DCGAN Example
MNIST Generator

ARTIFICIAL
INTELLIGENCE
RESEARCH INSTITUTE

2017.10.17

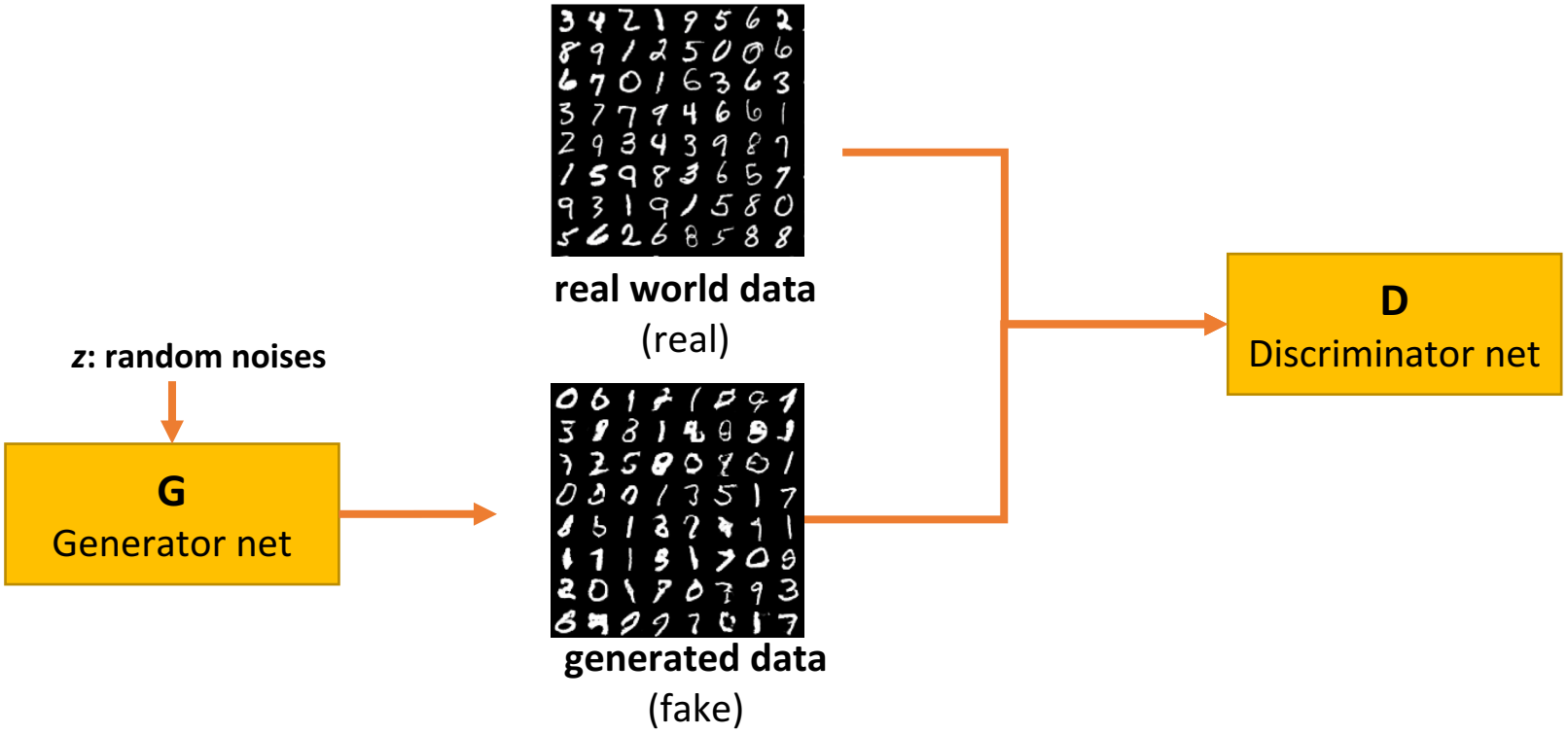
노형기 책임연구원
hyungkiroh@airi.kr

내용

- Introduction
- DCGAN 리뷰
- 구현 주요 사항
- 실습

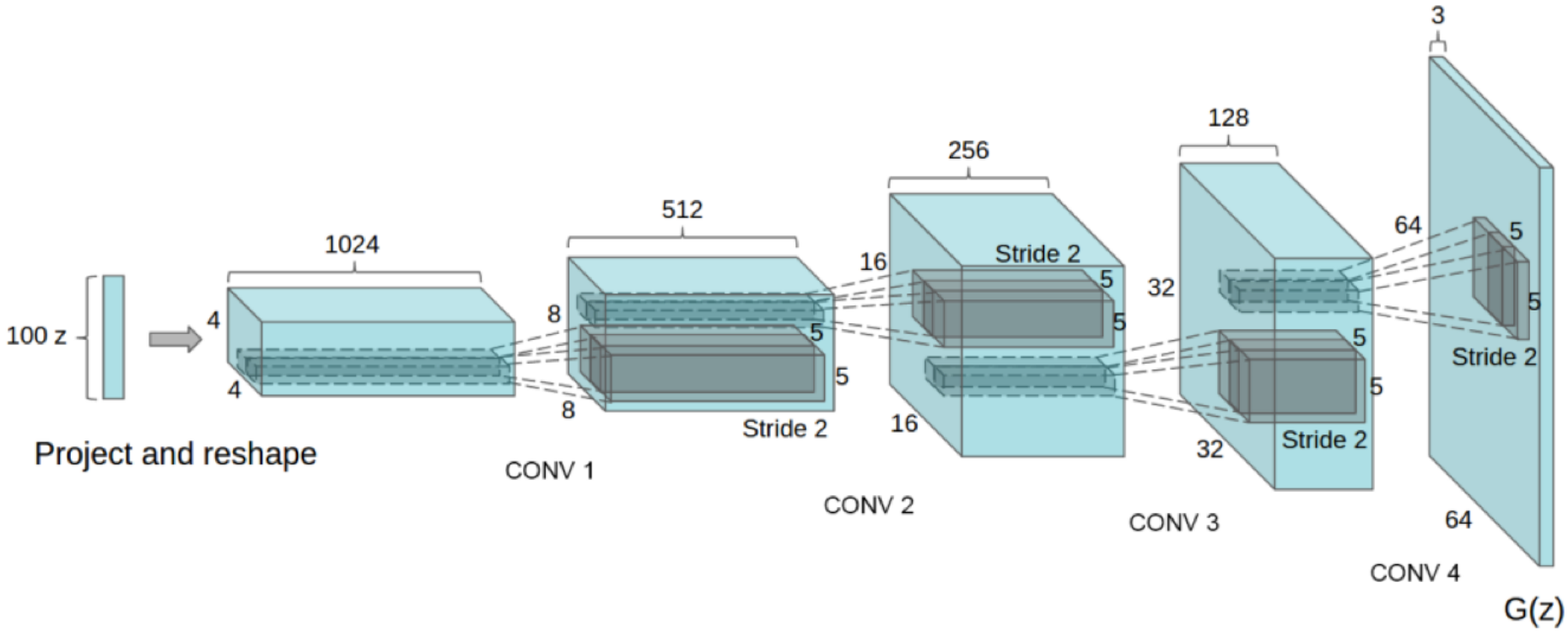
Introduction

- **GAN을 이용한 MNIST 데이터 생성**



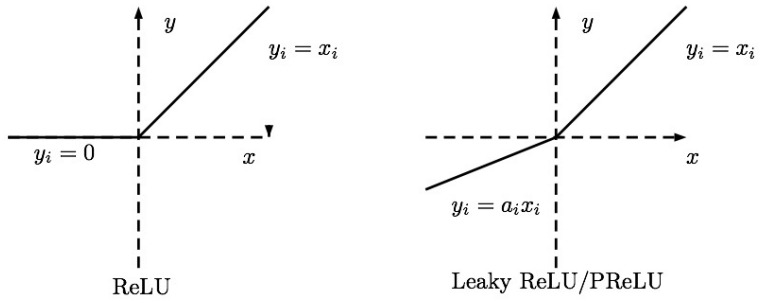
DCGAN 리뷰

- **DCGAN: Deep Convolutional GAN (A. Radford, 2016)**



DCGAN 리뷰

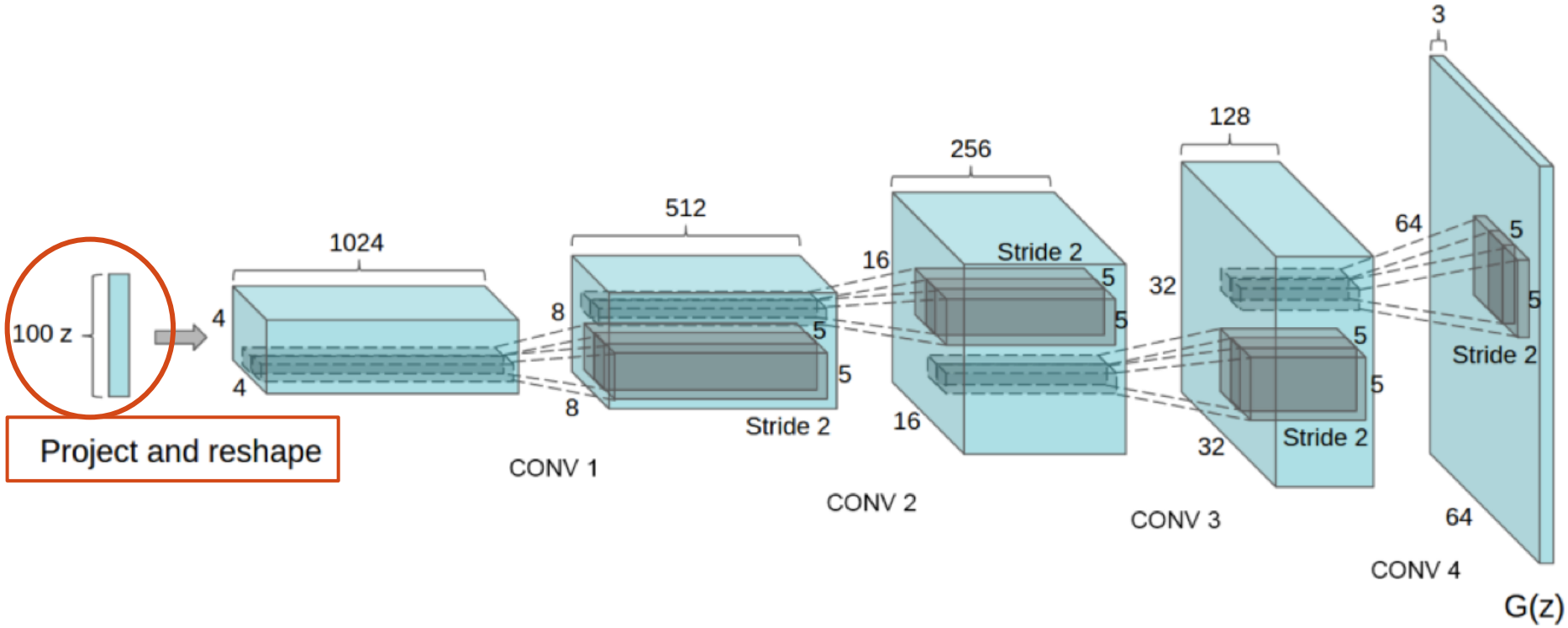
- **DCGAN**
  - 각 레이어에 Convolution layer를 사용
  - Pooling layer, Fully connected layer는 사용하지 않음
  - 레이어 계산 결과에 Batch Normalization 사용
  - Activation function으로 ReLU 대신 LeakyReLU 사용 (Discriminator)



  - Adam Optimizer 사용 (Adaptive moment estimation)

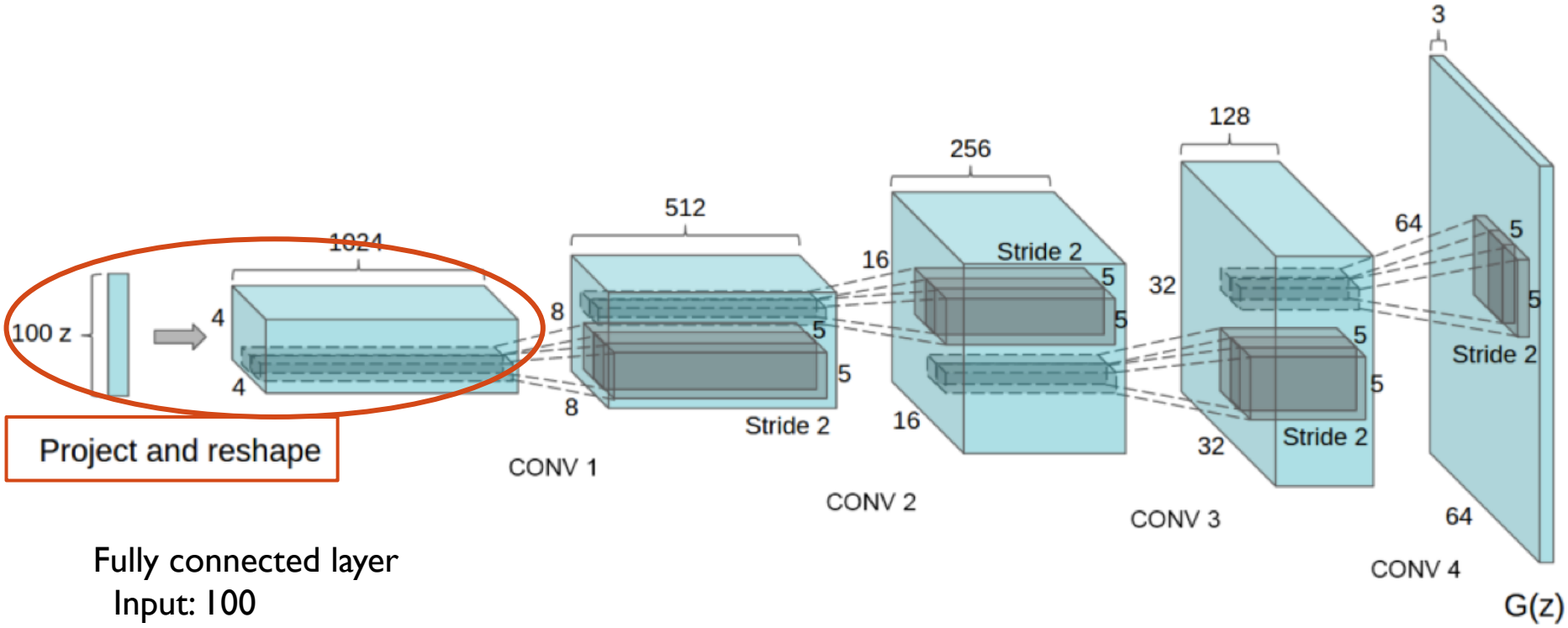
구현 주요 사항

- **랜덤 노이즈 z**
  - 생성할 이미지 1개당 100개의 랜덤 노이즈 대응
  - 100개의 랜덤 데이터에서 첫번째 필터맵 크기만큼 fully-connected layer 선언
  - 첫번째 필터맵의 차원에 맞게 reshape 수행 (예: 4x4x1024)



구현 주요 사항

- **Project and Reshape**

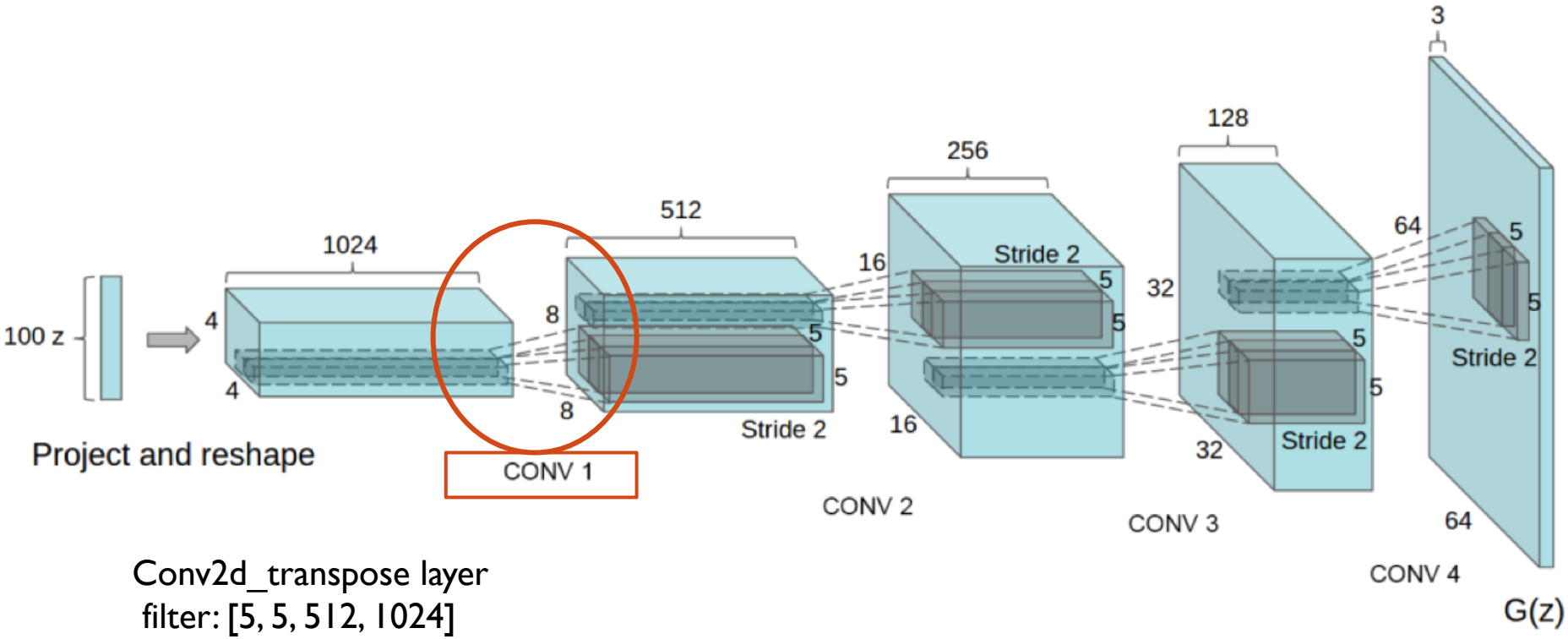


Fully connected layer
  Input: 100
  Output: 4x4x1024=16384
  Normalize: None
  Activation Function: None

Reshape 16384 output to [4, 4, 1024]

주요 구현 사항

- **Deconvolution Layer**

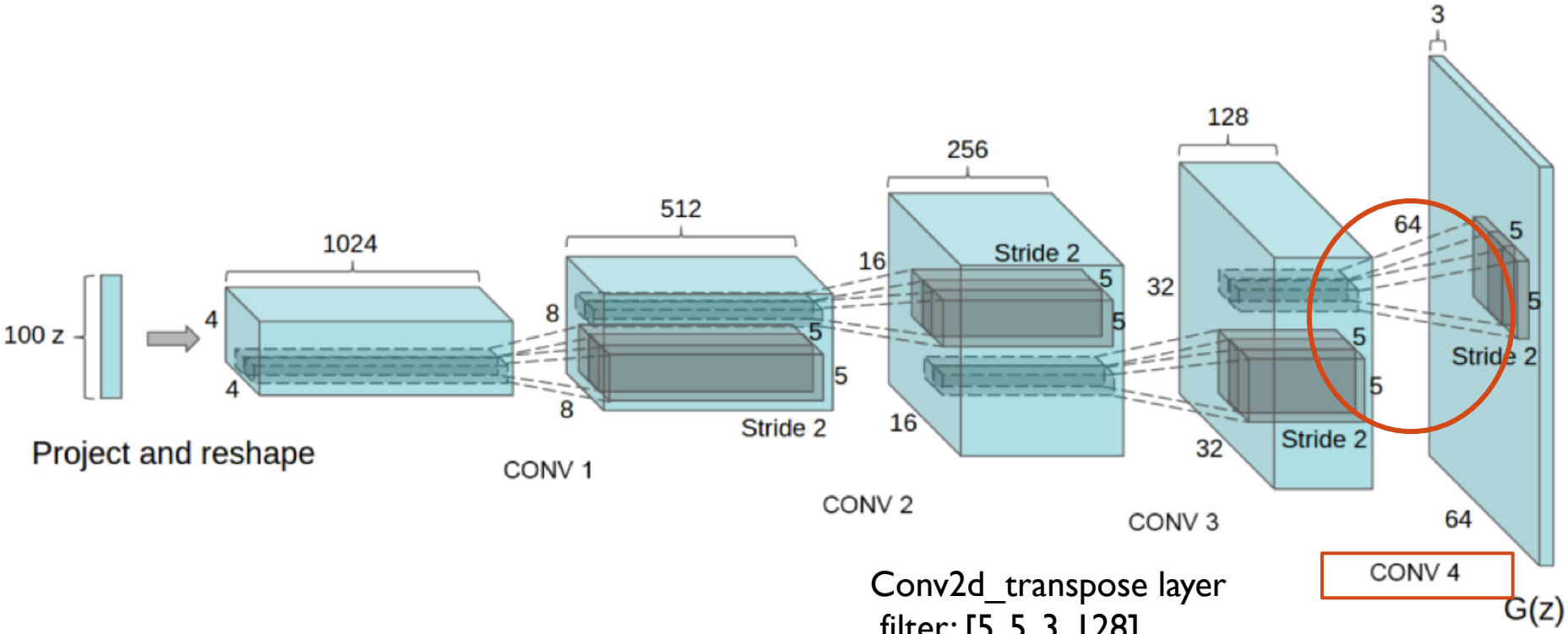


Conv2d_transpose layer
filter: [5, 5, 512, 1024]
output_shape: [batch_size, 8, 8, 512]
stride: [1, 2, 2, 1]
padding: SAME (default)

Normalize: BatchNorm
Activation: ReLU

주요 구현 사항

- **Image Generation**

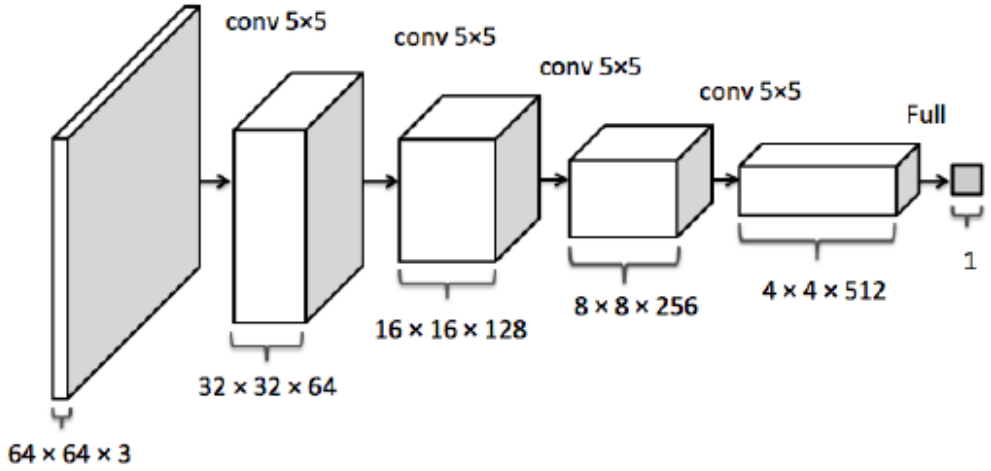


Conv2d_transpose layer
filter: [5, 5, 3, 128]
output_shape: [batch_size, 64, 64, 3]
stride: [1, 2, 2, 1]
padding: SAME (default)

Normalize: None
Activation: **tanh**

주요 구현 사항

- **Discriminator**
  - 일반적인 Convolution layer로 구현
  - Max pooling을 하지 않고 conv2d의 stride=2로 설정하여 크기를 줄임
  - 최종 레이어를 제외하고 Batch Normalize 사용
  - ReLU 대신 LeakyReLU를 사용
  - 최종 레이어는 직전 conv 레이어를 reshape한 뒤 fully connected로 1개 출력
  - 최종 레이어에서는 sigmoid를 사용하여 확률값(0~1)을 리턴



주요 구현 사항

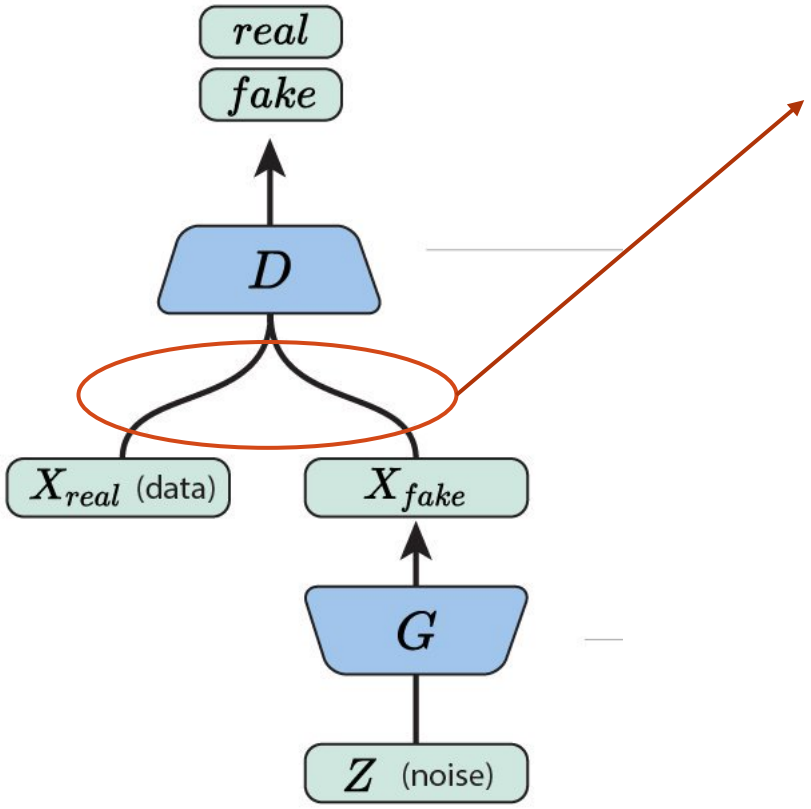
- 이미지 Normalize
  - 별도의 전처리는 하지 않음
  - Tanh 활성 함수와 맞추기 위해서 픽셀값은 [-1, 1] 사이의 값으로 정규화
- Leaky ReLU 구현
  - tf.maximum(x, 0.2 * x)
  - DCGAN 논문에서 권장하는 LeakyReLU의 leak값은 0.2로 되어 있음

주요 구현 사항

- **Discriminator Loss 함수**
  - Sigmoid_cross_entropy 사용
  - 주의: sigmoid_cross_entropy는 sigmoid를 내부적으로 포함하므로 반드시 sigmoid 하기 전의 결과 값을 넣어주어야 한다.
- **Discriminator 훈련**
  - Label은 real 데이터의 경우 1.0, generated 데이터의 경우 0.0을 준다. Label smoothing 기법이라고 하여 0.0, 1.0 대신에 0.1, 0.9를 주는 것이 효과적임
  - 미니배치 상으로 real 데이터만으로 구성하여 한번, generated 데이터만으로 구성하여 한번 이렇게 2회 훈련시킨다

주요 구현 사항

- **데이터 분류하여 넣어주기**



어떻게 구현할 것인가?
다음 방법에서 선택
1. tf.where 또는 tf.cond 이용
2. 임의 노드로 feed 하기

또 다른 구현 방법
D를 2개로 구현하고 weight 공유하기

주요 구현 사항

- **tf.where(condition, x, y)**

```
# where 예제
add_ab = tf.add(a, b)
square_b = tf.square(b)
result = tf.where(a < b, add_ab, square_b)
# a < b 가 참이면 add_ab가 연결되고 아니면 square_b가 연결된다
```

```
# 실제 사용예
# G 네트워크에서 생성한 이미지
generated_image = tf.tanh(net)

# 데이터파이프라인 또는 placeholder로 받은 real image
input_data = make_data_pipeline(. . .)

d_input = tf.where(is_real, input_data, generated_image)
# is_real은 tf.bool형 placeholder로 실제 데이터로 훈련시 True값을 입력한다.
```

주요 구현 사항

- **tf.cond(pred, true_fn, false_fn)**

```
# cond 예제
result = tf.cond(a < b, lambda: tf.add(a, b), lambda: tf.square(b))
# a < b 가 참이면 tf.add(a, b)가 수행되고 아니면 tf.square(b)가 실행된다
```

```
# 실제 사용예
# G 네트워크에서 생성한 이미지
generated_image = tf.tanh(net)

# 데이터파이프라인 또는 placeholder로 받은 real image
input_data = make_data_pipeline(. . .)

d_input = tf.cond(is_real, lambda: input_data, lambda: generated_image)
# is_real은 tf.bool형 placeholder로 실제 데이터로 훈련시 True값을 입력한다.
```

주요 구현 사항

- **임의 노드로 real 데이터를 feed하여 구현하기**

```
# 실제 사용 예
# G 네트워크에서 생성한 이미지
generated_image = tf.tanh(net)

d_input = generated_image

. . .

# training loop
sess.run([update, loss], feed_dict={d_input: real_image, . . .})
```

주요 구현 사항

- **훈련**
  - **G와 D는 연결되도록 구현하지만 훈련은 각기 따로시킴**
  - **특정 네트워크에 속한 변수들만 훈련되도록 Optimizer를 설정해야 함**

```
# 네트워크 정의시에 variable_scope로 각 네트워크 구별

with tf.variable_scope("Generator"):
  self.L0 = fully_connected(self.input_noise, 7 * 7 * 128, "linear0")
  self.L0 = tf.reshape(self.L0, [self.batch_size, 7, 7, 128])

# . . .
with tf.variable_scope("Discriminator"):
  self.inputD = self.generated
  self.L4 = conv2d(self.inputD, 64, name="conv4")
  self.L4 = batch_norm(self.L4, name="batch_norm4")
  self.L4 = lrelu(self.L4, name="lrelu4")

# . . .
```

주요 구현 사항

- **훈련**

```
# Optimizer 정의시에 해당 이름을 가진 변수만 지정함

t_vars = tf.trainable_variables()

# Discriminator 용 Optimzer
self.d_vars = [var for var in t_vars if 'Discriminator' in var.name]
self.d_optim = tf.train.AdamOptimizer(learning_rate=0.0002, beta1=0.5).
minimize(self.loss, var_list=self.d_vars)

# Generator 용 Optimzer
self.g_vars = [var for var in t_vars if 'Generator' in var.name]
self.g_optim = tf.train.AdamOptimizer(learning_rate=0.0002, beta1=0.5).
minimize(self.loss, var_list=self.g_vars)
```

참고:AdamOptimizer의 learning_rate와 beta1 값은 논문에 제시한 대로 설정해야 훈련이 용이하다

주요 구현 사항

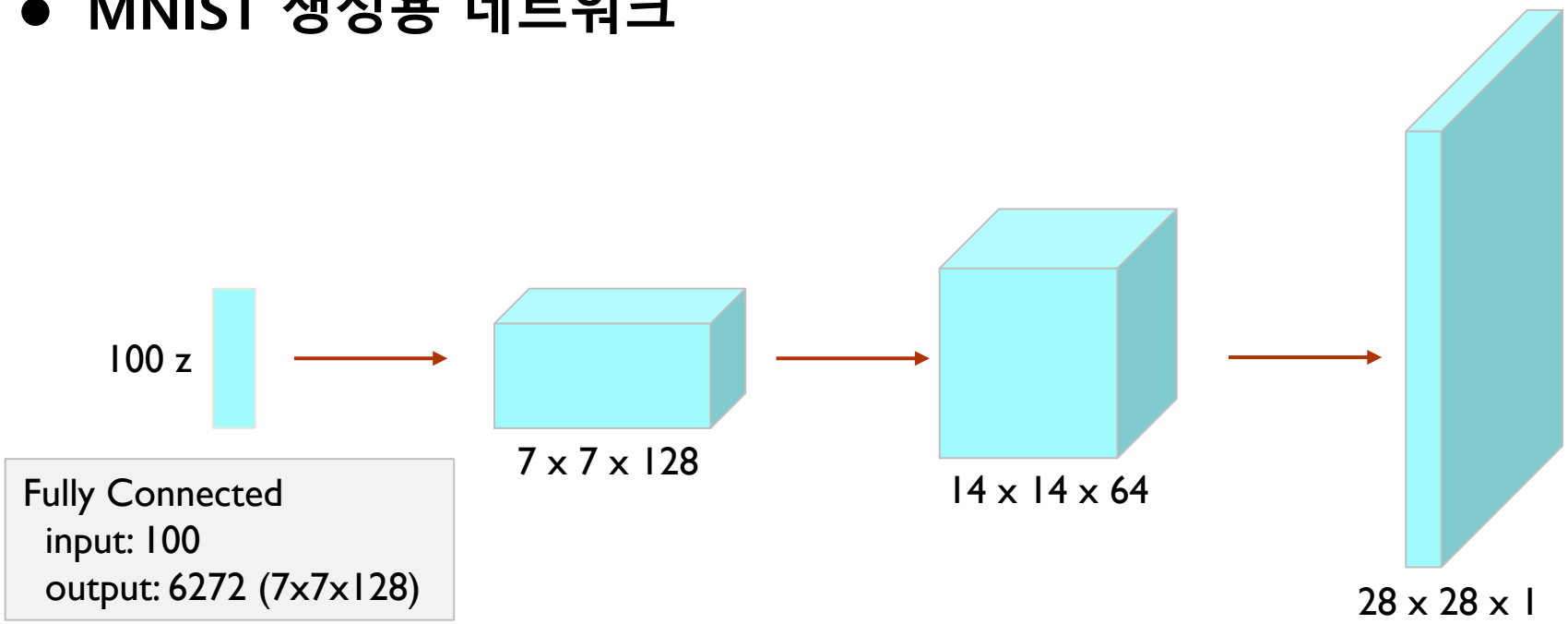
- **훈련**
  - D 훈련, G 훈련의 순서로 학습시킴
  - Real / fake를 나누어 미니배치 구성
  - G 네트워크 훈련은 D와 반대의 label를 줌
  - G 네트워크 훈련에 real 입력은 필요 없음 (대신 fake 2회 훈련)

```
_, d_loss1 = sess.run([dcgan.d_optim, dcgan.loss],
  feed_dict={dcgan.input_noise:z, dcgan.labelD: label_fake})
_, d_loss2 = sess.run([dcgan.d_optim, dcgan.loss],
  feed_dict={dcgan.inputD:batch_input, dcgan.labelD: label_real})

_, g_loss1 = sess.run([dcgan.g_optim, dcgan.loss],
  feed_dict={dcgan.input_noise:z, dcgan.labelD: label_real})
_, g_loss2 = sess.run([dcgan.g_optim, dcgan.loss],
  feed_dict={dcgan.input_noise:z, dcgan.labelD: label_real})
```

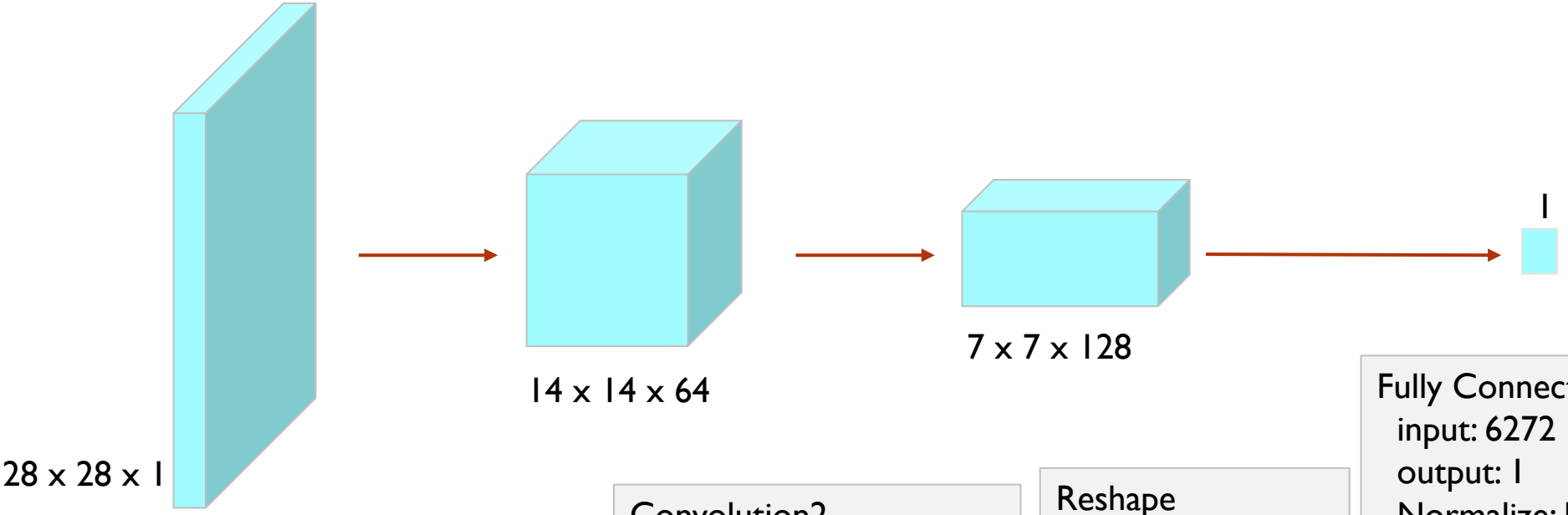
주요 구현 사항

- **MNIST 생성용 네트워크**



주요 구현 사항

- **MNIST 판별용 네트워크**



Convolution1
filter: [5, 5, 1, 64]
stride: [1, 2, 2, 1]
Normalize: BatchNorm
Activation: LReLU (0.2)

Convolution2
filter: [5, 5, 64, 128]
stride: [1, 2, 2, 1]
Normalize: BatchNorm
Activation: LReLU (0.2)

Reshape
input: [7, 7, 128]
output: 6272

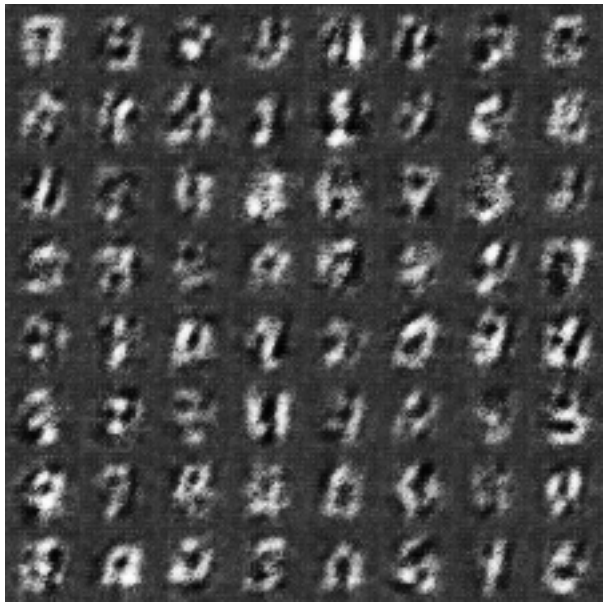
Fully Connected
input: 6272
output: 1
Normalize: None
Activation: sigmoid

주요 구현 사항

- **고려 사항**
  - 고품질 영상을 위해서 필터 개수를 128에서 더 높여본다
  - 고품질 영상을 위해서 convolution 레이어를 늘려본다. (단 해상도를 더 줄일 수 없으므로 추가되는 레이어는 stride=1로 설정해야 할 것이다)
- **유틸리티 파일**
  - image_util.py: 이미지 관련 유틸, 함수 중에 save_images를 사용하면 여러개의 이미지를 타일링 하여 하나의 이미지로 저장할 수 있다

```
save_images('output.jpg', output_images[0:64], [8, 8])
```

구현예시



Fist minibatch

Epoch 5

구현예시

Epoch 50

Epoch 200

실습

- **Do It Yourself**

참고 자료

- **DCGAN 논문**
  - **Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks (A. Radford et al, 2016) arXiv:1511.06434**
- **DCGAN 코드**
  - https://github.com/carpedm20/DCGAN-tensorflow
  - https://github.com/openai/improved-gan
- **기타 참고자료**
  - https://github.com/soumith/ganhacks