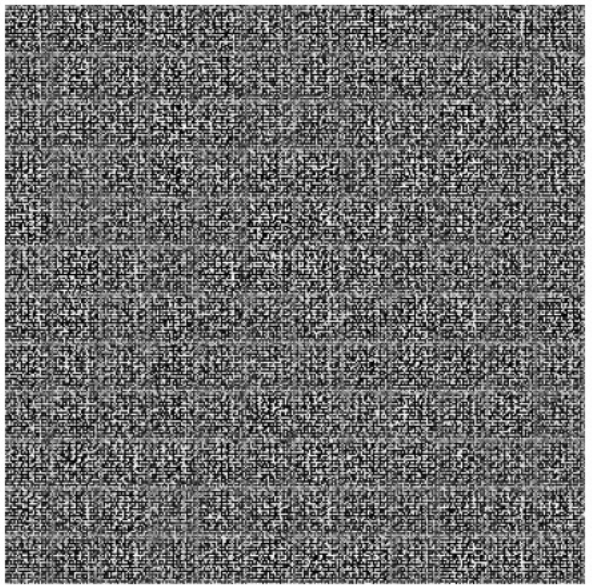
Generative Adversarial Network



CS-EJ3311 - Deep Learning with Python
24.10.-11.12.2022
Aalto University & FiTech.io

5.12.2022 Shamsi Abdurakhmanova

Task A (source)

Domain $\mathcal{D}_s = \{\mathcal{X}_s, \mathrm{P}(\mathcal{X}_s)\}$

Task $\mathcal{T}_s = \{\mathcal{Y}_s, P(\mathcal{Y}_s|\mathcal{X}_s)\}$

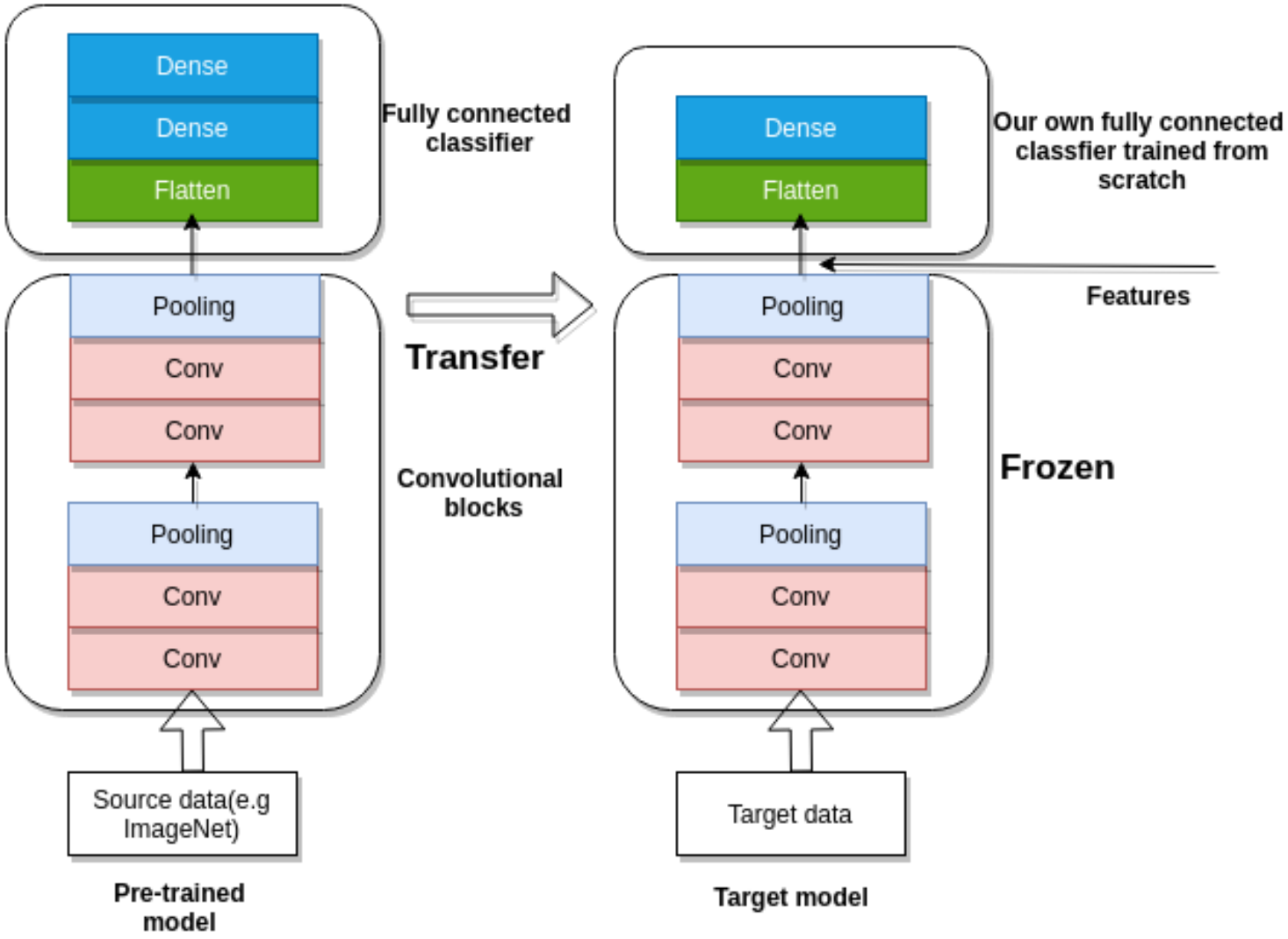
Task B (target)

Domain $\mathcal{D}_T = \{\mathcal{X}_\mathrm{T}, \mathrm{P}(\mathcal{X}_T)\}$

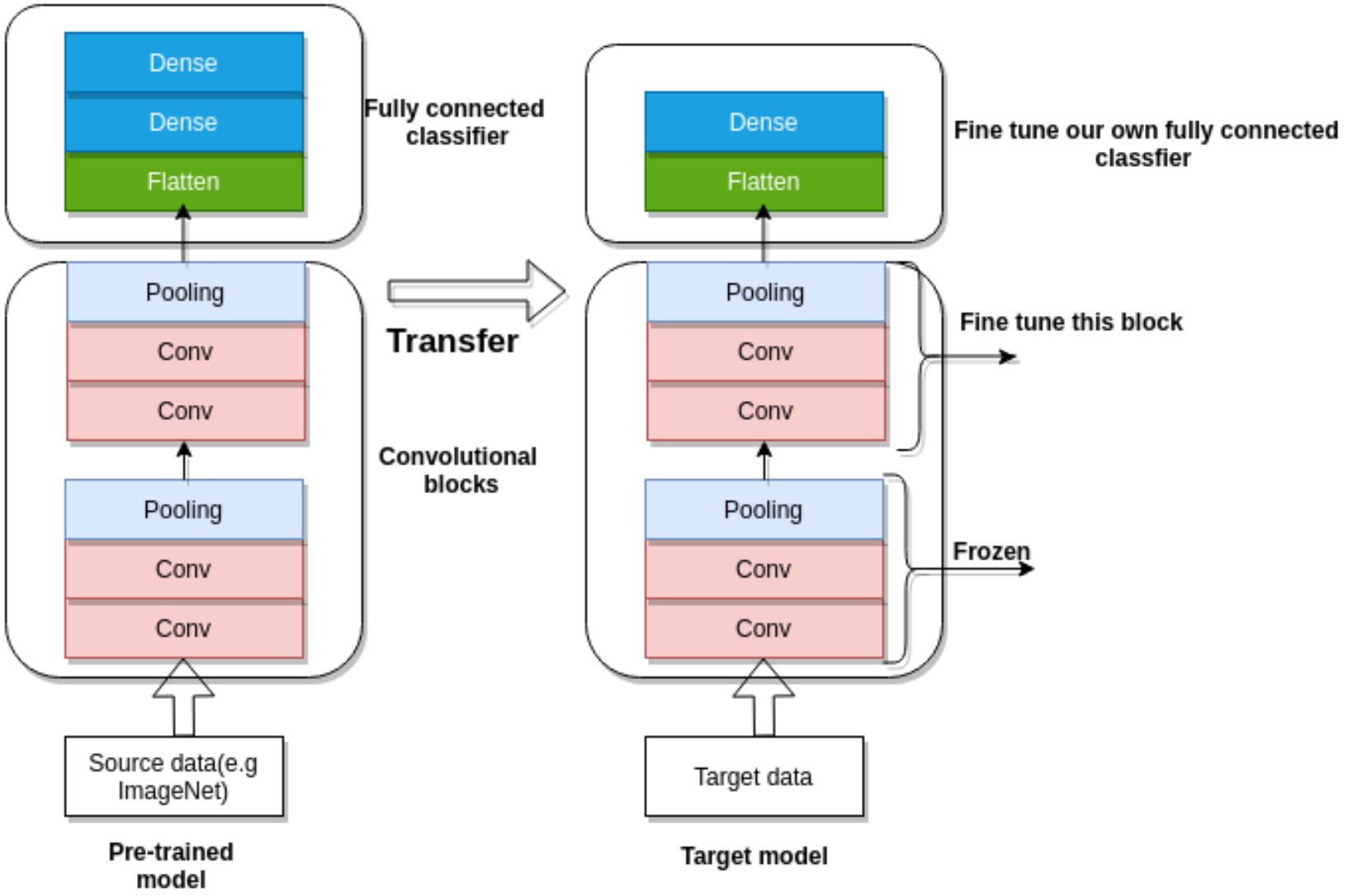
Task $\mathcal{T}_T = \{\mathcal{Y}_T, P(\mathcal{Y}_T|\mathcal{X}_T)\}$

Transfer learning – learn $P(\mathcal{Y}_T|\mathcal{X}_T)$ using knowledge $\mathcal{D}_s, \mathcal{T}_s$

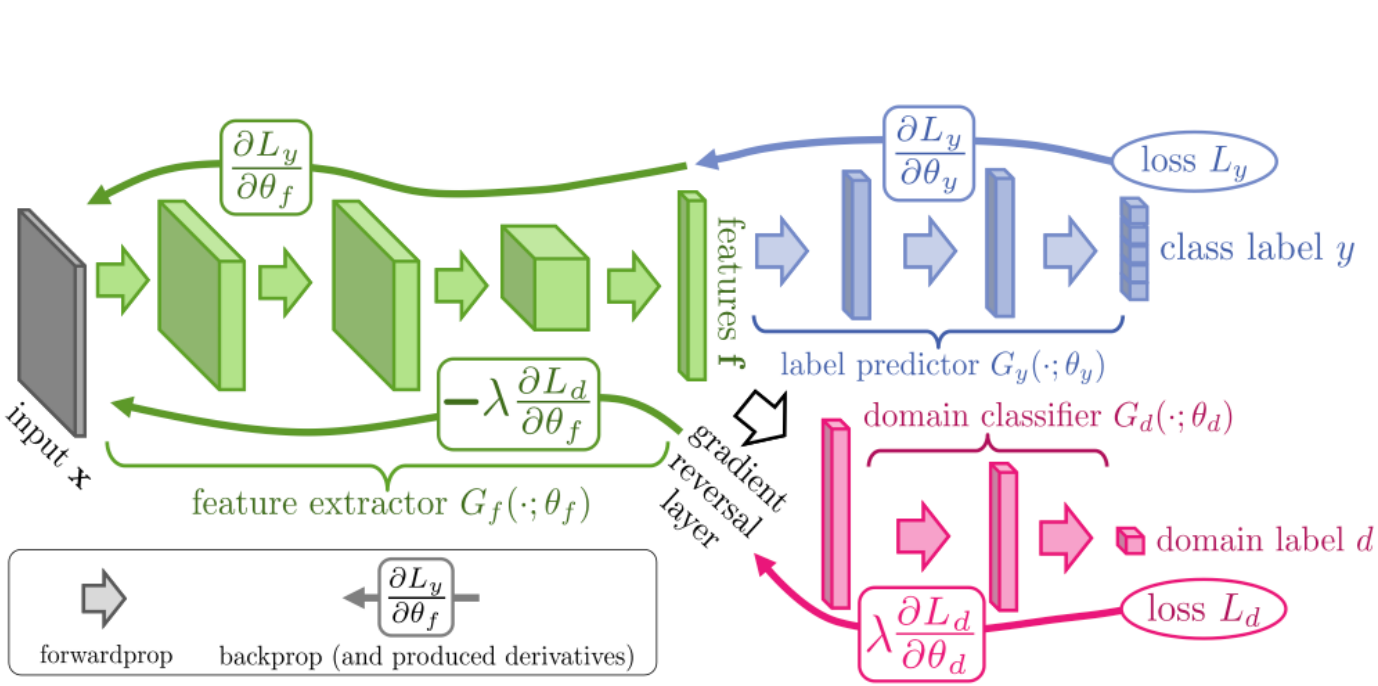
Pre-trained model for feature extraction



Fine-tuning pre-trained model



Transfer learning when no labels are available



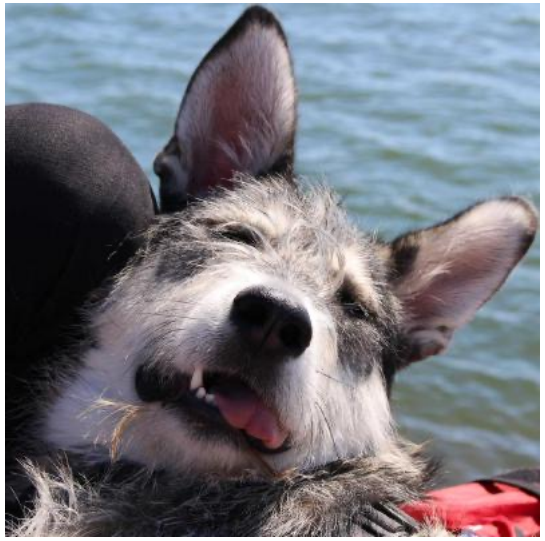
0
1
2
...
9

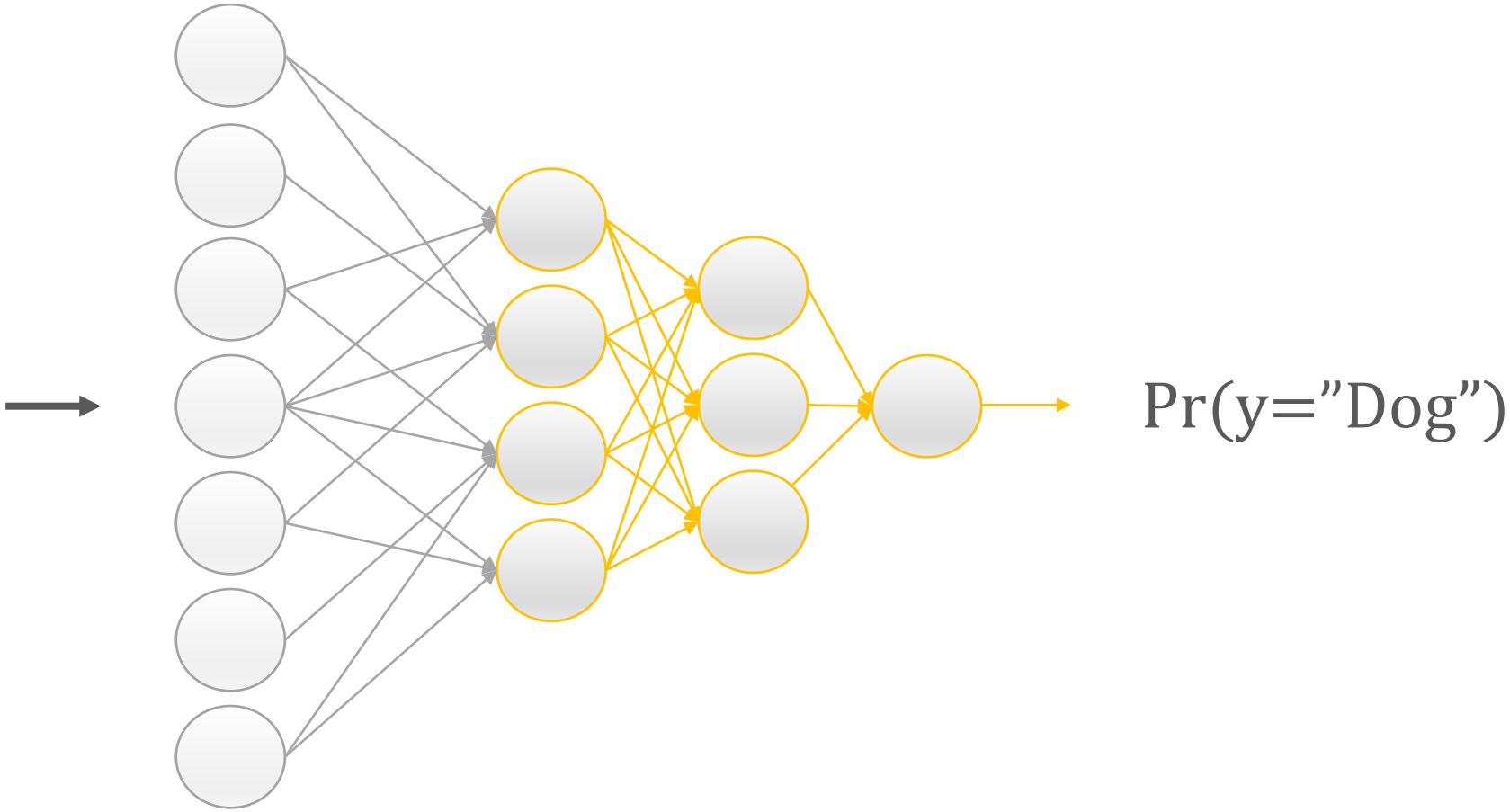
MNIST

MNIST-M

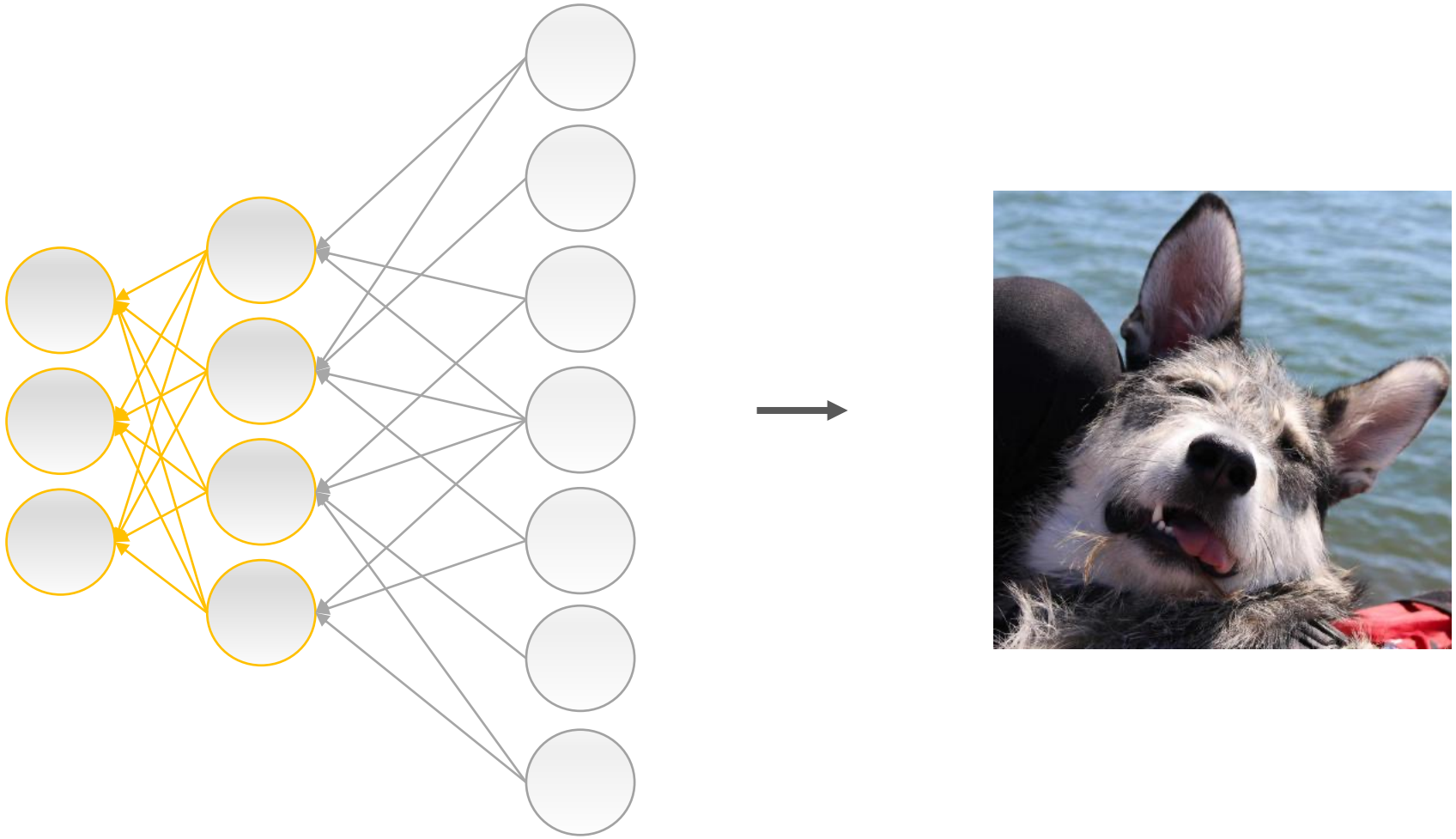
Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., ... & Lempitsky, V. (2016). Domain-adversarial training of neural networks. The journal of machine learning research, 17(1), 2096-2030.

Image classifying ANN



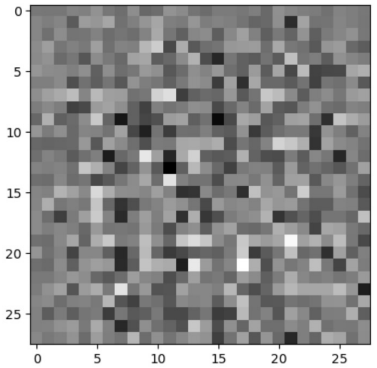


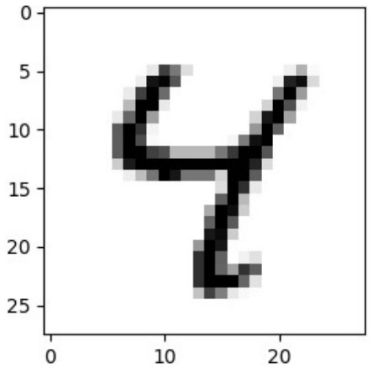
Data (image) generating ANN



generated sample

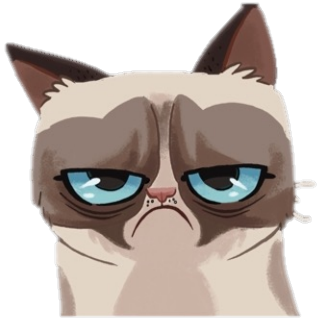
targer sample





Generator
“Artist”

LOSE



Discriminator
“Critic”

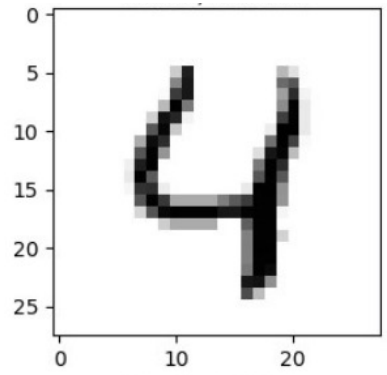
“Fake!”

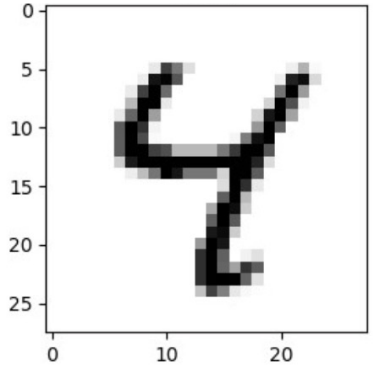
“Real!”

WIN

generated sample

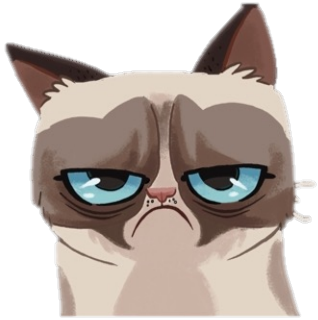
training sample





Generator
"Artist"

WIN

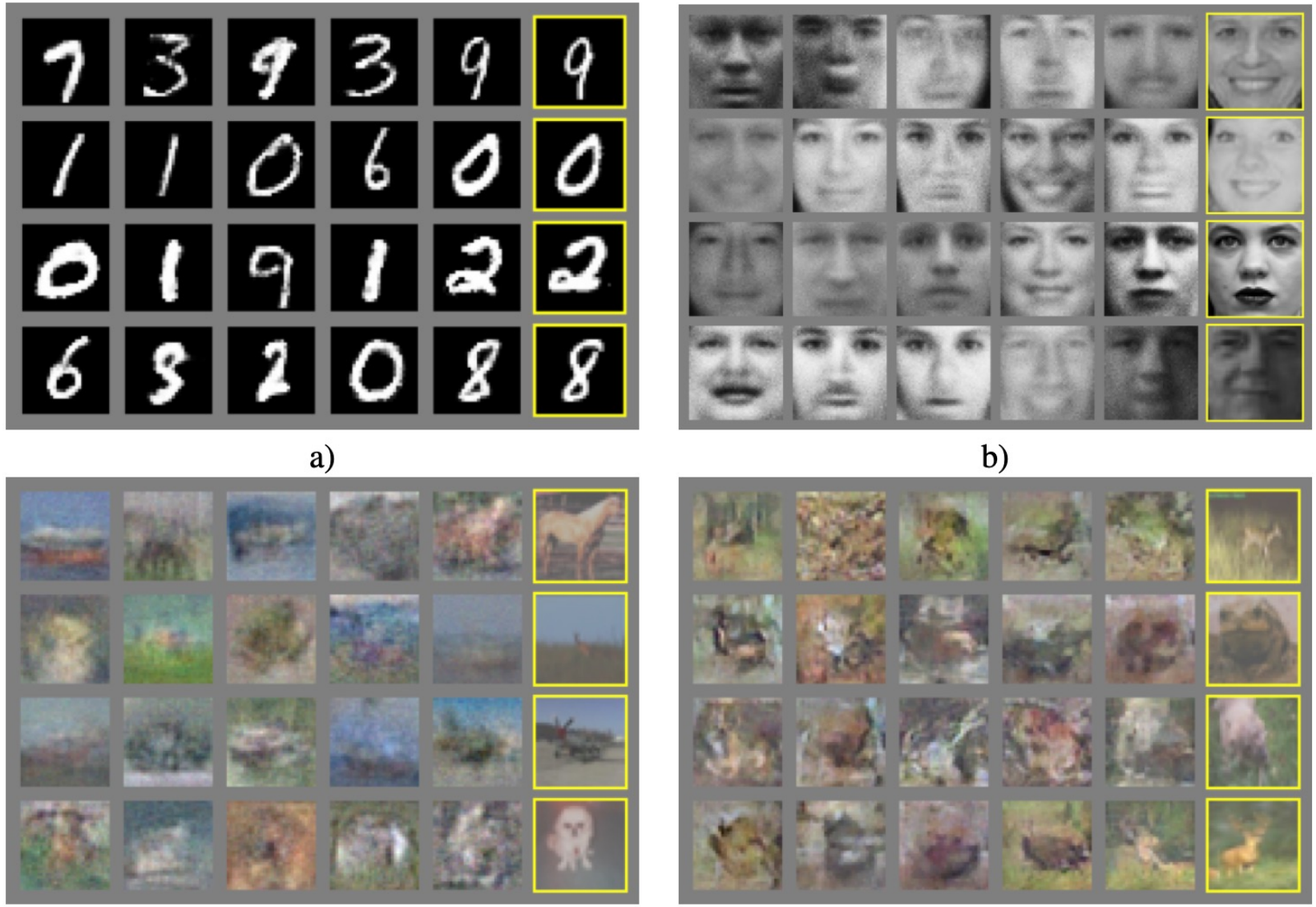


Discriminator
"Critic"

"Real!"

"Real!"

LOSE



a)

b)

c)

d)

“Generative Adversarial Nets” Ian Goodfellow 2014

GAN PROGRESS ON FACE GENERATION

Source: Goodfellow et al., 2014; Radford et al., 2016; Liu & Tuzel, 2016; Karras et al., 2018; Karras et al., 2019; Goodfellow, 2019; Karras et al., 2020; AI Index, 2021

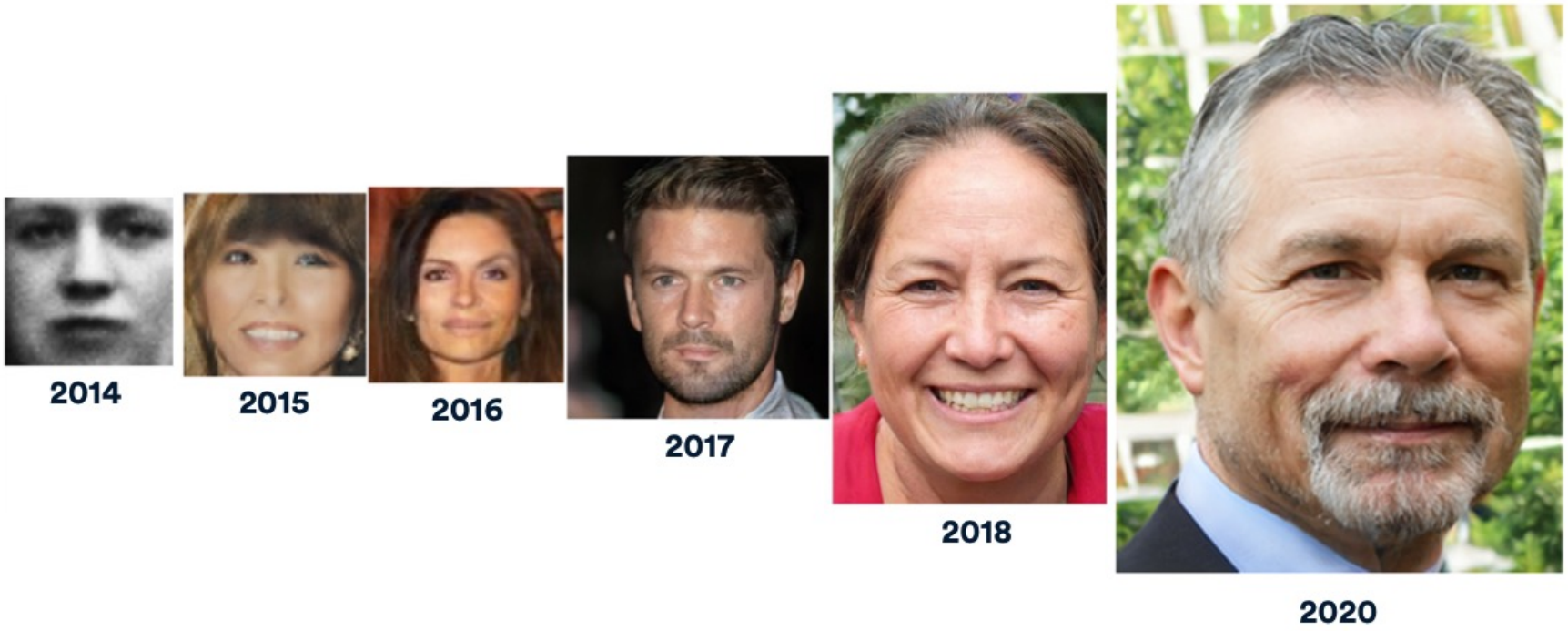
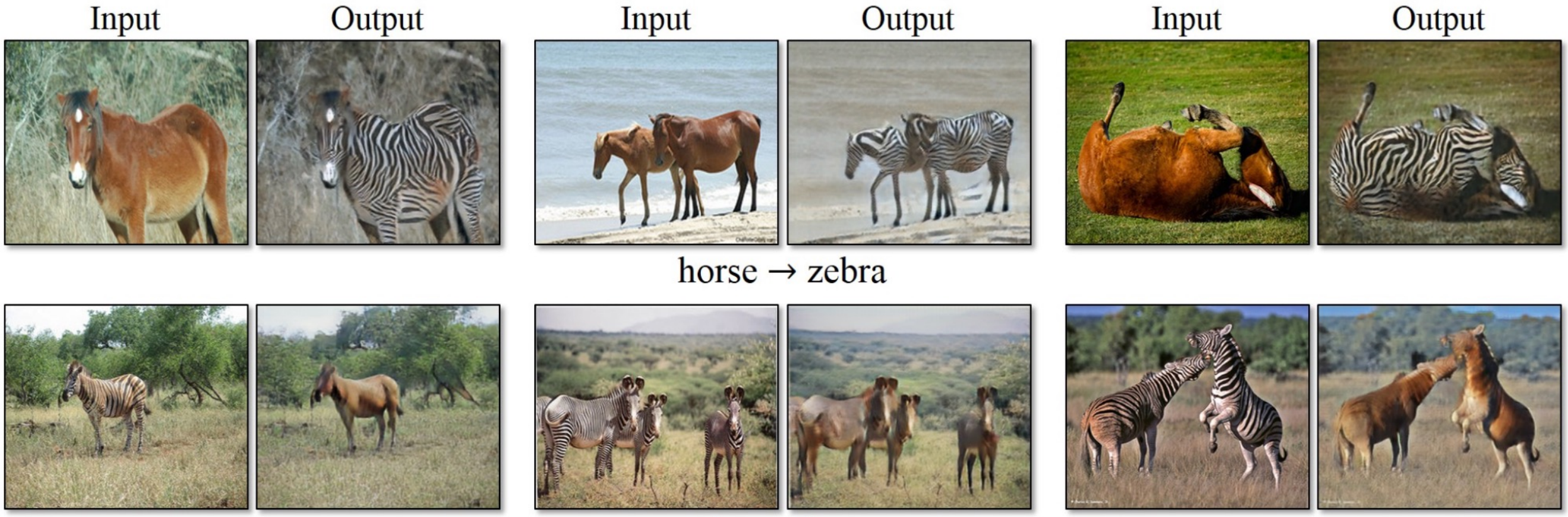


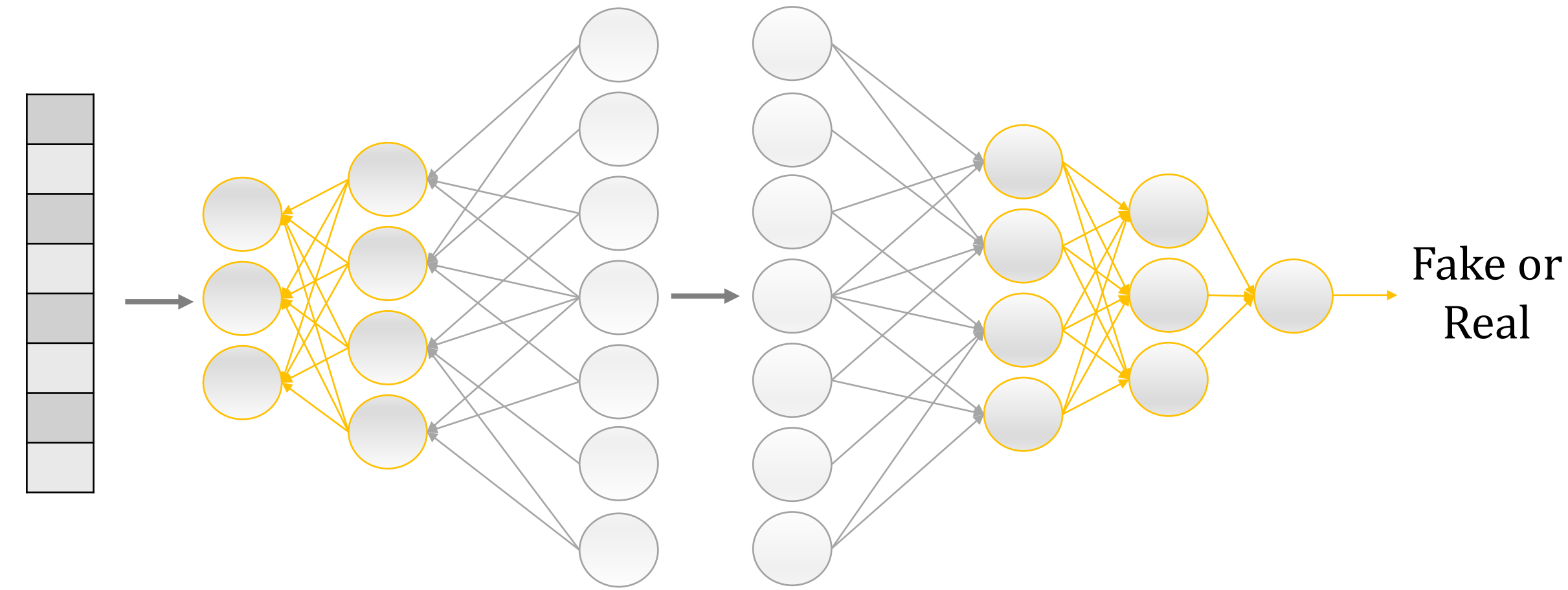
Figure 2.1.7

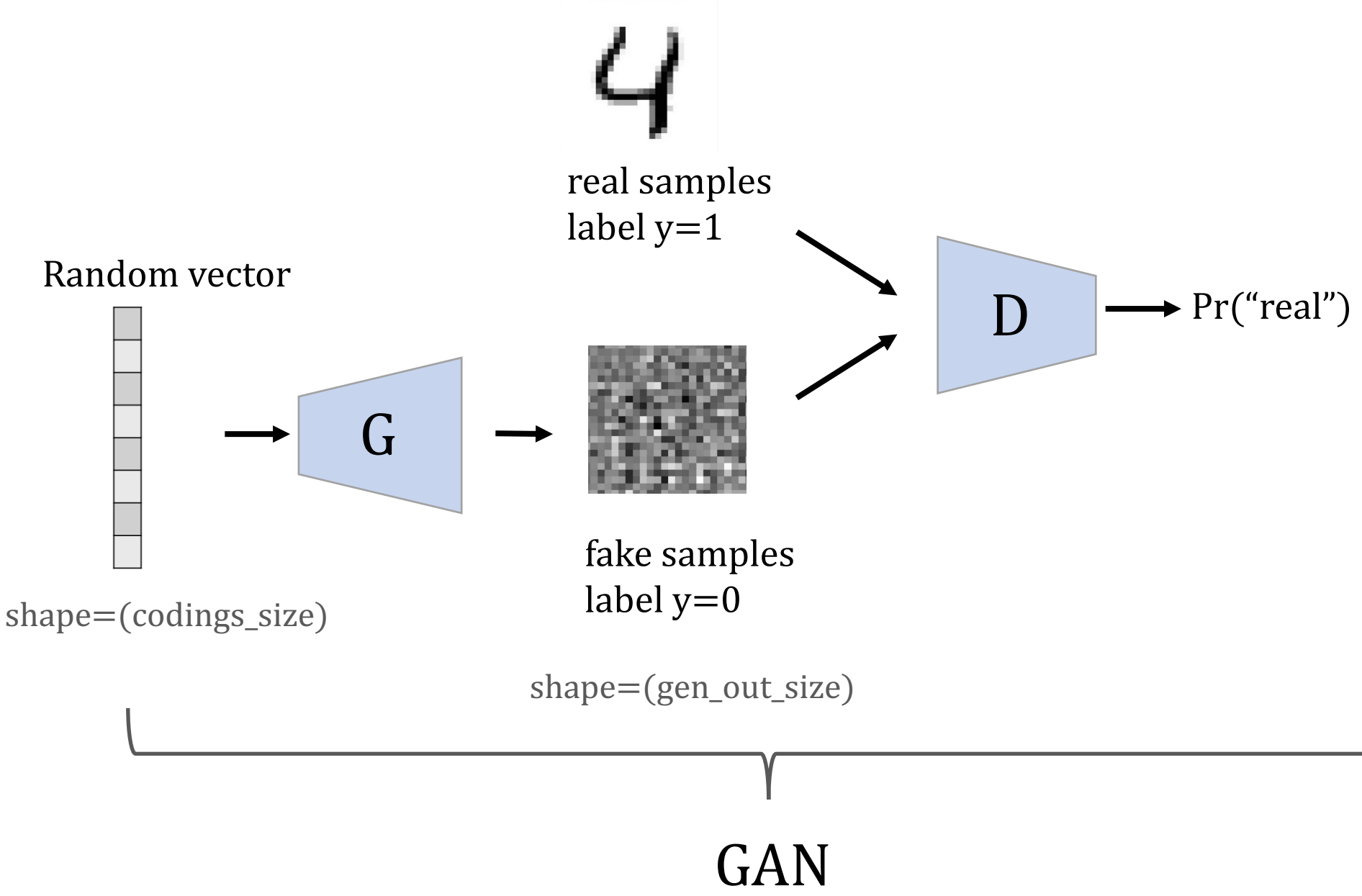
https://aiindex.stanford.edu/wp-content/uploads/2021/11/2021-AI-Index-Report_Master.pdf

https://thispersondoesnotexist.com

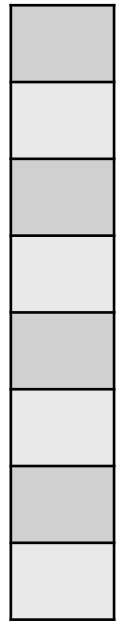


https://junyanz.github.io/CycleGAN/

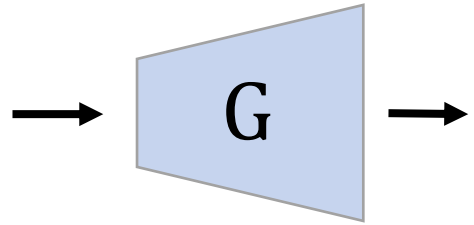

Random vector
Generator
Discriminator
Fake or Real


real samples
label y=1
Random vector
D
Pr("real")
G
fake samples
label y=0
shape=(codings_size)
shape=(gen_out_size)
GAN

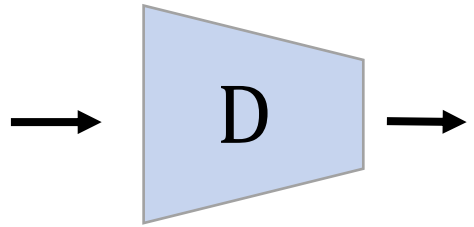
```
# random vector
codings_size=…


# sample random vectors
noise = tf.random.normal(shape=[batch_size, codings_size])


gen_out_size=…   # shape of the generated sample
```



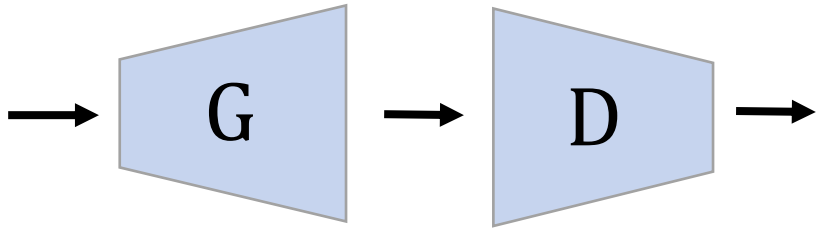
```
generator = tf.keras.models.Sequential([
    tf.keras.layers.Dense(..., input_shape=[codings_size]),
    tf.keras.layers.Dense(...),
    tf.keras.layers.Dense(gen_out_size)
])
```



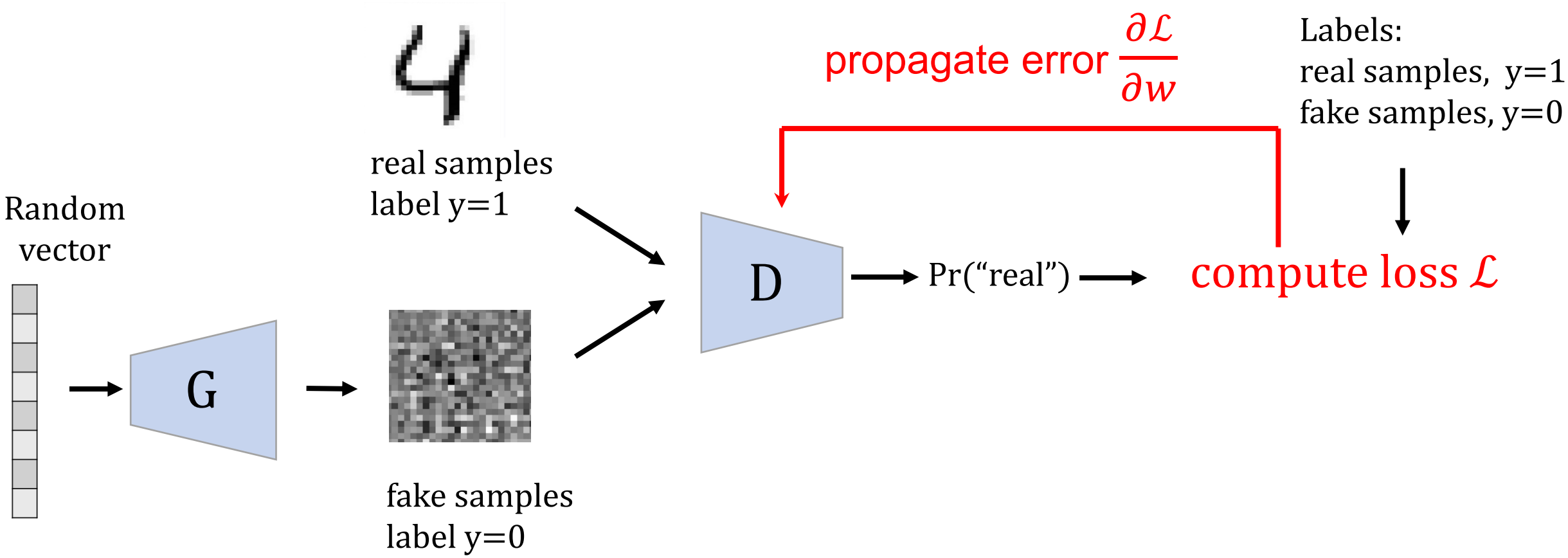

```
discriminator = tf.keras.models.Sequential([
    tf.keras.layers.Dense(..., input_shape=[gen_out_size ]),
    tf.keras.layers.Dense(...),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```



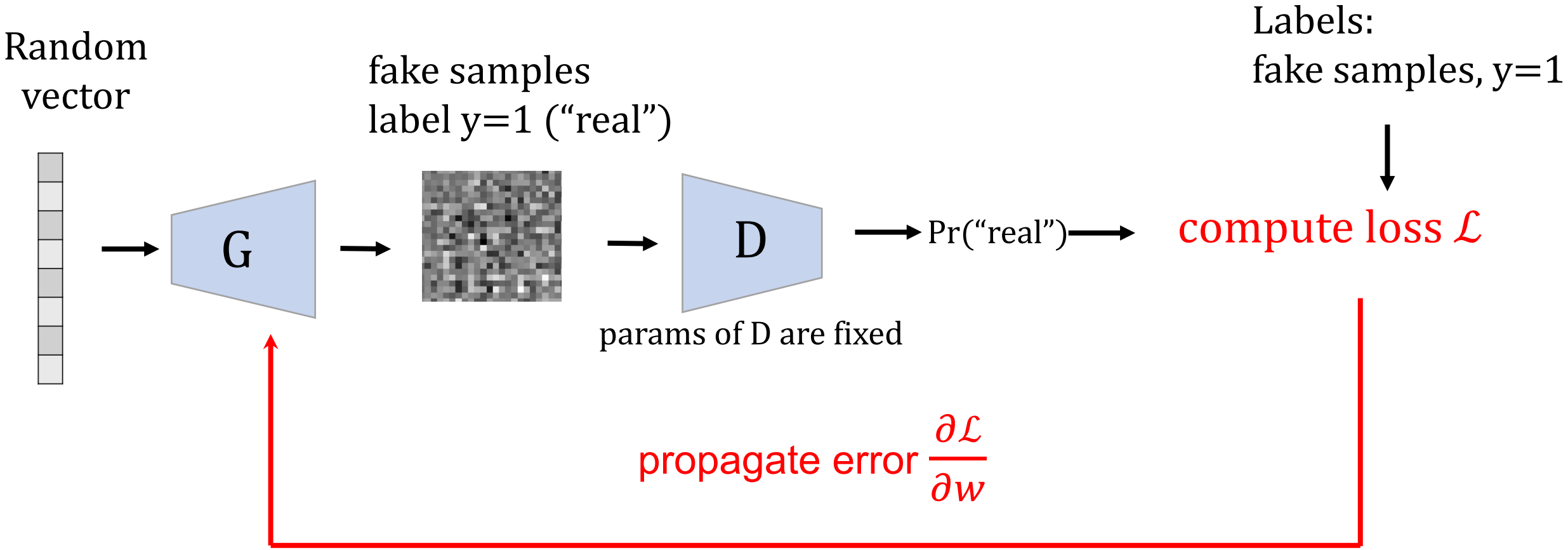

```
gan = tf.keras.models.Sequential([generator, discriminator])
```



Phase 1. Discriminator training



Phase 2. Generator training



Discriminator

maximize loss:

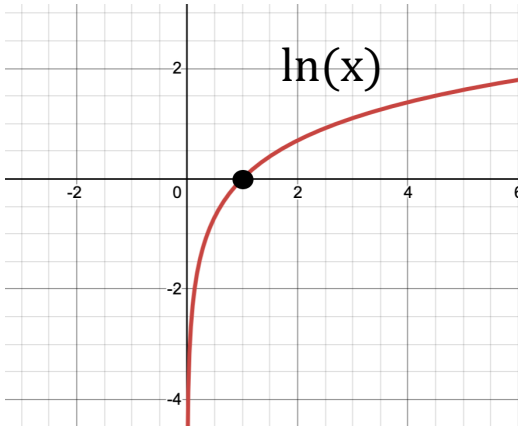
$$\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

minimize
binary CE

$$L = -\frac{1}{m}\sum_{i=1}^{m}(y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i))$$

$x^{(i)}$ - real data point, assign lable $y = 1$

$G(z^{(i)})$ – generated “fake” data point, assign lable $y = 0$



Generator

minimize loss:

$$\log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

minimize
binary CE

$$L = -\frac{1}{m}\sum_{i=1}^{m} y_i \cdot \log(\hat{y}_i)$$

$G(z^{(i)})$ – generated “fake” data point, assign lable $\boldsymbol{y = 1}$

```
# build one model
gan = tf.keras.models.Sequential([generator, discriminator])

# compile discriminator
# discriminator.trainable=True when training discriminator
discriminator.compile(loss="binary_crossentropy",
optimizer="rmsprop")

# compile gan
# discriminator.trainable=False when training gan
discriminator.trainable = False
gan.compile(loss="binary_crossentropy", optimizer="rmsprop")
```

```
def train_gan(gan, dataset, batch_size, codings_size, n_epochs=90):
    generator, discriminator = gan.layers
    for epoch in range(n_epochs):
        for X_batch in dataset:

            # phase 1 - training the discriminator
            noise = tf.random.normal(shape=[batch_size, codings_size])
            gen_samples = generator(noise)
            X_fake_and_real = tf.concat([gen_samples, tf.cast(X_batch, tf.float32)], axis=0)
            y1 = tf.constant([[0.]] * batch_size + [[1.]] * batch_size)
            discriminator.train_on_batch(X_fake_and_real, y1)

            # phase 2 - training the generator
            noise = tf.random.normal(shape=[batch_size, codings_size])
            y2 = tf.constant([[1.]] * batch_size)
            gan.train_on_batch(noise, y2)
```

random vector

fake samples

labels:

real y=1

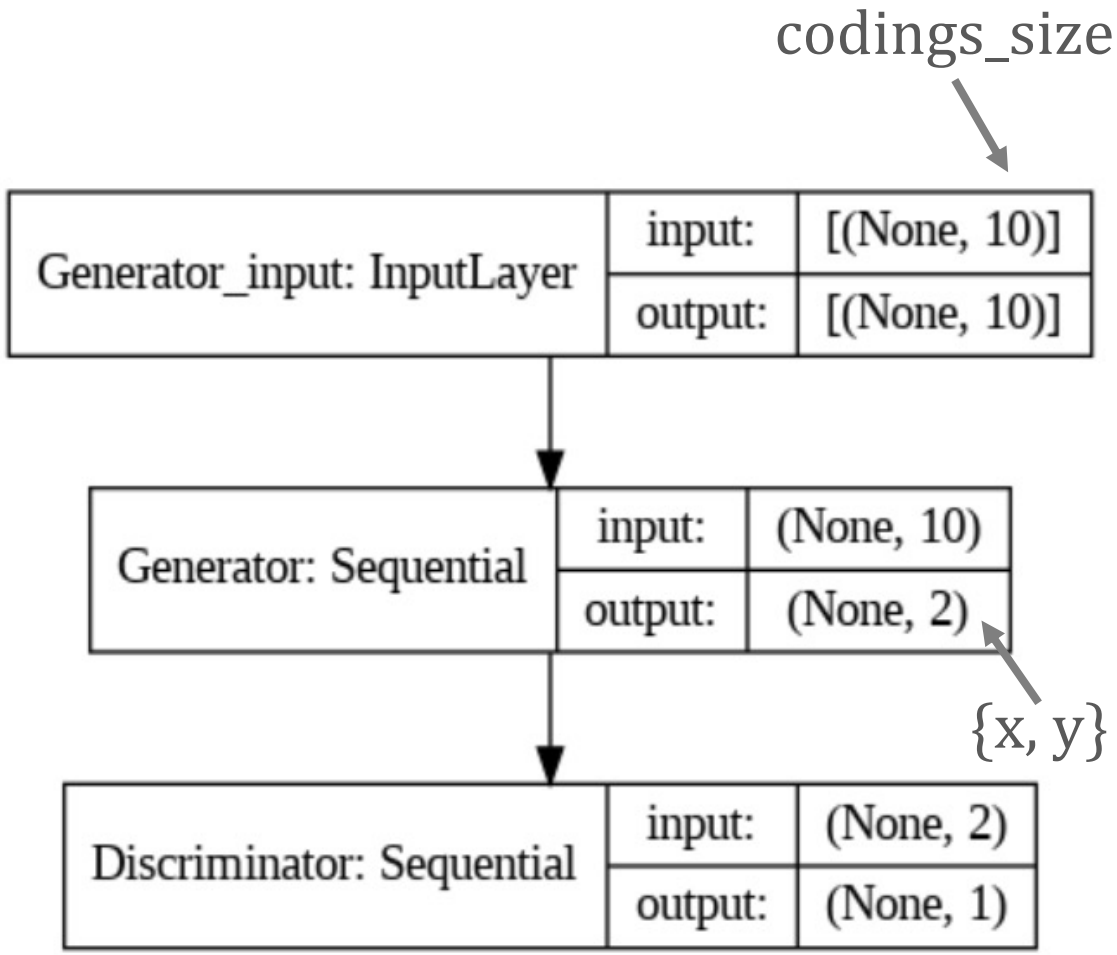
fake y=0

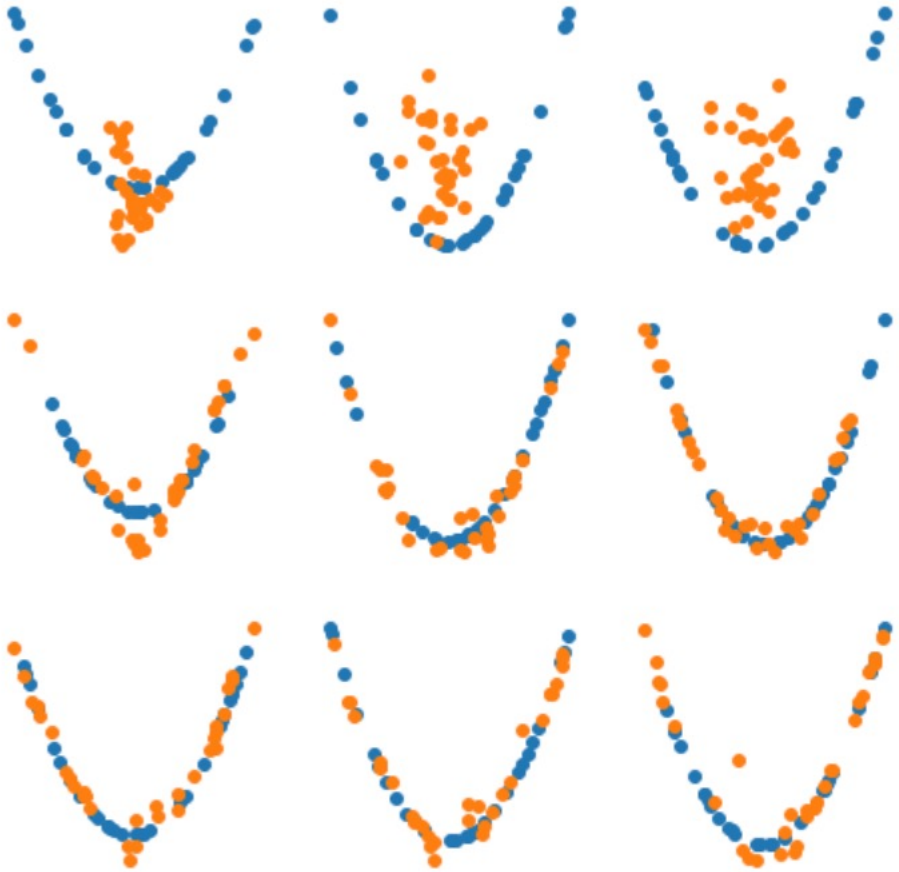
random vector

labels:

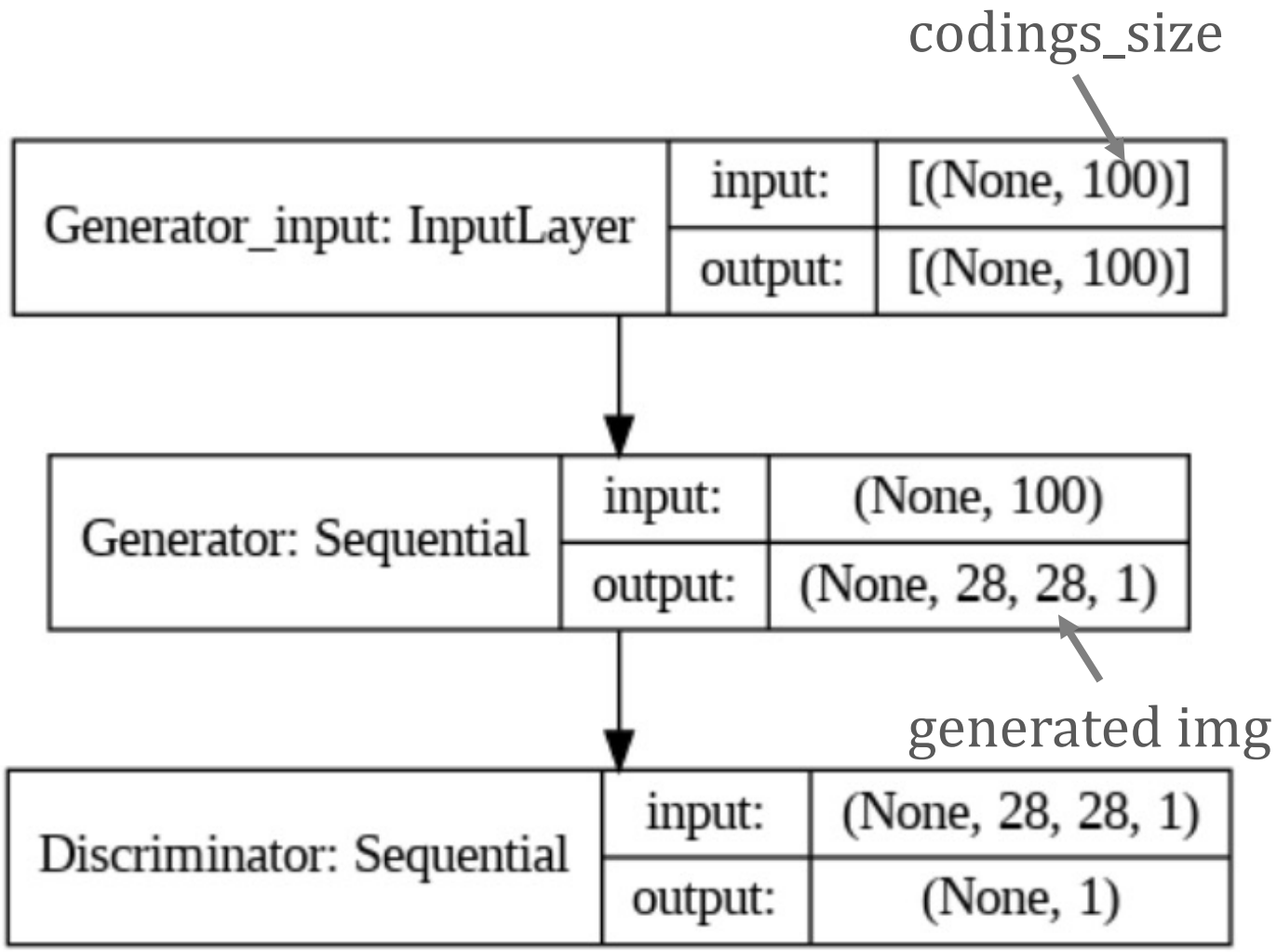
fake y=1

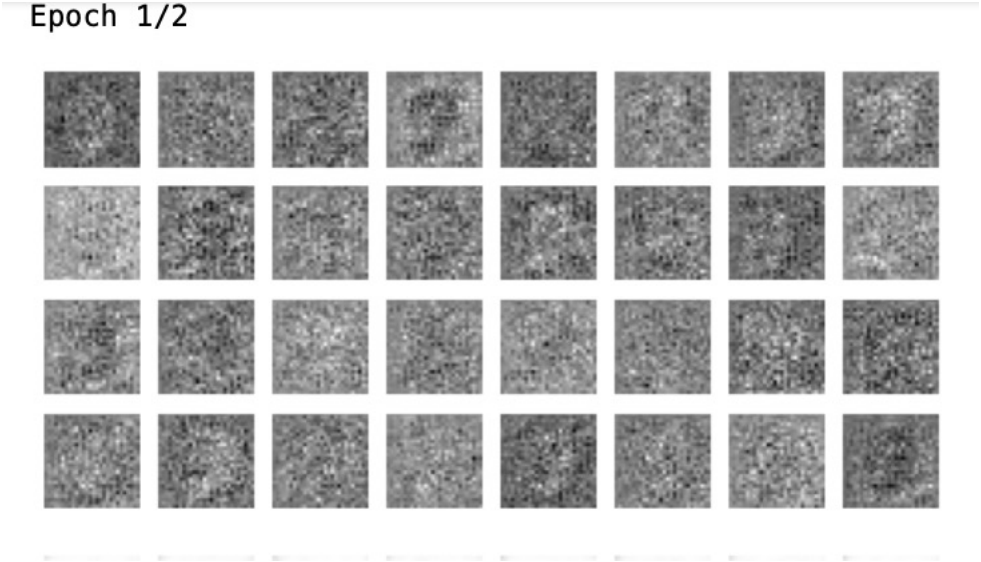
Generate samples from quadratic distribution.

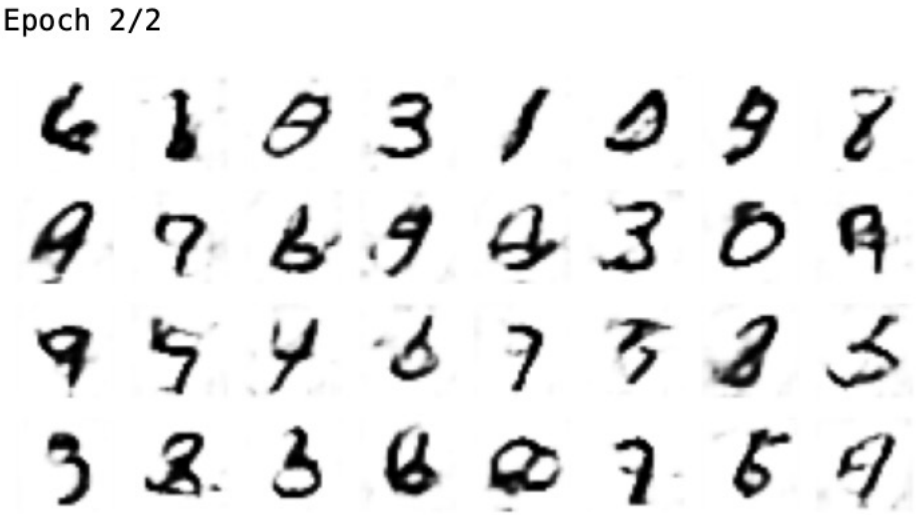




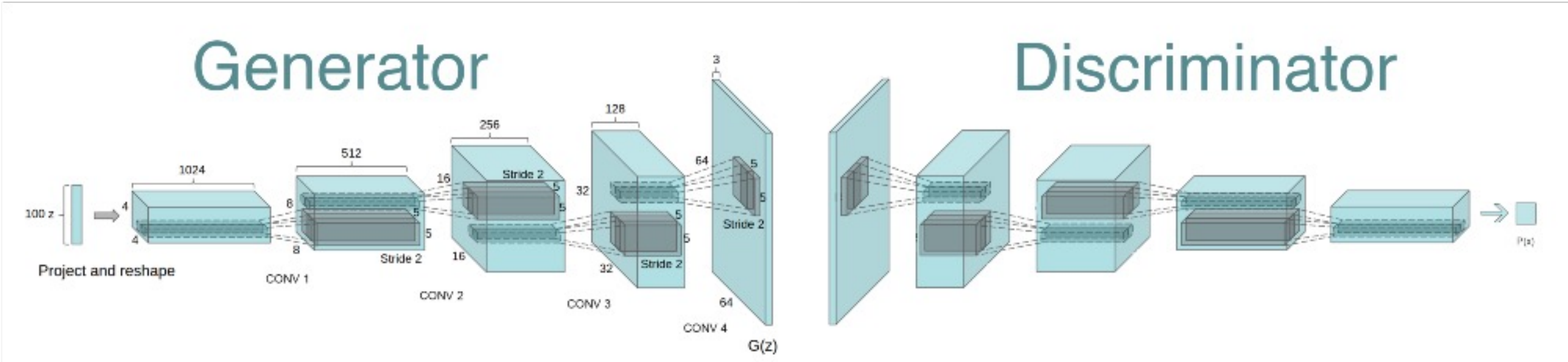
DCGAN on MNIST dataset



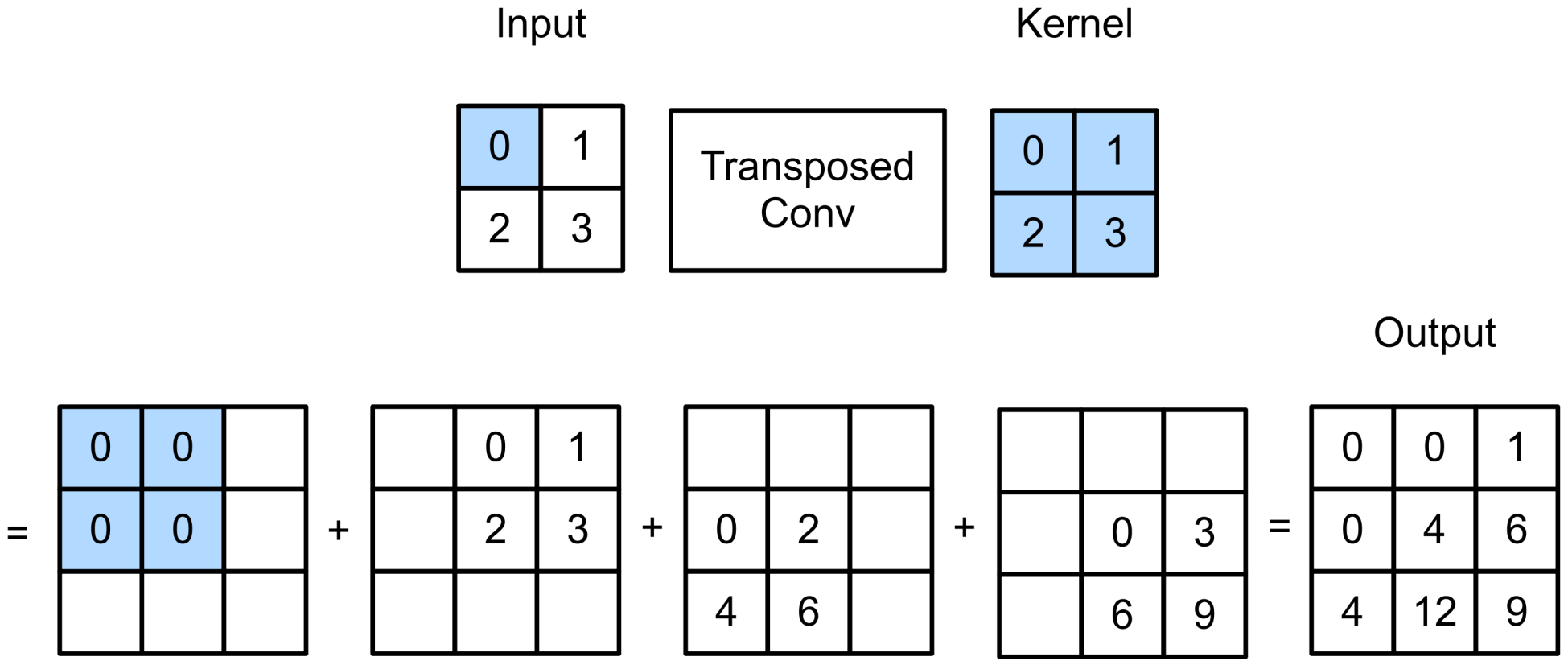




DCGAN – Deep Convolutional GAN

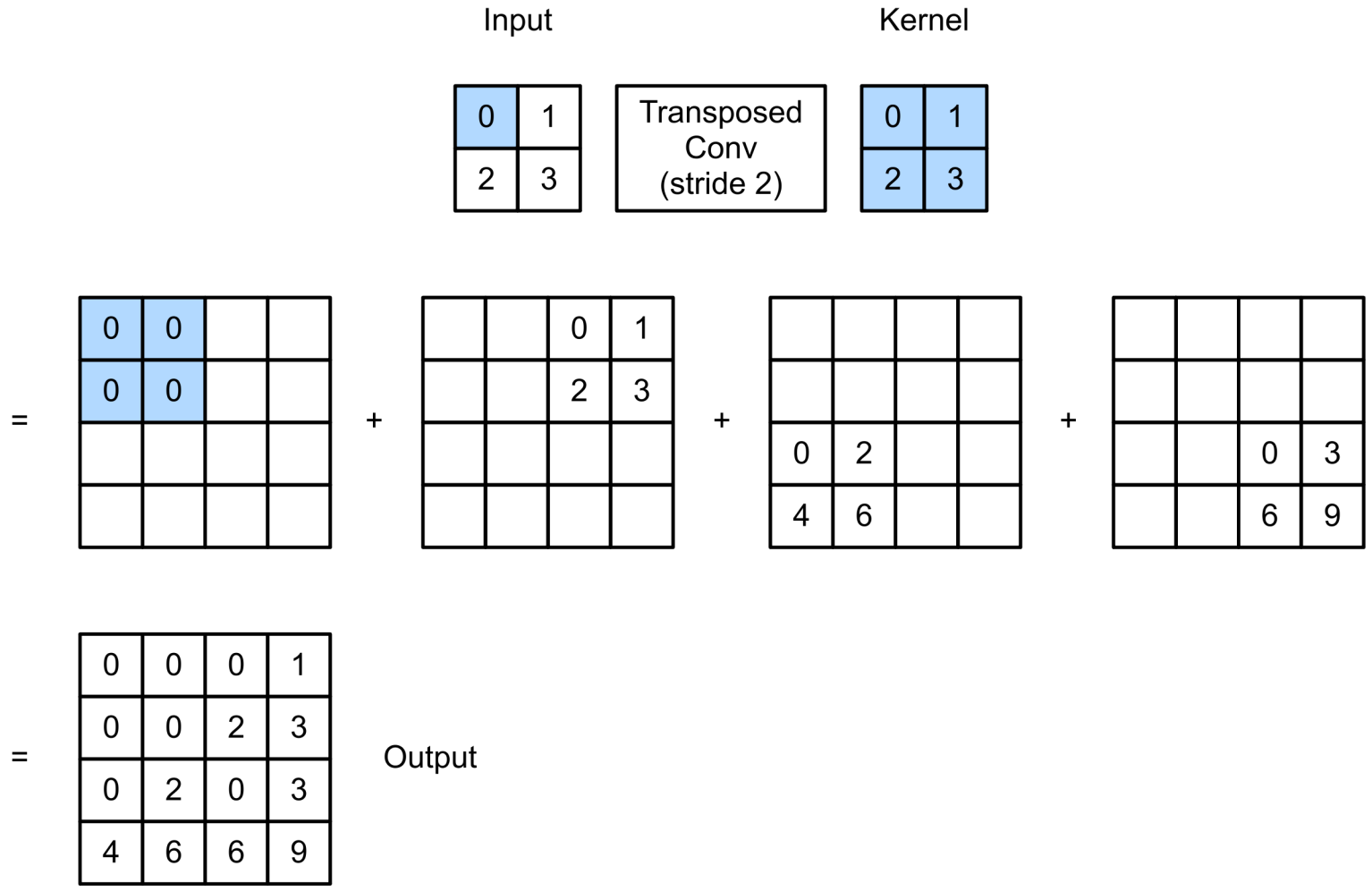


DCGAN – Transposed convolution



http://d2l.ai/chapter_computer-vision/transposed-conv.html

DCGAN – Transposed convolution



http://d2l.ai/chapter_computer-vision/transposed-conv.html