HW2: Convolutional Autoencoder

王昊文

2021 年 12 月 15 日

1 Overview

In this homework, I will be training a simple autoencoder, that will extract latent codes of the **Wafer map** dataset. Our goal is to generate simialr images to the wafer dataset, by adding Gaussian noise into the latent code.

The whole homework strictly follows the object-orientated philosophy, which treats *datatset*, *data loader*, *trainer*, *logger* as different objects, so that it can be very convinient to write your own custom objects. You can combine all your settings and put them under `config.json` inside the root folder.

Also, you will find that the structure is highly similar to pytorch template, which allows me to have a basic building block to build this project.

2 Details

2.1 Data

In the Wafer dataset, we have a total of 1281 of data, testing data is not required in this project since we have to generate images directly from the training data.

Each image has a height and width of 26, and each image contains three channels, however the three channels does not indicate RGB, but denotes whether the region is *defected* , *normal* or it is the *boundary* of the wafer. Each channel can only be boolean values, that is, only 0 or 1, to indicate the status of the area under.

There are 10 types of the wafer, there will be an obvious architecture between the 10 types of wafer. We will demonstrate the results from 10 different types of wafer in the result section.

2.2 Model

2.2.1 Intuition and loss function selection

Recall that our task is to generate similar images by adding Gaussian noise into the latent code generated by the autoencoder. Hence, when training, we try to let the output image be identical to the input image. Note that the model can be easily learning identity if our encoder and decoder have the same architecture, hence we must use an appropriate architecture design for our encoder and decoder.

In this case, for image data, convolution neural network on the decoder side and transpose-convolution at the encoder side fits well for this task. Since we are trying to match the input and output image, **MSE loss** will be used in this homework.

2.2.2 Architecture

The model architecture is shown in figure 1. Note that is defined under `model.py`.

In figure 1, the yellow module shows a convolution or a transpose-convolution(for convinience, I denoted it as DeConv). Each convolution and transpose-convolution has a stride of 1 and a kernel size of 3, by setting it this way, we can prevent the shape from decreasing. One thing to note is that we will add a maxpooling at the decoder side and an upsampling at the encoder to form a structure for autoencoders, hence at every convolution/transpose-convolution, I will increase/decrease the output channel, to preserve more information with response to the decrease/increase of the output size.

The blue module denotes either a Maxpooling or upsampling module. More details are listed in table 1.

2.3 Training and validation

As shown in figure 2, the loss curve is plot via tensorboard. Note that the horizontal axis is drawn in unit of steps. When training, I set the batch size to 128. Since there are 1281 samples of the training data,

Layer	Output Shape	Param Num.
Conv2d-1	[128, 16, 26, 26]	448
BatchNorm2d-2	[128, 16, 26, 26]	32
ReLU-3	[128, 16, 26, 26]	0
Conv2d-4	[128, 32, 26, 26]	4,640
BatchNorm2d-5	[128, 32, 26, 26]	64
ReLU-6	[128, 32, 26, 26]	0
Conv2d-7	[128, 64, 26, 26]	18,496
BatchNorm2d-8	[128, 64, 26, 26]	128
ReLU-9	[128, 64, 26, 26]	0
MaxPool2d-10	[128, 64, 13, 13]	0
ConvTranspose2d-11	[128, 32, 13, 13]	18,464
BatchNorm2d-12	[128, 32, 13, 13]	64
ReLU-13	[128, 32, 13, 13]	0
ConvTranspose2d-14	[128, 16, 13, 13]	4,624
BatchNorm2d-15	[128, 16, 13, 13]	32
ReLU-16	[128, 16, 13, 13]	0
Interpolate-17	[128, 16, 26, 26]	0
ConvTranspose2d-18	[128, 3, 26, 26]	435
BatchNorm2d-19	[128, 3, 26, 26]	6
ReLU-20	[128, 3, 26, 26]	0

Table 1: A list of the layers of the current model

that is approximately 11 steps for epoch, and this curve is drawn in total of 150 epochs, that is around 1.6k steps. During training, I used adam as the optimizer, with weight decay of 0.001. Also, the learning rate is scheduled to decay at a rate of 0.5 for every 40 epoch, in order to further decrease the loss value.

2.4 Performace

I have provided a best model `checkpoint-epoch300.pth` and the config `config.json` under the root folder. You can easily visualize the results by using the `test.py` given under the root folder. Several images are already stored in `img/` folder, you can see the quality of the image generated is pretty good.

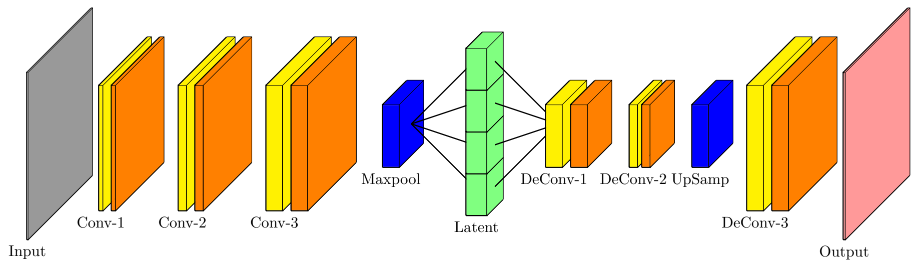

Figure 1: High-level of our model architecture

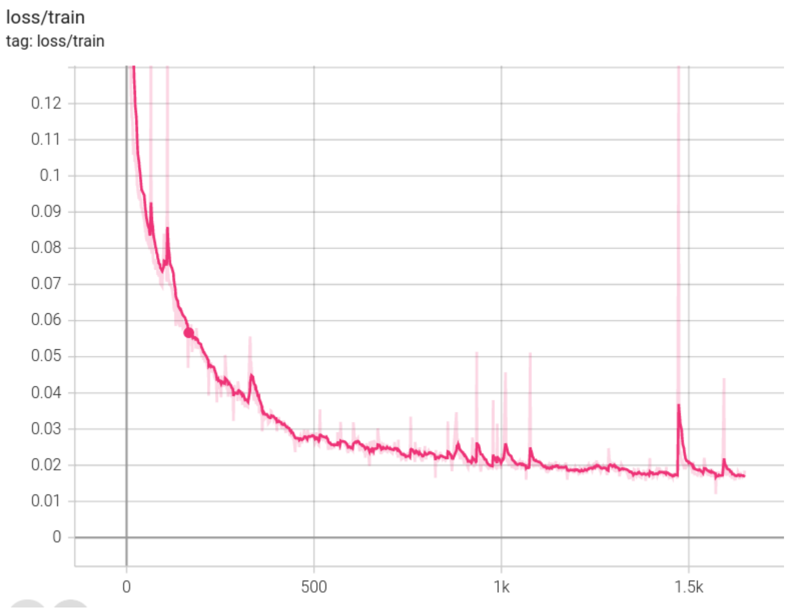

Figure 2: The MSE loss curve of the model

3 Result Visualization

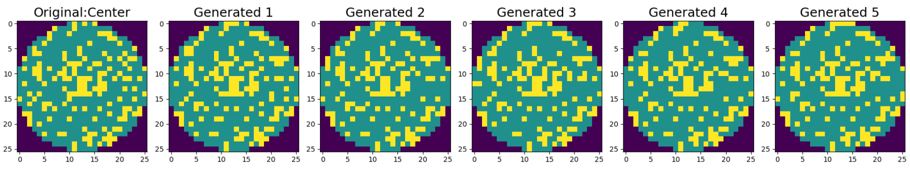

Figure 3: Center

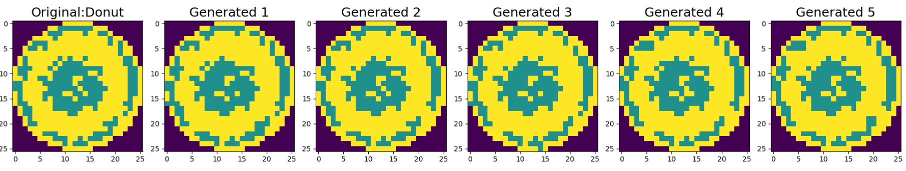

Figure 4: Donut

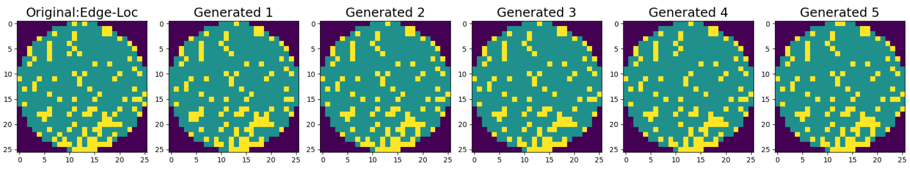

Figure 5: Edge-Loc

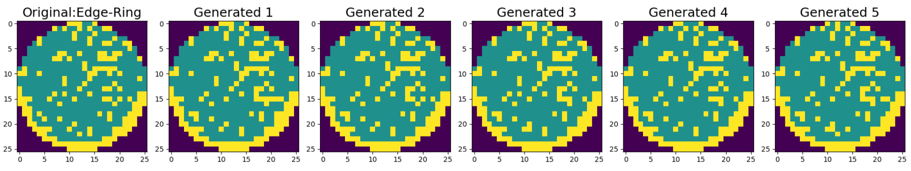

Figure 6: Edge-Ring

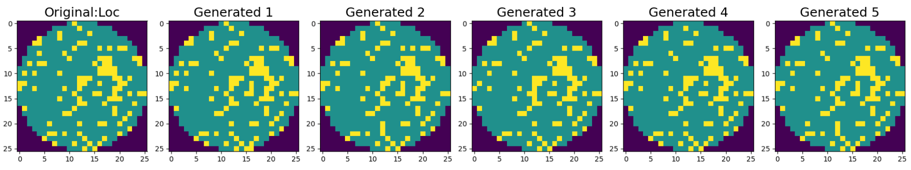

Figure 7: Loc

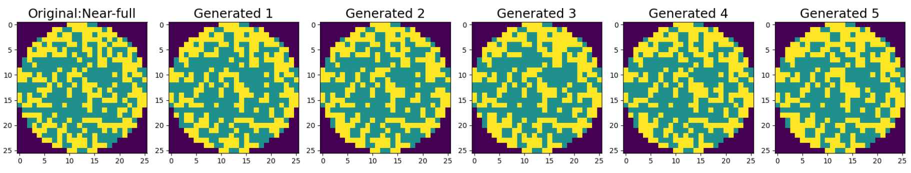

Figure 8: Near-full

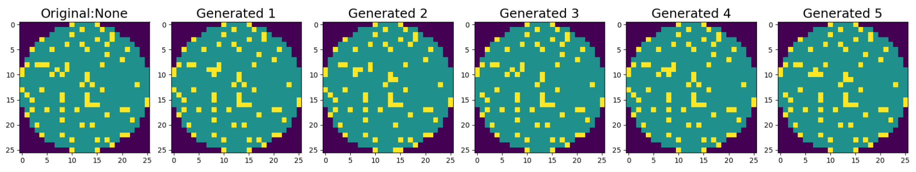

Figure 9: None

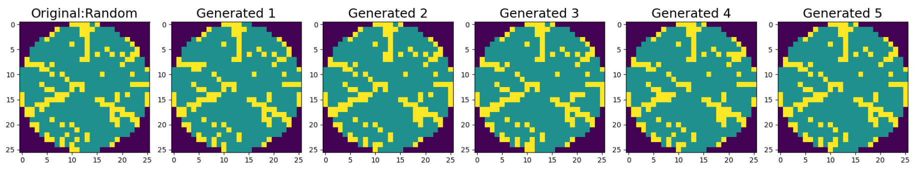

Figure 10: Random

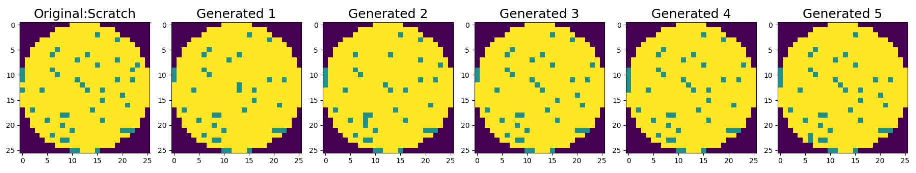

Figure 11: Scratch