cGAN 기반
무작위 창호 도식 데이터셋을 활용한
이미지 생성 모델

강해성 건축학 전공 김찬규 건축공학 전공

연구 배경 및 목표

연구 배경
cGAN의 건축 산업에의 활용

1. 최근 논란이 있었던 **공동주택 하자 판별**을 위한 인공지능 모델과 모델의 학습 데이터셋 구축의 필요성

2. 공동주택의 하자 판별을 위해 **'하자 없는 이미지'**와 **'하자 있는 이미지'** 데이터셋 필요

3. cGAN으로 생성된 가상의 레퍼런스를 건물 구성요소가 갖는 형태적 다양성을 보이고 **하자 판별 모델 구축의 기초 기반 마련**

연구 목표
모델 학습을 위한 데이터셋의 생성

1. 모델의 식별에 용이하도록 **비교적 유형화된 형태와 크기**를 갖는 건물의 **창호**를 대상으로 설정

2. 창호 하자 판별에 필요한 **'하자 없는 이미지'**의 데이터셋의 부족을 타개하기 위해 해당 데이터셋 생성 모델 구축

3. 실제 창호 이미지 촬영의 어려움으로 인해 **연구자에 의해 생성**된 **창호의 형태를 간략화한 하자 도식 이미지**를 학습하여 다양한 창호 하자 이미지를 생성하는 모델 구축

그림. 공동 주택 하자 판별 모델의 간략화된 절차도

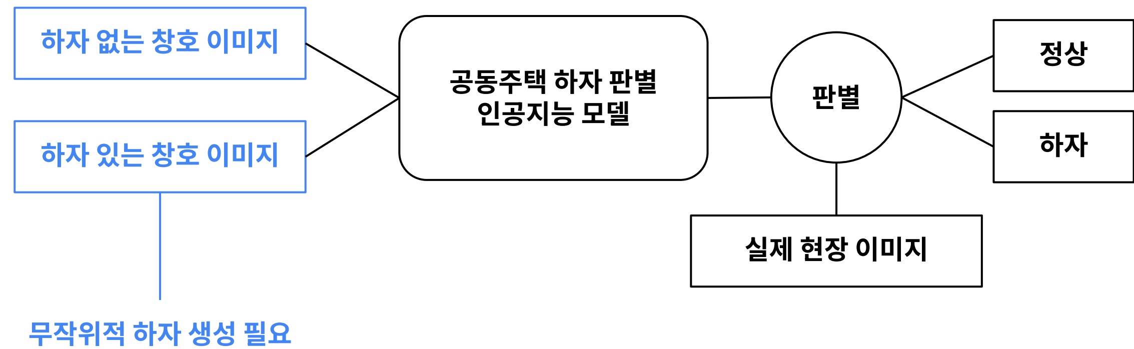
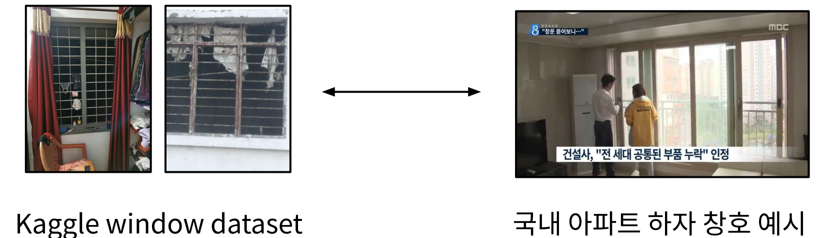


그림. 오픈 데이터셋과 현실 수요와의 차이



연구 배경 및 목표

시행착오
하자 있는 창호의 generator 학습의 어려움

발견사항
1 Epoch가 커져도 큰 변화가 없으며 학습이 진행되었는지 평가하기 어려움
2 데이터셋의 문제로 파악

그림. 무작위로 생성된 하자 있는 창호 도식 데이터셋

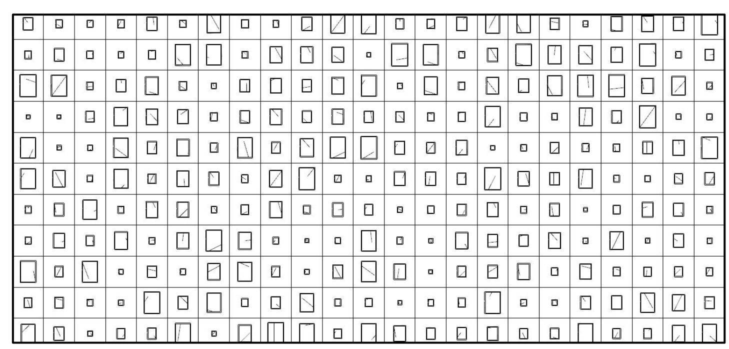


그림. 하자 있는 창호 도식

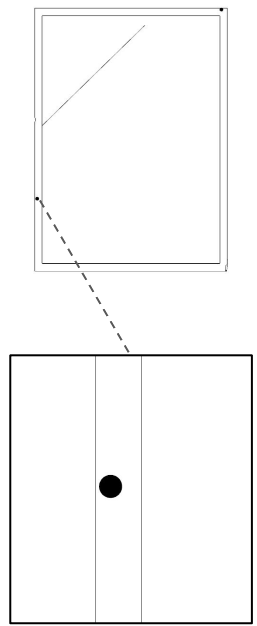
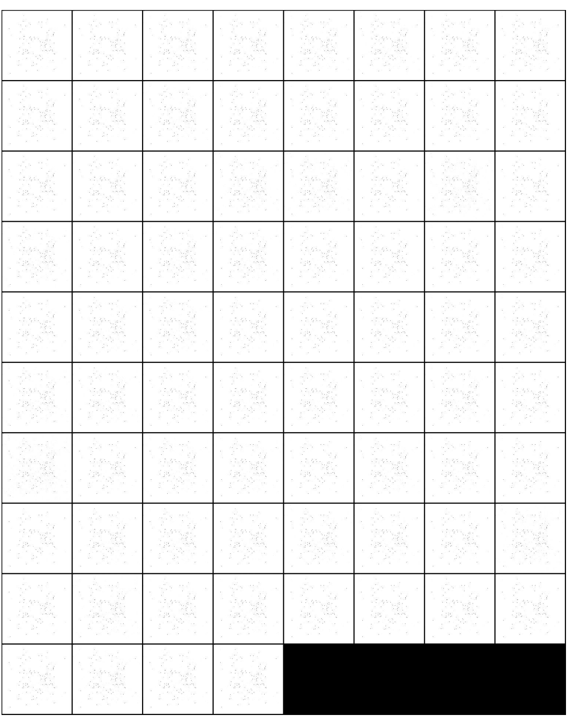
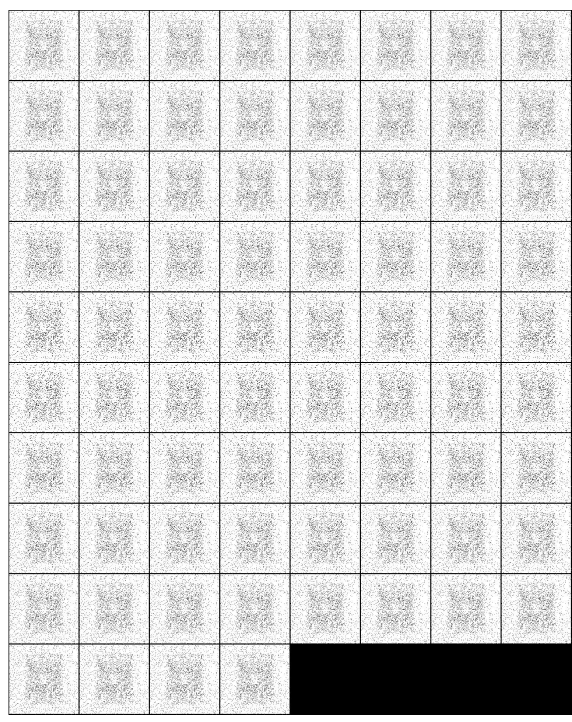


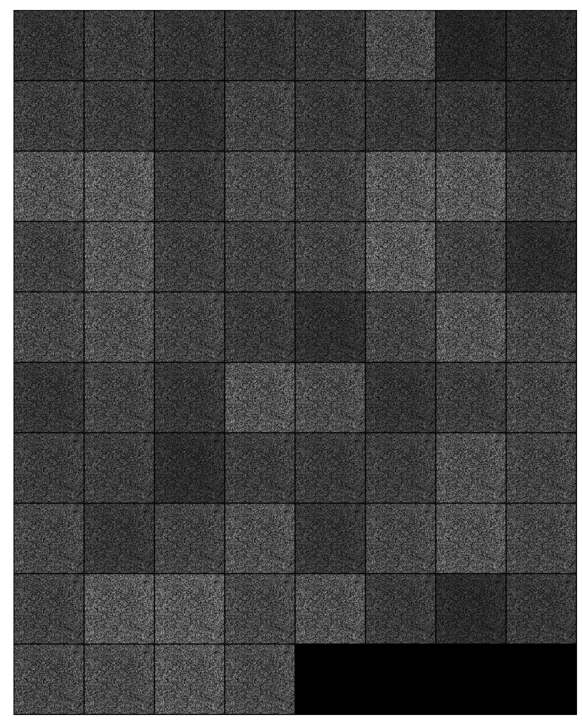
그림. Conditional GAN – 1st try



Epoch: 100



Epoch: 265



Epoch: 300

연구 방법

창호 하자 이미지와 조건을 연동한 모델 학습과 이미지 생성

1. 창호를 도식화한 데이터셋을 제작

2. 데이터셋 학습

3. 창호 도식 이미지 생성

그림. 연구 절차

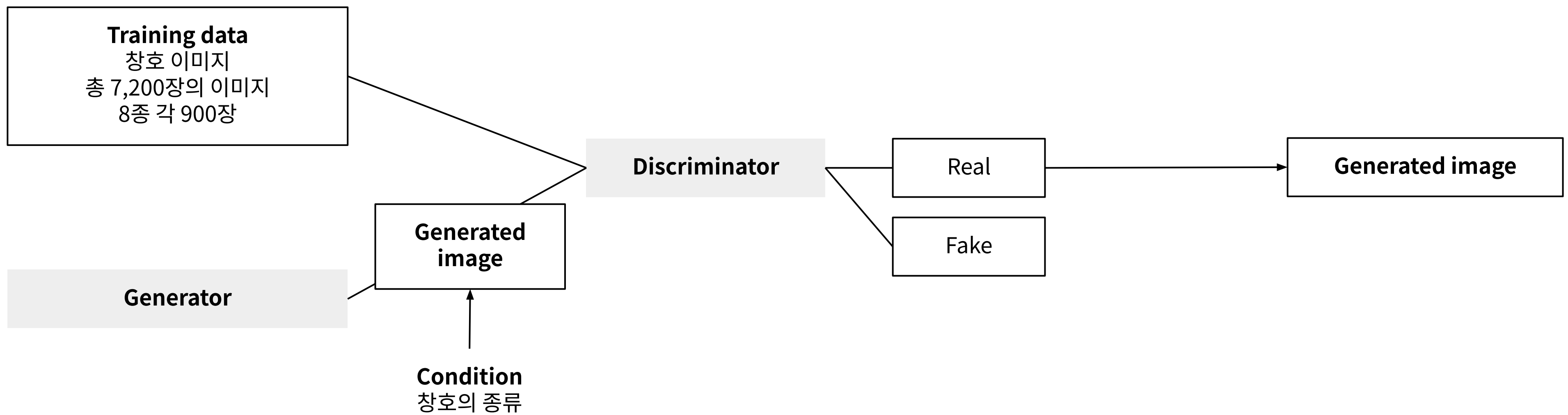
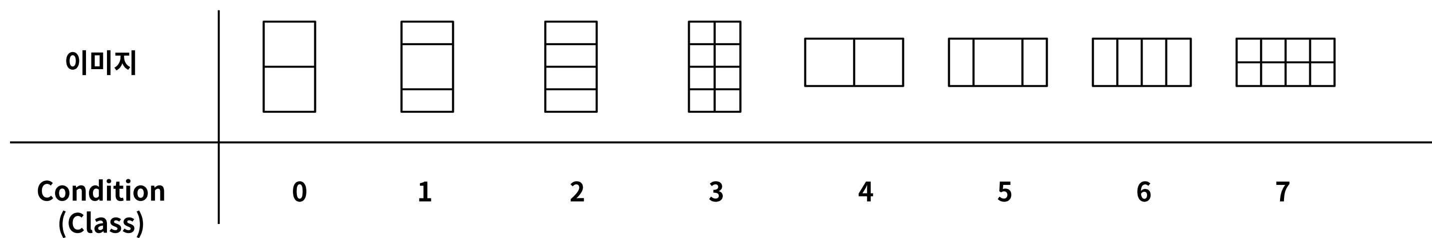


그림. 이미지당 부여된 조건



연구 방법

모델 학습을 위한 데이터셋 준비 방법

1. 창호를 표현하는 도식화 규칙 설정 및 스크립팅	라이노와 그래스호퍼 이용
2. 난수를 부여하여 각 유형 내에서 창호 이미지를 무작위로 생성	라이노와 그래스호퍼 이용
3. 생성된 이미지를 일정 크기로 분할하여 저장	파이썬과 라이브러리 Pillow 이용

그림. 창호 도식화

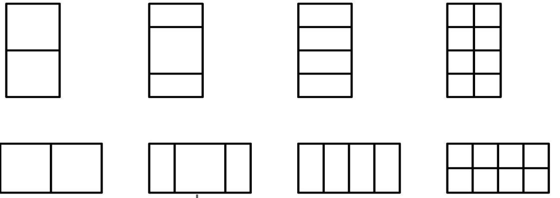


그림. 난수에 의해 무작위로 생성된 창호 도식

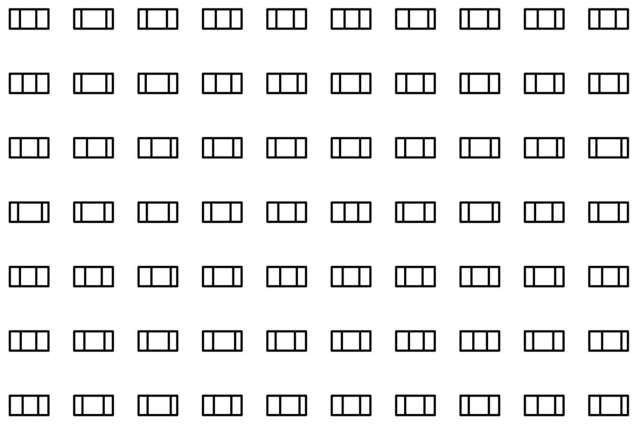


표. 시행착오와 MNIST 데이터셋의 교훈으로부터 도출한 창호 도식화 규칙

창호 도식화 규칙	목적
학습에 도움이 되도록 창호의 프레임이 일관된 위상을 갖는 범위 내에서 무작위로 이미지를 생성	도형의 형태가 무작위로 나타날 경우 모델이 데이터에서 규칙성을 찾아내어 생성하지 못하므로 이를 예방
창호 프레임에 의한 분절되는 창의 크기는 다양하게 하되 창호 외곽 치수는 동일하게 유지	창호 외곽선의 크기가 무작위로 나타날 경우 학습에 어려움을 보였기 때문에 동일하게 맞추어 학습 시행
가로 방향(landscape)과 세로 방향(portrait)에 따른 창호 도식의 형태 차이	형태의 확실한 차이를 주어 모델이 잘 학습할 수 있게끔 유도할 목적

그림. 일관된 위상을 갖는 무작위 창호 도식 생성

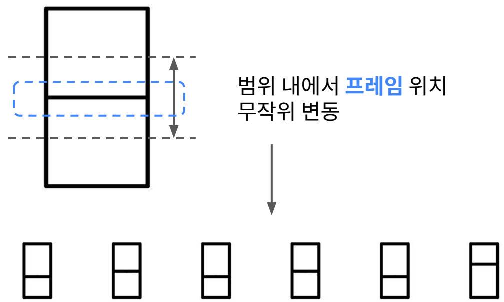
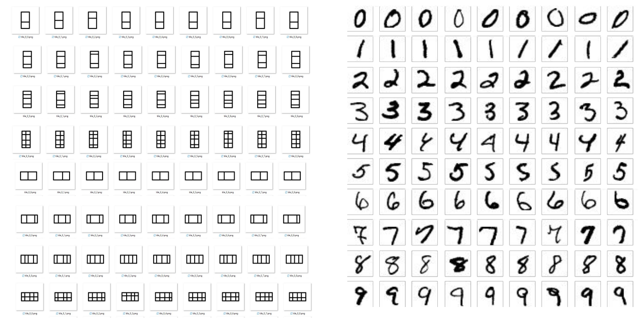


그림. 본 연구의 데이터셋과 MNIST 데이터셋 비교



연구 방법

모델 학습을 위한 데이터셋 준비 방법

1. 창호를 표현하는 도식화 규칙 설정 및 스크립팅	라이노와 그래스호퍼 이용
2. 난수를 부여하여 각 유형 내에서 창호 이미지를 무작위로 생성	라이노와 그래스호퍼 이용
3. 생성된 이미지를 일정 크기로 분할하여 저장	파이썬과 라이브러리 Pillow 이용

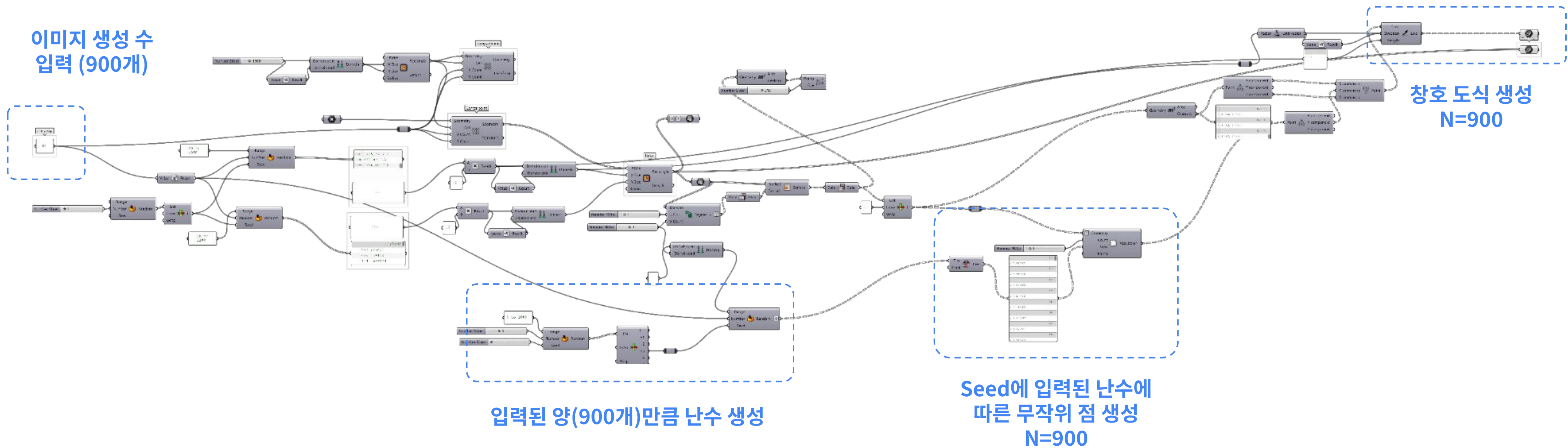
그림. 난수 부여하여 이미지 대량 생성

연구 방법

모델 학습을 위한 데이터셋 준비 방법

1. 창호를 표현하는 도식화 규칙 설정 및 스크립팅	라이노와 그래스호퍼 이용
2. 난수를 부여하여 각 유형 내에서 창호 이미지를 무작위로 생성	라이노와 그래스호퍼 이용
3. 생성된 이미지를 일정 크기로 분할하여 저장	파이썬과 라이브러리 Pillow 이용

그림. 그래스호퍼 스크립트



연구 방법

모델 학습을 위한 데이터셋 준비 방법

1. 창호를 표현하는 도식화 규칙 설정 및 스크립팅 — 라이노와 그래스호퍼 이용
2. 난수를 부여하여 각 유형 내에서 창호 이미지를 무작위로 생성 — 라이노와 그래스호퍼 이용
3. **생성된 이미지를 일정 크기로 분할하여 저장** — 파이썬과 라이브러리 Pillow 이용

그림. 그래스호퍼 생성 이미지
10,630px*10,630px

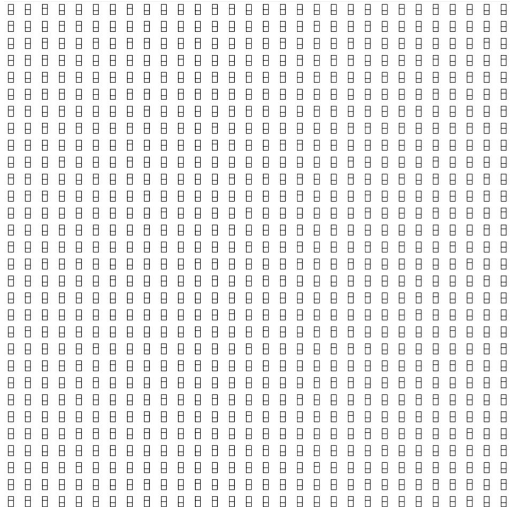


그림. 이미지 분할과 데이터셋 생성

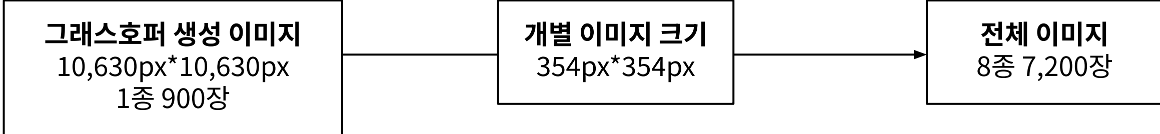
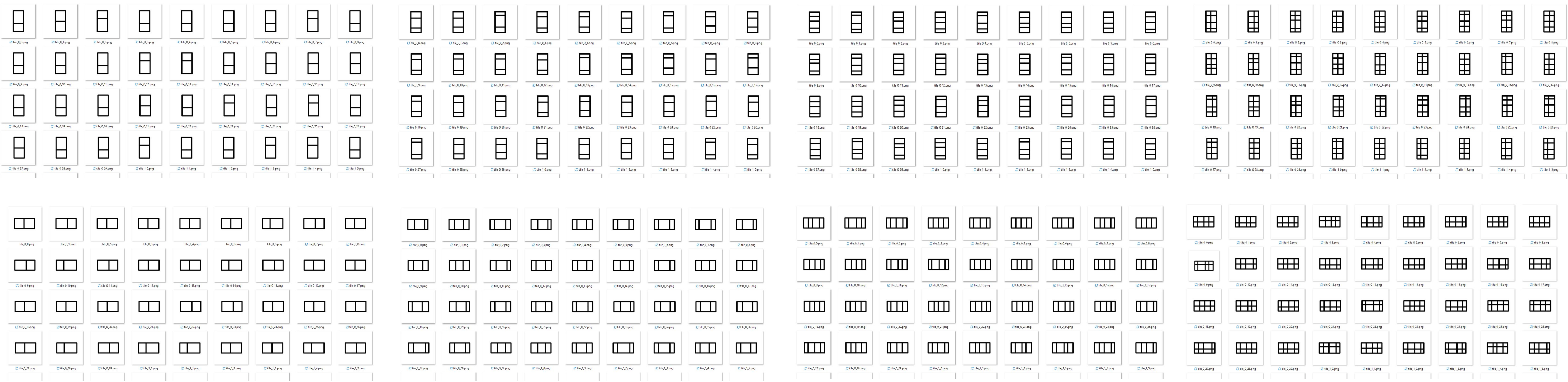


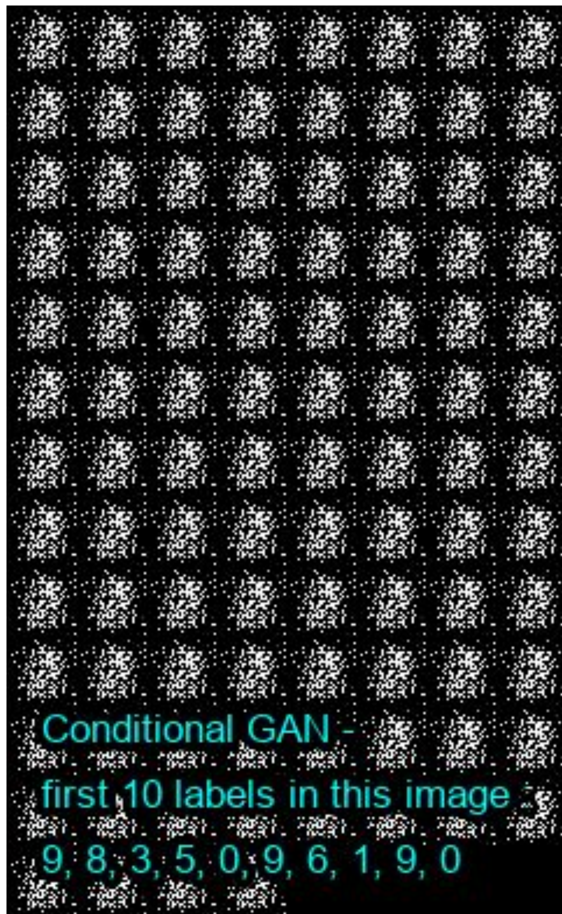
그림. 저장된 8종 창호 도식 데이터셋 (N=7200)



Conditional GAN study - MNIST

Epoch가 커질수록 generator의 성능이 좋아짐을 확인할 수 있었음

그림. Epoch에 따른 생성 이미지 모습



Epoch: 1

Epoch: 50

Epoch: 100

연구 방법

Part 1. Import tools

```
import os
import torch.nn as nn
import torch.utils.data
import torch.nn.functional as F
import torchvision
import numpy as np
from torchvision import transforms
from torchvision.utils import save_image
from PIL import Image, ImageFont, ImageDraw
import matplotlib.pyplot as plt
```

Part 2. Hyper-parameters & variables setting

```
num_epoch = 500 #시행착오
batch_size = 30
learning_rate = 0.0004
img_size_1d = 354
img_size = img_size_1d * img_size_1d

num_channel = 1
dir_name = "CGAN_results_window8"

noise_size = 100
hidden_size1 = 256
hidden_size2 = 512
hidden_size3 = 1024
```

Part 3. Other setting & discriminator

```
# Device setting
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print("Now using {} devices".format(device))

# Create a directory for saving samples
if not os.path.exists(dir_name):
  os.makedirs(dir_name)

# Define discriminator
class Discriminator(nn.Module):
  def __init__(self, condition_size):
    super(Discriminator, self).__init__()
    self.linear1 = nn.Linear(img_size + condition_size, hidden_size3)
    self.linear2 = nn.Linear(hidden_size3, hidden_size2)
    self.linear3 = nn.Linear(hidden_size2, hidden_size1)
    self.linear4 = nn.Linear(hidden_size1, 1)
    self.leaky_relu = nn.LeakyReLU(0.2)
    self.sigmoid = nn.Sigmoid()

  def forward(self, x):
    x = self.leaky_relu(self.linear1(x))
    x = self.leaky_relu(self.linear2(x))
    x = self.leaky_relu(self.linear3(x))
    x = self.linear4(x)
    x = self.sigmoid(x)
    return x
```

연구 방법

Part 4. Generator, transform and etc

```
# Define generator
class Generator(nn.Module):
   def __init__(self, condition_size):
      super(Generator, self).__init__()
      self.linear1 = nn.Linear(noise_size + condition_size, hidden_size1)
      self.linear2 = nn.Linear(hidden_size1, hidden_size2)
      self.linear3 = nn.Linear(hidden_size2, hidden_size3)
      self.linear4 = nn.Linear(hidden_size3, img_size)
      self.relu = nn.ReLU()
      self.tanh = nn.Tanh()

   def forward(self, x):
      x = self.relu(self.linear1(x))
      x = self.relu(self.linear2(x))
      x = self.relu(self.linear3(x))
      x = self.linear4(x)
      x = self.tanh(x)
      return x

# For checking CGAN's validity in final step #컨디션에 따른 이미지 생성
def check_condition(_generator, condition_size):
   test_image = torch.empty(0).to(device)
   for i in range(condition_size):
      test_label = torch.tensor(list(range(condition_size)))
      test_label_encoded = F.one_hot(test_label,
num_classes=condition_size).float().to(device)
      _z = torch.randn(condition_size, noise_size).to(device)
      _z_concat = torch.cat((_z, test_label_encoded), 1)
      test_image = torch.cat((test_image, _generator(_z_concat)), 0)
   _result = test_image.reshape(condition_size, 1, img_size_1d, img_size_1d)
   save_image(_result, os.path.join(dir_name, 'CGAN_test_result.png'), nrow=10)
```

```
# Dataset transform setting
transform = transforms.Compose([
   transforms.Resize((img_size_1d, img_size_1d)), # 이미지 크기 변환 추가
   transforms.Grayscale(num_output_channels=1),
   transforms.ToTensor(),
   transforms.Normalize(0.5, 0.5)])
# Custom dataset
dataset_dir = r'C:\Users\PC\Desktop\석사과정\수업\건축과
AI\finalfinal_project\dataset_240612_0144'
custom_dataset = torchvision.datasets.ImageFolder(root=dataset_dir,
transform=transform)

# Check number of classes in the dataset
num_classes = len(custom_dataset.classes)
if num_classes < 2:
   condition_size = 1
else:
   condition_size = num_classes

# Data loader
data_loader = torch.utils.data.DataLoader(dataset=custom_dataset,
                                          batch_size=batch_size,
                                          shuffle=True)

# Initialize generator/Discriminator
discriminator = Discriminator(condition_size)
generator = Generator(condition_size)

# Device setting
discriminator = discriminator.to(device)
generator = generator.to(device)

# Loss function & Optimizer setting
criterion = nn.BCELoss() # Binary Cross Entropy
d_optimizer = torch.optim.Adam(discriminator.parameters(), lr=learning_rate)
g_optimizer = torch.optim.Adam(generator.parameters(), lr=learning_rate)
```

Part 6. Train generator & discriminator

```
# Training part
for epoch in range(num_epoch):
  for i, (images, label) in enumerate(data_loader):

    # make ground truth (labels) -> 1 for real, 0 for fake
    real_label = torch.full((batch_size, 1), 1, dtype=torch.float32).to(device)
    fake_label = torch.full((batch_size, 1), 0, dtype=torch.float32).to(device)

    # reshape real images from dataset
    real_images = images.reshape(batch_size, -1).to(device)

    # Encode label's with 'one hot encoding'
    label_encoded = F.one_hot(label, num_classes=condition_size).float().to(device)
    # concat real images with 'label encoded vector'
    real_images_concat = torch.cat((real_images, label_encoded), 1)

    # train Generator #

    # Initialize grad
    g_optimizer.zero_grad()
    d_optimizer.zero_grad()

    # make fake images with generator & noise vector 'z'
    z = torch.randn(batch_size, noise_size).to(device)

    # concat noise vector z with encoded labels
    z_concat = torch.cat((z, label_encoded), 1)
    fake_images = generator(z_concat)
    fake_images_concat = torch.cat((fake_images, label_encoded), 1)

    # Compare result of discriminator with fake images & real labels
    # If generator deceives discriminator, g_loss will decrease
    g_loss = criterion(discriminator(fake_images_concat), real_label)

    # Train generator with backpropagation
    g_loss.backward()
    g_optimizer.step()
```

```
    # train Discriminator #

    # Initialize grad
    d_optimizer.zero_grad()
    g_optimizer.zero_grad()

    # make fake images with generator & noise vector 'z'
    z = torch.randn(batch_size, noise_size).to(device)

    # concat noise vector z with encoded labels
    z_concat = torch.cat((z, label_encoded), 1)
    fake_images = generator(z_concat)
    fake_images_concat = torch.cat((fake_images, label_encoded), 1)

    # Calculate fake & real loss with generated images above & real images
    fake_loss = criterion(discriminator(fake_images_concat), fake_label)
    real_loss = criterion(discriminator(real_images_concat), real_label)
    d_loss = (fake_loss + real_loss) / 2

    # Train discriminator with backpropagation
    d_loss.backward()
    d_optimizer.step()

    d_performance = discriminator(real_images_concat).mean()
    g_performance = discriminator(fake_images_concat).mean()

    if (i + 1) % 150 == 0:
      print("Epoch [ {}/{} ]  Step [ {}/{} ]  d_loss : {:.5f}  g_loss : {:.5f}"
          .format(epoch + 1, num_epoch, i+1, len(data_loader), d_loss.item(),
g_loss.item()))
```

연구 결과

Conditional GAN – 1 class

1. 초기에 이미지가 너무 어두워 transform 함수를 수정
2. 학습이 진행될수록 점점 선명해지지만 데이터셋의 무작위적 형태가 그대로 학습된 듯 한 모습을 보임

그림. 학습에 쓰인 창호 도식

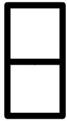
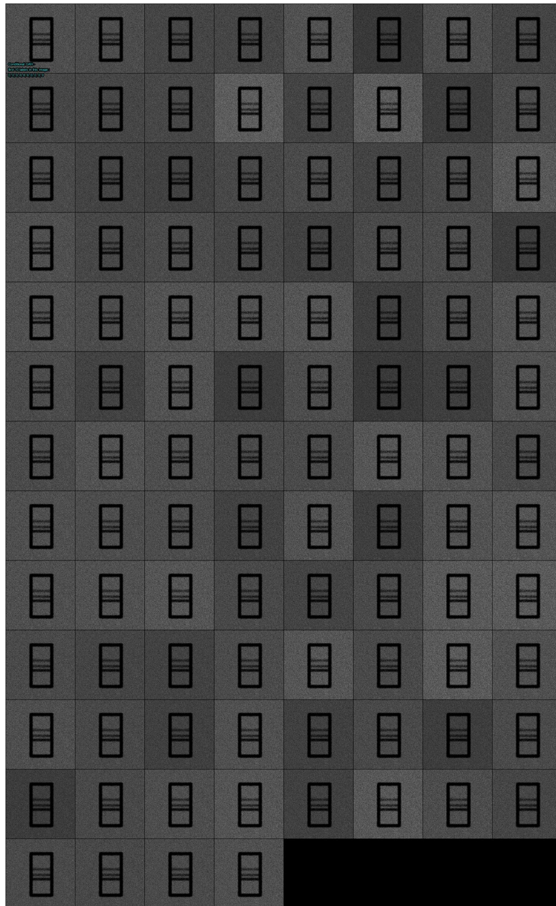
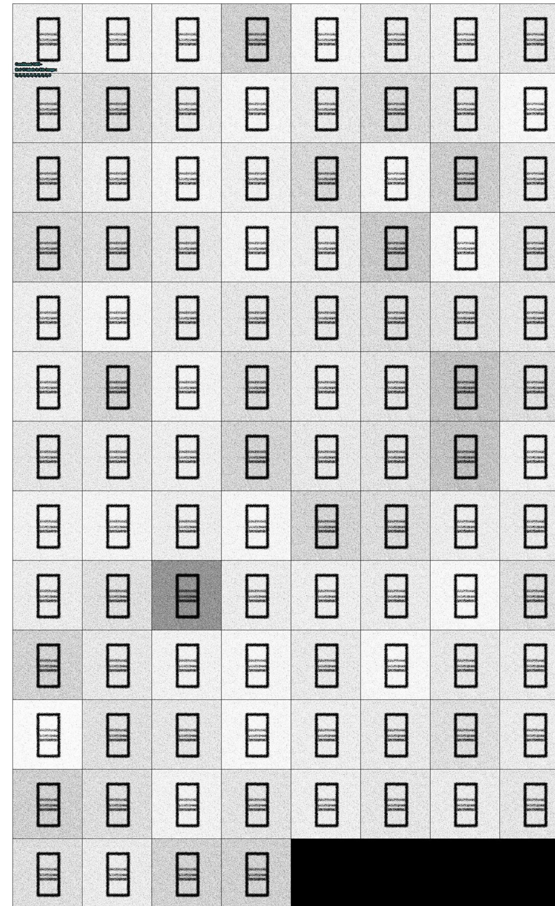


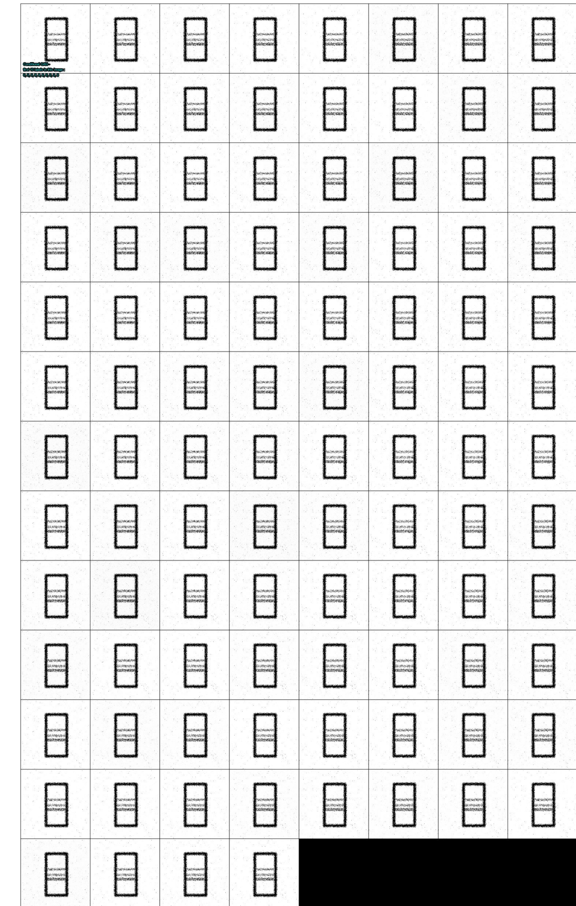
그림. Epoch에 따른 생성 이미지 모습



Epoch: 7



Epoch: 76



Epoch: 142

연구 결과

Conditional GAN – 8 class

1. 8개의 창 종류를 추가한 후 학습 시 30분-60분당 1 epoch 정도의 학습 속도
2. 현재로서는결과물을 파악하기에 어려움이 있음
3. 학습 속도가 느린 이유는 이미지 크기의 영향보다는 learning rate, hidden layer의 크기로 추정

그림. 학습에 쓰인 창호 도식

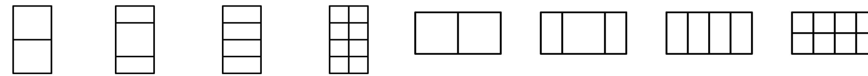
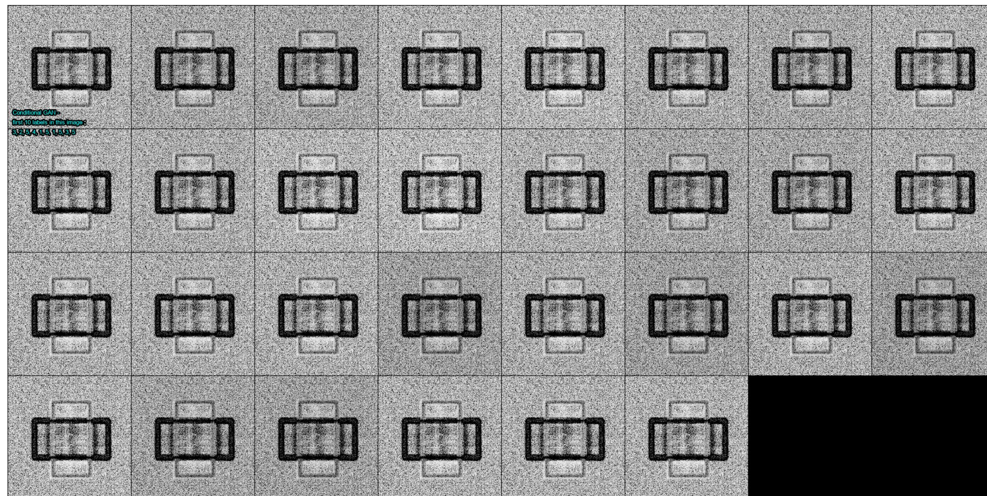
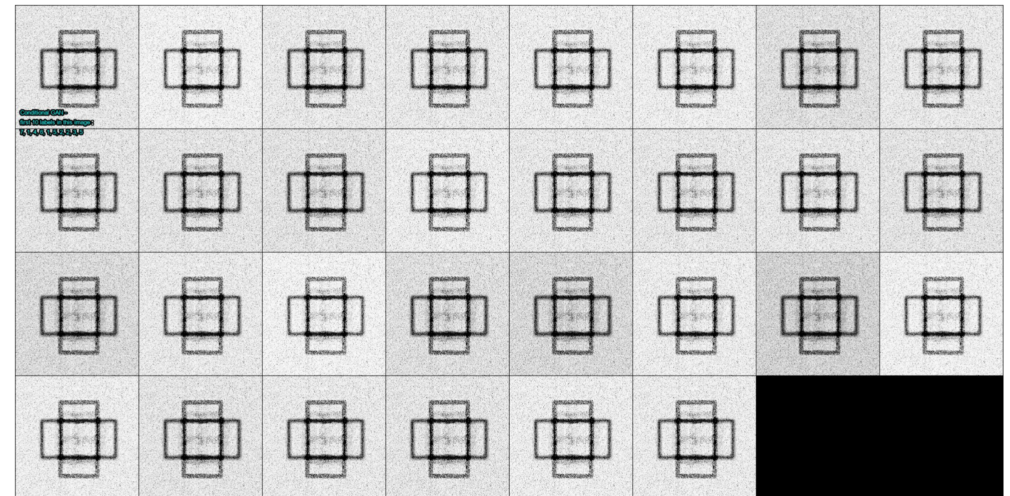


그림. Epoch에 따른 생성 이미지 모습



Epoch: 13



Epoch: 27

결론 및 시사점

연구의 시행착오 및 cGAN 모델 학습을 통해 도출한 결론 및 시사점

1. cGAN 학습을 위한 데이터셋 생성시 유의점 도출

창호의 하자와 같이 도형의 일부 형태가 무작위로 나타날 경우 모델이 데이터에서 **규칙성**을 찾아내어 이미지를 생성하지 못함
→ 학습에 도움이 되도록 창호의 프레임이 일관된 위상을 갖는 범위 내에서 무작위로 이미지를 생성

도형의 **크기**가 무작위로 나타날 경우 학습에 어려움이 생김
→ 창호 프레임에 의한 분절되는 창의 크기는 다양하게 하되 창호 외곽 치수는 동일하게 유지

2. 연구자가 생각하는 이미지 식별 및 생성에 적합한 데이터셋과 GAN 모델에 적합한 데이터셋의 차이의 인식 필요

학습된 데이터셋과 생성된 이미지의 차이 발생
→ 모델의 학습에는 연구자가 생각한 것보다 이미지의 특성을 판별하는 **더 세분화된 기준이 필요**

낮은 epoch에서는 주어진 조건에 따라 확연히 구분되는 이미지를 생성하지 못함
→ 학습 속도 향상 필요하며 **세분화된 클래스로 분류**된 데이터셋을 제작하여 학습했다면 학습 효율을 높일 수 있음

3. 모델의 학습 속도 향상 방법 필요

학습 속도가 느린 이유는 **learning rate, hidden layer의 크기**로 추정되므로 이를 조절하여 학습 속도 개선 가능

그림. 창호 도식 데이터셋 개선 예시

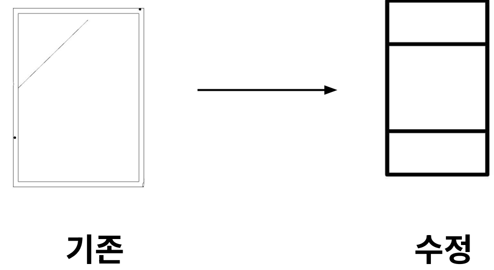
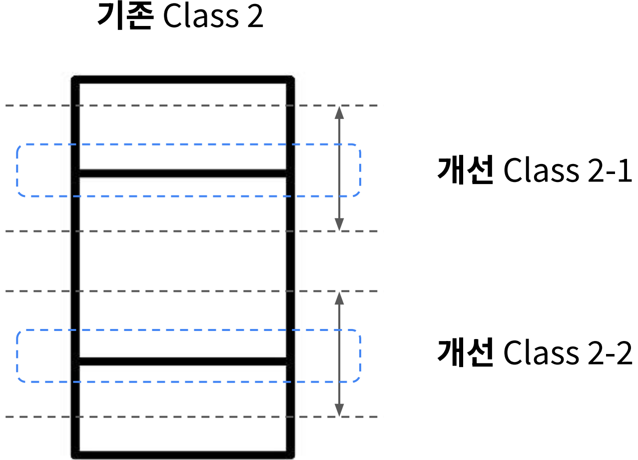


그림. 클래스 세분화 예시



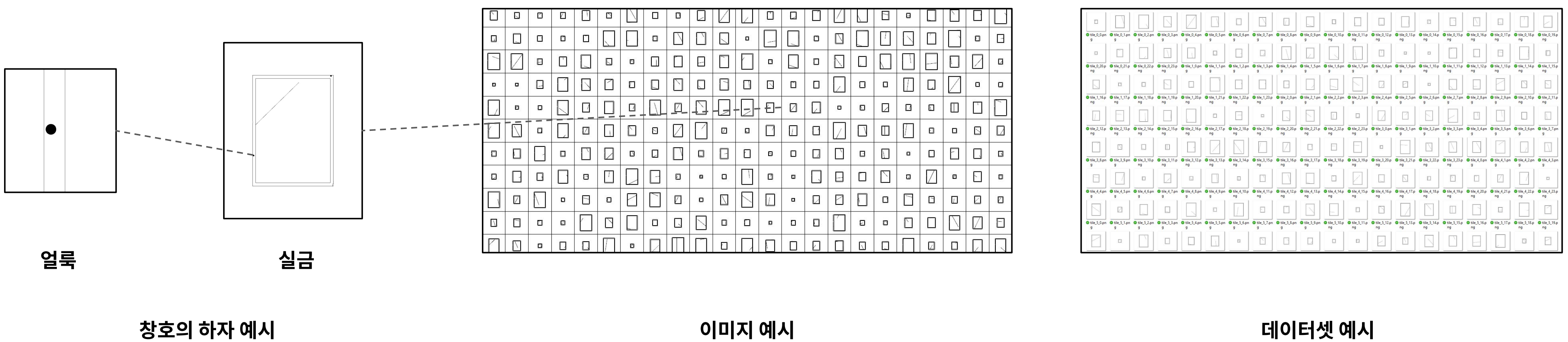
개선 방안 범위 내에서 무작위 변동하는 **프레임** 위치를 고려하여 변동 범위별 클래스 세분화

부록

모델 학습을 위한 데이터셋 준비 방법

1. 창호 하자를 표현하는 도식화 규칙 설정 및 스크립팅	라이노와 그래스호퍼 이용
2. 난수를 부여하여 창호 하자 이미지를 무작위로 생성	라이노와 그래스호퍼 이용
3. 생성된 이미지를 일정 크기로 분할하여 저장	파이썬과 라이브러리 Pillow 이용

초기 시도



창호의 하자 예시

이미지 예시

※창호의 크기를 모두 다르게 생성하였으나 모델 학습의
어려움으로 인해 **크기를 통일하여 진행**

데이터셋 예시

MNIST처럼 저용량의 이미지를 다량 생성