Generative Adversarial Networks (GAN)

Alexander Binder

University of Oslo (UiO)

April 28, 2021

UiO : Department of Informatics
University of Oslo

Learning goals

- loss analysis: understanding what losses in classic GAN minimize in a special case
- a better loss: Wasserstein-GANs [loss design]
- problems of GANs
- a better architecture: StyleGAN (there is StyleGAN2 out btw) [architecture design]
- a better application: using GAN to inpaint and to create with constraints [human in the loop, usability]
- a better training: small scale training [training with limited data]

Recap on GANs: – the generator (for images)

- $G(\cdot)$ generates output sample (image)
- $G(\cdot)$ maps a random input vector $v \sim P_v$ onto an output image $z = G(v)$
- based on layers of upscale+convolution OR fractionally strided convolutions + other layers
- trainable parameters

Recap on classic GANs with non-saturating discriminator loss: – the discriminator

- $D(\cdot)$ critic for the quality of the generated samples (images)
- $D(\cdot)$ maps an input sample x onto a classification probability $D(x) \in [0, 1]$
- classification problem: $D(x) = 1$ – is certain that the input is from the training dataset and not from $G(\cdot)$
- for images typically a vanilla CNN: conv layers, batch norm, residual connections
- trainable parameters

Iterate:

- one step of generator: update parameters to create more realistic images, to increase discriminator loss
- one step of discriminator: update parameters to be able to discriminator generator images from real dataset images, to decrease discriminator loss

$$\min_{w(G)} L_G, \; L_G = E_{z \sim P_z}[-\ln D(G_{w(G)}(z))]$$

$$D(G(z)) = 1 \Rightarrow -\ln D(\cdot) = 0, L_G = 0$$

$$\min_{w(D)} L_D, \; L_D = E_{x \sim P_{data}}[-\ln(D(x))] + E_{z \sim P_z}[-\ln(1 - D(G(z)))]$$

$$D(x) = 1, D(G(z)) = 0 \Rightarrow L_D = 0$$

next step:

- for a special choice of generator loss L_G, show that training of generator and discriminator can be written as a max – min-formulation over a single objective function

$$\min_{w(G)} L_G,\ L_G = E_{z\sim P_z}[-\ln D(G(z))]$$

$$\min_{w(D)} L_D,\ L_D = E_{x\sim P_{data}}[-\ln(D(x))] + E_{z\sim P_z}[-\ln(1-D(G(z)))]$$

Consider the original GAN formulation: replace the first term by $E_{z\sim P_z}[\ln(1-D(G_{w(G)}(z)))]$ and Switch the second min to a max

(1) $\min_{w(G)} L_G,\ L_G = E_{z\sim P_z}[\ln(1-D(G(z)))]$

(2) $\max_{w(D)} L_D,\ L_D = E_{x\sim P_{data}}[\ln(D(x))] + E_{z\sim P_z}[\ln(1-D(G(z)))]$

Have now this modified GAN formulation:

$$(1)\ \min_{w(G)} L_G,\ L_G = E_{z\sim P_z}[\ln(\,1 - D(G(z))\,)]$$

$$(2)\ \max_{w(D)} L_D,\ L_D = E_{x\sim P_{data}}[\ln(D(x))] + E_{z\sim P_z}[\ln(1 - D(G(z)))]$$

Minimizing (2) wrt G is equivalent to minimizing (1) wrt G.

Therefore can use eq (2) for both:

$$\max_{w(D)}\min_{w(G)} L_{GD},\ L_{GD} = E_{x\sim P_{data}}[\ln(D(x))] + E_{z\sim P_z}[\ln(\,1 - D(G_{w(G)}(z))\,)]$$

Result: optimization problem with one loss function, as max – min

A special case of GAN objective as max – min formulation

If one uses the modified generator loss $L_G = E_{z\sim P_z}[\ln(1 - D(G(z)))]$,
and above discriminator loss $L_D = E_{x\sim P_{data}}[-\ln(D(x))] + E_{z\sim P_z}[-\ln(1 - D(G(z)))]$,

then one obtains as GAN objective:

$$\max_{w(D)} \min_{w(G)} L_{GD},\ L_{GD} = E_{x\sim P_{data}}[\ln(D(x))] + E_{z\sim P_z}[\ln(1 - D(G_{w(G)}(z)))]$$

Next steps:

- what is the optimal solution for the discriminator $D(\cdot)$?
- given that optimal solution, what does the generator $G(\cdot)$ try to optimize?

You can ask: what is for each x the best solution for the discriminator $D(\cdot)$?

$$
\begin{aligned}
L_{GD} &= E_{x\sim P_{data}}[\ln(D(x))] + E_{z\sim P_z}[\ln(1 - D(G_{w(G)}(z)))] \\
&= \int_x P_{data}(x)\ln(D(x))dx + \int_z P_z(z)\ln(1 - D(G(z)))dz \\
&= \int P_{data}(x)\ln(D(x)) + P_G(x)\ln(1 - D(x))dx
\end{aligned}
$$

$$
\operatorname{argmax}_{D(x)}\ a\ln(D(x)) + b\ln(1 - D(x)) = ?
$$

$$
D^{(*)}(x) = \frac{a}{a+b} = \frac{P_{data}(x)}{P_{data}(x) + P_G(x)}
$$

Have: solution for the optimal discriminator $D(\cdot)$

for the above loss function L_{GD} for max-min-optimization, the optimal discriminator would predict

$$D^{(*)}(x) = \frac{P_{data}(x)}{P_{data}(x) + P_G(x)}$$

next:

- relationship of the optimization function L_{GD} to Jensen-Shannon divergence.

compare Jensen-Shannon divergence to KL-divergence:

- **KL-Divergence** (0 if equal, positive if different, a dissimilarity measure):

$$D_{KL}(p||q) = \int p(x) \log\left(\frac{p(x)}{q(x)}\right) dx \text{ if has a density}$$

$$= \sum_i p_i \log\left(\frac{p_i}{q_i}\right) \text{ if discrete}$$

- **JS-Divergence** is a symmetrized variant (measure left and right versus the average $\frac{p+q}{2}$):

$$D_{JS}(p||q) = \frac{1}{2} D_{KL}\left(p||\frac{p+q}{2}\right) + \frac{1}{2} D_{KL}\left(q||\frac{p+q}{2}\right)$$

- relationship of the optimization function L_{GD} to Jensen-Shannon divergence

plug $D^{(*)}(x) = \frac{P_{data}(x)}{P_{data}(x)+P_G(x)}$ in for D:

$$
\begin{aligned}
L_{GD} &= \int P_{data}(x)\ln(D(x)) + P_G(x)\ln(\,1 - D(x)\,)dx \\
&= \int P_{data}(x)\ln(\frac{P_{data}(x)}{P_{data}(x)+P_G(x)}) + P_G(x)\ln(\,1 - \frac{P_{data}(x)}{P_{data}(x)+P_G(x)}\,) \\
&= \int P_{data}(x)\ln(\frac{P_{data}(x)}{P_{data}(x)+P_G(x)}) + P_G(x)\ln(\frac{P_G(x)}{P_{data}(x)+P_G(x)}) \\
&= D_{KL}\left(p||p+q\right) + D_{KL}\left(q||p+q\right)
\end{aligned}
$$

looks already almost like JS-divergence already: we need $D_{KL}\left(\cdot||\frac{p+q}{2}\right)$

- relationship of the optimization function L_{GD} to Jensen-Shannon divergence

plug $D^{(*)}(x) = \frac{P_{data}(x)}{P_{data}(x)+P_G(x)}$ in for D:

$$
\begin{aligned}
L_{GD} &= \int P_{data}(x) \ln\left(\frac{P_{data}(x)}{P_{data}(x) + P_G(x)}\right) + P_G(x) \ln\left(\frac{P_G(x)}{P_{data}(x) + P_G(x)}\right) \\
&= \int P_{data}(x) \ln\left(\frac{\frac{1}{2}P_{data}(x)}{(P_{data}(x) + P_G(x))/2}\right) + P_G(x) \ln\left(\frac{\frac{1}{2}P_G(x)}{(P_{data}(x) + P_G(x))/2}\right) \\
&= \int P_{data}(x) \ln\left(\frac{P_{data}(x)}{(P_{data}(x) + P_G(x))/2}\right) + P_G(x) \ln\left(\frac{P_G(x)}{(P_{data}(x) + P_G(x))/2}\right) \\
&+ \int P_{data}(x) \ln\left(\frac{1}{2}\right) + P_G(x) \ln\left(\frac{1}{2}\right) \\
&= 2D_{JS}(P_{data}||P_G) + 2\ln\left(\frac{1}{2}\right)
\end{aligned}
$$

The consequence from $L_{GD} = 2D_{JS}(P_{data}||P_G) + C$:

A special case of GAN objective as max – min formulation

Suppose we use above modified GAN losses, and the discriminator is the optimal one $D^{(*)}(x)$, then

the generator tries to learn to minimize the Jensen-Shannon-Divergence between $D_{JS}(P_{data}||P_G)$ to the true data distribution P_{data}.

Therefore the optimal generator solution would be $P_G = P_{data}$.

However often this cannot be optimized well.

A special case of GAN objective as max – min formulation

Suppose we use above modified GAN losses, and the discriminator is the optimal one $D^{(*)}(x)$, then

the generator tries to learn to minimize the Jensen-Shannon-Divergence between $D_{JS}(P_{data}||P_G)$ to the true data distribution P_{data}.

Therefore the optimal generator solution would be $P_G = P_{data}$.

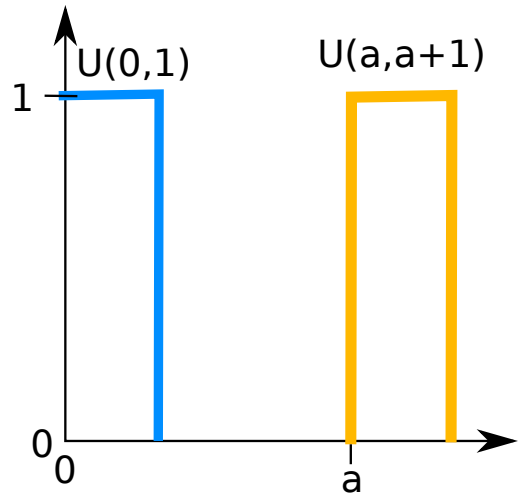
next step:

- What is the problem with this ??

Problem: Suppose your two distributions have no overlap in their probability supports, then the KL-divergence is constant, thus its derivative will be zero.

Example:

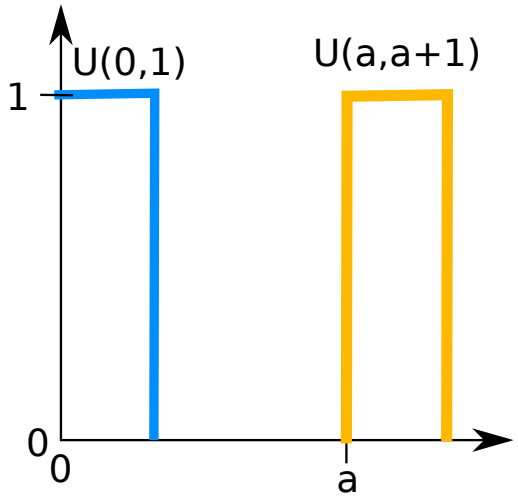
$$P_{data}(x) = U([0,1]) = \mathbb{1}[0 \leq x \leq 1]$$
$$P_{G_a}(x) = U([a, a+1]) = \mathbb{1}[a \leq x \leq a+1]$$



$$a \notin [0,1] \Rightarrow 2D_{JS} = D_{KL}(U_0 || \frac{U_0 + U_a}{2}) + D_{KL}(U_a || \frac{U_0 + U_a}{2}) = 1$$

What is the consequence? Since the objective function is constant, and not depending on a, there is no way by gradient methods to make a converge so that $P_{G_a} = P_{data}$ (given we use the optimal discriminator ...).

classic GAN training – loss analysis: when it can fail to learn



What is the consequence? For disjoint supports, or if the supports are not disjoint but have a very small probability overlap (eg replace the two uniform distributions by normal distributions which are far away from each other),

- then the learning of the generator suffers from vanishing gradients, and does not progress in time horizons of PhD students anymore

- mode collapse (last lecture)
- poor convergence (see above)
- hyperparameter sensitivity
- a too good discriminator $D(\cdot)$ can lead to gradient vanishing in $G(\cdot)$

- **GANs are very sensitive to hyperparameter choices** for more details https://papers.nips.cc/paper/7350-are-gans-created-equal-a-large-scale-study.pdf.
- a too good discriminator $D(\cdot)$ can lead to gradient vanishing in $G(\cdot)$. This happens if $D(G(y)) = 0$ for a whole region $\{z : \|z - G(y)\| < \epsilon\}$ around your current point $G(y)$.

- a too good discriminator $D(\cdot)$ can lead to gradient vanishing in $G(\cdot)$. This happens if $D(G(y)) \overset{=}{\approx} 0$ for a whole region $\{z : \|z - G(y)\| < \epsilon\}$ around your current point $G(y)$.

$$
\begin{aligned}
&\forall\, z : \|z - G(y)\| < \epsilon : \;\; D(G(y)) = 0 \\
&\Rightarrow \nabla^{(z)} D(z = G(y)) = 0 \\
&\Rightarrow \frac{\partial D(G(y))}{\partial w_d} = \nabla^{(z)} D(z = G(y)) \frac{\partial G}{\partial w_d} = 0
\end{aligned}
$$

Outline

- result: a different loss function for the generator, replacing the discriminator $D(\cdot)$
- idea: measure and minimize a distance between probability distributions μ of training data and τ of generated samples.

$$W_1(\mu, \tau) = \inf_{\gamma \in \Gamma(\mu,\tau)} e(\gamma) = \inf_{\gamma \in \Gamma(\mu,\tau)} \sum_{i=1}^{n} \sum_{k=1}^{n} d(x_i, x_k)\gamma(x_i, x_k)$$

The samples are x_i, x_k. $d(x_i, x_k)$ is a distance between samples. $\gamma \in \Gamma(\mu, \tau)$ is a joint probability distribution, such that

$$\sum_k \gamma(x_i, x_k) = \mu(x_i)$$

$$\sum_i \gamma(x_i, x_k) = \tau(x_k)$$

Looks complicated, will be explained in details!

Wasserstein distance preliminaries: the cost of moving water

- Have n cauldrons $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$.
  Cauldron x_i contains the amount $\mu(x_i)$ of water.
  For simplicity: $\sum_{x_i} \mu(x_i) = 1$ (liter?)
- We want to redistribute the water. Distributing an amount of 1 water from x_i to x_k **costs** us $d(x_i, x_k)$.
  - We have a distribution function:
    $\gamma(x_i, x_k)$ is the **amount redistributed** $x_i \to x_k$.
  - It will have the following properties:
    (P1): $\sum_{\text{targets } k} \gamma(x_i, x_k) = \mu(x_i)$ – no spilling of water
    (P2): the final amount after distribution will be
    $\sum_{\text{sources } i} \gamma(x_i, x_k) =: \tau(x_k)$.
  - what is the cost of applying $\gamma(x_i, x_k)$?

Wasserstein distance preliminaries: the cost of moving water

- Have n cauldrons $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$.
  Cauldron x_i contains the amount $\mu(x_i)$ of water.
  For simplicity: $\sum_{x_i} \mu(x_i) = 1$ (liter?)
- We want to redistribute the water. Distributing an amount of 1 water from x_i to x_k **costs** us $d(x_i, x_k)$.
  - We have a distribution function:
    $\gamma(x_i, x_k)$ is the **amount redistributed** $x_i \to x_k$.
  - It will have the following properties:
    (P1): $\sum_{\text{targets } k} \gamma(x_i, x_k) = \mu(x_i)$ – no spilling of water
    (P2): the final amount after distribution will be
    $\sum_{\text{sources } i} \gamma(x_i, x_k) =: \tau(x_k)$.
  - what is the cost of applying $\gamma(x_i, x_k)$?

$$e(\gamma) = \sum_{\text{sources } i=1}^{n} \sum_{\text{targets } k=1}^{n} \gamma(x_i, x_k) d(x_i, x_k)$$

amount-weighted sum of costs

- for initial distribution $\mu(x_i)$,
  per unit movement cost $d(x_i, x_k)$, redistribution function $\gamma(x_i, x_k)$,
  resulting in final distribution $\tau(x_k) := \sum_i \gamma(x_i, x_k)$, we have as cost:

$$e(\gamma) = \sum_{i=1}^{n} \sum_{k=1}^{n} \gamma(x_i, x_k) d(x_i, x_k)$$

- Towards wasserstein distance: what is if ?? ... we have initial distribution $\mu(x_i)$, final distribution $\tau(x_k)$ and we want to find the least-cost redistribution $\gamma(x_i, x_k)$ from $\mu(x_i)$ to $\tau(x_k)$?
  We assume $\sum_i \mu(x_i) = \sum_k \tau(x_k) = 1$
- any **valid** redistribution $\gamma(x_i, x_k)$ must satisfy:
  (P1): $\sum_k \gamma(x_i, x_k) = \mu(x_i)$
  (P2): $\tau(x_k) = \sum_i \gamma(x_i, x_k)$.
- let be $\Gamma(\mu, \tau)$ the space of all redistributions satisfying (P1) and (P2)
- then the least cost redistribution is given by

$$\inf_{\gamma \in \Gamma(\mu,\tau)} e(\gamma) = \inf_{\gamma \in \Gamma(\mu,\tau)} \sum_{i=1}^{n} \sum_{k=1}^{n} \gamma(x_i, x_k) d(x_i, x_k)$$

- for initial distribution $\mu(x_i)$,
  per unit movement cost $d(x_i, x_k)$, redistribution function $\gamma(x_i, x_k)$,
  resulting in final distribution $\tau(x_k) := \sum_i \gamma(x_i, x_k)$, we have as cost:

$$e(\gamma) = \sum_{i=1}^{n} \sum_{k=1}^{n} \gamma(x_i, x_k) d(x_i, x_k)$$

- Towards wasserstein distance: what is if ?? ... we have initial distribution $\mu(x_i)$, final distribution $\tau(x_k)$ and we want to find the least-cost redistribution $\gamma(x_i, x_k)$ from $\mu(x_i)$ to $\tau(x_k)$?
  We assume $\sum_i \mu(x_i) = \sum_k \tau(x_k) = 1$
- any **valid** redistribution $\gamma(x_i, x_k)$ must satisfy:
  (P1): $\sum_k \gamma(x_i, x_k) = \mu(x_i)$
  (P2): $\tau(x_k) = \sum_i \gamma(x_i, x_k)$.
- let be $\Gamma(\mu, \tau)$ the space of all redistributions satisfying (P1) and (P2)
- then the least cost redistribution is given by

$$\inf_{\gamma \in \Gamma(\mu,\tau)} e(\gamma) = \inf_{\gamma \in \Gamma(\mu,\tau)} \sum_{i=1}^{n} \sum_{k=1}^{n} \gamma(x_i, x_k) d(x_i, x_k)$$

- for initial distribution $\mu(x_i)$,
  per unit movement cost $d(x_i, x_k)$, redistribution function $\gamma(x_i, x_k)$,
  resulting in final distribution $\tau(x_k) := \sum_i \gamma(x_i, x_k)$, we have as cost:

$$e(\gamma) = \sum_{i=1}^{n} \sum_{k=1}^{n} \gamma(x_i, x_k) d(x_i, x_k)$$

- Towards wasserstein distance: what is if ?? ... we have initial distribution $\mu(x_i)$, final distribution $\tau(x_k)$ and we want to find the least-cost redistribution $\gamma(x_i, x_k)$ from $\mu(x_i)$ to $\tau(x_k)$?
  We assume $\sum_i \mu(x_i) = \sum_k \tau(x_k) = 1$
- any **valid** redistribution $\gamma(x_i, x_k)$ must satisfy:
  (P1): $\sum_k \gamma(x_i, x_k) = \mu(x_i)$
  (P2): $\tau(x_k) = \sum_i \gamma(x_i, x_k)$.
- let be $\Gamma(\mu, \tau)$ the space of all redistributions satisfying (P1) and (P2)
- then the least cost redistribution is given by

$$\inf_{\gamma \in \Gamma(\mu,\tau)} e(\gamma) = \inf_{\gamma \in \Gamma(\mu,\tau)} \sum_{i=1}^{n} \sum_{k=1}^{n} \gamma(x_i, x_k) d(x_i, x_k)$$

special case of Wasserstein distance for the discrete case

- Suppose we have a discrete space $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$, we have a distance on it: $d(x_i, x_k)$
  (e.g. euclidean distance $d(x_i, x_k) = \|x_i - x_k\|$).
- We are given two probability distributions on μ, τ on $\mathcal{X}$, for which we want to compute a distance $W_1(\mu, \tau)$ between them.
- Furthermore let be $\Gamma(\mu, \tau)$ the space of all joint distributions $\gamma(x_i, x_k)$ over $\mathcal{X} \times \mathcal{X}$ such that ...
  they have μ and τ as their marginal distribution,
  meaning: $\sum_k \gamma(x_i, x_k) = \mu(x_i)$ and $\sum_i \gamma(x_i, x_k) = \tau(x_k)$

Then we can define the Wasserstein distance as:

$$W_1(\mu, \tau) = \inf_{\gamma \in \Gamma(\mu,\tau)} e(\gamma) = \inf_{\gamma \in \Gamma(\mu,\tau)} \sum_{i=1}^{n} \sum_{k=1}^{n} d(x_i, x_k)\gamma(x_i, x_k)$$

$$W_1(\mu, \tau) = \inf_{\gamma \in \Gamma(\mu,\tau)} e(\gamma) = \inf_{\gamma \in \Gamma(\mu,\tau)} \sum_{i=1}^{n} \sum_{k=1}^{n} d(x_i, x_k)\gamma(x_i, x_k)$$

- the Wasserstein distance measures a distance between probability distributions but it depends on a distance metric $d(x_i, x_k)$ between samples!
  (e.g. for images one could use for x_i feature maps from a neural networks)
- has a clear interpretation as the least cost to shift a distribution $\mu(\cdot)$ into a distribution $\tau(\cdot)$ when the cost to shift one unit from x_i to x_k is given as $d(x_i, x_k)$
- $\gamma(x_i, x_k)$ is the redistribution function, is a joint probability

discrete version – countable space $\mathcal{X} = \{x_1, x_2, \ldots\}$ with probability for each element $p(x)$:

$$W_1(\mu, \tau) = \inf_{\gamma \in \Gamma(\mu,\tau)} e(\gamma) = \inf_{\gamma \in \Gamma(\mu,\tau)} \sum_{i=1}^{n} \sum_{k=1}^{n} d(x_i, x_k)\gamma(x_i, x_k)$$

continuous version (no exam) – non-countable space like $\mathbb{R}$ with a density $p(x)$:

$$W_1(\mu, \tau) = \inf_{\gamma \in \Gamma(\mu,\tau)} e(\gamma) = \inf_{\gamma \in \Gamma(\mu,\tau)} \int d(x, x')\gamma(x, x')dxdx'$$

discrete – countable space $\mathcal{X} = \{x_1, x_2, \ldots\}$ with probability for each element $p(x)$:

$$\sum_i p(x_i) f(x_i)$$

$$\sum_{i,k,l} p(x_i, x_k, x_l) f(x_i, x_k, x_l)$$

continuous version – non-countable space like $\mathbb{R}$ with a density $p(x)$:

$$\int p(x) f(x) dx$$

$$\int p(x, x', x'') f(x, x', x'') dx dx' dx''$$

$$\sum_i p(x_i) \ldots \overset{!}{\longleftrightarrow} \int_x p(x) \ldots dx$$

Steps to obtain a Wasserstein GAN

(A) Write the Wasserstein distance as a difference of two expectations of a function f, maximized over the function f

$$W_1(P_{data}, P_{GAN}) = \max_{f \in Lip_1} E_{x \sim P_{data}}[f(x)] - E_{y \sim P_{GAN}}[f(y)]$$

(B) Replace the expectations $E[\cdot]$ by mini-batch averages, one time over the real data (for $x \sim P_{data}$), one time over the data from the generator (for $y \sim P_{GAN}$)

(C) Represent the function f, the so-called critic, by a neural net.

(D) Train the critic neural net using above max-objective turned into a min (typically with some gradient norm penalty)

(E) Interleave the train-critic-step with the step which optimizes the generator

Kantorovich-Rubinstein Duality Theorem

The Wasserstein distance for a metric can be computed as (Kantorovich-Rubinstein Duality Theorem, see a theorem in chapter 5 of the book by Cedric Villani, *Optimal Transport: Old and New*):

$$W_1(\mu,\tau) = \inf_{\gamma\in\Gamma(\mu,\tau)} \int_{\mathcal{X}\times\mathcal{X}} d(x,y)\gamma(x,y)dxdy$$
$$= \sup_{f\in Lip_1} E_{x\sim\mu}[f(x)] - E_{y\sim\tau}[f(y)]$$
$$f \in Lip_1 \Leftrightarrow |f(x) - f(y)| \leq 1 \cdot \|x-y\|_2 \quad \forall x, y$$

use $\mu = P_{data} \approx Unif(D_n)$, $\tau = P_{GAN}$

$$W_1(\mu, \tau) = \sup_{f \in Lip_1} E_{x \sim P_{data}}[f(x)] - E_{y \sim P_{GAN}}[f(y)]$$

$$\approx \sup_{f \in Lip_1} \frac{1}{b} \sum_{x_i \sim D_n}^{b} f(x_i) - \frac{1}{b} \sum_{v_i \sim \text{ input codes}}^{b} f(G(v_i))$$

Thus the generator objective becomes:

$$\inf_{w(G)} \sup_{f \in Lip_1} \frac{1}{b} \sum_{x_i \sim D_n}^{b} f(x_i) - \frac{1}{b} \sum_{v_i \sim \text{ input codes}}^{b} f(G(v_i))$$

Generator loss? we want to minimize the above, thus optimize for a given $f(\cdot)$: Optimize for weights $w(G)$ of G:

$$\inf_{w(G)} \frac{1}{b} \sum_{x_i \sim D_n}^{b} f(x_i) - \frac{1}{b} \sum_{v_i \sim \text{ input codes}}^{b} f(G(v_i))$$

this does not depend on D_n or the first term as a whole, thus (and replacing inf by a finite set min):

The generator loss for a Wasserstein GAN

$$\min_{w(G)} L(G), \; L(G) = -\frac{1}{b} \sum_{v_i \sim \text{input codes}}^{b} f(G(v_i))$$

discriminator loss?

Note: $f(\cdot) = D(\cdot)$ for the above function f

We want to maximize the distance for a given G by changing $D(\cdot)$

$$\sup_{D \in Lip_1} \frac{1}{b} \sum_{x_i \sim D_n}^{b} D(x_i) - \frac{1}{b} \sum_{v_i \sim \text{ input codes}}^{b} D(G(v_i))$$

if you want a minimum problem, multiply above with -1

The discriminator loss for a Wasserstein GAN

Optimize for weights $w(D)$ for D – note $D(\cdot) = f(\dot{)}$:

$$\min_{w(D): D \in Lip_1} -\frac{1}{b} \sum_{x_i \sim data}^{b} D(x_i) + \frac{1}{b} \sum_{v_i \sim \text{input codes}}^{b} D(G(v_i))$$

steps (C,D): Represent the function f (the critic) by a neural net.


```
f= someCNN()
f.loadweights()
critic_optimizer=torch.nn.optim.AdamW(f.parameters(),lr=...)
...
later in the code during training of the GAN:
...
samples=generator.sample().detach()
L= #that diff of averages with f + regularizer term
L.backward()
critic_optimizer.step()
```

Problem: need to ensure $f \in Lip_1$. Two options:

(1) Spectral normalization of every trainable convolution weight W https://arxiv.org/abs/1802.05957 (Miyato et al., ICLR 2018) – (details for this are out of class)

(2) add to $\hat{L}_D$ a gradient penalty term $R(f)$ like WGAN-GP Gulrajani et al. https://arxiv.org/abs/1704.00028 or WGAN-LP Petzka et al. https://openreview.net/pdf?id=B1hYRMbCW

$$R(f) = \frac{1}{n} \sum_{z_i} [(\|\nabla f(z_i)\| - 1)^2] \text{ (WGAN-LP)}$$

$$R(f) = \frac{1}{n} \sum_{z_i} [max(0, \|\nabla f(z_i)\| - 1)^2] \text{ (WGAN-LP)}$$

$$z_i = t x_i + (1-t) y_i, t \sim U([0,1]), x_i \sim D_n, y_i \sim G(\cdot)$$

For a quick intuition why restraining the gradient norm works:

if in the one-dimensional case $|f'(z)| \leq 1$, then

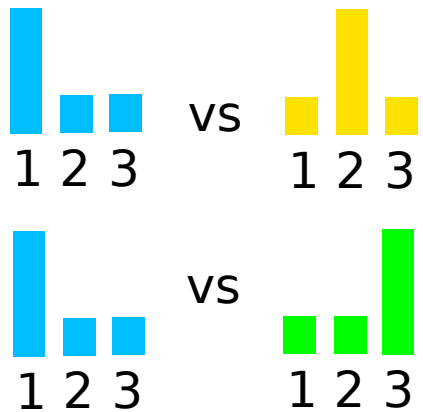
$$
\begin{aligned}
f(x) - f(y) &= \int_{z=y}^{z=x} f'(z)dz \\
\Rightarrow |f(x) - f(y)| &= |\int_{z=y}^{z=x} f'(z)dz| \leq \int_{z=y}^{z=x} |f'(z)|dz \\
&\leq \int_{z=y}^{z=x} 1dz = x - y \leq |x - y| \\
\Rightarrow |f(x) - f(y)| &\leq |x - y|
\end{aligned}
$$

for more insights on why Lipshitz regularization leads to better GAN training:
https://arxiv.org/pdf/1811.09567.pdf

https://github.com/jalola/improved-wgan-pytorch

WGAN-GP (Gulrajani et al.) https://arxiv.org/pdf/1704.00028.pdf and WGAN-LP (Petzka et al) https://openreview.net/pdf?id=B1hYRMbCW. GAN training can take days for one hyperparameter setting!!!

- mathematical difference:
  KL-Div $D_{KL}(p||q) = \sum_x p(x) \log p(x) - p(x) \log q(x)$ is not symmetric, does not satisfy triangle inequality. Wasserstein distance is a distance metric (symmetric, triangle inequality).
- practical advantage of Wasserstein-distance: takes metric $d(\cdot)$ in sample space for the samples x_i into account



both comparisons will have same KL-Divergence

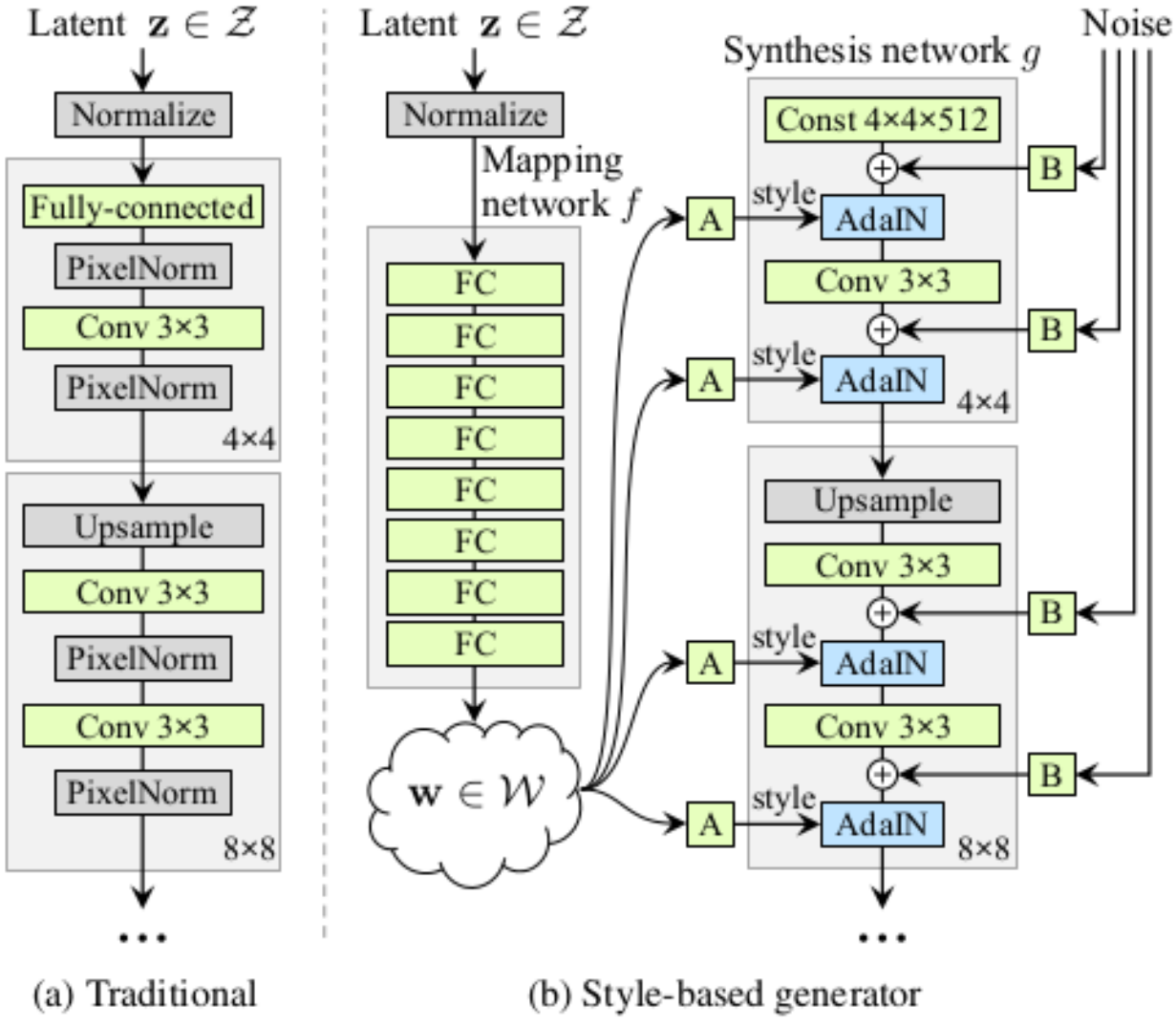
Using a CNN as a discriminator and above loss formulation, will we be able to compute such a minimum over this function class?

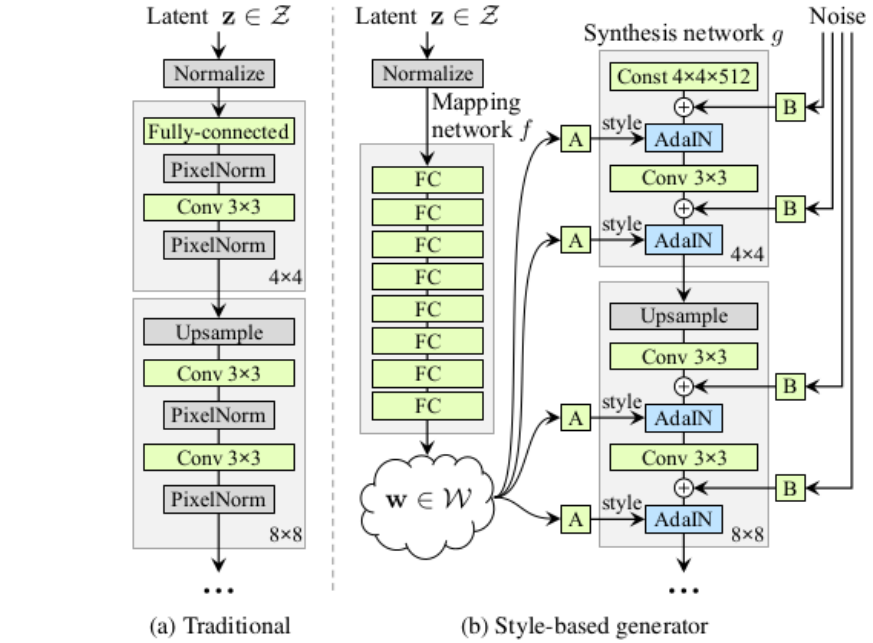
$$\sup_{f \in Lip_1} f(z) \ ?$$

If not, what are possible deviations?

Outline

Karras et al. https://arxiv.org/abs/1812.04948





(a) Traditional (b) Style-based generator

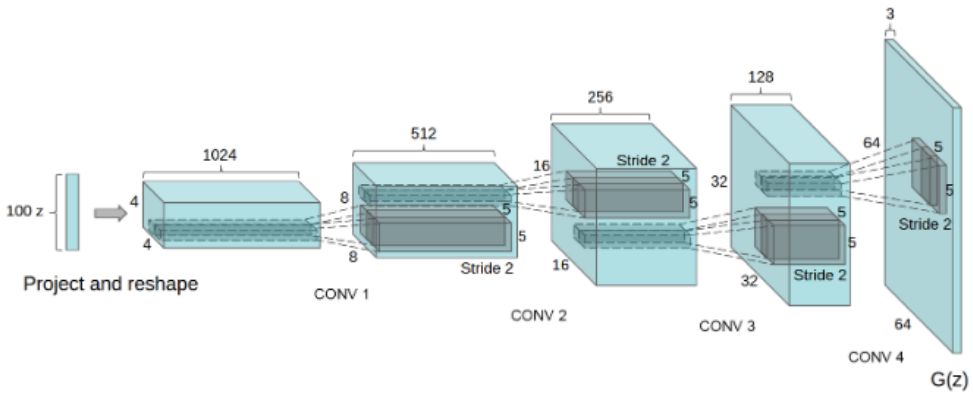
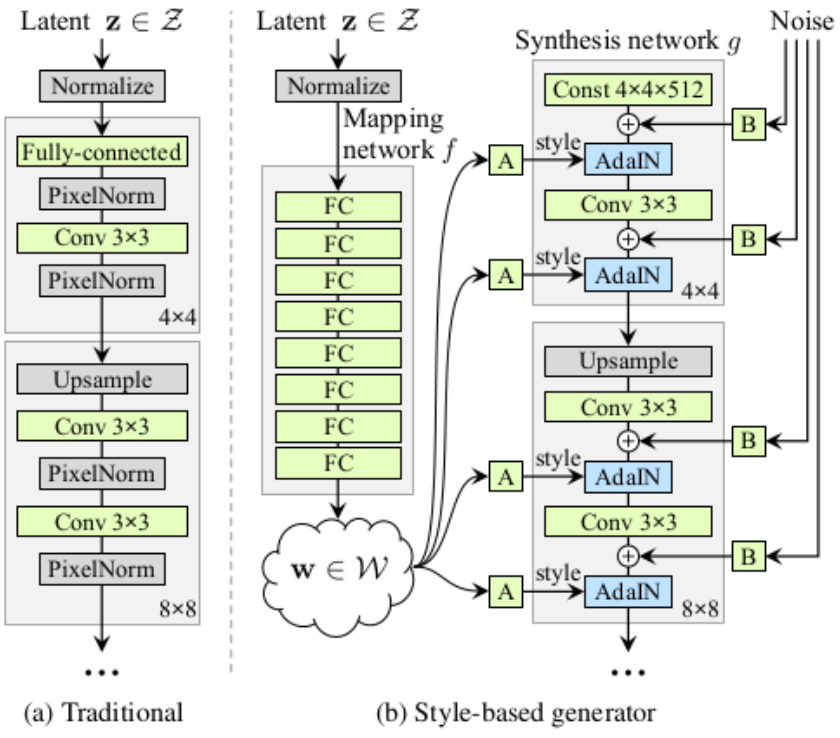
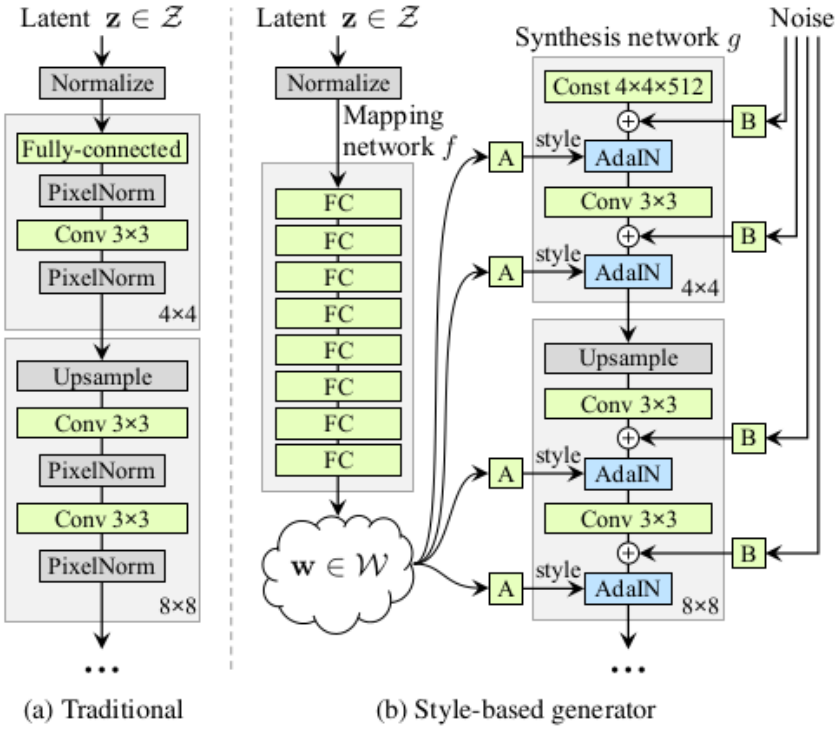


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.



Karras et al. https://arxiv.org/abs/1812.04948

- first idea: do not input a latent vector to get a random output.
- Instead add random noise (B) to every feature map.
- Reason: randomness from input vector can get smoothed out as it passes through many layers



Karras et al. https://arxiv.org/abs/1812.04948

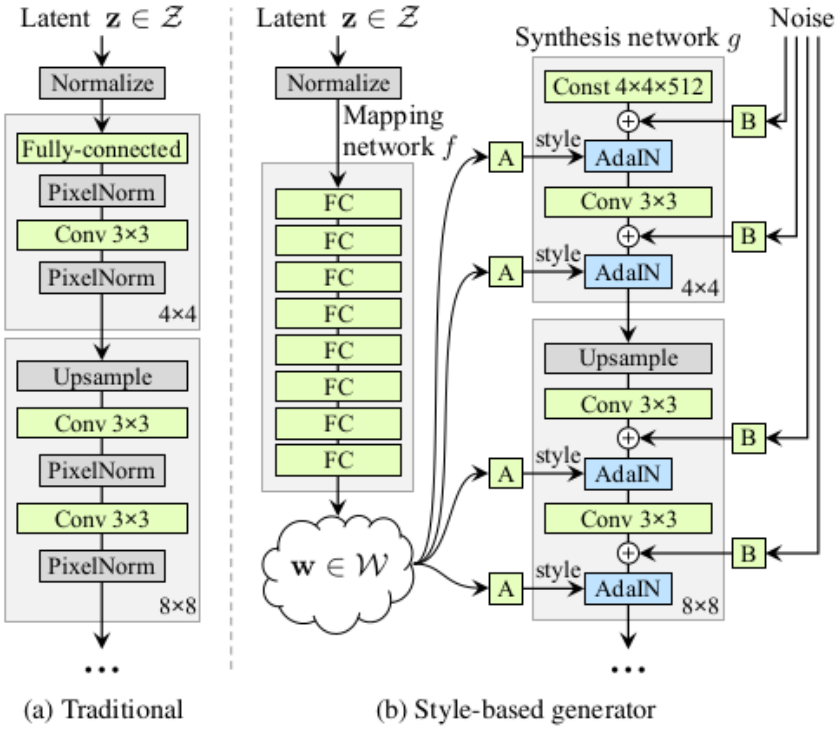
- second big idea: input styles using adaptive instance normalization

$$AI(x, y) = y_s \frac{x - \mu(x)}{\sigma(x)} + y_b$$

$$AI(x, y) = y_s \frac{x - \mu(x)}{\sigma(x)} + y_b$$

Difference to batchnormalization?

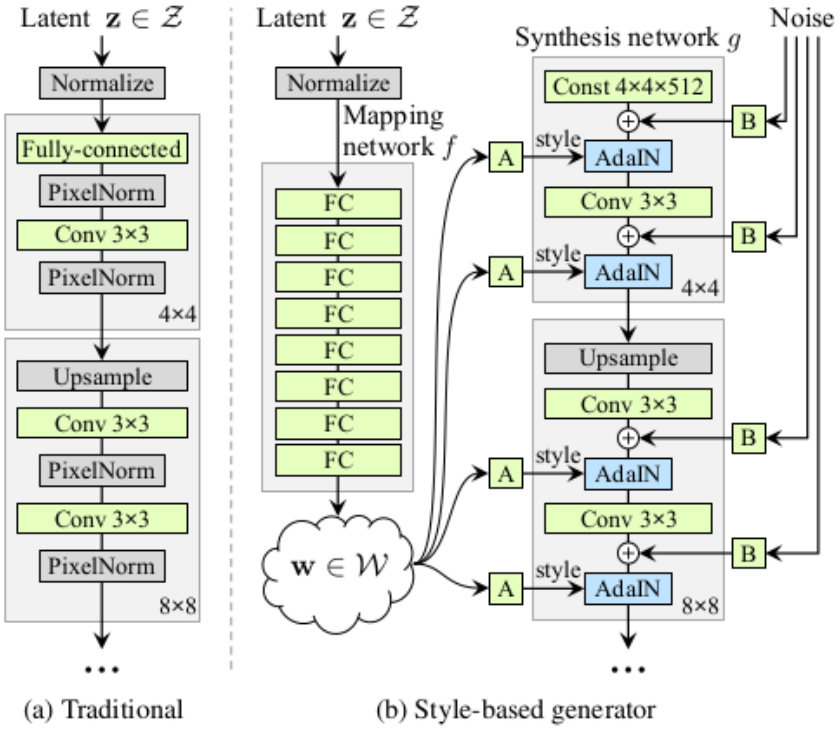
- batchsize$= 1$, same as layer normalization
- y_s, y_b are not directly trainable parameters as in BN but come as inputs from a trainable network



(a) Traditional (b) Style-based generator

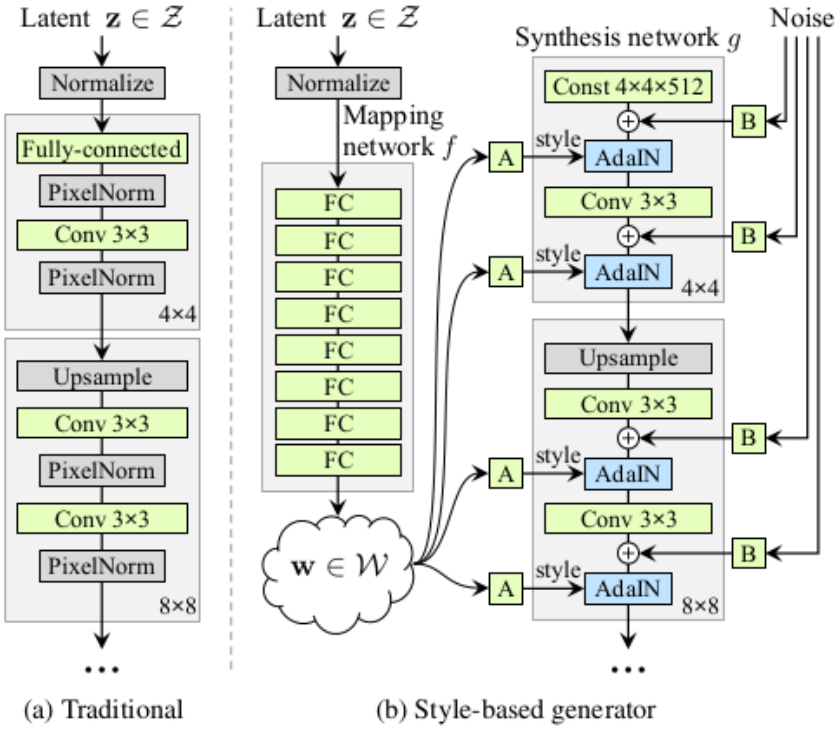
Karras et al. https://arxiv.org/abs/1812.04948

- style vectors y_s, y_b are learnt as an output of an 8-layer FC network.



- third big idea: uses progressive growing

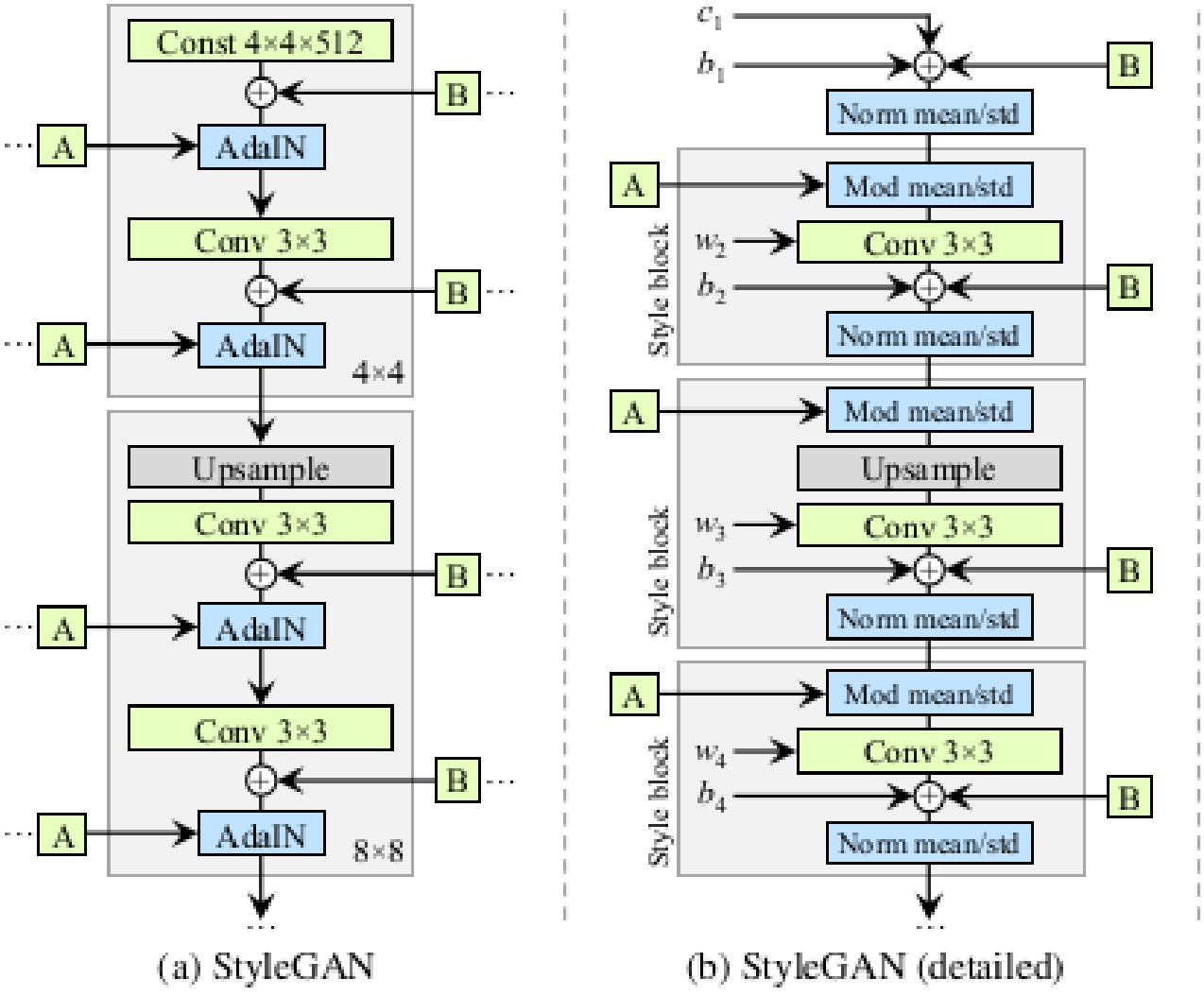
Karras et al. https://arxiv.org/abs/1812.04948



(a) Traditional (b) Style-based generator

Karras et al. https://arxiv.org/abs/1812.04948

- smaller tricks (out of class): 1.using bilinear up- and downsampling. 2. cross-apply style vectors from different runs. 3. Truncate large variations in the style space $\mathcal{W}$.



(a) StyleGAN (b) StyleGAN (detailed)

Style-vectors can be recombined from two different runs at different levels

another set of large changes: StyleGan2 (out of class)
https://arxiv.org/abs/1912.04958

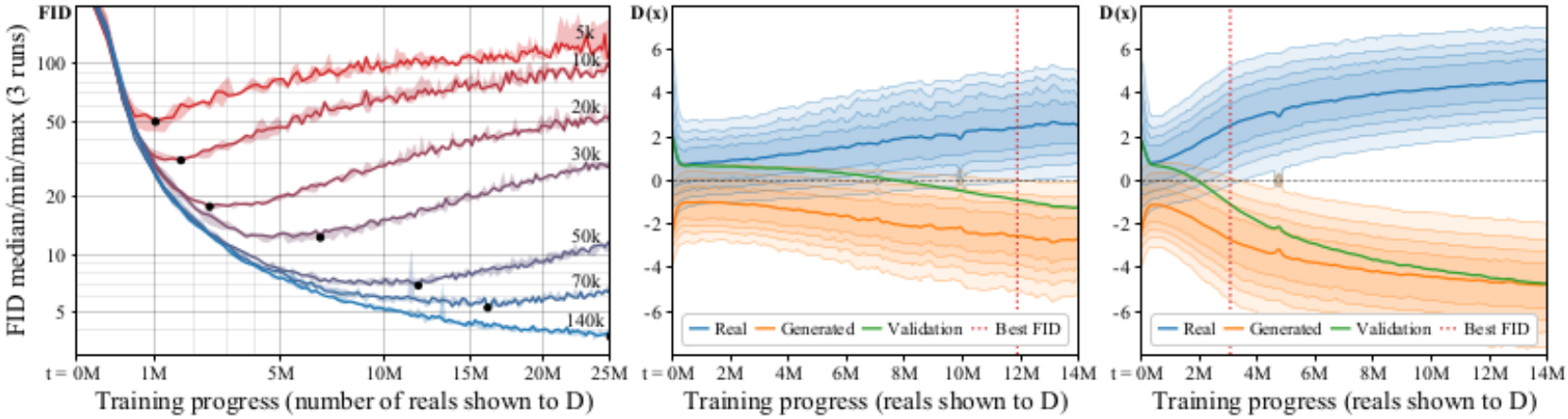
including giving up progressive growing

Outline

Karras et al. Training Generative Adversarial Networks with Limited Data, NeurIPS 2020, https://papers.nips.cc/paper/2020/hash/8d30aa96e72440759f74bd2306c1fa3d-Abstract.html

Takeaway for in-class

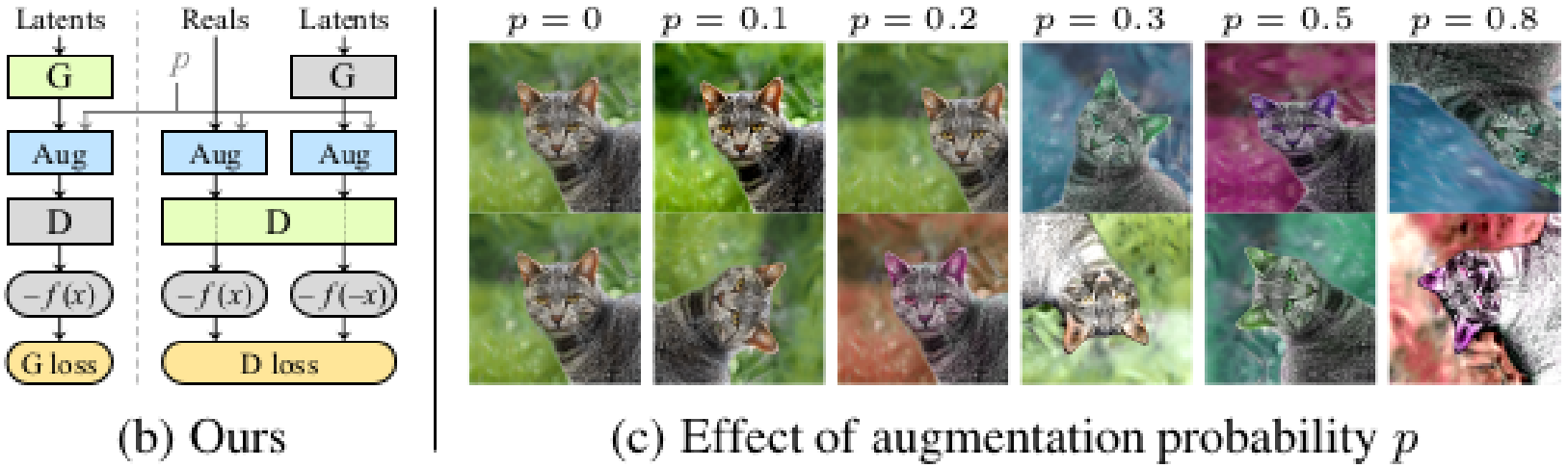
- how to create measures to quantify overfitting
- how to use data augmentation in GANs so that the GANs do not start to reproduce the augmentations.
- how to tune augmentation probabilities based on overfitting measures
- how one can apply transfer learning in GANs: to discriminator and freezing lowest layers



(a) Convergence of FFHQ (256×256) (b) Discriminator outputs, 50k (c) Discriminator outputs, 20k

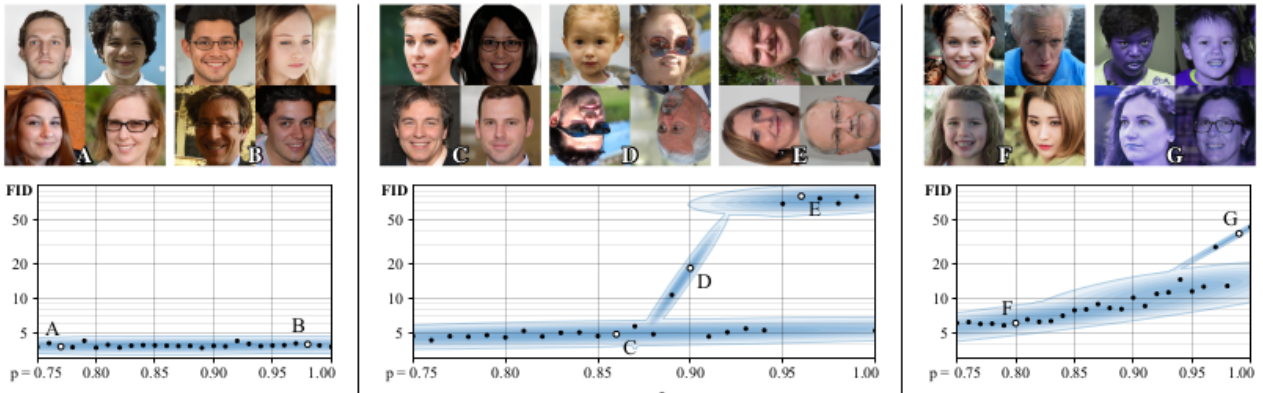
Figure 1: (a) Convergence with different training set sizes. "140k" means that we amplified the 70k dataset by $2\times$ through x-flips; we do not use data amplification in any other case. (b,c) Evolution of discriminator outputs during training. Each vertical slice shows a histogram of $D(x)$, i.e., raw logits.

- GANs overfit
- overfitting can be measured by comparing discriminator scores on: training data, generated data, withheld real images



(b) Ours (c) Effect of augmentation probability p

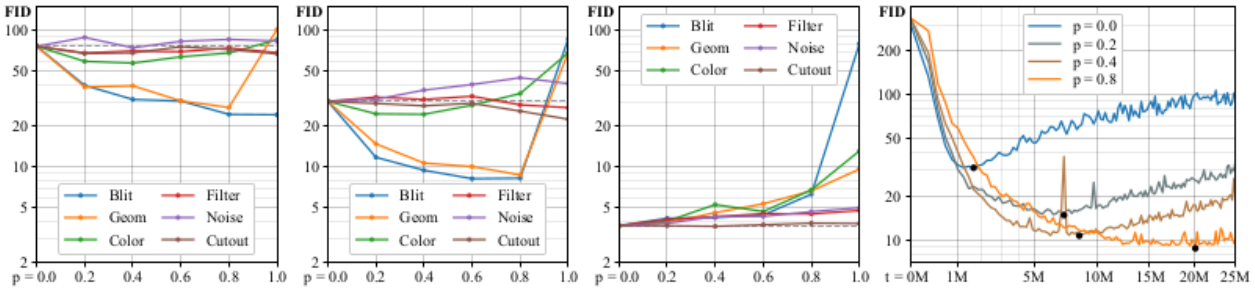
- solution is to use data augmentation for generator and discriminator
- examples: horizontal flip, rotations, color augmentation
- risk/problem is: the generator may learn to produce augmented images.



(a) Isotropic image scaling (b) Random 90° rotations (c) Color transformations

Figure 3: Leaking behavior of three example augmentations, shown as FID w.r.t. the probability of executing the augmentation. Each dot represents a complete training run, and the blue Gaussian mixture is a visualization aid. The top row shows generated example images from selected training runs, indicated by uppercase letters in the plots.

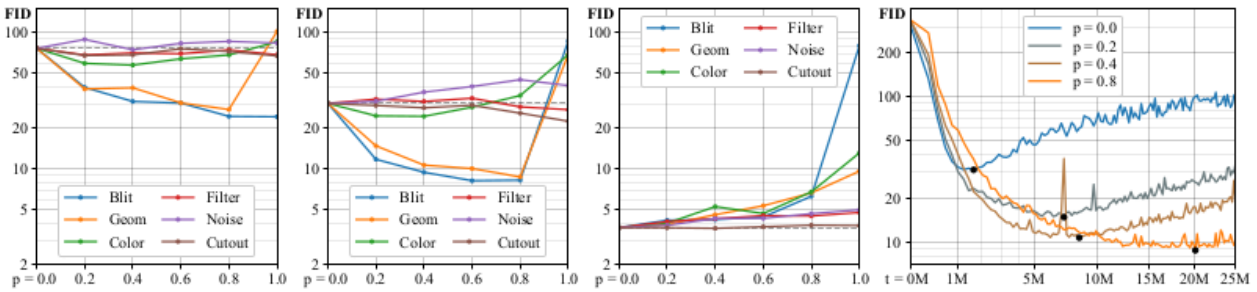
- solution is to use data augmentation for generator and discriminator
- risk/problem is: the generator may learn to produce augmented images.
- solution: allow to not execute each data augmentation with a probability $1 - p \in [0, 1]$
- if p is too large, then it starts to leak the augmentation into the generator



(a) FFHQ-2k (b) FFHQ-10k (c) FFHQ-140k (d) Convergence, 10k, Geom

Figure 4: (a-c) Impact of p for different augmentation categories and dataset sizes. The dashed gray line indicates baseline FID without augmentations. (d) Convergence curves for selected values of p using geometric augmentations with 10k training images.

- solution is to use data augmentation for generator and discriminator
- risk/problem is: the generator may learn to produce augmented images.
- solution: allow to not execute each data augmentation with a probability $1 - p \in [0, 1]$
- observation: different augmentations are differently effective, depends on: dataset size and p.



(a) FFHQ-2k (b) FFHQ-10k (c) FFHQ-140k (d) Convergence, 10k, Geom

Figure 4: (a-c) Impact of p for different augmentation categories and dataset sizes. The dashed gray line indicates baseline FID without augmentations. (d) Convergence curves for selected values of p using geometric augmentations with 10k training images.

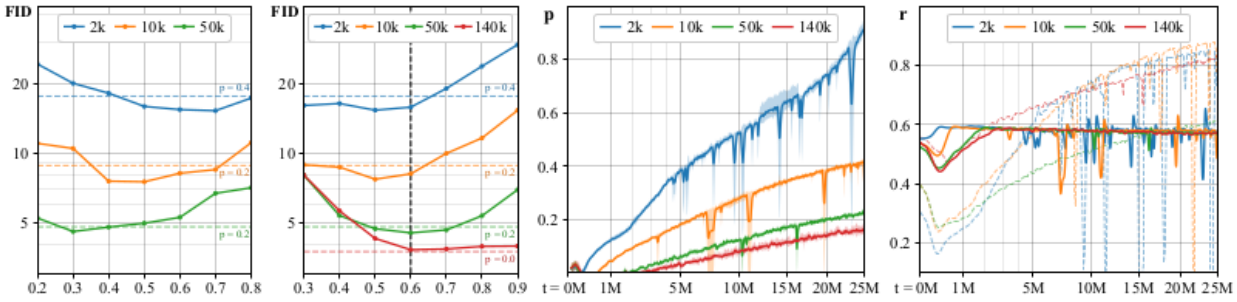
- solution is to use data augmentation for generator and discriminator
- risk/problem is: the generator may learn to produce augmented images.
- solution: allow to not execute each data augmentation with a probability $1 - p \in [0, 1]$
- next problem: how to select the best p?

- solution is to use data augmentation for generator and discriminator
- risk/problem is: the generator may learn to produce augmented images.
- solution: allow to not execute each data augmentation with a probability $1 - p \in [0, 1]$
- next problem: how to select the best p?
- define two measures to measure the overfitting, $D(x) \in (-\infty, +\infty)$ as logits:

$$r_v = \frac{E[D_{train}] - E[D_{val}]}{E[D_{train}] - E[D_{generated}]}$$

$$r_t = E_{x \sim \mathrm{TrainData}}[\,\mathrm{sign}(D_{train}(x))\,]$$

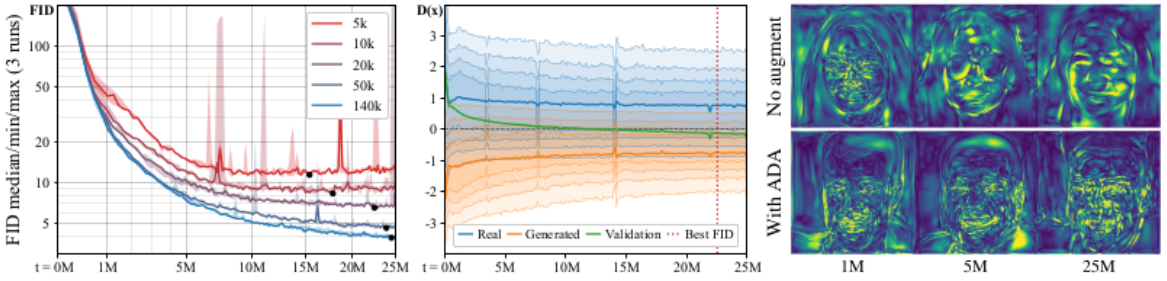
- 0 - no overfitting, 1 - bad overfitting



(a) r_v target sweep (b) r_t target sweep (c) Evolution of p over training (d) Evolution of r_t

Figure 5: Behavior of our adaptive augmentation strength heuristics in FFHQ. (a,b) FID for different training set sizes as a function of the target value for r_v and r_t. The dashed horizontal lines indicate the best fixed augmentation probability p found using grid search, and the dashed vertical line marks the target value we will use in subsequent tests. (c) Evolution of p over the course of training using heuristic r_t. (d) Evolution of r_t values over training. Dashes correspond to the fixed p values in (b).

- if one keeps r_v or r_t approximately constant by
  - initializing $p = 0$
  - adjusting p every 4 epochs ...



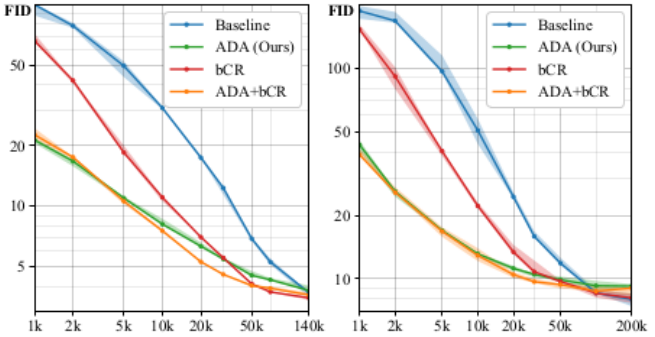
(a) With adaptive augmentation (b) Discriminator outputs, 20k (c) Discriminator gradients, 10k

Figure 6: (a) Training curves for FFHQ with different training set sizes using adaptive augmentation. (b) The supports of real and generated images continue to overlap. (c) Example magnitudes of the gradients the generator receives from the discriminator as the training progresses.

- solution is to use data augmentation for generator and discriminator
- allow to not execute each data augmentation with a probability $1 - p \in [0, 1]$
- Define two measures in $[0, 1]$ to measure the overfitting:

$$r_t = E_{x \sim \text{TrainData}}[\text{sign}(D_{train}(x))]$$

- run an algorithm: initializing $p = 0$, adjusting p every 4 epochs to keep $r_t \approx 0.6$



(a) FFHQ (256 × 256) (b) LSUN CAT (256 × 256)

Dataset		Baseline	ADA	+ bCR
FFHQ	1k	100.16	**21.29**	22.61
	5k	49.68	10.96	**10.58**
	10k	30.74	8.13	**7.53**
	30k	12.31	5.46	**4.57**
	70k	5.28	4.30	**3.91**
	140k	3.71	3.81	**3.62**
LSUN CAT	1k	186.91	43.25	**38.82**
	5k	96.44	16.95	**16.80**
	10k	50.66	13.13	**12.90**
	30k	15.90	10.50	**9.68**
	100k	**8.56**	9.26	8.73
	200k	**7.98**	9.22	9.03

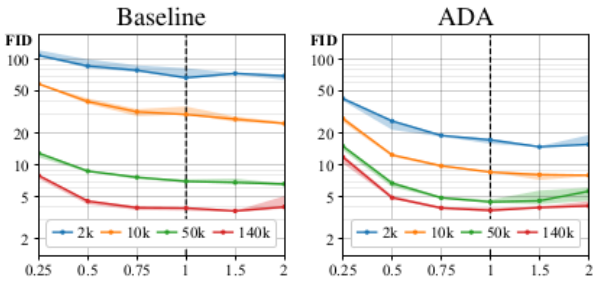
(c) Median FID

(d) Mean image

Figure 7: (a-c) FID as a function of training set size, reported as median/min/max over 3 training runs. (d) Average of 10k random images generated using the networks trained with 5k subset of FFHQ. ADA matches the average of real data, whereas the xy-translation augmentation in bCR [43] has leaked to the generated images, significantly blurring the average image.

FFHQ (256 × 256)	2k	10k	140k
Baseline	78.80±2.31	30.73±0.48	3.66±0.10
PA-GAN [39]	56.49±7.28	27.71±2.77	3.78±0.06
WGAN-GP [13]	79.19±6.30	35.68±1.27	6.54±0.37
zCR [43]	71.61±9.64	23.02±2.09	**3.45**±0.19
Auxiliary rotation [6]	66.64±3.64	25.37±1.45	4.16±0.05
Spectral norm [24]	88.71±3.18	38.58±3.14	4.60±0.19
Shallow mapping	71.35±7.20	27.71±1.96	3.59±0.22
Adaptive dropout	67.23±4.76	23.33±0.98	4.16±0.05
ADA (Ours)	**16.49**±0.65	**8.29**±0.31	3.88±0.13

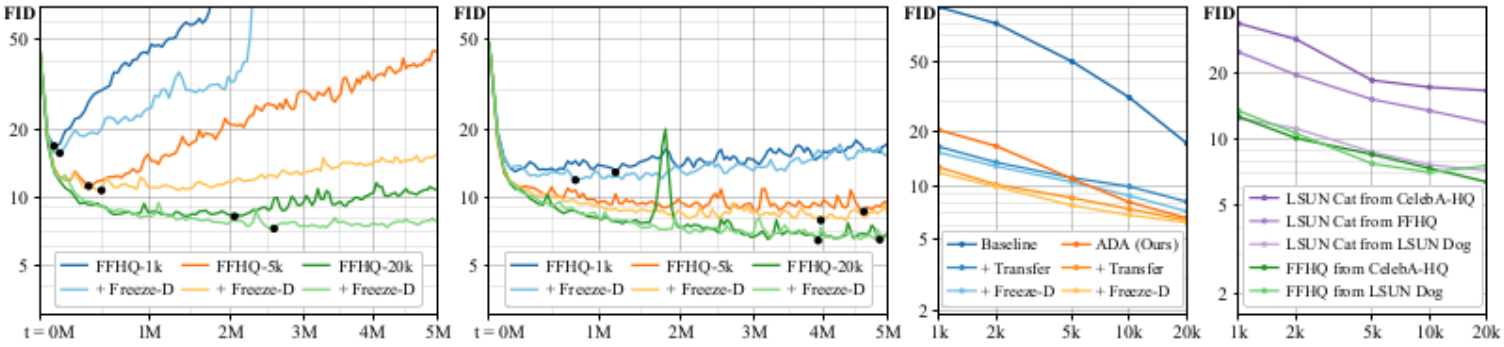
(a) Comparison methods



(b) Discriminator capacity sweeps

Figure 8: (a) We report the mean and standard deviation for each comparison method, calculated over 3 training runs. (b) FID as a function of discriminator capacity, reported as median/min/max over 3 training runs. We scale the number of feature maps uniformly across all layers by a given factor (x-axis). The baseline configuration (no scaling) is indicated by the dashed vertical line.

- solution is to use data augmentation for generator and discriminator
- run an algorithm: initializing $p = 0$, adjusting p every 4 epochs to keep $r_t \approx 0.6$



(a) Without ADA (b) With ADA (c) Dataset sizes (d) Datasets

Figure 9: Transfer learning FFHQ starting from a pre-trained CELEBA-HQ model, both 256×256. (a) Training convergence for our baseline method and Freeze-D [26]. (b) The same configurations with ADA. (c) FIDs as a function of dataset size. (d) Effect of source and target datasets.

- the effect of transfer learning for the discriminator
- freeze only the layers for the 3-4 highest resolutions, its a tunable parameter

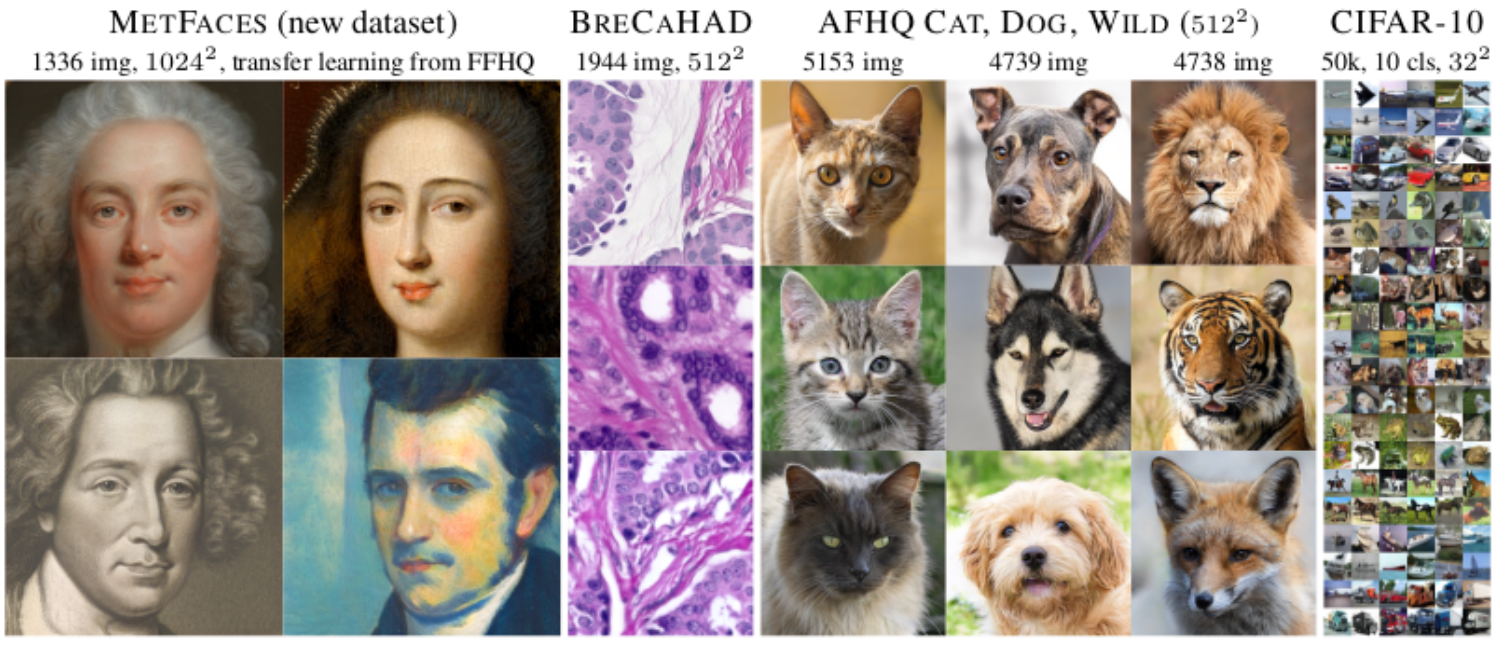


Figure 10: Example generated images for several datasets with limited amount of training data, trained using ADA. We use transfer learning with METFACES and train other datasets from scratch. See Appendix A for uncurated results and real images, and Appendix D for our training configurations.

- the effect of transfer learning for the discriminator
- freeze only the layers for the 3-4 highest resolutions, its a tunable parameter

Outline

Why inpainting?

What is required to know for GAN inpainting?

- what problem it solves
- challenges for this setup
- not the model details

Why inpainting?

- create content subject to constraints
- can iteratively postprocess bad regions using different+specialized generators (e.g. for vegetation, building facades, ears of humans, teeth, water waves)

Challenges in inpainting I

- local consistency of the inpainted patch

- solution: use a discriminator on the patch

- global consistency of the image as a whole

- solution: use a discriminator on the whole image

- informing the generators about local statistics of the image

- e.g. what textures / structures should appear in the white patches?
- how to build a model that can provide such an information to the generative part of it?

https://arxiv.org/abs/1801.07892 Yu et al. CVPR 2018

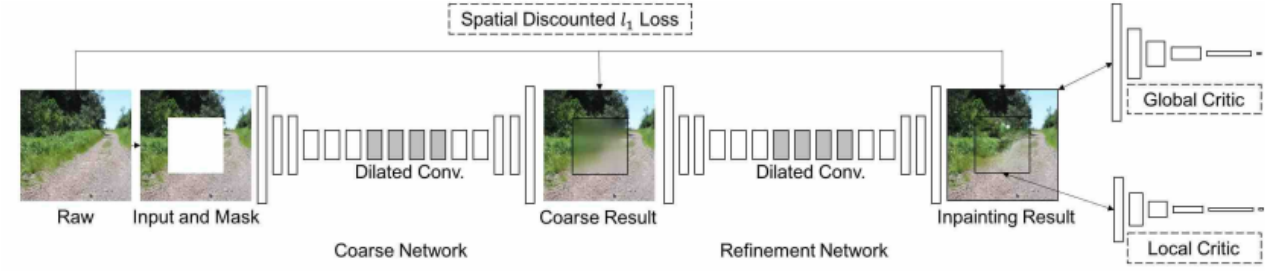
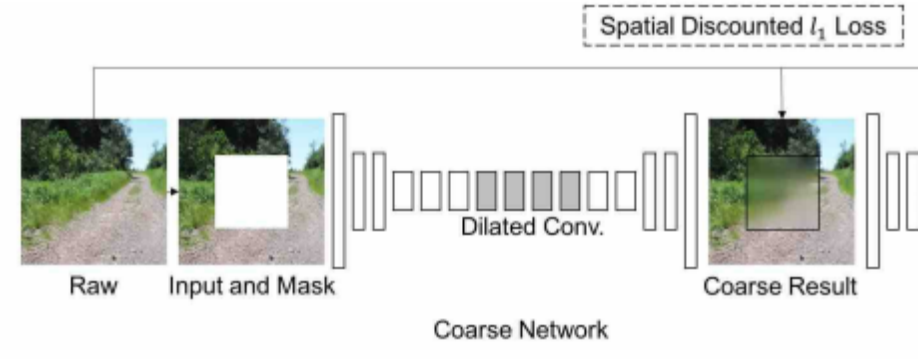


Figure 2: Overview of our improved generative inpainting framework. The coarse network is trained with reconstruction loss explicitly, while the refinement network is trained with reconstruction loss, global and local WGAN-GP adversarial loss.

Two stage architecture

- stage 1: direct, coarse reconstruction
- stage 2: apply reconstruction loss and two discriminators

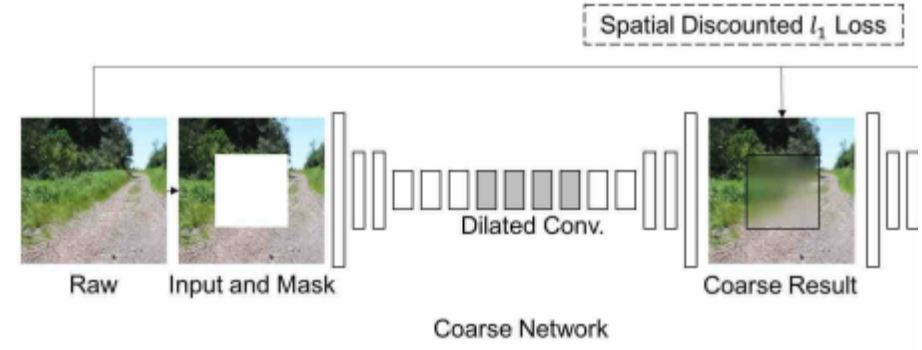


- stage 1 takes an image x as input, adds white patches $x \mapsto h(w)$ in random positions as input and reconstructs the image x based on $h(x)$ as $f(h(x))$ using the model $f(\cdot)$
- Training loss: weighted pixel-wise reconstruction loss

$$L(x, f(x)) = \|(f(h(x)) - x) \odot \gamma(h(x))\|_1$$

The weight $\gamma(h(x))$ is 1 on pixels without white patches, and on white patches it decreases to zero as a function of the distance to the nearest non-masked pixel:

$$\gamma(h(x)) = \gamma^l, \; l = \text{pixel distance to nearest non-masked pixel}$$

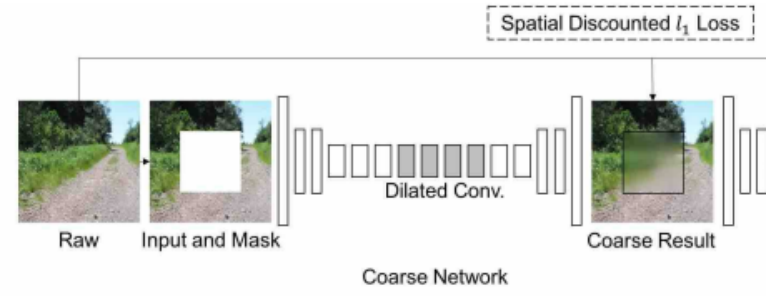


- stage 1 takes an image x as input, adds white patches $x \mapsto h(w)$ in random positions as input and reconstructs the image x based on $h(x)$ as $f(h(x))$ using the model $f(\cdot)$
- Training loss: weighted pixel-wise reconstruction loss

$$L(x, f(x)) = \|(f(h(x)) - x) \odot \gamma(h(x))\|_1$$

$$\gamma(h(x)) = \gamma^l, \; l = \text{pixel distance to nearest non-masked pixel}$$

- Idea: less weight on reconstruction quality for the coarse first stage for pixels far away from a non-masked pixel. Enforce a more truthful reconstruction on unmasked pixels and close to them.



Inpainting network Inpainting network has two encoder-decoder architecture stacked together, with each encoder-decoder of network architecture:
K5S1C32 - K3S2C64 - K3S1C64 - K3S2C128 - K3S1C128 - K3S1C128 - K3D2S1C128 - K3D4S1C128 - K3D8S1C128 - K3D16S1C128 - K3S1C128 - K3S1C128 - resize (2×) - K3S1C64 - K3S1C64 - resize (2×) - K3S1C32 - K3S1C16 - K3S1C3 - clip.

Prediction model:

- takes an image with random white patches, outputs an image
- start: 5 standard convolution layers, 2 with stride 2. Result: feature map compressed to 1/4.

  Creates a bottleneck: lower resolution, higher no. feature channels.
- then 4 dilated convolution layers.
- then (conv-conv-upsample 2x) – repeated two times. to obtain the original image size back.
- then 3x convolutions, last with output with 3 channels

Dilated convolutions?
https://github.com/vdumoulin/conv_arithmetic

- dilation factor 1: take every pixel from input feature map (the usual!)
- dilation factor 2: take every second pixel from input feature map
- dilation factor n: take every $n-th$ pixel from input feature map
- allows to process larger fields of view – good for reconstruction tasks to obtain a multiscale global view of the information in the image

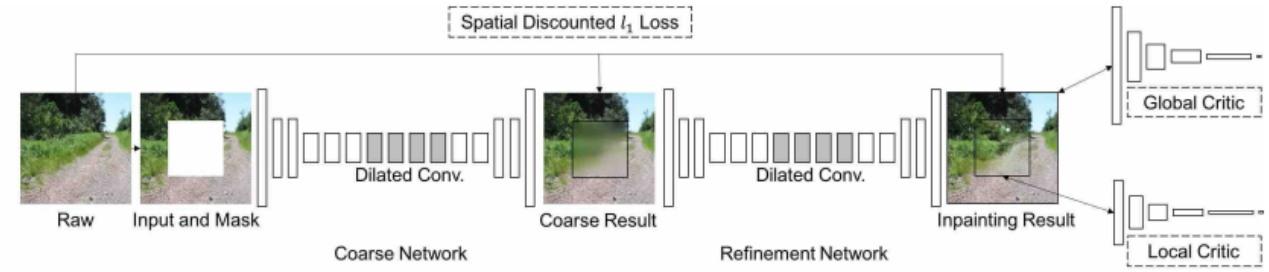


Figure 2: Overview of our improved generative inpainting framework. The coarse network is trained with reconstruction loss explicitly, while the refinement network is trained with reconstruction loss, global and local WGAN-GP adversarial loss.

Training loss: three losses summed

- loss 1: weighted pixel-wise reconstruction loss $\|(f(h(x)) - x) \odot \gamma(h(x))\|_1$
- loss 2: discriminator on patch. WGAN-GP :)
- loss 3: discriminator on the whole image. WGAN-GP

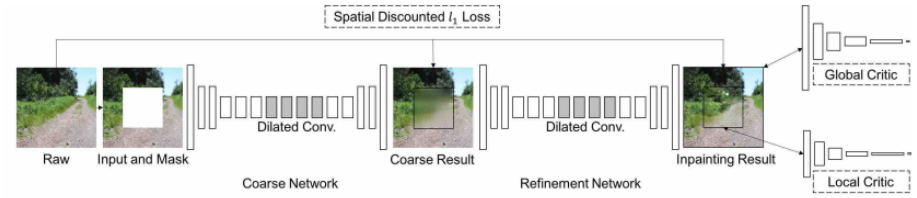
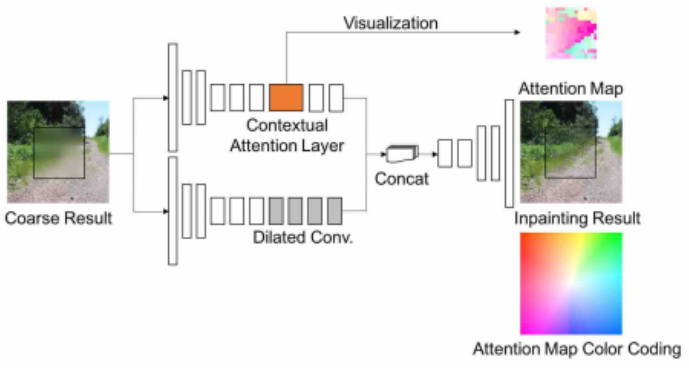
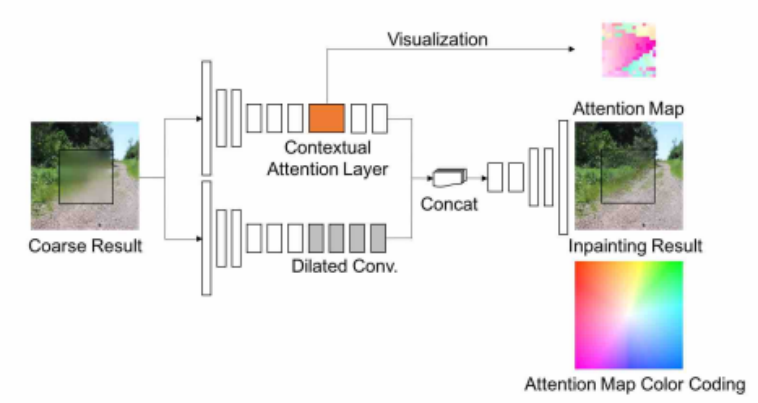


Figure 2: Overview of our improved generative inpainting framework. The coarse network is trained with reconstruction loss explicitly, while the refinement network is trained with reconstruction loss, global and local WGAN-GP adversarial loss.

Finer stage prediction model:

- two streams, concatenated before final 3 convolutions down to three channels, which are same final convolutions as in stage 1 model.
- stream 1: 5 convolutions, dilations, (conv-conv-upsample) twice repeated as in stage 1 model. Without stream 2 it would be the same model again.
- stream 2: informs about patch statistics from non-masked regions, explained next!





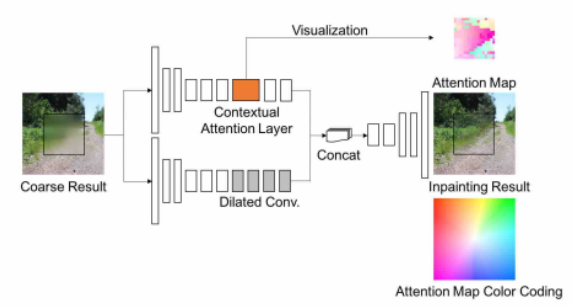
Stream 2: contextual attention. idea: compute how good a patch from coarsely reconstructed image matches any of the background patches from the non-masked regions of this image.

- if we had a background patch of spatial size 3×3 in some feature space f_b from the non-masked regions of this image, and a patch f_v from the coarsely reconstructed image, then we could compute the cosine angle as similarity measure

$$s(f_b, f_v) = \frac{f_b \cdot f_v}{\|f_b\| \|f_v\|}$$

- if we had N background patches, we could compute a softmax of similarities over all those N patches $f_b^{(i)}$ from non-masked regions:

$$p(f_b^{(1)}, f_b^{(2)}, \ldots, f_b^{(N)} | f_v) = softmax\left(s(f_b^{(1)}, f_v), \ldots, s(f_b^{(N)}, f_v)\right)$$



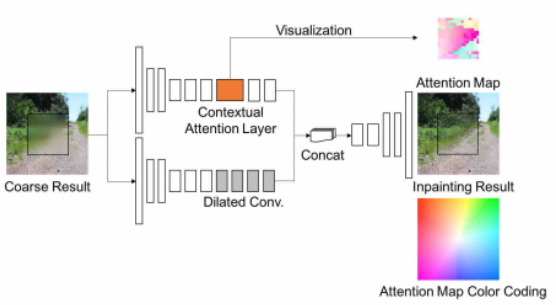
Stream 2: contextual attention. idea: compute how good a patch from coarsely reconstructed image matches any of the background patches from the non-masked regions of this image.

- if we had N background patches, we could compute a softmax of similarities over all those N patches $f_b^{(i)}$ from non-masked regions:

$$s(f_b, f_v) = \frac{f_b \cdot f_v}{\|f_b\| \|f_v\|}$$

$$p(f_b^{(1)}, f_b^{(2)}, \ldots, f_b^{(N)} | f_v) = softmax\left(s(f_b^{(1)}, f_v), \ldots, s(f_b^{(N)}, f_v)\right)$$

- This tells a probability of similarities of background patches (non-masked regions) to f_v
- how to implement that efficiently?

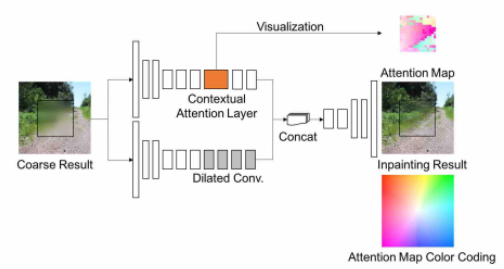


Stream 2: contextual attention. idea: compute how good a patch from coarsely reconstructed image matches any of the background patches from the non-masked regions of this image.

- if we had N background patches, we could compute a softmax of similarities over all those N patches $f_b^{(i)}$ from non-masked regions:

$$p(f_b^{(1)}, f_b^{(2)}, \ldots, f_b^{(N)} | f_v) = softmax\left(s(f_b^{(1)}, f_v), \ldots, s(f_b^{(N)}, f_v)\right)$$

- This tells a probability of similarities of bg patches to f_v. How to implement that efficiently?
- extract N background patches of spatial size 3×3 in some feature space with c channels (after 5 initial convolutions). result: a feature map of shape $(n, c, 3, 3)$. Make them kernels in a convolution with n output channels

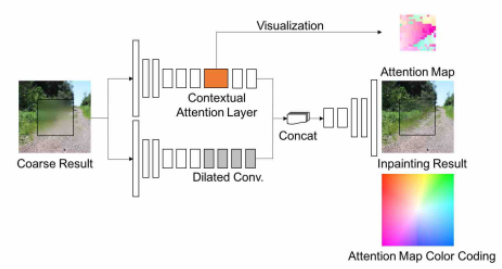


Stream 2: contextual attention. idea: compute how good a patch from coarsely reconstructed image matches any of the background patches from the non-masked regions of this image.

- if we had N background patches, we could compute a softmax of similarities over all those N patches $f_b^{(i)}$ from non-masked regions:

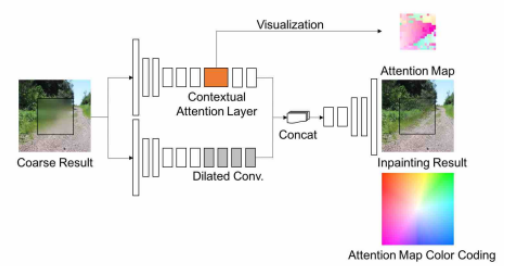
$$p(f_b^{(1)}, f_b^{(2)}, \ldots, f_b^{(N)} | f_v) = softmax\left(s(f_b^{(1)}, f_v), \ldots, s(f_b^{(N)}, f_v)\right)$$

- How to implement that efficiently?
- extract N background patches of spatial size 3×3 in some feature space with c channels (after 5 initial convolutions). Result: a feature map of shape $(n, c, 3, 3)$. Make them kernels in a convolution with n output channels
- run this convolution over the whole feature map to get inner product scores $(f_b^{(1)} \cdot f_v, \ldots, f_b^{(N)} \cdot f_v)$ for the whole image



Stream 2: contextual attention. idea: compute how good a patch from coarsely reconstructed image matches any of the background patches from the non-masked regions of this image.

- extract N background patches of spatial size 3×3 in some feature space with c channels (after 5 initial convolutions). Result: a feature map of shape $(n, c, 3, 3)$. Make them kernels in a convolution with n output channels
- run this convolution over the whole feature map to get inner product scores $(f_b^{(1)} \cdot f_v, \ldots, f_b^{(N)} \cdot f_v)$ for the whole image
- computations: $(N, c, 3, 3) \star (1, c, h, w) \mapsto (1, N, h, w)$ (...padding). N channels. At position (h', w') of the feature map $(1, N, h, w)$ it tells N similarities to the N background patches - each in its own channel $i \in \{1, \ldots, N\}$.

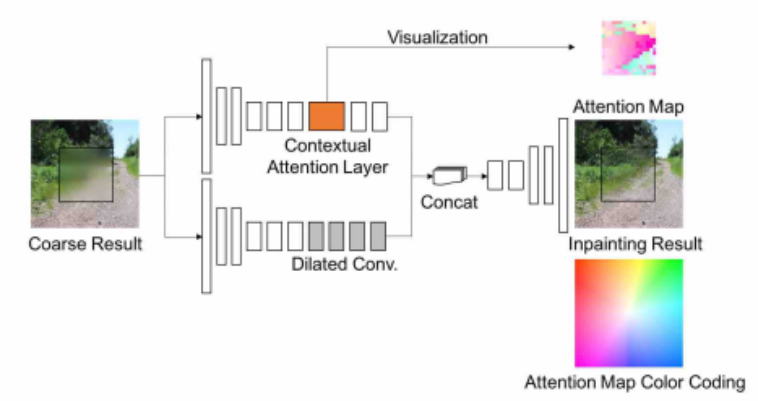


Stream 2: contextual attention. idea: compute how good a patch from coarsely reconstructed image matches any of the background patches from the non-masked regions of this image.

- if we had N background patches, we could compute a softmax of similarities over all those N patches $f_b^{(i)}$ from non-masked regions:

$$p(f_b^{(1)}, f_b^{(2)}, \ldots, f_b^{(N)} | f_v) = softmax\left(s(f_b^{(1)}, f_v), \ldots, s(f_b^{(N)}, f_v)\right)$$

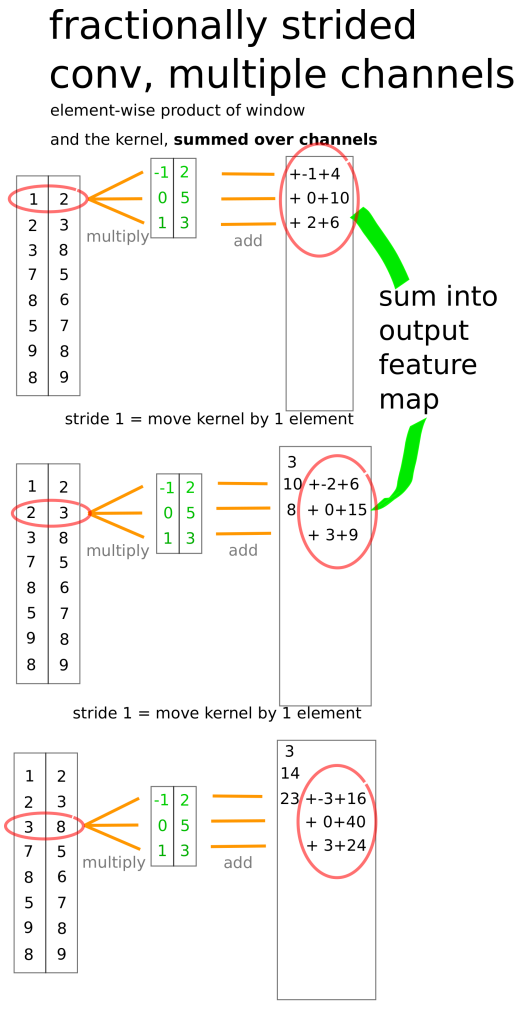
- extract N background patches of spatial size 3×3 in some feature space with c channels (after 5 initial convolutions). Result: a feature map of shape $(n, c, 3, 3)$. Make them kernels in a convolution with n output channels
- run this convolution over the whole feature map to get inner product scores $(f_b^{(1)} \cdot f_v, \ldots, f_b^{(N)} \cdot f_v)$ for the whole image
- normalize feature maps, apply channel-wise softmax to get $p(f_b^{(i)} | f_v), i \in \{1, \ldots, N\}$

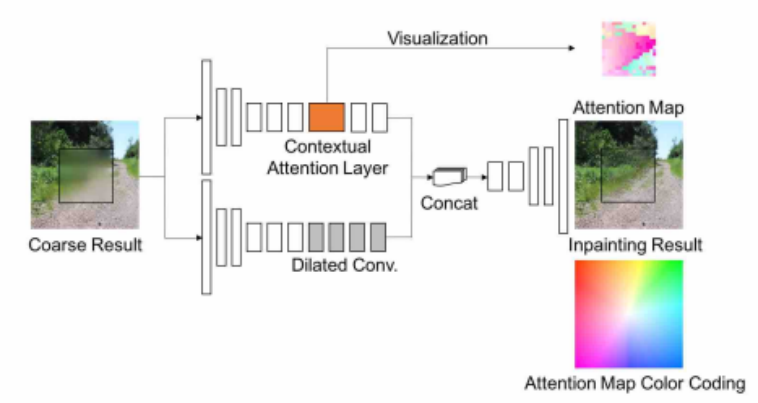


Stream 2: contextual attention. idea: compute how good a patch from coarsely reconstructed image matches any of the background patches from the non-masked regions of this image.

- now we have the softmax scores $p(f_b^{(i)}|f_v)$ for every position in the image
- smoothen the scores by sum pooling over neighboring pixels ('attention propagation')
- use them as input for a fractionally strided convolution layer (deconvolutional filters in the paper). The weights of the fractionally strided convolution layer are fixed to be the extracted background patches $f_b^{(1)}, f_b^{(2)}, \ldots, f_b^{(N)}$
- What ? This computes then in the resulting output feature map a weighted copy of the kernels!

For multiple input channels the working principle is analogous.





Stream 2: contextual attention. idea: compute how good a patch from coarsely reconstructed image matches any of the background patches from the non-masked regions of this image.

- now we have the softmax scores $p(f_b^{(i)}|f_v)$ for every position in the image
- use them as input for a fractionally strided convolution layer. The weights of the fractionally strided convolution layer are fixed to be the extracted background patches $f_b^{(1)}, f_b^{(2)}, \ldots, f_b^{(N)}$

  This computes then in the resulting output feature map a weighted copy of the kernels!

  !

- example on how to use neural network toolbox layers to achieve a similarity measuring task within a network.

How to include an object type GAN ?

- as third stream in the second stage
- use a feature map over the coarse result to input local image statistics into the GAN generator code z.
- in principle a user could select from which region to sample background patches to inform the inpainter. Does not need to be fixed!
- fusing GANs with neural style transfer can be useful in GAN inpainting beyond pure artistic creation goals.

Points taken here from:

- local critic (masked patch level)
- global critic
- how to obtain local statistics and use them to inform a generative process
- how to do that with neural network layers (convolution, fractionally strided convolution)
- think how to put a user in the loop at testing time (beyond PhD research)

Questions?!