Generative Learning

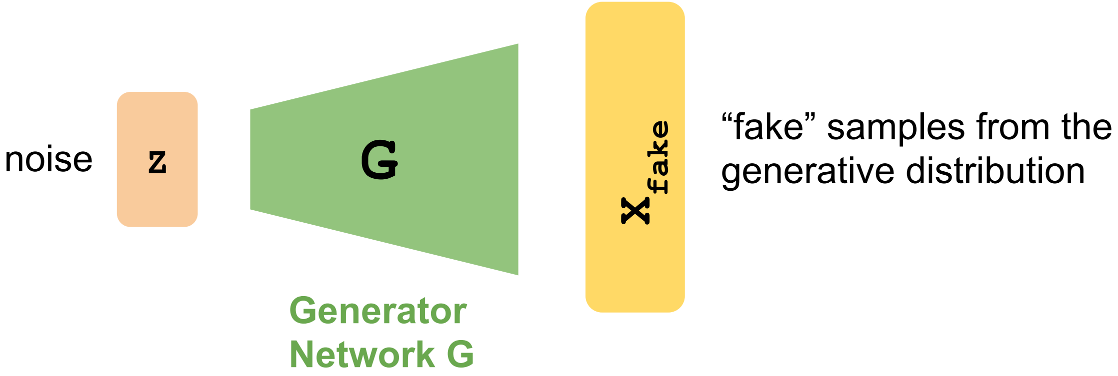
Advances in Generative Adversarial Networks

N. Rich Nguyen, PhD

SYS 6016

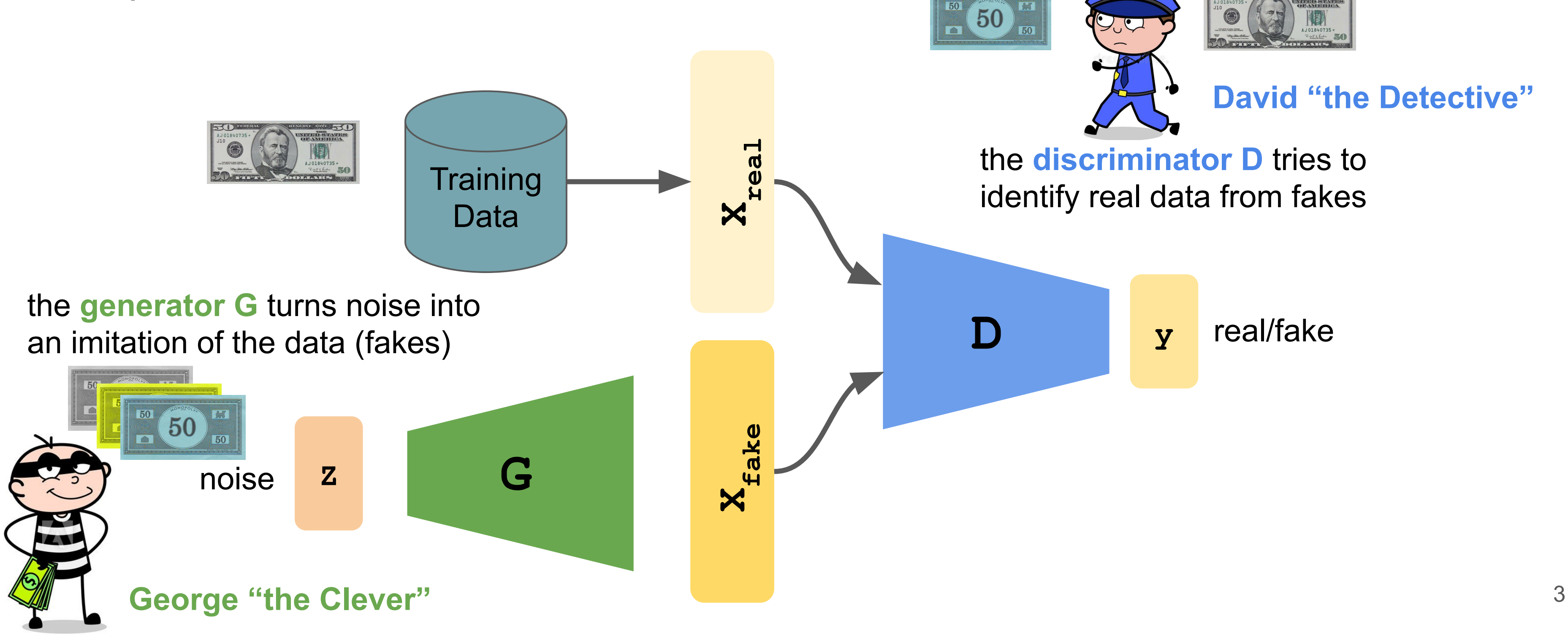
Density distribution: Building or Sampling?

- **Problem**: we tried to generate samples from a complex density distribution, but it's hard to learn such a distribution directly!
- **Solution**: instead of attempt to model the density distribution, just generate new instances by sampling from something simple (like a noise distribution), and then learn a transformation to a desired distribution.



Generative Adversarial Networks (GANs)

GANs are a way to make a generative model by having two neural networks compete with each other as **adversaries**.



Designing a simple GAN for Fashion MNIST

```
codings_size = 30

generator = keras.models.Sequential([
    keras.layers.Dense(100, activation="selu", input_shape=[codings_size]),
    keras.layers.Dense(150, activation="selu"),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
discriminator = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(150, activation="selu"),
    keras.layers.Dense(100, activation="selu"),
    keras.layers.Dense(1, activation="sigmoid")
])
gan = keras.models.Sequential([generator, discriminator])
```

Training GANs

- **Generator G (George)** tries to create fake data to trick **discriminator D (David)**
- **David** tries to identify real data from fakes created by **George**
- **George** and **David** are trained jointly by the below objective function (called a **minimax game)**:

$$\min_{G} \max_{D} V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

- **David** wants to **maximize** the objective s.t. `D(x)` close to 1, and `D(G(z))` close to 0.
- **George** wants to **minimize** the same objective s.t. `D(G(z))` close to 1.
- Convergence: `D(x)`=`D(G(z))`=0.5 (**David** cannot tell real/fake anymore)

Intuition behind GANs

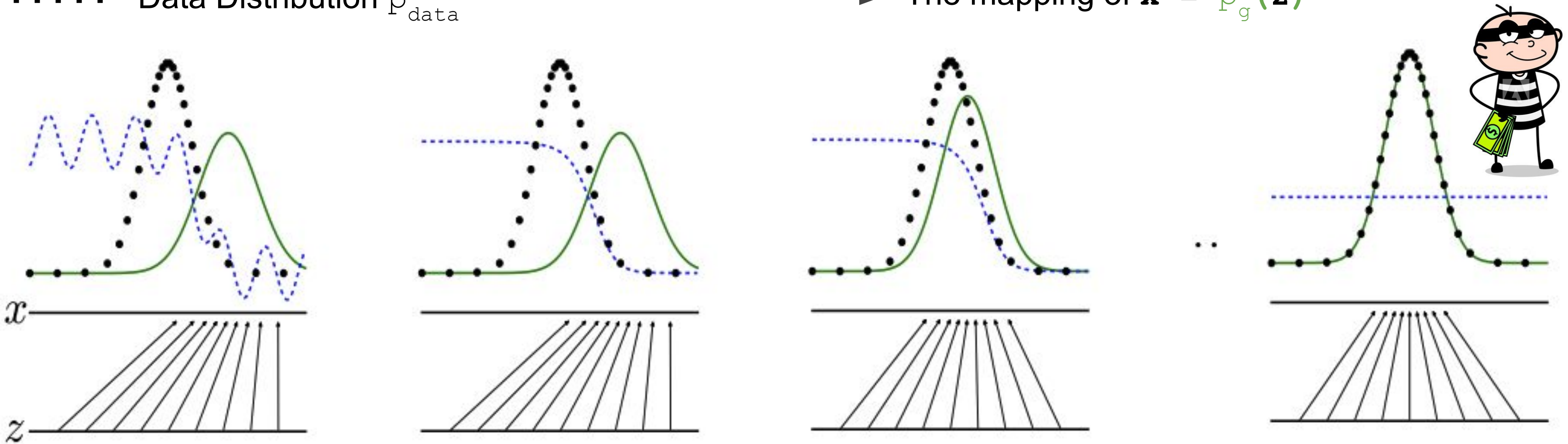
GANs are trained by simultaneously updating **David** so that he discriminates between real data and fake data which is generated by **George**.

Discriminative Distribution (**David**)

Generative Distribution p_g (**George**)

Data Distribution p_{data}

The mapping of $\mathbf{x} = p_g(\mathbf{z})$



z is sampled uniformly
x is generated by p_g

Train David to discriminate p_g and p_{data}

Train George to change p_g to fool **David**

Convergence: **David** is *uniform* (p_g=p_{data})

Implementing the training of GAN

```
batch_size = 32
dataset = tf.data.Dataset.from_tensor_slices(X_train).shuffle(1000)
dataset = dataset.batch(batch_size, drop_remainder=True).prefetch(1)
```

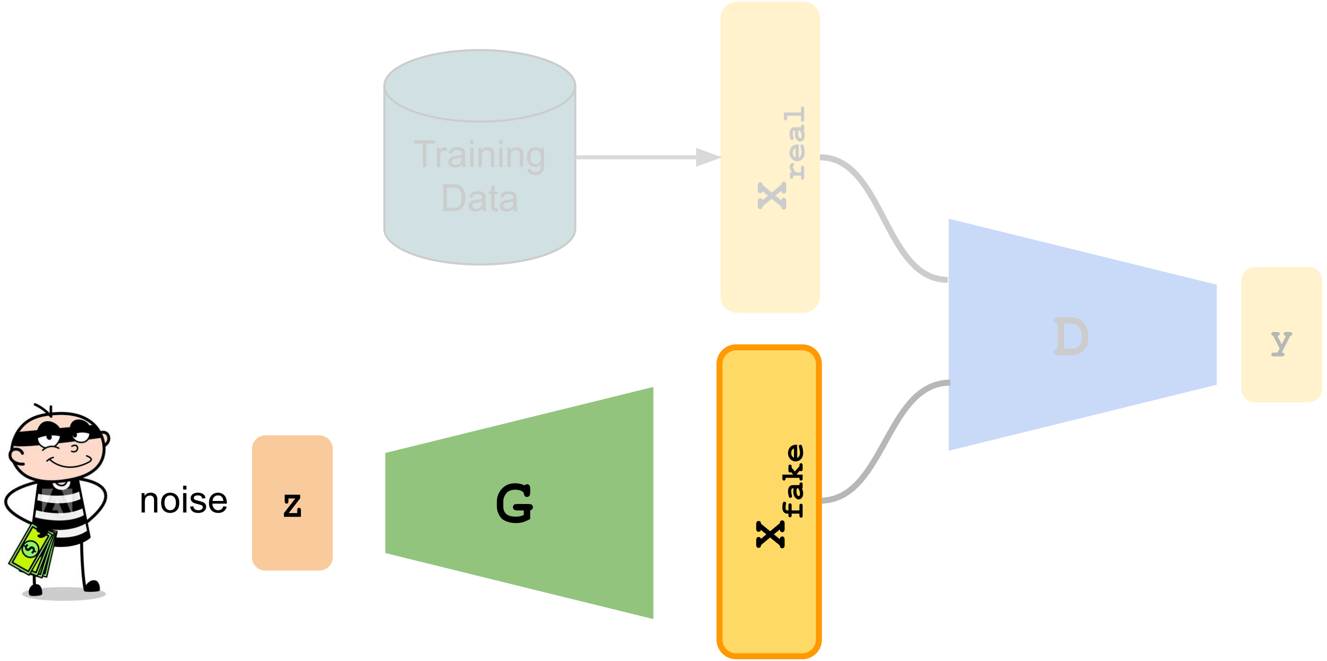
```
discriminator.compile(loss="binary_crossentropy", optimizer="rmsprop")
discriminator.trainable = False
gan.compile(loss="binary_crossentropy", optimizer="rmsprop")
```

```
def train_gan(gan, dataset, batch_size, codings_size, n_epochs=50):
    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])
            generated_images = generator(noise)
            X_fake_and_real = tf.concat([generated_images, X_batch], axis=0)
            y1 = tf.constant([[0.]] * batch_size + [[1.]] * batch_size)
            discriminator.trainable = True
            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)
            discriminator.trainable = False
            gan.train_on_batch(noise, y2)

train_gan(gan, dataset, batch_size, codings_size)
```

Generating new data with trained GANs

- After training, use **George** to create new data that's never been seen before.
- If it's trained *effectively*, the generated data will be **indistinguishable** to real data (to both **David** and humans in general)

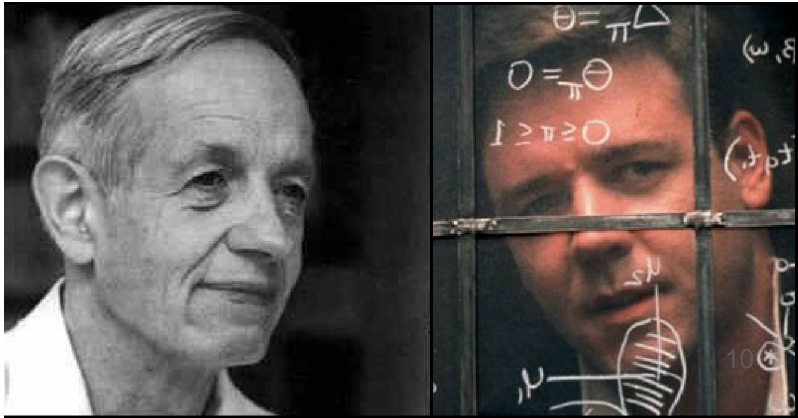


Images generated after one training epoch

Unfortunately, the images never really get much better than this. It turns out that training a GAN is very **challenging**!

The difficulties of training GANs

- During training, both **George** and **David** constantly try to outsmart each other, in a **zero-sum game**!
- As training advances, the game may end up in a state that game theorists call a **Nash Equilibrium** named after John Nash (played by Russell Crowe in *Beautiful Mind*). Ether **George** nor **David** would be better off changing his strategy assuming another player do not change his.
- However, it's not that simple: **nothing guarantees** Nash Equilibrium can ever be reached!

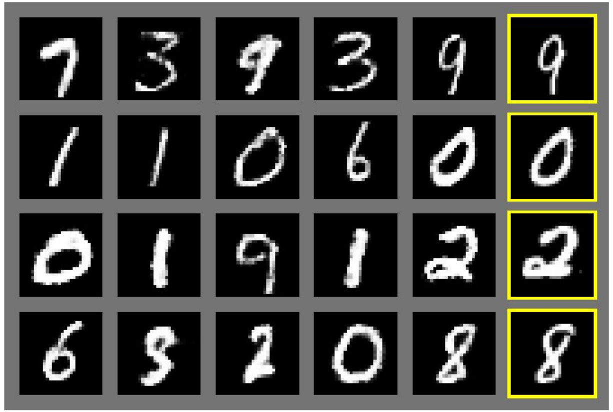


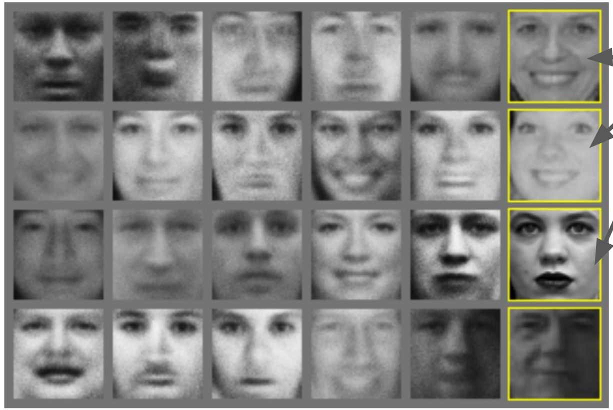
The difficulties of training GANs (cont)

- **Mode Collapse:** this happens when the **George**'s output gradually becomes less diverse. For example, *suppose **George** has successfully fool **David** with image of shoes, and this will encourage him to produce even more images of shoes and forgets to produce anything else.*
- **Hyperparameter**: Because **George** and **David** are constantly compete, their parameters may end up *oscillating* and become *unstable*. GANs are very sensitive to hyperparameters → lots of effort fine tuning them.
- **Experience Replay:** storing images produced by **George** at each iteration in a *replay buffer*, and the training of **David** use images drawn from this buffer instead of produced by the current state of **George**.
- The *dynamics* of **George** and **David** are still not *well* understood.

Original GANs (2014 by Goodfellow*)

MNIST

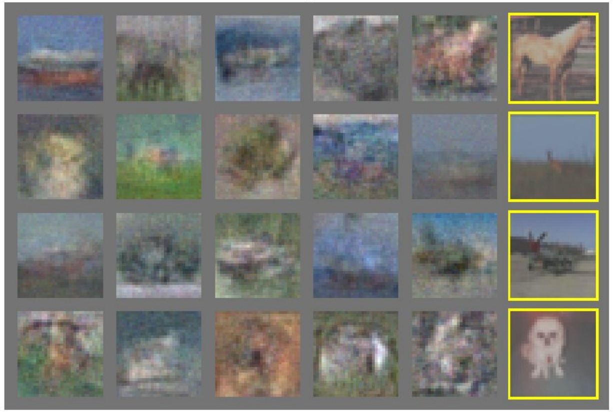


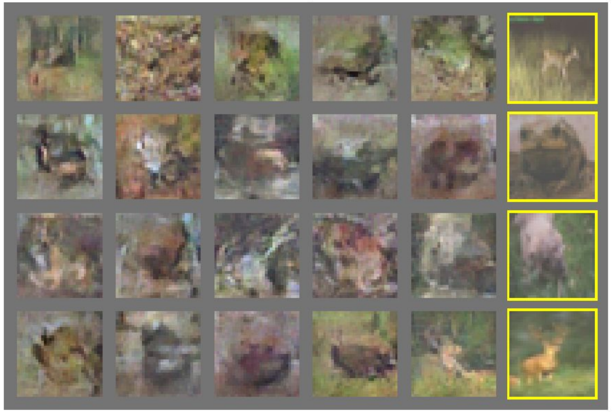


Nearest training sample to the generated image on its left.

TFD

CIFAR-10 (Dense Model)



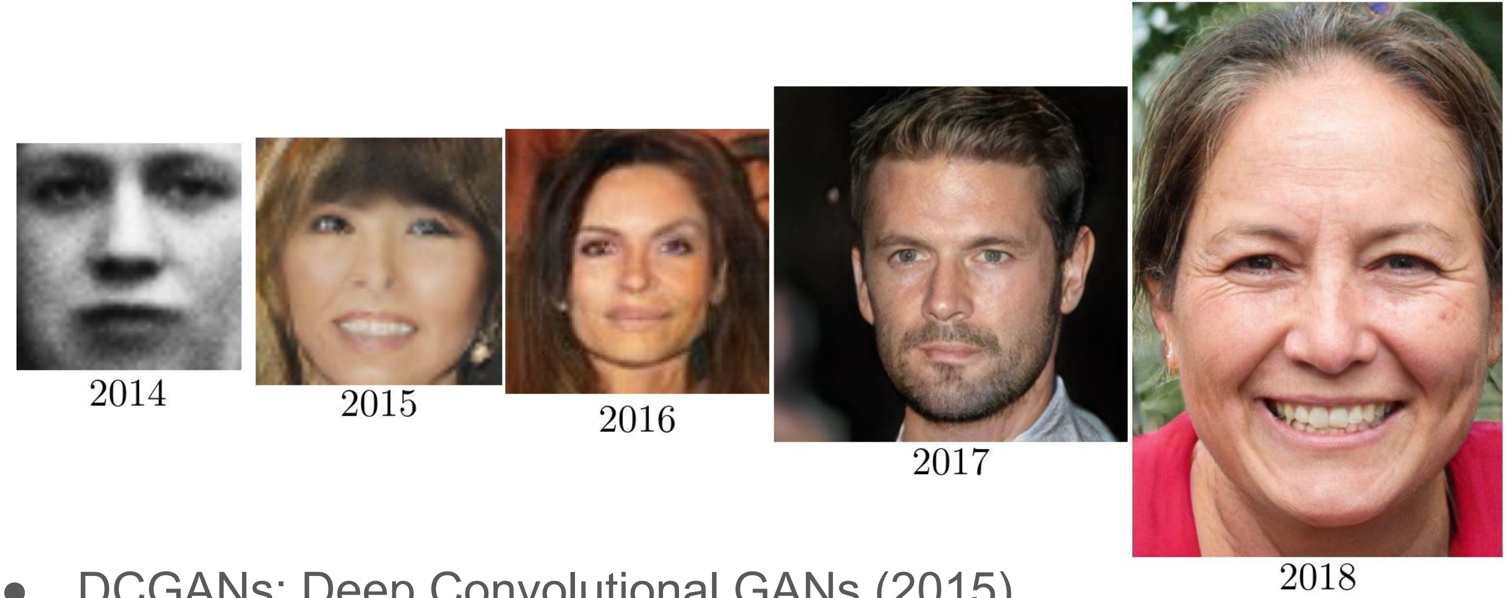


CIFAR-10 (Conv D and Deconv G)

*Goodfellow, Ian, et al. "Generative adversarial nets." *Advances in neural information processing systems*. 2014.

Recent Advances in GANs

GAN quality has progressed rapidly (adopted from a tweet by Ian Goodfellow)



- DCGANs: Deep Convolutional GANs (2015)
- CycleGAN: domain transformation (2017)
- Progressive Growing GANs (2018)
- StyleGAN: style-based generator (2019)

Deep Convolutional GANs (DCGANs*)

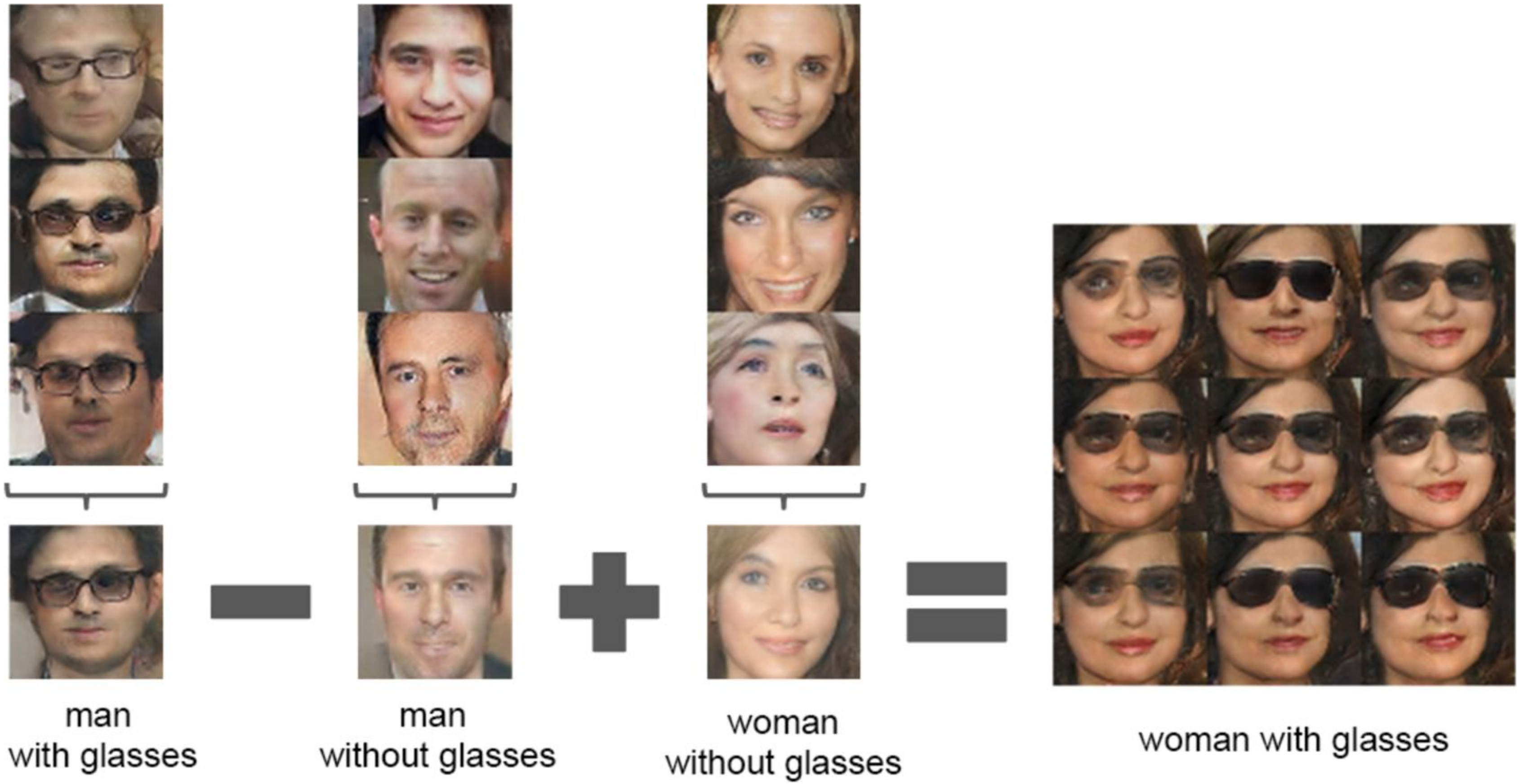
Making some guidelines for building a stable convolutional GANs:

- Replace pooling layers with strided convolutions in **David** and transposed convolutions in **George**.
- Use Batch Normalization in both **George** and **David**
- Use ReLU for **George** for all layers except output layer
- Use leaky ReLU in **David** for all layers
- Remove Dense layers

⇒ **Fairly realistic** images, they will get better with more data and deeper networks!

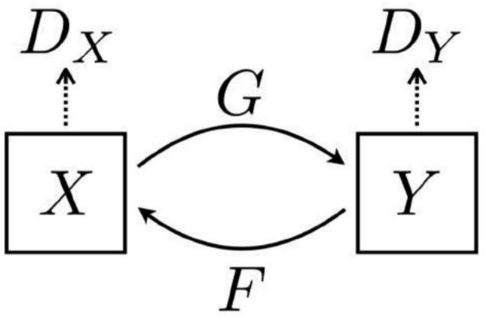
*Radford, Alec et al "Unsupervised representation learning with deep convolutional generative adversarial networks." *arXiv preprint arXiv:1511.06434* (2015).

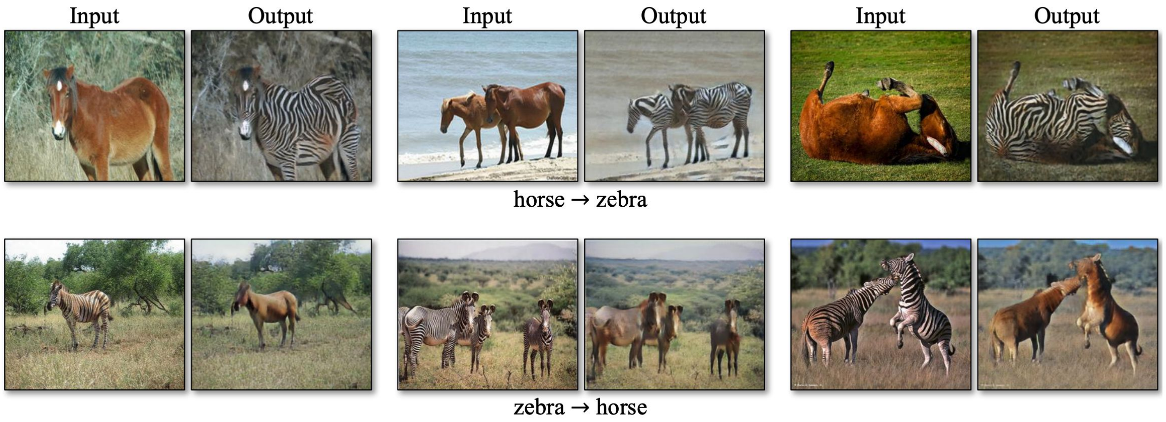
DCGANs: Latent Vector Arithmetics



CycleGAN: domain transformation*

CycleGAN learns transformation across domains with unpaired data



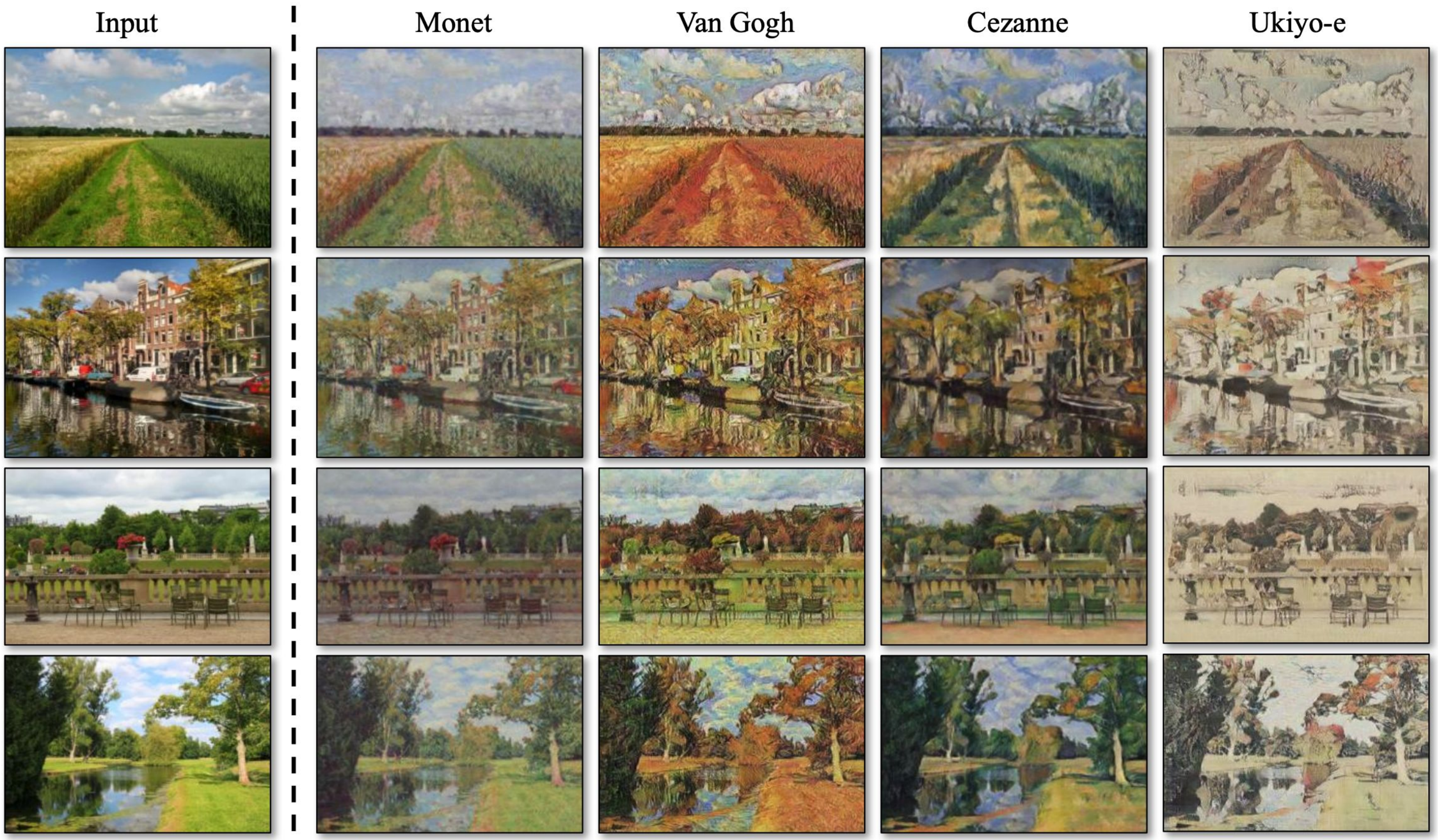


Not only just adding/removing **stripes…**

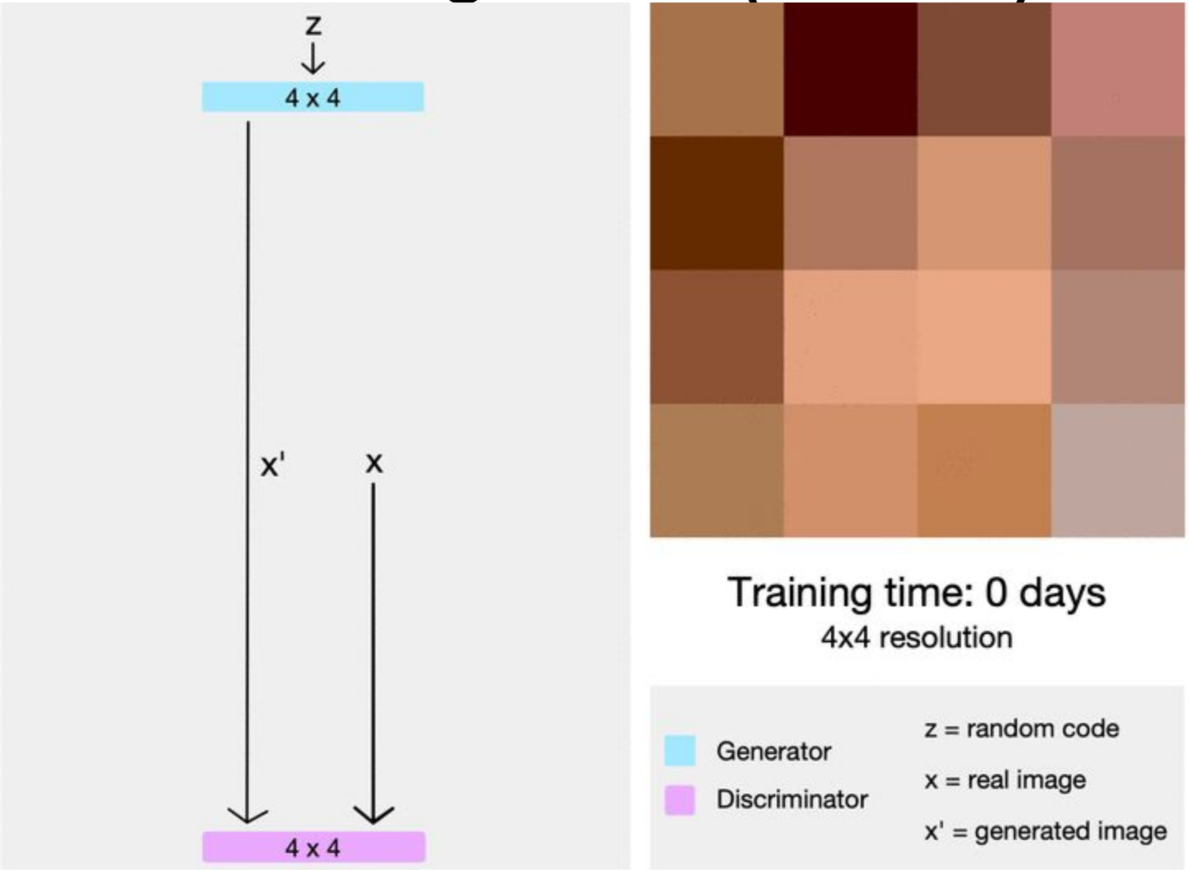
Also changes the **grass** to more yellow and dry (typically found in zebra's habitat in African savanna)

*Zhu, Jun-Yan, et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks." *ICCV* 2017.

CycleGAN demo



Progressive Growing GANs (NVIDIA)*



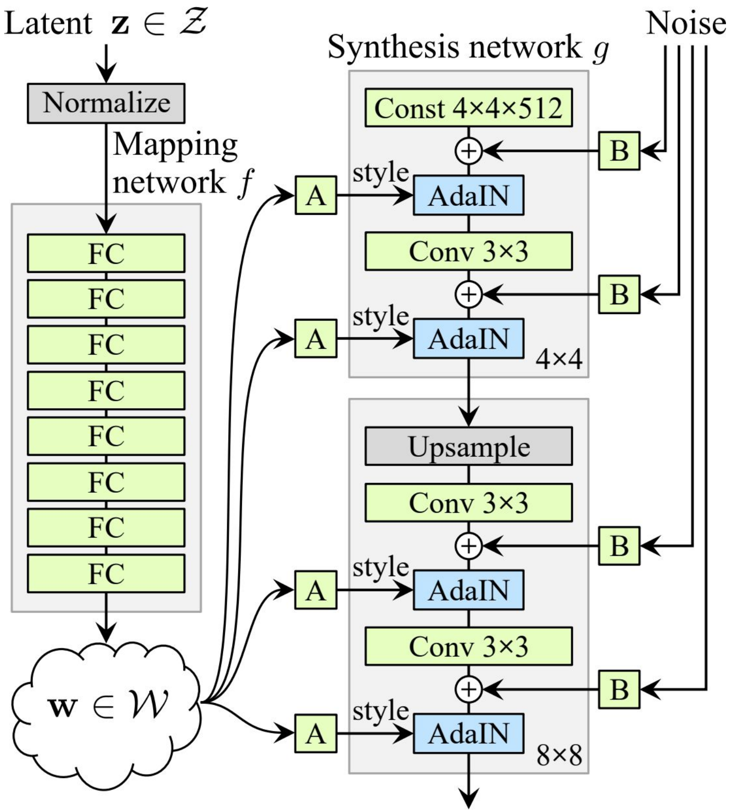
*Karras, Tero, et al. "Progressive Growing of GANs for Improved Quality, Stability, and Variation, ICLR 2018."

Progressive Growing GANs: on CelebA-HQ

Karras, Tero, et al. "Progressive Growing of GANs for Improved Quality, Stability, and Variation, ICLR 2018."

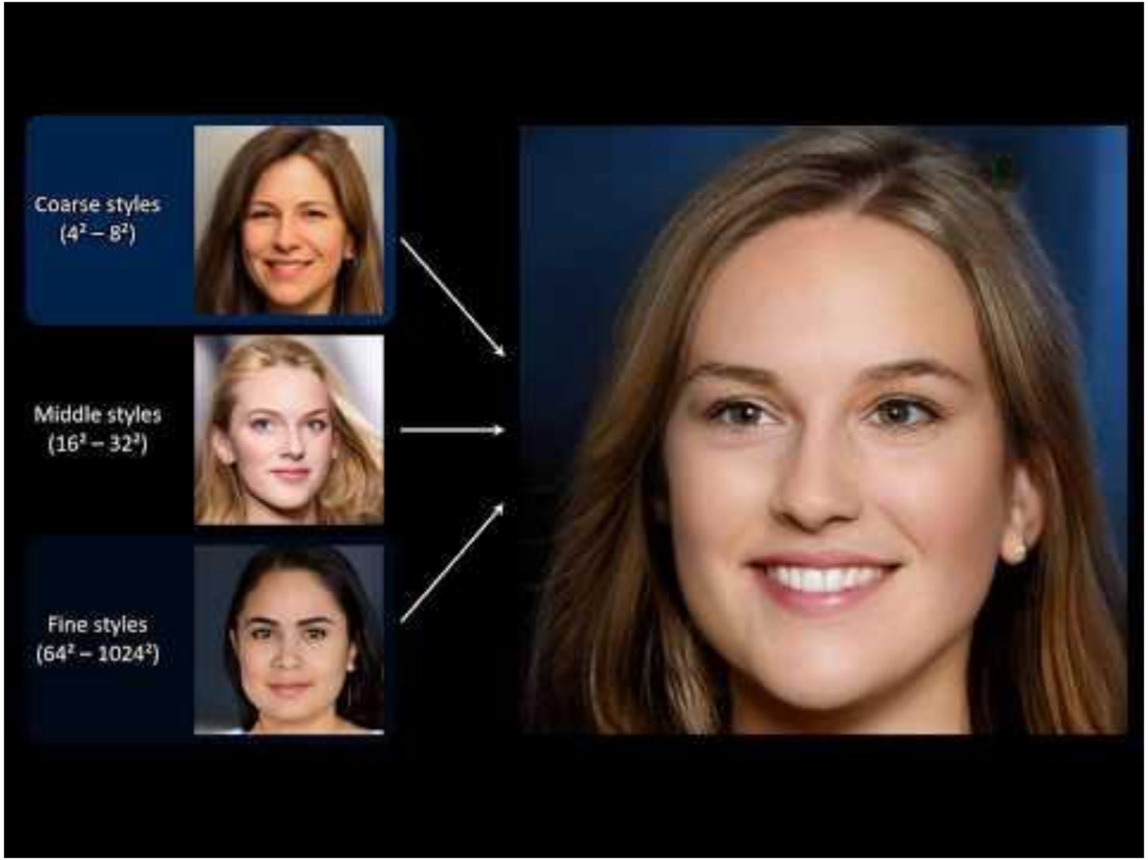
StyleGAN: Style-based generator

- Use style transfer techniques in generator **George** to ensure generated images have the same local structure as the training images.
- The discriminator **David** and loss function were not modified.
- **George** now consists of two networks:
  - **Mapping Network**: maps the latent representation z to multiple style vectors
  - **Synthesis Network**: generates images through multiple convolutional and upsampling layers.
- Adding noise independently



Karras, Tero, et al. "A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR 2019."

StyleGAN: Style-based generator



Karras, Tero, et al. "A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR 2019."

StyleGAN: Style-based transfer results

Karras, Tero, et al. "A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR 2019."

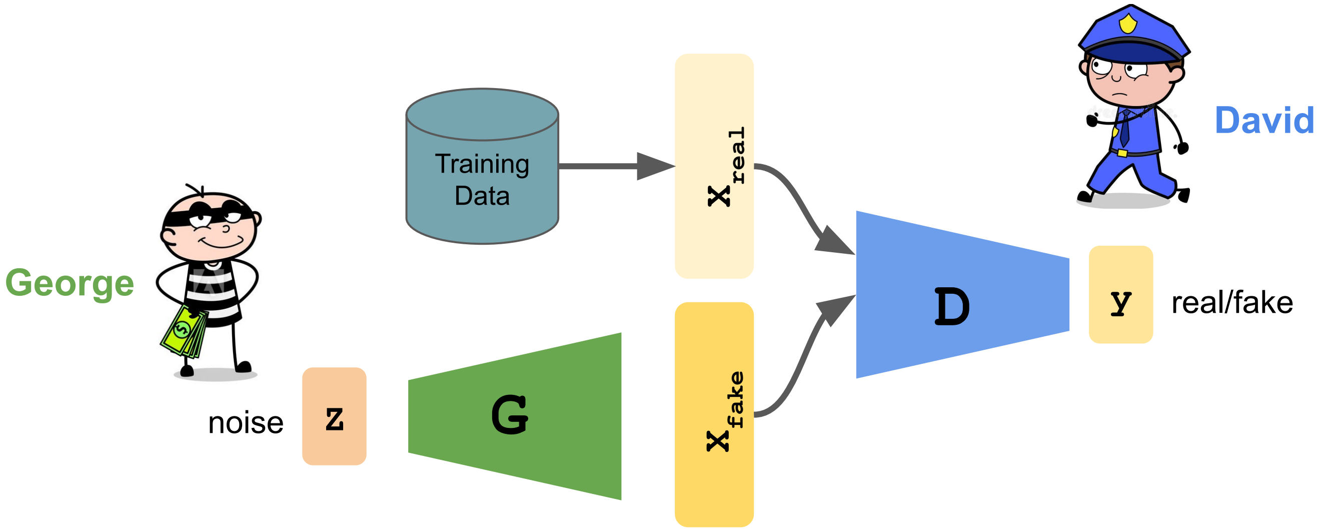
Image2StyleGAN

Ryan Gosling ⇒ Barack Obama

Oprah Winfrey ⇒ Scarlett Johansson

Summary: Generative Adversarial Network

- GANs are “*the most interesting idea in the last ten years in ML*” -- Yann LeCun
- GANs generate *realistic* images that are ***indistinguishable*** from real images
- The *dynamics* of adversaries **George** and **David** are still not *well* understood
- There are such a variety of GANs out there and still growing
- This hopefully gives you an introduction and perhaps a desire to learn more!



Bonus Content

DCGANs: Implementation

```
codings_size = 100

generator = keras.models.Sequential([
    keras.layers.Dense(7 * 7 * 128, input_shape=[codings_size]),
    keras.layers.Reshape([7, 7, 128]),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(64, kernel_size=5, strides=2, padding="same",
                                 activation="selu"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(1, kernel_size=5, strides=2, padding="same",
                                 activation="tanh")
])
discriminator = keras.models.Sequential([
    keras.layers.Conv2D(64, kernel_size=5, strides=2, padding="same",
                        activation=keras.layers.LeakyReLU(0.2),
                        input_shape=[28, 28, 1]),
    keras.layers.Dropout(0.4),
    keras.layers.Conv2D(128, kernel_size=5, strides=2, padding="same",
                        activation=keras.layers.LeakyReLU(0.2)),
    keras.layers.Dropout(0.4),
    keras.layers.Flatten(),
    keras.layers.Dense(1, activation="sigmoid")
])
gan = keras.models.Sequential([generator, discriminator])
```