Evaluation of GANs on Texture Generation for Computer Graphics

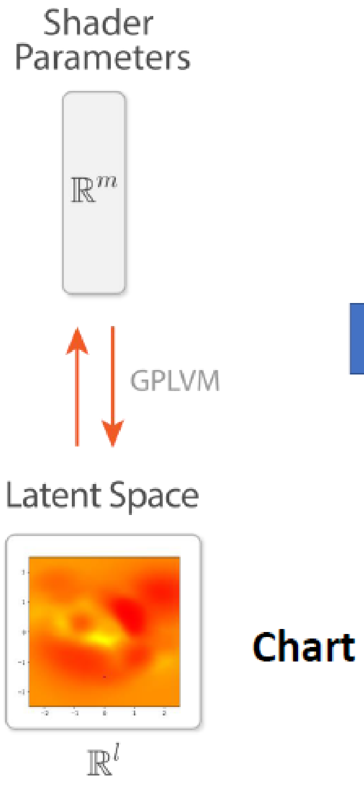
CS 482 Project Final Presentation

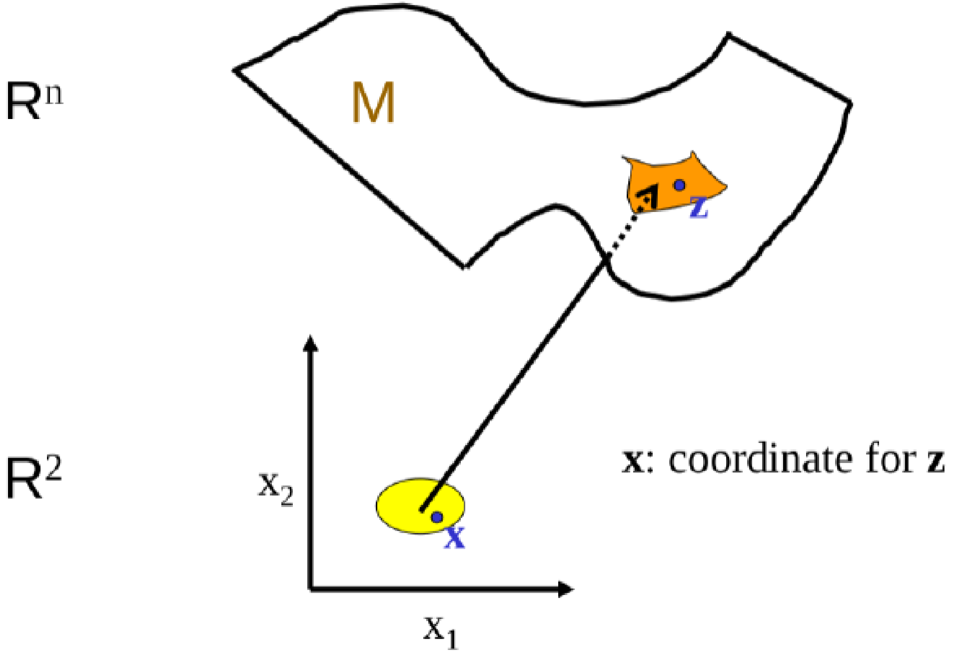
MinKu Kang

Goal of the Project

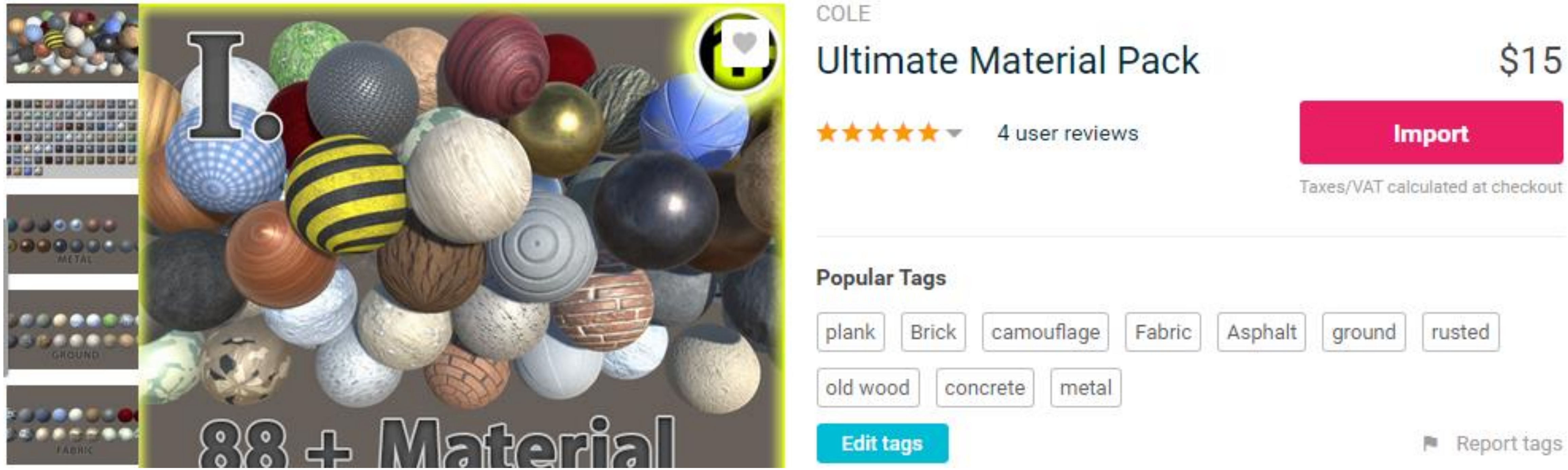
1. Find (learn) a meaningful **manifold**
2. FInd (learn) a **mapping** between the lowdimensional chart and the manifold

Material Training Data from Users

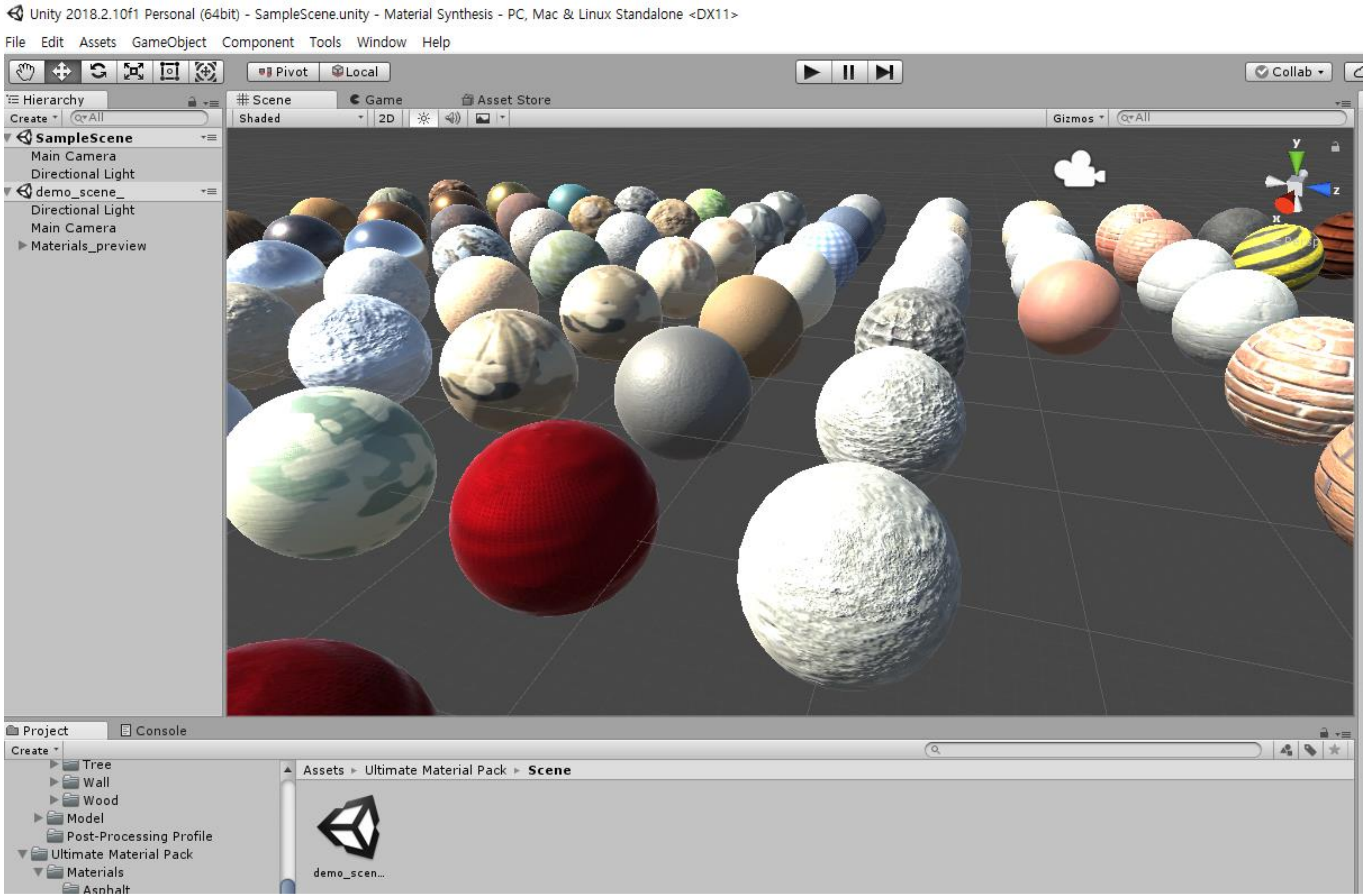




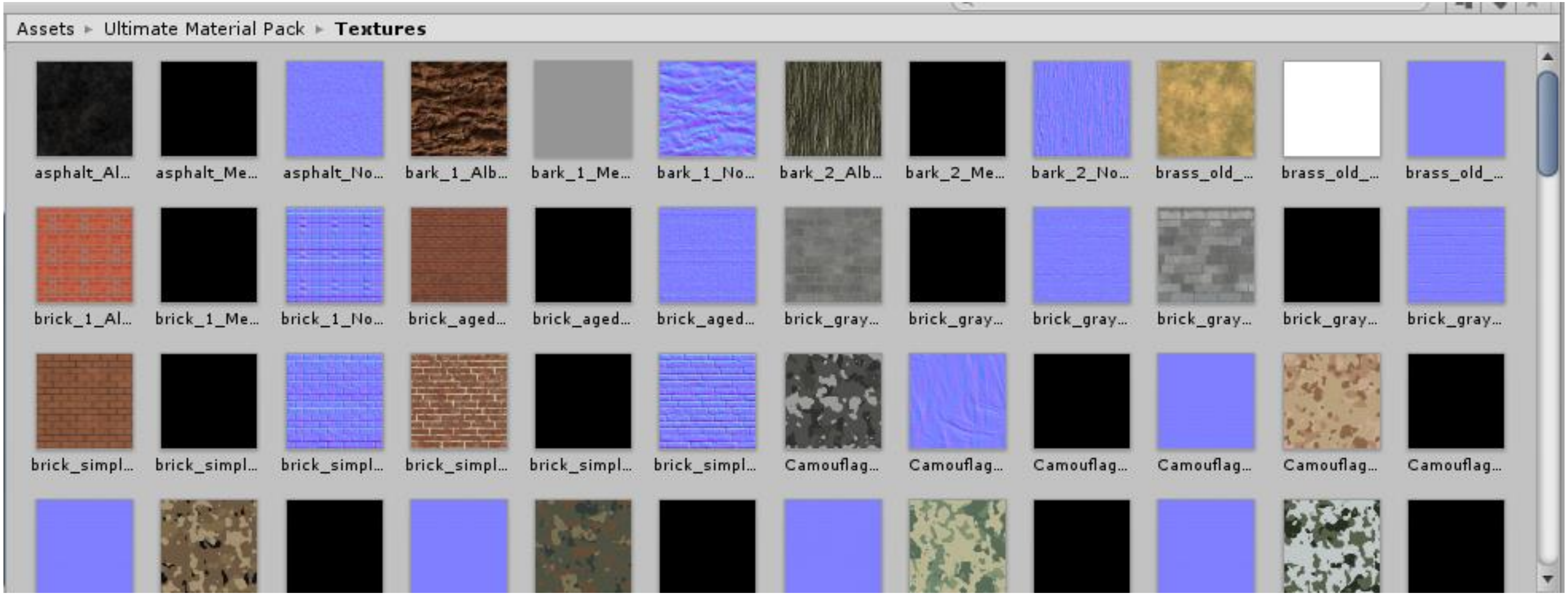
Purchased Material Pack from Unity Asset Store



Purchased Material Pack from Unity Asset Store

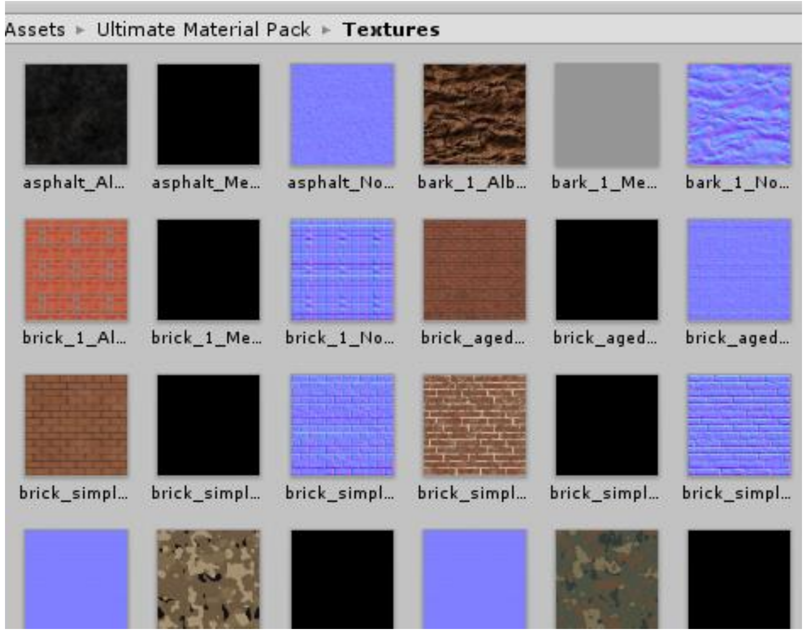


Purchased Material Pack from Unity Asset Store



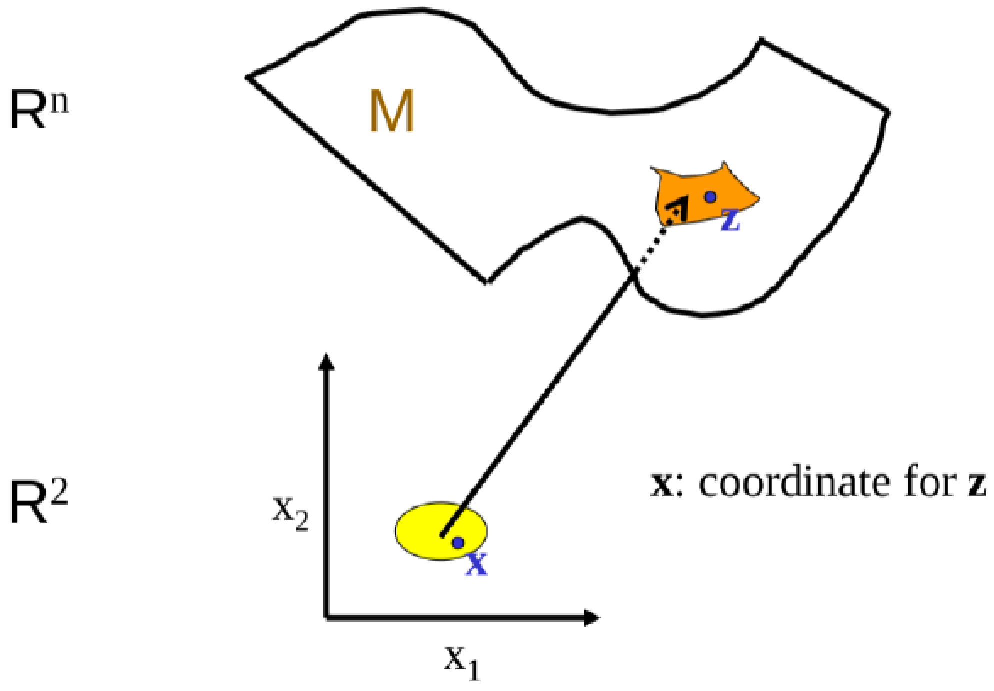
Various **texture maps** to give **fine surface details** on low-poly meshes

Modified Goal of the project



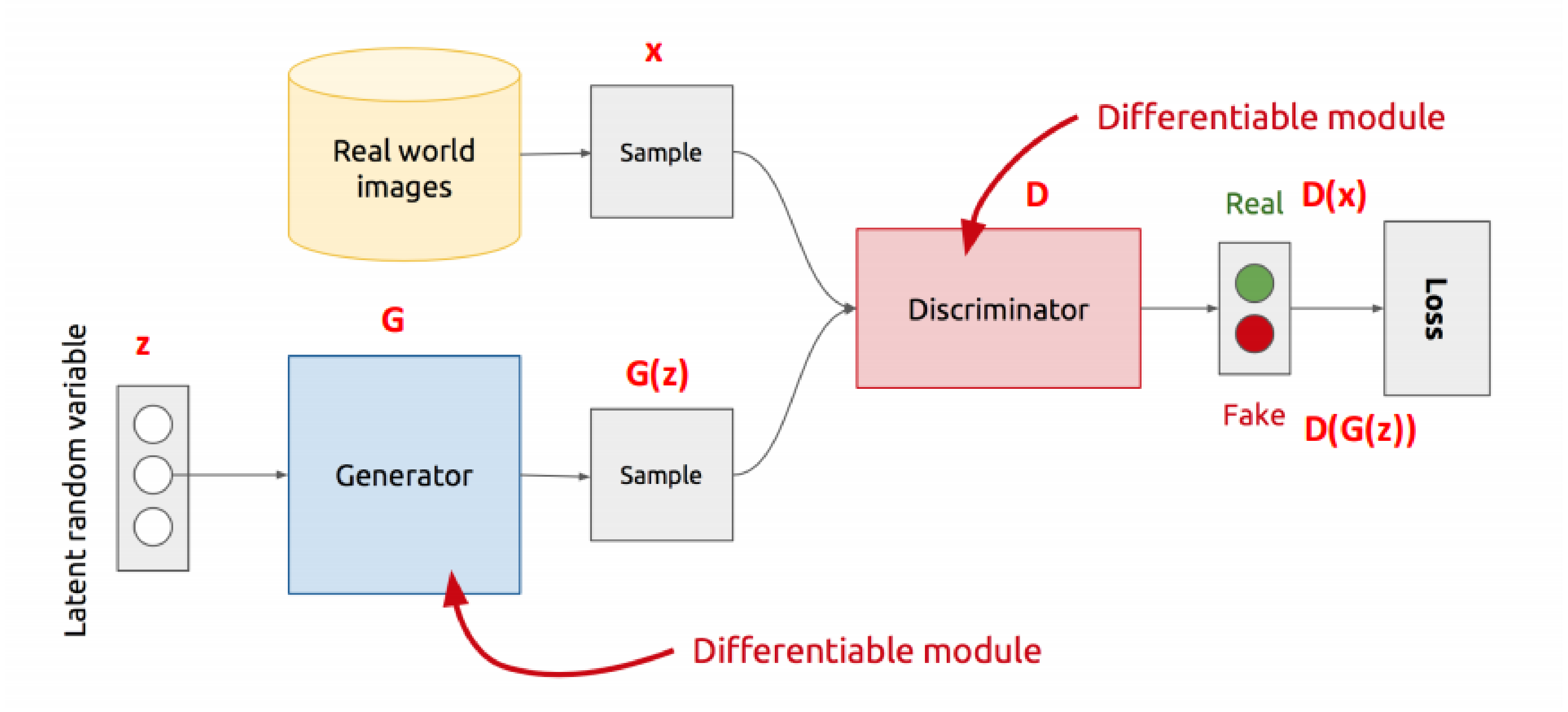
Use it as training data
for **material texture generation**

1. Find (learn) a meaningful **manifold**
2. FInd (learn) a **mapping** between the lowdimensional chart and the manifold



Brief Introduction on GAN

GAN's Architecture

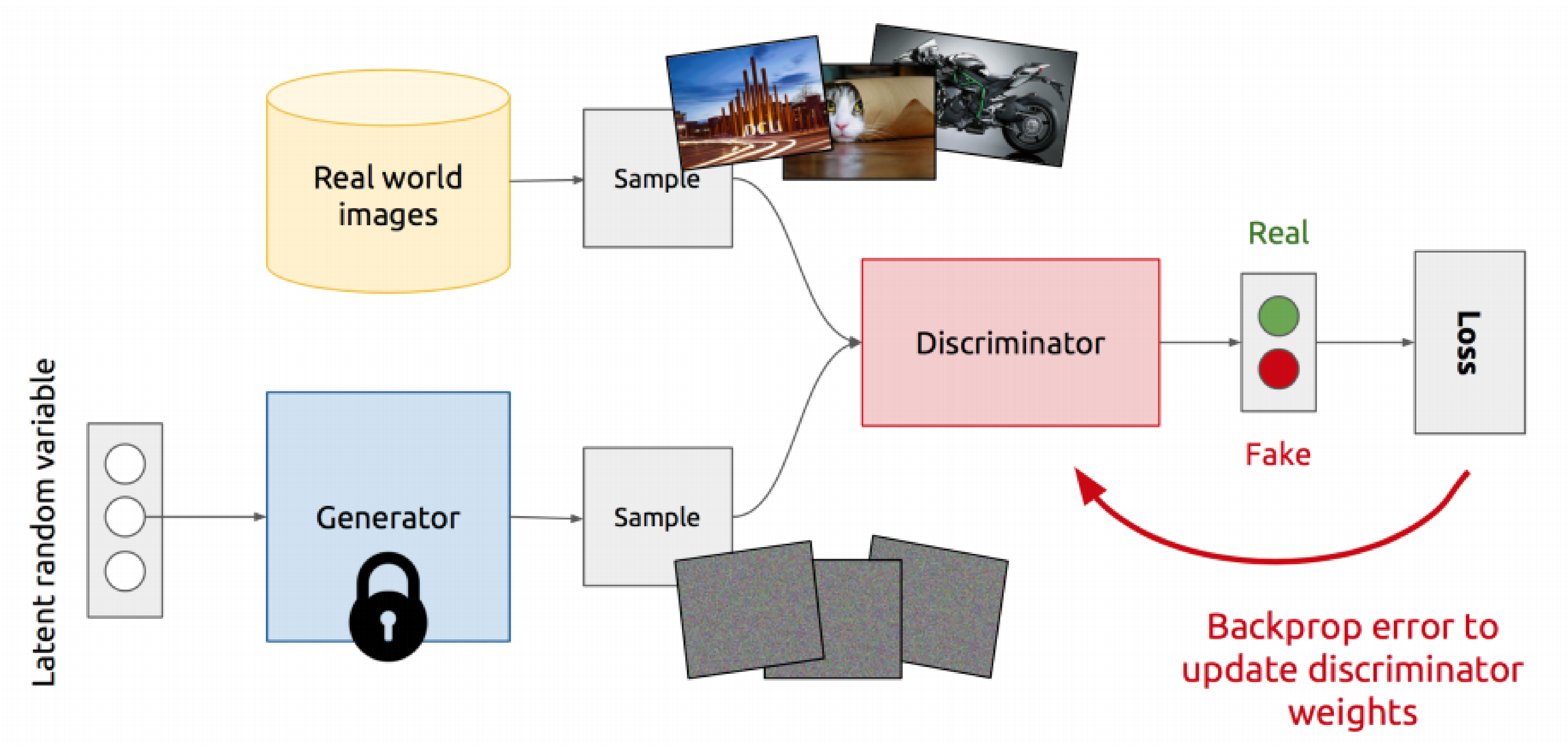


- **Z** is some random noise (Gaussian/Uniform).
- **Z** can be thought as the latent representation of the image.

https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

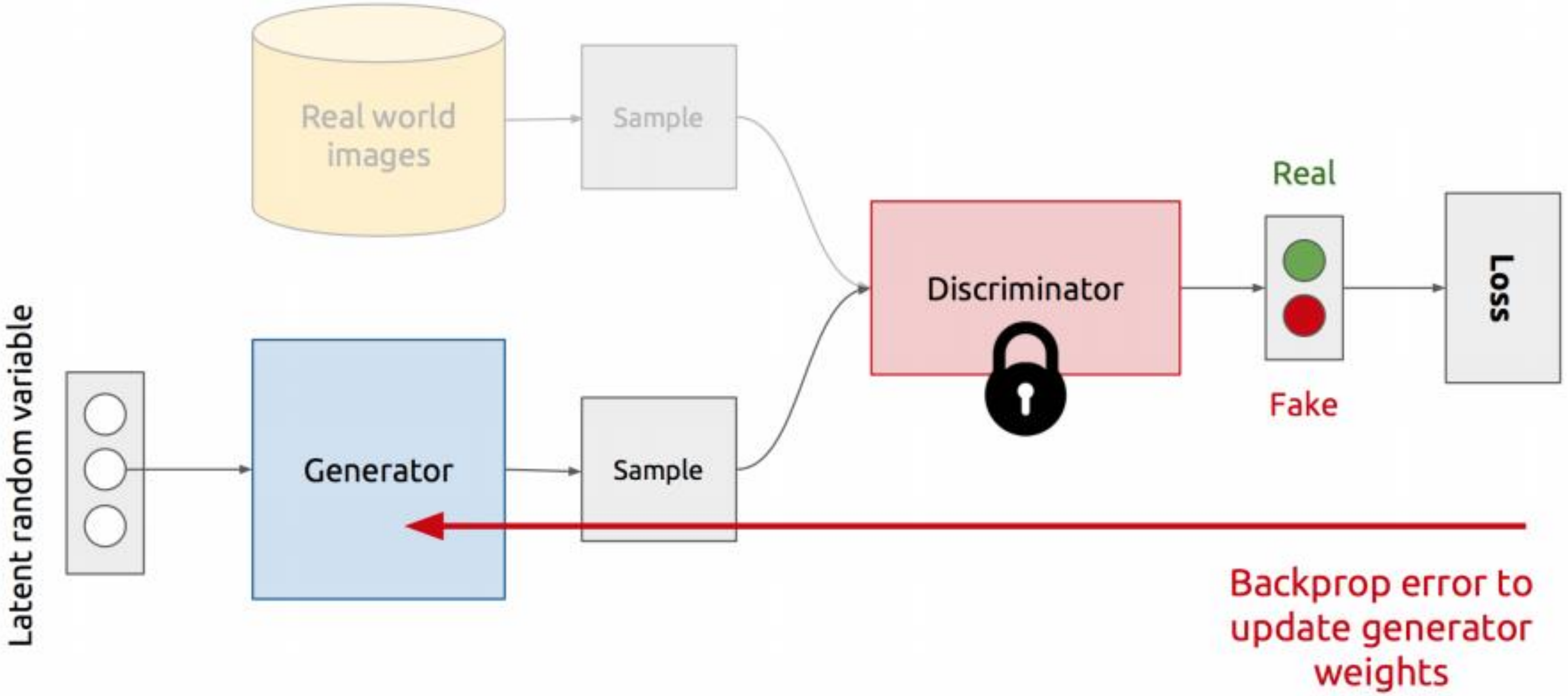
http://slazebni.cs.illinois.edu/spring17/lec11_gan.pdf

Training Discriminator



https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Training Generator



https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

GAN's formulation

$$\min_{G} \max_{D} V(D, G)$$

- It is formulated as a **minimax game**, where:
  - The Discriminator is trying to maximize its reward $\boldsymbol{V(D, G)}$
  - The Generator is trying to minimize Discriminator's reward (or maximize its loss)

$$V(D, G) = \boxed{\mathbb{E}_{x \sim p(x)}[\log D(x)]} + \boxed{\mathbb{E}_{z \sim q(z)}[\log(1 - D(G(z)))]}$$

- The Nash equilibrium of this particular game is achieved at:
  - $P_{data}(x) = P_{gen}(x) \ \forall x$
  - $\mathrm{D}(x) = \frac{1}{2} \ \forall x$

http://slazebni.cs.illinois.edu/spring17/lec11_gan.pdf

- **Deep Learning models (in general) involve a single player**
  - The player tries to maximize its reward (minimize its loss).
  - Use SGD (with Backpropagation) to find the optimal parameters.
  - SGD has convergence guarantees (under certain conditions).
  - **Problem:** With non-convexity, we might converge to local optima.

$$\min_{G} L(G)$$

- **GANs instead involve two (or more) players**
  - Discriminator is trying to maximize its reward.
  - Generator is trying to minimize Discriminator's reward.

$$\min_{G} \max_{D} V(D, G)$$

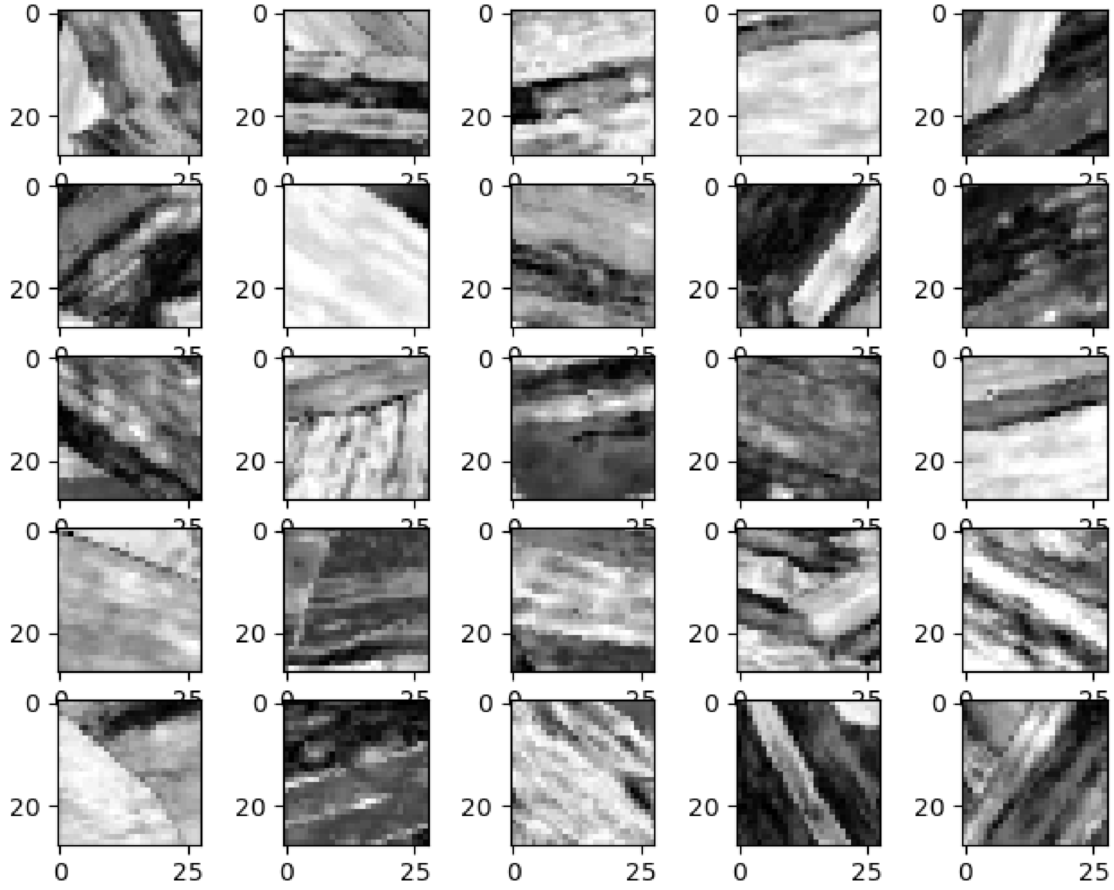
  - SGD was not designed to find the Nash equilibrium of a game.
  - **Problem:** We might not converge to the Nash equilibrium at all.

Salimans, Tim, et al. "Improved techniques for training gans." *Advances in Neural Information Processing Systems*. 2016.

Sample Collection

Sample Patches from a texture image

2048 x 2048 px
A single large texture

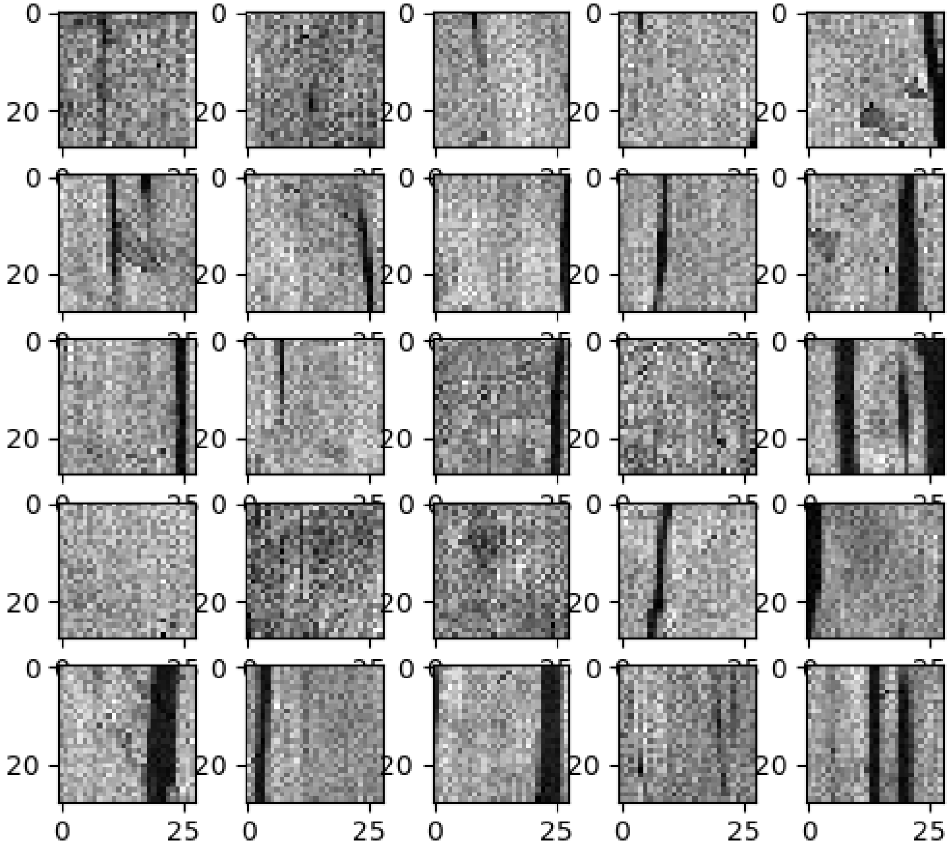


5,000 patches,
28 x 28 px each
gray-scaled

A half for training,
the other half for
testing

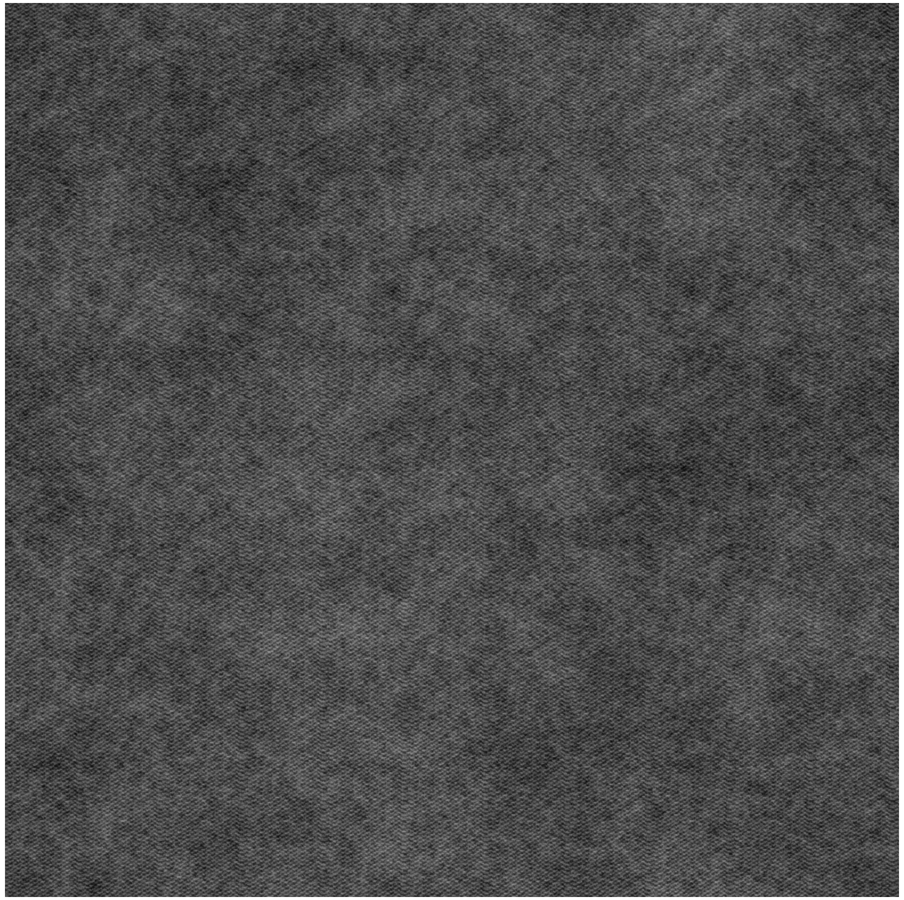
Sample Patches from a texture image

2048 x 2048 px
A single large texture

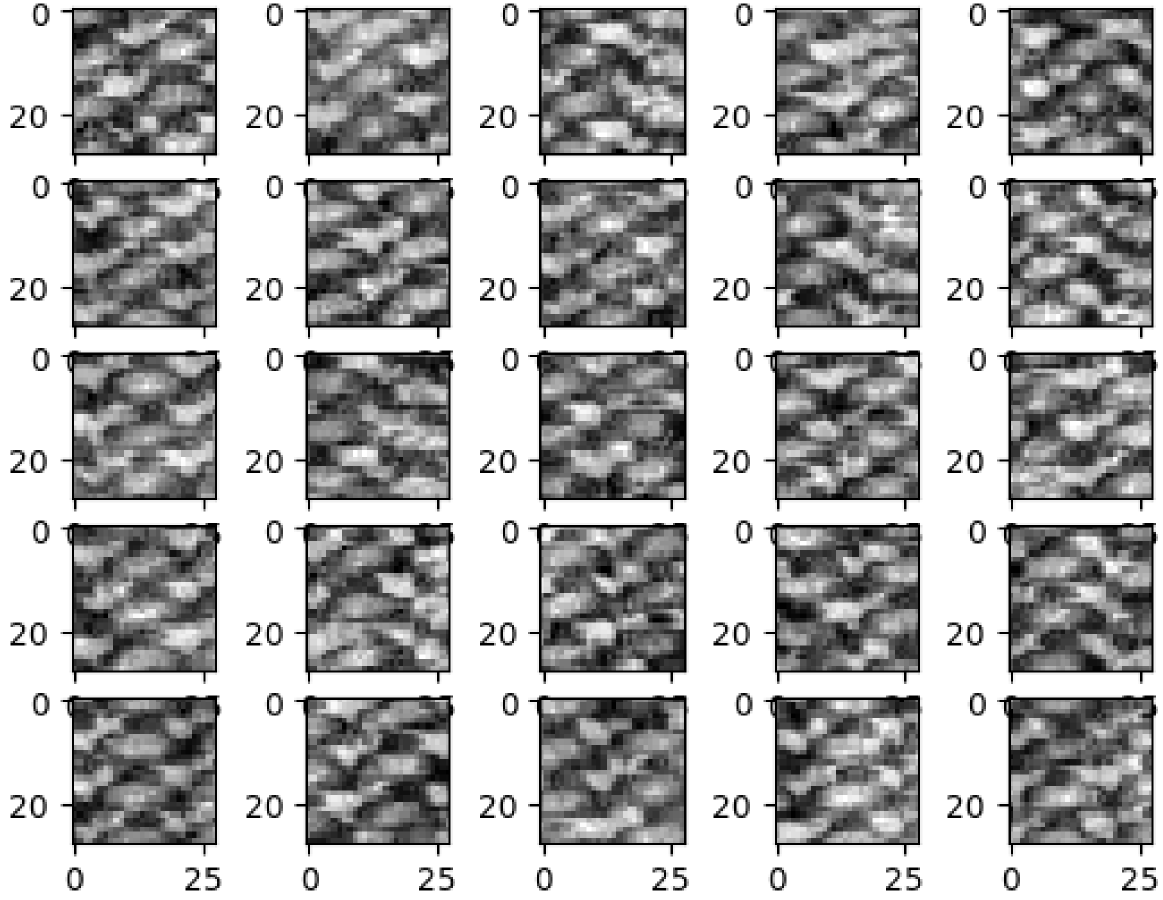


5,000 patches,
28 x 28 px each
gray-scaled

Sample Patches from a texture image

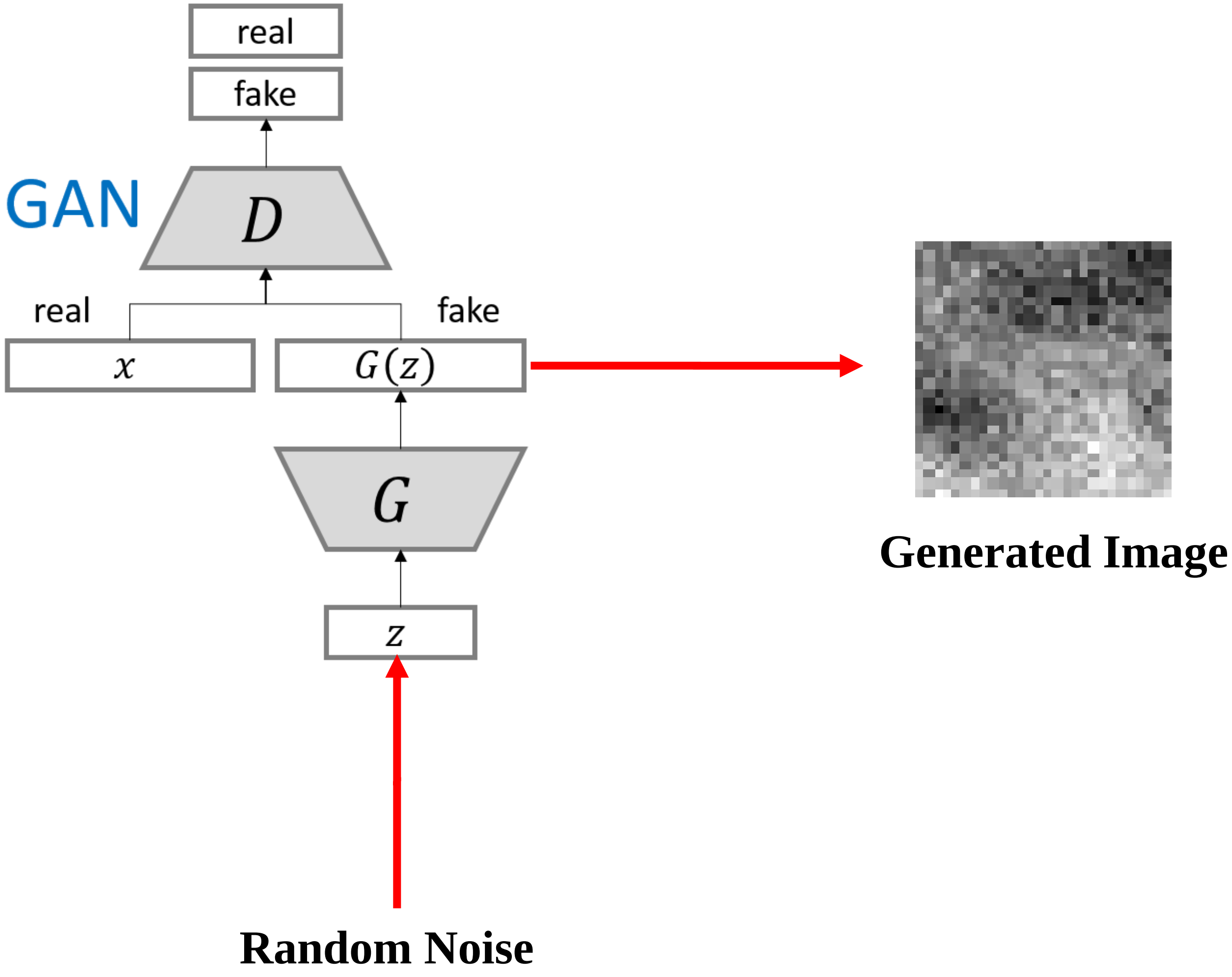


2048 x 2048 px
A single large texture



5,000 patches,
28 x 28 px each
gray-scaled

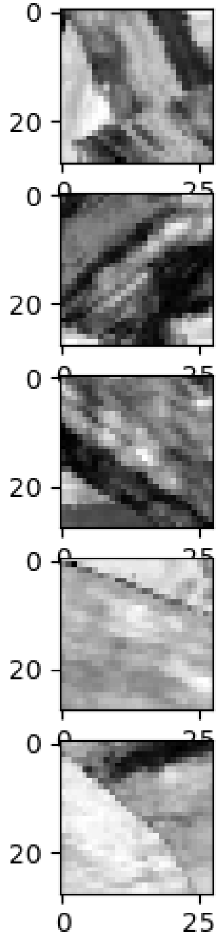
Texture Generation



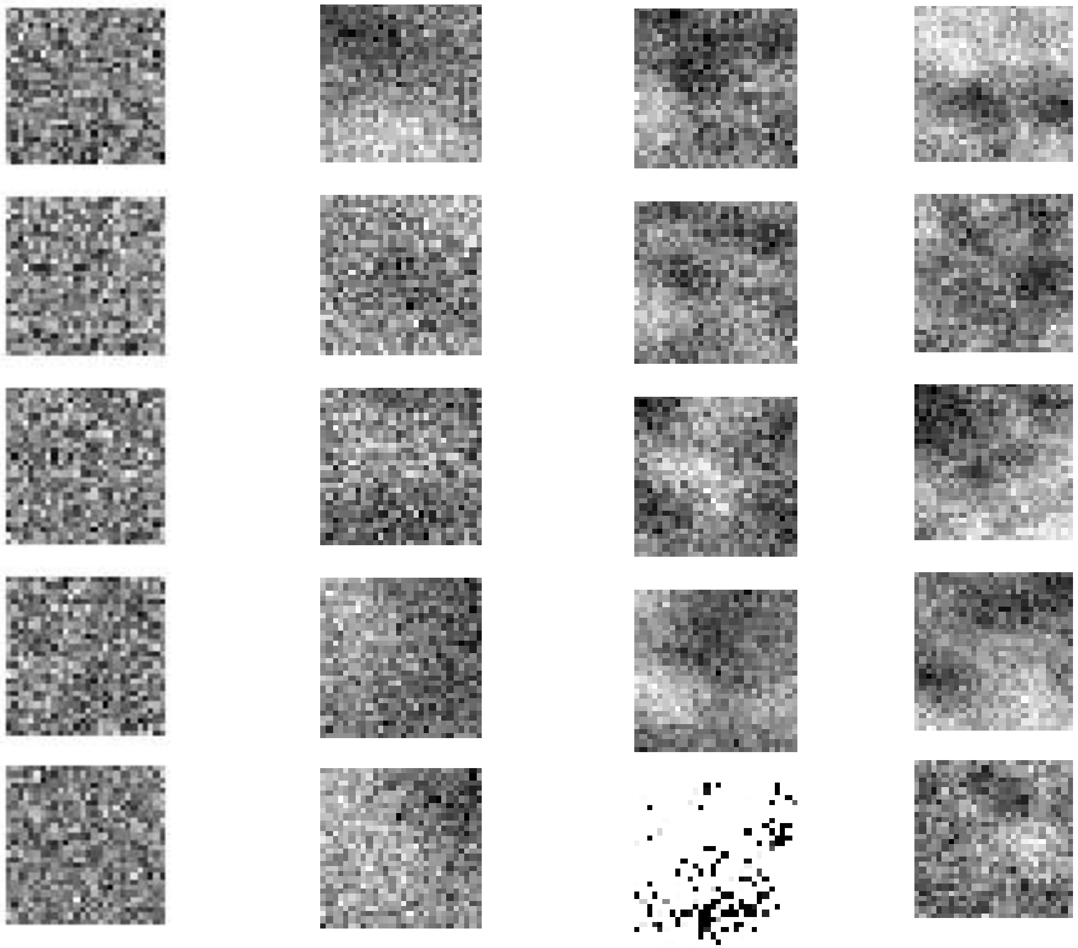
GAN Diagram from
https://github.com/hwalsuklee/tensorflow-generative-model-collections

Vanilla GAN for texture generation

wood



Reference



Iter.: 0

Iter.: 4,000

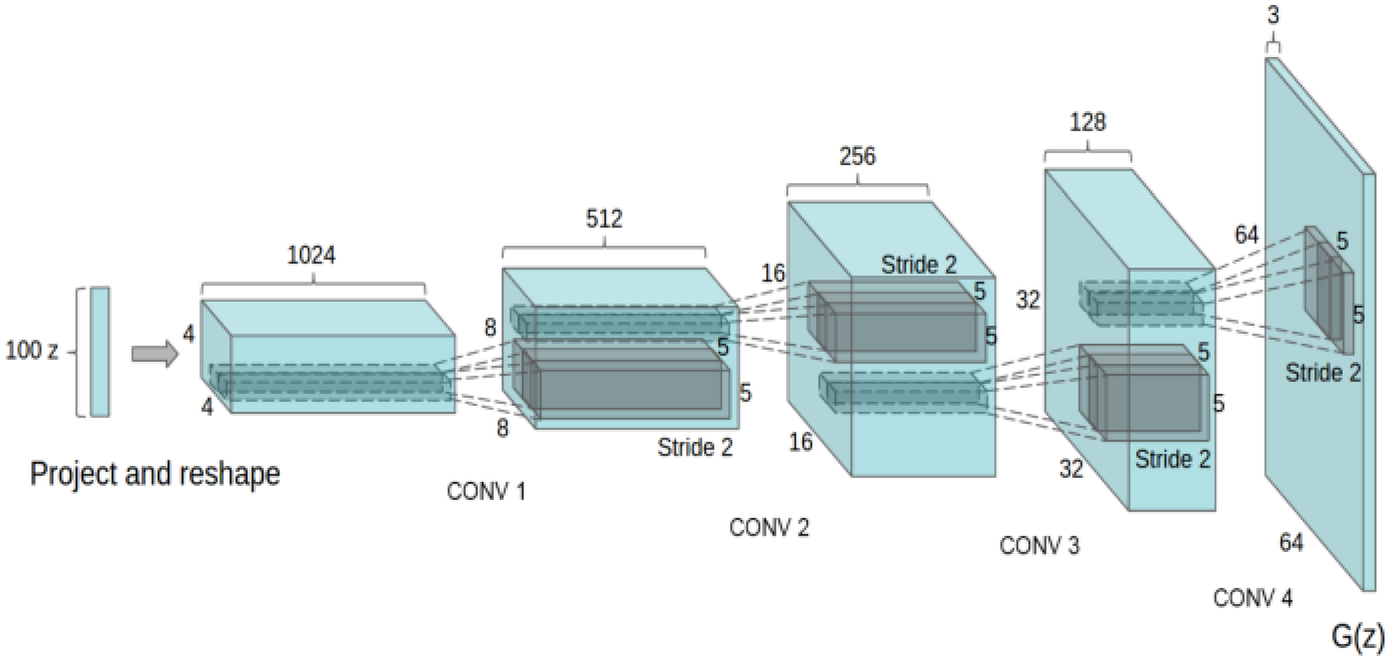
Iter.: 8,000

Iter.: 12,000

Training

Deep Convolutional GANs (DCGANs)

Generator Architecture



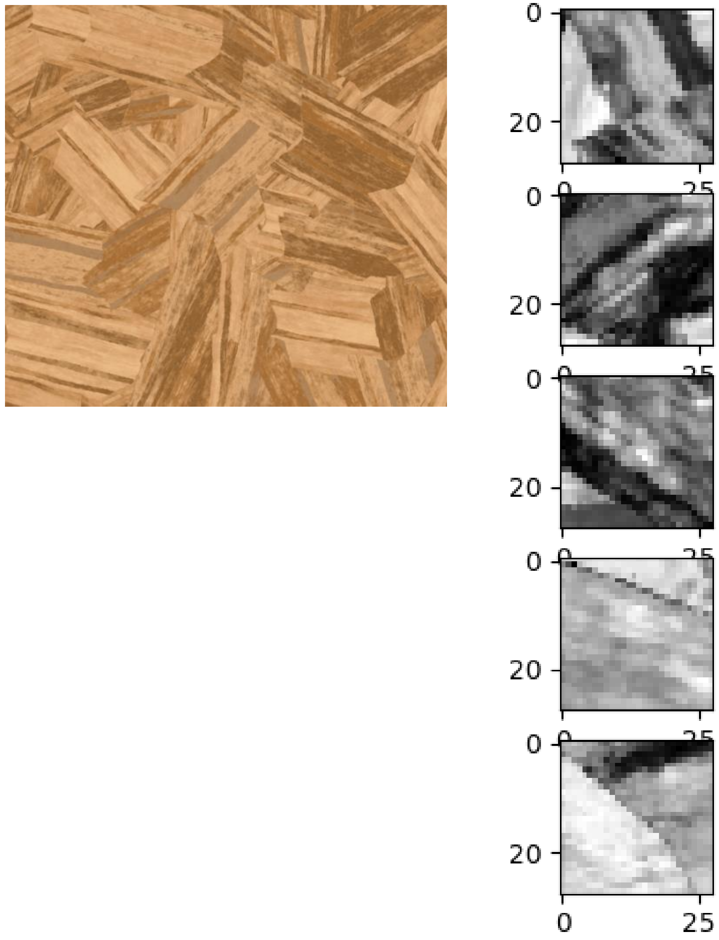
Key ideas:

- Replace FC hidden layers with Convolutions
  - **Generator:** Fractional-Strided convolutions
- Use Batch Normalization after each layer
- **Inside Generator**
  - Use ReLU for hidden layers
  - Use Tanh for the output layer

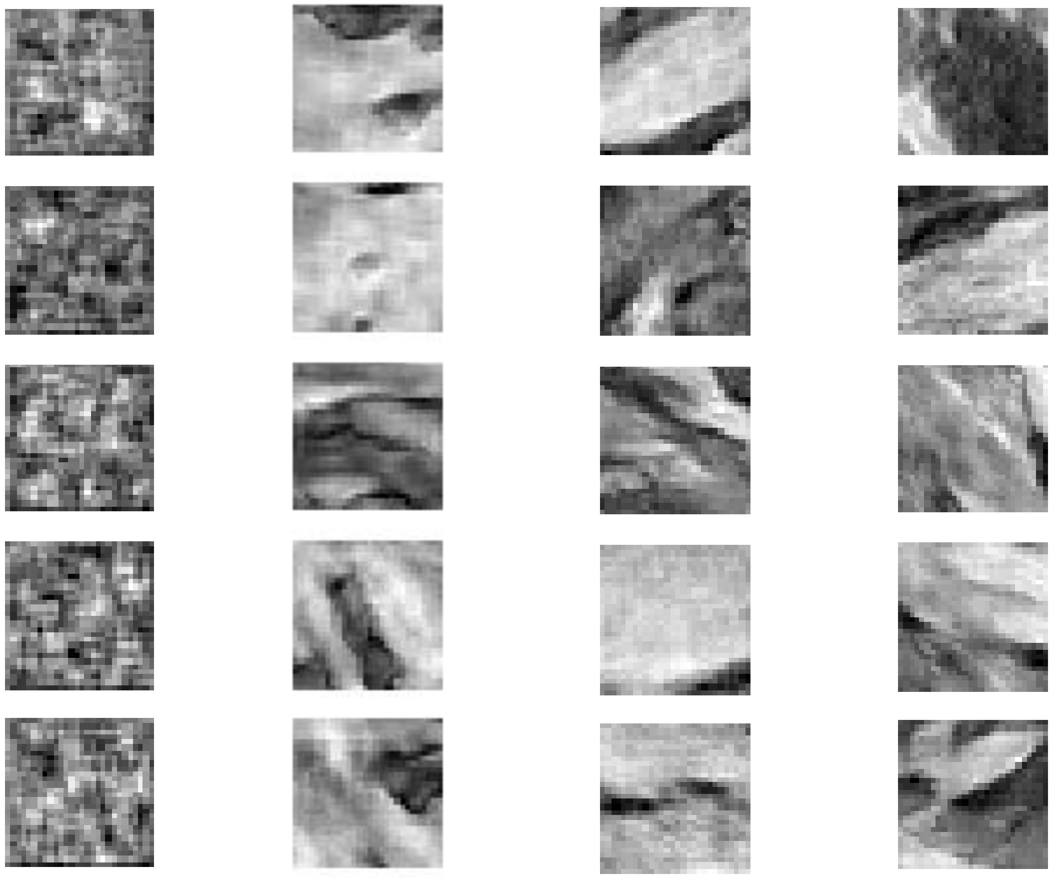
Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv:1511.06434 (2015).

http://slazebni.cs.illinois.edu/spring17/lec11_gan.pdf

Deep Convolutional GAN (DCGAN) for texture generation



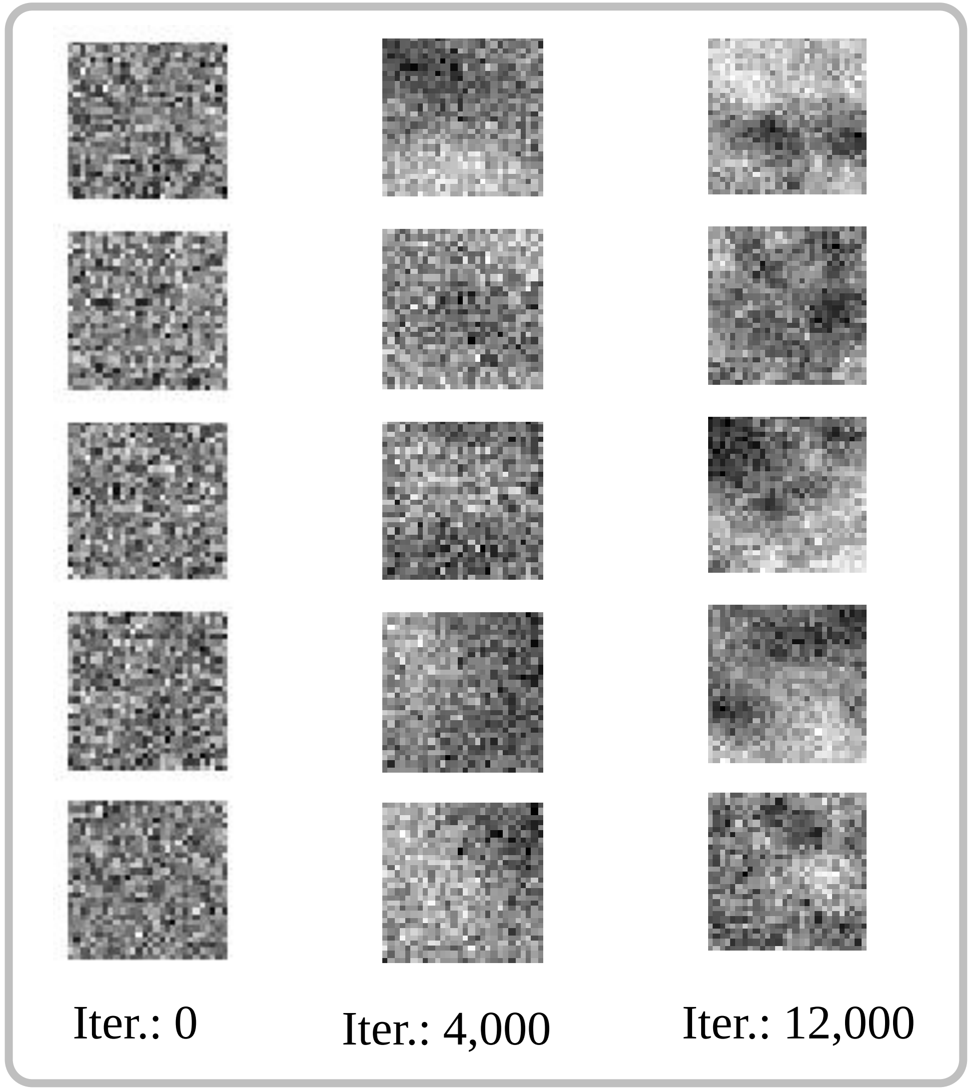
Reference

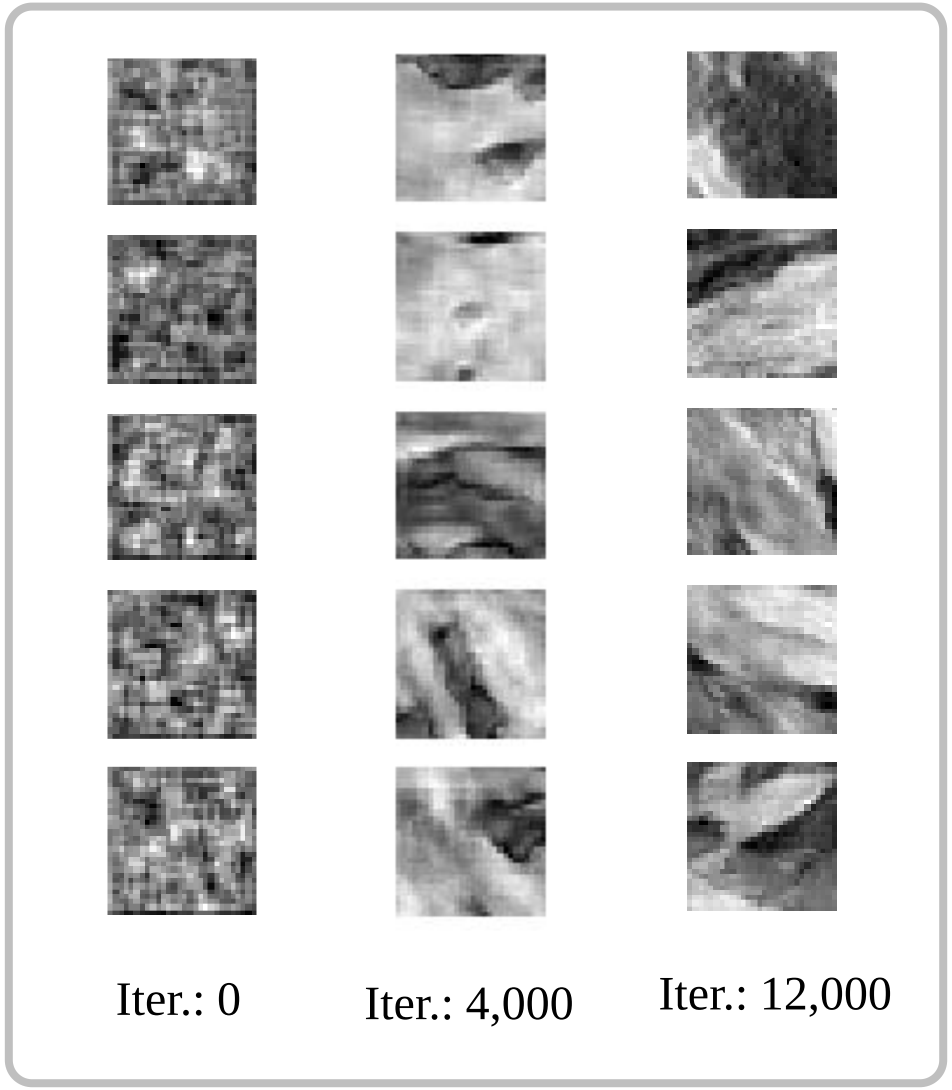


Iter.: 0 | Iter.: 4,000 | Iter.: 8,000 | Iter.: 12,000

Training

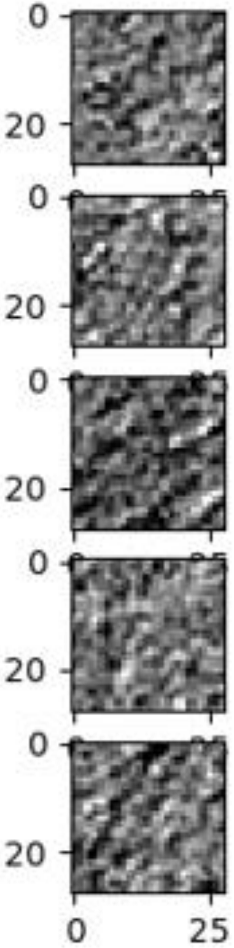
Vanilla GAN vs. DCGAN



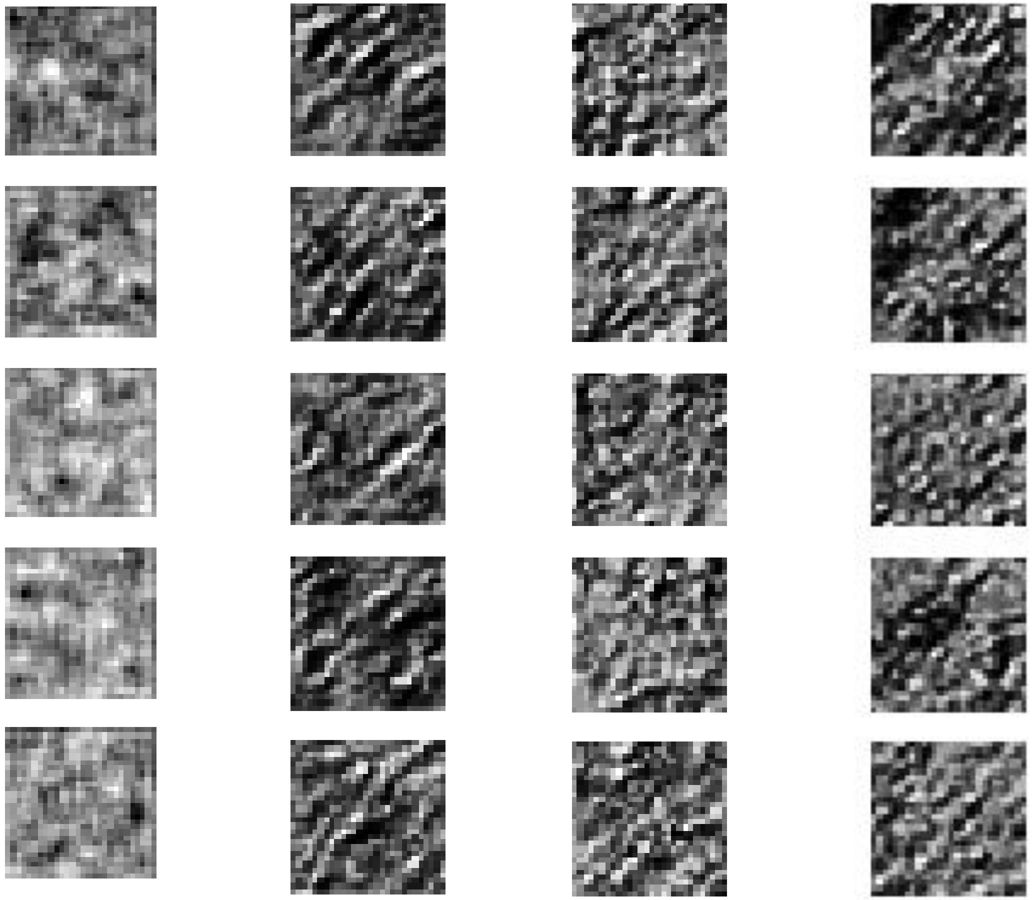


Deep Convolutional GAN for texture generation

ground_dirt



Reference



Iter.: 0

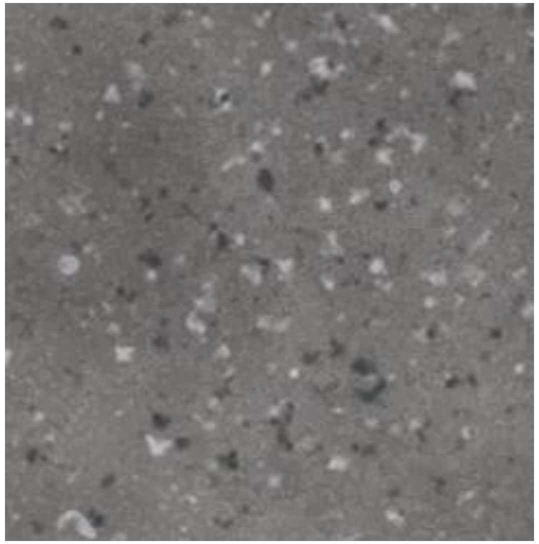
Iter.: 400

Iter.: 2,000

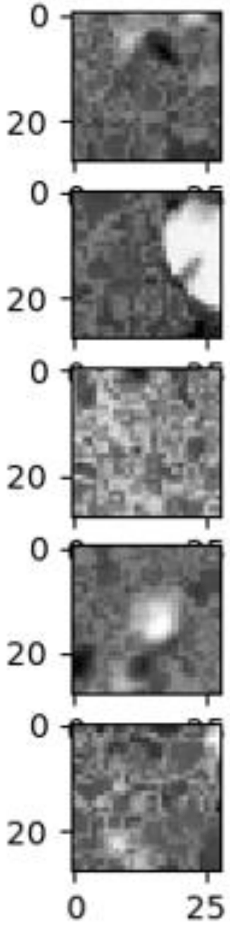
Iter.: 4,000

Training

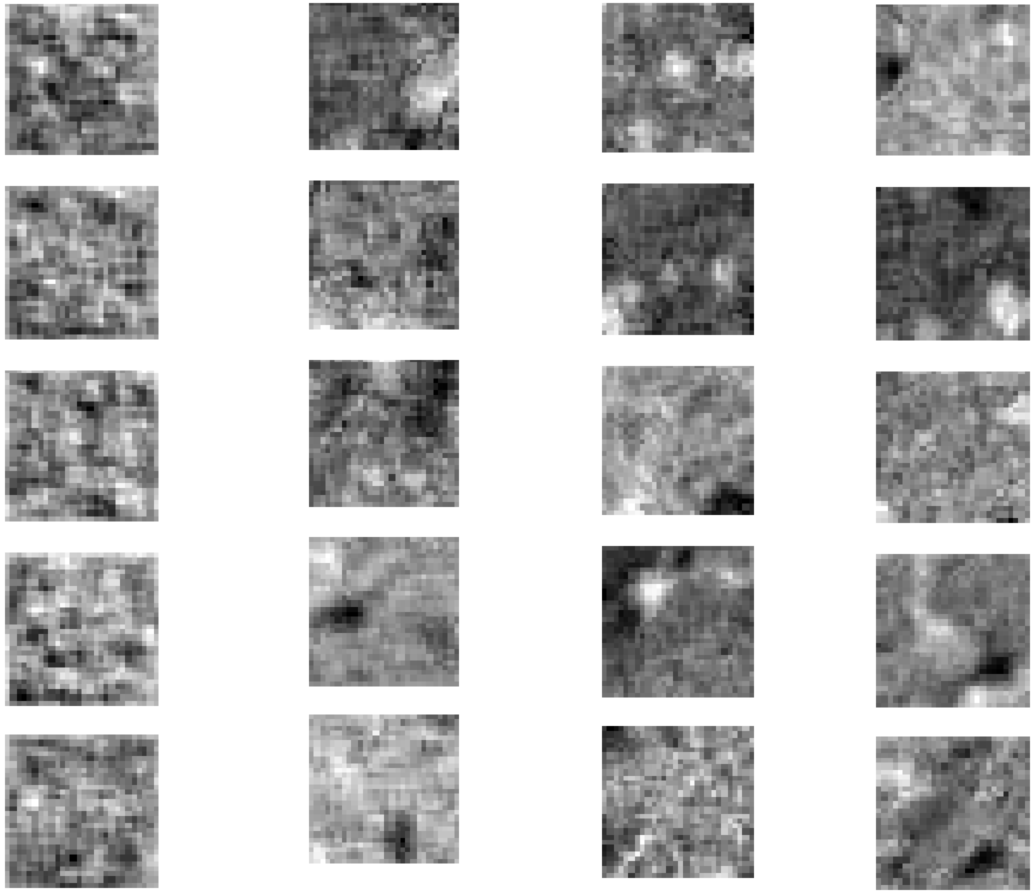
Deep Convolutional GAN for texture generation



ground_rock



Reference

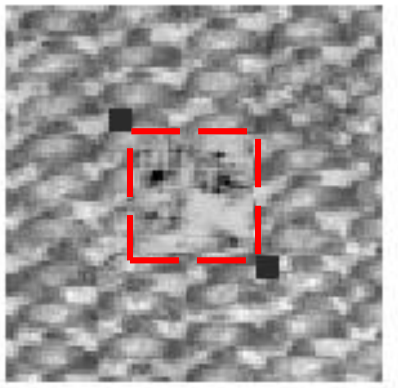


Iter.: 0 | Iter.: 2,000 | Iter.: 4,000 | Iter.: 8,000

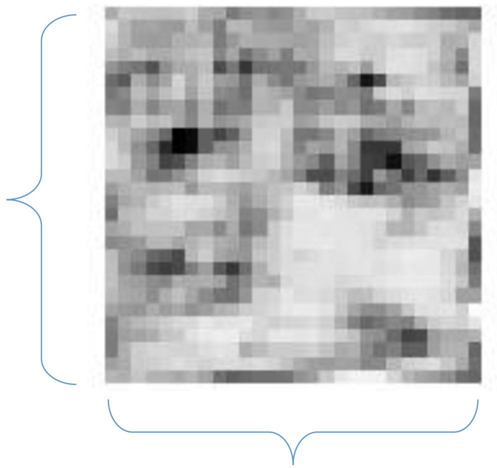
Training

Texture Generation with **Boundary Condition**

How to make a Boundary-Respecting Generator ?



Region of Interest



H:28

W:28

Boundary condition

Boundary condition vector:
2*w + 2*(h-2) pixels

How to make a **Boundary-Respecting** Generator ?

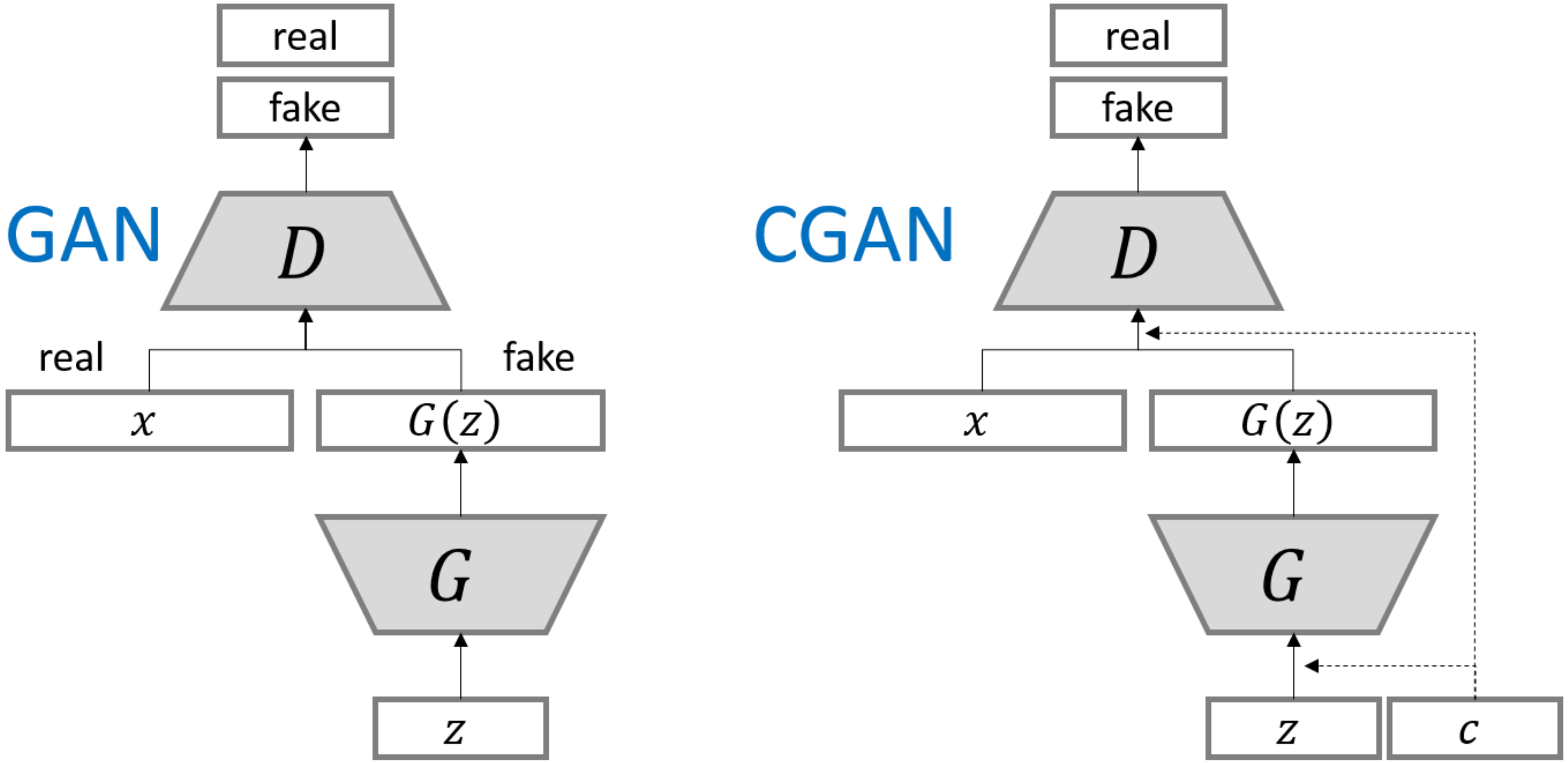
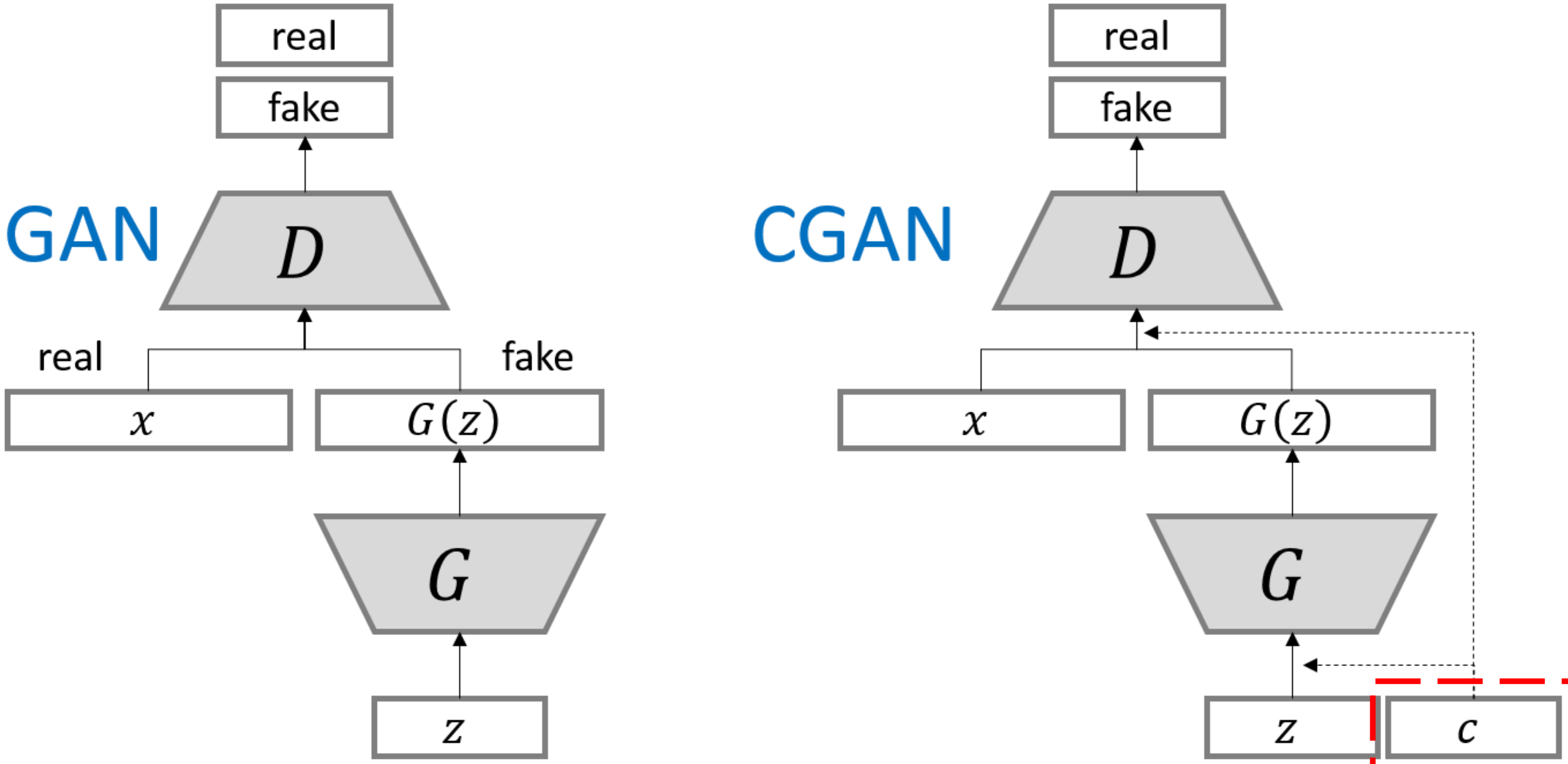


Image from
https://github.com/hwalsuklee/tensorflow-generative-model-collections

How to make a **Boundary-Respecting** Generator ?
: Utilize Conditional GAN (CGAN)

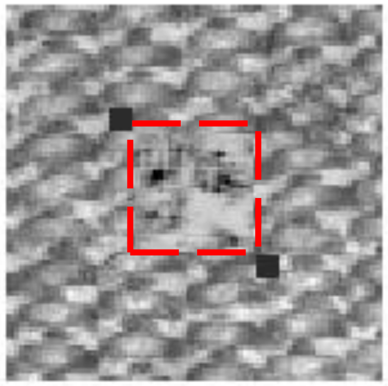


Additional Context

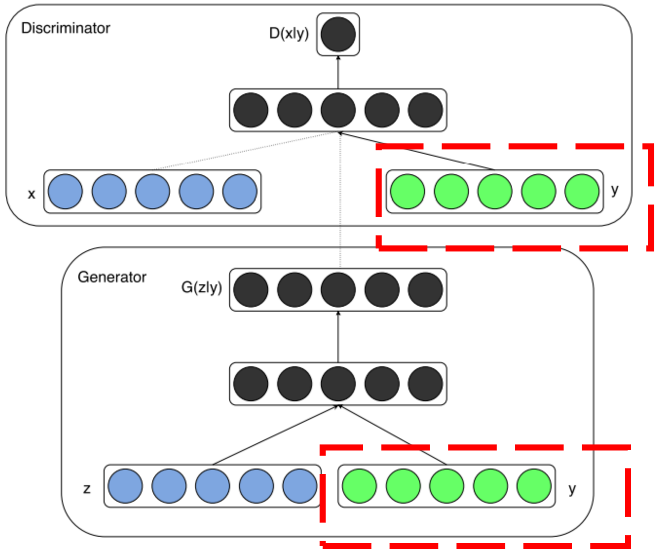
Boundary condition vector:
2*W + 2*(H-2) pixels

Image from
https://github.com/hwalsuklee/tensorflow-generative-model-collections

How to make a Boundary-Respecting Generator ?
: Utilize Conditional GAN (CGAN)

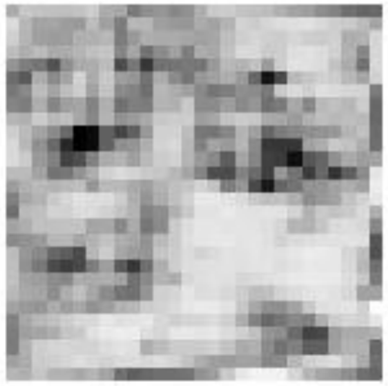


Region of
Interest



CGAN:
Mirza, Mehdi, and Simon Osindero. "Conditional generative adversarial nets." *arXiv preprint arXiv:1411.1784* (2014).

H:28



W:28

Boundary condition

Boundary condition vector:
2*W + 2*(H-2) pixels

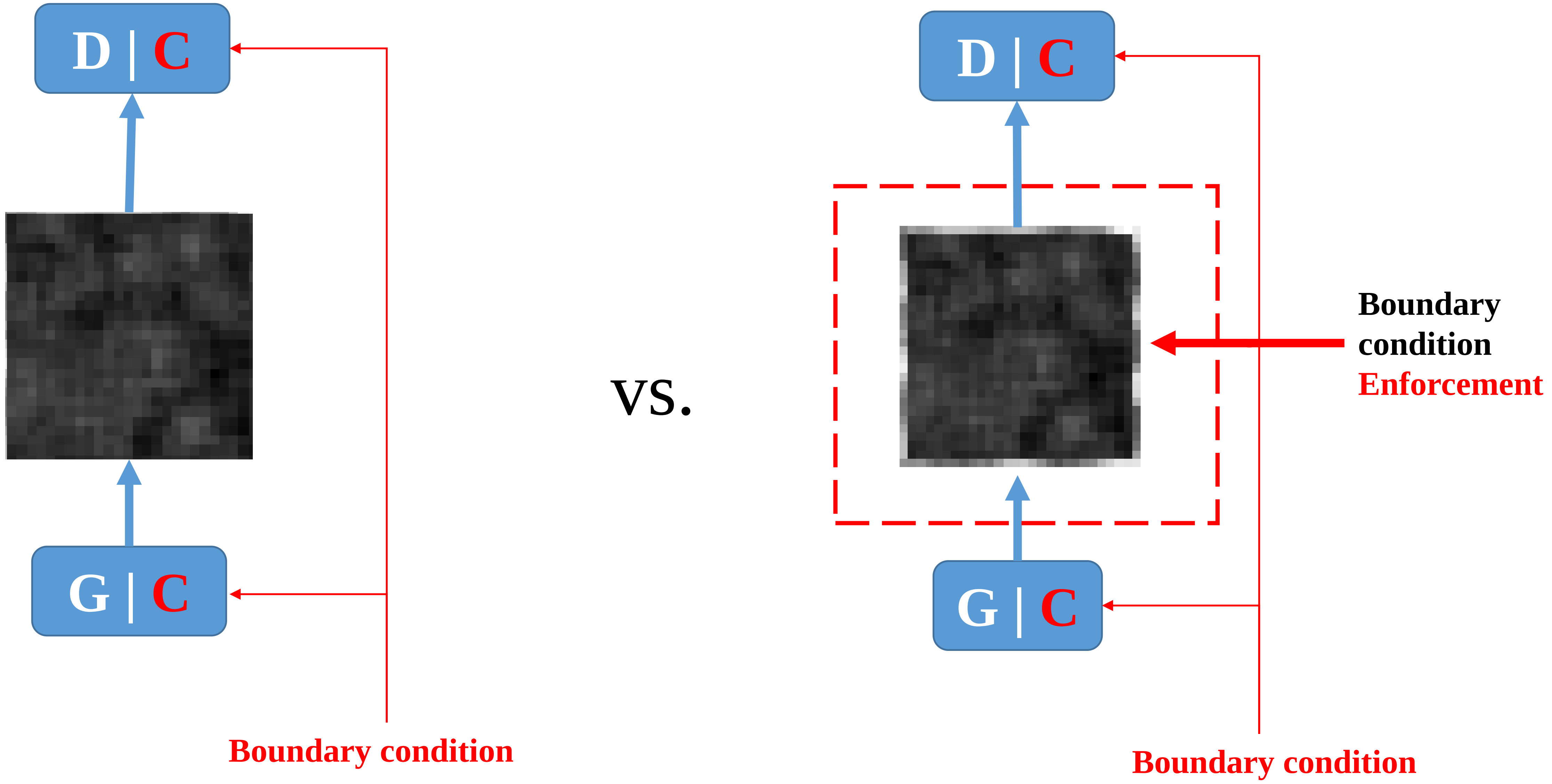
$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

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

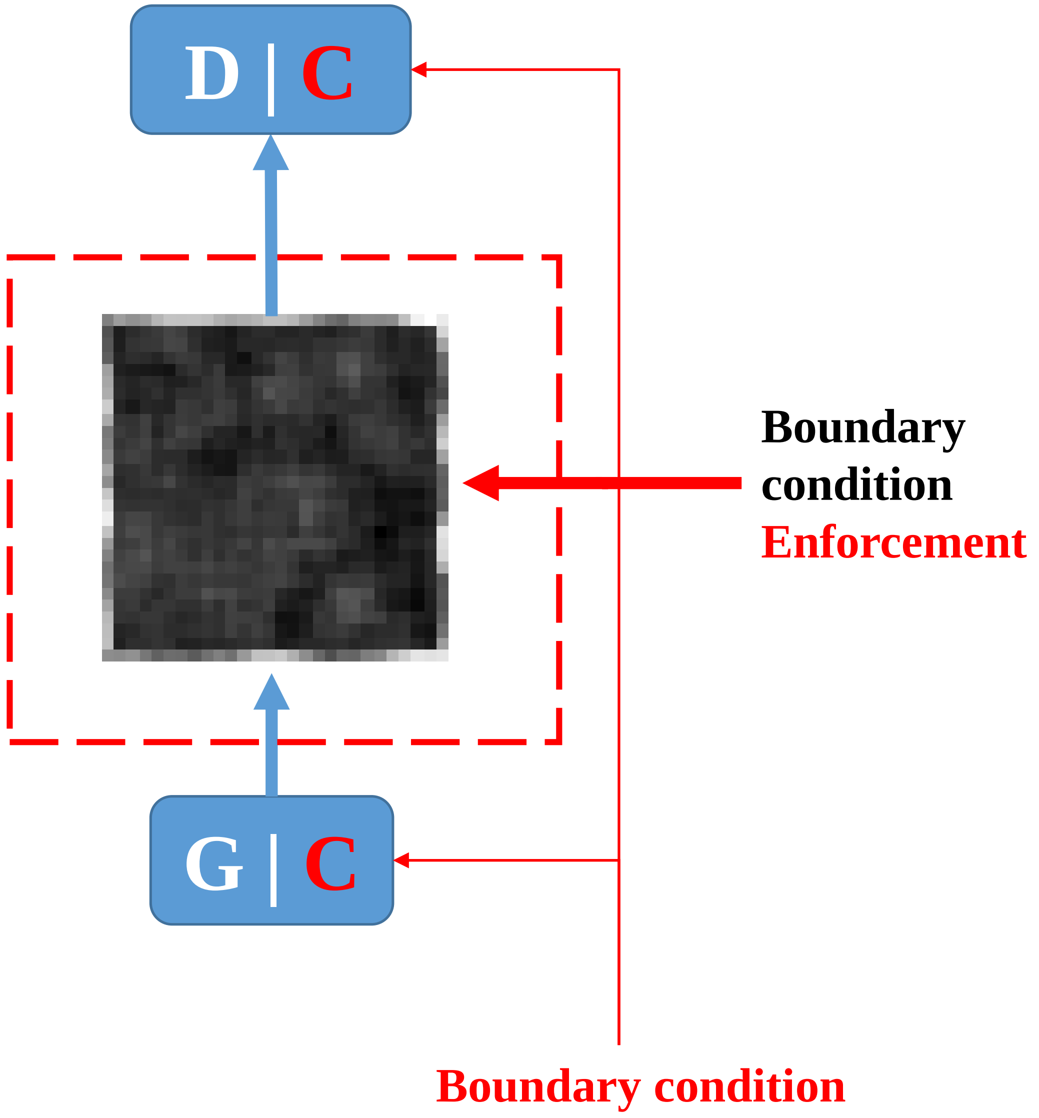
Boundary condition

Boundary condition

CGAN vs. Boundary Enforced CGAN (BECGAN, proposed)



Boundary Enforced CGAN (BECGAN, proposed)



A generated image with boundary pixels enforced is passed to the discriminator

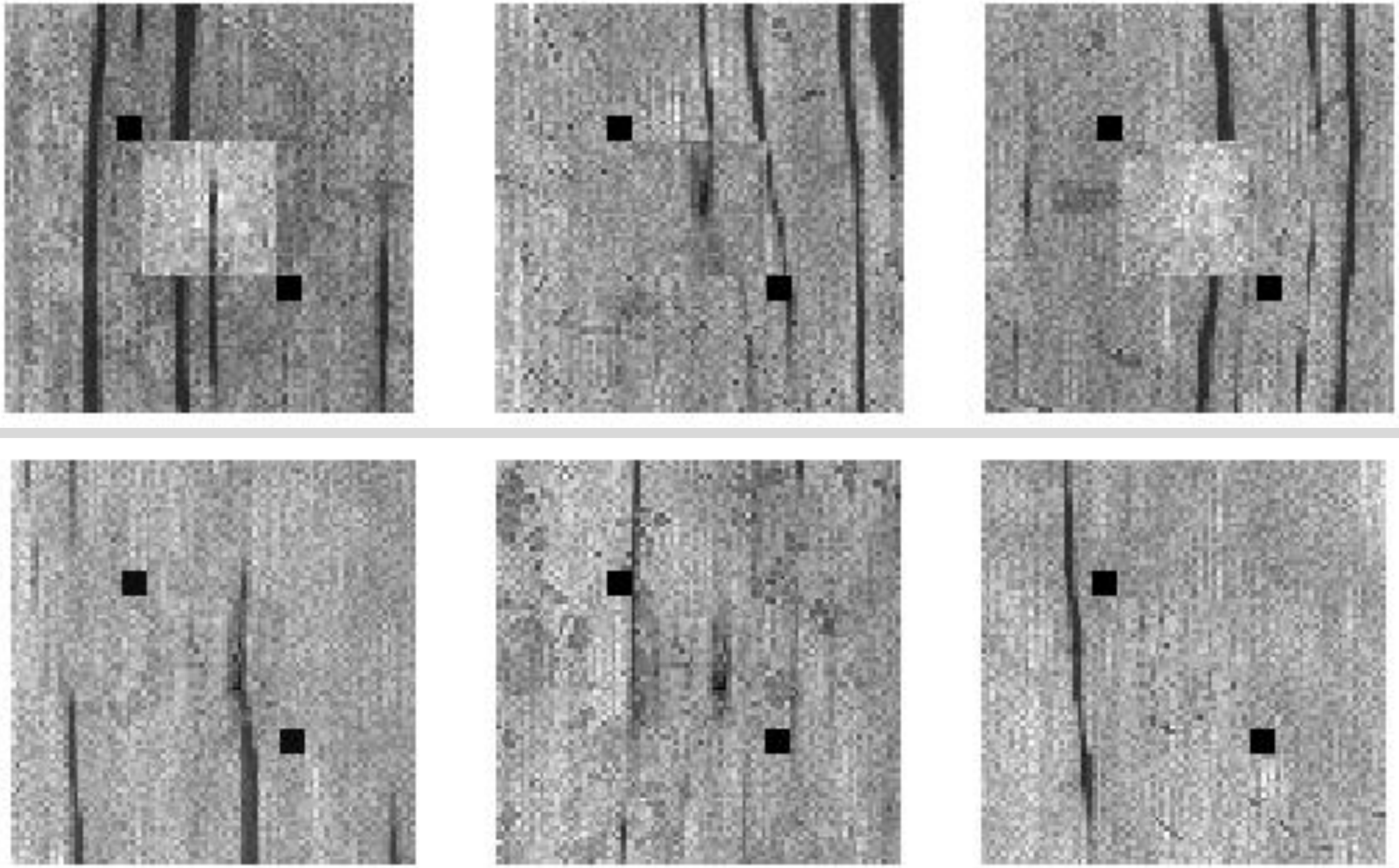
Discriminator would focus on the boundary region since it looks quite **artificial**

Generator would **also focus on the boundary region** to fool the discriminator.

CGAN vs. **Boundary Enforced** CGAN

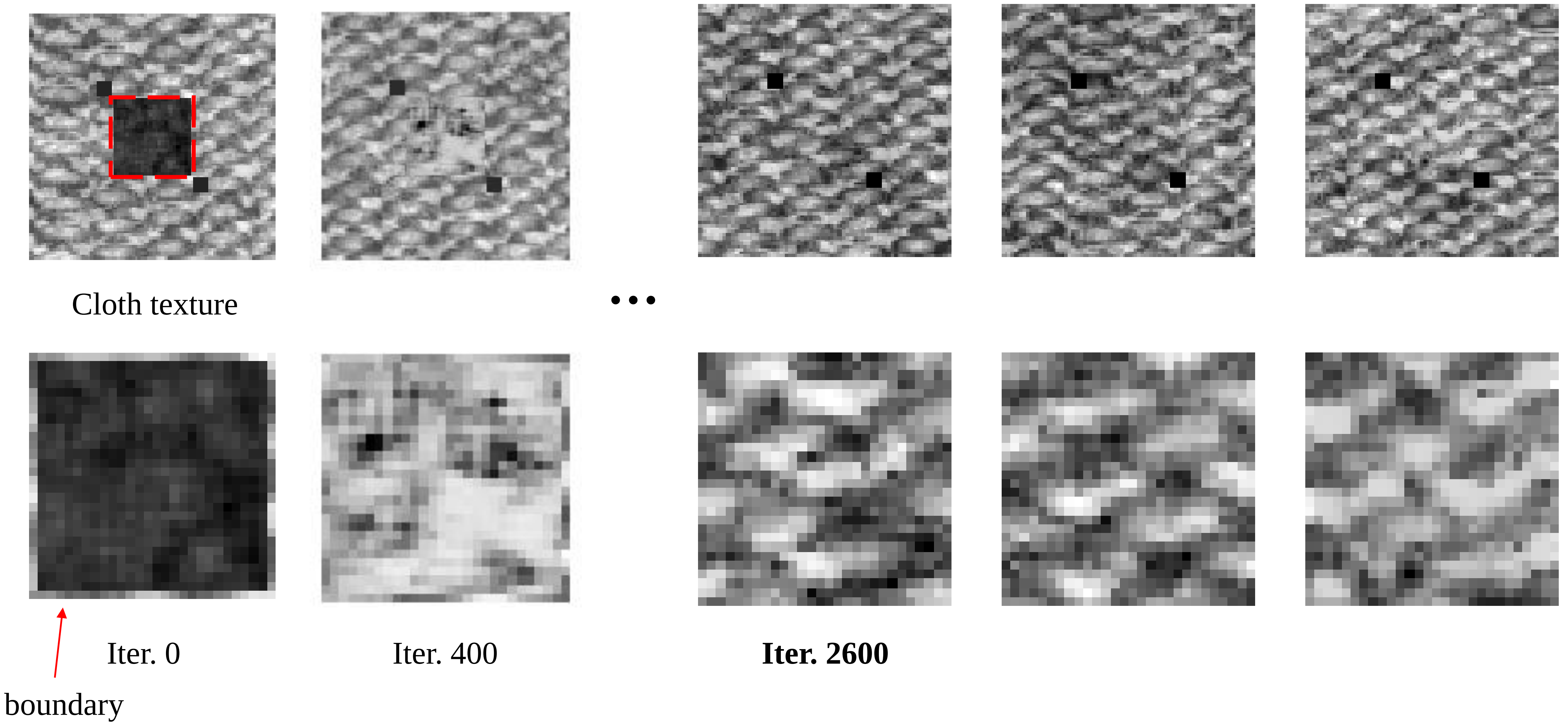
CGAN

Boundary Enforced CGAN **(proposed)**

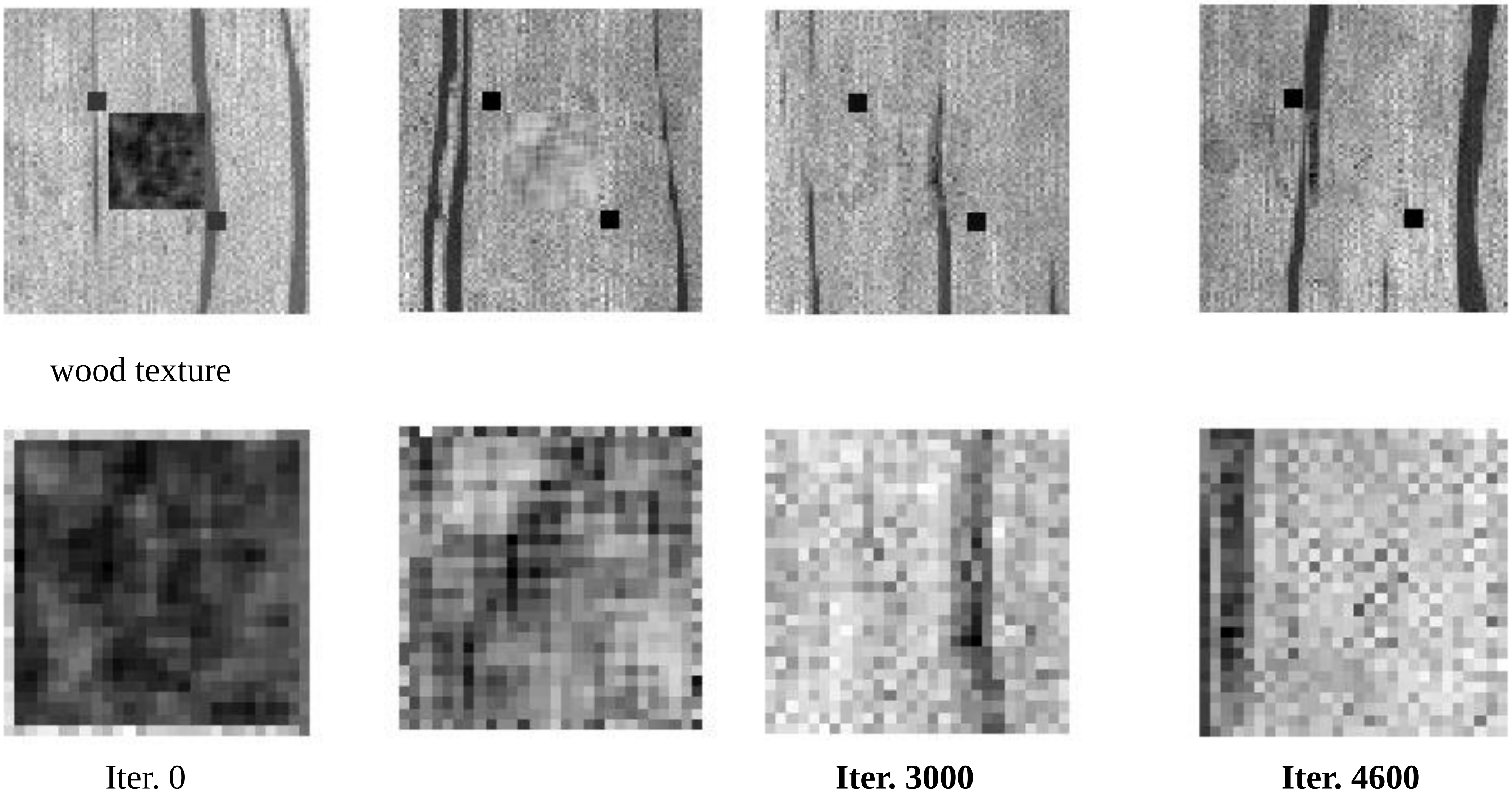


Iter. 3,000

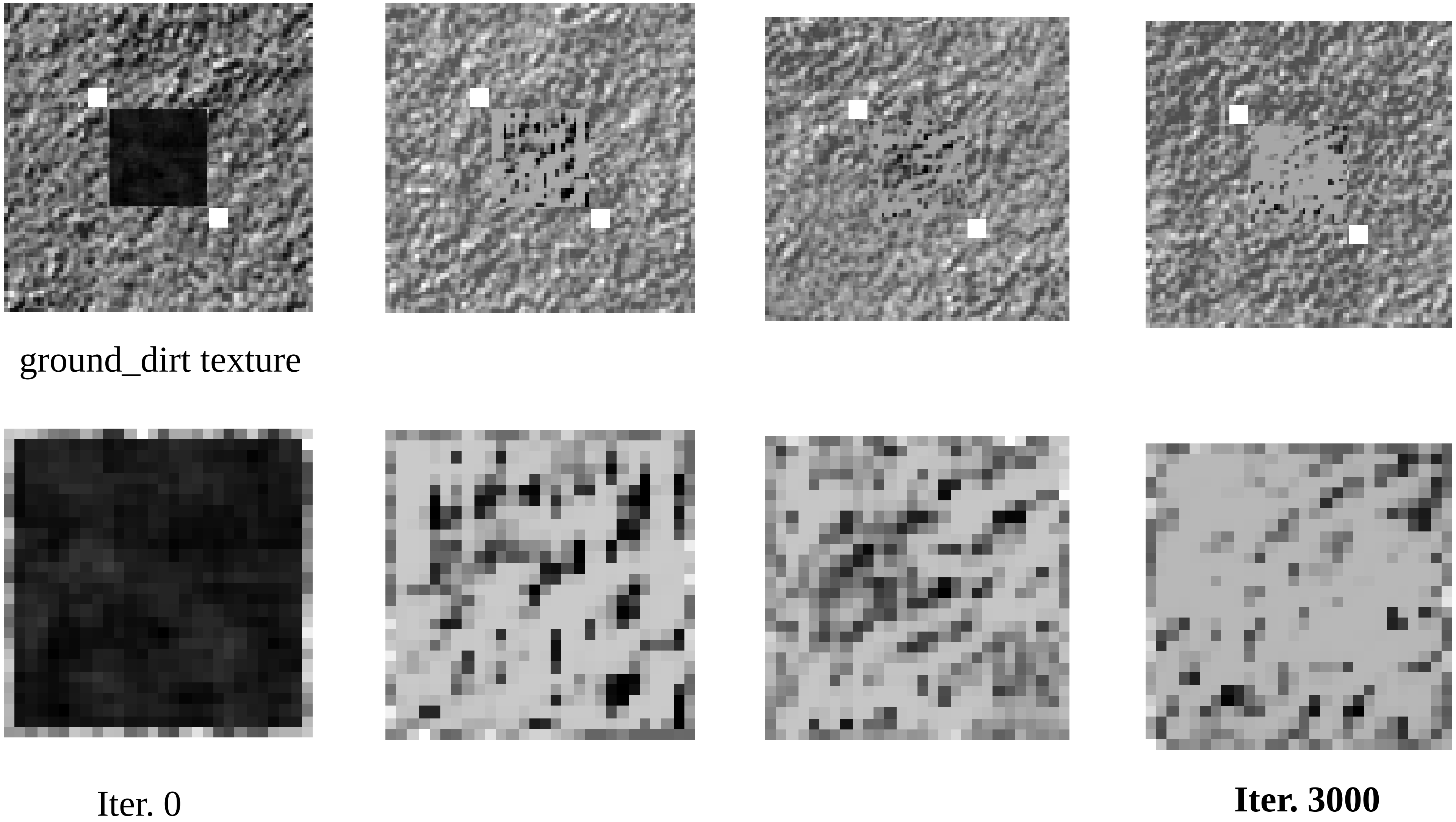
BECGAN improves over learning iterations



BECGAN improves over learning iterations



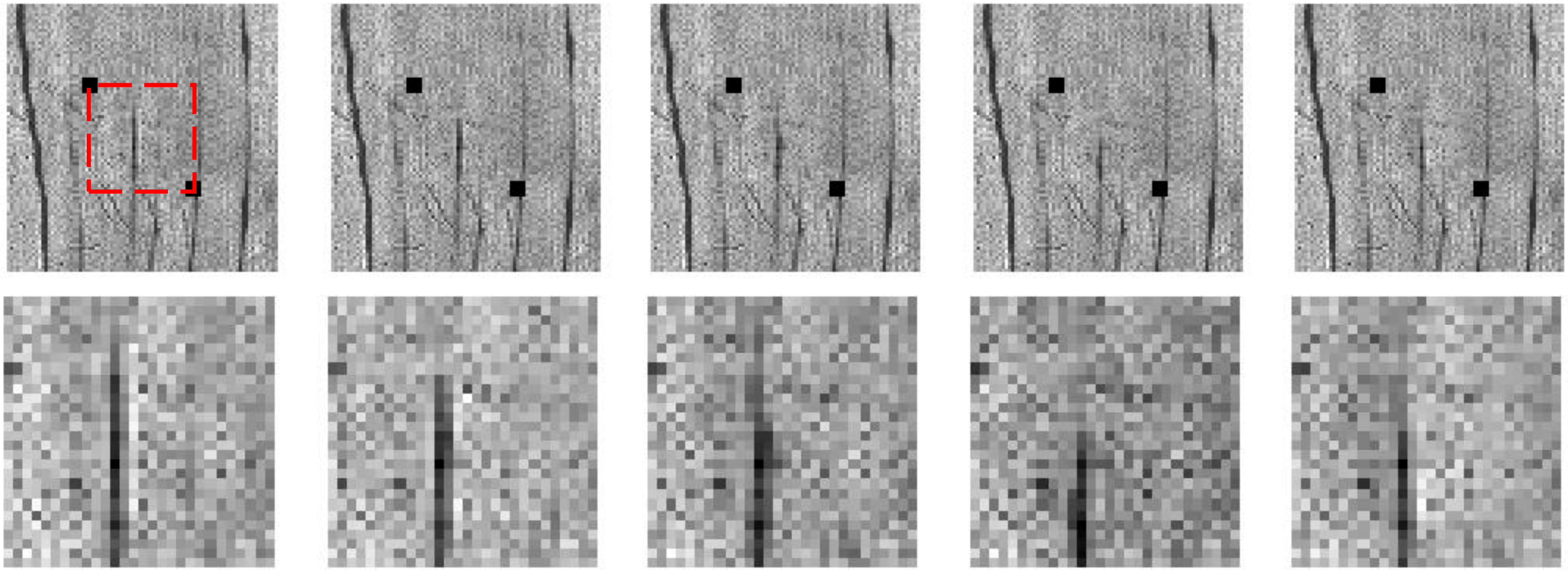
Some **failure** cases … (possibly due to **GAN** **instability** in learning)

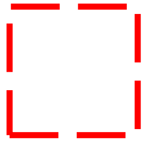


Diversity
given a same
Boundary Condition

: Does the generator memorize the data or not ?

Diversity given a same boundary condition

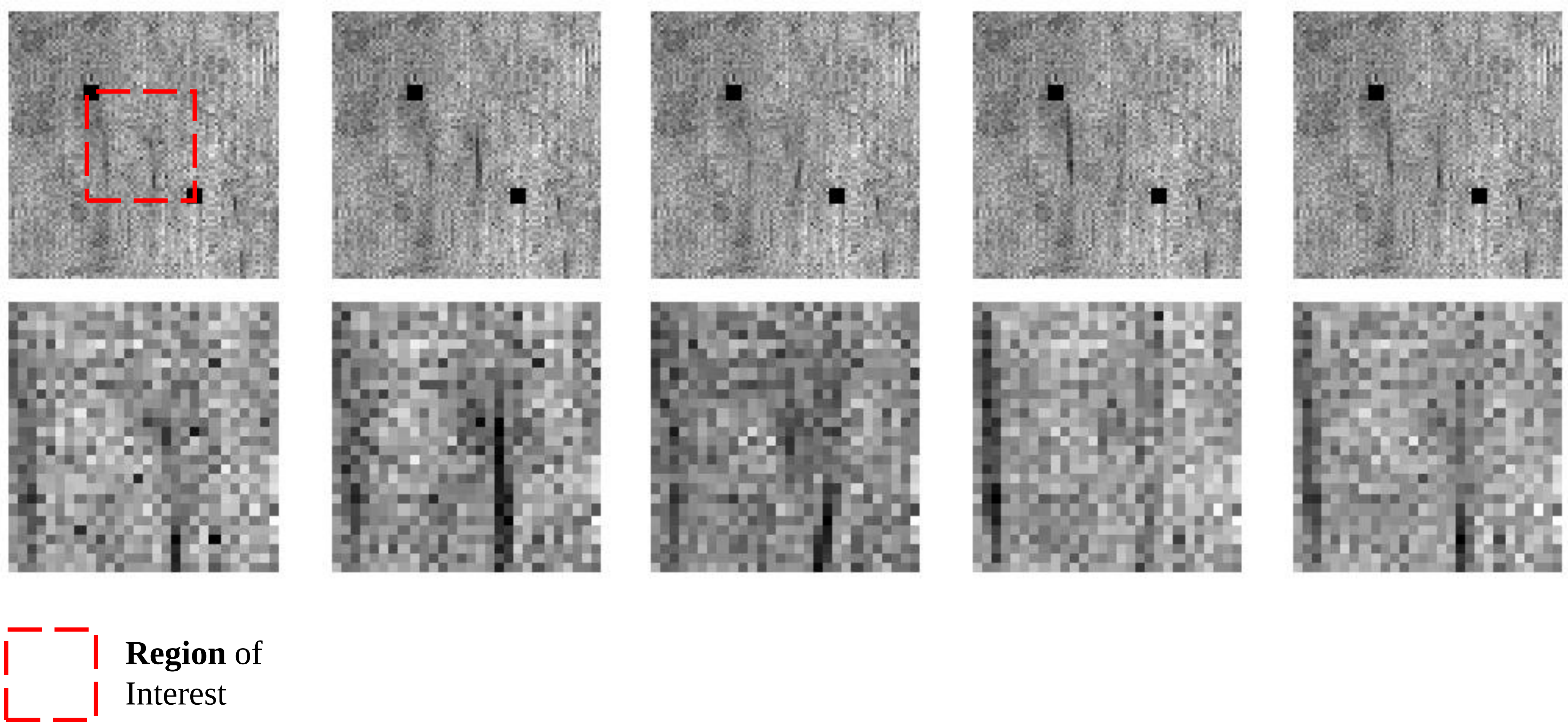




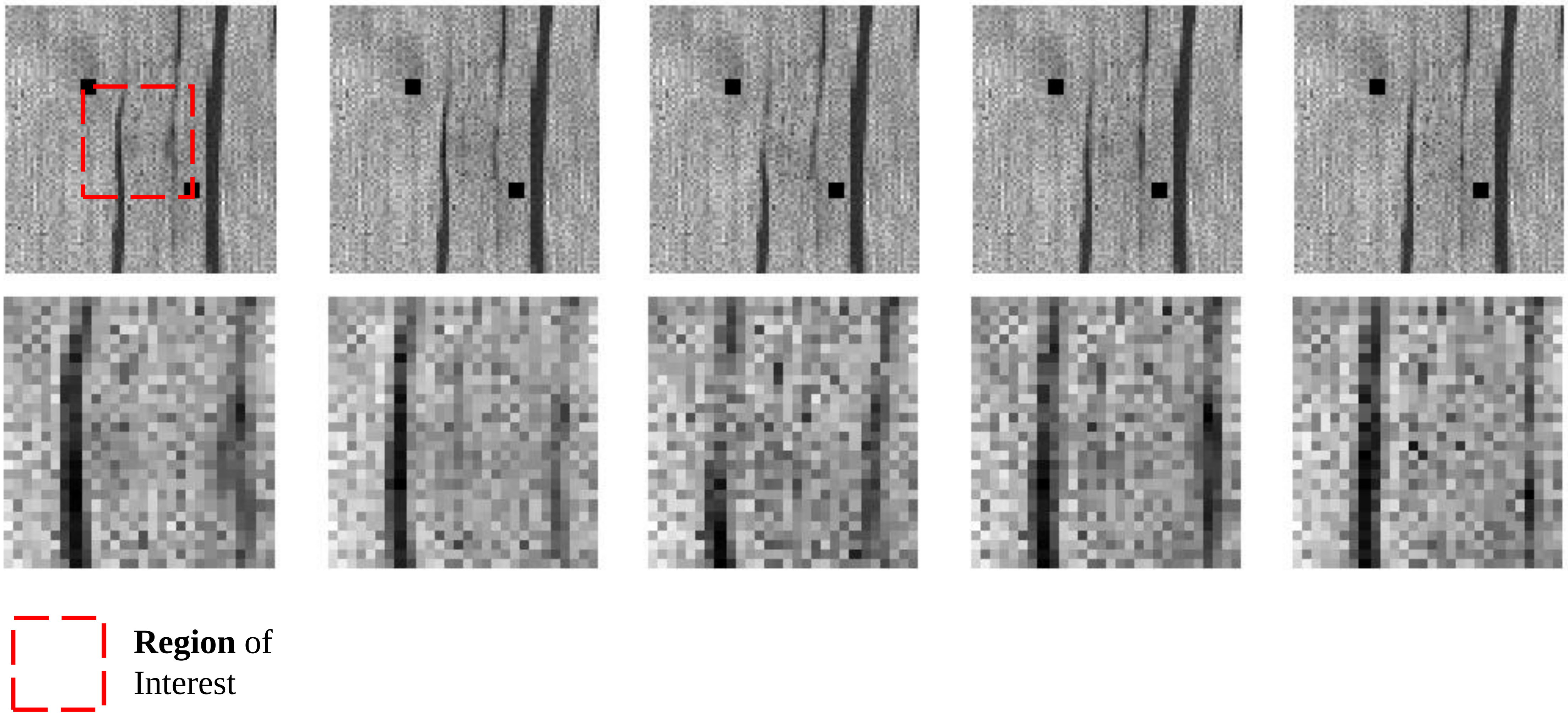
Region of Interest

- generated **highlight** effect
- generated different **crack** patterns

Diversity given a same boundary condition

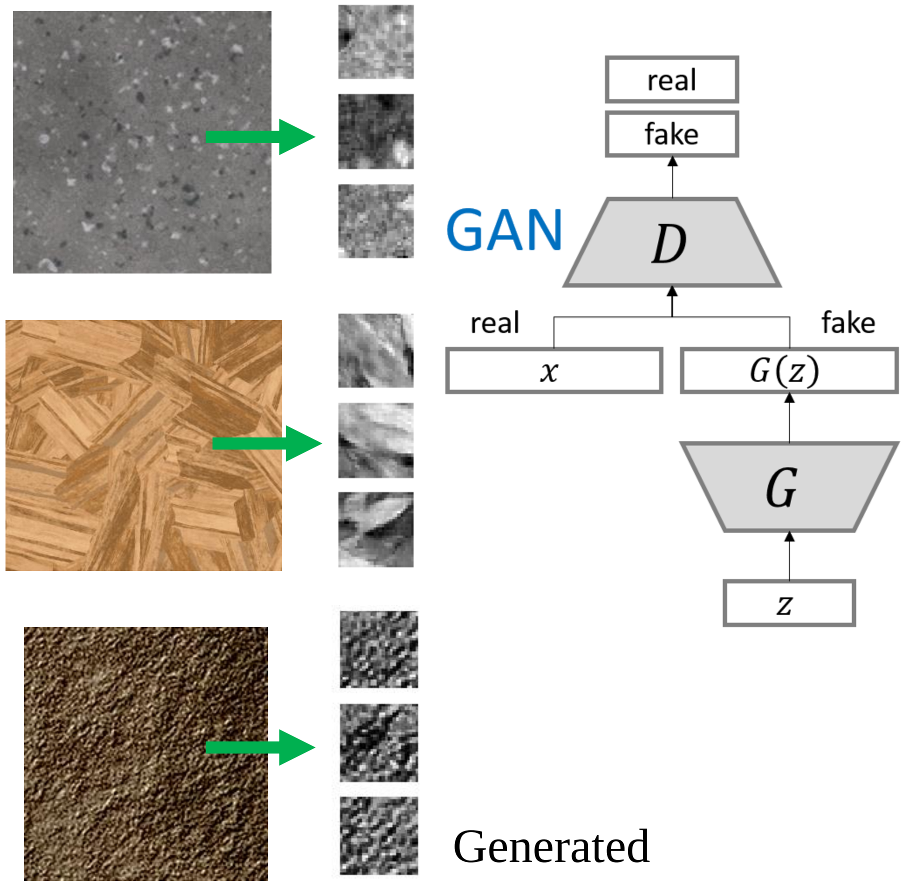


Diversity given a same boundary condition

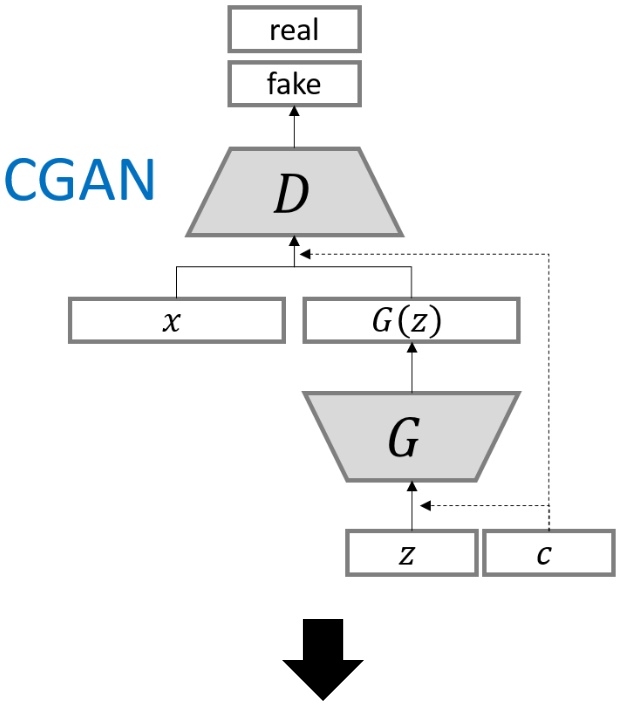


Summary

Deep Convolutional GAN
for **Unconstrained** Texture Generation



Conditional GAN
for **Boundary Conditioned** Texture Generation



Boundary Enforced CGAN
(BECGAN, **proposed**)