Progressive Growing of GANs

Joris ten Tusscher

February 18, 2019

The paper

- "PROGRESSIVE GROWING OF GANS FOR IMPROVED QUALITY, STABILITY, AND VARIATION"[1]
- Tero Karras, Timo Aila, Samuli Laine, Jaakko Lehtinen
- NVIDIA research
- Presented in 2018 at ICLR (International Conference on Learning Representations)

Goal

- Generate new, realistic samples from a high-dimensional data distribution.
- **Example**: modelling the distribution of real images, and use that to generate new, real looking images.

Approaches

1. Autoregressive (AR) models
2. Variational Autoencoders (VAE)
3. Generative Adversarial Network (GAN)

Approaches

1. **Autoregressive (AR) models**
2. Variational Autoencoders (VAE)
3. Generative Adversarial Network (GAN)

Autoregressive (AR) models

- Regression function: $X \rightarrow W \rightarrow \hat{y}$
  - X: model input (list of independent variables x_1, x_2, ..., x_n).
  - W: model coefficients w_0, $w_1, ..., w_n$.
  - $\hat{y}$: prediction
  - e.g. $\hat{y} = w_0 + w_1 x_1 + w_2 x_2^2 + w_3 x_3^3 + ...$
- Example: predict colloquium turn-out $\hat{y}$ based on time of day (x_0), day of week (x_1), number of subsequent colloquium talks (x_2).
- Autoregression:
  - X consists of observations from n previous time steps.
  - $\hat{y}$ is usually renamed to $\hat{x}_{n+1}$.

PixelCNN (AR image generator model)^2 (2016)

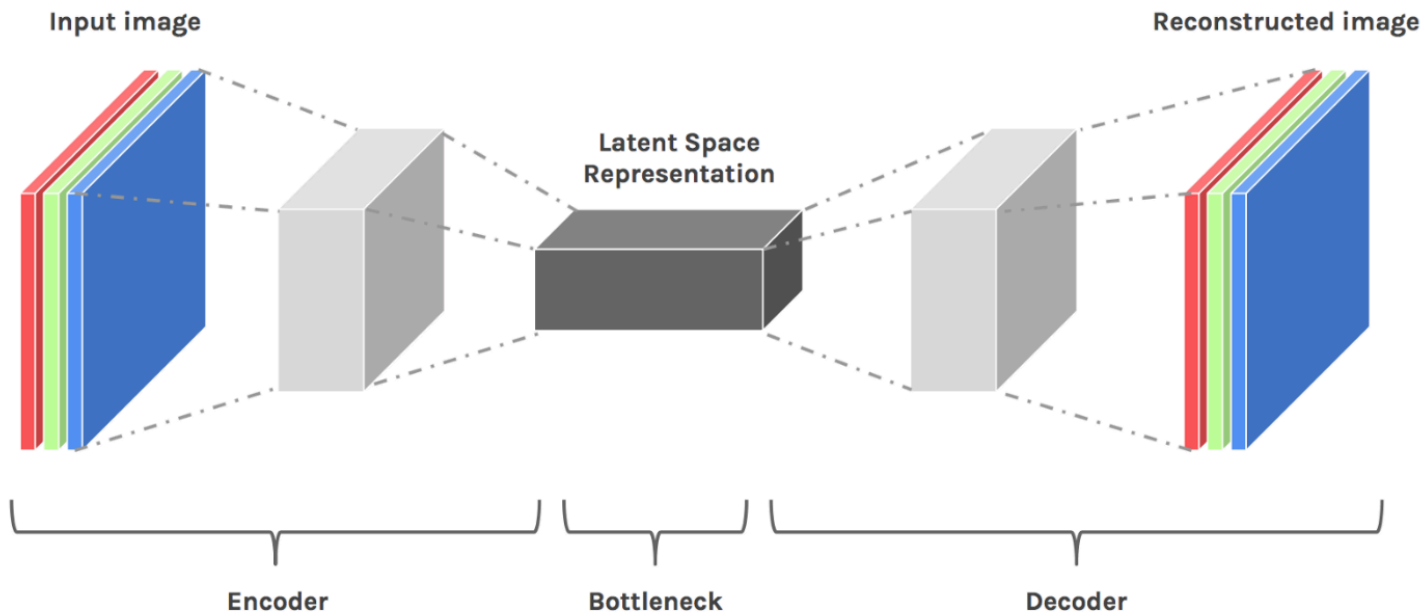
- model joint distribution of pixels over an image X as product of conditional distributions:
  $p(X) = \prod_{i=1}^{n^2} p\left(x_i | x_1, \ldots, x_{i-1}\right)$
- Gives sharp images, but obviously **very** slow (one neural network per pixel!).

PixelCNN (AR image generator model)^2 (2016)

- model joint distribution of pixels over an image X as product of conditional distributions:
  $p(X) = \prod_{i=1}^{n^2} p\left(x_i | x_1, \ldots, x_{i-1}\right)$
- Gives sharp images, but obviously **very** slow (one neural network per pixel!).
- Add extra dependencies, e.g. **latent representation** F of some random input image:
  $p(X) = \prod_{i=1}^{n^2} p\left(x_i | x_1, \ldots, x_{i-1}, \mathbf{F}\right)$

Latent representations

- Latent space is intermediate representation in autoencoder.
- Autoencoder:
  - $\phi : X \to F$
  - $\psi : F \to X$
  - $\phi, \psi = \underset{\phi,\psi}{\arg\min} \|X - (\psi \circ \phi)X\|^2$
  - Visualised:



PixelCNN (AR image generator model)^2 (2016)

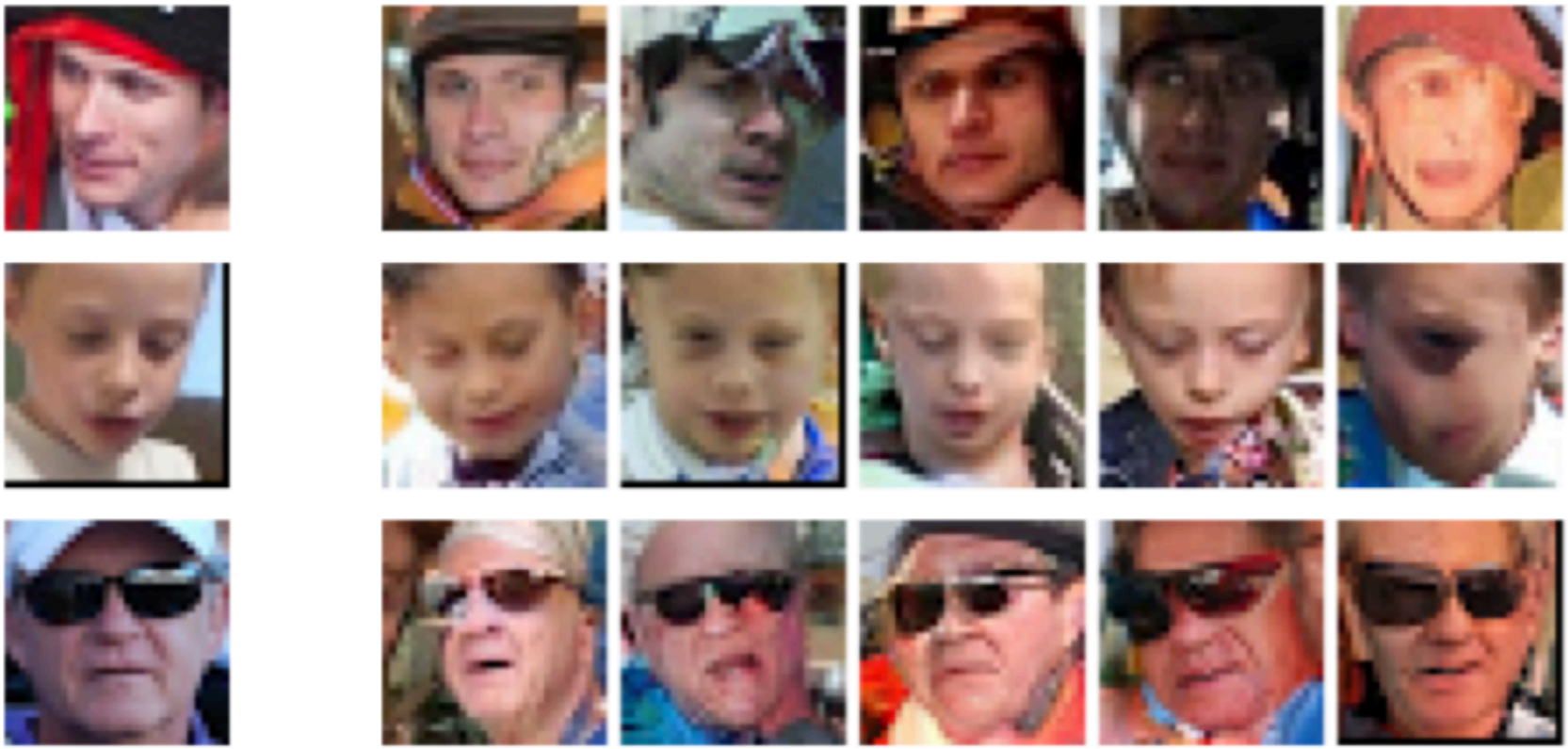


Figure 1: Images generated using PixelCNN^2 with extra constraint from latent space.

Approaches

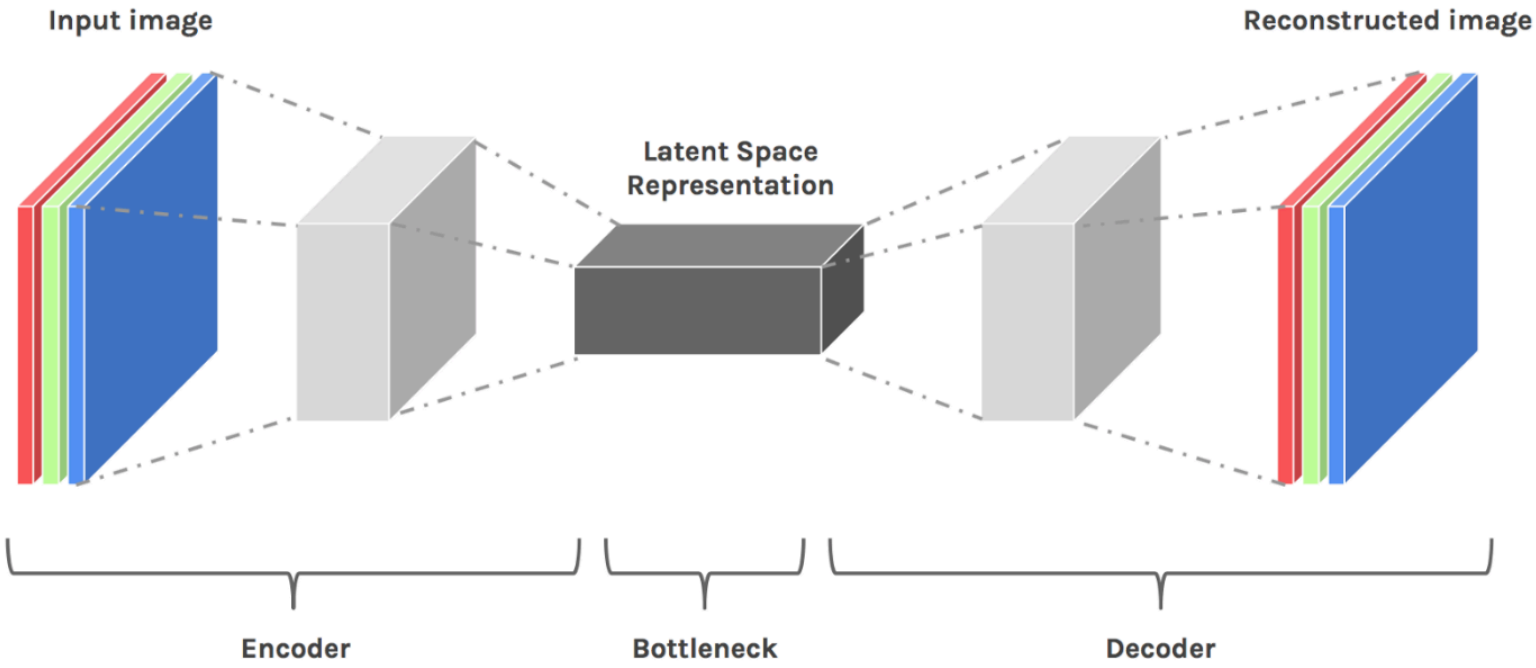
1. **Autoregressive (AR) models**
2. Variational Autoencoders (VAE)
3. Generative Adversarial Network (GAN)

Approaches

1. Autoregressive (AR) models:
   - High quality
   - Extremely slow
2. **Variational Autoencoders (VAE)**
3. Generative Adversarial Network (GAN)

Variational Autoencoders (VAE)

- Recall the AE:



- Observation: we know nothing about the latent space distribution.
- Fix: force encoder to generate vectors that roughly follow certain distribution (e.g. isotropic multivariate $\mathcal{N}(\mu = 0, \sigma^2 = 1)$).

Variational Autoencoders (VAE)

- Fast generation: only 1 neural network
- Blurry output due to latent space distribution constraint

Approaches

1. Autoregressive (AR) models:
   - High quality
   - Extremely slow
2. **Variational Autoencoders (VAE)**
3. Generative Adversarial Network (GAN)

Approaches

1. Autoregressive (AR) models:
   - High quality
   - Extremely slow
2. Variational Autoencoders (VAE)
   - Blurry
   - Fast
3. **Generative Adversarial Network (GAN)**

Generative Adversarial Networks

- Proposed in 2014 by Goodfellow et al.[3]
- A GAN is a two player minimax game:
  - Generator **G** captures data distribution
  - Discriminator **D** discriminates between training data and data from G.
- G and D constantly try to beat each other.

On G and D

- Generator can be any arbitrary function $Z \rightarrow X$
  - Z: noise vector
  - X: output (in our context: image)
- Discriminator can be any arbitrary function $X \rightarrow P$
  - P: probability that X comes from real data and not G.

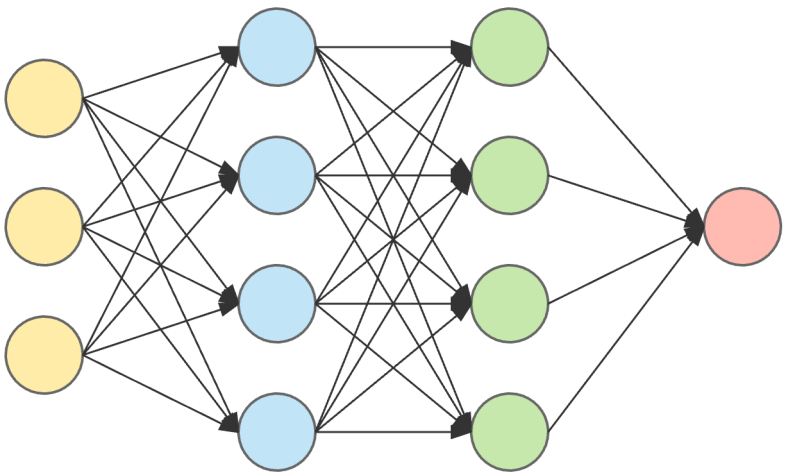
The minimax game

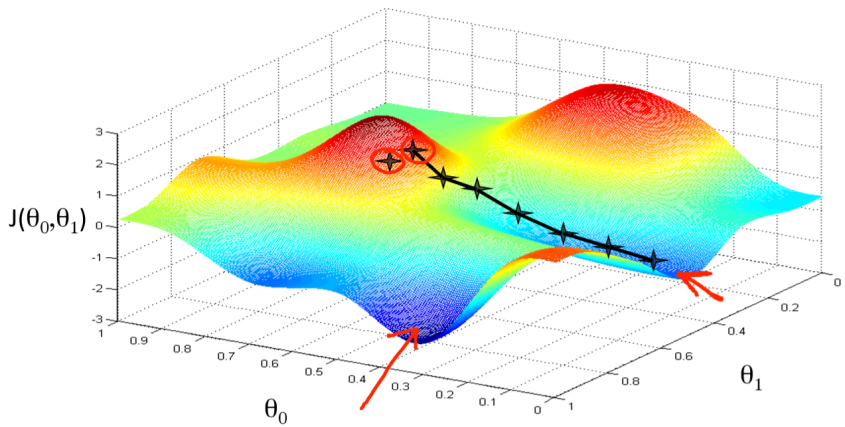
- For D: maximise probability that D predicts correct class for random inputs.
- For G: minimise $1 - D(G(Z))$.
- Formally: let's say that $V(D, G)$ is the valuation function of our minimax game, then:

$$\min_G \max_D V(D, G) = \mathbb{E}_{X \sim p_{data}(X)}[\log D(X)] + \mathbb{E}_{Z \sim p_Z(Z)}[\log(1 - D(G(Z)))]$$

How to train GANs?

- How to train NNs in general?
  - Realise that loss of NN is function $\mathcal{L}$ of network parameters θ
  - Calculate gradient $\nabla\mathcal{L}$
  - Move in opposite direction of $\nabla\mathcal{L}$
  - But don't actually do this.





Iteratively training a GAN:[3] training algorithm

1. Repeat until happy
   - 1.1 Build minibatch Z of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ using p_Z
   - 1.2 Build minibatch X of m data samples $\{x^{(1)}, \ldots, x^{(m)}\}$ using p_{data}
   - 1.3 Update D by ascending its stochastic gradient: $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D(x^{(i)}) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right]$
   - 1.4 Resample Z.
   - 1.5 Update G by descending its stochastic gradient: $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$

Iteratively training a GAN:[3] training algorithm visualisation

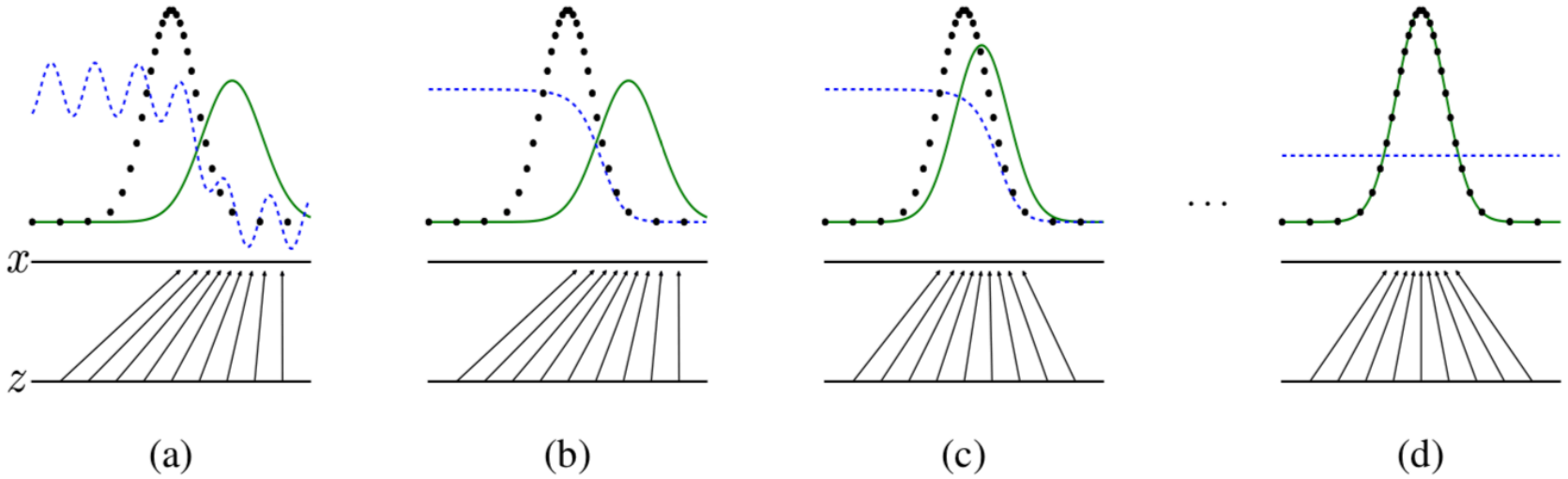


Figure 2: p_{data} (black), p_D (blue) and p_G (green). Horizontal lines are (partial) domains of Z and X. Arrows show G's mapping.

Iteratively training a GAN:[3] training algorithm visualisation

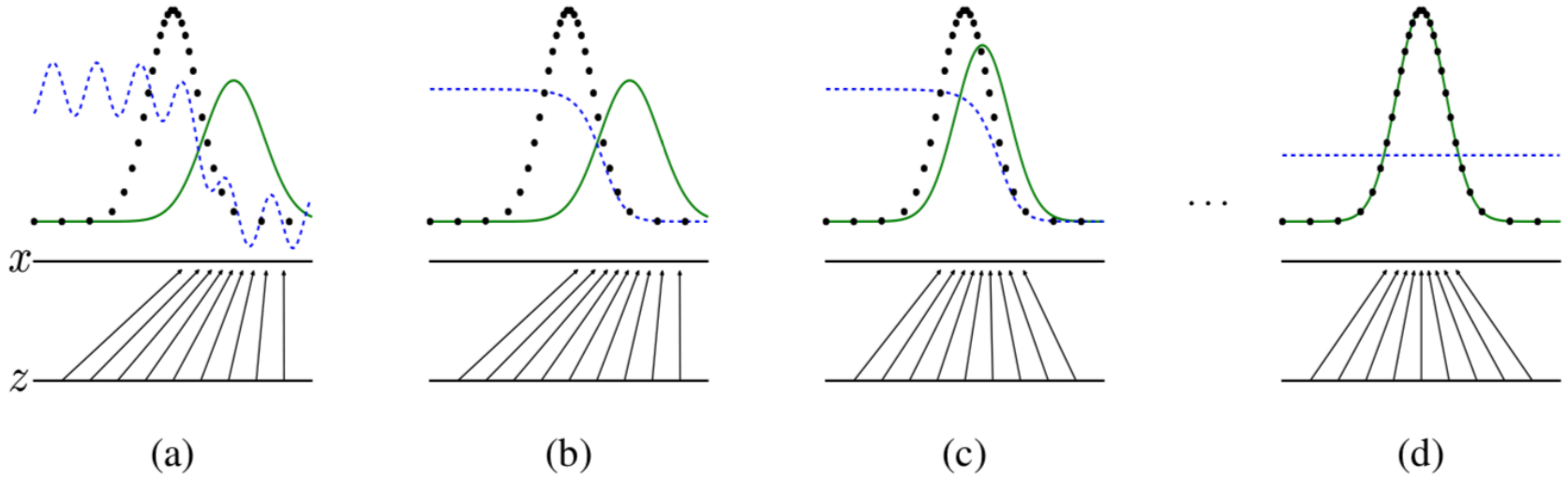


Figure 2: p_{data} (black), p_D (blue) and p_G (green). Horizontal lines are (partial) domains of Z and X. Arrows show G's mapping.

a) distributions at start of iteration.

b) updated p_D

c) updated p_G

d) The ideal post-training scenario: equilibrium, i.e. $p_{data} = p_G \wedge \forall X : D(X) = \frac{1}{2}$

‘Face generation’ using GANs

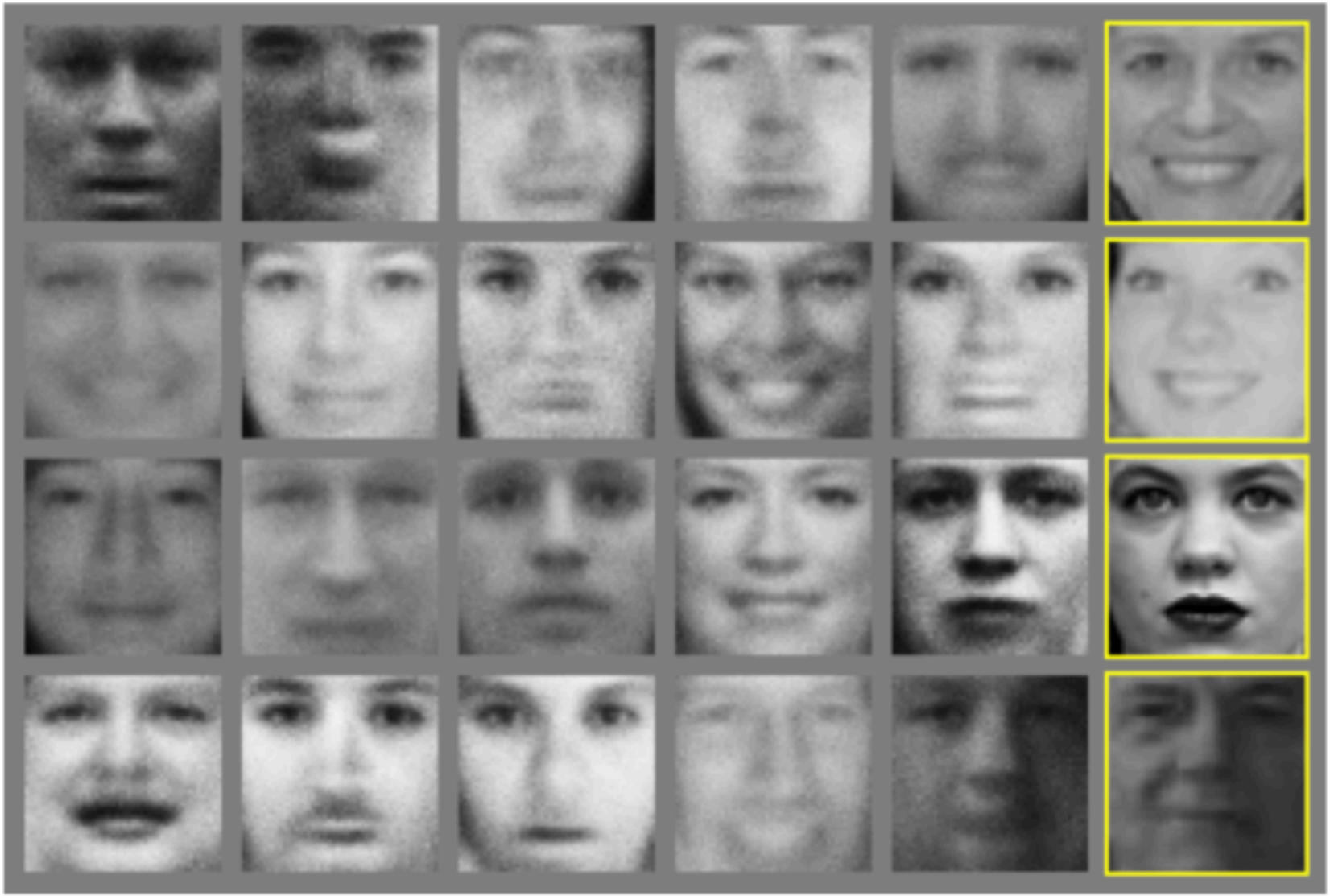


Figure 3: Faces generated using GANs in original paper[3]

Problems with GANs

- Observation: at first, the difference between p_G and p_{data} will be huge. Effect:

  $\nabla \mathcal{L}_G$ will be all over the place.
- Even more problematic for higher resolutions: initial difference between p_G and p_{data} will only grow.
- Idea: slowly increase resolution for both G and D.
  - G will first learn macro aspects of p_{data}, e.g. a face always has two eyes.
  - Add layers to G and D to slowly learn meso and micro aspects.

Progressive growing of GANs[1]

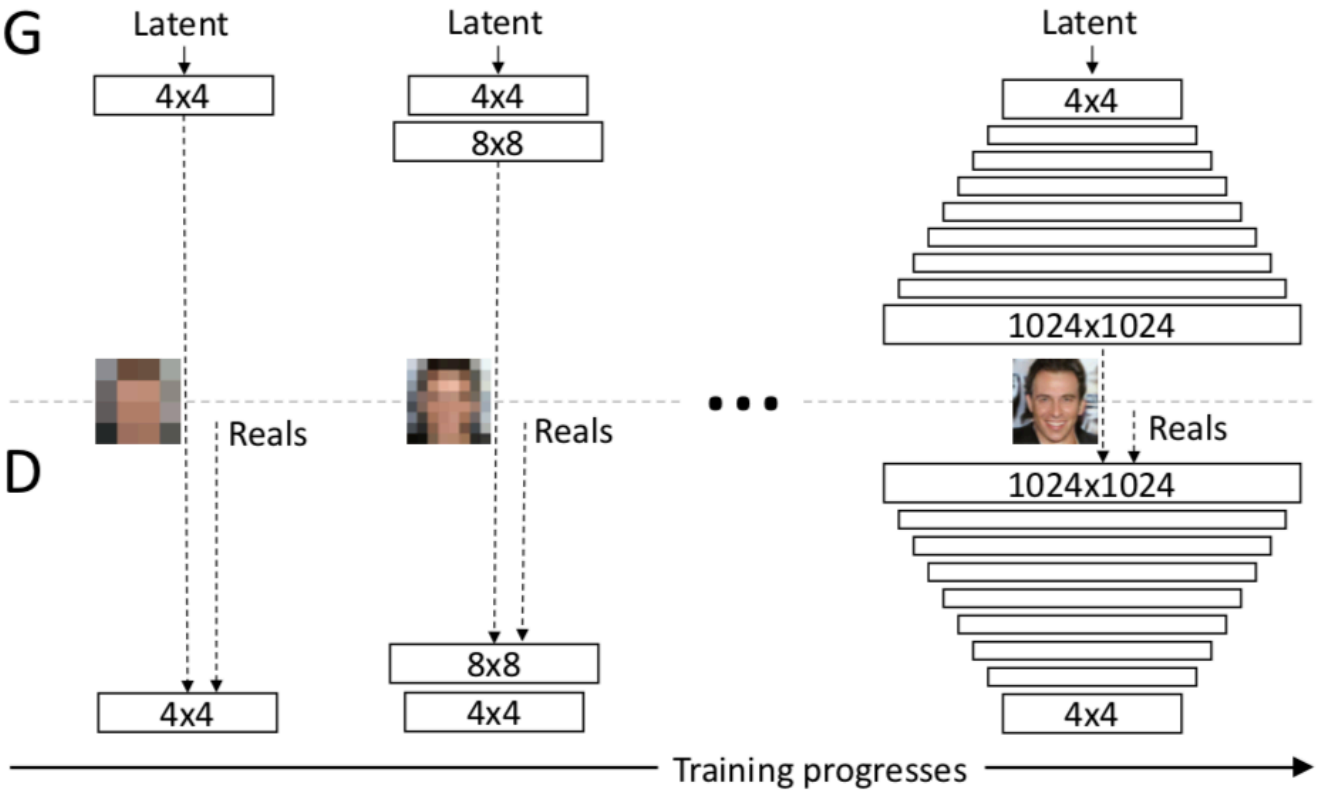


Figure 4: Visualisation of progressive growing of GANs.

- Latent vector size stays constant.
- Training data is downsampled.

Adding a new layer

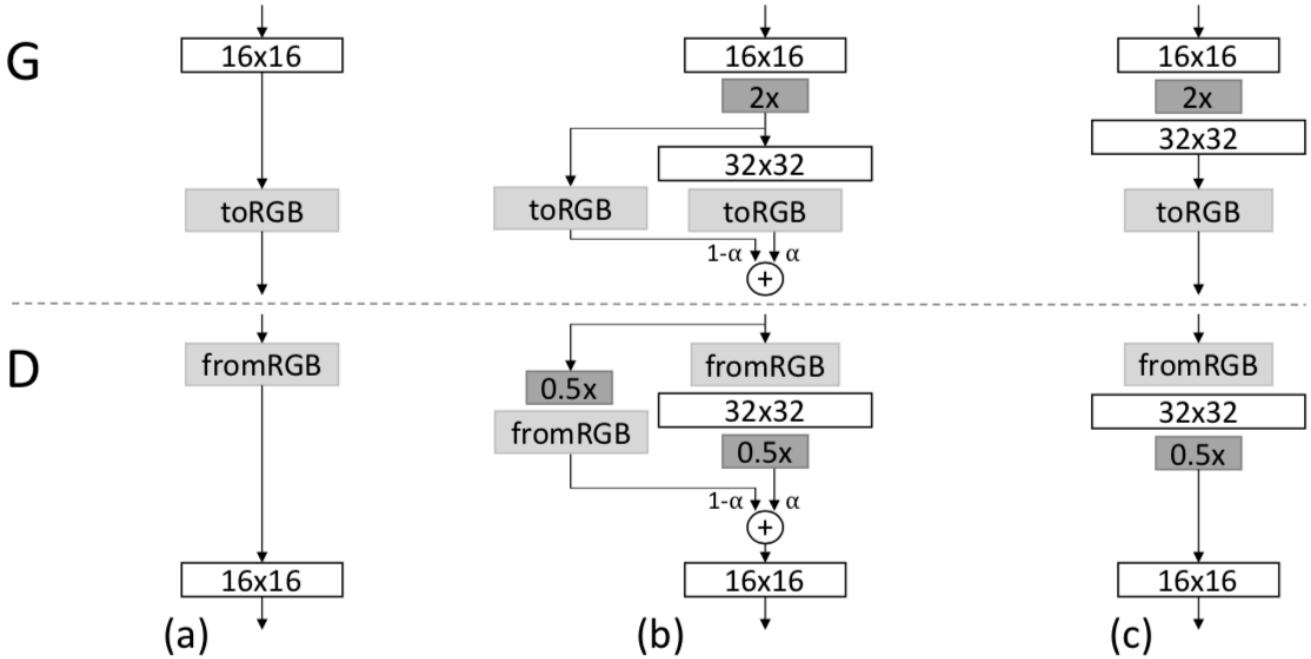


Figure 5: Fading in a layer.[1]

- Upsampling: nearest-neighbour interpolation
- Downsampling: average pooling
- θ for new layer is initialised randomly.
- α: linear transition parameter.

Progressive growing of GANs:[1] results

Figure 6: ‘Celebrities’ generated using a progressive GAN

How random is the final G?

Figure 7: Nearest neighbours of the generated faces in the training set.

Different ages

Mistakes

Bonus: interpolation of latent input

https://youtu.be/X0xxPcy5Gr4?t=107

Approaches

1. Autoregressive (AR) models:
   - High quality
   - Extremely slow
2. Variational Autoencoders (VAE)
   - Blurry
   - Fast
3. Progressively grown Generative Adversarial Network (GAN)
   - High quality
   - Fast (only 1 GAN, progressively grown)
   - Increased GAN training stability

Thank you. Bibliography


Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen.
Progressive growing of gans for improved quality, stability, and variation.
arXiv preprint arXiv:1710.10196, 2017.


Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al.
Conditional image generation with pixelcnn decoders.
In *Advances in Neural Information Processing Systems*, pages 4790–4798, 2016.


Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio.
Generative adversarial nets.
In *Advances in neural information processing systems*, pages 2672–2680, 2014.