GAN with Pokemon

2017010698
수학과 오서영

Classification vs GAN

일반적인 classification

: 대상을 판별하는 판별망 (Discriminative Network)
-> 입력데이터 x에 대하여 y가 될 조건부 확률을 구하는 것

(**ex**) 입력 데이터 : 고양이사진
-> 예측 : "고양이"

GAN (Generative Adversarial Networks)

: 특정 확률분포를 갖는 데이터로 학습을 시키면,
이것과 유사한 분포를 갖는 데이터를 생성하는 것

-> label이 없더라도 스스로 데이터 속 **중요한 정보**를 찾아
새로운 데이터를 만들어냄

GAN 이란?

생성망 (Generator)

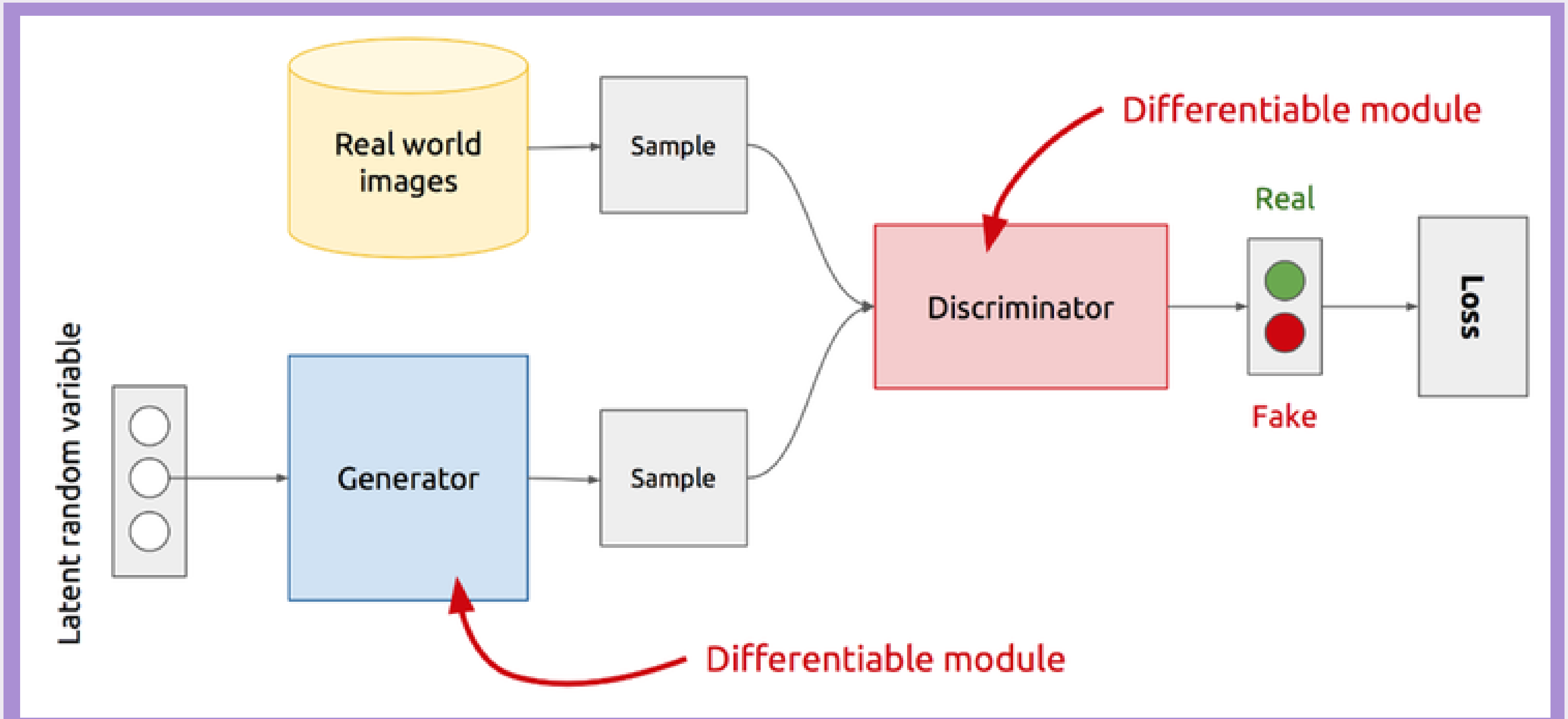
: **Discriminator**를 속일 수 있을 정도로 진짜 데이터와 비슷한 가짜 데이터를 만들어 냄

판별망 (Discriminator)

: 실제 학습 데이터와 **Generator**를 거쳐 만들어진 가짜 데이터를 이용하여 학습
-> 데이터가 진짜인지 가짜인지 구별하는 역할

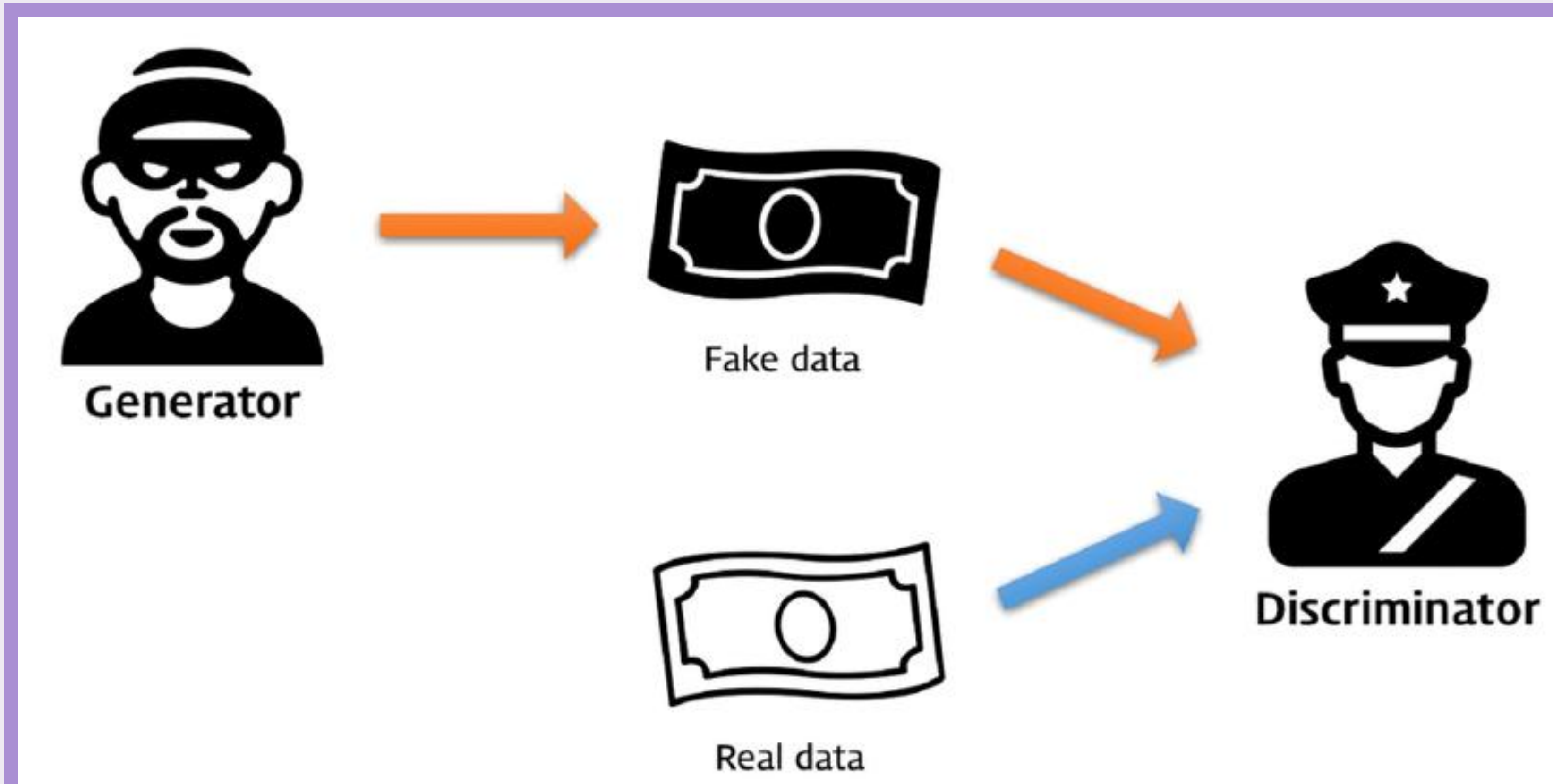
Discriminator는 판별을 잘하는 방향으로,
Generator는 Discriminator를 잘 속이는 방향으로 학습됨

GAN 이란?



Real world images
Sample
Differentiable module
Real
Discriminator
Loss
Fake
Latent random variable
Generator
Sample
Differentiable module

GAN 이란?



위조 지폐를 만들어 내는 사기꾼과
위조 지폐 여부를 판별하는 형사 ?

GAN 학습

1. Generator 고정 -> Discriminator 학습
2. Discriminator 고정 -> Generator 학습
위 과정을 반복하면 평형 상태에 도달

상호 협조적이지 않는 상대가
서로 최적에 도달하려고 노력 -> **내쉬 평형** 상태

어떤 기준으로 최적에 도달했는지
명확하게 판단할 근거가 부족
-> **사람의 개입**이 필요

Data Exploration

Import Packages

```
import numpy as np
import matplotlib.pyplot as plt
import os
from PIL import Image
import imageio

import tensorflow as tf
from keras import layers
from keras.datasets import mnist
from keras.models import Sequential, Model, load_model
from keras.optimizers import Adam
```

Data Exploration

Make Dataset

```
def load_and_preprocessing(dir):
    data = []
    img_list = os.listdir(dir)
    for name in img_list :

        png = imageio.imread(dir + name)
        png = Image.fromarray(png)
        png.load()  # for splitting

        # convert RGBA to RGB -> alpha channel
        if(len(png.split()) == 4):
            img = Image.new('RGB', png.size, (255, 255, 255)) # white
            img.paste(png, mask = png.split()[3])
        else:
            img = png

        images = tf.keras.preprocessing.image.img_to_array(img)
        images /= 255.  # preprocessing

        data.append(images)

    return np.stack(data)
```

Data Exploration

Make Dataset

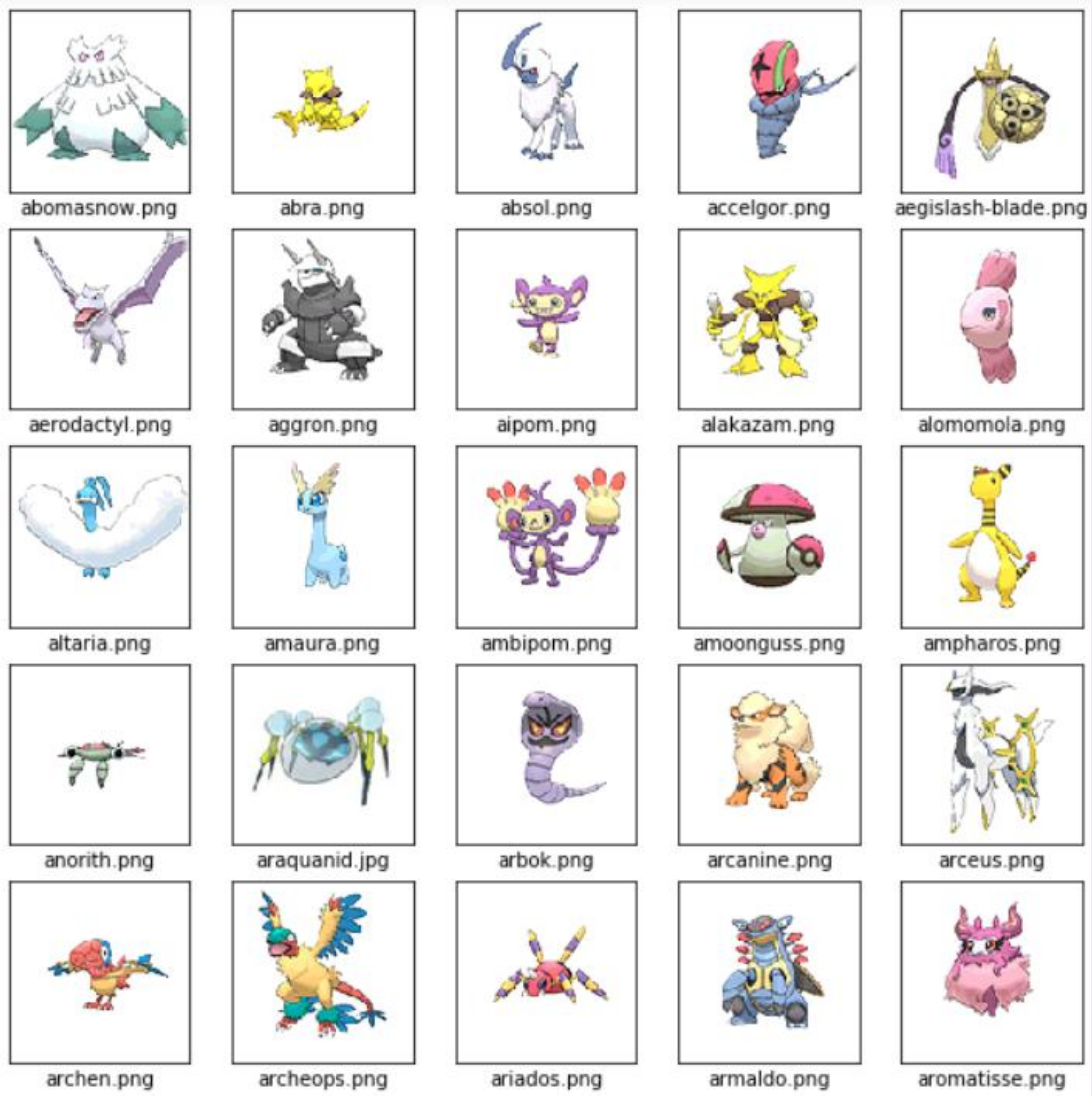
```
dir = "images/"
img_list = os.listdir(dir)
img_len = len(os.listdir(dir))

print("The number of images :",img_len)
print(img_list[0:10])
```

```
The number of images : 809
['abomasnow.png', 'abra.png', 'absol.png', 'accelgor.png',
m.png', 'alomomola.png']
```

```
# Make dataset
dataset = load_and_preprocessing(dir)
print("Shape of dataset :", dataset.shape)
```

```
Shape of dataset : (809, 120, 120, 3)
```

Data Exploration



Create Generator

Generator

```
# params
latent_dim = 100
height = 120
width = 120
channels = 3
```

```
generator_input = layers.Input(shape=(latent_dim,))
g = layers.Dense(128 * 15 * 15)(generator_input)
g = layers.Reshape((15, 15, 128))(g)

g = layers.Conv2DTranspose(128, 3, strides=2, padding='same')(g)
g = layers.BatchNormalization(momentum=0.8)(g)
g = layers.ReLU()(g)

g = layers.Conv2DTranspose(128, 3, strides=2, padding='same')(g)
g = layers.BatchNormalization(momentum=0.8)(g)
g = layers.ReLU()(g)

g = layers.Conv2DTranspose(64, 3, strides=2, padding='same')(g)
g = layers.BatchNormalization(momentum=0.8)(g)
g = layers.ReLU()(g)

g = layers.Conv2D(channels, 3, activation='tanh', padding='same')(g)

generator = Model(generator_input, g)
generator.summary()
```

Create Generator

```
Model: "model_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_26 (InputLayer)        (None, 100)               0
_________________________________________________________________
dense_18 (Dense)             (None, 28800)             2908800
_________________________________________________________________
reshape_12 (Reshape)         (None, 15, 15, 128)       0
_________________________________________________________________
conv2d_transpose_24 (Conv2DT (None, 30, 30, 128)       147584
_________________________________________________________________
batch_normalization_39 (Batc (None, 30, 30, 128)       512
_________________________________________________________________
re_lu_21 (ReLU)              (None, 30, 30, 128)       0
_________________________________________________________________
conv2d_transpose_25 (Conv2DT (None, 60, 60, 128)       147584
_________________________________________________________________
batch_normalization_40 (Batc (None, 60, 60, 128)       512
_________________________________________________________________
re_lu_22 (ReLU)              (None, 60, 60, 128)       0
_________________________________________________________________
conv2d_transpose_26 (Conv2DT (None, 120, 120, 64)      73792
_________________________________________________________________
batch_normalization_41 (Batc (None, 120, 120, 64)      256
_________________________________________________________________
re_lu_23 (ReLU)              (None, 120, 120, 64)      0
_________________________________________________________________
conv2d_50 (Conv2D)           (None, 120, 120, 3)       1731
=================================================================
Total params: 3,280,771
Trainable params: 3,280,131
Non-trainable params: 640
```

Create Discriminator

Discriminator

```
discriminator_input = layers.Input(shape=(height, width, channels))

d = layers.Conv2D(128, 3, strides=2, padding='same')(discriminator_input)
d = layers.LeakyReLU(alpha=0.2)(d)

d = layers.Conv2D(128, 3, strides=2, padding='same')(d)
d = layers.BatchNormalization(momentum=0.8)(d)
d = layers.LeakyReLU(alpha=0.2)(d)

d = layers.Conv2D(64, 3, strides=2, padding='same')(d)
d = layers.BatchNormalization(momentum=0.8)(d)
d = layers.LeakyReLU()(d)

d = layers.Conv2D(64, 3, strides=2, padding='same')(d)
d = layers.BatchNormalization(momentum=0.8)(d)
d = layers.LeakyReLU()(d)

d = layers.Flatten()(d)
d = layers.Dense(1, activation='sigmoid')(d)

discriminator = Model(discriminator_input, d)
discriminator_optimizer = Adam(lr=0.0002, beta_1=0.5, clipvalue=1.0)
discriminator.compile(optimizer=discriminator_optimizer, loss='binary_crossentropy', metrics=['accuracy'])
discriminator.summary()
```

Create Discriminator

```
Model: "model_26"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_31 (InputLayer)        (None, 120, 120, 3)       0
_________________________________________________________________
conv2d_59 (Conv2D)           (None, 60, 60, 128)       3584
_________________________________________________________________
leaky_re_lu_33 (LeakyReLU)   (None, 60, 60, 128)       0
_________________________________________________________________
conv2d_60 (Conv2D)           (None, 30, 30, 128)       147584
_________________________________________________________________
batch_normalization_48 (Batc (None, 30, 30, 128)       512
_________________________________________________________________
leaky_re_lu_34 (LeakyReLU)   (None, 30, 30, 128)       0
_________________________________________________________________
conv2d_61 (Conv2D)           (None, 15, 15, 64)        73792
_________________________________________________________________
batch_normalization_49 (Batc (None, 15, 15, 64)        256
_________________________________________________________________
leaky_re_lu_35 (LeakyReLU)   (None, 15, 15, 64)        0
_________________________________________________________________
conv2d_62 (Conv2D)           (None, 8, 8, 64)          36928
_________________________________________________________________
batch_normalization_50 (Batc (None, 8, 8, 64)          256
_________________________________________________________________
leaky_re_lu_36 (LeakyReLU)   (None, 8, 8, 64)          0
_________________________________________________________________
flatten_9 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_21 (Dense)             (None, 1)                 4097
=================================================================
Total params: 267,009
Trainable params: 266,497
Non-trainable params: 512
```

GAN - Training

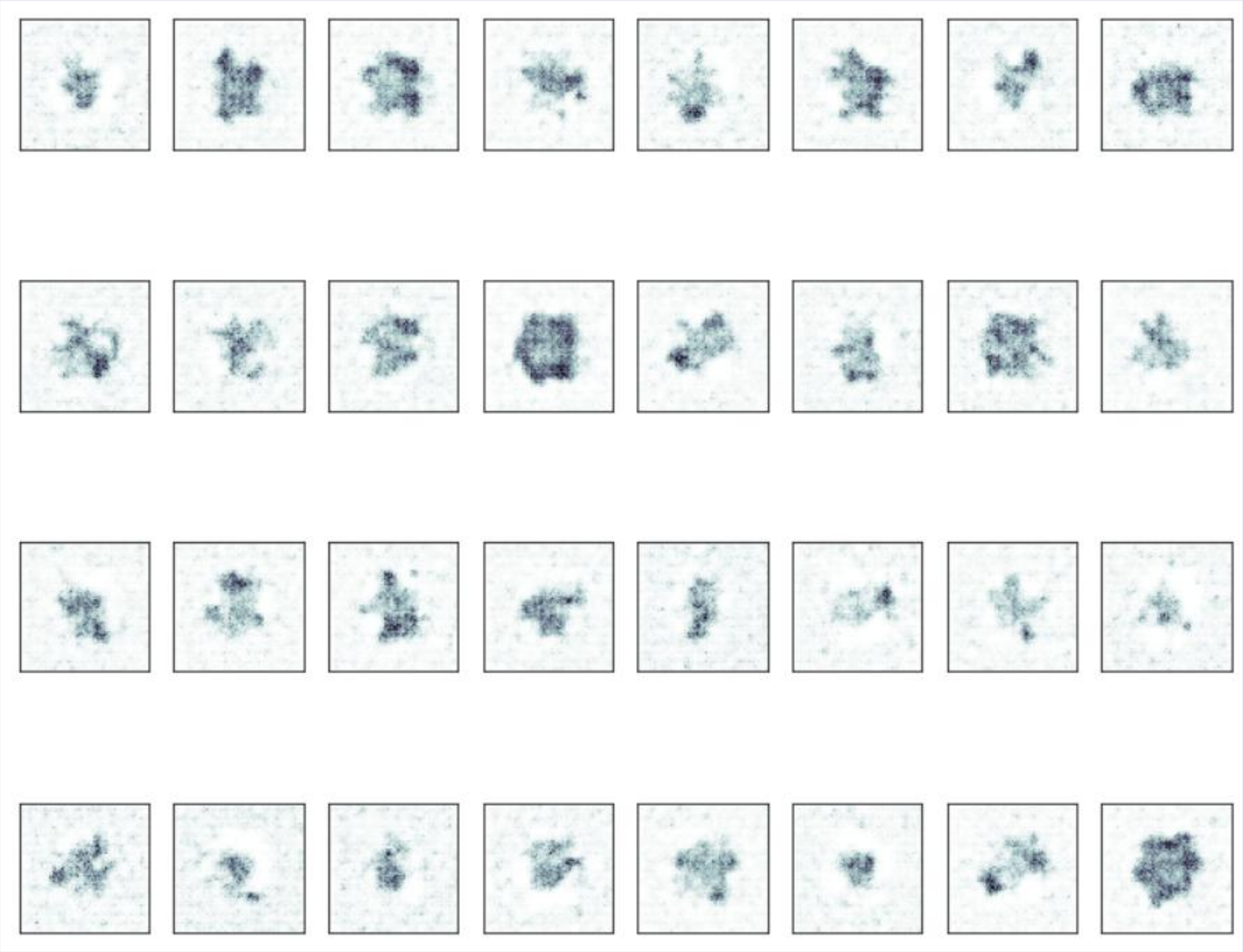
GAN

```
gan_input = layers.Input(shape=(latent_dim,))
discriminator.trainable = False

gan_output = discriminator(generator(gan_input))
gan = Model(gan_input, gan_output)

gan_optimizer = Adam(lr=0.0002, beta_1=0.5, clipvalue=1.0)
gan.compile(optimizer = gan_optimizer, loss = 'binary_crossentropy', metrics=['accuracy'])
```

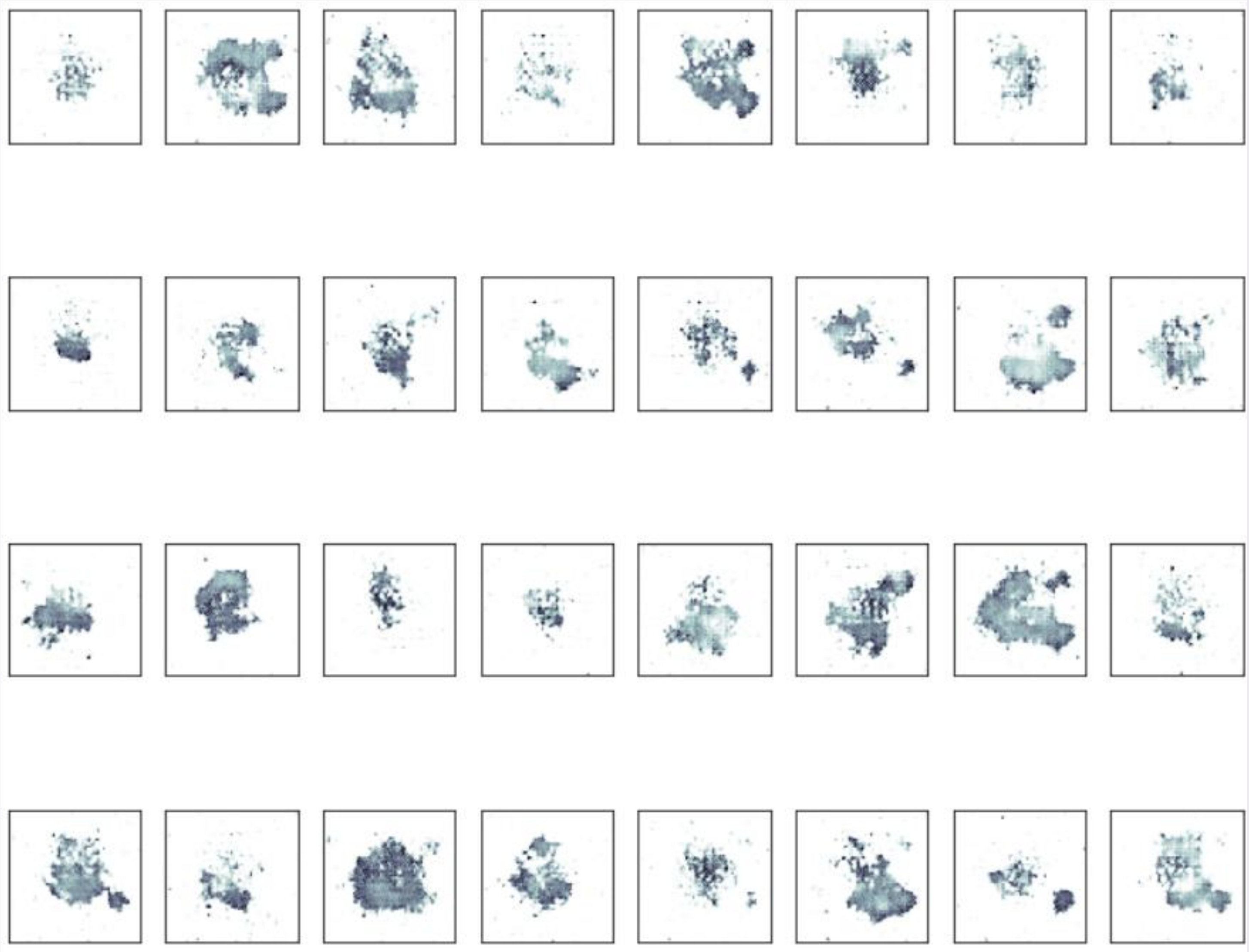
```
%%time
hist_1000 = train(1000, 1)
```

```
56 iteration - discriminator loss: 5.479, generator loss: 4.935
57 iteration - discriminator loss: 1.801, generator loss: 1.393
58 iteration - discriminator loss: 0.806, generator loss: 2.114
59 iteration - discriminator loss: 0.177, generator loss: 3.102
60 iteration - discriminator loss: 0.162, generator loss: 2.834
61 iteration - discriminator loss: 0.224, generator loss: 2.583
62 iteration - discriminator loss: 0.167, generator loss: 2.694
63 iteration - discriminator loss: 0.148, generator loss: 2.271
64 iteration - discriminator loss: 0.294, generator loss: 4.602
65 iteration - discriminator loss: 1.343, generator loss: 6.806
66 iteration - discriminator loss: 1.140, generator loss: 1.308
67 iteration - discriminator loss: 0.031, generator loss: 2.293
68 iteration - discriminator loss: 0.117, generator loss: 3.423
```

Visualization – 100 iterations



Visualization – 750 iterations



Visualization – 1500 iterations

Dataset

[1] Pokemon Image Dataset,
https://www.kaggle.com/vishalsubbiah/pokemon-images-and-types

Reference

[1] 라온피플 ML Academy,
https://laonple.blog.me/221190581073