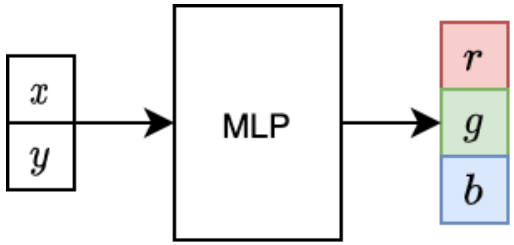
Meta Generation

Ivan Skorokhodov

August 31, 2020

Implicit Neural Representations

- INR is a neural network $f_\varphi(x, y)$ which takes coordinates (x, y) and produces a pixel value:



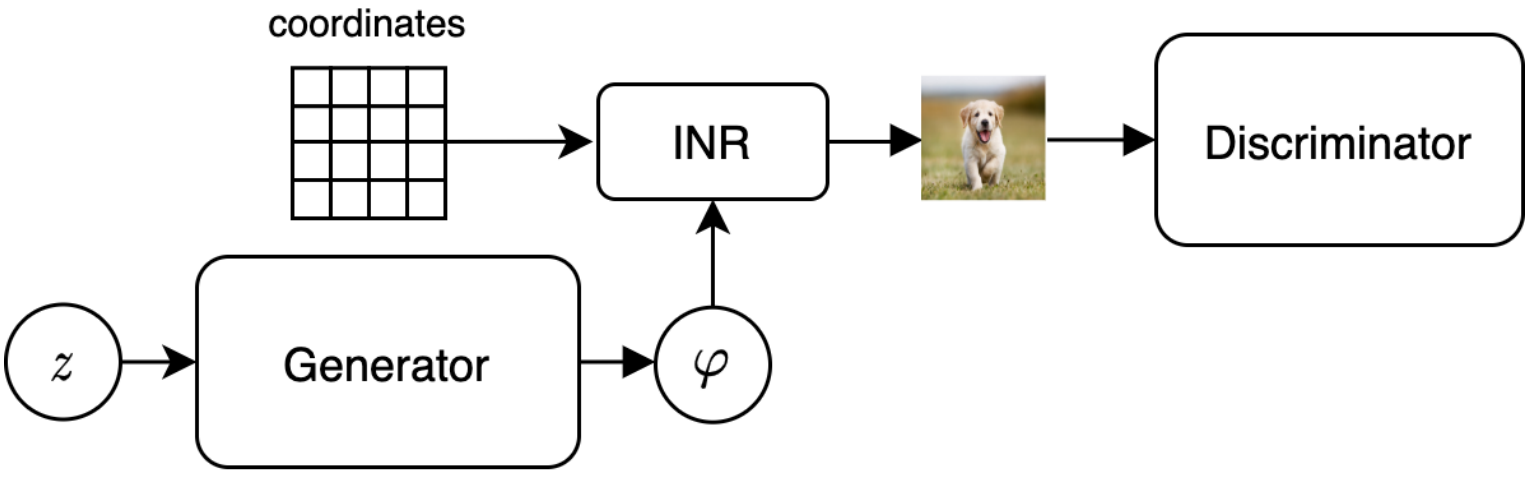
- We can generate the whole image by computing the value of $f_\varphi(x, y)$ at each coordinate
- I.e. we have *1 image = 1 INR*

Benefits of INRs

- They carry more "complete" information about a signal:
  - Usual 2D arrays of pixels are discretized version of a signal
  - INRs are continuous and carry information "between" the pixels
  - This allows one to have "superresolution out-of-the-box"
- They are popular in 3D deep learning since they are cheaper to generate and operate with than 3D objects

INR-GAN

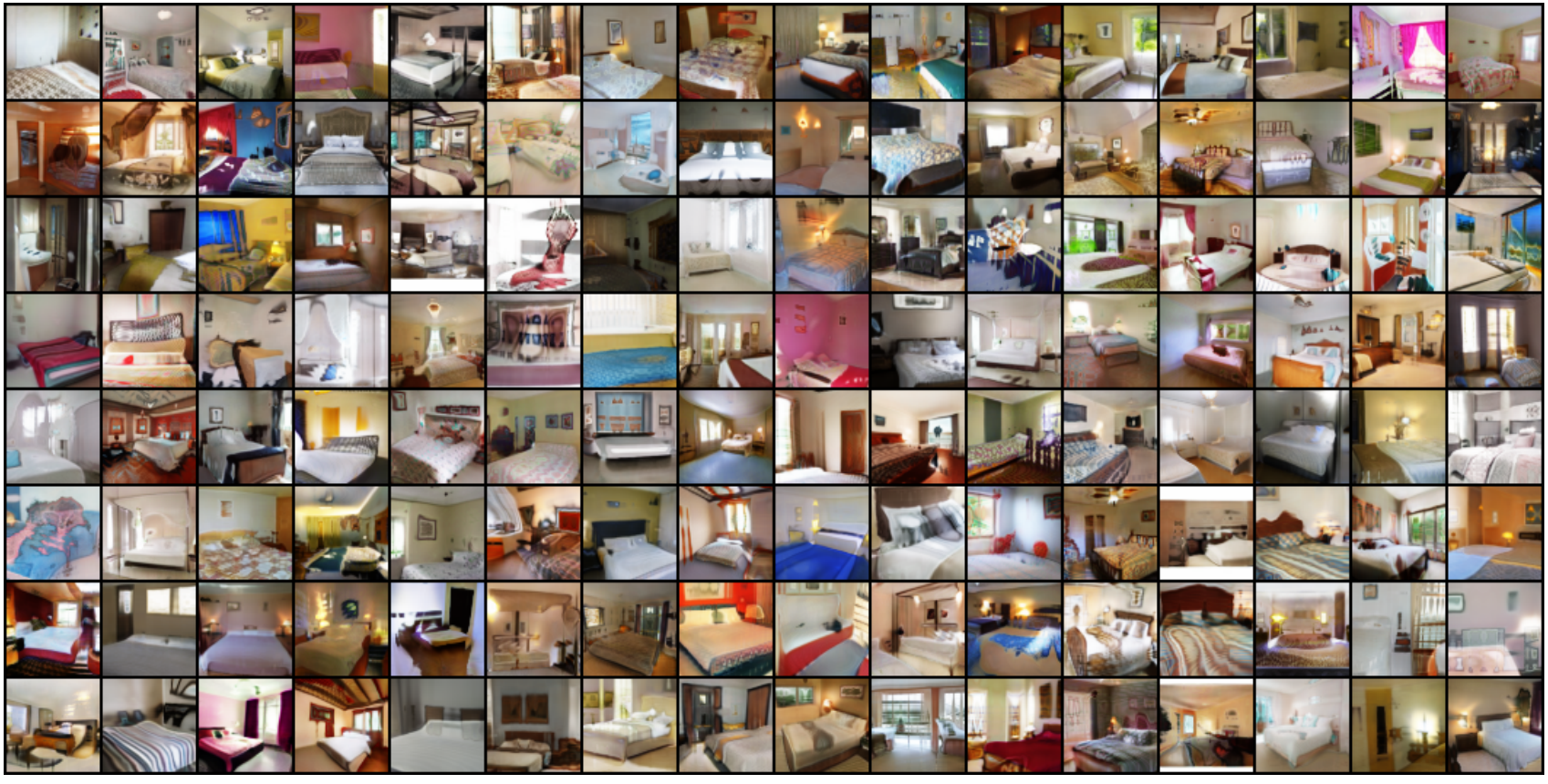
Train a generator to produce INRs



Advantages compared to traditional GANs

- A unified generator architecture for images/audio/video/3D-shapes
- Better biological plausibility
- We can generate different resolutions for different parts of an image
  - Imagine that we generate an image of a human on a sky background
  - We can use dense set of coordinates for a human and a sparse set of coordinates for the sky
  - It is good since it is impossible to do so for a normal GAN
- We can generate spherical images (and images on any surface)
- We can use "random resolutions" loss to train a high-resolution GAN
- We can train our generator on images of extreme resolution by using a "super-resolution" loss:
  - Train an INR-GAN normally on, for example, 256×256 images
  - Additionally, compute random dense patches and make the discriminator to distinguish between real/fake dense patches
  - (Provide global context if needed)
- It should be cheaper to use for some domains (audio, 3D, video, extreme-resolution images)
- Progressive growing is easier to incorporate

Samples on LSUN bedroom 64x64 (FID: 12.75)



Q: FourierINR vs SIREN

SIREN:

- Use $\sin(x)$ activation function everywhere
- Initialize the weights with $\sigma = \sqrt{2/n_{\text{in}}}$
- For the first layer, use $\sigma_1 = \sigma/30$ and multiply the activations by 30 afterwards. Why? Do we suffer from small gradients when we use $\sin(x)$?

Fourier INR:

- Use $\sin(x)$ and $\cos(x)$ for the first layer (and concat them together)
- Use traditional layers and activations on top of that
- Initialize the first layer with $\sigma \sim 100$.

Which one do you like more?

Size/quality tradeoff for INR architecture

Type	MAE (50 imgs)	#params
Linear	0.047	$n^2 + n$
SELinear	0.050	$n^2 + n$
mFiLM	0.914	$3n$
mFiLM -norm	0.656	$3n$
mFiLM -norm +scale	0.089	$3n$
Factorized ($r = 10$)	0.285	$2nr + n$
FactorizedSE ($r = 10$)	0.059	$2nr + n$
AdaIN	0.089	$2n$

All of them ran in 3.6 iters/sec on 1080 Ti and used 10 GB GPU memory[1].

[1] Except for AdaIN, which took more memory for some reason.

Changes overview

Implementing some tricks increased FID from ≈150 to 58 at 20k iterations on LSUN conference room[2]:

- SIREN → FourierINR
- Generator EMA (had to remove BatchNorm)
- WGAN-GP → R1
- D5:G1 → D1:G1 ticks
- batch size 32 → batch size 64
- LRs 0.002/0.0002 → 0.0001/0.0002 for G/D (like in StyleGAN)[3]
- INR size $64 \times 5 \rightarrow 256 \times 5$
- INRLinear → INRFactorizedSELinear
- WGAN loss → NS loss (results are in progress)
- Residual weight 0.0 → 0.1

[2] I didn't test each change individually

[3] What LR should we use for G? Is it equivalent to the mapping network?

Model samples for LSUN bedroom 128x128

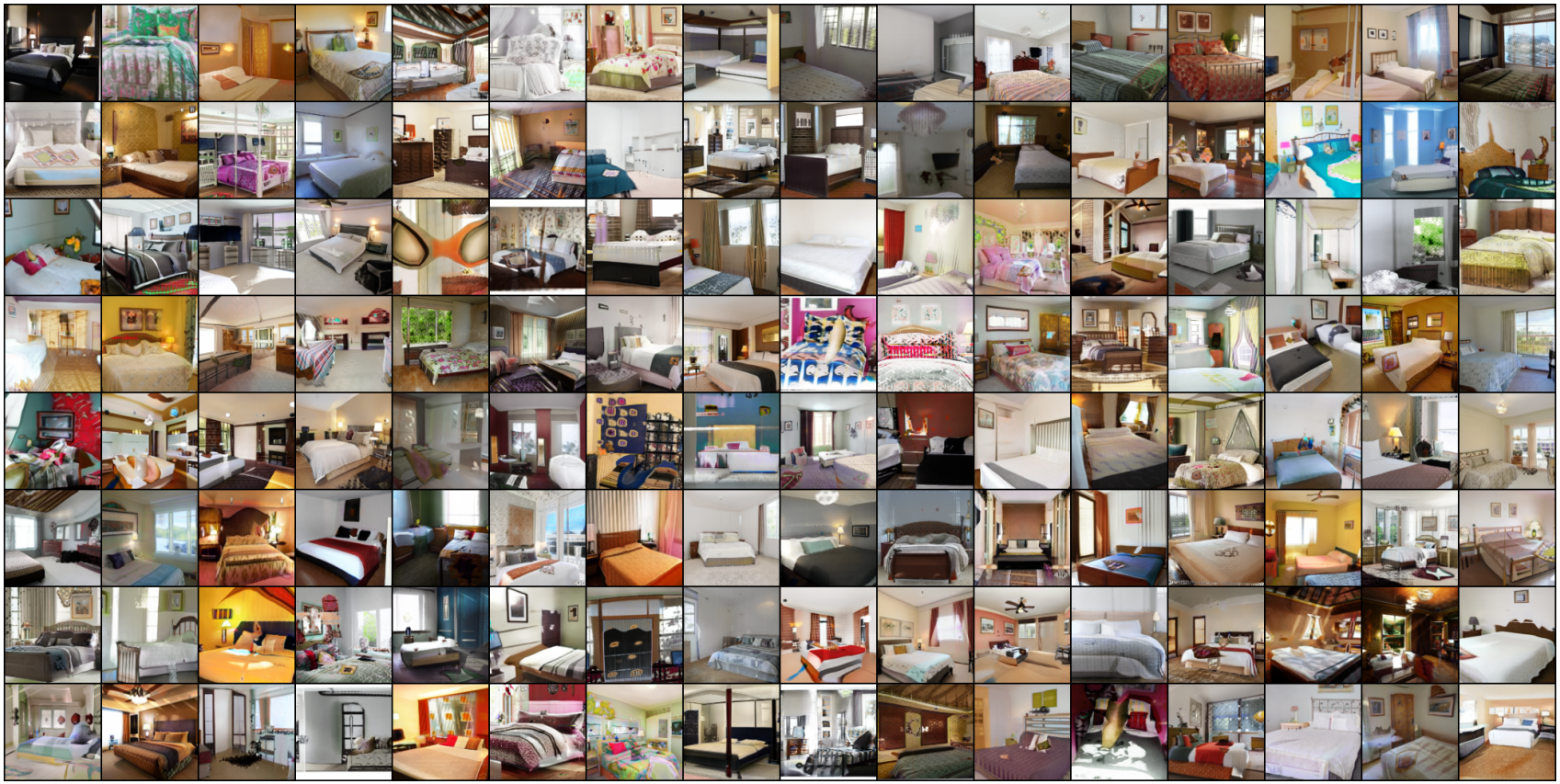


Figure: Model samples for LSUN bedroom 128x128 after 500k iterations. FID is 7.2

Bluntly increasing the model size

Bluntly increasing the model size does not work:

- For INR, increasing the size from $256 \times 5 \to 256 \times 10$ resulted in FID degradation: $12 \to 15$ FID after 40k iters
- For G, switching from 1024×5 to:
  - 2048×3 backbone
  - 7 independent heads of 1024×3

  resulted in FID degradation: $12 \to 35$ after 50k iters
- For D, increasing the size from 10M parameters to 30M parameters didn't change FID

INR natural upsampling vs bilinear upsampling

Consider the following setup:

- Train an INR-GAN on 128×128 images
- Upsample to 256×256 with bilinear upsampling
- Upsample to 256×256 with INR "natural" upsampling (i.e. just use a denser coordinate grid as an INR input)

Result:

- FID for INR natural upsampling is ~ 2 times better than for the bilinear upsampling:
  - 22 vs 41.4 after 70k iterations on LSUN bedroom
- But it is ~ 2 times worse than model's FID score on 128×128 images
  - 22 vs 10.7 after 70k iterations on LSUN bedroom

How to improve INR natural upsampling?

Clipping G output

- Instead of $w = G(z)$ compute INR weights with:

$$w = \text{clip}(G(z), -2, 2)$$

- This is approximately $\pm 2\sigma$ for $\mathcal{N}(0, 1)$.

This does not change FID much ($8 \rightarrow 7.6$ after 300k iters on LSUN bedroom), but makes interpolations awful:

Deciding upon the contributions

What we are going to sell:

- Best known scores for a GAN with G consisting entirely from fully-connected layers
- We investigated different INR parametrizations and proposed novel FactorizedSE paramtrization
- We show that INR interpolates much better than bilinear/nearest/cubic interpolations
- We investigated scaling inside the INR
- We argue that INRs has good potential and will replace conv-based decoders someday (but leave these claims for future work):
  - an ability to train the model on extreme-scale images (4096×4096)
  - out-of-the-box interpolation
  - universal G architecture for any domain (images, video, audio, etc)

Overall, we pushed INR-GANs for image domain quite far. Our work democratizes INR-GANs.

Next steps

- Local patch discriminator: additional D that operates on local upsampled patches
  - This would make training 2 times slower. Can we alleviate this?
- Training on variable-sized images
  - We hope that traditional convolutions would have problems producing different-ratio images