한국소음진동공학회 AI혁신위원회 주관
2022년 인공지능 강습회

AI 기계이상진단을 위한 인공지능 학습 기법

제 3강 재현 태스크를 사용한 이상진단 실습

한국과학기술원 전기및전자공학부

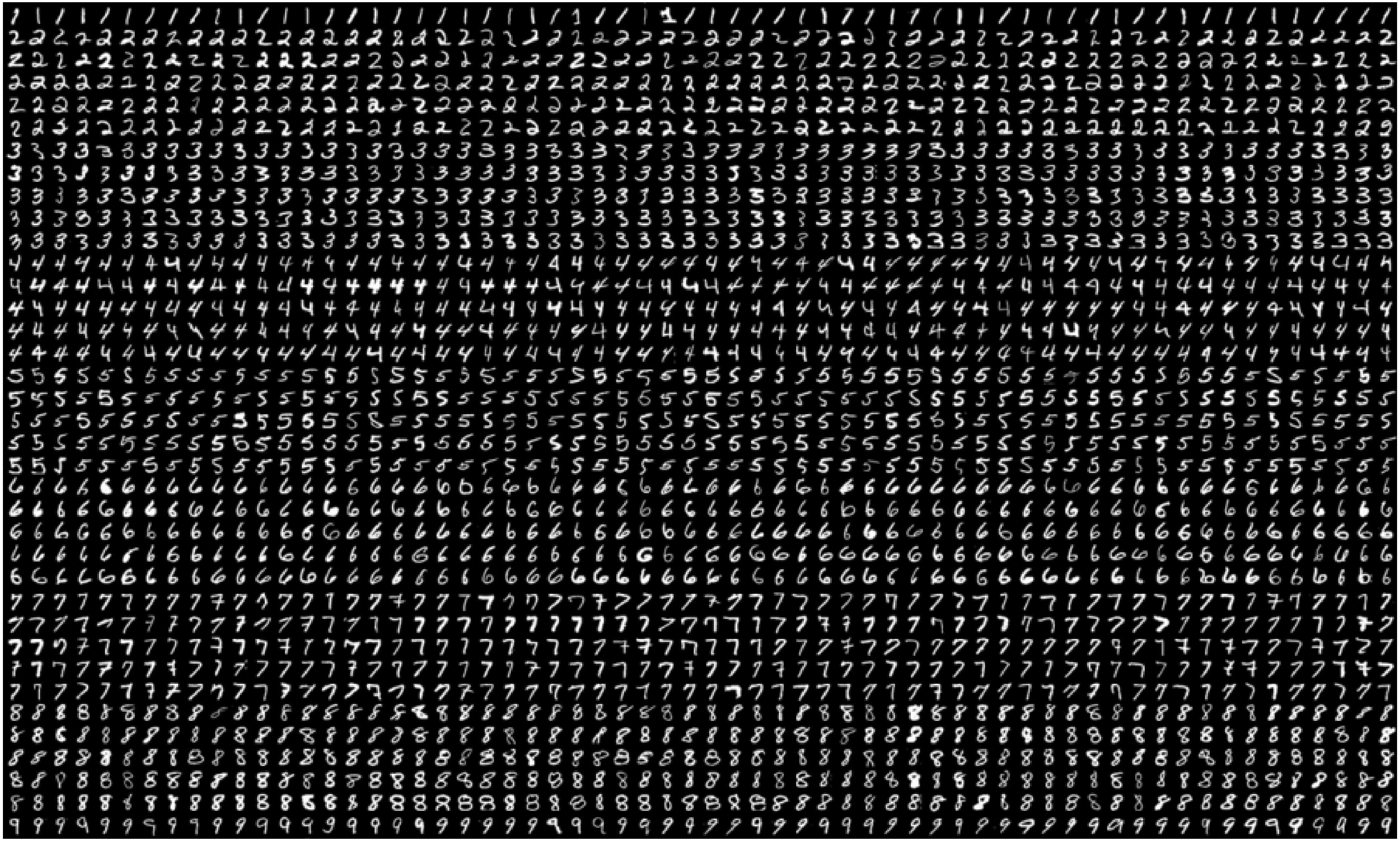
최정우
jwoo@kaist.ac.kr

KAIST EE

목차

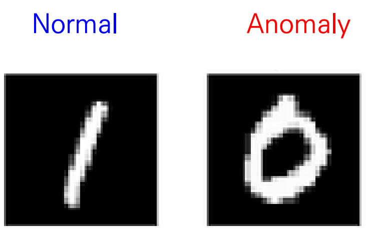
- MNIST 데이터 셋을 사용한 이상진단 실습
- 오토인코더 + 재현 태스크
- 정상 데이터의 다양성 vs. 모델 수용성
- 합성곱 오토인코더 (convolutional autoencoder)

MNIST dataset

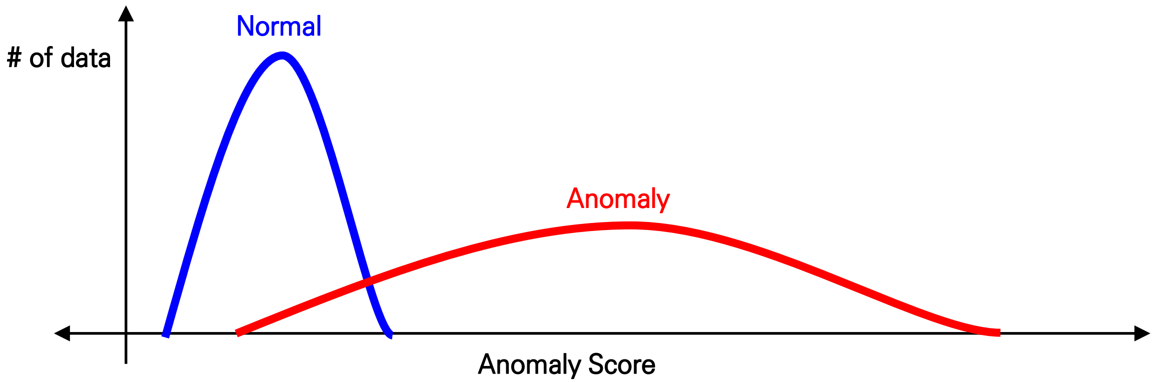


문제 정의

- Anomaly Detection with MNIST Dataset
  - Normal : 1
  - Anomaly : 0

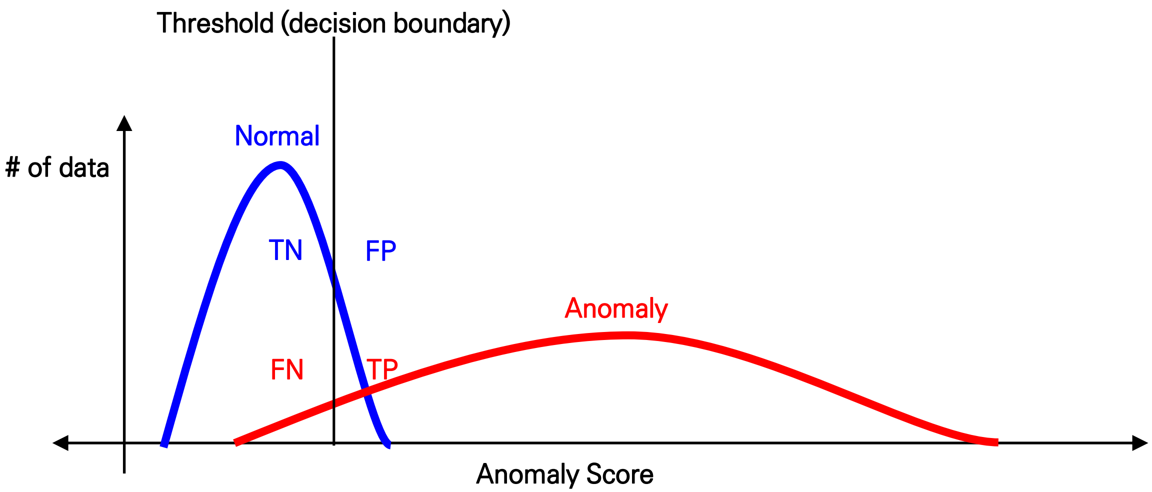


  - Anomaly score of Normal / Anomaly data

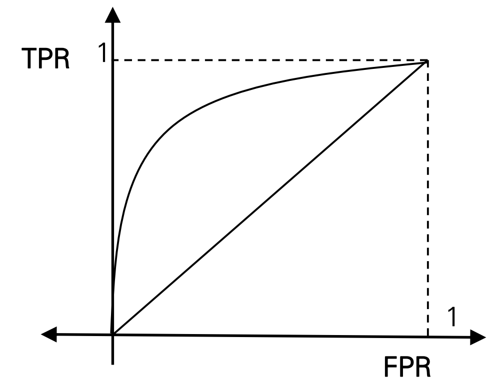


문제 정의

- Anomaly Detection with MNIST Dataset
  - TPR & FPR and ROC-AUC



$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{N} = \frac{FP}{FP + TN}$$

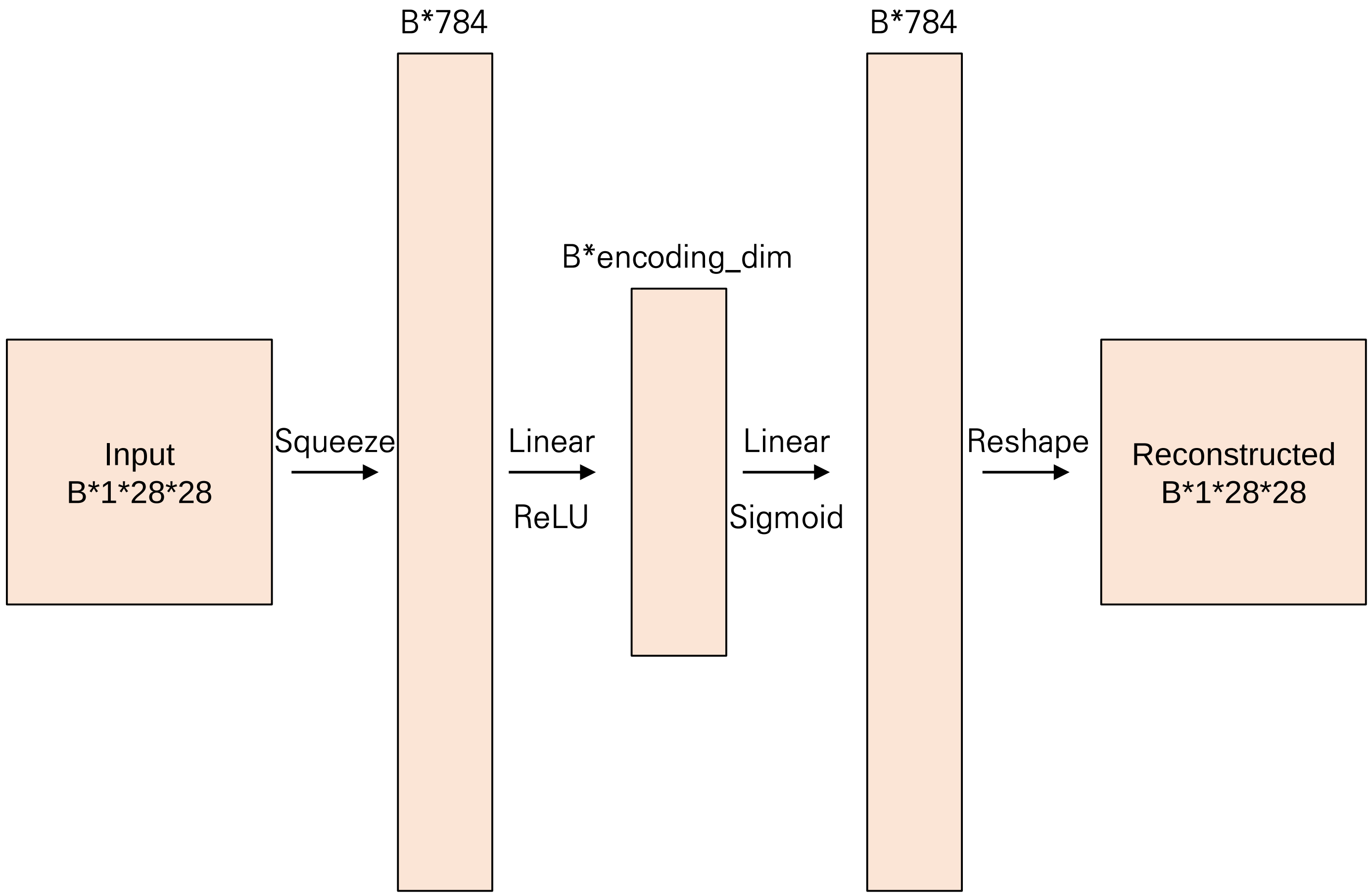


이상 진단 실습

Practice 1 : Reconstruction

Practice 1 : Reconstruction

- Autoencoder : Model architecture



Code 설명 – 1. Import packages

- 설치에 필요한 package 로드

```
[ ]  !pip install torchinfo # install torchinfo module
```

```
Requirement already satisfied: torchinfo in /usr/local/lib/python3.7/dist-packages (1.6.5)
```

```
import torch
from torch import nn
from torchvision import datasets          # https://pytorch.org/vision/stable/datasets.html
from torchvision.transforms import ToTensor
from torch.utils.data import DataLoader, TensorDataset, Subset
from sklearn import metrics               # for Anomaly scores (https://scikit-learn.org/sta
import matplotlib.pyplot as plt           # for plotting (https://matplotlib.org/3.5.1/api/_
from torchinfo import summary             # for model summary (https://github.com/TylerYep/t
import numpy as np
```

```
[ ]  device = 'cuda' if torch.cuda.is_available() else 'cpu'  ## use gpu if available
     print(f'Using {device} device')
```

```
Using cuda device
```

Code 설명 – 2. Hyperparameters

- 훈련의 number of epochs, batch size 설정
- Anomaly, Normal 로 사용할 MNIST의 숫자 설정
- Autoencoder의 latent variable 개수 설정

2. Hyperparameters

You can change the hyperparameter below. Parameter `ANOMALY_NUM` means which number to set as an anomaly.

```
#@title Hyperparameters
EPOCHS = 5          # Number of epochs to train
BATCH = 32          # Minibatch size
ANOMALY_NUM = [0]   # (list) Data of digit "0" will be used as anomalous data
NORMAL_NUM =  [1,2,3]   # (list) Data of digit "1" will be used as normal data
LATENT_DIM = 64     # Latent dimension
```

The dataset for anomaly detection is composed of normal data and anomaly data. Normal data can be seen in the train, validation, and test dataset, but anomaly data is not in the train dataset. Here digit `1` will be used as `normal data`, and `0` will be used as `anomaly data`.

Code 설명 – 3. Dataset과 DataLoader

(v)

Here we use MNIST dataset. The datatype of the dataset is a TorchTensor of tuples: (`image tensor`, `target label`)

The dimensions of an image tensor is (`channel` =1, `width` =28, `height` =28)

Here, we use `Subset` package in `torch.utils.data` to build two sub-datasets (validation, test).

```
[ ]  # Get MNIST train and test data (https://pytorch.org/vision/stable/generated/torchvision.datasets.MN
     mnist_train = datasets.MNIST(root='MNIST_data/', train=True, transform=ToTensor(), download=True)
     mnist_test  = datasets.MNIST(root='MNIST_data/', train=False, transform=ToTensor(), download=True)
```

```
[ ]  # define training & test dataset
     train_idx = [i for i,v in enumerate(mnist_train) if v[1] in NORMAL_NUM]       # get a list of indi
     train_dataset = Subset(mnist_train,train_idx)                                 # get a subset of da

     testdigit = ANOMALY_NUM + NORMAL_NUM                                          # join two list
     test_idx =[i for i,v in enumerate(mnist_test) if v[1] in testdigit]
     Num_test = int(len(test_idx)/2);
```

- Dataloader: 대용량의 데이터를 처리하기 위한 데이터 로더
- Subset: 큰 데이터 세트 중의 일부를 서브셋으로 정의
- Test set 중 절반은 validation, 나머지 절반을 test에 사용

Code 설명 – 3. Dataset과 DataLoader

```
test_dataset = Subset(mnist_test,test_idx[:Num_test])
val_dataset = Subset(mnist_test,test_idx[Num_test:])

train_dataloader = DataLoader(train_dataset, batch_size=BATCH, shuffle=True)
val_dataloader = DataLoader(val_dataset, batch_size=BATCH)
test_dataloader = DataLoader(test_dataset, batch_size=BATCH)
```

```
[ ]  print(len(train_dataset))
     print(len(val_dataset))
     print(len(test_dataset))
```

```
6742
1058
1057
```

- DataLoader 객체 생성: train, validation, test

Code 설명 – 4. DNN 모델 정의

```
## Define the autoencoder class
class AutoEncoder(nn.Module):
    def __init__(self, encoding_dim):
        super(AutoEncoder, self).__init__()
        self.encoding_dim = encoding_dim
        self.encoder = nn.Sequential(
            nn.Linear(28*28, self.encoding_dim),
            nn.ReLU()
        )
        self.decoder = nn.Sequential(
            nn.Linear(self.encoding_dim, 784),
            nn.Sigmoid()
        )

    def forward(self, x):
        out = x.reshape(x.size(0), -1)
        out = self.encoder(out)
        out = self.decoder(out)
        out = out.view(x.size())
        return out
```

nn.Module을 template으로 상속하여 AutoEncoder 생성

super class 초기화 함수(상속)

class 내부에 encoding_dim 변수 저장

encoder 구성
nn.Linear : fully connected network (in size, out size)

decoder 구성

forward propagation 정의

- AutoEncoder 정의

Code 설명 – DNN 모델 생성

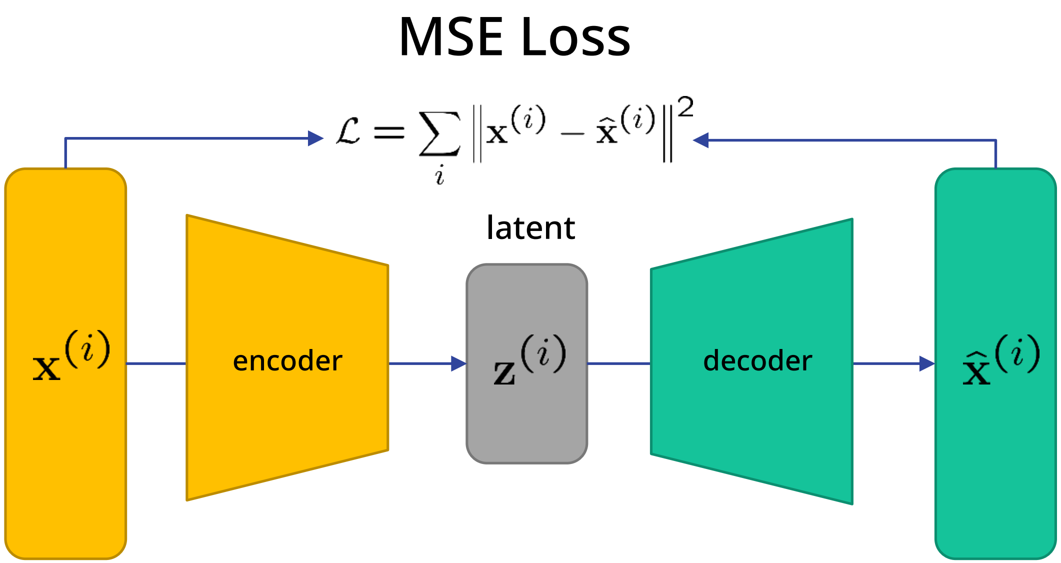
```
## Create autoencoder object
model = AutoEncoder(LATENT_DIM).to(device)
summary(model,input_size=(BATCH,1,28,28))  # summary(model, input_size)
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
AutoEncoder                              --                        --
├─Sequential: 1-1                          [32, 64]                  --
│    └─Linear: 2-1                          [32, 64]                  50,240
│    └─ReLU: 2-2                            [32, 64]                  --
├─Sequential: 1-2                          [32, 784]                 --
│    └─Linear: 2-3                          [32, 784]                 50,960
│    └─Sigmoid: 2-4                         [32, 784]                 --
==========================================================================================
Total params: 101,200
Trainable params: 101,200
Non-trainable params: 0
Total mult-adds (M): 3.24
==========================================================================================
Input size (MB): 0.10
Forward/backward pass size (MB): 0.22
Params size (MB): 0.40
Estimated Total Size (MB): 0.72
==========================================================================================
```

Code 설명 – Loss function 정의

```
[ ]   loss_fn = nn.MSELoss()
      anomaly_score = nn.MSELoss(reduction='none')
      optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
```

- MSE loss
  - reduction = ‘none’ : MSE of each data (without averaging across all data)



$$\mathcal{L} = \sum_i \left\| \mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)} \right\|^2$$

Code 설명 – 5. Train module 정의

```
## Define trainer object
def train(dataloader, model, loss_fn, optimizer):
    model.train()                                        ← Train mode로 전환

    size = len(dataloader.dataset)
    losses = []
    for batch, X in enumerate(dataloader):
        X = X[0].to(device)                              ← batch의 image tensor를 gpu로 보내기
        pred = model(X)                                  ← Forward propagation [Nbatch, img_x, img_y]
        loss = loss_fn(pred, X)                          ← loss 계산

        for idata in range(pred.shape[0]):               ← for each image index in batch
            iloss = loss_fn(pred[idata,:,:], X[idata,:,:])   ← image 별 loss 계산
            losses.append(iloss)

        optimizer.zero_grad()                            ← gradient 초기화
        loss.backward()                                  ← backward propagation
        optimizer.step()

        if batch % 300 == 0:                             ← 300번 iteration마다 손실함수 출력
            loss, current = loss.item(), batch * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")

    losses = [i.item() for i in losses]                  ← tensor를 list로 변환
    return np.mean(losses)
```

Code 설명 – 6. Test module 정의

```
## Define test object: evaluate the model and plot results
def test(dataloader, model, loss_fn, anomaly_score, draw_mode = False):
    model.eval()                                          ← Evaluation mode로 전환

    y_true, y_pred = [], []

    normal_loss = []
    anomaly_loss = []

    with torch.no_grad():                                 ← Gradient tracker off
        for X, y in dataloader:                           ← X: image, y: label
            X, y = X.to(device), y.to(device)
            output = model(X)                             ← Reconstructed image (batch)

            for idata in range(output.shape[0]):          ← for each image in batch
                loss = loss_fn(output[idata,:,:], X[idata,:,:])   ← individual loss
                if y[idata] in NORMAL_NUM:                ← 현재 데이터의 digit이 normal 이면 digit을 1로 변경
                    y[idata] = 0
                    normal_loss.append(loss)
                elif y[idata] in ANOMALY_NUM:             ← 현재 데이터의 digit이 anomaly 이면 digit을 0으로 변경
                    y[idata] = 1
                    anomaly_loss.append(loss)

            score = torch.mean(anomaly_score(X, output), (1, 2, 3))   ← Anomaly score 계산
```

Code 설명 – 6. Test module 정의

```python
                y_true.extend(y.tolist())                    # torchtensor -
                y_pred.extend(score.tolist())

    roc_auc = metrics.roc_auc_score(y_true, y_pred)

    if draw_mode:
        fpr, tpr, _ = metrics.roc_curve(y_true,y_pred)
        plt.figure(figsize=(5,5))
        plt.plot(fpr,tpr)
        plt.title('ROC curve')
        plt.xlabel('FPR')
        plt.ylabel('TPR')


    print(f'ROC AUC: {roc_auc:>0.3f}')
    normal_loss = [i.item() for i in normal_loss]
    anomaly_loss = [i.item() for i in anomaly_loss]
    normal_loss_mean = np.mean(normal_loss)
    anomaly_loss_mean = np.mean(anomaly_loss)
    print(f'normal loss : {normal_loss_mean}')
    print(f'anomaly loss : {anomaly_loss_mean}')
    return roc_auc.item(), normal_loss_mean, anomaly_loss_mean, normal_loss, anomaly_loss
```

list로 변환해서 y_true와 y_predict에 추가

ROC-AUC 계산

Code 설명 – Trainer 실행

```
aucs = []
train_normal_losses = []
val_normal_losses = []
val_anomaly_losses = []
best_auc = 0
best_model = model.to(device)  ## allocate model to GPU or CPU

for t in range(EPOCHS):
    print(f"Epoch {t+1}₩n-------------------------------")

    # train for single epoch
    train_normal_loss = train(train_dataloader, model, loss_fn, optimizer)
    # validation for single epoch
    auc, val_normal_loss, val_anomaly_loss, _, _ = test(val_dataloader, model, loss_fn, anomaly_score)

    train_normal_losses.append(train_normal_loss)
    aucs.append(auc)
    val_normal_losses.append(val_normal_loss)
    val_anomaly_losses.append(val_anomaly_loss)
    if best_auc < auc:                                      # as num of epoch increases
        best_model = model
        best_auc = auc
```

← 매 epoch 마다 더 나은 모델로 업데이트 (best_model = model.to(device))

← Trainer 실행 (train_normal_loss = train(...))

← Validation 실행 (test(...))

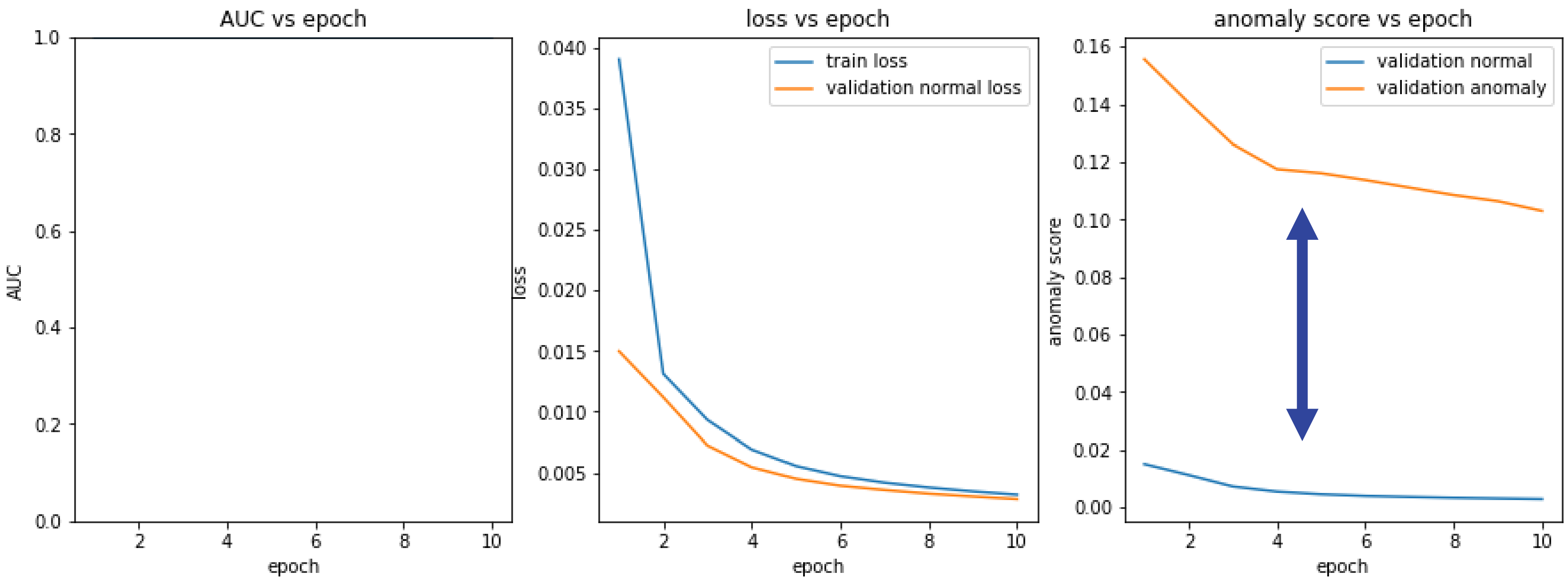
← 매 epoch 마다 더 나은 모델로 업데이트 (best_auc = auc)

Code 설명 – Trainer 실행

```
Epoch 1
-------------------------------
loss: 0.239662  [    0/ 6742]
ROC AUC: 1.000
normal loss : 0.014711812181736936
anomaly loss : 0.1579014242692726
Epoch 2
-------------------------------
loss: 0.013732  [    0/ 6742]
ROC AUC: 1.000
normal loss : 0.010429973634467883
anomaly loss : 0.138356912441141
Epoch 3
-------------------------------
loss: 0.009401  [    0/ 6742]
ROC AUC: 1.000
normal loss : 0.007235441277946599
anomaly loss : 0.12779102024922925
Epoch 4
-------------------------------
loss: 0.009527  [    0/ 6742]
ROC AUC: 1.000
normal loss : 0.005520757522899658
anomaly loss : 0.122651199223547
```

Practice 1 : Reconstruction

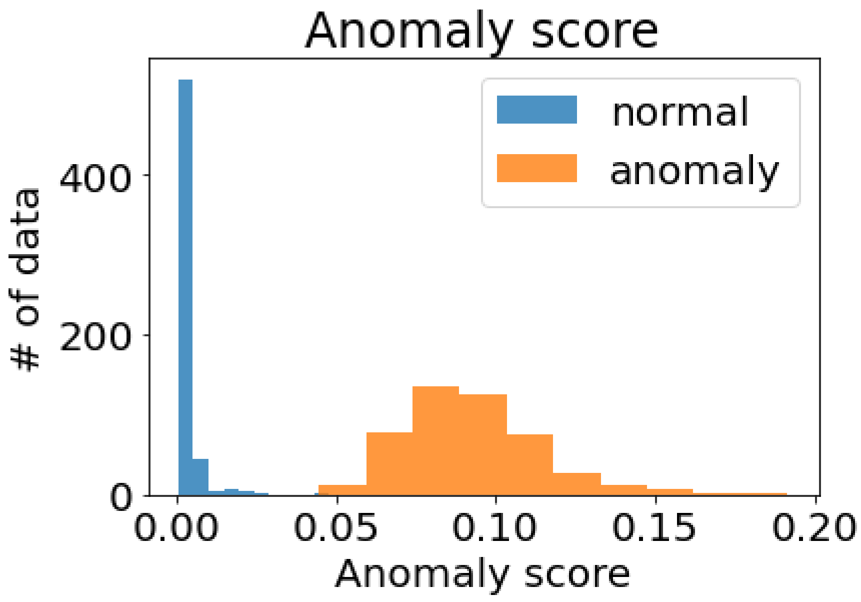
- Autoencoder : activity 1 – run the given code

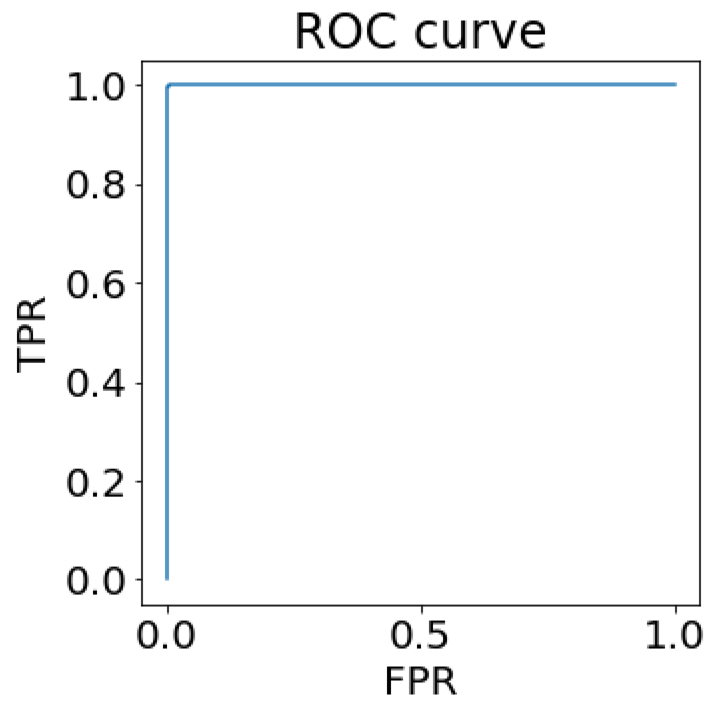


  – Q : Why there is gap between anomaly score of normal and anomaly?
    A : Model is not trained to reconstruct anomaly data (unseen).

Practice 1 : Reconstruction

- autoencoder : activity 1 –(A0 vs. N1, encoding_dim =64)

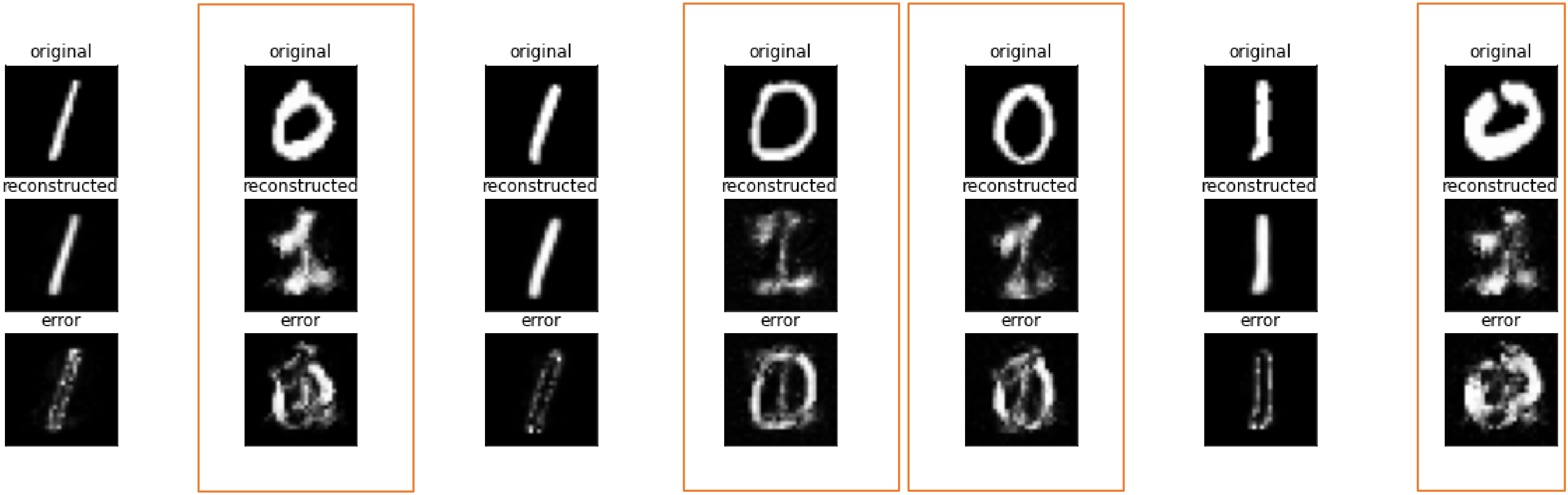




ROC AUC = 1.0

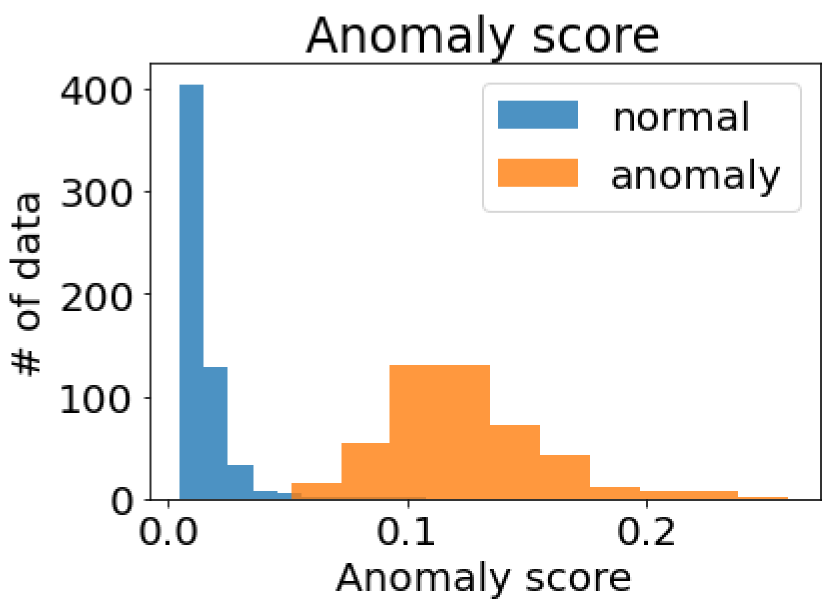
Practice 1 : Reconstruction

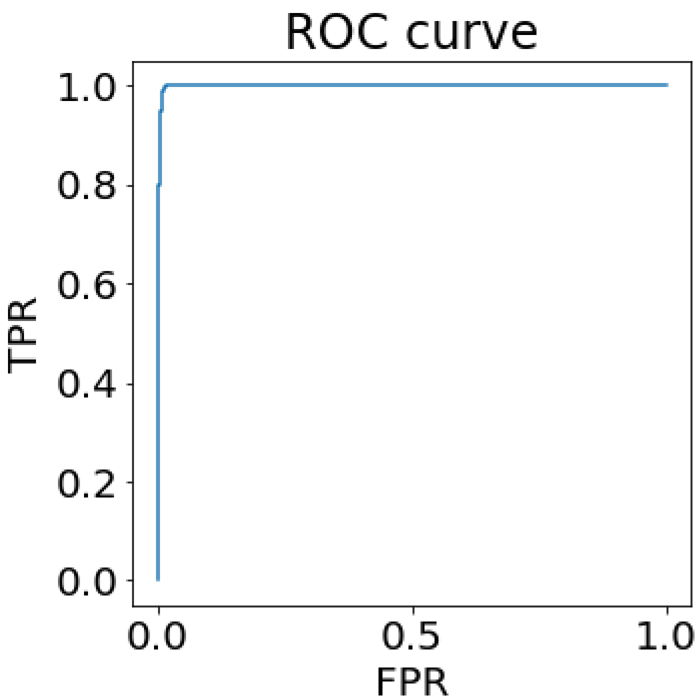
- autoencoder : activity 1 –(A0 vs. N1, encoding_dim =64)



Practice 1 : Reconstruction

- autoencoder : activity 2 – change encoding_dim (encoding_dim = 4)

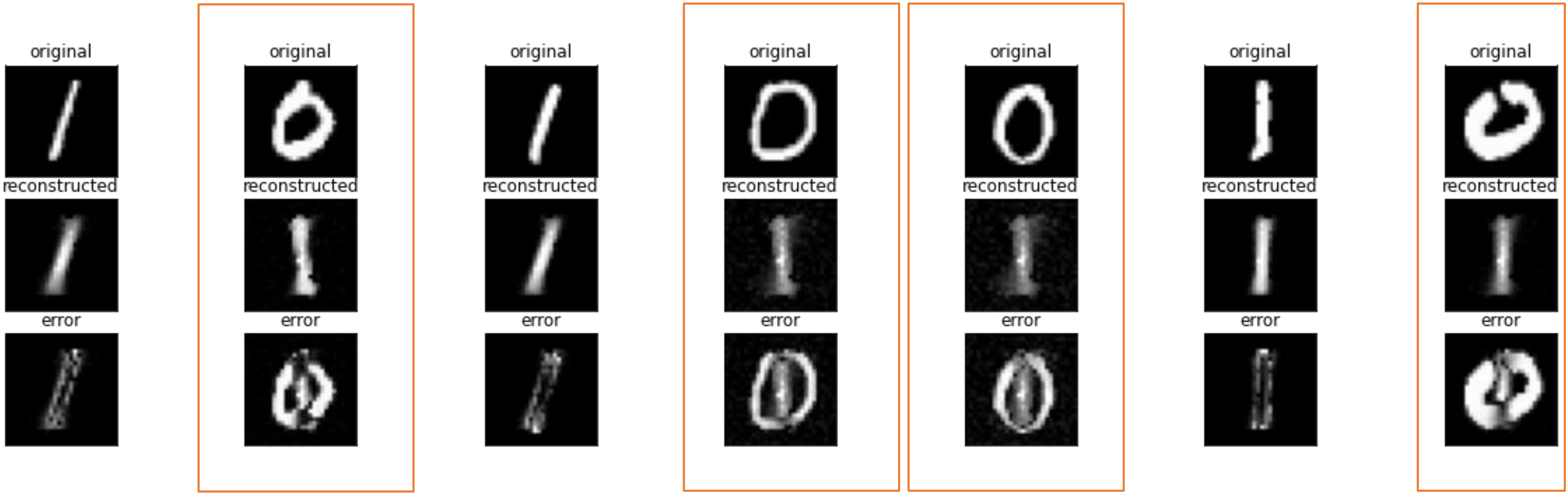




ROC AUC = 0.999

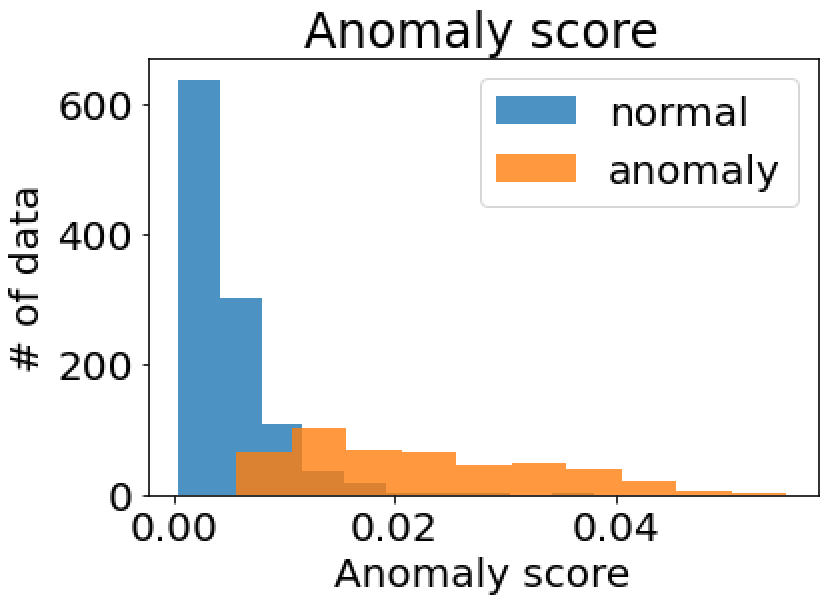
Practice 1 : Reconstruction

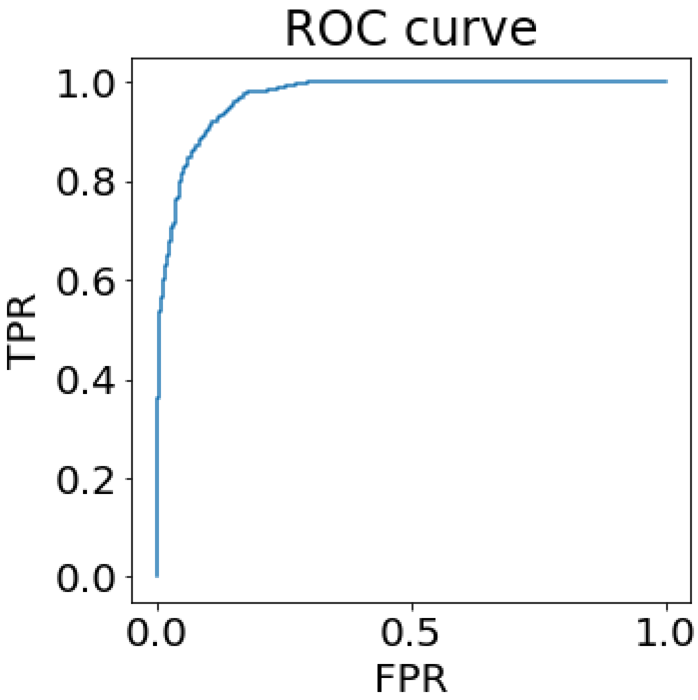
- autoencoder : activity 2 –(A0 vs. N1, encoding_dim =4)



Practice 1 : Reconstruction

- autoencoder : activity 3 – (Anomaly 0 vs Normal 1,2 encoding_dim = 64)

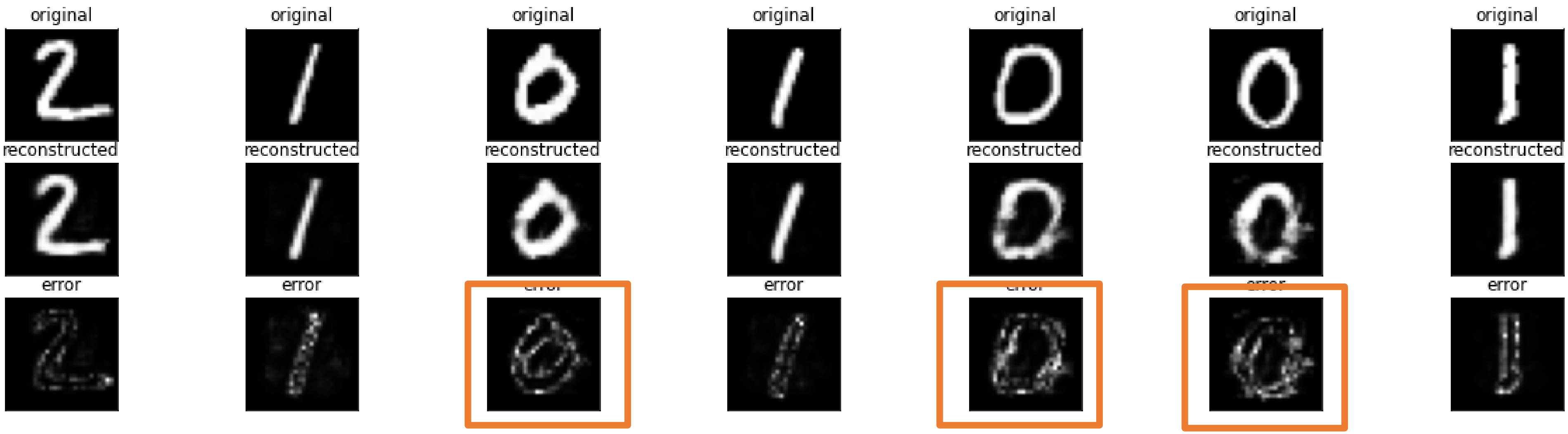




ROC AUC = 0.970

Practice 1 : Reconstruction

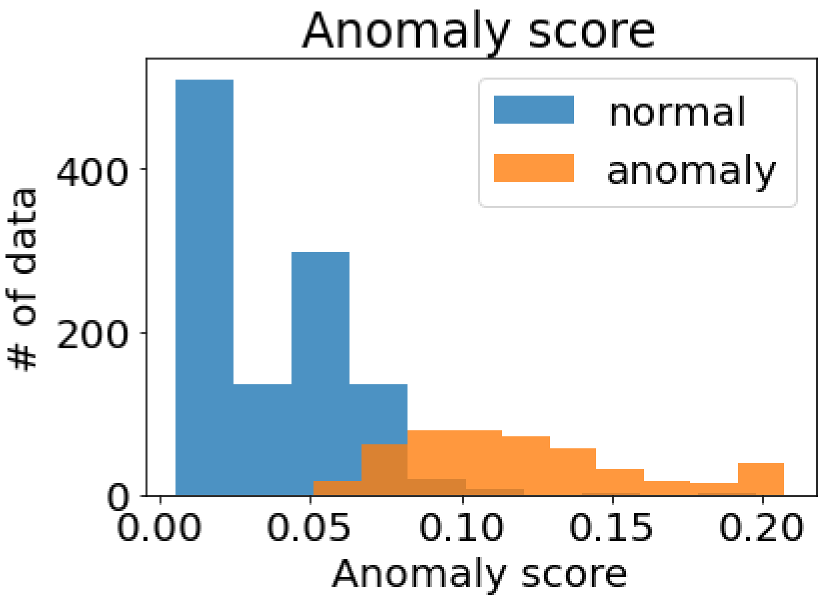
- autoencoder : activity 2 – (A0 vs N1,2 encoding_dim = 64)

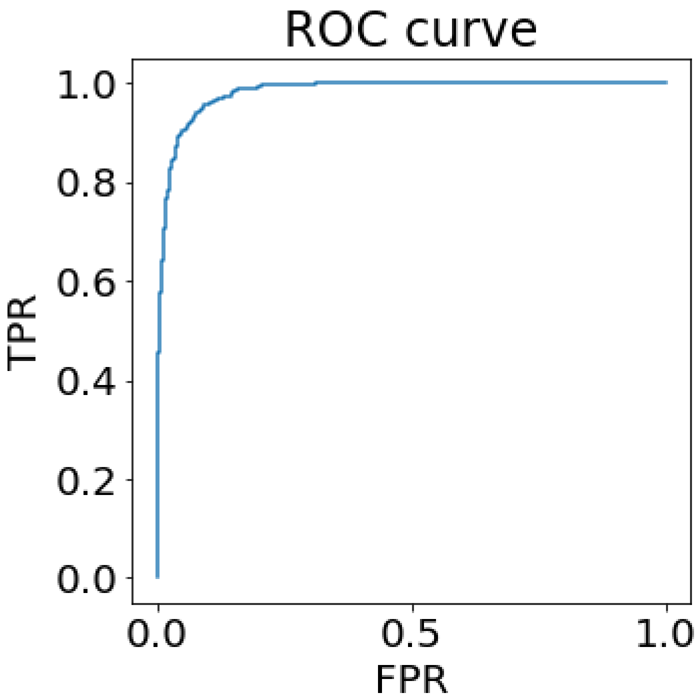


  – Due to the inclusion of “2” to normal data, the model is trained to express more versatile features

Practice 1 : Reconstruction

- autoencoder : activity 3 – (A0 vs N1,2 encoding_dim = 4)

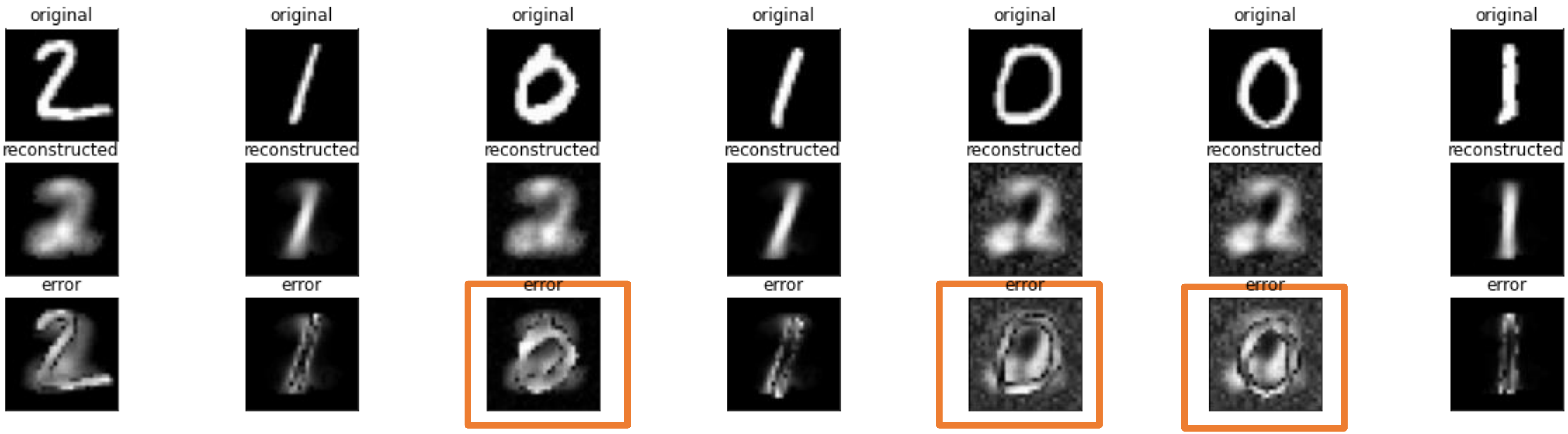




ROC AUC = 0.982

Practice 1 : Reconstruction

- autoencoder : activity 2 – (A0 vs N1,2 encoding_dim = 4)



  - Due to the small latent dimension (4), the number of expressible features is extremely small
  - Blurry image even for the normal data
  - However, few latent variables increase the reconstruction error for the anomaly data dramatically

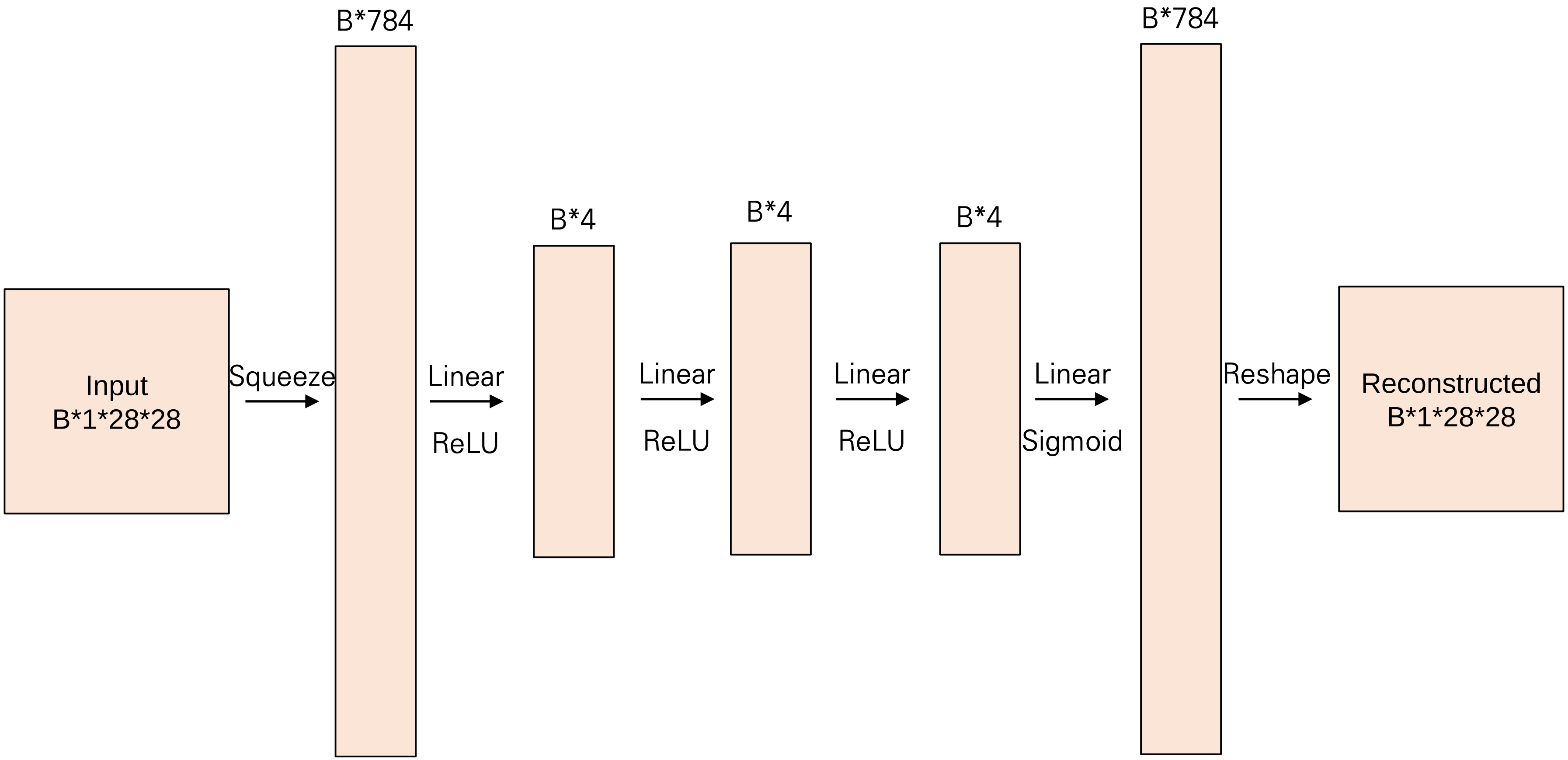
Practice 1 : Reconstruction

- Diversity of normal data VS Model capacity

		Diversity of Normal Data	
		Low	High
Model capacity	Low		
	High		

Practice 1 : Reconstruction

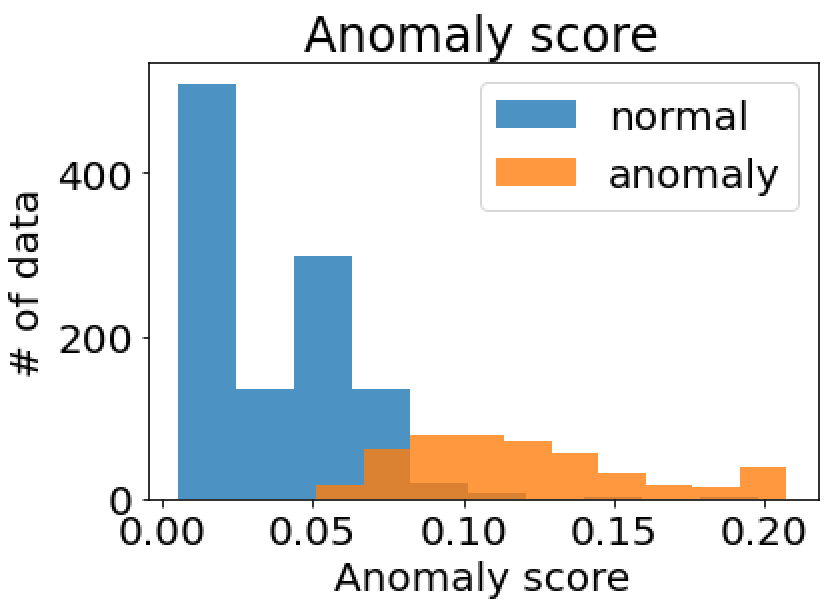
- autoencoder : activity 3 – make it deeper



Practice 1 : Reconstruction

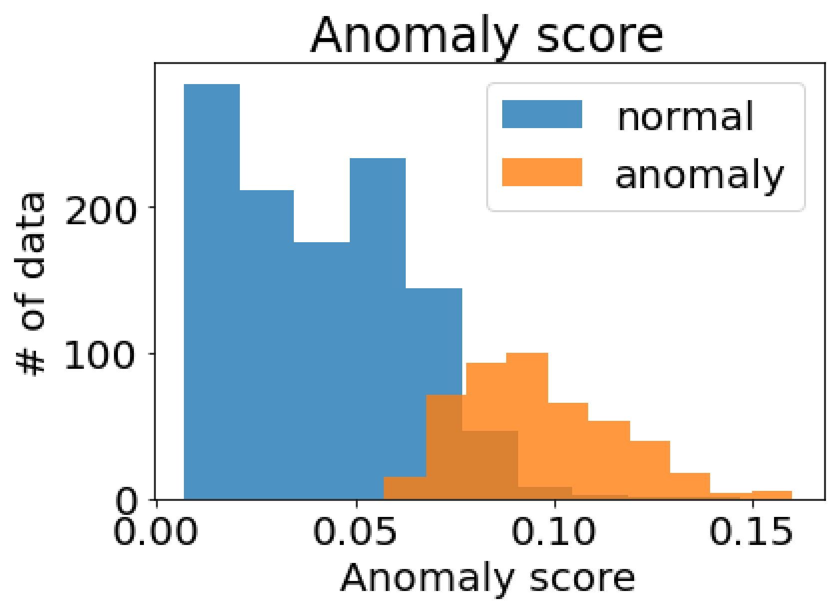
- autoencoder : activity 3 – make it deeper (encoding_dim = 4)

Shallow model



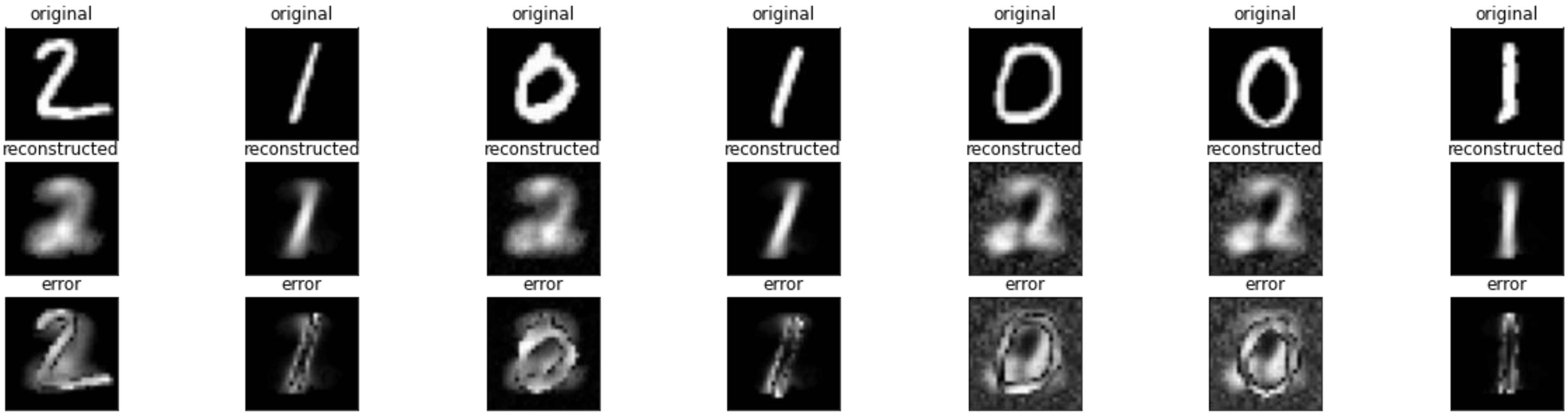
ROC AUC = 0.982

Deep model



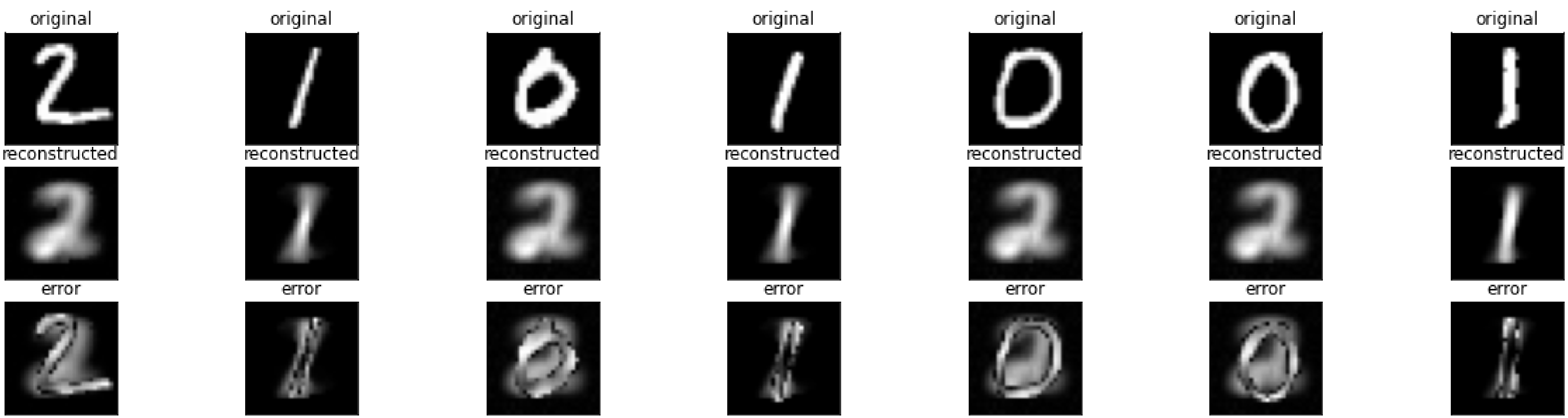
ROC−AUC: 0.976

Practice 1 : Reconstruction



Shallow model

Deep model

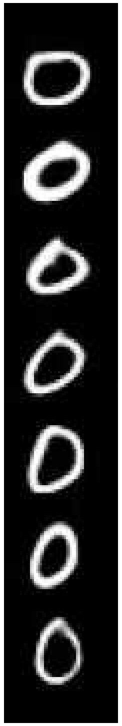


Practice 1 : Reconstruction

- autoencoder : activity 4 – High diversity of normal data
- (A0 vs N1,2,3,4,5 encoding_dim = 64, num_layer=4)

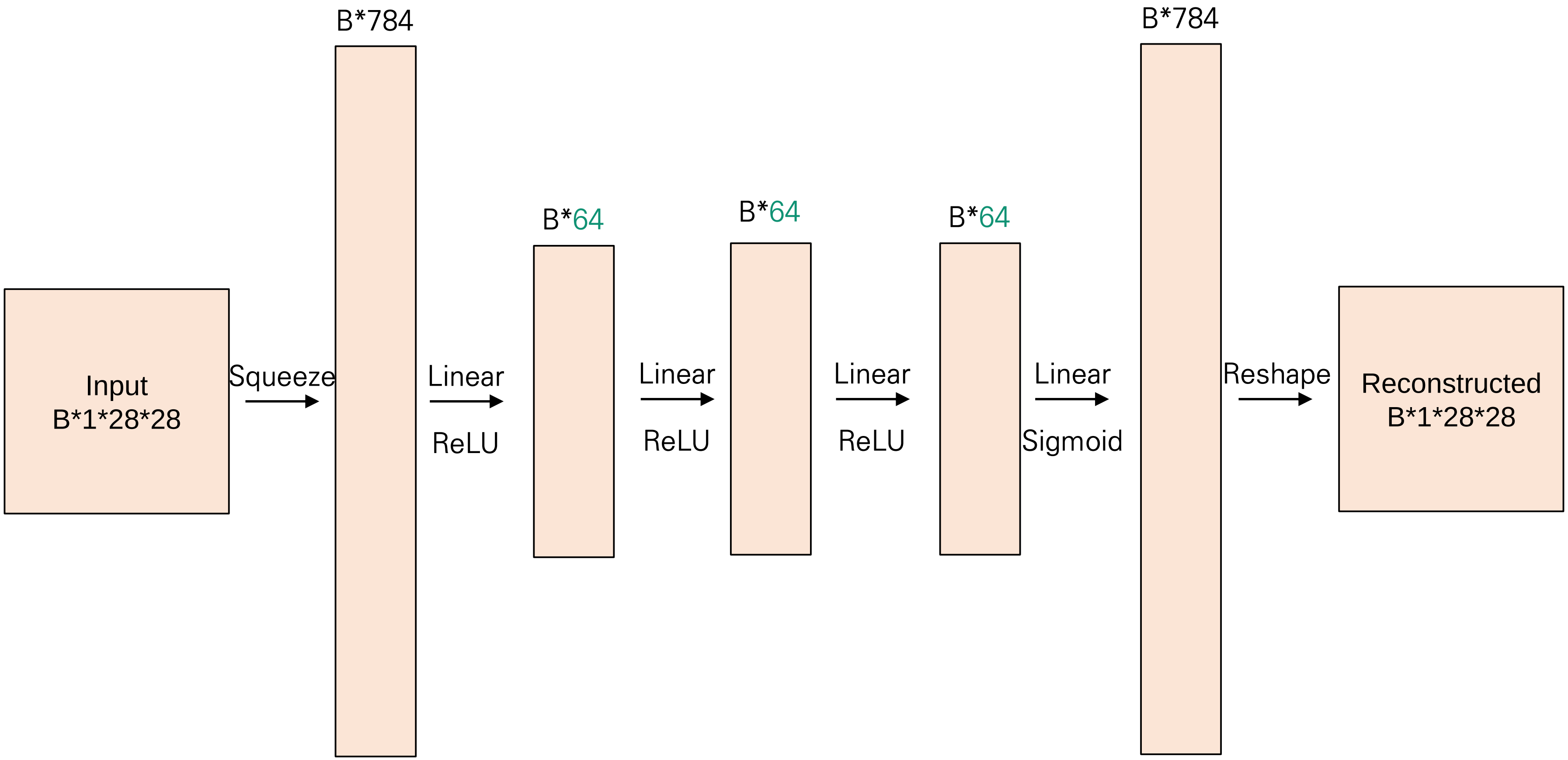
Normal

Anomaly



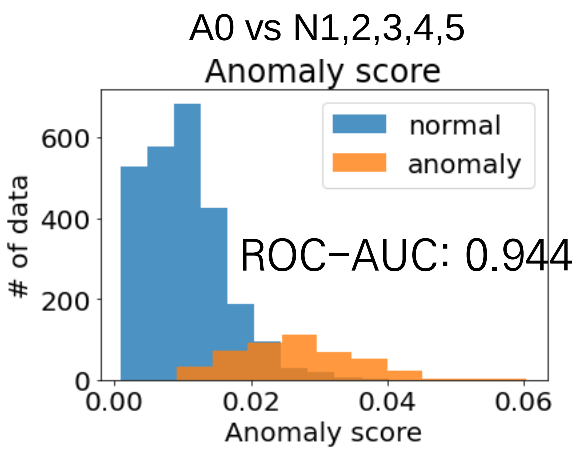
Practice 1 : Reconstruction

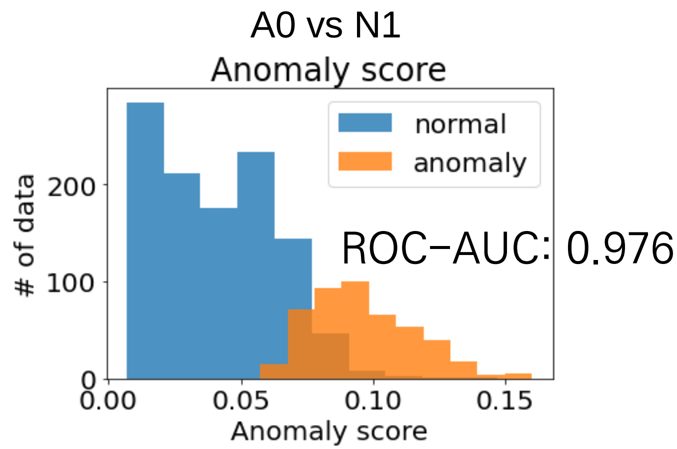
- autoencoder : activity 4 – High diversity of normal data

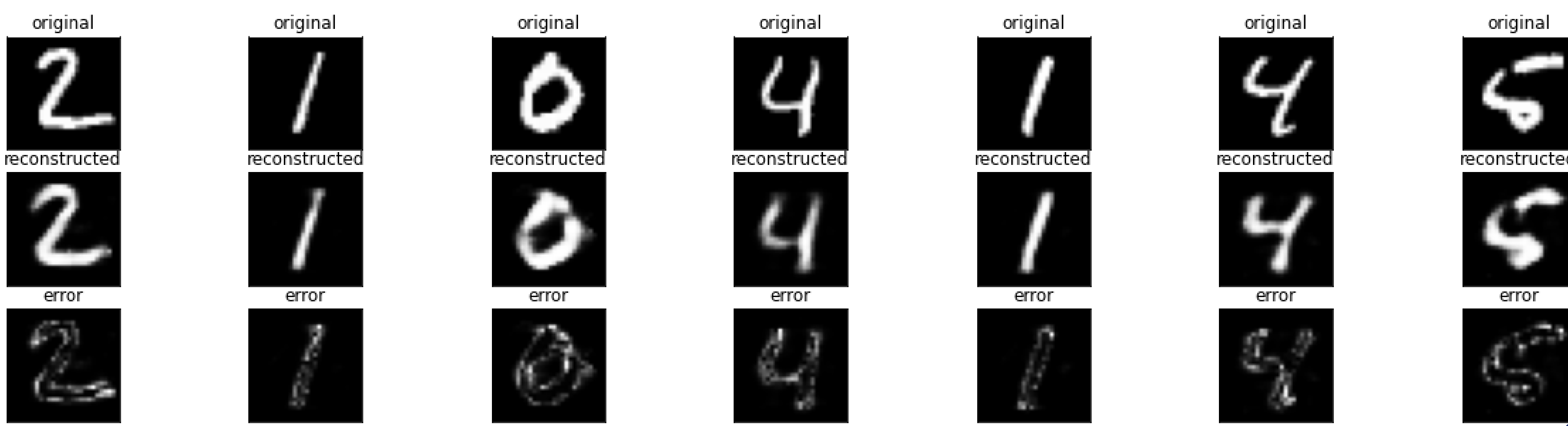


Practice 1 : Reconstruction

- autoencoder : activity 4 (A0 vs N1,2,3,4,5 encoding_dim = 64, num_layer=4)
- Lowest ROC-AUC!





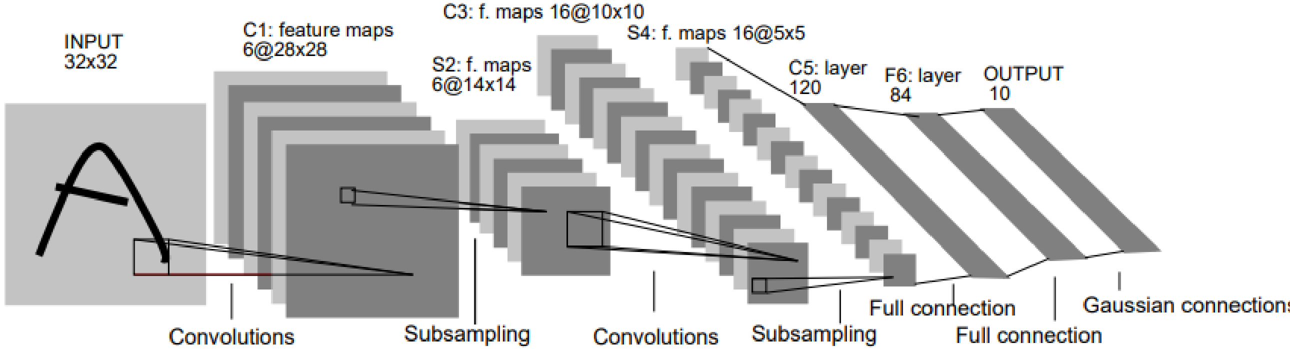


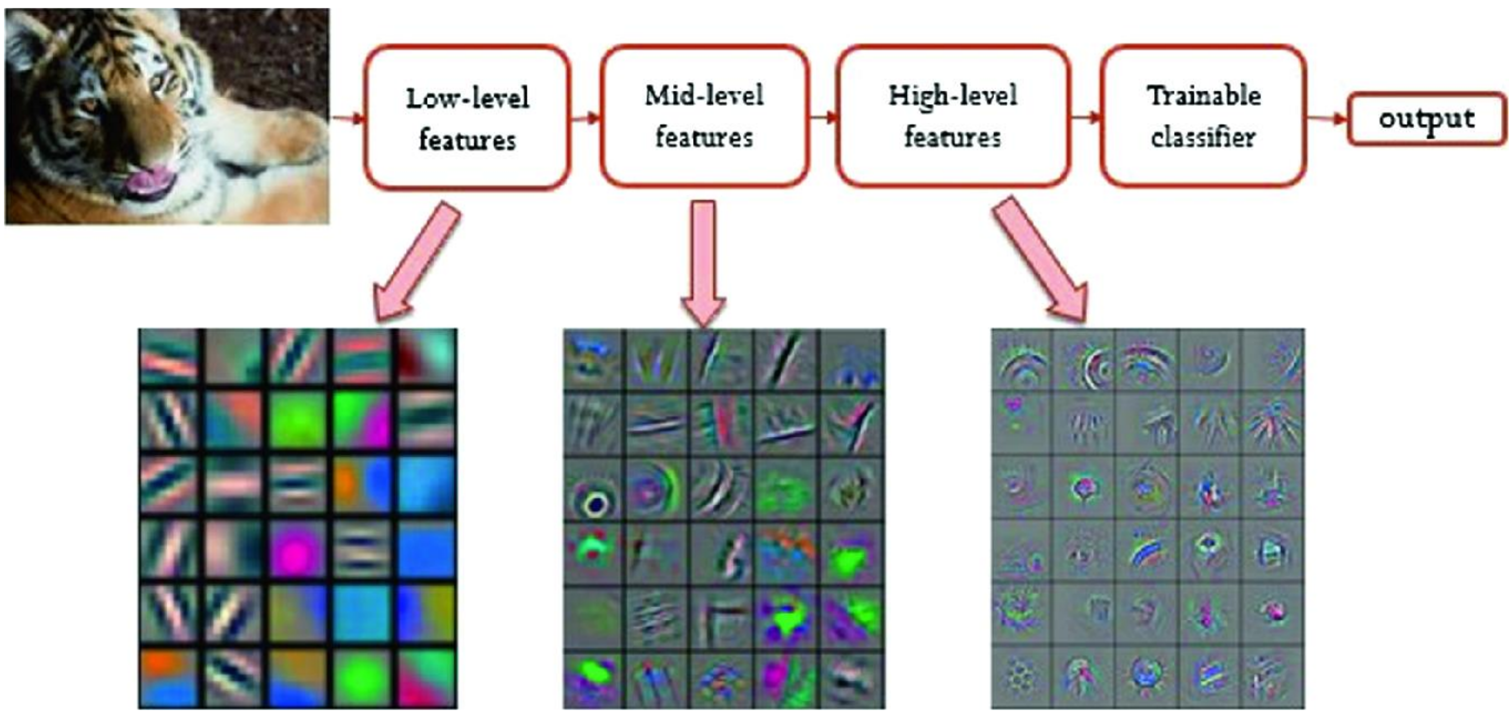
Practice 1 : Reconstruction

- So, what is the good reconstruction model?
  - The model's latent space should be small enough to capture only the essence of the training (normal) data.
  - How small it should be? → No golden rule
    To keep the latent small, the DNN structure should be able to capture the "context" o f the normal data.

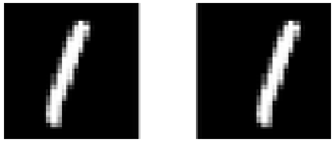
- What can else we can change?
  - Using convolutional neural network (CNN) helps to capture shift-invariant features
  - More comprehensive task than the bottleneck + reconstruction
  - Epoch / Optimizer / LR scheduler … / Batch size

Convolutional Network



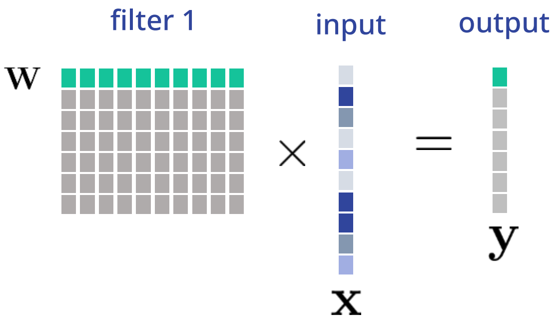


Convolutional Network



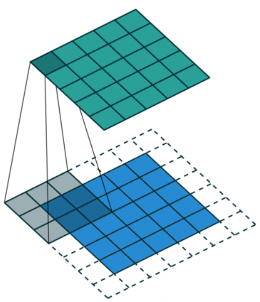
- Shift invariance (translation invariance)
  - Linear layer (MLP, FCN)

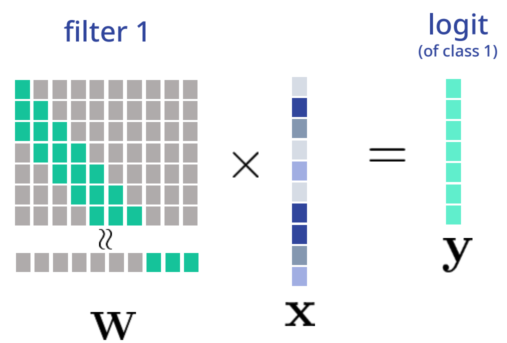
$$\mathbf{y} = \mathbf{W}\mathbf{x}$$



  - Convolutional layer

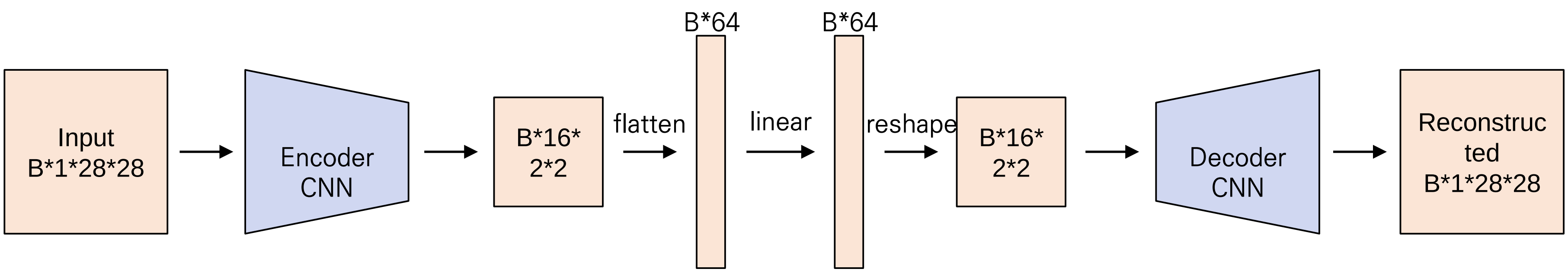
$$\mathbf{y} = \mathbf{w} * \mathbf{x}$$



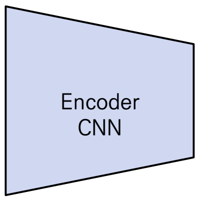


Practice 1 : Reconstruction

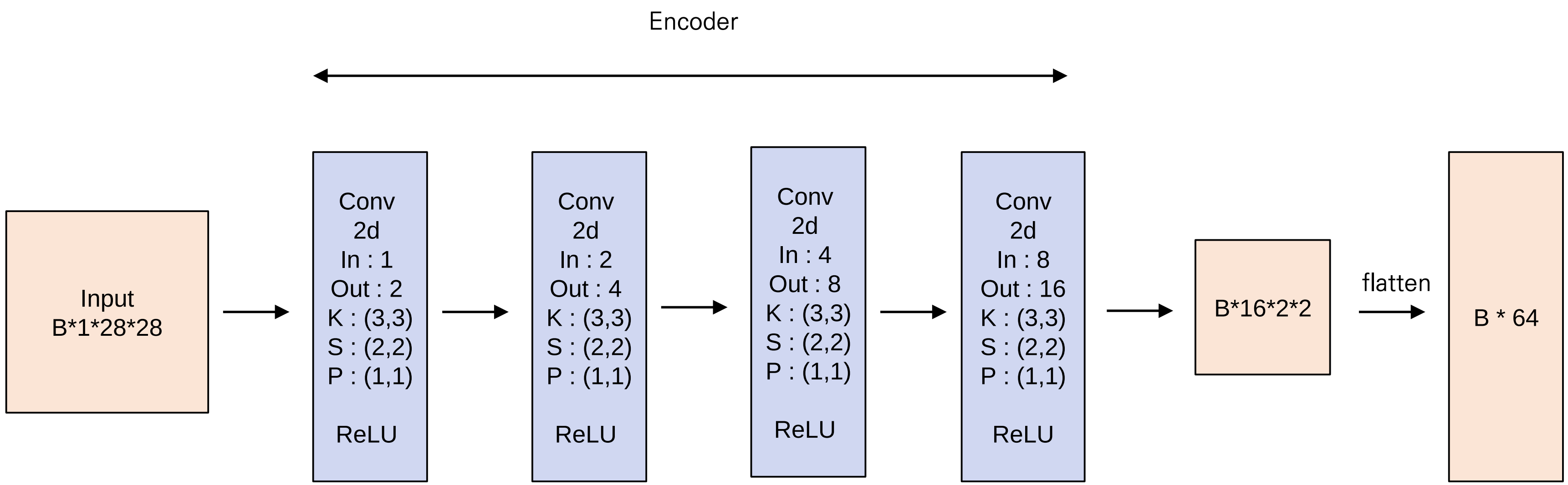
- Conv autoencoder



Practice 1 : Reconstruction

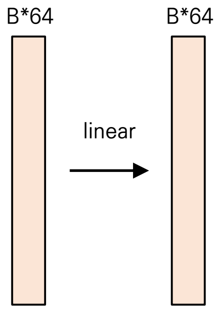


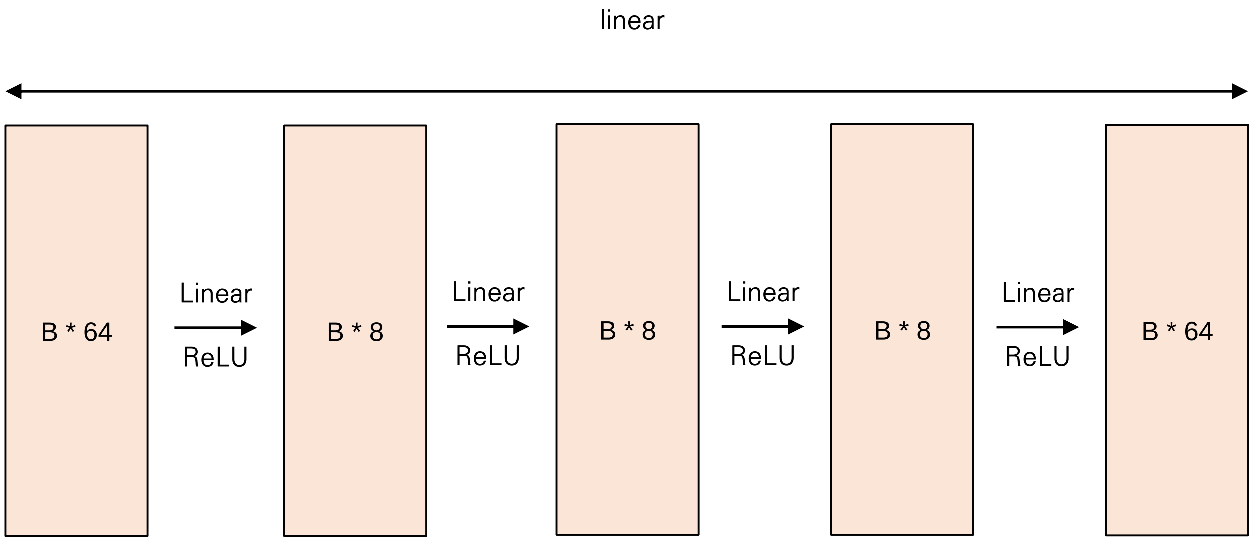
- Conv autoencoder : Encoder (Assignment)



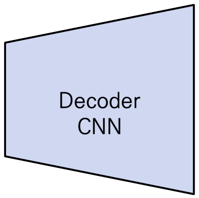
Practice 1 : Reconstruction

- Conv autoencoder : Linear (Assignment)

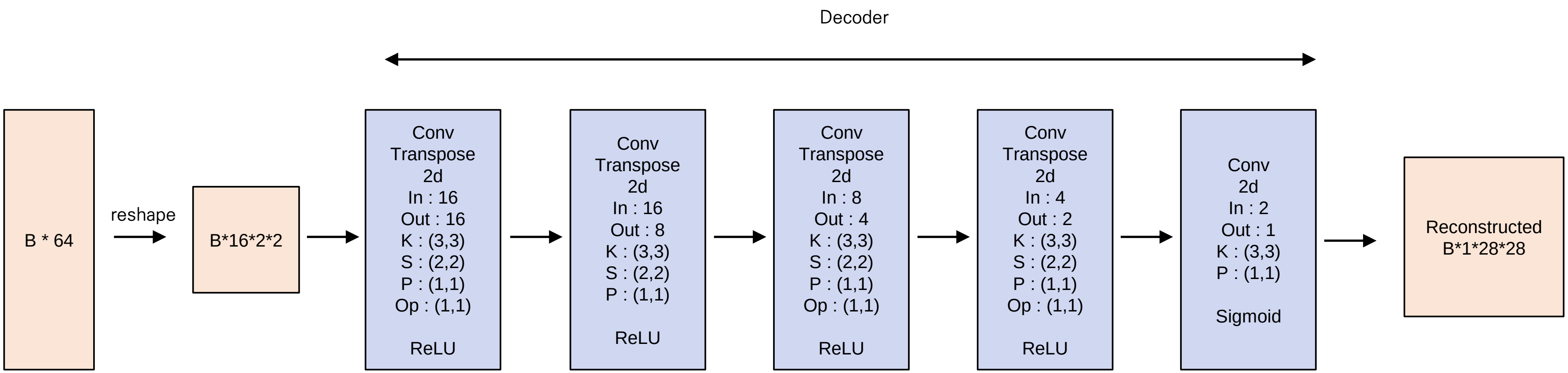




Practice 1 : Reconstruction

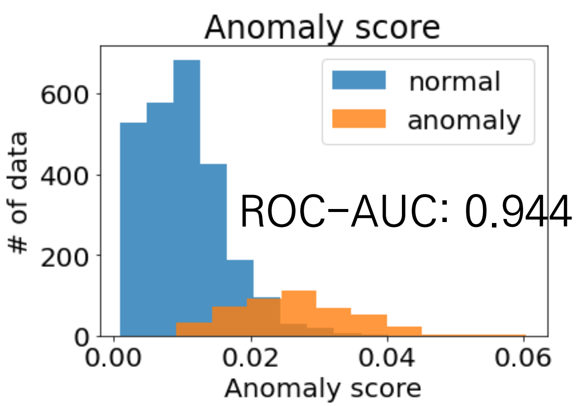


- Conv autoencoder : Decoder (Assignment)

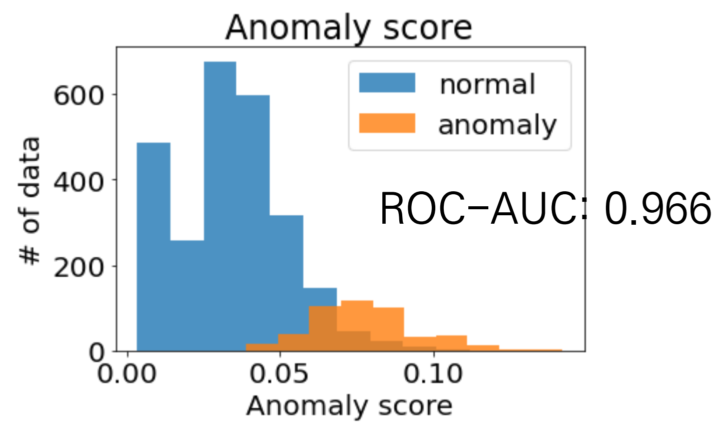


ConvAutoEncoder

A0 vs N1,2,3,4,5



Autoencoder



ConvAutoencoder