

심층 합성곱 생성적 적대 신경망(DCGAN)기반 웨이퍼 결함 패턴 생성 및 평가 알고리즘 개발

2019.7~2019.9

github: <https://github.com/joyoon1110/generator>

Table of Contents	Page No
적대적 생성 신경망에 대해 알아보고, 이를 MNIST 데이터 셋에 적용하여 이미지 생성 알고리즘 실습을 진행한다.	4
앞서 구현했던 적대적 생성 신경망을 이용하여 wafer dataset에 적용하여 이미지를 생성하는 실험을 진행한다.	6
적대적 생성 신경망을 이용하여 wafer dataset에 적용하여 이미지를 생성하는 실험을 진행	8
적대적 생성 신경망의 구조를 다시 살펴보고, wafer 이미지를 생성하기 위한 네트워크와 하이퍼파라미터를 찾고, 데이터를 생성	9
기존 적대적 생성 신경망에 batch normalization기법을 추가하여 데이터 생성 실험을 진행	12
데이터 셋으로 웨이퍼 데이터 셋(WM-811K)을 사용하여 이미지 생성 실험을 수행한다.	14
MNIST 데이터 셋을 사용하여 이미지를 생성하고, 학습에 따른 오차 변화를 관찰한다.	15
MNIST 데이터 셋과 웨이퍼 데이터 셋(WM-811K)의 이미지 유효성 평가 결과를 비교한다.	17
웨이퍼 데이터 셋의 생성 데이터에 대한 유효성 평가 척도 값을 낮추기 위한 실험을 진행한다.	19
이진화 임계값을 기준으로 FID 측정값을 낮추기 위한 실험을 진행한다.	21
이미지 생성 알고리즘을 실제 서비스에 적용하기 위해 CPU 및 GPU 환경에서 시간을 측정한다.	23
원본 데이터 셋으로 학습한 분류 신경망을 이용하여 생성 데이터를 분류하는 실험을 수행한다.	24
실제 서비스 배포를 위한 응용사례를 조사하고 추후 적용을 위한 디자인을 고안한다.	26
데이터 생성에 필요한 클래스별 최소 이미지 수 파악을 추가로 수행한다.	28
이전 연구에서 정한 트레이닝 이미지 수를 이용하여 데이터 생성 실험을 수행한다.	29
이전 연구에서 생성된 이미지 유효성으로 평가한 FID 측정값을 낮추는 실험과 알고리즘의 동작 시간을 줄이기 위한 트레이닝 수를 줄이는 실험을 진행한다.	31
WM-811K 데이터 셋에서 기존 이미지와 다른 이미지를 대상으로 생성 실험을 진행하고, FID 값을 측정함으로써 생성된 이미지가 유효함을 확인한다.	33
사용자 정의 클래스 이미지를 정의하고 분류기를 훈련하여, 생성한 이미지 분류를 위한 모델을 생성한다.	34
Scratch 패턴 이미지 생성 실험을 하고 클래스 분류를 위한 모델을 얻기 위한 학습을 진행한다.	36
Scratch 패턴의 이미지 데이터 증식 방법을 변경하여 분류 실험을 진행한다.	37
ImageDataGenerator 클래스에서 제공하는 이미지 변환 인자를 이용	38



# 연구 노트

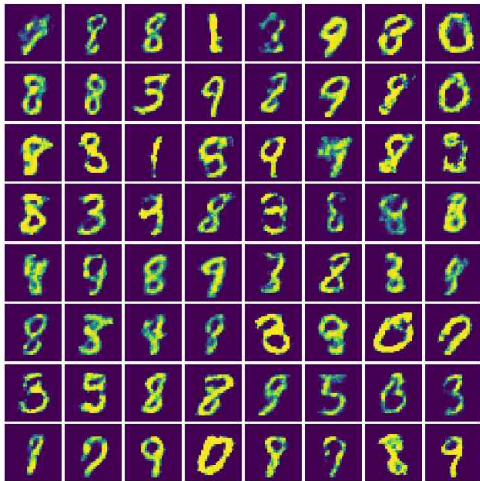
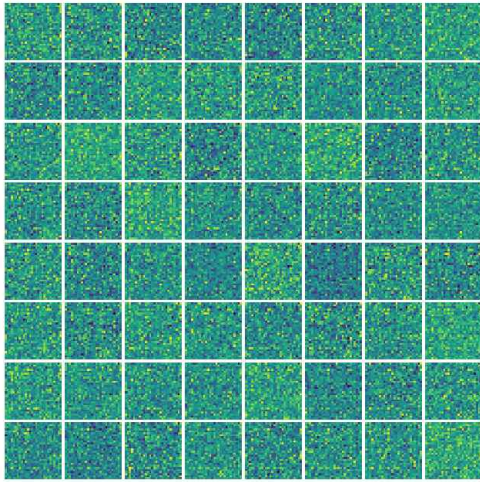
## 연구 목표

적대적 생성 신경망에 대해 알아보고, 이를 MNIST 데이터 셋에 적용하여 이미지 생성 알고리즘 실습을 진행한다.

## 개념

### 1. 적대적 생성 신경망(GAN)

적대적 생성 신경망(GAN)은 생성된 이미지가 실제 이미지와 통계적으로 거의 구분이 되지 않도록 강제하여 아주 실제 같은 합성 이미지를 생성한다.



GAN의 네트워크 2개는 아래와 같다

- 생성자 네트워크(Generative network): 랜덤 벡터(잠재 공간의 무작위한 포인트)를 입력으로 받아 이를 합성된 이미지로 디코딩한다.
- 판별자 네트워크(Discriminator network)(또는 상대 네트워크): 이미지(실제 또는 합성 이미지)를 입력으로 받아 훈련 세트에서 온 이미지인지, 생성자 네트워크가 만든 이미지인지 판별한다.

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]. \quad (1)$$

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m [\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))]$$

생성자 네트워크는 판별자 네트워크를 속이도록 훈련한다. 훈련이 계속될수록 점점 더 실제와 같은 이미지를 생성하게 된다. 실제 이미지와 구분할 수 없는 인공적인 이미지를 만들어 판별자 네트워크가 두 이미지를 동일하게 보도록 만든다. 한편 판별자 네트워크는 생성된 이미지가 실제인지 판별하는 기준을 설정하면서 생성자의 능력 향상에 적응해 간다. 훈련이 끝나면 생성자는 입력 공간에 있는 어떤 포인트를 그럴듯한 이미지로 변환한다.

위의 그림은 차례대로 훈련 초기(epoch 0)에서 시작하여 점진적으로 이미지를 생성하는 과정을 나타낸다. 이때 epochs는 100,000번, batch size는 64로 진행하였다.

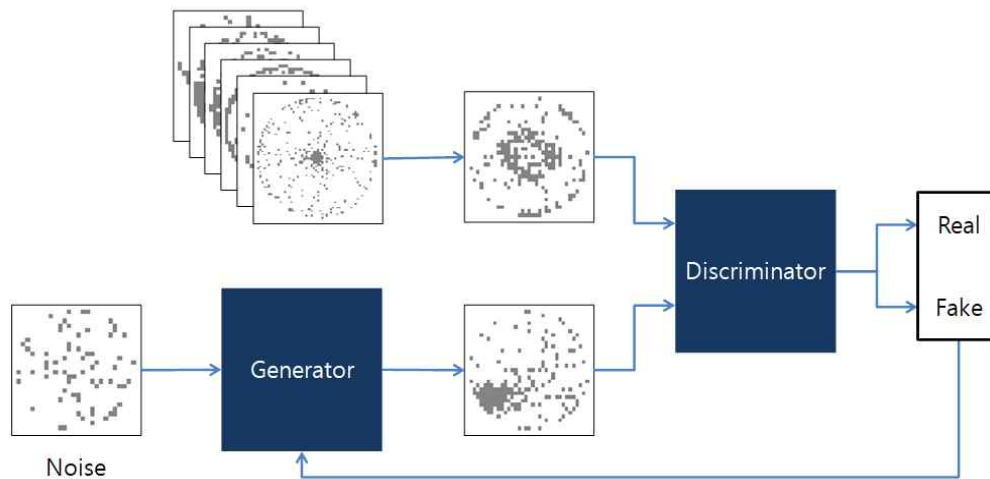
## 연구 노트

### 연구 목표

앞서 구현했던 적대적 생성 신경망을 이용하여 wafer dataset에 적용하여 이미지를 생성하는 실험을 진행한다.

### 개념

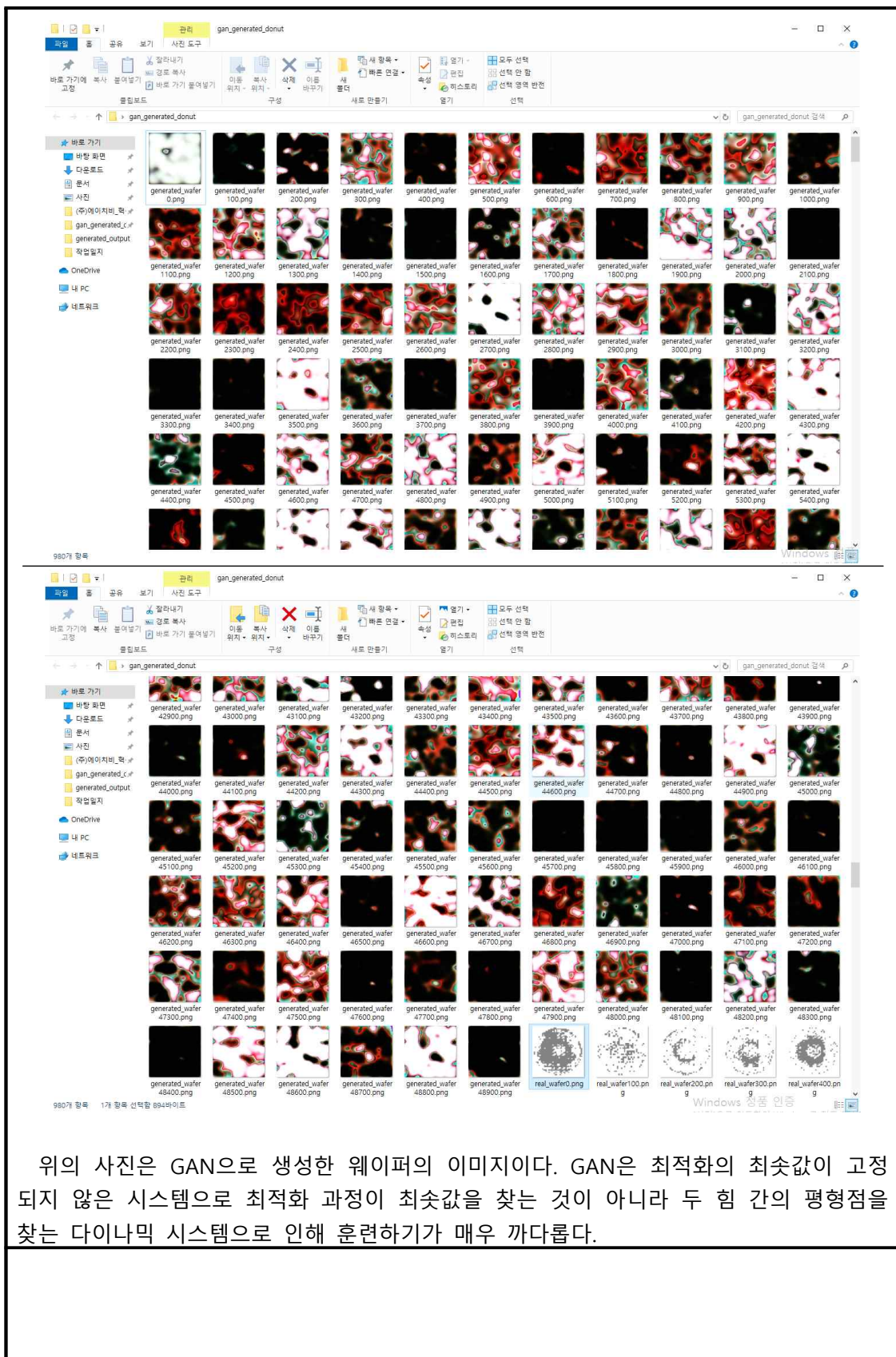
#### 1. 적대적 생성 신경망을 이용한 wafer dataset 이미지 생성



위의 그림은 wafer dataset을 생성하기 위한 적대적 생성 신경망의 구조로 랜덤한 잠재 공간의 벡터를 이미지로 변환하고 판별자는 실제 이미지와 생성된 이미지를 구분. 생성자는 판별자를 속이도록 훈련한다.



위의 웨이퍼 이미지는 웨이퍼가 가진 결함중 도넛의 형태를 나타내는 이미지의 예이다.



위의 사진은 GAN으로 생성한 웨이퍼의 이미지이다. GAN은 최적화의 최솟값이 고정되지 않은 시스템으로 최적화 과정이 최솟값을 찾는 것이 아니라 두 힘 간의 평형점을 찾는 다이내믹 시스템으로 인해 훈련하기가 매우 까다롭다.



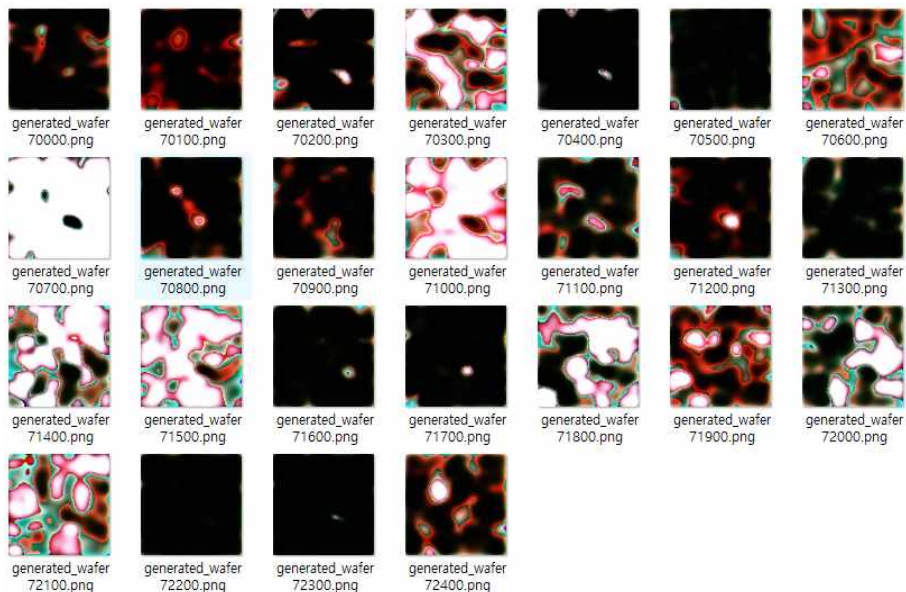
# 연구 노트

## 연구 목표

적대적 생성 신경망을 이용하여 wafer dataset에 적용하여 이미지를 생성하는 실험을 진행하고 GAN의 훈련의 어려움을 이해한다.

## 개념

적대적 생성 신경망을 이용한 실험으로 기초 데이터(MNIST)에 사용한 네트워크의 구조를 동일하게 사용하여 이미지 생성을 진행하였다. 아래 사진에서와 같이 실험에 사용한 데이터와는 너무 다른 모양으로 데이터가 생성되는 것을 알 있고, GAN의 단점을 아래와 같이 정리하였음.



## [단점]

1. GAN은 고정된 손실 공간에서 수행하는 단순한 경사하강법 과정이 아니라 동적과정으로 인하여 훈련하기 어렵다.
2. 잠재 공간이 연속된 구조를 가지지 않아 잠재 공간의 개념 벡터를 사용하여 이미지를 변경하는 등의 실용적인 특정 애플리케이션에는 잘 맞지 않는다.



# 연구 노트

## 연구 목표

적대적 생성 신경망의 구조를 다시 살펴보고, wafer 이미지를 생성하기 위한 네트워크와 하이퍼파라미터를 찾고, 데이터를 생성한다.

## 개념

### 1. GAN의 네트워크 훈련 방법

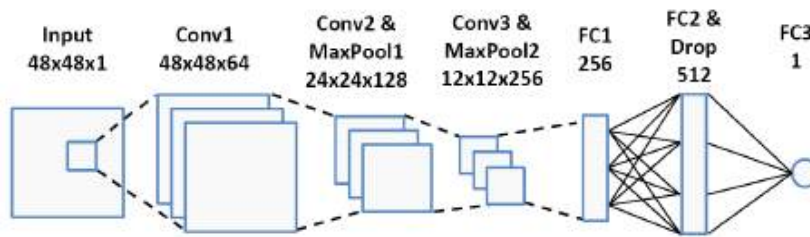


Figure 5. CNN architecture for the discriminator

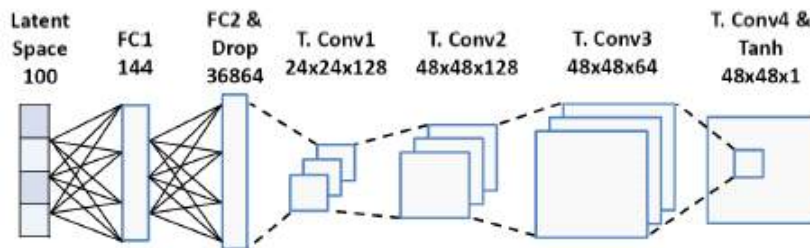
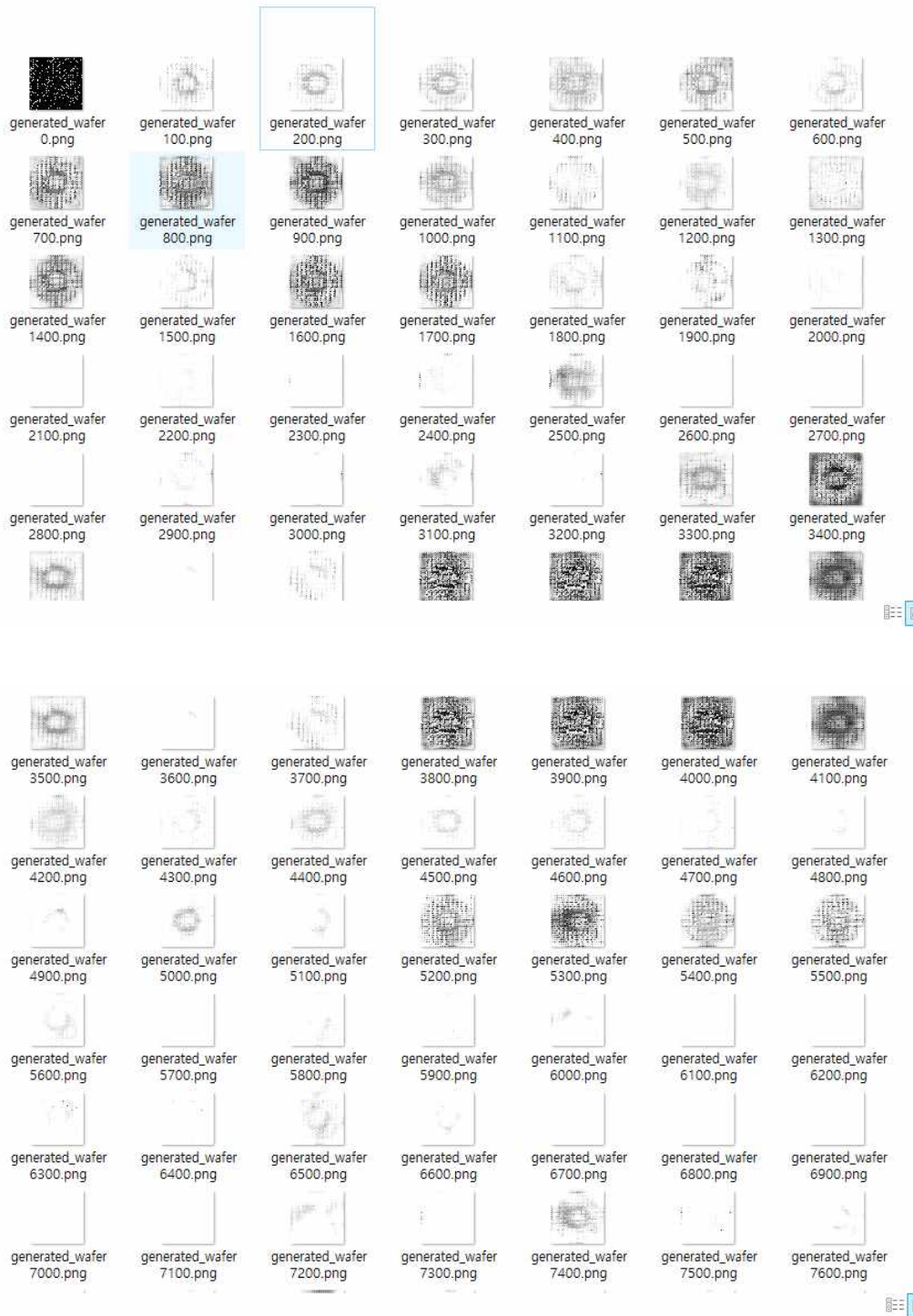
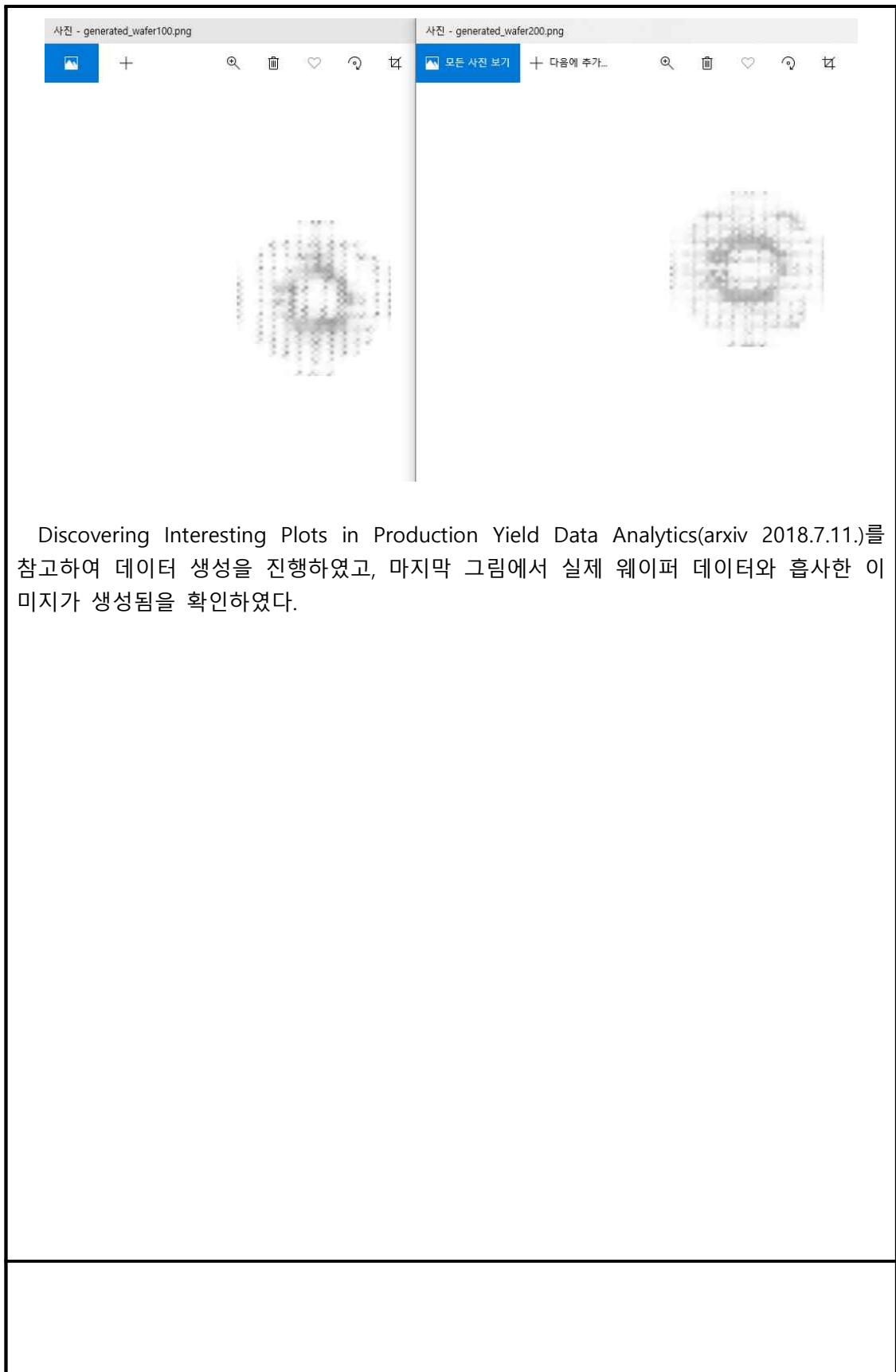


Figure 6. CNN architecture for the generator

- 잠재 공간에서 무작위로 포인트를 뽑는다(랜덤 노이즈).
- 이 랜덤 노이즈를 사용하여 generator에서 이미지를 생성한다.
- 생성된 이미지와 진짜 이미지를 섞는다.
- 진짜와 가짜가 섞은 이미지와 이에 대응하는 타깃을 사용하여 discriminaotr를 훈련한다.
- 잠재 공간에서 무작위로 새로운 포인트를 뽑는다.
- 이 랜덤 벡터를 사용하여 gan을 훈련한다. 모든 타깃은 'real'로 설정한다. 판별자가 생성된 이미지를 모두 'rea'로 예측하도록 생성자의 가중치를 업데이트 한다.

## 2. 생성된 wafer 이미지





Discovering Interesting Plots in Production Yield Data Analytics(arxiv 2018.7.11.)를 참고하여 데이터 생성을 진행하였고, 마지막 그림에서 실제 웨이퍼 데이터와 흡사한 이미지가 생성됨을 확인하였다.

# 연구 노트

## 연구 목표

기존 적대적 생성 신경망에 batch normalization 기법을 추가하여 데이터 생성 실험을 진행한다.

## 개념

### 1. batch normalization

Normalization을 사용하는 것이 거의 루틴과 같이 정형화되어있다. 다만 모든 layer에 BN을 추가하면 문제가 생기고 Generator의 output layer와 discriminator의 input layer에는 BN을 넣지 않는다. 이유를 추측해보자면 아무래도 Generator가 생성하는 이미지가 BN을 지나면서 다시 normalize 되면 아무래도 실제 이미지와는 값의 범위가 다를 수 밖에 없다. 저자들은 BN을 넣으면 GAN의 고질적 문제 중 하나인 Mode collapsing 문제를 어느 정도 완화해준다고 하는데 이 부분은 사실 아직 해결되었다고 보기는 어렵다. NIPS 2016 Tutorial에서도 언급되었지만 BN이 거꾸로 특정 문제를 일으키기도 하는 경우가 있다.

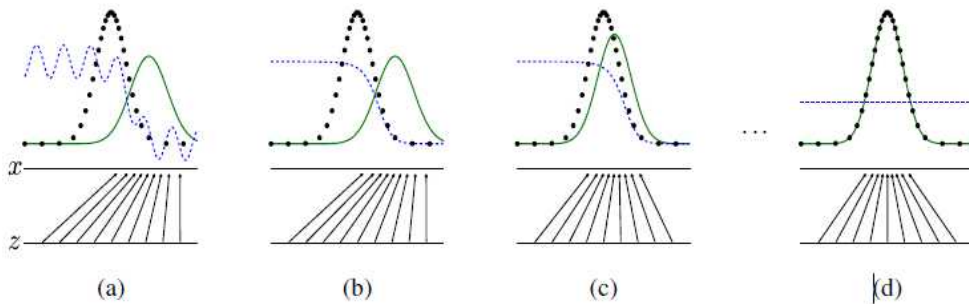
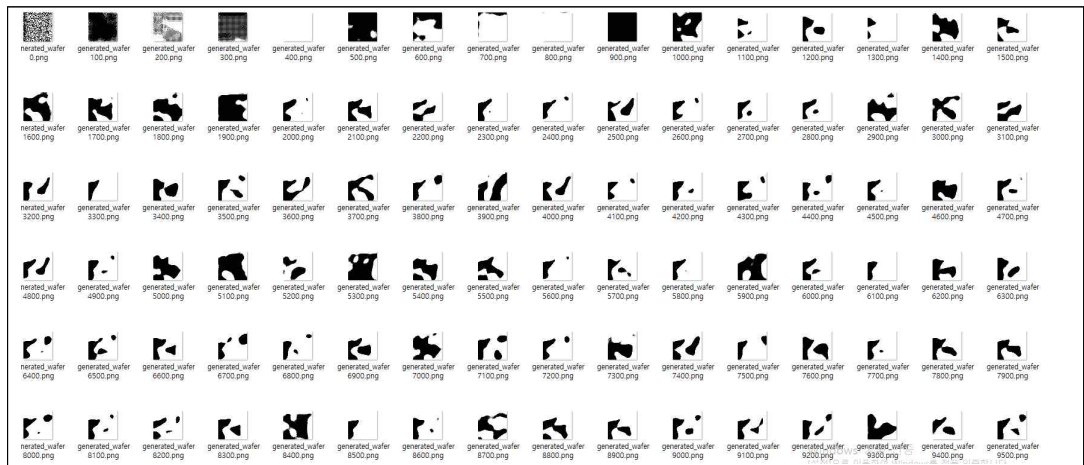
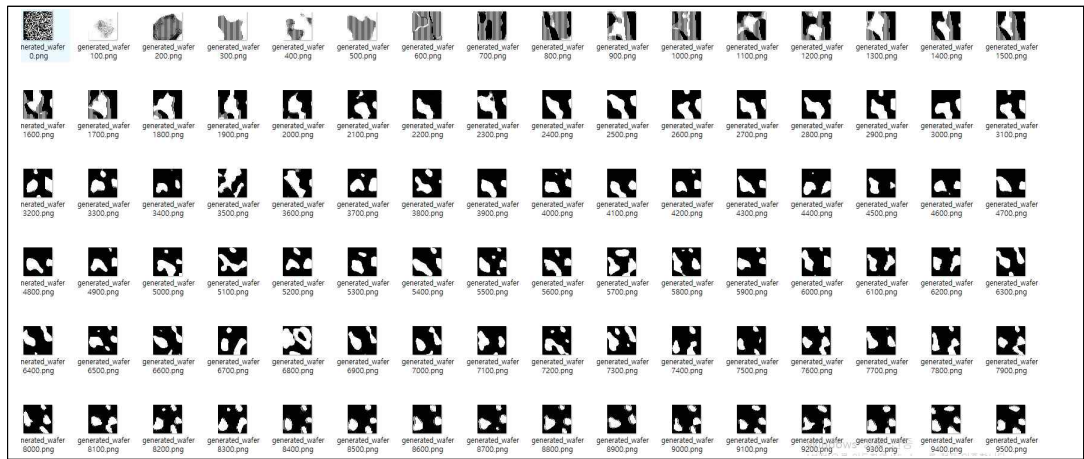
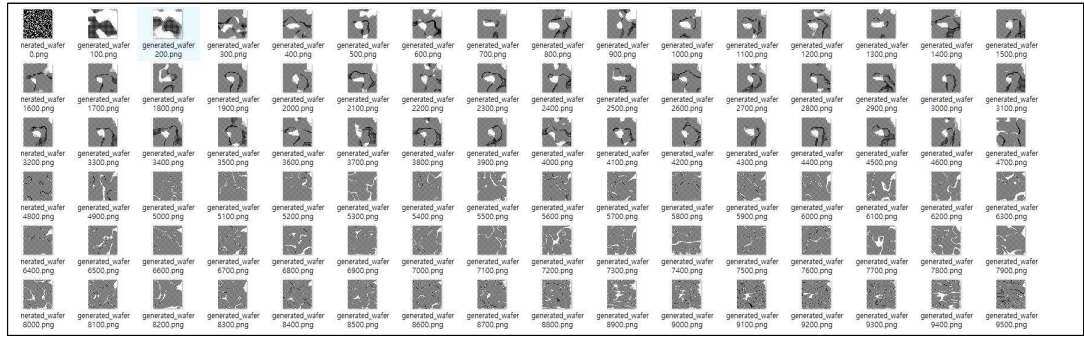


Figure 1: Generative adversarial nets are trained by simultaneously updating the discriminative distribution ( $D$ , blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line)  $p_x$  from those of the generative distribution  $p_g$  ( $G$ ) (green, solid line). The lower horizontal line is the domain from which  $z$  is sampled, in this case uniformly. The horizontal line above is part of the domain of  $x$ . The upward arrows show how the mapping  $x = G(z)$  imposes the non-uniform distribution  $p_g$  on transformed samples.  $G$  contracts in regions of high density and expands in regions of low density of  $p_g$ . (a) Consider an adversarial pair near convergence:  $p_g$  is similar to  $p_{\text{data}}$  and  $D$  is a partially accurate classifier. (b) In the inner loop of the algorithm  $D$  is trained to discriminate samples from data, converging to  $D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}$ . (c) After an update to  $G$ , gradient of  $D$  has guided  $G(z)$  to flow to regions that are more likely to be classified as data. (d) After several steps of training, if  $G$  and  $D$  have enough capacity, they will reach a point at which both cannot improve because  $p_g = p_{\text{data}}$ . The discriminator is unable to differentiate between the two distributions, i.e.  $D(x) = \frac{1}{2}$ .



위의 생성 이미지는 적대적 생성 네트워크에 batch normalization을 추가하여 생성한 wafer 이미지 데이터로 기존에 생성하였던 이미지보다 결과가 좋지 않음을 확인하였다.

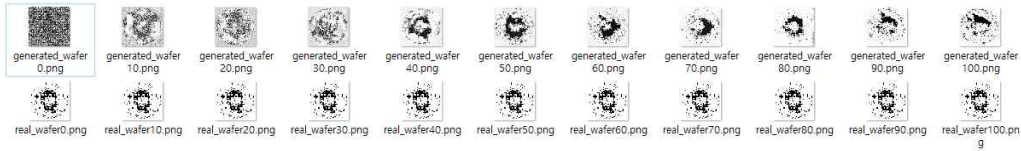


# 연구 노트

## 연구 목표

데이터 셋으로 웨이퍼 데이터 셋(WM-811K)을 사용하여 이미지 생성 실험을 수행한다.

## 개념



(a) 생성 이미지와 원본 이미지의 비교

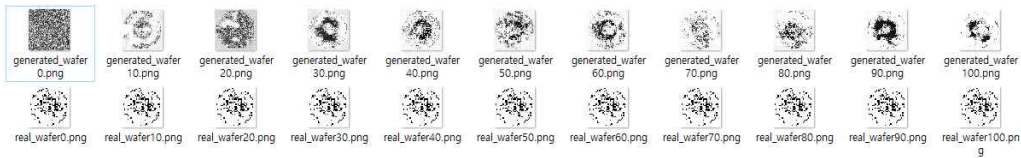


(b) 초기 생성 이미지



(c) 생성 이미지

그림 1. 실험 1에서 생성 이미지의 변화 모습



(a) 학습 횟수에 따른 생성 이미지의 변화2



(b) 생성 이미지



(c) 원본 이미지

그림 2. 실험 2에서 생성 이미지와 원본 이미지의 비교

이미지의 생성은 도메인에 따라 네트워크의 구조 이외에도 학습량에 영향을 많이 받음을 확인하고, 아래 그림에 생성 이미지의 변화와 원본 이미지와 차이를 나타냄.

# 연구 노트

## 연구 목표

MNIST 데이터 셋을 사용하여 이미지를 생성하고, 학습에 따른 오차변화를 관찰한다.

## 개념

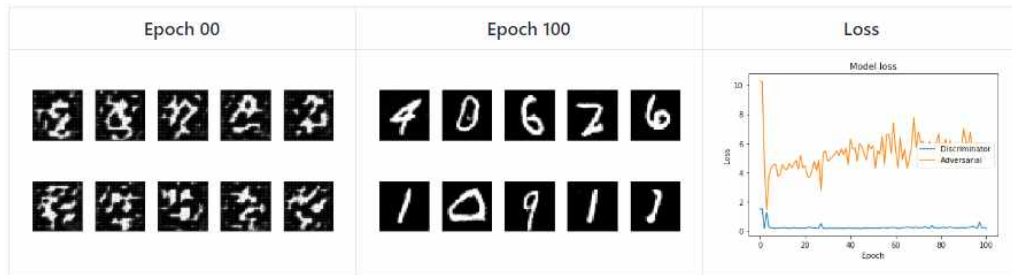


그림 1. MNIST 데이터 셋의 생성 신경망 학습에 따른 생성 이미지

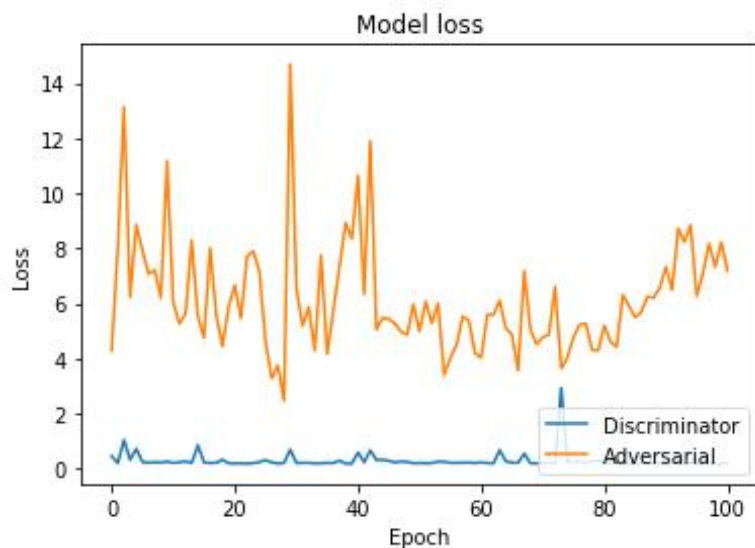


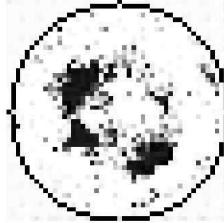
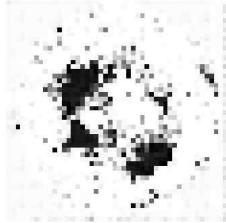
그림 2. MNIST 데이터 셋의 생성 신경망 학습 오차 변화

이전 연구에서 학습량에 따른 생성 이미지의 변화를 확인하였고, 이를 MNIST 데이터셋에 적용하여 학습량에 따른 오차변화를 관찰하였음. 그림1에서 Epoch가 00일때와 100일때를 비교하면 학습에 따라 생성 이미지의 변화가 관찰되었고, 이때 Loss는 MNIST 데이터 셋의 신경망의 오차를 나타냄. Discriminator 네트워크의 Loss는 크게 감소하는 반면, Adversarial 네트워크의 오차는 변동이 심하게 나타남. 이론적으로 두 오차가 같은 Loss 값을 가져야 하지만 실제 실험에서는 관찰되지 않았음.



$$FID = \|m - m_w\|_2^2 + Tr(C + C_w - 2(CC_w)^{1/2}),$$

수식 1. FID(Frechet Inception Distance)



(a) 생성된 원본 이미지      (b) 지름이 r인 직경 생성      (c) 이진화 이미지

그림 3. 임계값(threshold value)에 따른 이미지 변화 과정

생성 이미지의 유효성을 평가하기 위해 수식 1을 사용하여 FID값을 측정함. 수식은 생성 이미지와 원본 이미지 간의 정규 분포의 차이를 측정하는 원리를 이용함.

생성 이미지 유효성 평가에 앞서 생성 이미지에서 데이터 셋의 범위를 벗어난 픽셀을 확인하였고, 이를 제거하기 위해 오프 이진화(Off-thresholding)를 사용함. 흑백 이미지의 픽셀값의 범위(0~255) 내에서 변화를 시켜가며 확인하였고, 임계값(threshold value)이 210일 때 데이터 셋의 범위를 벗어난 픽셀들이 사라지는 것을 확인하였음.

## 연구 노트

### 연구 목표

MNIST 데이터 셋과 웨이퍼 데이터 셋(WM-811K)의 이미지 유효성 평가 결과를 비교한다.

### 개념

$A_1$	$A_2$	FID ( $d^2$ )
MNIST train	MNIST train	6.8959054997e-11
MNIST train	MNIST val	7.05136659744
MNIST val	MNIST train	7.0513665973
MNIST train	ImageNet 10 classes	338.586062646
MNIST train	ImageNet 6 classes	346.218188602
ImageNet 10 classes	ImageNet 6 classes	67.1025959331

(a)

FID between MNIST train and val : 7.051364031219645

FID between MNIST val and train : 7.051364031192733

FID between MNIST train and train : -4.4830549676303866e-11

(b)

그림 1. MNIST 데이터 셋의 FID 측정 값

웨이퍼 데이터 셋의 생성 이미지 유효성을 평가하기 위하여 FID 값을 측정하여 기준이 되는 MNIST 데이터 셋과 비교하였음. MNIST 데이터 셋의 FID 값의 경우, 원본 이미지와 생성 이미지 간의 정규 분포의 차이가 0과 가까운 값을 가지는 것을 확인하였고, 이는 즉 생성된 이미지가 유효하다는 것을 의미함. 또한, MNIST 데이터 셋과 ImageNet 데이터 비교를 통해서 두 데이터 셋의 관계가 없을 경우 큰 FID 값을 가지는 것을 확인할 수 있었음.

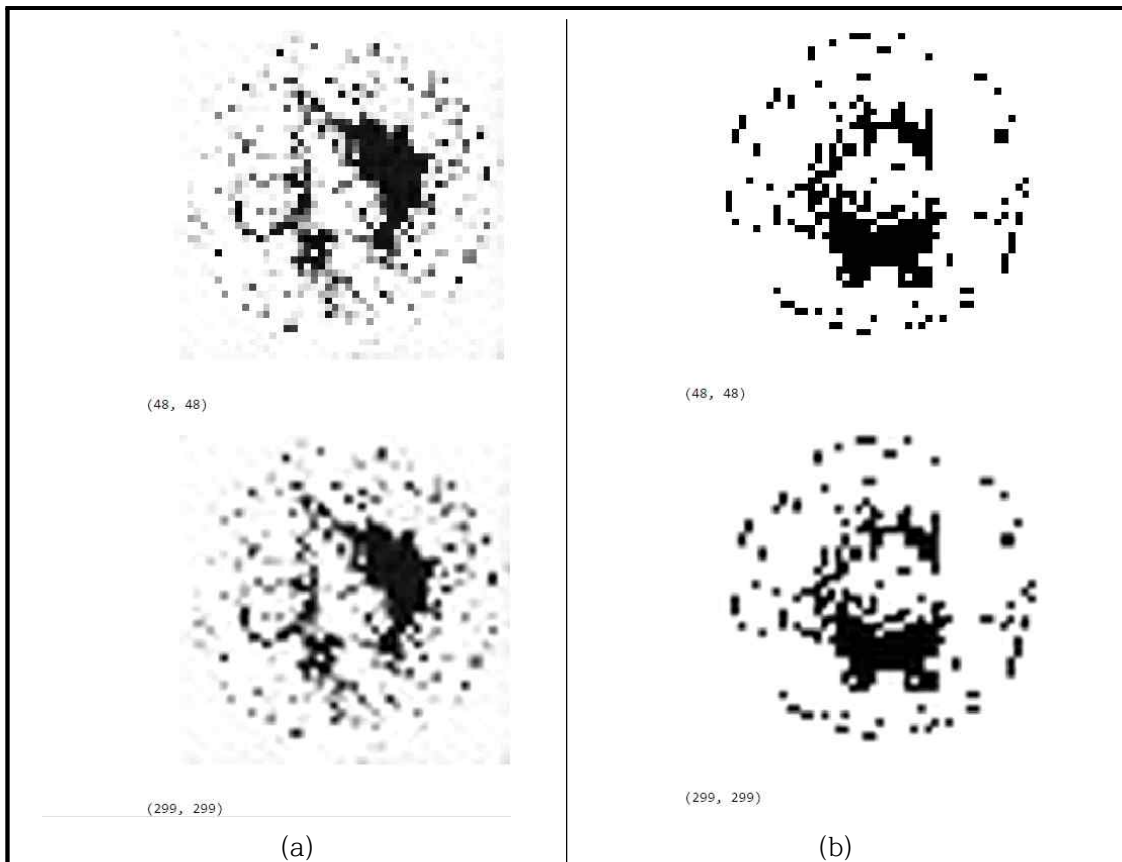


그림 2. 웨이퍼 데이터 셋 생성 이미지

```

/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:104: ComplexWarning: Casting complex values to real discards the imaginary part
FID between WAFER train and val : value=334.1
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:105: ComplexWarning: Casting complex values to real discards the imaginary part
FID between WAFER val and train : value=334.1
FID between WAFER train and train : value=-0.0
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:106: ComplexWarning: Casting complex values to real discards the imaginary part

```

그림 3. 웨이퍼 데이터 셋 생성 이미지 FID 측정 값

그림 2의 (a)는 이진화 전 생성 이미지의 사이즈에 따른 이미지를 보여줌. 웨이퍼 데이터 셋을 생성하기 위해 48x48로 리사이즈된 이미지를 사용하고, FID 값을 측정하기 위하여 299x299로 다시 리사이즈 함. 그림 2의 (b)는 이진화 후 (a)와 마찬가지로 생성 이미지의 사이즈에 따른 이미지를 보여줌. (a)와 동일하게 리사이즈하는 과정을 거침.

그림 3에서 웨이퍼 데이터 셋의 생성 이미지는 MNIST 데이터 셋 생성 이미지와 비교하여 FID 값이 다른 클래스와 분포를 비교하듯 큰 값을 가지는 것을 확인하였음.

# 연구 노트

## 연구 목표

웨이퍼 데이터 셋의 생성 데이터에 대한 유효성 평가 척도 값을 낮추기 위한 실험을 진행한다.

## 개념

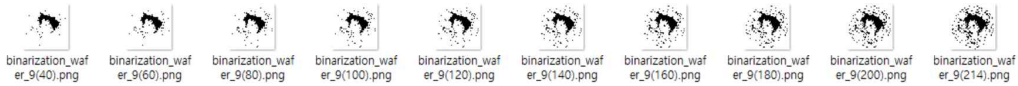


그림 1. 이진화 임계값 변화에 따른 생성 이미지의 변화

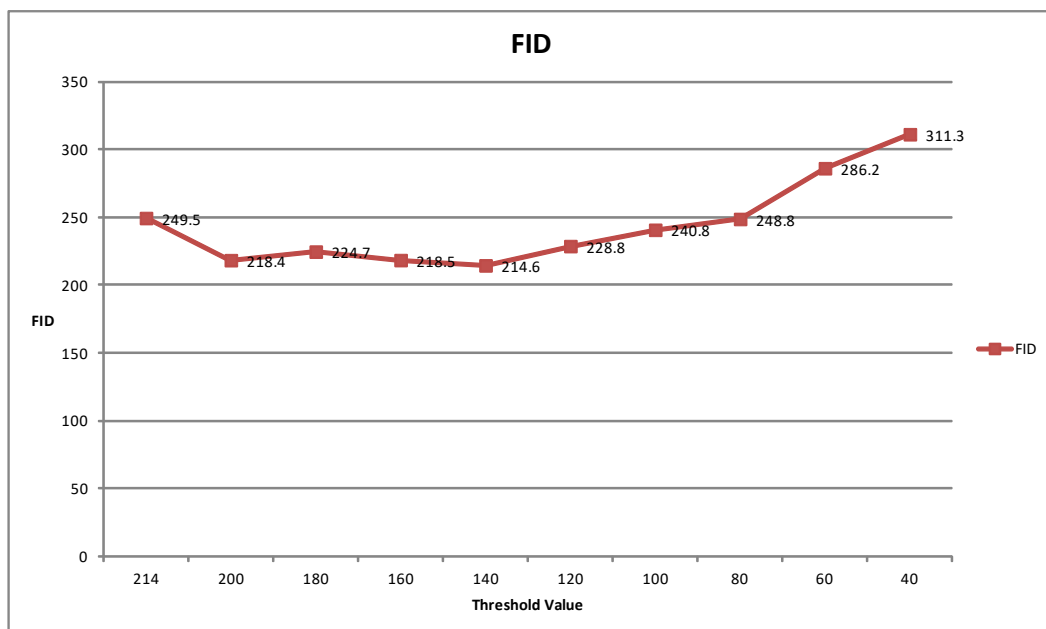


그림 2. 이진화 임계값 변화에 따른 생성 이미지 FID 측정값의 변화

```
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:103: ComplexWarning: Casting complex values to real discards the imaginary part
FID between WAFER train and val : value=214.6
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:104: ComplexWarning: Casting complex values to real discards the imaginary part
FID between WAFER val and train : value=214.6
FID between WAFER train and train : value=-0.0
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:105: ComplexWarning: Casting complex values to real discards the imaginary part
```

그림 3. 임계값이 140일 때 생성 이미지의 FID 측정값

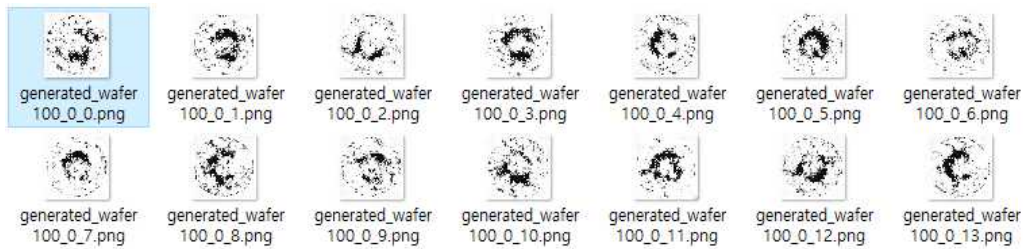
그림 1은 이진화 임계값 변화에 따른 생성 이미지의 변화를 나타낸다. 임계값이 작아질수록 이미지의 노이즈가 많이 사라지는 것을 확인할 수 있다. 그에 반하여 FID 측정값은 임계값의 범위가 140~200 인 구간에서는 감소하는 추세를 보였으나 40~120으로 임계값이 더욱 작아질수록 증가하는 추세를 확인하였다. 이는 이미지의 노이즈 제거에 따른 원본 이미지와의 정규 분포의 차이로 인한 것으로 해석된다.

# 연구 노트

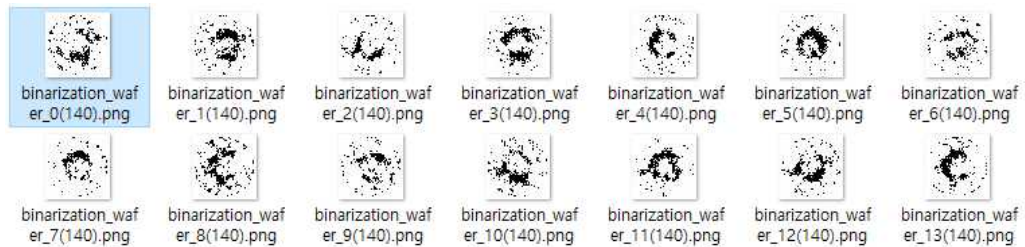
## 연구 목표

이진화 임계값을 기준으로 FID 측정값을 낮추기 위한 실험을 진행한다.

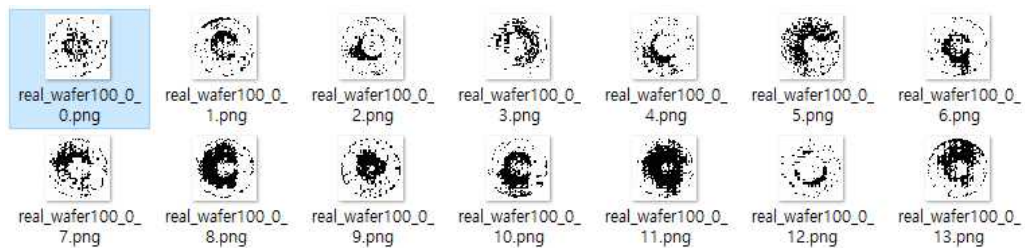
## 개념



(a) 생성 데이터 이미지



(b) 생성 데이터 이진화 이미지



(c) 원본 데이터 이미지

그림 1. 웨이퍼 데이터 셋 생성 및 원본 이미지

그림 1의 (a)는 웨이퍼 데이터 셋에서 생성된 데이터 이미지의 사진이다. 이를 사용하여 임계값 140으로 이진화한 데이터 이미지를 그림 1의 (b)에 나타내었다. 그림 1의 (b)와 (c)를 유관으로 확인하면 유효한 데이터가 생성 되었음을 확인할 수 있다.

```

print("FID between WAFER REAL and FAKE : value=%4.1f" %(fid(h_real, h_fake)))
print("FID between WAFER FAKE and REAL : value=%4.1f" %(fid(h_fake, h_real)))
print("FID between WAFER REAL and REAL : value=%4.1f" %(fid(h_real, h_real)))

/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:18: ComplexWarning: Casting complex values to real discards the imaginary part
FID between WAFER REAL and FAKE : value=133.2
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:19: ComplexWarning: Casting complex values to real discards the imaginary part
FID between WAFER FAKE and REAL : value=133.2
FID between WAFER REAL and REAL : value=-0.0
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:20: ComplexWarning: Casting complex values to real discards the imaginary part

```

그림 2. 생성 데이터 이미지 FID 측정값

이번 실험에서 고려하였던 점은 MNIST 데이터 셋의 데이터 양과 비교하여 웨이퍼 데이터 생성 이미지를 늘려서 진행하였고, 데이터의 양을 늘림으로써 유효한 값이 나오는 것을 확인하였다.



# 연구 노트

## 연구 목표

이미지 생성 알고리즘을 실제 서비스에 적용하기 위해 CPU 및 GPU 환경에서 시간을 측정한다.

## 개념

```
epoch = 93/100, d_loss=0.235, g_loss=5.348
epoch = 94/100, d_loss=0.224, g_loss=5.051
epoch = 95/100, d_loss=0.225, g_loss=4.829
epoch = 96/100, d_loss=0.194, g_loss=7.085
epoch = 97/100, d_loss=0.219, g_loss=6.442
epoch = 98/100, d_loss=0.231, g_loss=5.186
epoch = 99/100, d_loss=0.185, g_loss=6.507
epoch = 100/100, d_loss=0.184, g_loss=6.710
CPU times: user 1min 21s, sys: 49.8 s, total: 2min 10s
Wall time: 1min 36s
```

(a)

```
epoch = 1/10, d_loss=0.311, g_loss=4.325
1 epoch time: 54.78855872154236
epoch = 2/10, d_loss=0.214, g_loss=8.225
1 epoch time: 347.91906094551086
epoch = 3/10, d_loss=0.290, g_loss=16.118
1 epoch time: 51.526278018951416
epoch = 4/10, d_loss=0.195, g_loss=16.118
1 epoch time: 51.39961624145508
epoch = 5/10, d_loss=0.189, g_loss=16.118
1 epoch time: 51.24104022979736
epoch = 6/10, d_loss=0.175, g_loss=16.115
1 epoch time: 51.24602675437927
epoch = 7/10, d_loss=0.171, g_loss=16.118
1 epoch time: 51.246026039123535
epoch = 8/10, d_loss=0.173, g_loss=16.118
1 epoch time: 51.30786156654358
epoch = 9/10, d_loss=0.166, g_loss=16.118
1 epoch time: 51.41956305503845
epoch = 10/10, d_loss=0.170, g_loss=16.118
1 epoch time: 51.457462310791016
Wall time: 13min 33s
```

(b)

그림 1. CPU 및 GPU 코드 실행 시간 비교

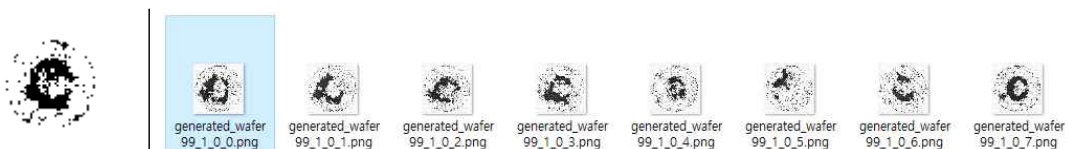


그림 2. 원본 이미지 및 생성 이미지 비교

그림 1은 CPU 및 GPU 환경에서 코드의 실행 속도를 측정하고 비교하여 나타낸 것이다. CPU 환경에서는 학습 1회시 51초의 시간이 소요되었고, GPU 환경에서는 학습 1회시 1초의 시간이 소요됨을 확인하였다.

# 연구 노트

## 연구 목표

원본 데이터 셋으로 학습한 분류 신경망을 이용하여 생성 데이터를 분류하는 실험을 수행한다.

## 개념

I	패키지 로드	이미지 로드	데이터 생성	특징추출 (Batch 128)	Training (30 epochs, Batch 128)	SUM
4,000 장	2.31 -> 2.45 (0.14 G)	2.45 -> 2.47 (0.02 G)	2.50 -> 2.60 (0.02 G)	2.66 -> 3.06 (0.51 G)	2.92 -> 3.05 (0.13 G)	0.02 G

그림 1. 데이터 생성 알고리즘 메모리 소요 측정

```
0_center: 2000
1_donut: 409
2_edge_loc: 2000
3_edge_ring: 2000
4_near_full: 54
5_random: 609
6_scratch: 500
7_none: 2000
8_loc: 1620

SUM: 11,192
```

그림 2. 분류 모델을 생성하는데 필요한 클래스 별 웨이퍼 데이터 셋

```
Epoch 30/30
11192/11192 [=====] - 10s 889us/step - loss: 0.0338 - acc: 0.9882 - val_loss: 1.1197 - val_acc: 0.7472
```

그림 3. 분류 신경망 정확도

```

(384, 120, 120, 1)
[1 1 1 1 1 1 1 1 1 1 1 5 1 1 1 8 1 1 1 1 1 1 1 5 1 1 1 1 5 1 1 8 1 1 1 5 1 1
1 1 8 1 1 1 1 1 1 1 1 1 1 1 1 8 1 1 1 1 1 1 1 5 1 1 1 1 1 5 1 1 1 1 1 1 1 1
1 1 1 5 1 5 1 5 1 1 1 1 1 1 1 5 1 8 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 5 1
1 1 1 5 1 1 1 1 1 1 1 1 7 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 5 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 5 7 1 1 1 1 1 1 1 1 1 1 1 5 5 1 1 1 1 1 1 1 1
1 1 5 1 1 1 1 1 1 1 1 1 1 1 1 1 8 1 1 1 1 1 1 1 1 1 1 1 1 1 5 1 1 1 1 1 1 1 8
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 8 1 1 1 1 5 1 1 1 1 8 1 8 1 1 1 1
1 1 1 1 1 1 1 1 1 1 5 5 1 1 1 1 1 1 1 1 1 1 1 1 0 5 1 1 1 1 1 1 8 5 5 1 1 1 8 5
1 1 1 1 1 5 1 1 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 5 1 5 1 1 1 1 1 1 5 1
1 1 1 1 1 8 1 1 1 1 1 1 1 1 1 1 5 1 1 1 1 1 7 1 1 1 5 1 1 1 5 5 8 1 8 1 1 1 1
1 1 1 1 1 1 1 1 1 1 5 1 1 5 1]

```

그림 4. 클래스 분류

생성 알고리즘을 수행하기 위해서 메모리 소비를 측정한 과정은 그림 1의 순서와 같다. 4000장의 이미지를 생성하기 위해서 총 0.82 G 메모리가 필요한 요구량이다. 생성 알고리즘을 수행하기 위한 순서는 그림 1과 같이 패키지 로드, 이미지 로드, 데이터 생성, 특징 추출, 훈련의 과정을 거치며, 이때 훈련 수는 30, Batch의 크기는 128로 지정하였다.

생성한 이미지의 분류 정확도를 평가하기 위해 분류 신경망 모델을 훈련한다. 이때 사용한 데이터 셋은 원본 데이터 셋에서 각 클래스별 2000장을 분류하여 훈련에 사용하였고, 일부 클래스의 경우 데이터 부족으로 인하여 데이터 클래스 편차가 존재한다. 결과적으로 9개의 클래스에서 추출한 약 11,000 장의 이미지 데이터 셋을 훈련데이터로 사용하였다.

30회 트레이닝을 거쳐 훈련정확도 98%, 평가 정확도 74%인 분류 모델을 만들었고, 이 모델에 생성 데이터를 분류한 결과 약 85 % 의 결과로 분류하는 것을 확인하였다.

# 연구 노트

## 연구 목표

실제 서비스 배포를 위한 응용사례를 조사하고 추후 적용을 위한 디자인을 고안한다.

## 개념

### 2. ImageGen™은?

(1) 샘플 영상으로부터 실제 환경 변화를 반영할 수 있는 다양한 테스트 영상 생성

(2) 주요특징

실제 환경 모델링

복합 환경 모델링

환경 요소 강도 설정

12,000개 이미지 생성(1시간 5M기준)

4~22개 이미지 생성 가능(17,592,186,044,416개)

그림 1. 라온피플 ImageGEN의 주요 특징

데이터 생성 관련하여 응용사례를 참고한 회사는 라온피플이다. 라온피플은 머신러닝 전문 회사로 반도체, 식물, 납 도포 불량 검사 등 딥러닝을 서비스에 효과적으로 적용한 회사중에 하나이다.

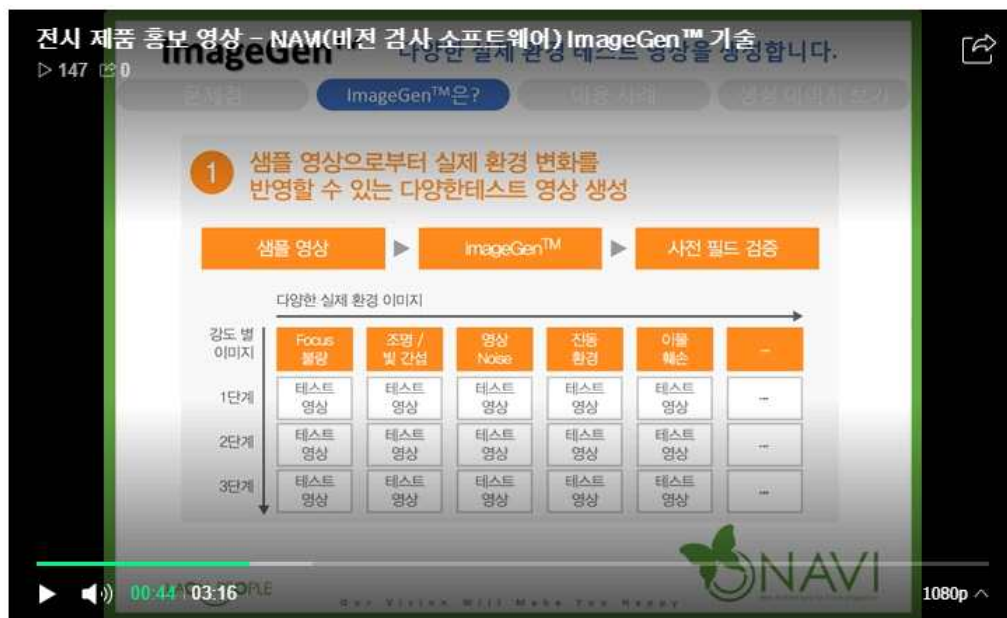


그림 2. 데이터 생성 방법

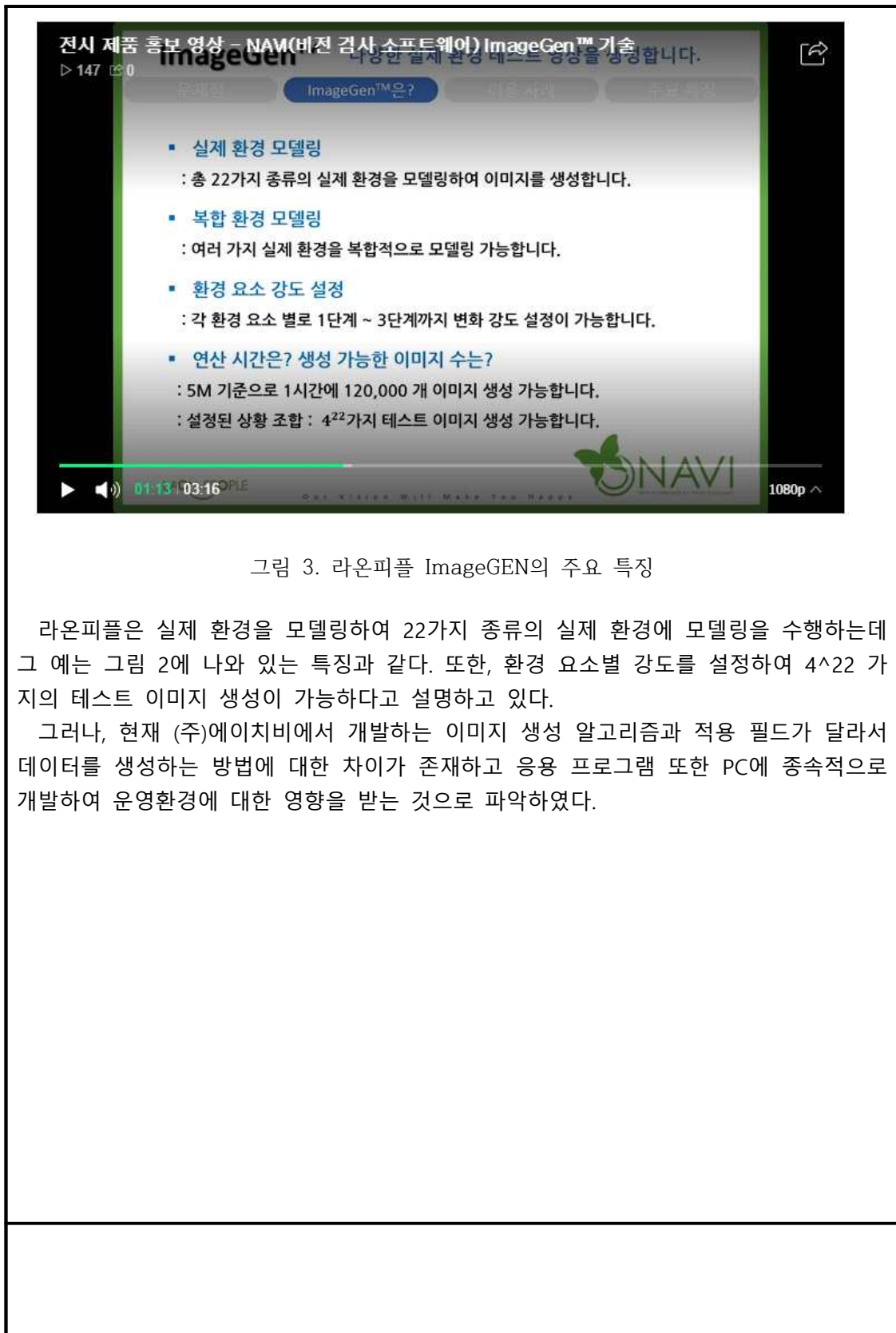


그림 3. 라온피플 ImageGEN의 주요 특징

라온피플은 실제 환경을 모델링하여 22가지 종류의 실제 환경에 모델링을 수행하는데 그 예는 그림 2에 나와 있는 특징과 같다. 또한, 환경 요소별 강도를 설정하여  $4^{22}$  가지의 테스트 이미지 생성이 가능하다고 설명하고 있다.

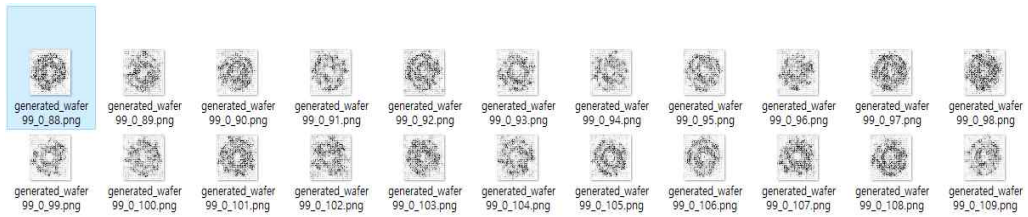
그러나, 현재 (주)에이치비에서 개발하는 이미지 생성 알고리즘과 적용 필드가 달라서 데이터를 생성하는 방법에 대한 차이가 존재하고 응용 프로그램 또한 PC에 종속적으로 개발하여 운영환경에 대한 영향을 받는 것으로 파악하였다.

# 연구 노트

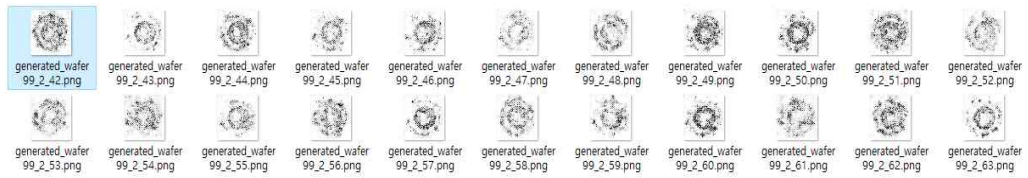
## 연구 목표

데이터 생성에 필요한 클래스별 최소 이미지 수 파악을 추가로 수행한다.

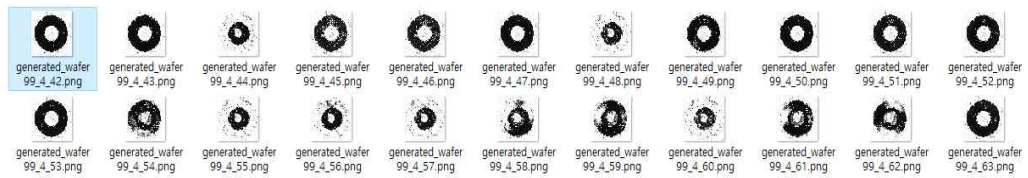
## 개념



(a) 110



(b) 220



(c) 330



(d) 440

그림 1.

실제 산업 현장에서는 이미지 분류 및 생성을 적용하기 위해 데이터 획득이 사실상 어려운 상황에 놓여있는 곳이 많다. 그러한 이유로 소수의 데이터를 이용하여 데이터 증식 후 생성 알고리즘에 적용하기 위한 데이터 증식 실험을 수행하였다.



# 연구 노트

## 연구 목표

이전 연구에서 정한 트레이닝 이미지 수를 이용하여 데이터 생성 실험을 수행한다.

## 개념



그림 1. 웨이퍼 데이터 이미지

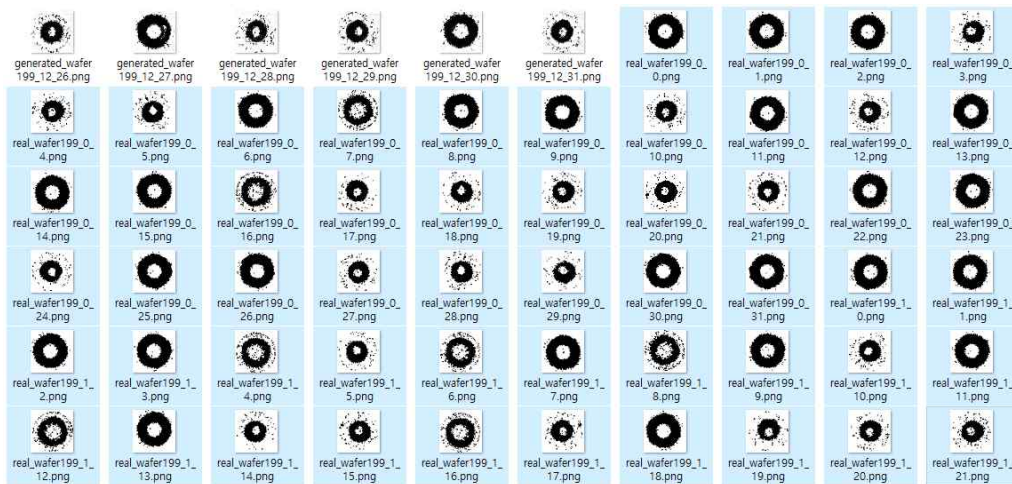


그림 2. 웨이퍼 데이터 생성 이미지



```
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:18: ComplexWarning: Casting complex values to real discards the imaginary part
FID between WAFER READ and FAKE : value=92.0
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:19: ComplexWarning: Casting complex values to real discards the imaginary part
FID between WAFER FAKE and REAL : value=92.0
FID between WAFER REAL and REAL : value=-0.0
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:20: ComplexWarning: Casting complex values to real discards the imaginary part
```

그림 3. FID 측정값

이전 실험에서 데이터 생성에 필요한 이미지의 최소 증식 수로 440으로 결정하였고 이에 따라 데이터 생성 실험을 수행하였다. 최소의 데이터 22장에 대하여 440 장으로 증식하는데 걸리는 시간은 2.5s로 측정되었으며 학습 및 저장을 위한 시간은 학습량 200 epoch를 기준으로 패키지 로드 시간을 포함하여 230s로 측정되었다.

또한, 이진화 및 리사이즈의 추가 과정을 더하여 웨이퍼 데이터 셋 하나의 클래스에 대하여 데이터를 생성하는데 GPU 환경에서 250 s가 걸리는 것으로 확인되었고, 추후 트레이닝 수 및 시간을 줄이는 실험에 대한 필요성이 있음을 확인하였다.

# 연구 노트

## 연구 목표

이전 연구에서 생성된 이미지 유효성으로 평가한 FID 측정값을 낮추는 실험과 알고리즘의 동작 시간을 줄이기 위한 트레이닝 수를 줄이는 실험을 진행한다.

## 개념

### 실험 1. FID value 값 낮추는 실험.

```
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:18: ComplexWarning: Casting complex values to real discards the imaginary part
FID between WAFER READ and FAKE : value=108.7

/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:19: ComplexWarning: Casting complex values to real discards the imaginary part
FID between WAFER FAKE and REAL : value=108.7
FID between WAFER REAL and REAL : value=-0.0

/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:20: ComplexWarning: Casting complex values to real discards the imaginary part
```

#### (a) 실험 1-1

```
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:18: ComplexWarning: Casting complex values to real discards the imaginary part
FID between WAFER READ and FAKE : value=209.3

/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:19: ComplexWarning: Casting complex values to real discards the imaginary part
FID between WAFER FAKE and REAL : value=209.3
FID between WAFER REAL and REAL : value=-0.0

/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:20: ComplexWarning: Casting complex values to real discards the imaginary part
```

#### (b) 실험 1-2

그림 1. FID 측정값을 낮추기 위한 실험에서 FID 측정값

	이미지 수	FID 측정값
실험 1-1	506	108.7
실험 1-2	615	209.3

표 1. 이미지 수에 따른 FID 측정값

FID 측정값을 낮추기 위해 생성 이미지를 위한 학습 이미지 수를 늘려서 실험을 진행함. 학습을 위해 동일한 훈련횟수를 적용하고, 이때 FID 측정값을 비교하였음. 이미지 수의 차이로 훈련시간에서 실험 1-1은 270 s, 실험 1-2는 345 s의 시간이 소요되었고, 약 75 s초의 시간 차이가 발생함. 그림 1과 표 1에서와 같이 이미지 수의 증가가 FID 측정값을 낮추는 결과를 가져오지 않음.

## 실험 2. 트레이닝 수 줄이는 실험

```
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:18: ComplexWarning: Casting complex values to real discards the imaginary part
FID between WAFER READ and FAKE : value=87.3
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:19: ComplexWarning: Casting complex values to real discards the imaginary part
FID between WAFER FAKE and REAL : value=87.3
FID between WAFER REAL and REAL : value=-0.0
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:20: ComplexWarning: Casting complex values to real discards the imaginary part
```

### (a) 실험 2-1

```
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:18: ComplexWarning: Casting complex values to real discards the imaginary part
FID between WAFER READ and FAKE : value=87.7
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:19: ComplexWarning: Casting complex values to real discards the imaginary part
FID between WAFER FAKE and REAL : value=87.7
FID between WAFER REAL and REAL : value=-0.0
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:20: ComplexWarning: Casting complex values to real discards the imaginary part
```

### (b) 실험 2-2

그림 2. 트레이닝 수를 줄이기 위한 실험에서 FID 측정값

	트레이닝 수	학습 시간	FID 측정값
실험 2-1	150	180 s	87.3
실험 2-2	100	120 s	87.7

표 2. 트레이닝 수에 따른 FID 측정값

트레이닝 수를 줄이기 위해 생성 이미지를 위한 학습 횟수를 변화시켜 실험을 진행함. 학습을 위한 트레이닝 수는 '100'과 '150' 두 값을 비교하였고, 이때 훈련에 사용되는 이미지는 400장으로 동일한 이미지를 사용하였음. 트레이닝 수의 차이로 학습 시간에서 실험 2-1은 180 s, 실험 2-2는 120 s의 시간이 소요되었고, 60 s의 시간 차이가 발생하는 것을 확인하였고, 이때 FID 측정값에서 기존 실험 1에서 측정했던 값보다 작은 값이 각각 87.3과 87.7로 측정되는 것을 또한 확인하였음.

이를 벤치마킹 데이터 셋인 MNIST 이미지 데이터 셋과 비교하였을 때, MNIST 데이터의 이미지 셋 6000장을 기준으로 50회 훈련하는데 160 초, 60000 장을 기준으로 1600 초로 훈련 1회 당 각각 3.2초, 32초의 시간이 소요됨. 웨이퍼 데이터 셋 400장의 이미지를 MNIST 데이터 셋 6000 장의 이미지와 비교하였을 때, 웨이퍼 데이터 셋에서 훈련 1회에 1.1 초의 시간이 소요되는 것과 비교하였을 때, 훈련 시간이 적게 소요되는 것 또한 확인할 수 있었음.

# 연구 노트

## 연구 목표

WM-811K 데이터 셋에서 기존 이미지와 다른 이미지를 대상으로 생성 실험을 진행하고, FID 값을 측정함으로써 생성된 이미지가 유효함을 확인한다.

## 개념

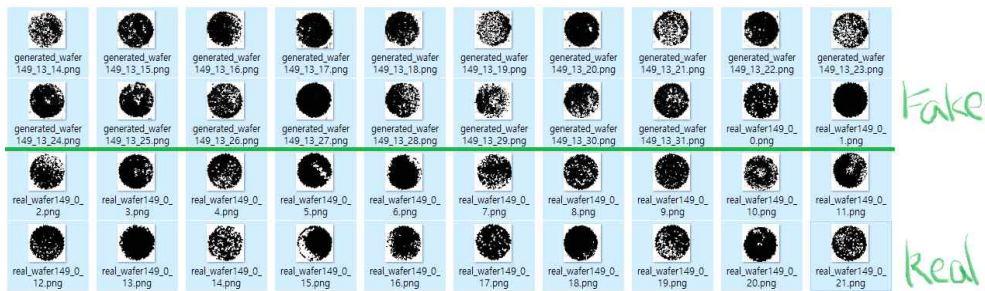


그림 1. 생성 이미지와 원본 이미지의 비교

```
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:18: ComplexWarning: Casting complex values to real discards the imaginary part
FID between WAFER READ and FAKE : value=87.9
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:19: ComplexWarning: Casting complex values to real discards the imaginary part
FID between WAFER FAKE and REAL : value=87.9
FID between WAFER REAL and REAL : value=-0.0
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:20: ComplexWarning: Casting complex values to real discards the imaginary part
```

그림 2. 생성 이미지와 원본 이미지 간의 FID 측정값

그림 1에서 초록색 구분선을 기준으로 초록색 구분선 위의 이미지(Fake로 표시)는 생성된 이미지를 의미하며 초록색 구분선 아래의 이미지(Real로 표시)는 원본 이미지를 의미함. 그림에서와 같이 생성된 이미지와 원본 이미지 간의 구분이 어려울 정도로 비슷한 이미지가 생성되었음.

이때 FID 측정값은 87.9의 값으로 측정되었고, 원본 이미지와 생성 이미지간의 비슷한 정도를 수치적으로 측정함으로써 이미지가 잘 생성된 것을 확인하였음.

# 연구 노트

## 연구 목표

사용자 정의 클래스 이미지를 정의하고 분류기를 훈련하여, 생성한 이미지 분류를 위한 모델을 생성한다.

## 개념

```
Epoch 30/30
3616/3616 [=====] - 3s 797us/step - loss: 0.0078 - acc: 0.9978 - val_loss: 4.1980 - val_acc: 0.5977
Out[5]: <keras.callbacks.History at 0x7f199e1b07b8>
```

### (a) 실험 1

```
Epoch 30/30
3616/3616 [=====] - 3s 817us/step - loss: 0.0129 - acc: 0.9959 - val_loss: 0.5956 - val_acc: 0.8878
Out[5]: <keras.callbacks.History at 0x7ff98a1b07b8>
```

### (b) 실험 2

그림 1. 사용자 정의 클래스 학습 정확도 및 테스트 정확도

index	Pattern	Accuracy(%)	index	Pattern	Accuracy
1	0_circle	100	6	5_nothing	0
2	1_point	0	7	6_one_direct_loc	84.5
3	2_donut	87.4	8	7_half_defect	96
4	3_ring	94.7	9	8_one_scratch	72.6
5	4_full	0		평균	59.77

표 1. 실험 1 클래스별 정확도

index	Pattern	Accuracy(%)	index	Pattern	Accuracy
1	0_circle	100	6	5_nothing	100
2	1_point	83.6	7	6_one_direct_loc	89.1
3	2_donut	78.9	8	7_half_defect	93.7
4	3_ring	100	9	8_one_scratch	56
5	4_full	100		평균	88.78

표 2. 실험 2 클래스별 정확도

이미지 분류 모델을 만들기 위해 임의의 사용자로부터 9개의 클래스 이미지를 분류 받아 설계 및 실험을 진행하였음. 그림 1은 실험 1과 실험 2에서 사용자 정의 클래스 학습 및 테스트 정확도와 학습 손실과 테스트 손실을 나타냄. 실험 1과 실험 2에서 학습 정확도는 100에 근접하였지만 평가 정확도에서 각각 0.59977과 0.8878로 측정되어 편차가 크게 발생하였음. 표1과 표2는 각각 실험 1과 실험 2에서 클래스별 정확도를 표로 나타내었음.

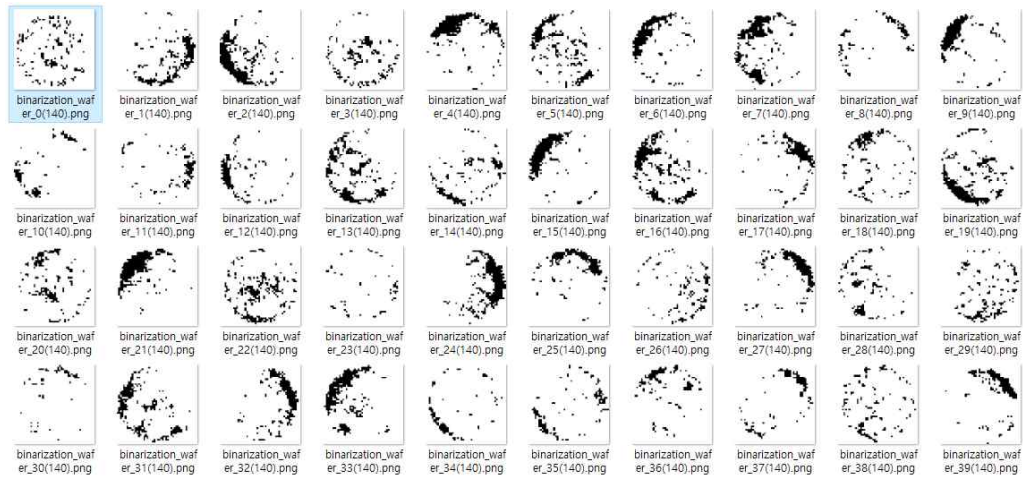


그림 2. one\_scratch 클래스 원본 이미지

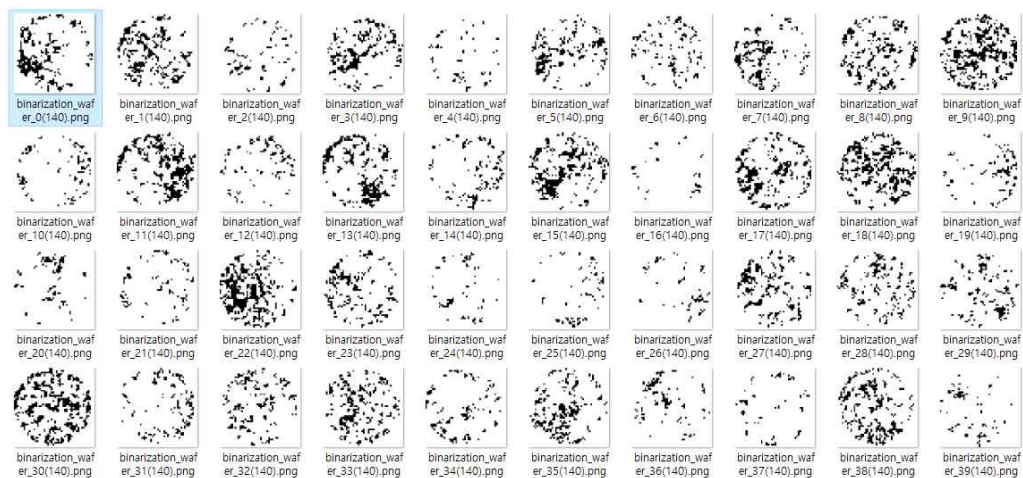


그림 3. one\_scratch 클래스 생성 이미지

실험 2에서 one\_scratch 패턴 이미지에서 클래스 분류 정확도가 낮게 나오는 것을 확인하기 위해서 그림2와 그림 3에 one\_scratch에 대한 원본 이미지와 생성 이미지를 비교함. 본격적으로 생성 알고리즘을 돌리기 전 각 클래스별 패턴을 400장으로 증식하는 과정에서 그림 2에서와 같이 이미지 회전 등에 따른 local 패턴이 상이하여 그림 3과 같은 생성 이미지를 획득하였고, 그에 따라 분류 정확도가 낮게 나온 것으로 확인함. one\_scratch 패턴에 대한 생성 이미지 추가 실험 필요할 것으로 판단하였음.



# 연구 노트

## 연구 목표

Scratch 패턴 이미지 생성 실험을 하고 클래스 분류를 위한 모델을 얻기 위한 학습을 진행한다.

## 개념

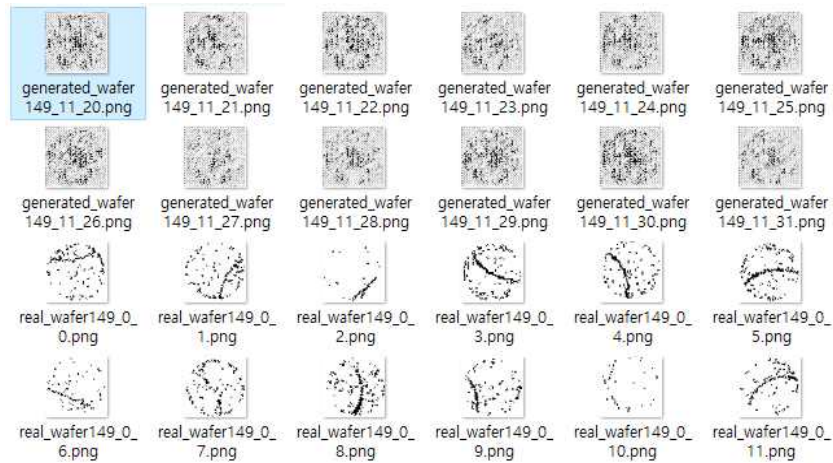


그림 1. Scratch 패턴 생성 이미지와 원본 이미지

index	Pattern	Accuracy(%)	index	Pattern	Accuracy
1	0_circle	100	6	5_nothing	100
2	1_point	51.5	7	6_one_direct_loc	97.1
3	2_donut	55.4	8	7_half_defect	96
4	3_ring	95.3	9	8_one_scratch	14.9
5	4_full	100			

표 1. 사용자 정의 클래스별 정확도

사용자 정의 클래스에서 one\_scratch 패턴 이미지를 분리하여 생성 실험을 진행하였음. one\_scrach 패턴의 경우 이미지의 분포가 다른 패턴들과 달리 일정하지 않고 다양한 유형으로 존재하는 것으로 확인되었음. 그에 따라서 그림 1에서와 같이 원본 이미지의 결함 분포와 상이한 패턴으로 이미지가 생성되는 것을 확인하였음. 또한, 표1 에서도 scract 패턴에서도 분류결과가 다른 클래스와 비교하여 현저히 낮은 것을 확인함.



# 연구 노트

## 연구 목표

Scratch 패턴의 이미지 데이터 증식 방법을 변경하여 분류 실험을 진행한다.

## 개념

```
Epoch 30/30
3636/3636 [=====] - 3s 759us/step - loss: 0.0028 - acc: 0.9992 - val_loss: 0.4874 - val_acc: 0.9194
<keras.callbacks.History at 0x7f0366855550>
```

그림 1. 그림 1. 사용자 정의 클래스 학습 정확도 및 테스트 정확도

index	Pattern	Accuracy(%)	index	Pattern	Accuracy
1	0_circle	88.9	6	5_nothing	29.8
2	1_point	100	7	6_one_direct_loc	89.1
3	2_donut	87.4	8	7_half_defect	76
4	3_ring	100	9	8_one_scratch	100
5	4_full	29.8		평균	91.94

표 1. 사용자 정의 클래스별 정확도

앞에서 Scratch 패턴 이미지의 경우, 이미지에서 결함의 분포가 다양한 유형으로 존재하는 것을 확인하였고, 그에 따라 Scratch 패턴과 같은 이미지의 경우 생성 방법을 기존 클래스와는 다르게 하여야 한다고 판단하였고, 그에 따라 획득한 생성 이미지를 이용하여 기존과 동일한 실험을 진행하였음.

이미지 생성을 위한 방법으로 딥러닝 프레임워크에서 제공하는 ImageDataGenerator 함수를 사용하였음. 딥러닝 프레임워크의 ImageDataGenerator는 사용자 정의 클래스 이미지에서 회전, 이미지 확대/축소, 상하/좌우 대칭 등 여러 가지 인자를 제공하여 다양한 방법으로 데이터를 증식할 수 있음.

기존 클래스는 기존과 동일한 방법으로 데이터 생성 알고리즘을 이용하여 데이터 증식을 진행하였고, one\_scratch 클래스에 대해서만 ImageDataGenerator 함수를 사용하여 데이터 증식을 진행함. 표1의 실험 결과로부터 one\_scratch 분류 정확도 및 평균 정확도가 증가한 것을 확인하였음.

## 연구 노트

### 연구 목표

ImageDataGenerator 클래스에서 제공하는 이미지 변환 인자를 이용하여 이미지 변환 실험을 하고, 유효한 변환 인자를 확인한다.

### 개념

featurewise_center	데이터셋에 대해 특성별로 인풋의 평균이 0이 되도록 함.
samplewise_center	각 샘플의 평균이 0이 되도록함.
featurewise_std_normalization	인풋을 각 특성 내에서 데이터셋의 표준편차로 나눔.
samplewise_std_normalization	각 인풋을 표준편차로 나눔.
zca_epsilon	영위상 성분분석 백색화의 엡실론 값.
zca_whitening	영위상 성분분석 백색화를 적용할지 여부.
rotation_range	무작위 회전의 각도 범위.
width_shift_range	
height_shift_range	
brightness_range	밝기 정도를 조절할 값의 범위.
shear_range	충밀리기의 강도.
zoom_range	무작위 줌의 범위.
channel_shift_range	무작위 채널 이동의 범위.
fill_mode	인풋 경계의 바깥 공간은 모드에 따라 다르게 채워짐.
cval	'fill_mode = "constance"'인 경우 경계 밖 공간에 사용하는 값.
horizontal_flip	인풋을 무작위로 가로로 뒤집음.
vertical_flip	인풋을 무작위로 세로로 뒤집음.
rescale	크기 재조정 인수.
preprocessing_function	각 인풋에 적용되는 함수.
data_format	이미지 데이터 형식.
validation_split	검증의 용도로 남겨둘 이미지의 비율.
dtype	생성된 배열에 사용할 자료형.

표 1. ImageDataGenerator 클래스에서 제공하는 이미지 변환 인자

케라스 ImageDataGenerator에서 이미지 변환을 위해 제공하는 인자 목록을 표1에 나타냄. 실시간 데이터 증감을 사용해서 텐서 이미지 데이터 배치를 생성하고, 데이터에 대해 루프가 순환됨.

```

1 (x_train, y_train), (x_test, y_test) = cifar10.load_data()
2 y_train = np_utils.to_categorical(y_train, num_classes)
3 y_test = np_utils.to_categorical(y_test, num_classes)
4
5 datagen = ImageDataGenerator(featurewise_center=True,
6                             featurewise_std_normalization=True,
7                             rotation_range=20,
8                             width_shift_range=0.2,
9                             height_shift_range=0.2,
10                            horizontal_flip=True)
11
12 datagen.fit(x_train)
13
14 model.fit_generator(datagen.flow(x_train, y_train, batch_size=32),
15                   steps_per_epoch=len(x_train) / 32, epochs=epochs)
16
17 for e in range(epochs):
18     print('Epoch', e)
19     batches = 0
20     for x_batch, y_batch in datagen.flow(x_train, y_train, batch_size=32):
21         model.fit(x_batch, y_batch)
22         batches += 1
23     if batches >= len(x_train) / 32:
24         break

```

그림 1. flow(x, y) 사용

```

1 train_datagen = ImageDataGenerator(rescale=1./255,
2                                   shear_range=0.2,
3                                   zoom_range=0.2,
4                                   horizontal_flip=True)
5
6 test_datagen = ImageDataGenerator(rescale=1./255)
7
8 train_generator = train_datagen.flow_from_directory('data/train',
9                                                    target_size=(150, 150),
10                                                    batch_size=32,
11                                                    class_mode='binary')
12
13 validation_generator = test_datagen.flow_from_directory('data/validation',
14                                                        target_size=(150, 150),
15                                                        batch_size=32,
16                                                        class_mode='binary')
17
18 model.fit_generator(train_generator,
19                   steps_per_epoch=2000,
20                   epochs=50,
21                   validation_data=validation_generator,
22                   validation_steps=8000)

```

그림 2. flow\_from\_directory(directory) 사용

# 1. 사용자관리 화면 디자인(추가)

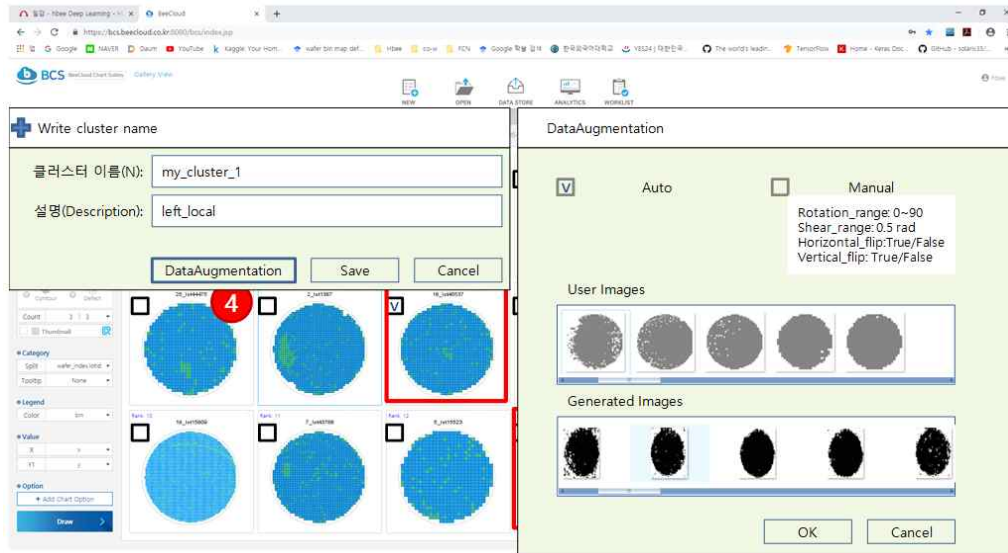


그림 3. 데이터 증식 UI Concept

Epoch 30/30  
3636/3636 [=====] - ls 407us/step - loss: 0.0343 - acc: 0.9901 - val\_loss: 2.3117 - val\_acc: 0.7021

그림 4. 사전 훈련 모델을 이용한 분류기 훈련

ImageDataGenerator 클래스를 이용하여 데이터 생성 코드를 그림1과 그림 2에 나타냄. 이때 웨이퍼 데이터 셋에 적용할 수 있는 유효한 인자는 rotation\_range, shear\_range, horizontal\_flip, vertical\_flip 4가지로 확인됨.

이전 연구를 바탕으로 그림 3에 데이터 증식을 위한 UI를 설계함. 그림 3의 왼쪽그림에서 ‘클러스터 이름’과 ‘설명’란을 적고 ‘DataAugmentation’을 클릭하면 그림 3의 오른쪽 그림과 같은 새로운 창이 열림. 데이터 생성 모드가 'Auto'인지 'Manual'인지 여부에 따라서 생성 알고리즘 또는 ImageDataGenerator 함수를 사용하고, 'Manual'일 경우 앞서 확인된 유효한 인자의 입력값을 인풋으로 사용하여 알고리즘을 실행함. 'User Images'란에는 증식의 대상이 되는 원본 이미지가 찍어지고, 'Generated Images'란에는 생성 알고리즘 또는 ImageDataGenerator로 생성된 이미지가 찍어짐.

또한, 그림 4의 출력결과와 같이 테스트 정확도가 약 70%인 사전 훈련 모델을 이용한 분류기 모델 만드는 실험을 수행하였음.

# 연구 노트

## 연구 목표

데이터 생성과 관련하여 타사의 데이터 생성 솔루션과 비교하고, Scatter Analytics 버전의 UI를 설계한다.

## 개념

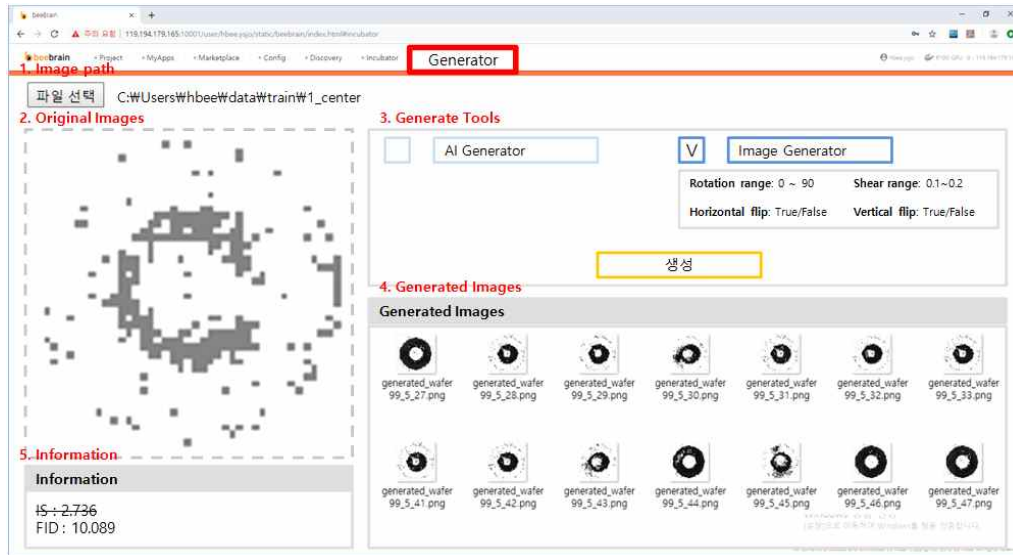


그림 1. 데이터 생성을 위한 Generator UI

데이터 생성을 위한 Scatter Analytics 솔루션을 Image path, Original Images, Generate Tools, Generated Images, Information으로 구성하였고, 설계한 Generator UI 는 그림 1과 같음.

Image path는 사용자가 정의한 클래스에 이미지를 업로드한 후 사용자 정의 클래스 이미지가 저장되어 있는 경로를 말함. Original Images는 업로드한 이미지를 미리 볼 수 있도록 공간을 구성하였음. Generate Tools에서는 데이터 생성을 AI 또는 User의 선택에 따라 증식할 수 있도록 선택할 수 있음. 마지막으로 Generated Images는 생성된 이미지를 볼 수 있도록 하고, Information에서는 생성 이미지의 유효성 평가할 수 있도록 평가값을 제공함.

# 연구 노트

## 연구 목표

생성 알고리즘의 성능을 평가하기 인셉션 점수(Inception score, IS)를 추가로 측정하여 생성 웨이퍼 데이터 셋을 평가한다.

## 개념

$$IS(G) = \exp(\mathbb{E}_{\hat{x} \sim p_g} KL(p(y|\hat{x})||p(y)))$$

그림 1. 인셉션 점수

**Table 3.2** Average Inception score for various data, dimensions and GANs

Data	Model	1	2	3	10	50	100
Mnist	<b>GAN-MM</b>	1.15	2.17	2.17	<b>2.19</b>	2.18	<b>2.19</b>
	GAN-NS	1.76	<b>2.22</b>	2.18	2.19	2.19	2.19
	LSGAN	1.82	2.15	<b>2.29</b>	2.19	2.17	2.18
	WGAN	1.92	2.14	2.15	2.31	2.23	2.23
	WGAN-GP	1.92	<b>2.27</b>	2.22	2.19	2.20	2.22
	DRAGAN	1.68	2.14	<b>2.25</b>	2.19	2.19	2.17
	BEGAN	1.90	<b>2.26</b>	2.17	2.21	<b>2.26</b>	<b>2.26</b>
Fashion -Mnist	GAN-MM	1.50	4.02	4.06	<b>4.43</b>	4.33	4.35
	GAN-NS	2.46	4.11	4.31	<b>4.43</b>	4.37	4.40
	LSGAN	1.36	4.04	<b>4.39</b>	4.32	4.33	4.34
	<b>WGAN</b>	3.19	3.61	<b>3.91</b>	3.89	3.79	3.76
	WGAN-GP	2.74	3.65	4.03	4.20	<b>4.28</b>	4.24
	DRAGAN	2.32	3.72	4.04	<b>4.42</b>	4.02	4.05
	BEGAN	2.82	3.82	<b>4.26</b>	4.25	4.02	4.05
Cifar-10	GAN-MM	1.01	2.46	3.15	4.57	5.12	<b>5.29</b>
	GAN-NS	2.13	2.81	3.28	4.92	4.67	<b>5.19</b>
	LSGAN	1.72	2.85	3.05	4.70	<b>5.04</b>	4.83
	WGAN	2.34	2.95	3.09	<b>3.10</b>	2.69	2.85
	<b>WGAN-GP</b>	1.67	1.89	2.18	2.21	2.38	<b>2.49</b>
	DRAGAN	1.90	2.67	2.80	3.79	<b>4.28</b>	4.16
	BEGAN	1.61	2.40	2.92	<b>4.32</b>	3.62	4.26
CelebA	GAN-MM	1.58	1.52	1.01	2.13	2.20	<b>2.43</b>
	GAN-NS	<b>2.58</b>	2.02	2.15	2.37	2.19	2.26
	LSGAN	2.05	2.24	2.02	2.23	<b>2.9</b>	2.24
	WGAN	<b>2.77</b>	2.25	2.39	2.27	2.24	2.26
	WGAN-GP	1.72	2.47	2.55	<b>3.03</b>	2.65	2.45
	DRAGAN	2.22	2.00	<b>2.58</b>	2.21	2.21	2.24
	<b>BEGAN</b>	1.47	<b>2.35</b>	2.12	2.14	2.22	2.22

그림 2. 입력 데이터, 차원, GAN 종류에 따른 평균 인셉션 점수

```
inception_score = get_inception_score(image_array, splits=10)
```

```
Calculating Inception Score with 416 images in 10 splits  
Inception Score calculation time: 45.427537 s
```

```
# Inception score of GAN  
print(inception_score)
```

```
(1.0099909, 0.004884649)
```

```
# Inception score of ImageDataGenerator  
print(inception_score)
```

```
(1.1343493, 0.11047577)
```

그림 3. WM-811K 생성 데이터 인셉션 점수

인셉션 모형(Inception model, Szegedy, 2016)은 1,000개의 클래스와 120만개의 이미지로 구성된 ImageNet(Deng, 2009) 자료를 사전학습(pre-trained)한 CNN모형으로 이미지가 입력되면 각 1,000개의 클래스에 속할 확률 벡터를 출력함. 이는 전이학습(transfer learning)과 미세조정(fine tuning)에 널리 사용됨. 생성 데이터를 인셉션 모형에 입력한 결과를 이용하여 그림 1과 같이 IS를 계산하여 생성 모형을 평가하는데 값이 클수록 좋은 품질을 의미함(Barratt and Sharma, 2018). 여기서  $p(y|\hat{x})$ 는 조건부 클래스 분포(conditional class distribution)이고  $p(y)$ 는 주변 클래스 분포(marginal class distribution)로 IS는 1이상 1,000이하의 값을 가질 수 있지만 대개는 2 근방의 값을 가짐(Barratt and Sharma, 2018).

대부분의 모형과 데이터 셋에서 2차원부터 IS 값이 2 이상의 값을 가지는 것을 그림 2에서 알 수 있음. 차원을 줄이면 계산에 필요한 공간과 시간을 절약할 수 있으나 많은 연구에서 전형적으로 잠재공간의 크기를 100차원으로 사용함.

WM-811K 데이터 셋을 이용하여 GAN과 ImageDataGenerator 알고리즘으로 생성한 데이터 셋에 대하여 각각 IS값을 측정하였음. 측정한 값은 그림 3과 같이 각각 1.009, 1.134로 측정되었고, 2근방의 값을 가지지 못하는 것에 대해서 이미지의 크기가 벤치마크로 제공하는 데이터와의 차이로 인하여 낮게 나온 것으로 평가하였음.

\* 참고문헌: Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

\* 참고문헌: Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255). Ieee.

\* 참고문헌: Barratt, S., & Sharma, R. (2018). A note on the inception score. arXiv preprint arXiv:1801.01973.



## 연구 노트

### 연구 목표

IS(Inception Score)을 계산하는 script를 추가하여 표준출력으로 나타내고, 데이터 생성 시 필요한 저장 공간, 실행 속도 및 메모리 점유율을 각각 측정한다.

### 개념

original images 60 장 기준 / 1 class

generated images: 400 장 기준

저장 공간			
GAN		ImageDataGenerator	
original_image: 148KB			
1. down size	26KB	1. increasing	1.64MB
2. increasing	1.64MB		
3. generated	1.62MB		
4. binarize	169KB		
5. up size	1.37MB		
increasing_image: 1.65MB			
inception model: 169MB			
모델 제외할 경우	6MB	모델 제외할 경우	3MB
합계	약 175MB	합계	약 172MB

그림 1. 데이터 생성 시 필요한 저장 공간

original images 60 장 기준 / 1class

generated images: 400 장 기준

실행 위치: P100

소요 메모리			
GAN		ImageDataGenerator	
사용량	3GB	사용량	3GB

\* FID값 계산할 때 44개의 core에서 CPU 점유율 100%까지 올라감.

그림 2. 데이터 생성 시 소요 메모리

original images 60 장 기준 / 1class

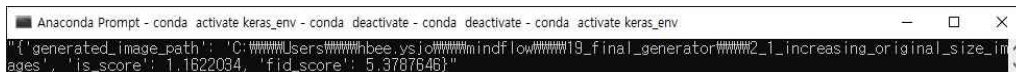
generated images: 400 장 기준

실행 위치: P100

소요 시간			
GAN		ImageDataGenerator	
패키지 로드: 1.9s			
1. down size	0.06s	1. increasing	1.2s
2. increasing	0.4s		
3. generated (150 epochs)	195s		
4. binarize	0.8s		
5. up size	0.3s		
increasing image for compare: 1.2s (FID 구할 때 사용)			
SI: 3.5s			
FID: 약 58s			
합계	약 261s (4분 21초)	합계	약 66s (1분 6초)

그림 3. 데이터 생성 시 script 실행에 소요되는 시간

IS(Inception Score)을 계산하는 script를 추가하여 표준출력으로 출력하였음. 표준출력으로는 아래 그림 4와 같이 생성된 이미지의 저장경로와 IS 및 FID 값과 함께 출력됨.



```

{"generated_image_path": "C:\\Users\\bee.ysj\\AppData\\Local\\Temp\\mindflow\\19_final_generator\\2_1_increasing_original_size_images", "is_score": 1.1622034, "fid_score": 5.3787646}

```

그림 4. 표준 출력 예시

데이터 생성 script 실행 시 필요한 저장 공간은 그림 1과 같음. 사용자 선택에 따라 AI generator(GAN) 또는 user generator(ImageDataGenerator)를 실행하며 AI generator의 경우, 그림 1과 같이 5가지의 과정이 순차적으로 실행됨. 또한, 데이터 생성에 소요되는 시간은 그림 3과 같으며 AI generator(GAN)의 경우 데이터 생성에 필요한 학습으로 인하여 약 3분 가량이 더 소요되는 것을 확인하였음. 마지막으로 메모리 사용량의 경우 AI generator와 user generator 모두 약 3GB의 메모리가 동일하게 소요되는 것을 측정하였고, FID 값을 계산할 경우 CPU 점유율이 최대치로 올라가는 것 또한 확인되었음.

# 연구 노트

## 연구 목표

기존 데이터 증식 화면의 구성을 수정하고, 화면 구성에 따른 스크립트 실행을 점검한다.

## 개념

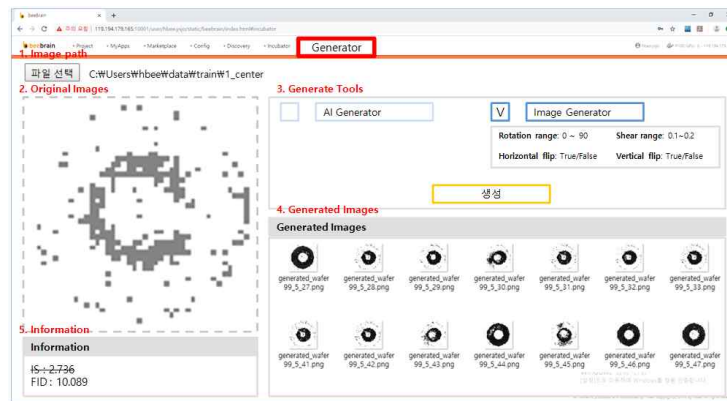


그림 1. 수정 전 데이터 증식 화면 구성

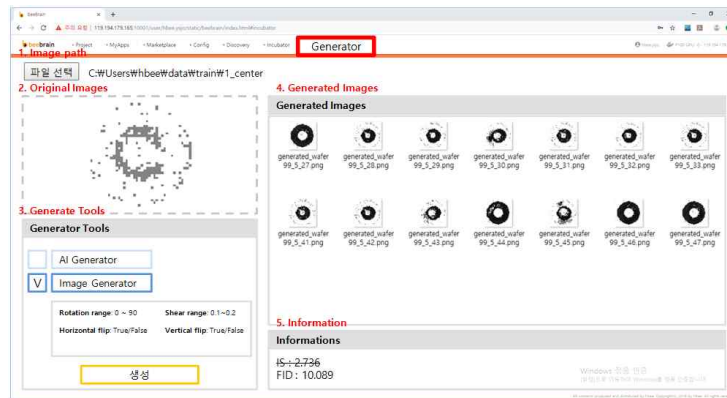


그림 2. 수정 후 데이터 증식 화면 구성

기존 데이터 증식 화면의 구성은 그림 1과 같음. 수정 전 데이터 증식 화면 구성의 경우 파일선택, 원본 이미지, 생성 도구, 생성 이미지, 생성 정보가 사용자의 시선을 분산시켜서 생성 이미지에 대한 정보 전달의 효과가 미비하였으나 그림 2와 같이 생성도구와 생성 정보를 원본이미지 및 생성 이미지 아래에 위치시켜서 생성 이미지의 정보 전달의 효과를 높게 구성하고자 하였고 기존 스크립트 실행 동작과 동일하게 실행할 수 있도록 함.