

    model = models.Model(input_tensor, output_tensor)
    model.compile(optimizer='Adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

CNN 구조

주요 코드 및 실행 결과

- 프레임워크(tensorflow, keras)

```
# Encoder
input_shape = (26, 26, 3)
input_tensor = Input(input_shape)
encode = layers.Conv2D(64, (3,3), padding='same', activation='relu')(input_tensor)

latent_vector = layers.MaxPool2D()(encode)

# Decoder
decode_layer_1 = layers.Conv2DTranspose(64, (3,3), padding='same', activation='relu')
decode_layer_2 = layers.UpSampling2D()
output_tensor = layers.Conv2DTranspose(3, (3,3), padding='same', activation='sigmoid')

# connect decoder layers
decode = decode_layer_1(latent_vector)
decode = decode_layer_2(decode)

ae = models.Model(input_tensor, output_tensor(decode))
ae.compile(optimizer = 'Adam',
              loss = 'mse',
             )
```

CNN 구조

주요 코드 및 실행 결과(실습)

```
1 import os
2 from os.path import join
3
4 import numpy as np
5 import pandas as pd
6
7 import tensorflow as tf
8 import keras
9 from keras import layers, Input, models
10 from tensorflow.keras.utils import to_categorical
11 from keras.wrappers.scikit_learn import KerasClassifier
12 from sklearn.model_selection import KFold
13 from sklearn.model_selection import cross_val_score
14
15 from sklearn.model_selection import train_test_split
16
17 import matplotlib.pyplot as plt
18
19 import warnings
20 warnings.filterwarnings("ignore")
```

```
[2]  1 from google.colab import drive
     2 drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[3]  1 !cp '/content/drive/MyDrive/tmp/archive.zip' ./
```

```
[4]  1 !unzip archive.zip
```

```
Archive:  archive.zip
  inflating: LSWMD.pkl
```

```
[5]  1 !ls -alh
```

✓ 22초 오후 10:17에 완료됨

학습 방법 및 결과

딥러닝 학습 조건

- 노트북 PC 사양, 학습시간

CPU : Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz 2

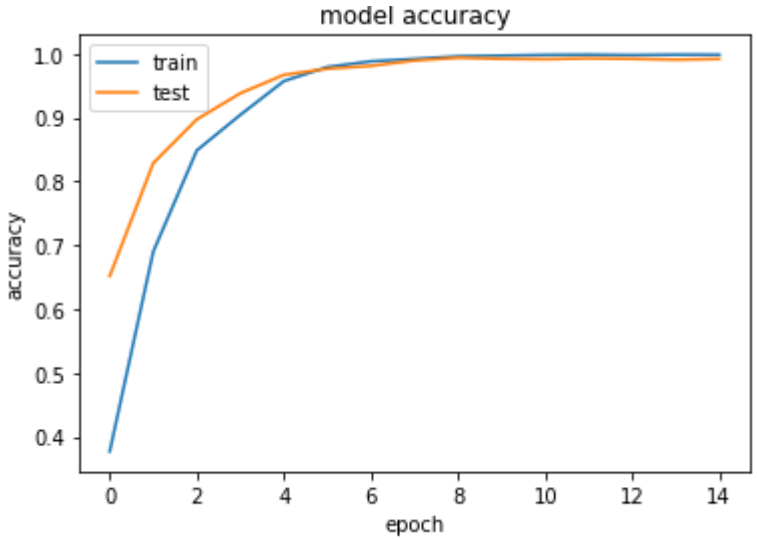
RAM : 8GB

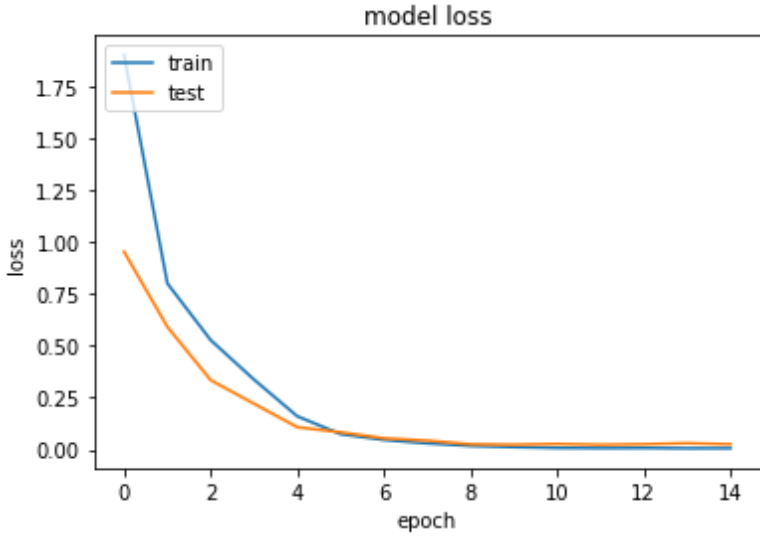
GPU : Intel(R)HD Graphics 520

- 하이퍼파라미터 : epoch=15, batch size=1024,
  optimizer = 'Adam', loss 함수 = 'categorical_crossentropy'

소요시간 : 약 3시간

추이 그래프



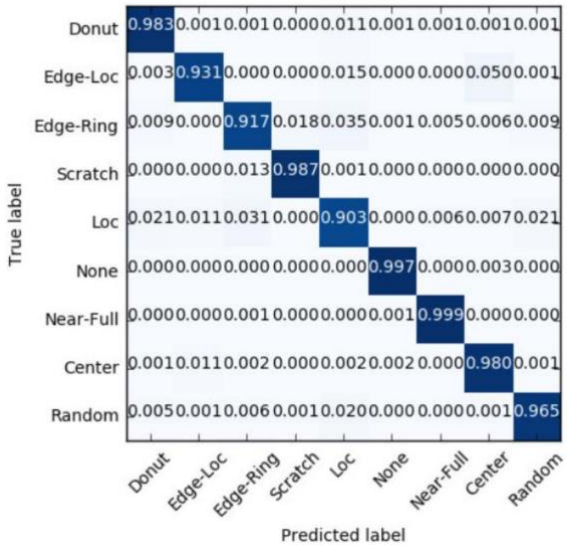


ACC = 99%

결과 및 토의

분류 성능

- Confusion matrix 및 평가지표 (자료를 찾아 봤지만 방법을 몰라 확인 못함)



OVERALL PERFORMANCE COMPARISON OF VARIOUS CLASSIFIERS (%)

Classifier	Training Acc	Validation Acc	Testing Acc	Precision	Recall	F1-Score
CNN-WDI	98.9	**96.4**	**96.2**	**96.2**	**96.2**	**96.2**
CNN-D	97.6	95.5	95.2	95.2	95.2	95.2
CNN-BN	**99.4**	95.6	95.6	95.6	95.6	95.6
CNN-SD	98.6	94.7	94.8	94.8	94.8	94.8
VGG-16	82.3	80.0	80.1	80.3	80.1	79.9
ANN	95.9	95.9	72.0	95.2	95.9	95.4
SVM	91.3	91.0	32.6	87.5	91.0	88.0

Note: Boldface numbers denote the highest values of different performance measures and Acc accuracy

결과 및 토의

1. 느낀점

- 반도체 제조 공정에서 웨이퍼 불량의 조기 진단을 위해 딥러닝 기술을 이용한다는 걸 처음 알았고, 현재 재직중인 회사도 제조업이어서 불량에 대해 딥러닝 기술을 활용할 수 있는지 확인 해 볼 예정임.

2. 아쉬움 점

- 코딩 능력 부족으로 Colab으로 데이터 셋을 불러와서 캐글이나 github 자료를 보고 순차적으로 실행 해보고 공부를 했다는 걸로 의미를 두었음.
- 코딩을 할 수 있었으면 Project#1 주제에 대해 좀 더 흥미롭게 프로젝트를 진행 했을 것 같다는 아쉬운 생각이 들었음.

감사합니다