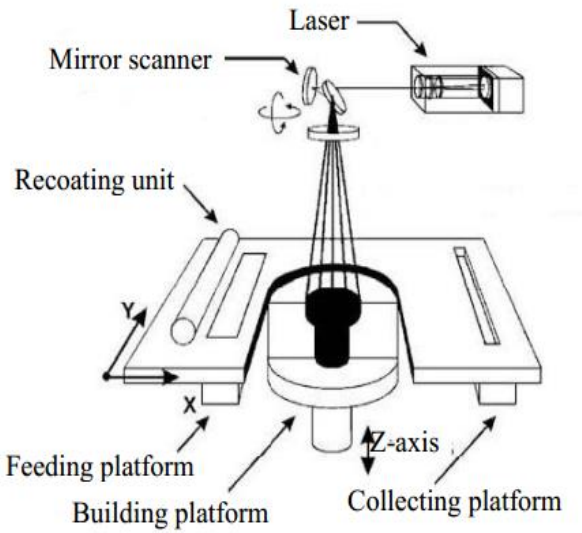
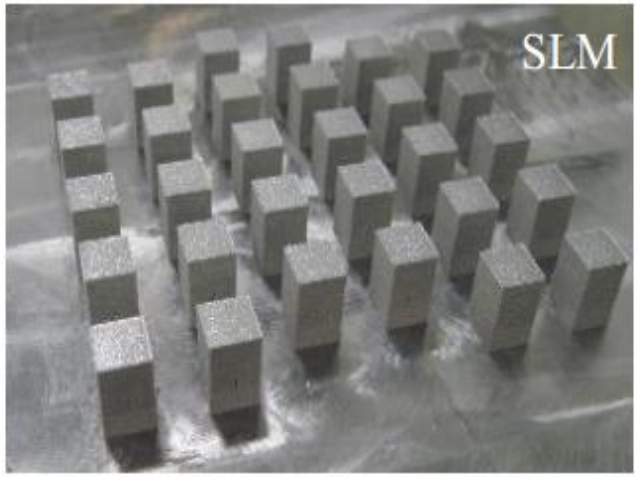
Application CGAN on Addictive Manufacturing

高园园 工学院 1801111733

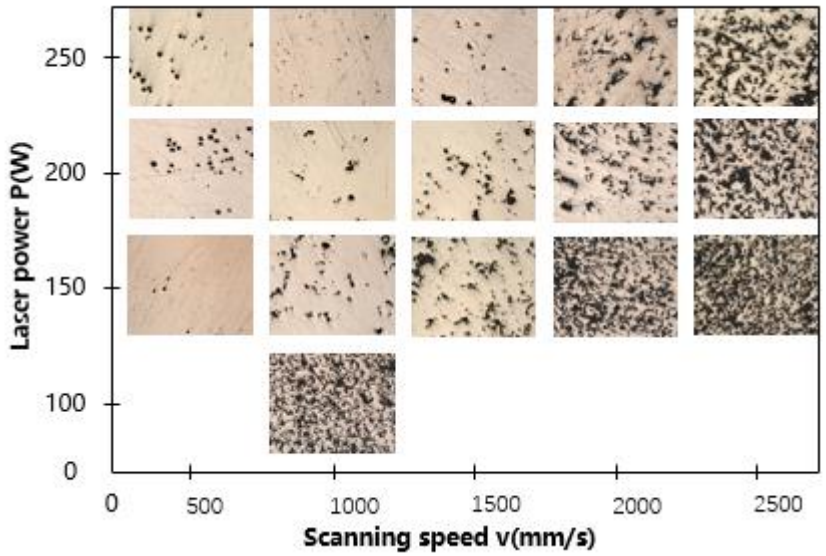
Introduction



SLM



Samples

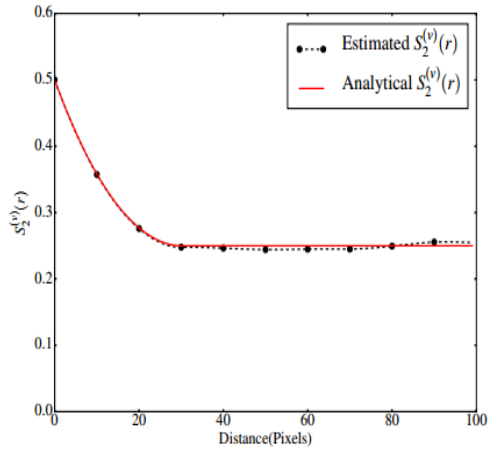


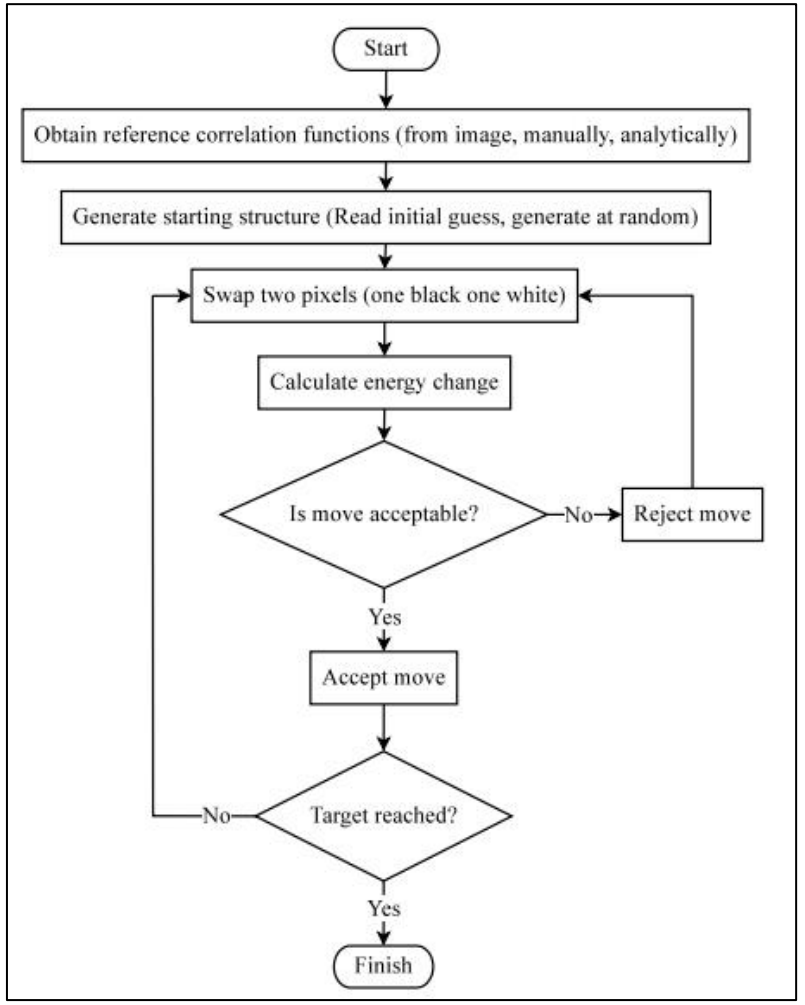
Microstructure

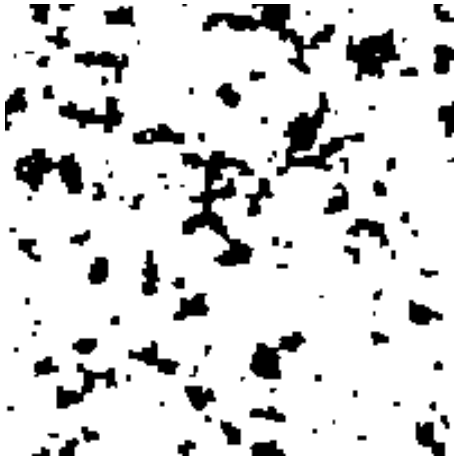
Related Work

$$E_t = \sum_k [f_t - f_0]^2$$

$$p(\Delta E_t) = \begin{cases} 1, \Delta E_t \le 0 \\ \exp\left(-\frac{\Delta E_t}{T_t}\right), \Delta E_t > 0 \end{cases}$$







Size:200*200

Reconstruction time： 3h

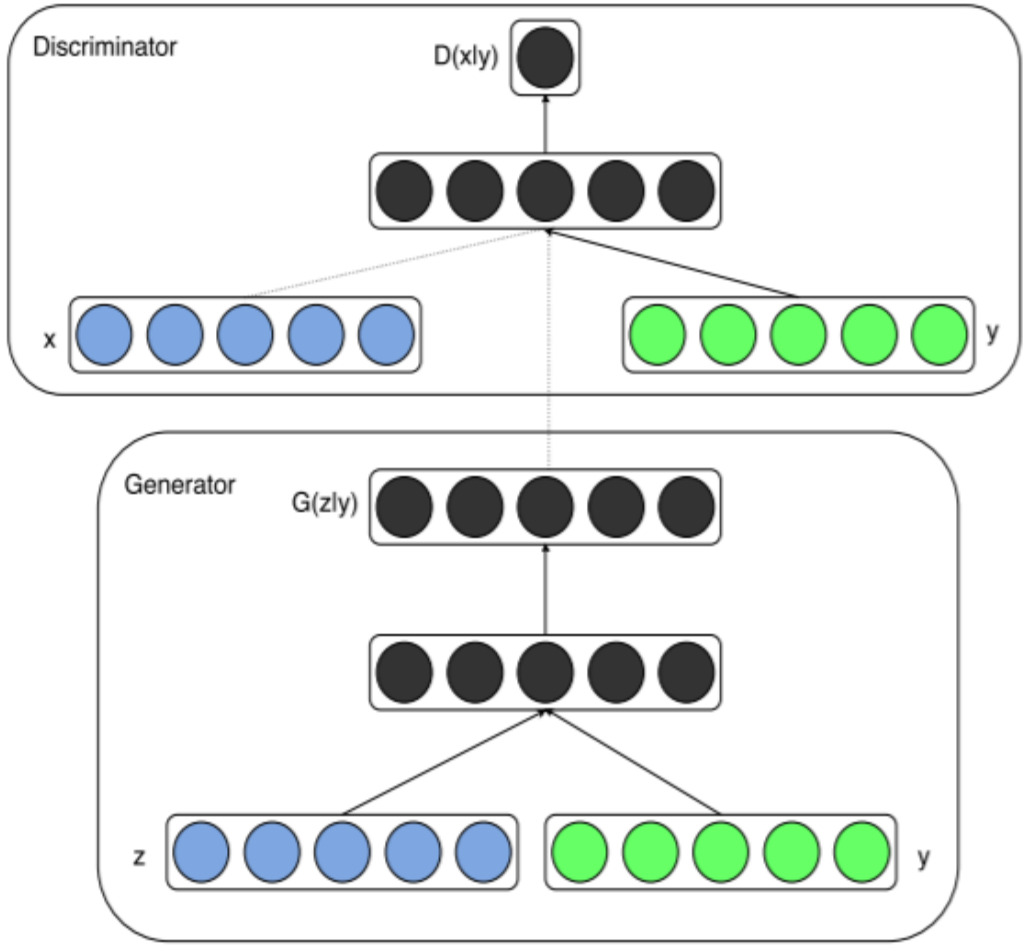
Method

x: microstructure
y: processing parameters
z: a noise prior

Processing parameters:

Power(W)	Speed(mm/s)		
100	800	300	1300
200	300	300	2000
200	800	300	3000
200	1300	400	300
200	2000	400	800
200	3000	400	1300
300	300	400	2000
300	800	400	3000

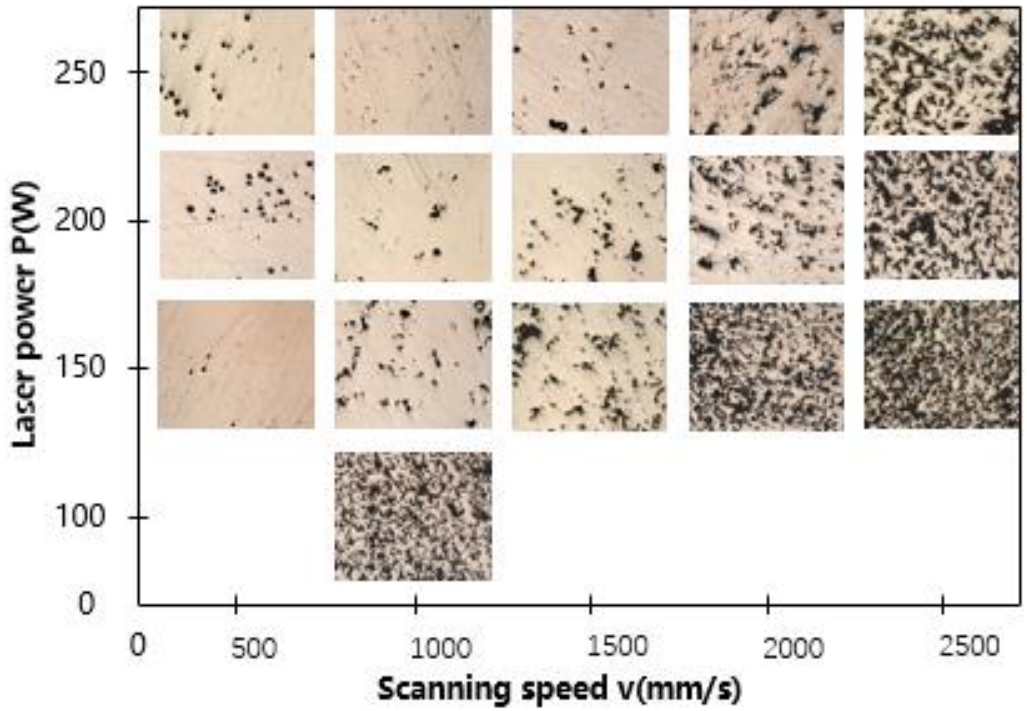
Our goal: G(X|Z,Y,θ)

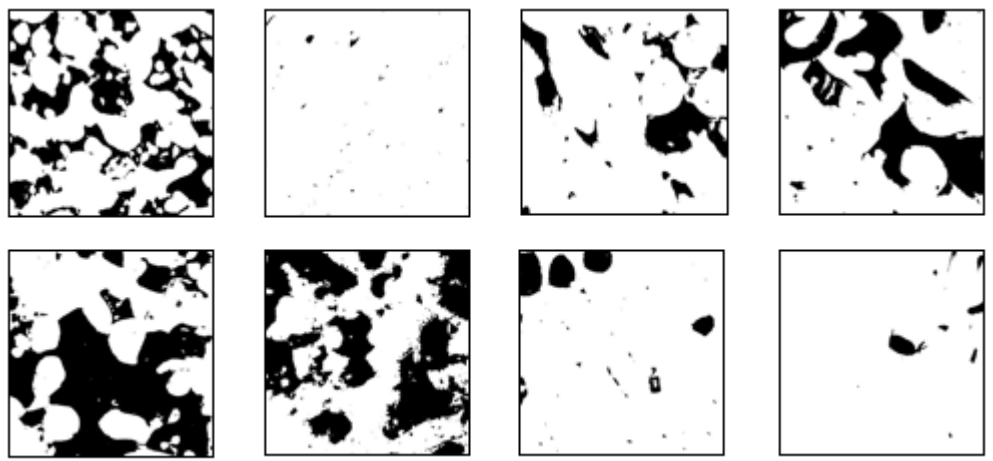


Method

Original: 16*16
Image size: 2752* 2208

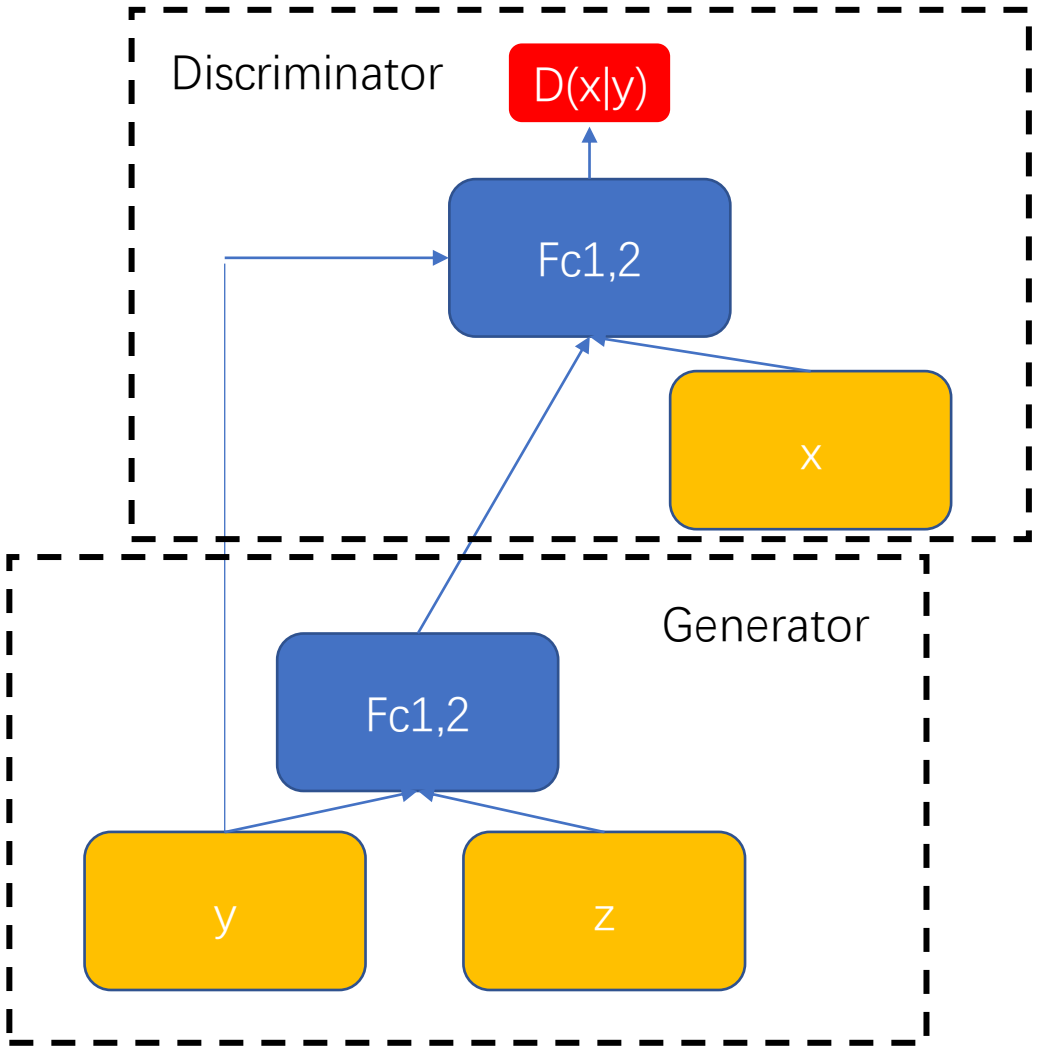
Preprocessing:
16*16
Image size: 20*550*550





Method

x: microstructure
y: processing parameters
z: a noise prior



Method

Model structure:

```
generator(
  (fc1_1): Linear(in_features=100, out_features=256, bias=True)
  (fc1_1_bn): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (fc1_2): Linear(in_features=2, out_features=256, bias=True)
  (fc1_2_bn): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (fc2): Linear(in_features=512, out_features=512, bias=True)
  (fc2_bn): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (fc3): Linear(in_features=512, out_features=2048, bias=True)
  (fc3_bn): BatchNorm1d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (fc4): Linear(in_features=2048, out_features=40000, bias=True)
)
```

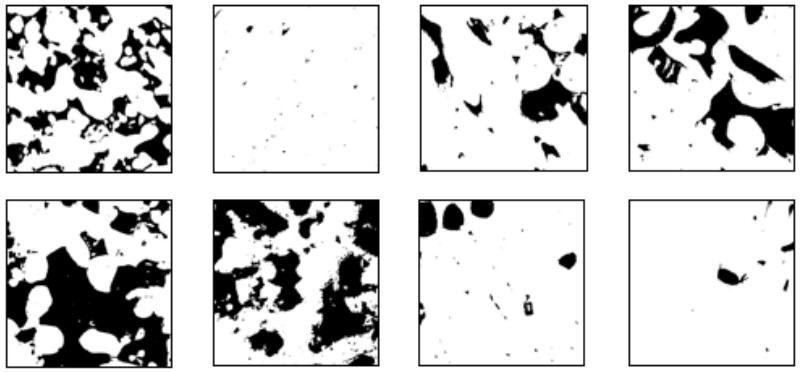
```
In [459]: D
Out[459]:
discriminator(
  (fc1_1): Linear(in_features=40000, out_features=1024, bias=True)
  (fc1_2): Linear(in_features=2, out_features=1024, bias=True)
  (fc2): Linear(in_features=2048, out_features=512, bias=True)
  (fc2_bn): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (fc3): Linear(in_features=512, out_features=256, bias=True)
  (fc3_bn): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (fc4): Linear(in_features=256, out_features=1, bias=True)
)
```

Optimizer:

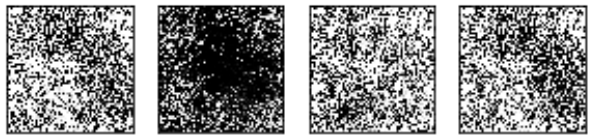
```
# Adam optimizer
G_optimizer = optim.Adam(G.parameters(), lr=lr, betas=(0.5, 0.999))
D_optimizer = optim.Adam(D.parameters(), lr=lr, betas=(0.5, 0.999))
```

Evaluation

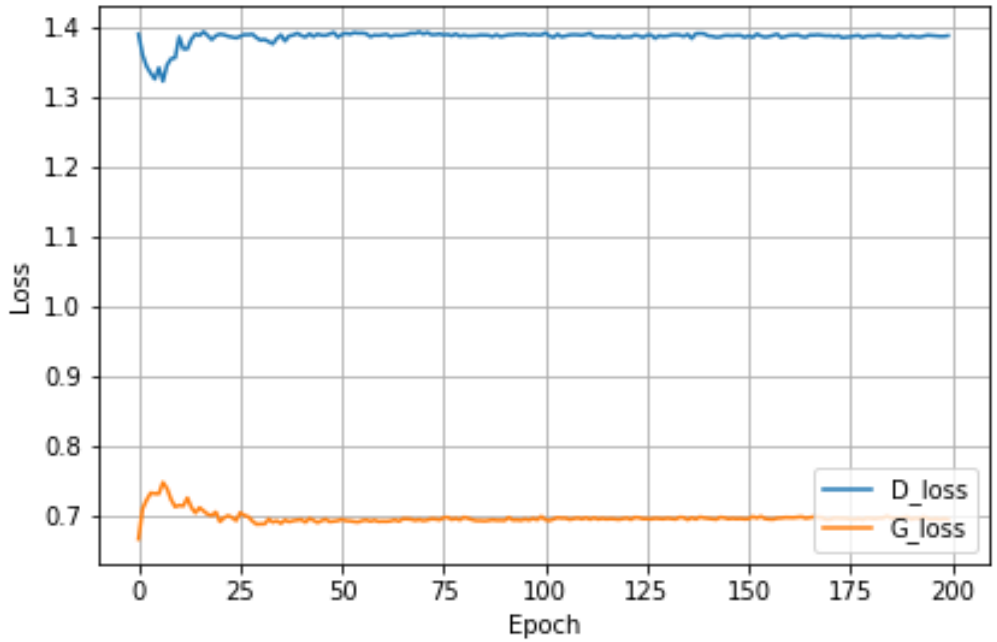
Original:



Reconstruction:

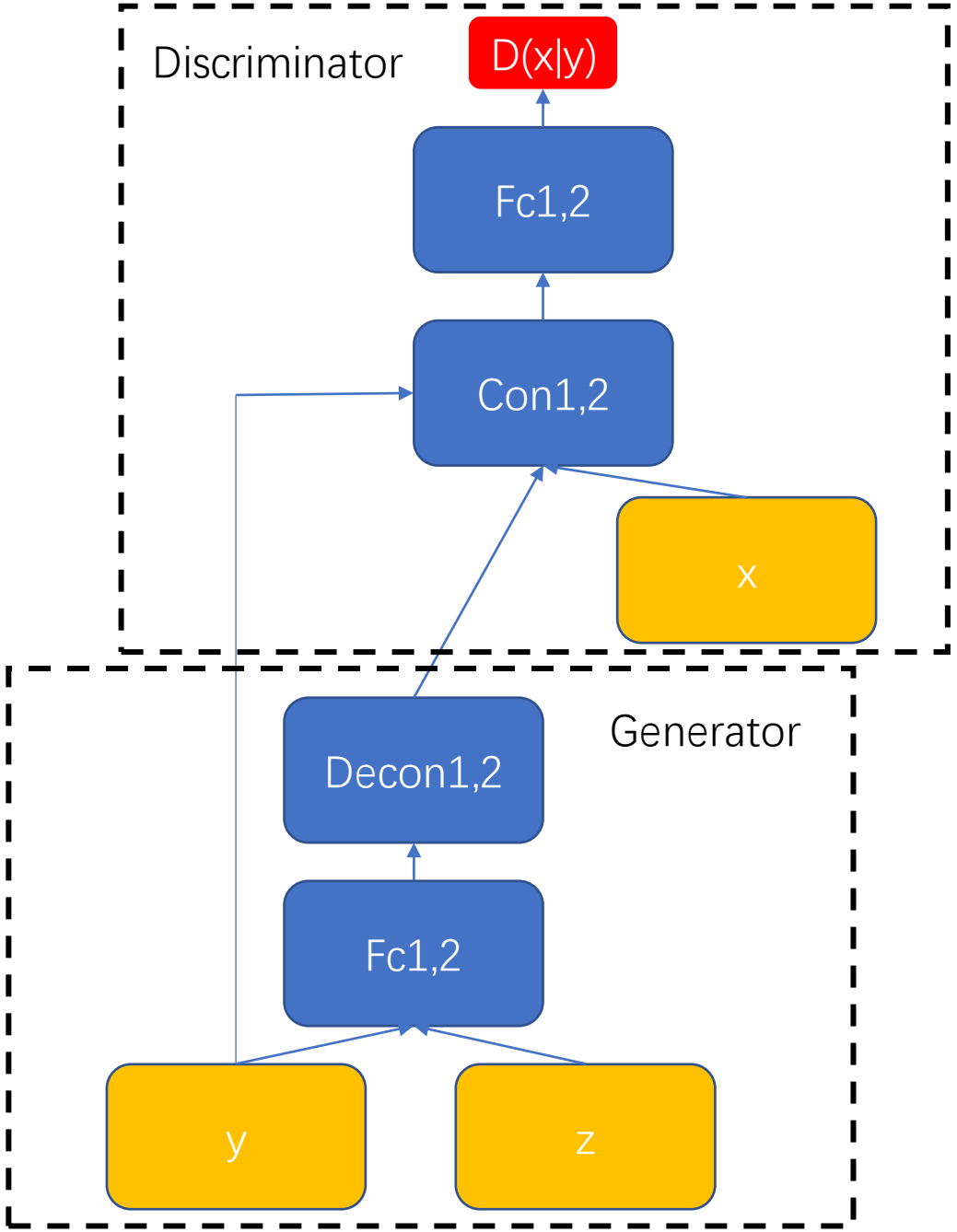


Loss:



Method

x: microstructure
y: processing parameters
z: a noise prior




```
# Adam optimizer
G_optimizer = optim.Adam(G.parameters(), lr=lr, betas=(0.5, 0.999))
D_optimizer = optim.Adam(D.parameters(), lr=lr, betas=(0.5, 0.999))
```

Method

Model structure:

```
generator(
  (fc): Sequential(
    (0): Linear(in_features=102, out_features=1024, bias=True)
    (1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): Linear(in_features=1024, out_features=320000, bias=True)
    (4): BatchNorm1d(320000, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): ReLU()
  )
  (deconv): Sequential(
    (0): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2),
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_
    (2): ReLU()
    (3): ConvTranspose2d(64, 1, kernel_size=(4, 4), stride=(2, 2), pa
    (4): Tanh()
  )
)
```

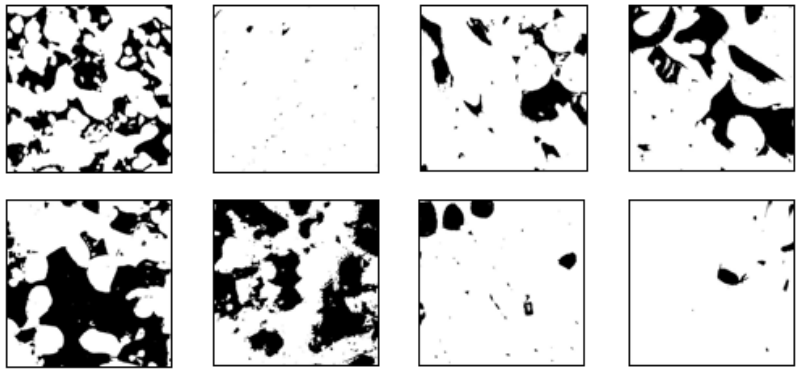
```
Out[456]:
discriminator(
  (conv): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running
    (4): LeakyReLU(negative_slope=0.2)
  )
  (fc): Sequential(
    (0): Linear(in_features=320000, out_features=1024, bias=True)
    (1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2)
    (3): Linear(in_features=1024, out_features=1, bias=True)
    (4): Sigmoid()
  )
)
```

Optimizer:

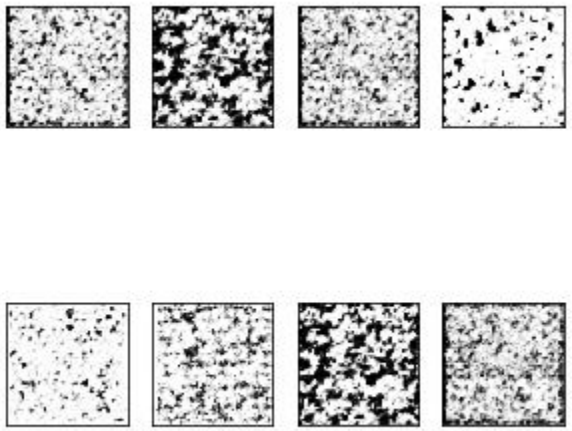
```
# Adam optimizer
G_optimizer = optim.Adam(G.parameters(), lr=lr, betas=(0.5, 0.999))
D_optimizer = optim.Adam(D.parameters(), lr=lr, betas=(0.5, 0.999))
```

Evaluation

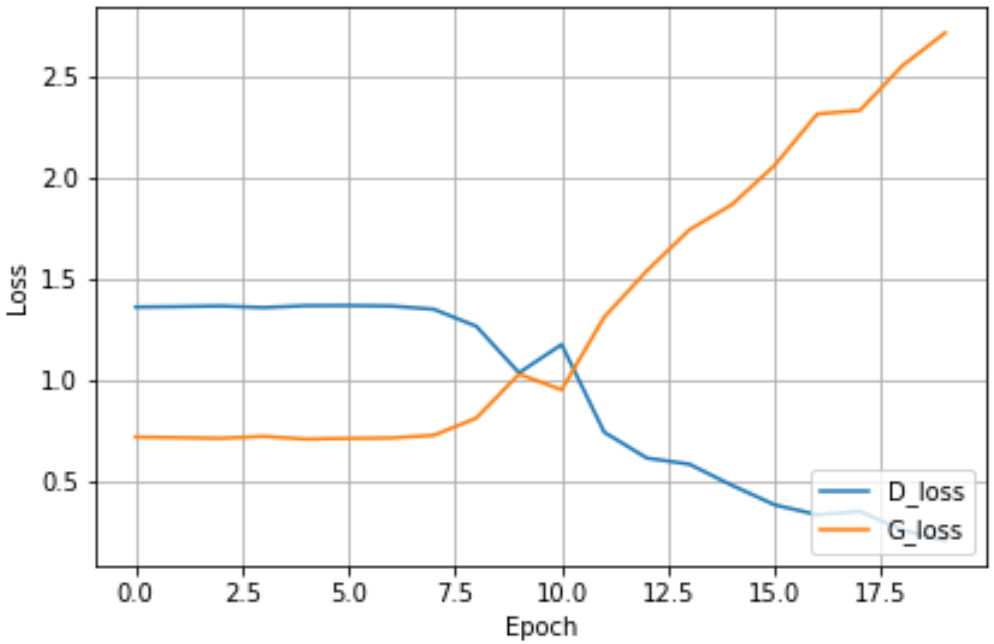
Original:



Reconstruction:



Loss:



Any suggestion?