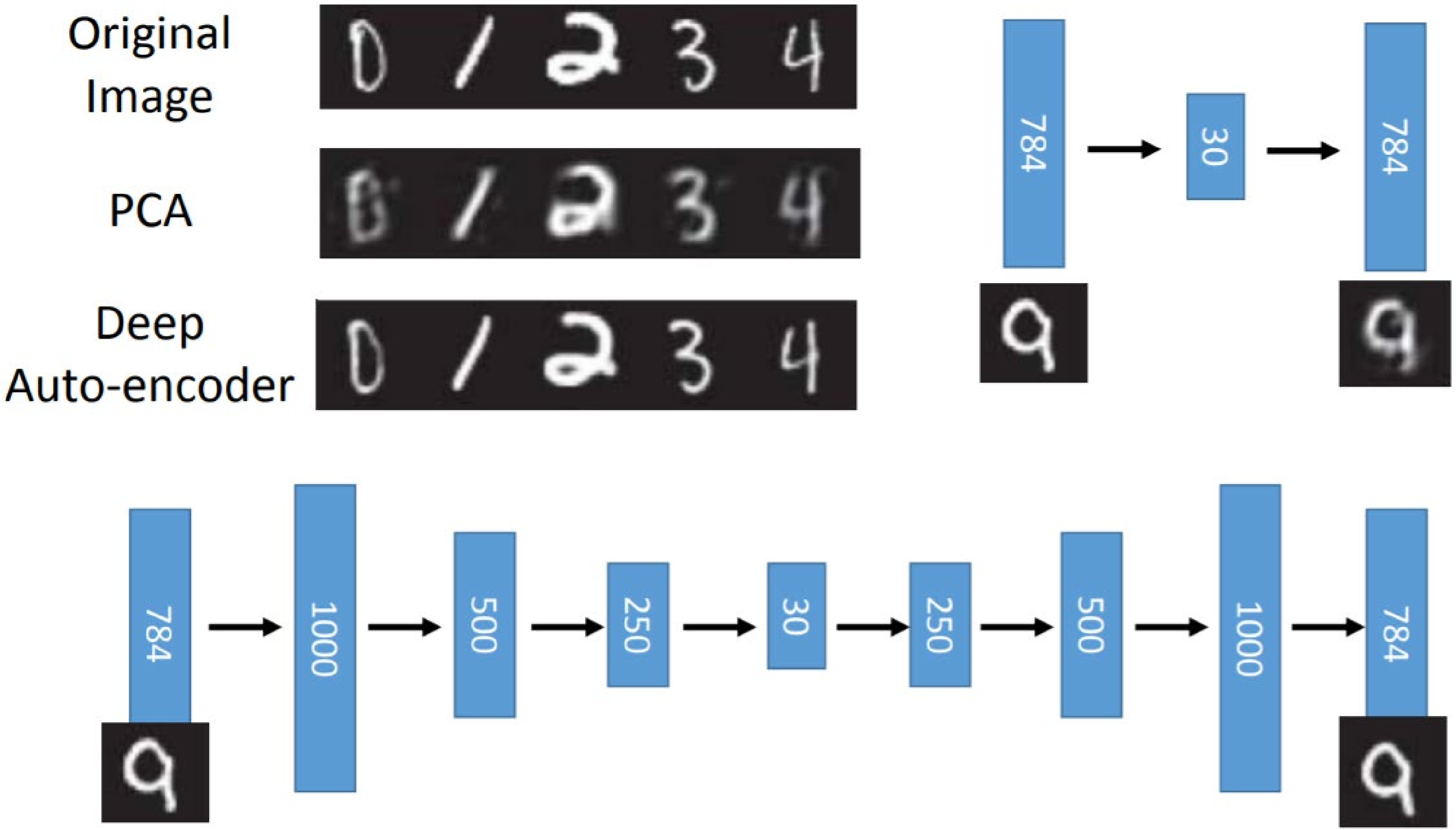
Auto-encoder

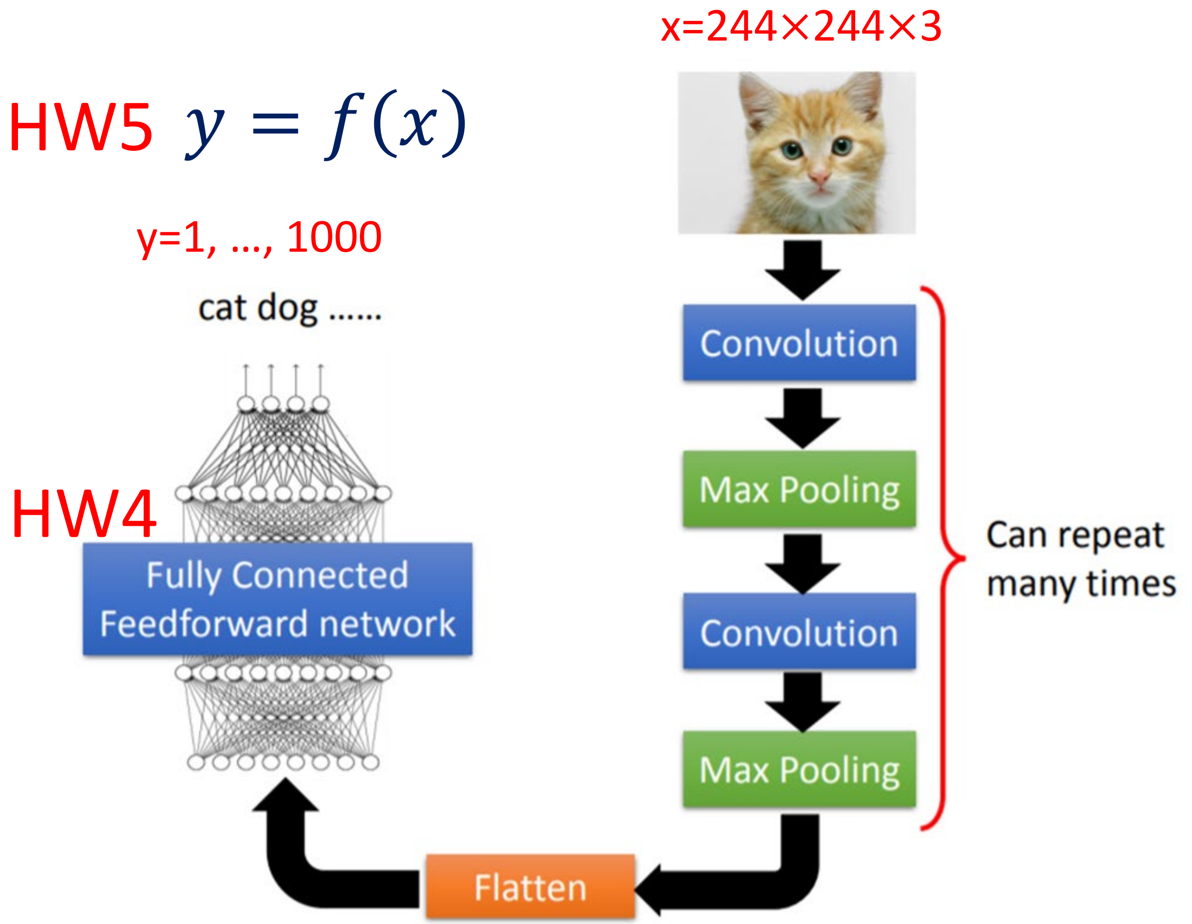
MLP based autoencoder

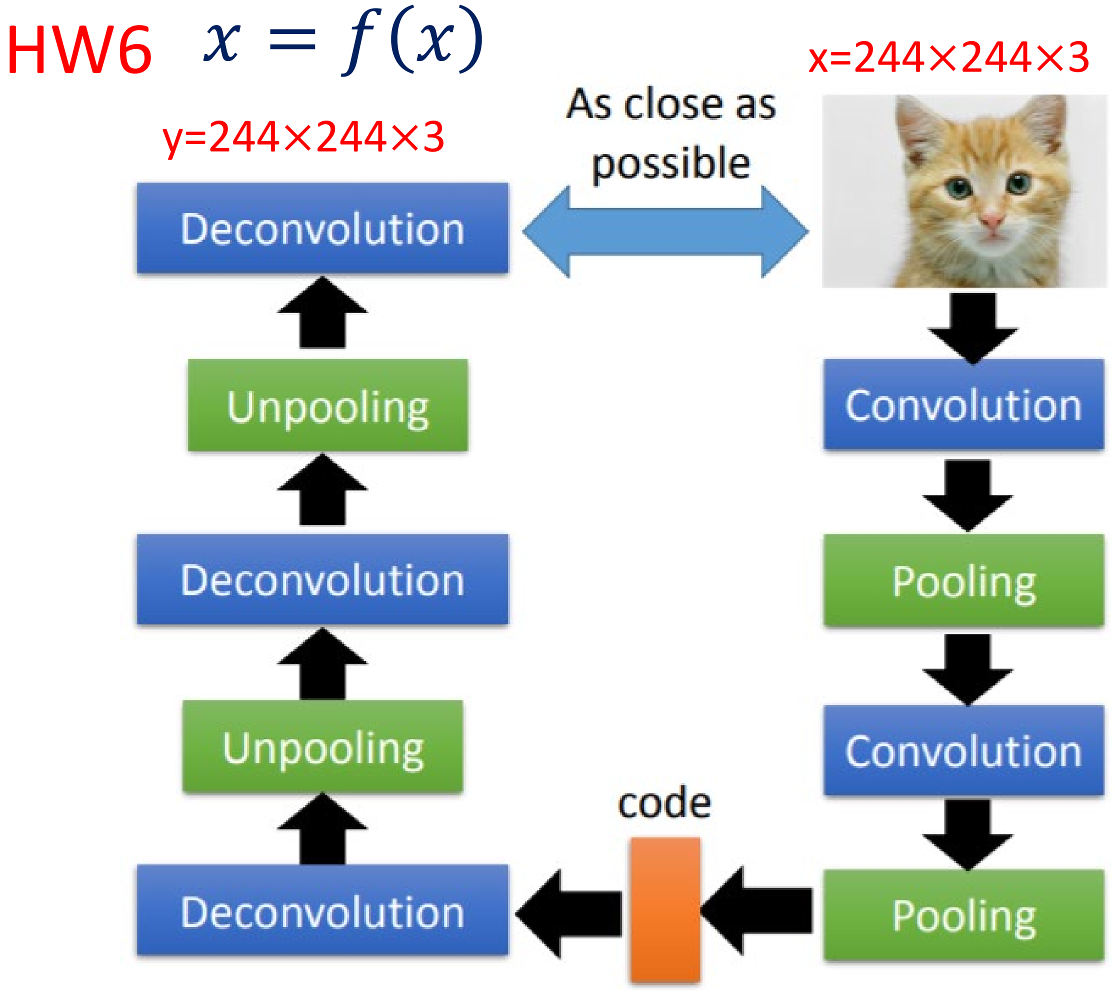


Reference: Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." *Science* 313.5786 (2006): 504-507

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

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



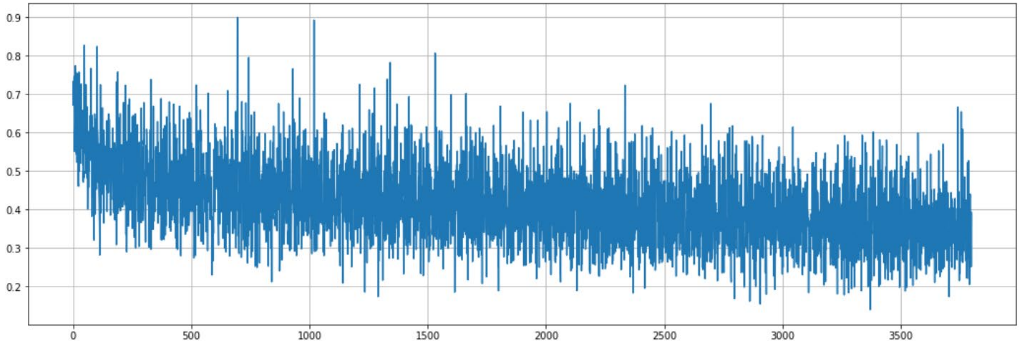


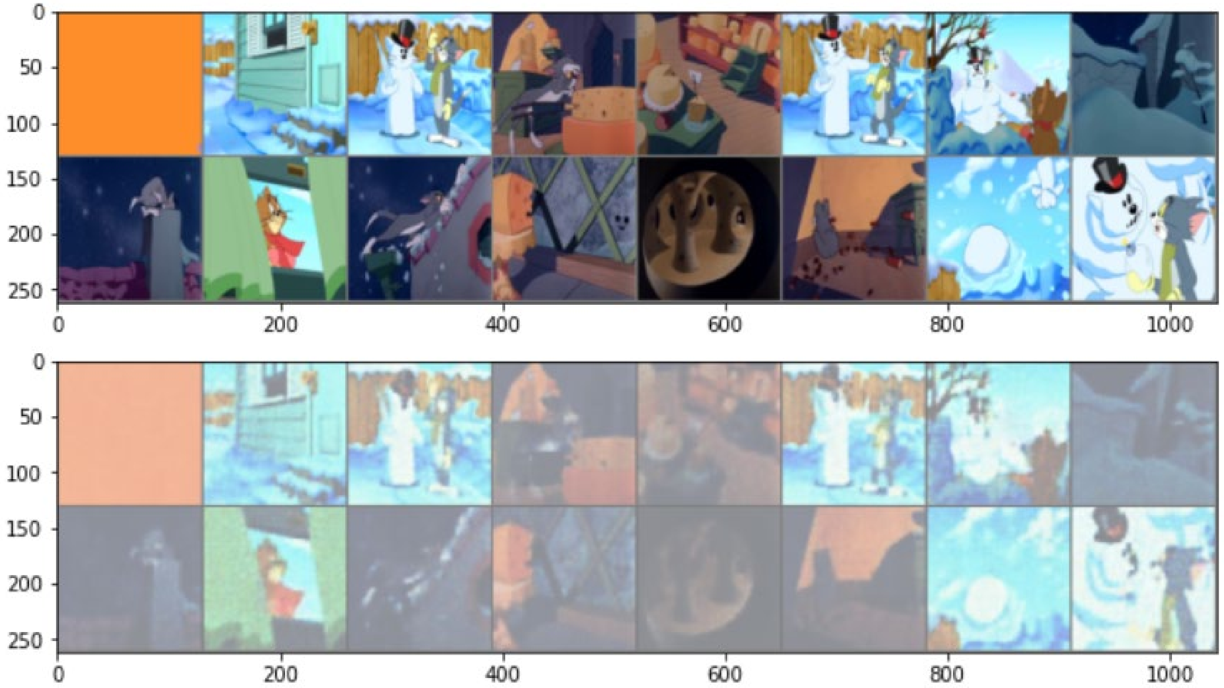
Reference: 李弘毅 ML Lecture 16 https://youtu.be/Tk5B4seA-AU

Practice

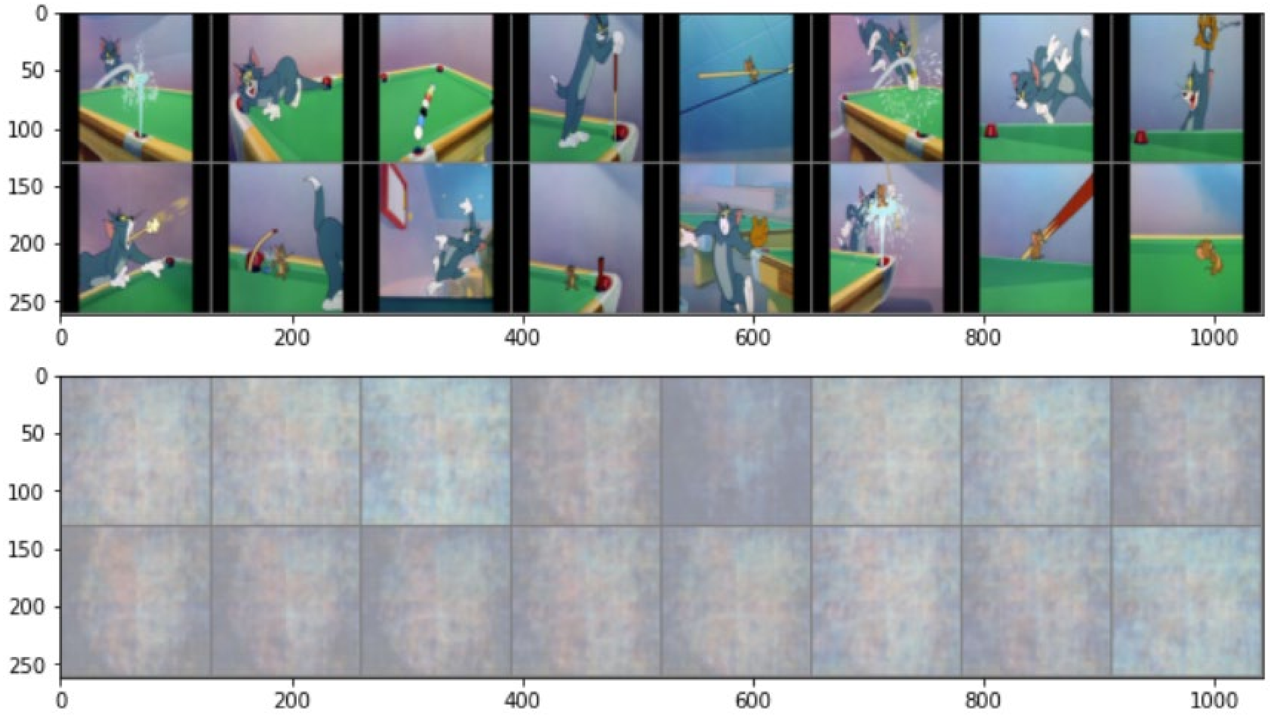
- Run “7.1.Conv_AE.ipynb"

Train 200 epochs



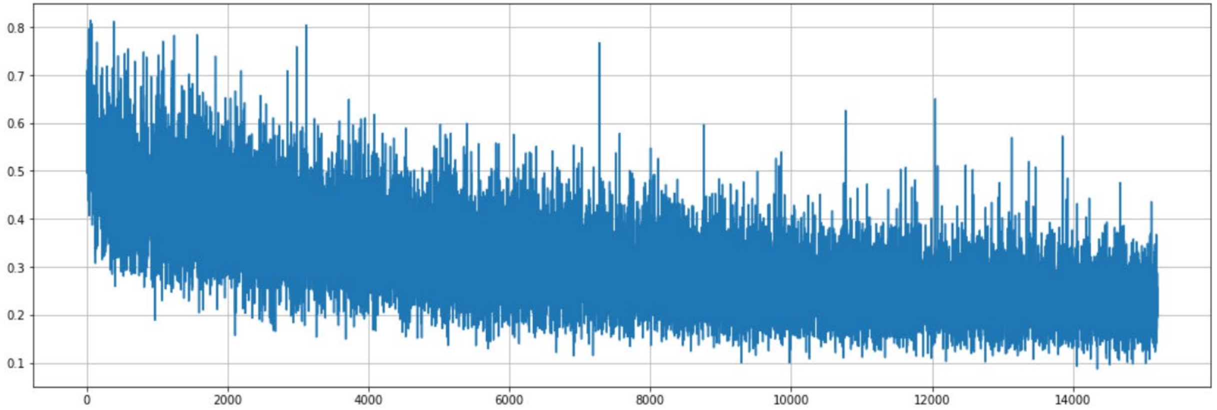


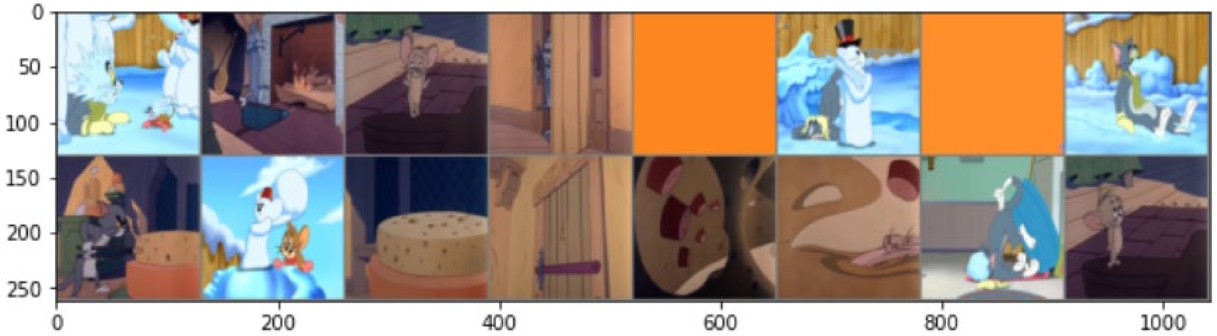
Able to recover the training images

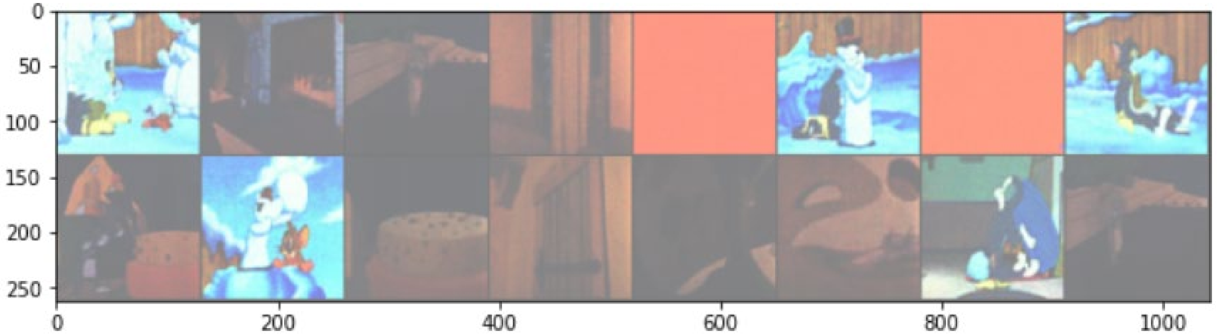


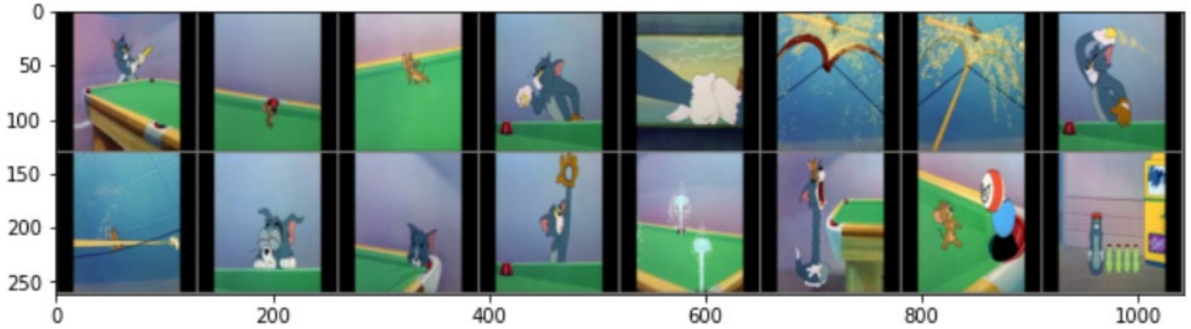
Fails to reconstruct the test images

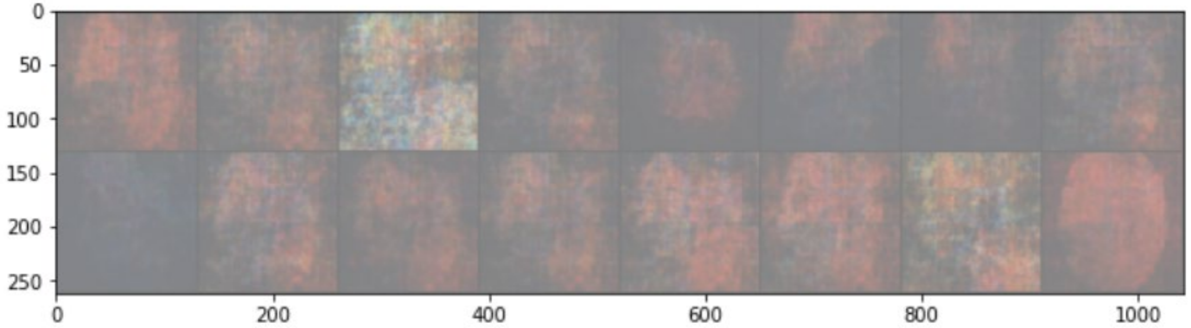
Train 800 epochs





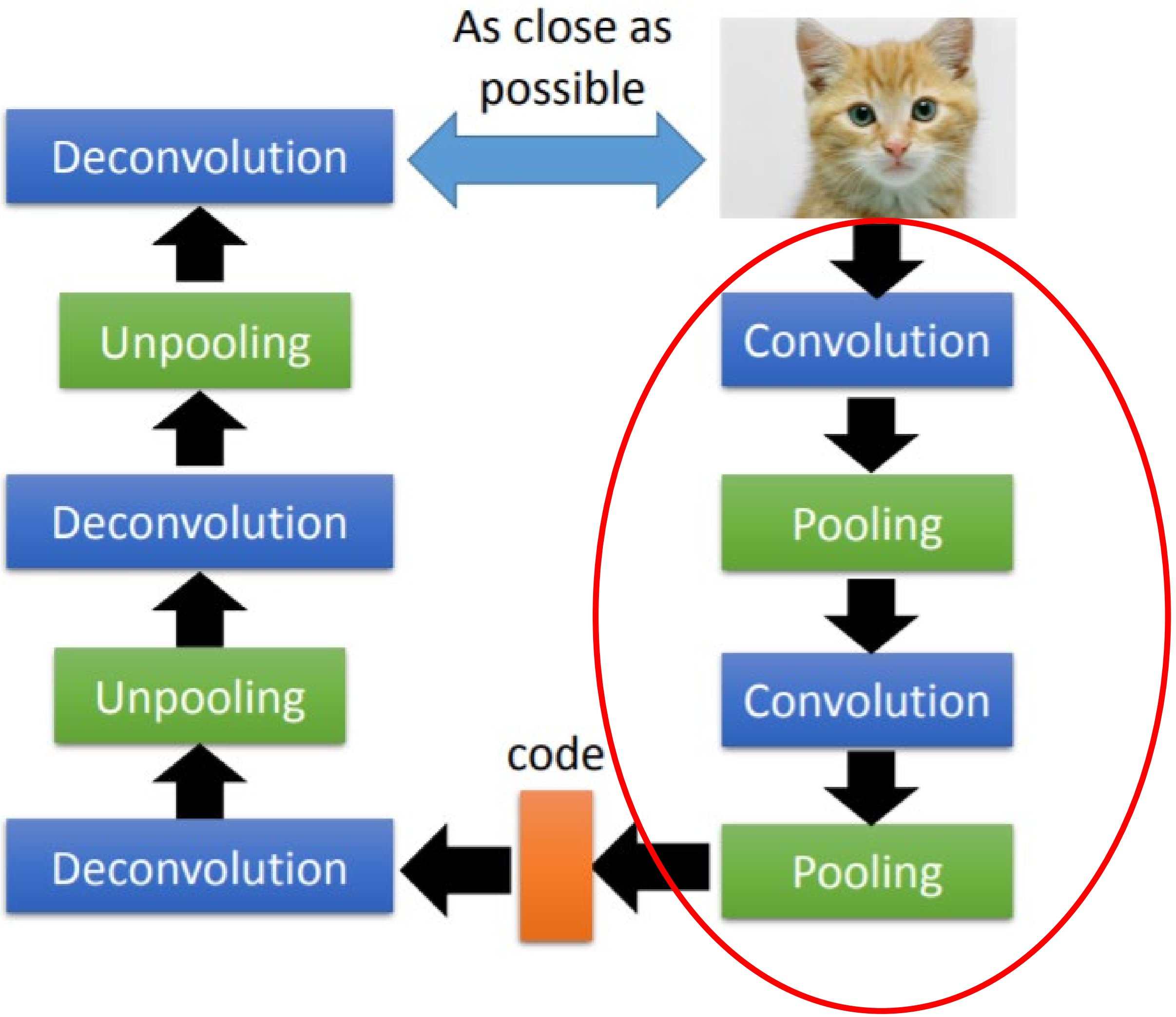






Still fails to reconstruct the test images

Encoder



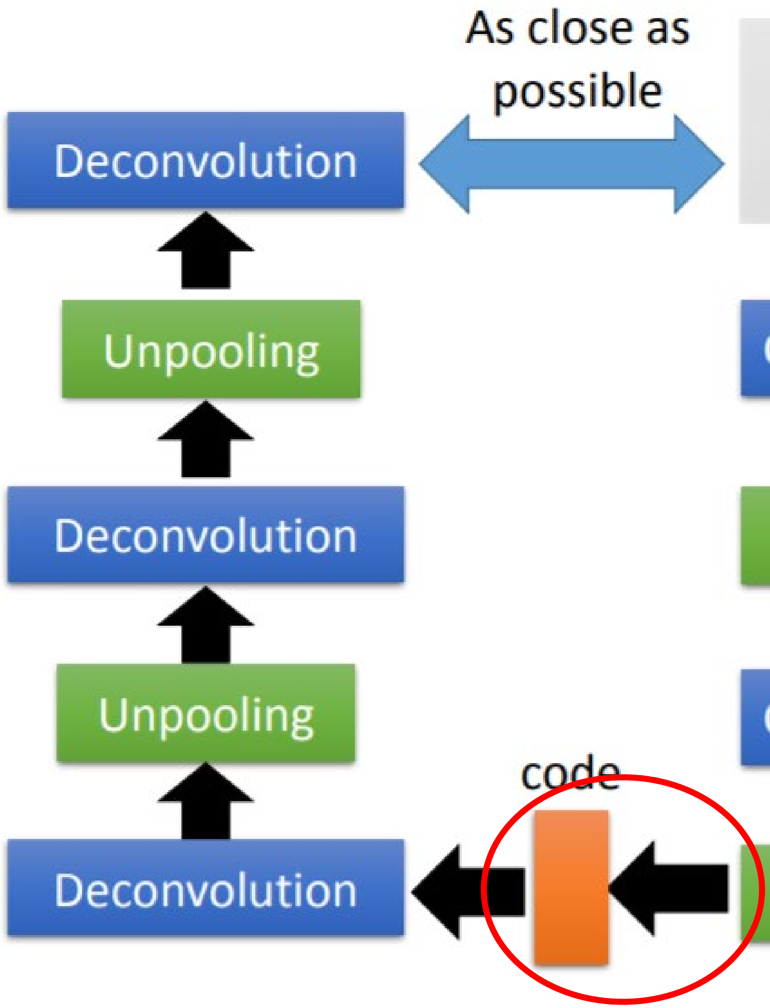

```
self.encoder = nn.Sequential(
  nn.Conv2d(3, 32, kernel_size=2, stride=2),
  nn.BatchNorm2d(32, eps=1e-05, momentum=0.1, af
  nn.ReLU(),
  nn.Conv2d(32, 64, kernel_size=2, stride=2),
  nn.BatchNorm2d(64, eps=1e-05, momentum=0.1, af
  nn.ReLU(),
  nn.Conv2d(64, 128, kernel_size=2, stride=2),
  nn.BatchNorm2d(128, 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 to write down the feature map size and the results after flatten

- 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




```
class autoencoder(nn.Module):
  def __init__(self,i=1024,o=64):
    super(autoencoder, self).__init__()
    self.encoder = nn.Sequential(
      nn.Conv2d(3, 32, kernel_size=2, stride=
      nn.BatchNorm2d(32, eps=1e-05, momentum=
      nn.ReLU(),
```

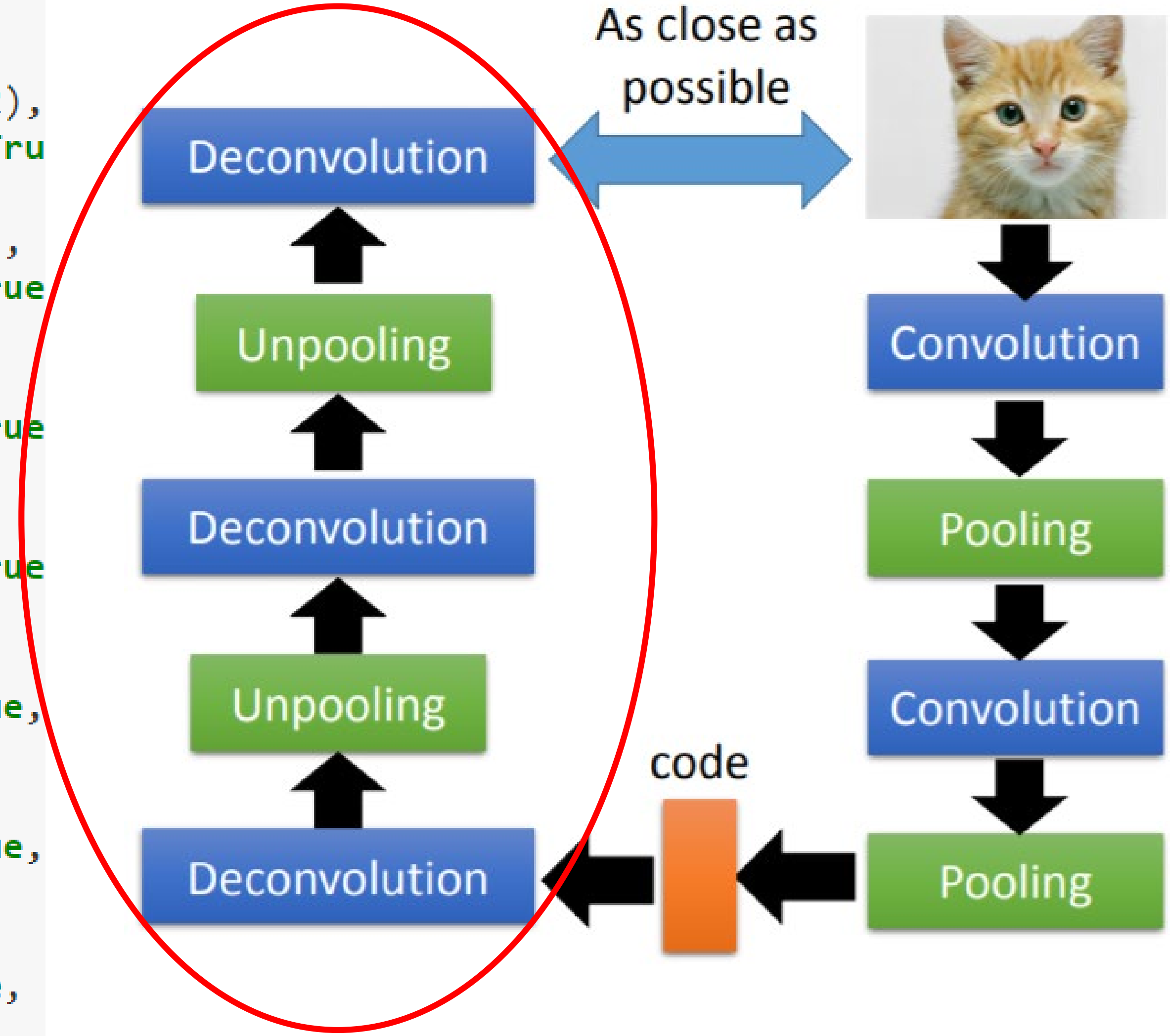
```
    nn.BatchNorm2d(1024, eps=1e-05, momentum=0.
    nn.ReLU(),
    nn.Conv2d(1024, 1024, kernel_size=2, stride
    nn.BatchNorm2d(1024, eps=1e-05, momentum=0.
    nn.ReLU(),
    Flatten(),
    nn.Linear(in_features=i, out_features=o),
)
```

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

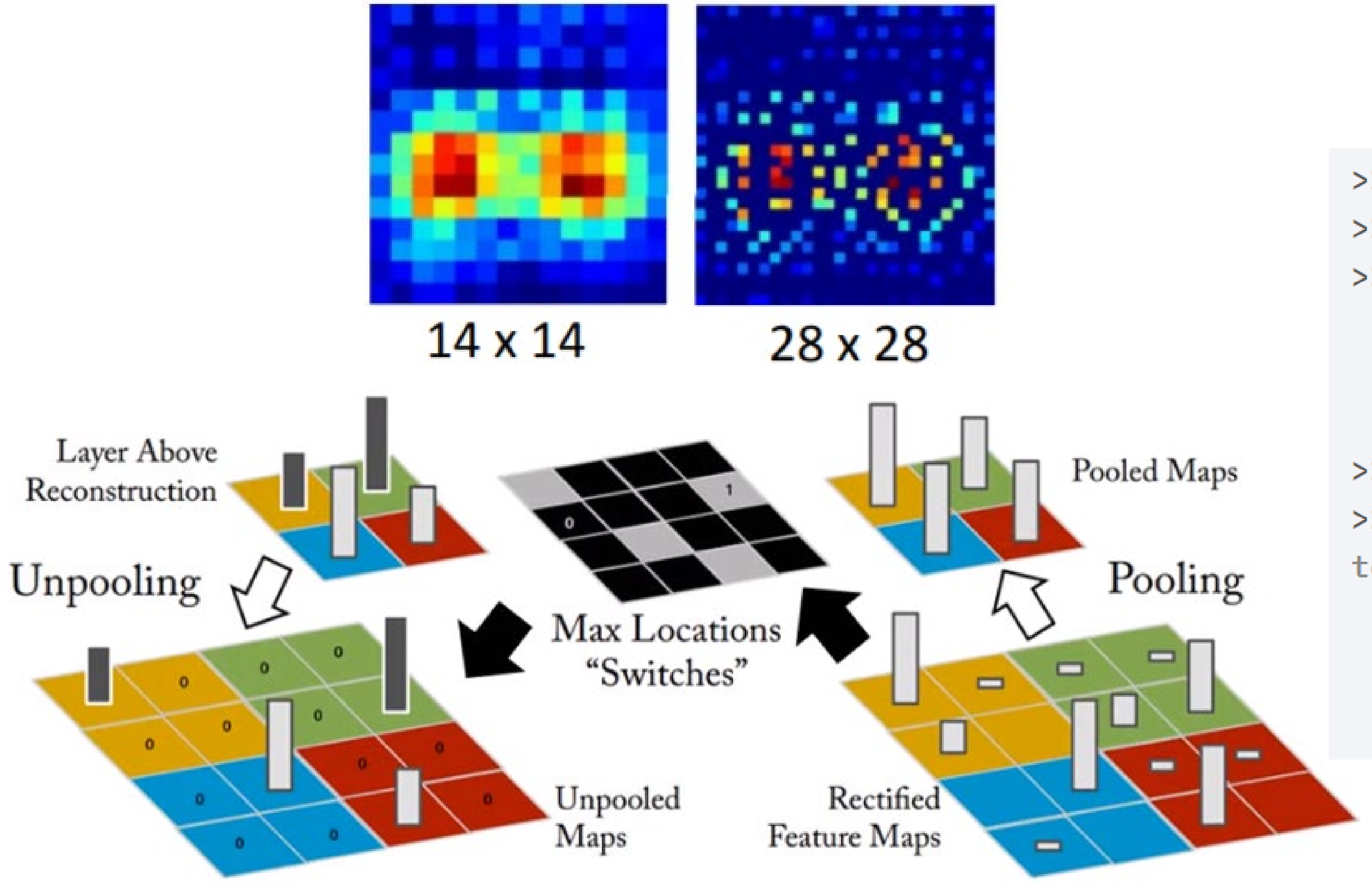
Reference: 李弘毅 ML Lecture 16 https://youtu.be/Tk5B4seA-AU

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



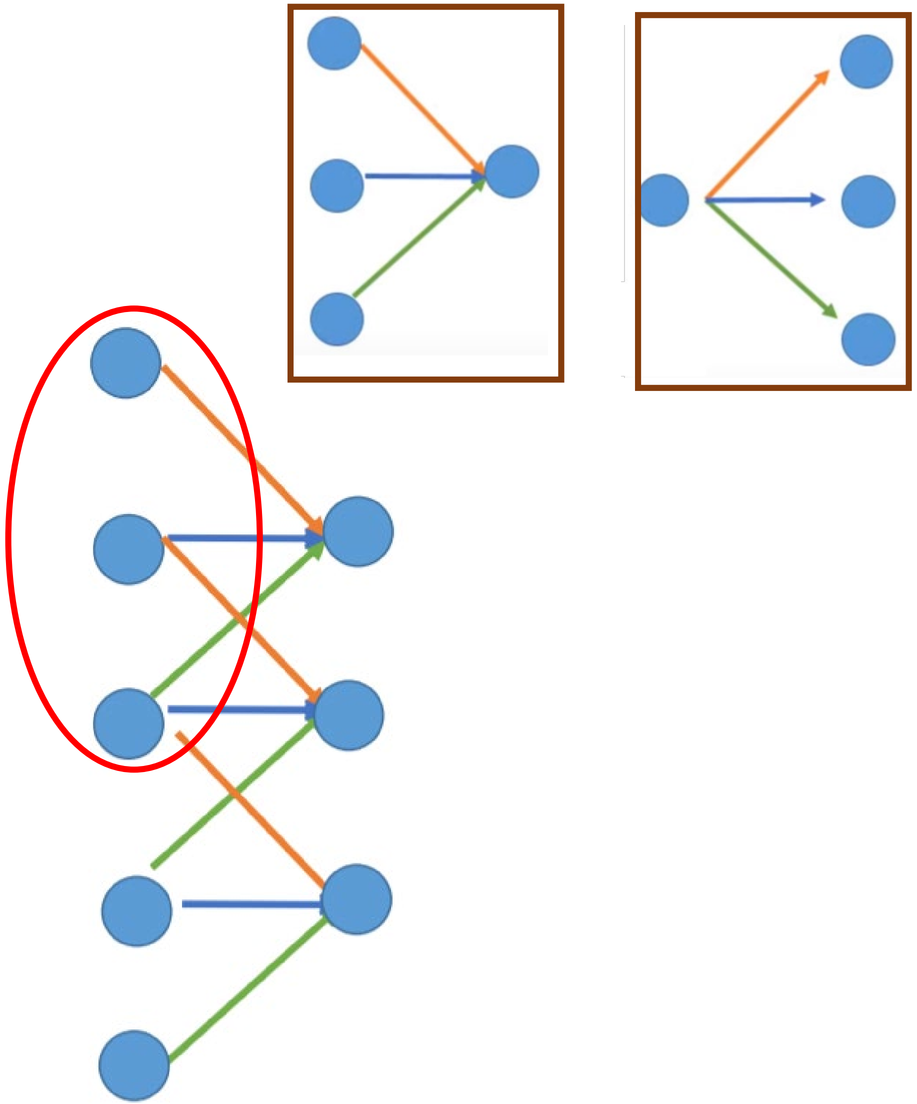

```
>>> pool = nn.MaxPool2d(2, stride=2, return_indices=True)
>>> unpool = nn.MaxUnpool2d(2, stride=2)
>>> input = torch.tensor([[[[ 1.,  2,  3,  4],
                            [ 5,  6,  7,  8],
                            [ 9, 10, 11, 12],
                            [13, 14, 15, 16]]]])
>>> output,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

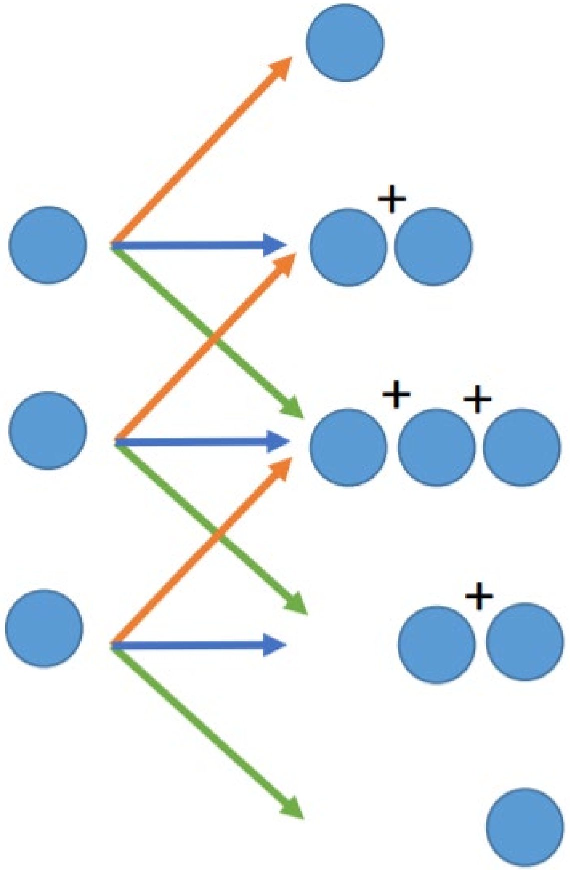
Reference: 李弘毅 ML Lecture 16 https://youtu.be/Tk5B4seA-AU

Deconvolution

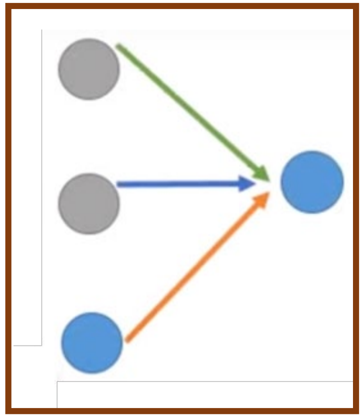
1D convolution, k=3

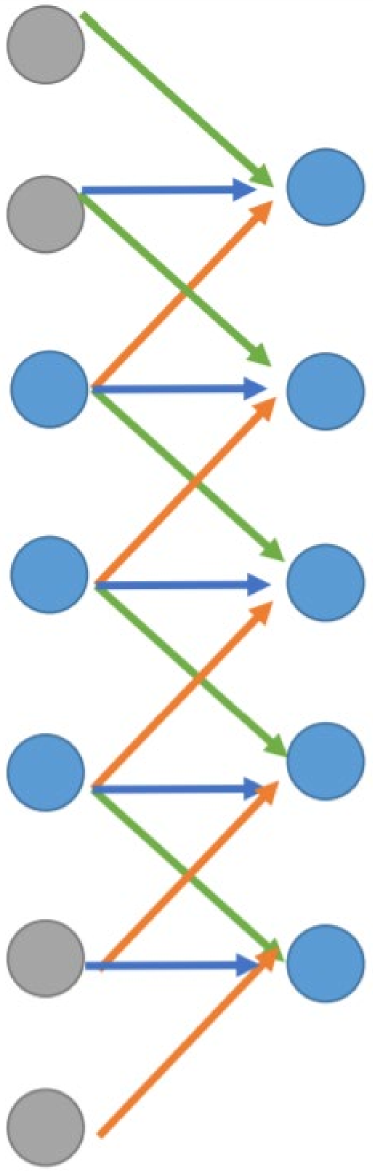
1D deconvolution, k=3





1D convolution, k=3





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

We only use deconvolution for up sampling, un-pooling is not used

```
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(),
)
```

Practice to write down the feature map size after deconvolution

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

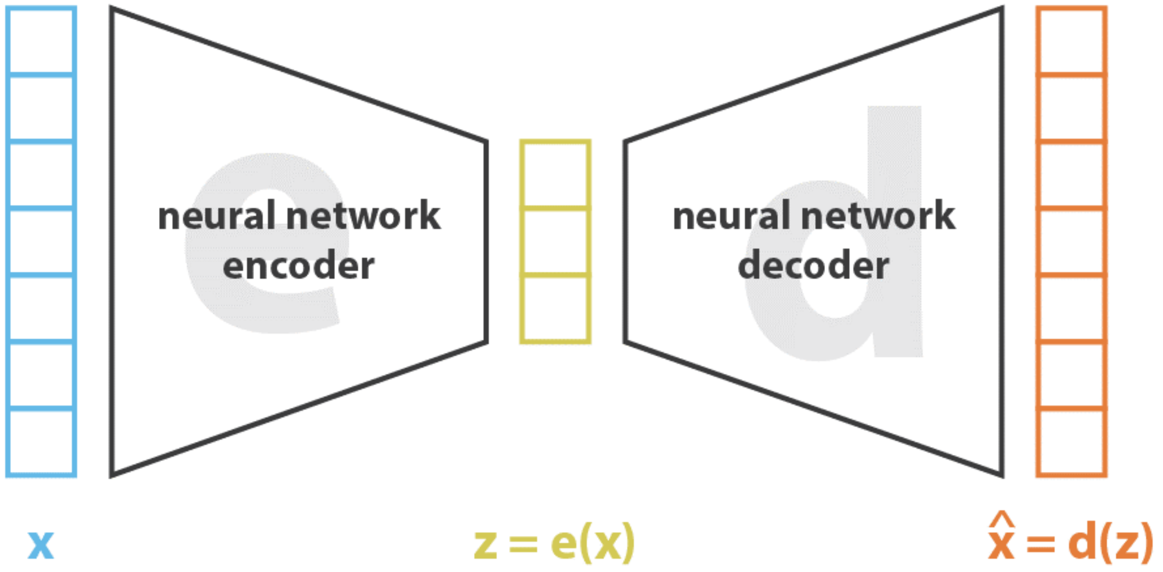
PYTORCH

Feature map size after 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

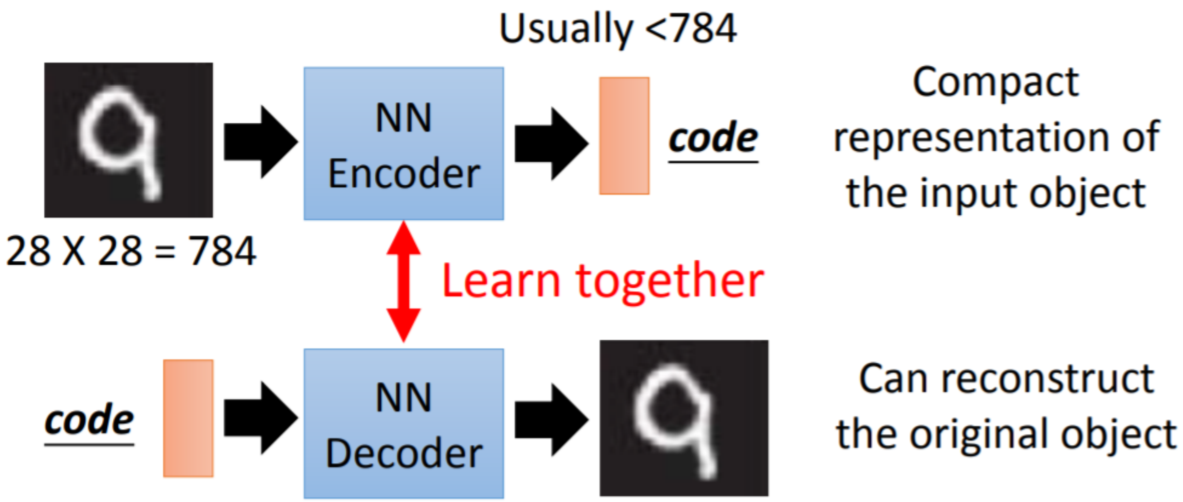
```
[13]: for batchX, _ in loader:
        break;
      print(batchX.shape)

      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
```

Save and load PyTorch model

Pass images to AE to get their compact representation (latent vectors)

[37]:
```python
for step, (batchX, batchY) in enumerate(loader):
  tensorY = model.encoder(batchX.to(device))
  if(step==0):
    arrayX = np.array(tensorY.cpu().detach().numpy())
    arrayY = batchY.cpu().detach().numpy()
  else:
    arrayX = np.concatenate((arrayX, tensorY.cpu().detach().numpy()))
    arrayY = np.concatenate((arrayY, batchY.cpu().detach().numpy()))
print(arrayX.shape, arrayY.shape)
```

```
(298, 64) (298,)
```



Autoencoder learns a compact representation of the input image

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

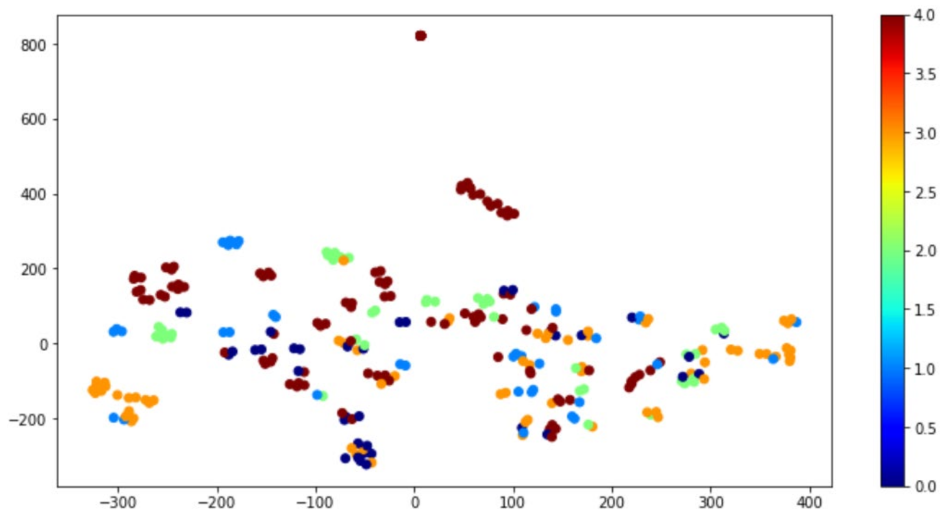
Use *t*-SNE to reduce the latent vector dimensions from 64 to 2

[38]:
```
from sklearn.manifold import TSNE
tsne = TSNE(perplexity=5, n_components=2, init='pca', n_iter=5000)
# try perlexity = 5, 10, 30, 50
```

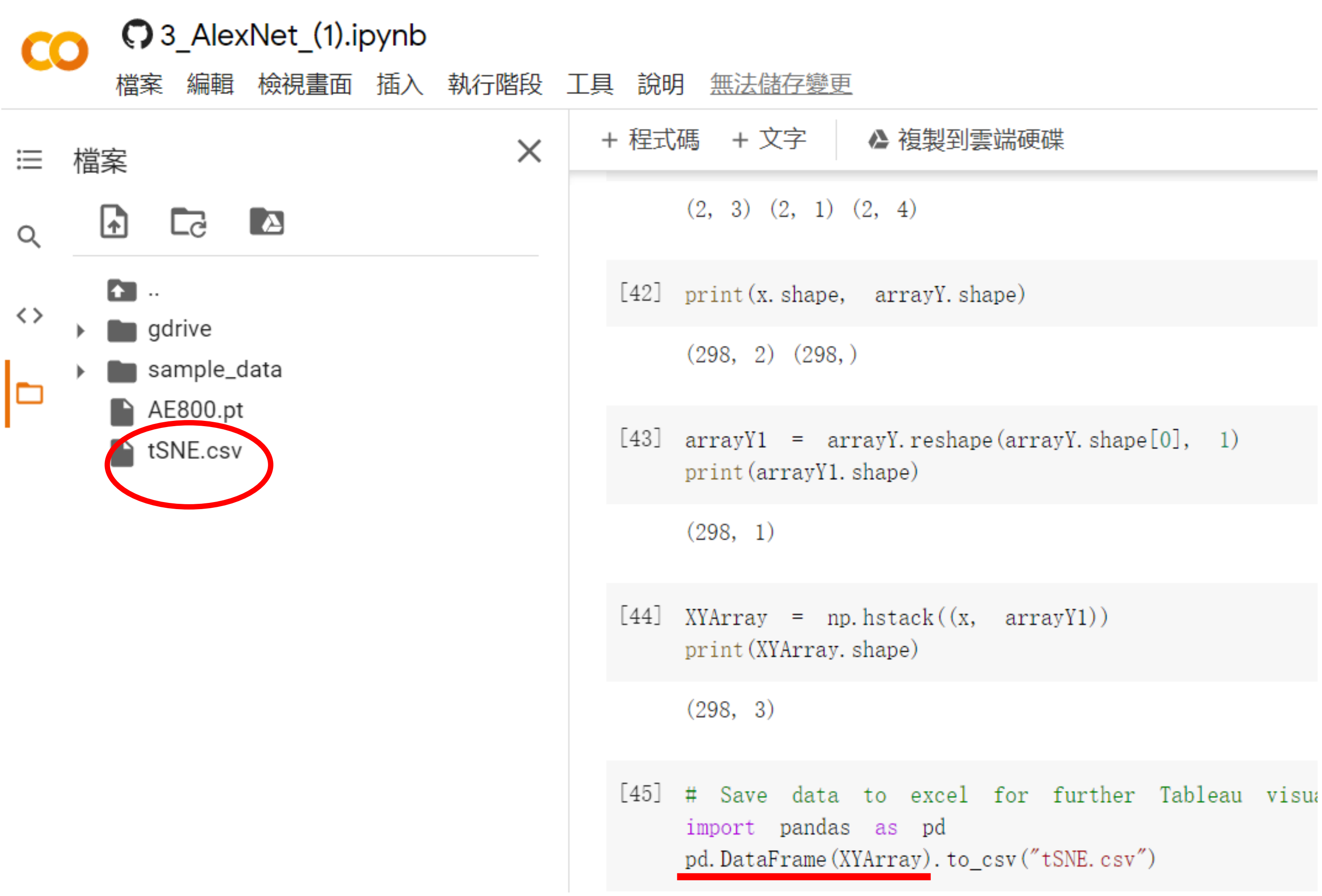
[39]:
```
x=tsne.fit_transform(arrayX)
print(x.shape)
```

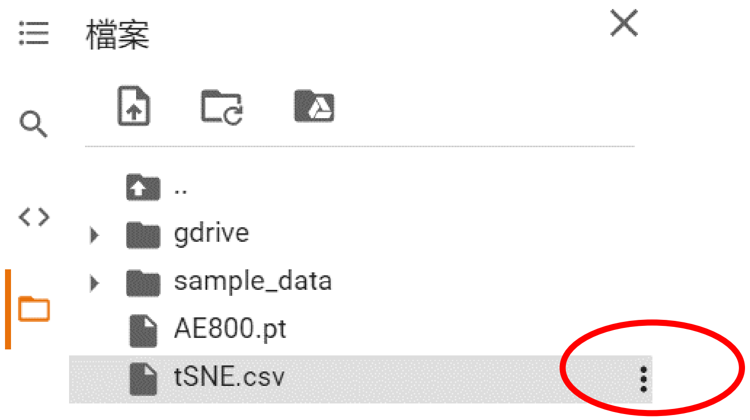
```
(298, 2)
```

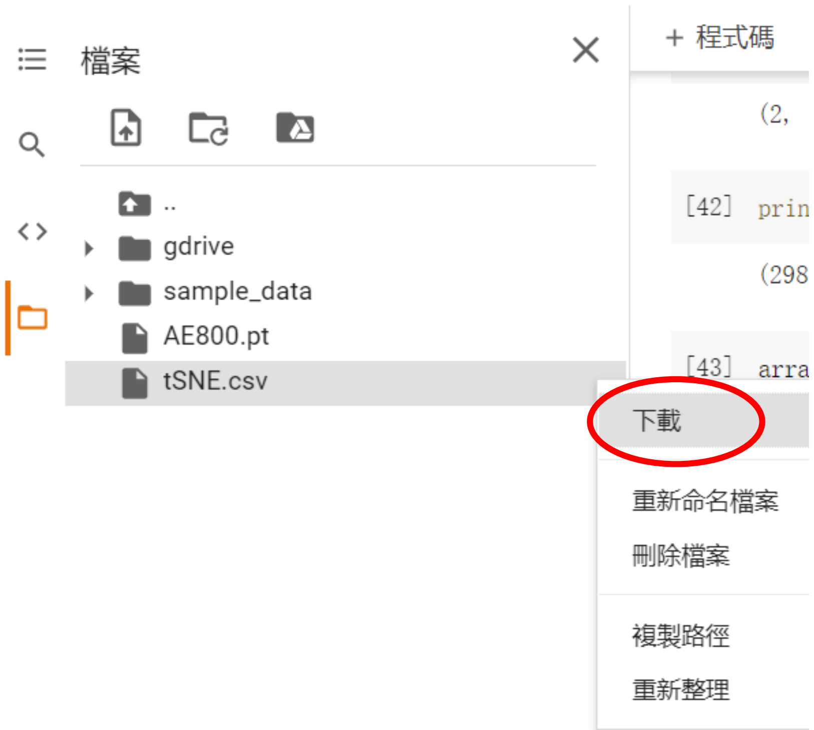
[40]:
```
plt.figure(figsize=(18,9))
plt.scatter(x[:, 0], x[:, 1], c= arrayY)
plt.show()
```



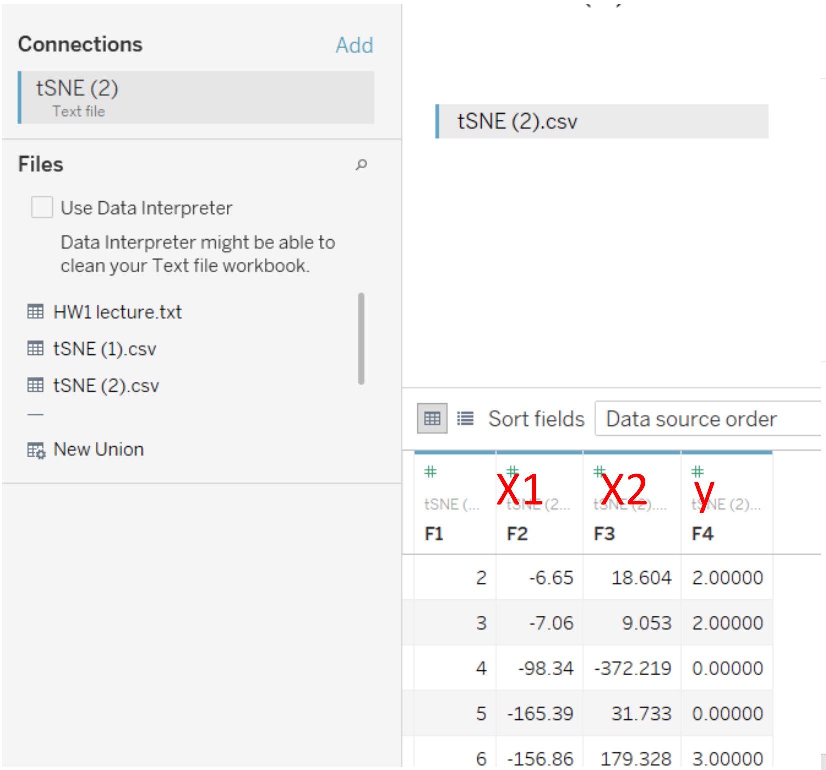
Save the results to csv file

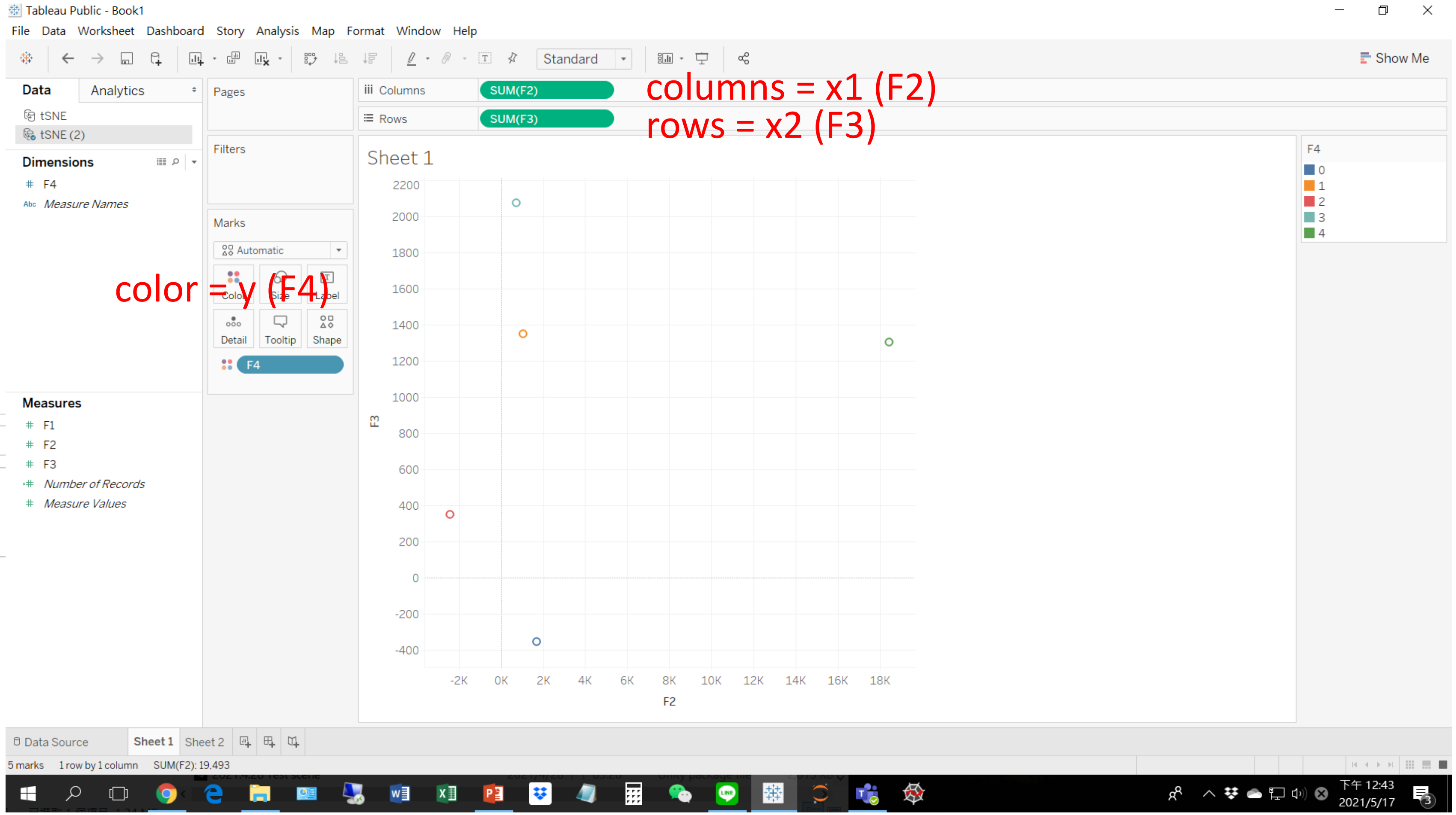






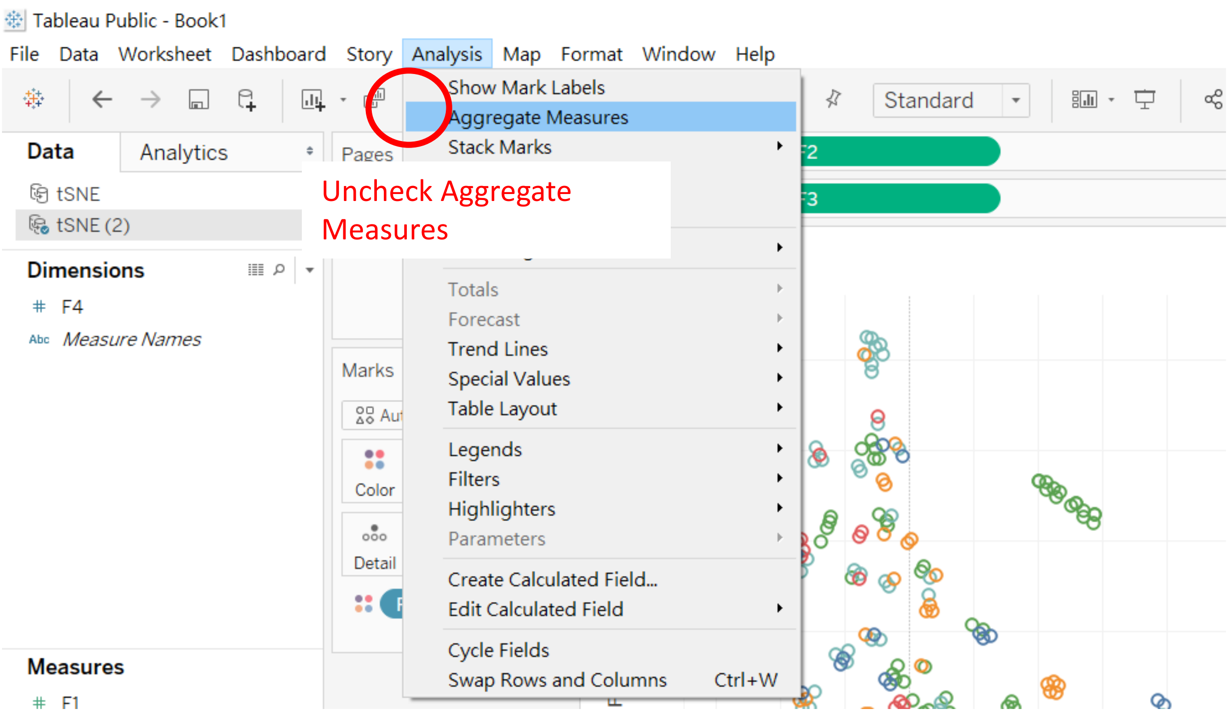
Visualize the downloaded file in Tableau public



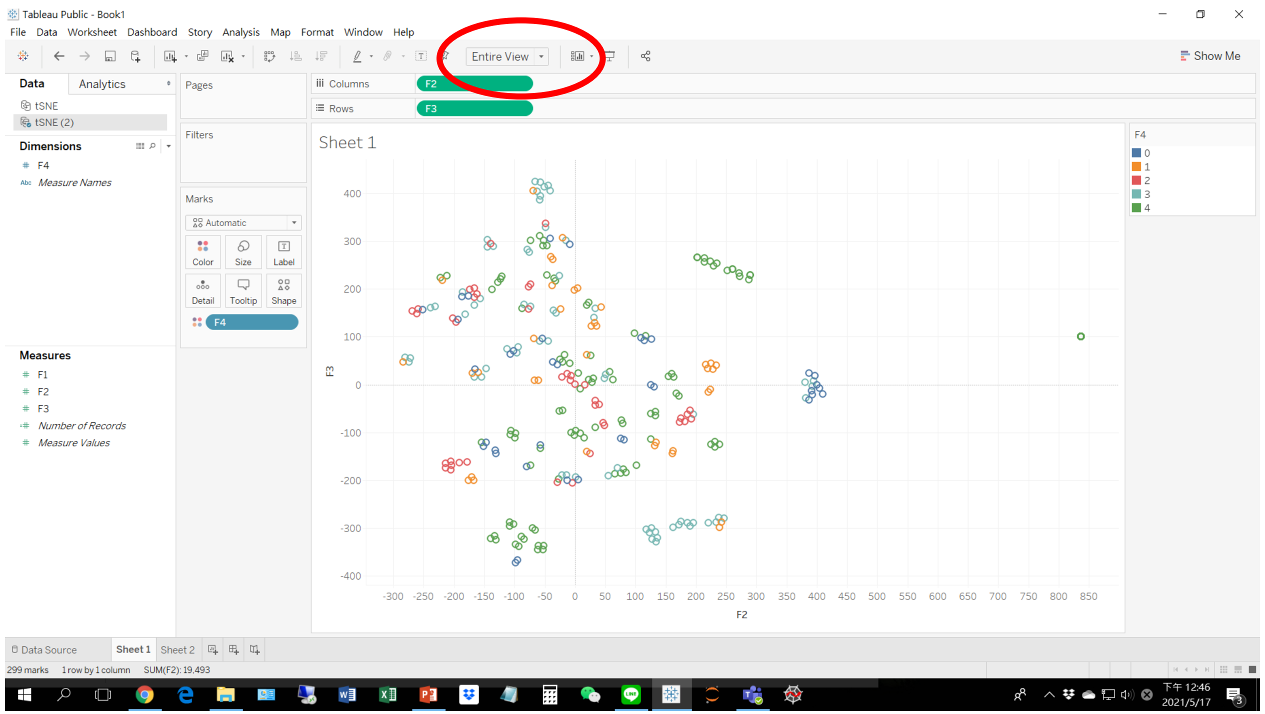


Visualize the downloaded file in Tableau public

disable Aggregate Measures



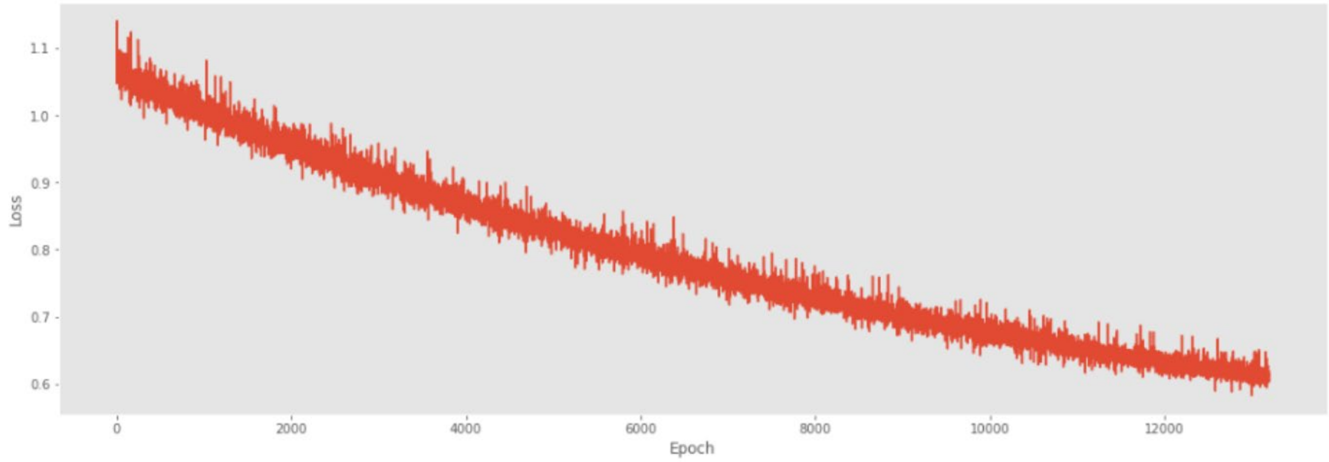
Select entire veiw



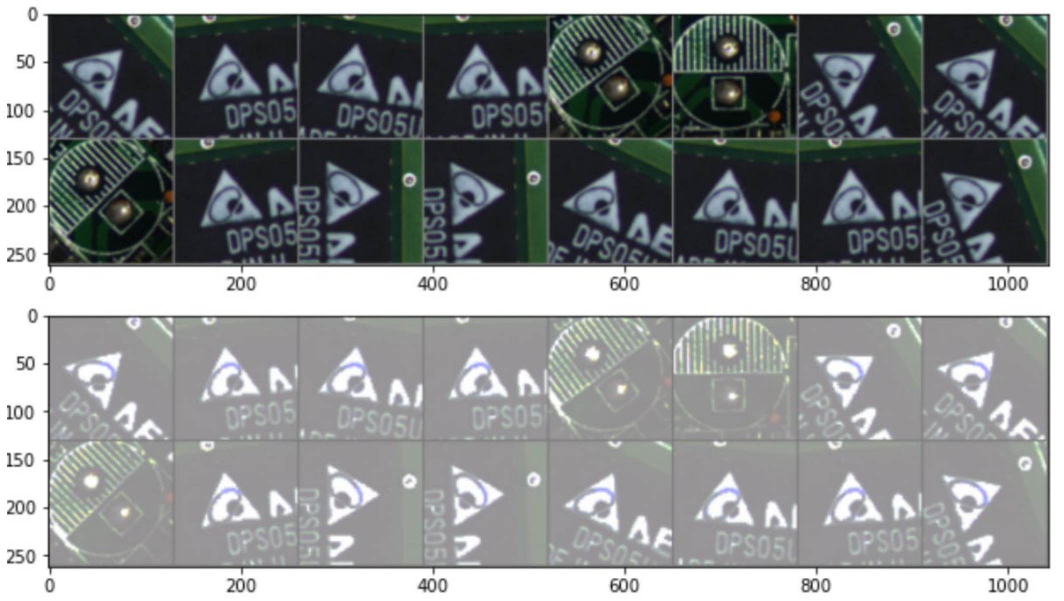
HW6 (1)

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

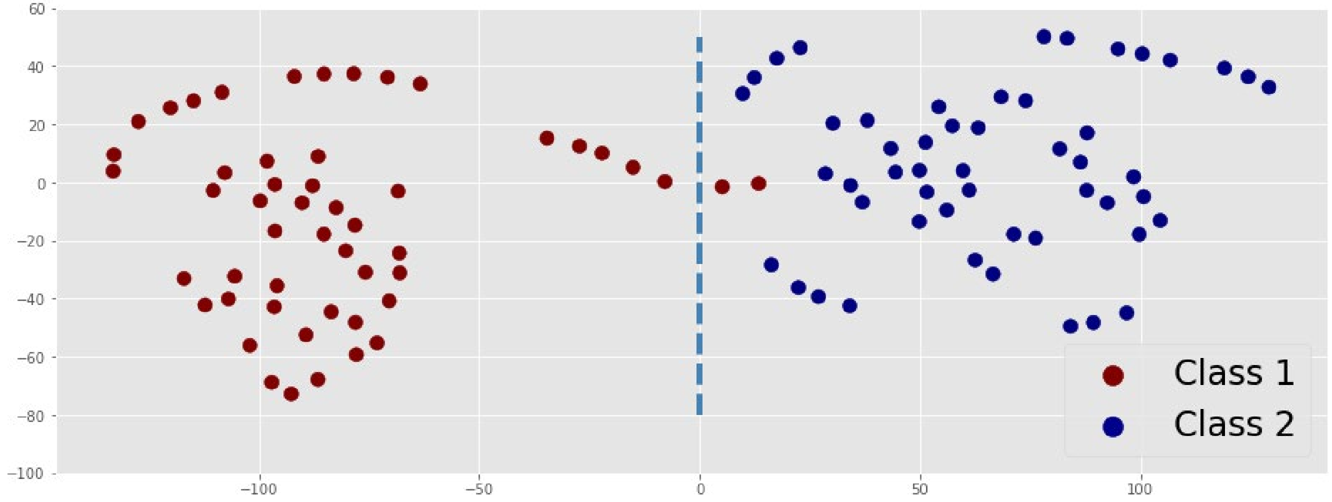
Class 1 = 100, Class 2 = 100, Latent vector size = 64, 2000 epochs, batch=32, Image size = 180x180x3



Recovered training images

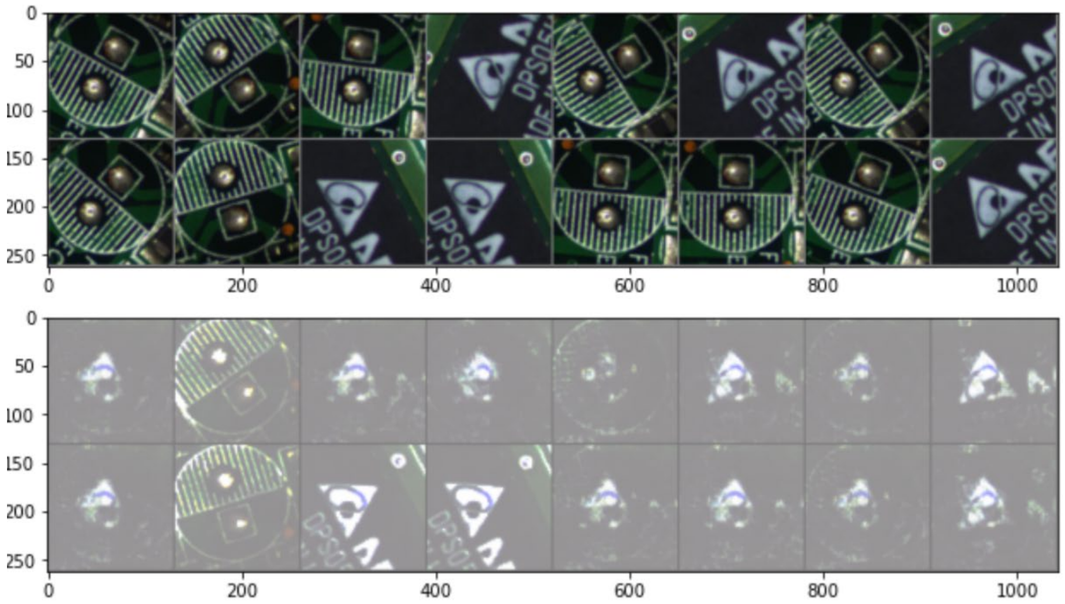


t-SNE (perplexity=10) results of training images

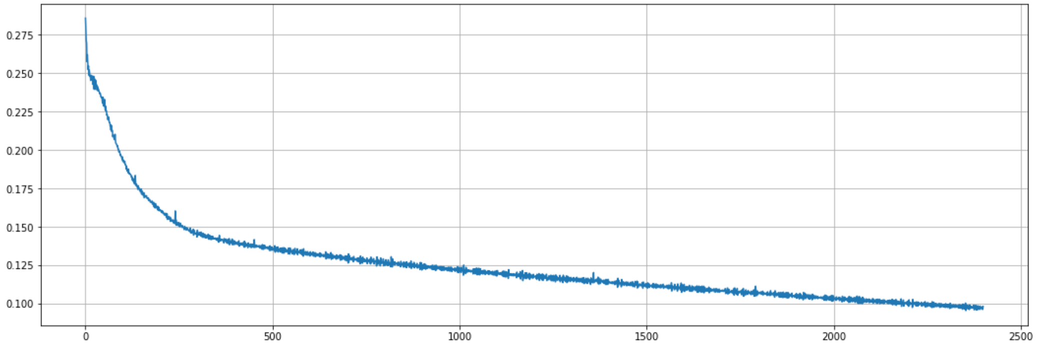


1085442 Carlos

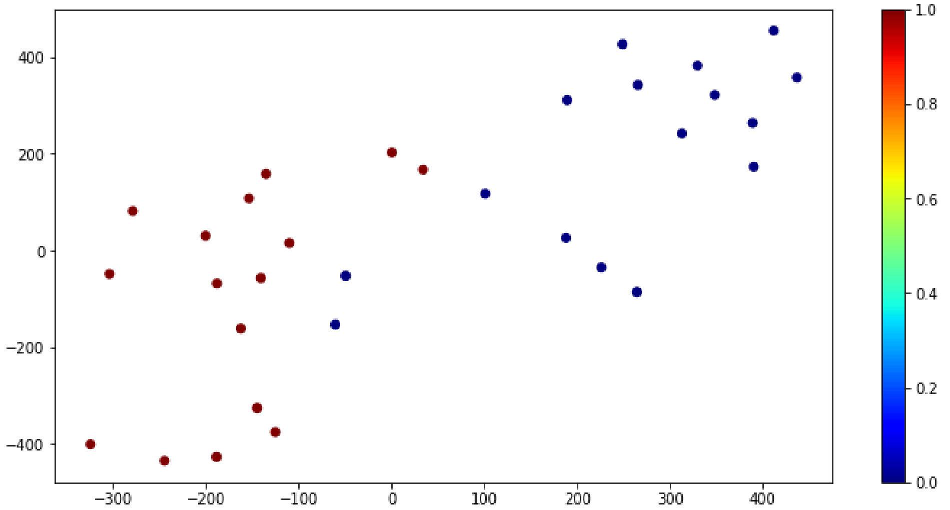
Recovered test images

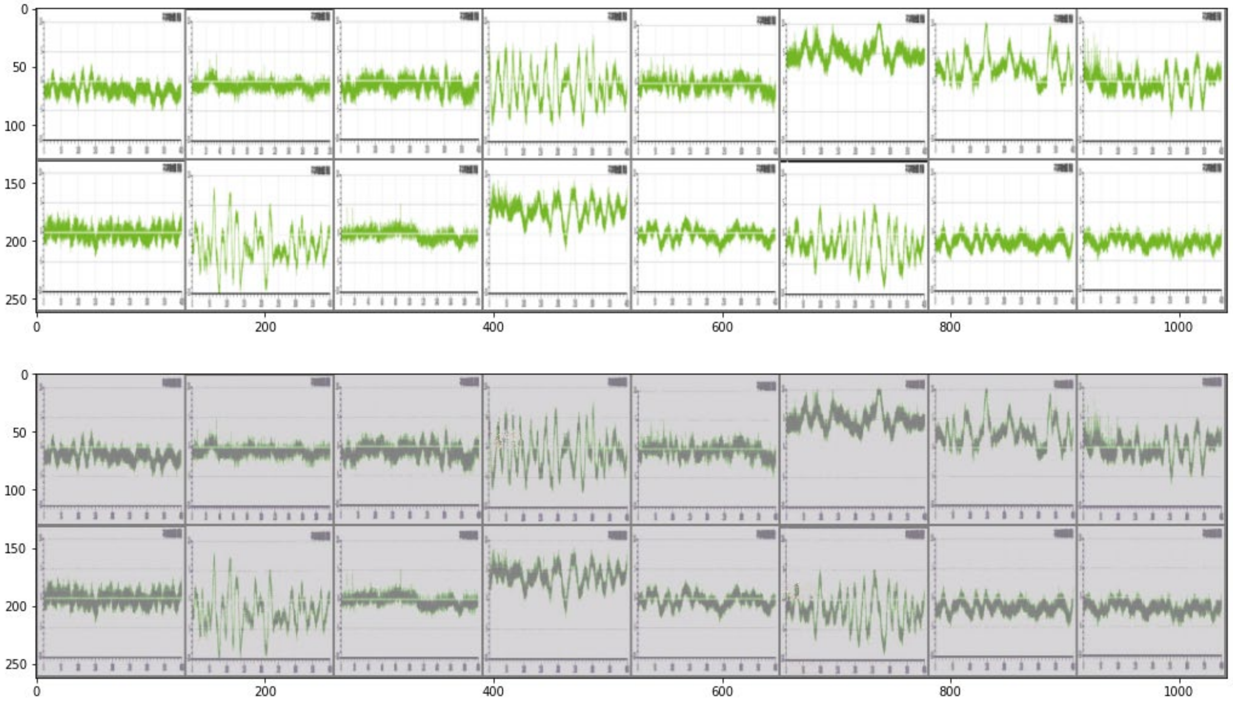


Normal = 16, Abnormal = 16, Latent vector size = 32, 1200 epochs

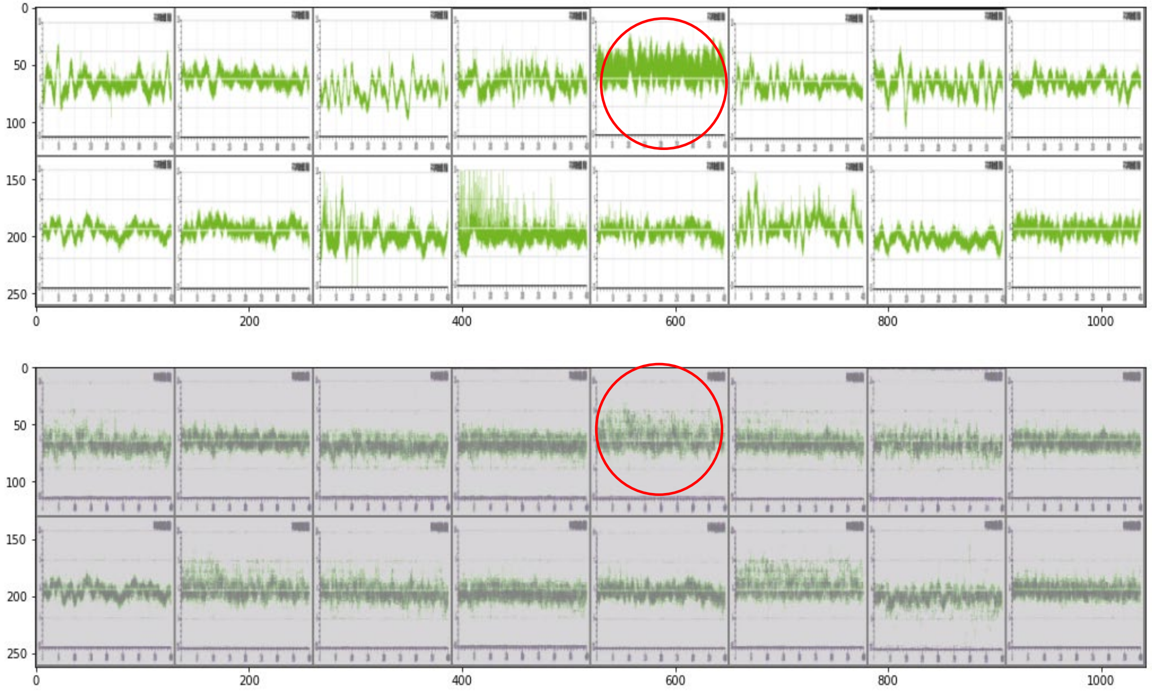


t-SNE (perplexity=?) results of training images



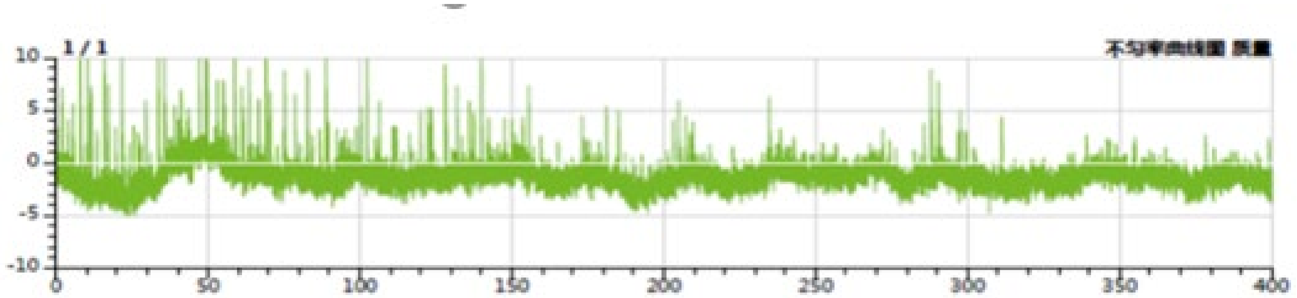


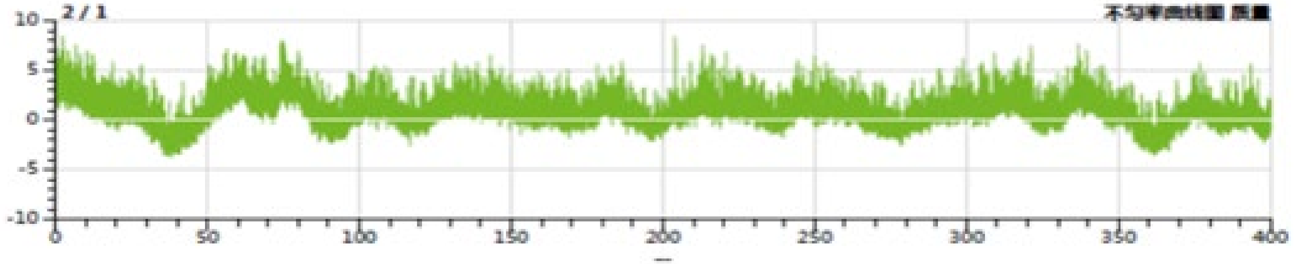
Recovered training images

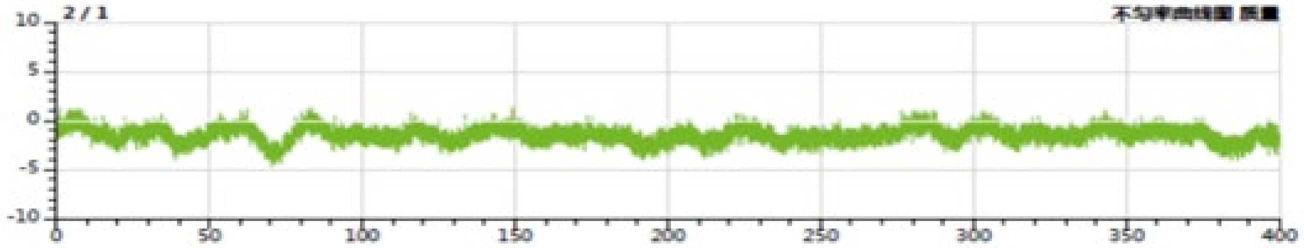


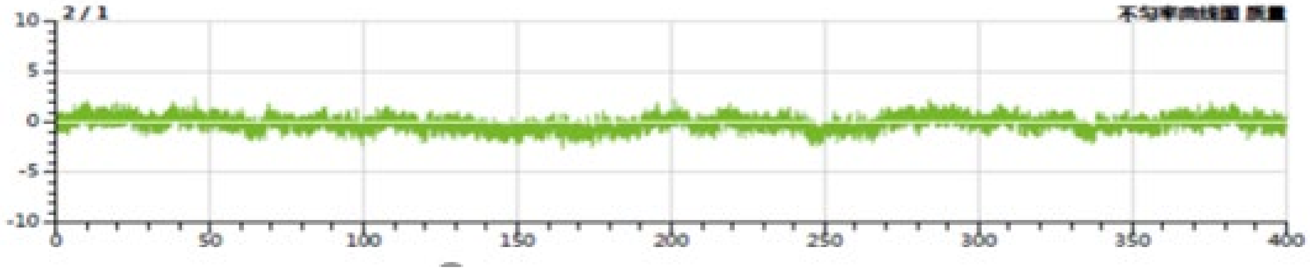
Recovered un-seen test images

Image size = 128*128*3
Class1 = 16，Class2 = 16
Latent vector size = 128
Batch = 128
epoch：20000

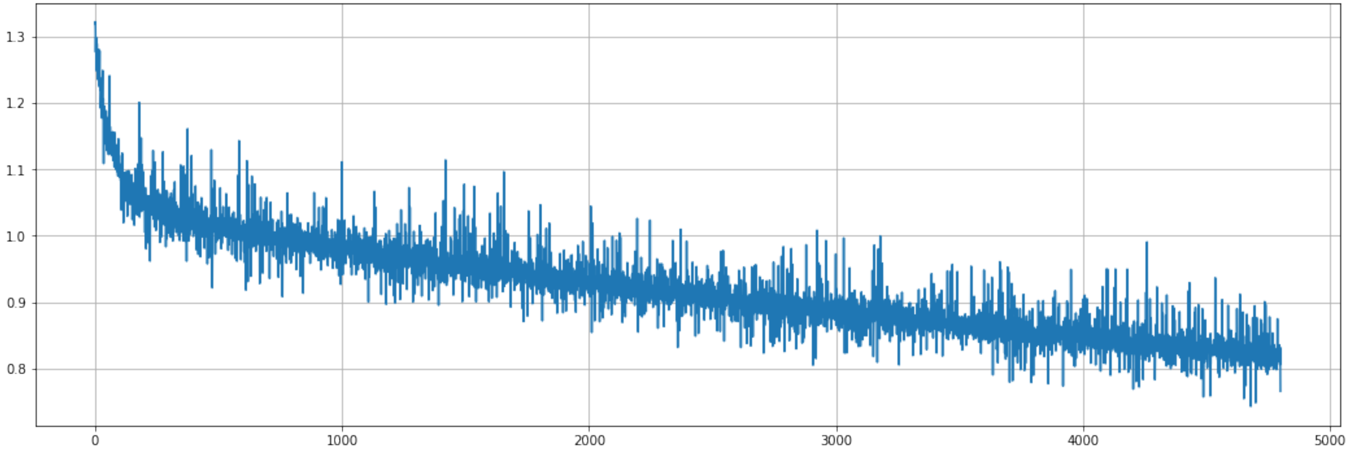




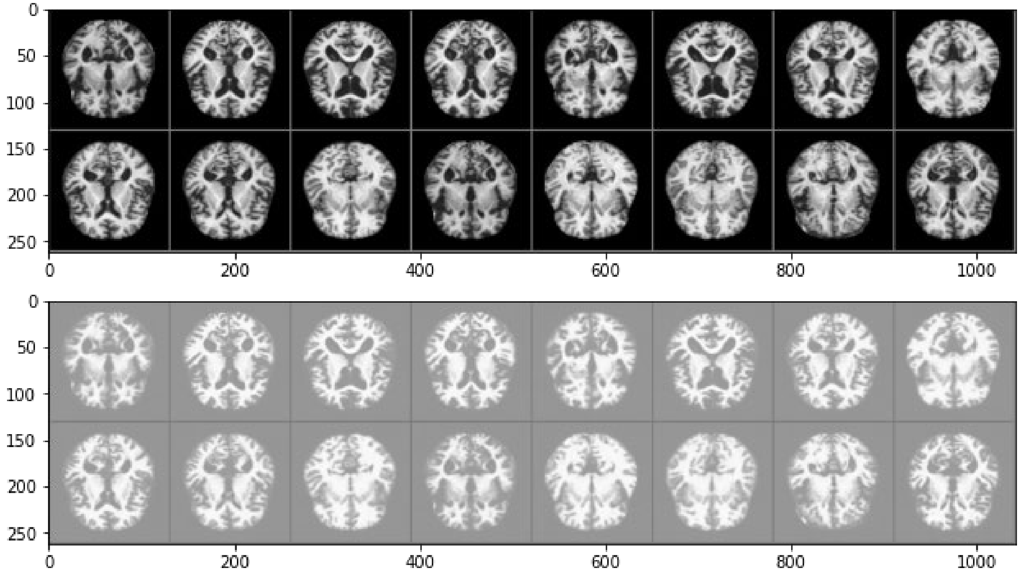




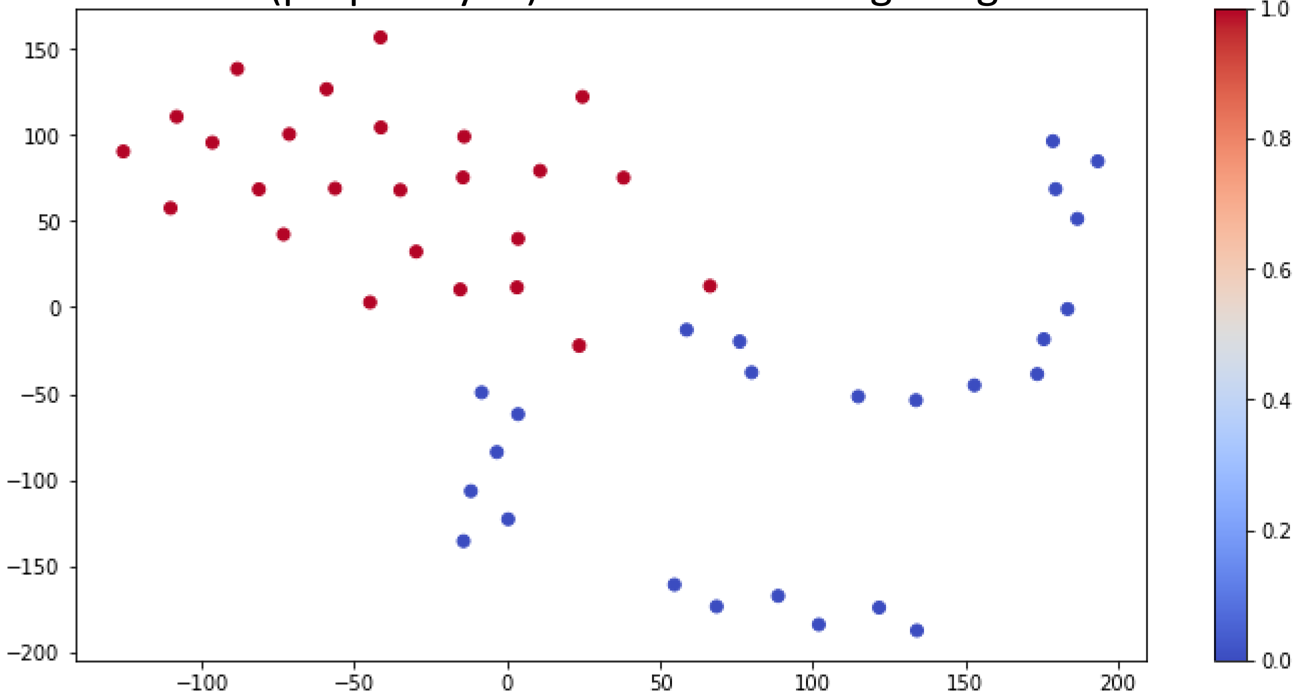
Dementia = 25, health = 25, Latent vector size = 64, 1200 epochs, batch= , Image size = 128x128x3



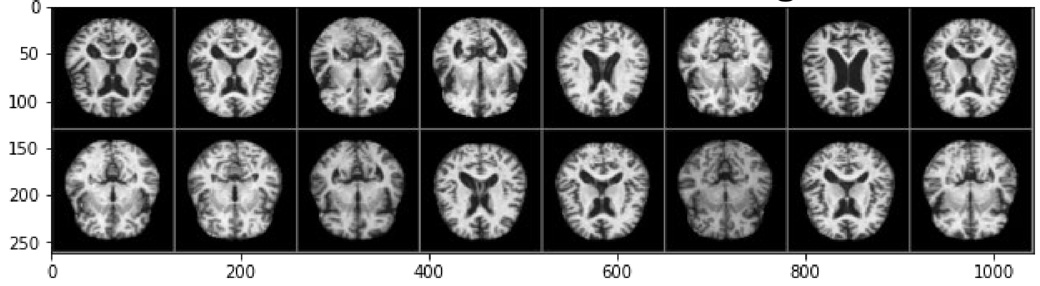
Recovered training images

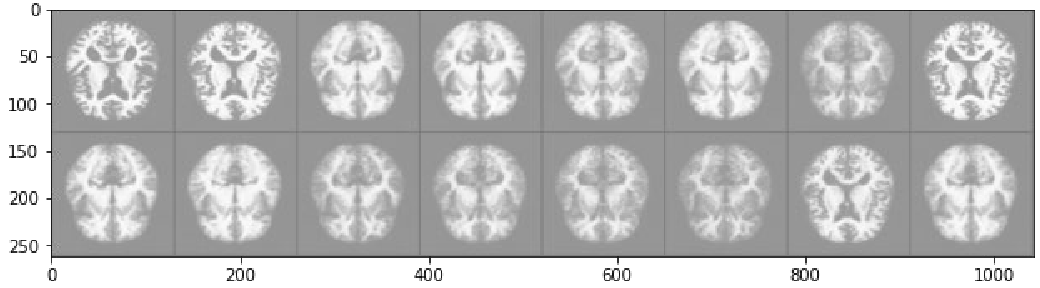


t-SNE (perplexity=5) results of training images

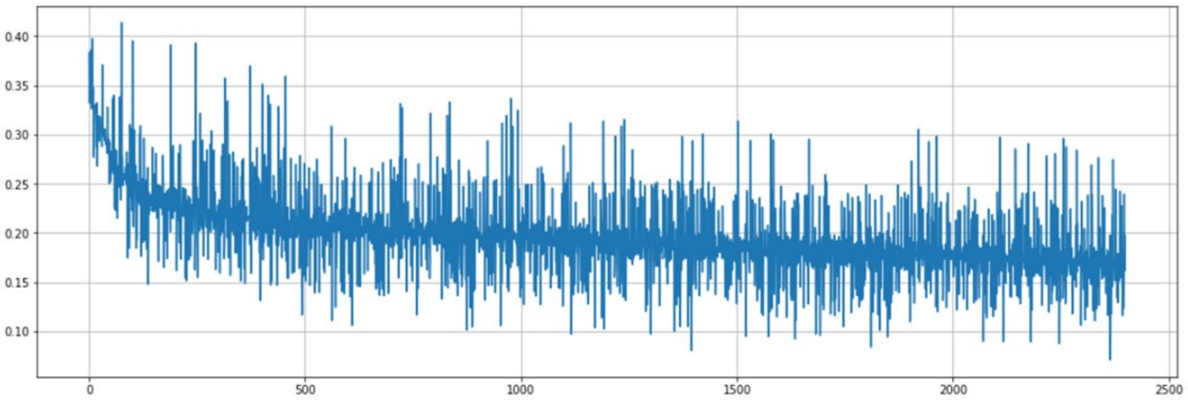


Recovered un-seen test images

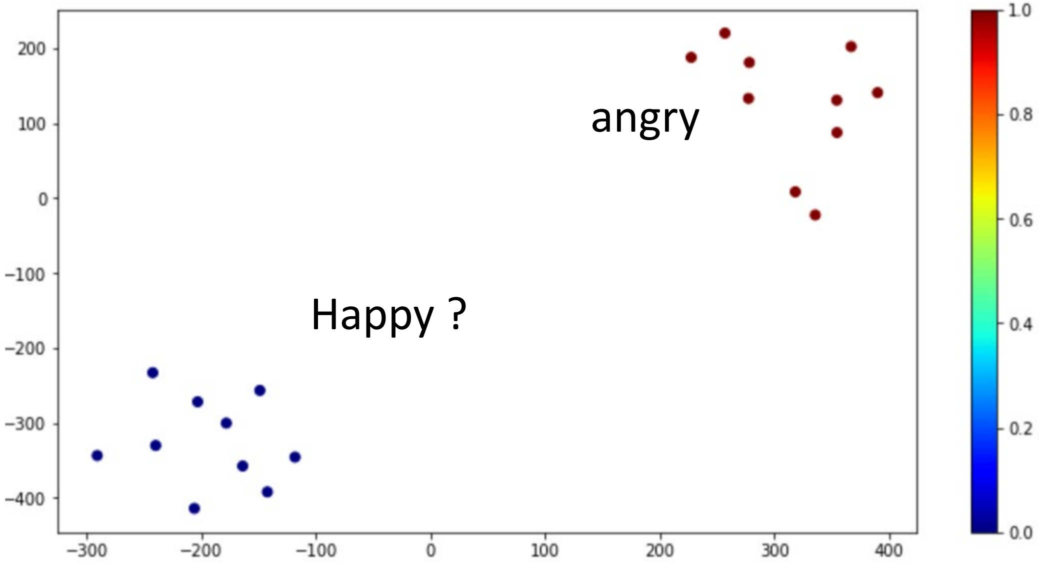




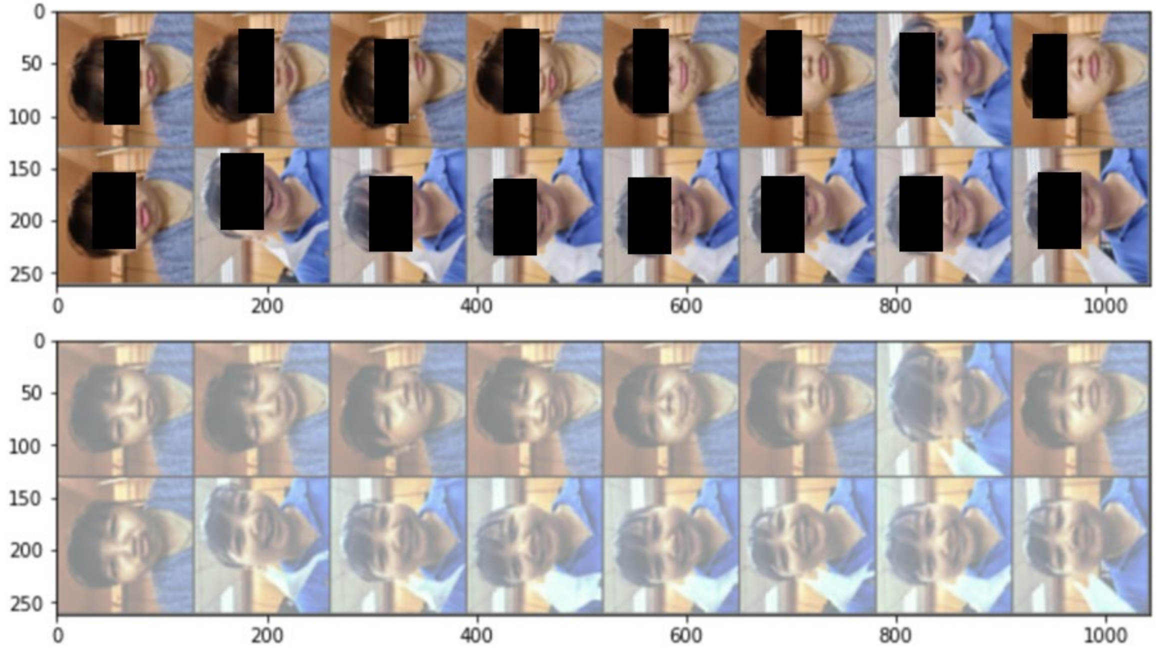
Happy = 10, Angry = 10, Latent vector size = 64, 1160 epochs, batch=16, Image size = 128x128x3



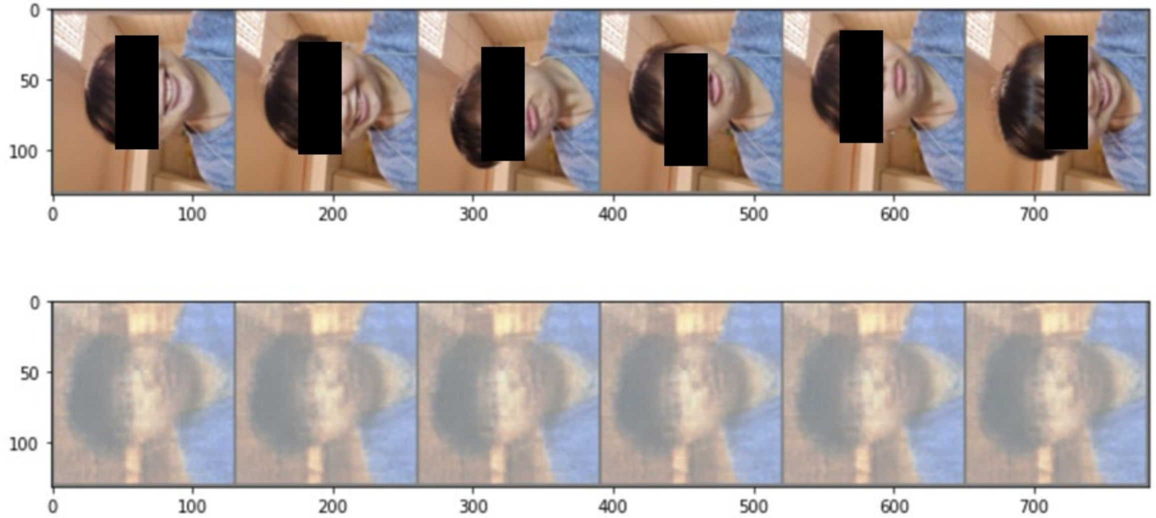
t-SNE (perplexity=5) results of training images



Recovered training images

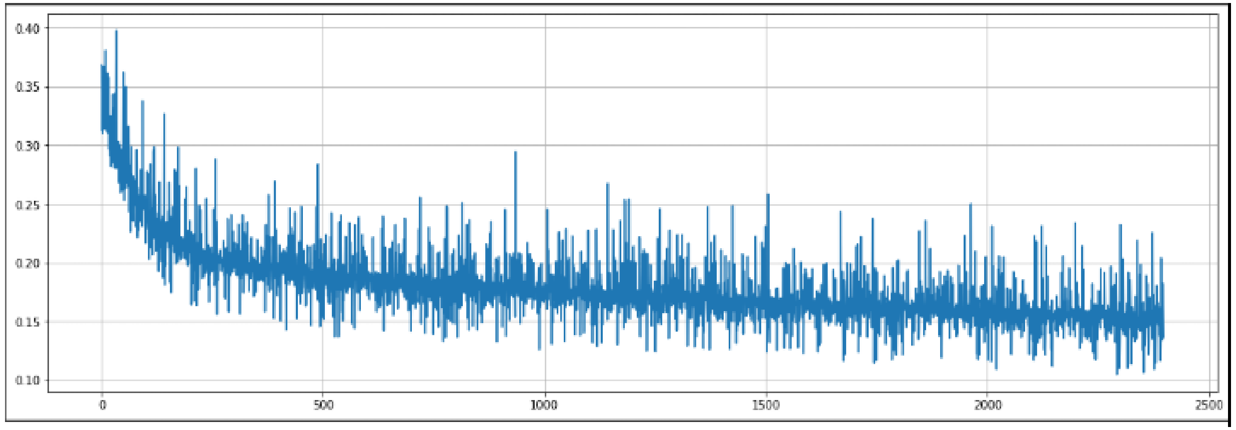


Recovered un-seen test images

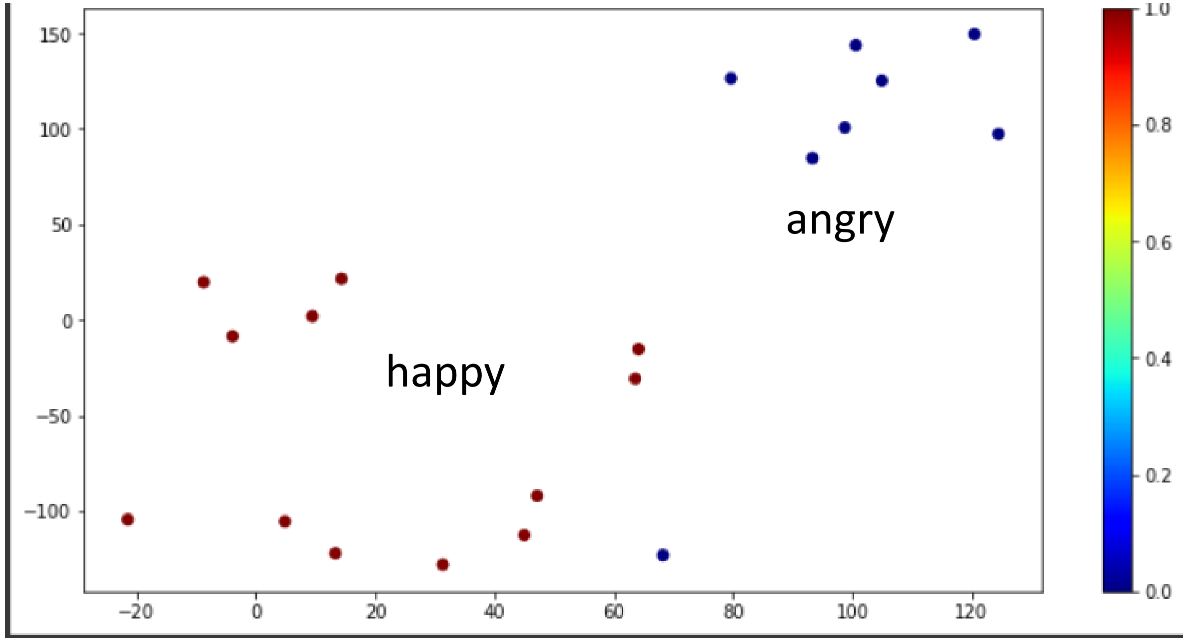


1061259 Keren

Happy = 12, Angry = 8, Latent vector size = 20, 1200 epochs,
batch=?, Image size = ?x?x3



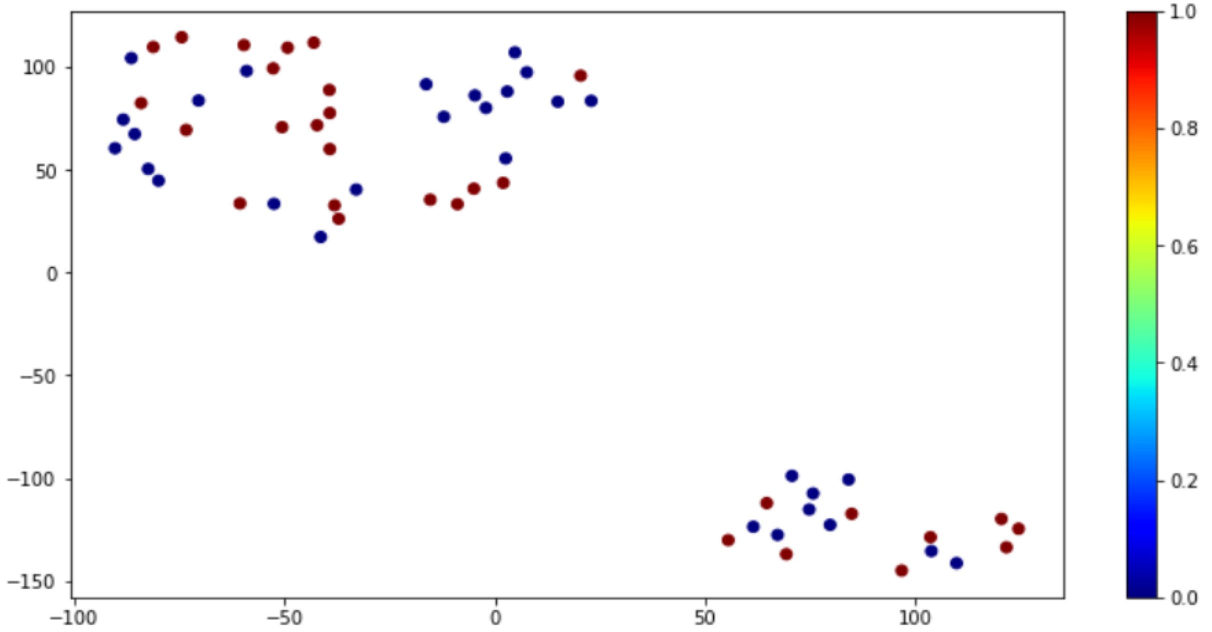
t-SNE (perplexity=5) results of training images

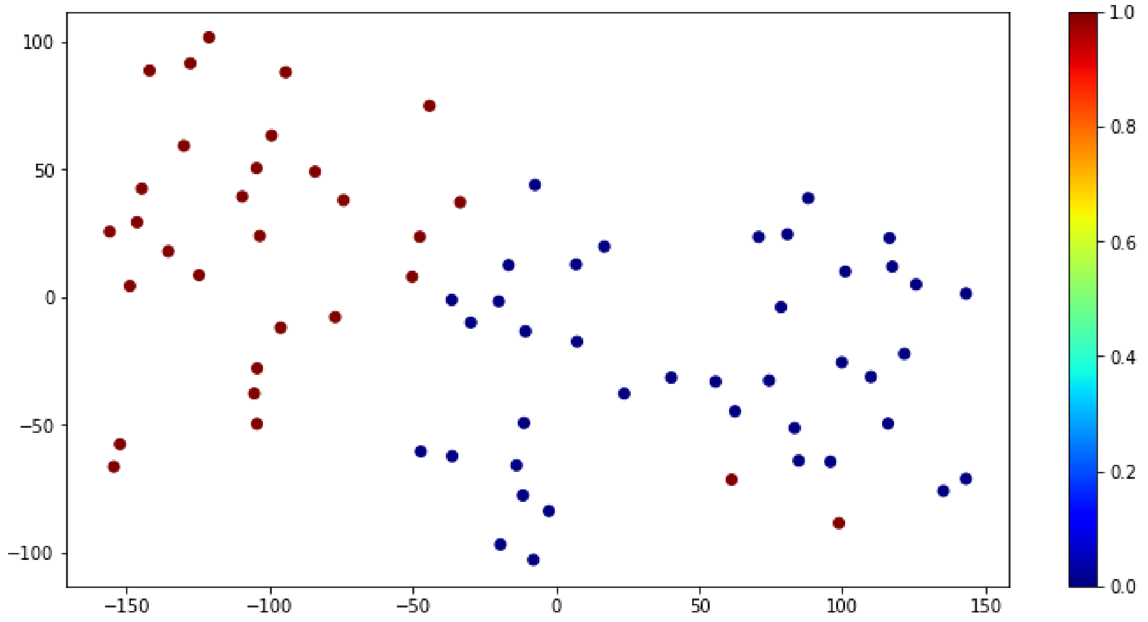


Recovered training images

Recovered un-seen test images

AE's classification idea is different from human's

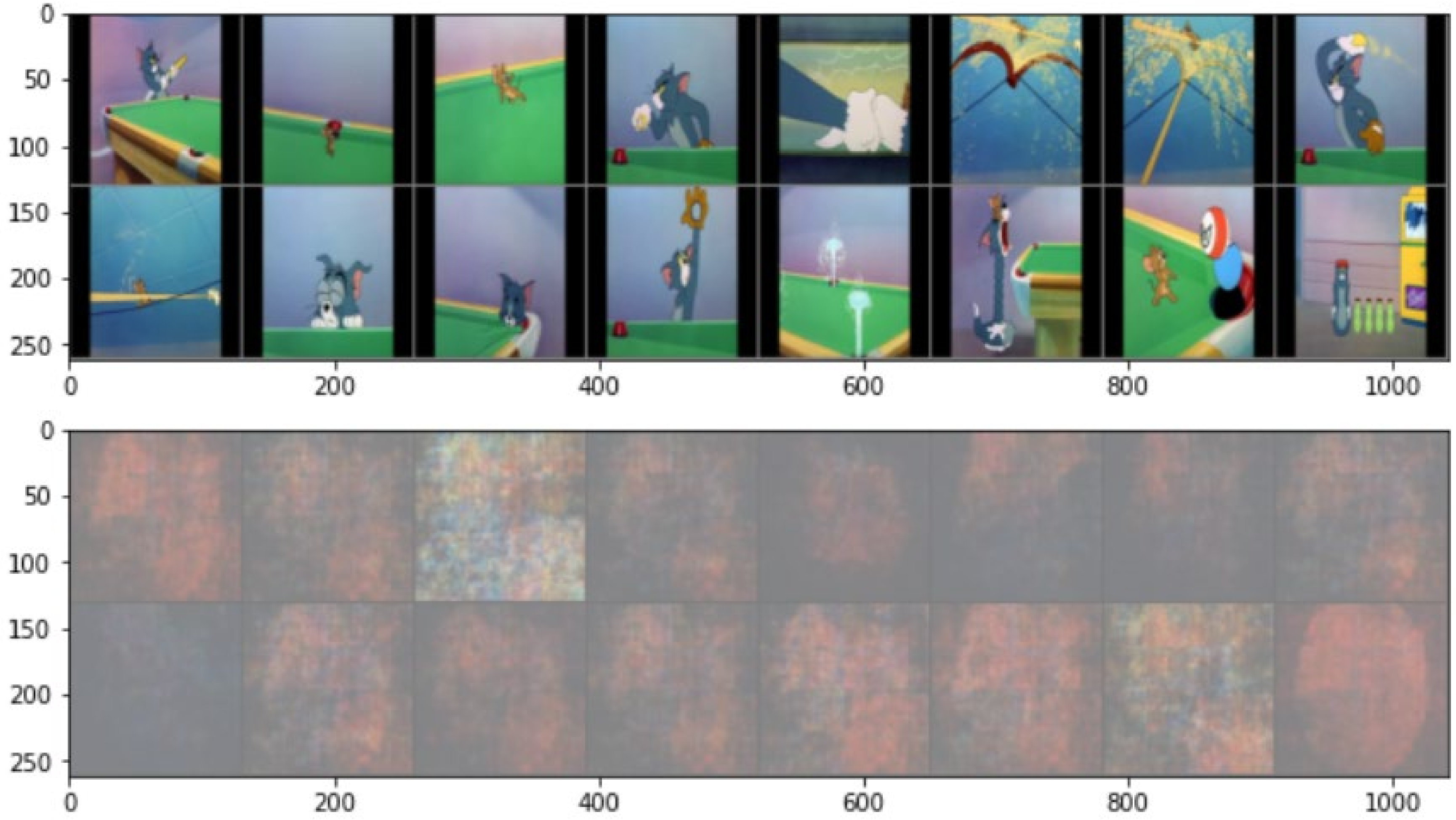




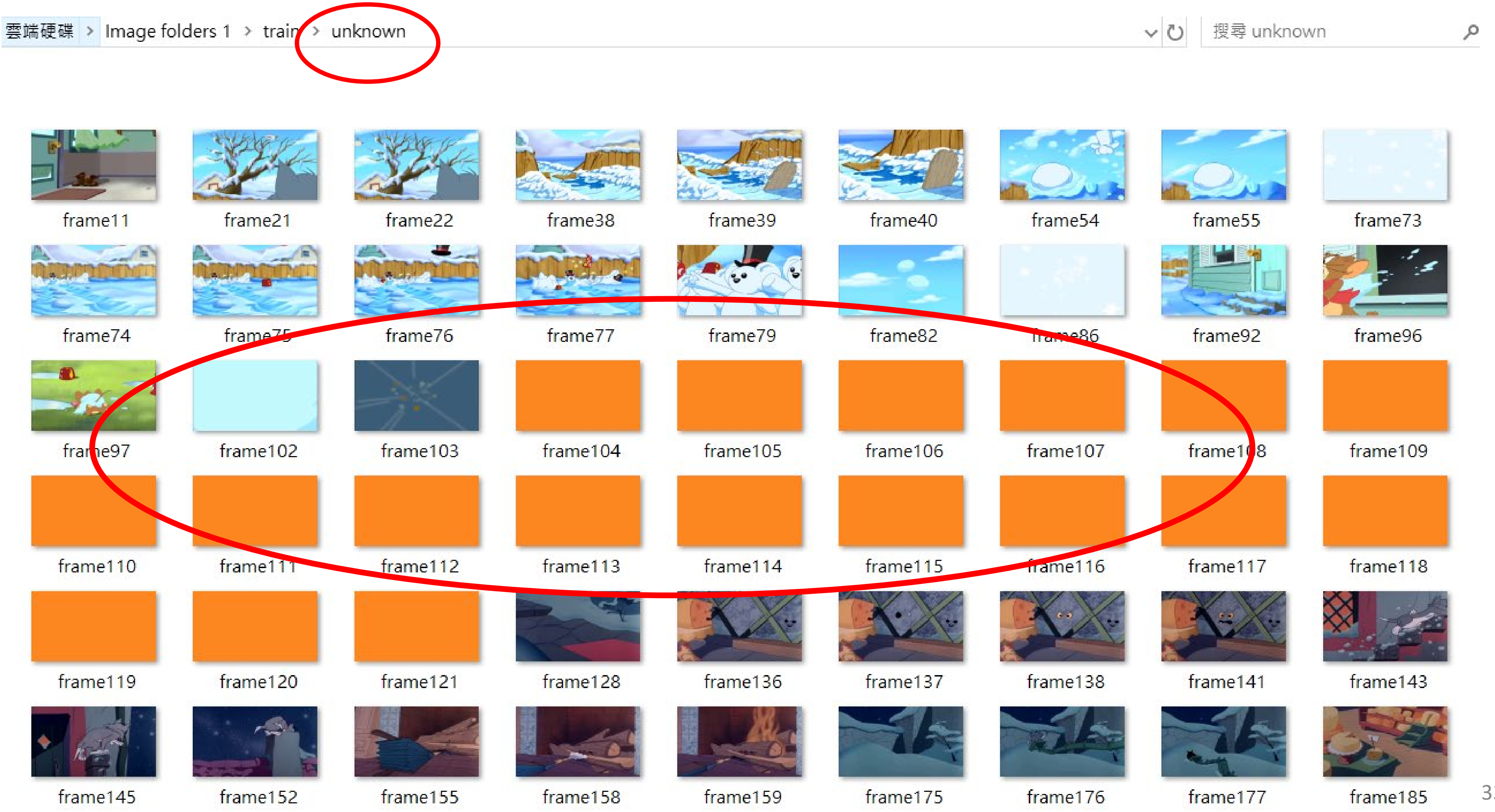
More on using AE on Tom & Jerry images

Results are still not good after 1200 epochs

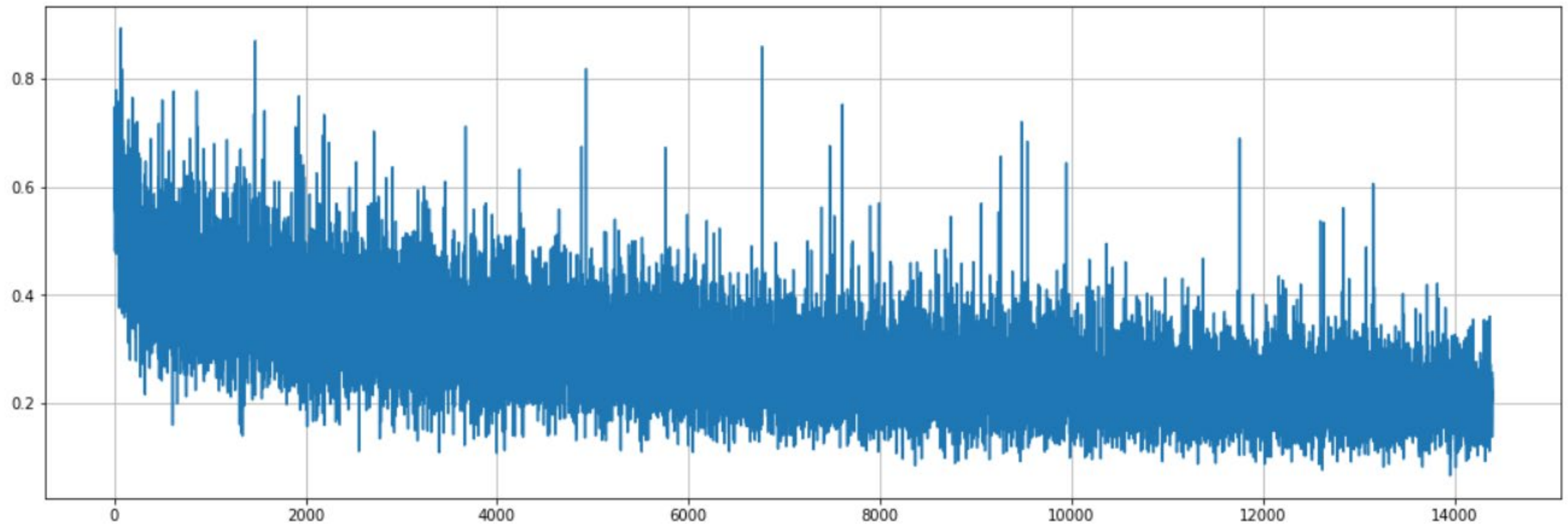
Test on un-seen images – fails to reconstruct the input images

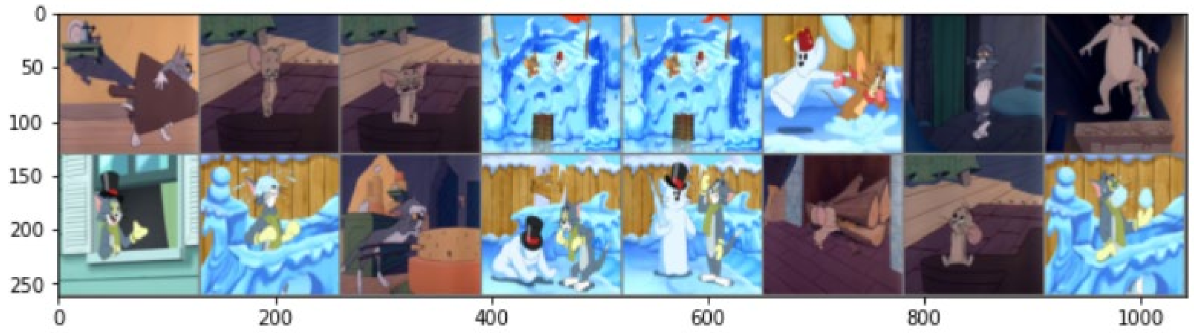


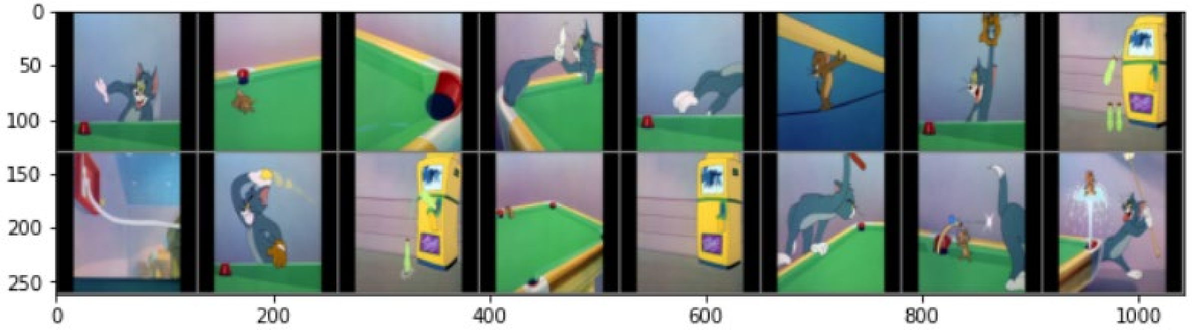
Remove the "unknown" folder?

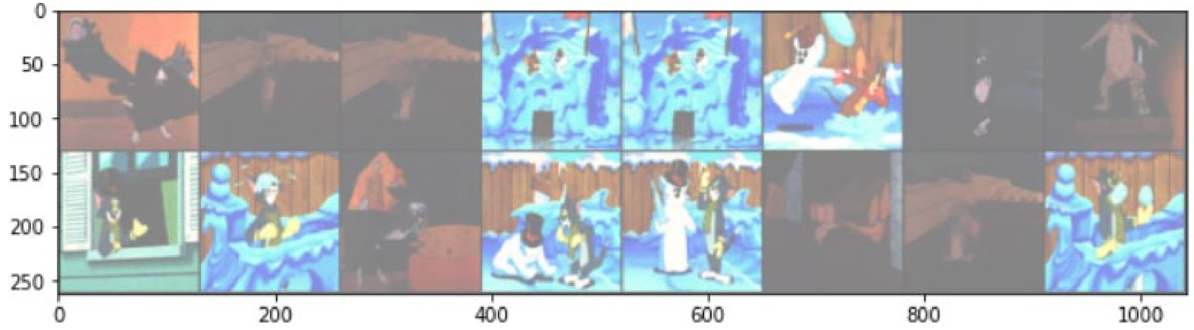


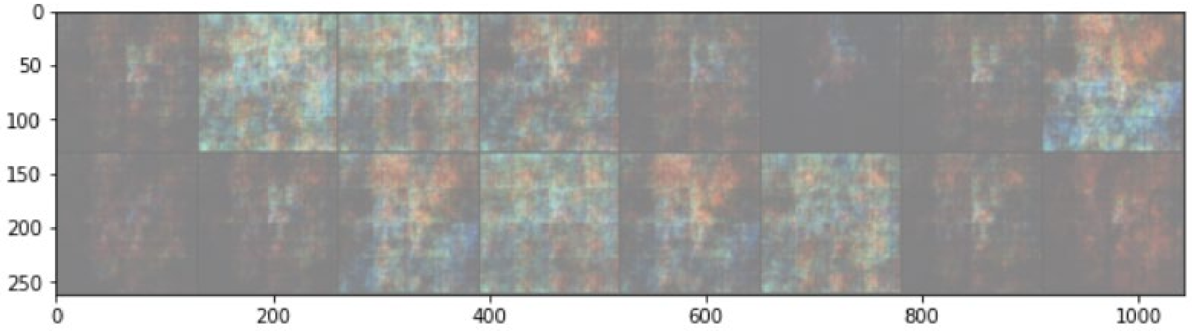
Train 1200 epochs after removing the "unknown" folder



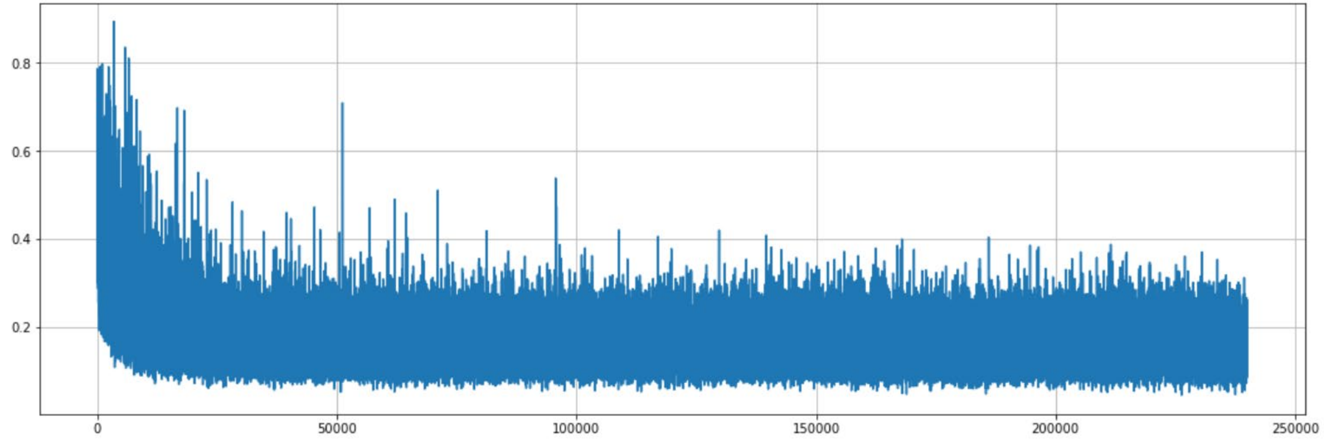




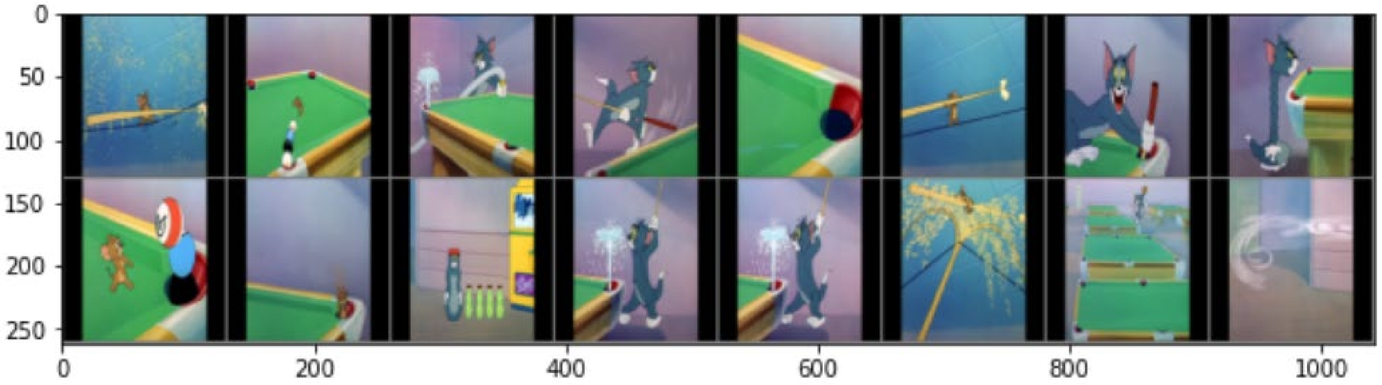


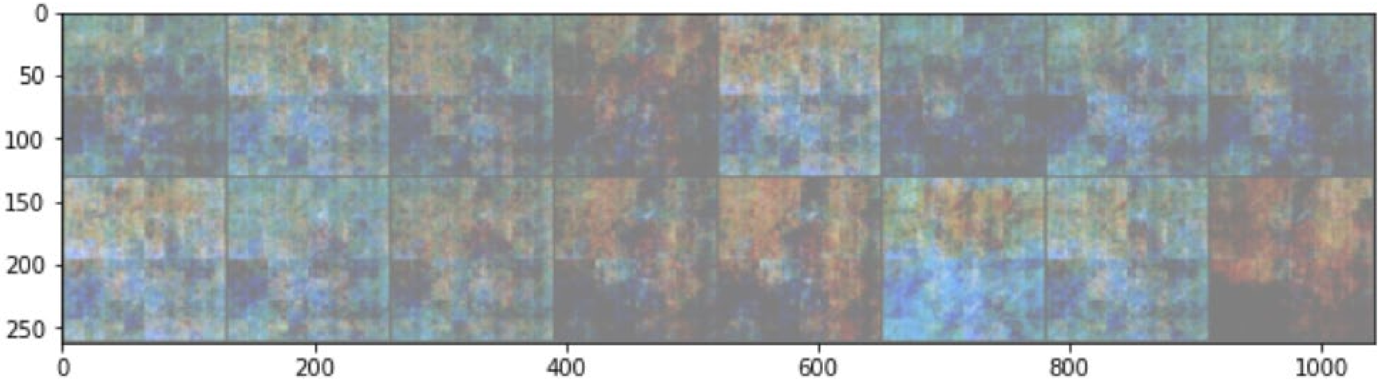


Train 20,000 epochs



1071226 陳仁慶





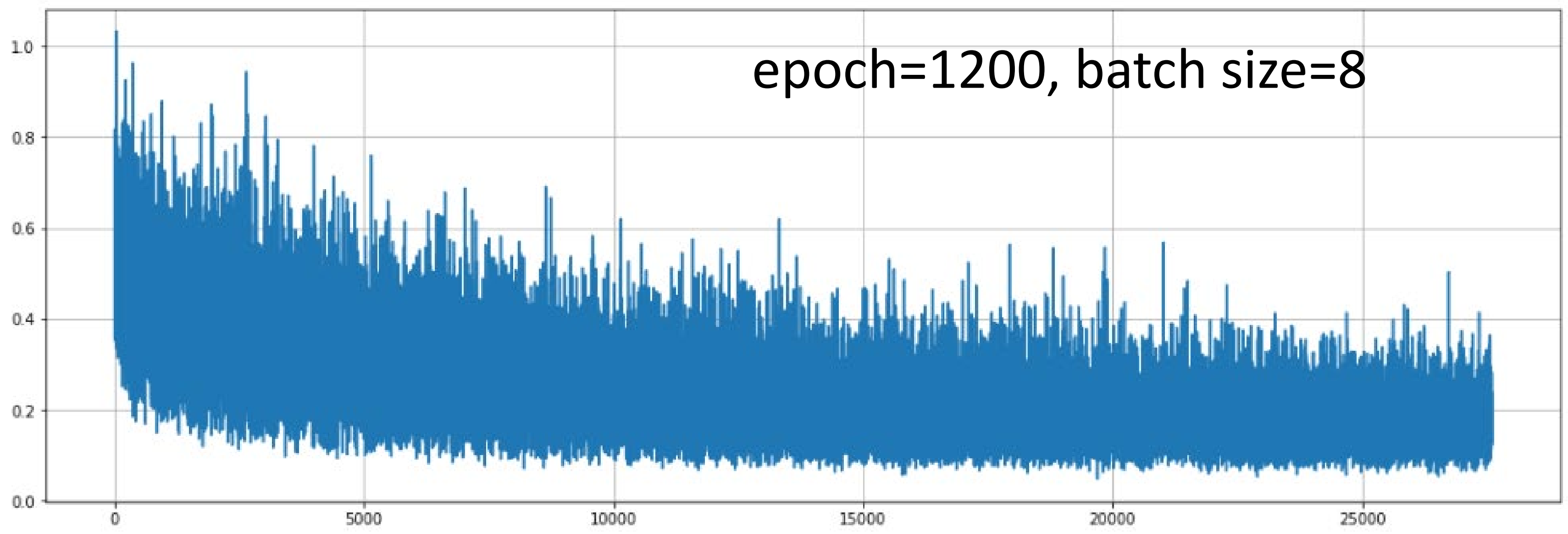
Still failed to recover un-seen images

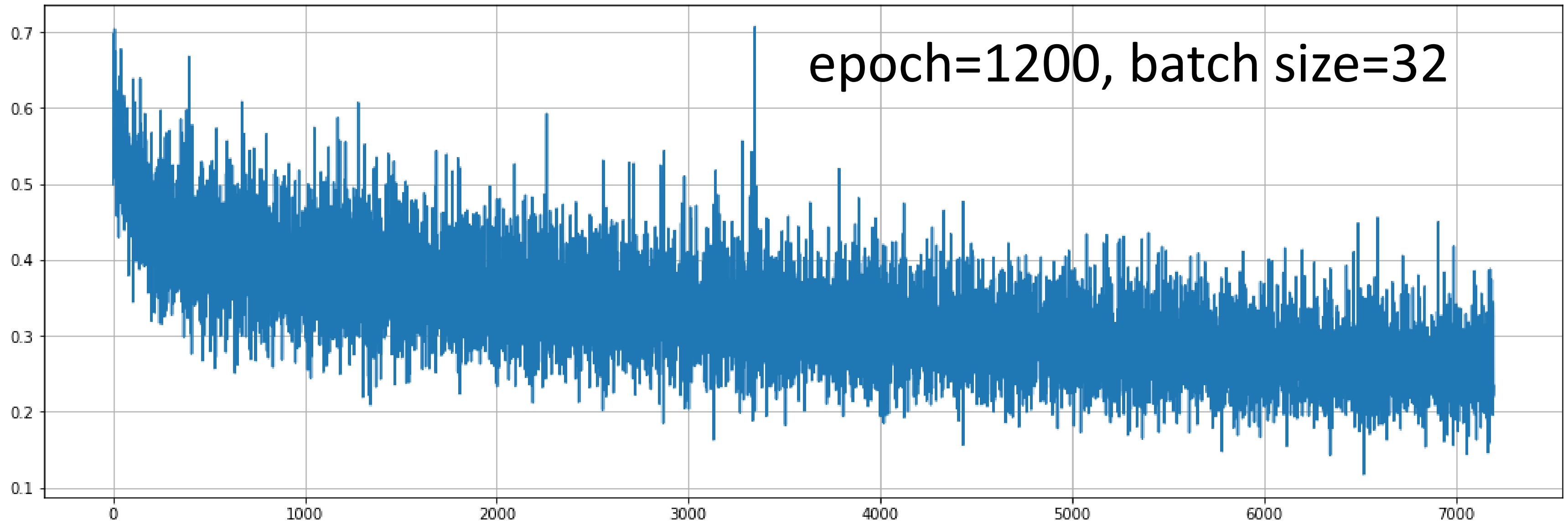
How about adjusting batch size?

```
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),
)
```

[12]:

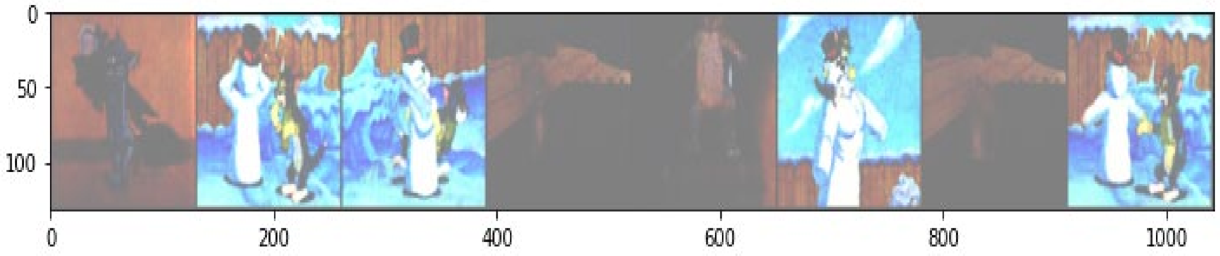
```
import torch.utils.data as Data
loader = Data.DataLoader(
    dataset=train_dataset,
    batch_size=16,
    shuffle=True)
```


epoch=1200, batch size=8
1.0
0.8
0.6
0.4
0.2
0.0
0
5000
10000
15000
20000
25000

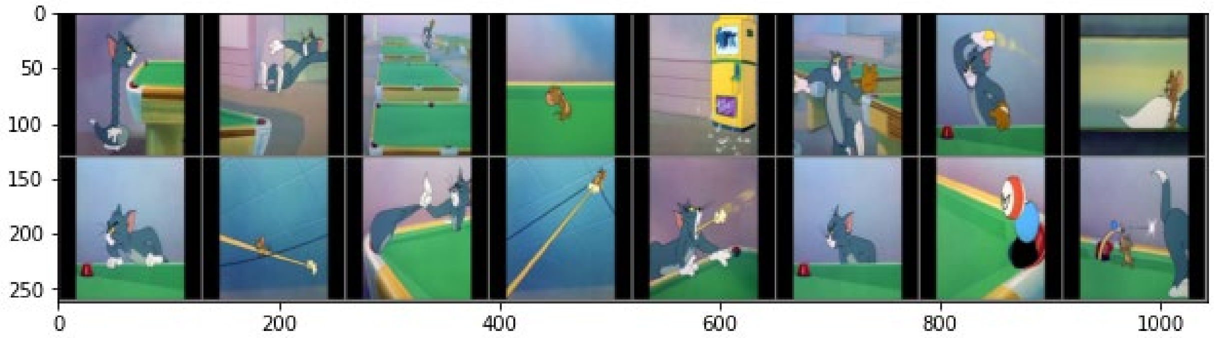

epoch=1200, batch size=32
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1000
2000
3000
4000
5000
6000
7000

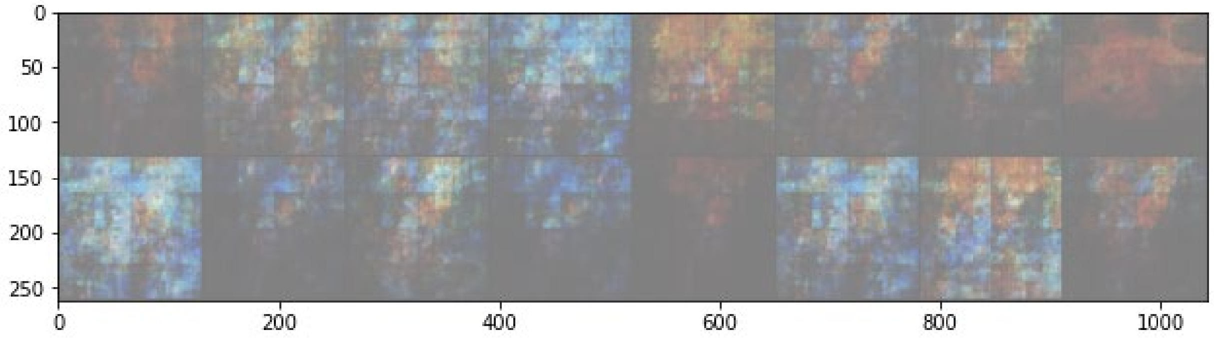
epoch=1200, batch size=8

Train:



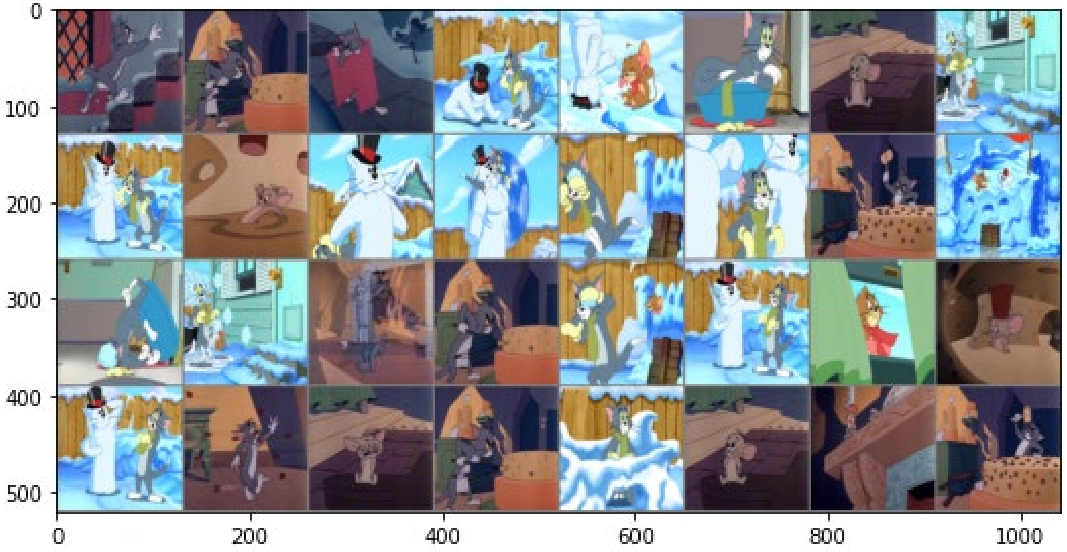
Test:





epoch=1200, batch size=32

Train:



Test: