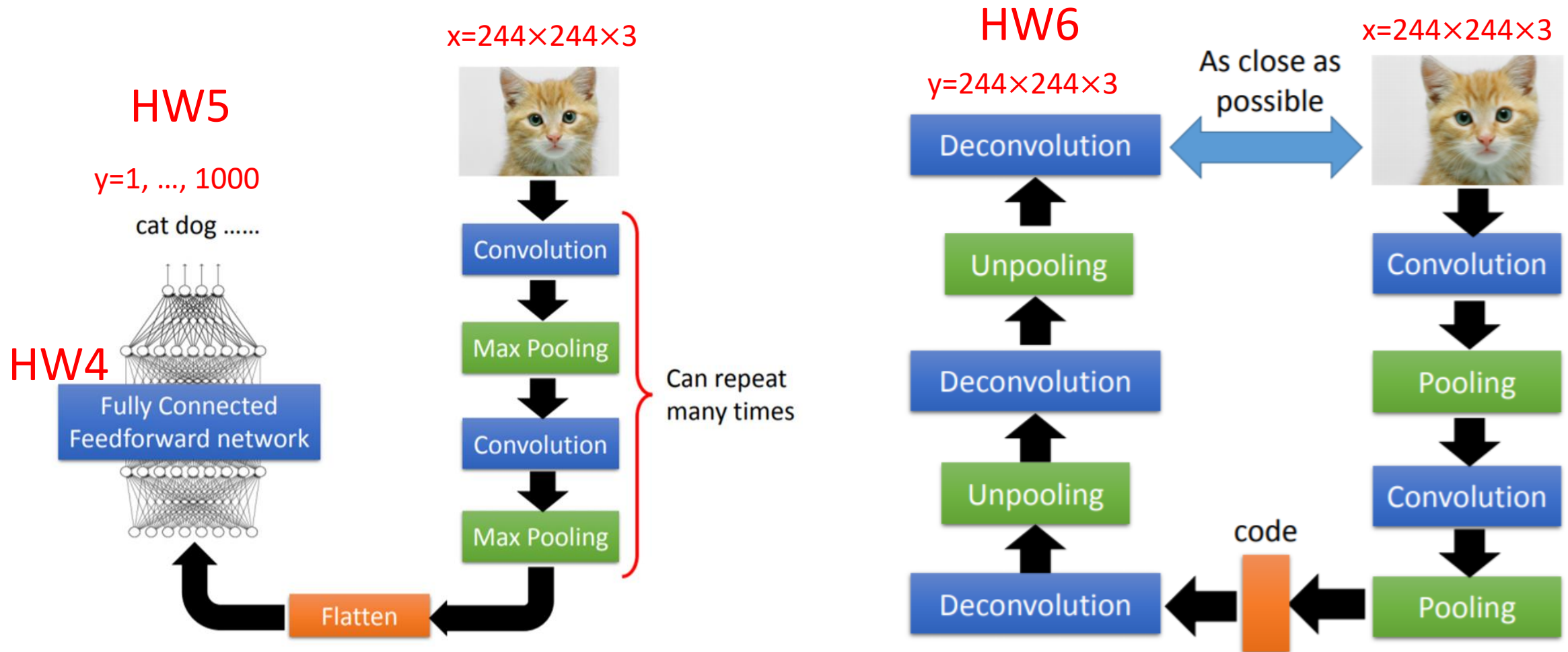
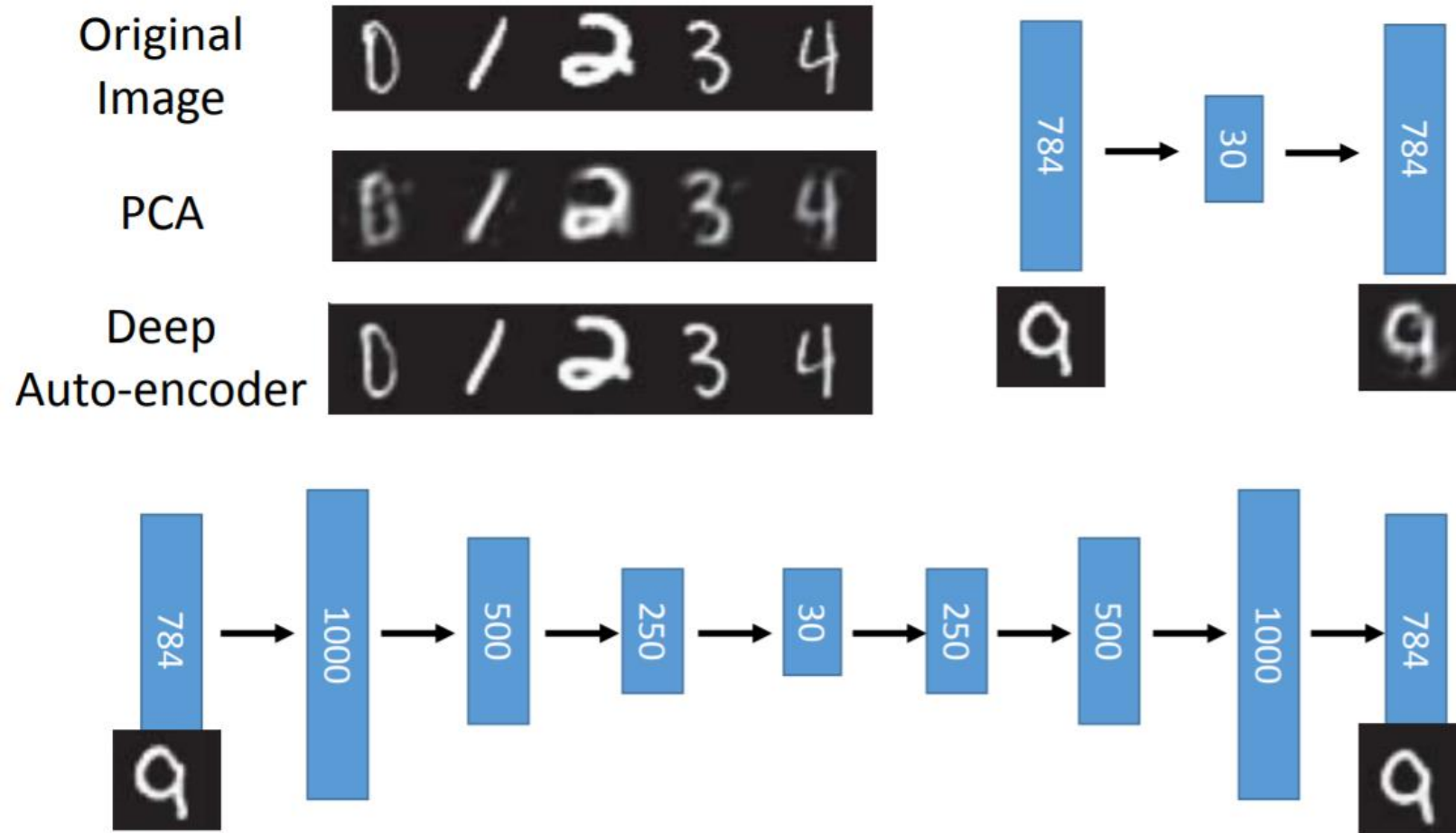


Auto-encoder

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

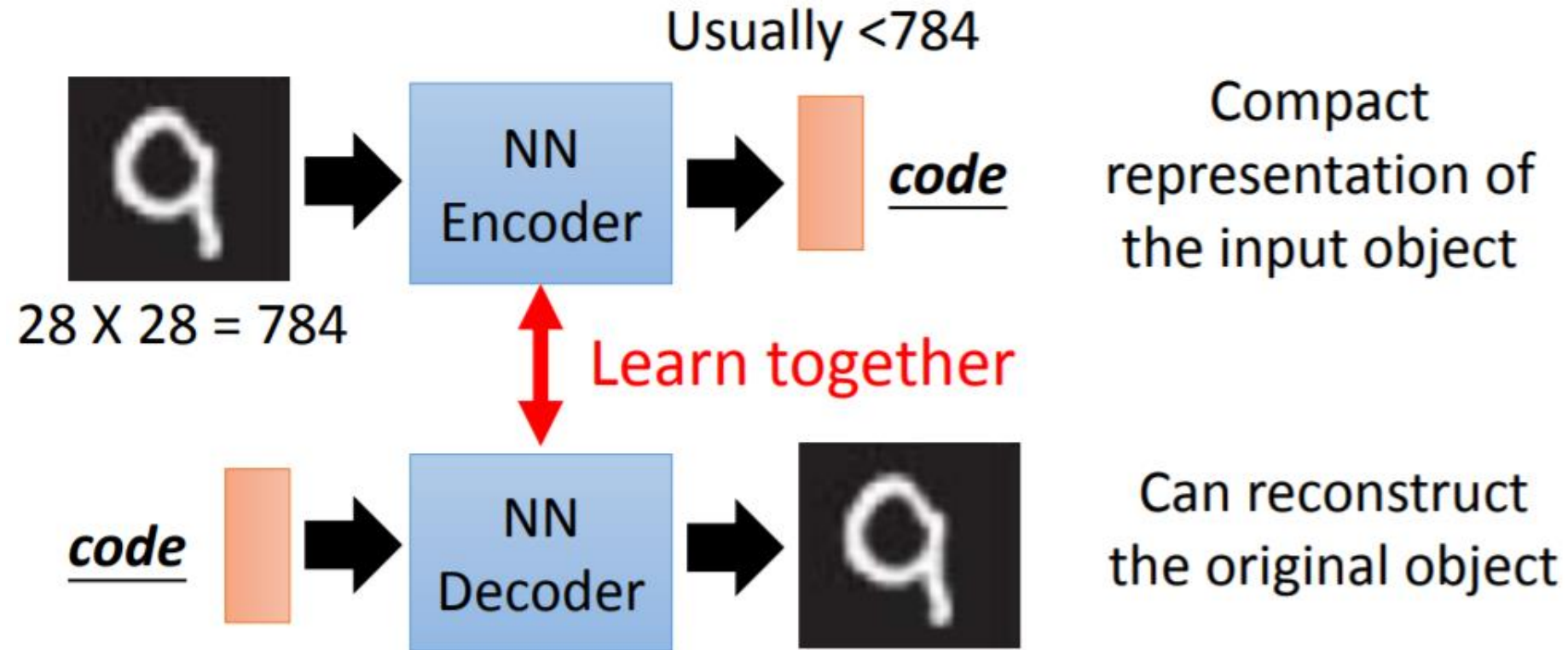


MLP based autoencoder



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

Autoencoder learns a compact representation of the input image

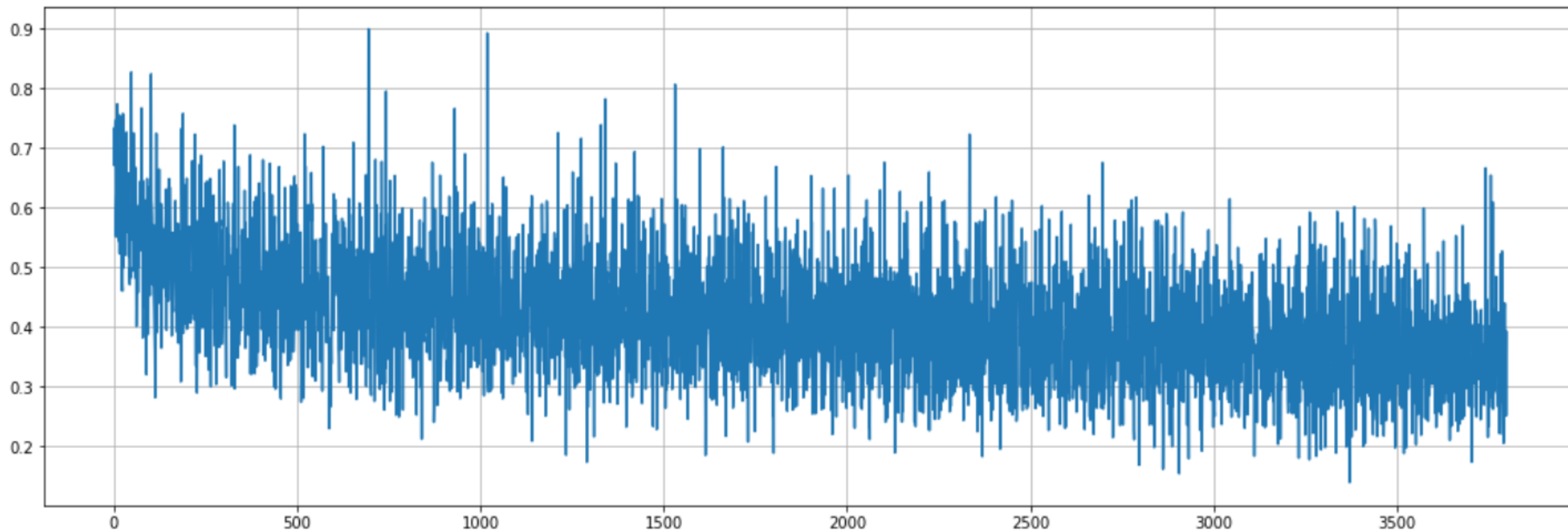


Practice

- Run "7.1.Conv_AE.ipynb"

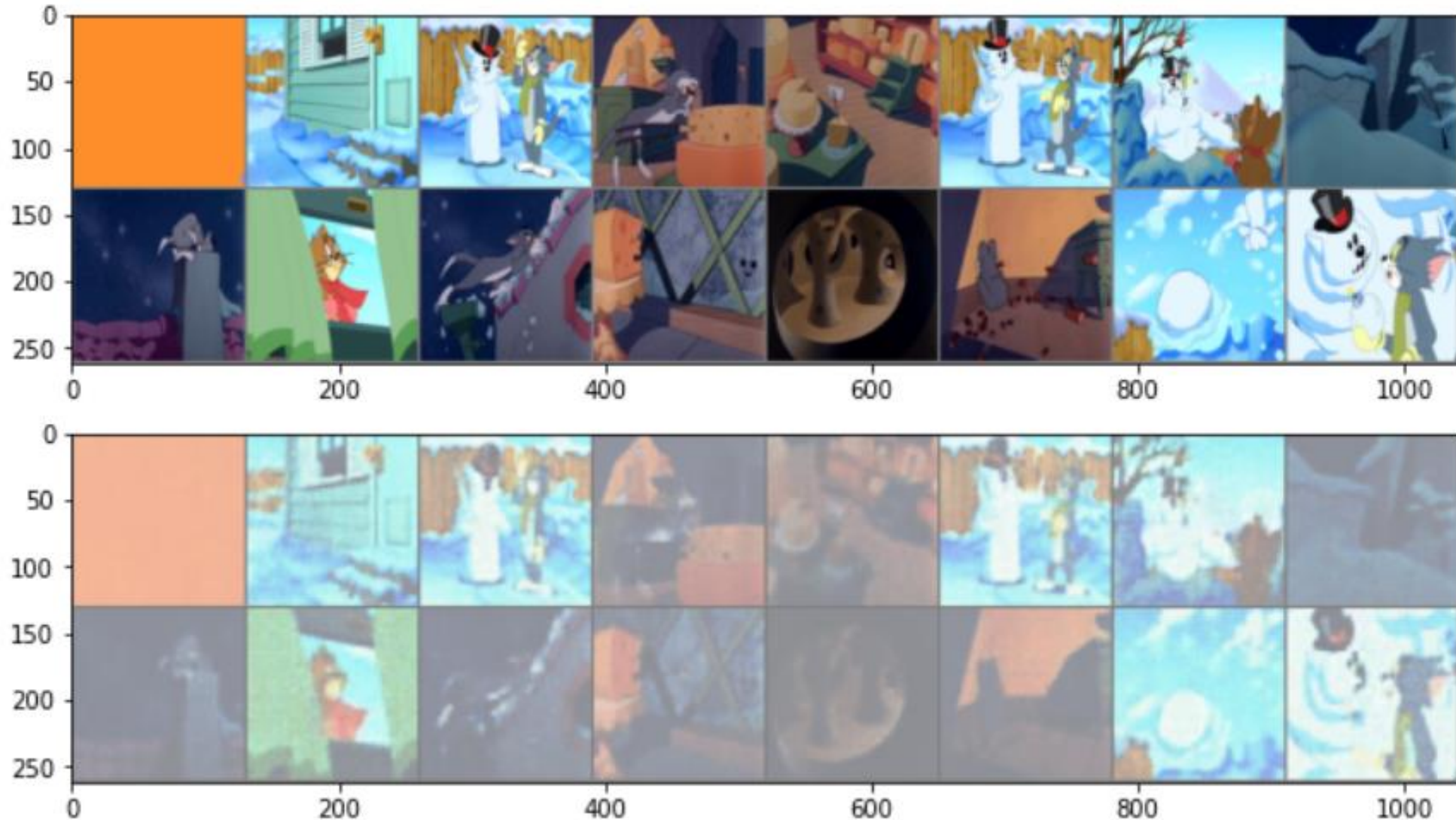


Train 200 epochs



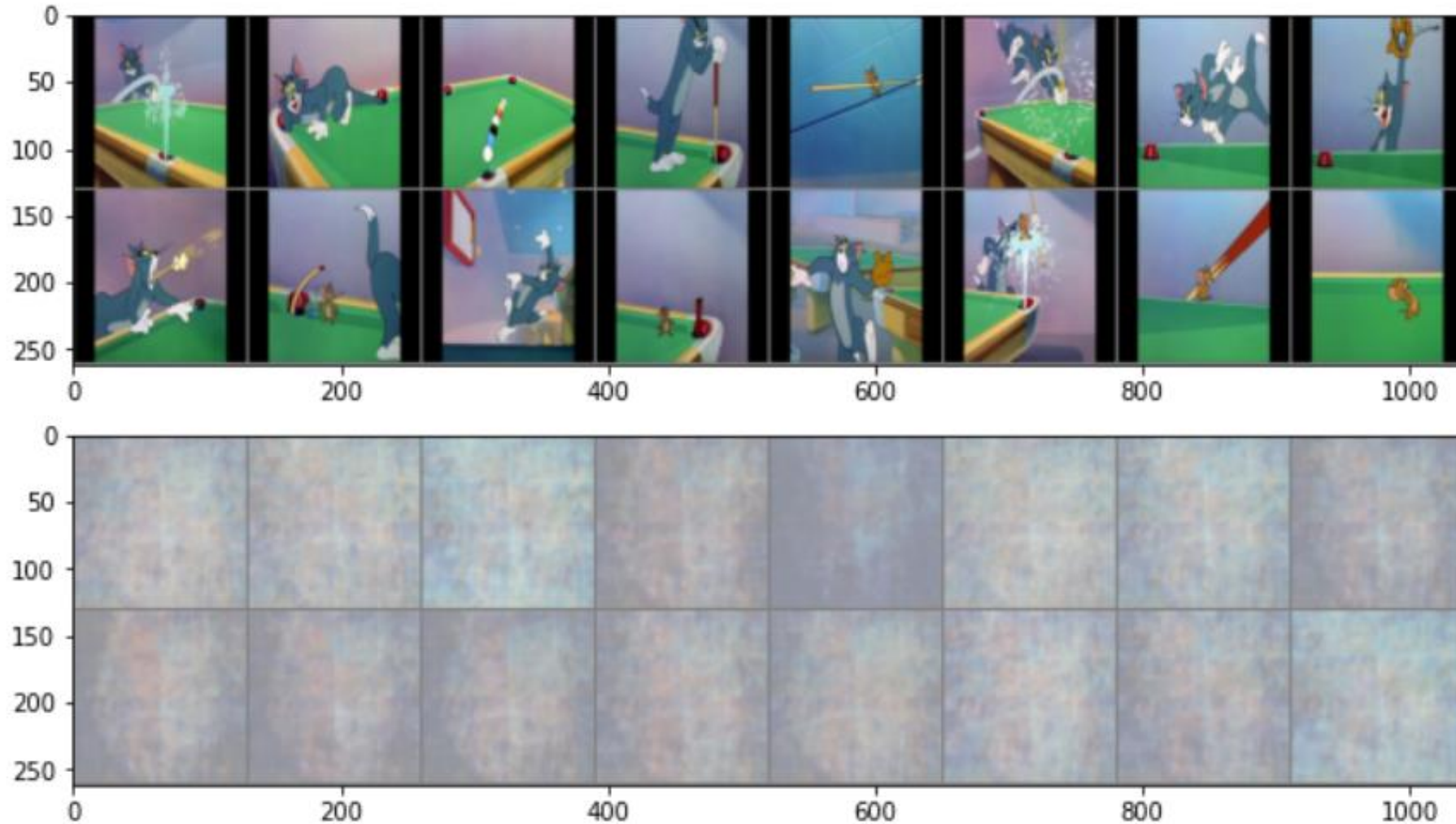
Train 200 epochs

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

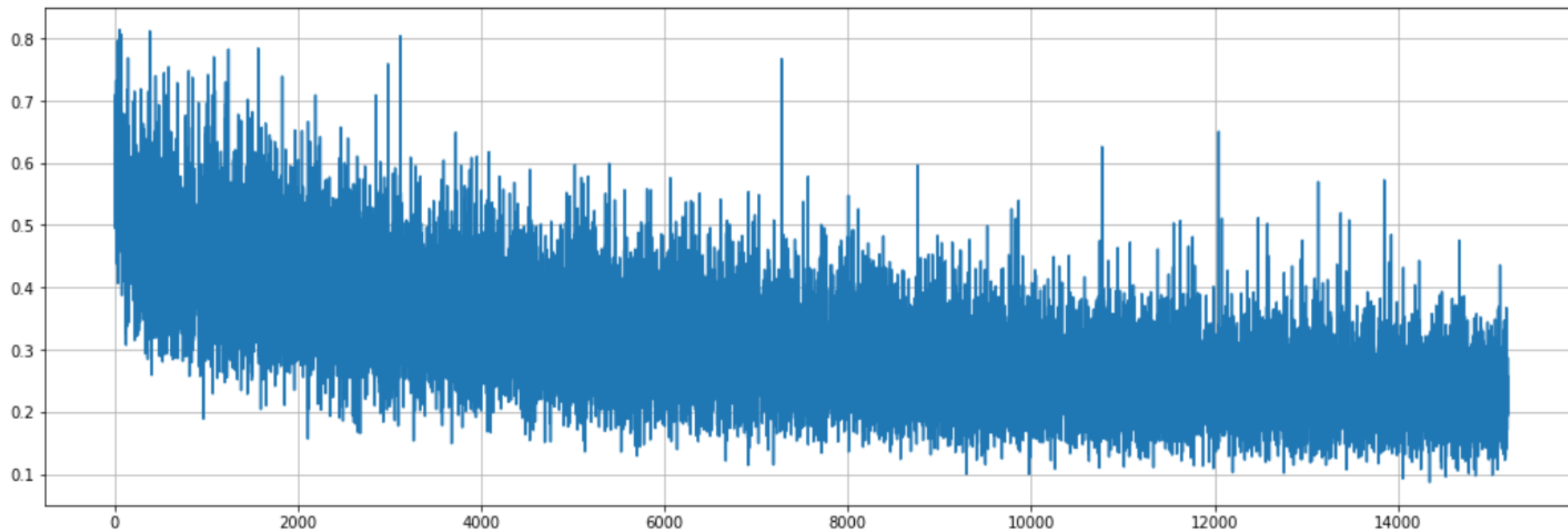


Train 200 epochs

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

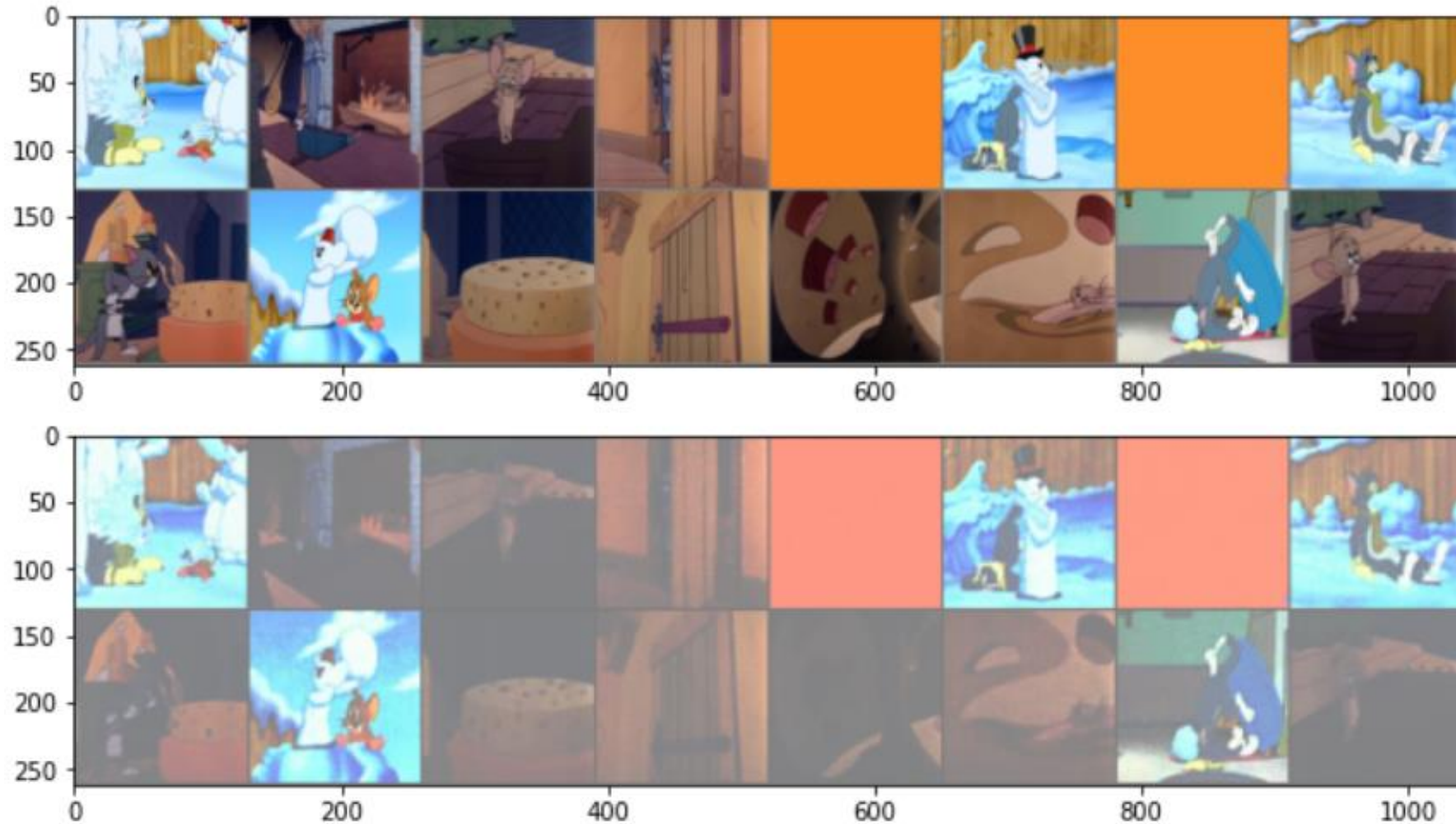


Train 800 epochs



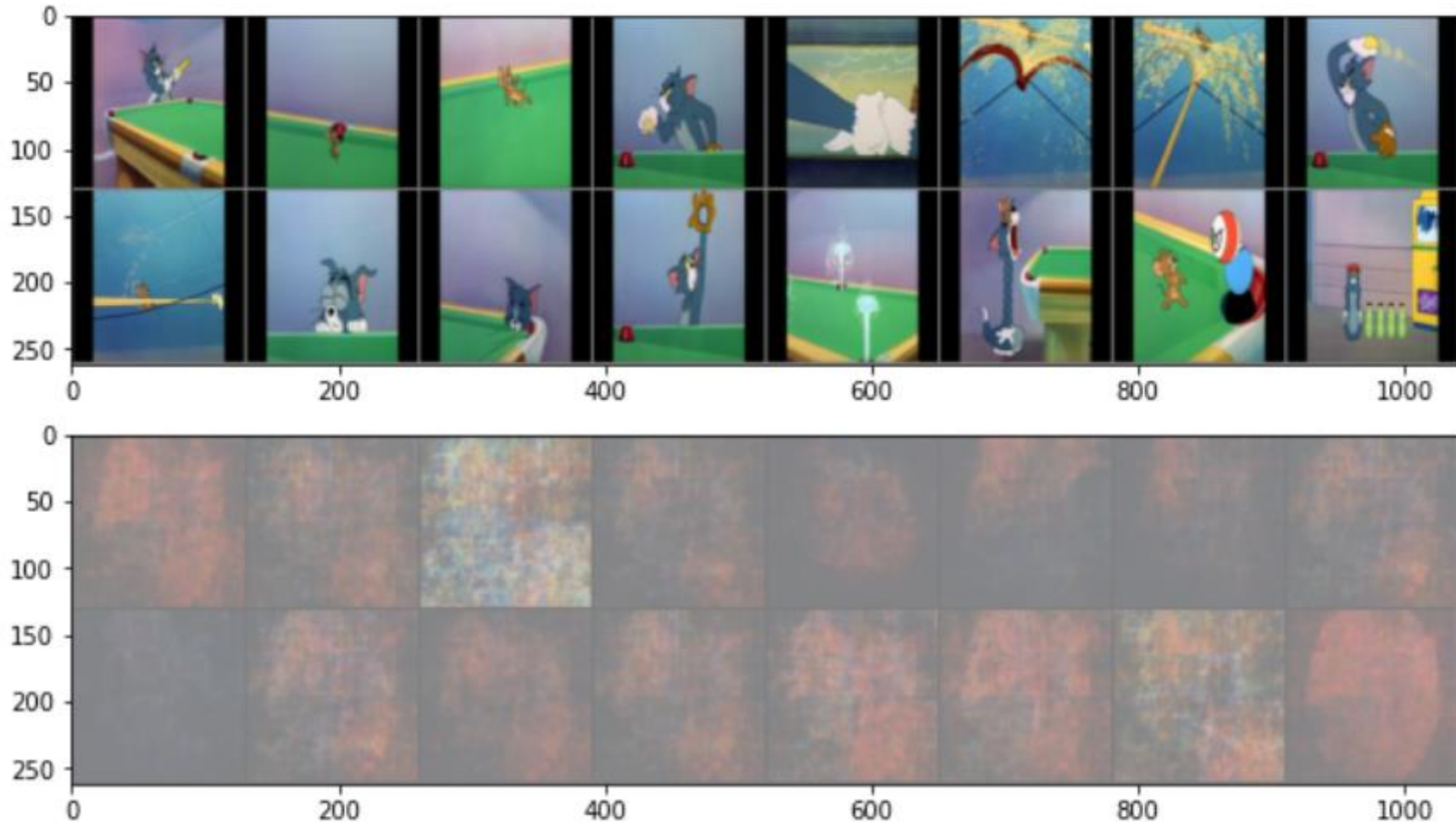
Train 800 epochs

Test on training images

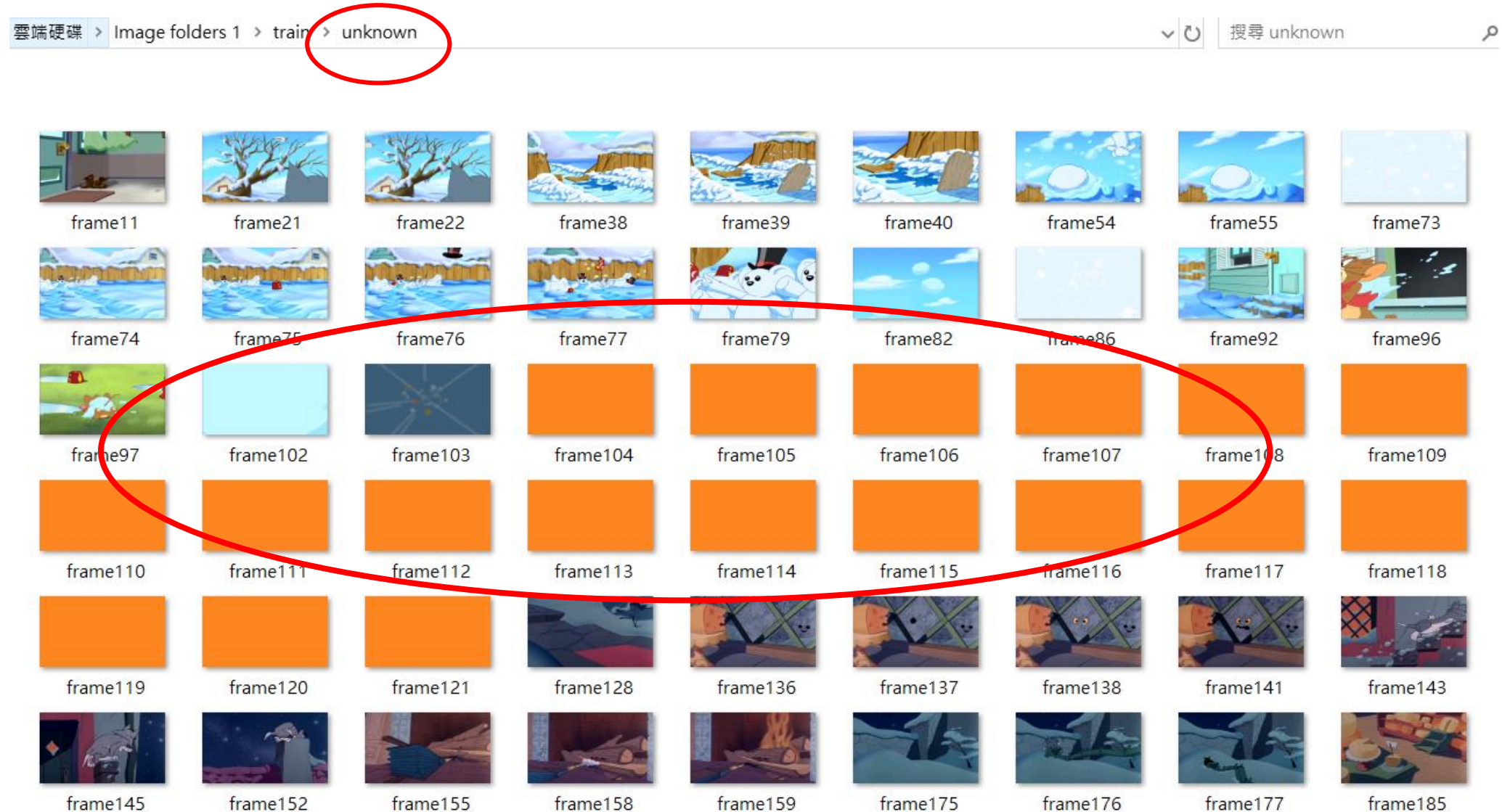


Train 800 epochs (same results when train 1200 epochs)

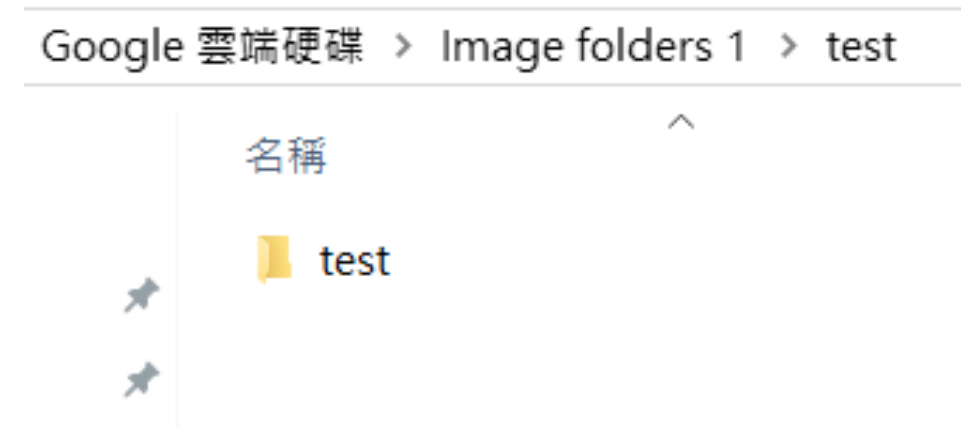
Test on un-seen images – fails to reconstruct the input images



Your AI model is as good as your data, and as bad as your data too

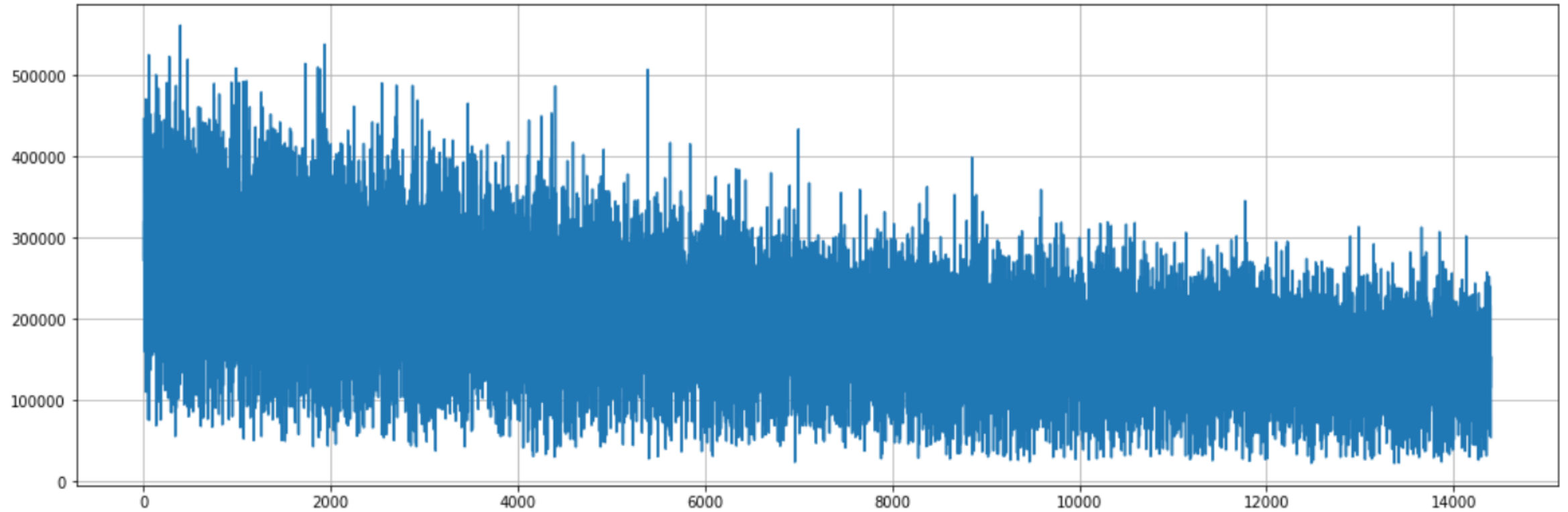


Create a new folder "Image folder 1", remove the "Unknown" sub-folder

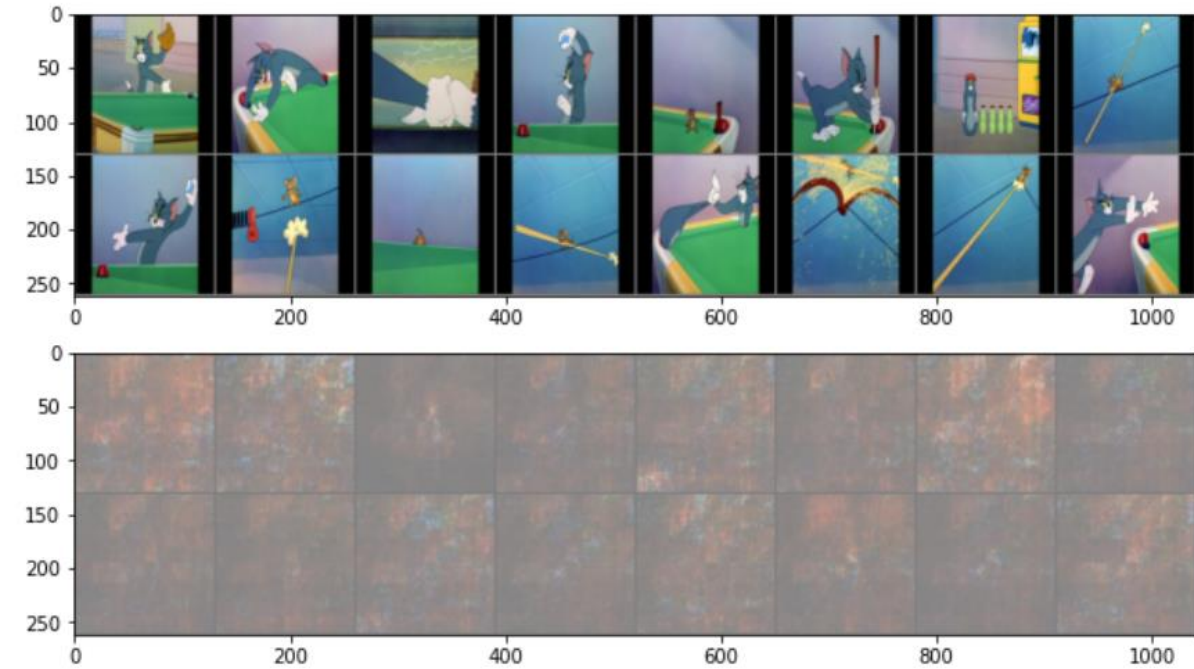
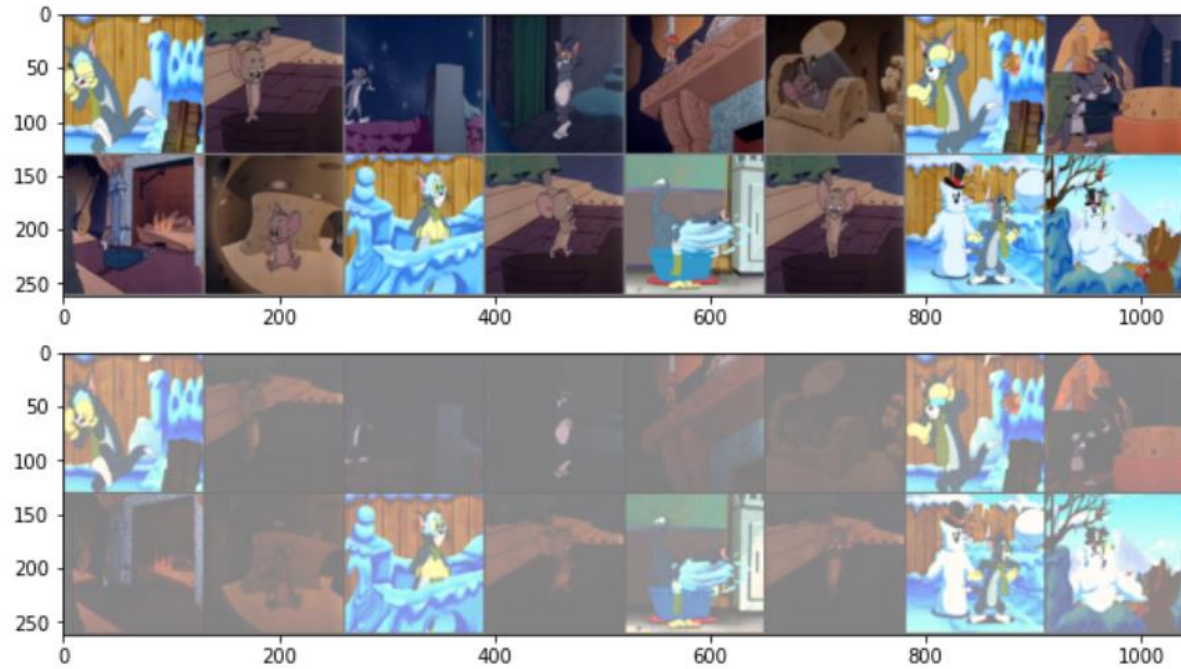


Test folder remains the same

Train 1200 epochs after removing the "unknown" folder

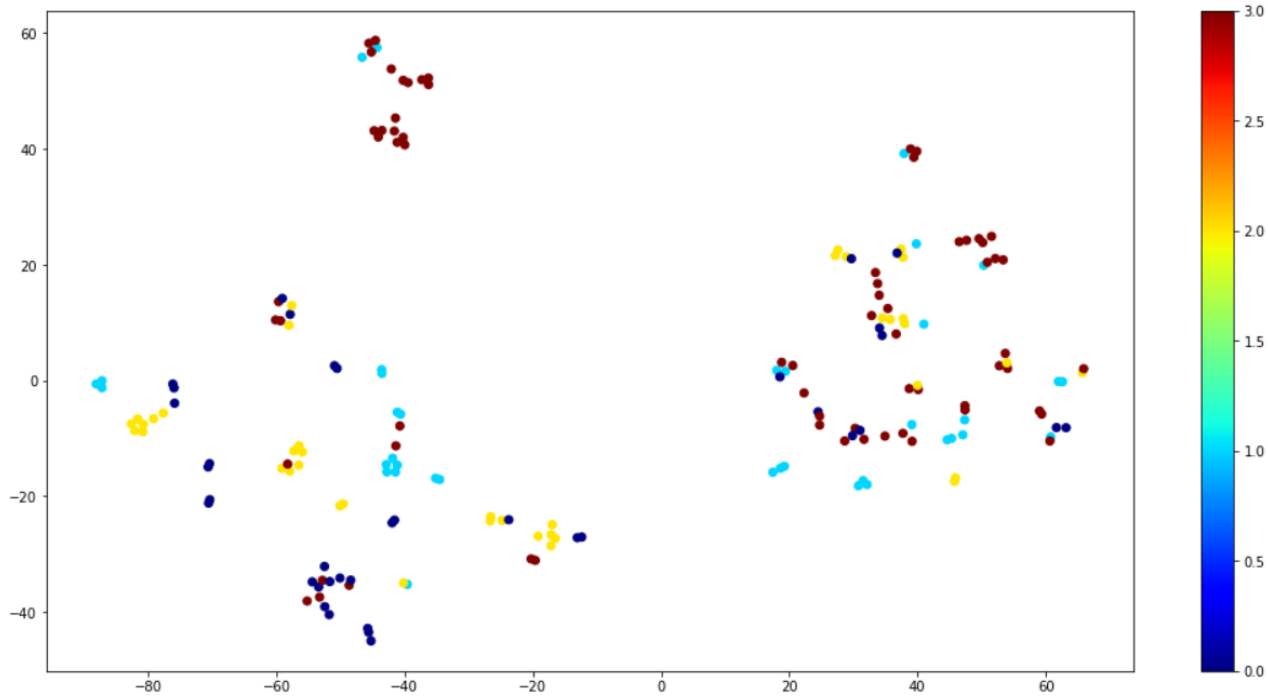


Train 1200 epochs after removing the "unknown" folder



Train 1200 epochs after removing the "unknown" folder

perplexity = 5



perplexity = 10, 30, 50

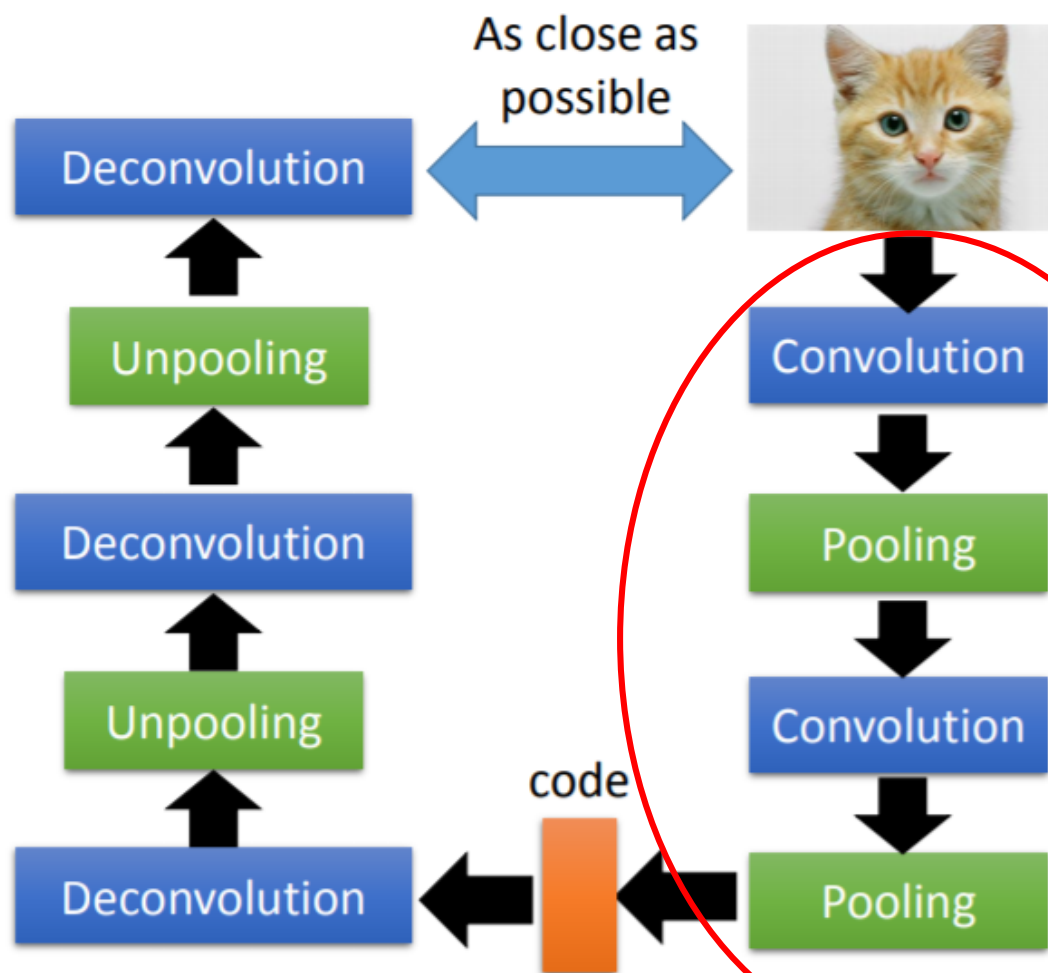
How about batch size? Increase or decrees ?

```
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=32 or 16

Encoder



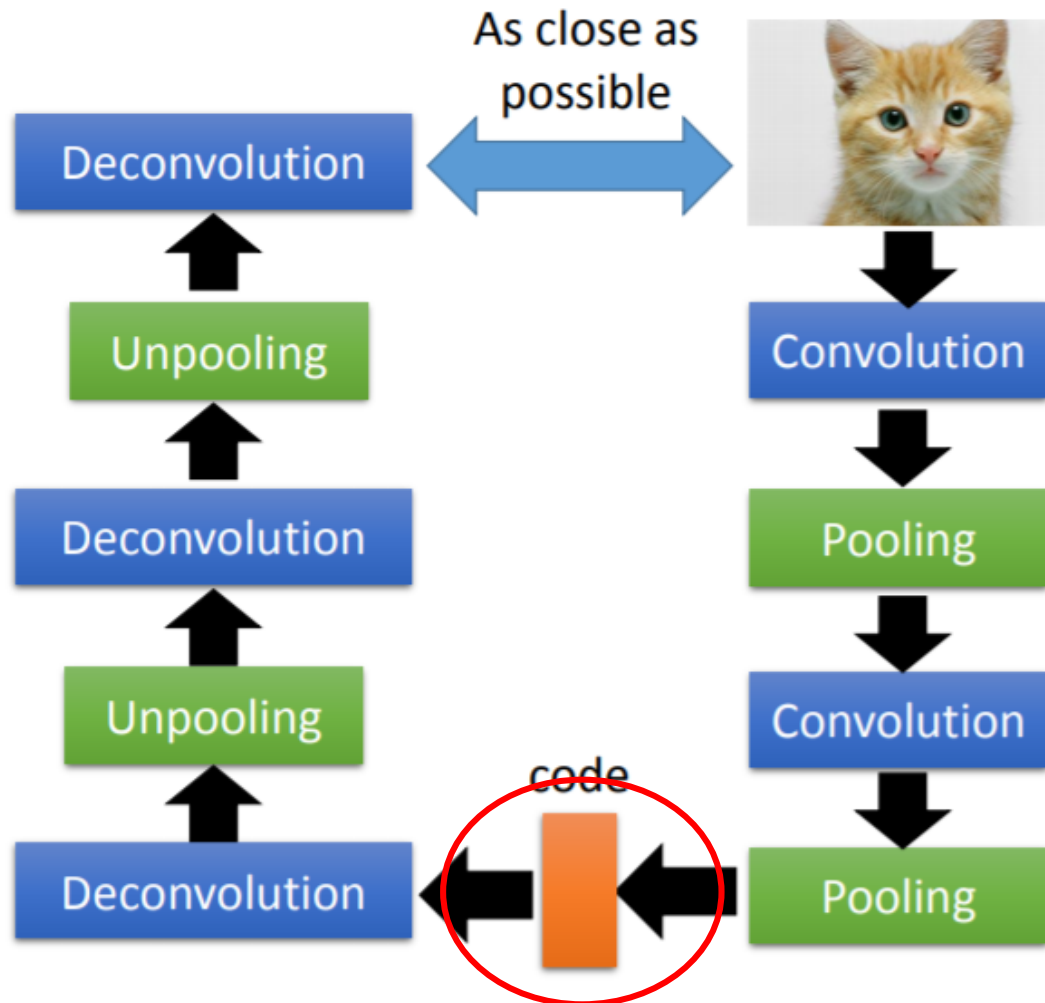
```
self.encoder = nn.Sequential(  
    nn.Conv2d(3, 32, kernel_size=2, stride=2),  
    nn.BatchNorm2d(32, eps=1e-05, momentum=0.1, af  
    nn.ReLU(),  
    nn.Conv2d(32, 64, kernel_size=2, stride=2),  
    nn.BatchNorm2d(64, eps=1e-05, momentum=0.1, af  
    nn.ReLU(),  
    nn.Conv2d(64, 128, kernel_size=2, 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),  
)
```

Practice: Draw the feature maps of encoder

- Let input image = $224 \times 224 \times 3$
- Draw the feature maps (H, W, depth) after each convolution and max pooling
- What is the number of nodes after flatten?



Latent vector

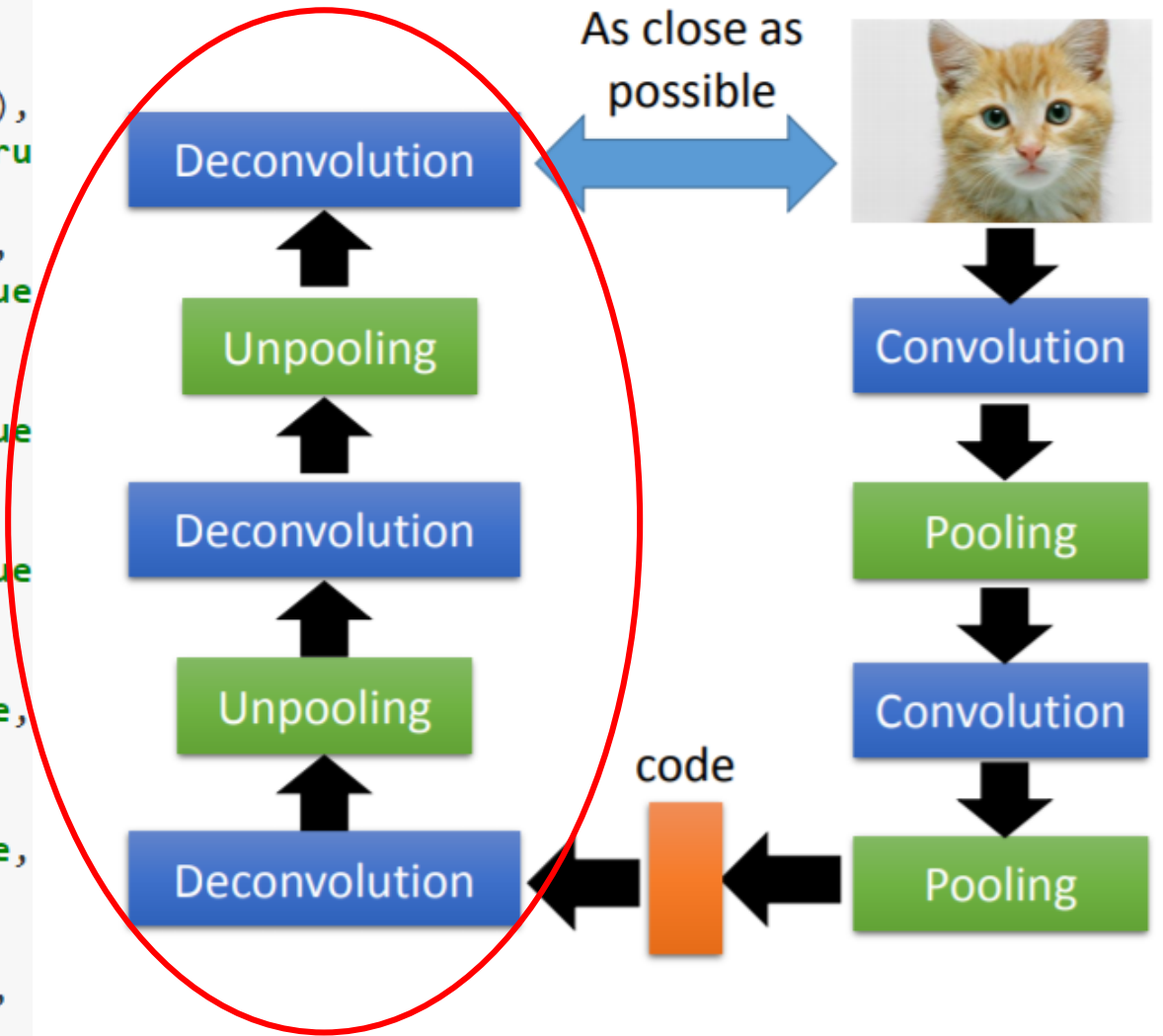


Flatten-22
Linear-23
Linear-24
UnFlatten-25

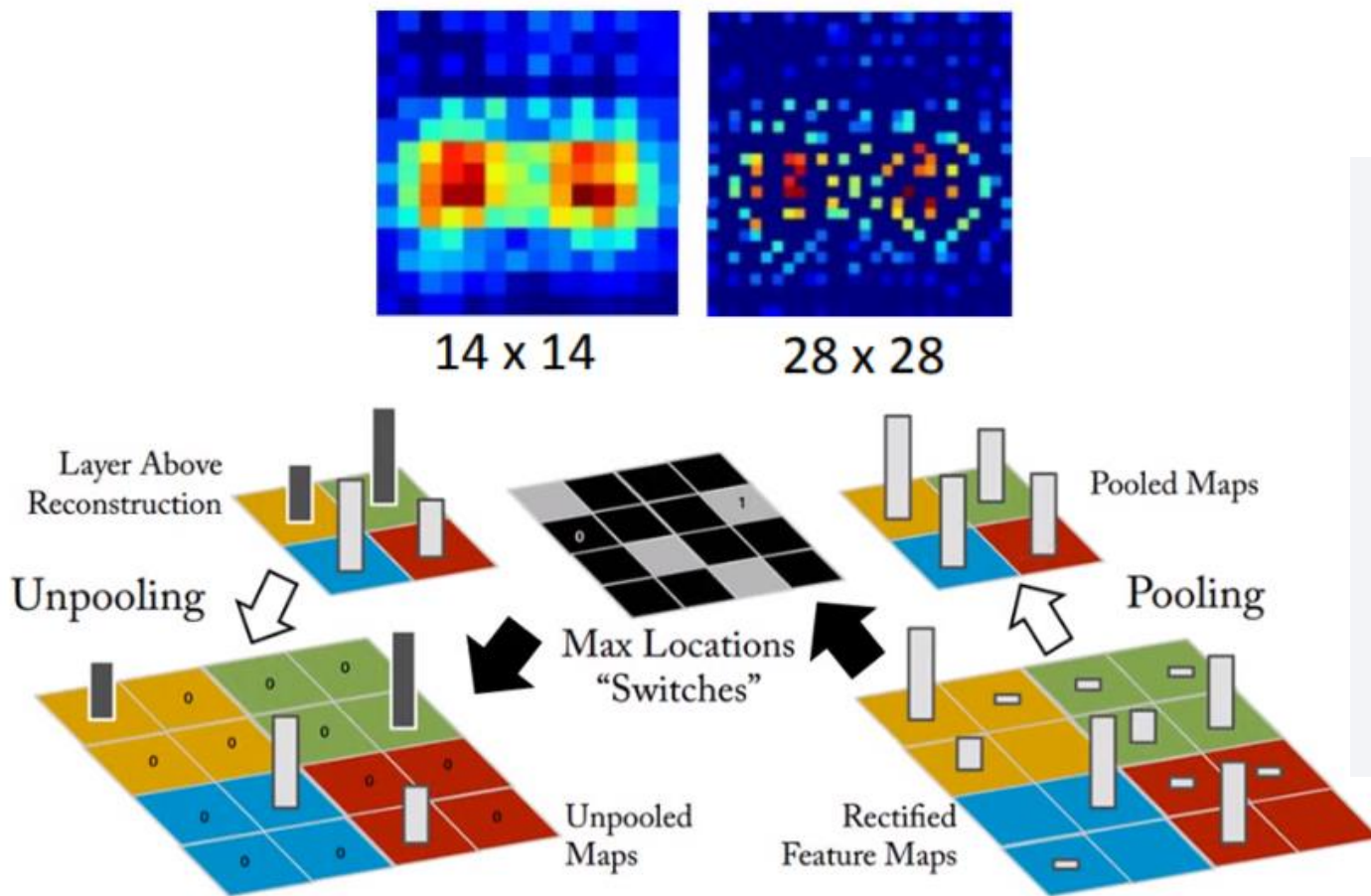
$[-1, 1024]$
 $[-1, 64]$
 $[-1, 1024]$
 $[-1, 1024, 1, 1]$

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=True),  
    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



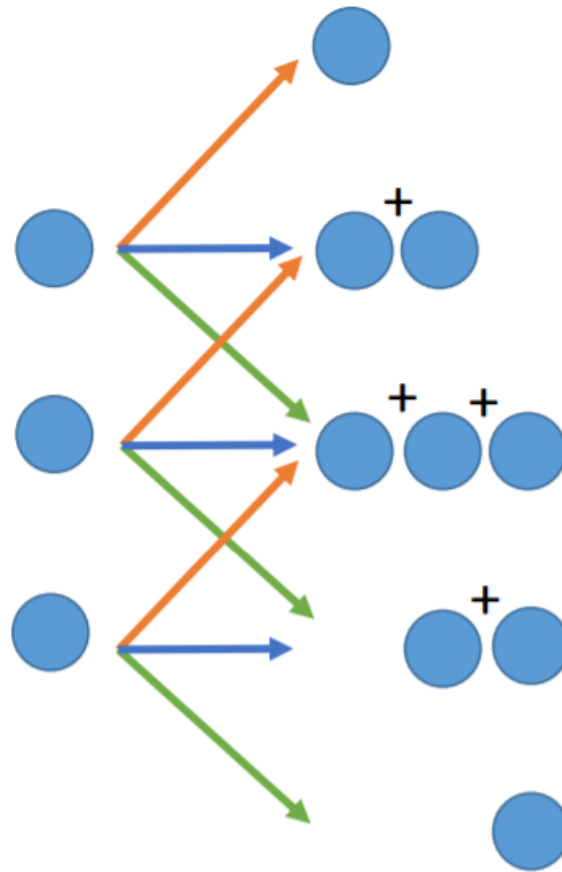
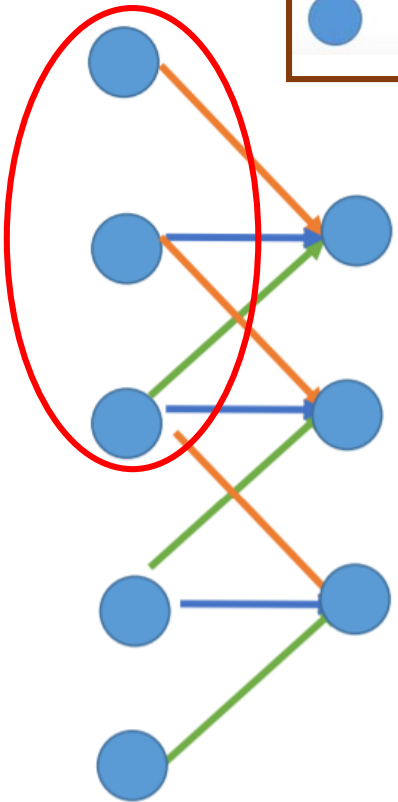
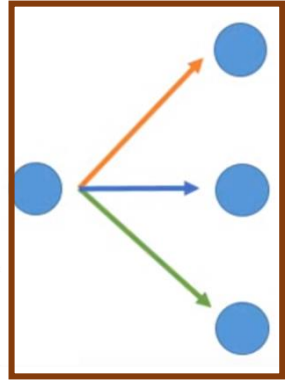
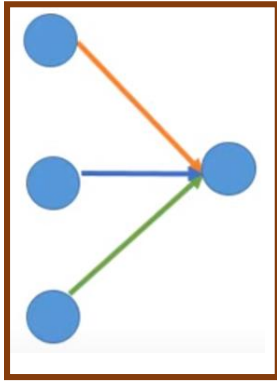
```
>>> pool = nn.MaxPool2d(2, stride=2, return_indices=True)
>>> unpool = nn.MaxUnpool2d(2, stride=2)
>>> input = torch.tensor([[[[ 1.,  2,  3,  4],
                             [ 5,  6,  7,  8],
                             [ 9, 10, 11, 12],
                             [13, 14, 15, 16]]]])

>>> output, indices = pool(input)
>>> unpool(output, indices)
tensor([[[[ 0.,  0.,  0.,  0.],
           [ 0.,  6.,  0.,  8.],
           [ 0.,  0.,  0.,  0.],
           [ 0., 14.,  0., 16.]]]]])
```

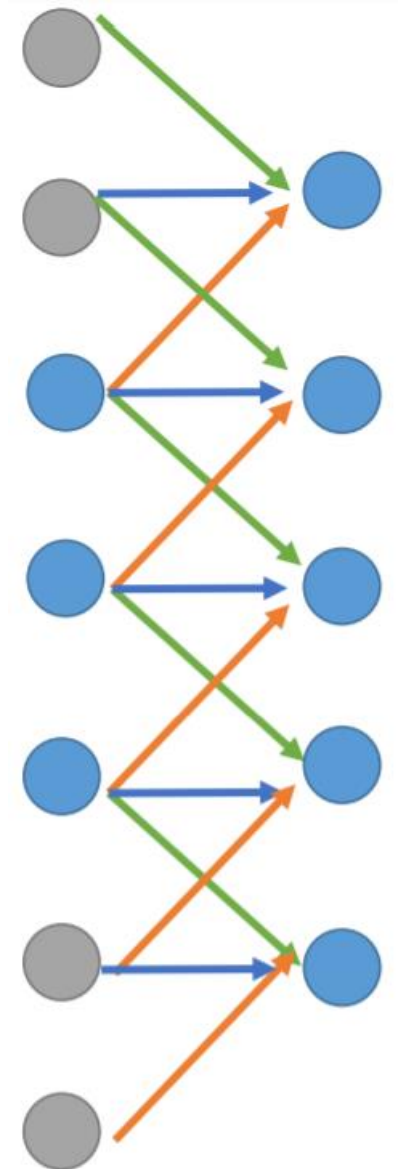
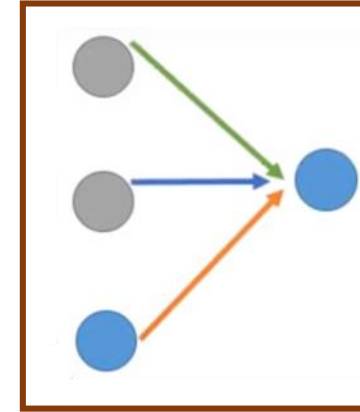
Reference: 李弘毅 ML Lecture 16 <https://youtu.be/Tk5B4seA-AU>

Deconvolution

1D convolution, $k=3$ 1D deconvolution, $k=3$



1D convolution, $k=3$



In this ConvAE example, we only use deconvolution for up sampling, no un-pooling is 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=True),  
    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: Draw the feature maps of decoder

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

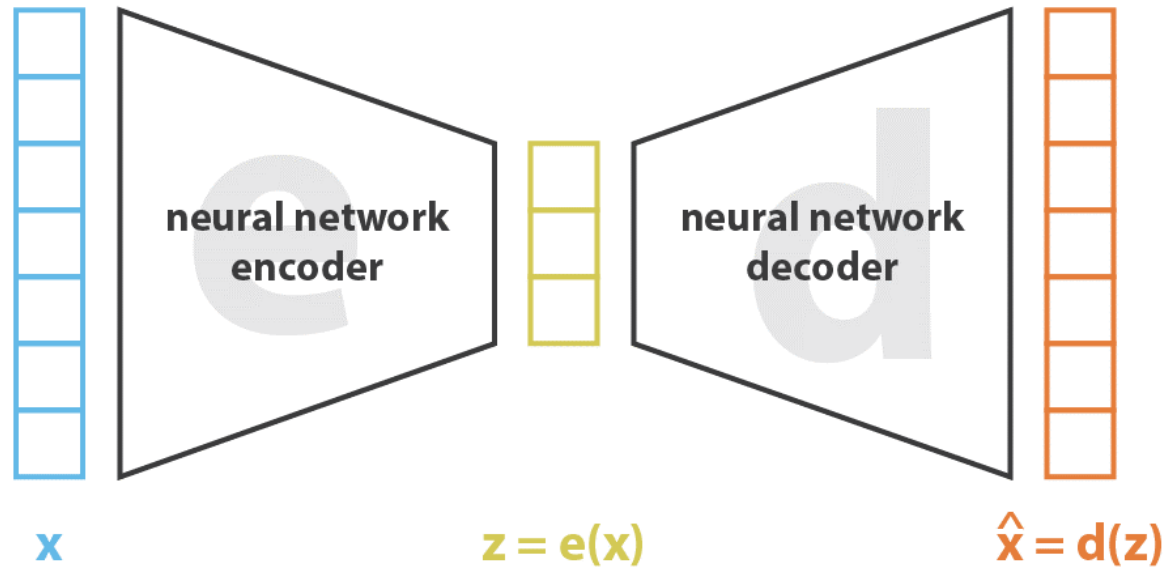


Deconvolution

```
(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
```

HW6 (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 to see whether they form clusters.



Save and load PyTorch model

```
[27]: torch.save(model.state_dict(), "AE800.pt")
```

```
[28]: model=autoencoder() #build NN architecture  
      model.load_state_dict(torch.load("AE800.pt")) #Load model weights  
      model.to(device)  
      model.eval()
```



Save and load PyTorch model



The screenshot displays a Jupyter Notebook interface. At the top, the notebook is titled "3_AlexNet_(1).ipynb". Below the title, there are tabs for "檔案" (Files), "編輯" (Edit), "檢視畫面" (View), "插入" (Insert), "執行階段" (Runtime), "工具" (Tools), "說明" (Help), and "無法儲存變更" (Cannot save changes). The "檔案" tab is active, showing a file explorer on the left. The file explorer has a search bar and icons for file operations. The file list includes "..", "gdrive", "sample_data", "AE800.pt", and "tsne.csv". The "AE800.pt" file is highlighted with a red circle. The main area of the notebook shows two code cells. The first cell, labeled [27], contains the code `torch.save(model.state_dict(), "AE800.pt")`, where the filename "AE800.pt" is circled in red. The second cell, labeled [28], contains the code `model=autoencoder() #build NN architecture`, `model.load_state_dict(torch.load("AE800.pt")) #load`, `model.to(device)`, and `model.eval()`. Below this, the definition of the `autoencoder` function is shown, starting with `autoencoder(` and `(encoder): Sequential(`.

3_AlexNet_(1).ipynb

檔案 編輯 檢視畫面 插入 執行階段 工具 說明 無法儲存變更

檔案

..

gdrive

sample_data

AE800.pt

tsne.csv

+ 程式碼 + 文字 複製到雲端硬碟

Save and load a PyTorch model (IT

```
[27] torch.save(model.state_dict(), "AE800.pt")
```

```
[28] model=autoencoder() #build NN architecture
      model.load_state_dict(torch.load("AE800.pt")) #load
      model.to(device)
      model.eval()

      autoencoder(
        (encoder): Sequential(
          (0): Conv2d(3, 32, kernel_size=(2, 2), stride=
          (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, a
          (2): ReLU()
```

Get latent vectors of all training images

```
[37]: 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,)



Use t -SNE to reduce dimensions from 64 to 2

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

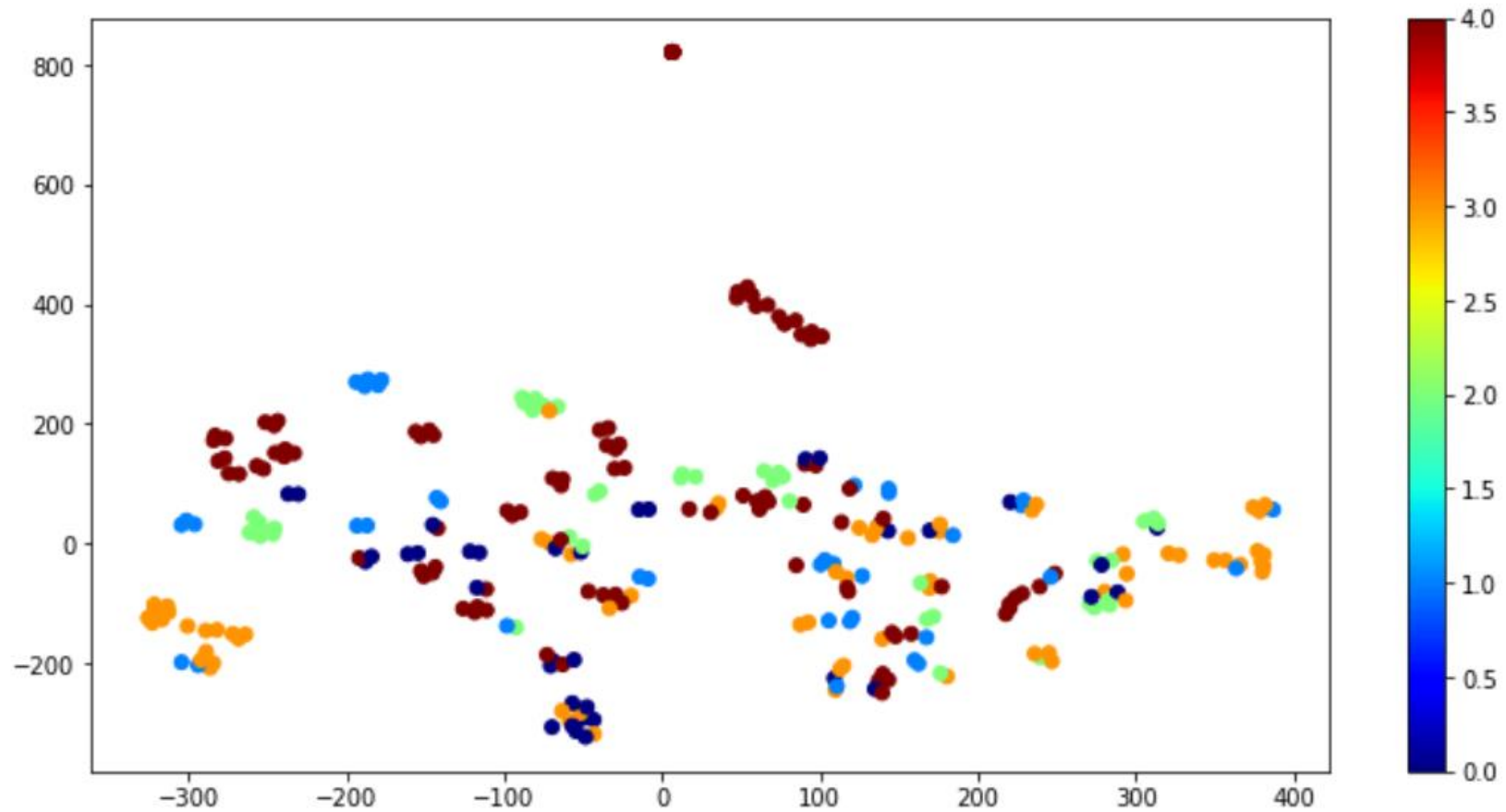
```
[39]: x=tsne.fit transform(arrayX)  
      print(x.shape)
```

(298, 2)

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



Use t -SNE to reduce dimensions from 64 to 2



Save data to csv file

```
[42] print(x.shape, arrayY.shape)
```

```
(298, 2) (298,)
```

```
[43] arrayY1 = arrayY.reshape(arrayY.shape[0], 1)
      print(arrayY1.shape)
```

```
(298, 1)
```

```
[44] XYArray = np.hstack((x, arrayY1))
      print(XYArray.shape)
```

```
(298, 3)
```

```
[45] # Save data to excel for further Tableau visualization
      import pandas as pd
      pd.DataFrame(XYArray).to_csv("tSNE.csv")
```

Save data to csv file

CO 3_AlexNet_(1).ipynb

檔案 編輯 檢視畫面 插入 執行階段 工具 說明 無法儲存變更

檔案

gdrive
sample_data
AF800.pt
tSNE.csv

+ 程式碼 + 文字 複製到雲端硬碟

```
(2, 3) (2, 1) (2, 4)
```

```
[42] print(x.shape, arrayY.shape)
```

```
(298, 2) (298,)
```

```
[43] arrayY1 = arrayY.reshape(arrayY.shape[0], 1)
```

```
print(arrayY1.shape)
```

```
(298, 1)
```

```
[44] XYArray = np.hstack((x, arrayY1))
```

```
print(XYArray.shape)
```

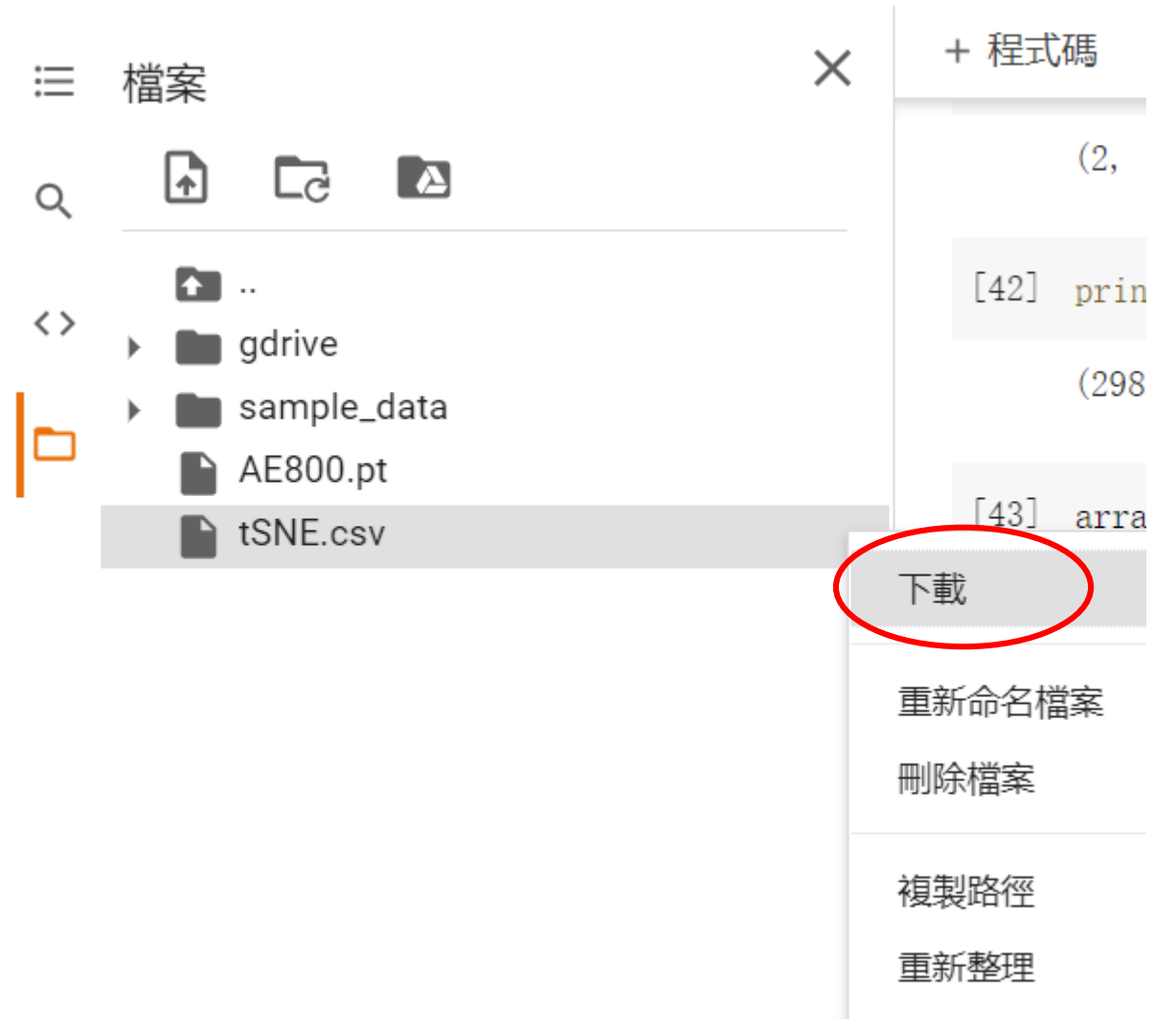
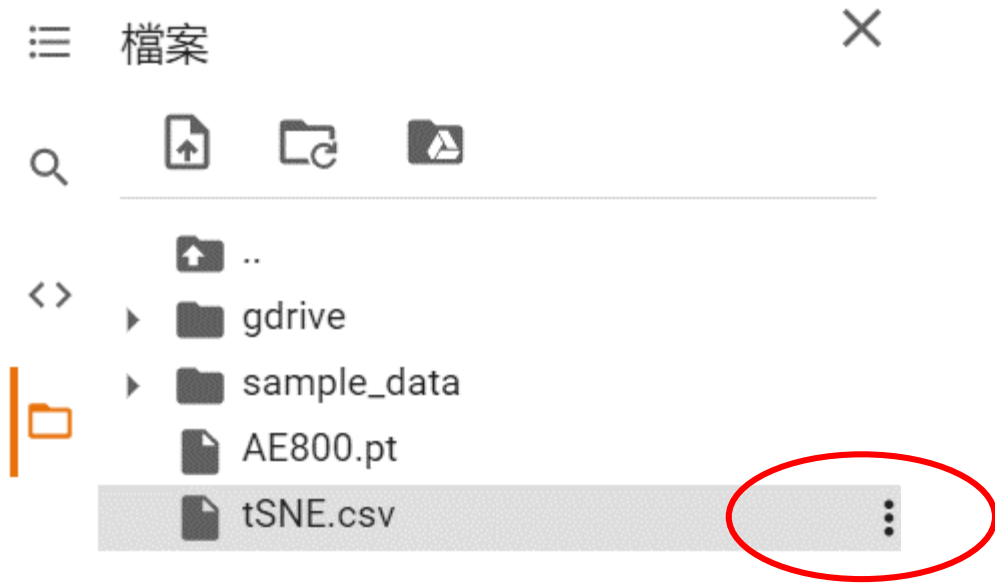
```
(298, 3)
```

```
[45] # Save data to excel for further Tableau visual
```

```
import pandas as pd
```

```
pd.DataFrame(XYArray).to_csv("tSNE.csv")
```

Download csv file



Visualize in Tableau public

ConnectionsAdd

tSNE (2)
Text file

Files

☐ Use Data Interpreter

Data Interpreter might be able to clean your Text file workbook.

HW1 lecture.txt

tSNE (1).csv

tSNE (2).csv

—

New Union

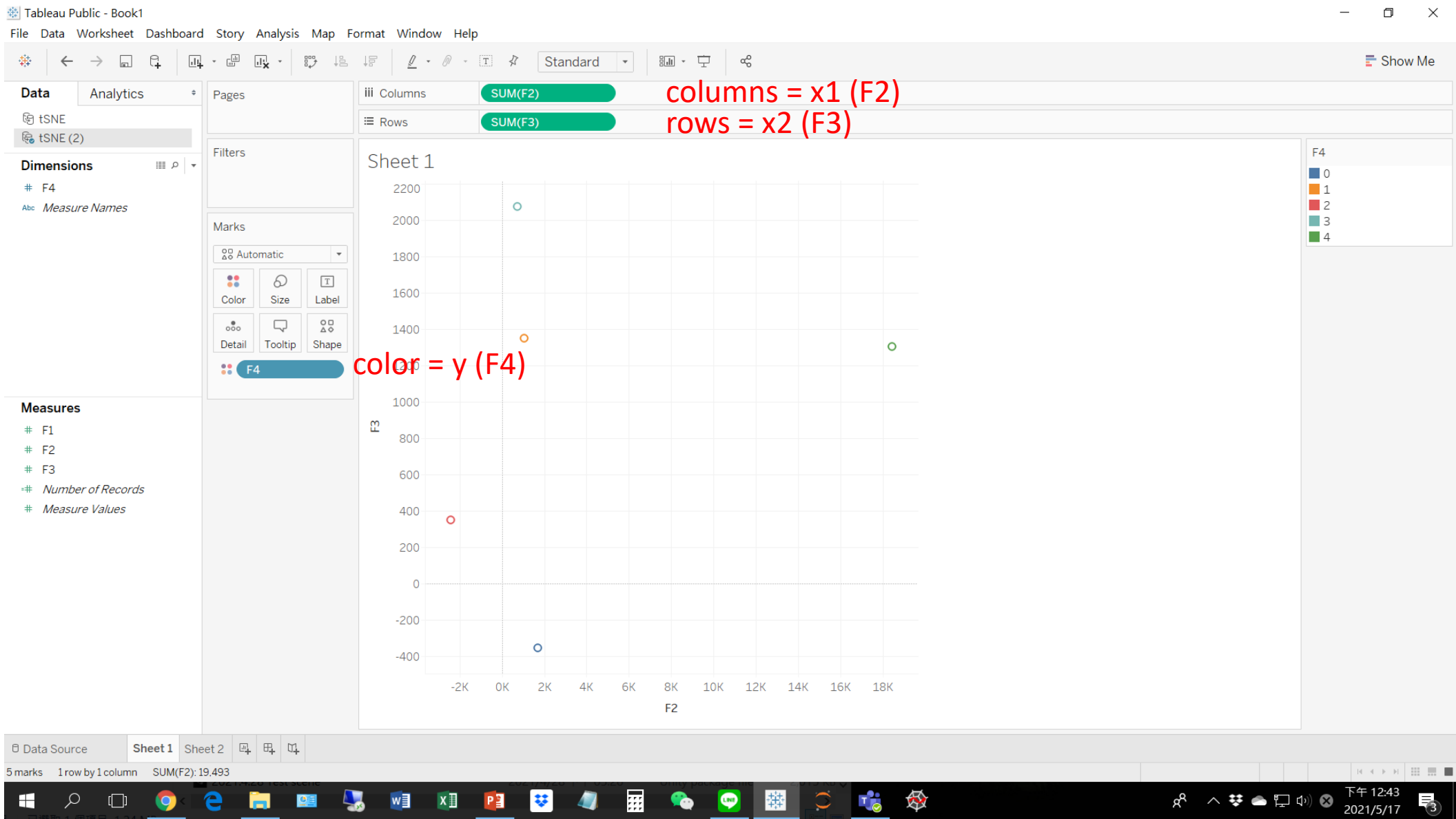
tSNE (2).csv

Sort fields

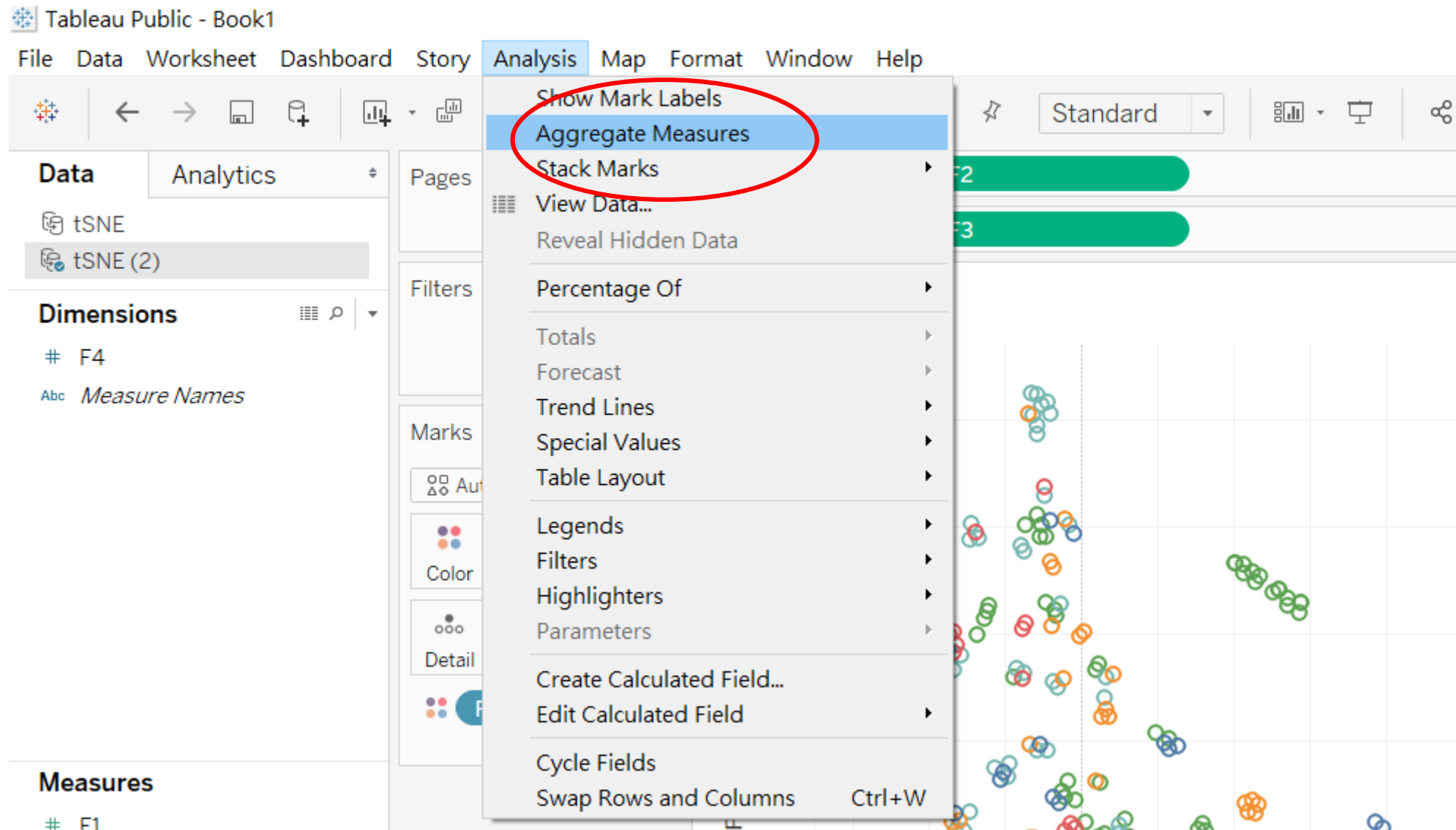
Data source order

#	# X1	# X2	# y
tSNE (...)	tSNE (2...)	tSNE (2)....	tSNE (2)...
F1	F2	F3	F4
2	-6.65	18.604	2.00000
3	-7.06	9.053	2.00000
4	-98.34	-372.219	0.00000
5	-165.39	31.733	0.00000
6	-156.86	179.328	3.00000

Visualize in Tableau public



Visualize in Tableau public



Visualize in Tableau public

