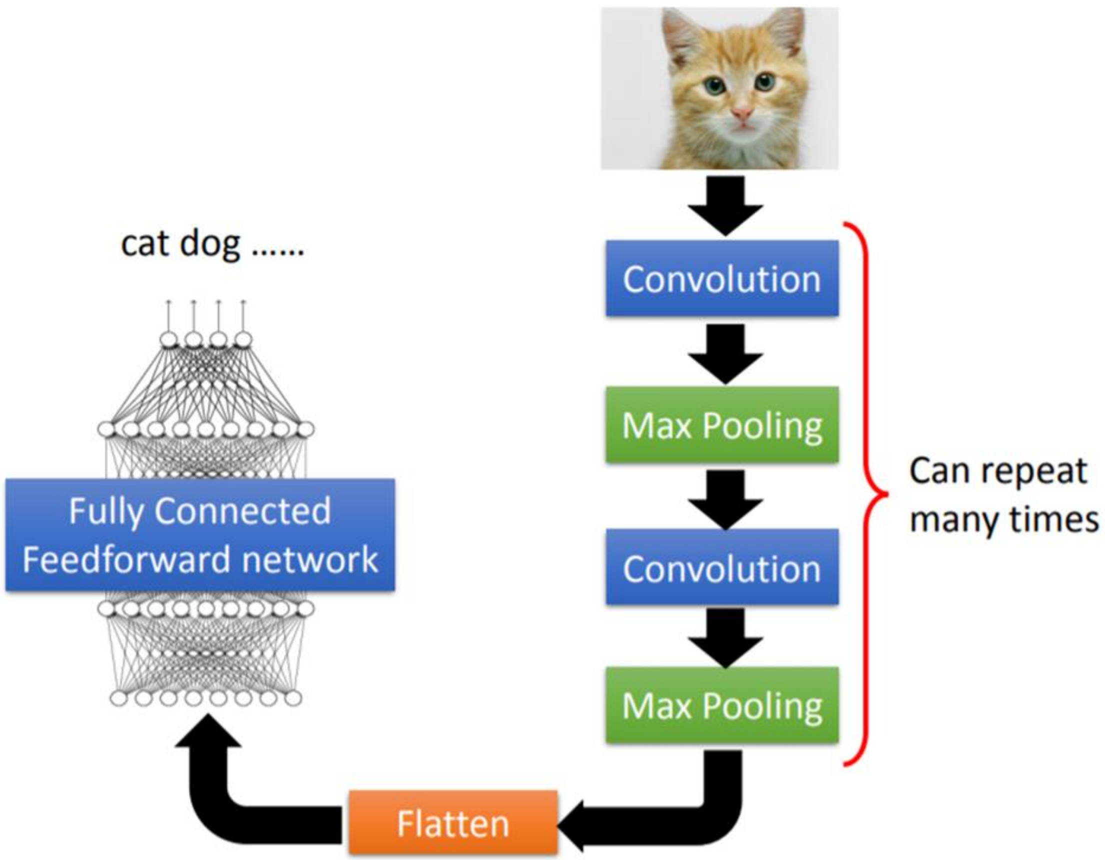
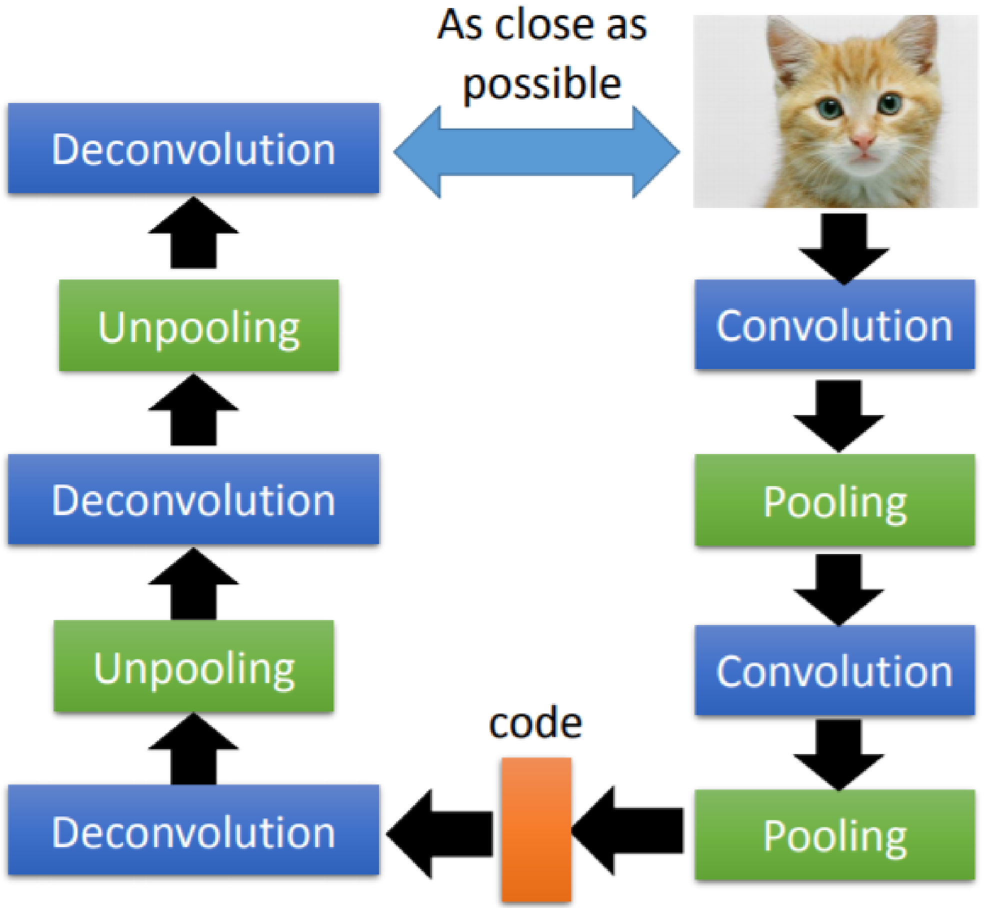
Auto-encoder

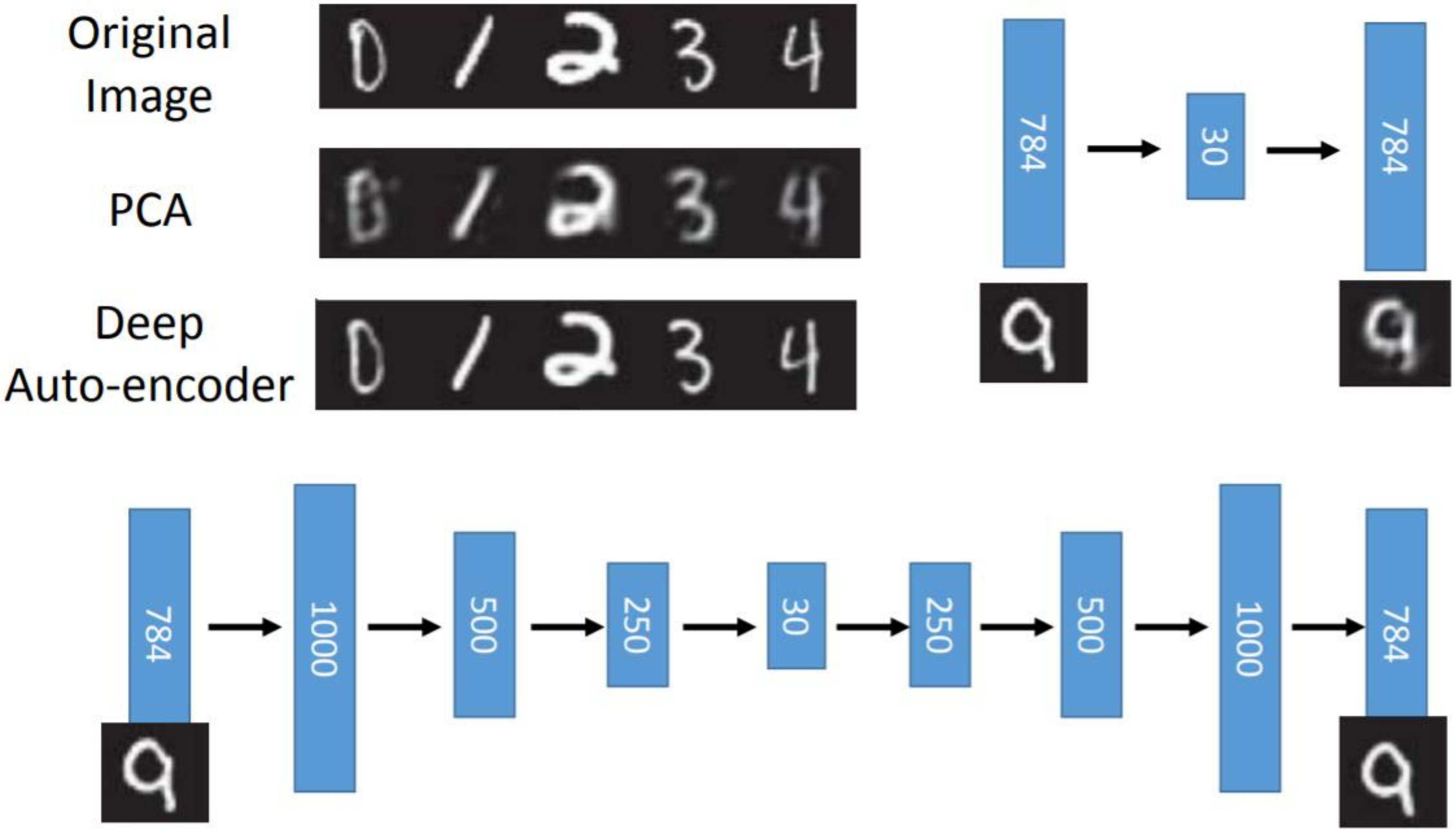
- CNN Image Classifier – Convolution section + MLP classifier
- CNN Autoencoder – Convolution section + Deconvolution section to recover the input image





Reference: 李弘毅 ML Lecture 16 https://youtu.be/Tk5B4seA-AU

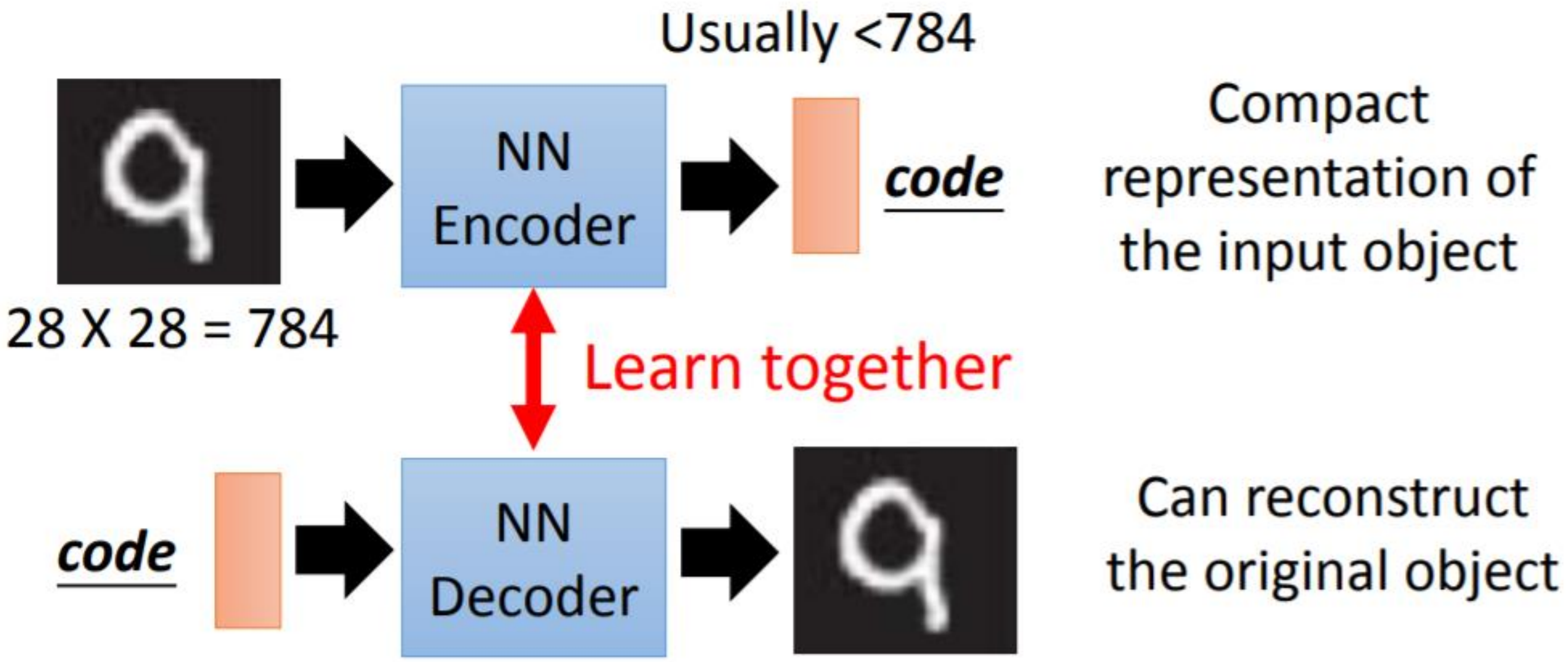
MLP based autoencoder



Reference: Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." *Science* 313.5786 (2006): 504-507

Reference: 李弘毅 ML Lecture 16 https://youtu.be/Tk5B4seA-AU

Autoencoder learns a compact representation of the input image



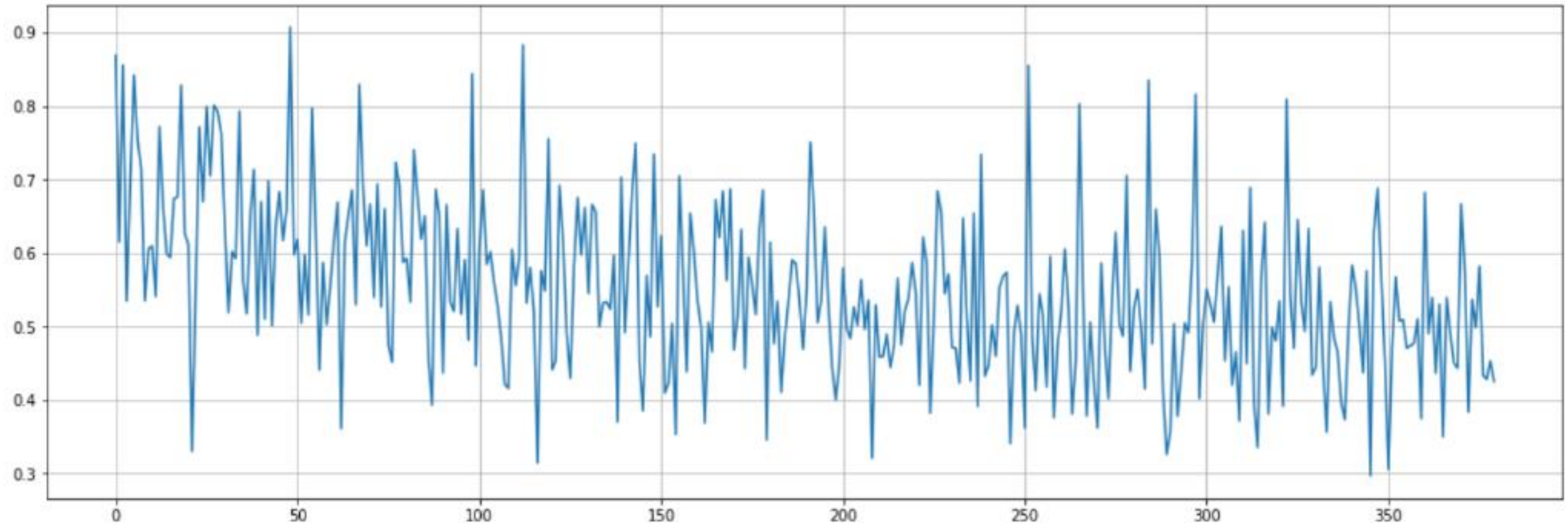
Reference: 李弘毅 ML Lecture 16 https://youtu.be/Tk5B4seA-AU

Practice

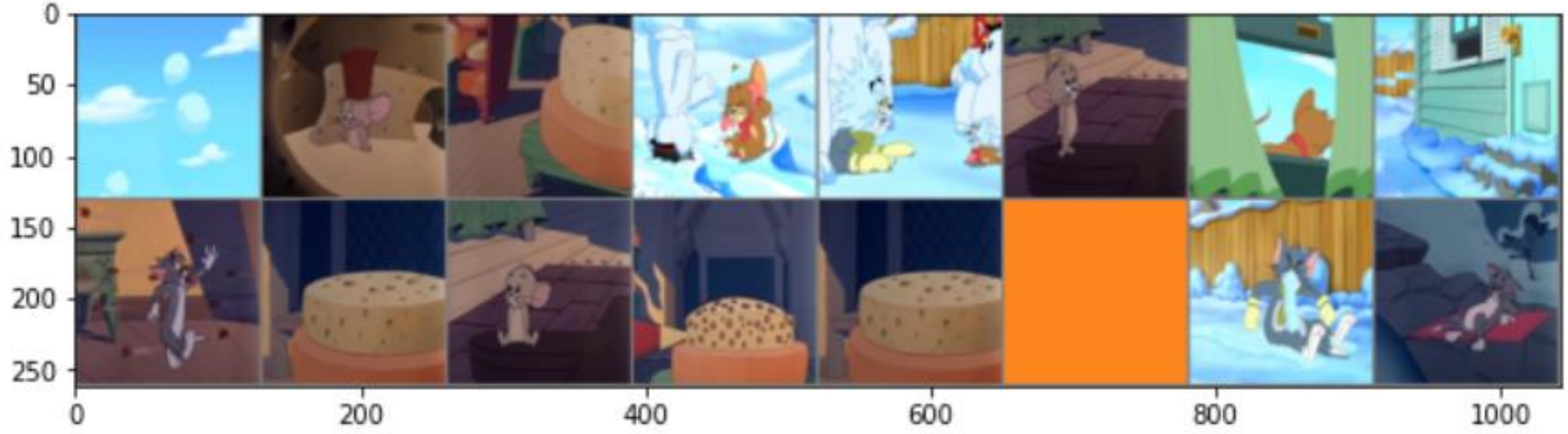
- Run “7.1.Conv_AE.ipynb"

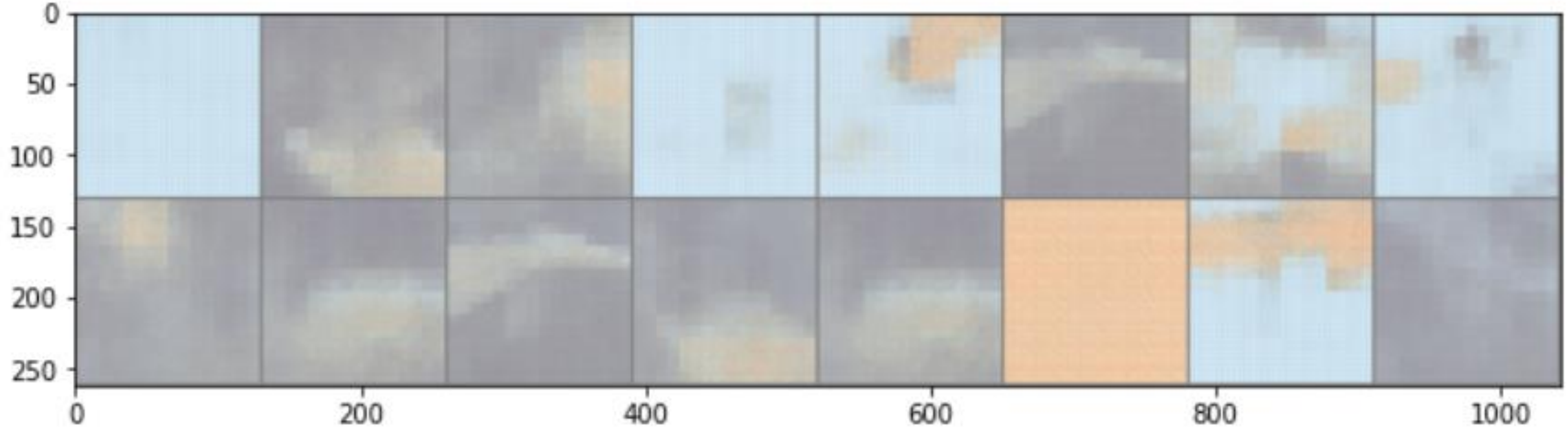
After train 20 epochs

Input size=128x128, batch size=16

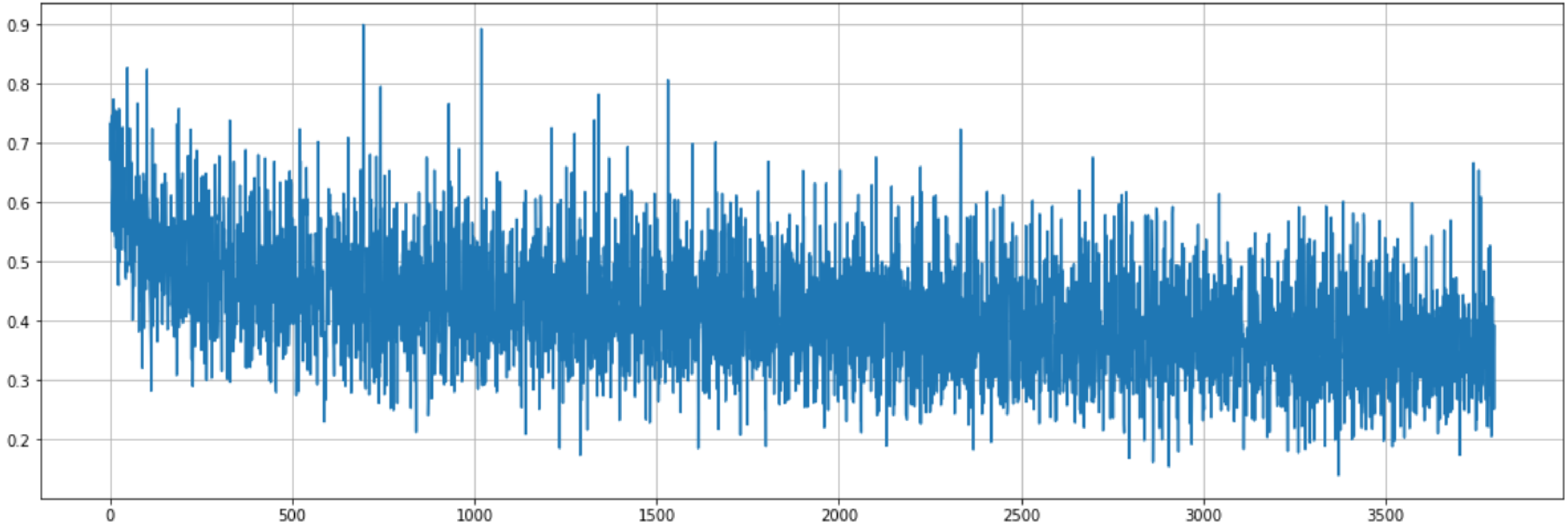


After train 20 epochs



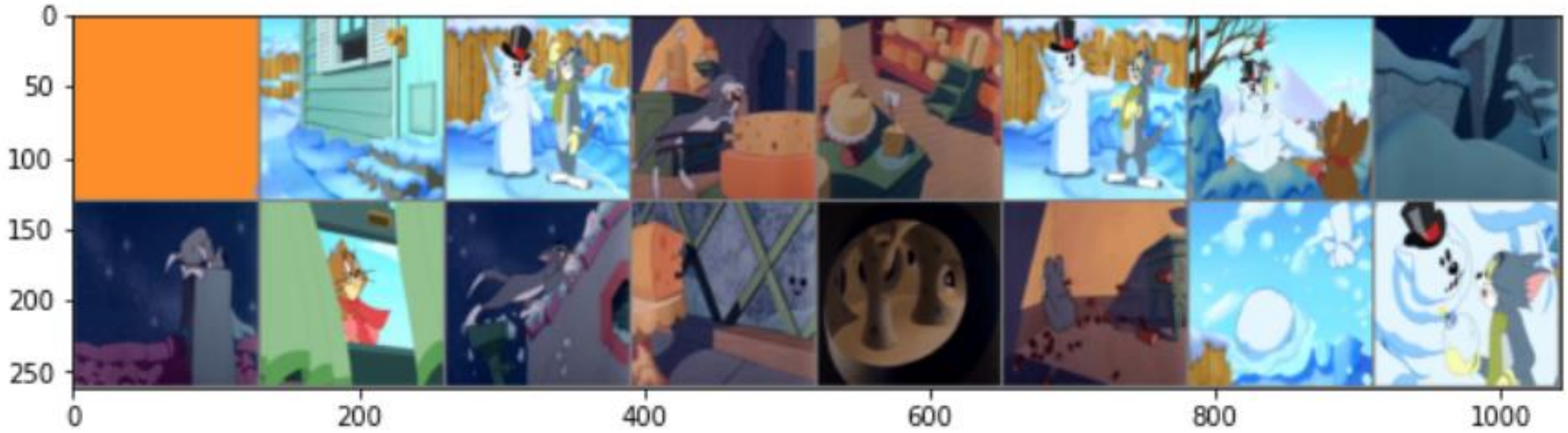


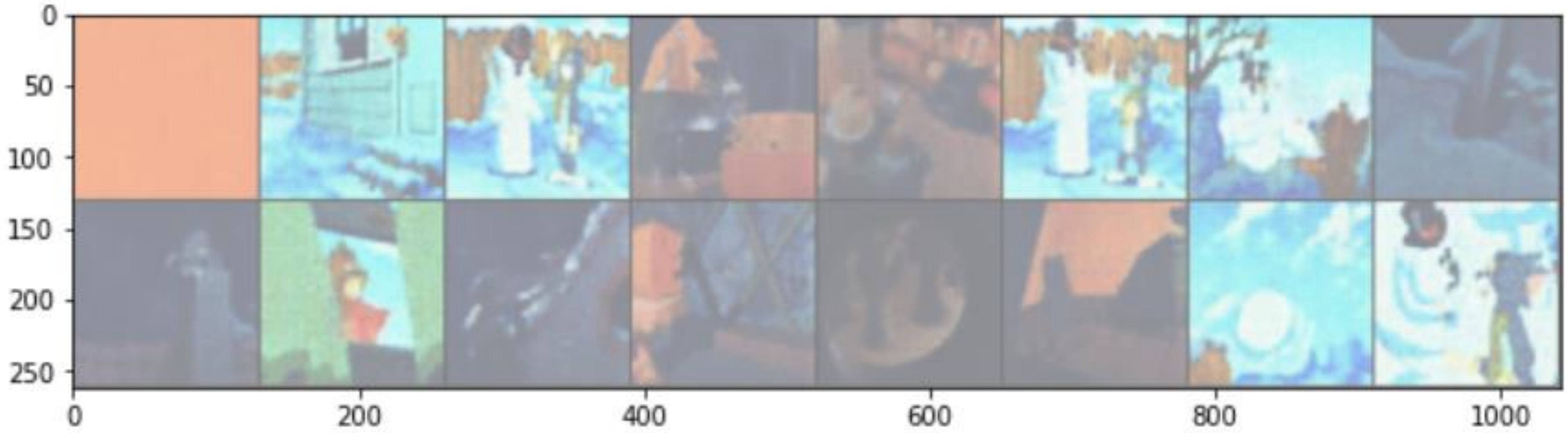
After train 200 epochs



After train 200 epochs

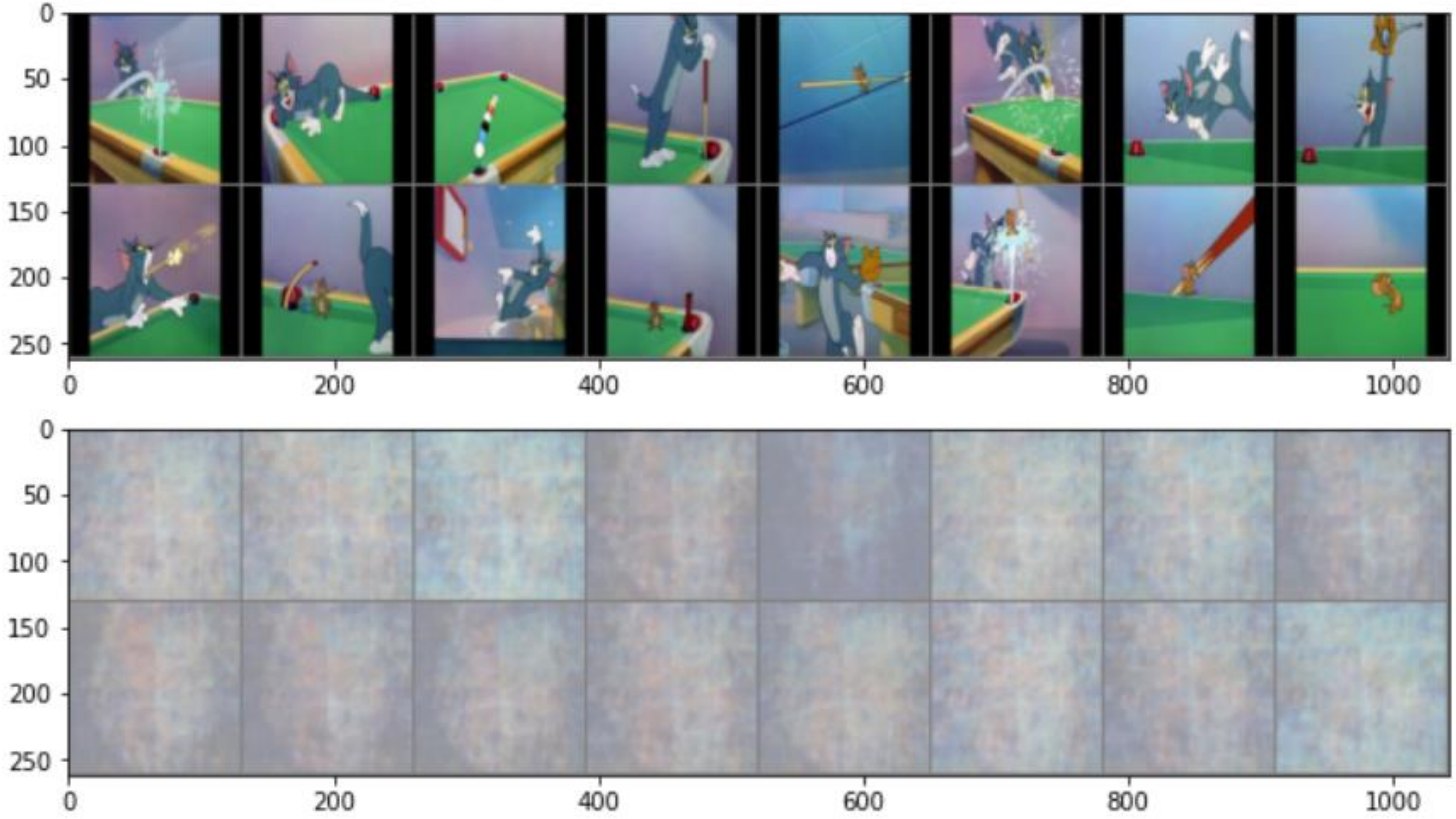
Test on training images – the NN is able to recover more from the input images





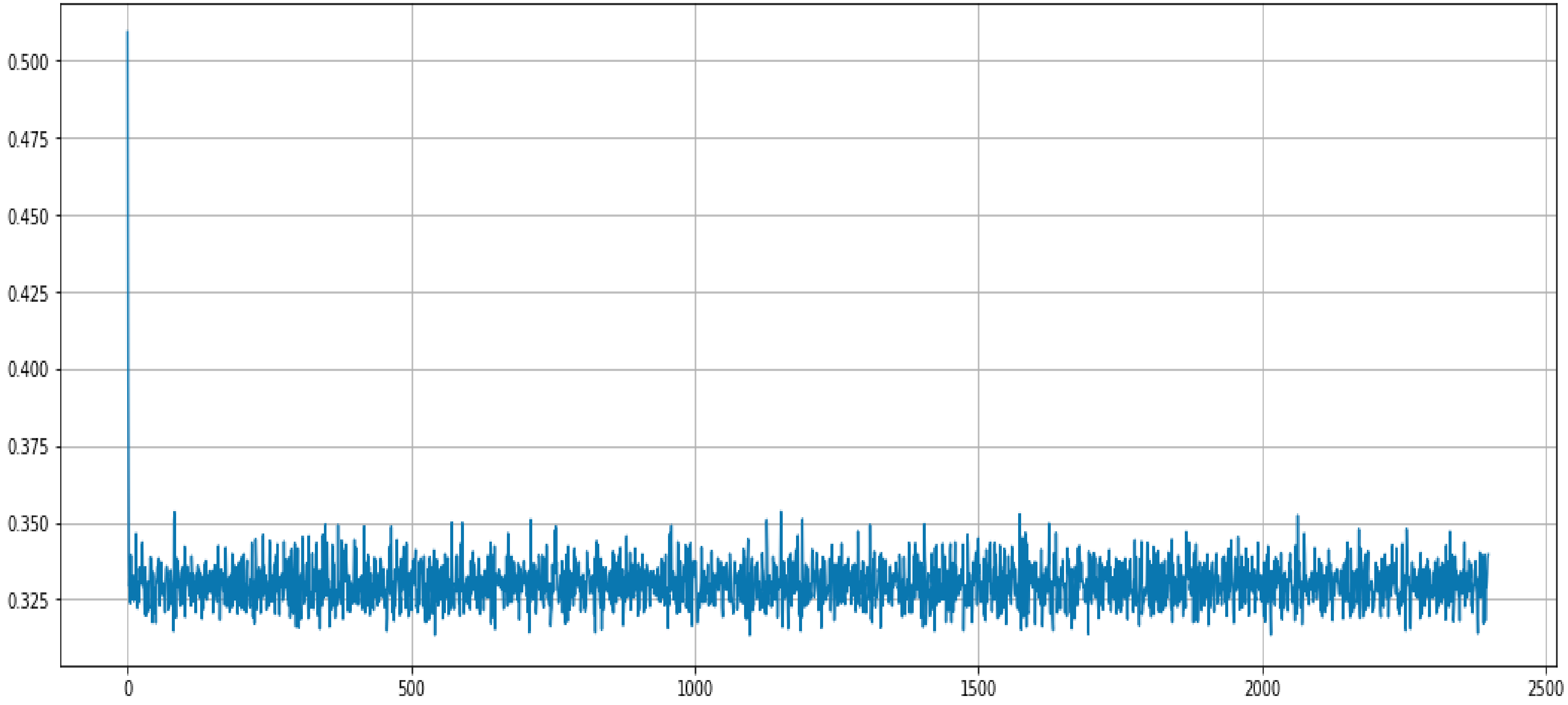
After train 200 epochs

Test on un-seen images – fails to reconstruct the input images

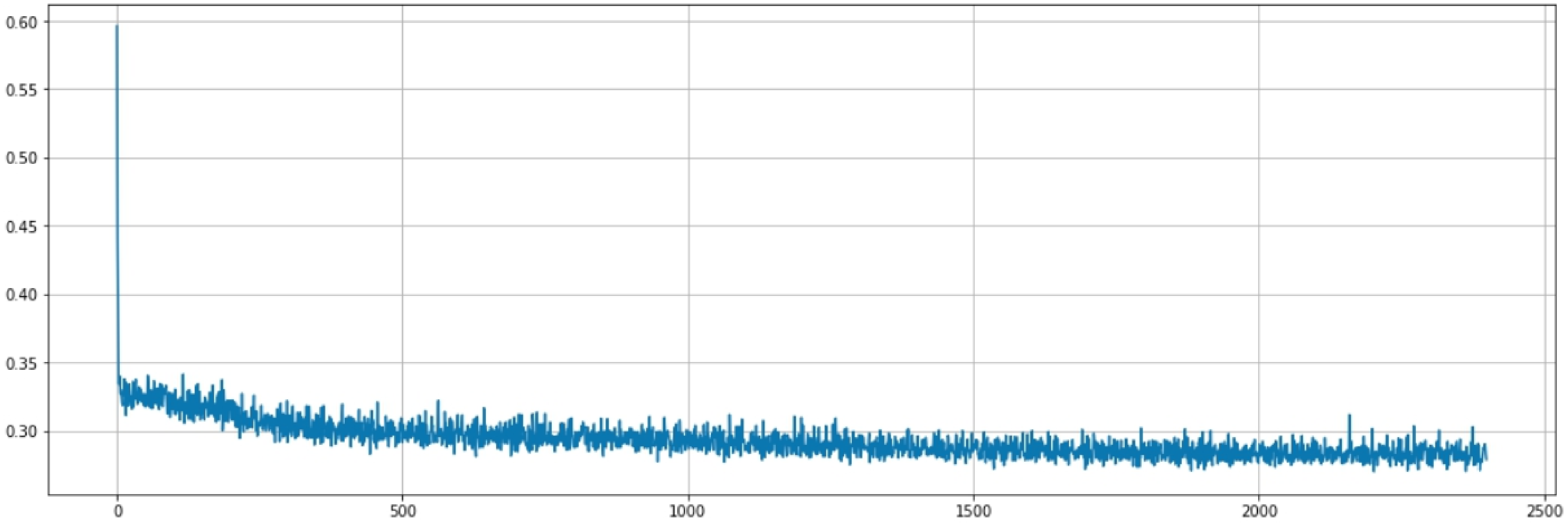


Adding another 200 epochs (Total = 400)

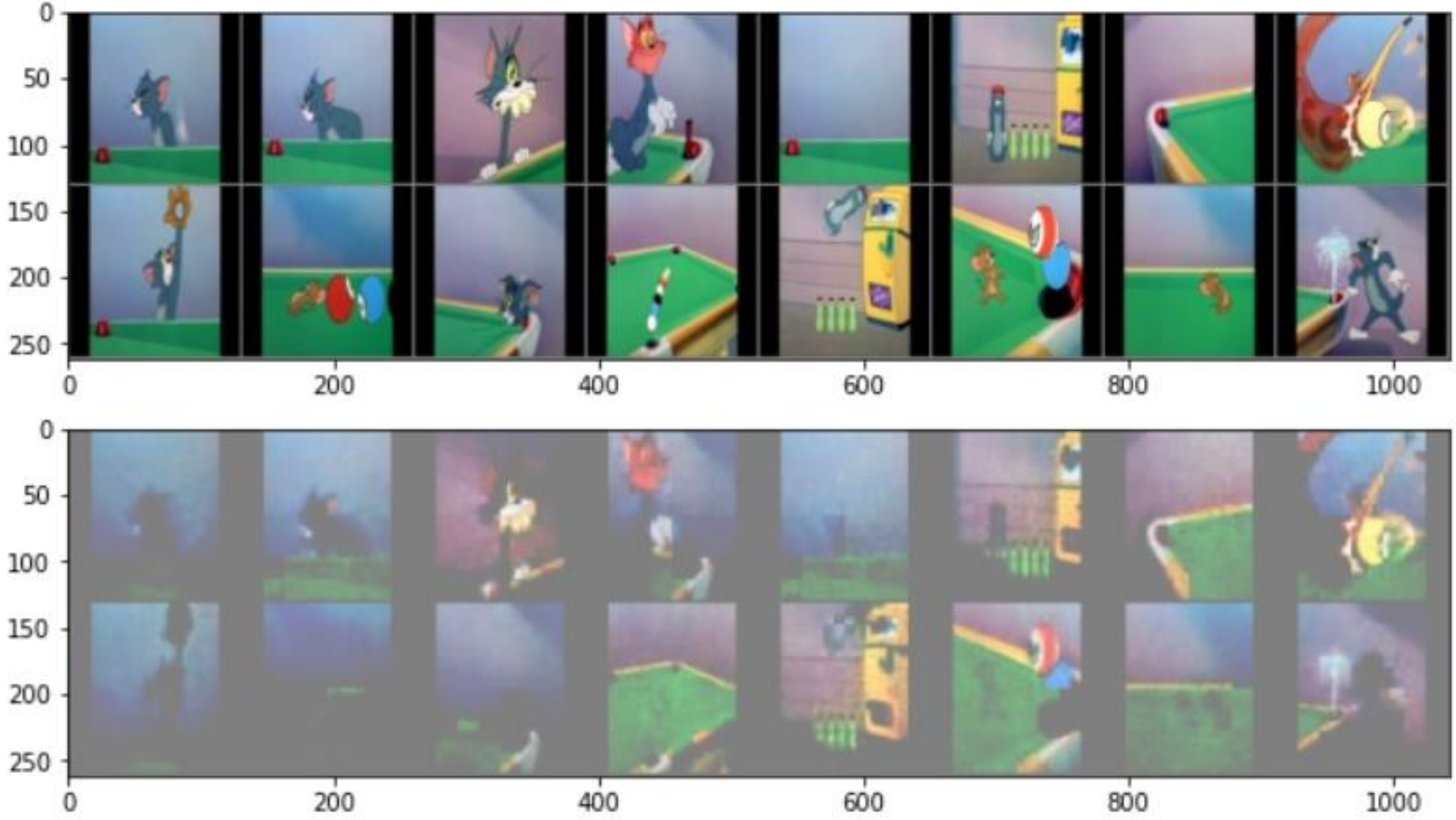
Most students have this loss plot for epoch 200~400 and the AE is not able to recover the test images. Why?



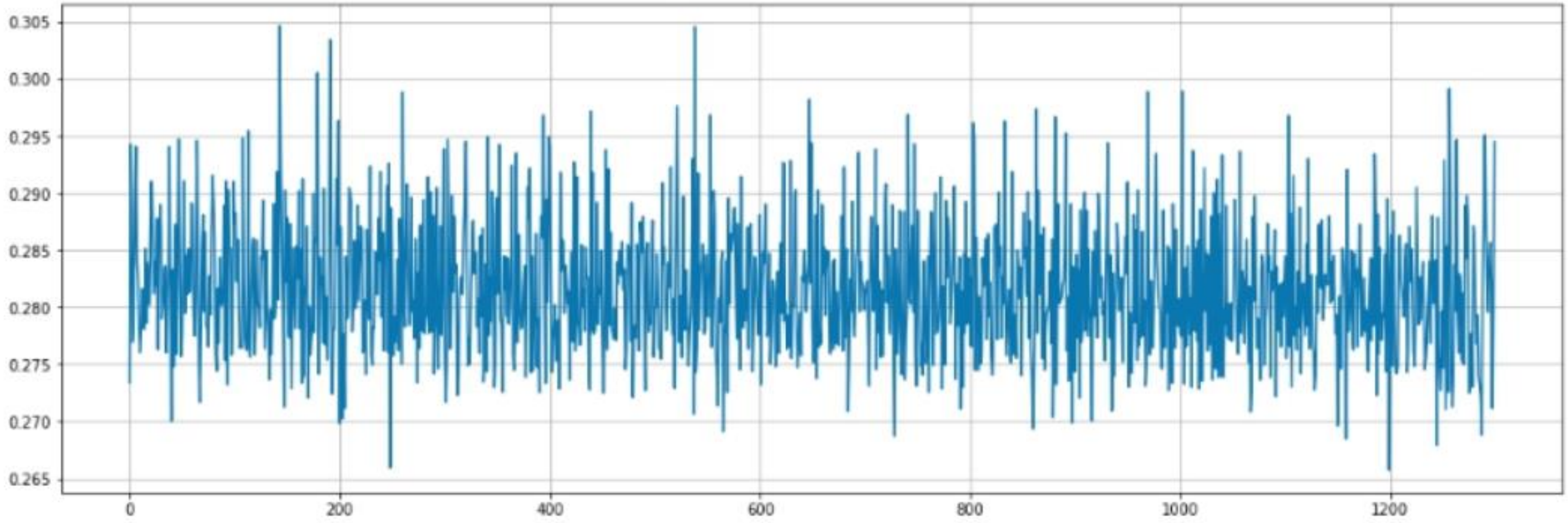
One student has this loss plot for epoch 200~400 and the AE is able to recover the test images. Why he can succeed?



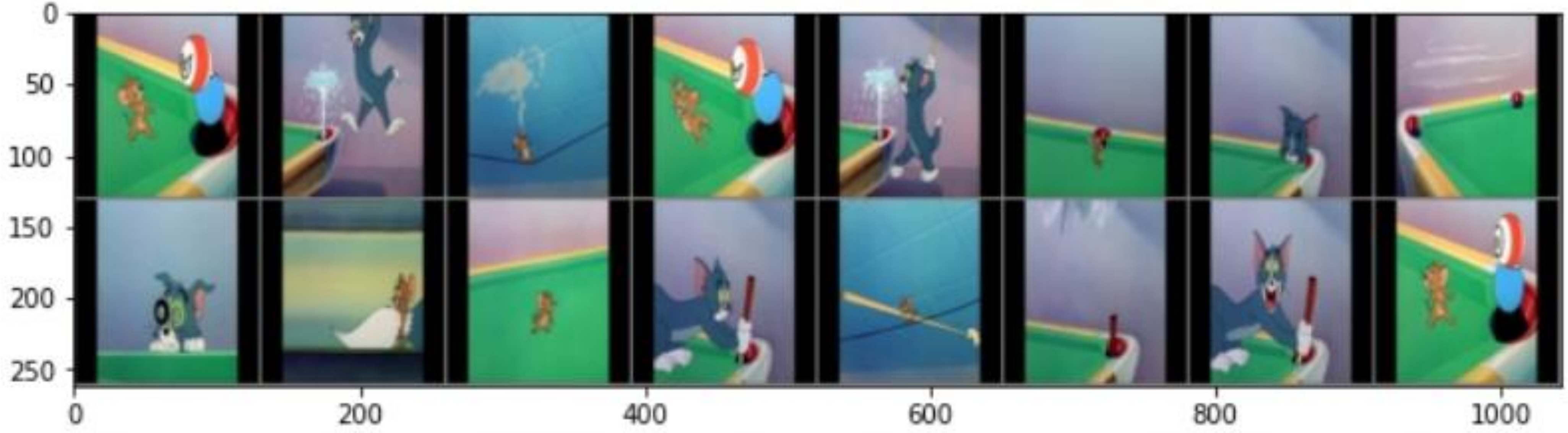
One student has this loss plot for epoch 200~400 and the AE is able to recover the test images. Why he can succeed?

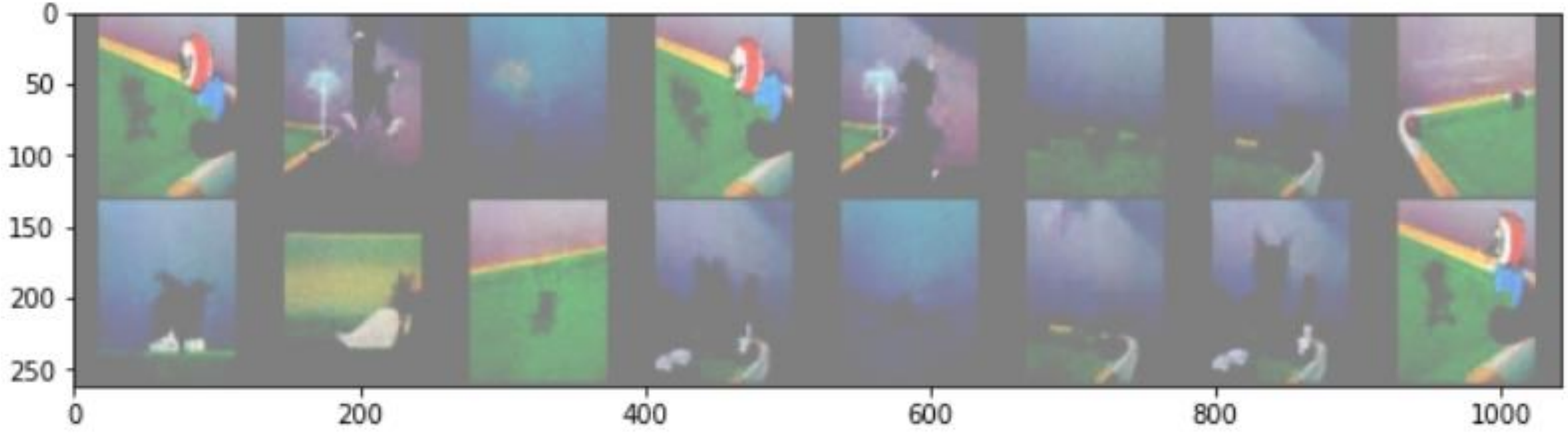


He keeps training another 100 epochs (Total = 500 epochs). The loss plot for epoch 400-500

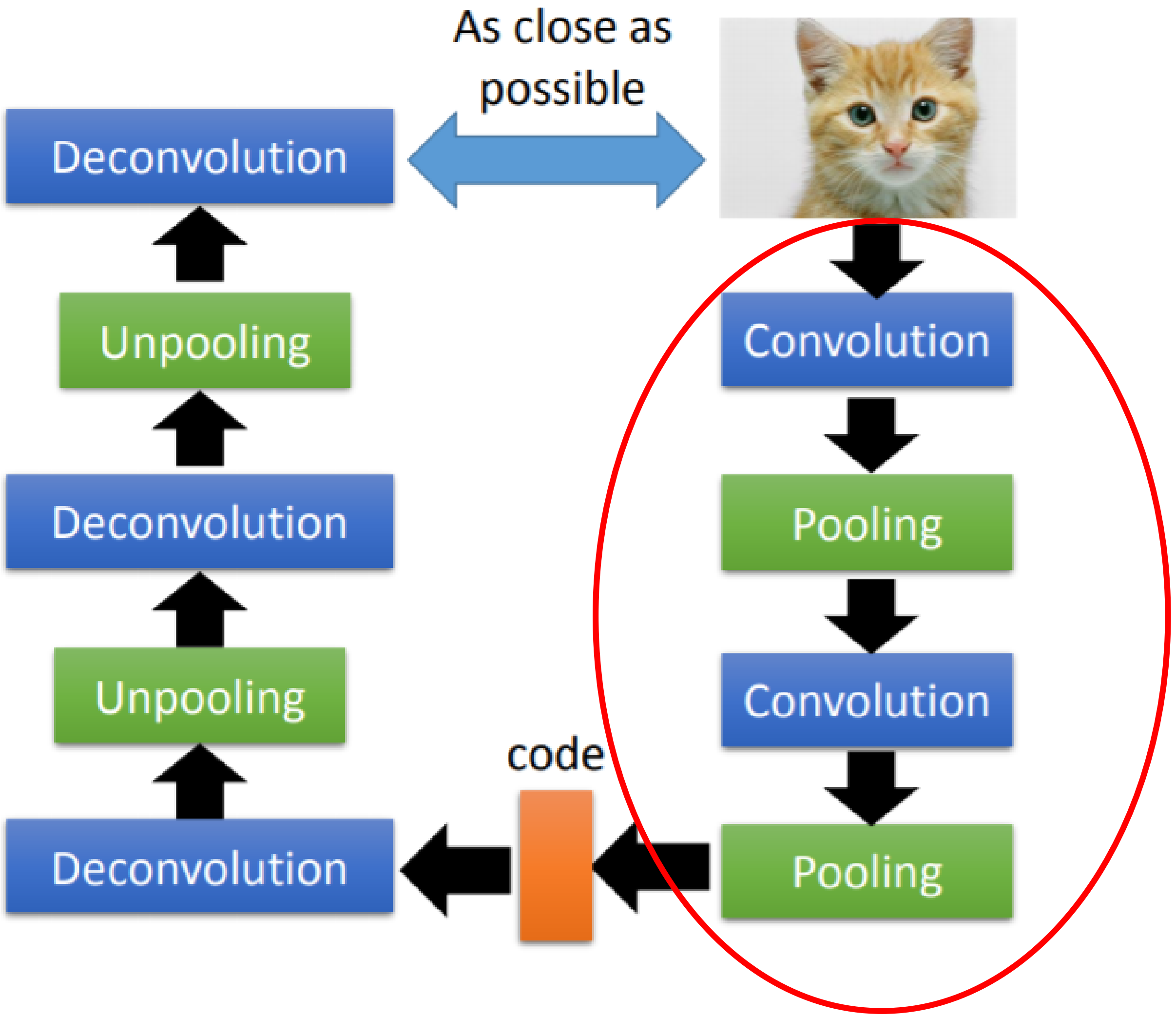


He keeps training another 100 epochs (Total = 500 epochs). The loss plot for epoch 400-500





Encoder



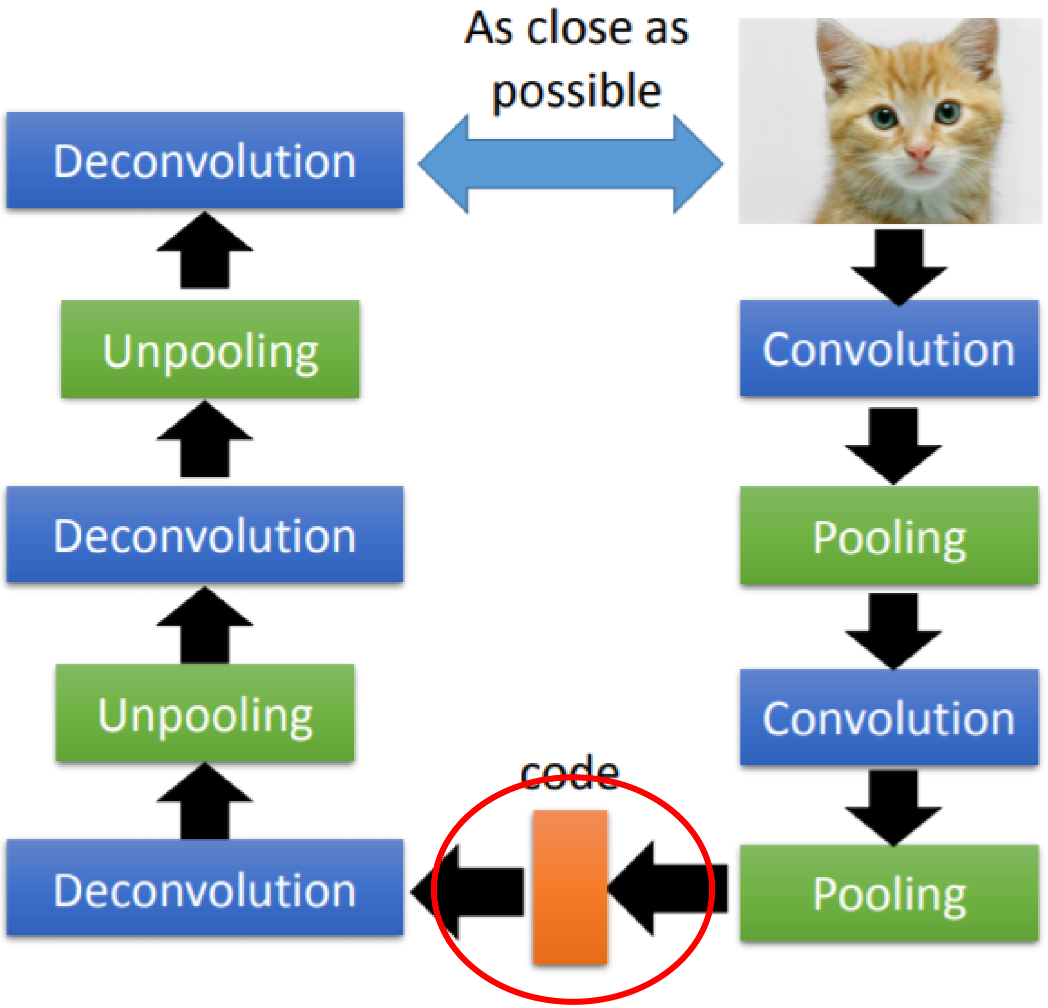

```
self.encoder = nn.Sequential(
  nn.Conv2d(3, 32, kernel_size=2, stride=2),
  nn.BatchNorm2d(32, eps=1e-05, momentum=0.1, af
  nn.ReLU(),
  nn.Conv2d(32, 64, kernel_size=2, stride=2),
  nn.BatchNorm2d(64, eps=1e-05, momentum=0.1, af
  nn.ReLU(),
  nn.Conv2d(64, 128, kernel_size=2, stride=2),
  nn.BatchNorm2d(128, eps=1e-05, momentum=0.1, a
  nn.ReLU(),
  nn.Conv2d(128, 256, kernel_size=2, stride=2),
  nn.BatchNorm2d(256, eps=1e-05, momentum=0.1, a
  nn.ReLU(),
  nn.Conv2d(256, 512, kernel_size=2, stride=2),
  nn.BatchNorm2d(512, eps=1e-05, momentum=0.1, a
  nn.ReLU(),
  nn.Conv2d(512, 1024, kernel_size=2, stride=2),
  nn.BatchNorm2d(1024, eps=1e-05, momentum=0.1,
  nn.ReLU(),
  nn.Conv2d(1024, 1024, kernel_size=2, stride=2)
  nn.BatchNorm2d(1024, eps=1e-05, momentum=0.1,
  nn.ReLU(),
  Flatten(),
  nn.Linear(in_features=i, out_features=o),
)
```

Reference: 李弘毅 ML Lecture 16 https://youtu.be/Tk5B4seA-AU

Practice: Draw the feature maps of encoder

- Let input image = 224x224x3
- Draw the feature maps (H, W, depth) after each convolution and max pooling
- What is the number of nodes after flatten?

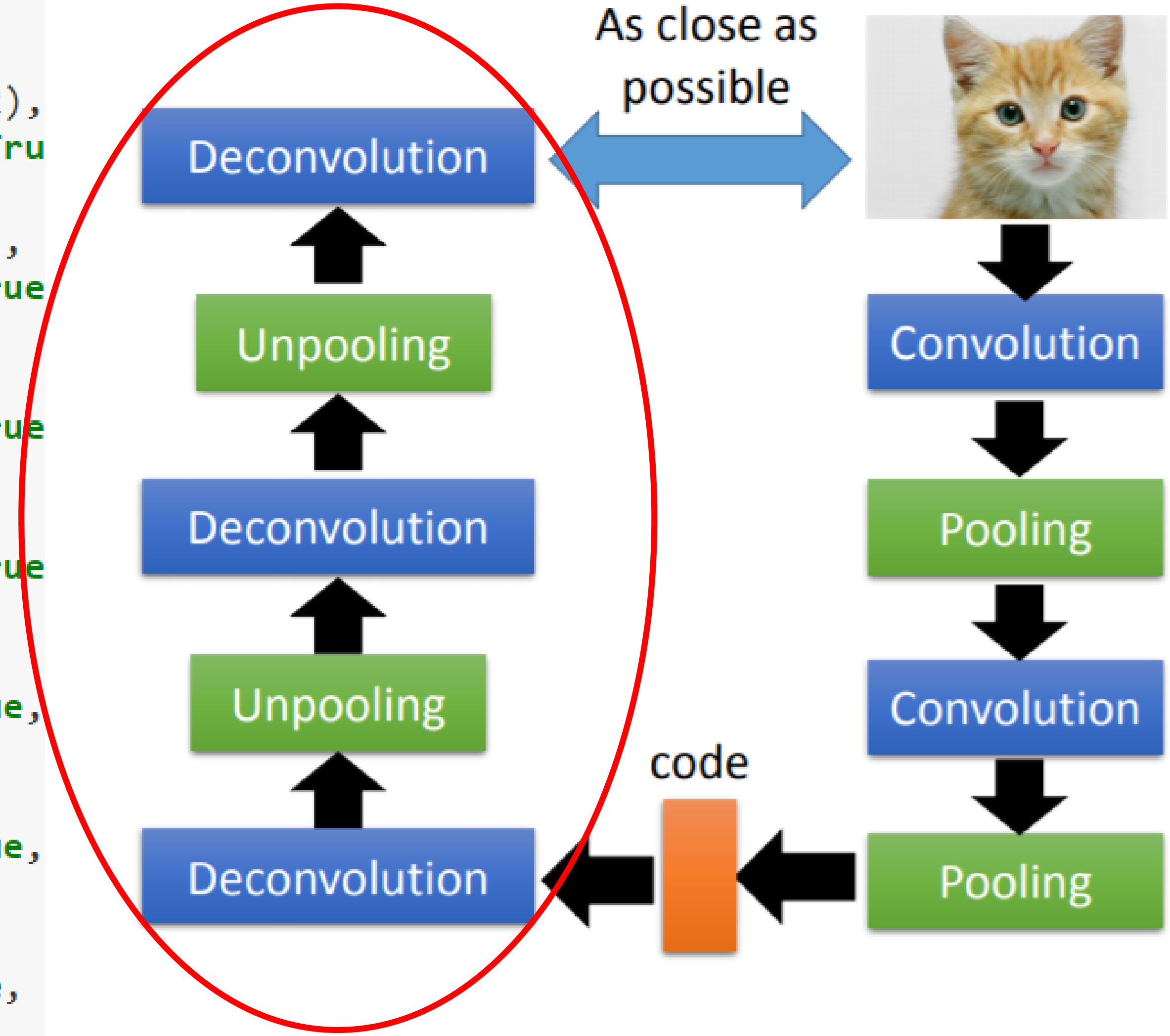
Latent vector



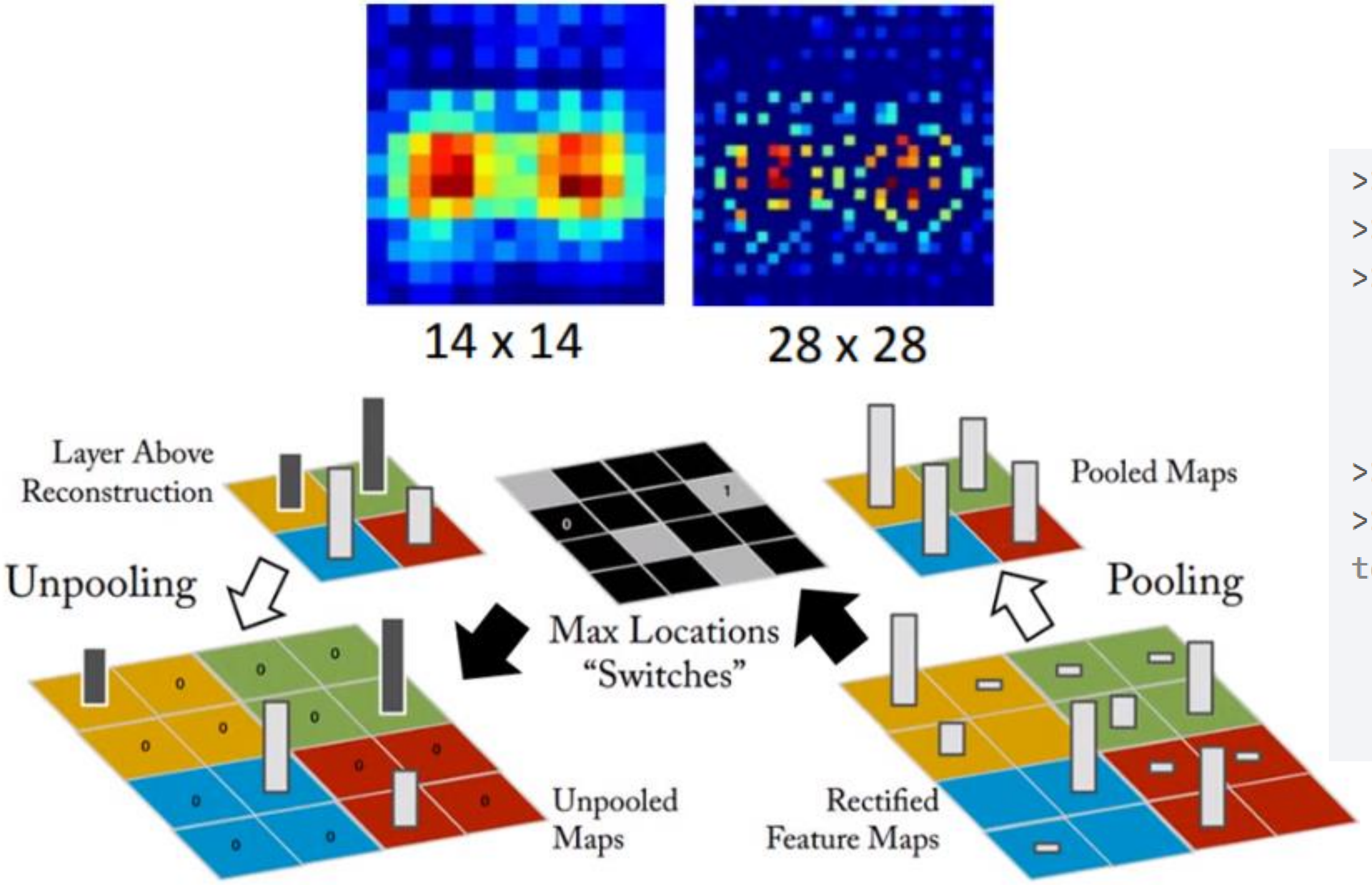

```
     Flatten-22                  [-1, 1024]
      Linear-23                    [-1, 64]
      Linear-24                  [-1, 1024]
   UnFlatten-25            [-1, 1024, 1, 1]
```

Decoder

```
self.decoder = nn.Sequential(
  nn.Linear(in_features=o, out_features=i),
  UnFlatten(),
  nn.ConvTranspose2d(1024, 1024, kernel_size=2, stride=2),
  nn.BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=Tru
  nn.ReLU(),
  nn.ConvTranspose2d(1024, 512, kernel_size=2, stride=2),
  nn.BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True
  nn.ReLU(),
  nn.ConvTranspose2d(512, 256, kernel_size=2, stride=2),
  nn.BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True
  nn.ReLU(),
  nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2),
  nn.BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True
  nn.ReLU(),
  nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2),
  nn.BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
  nn.ReLU(),
  nn.ConvTranspose2d(64, 32, kernel_size=2, stride=2),
  nn.BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
  nn.ReLU(),
  nn.ConvTranspose2d(32, 3, kernel_size=2, stride=2),
  nn.BatchNorm2d(3, eps=1e-05, momentum=0.1, affine=True,
  nn.Sigmoid(),
)
```



Unpooling

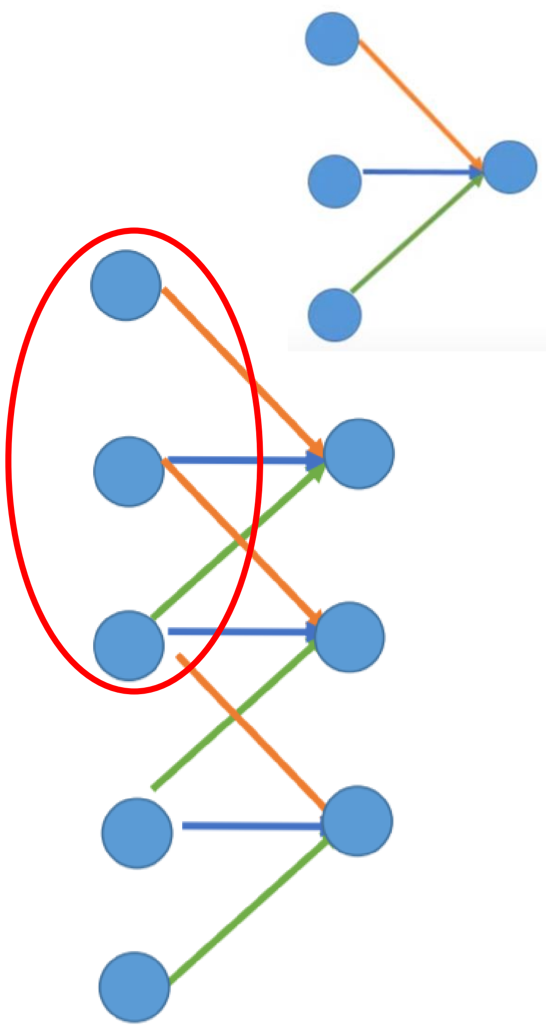



```
>>> pool = nn.MaxPool2d(2, stride=2, return_indices=True)
>>> unpool = nn.MaxUnpool2d(2, stride=2)
>>> input = torch.tensor([[[[ 1.,  2,  3,  4],
                            [ 5,  6,  7,  8],
                            [ 9, 10, 11, 12],
                            [13, 14, 15, 16]]]])
>>> output, indices = pool(input)
>>> unpool(output, indices)
tensor([[[[  0.,   0.,   0.,   0.],
          [  0.,   6.,   0.,   8.],
          [  0.,   0.,   0.,   0.],
          [  0.,  14.,   0.,  16.]]]])
```

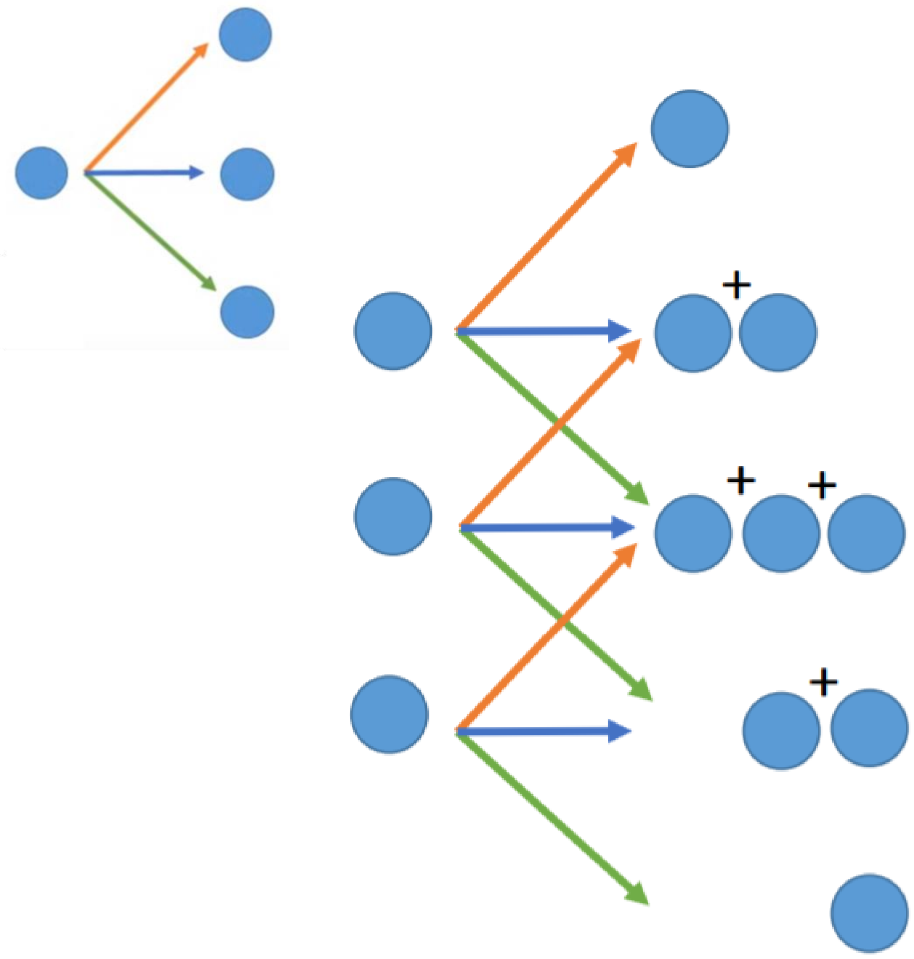
Reference: 李弘毅 ML Lecture 16 https://youtu.be/Tk5B4seA-AU

Deconvolution

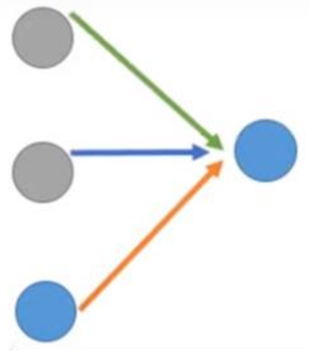
1D convolution, filter size=3

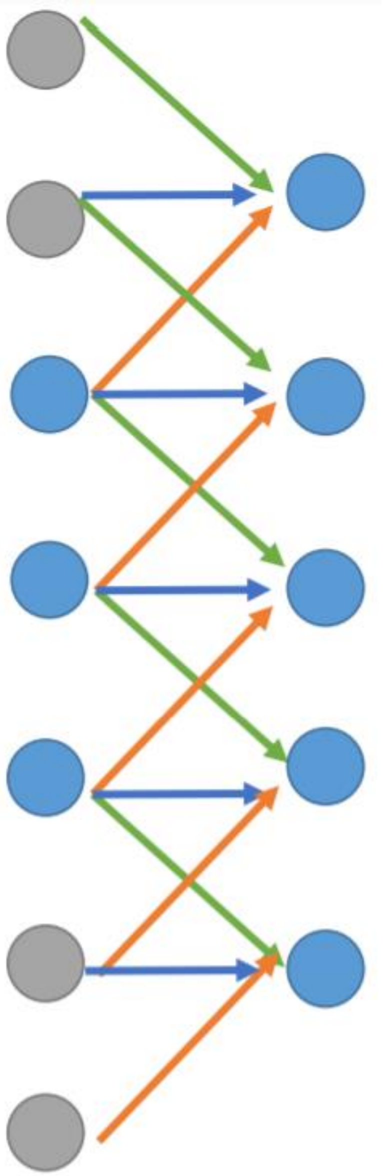


1D deconvolution, filter size=3



1D convolution,
filter size=3





Reference: 李弘毅 ML Lecture 16 https://youtu.be/Tk5B4seA-AU

Practice: Draw the feature maps of decoder

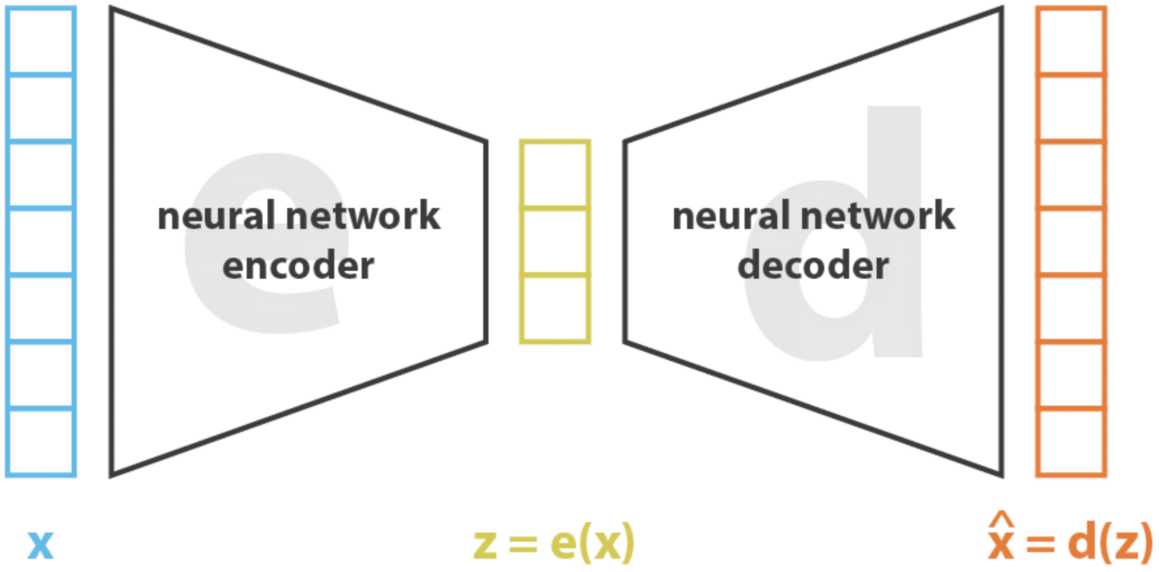
- Input – the number of nodes after un-flattern
- Draw feature maps (H, W, depth) after each de-convolution and un-max pooling

Deconvolution

```
(2): ConvTranspose2d(1024, 1024, kernel_size=(2, 2), stride=(2, 2))
(3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_r
(4): ReLU()
(5): ConvTranspose2d(1024, 512, kernel_size=(2, 2), stride=(2, 2))
(6): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
(7): ReLU()
(8): ConvTranspose2d(512, 256, kernel_size=(2, 2), stride=(2, 2))
(9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
(10): ReLU()
```

```
    ConvTranspose2d-26           [-1, 1024, 2, 2]
        BatchNorm2d-27           [-1, 1024, 2, 2]
               ReLU-28           [-1, 1024, 2, 2]
    ConvTranspose2d-29            [-1, 512, 4, 4]
        BatchNorm2d-30            [-1, 512, 4, 4]
               ReLU-31            [-1, 512, 4, 4]
    ConvTranspose2d-32            [-1, 256, 8, 8]
        BatchNorm2d-33            [-1, 256, 8, 8]
               ReLU-34            [-1, 256, 8, 8]
    ConvTranspose2d-35          [-1, 128, 16, 16]
        BatchNorm2d-36          [-1, 128, 16, 16]
               ReLU-37          [-1, 128, 16, 16]
    ConvTranspose2d-38           [-1, 64, 32, 32]
        BatchNorm2d-39           [-1, 64, 32, 32]
               ReLU-40           [-1, 64, 32, 32]
```

Loss function



$$\text{loss} = \| x - \hat{x} \|^2 = \| x - d(z) \|^2 = \| x - d(e(x)) \|^2$$

Source: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

[13]:
```
for batchX, _ in loader:
  break;
print(batchX.shape)
```

```
torch.Size([16, 3, 128, 128])
```

[14]:
```
tensorY=model(batchX.to(device))
print(tensorY.shape)
```

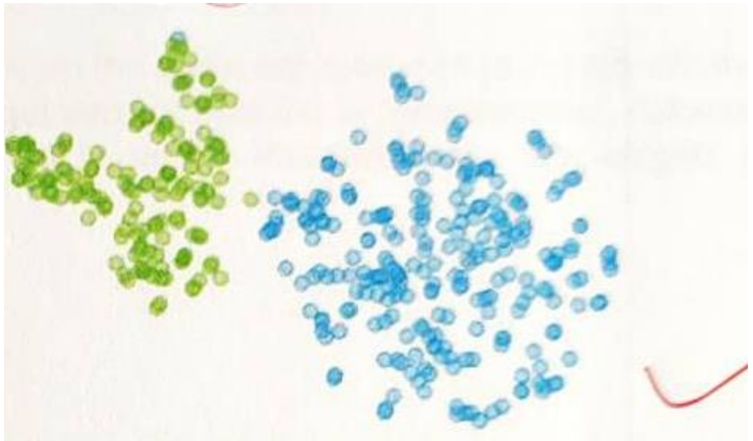
```
torch.Size([16, 3, 128, 128])
```

[15]:
```
loss = loss_func(tensorY, batchX.to(device))
print(loss)
```

```
tensor(0.6961, device='cuda:0', grad_fn=<MseL
```

HW5 (1)

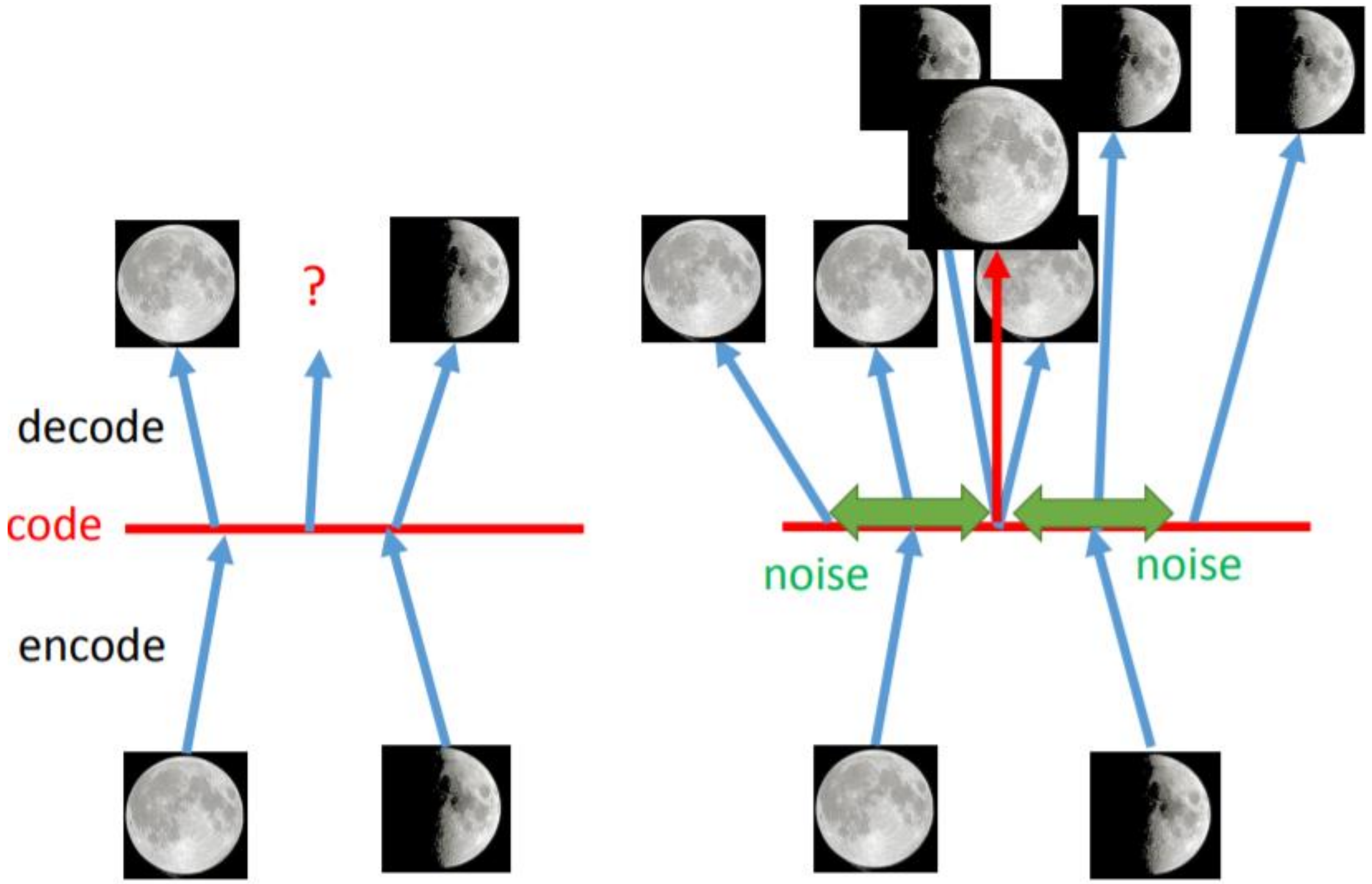
- Train an AE to learn a compact representation (try latent vector of size 20, 30, 50) of your facial expression. Test with 10 happy and 10 angry faces.
- Show the recovered image.
- Send the latent vectors to *t*-SNE or PCA to see whether they form clusters.



Vibrational Auto-Encoder (VAE)

Why VAE?

Assume 1-d code

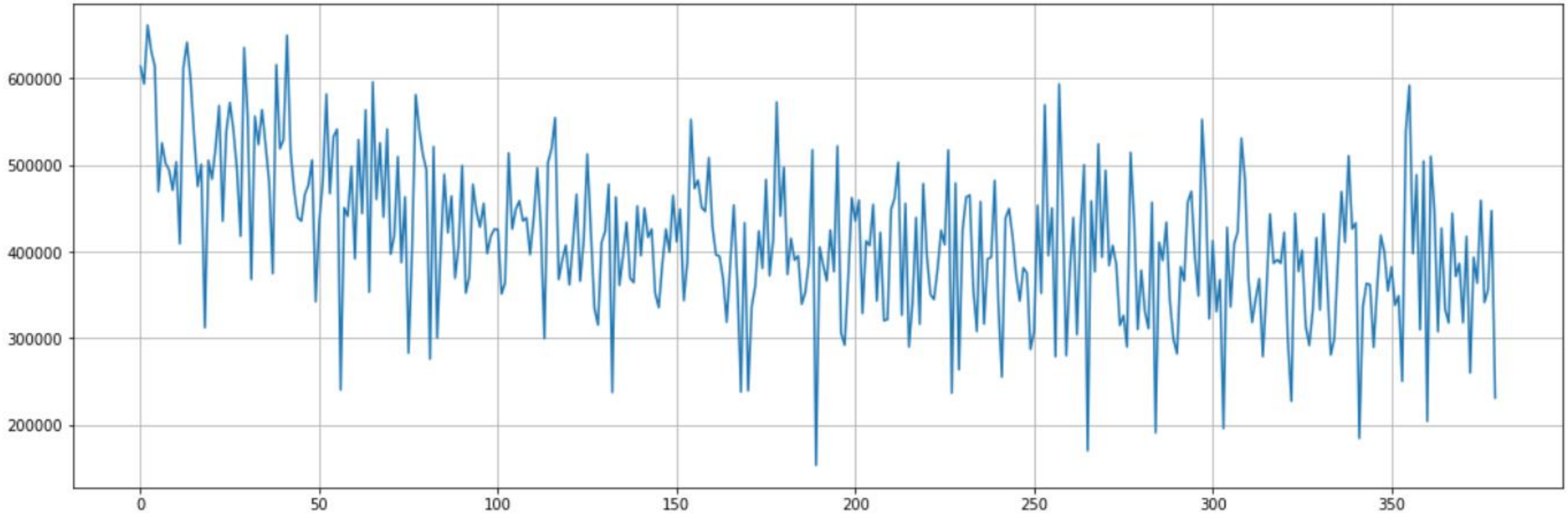


Practice

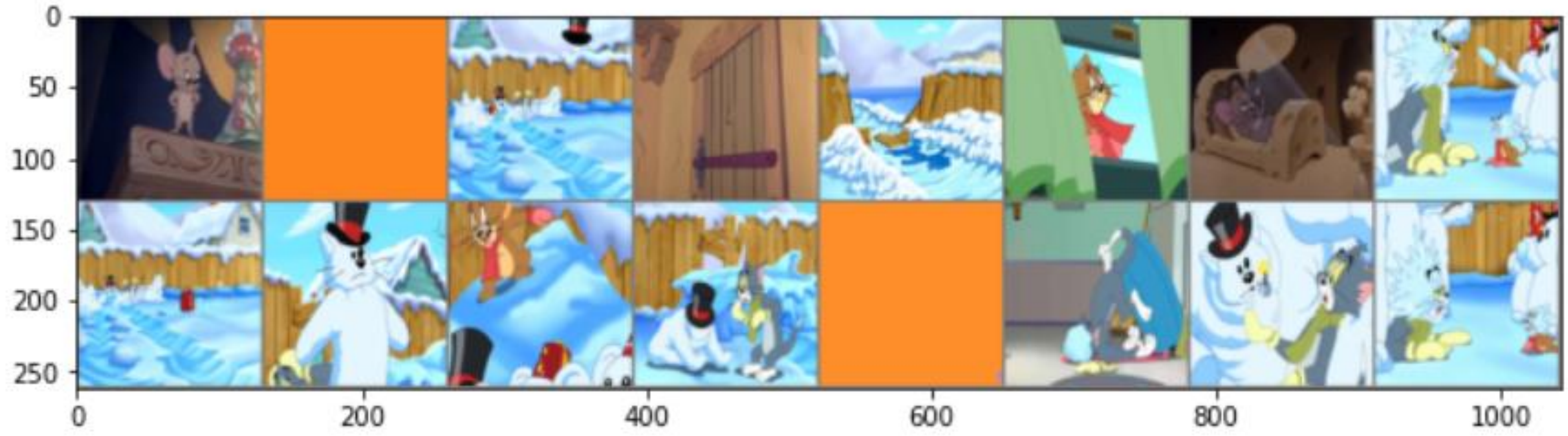
- Run “7.2.Conv_VAE.ipynb"

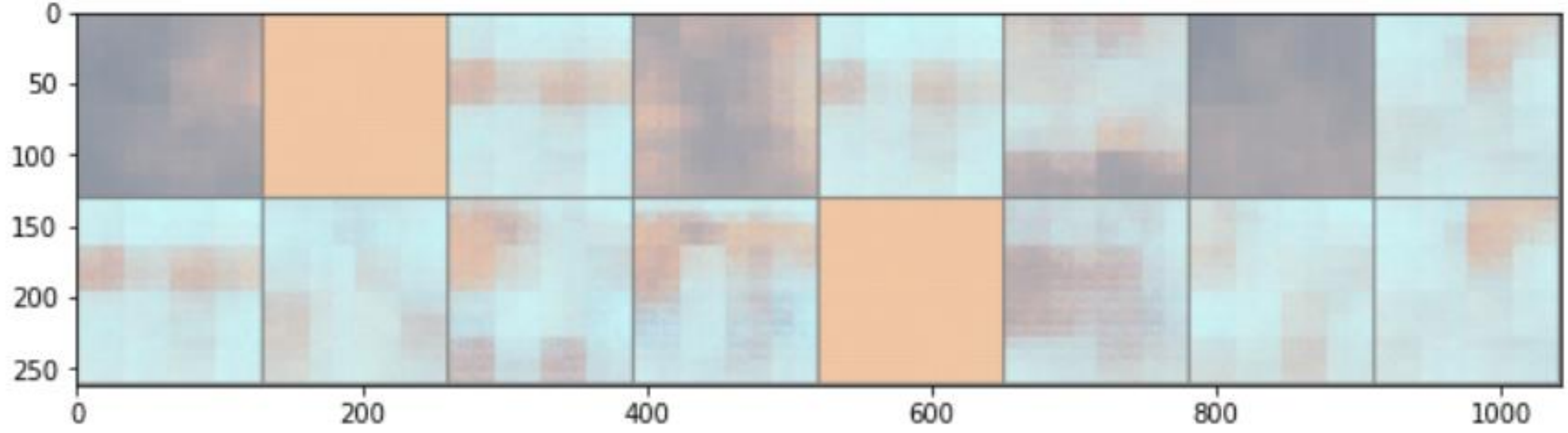
After train 20 epochs

Input size=128x128, batch size=16

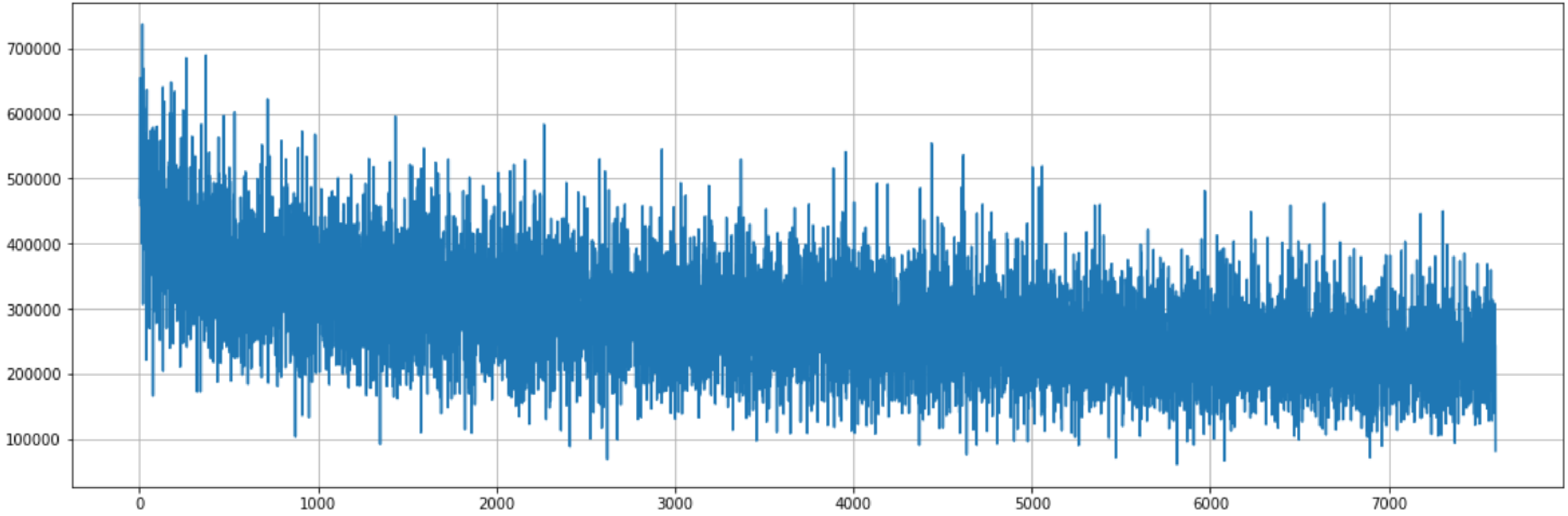


After train 20 epochs

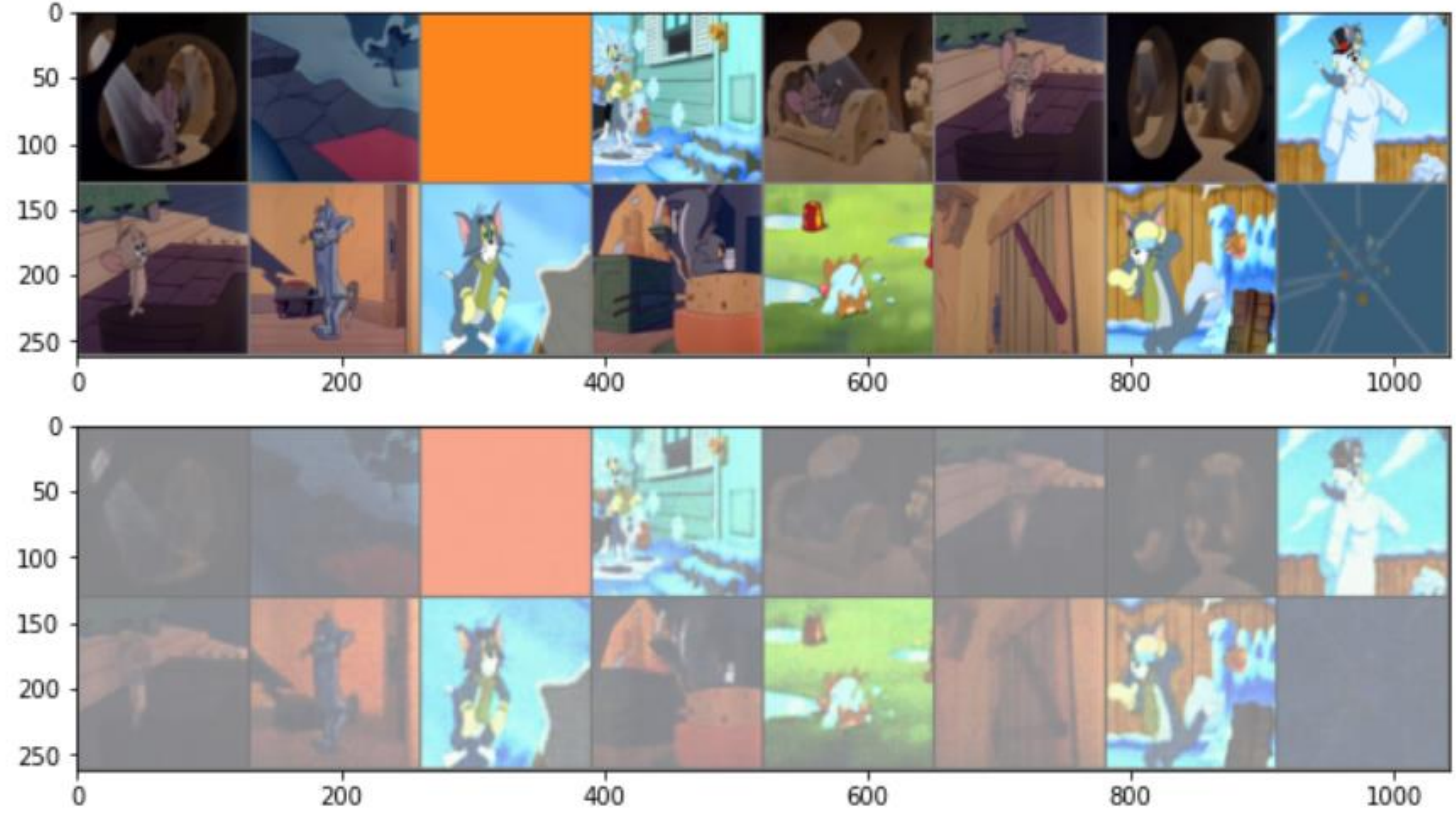




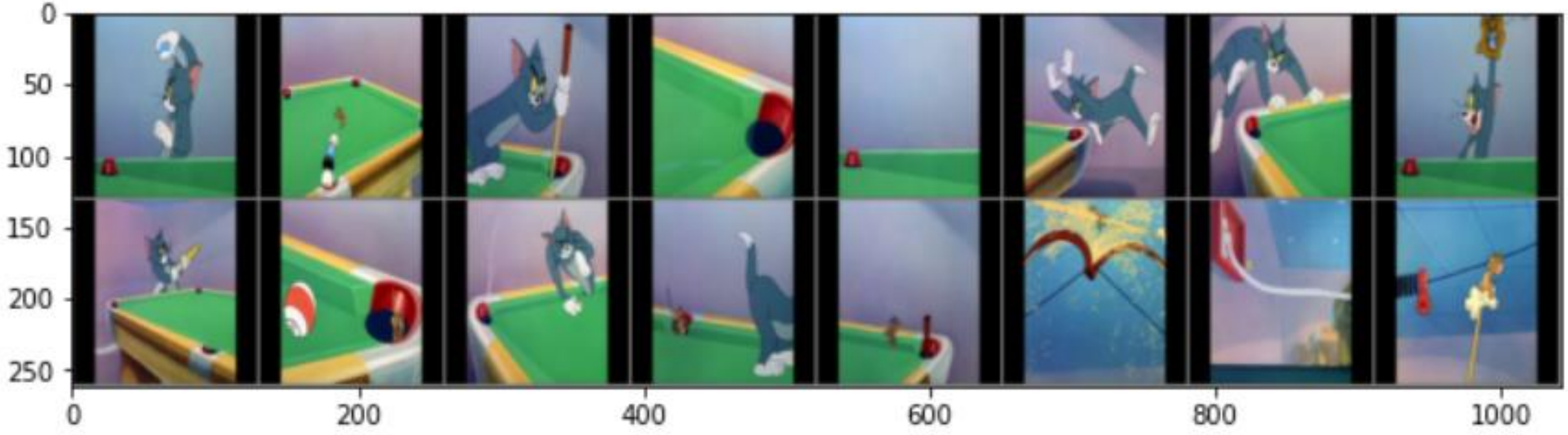
Loss plot of epoch 0-400

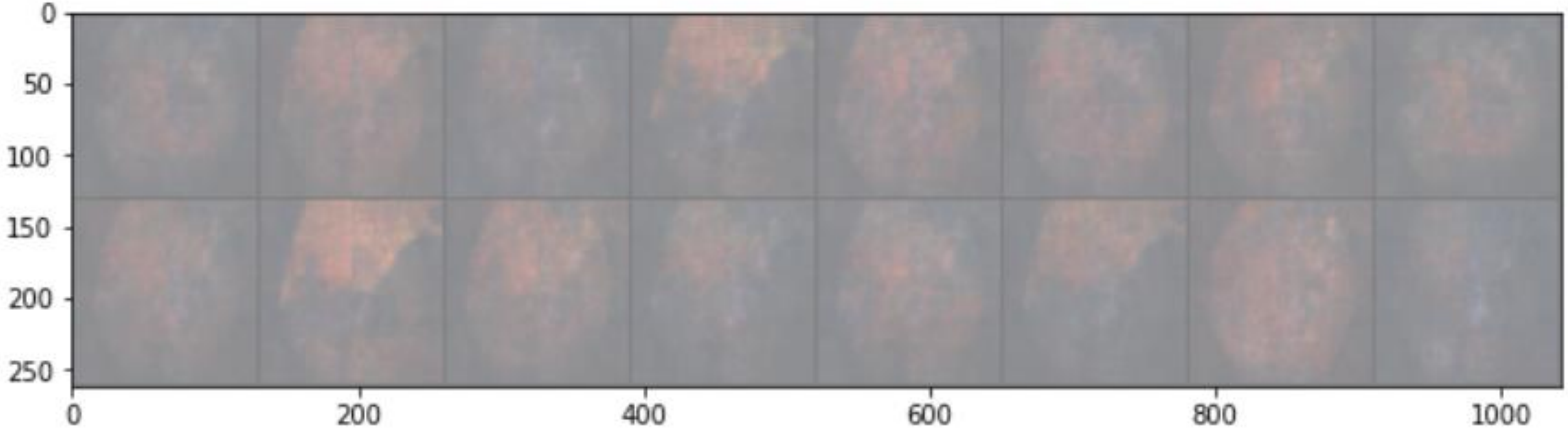


Training images recovered after training for 400 epochs

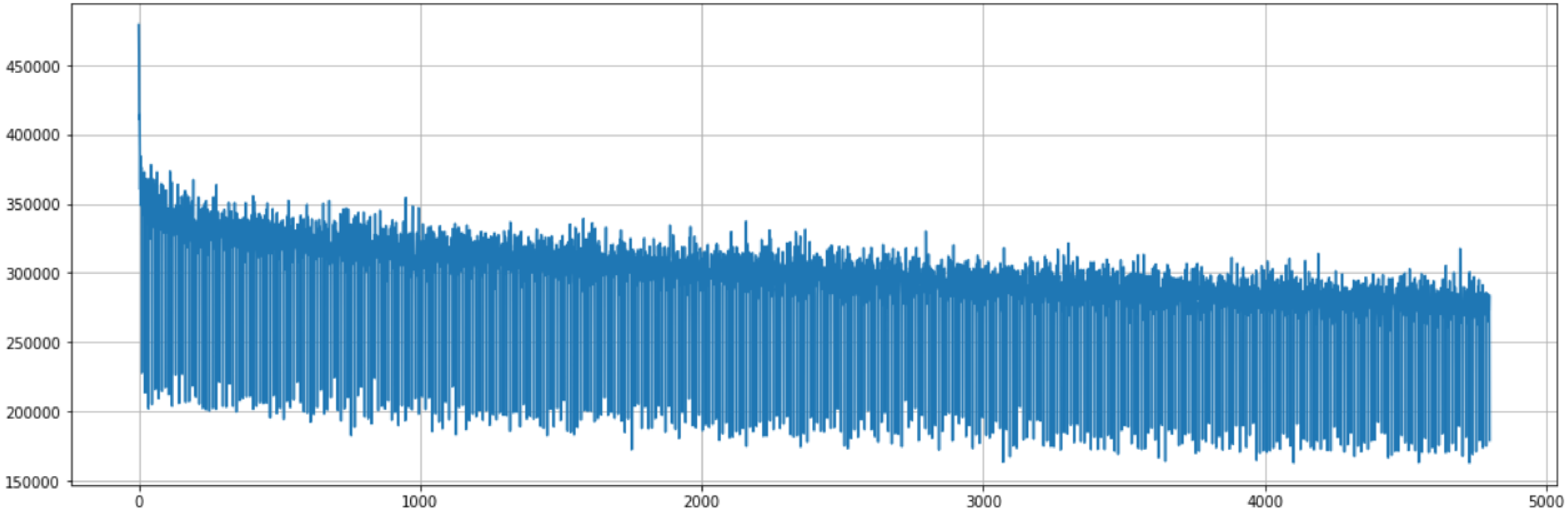


NN fails to recover un-seen test images if trained for 400 epochs

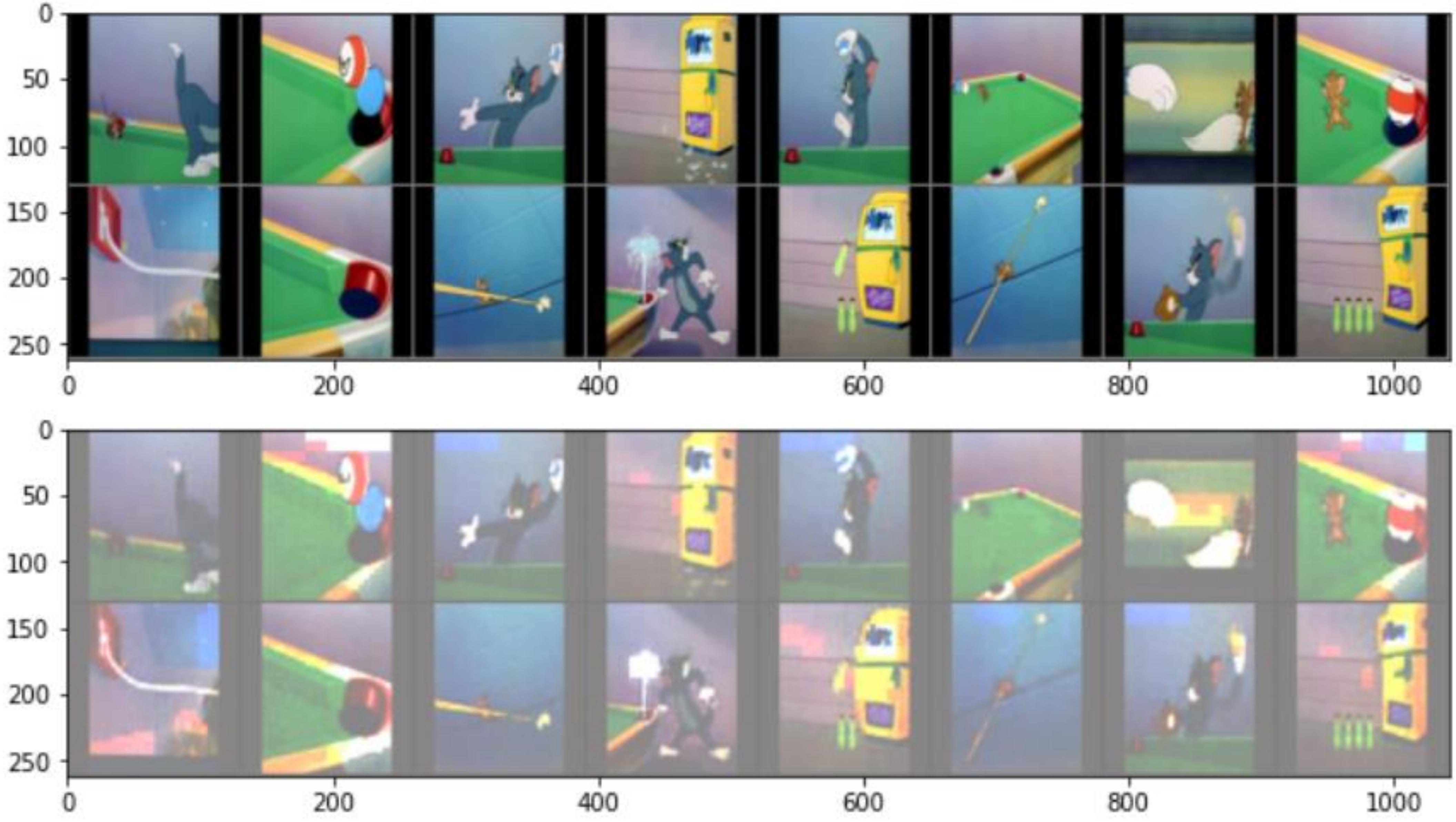




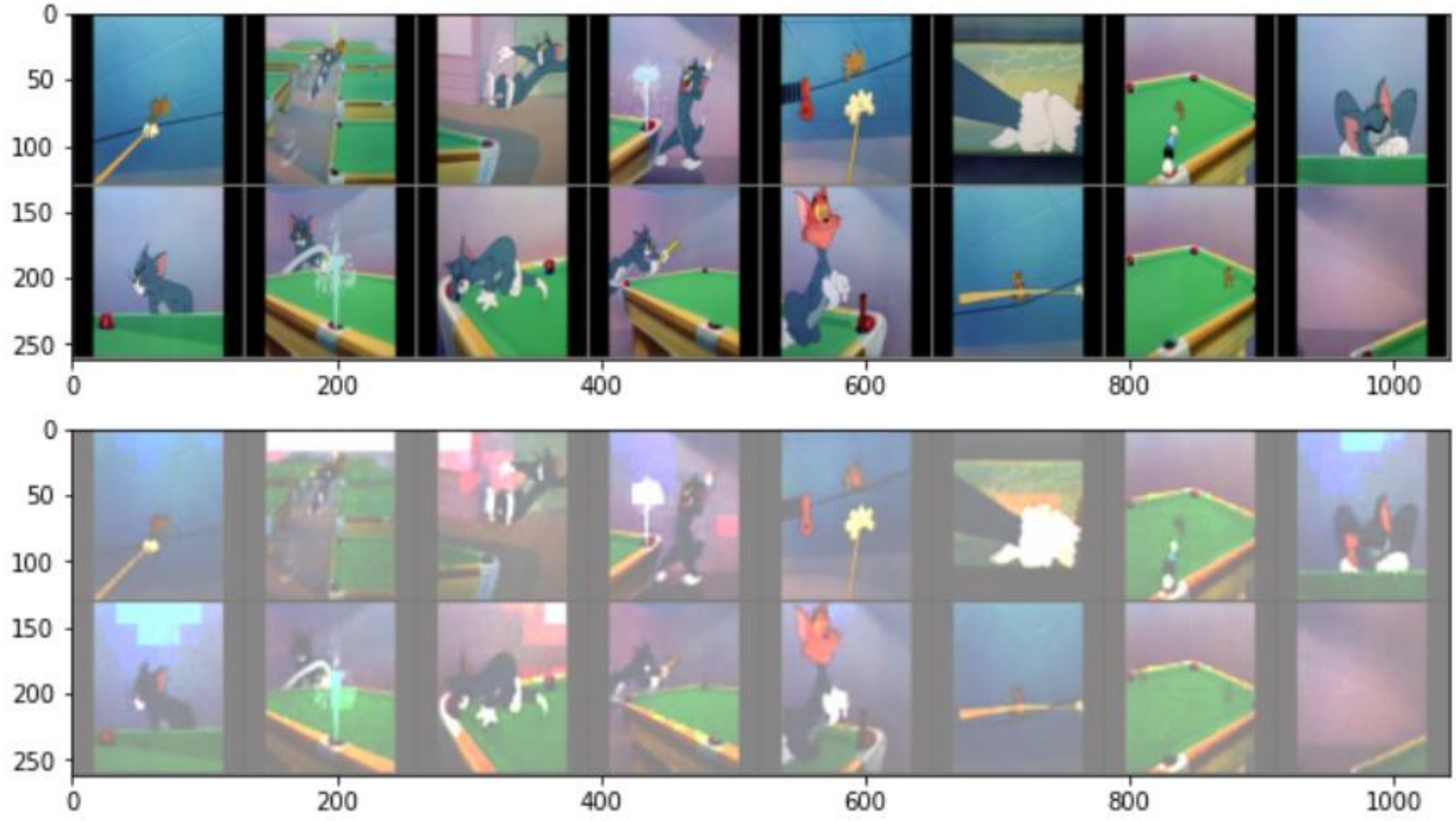
Loss plot of epoch 400-800



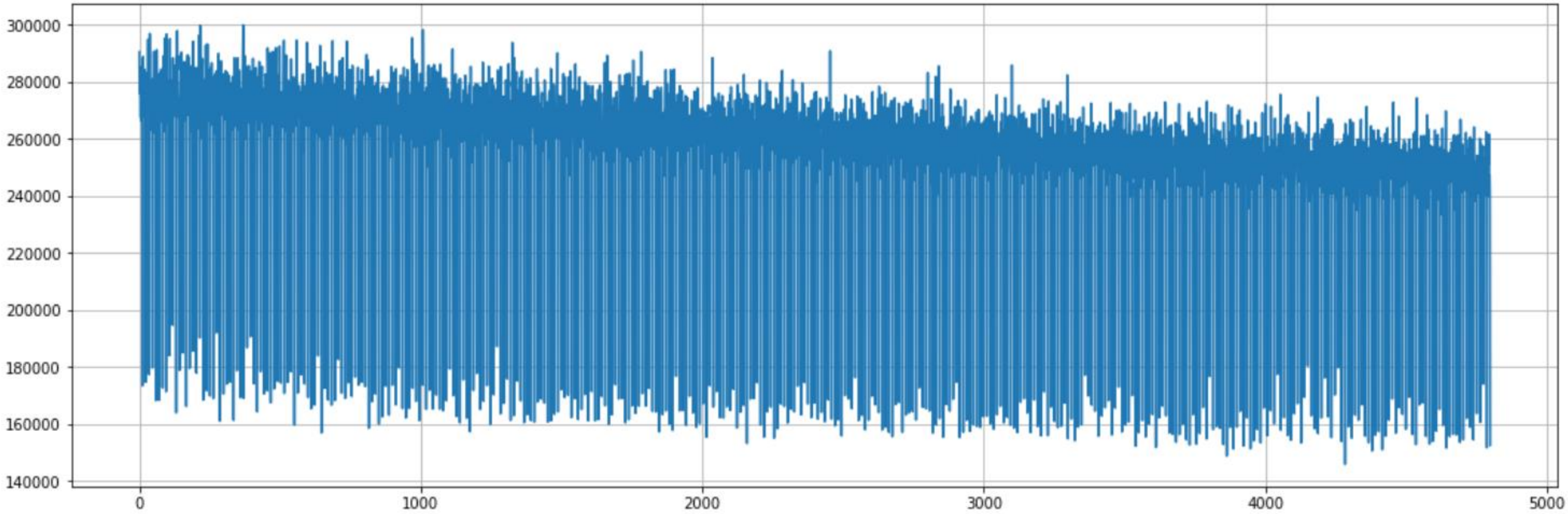
Training images recovered after training for 800 epochs



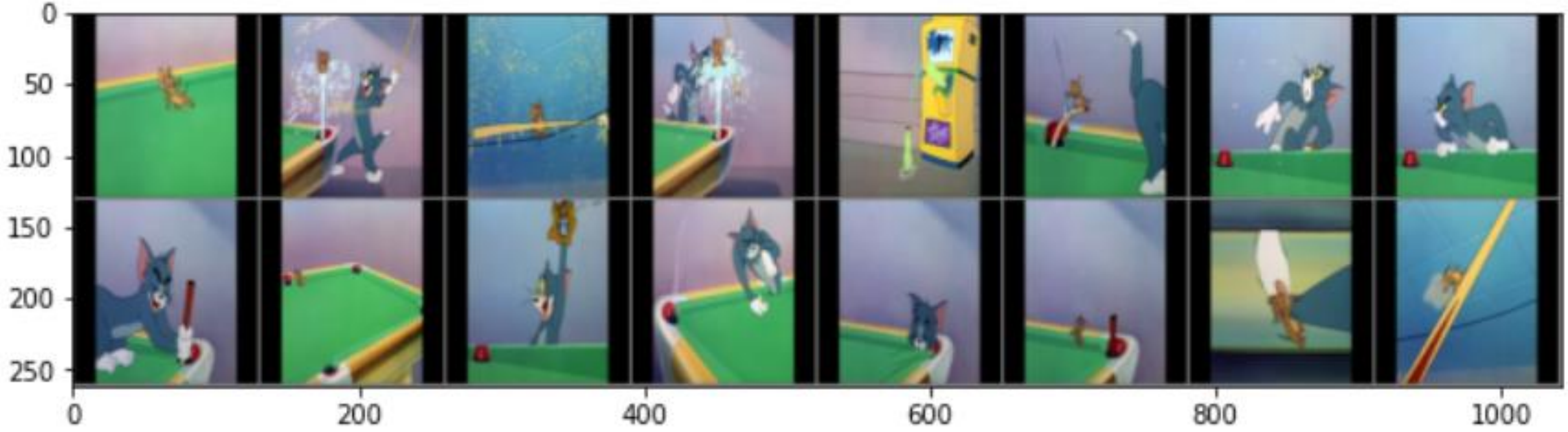
After training for 800 epochs, the NN can recover un-seen test images

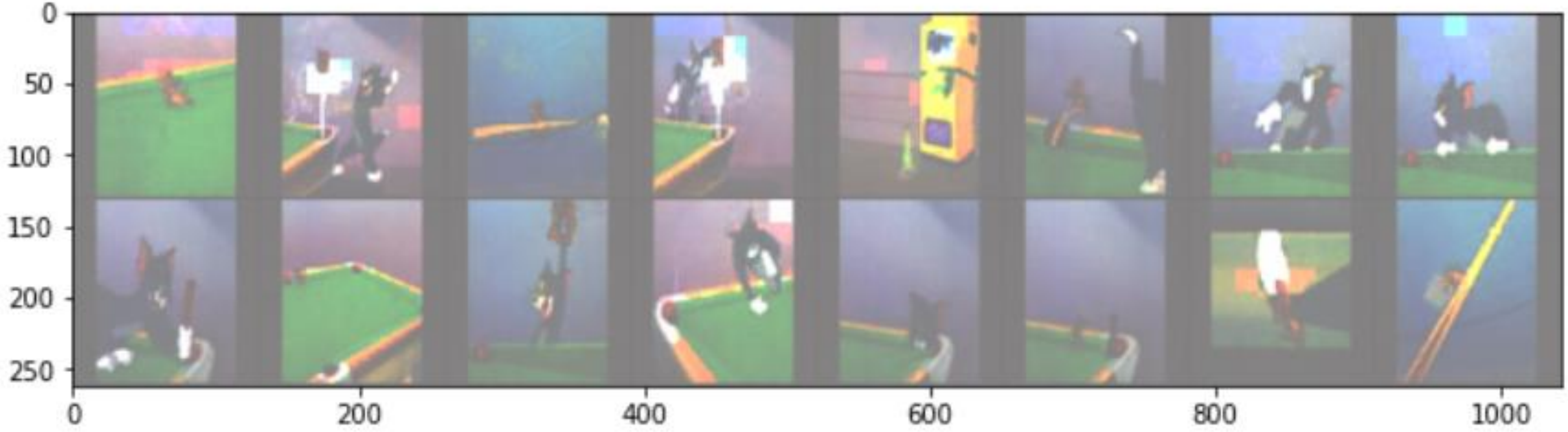


Loss plot of epoch 800-1200

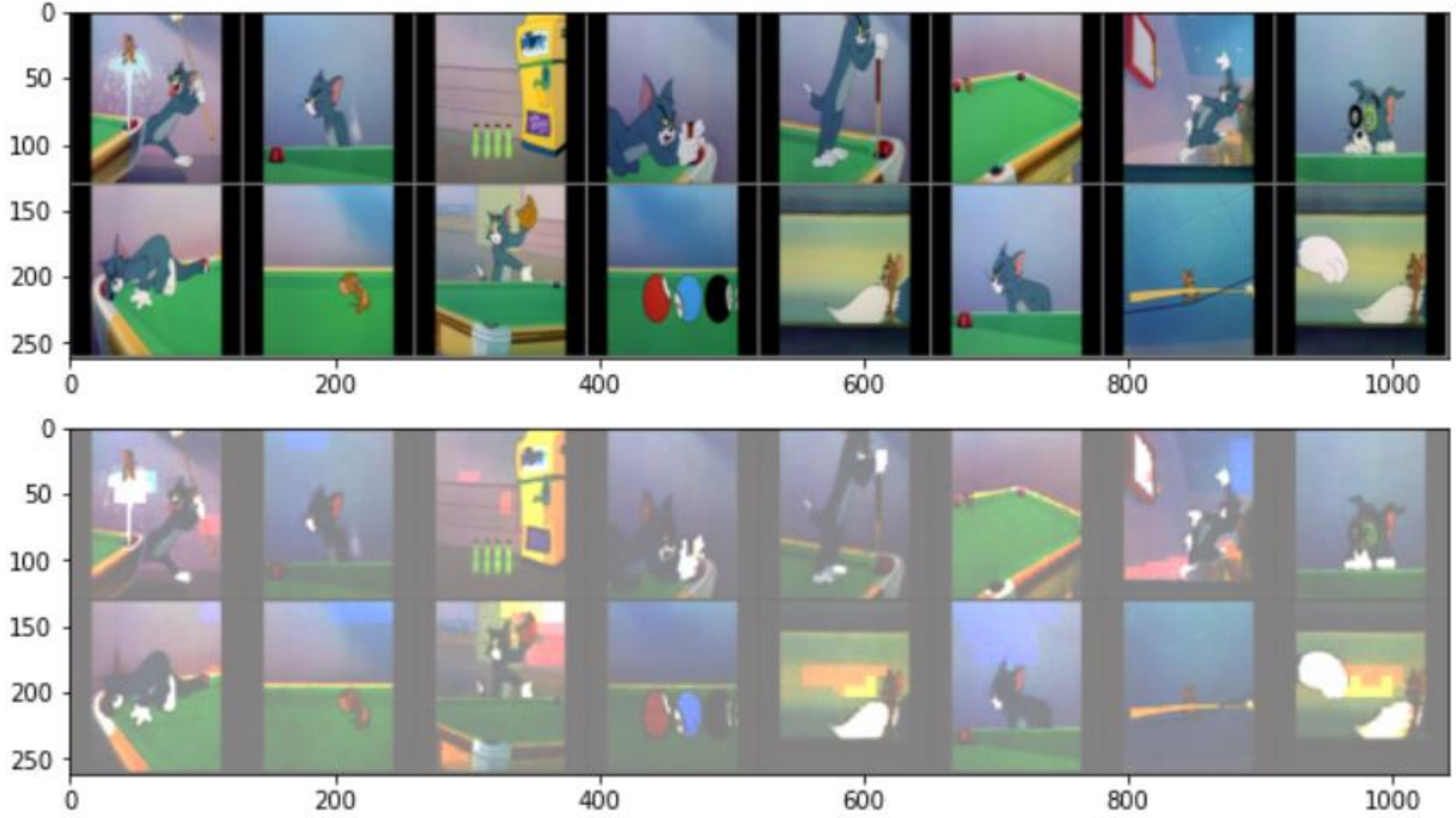


Training images recovered after training for 1200 epochs

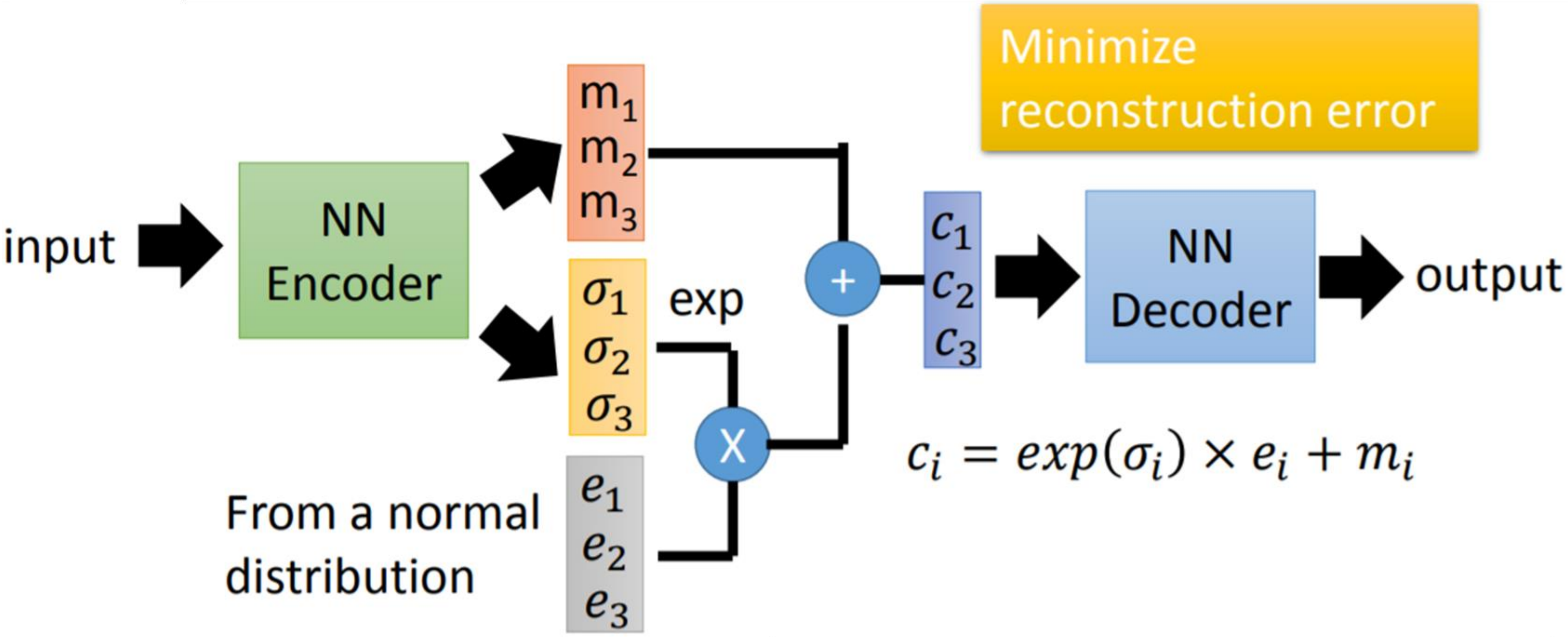




After training for 1200 epochs, the NN can recover un-seen test images



VAE

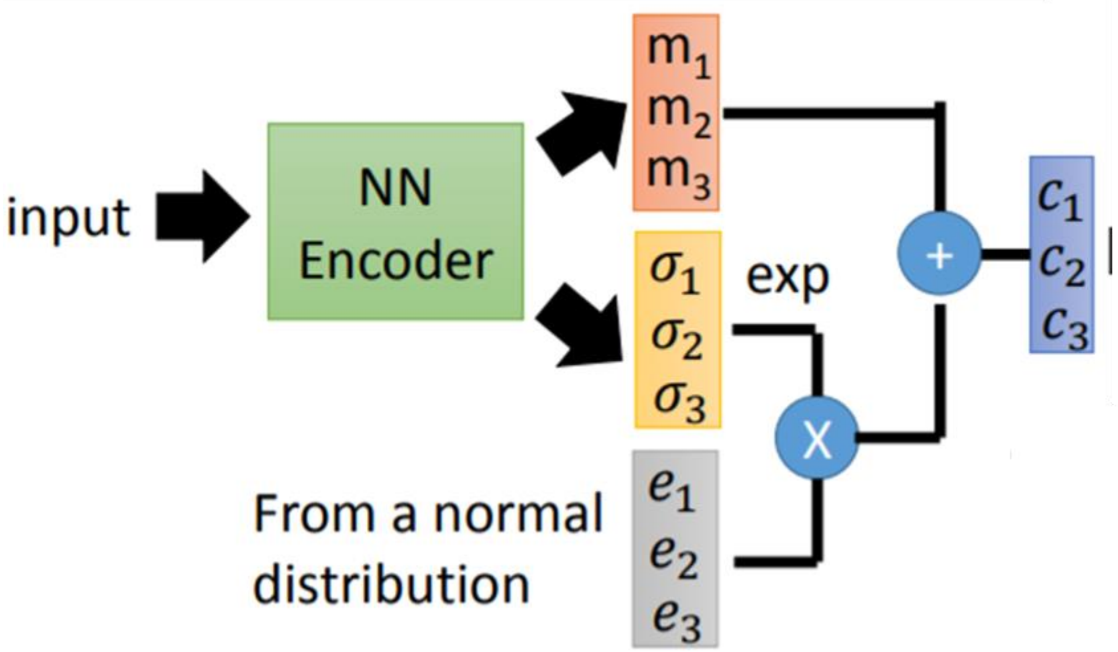


Reference: 李弘毅 ML Lecture 17 https://youtu.be/YNUek8ioAJk

Encoder

```
[15]: for batchX, _ in loader:
          break;
      print(batchX.shape)

      torch.Size([16, 3, 128, 128])
```

```
(fc1): Linear(in_features=1024, out_features=64,
(fc2): Linear(in_features=1024, out_features=64,
(fc3): Linear(in_features=64, out_features=1024,
```




```
16]: h = model.encoder(batchX.to(device))
     print(h.shape)

     torch.Size([16, 1024])
```

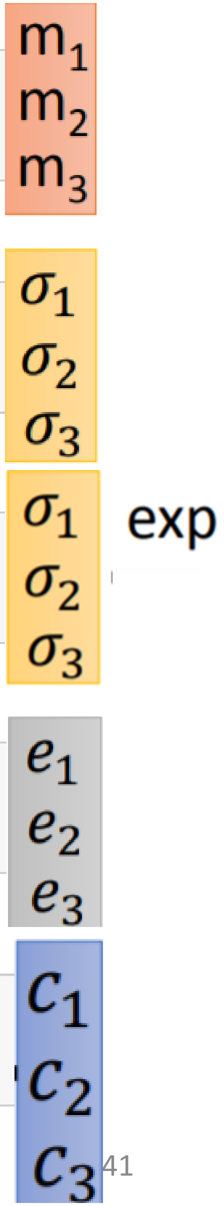
```
[17]: mu=model.fc1(h)
      print(mu.shape)

      torch.Size([16, 64])
```

```
[18]: logvar=model.fc2(h)
      print(logvar.shape)

      torch.Size([16, 64])
```

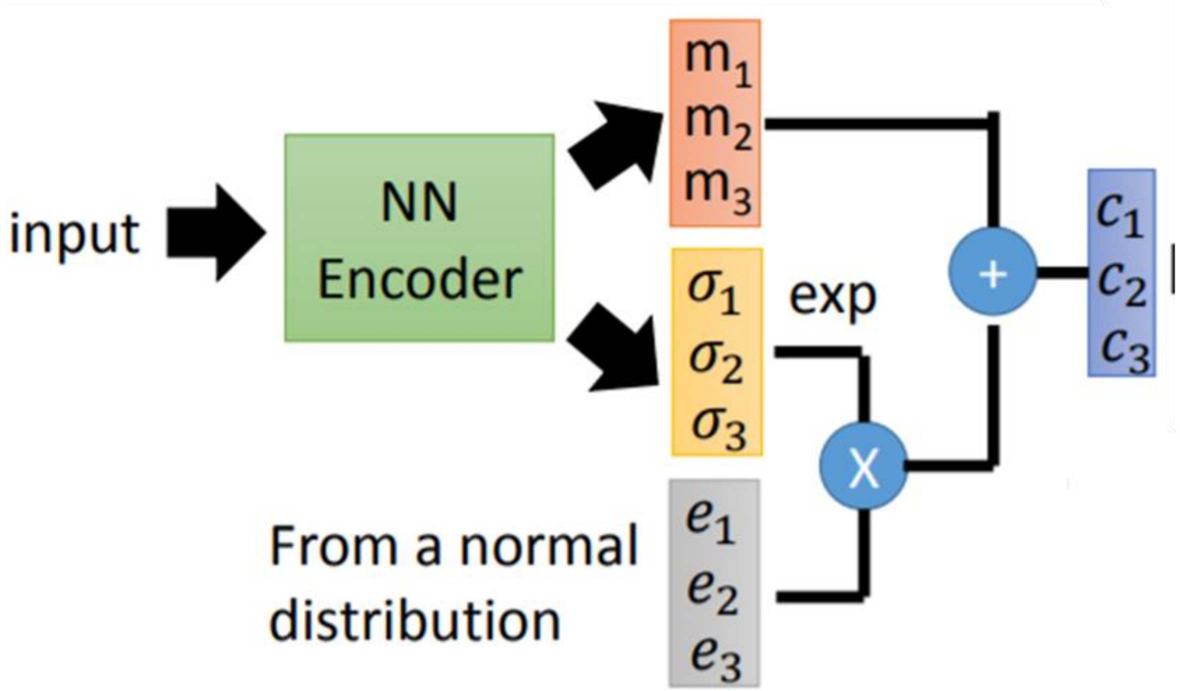
```
[19]: std = logvar.mul(0.5).exp_()
      print(std.shape)

      torch.Size([16, 64])
```

```
[20]: esp=torch.randn(*mu.size())
      print(esp.shape)

      torch.Size([16, 64])
```

```
[21]: z=mu+std*esp.to(device)
      print(z.shape)

      torch.Size([16, 64])
```



Loss function



We want σ_i close to 0
(variance close to 1)

Minimize

$$\sum_{i=1}^{3}\left(exp(\sigma_i) - (1+\sigma_i) + (m_i)^2\right)$$

L2 regularization

Loss function

[9]:
```
def loss_fn(recon_x, x, mu, logvar):
    #BCE = F.binary_cross_entropy(recon_x, x, size_average=False).to(device)
    MSE = F.mse_loss(recon_x, x, reduction='sum')
    # see Appendix B from VAE paper:
    # Kingma and Welling. Auto-Encoding Variational Bayes. ICLR, 2014
    # 0.5 * sum(1 + log(sigma^2) - mu^2 - sigma^2)
    KLD = -0.5*torch.mean(1+logvar-mu.pow(2)-logvar.exp()).to(device)
    return MSE+KLD, MSE, KLD
```

Minimize

$$\sum_{i=1}^{3}(\underline{exp(\sigma_i)} - \underline{(1+\sigma_i)} + \underline{(m_i)^2})$$

L2 regularization

[23]:
```
tensorY,mu,logvar = model(batchX.to(device))
print(tensorY.shape)
```

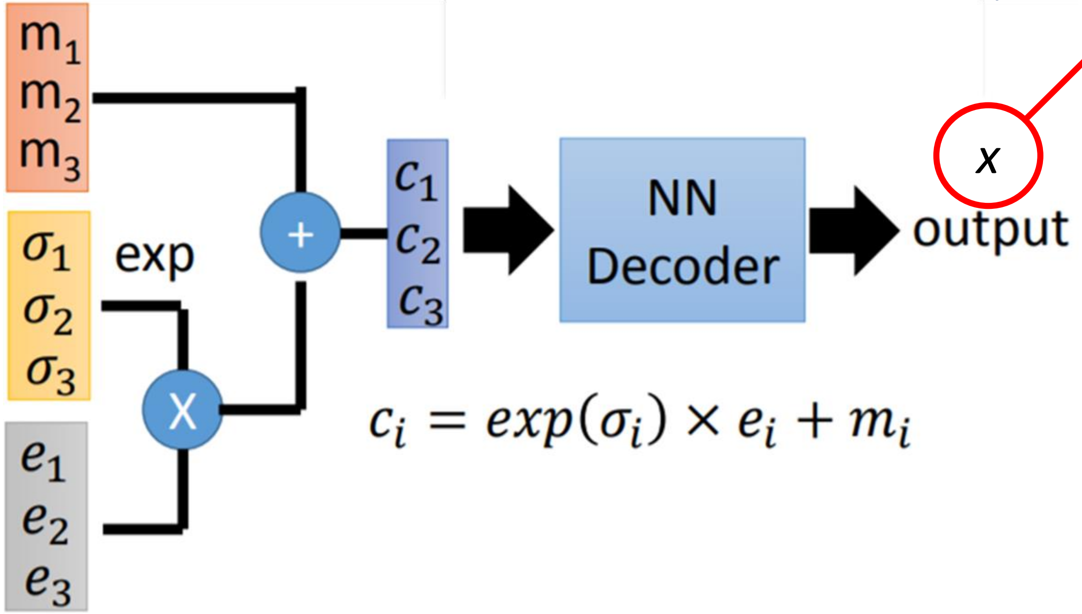
```
torch.Size([16, 3, 128, 128])
```

[24]:
```
loss, mse,kld = loss_fn(tensorY, batchX.to(device), mu, logvar)
print(loss)
```

```
tensor(627375.3750, device='cuda:0', grad_fn=<AddBackward0>)
```

The decoder part of VAE can be modelled as a Gaussian mixture model sampled from the latent vector z

If x is close to training image, then P(x) should be large.



$$c_i = exp(\sigma_i) \times e_i + m_i$$

Gaussian Mixture Model

$$P(x) = \sum_m P(m)P(x|m)$$

How to sample?

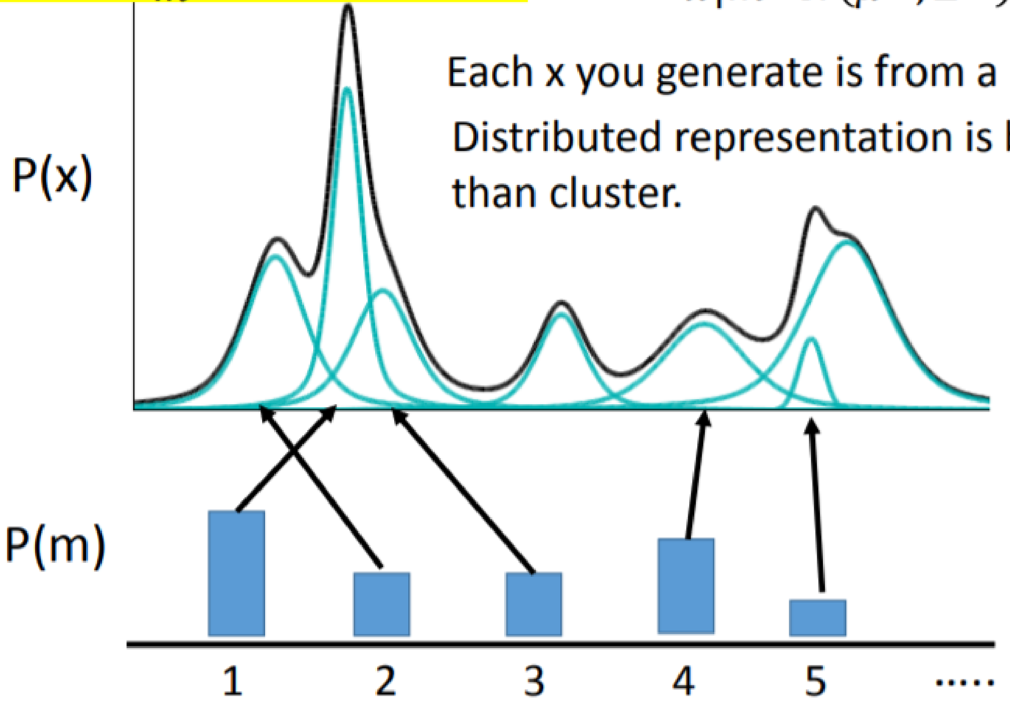
$m \sim P(m)$ (multinomial)

m is an integer

$x|m \sim N(\mu^m, \Sigma^m)$

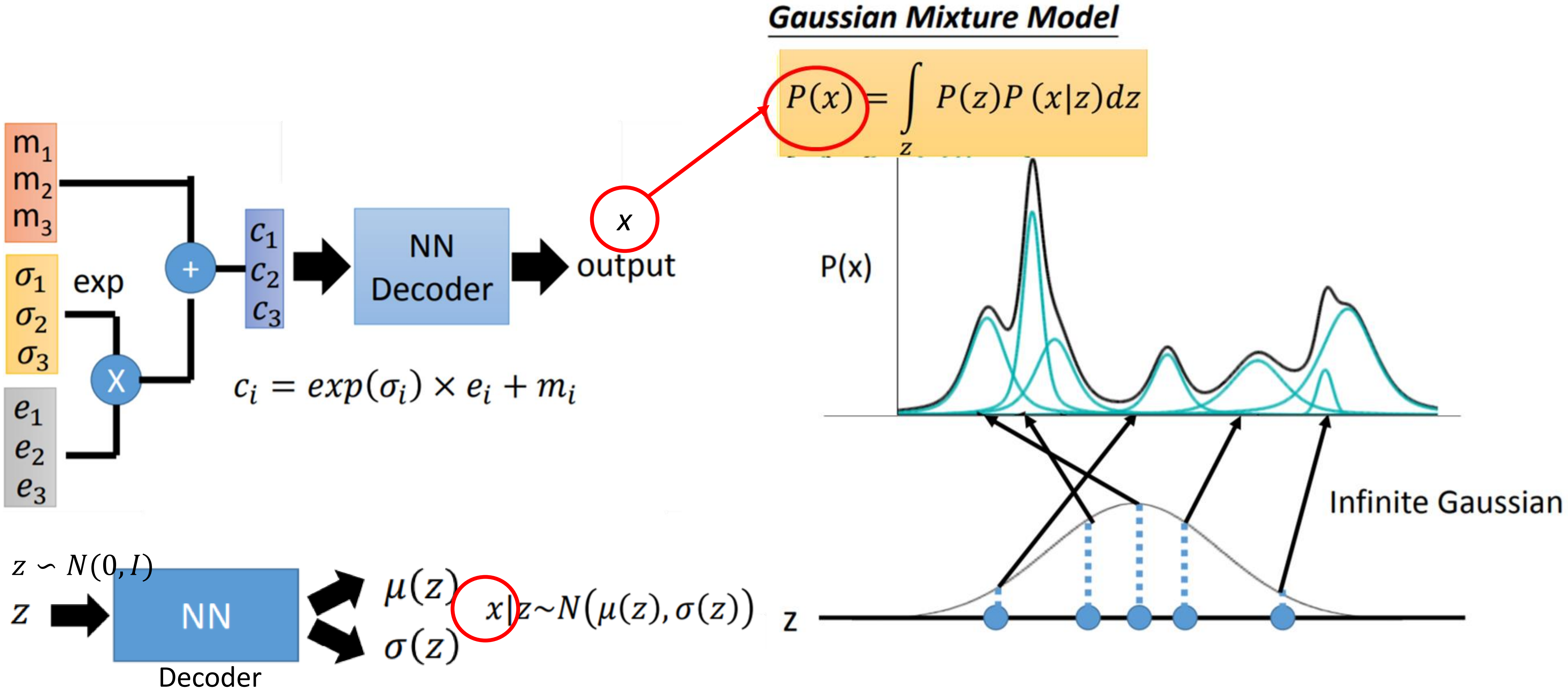
Each x you generate is from a mixture

Distributed representation is better than cluster.



Reference: 李弘毅 ML Lecture 18 https://youtu.be/8zomhgKrsmQ

The decoder of VAE can be modelled as a Gaussian mixture model sampled from the latent vector z



Reference: 李弘毅 ML Lecture 18 https://youtu.be/8zomhgKrsmQ

Given a set of training images x, we want to find $\mu(z)$ and $\sigma(z)$ that maximize $P(x)$.

Maximizing Likelihood

$$P(x) = \int_z P(z)P\,(x|z)dz$$

P(z) is normal distribution

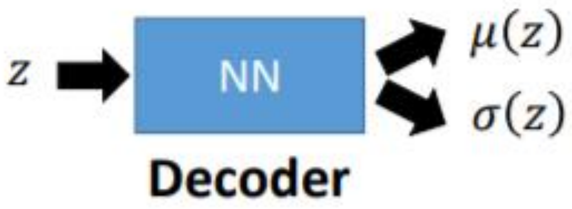
$x|z \sim N\big(\mu(z), \sigma(z)\big)$

$\mu(z), \sigma(z)$ is going to be estimated

$$L = \sum_x logP(x)$$

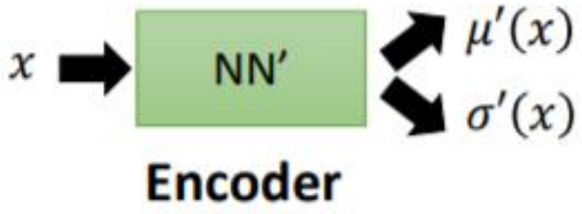
Maximizing the likelihood of the observed x

Tuning the parameters to maximize likelihood L



We need another distribution q(z|x)

$$z|x \sim N\big(\mu'(x), \sigma'(x)\big)$$



Reference: 李弘毅 ML Lecture 18 https://youtu.be/8zomhgKrsmQ

Recap: Use maximum likelihood to derive loss function for logistic regression

Training Data	x^1	x^2	x^3	……	x^N
	C_1	C_1	C_2		C_1

max $\quad L(w,b) = f_{w,b}(x^1) f_{w,b}(x^2) \left(1 - f_{w,b}(x^3)\right) \cdots f_{w,b}(x^N)$

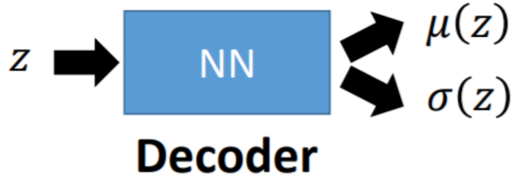
min $\quad -lnL(w,b) = -lnf_{w,b}(x^1) - lnf_{w,b}(x^2) - ln\left(1 - f_{w,b}(x^3)\right) \cdots$

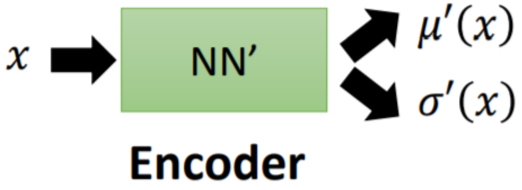
$\hat{y}^n$: 1 for class 1, 0 for class 2

$$= \sum_n -\underbrace{\left[\hat{y}^n lnf_{w,b}(x^n) + (1-\hat{y}^n) ln\left(1 - f_{w,b}(x^n)\right)\right]}_{\text{Cross entropy between two Bernoulli distribution}}$$

Reference: 李弘毅 ML Lecture 5 https://youtu.be/hSXFuypLukA

Rewrite $\log P(x)$ as KL divergence of P($z|x$) and $q(z|x)$





$$logP(x) = \int_z q(z|x)logP(x)dz$$

q(z|x) can be any distribution

$$= \int_z q(z|x)log\left(\frac{P(z,x)}{P(z|x)}\right)dz = \int_z q(z|x)log\left(\frac{P(z,x)}{q(z|x)}\frac{q(z|x)}{P(z|x)}\right)dz$$

$$= \int_z q(z|x)log\left(\frac{P(z,x)}{q(z|x)}\right)dz + \underbrace{\int_z q(z|x)log\left(\frac{q(z|x)}{P(z|x)}\right)dz}_{KL(q(z|x)||P(z|x))}$$

≥ 0

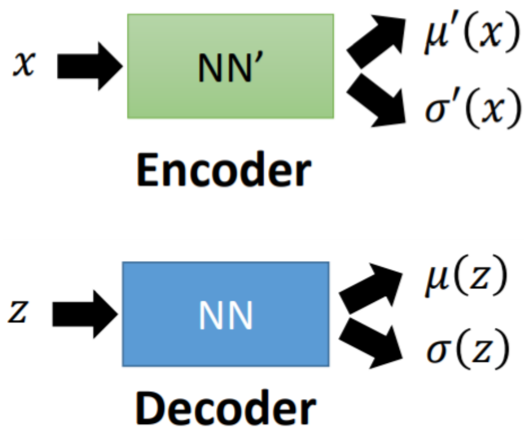
$$\geq \int_z q(z|x)log\left(\frac{P(x|z)P(z)}{q(z|x)}\right)dz$$

lower bound L_b

$$D_{KL}(q||p) = \sum_{i=1}^{N} q(x_i)\log(\frac{q(x_i)}{p(x_i)})$$

Reference: 李弘毅 ML Lecture 18 https://youtu.be/8zomhgKrsmQ

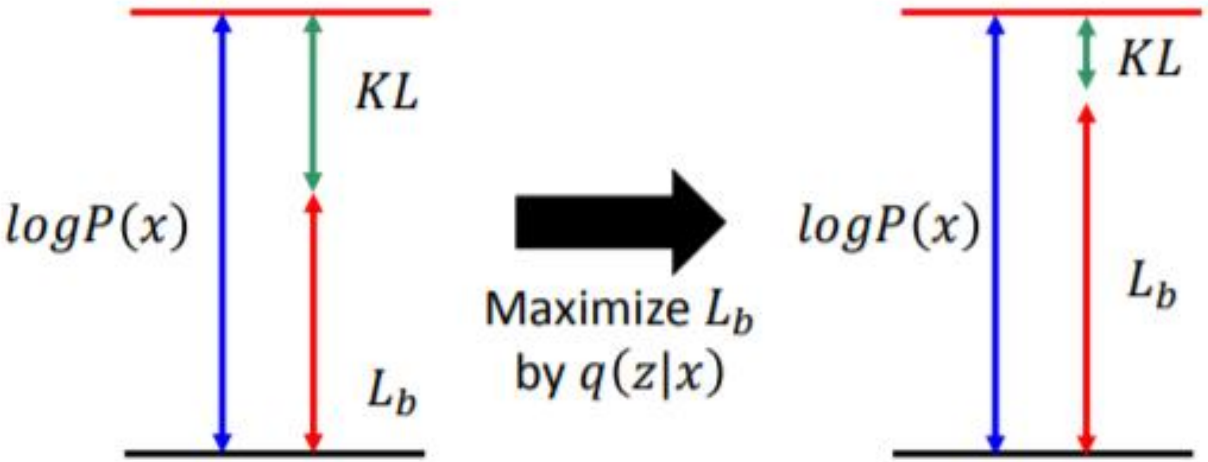
If $P(x|z)$ and $q(z|x)$ can maximum L_b, then we can maximize $P(x)$ by maximizing L_b



$$logP(x) = L_b + KL\big(q(z|x)||P(z|x)\big)$$

$$L_b = \int_z q(z|x)log\left(\frac{P(x|z)P(z)}{q(z|x)}\right)dz$$

Find $P(x|z)$ and $q(z|x)$ maximizing L_b



$q(z|x)$ will be an approximation of $p(z|x)$ in the end

Rewrite the lower bound L_b as $KL(q(z|x)||P(z))$

$$L_b = \int_z q(z|x) \log\left(\frac{P(z,x)}{q(z|x)}\right) dz = \int_z q(z|x) \log\left(\frac{P(x|z)P(z)}{q(z|x)}\right) dz$$

$$= \underbrace{\int_z \boxed{q(z|x)} \log\left(\frac{P(z)}{\boxed{q(z|x)}}\right) dz}_{-KL(q(z|x)||P(z))} + \int_z q(z|x) \log P(x|z) dz$$

$$z|x \sim N(\mu'(x), \sigma'(x))$$

How the loss function is derived

Minimizing $KL(q(z|x)||P(z))$

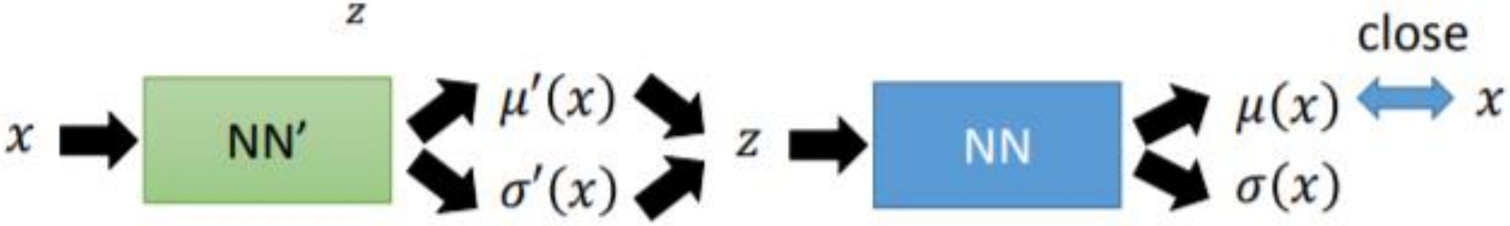
Minimize

$$\sum_{i=1}^{3}(exp(\sigma_i) - (1 + \sigma_i) + (m_i)^2)$$

(Refer to the Appendix B of the original VAE paper)

Maximizing

$$\int_z q(z|x)logP(x|z)dz = E_{q(z|x)}[logP(x|z)]$$



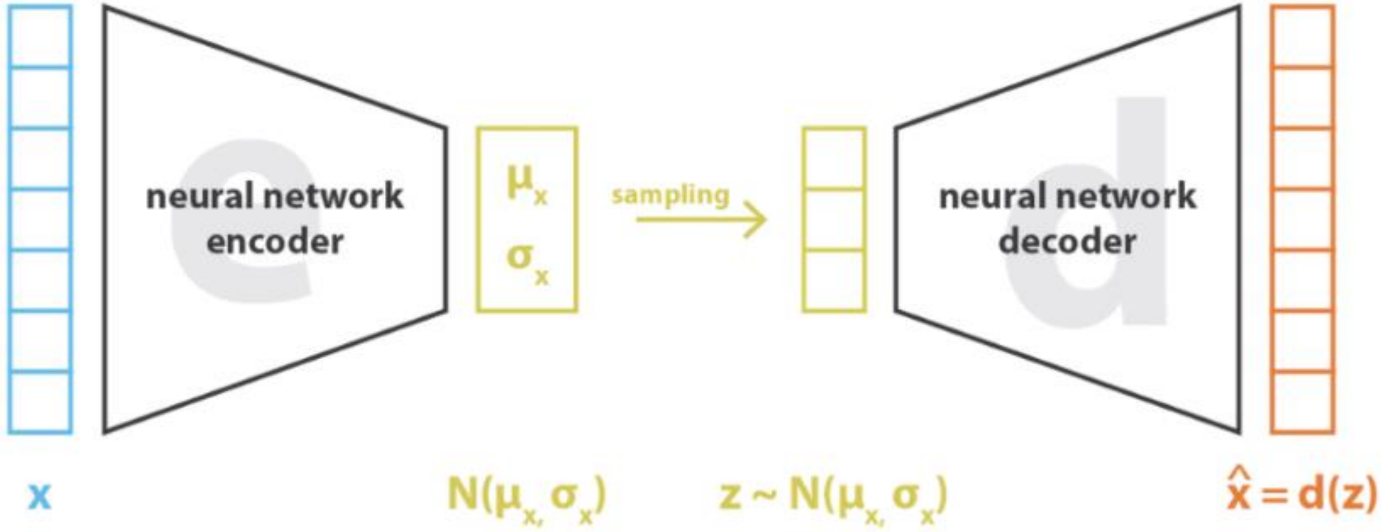
q(z|x)

P(x|z)

If we consider mean only

Loss function

Source: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73



$$\text{loss} = \| x - \hat{x} \|^2 + \mathrm{KL}[\, N(\mu_x, \sigma_x), N(0, I)\,] = \| x - d(z) \|^2 + \mathrm{KL}[\, N(\mu_x, \sigma_x), N(0, I)\,]$$

$$D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i) log(\frac{p(x_i)}{q(x_i)})$$

Minimize

$$\boxed{\sum_{i=1}^{3} (exp(\sigma_i) - (1 + \sigma_i) + (m_i)^2)}$$

HW5 (2)

- Train an VAE to learn a compact representation (try latent vector of size 20, 30, 50) of your facial expression. Test with 10 happy and 10 angry faces.
- Show the recovered image.
- Send the latent vectors to *t*-SNE or PCA to see whether they form clusters.