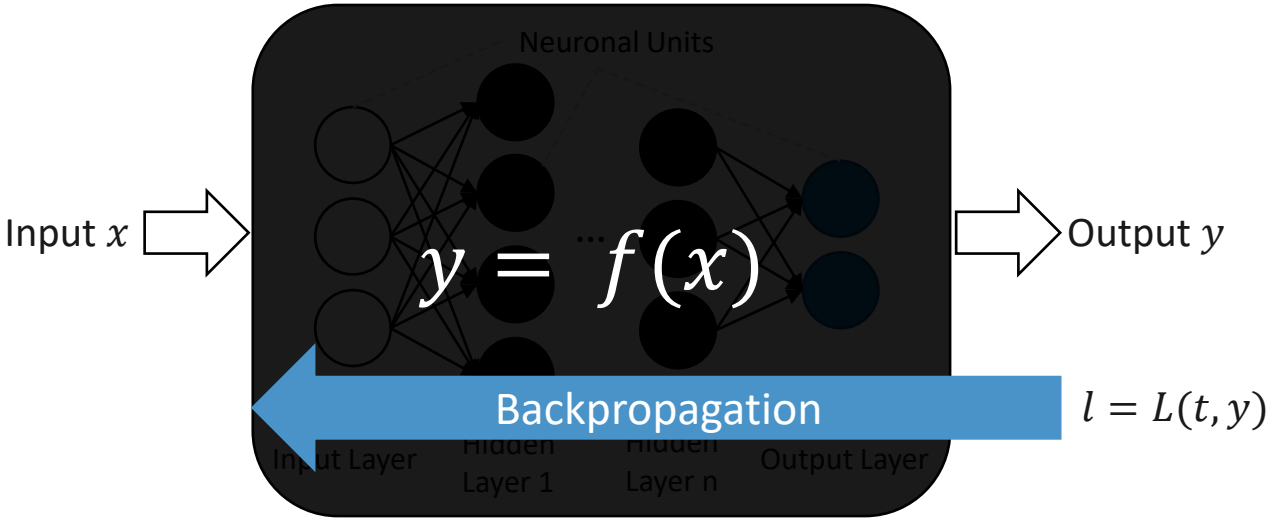
L12 Generative Adversarial Networks (GANs)

B. Stuhr, J. Haselberger, D. Schneider

Data Science & Artificial Intelligence

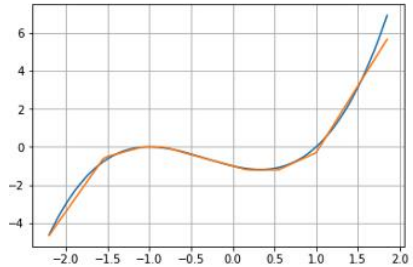
Summer Term 2022

Recap: Neural Nets are learnable Functions



Universal Approximator

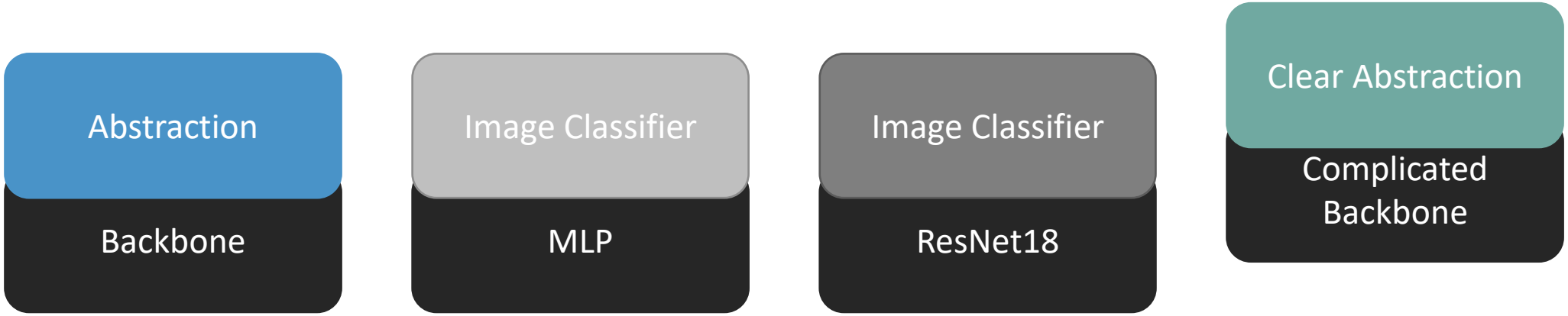
- Theoretically, there is a neural network that can approximate any (continuous) function [Cybenko1989] [Hornik1991]
- Theoretical foundation for why neural networks work




```
def feed_forward(input):
    Zh = np.dot(input, Wh)
    Ah = relu(Zh + Bh)
    Zo = np.dot(Ah, Wo)
    Ao = Zo + Bo
    return Ao
```

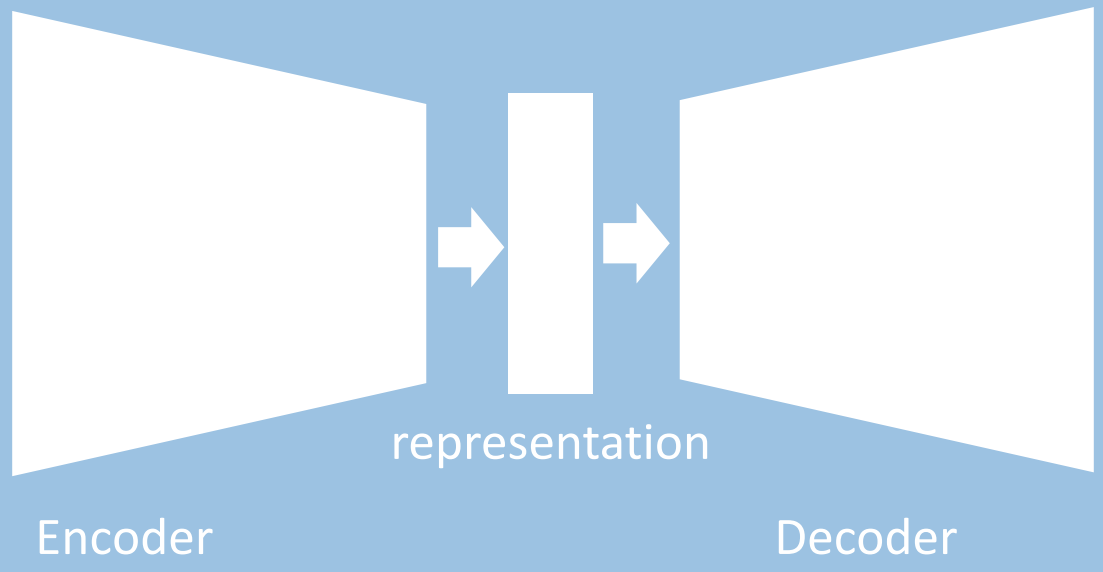
Approximating a Function with a Neural Network [Link]

Recap: Abstractions and Backbones



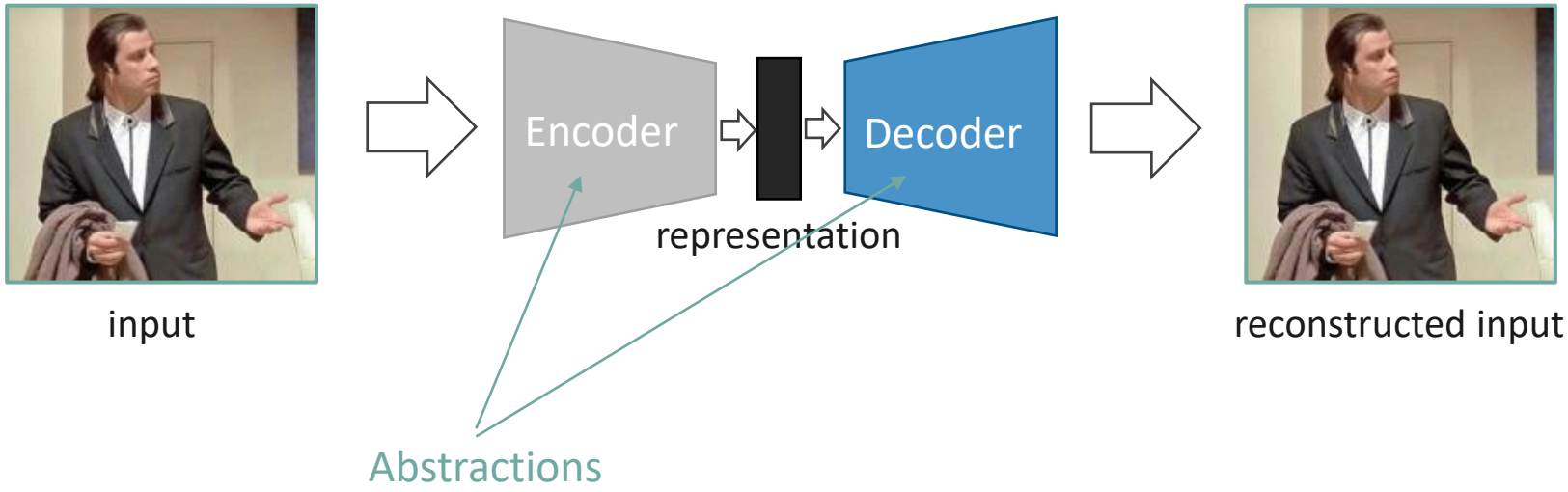
Abstraction: Describes the functionality or output of the block

Backbone: Concrete network used in a part of the DL architecture



L12.1 Recap: Autoencoders

General Architecture of an Autoencoder



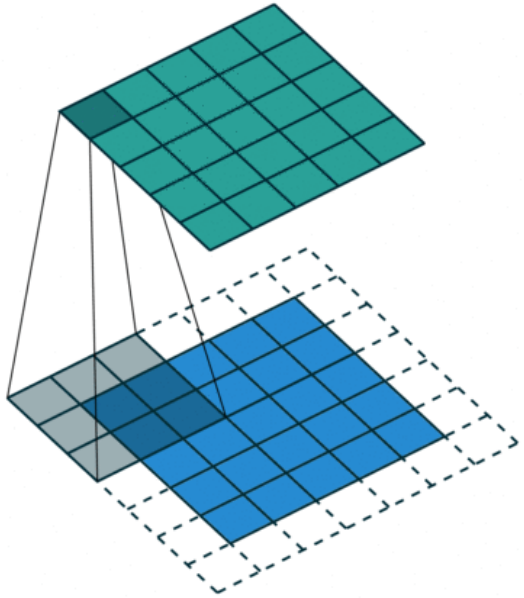
Encoder: converts the input into a different representation
Decoder: converts the representation back to the original input

Unsupervised Learning: No labels required (t=x)

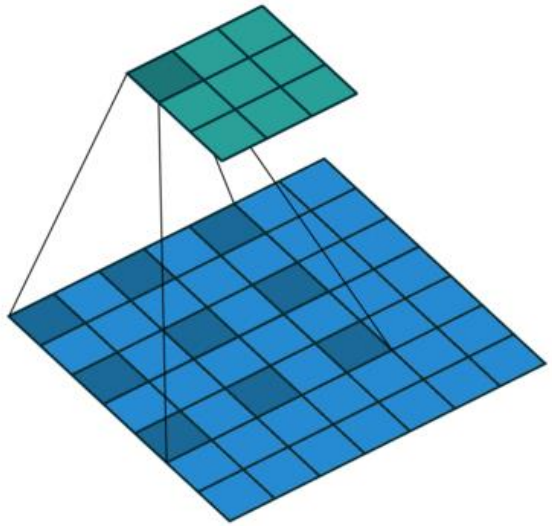
- Trained such, that the representations have “nice” properties
- Different kinds of Autoencoder aim to achieve different kinds of properties

Important Building Blocks: Convolutions/Upscaling

Downscaling (Encoder/Discriminator)

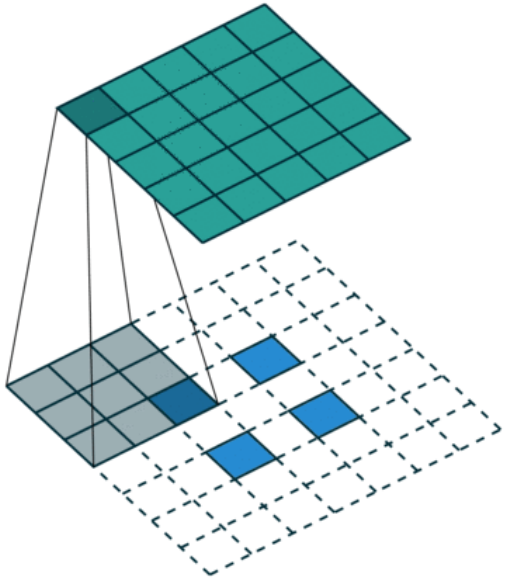


Convolution

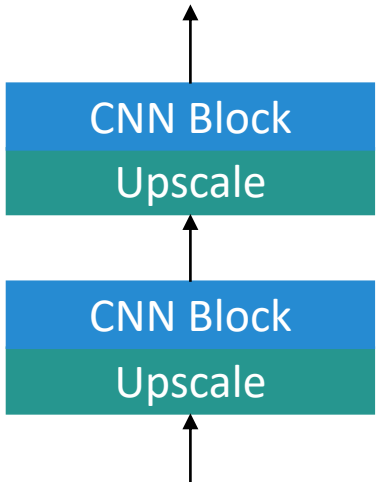


Dilated/Atrous Convolution

Upscaling (Decoder/Generator)



Transposed/De-
/Up-/fractionally
strided Convolution



Upscaling + CNN Block

E.g. Nearest-neighbor
interpolation + cov2d

Autoencoders: Losses

- Mean Squared Error (MSE): $L_{MSE}(t, y) = \frac{1}{N}\sum_{i}^{N}(t_i - y_i)^2$

MSE ( , )

input reconstructed input

- L1 Loss: $\text{L1}(t, y) = \frac{1}{N}\sum_{i}^{N}|t_i - y_i|$:
- Binary Cross Entropy Loss
- ❖ And many other Losses!

! **Variational Autoencoders (VAEs):** Latent Space (space of the representation) is forced to resemble a distribution
- Better latent space for image generation
- Sample a representation from the latent space

Autoencoders: Applications

- Dimensionality Reduction
- Feature Extraction
- Image Compression
- Superresolution
- Image Restoration
- Image Generation
- Sequence to Sequence Prediction
- …

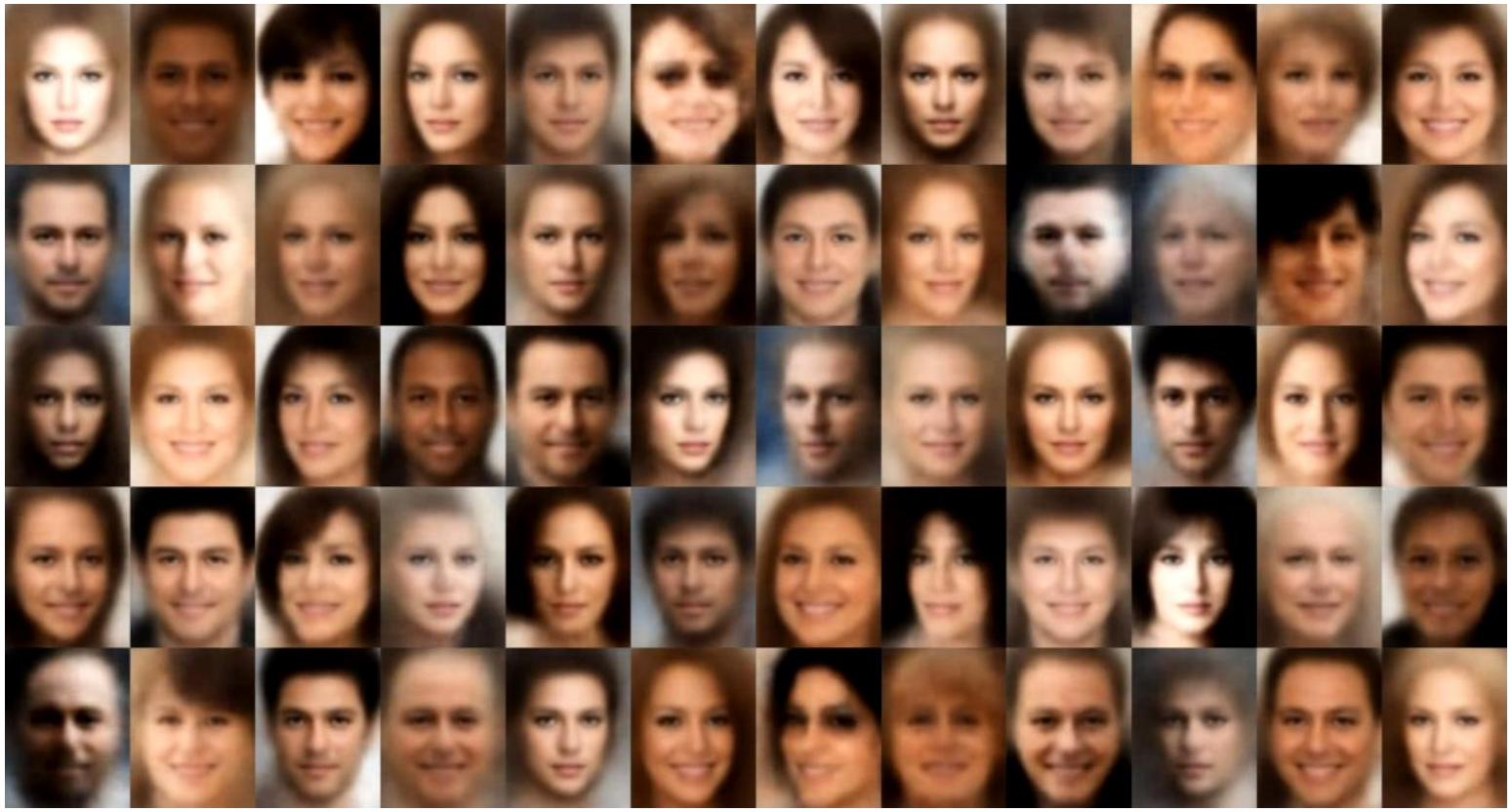
IMAGE NOISE REDUCTION

Before After

IMAGE COLORING

Before After

But: Generated images are often blurry



Mean Squared Error regresses to the mean
→ Blurry Images

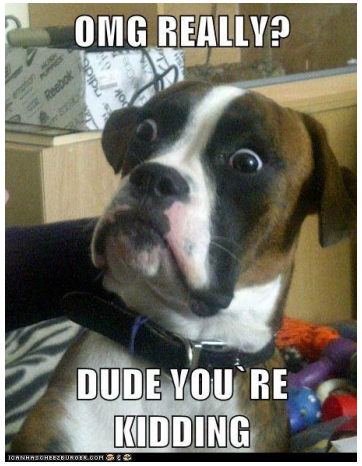
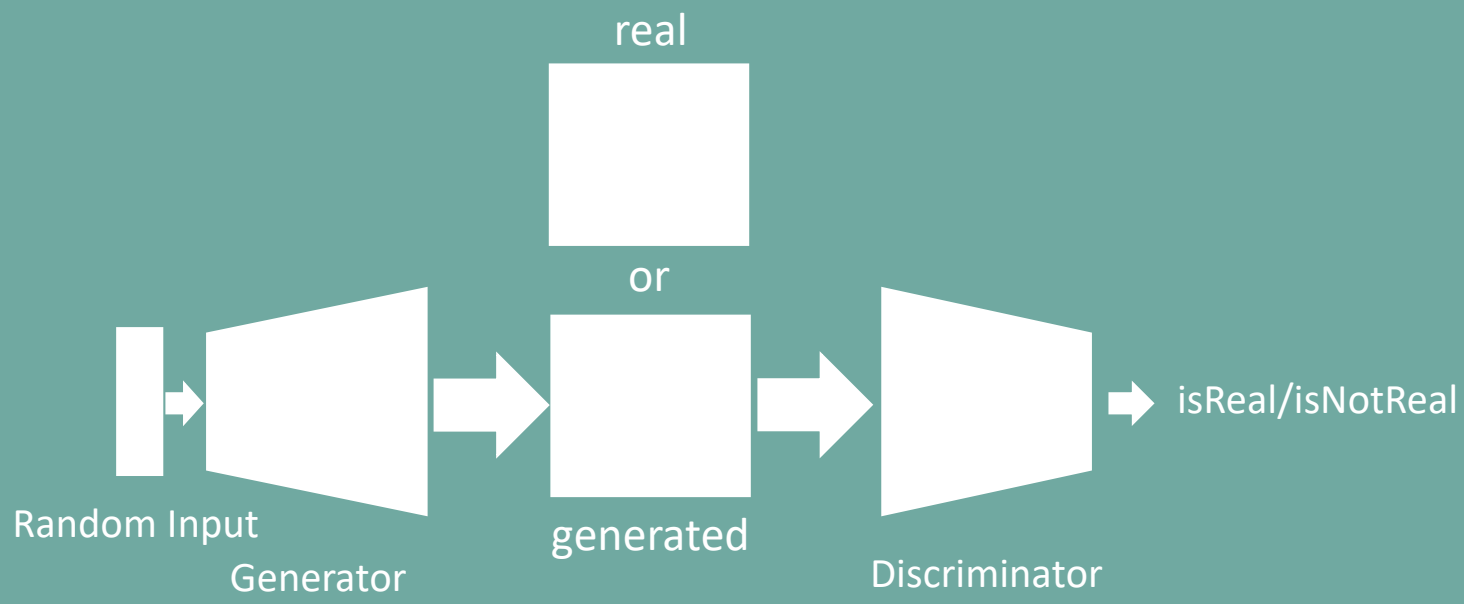
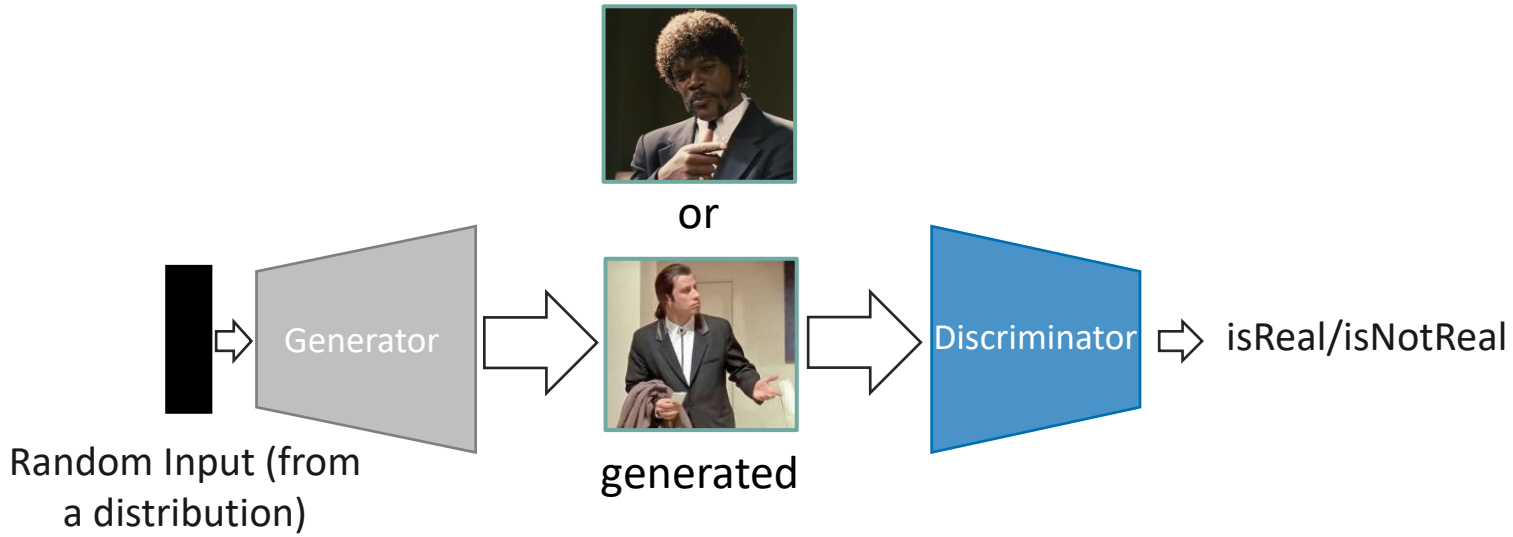


Image from [Link]



L12.2 Generative Adversarial Networks (GANs)

GANs: Learn the Loss Function with a Discriminator



Generator (G): Tries to generate a realistic looking image
Discriminator (D): Tries to distinguish real images from generated images

Unsupervised Learning: No labels required
- Trained such, that generator and discriminator learn in balance
- Different kinds of GANs have different training schemes

Source: Original Paper from 2014 of Ian Goodfellow

GANs original Loss

Discriminator (D) Loss:

Classification Loss on data (images)

Real Data (label 1)

Fake Data (label 0)

maximize $L^{(D)} = \mathbb{E}_{x \sim p_{data}} \log(D(x)) + \mathbb{E}_z \log(1 - D(G(z)))$

Generator (G) Loss:

minimize $L^{(G)} = L^{(D)}$

The generator can't directly affect the real data part of the term, therefore:

minimize $L^{(G)} = \mathbb{E}_z \log(1 - D(G(z)))$

Optimization strategy: Minimax Game

- Two-player, turn-based Game

D tries maximize the probability of being correct in real/fake classification
G tries to minimize the probability that D is correct by generation realistic examples

- D provides gradients for G

❖ Formula derives from the cross-entropy between the real and generated distributions (often used in implementations)

Source: Original Paper from 2014 of Ian Goodfellow

GANs original Pseudocode

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Discriminator Training

Generator Training

❖ Other Training Schemes are possible (and used) as well!

Source: Original Paper from 2014 of Ian Goodfellow

GANs: Non-Saturating Loss

Figure 5.5. A sketch of what the hypothesized relationships are meant to look like in theory. The y-axis is the loss function for the Generator, whereas D(G(z)) is the Discriminator's "guess" for the likelihood of the generated sample. You can see that Minimax (MM) stays flat for too long, thereby giving the Generator too little information—the gradients vanish.

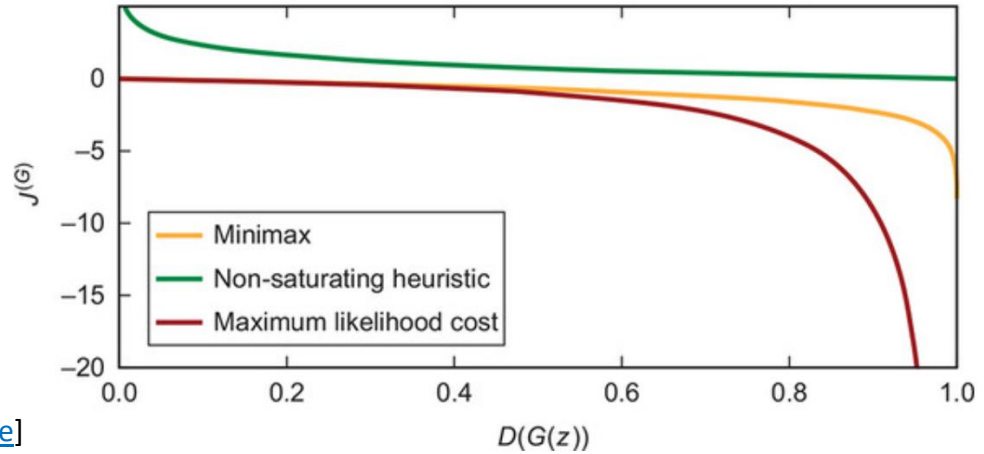


Image from [here]

When D is winning, we're at the left side of the graph, since D outputs the probability of the sample being from the *true* data distribution.

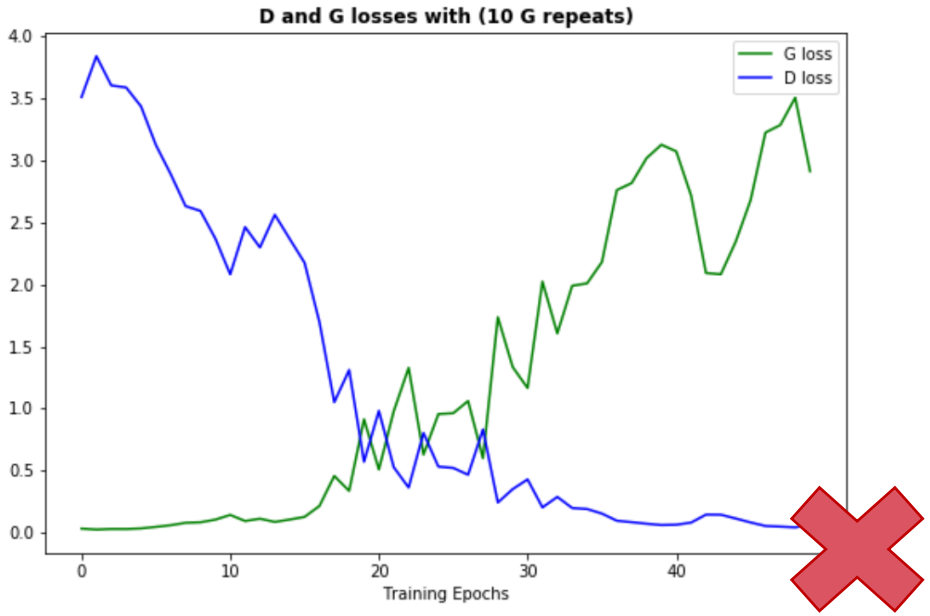
Minimax loss can cause the GAN to get stuck in early training stages when the discriminator's job is too easy

Heuristic Hack:

- G maximizes the log probabilities of D being false instead of minimizing the log probabilities of D being correct
- G can still learn, even when D is 100% correct (Loss is 0)

Source: Original Paper from 2014 of Ian Goodfellow

Example GAN Training Curves



- There are GAN Training Objectives that are designed to converge as well!

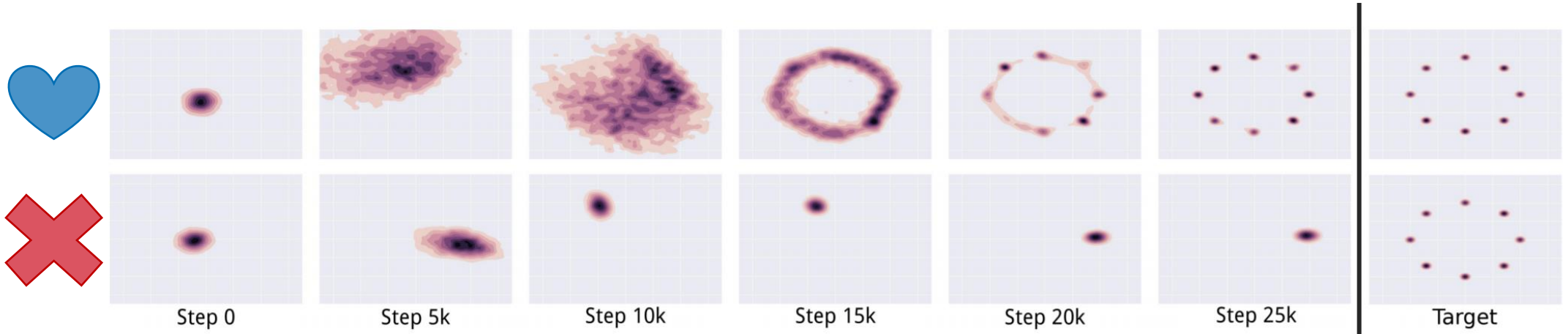
Many other Losses and Tweaks proposed!

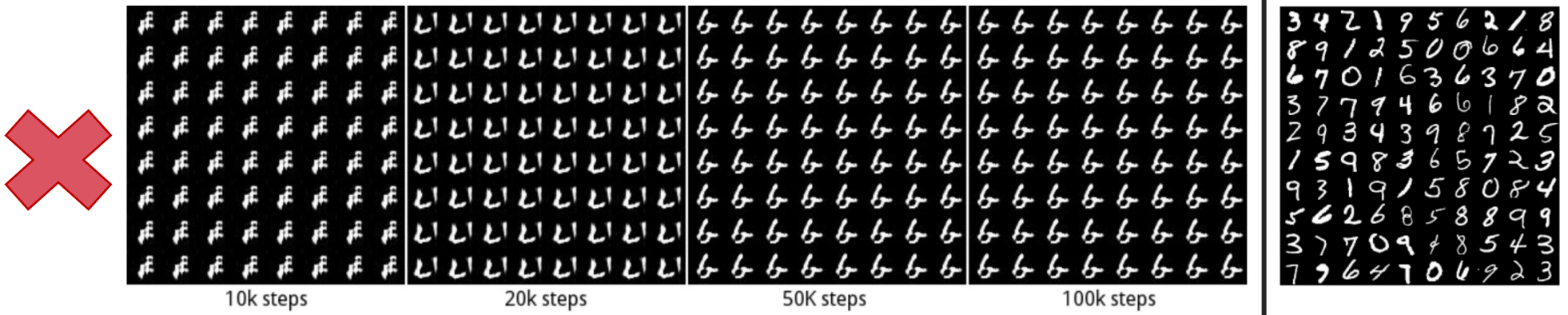
GAN	Discriminator Loss	Generator Loss
MM GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{GAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
NS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{NSGAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{NSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[\log(D(\hat{x}))]$
WGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} = -\mathbb{E}_{x\sim p_d}[D(x)] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
WGAN GP	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGANGP}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_g}[(\|\nabla D(\alpha x + (1-\alpha\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGANGP}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
LS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{LSGAN}} = -\mathbb{E}_{x\sim p_d}[(D(x)-1)^2] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{LSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[(D(\hat{x}-1))^2]$
DRAGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{DRAGAN}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_d+\mathcal{N}(0,c)}[(\|\nabla D(\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{DRAGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
BEGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{BEGAN}} = \mathbb{E}_{x\sim p_d}[\|x - \mathrm{AE}(x)\|_1] - k_t\mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x} - \mathrm{AE}(\hat{x})\|_1]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{BEGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x} - \mathrm{AE}(\hat{x})\|_1]$

Large-scale study shows that **Hyperparameter Optimization seems more important than loss functions**

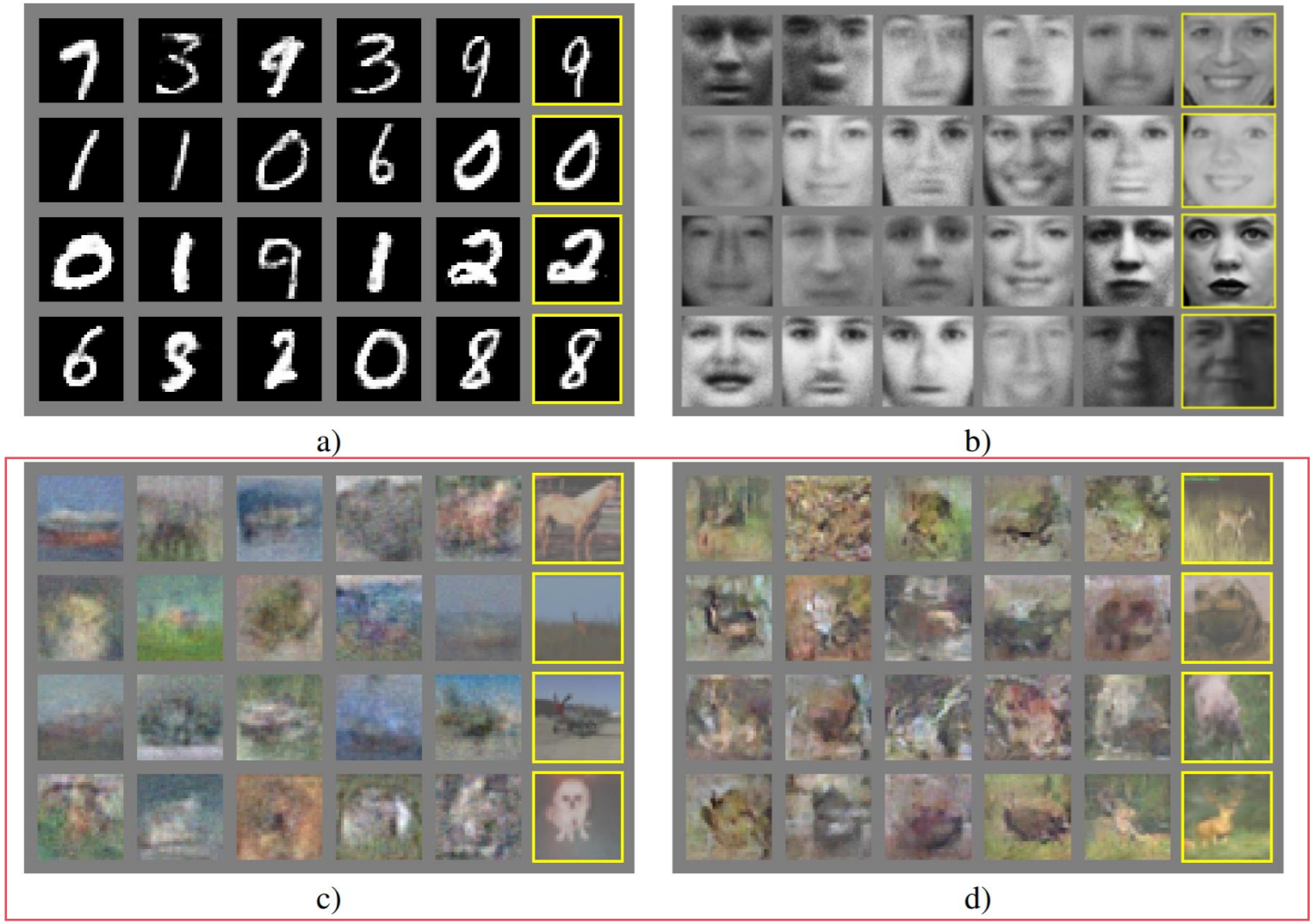
- Study: Are GANs Created Equal? A Large-Scale Study

Vanilla GANs struggle from Mode Collapse





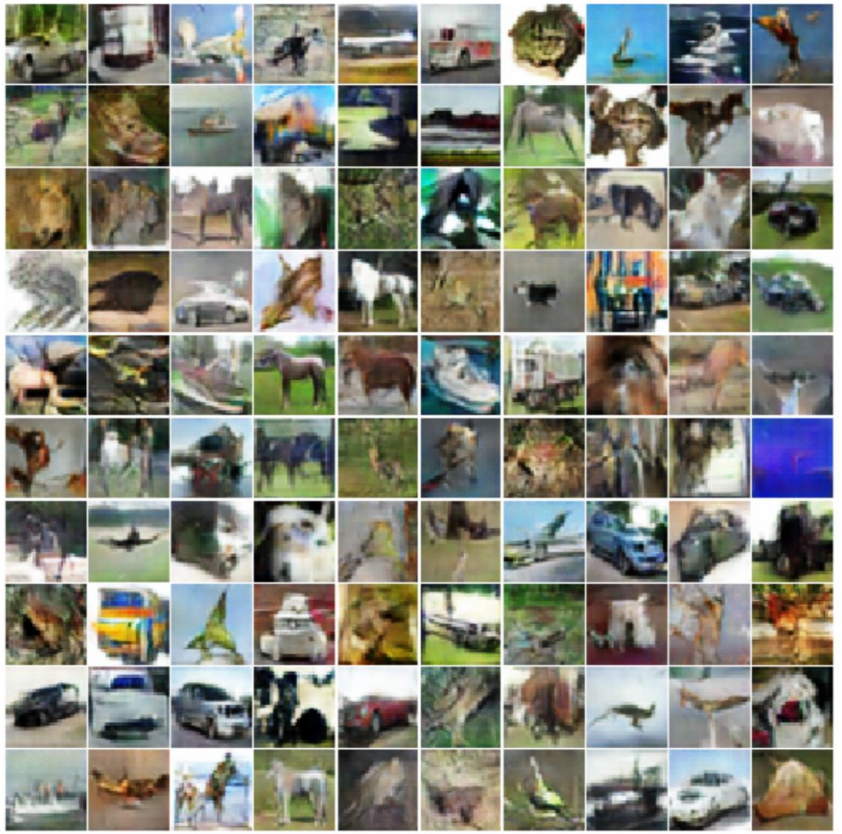
Vanilla GANs struggle with Global Structure



Source: Original Paper from 2014 of Ian Goodfellow

Improved Techniques for Training GANs Paper (2016)

Minibatch Discriminator
- Looks at multiple examples in combination



Other Improvements: Build up on other work
- E.g., DCGAN (GAN with convolutional layers)

Source: [Improved Techniques for Training GANs]

But: High-Res. Images are still difficult

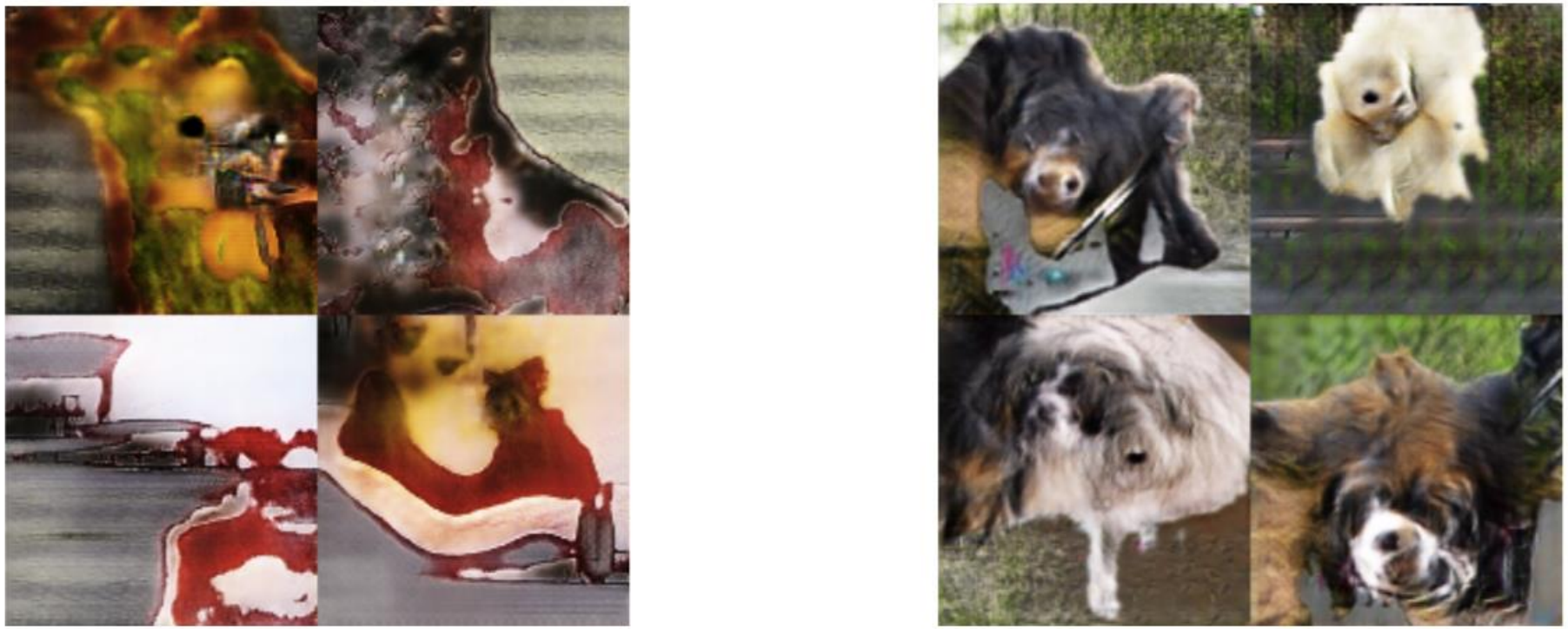
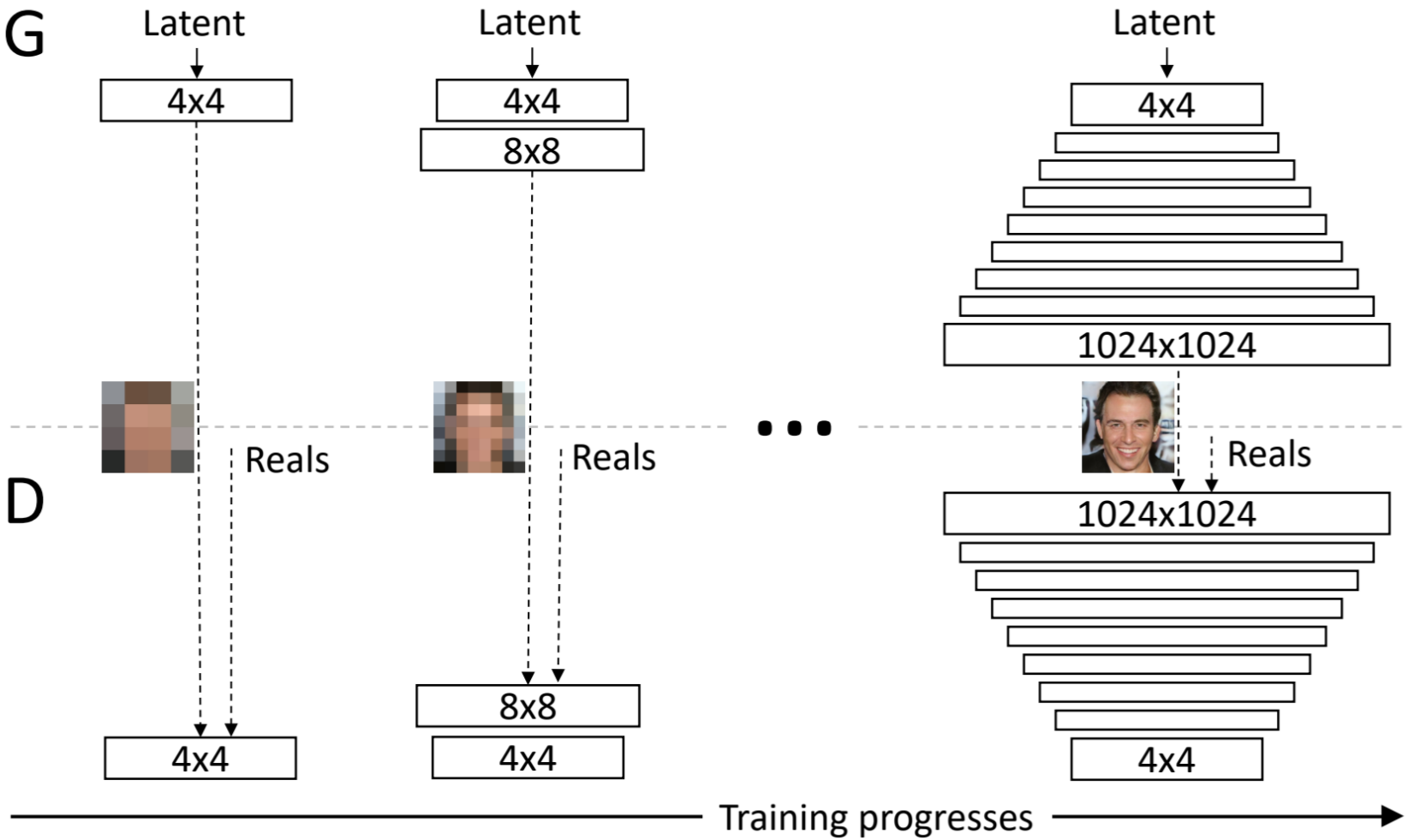


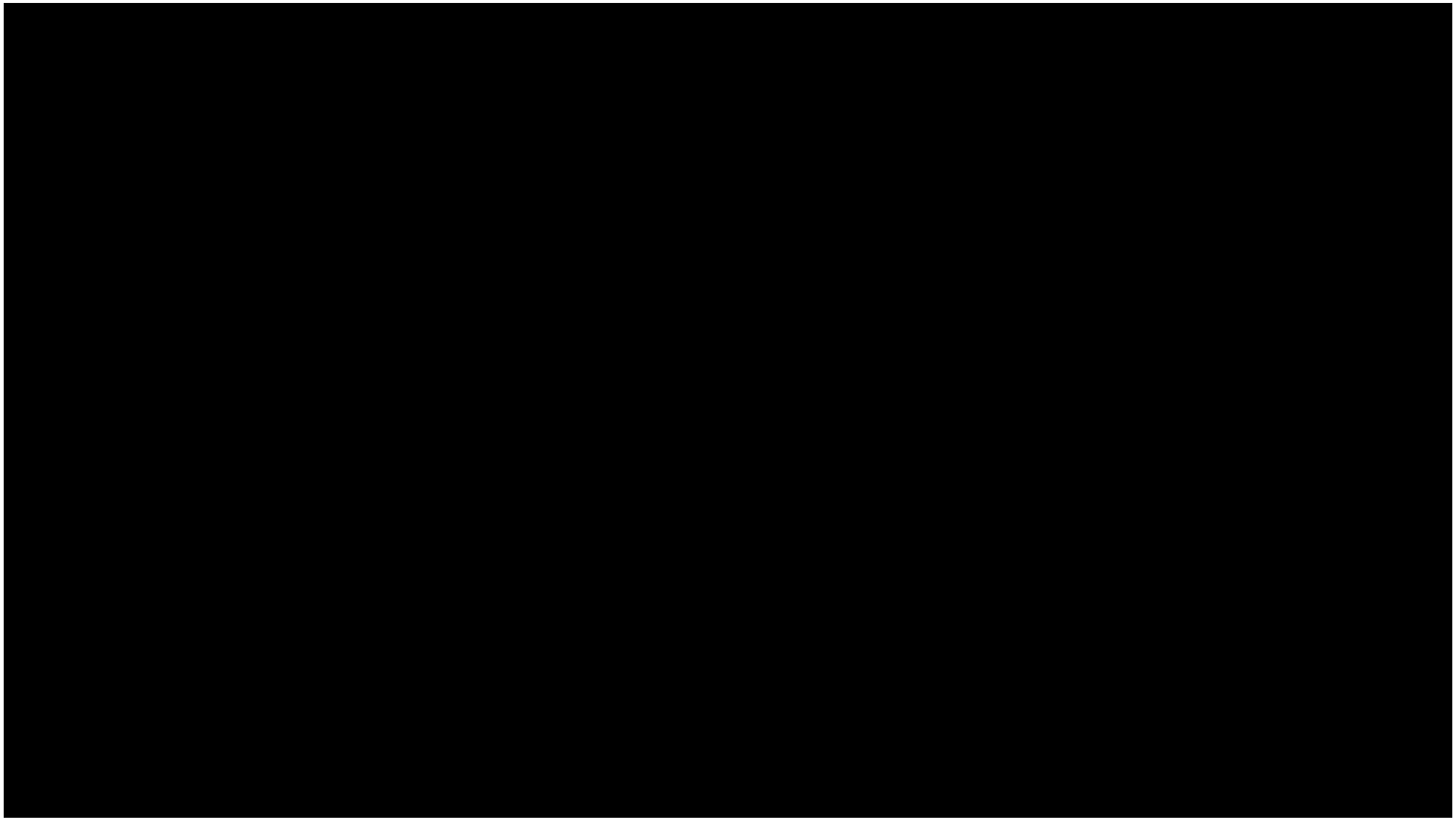
Figure 6: Samples generated from the ImageNet dataset. *(Left)* Samples generated by a DCGAN. *(Right)* Samples generated using the techniques proposed in this work. The new techniques enable GANs to learn recognizable features of animals, such as fur, eyes, and noses, but these features are not correctly combined to form an animal with realistic anatomical structure.

One Solution: Progressive Growing (2017)



Source: [Progressive Growing of GANs for Improved Quality, Stability, and Variation](#) (Nvidia)

Progressive Growing Training Process



More results can be found here

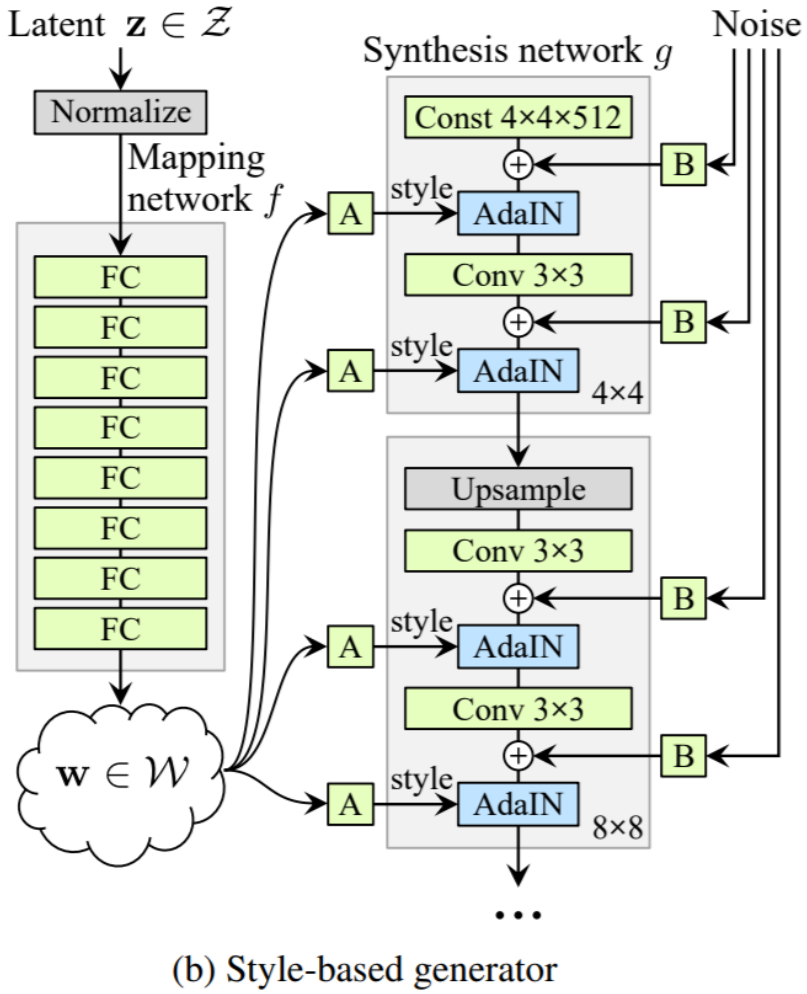
Progressive Growing Results: Global Structure

Mao et al. (2016b) (128 × 128) Gulrajani et al. (2017) (128 × 128) Our (256 × 256)

Nvidia: Progressive Growing

More results can be found here

StyleGAN (2019)



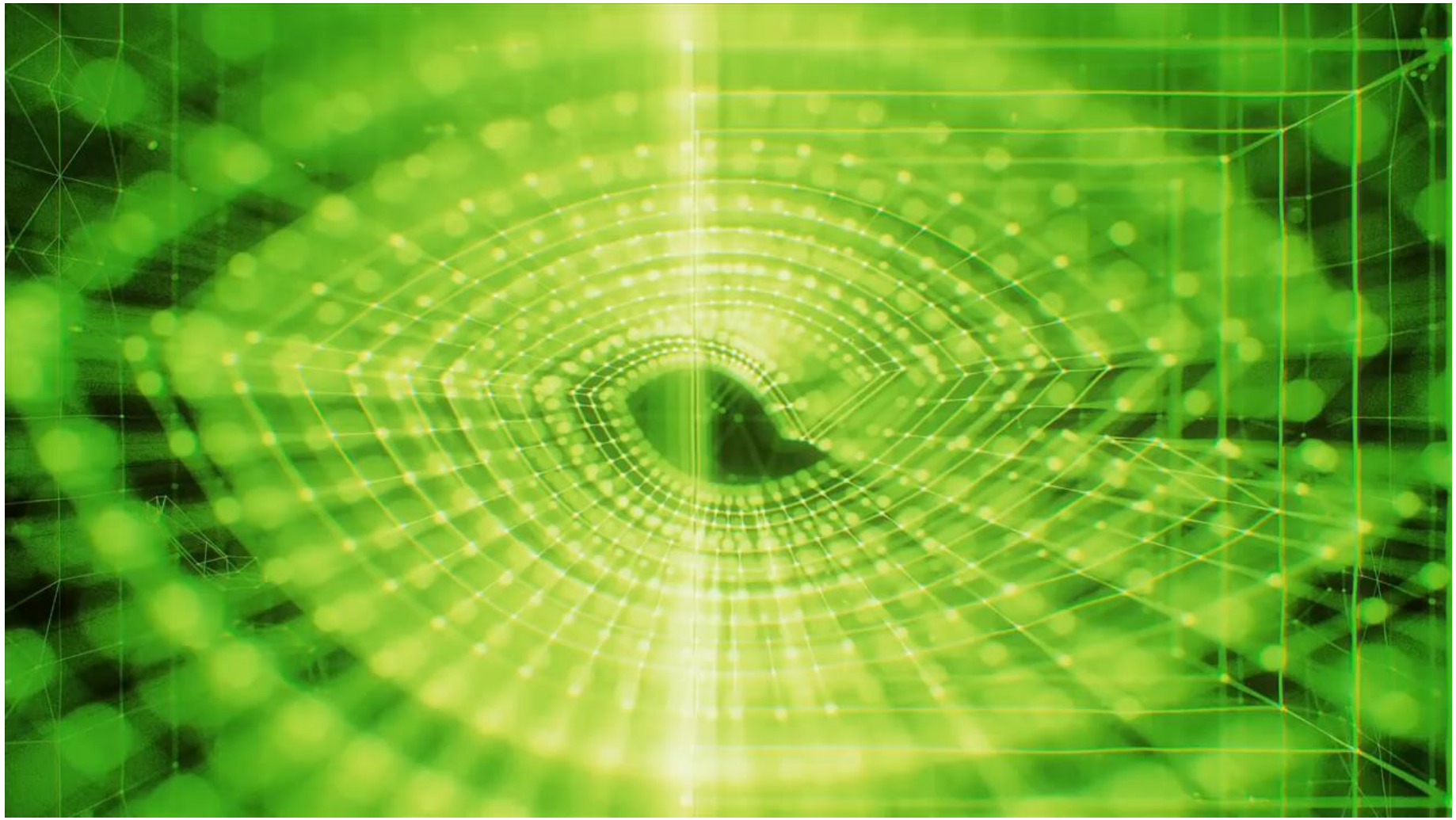
(b) Style-based generator

Feeds the input vector as style to different layers

- Separation of high-level attributes
  - E.g., pose and identity when trained on human faces

Source: [A Style-Based Generator Architecture for Generative Adversarial Networks](#) (Nvidia)

StyleGAN Results



More results can be found here

But: Images contain Droplets

Figure 1. Instance normalization causes water droplet -like artifacts in StyleGAN images. These are not always obvious in the generated images, but if we look at the activations inside the generator network, the problem is always there, in all feature maps starting from the 64x64 resolution. It is a systemic problem that plagues all StyleGAN images.

Source: Analyzing and Improving the Image Quality of StyleGAN (Nvidia)

But: Images contain “Phase” Artifacts

Figure 6. Progressive growing leads to “phase” artifacts. In this example the teeth do not follow the pose but stay aligned to the camera, as indicated by the blue line.

Source: Analyzing and Improving the Image Quality of StyleGAN (Nvidia)

StyleGANv2

Revisits the architecture and training of StyleGAN:

No instance norm
- Updates blocks that fuse the style into the model
- Prevents droplets

No progressive growing
- Progressively grown generator appears to have a strong location preference for detail
  - E.g., teeth

Generate your own image here:
https://thispersondoesnotexist.com/

More results can be found here

Source: Analyzing and Improving the Image Quality of StyleGAN (Nvidia)

StyleGANv3

Revisits architecture and training of StyleGAN2:

GANs like StyleGANv2 depend on absolute pixel coordinates in an unhealthy manner

- ➢ Interpreting all signals in the network as continuous
  - E.g., with Fourier features

Source: Alias-Free Generative Adversarial Networks (Nvidia)

Further Reading

- Deep Learning Book: https://www.deeplearningbook.org/
- The newest Papers (there are way too many)
- YouTube Lectures of other Unis, e.g.:
  - MIT: https://www.youtube.com/watch?v=5tvmMX8r_OM&list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI
  - Stanford: https://www.youtube.com/watch?v=vT1JzLTH4G4&list=PLC1qU-LWwrF64f4QKQT-Vg5Wr4qEE1Zxk
  - TUM: https://www.youtube.com/watch?v=QLOocPbztuc&list=PLQ8Y4kIIbzy_OaXv86lfbQwPHSomk2o2e
  - TUM (advanced): https://www.youtube.com/watch?v=Bt5O1HjT9cI&list=PLog3nOPCjKBkngkkF552-Hiwa5t_ZeDnh
  - Tübingen: https://www.youtube.com/watch?v=OCHbm88xUGU&list=PL05umP7R6ij3NTWIdtMbfvX7Z-4WEXRqD
  - ….

www.hs-kempten.de/ifm