```
#설치 패키지
#!pip install pandas
#!pip install numpy
#!pip install matplotlib
#!pip install imgaug


import numpy as np   # linear algebra
import pandas as pd  # data processing, CSV file I/O (e.g. pd.read_csv)
import os


import imgaug as ia             # 이미지 증강 패키지
import imgaug.augmenters as iaa # 이미지 증강
import matplotlib.pyplot as plt # Python PL 언어 및 수학적 확장 NumPy 라이브러리를 활용한 플로팅 라이브러리
%matplotlib inline
# 노트북을 실행한 브라우저에서 바로 시각화


from sklearn.model_selection import train_test_split  # 데이터 셋 분할 패키지
from sklearn.preprocessing import OneHotEncoder       # 데이터 전처리 (원핫 인코딩 수행)
```

# 1. 데이터 확인

## 데이터 세트는 웨이퍼 다이 크기, 랏 이름 및 웨이퍼 인덱스와 같은 추가 정보와 함께 811,457개의 웨이퍼 맵으로 구성됨

```
from google.colab import drive
drive.mount('/content/drive')
```

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

```
df = pd.read_pickle('/content/drive/MyDrive/1.capstone/LSWMD.pkl') #pickle 읽기
df.info() # 데이터 정보 표시
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 811457 entries, 0 to 811456
Data columns (total 6 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   waferMap        811457 non-null  object
 1   dieSize         811457 non-null  float64
 2   lotName         811457 non-null  object
 3   waferIndex      811457 non-null  float64
 4   trianTestLabel  811457 non-null  object
 5   failureType     811457 non-null  object
dtypes: float64(2), object(4)
memory usage: 37.1+ MB
```

```
df.head() # 최상단 Top 5개만 표시
```

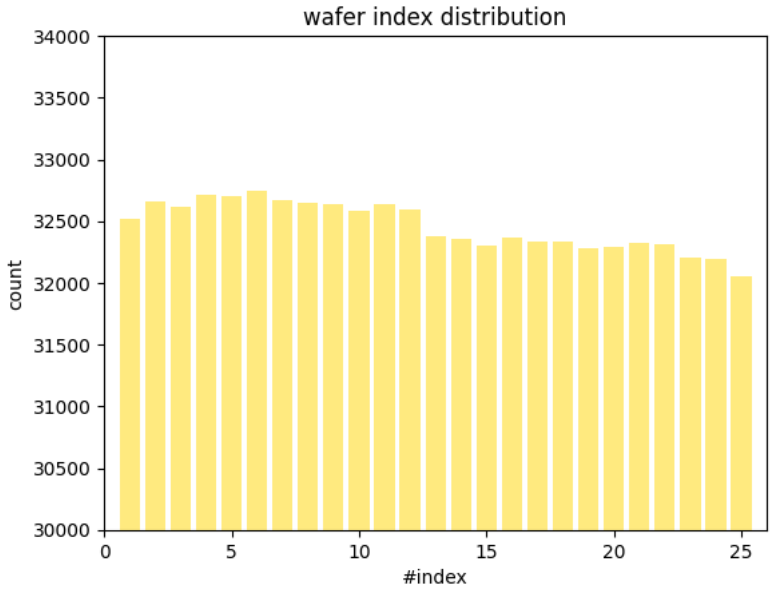
| | waferMap | dieSize | lotName | waferIndex | trianTestLabel | failureType |
|---|---|---|---|---|---|---|
| **0** | [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... | 1683.0 | lot1 | 1.0 | [[Training]] | [[none]] |
| **1** | [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... | 1683.0 | lot1 | 2.0 | [[Training]] | [[none]] |
| **2** | [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... | 1683.0 | lot1 | 3.0 | [[Training]] | [[none]] |
| **3** | [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... | 1683.0 | lot1 | 4.0 | [[Training]] | [[none]] |
| **4** | [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... | 1683.0 | lot1 | 5.0 | [[Training]] | [[none]] |

```
df.tail() # 최하단 Top 5개만 표시
```

| | waferMap | dieSize | lotName | waferIndex | trianTestLabel | failureType |
|---|---|---|---|---|---|---|
| **811452** | [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 1, 1,... | 600.0 | lot47542 | 23.0 | [[Test]] | [[Edge-Ring]] |
| **811453** | [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 1, 1,... | 600.0 | lot47542 | 24.0 | [[Test]] | [[Edge-Loc]] |
| **811454** | [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 1, 1,... | 600.0 | lot47542 | 25.0 | [[Test]] | [[Edge-Ring]] |
| **811455** | [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,... | 600.0 | lot47543 | 1.0 | [] | [] |
| **811456** | [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 1, 1,... | 600.0 | lot47543 | 2.0 | [] | [] |

▾ 모든 Lot은 25웨이퍼 맵을 가지고 있지 않다는 것을 알 수 있고, waferindex가 없어도 결과는 동일한 것을 알 수 있음.

```
# 웨이퍼 인덱스 분포도 확인
unique_index = np.unique(df.waferIndex, return_counts=True) # 중복된 값 제거후 인덱스 반환
plt.bar(unique_index[0], unique_index[1], color='gold', align='center', alpha=0.5)
plt.title('wafer index distribution')
plt.xlabel('#index')
plt.ylabel('count')
plt.xlim(0, 26)
plt.ylim(30000, 34000)
plt.show()
```




```
df = df.drop(['waferIndex'], axis = 1)
```

```
def find_dim(x):
    dim0 = np.size(x, axis=0)
    dim1 = np.size(x, axis=1)
    return (dim0, dim1)

df['waferMapDim'] = df.waferMap.apply(find_dim)
df.sample(5)
```

| | waferMap | dieSize | lotName | trianTestLabel | failureType | waferMapDim |
|---|---|---|---|---|---|---|
| 486739 | [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 2,... | 710.0 | lot30132 | [] | [] | (32, 29) |
| 416201 | [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... | 2367.0 | lot24781 | [] | [] | (51, 59) |
| 361094 | [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,... | 1080.0 | lot21620 | [] | [] | (35, 40) |
| 588856 | [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 1, 2,... | 515.0 | lot36739 | [] | [] | (25, 27) |
| 711068 | [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,... | 516.0 | lot43470 | [[Test]] | [[none]] | (25, 27) |

```
max(df['waferMapDim']), min(df['waferMapDim'])
```

```
((300, 202), (6, 21))
```

```
unique_waferDim = np.unique(df['waferMapDim'])
unique_waferDim.shape
```

```
(632,)
```

```
df['failureNum']=df.failureType
df['trainTestNum']=df.trianTestLabel

mapping_type={
    'Center':0,
    'Donut':1,
    'Edge-Loc':2,
    'Edge-Ring':3,
    'Loc':4,
    'Random':5,
```

```
    'Scratch':6,
    'Near-full':7,
    'none':8}
mapping_traintest={'Training':0,'Test':1}
df=df.replace({'failureNum':mapping_type, 'trainTestNum':mapping_traintest})
```

```
/usr/local/lib/python3.9/dist-packages/pandas/core/array_algos/replace.py:86: FutureWarning: elementwise comparison faile
  op = lambda x: operator.eq(x, b)
```

```
df_label = df[(df['failureNum']>=0) & (df['failureNum']<=8)]
df_label = df_label.reset_index()
df_pattern = df[(df['failureNum']>=0) & (df['failureNum']<=7)]
df_pattern = df_pattern.reset_index()
df_none = df[(df['failureNum']==8)]

# 1) 불량 총 갯수, 2) 불량 패턴, 3) none
df_label.shape[0], df_pattern.shape[0], df_none.shape[0]
```

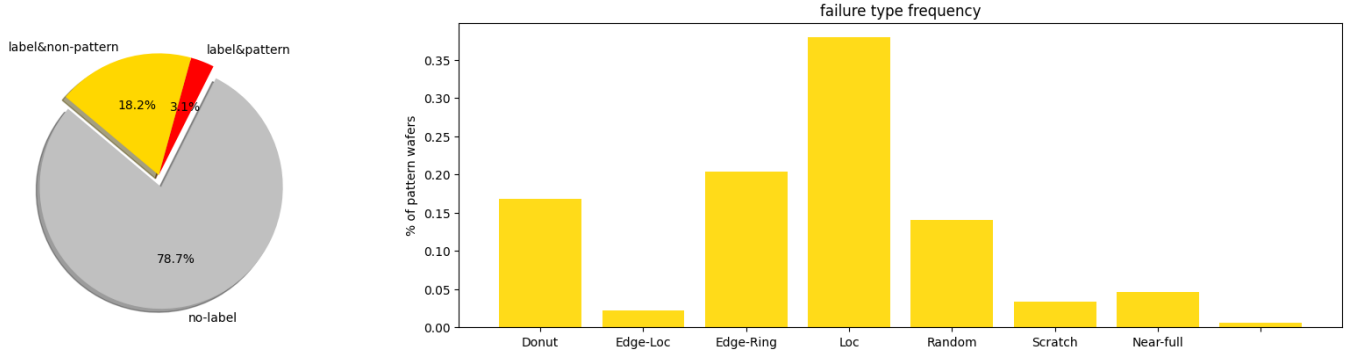
```
(172950, 25519, 147431)
```

```
tol_wafers = df.shape[0]
tol_wafers
```

```
811457
```

```
import matplotlib.pyplot as plt
%matplotlib inline

from matplotlib import gridspec # GridSpec을이용한 복잡한그래프영역배치, GridSpec 함수를 이용하여 subplot의 크기와 위치를 지정

fig = plt.figure(figsize=(20, 4.5))
gs = gridspec.GridSpec(1, 2, width_ratios=[1, 2.5])
#gridspec.GridSpec 사용법 (행, 열, )
#gs = gridspec.GridSpec(nrows = 2,  ncols = 2,  height_ratios = [5, 10], width_ratios = [10, 5])

ax1 = plt.subplot(gs[0])
ax2 = plt.subplot(gs[1])

no_wafers=[tol_wafers-df_label.shape[0], df_pattern.shape[0], df_none.shape[0]]

colors = ['silver', 'red', 'gold']  # 색상 지정
explode = (0.1, 0, 0)               # 부채꼴이 파이 차트의 중심에서 벗어나는 정도 (no-label이 0.1도 벗어나게 설정)
labels = ['no-label', 'label&pattern', 'label&non-pattern'] # 라벨지정
ax1.pie(no_wafers, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=140)

uni_pattern=np.unique(df_pattern.failureNum, return_counts=True)
# set_xticklables 0부터 시작되서 '빈'라벨이 들어가는 경우가 존재
labels2 = ['Center','Donut','Edge-Loc','Edge-Ring','Loc','Random','Scratch','Near-full']

ax2.bar(uni_pattern[0],uni_pattern[1]/df_pattern.shape[0], color='gold', align='center', alpha=0.9)
ax2.set_title("failure type frequency")
ax2.set_ylabel("% of pattern wafers")
ax2.set_xticklabels(labels2)

plt.show()
```

```
<ipython-input-22-f22045e8201b>:28: UserWarning: FixedFormatter should only be used together with FixedLocator
  ax2.set_xticklabels(labels2)
```




```
# 불량패턴 8가지
fig, ax = plt.subplots(nrows = 10, ncols = 10, figsize=(20, 20))
ax = ax.ravel(order='C')
for i in range(100):
    img = df_pattern.waferMap[i]
```

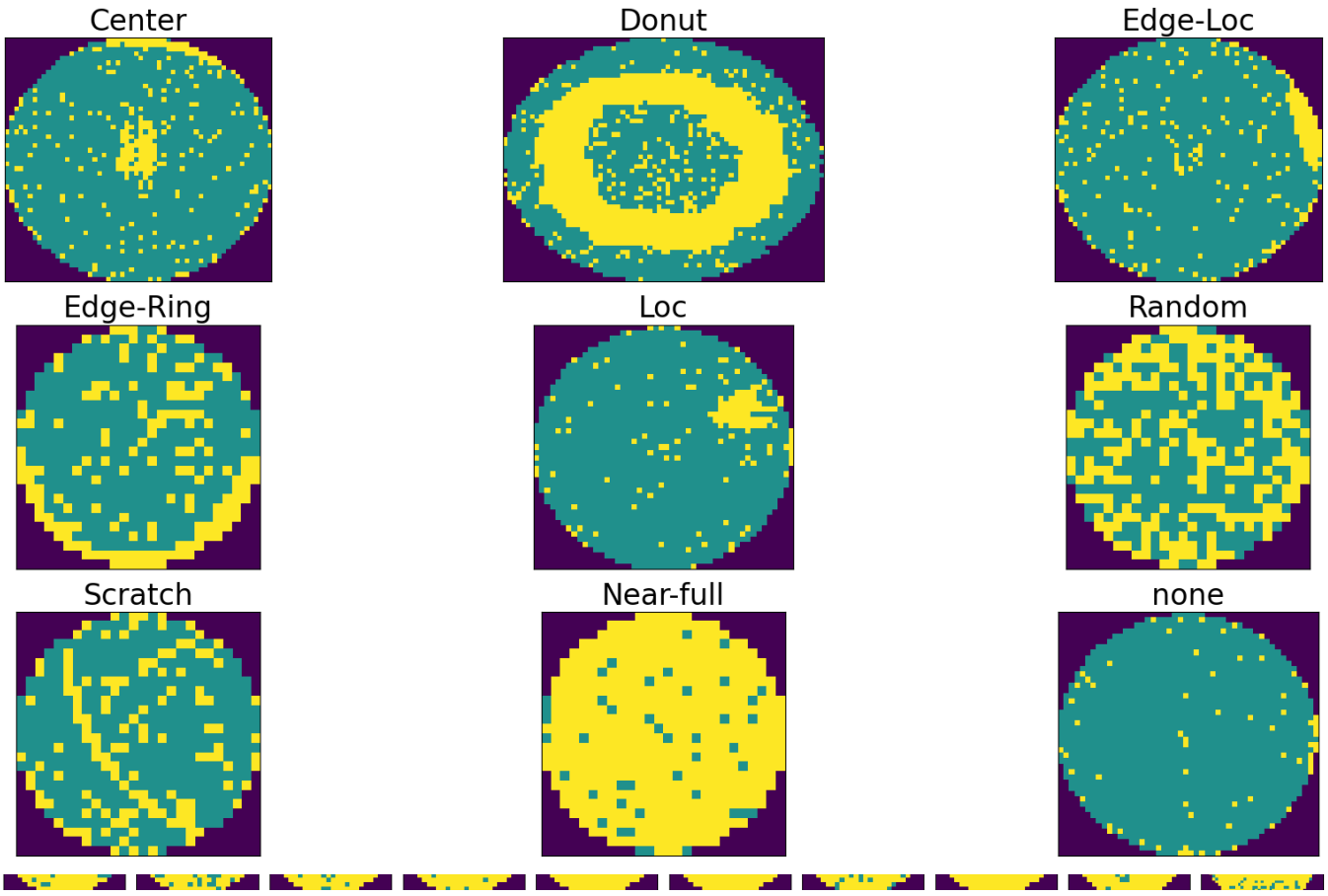
```
        ax[i].imshow(img)
        ax[i].set_title(df_pattern.failureType[i][0][0], fontsize=10)
        ax[i].set_xlabel(df_pattern.index[i], fontsize=8)
        ax[i].set_xticks([])
        ax[i].set_yticks([])
    plt.tight_layout()
    plt.show()

    # 불량패턴 none
    fig, ax = plt.subplots(nrows = 1, ncols = 10, figsize=(20, 20))
    ax = ax.ravel(order='C')
    for i in range(10):
        img = df_none.waferMap[i]
        ax[i].imshow(img)
        ax[i].set_title(df_none.failureType[i][0][0], fontsize=10)
        ax[i].set_xlabel(df_none.index[i], fontsize=8)
        ax[i].set_xticks([])
        ax[i].set_yticks([])
    plt.tight_layout()
    plt.show()
```

```
    # 불량패턴 8가지
    x = [0,1,2,3,4,5,6,7]
    labels2 = ['Center','Donut','Edge-Loc','Edge-Ring','Loc','Random','Scratch','Near-full']

    for k in x:
        fig, ax = plt.subplots(nrows = 1, ncols = 10, figsize=(18, 12))
        ax = ax.ravel(order='C')
```

```
        for j in [k]:
            img = df_pattern.waferMap[df_pattern.failureType==labels2[j]]
            for i in range(10):
                ax[i].imshow(img[img.index[i]])
                ax[i].set_title(df_pattern.failureType[img.index[i]][0][0], fontsize=10)
                ax[i].set_xlabel(df_pattern.index[img.index[i]], fontsize=10)
                ax[i].set_xticks([])
                ax[i].set_yticks([])
        plt.tight_layout()
        plt.show()

    # 불량패턴 none
    fig, ax = plt.subplots(nrows = 1, ncols = 10, figsize=(18, 12))
    ax = ax.ravel(order='C')
    img = df_none.waferMap[df_none.failureType=='none']
    for i in range(10):
          ax[i].imshow(img[img.index[i]])
          ax[i].set_title(df_none.failureType[img.index[i]][0][0], fontsize=10)
          ax[i].set_xlabel(df_none.index[img.index[i]], fontsize=10)
          ax[i].set_xticks([])
          ax[i].set_yticks([])
    plt.tight_layout()
    plt.show()
```

```
    x = [9,340, 3, 16, 0, 25, 84, 37, 7]
    labels2 = ['Center','Donut','Edge-Loc','Edge-Ring','Loc','Random','Scratch','Near-full','none']

    #ind_def = {'Center': 9, 'Donut': 340, 'Edge-Loc': 3, 'Edge-Ring': 16, 'Loc': 0, 'Random': 25,  'Scratch': 84, 'Near-full': 37
    fig, ax = plt.subplots(nrows = 3, ncols = 3, figsize=(20, 10))
    ax = ax.ravel(order='C')
```

```
for i in range(9):
    if i < 8 :
      img = df_pattern.waferMap[x[i]]
      ax[i].imshow(img)
      ax[i].set_title(df_pattern.failureType[x[i]][0][0],fontsize=24)
      ax[i].set_xticks([])
      ax[i].set_yticks([])
    else :
      img = df_none.waferMap[x[i]]
      ax[i].imshow(img)
      ax[i].set_title(df_none.failureType[x[i]][0][0],fontsize=24)
      ax[i].set_xticks([])
      ax[i].set_yticks([])

plt.tight_layout()
plt.show()
```




## 2. 모델정의

```
del df
del df_pattern
del df_none


import chardet
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Input, Conv2D, Dense, MaxPool2D
from tensorflow.keras.layers import Flatten, Softmax, SpatialDropout2D
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam


model = Sequential([
    # Input
    Input(shape=(224, 224, 3)),

    # block 1
    Conv2D(filters=16, kernel_size=(3,3), activation='relu', kernel_initializer='he_normal'),
    BatchNormalization(),
    MaxPool2D(pool_size=(2,2)),
    Conv2D(filters=16, kernel_size=(3,3), activation='relu', padding='same', kernel_initializer='he_normal'),
    BatchNormalization(),
```

```
    # block 2
    Conv2D(filters=32, kernel_size=(3,3), activation='relu', padding='same', kernel_initializer='he_normal'),
    BatchNormalization(),
    MaxPool2D(pool_size=(2,2)),
    Conv2D(filters=32, kernel_size=(3,3), activation='relu', padding='same', kernel_initializer='he_normal'),
    BatchNormalization(),

    # block 3
    Conv2D(filters=64, kernel_size=(3,3), activation='relu', padding='same', kernel_initializer='he_normal'),
    BatchNormalization(),
    MaxPool2D(pool_size=(2,2)),
    Conv2D(filters=64, kernel_size=(3,3), activation='relu', padding='same', kernel_initializer='he_normal'),
    BatchNormalization(),

    # block 4
    Conv2D(filters=128, kernel_size=(2,2), activation='relu', padding='same', kernel_initializer='he_normal'),
    BatchNormalization(),
    MaxPool2D(pool_size=(2,2)),
    Conv2D(filters=128, kernel_size=(2,2), activation='relu', padding='same', kernel_initializer='he_normal'),
    BatchNormalization(), # 배치정규화

    # block 5
    SpatialDropout2D(rate=0.2), # 드롭아웃 0.2
    MaxPool2D(pool_size=(2,2)),

    Flatten(), #1차원 평탄화
    Dense(512, activation='relu'), # 출력값 512를 갖는 은닉층 생성
    Dense(9, activation='softmax') # 출력값 9개를 갖는 은닉층 생성
])
# 아담, 학습률 0.001, 다중 분류 손실 힘수
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 16)      448

 batch_normalization (BatchN  (None, 222, 222, 16)     64
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 111, 111, 16)     0
 )

 conv2d_1 (Conv2D)           (None, 111, 111, 16)      2320

 batch_normalization_1 (Batc  (None, 111, 111, 16)     64
 hNormalization)

 conv2d_2 (Conv2D)           (None, 111, 111, 32)      4640

 batch_normalization_2 (Batc  (None, 111, 111, 32)     128
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 55, 55, 32)       0
 2D)

 conv2d_3 (Conv2D)           (None, 55, 55, 32)        9248

 batch_normalization_3 (Batc  (None, 55, 55, 32)       128
 hNormalization)

 conv2d_4 (Conv2D)           (None, 55, 55, 64)        18496

 batch_normalization_4 (Batc  (None, 55, 55, 64)       256
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 27, 27, 64)       0
 2D)

 conv2d_5 (Conv2D)           (None, 27, 27, 64)        36928

 batch_normalization_5 (Batc  (None, 27, 27, 64)       256
 hNormalization)

 conv2d_6 (Conv2D)           (None, 27, 27, 128)       32896

 batch_normalization_6 (Batc  (None, 27, 27, 128)      512
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 13, 13, 128)      0
 2D)

 conv2d_7 (Conv2D)           (None, 13, 13, 128)       65664
```

```
batch_normalization_7 (Batc  (None, 13, 13, 128)      512
hNormalization)

spatial_dropout2d (SpatialD  (None, 13, 13, 128)      0
ropout2D)
```

## 이미지 증강 함수

```
import cv2

# 주어진 이미지를 높이와 너비에 따라 재구성하는 함수
def reshape_images(images, height, width):
    reshaped_images = np.zeros((len(images), height, width, 3))
    for n in range(len(images)):
        for h in range(height):
            for w in range(width):
                reshaped_images[n, h, w, images[n][h][w]] = 1
    return reshaped_images

# 주어진 이미지를 증강하는 함수 (imgaug 라이브러리를 사용, number=None이면 모든 이미지에 적용)
def augment_images(images, number=None):
    seq = iaa.Sequential([
        iaa.Fliplr(0.5), # 50% 확률로 이미지 좌우로 뒤집음
        iaa.Affine(
            scale={"x": (0.8, 1.2), "y": (0.8, 1.2)}, # 이미지의 크기를 0.8~1.2 사이에서 무작위로 조절
            translate_percent={"x": (-0.05, 0.05), "y": (-0.05, 0.05)}, #이미지를 -5%5% 범위 내에서 무작위로 이동
            rotate=(-180, 180), #이미지를 -180도180도 사이에서 무작위로 회전
            shear=(-8, 8) #수평 픽셀을 -8, 8 사이로 이동
        ),
    ], random_order=True) # 무작위 선택

    #images_input의 각 이미지를 리스트에 추가하고, 마지막으로 리스트를 넘파이 어레이로 변환
    images_input = np.random.choice(images, number) if number else images
    images_expanded = []
    for image in images_input:
        images_expanded.append(image)
    images_expanded = np.array(images_expanded)
    images_augmented = seq(images=images_expanded)
    return images_augmented
```

```
import gc #리소스 정리
gc.collect()
```

```
406
```

## 학습 진행

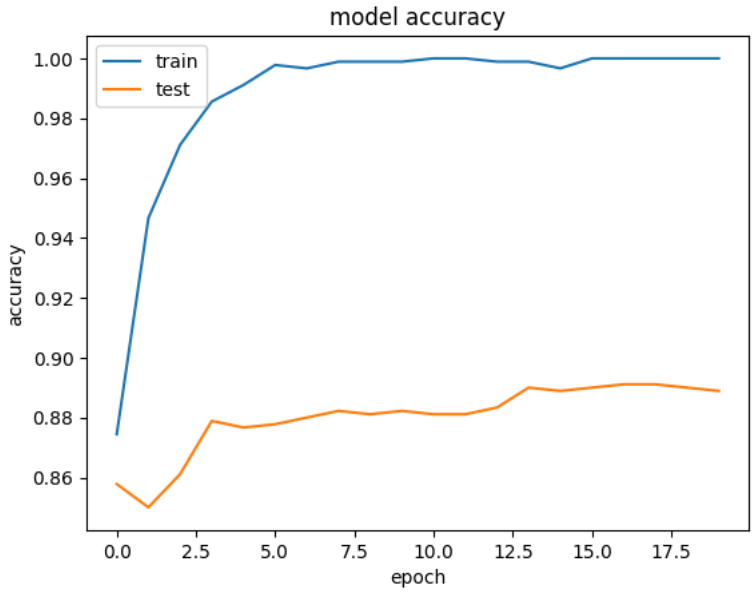
```
class_num = 9              #클래스 개수
dsize = (224, 224)         #이미지 크기
count_per_class_test = 100 #테스트 데이터에서 각 클래스마다 생성할 이미지 개수
count_per_class = 100      #학습 데이터에서 각 클래스마다 생성할 이미지 개수를 나타냅니다.

#테스트 데이터 생성
t_count = 0 # 테스트 증강이미지 체크용
x_test, y_test = [], []
for failureNum in range(class_num):  #count_per_class_test * class_num (1300 * 9 = 80,000   테스트 15%)
    extracted = df_label[df_label['failureNum'] == failureNum].sample(count_per_class_test, replace=True).waferMap
    print(df_label['failureNum'])
    resized = extracted.apply(lambda x:cv2.resize(x, dsize=dsize, interpolation=cv2.INTER_AREA))
    del extracted
    augmented = np.array(augment_images(resized))
    reshaped = reshape_images(augmented, dsize[1], dsize[0])
    del augmented

    labels = np.zeros((count_per_class_test, class_num)) #0으로 채워진 9개행과 20개열로 구성된 Array 생성
    for i in range(count_per_class_test):
        labels[i][failureNum] = 1
        t_count += 1
    x_test.extend(reshaped)
    y_test.extend(labels)
x_test = np.array(x_test)
y_test = np.array(y_test)
print(t_count)

#학습 데이터 생성
count = 0 # 증강이미지 체크용
for j in range(100): #데이터 증강 range * count_per_class * class_num (100 * 100 * 9 = 90,000 학습/검증 85%)
```

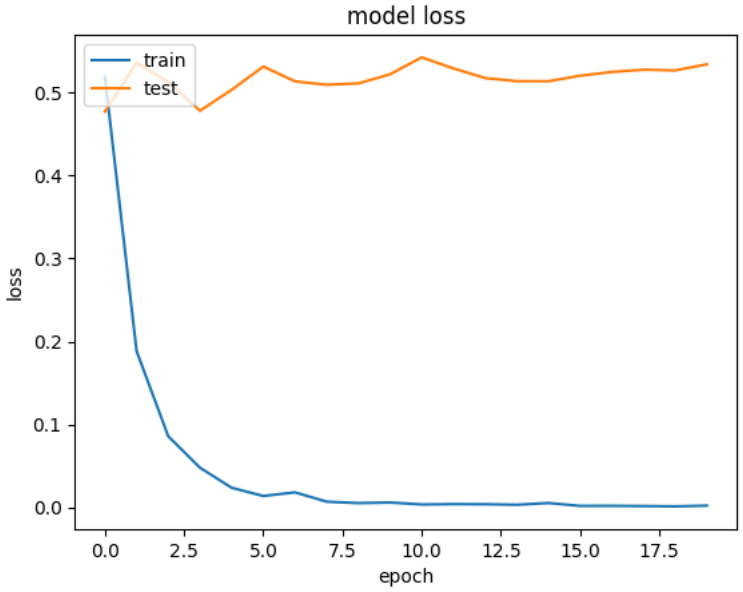
```
    x_train, y_train = [], []
    for failureNum in range(class_num):
        extracted = df_label[df_label['failureNum'] == failureNum].sample(count_per_class, replace=True).waferMap
        #print(df_label['failureType'])
        resized = extracted.apply(lambda x:cv2.resize(x, dsize=dsize, interpolation=cv2.INTER_AREA))
        del extracted
        augmented = np.array(augment_images(resized))
        reshaped = reshape_images(augmented, dsize[1], dsize[0])
        del augmented

        labels = np.zeros((count_per_class, class_num))
        for i in range(count_per_class):
            labels[i][failureNum] = 1
            count += 1
        x_train.extend(reshaped)
        y_train.extend(labels)

    history = model.fit(np.array(x_train), np.array(y_train), validation_data=(x_test, y_test), epochs=20, batch_size=100)
print(count)
```

```
0         8
1         8
2         8
3         8
4         8
         ..
172945    2
172946    2
172947    3
172948    2
172949    3
Name: failureNum, Length: 172950, dtype: object
0         8
1         8
2         8
3         8
4         8
         ..
172945    2
172946    2
172947    3
172948    2
172949    3
Name: failureNum, Length: 172950, dtype: object
0         8
1         8
2         8
3         8
4         8
         ..
172945    2
172946    2
172947    3
172948    2
172949    3
Name: failureNum, Length: 172950, dtype: object
0         8
1         8
2         8
3         8
4         8
         ..
172945    2
172946    2
172947    3
172948    2
172949    3
Name: failureNum, Length: 172950, dtype: object
0         8
1         8
2         8
3         8
4         8
         ..
172945    2
172946    2
```

```
#히스토리 저장
import pickle

with open('hist_v3.h5', mode = 'wb') as f :
    pickle.dump(history, f)
```

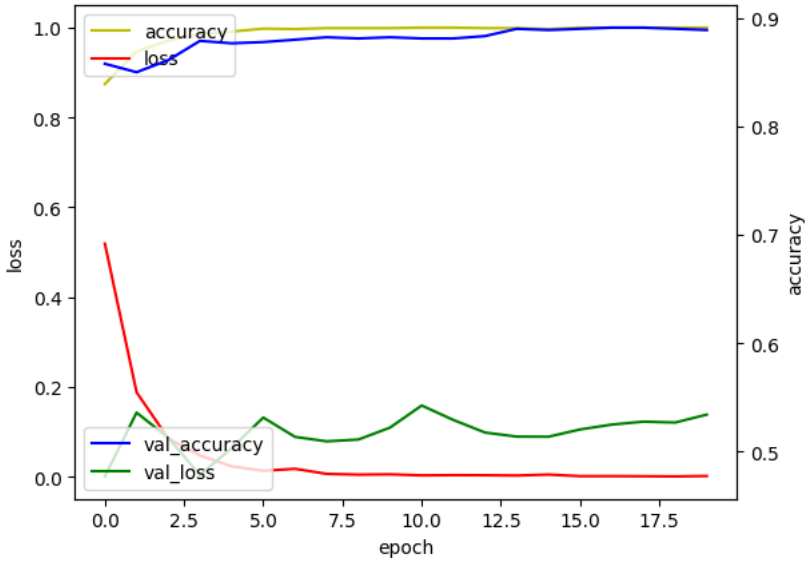
```
#히스토리 불러오기
#with open(file = 'hist_v3.pickle', mode = 'rb') as f :
#    history = pickle.load(f)


#for history in histories:
# accuracy plot
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.axis('auto')
plt.show()

# loss plot
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.axis('auto')
plt.show()
```




```
fig, loss_ax = plt.subplots()

acc_ax = loss_ax.twinx()

loss_ax.plot(history.history['accuracy'],'y',label='accuracy')
loss_ax.plot(history.history['loss'],'r',label='loss')
```

```
acc_ax.plot(history.history['val_accuracy'],'b',label='val_accuracy')
acc_ax.plot(history.history['val_loss'],'g',label='val_loss')

loss_ax.set_xlabel('epoch')
loss_ax.set_ylabel('loss')
acc_ax.set_ylabel('accuracy')

loss_ax.legend(loc='upper left')
acc_ax.legend(loc='lower left')
plt.show()
```




```
print("accracy      : " + str(history.history['accuracy'][19]))
print("val_accuracy : " + str(history.history['val_accuracy'][19]))
print("loss         : " + str(history.history['loss'][19]))
print("val_loss     : " + str(history.history['val_loss'][19]))

# 5. 모델 평가하기
print("")
print("-- Evaluate --")
scores = model.evaluate(x_test, y_test)
print("%s: %.2f%%" %(model.metrics_names[1], scores[1]*100))

print("")
print("-- Predict --")
scores = model.predict(x_test)
```

```
accracy      : 1.0
val_accuracy : 0.8888888955116272
loss         : 0.002047827932983637
val_loss     : 0.5341734886169434

-- Evaluate --
29/29 [==============================] - 1s 13ms/step - loss: 0.5342 - accuracy: 0.8889
accuracy: 88.89%

-- Predict --
29/29 [==============================] - 0s 7ms/step
```

```
#가중치 저장
model.save_weights("cnnWDI_v3_weight")


#가중치 불러오기
#model.load_weights("cnnWDI_v3_weight")

#모델 저장
model.save("cnnWDI_v3.h5")


#모델 불러오기
#keras.models.load_model('cnnWDI_v3.h5')
```