Fabian Scheipel, BSc

Pattern recognition on parametric wafer maps using generative adversarial networks

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor

Dipl.-Ing. Dr.techn. Roman Kern

Institute for Interactive Systems and Data Science
Head: Univ.-Prof. Dipl.-Inf. Dr. Stefanie Lindstaedt

Graz, May 2021

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date	Signature

Abstract

During manufacturing process in semiconductor industry, a large amount of data is produced. Measurements can be visualized with so called wafer maps. Different problems in manufacturing process may cause a decrease in production yield. Very often, patterns on the wafer map may indicate the problems in an early stage. A malfunctioning production tool may therefore be indicated by a well-known pattern. For standard pattern recognition tasks, a set of labelled data is needed in order to train classification architectures. This master thesis proposes a method using a generative adversarial network (GAN) called BigBiGAN, which creates a low dimensional representation of the input wafer map. By applying different clustering methods, sets of similar wafer maps can be clustered without prior knowledge. If the network is correctly parametrized, the data is properly pre-processed and the clustering methods are well suited, it is possible to generate labelled data sets from real world data of any size. This approach is able to replace the manual creation of data sets, which is a time consuming and error prone task.

Contents

1 Introduction

Semiconductor industry produce a large amount of data while manufacturing process. There are several methods to use this mass of data, for example standard statistical process control (SPC). The main purpose of evaluating the data is to increase the yield of the production. SPC provides several statistics of all different tests measured in the production cycle. Engineers on site need to use this information to make further classification. For example, if a test exceeds some limit or shows a strange behavior, human eye needs to classify which problem might have occurred. Therefore, SPC is mainly used for process stabilization and not directly as a quality measurement.
The aim of this work is, to create a tool based on machine learning algorithms, that recognizes different patterns on so called wafer maps. Therefore, it will be necessary

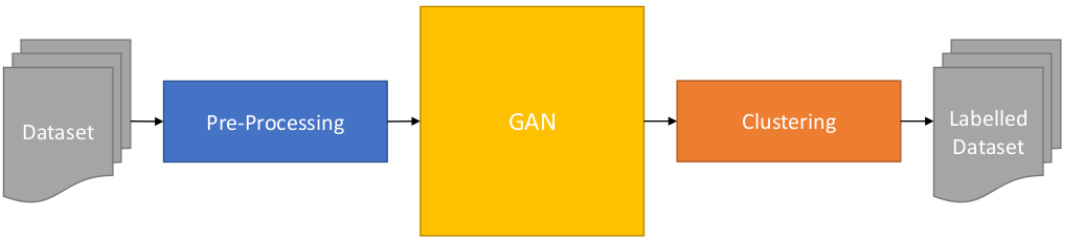


Figure 1.1: Schematic view of the process from data set to output labels. First step is, to pre-process the given data set followed by training a generative adversarial network (GAN), the encoder's output delivers a feature representation which can be clustered into different groups. This groups are used for labelling the input data.

to find out, which patterns are usually seen on wafers. Another open question is, how to handle patterns that have not been seen before but may also cause problems. Next to semiconductor specific questions, common neural network (NN) issues will arise, as missing data handling, training data composition, over-fitting and under-fitting issues and many more.

2 Background

As wafer maps are two dimensional arrays of values with multiple channels, standard pattern recognition tasks as applied to images can be used. The main difference between wafer maps and images is, that there are regions on the wafer map image that consists no data [1] [2] (e.g. not tested parts, regions outside the outlining circle). Also, values on the wafer maps may be either parametric values, which can be of any magnitude, labeled (binned) values which has no numeric quantity, or binary (pass/fail) values. This are the reasons, why pre-processing (see Figure 2.1) of the data is a major task for any pattern recognition task regarding wafer maps.

2.1 Wafer map images

In semiconductor industry, rectangular micro chips are processed on circular, so called wafers. During and after production progress, each micro chip is tested by various electronic or mechanical measurements. In order to visualize the test parameters, so called wafer maps are commonly used. Wafer maps $\mathcal{W} \in \mathbb{R}^{m \times n \times p}$ are two dimensional maps of size $m \times n$ of multidimensional p measurement values. As there are also binary or labelled (binned) measurements, a mapping from $\mathbb{Z} \rightarrow \mathbb{R}$ is necessary in this cases. Each individual measurement matrix $\mathcal{W}_{p_i} \in \mathbb{R}^{m \times n}$ can be treated as a gray scale image. In order to be able to transform a wafer map to the

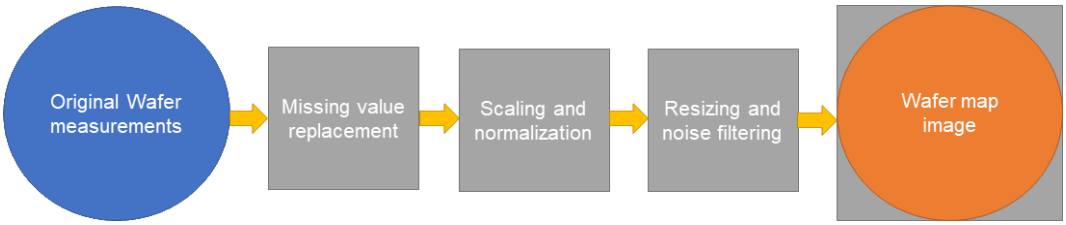


Figure 2.1: Full pre-process chain. Having original wafer measurements, some missing value replacement is necessary. As measurements are of various scale, some scaling and normalization method is required, to make the wafer map images comparable and usable. In order to fit into fixed input sizes of e.g. convolutional neural networks (CNNs) the images need to be resized accordingly.

image space, missing data replacements need to be considered. Wafer maps are in circular shape and outside the circular boundary there are no measurement values. Also chips may be skipped while testing or testing process as a whole is stopped after a certain threshold of failing chips.

2.1.1 Missing data

As an image pixel always consists of values in the interval $[0, 1]$, no missing data can be visualized without a corresponding replacement. A replacement $\rho(\mathbf{p}) = f(\mathbf{p}, \mathbf{q})$ is a function which is defined $\forall \mathbf{q} \in \omega_{\mathbf{p}}$ over a window $\omega_{\mathbf{p}} = \{q : \|\mathbf{p} - \mathbf{q}\|_\infty \leq r\}$ centered around a missing value pixel $\mathbf{p}$ with a radius r. There are multiple ways of defining f and r.

- Chip value
  If the missing chip value $\mathbf{p}$ is inside the circular boundary of a wafer map, it can be useful to replace the value with a dynamic function f within the window $\omega_{\mathbf{p}}$ (small r).

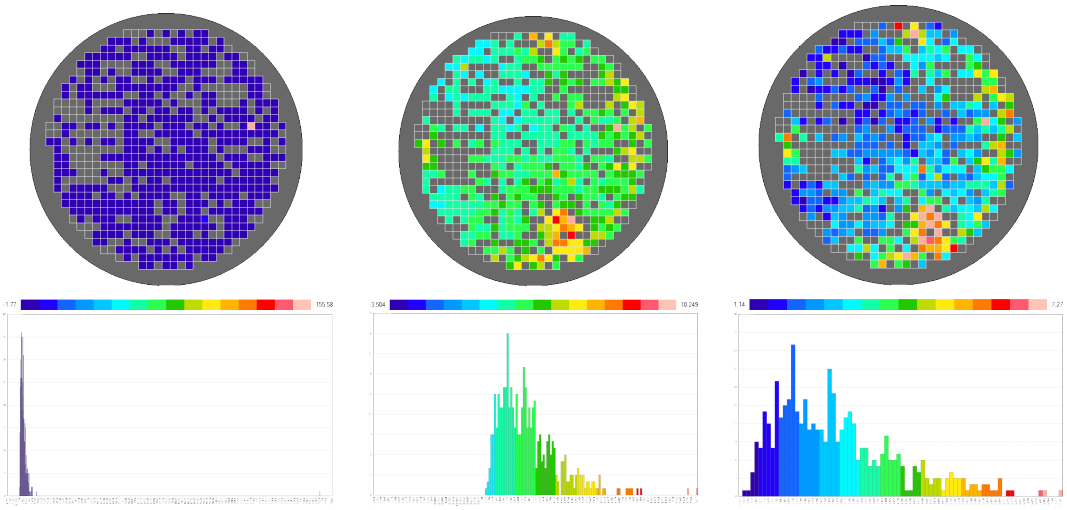
A missing chip value can be defined as erroneous or average chip. Depending on this definition, a function f can be chosen.
Examples:

  - Erroneous: $f(\mathbf{p}, \mathbf{q}) = \max\left(\mathcal{W}(\mathbf{q})\right) \; \forall \mathbf{q} \in \omega_{\mathbf{p}}$
  - Average: $f(\mathbf{p}, \mathbf{q}) = m(\mathcal{W}(\mathbf{q})) \; \forall \mathbf{q} \in \omega_{\mathbf{p}}$, where $m(\mathcal{W}(\mathbf{q}))$ is the *Median* value of the window.

- Background value
  If a missing data value $\mathbf{p}$ is outside of the circular boundary of a wafer map, it is not necessary to replace the value with a dynamic function f in a narrow neighborhood. Therefore a function f across a globally defined window (whole wafer map, $r = \infty$) will be used.
  Background missing values can be defined constantly in order to highlight the background equally in each wafer map. Another approach would be, to define the background as an average chip.
- Additional note to average chips
  The method f can also be a static value across multiple wafer maps. An example would be the median value of all measurement values of the same parameter.

2.1.2 Scaling and normalization

Gray scale images $\mathbf{I} \in \mathbb{R}^{m \times n}_{[0,1]}$ are defined in the interval $[0, 1]$, therefore a normalization operation $\mathbf{n} : \mathbb{R}^{m \times n} \to \mathbb{R}^{m \times n}_{[0,1]}$ is needed in order to transform wafer maps to the image space. As wafer map measurements contain outliers, a data scaling operation $\mathbf{s} : \mathbb{R}^{m \times n} \to \mathbb{R}^{m \times n}$ is applied to the original measurements in order to focus on statistically significant results.



(a) Full range wafer map $\mathcal{W}_{p_i}$ with corresponding histogram. The coloring of the wafer map image is based on the full range. Due to a small amount of outliers, quantization effects and limited color range lead to a mainly blue wafer map image. Therefore, no patterns are visible for the human eye. This is also visible in the histogram, having a small peak at the beginning, followed by no or very small counts at the rest of the histogram

(b) Scaled wafer map $\mathcal{W}_{p_i}^{\sigma}$ with corresponding histogram. The coloring of the wafer map image is based on the mean $\overline{\mathcal{W}_{p_i}}$ and a multiple k of the standard deviation $\sigma(\mathcal{W}_{p_i})$. As the outliers still have an effect on the coloring, not the whole color space is used, but some patterns can be seen. The histogram (e) shows, that lower values are not good represented.

(c) Scaled wafer map $\mathcal{W}_{p_i}^{IQR}$ with corresponding histogram. The coloring of the wafer map is based on the Interquartile range (IQR). As outliers are typically within the first or the fourth quartile, the color range can be used in a better way. Now also patterns in the blue area can be visualized better, which can be seen at the histogram (f), which is good balanced.

Figure 2.2: Different scaling modes of wafer map images.

As shown in Figure 2.2, different scaling modes can be applied. Figure 2.2a shows a wafer map visualized with its full range and its equally scaled histogram. Compared to Figures 2.2b and 2.2c, no patterns can be seen on the wafer map. This is due to the fact, that some values on the wafer map are outliers, which scales the significant values in an inappropriate way. Therefore, statistically meaningful scaling methods needs to be applied as shown in Figures 2.2b and 2.2c. It can be seen, that if the histogram is equally filled, more detailed information can be visualized from the

image. This procedure is also called *histogram equalization* in image processing [3]. Figure 2.2b shows the scaling operation σ_k defined in Equation 2.1.

$$\sigma_k : \mathcal{W}_{p_i}^{\sigma_k}(\mathbf{x}) = \begin{cases} \overline{\mathcal{W}_{p_i}} - k\sigma(\mathcal{W}_{p_i}), & \text{if } \mathcal{W}_{p_i}(\mathbf{x}) < \overline{\mathcal{W}_{p_i}} - k\sigma(\mathcal{W}_{p_i}) \\ \overline{\mathcal{W}_{p_i}} + k\sigma(\mathcal{W}_{p_i}), & \text{if } \mathcal{W}_{p_i}(\mathbf{x}) > \overline{\mathcal{W}_{p_i}} + k\sigma(\mathcal{W}_{p_i}) \\ \mathcal{W}_{p_i}(\mathbf{x}), & \text{otherwise} \end{cases} \tag{2.1}$$

, where $\overline{\mathcal{W}_{p_i}}$ is the *Arithmetic Mean* and $\sigma(\mathcal{W}_{p_i})$ is the *Standard Deviation* [4] of the wafer map values.

Figure 2.2c shows the scaling operation IQR defined in Equation 2.2.

$$\text{IQR} : \mathcal{W}_{p_i}^{IQR}(\mathbf{x}) = \begin{cases} Q_1(\mathcal{W}_{p_i}), & \text{if } \mathcal{W}_{p_i}(\mathbf{x}) < Q_1(\mathcal{W}_{p_i}) \\ Q_3(\mathcal{W}_{p_i}), & \text{if } \mathcal{W}_{p_i}(\mathbf{x}) > Q_3(\mathcal{W}_{p_i}) \\ \mathcal{W}_{p_i}(\mathbf{x}), & \text{otherwise} \end{cases} \tag{2.2}$$

, where $Q_1(\mathcal{W}_{p_i})$ is the *First Quartile* and $Q_3(\mathcal{W}_{p_i})$ is the *Third Quartile* of the wafer map values. The difference between Q_3 and Q_1 is called *Interquartile Range* (IQR) [5].

In order to transform the data to the gray scale image space in the interval $[0, 1]$ a normalization operation is needed. This can be achieved by the normalization operation $n : \mathcal{W}_{p_i}^n = \frac{\mathcal{W}_{p_i}^s - \min(\mathcal{W}_{p_i}^s)}{\max(\mathcal{W}_{p_i}^s) - \min(\mathcal{W}_{p_i}^s)}$, which is applied on the scaled wafer map image $\mathcal{W}_{p_i}^s$, where s can be an arbitrary scaling operator (e.g. σ, or IQR).

As wafer map measurements contains noise, also noise cancellation can be applied.

2.1.3 Additional pre-processing steps

In order to be able to compare wafer maps that differs in size and to feed the images to image processing algorithm or deep learning networks, some image scaling is

required. Therefore, standard interpolation [6] modes from image processing can be applied. Popular interpolation modes are *Bilinear interpolation* [7] or *Nearest neighbor interpolation* [8].
The last step of the wafer map image pre-processing chain is *Noise cancellation*. As wafer map measurements are not ideal, noise cancellation can improve results significantly. The *Gaussian Filter*[9] is used to eliminate normally distributed noise. An optimization on the gaussian filter is the *Bilateral filter*. For local outlier filtering, the *Median filter* can be used. These filtering methods are standard methods in classical image processing. As the pre-processing chain results in a gray scale wafer image, standard image processing steps can be applied for pattern recognition as well.

2.2 Classical pattern recognition

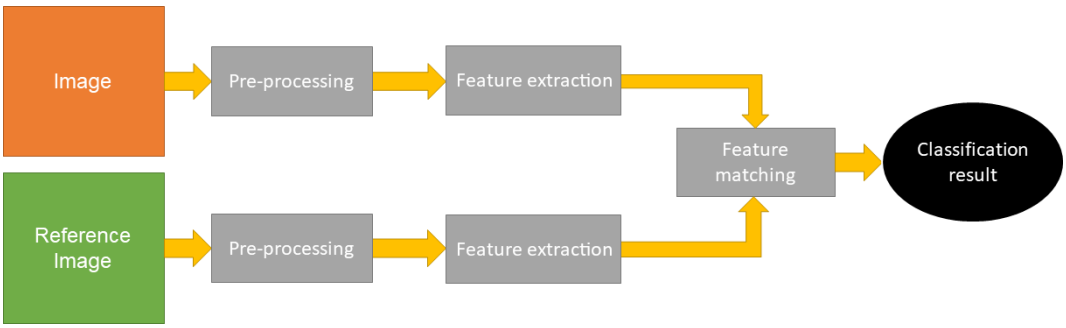


Figure 2.3: Schematic pattern recognition chain. For both, image and reference image, the same process is done, starting with pre-processing in order to fit to the needs of the problem (missing value replacement, scaling, normalization, resizing). After feature extraction, image features can be compared using feature matching algorithms to get a classification result.

As shown in Figure 2.3, standard pattern recognition consists of three parts. The first part is the pre-processing. This was already shown in 2.1. The second step

is the feature extraction step. In order to match a input image with a reference, similarities between images need to be found. As images consist of pixel values which represent color only, differences in illumination or orientation of the same objects in input and reference image makes it hard to find similarities on pixel level. Therefore, so called *Feature matching*[10] approaches are used. In order to match features from one image with another, some *Points of interest* (POI) or *Feature Points* (FP) needs to be detected. Depending on the algorithm used, FPs can for example consist of edges, corners (e.g. FAST [11], Harris Corners [12] or blobs (e.g. Difference of Gaussian (DoG) [13]). To be able to compare FPs between images, a descriptive information for a FP is needed. Therefore, *Feature Descriptors* (FD) need to be calculated for each FP. FDs are computed from a defined region around a FP. Popular algorithms are e.g. SIFT [14] or BRIEF [15]. FDs result in a so called *Feature Vector* (FV). For *Feature matching*, FVs are compared using some distance operator (e.g. $\mathcal{L}_2$ norm [16]).

In the last recent years, neural networks (NNs) outperformed classical pattern recognition techniques. For this thesis, convolutional neural networks (CNNs) are described.

2.3 Feed forward neural networks (NNs)

NNs [17] are quintessential deep learning methods used for learning an approximation of a function f^*. A NN maps an input $\mathbf{x}$ to an output $\mathbf{y}$ defined by a mapping function $f(\mathbf{x}, \theta)$. The goal of training a NN is to find out the best matching parameters θ for the approximation. Usually, multiple functions $f^{(0)}$, $f^{(1)}$, to $f^{(n)}$ are combined in a feed forward manner (see Figure 2.4), which means that no feedback loops are included within the network. The functions are commonly called *layers*.

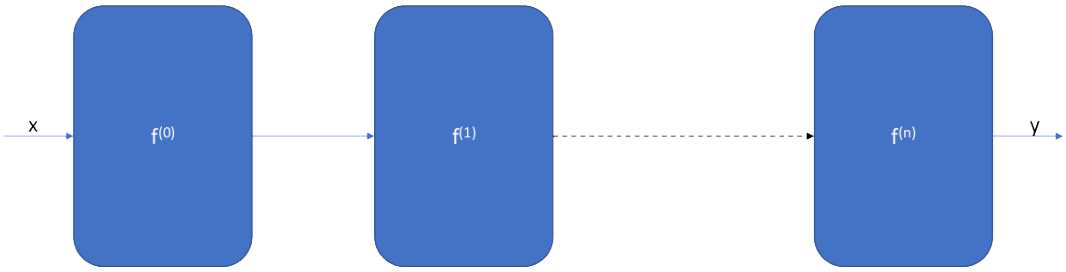


Figure 2.4: Feed forward network consisting of multiple mapping functions f without feedback loops. The set of mapping functions try to approximate the function f^* which can be achieved by training the function's parameters θ.

NNs are also called multi layer perceptrons (MLPs). MLPs are combinations of single layer perceptrons (SLPs) as shown in Figure 2.5. SLPs are also called *Neurons*.

2.3.1 Neurons

Each layer in a NN consists of a defined number of SLPs, where its inputs are coming from the previous layer and the outputs are passed to the next layer.

The characteristic of a SLP is, that the inputs from the previous layer are *weighted* and *biased* before entering a non-linear *activation function*. The approximation function $\mathbf{y} = f^*_{SLP}(\mathbf{x})$ can then be computed using a vector dot product ($\mathbf{y} = a(\mathbf{w}^\top \mathbf{x} + b)$). The weighting vector $\mathbf{w}$ and the bias value b are initially unknown and need to be trained accordingly in order to fit the approximation function f^*_{SLP}. Therefore, the trainable parameters are defined as $\theta_{SLP} = \{\mathbf{w}; \mathbf{b}\}$.

Activation functions

As the original function f to approximate can be any kind of mathematical function, also non-linear functions should be possible to approximate. As weighting and

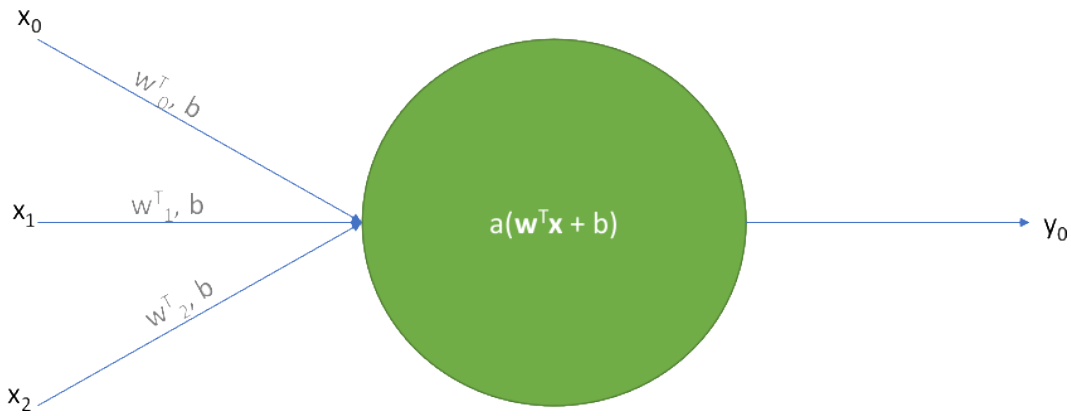
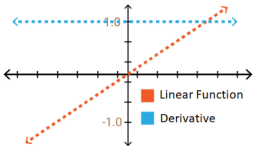


Figure 2.5: A single layer perceptron (SLP) weighs the components of an input vector $\mathbf{x}$ with the weight $\mathbf{w}$ by a vector dot-product and adds a bias b. The result is fed into a (piece-wise) differentiable activation function a which results in an output scalar value y_0

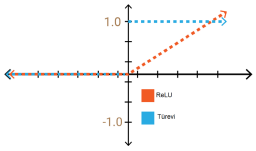
biasing are linear mathematical operations, only a linear regression [18] would be capable without adding some non-linear activation function. There are various activation functions as shown in Figure 2.6. Activation functions need to be (piece-wise) differentiable, as neural networks are usually trained using gradient descent [19] methods.

2.3.2 Training a neural network

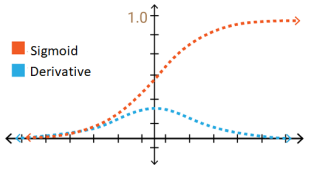
In order to train a NN, a *loss function* $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ needs to be defined. $\mathcal{L}$ describes the difference between the approximated output vector $\mathbf{y}$ and the desired ground truth vector $\hat{\mathbf{y}}$. A commonly used loss function is *Mean Squared Error* (MSE), which is defined as $\mathcal{L}_{MSE}(\mathbf{y}, \hat{\mathbf{y}}) = \|\mathbf{y} - \hat{\mathbf{y}}\|^2$. Training a NN means, to minimize the loss function $\mathcal{L}$ with respect to the network parameters θ.



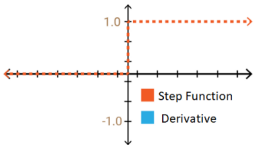
(a) Linear activation function with a constant derivative for all input values.



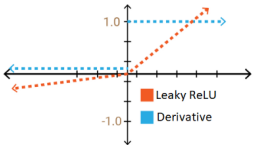
(b) Rectified Linear Unit (ReLU) which is zero for negative input values and a linear function for positive input values. The derivation is constant for positive input values and zero for negative ones.



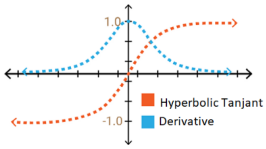
(c) Sigmoid activation function $\sigma(x) = \frac{1}{1+\exp^{-x}}$. Its derivative $\sigma'(x) = \sigma(x)(1 - \sigma(x))$ is defined for all input values.



(d) The step activation function is zero for negative input values and one for positive input values. Its derivation is zero for all values except the input value zero. In this single point the step functions' derivation is not defined.



(e) The Leaky ReLU function is similar to the ReLU function. The difference is, that also for negative input values, a linear function is defined with a lower slope. The derivation is a small constant for negative values and a higher constant for positive values.



(f) Hyperbolic tangent function $\tanh$ is a trigonometric function defined for all real input values. Its corresponding derivative is $\tanh' = \frac{1}{\cosh x^2}$ which is globally defined.

Figure 2.6: Different activation functions [20] including the corresponding derivatives.

Backpropagation and optimization

Minimizing a loss function can be achieved by applying gradient based methods [19]. The principal concept of gradient methods is, to change the desired parameter value in the direction of a local minimum. Local minima are defined as $\frac{\partial \mathbf{y}(\theta_0)}{\partial \theta} = 0$. Derived from linear approximation, the gradient descent method can be applied as $\theta^{i+1} = \theta^i - \gamma \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \theta}$. γ is defined to be the *Learning rate*. This value controls the speed of learning. If it is chosen too high, it can happen that no local minimum can be reached at all. If it is chosen to small, then training can be very slow. A major criterion for training NNs is to choose learning rates properly.
As shown in the gradient descent calculation, $\mathbf{y}$, as well as the loss function $\mathcal{L}$ need to be differentiable with respect to the network parameters θ. Furthermore, this requires also the activation functions to be differentiable with respect to θ. Another observation is, that the weighting matrices and the bias vectors within θ are distributed through the network in a feed forward way. Applying the gradients using gradient descent, can therefore only achieved in an iterative manner. This procedure is called *backpropagation* [21].
It can be shown, that if every part of the network is partially differentiable, all derivations of the loss functions with respect to each entry of θ can be calculated iteratively from the back to the end. This can then be used to apply gradient descent in order to train the parameters.

Application of neural networks

There is a variety of applications for neural networks. The general approach to approximate some nonlinear function with a neural network can be applied in different ways. The different approaches are varying mainly in the usage of different components of the network. Variations are possible in choosing different depths of the network, different activation functions, especially for output neurons or

different loss functions.
The *depth* of a network is defined by its number of *hidden layers* and the corresponding number of neurons per hidden layer. Hidden layers is a different name for intermediate layers between input and output neurons. As input and output neurons are given from input and output vectors, the number of hidden neurons per hidden layer can vary. Depending on the complexity of the function to approximate, the depth of the networks can be chosen.
By choosing different output activation functions, different kinds of problems can be addressed. For classification tasks, some output representing a percentage is used (e.g. Sigmoid). For regression tasks, a more linear activation function can be chosen (e.g. ReLu). Also the definition of the loss functions depend on the function a network is approximating. An example for classification tasks is the cross entropy loss function [22]. For regression tasks MSE can be chosen.
Besides variations of the simple feed forward neural networks, there are also some advanced topologies like convolutional neural networks (CNNs). They are commonly used in grid like data sets like images or regular time intervals.

2.4 Convolutional neural networks (CNNs)

Convolutional neural networks (CNNs) [23] are feed forward neural networks replacing the matrix multiplication of a NN with a convolution function at least in one of its layers.

2.4.1 Convolution function

As neural networks consists only of discrete components, the *discrete convolution* function s is applied. In 1D it is defined as $s(t) = \sum_{k=0}^{W-1} \tilde{\mathbf{x}}[k]\mathbf{w}[t-k]$, where $\tilde{\mathbf{x}}$ consists of the input data $\mathbf{x}$ and padded zeros to match the convolving kernels $\mathbf{w}$ size

W. For image processing, the 2D version of the convolution function S is needed. It is defined as $S(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \tilde{\mathbf{X}}[i-m, j-n]\mathbf{K}[m,n]$. $\tilde{X}$ consists of an 2D image array and zeros to match the 2D kernels $\mathbf{K}$ size ($M \times N$) while convolving. Convolution can also be seen as a *sliding window* over the image. Usually, the kernel size $M \times N$ is much smaller than the input size. The convolution function can also be written as a matrix multiplication, where the kernel matrix has a very special characteristic called *Toeplitz matrix* [24]. A row of a *Toeplitz matrix* is defined to be equal to the row above, shifted by one element. As the kernel $\mathbf{K}$ is usually of much smaller dimension than the input size of $\mathbf{X}$ the *Toeplitz matrix* consists of mostly zero values. Also the fact that rows have equal entries defines this matrix to be very sparse. By writing the convolution as a matrix multiplication, the relationship to the base NNs is shown.

2.4.2 Convolutional layers

A NN is defined to be a CNN if at least one of its layer consists of a convolutional layer. For image processing, only the 2D variant of the convolution function is required. A 2D convolutional layer consists of multiple kernel matrices $\mathbf{K} \in \mathbb{R}^{M \times N}$, where $M \times N$ defines the kernel size.
As described in 2.4.1, instead of a normal matrix multiplication, the kernel matrices are applied in a sliding window approach. As the kernel sizes are usually much smaller than the actual input sizes, the number of trainable parameters, within the network decrease significantly. Smaller matrices also lead to faster training.
The output of a convolutional layer consist of so called *feature maps*. This comes from traditional image processing, as the output of a trained networks convolutional layer represent significant shapes on spatial positions similar to edge detectors or other feature points. For classification tasks, CNNs need to abstract the information from the input to a classification label at the output. So convolutional layers at the front of the network deliver more information at pixel level. Layers at the end

of a network deliver more information at object level. This can be achieved by adding pooling layers to the network. Pooling layers reduce dimensionality with the goal to add abstraction to a CNN by simply dropping information in a well-defined way. Again, by application of a sliding window approach, the dimension of an input gets decreased. One example is *max-pooling*. Here, only the largest value within the pooling windows is used at the output. For information dropping, multiple approaches can be used. Further examples would be *average-pooling* or *median-pooling*.

Application on image data

For applying CNNs on image data to perform classification tasks, most of the networks used consist of multiple convolutional layers at the beginning, followed by pooling layers. At the end of the networks, usually a dense NN is appended in order to use the highly abstracted output of the last feature maps of the CNN for classification. As shown in Figure 2.3, classical image processing requires some steps that are also used implicitly at CNNs for classification tasks. As the classical image processing pattern recognition chain, also pre-processing of input images is required to fit the input size given by the CNN. Convolutional layers provide multiple feature maps, which fits to the *Feature extraction* part. The dense layers at the end of the network fulfill the feature matching tasks. The main difference is, that the network itself has the knowledge of the reference image stored in its weights which is a major advantage of CNNs in pattern recognition. By using inverse, so called deconvolutional layers, CNNs can also be used for regression or generation tasks. Generative adversarial networks make use of this approach.

2.5 Generative adversarial networks (GANs)

Generative adversarial networks (GANs) are competitive NNs. Two or more networks are trying to compete their opponent networks by solving their task better than the other. A simple example is *DCGAN* [25], which consists of two NNs. The first network is the so called generator, the second one is called the discriminator. The generator tries to imitate data from a given data set. The task of the discriminator is to find out whether the input data set is fake or not. The result of this approach is an fake data generator on the one hand and a fake data discriminator on the other hand.
Both generator and discriminator loss depend on the foreign networks, which lead to very dynamic learning curves. Also the networks are very sensitive to be wrongly configured, which can lead to effects like *Mode loss*. *Mode loss* is a state of a network, where always the same result is given, independent on the input. This is often due to having a discriminator, which always beats the generator. In such a case, the best loss a generator can achieve is to generate a static output. This is also called *vanishing gradient problem* which is also well-known in other, non-dynamic multi-layer configurations.
Stability for GANs can be achieved by applying stable loss functions like the Wasserstein loss [26]. The network used for this thesis is a GAN consisting of three NNs, called BigBiGAN.

2.5.1 BigBiGAN

The BigBiGAN network is a generative adversarial network consisting of three major blocks called *encoder* $\mathcal{E}$, *generator* $\mathcal{G}$ and the *discriminator* $\mathcal{D}$. The inputs of the network are a original data set $\mathbf{x}$ and a random vector $\mathbf{z}$. Its outputs are a feature representation $\hat{\mathbf{z}}$ of $\mathbf{x}$ and a fake data set $\hat{\mathbf{x}}$ generated from $\mathbf{z}$. BigBiGAN is usually used in order to generate fake images from a set of original images. It is

more robust against common GAN problems like mode losses or vanishing gradient issues than simpler networks like DCGAN.

The aim of the encoder network, is to learn a feature representation $\hat{\mathbf{z}}$ of the original input $\mathbf{x}$. The output of the encoder network is also called *latent vector* $\mathbf{z}$. The generators goal is to create data $\hat{\mathbf{x}}$ from a (random) feature representation, which is the inverse task of the encoder. Generator and encoder together are trying to trick the discriminator to believe that the generated data is the original one. The discriminator gets either a pair of original data and latent vectors generated by the encoder $(\mathbf{x}, \hat{\mathbf{z}})$ or the opposite combination of generated data by the generator and the corresponding random latent vector $(\hat{\mathbf{x}}, \mathbf{z})$. The data distribution is defined as $\mathbf{x} \sim P_x$ and the latent distribution is defined as $\mathbf{z} \sim P_z$. The discriminator itself consists of three neural networks F, H and J. The F network is usually a CNN with a fully connected layer as an output layer. It uses the original data $\mathbf{x}$ or the generated ones $\hat{\mathbf{x}}$ as inputs and has a scalar vector output $\mathbf{s}_x$. The H network is usually a common NN which uses the latent representations $\mathbf{z}$ or $\hat{\mathbf{z}}$ as an input and results in a vector output $\mathbf{s}_z$. The network J uses both scalar vector outputs $\mathbf{s}_x$ and $\mathbf{s}_z$ as an input and results in the joint score vector output $\mathbf{s}_{xz}$. The three scores $\mathbf{s}_x$, $\mathbf{s}_z$ and $\mathbf{s}_{xz}$ are then used to calculate the corresponding loss functions for all networks. The minimax objective of the BigBiGAN is defined in Equation 2.3.

$$\min_{\mathcal{GE}} \max_{\mathcal{D}} \{\mathbb{E}_{\mathbf{x}\sim P_x, \mathbf{z}\sim \mathcal{E}_\Theta(\mathbf{x})}[\log(\sigma(\mathcal{D}(\mathbf{x}, \mathbf{z}))] + \mathbb{E}_{\mathbf{z}\sim P_z, \mathbf{x}\sim \mathcal{G}_\Theta(\mathbf{z})}[\log(1 - \sigma(\mathcal{D}(\mathbf{x}, \mathbf{z}))]\} \tag{2.3}$$

It can be shown, that the optimized minimax objective minimizes the Jensen-Shannon divergence [28] between the joint distributions $P_{\mathbf{x}\mathcal{E}}$ and $P_{\mathcal{G}\mathbf{z}}$ match at the global optimum. This means, that if the network is ideally trained, the generator

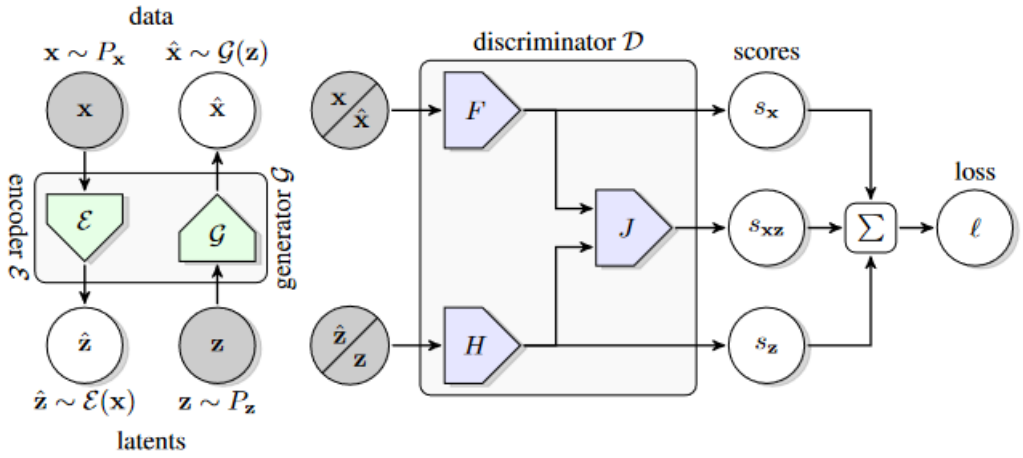


Figure 2.7: Graphical scheme of the BigBiGAN network [27]. It consists of a generator $\mathcal{G}$ and an encoder $\mathcal{E}$ network. The joint discriminator $\mathcal{D}$ computes the loss $\mathcal{L}$ from the encoder and generator outputs as well as the corresponding random vectors used for the generator and the original data source.

and the encoder generate the exact inverse outputs. So, if a original data point is passed to the encoder, the generator re-creates the same original data point using the latent vector output from the encoder. It is also worth mentioning, that the loss function used for training the encoder and the generator only depends on the discriminators outputs.

The equations 2.4, 2.5 and 2.6 define the outputs of the discriminators NN blocks. It can be seen from the equations 2.8 - 2.11 that all loss functions are only dependent on the discriminator output blocks and the data distributions P_x and P_z. y defines a label whether the input data of the discriminator is fake (-1) or real (+1). For calculating the encoder and generator loss $\mathcal{L}_{\mathcal{EG}}(P_x, P_z)$ the single sample loss $l_{\mathcal{EG}}(\mathbf{x}, \mathbf{z}, y)$ is used in a linear way. For computing the discriminator loss $\mathcal{L}_{\mathcal{D}}(P_x, P_z)$ the single sample loss $l_{\mathcal{D}}(\mathbf{x}, \mathbf{z}, y)$ is calculated using the hinge function shown in equation 2.7.

$$s_x(x) = F_\Theta(x) \quad (2.4)$$

$$s_z(z) = H_\Theta(x) \quad (2.5)$$

$$s_{xz}(\mathbf{x}, \mathbf{z}) = J_\Theta(F_\Theta(\mathbf{x}), H_\Theta(\mathbf{z})) \quad (2.6)$$

$$h(t) = \max(0, 1 - t) \quad (2.7)$$

$$l_{\mathcal{EG}}(\mathbf{x}, \mathbf{z}, y) = y(s_x(\mathbf{x}) + s_z(\mathbf{z}) + s_{xz}(\mathbf{x}, \mathbf{z})), \text{where } y \in \{-1, +1\} \quad (2.8)$$

$$\mathcal{L}_{\mathcal{EG}}(P_x, P_z) = \mathbb{E}_{\mathbf{x} \sim P_x, \hat{\mathbf{z}} \sim \mathcal{E}_\Theta(\mathbf{x})}[l_{\mathcal{EG}}(\mathbf{x}, \hat{\mathbf{z}}, +1)] + \mathbb{E}_{\mathbf{z} \sim P_z, \hat{\mathbf{x}} \sim \mathcal{G}_\Theta(\mathbf{z})}[l_{\mathcal{EG}}(\hat{\mathbf{x}}, \mathbf{z}, -1)] \quad (2.9)$$

$$l_{\mathcal{D}}(\mathbf{x}, \mathbf{z}, y) = h(y s_x(\mathbf{x})) + h(y s_z(\mathbf{z})) + h(y s_{xz}(\mathbf{x}, \mathbf{z})), \text{where } y \in \{-1, +1\} \quad (2.10)$$

$$\mathcal{L}_{\mathcal{D}}(P_x, P_z) = \mathbb{E}_{\mathbf{x} \sim P_x, \hat{\mathbf{z}} \sim \mathcal{E}_\Theta(\mathbf{x})}[l_{\mathcal{D}}(\mathbf{x}, \hat{\mathbf{z}}, +1)] + \mathbb{E}_{\mathbf{z} \sim P_z, \hat{\mathbf{x}} \sim \mathcal{G}_\Theta(\mathbf{z})}[l_{\mathcal{D}}(\hat{\mathbf{x}}, \mathbf{z}, -1)] \quad (2.11)$$

It can be shown, that BigBiGAN is very stable according to vanishing gradient problems and mode losses. It is able to deal with large scale input data and has a good reconstruction result. The setting of a competitive generator-encoder network reminds to variational autoencoders (VAEs) which are described in the next chapter.

2.6 Variational auto encoders (VAEs)

A variational autoencoder (VAE) consists of two deep learning networks trying to learn a statistical representation of a data set. The encoder part is trying to output a lower dimensional representation as an feature vector which is called latent vector $\mathbf{z}$. The distribution of all output values of the encoder network is also called (stochastic) *latent space*. The decoder part of the network uses the latent vector $\mathbf{z}$ to reconstruct the corresponding original data sample. Independent on the realization used, the loss functions are always calculated from the difference

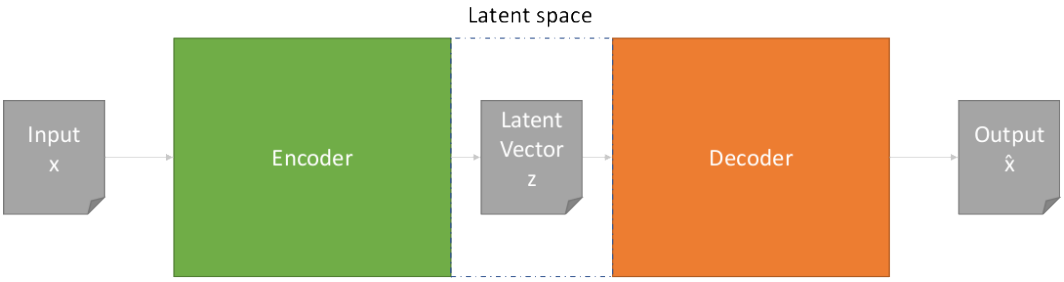


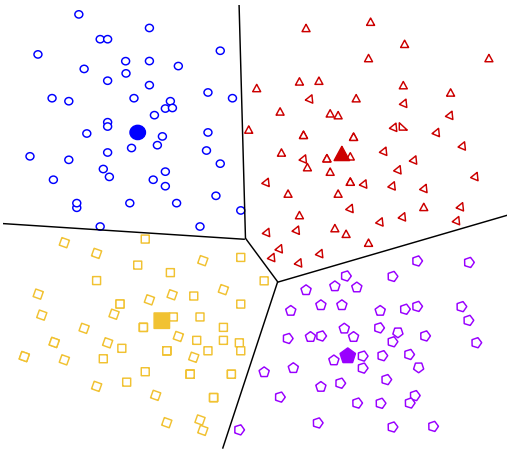
Figure 2.8: Graphical scheme of the VAE network. The encoder outputs a lower dimensional representative (latent vector) $\mathbf{z}$ of the input data $\mathbf{x}$. The encoder uses $\mathbf{z}$ to generate a approximated version $\hat{\mathbf{x}}$ of the original data.

between the decoder output $\hat{\mathbf{x}}$ and the encoder input $\mathbf{x}$. Once the VAE is fully trained, the latent outputs of training data inputs can be used to extract patterns by clustering algorithms. Networks as VAEs are also called competitive networks [29]. Other forms of competitive networks are for example *generative adversarial neural networks* (GANs) [30] which are based on *Turing Learning* [31] approaches. The main difference between GANs and VAEs is in having different loss function approaches. As GANs are only using the information whether the discriminator has been tricked or not, the VAEs are learning the difference between original and reconstructed data. This leads to VAEs having very good results at reconstructing data from the very same distribution. As VAEs are learning how the data is represented, reconstruction differences from data with a similar statistical distribution are lower compared to GANs. If the training data is from a different distribution, VAEs reconstruction results are usually worse than with GANs.

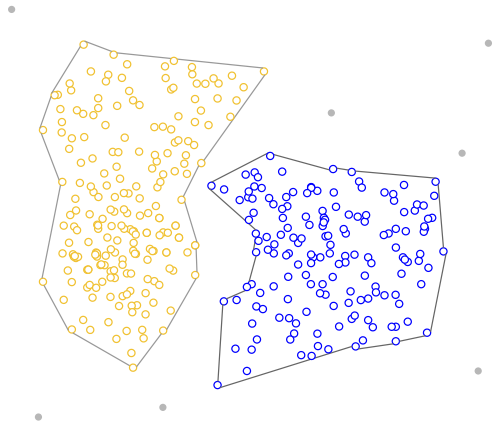
2.7 Clustering algorithms

In order to be able to cluster the feature or latent vectors of network outputs, some algorithms are used. In general, there are several different types of clustering

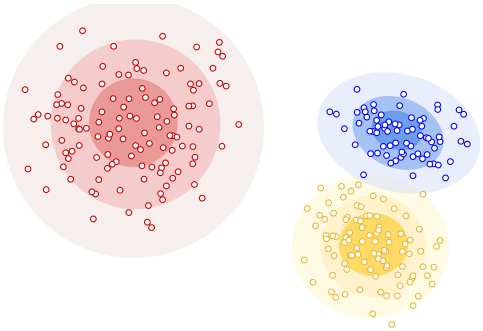
algorithm types [32], where this thesis is focussing on the four most commonly used types [33] as shown in Figure 2.9.



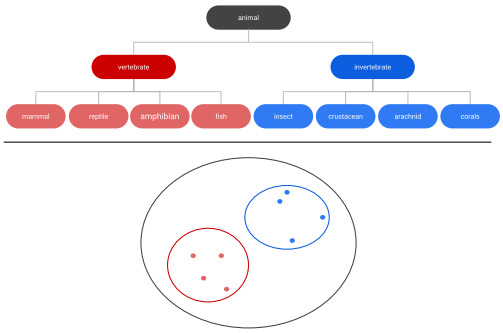
(a) Example of centroid based clustering. Centroids (filled symbols) are located in the center of the corresponding data clusters. Very efficient and simple, but sensitive to initial conditions and outliers.



(b) Example of density based clustering. Dense areas are combined to clusters. Approach allows arbitrary shapes and outliers are not assigned to clusters.



(c) Example of distribution based clustering based on gaussian distribution. This method assumes, that clusters are composed of statistical distributions. Prior knowledge about the data distribution is required.



(d) Example of hierarchical clustering. This method creates a tree of clusters, which fits well for hierarchical data.

Figure 2.9: Illustration of different clustering methods

2.7.1 Centroid based clustering

Centroid based clustering is one of the most commonly used clustering type. As shown in Figure 2.9a the data is split into a number of different regions having a centroid as a defining point in the center of the cluster. Centroid based clustering algorithms are usually very efficient and simple but tend to be sensitive against initial conditions and outliers. Centroid based clustering algorithms work best, if data has a circular shape as they are working mostly with euclidean distances. If the clusters shapes are different to circles, another type of clustering would be recommended. The major advantage of centroid based clustering methods, are, that the computational cost is low in most cases. A popular algorithm is the *K-means* [34]. Other techniques of the same type would be *K-medoids*, *PAM*, *CLARA* and *CLARANS*. [32].

2.7.2 Density based clustering

Density based clustering uses the spatial density of a data set in order to assign cluster labels. An illustration of density based methods is shown in Figure 2.9b. As outliers have a very low density per definition, those data points are not assigned to a cluster in this cases. Popular density based algorithms are *DBSCAN*, *OPTICS* and *Mean-shift* [32]. Density based clusterings are not restricted to shapes of the clusters as well as to the number of clusters. There are also efficient, linear algorithms for density based clustering (e.g. DBSCAN).

2.7.3 Distribution based clustering

Distribution based clustering assumes, that each cluster in the data set belongs to a different statistical distribution as shown in Figure 2.9c. This requires prior know-

ledge about the data set. If this prior knowledge is not available, this clustering type is not recommended. Figure 2.9c shows an example with a gaussian distribution. In the center of each cluster the expected value is located, as the probability is the highest. By moving from the center, the density and the corresponding probability decreases. Popular algorithms are *DBCLASD* or the *Gaussian mixture model algorithm* (GMM) [32].

2.7.4 Hierarchical clustering

Hierarchical clustering algorithms are creating trees of clusters with common properties as shown in Figure 2.9d. It fits well for taxonomy typed data. Common algorithms are *Hierarchical agglomerative clustering (HAC)*, *BIRCH*, *CURE*, *ROCK* and *Chameleon* [32].

3 Related work

The main question of this master thesis is, which different patterns can be found on a set of real world wafer maps in semiconductor industry. The most common approach is, that engineers are looking at visualizations of wafer map measurements and are able to see different kind of shapes depending on the scaling and the color mapping of the wafer map. As this restricts to common shapes human eyes are able to detect and also depends highly on the data visualization, some more scientific approaches been worked on. The first attempts was to use standard pattern recognition methods in order to compare features from new wafer maps with well-known features from reference wafer maps.

Many of this approaches have a common problem. They are in need of big sets of labelled data in order to perform e.g. supervised classification. This is usually hard manual human work. Furthermore, again only patterns the human eye can easily see are handled in such hand-made data sets.

Wang and Chan [35] show in their paper on how to handle rotated patterns on wafer maps using three different approaches for three different types of patterns. Polar masks to extract features of concentric patterns and line and arc masks to handle eccentric patterns like scratches. In order to get a good reference data set, different statistical approaches are needed to handle different kinds of patterns.

Schrunner, Bluder, Zernig, Kaestner and Kern show in their paper on how to extract patterns from wafer maps using Markov Random fields [36]. It turned out, that for

specific types of wafer maps, the method works good but for e.g. steep or sharp edges not.

Santos and Kern [37] showed in their paper that using VAEs, wafer map patterns can be extracted without prior knowledge. Therefore, the latent vector of the encoders output is fed to a clustering algorithm. The assumption is, that the spatial representation of the encoder output can be used to find similar patterns within a data set. This was the main idea of this thesis to try a similar approach using GANs instead of VAEs.

A similar approach using a VAE was used by Shon, Batbaatar, Cho and Choi [38]. The major difference is, that they are fine tuning the encoder after the unsupervised, autoencoder training step. This leads to more stability on the labelled data set.

4 Materials and method

This thesis handles the problem on how to extract patterns from semiconductor wafer map measurements without having prior knowledge about the actual possible patterns. This requires multiple stages as shown in Figure 1.1.

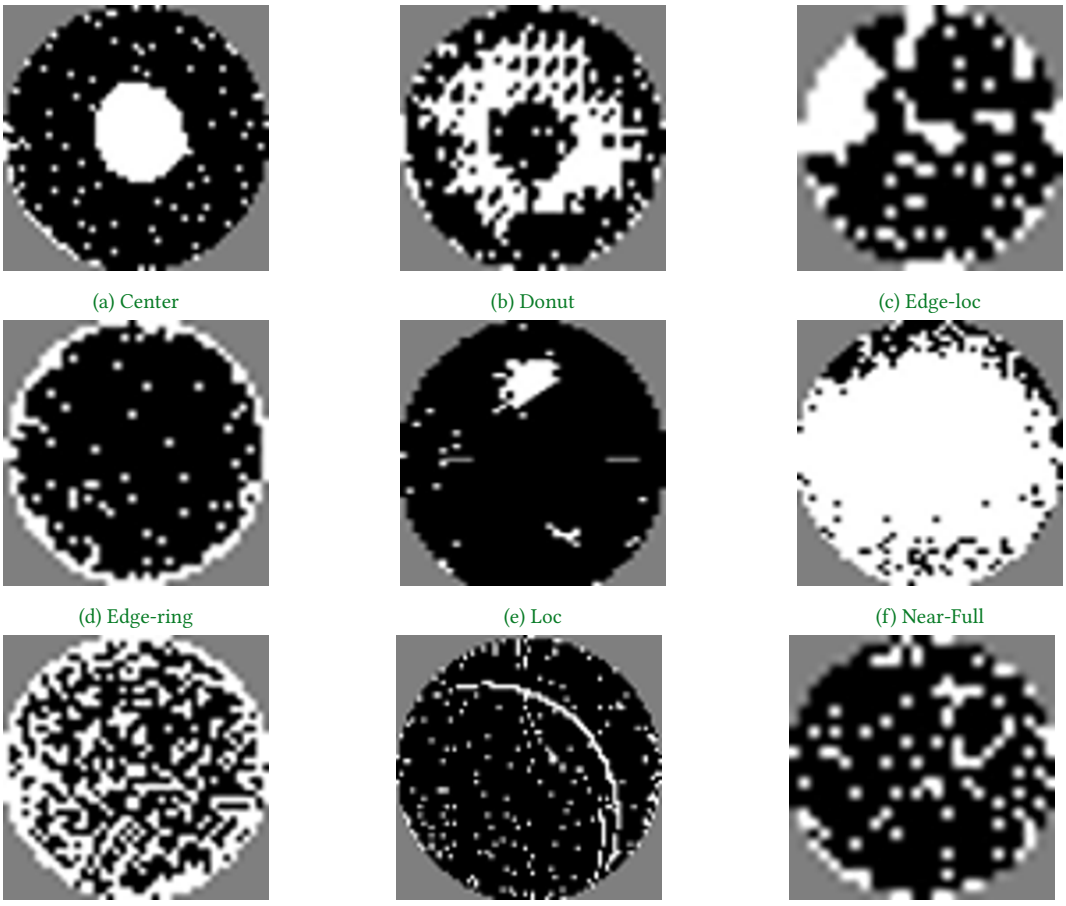
4.1 Data

The first, and actually the biggest implementational part of the project is the pre-processing of the given data sets.

4.1.1 Data sets

For the experiments, three different data sets has been used. The first data set (A) is a binary data set from real world data, consisting of 11,457 wafer maps collected from 46,393 lots in real-world fabrication [39]. It is called WM-811K and differs between nine pattern labels. Figure 4.1 shows a representation for each label. The data in this data set is unbalanced, which means that some labels have a significant bigger or lower amount of samples. Therefore, some balancing method is applied beforehand. In this case, a two-step approach was applied. The first step is to calculate the median $\bar{c}$ of the counts c_l of the labels l samples in the training data set. The second step is to pick only $\max(c_l, \bar{c})$ examples for further usage. The second data set (B)

is a simulated data set derived from real world data [40]. It consists of 1000 wafer maps consisting of parametric floating point values and five different pattern labels. Figure 4.2 shows a representation for each label. In order to make use of the data, some pre-processing is required.

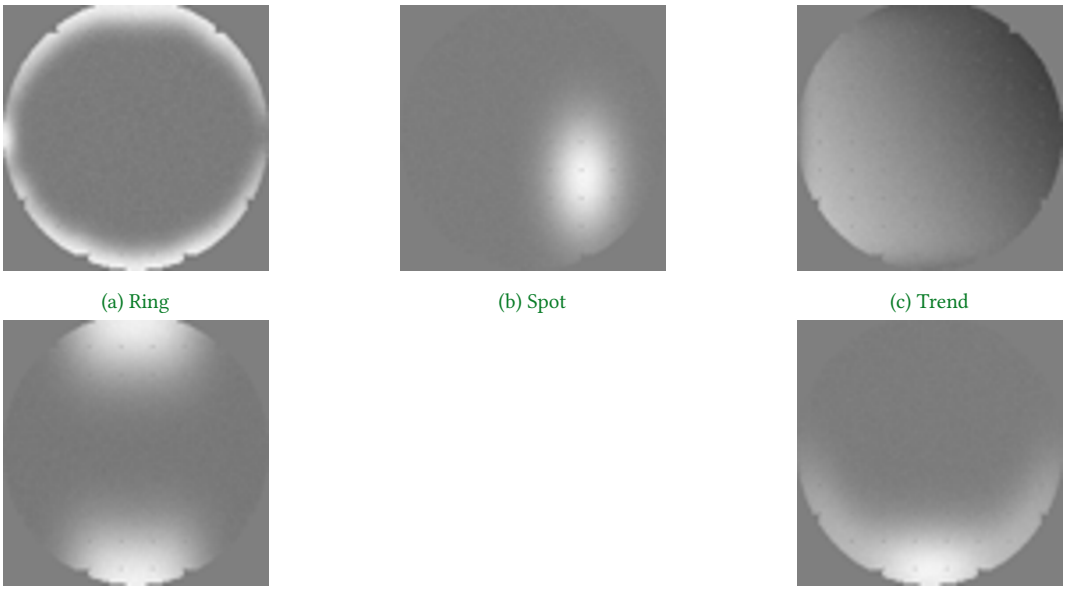


(a) Center (b) Donut (c) Edge-loc

(d) Edge-ring (e) Loc (f) Near-Full

(g) Random (h) Scratch (i) None

Figure 4.1: Data set (A). Labelled binary data set with 9 different labels.



(a) Ring (b) Spot (c) Trend

(d) Two-spots (e) Crescent

Figure 4.2: Data set (B). Labelled continous data set with 5 different labels.

4.1.2 Pre-processing

As shown in Section 2.1, wafer maps are circular arrays of numerical, binary or labeled values. In order to make them use within a NN, some pre-processing is required depending on the type of data. Figure 2.1 shows the whole chain of pre-processing.

Pre-processing of binary or labelled data

As binary or labelled data does not have a useful numerical value, a mapping from $\mathbb{Z} \rightarrow \mathbb{R}$ is required beforehand. As the data set (A) consists of binary data

(PASS/FAIL) only the following mapping was applied.

$$\mathbb{Z} \to \mathbb{R} : \begin{cases} PASS \to 0 \\ FAIL \to 1 \\ NOT - TESTED \to 0.5 \end{cases} \tag{4.1}$$

For mapping e.g. bin values, some priority mapping can be applied. Therefore, pass bins can be e.g. mapped negatively, and fail bins positively. The order of the bins is dependent on the significance of the failure. As this was not used within the experiments, no detailed description is provided in this thesis.

Missing data, scaling and normalization

Missing data handling, as well as scaling and normalization steps are dependent on the type of data used. If binary or labelled data is used, none of them is needed in case the mapping is already done in an appropriate way. For all parametric measurement values, all theory of Sections 2.1.1 and 2.1.2 is required. For missing data handling, different approaches has been tried. It turned out, that replacing missing values with a global median, worked best in this cases.
As shown in Section 2.1.2, histogram equalization improves visualization of parametric values also for NNs. Therefore, the IQR scaling operation as shown in Equation 2.2 has been applied. In order to fit into gray scale image space, the usual normalization operation $n : \mathcal{W}_{p_i}^n = \frac{\mathcal{W}_{p_i}^{IQR} - \min(\mathcal{W}_{p_i}^{IQR})}{\max(\mathcal{W}_{p_i}^{IQR}) - \min(\mathcal{W}_{p_i}^{IQR})}$ is applied to the data.

Additional pre-processing steps

As data from both data sets include noise, at the end of scaling and normalization, a median filter is applied. Also the size of the wafer map images is scaled to the

input size of the BigBiGANs generator using bilinear interpolation.

4.2 Network

4.2.1 BigBiGAN

For this thesis, the BigBiGAN network is used for all experiments. As in the definition of BigBiGAN there are some degrees of freedom on how to implement the different parts of the network. As a basis, a piece of code [1] has been used from GitHub. It is also mentioned at *Papers with Code* [2]. It is a framework coded in python using TensorFlow.

Encoder architecture

As shown in Figure 4.3, the encoder network consists of multiple layers. As an input, the encoder gets a set of size n images with the shape of $(32 \times 32 \times 1)$. The first layer upsamples the input images to a shape of $(64 \times 64 \times 1)$. The next step is a convolutional layer with a 32 kernels of size (7×7) with a stride of 2. The convolutional layer is followed by a max-pooling layer of size (3×3) with a stride of two. This results in a $(n \times 15 \times 15 \times 32)$ tensor which is passed to the following two *bottleneck residual* blocks, known from the RevNet [41] architecture which is special type of a residual neural network [42]. The two blocks are continued with an average-pooling layer of size (2×2). The combination of two bottleneck residual blocks and an average pooling layer is repeated once again, followed by additional two bottleneck residual blocks. At the end of the RevNet parts, a tensor of size

[1] https://github.com/LEGO999/BigBiGAN-TensorFlow2.0
[2] https://paperswithcode.com/method/bigbigan

$(n \times 4 \times 4 \times 128)$ results, followed by a reduce-mean block. This block averages over the (4×4) dimensions which results in a $(n \times 128)$ tensor. The next two layers are an aggregation of sublayers as shown in Figure 4.4. They are consisting of two dense layers followed by a dropout [43] and a LeakyReLU output activation added to a set of a single dense layers followed by a LeakyReLU output activation. The last layer of the network is a dense MLP with an output size with the length of the latent vector $\hat{\mathbf{z}}$.

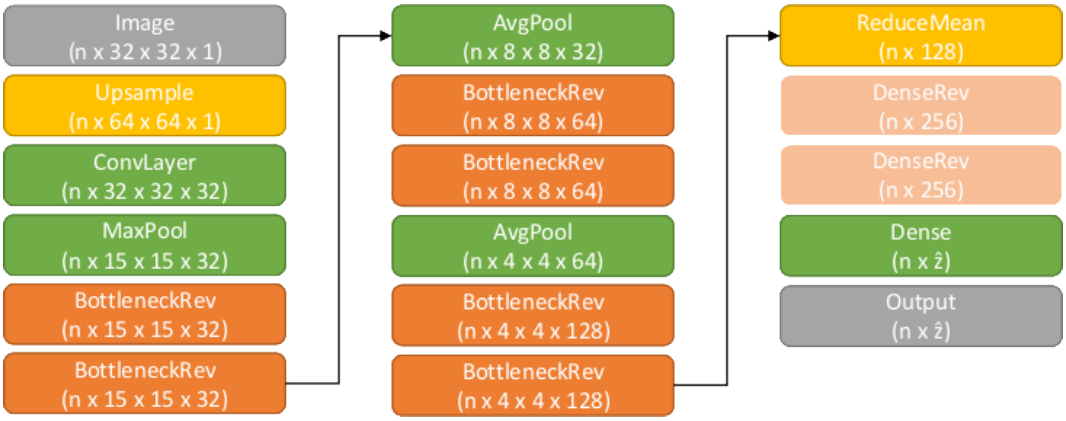


Figure 4.3: Graphical scheme of the encoder network. Upsampling, followed by ConvLayer and max-pooling. Three repetitions of BottleneckRev layers as defined in the RevNet [41] architecture and a combination of two DenseRev layers as shown in Figure 4.4 and a MLP result in the latent vector $\hat{\mathbf{z}}$

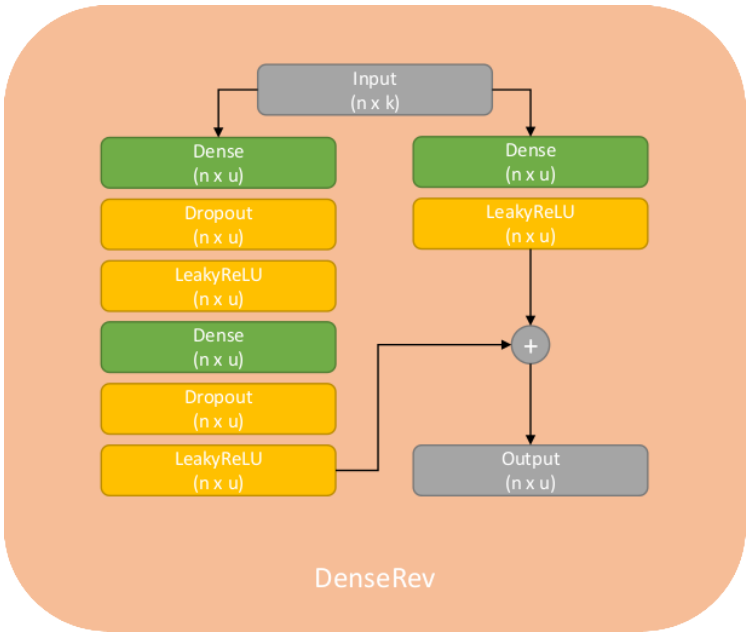


Figure 4.4: Graphical scheme of the DenseRev block. Two combined paths of layers are added for the final result. The first path consists of two dense blocks with a dropout layer and a LeakyReLU activation. The second path consists only of one of this combinations.

Generator architecture

The input for the generator is a set of n vectors of the size of the vector $\mathbf{z}$. For simplicity, this is noted as $(n \times z)$. Each vector is splitted into four parts of equal size. In case that the vector size of $\mathbf{z}$ is two, the input vector is repeated once in order to fit the size of four. The first split, Split 0 is passed to the first layer, which consists of a dense MLP layer wrapped by a Spectral Normalization layer [44]. As shown in Figure 4.5, the layer results in a tensor of size $(n \times 4096)$ which gets reshaped to size $(n \times 4 \times 4 \times 256)$. The reshaped tensor is then passed to two ResBlock layers, which are defined in the ResNet [42] architecture. The first ResBlock layer uses the input of the previous layer and the tensor of Split 1. Split 2 is then used with the second ResBlock layer followed by an Attention layer [45]. The resulting tensor of

size $(n \times 16 \times 16 \times 128)$ is passed to another ResBlock layer, which is using the last split, Split 3. After that, a BatchNormalization [46] with a ReLU output activation follows. The output layer is a transposed 2D convolutional layer, wrapped by a Spectral Normalization layer. The output of the generator is a image tensor of size $(n \times 32 \times 32 \times 1)$.

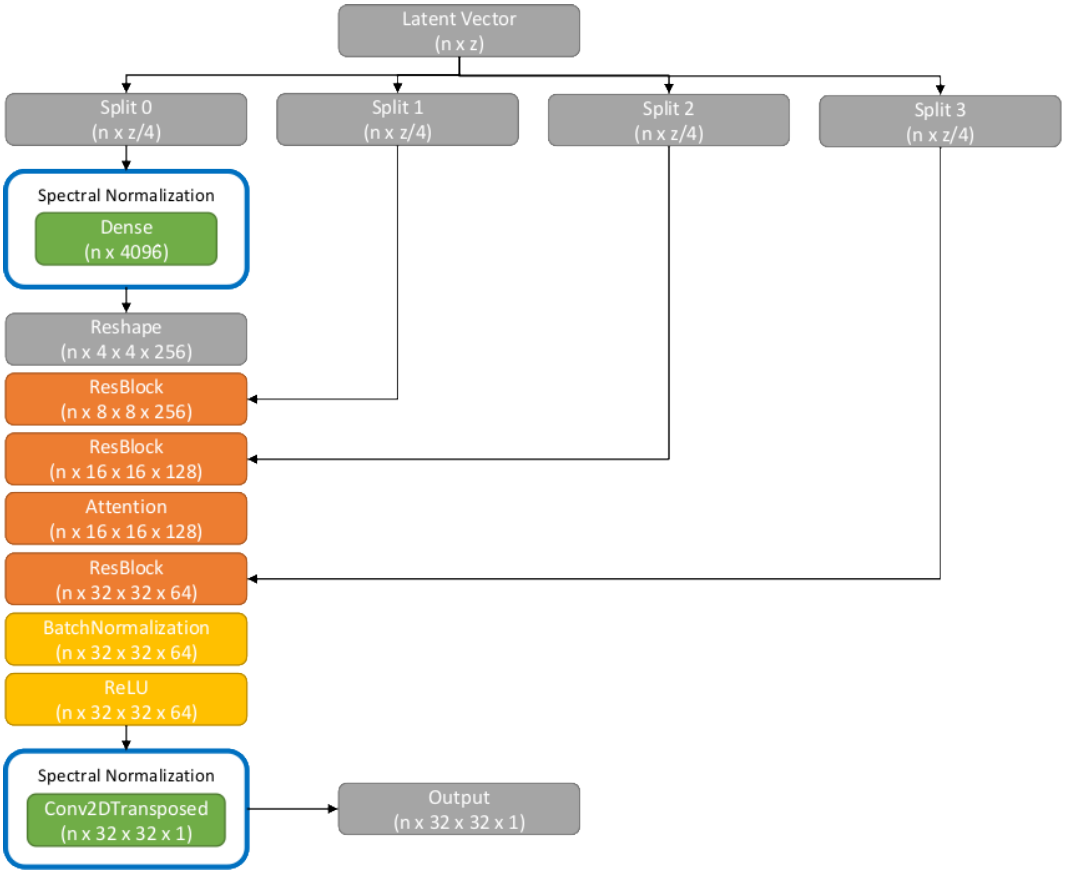


Figure 4.5: Graphical scheme of the generator network. The input vector $\mathbf{z}$ is splitted into four equally sized parts. The first part inputs into a MLP layer wrapped by a Spectral Normalization layer [44]. The reshaped output together with the other splits of the input is sequentially applied in ResBlocks interrupted by an Attention block. Batch Normalization with a ReLU activation follow. The last block consists of a Spectral Normalization wrapped transposed 2D convolutional layer which results a generated image $\hat{\mathbf{x}}$.

Discriminator architecture

As previously described, the discriminator of a BigBiGAN network consists of three parts F, H and J. As shown in Figure 4.6, the F network uses the output of the generator as an input and consists of a ResDown layer, which is described in the ResNet architecture [42], followed by an Attention layer [45]. The resulting tensor of size $(n \times 16 \times 16 \times 64)$ is passed in two subsequent ResDown layers, resulting in a tensor of size $(n \times 4 \times 4 \times 256)$. A reduce-mean block, averages over the (4×4) dimensions in order to get n vectors of size 256, which can be passed to the J network. An additional dense layer is provided, resulting in scalar score values $\mathbf{s}_x$ per sample used for the loss function.
The H network uses the output of the encoder as an input, and consists of two DenseRev blocks, which are aggregation of sublayers as shown in Figure 4.4. The output of the last DenseRev block has a size of $(n \times 64)$ and is also passed to the J network. An additional dense layer is provided, resulting in scalar score values $\mathbf{s}_z$ per sample used for the loss function.
The J network uses the concatenated outputs of the F and the H network as an input. Three subsequent DenseRev blocks and a final MLP dense block result in scalar scores value per sample $\mathbf{s}_{xz}$

4.2.2 Training

The BigBiGAN paper provides loss functions for all different networks as shown in Equations 2.10 and 2.8. Therefore, the loss functions are only dependent on the score values $\mathbf{s}_x$, $\mathbf{s}_z$ and $\mathbf{s}_x z$ and the fake or original information y. In order to perform training, some parameters need to be defined. For all following experiments the Adam [47] optimizer has been used. It requires some regularization parameters β^1 and β^2 listed in Table 4.1. The values has been chosen by an trial and error approach. As each types of GANs are very sensitive to training and optimization

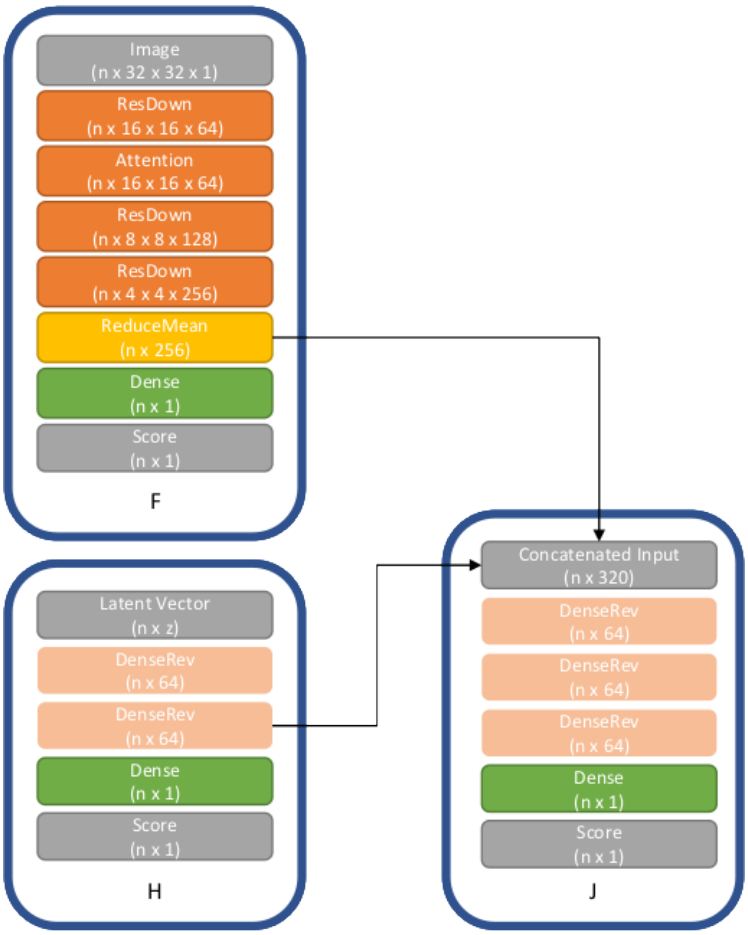


Figure 4.6: The F network is handling the output of the generator. It consists of three ResDown blocks interrupted once by an Attention block. By a reduce-mean operation, the resulting (4×4) part of the output tensor is reduced to a vector of size 256, this vector is furthermore used in the J network. By a final MLP, a single scalar value $\mathbf{s}_x$ per sample results. The H block handles the output of the encoder. It consists of two DenseRev blocks, where its output vectors are passed to the J network. Again, a MLP reduces the vector output to a single scalar score value $\mathbf{s}_z$ per sample. The J network combines both vector outputs of the F and H network, by three DenseRev blocks followed by a standard MLP to result in a scalar score value $\mathbf{s}_{xz}$ per sample. All three score values are used for computing the loss functions.

parametrizations, many iterations were necessary. The BigBiGAN paper proposes to use different optimizer parametrizations for encoder, generator and discriminator. At this experiments, the very same parametrizations performed best. The ratio of upgrading weights $R_{\mathcal{EG}}$ defines, how often the discriminator weights are updated until the encoder and generator weights are updated during training. In this case, the discriminator is updated twice until the generator and encoder parts are updated. This leads to a more stable training.

Parameter	Variable	Value
Learning rate for encoder and generator	$l_{\mathcal{EG}}$	$2 \cdot 10^{-4}$
Learning rate for discriminator	$l_{\mathcal{D}}$	$2 \cdot 10^{-4}$
Regularizer parameters for encoder and generator	$\beta^1_{\mathcal{EG}}$, $\beta^2_{\mathcal{EG}}$	0.5, 0.99
Regularizer parameters for discriminator	$\beta^1_{\mathcal{EG}}$, $\beta^2_{\mathcal{EG}}$	0.5, 0.99
Ratio of upgrading weights	$R_{\mathcal{EG}}$	2
Training batch size	B	256
Number of epochs	E	1000

Table 4.1: Parameters required for training and optimization of the encoder, generator and discriminator networks within the BigBiGAN architecture.

4.2.3 Noise modes

The distribution of the fake input noise of the generator P_z can be chosen arbitrarily. For the experiments two distributions has been defined. The first distribution is the uniform distribution $P_z = \mathcal{U}^N(0,1), z \in \mathbb{R}$, where N is the dimension of the vector. The main idea of using the uniform distribution instead of a normal distribution $\mathcal{N}$ as proposed in the BigBiGAN paper is, that $\mathcal{U}(0,1)$ fills more space within the latent space, which makes it easier for the clustering algorithms to cluster.
The second distribution is a random one-hot distribution based on the uniform distribution. A sample $\mathbf{z}$ is defined as an all zero vector having one nonzero entry

at random position i. In this case i is the random vector, defined by the uniform distribution $P_i = \mathcal{U}(0, N), i \in \mathbb{Z}$. The aim of the second distribution is, to provide a latent space, which intrinsically defines classification clusters.

4.3 Clustering

For clustering, three different methods are used for comparison. The first method is the centroid based algorithm K-means. It has a linear runtime and is therefore a fast approach. The main drawback is, that the number of clusters need do be defined beforehand. It is based on the euclidean distance from a centroid, which only allows compact shapes of the clusters. The second method is the density based algorithm DBSCAN. It is also a linear approach. If correctly parametrized, arbitrary shapes of the clusters are possible. The main drawback of the method is, that parametrization is hard and strongly dependent on the data. The last method is a hierarchical clustering algorithm called Hierarchical agglomerative clustering. It orders data points hierarchically by similarity. The actual clusters can be extracted by cutting the graph on a certain position. It is a very powerful, but non-linear approach. Also parametrization is hard.

5 Evaluation

5.1 Experiments

For the experiments, the BigBiGAN network is always trained with the given training and optimization parameters shown in Table 4.1. The experiments differ in data set, clustering method and noise mode. In order to compare the results, a cluster-performance metric $Cp(\mathbf{z}, C, L)$ is introduced as shown in Equation 5.1. The vector $\mathbf{z}$ is the latent vector. $C(\mathbf{z})$ returns the clusters for the input vector. $L(\mathbf{c})$ returns the given labels from the original data set for each data point.

$$Cp(z, C, L) = \frac{1}{|C|} \sum_{\mathbf{c} \in C(\mathbf{z})} \frac{maxcount(L(\mathbf{c}))}{|\mathbf{c}|} \tag{5.1}$$

The metric uses the count of the most available original label $maxcount$ within each cluster $\mathbf{c}$ divided by its length $|\mathbf{c}|$. The $maxcount$ function returns the count of the original label that appears most in the cluster. An ideal cluster consists only of one type of original label. In this case the metric results with 1. The result is the arithmetic mean of the individual cluster performances. This gives the opportunity to compare different cluster algorithms with each other.
There are three types of variations within the experiments. There are two different data sets, three different clustering methods and two different noise modes.

5.2 Results

5.2.1 Data set (A)

The evaluations of the cluster performance is done by using a test data set of 1024 wafers. No wafer of this set has been used during training before.

2D $\mathbb{U}(0,1)$ noise mode

Figure 5.1 shows the cluster performance over 1000 epochs for three different clustering algorithms using 2D uniformly distributed input vectors. Both HAC and DBSCAN perform better than K-means. Compared to data set B, the performance results is about 30% worse for all clustering types. It also shows, that clustering performs similar throughout all epochs.

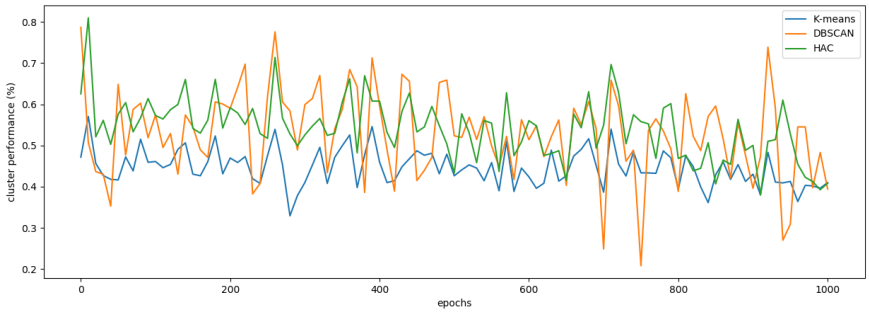


Figure 5.1: Cluster performance on three different clustering modes applied on the same latent outputs over 1000 epochs. Noise mode is the 2D uniform distribution.

The reconstruction loss in Figure 5.2 shows a steadily decreasing loss function for all cases. Only for the one-hot noise mode, it increases after around 150 epochs and results in a constant curve.

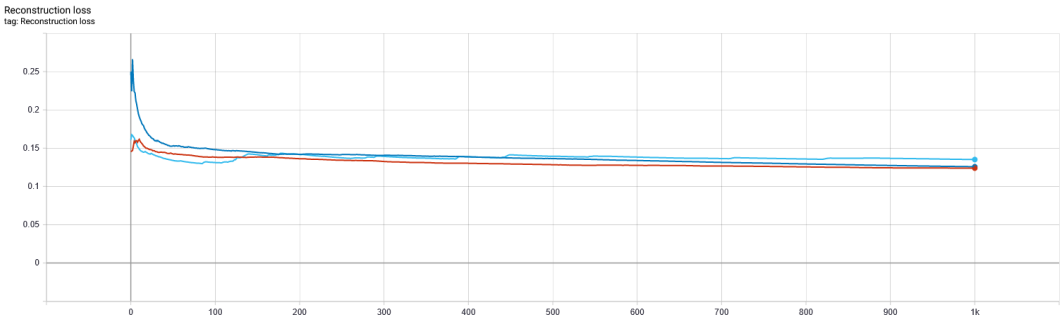


Figure 5.2: Reconstruction loss. The red curve uses the 2D uniform noise mode, the blue curve uses the 8D uniform noise mode and the light blue curve uses the 12D one-hot noise mode.

Figure 5.3 shows the cluster results per row for the K-means clustering algorithm. In this example, the output of epoch 0 is chosen, which shows already a good clustering at the first view.

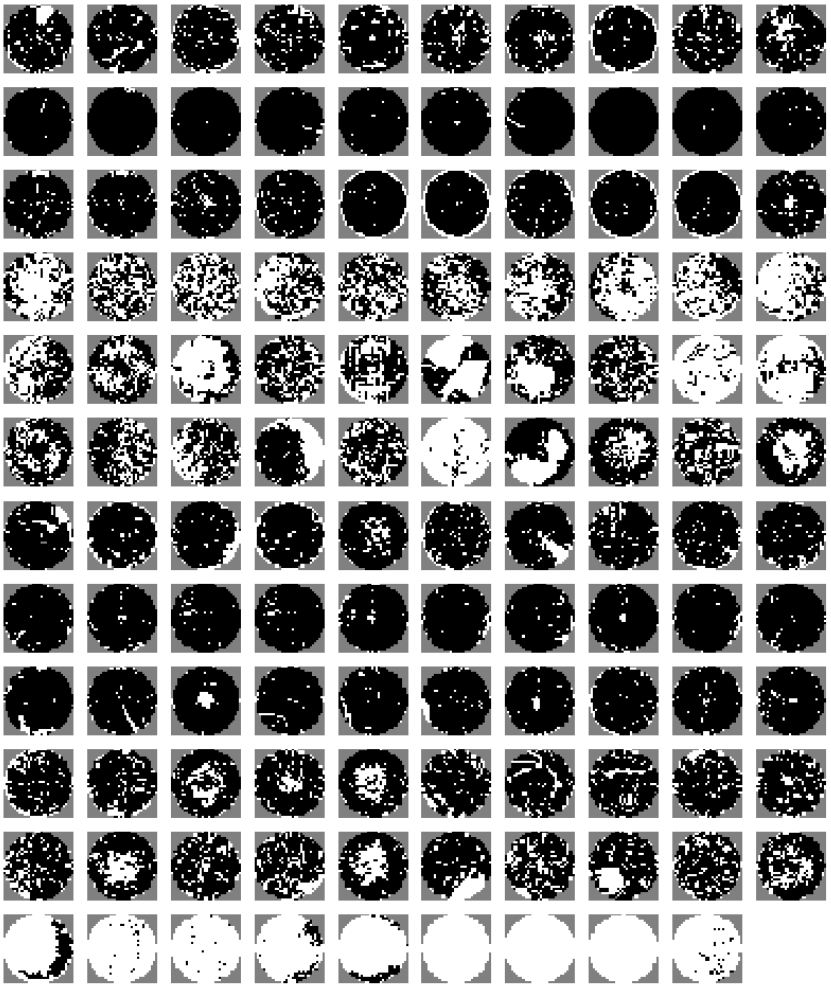


Figure 5.3: Wafer clusters using K-means clustering algorithm at epoch 0. Each row represents a sample of max 10 wafer map images of a cluster.

Figures 5.4 and Figure 5.5 show the cluster results per row for the DBSCAN and the HAC clustering algorithms. From human perspective, the clustering works good on a high level view.

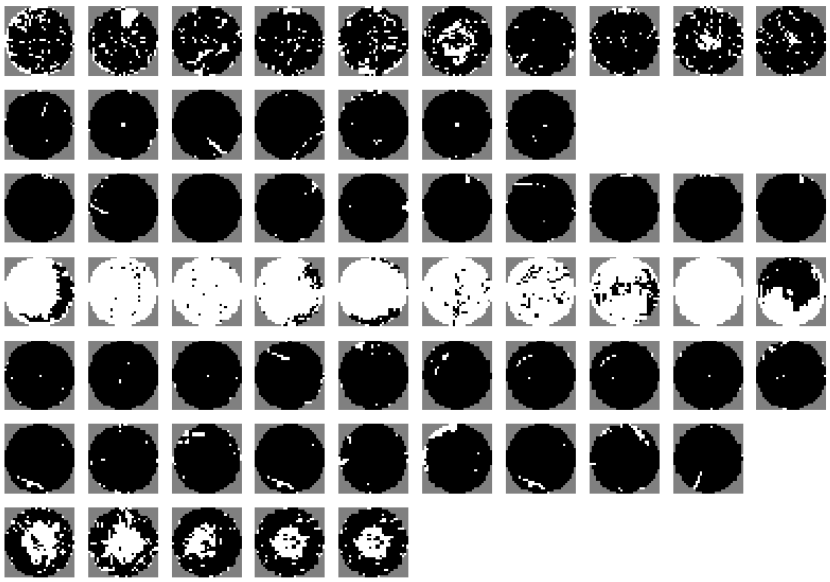


Figure 5.4: Wafer clusters using DBSCAN clustering algorithm at epoch 100. Each row represents a sample of max 10 wafer map images of a cluster.

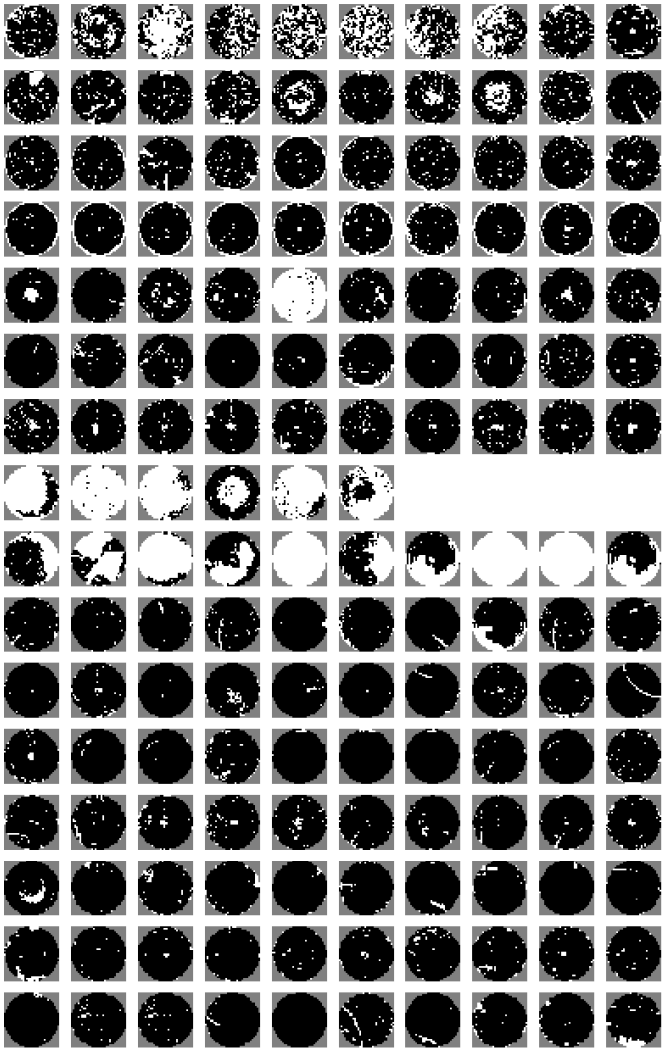


Figure 5.5: Wafer clusters using HAC clustering algorithm at epoch 590. Each row represents a sample of max 10 wafer map images of a cluster.

For all three clustering algorithms, the result is good at a high level view. Wafer map images with a higher density of black or white pixels at different regions are clustered, but fine shapes as scratches or surrounding ring shapes are hard to distinguish. Figure 5.6 shows the output latent space of the 2D encoder results with the corresponding clustering labels indicated by color.

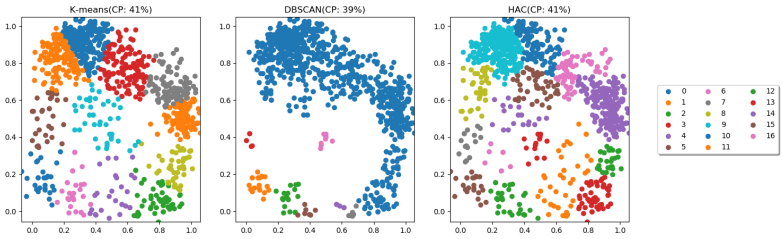


Figure 5.6: Latent space at epoch 1000 using the 2D uniform noise mode.

8D $\mathbb{U}(0,1)$ noise mode

Figure 5.7 shows the cluster performance over 1000 epochs for three different clustering algorithms using 8D uniformly distributed input vectors. HAC performs better than DBSCAN and K-means. DBSCAN performs bad after around 300 epochs, which is an example for the hardness of DBSCAN parametrization.

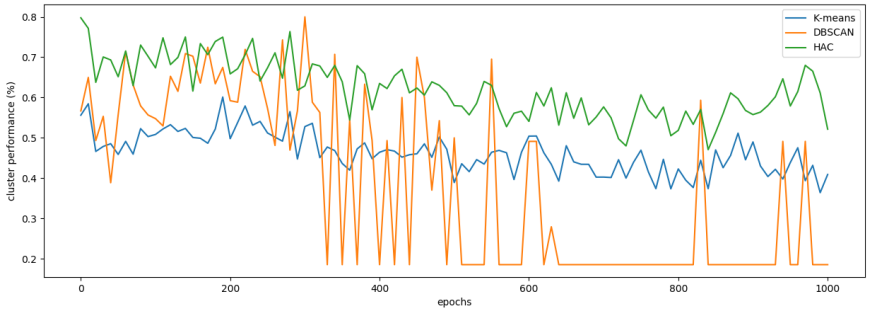


Figure 5.7: Cluster performance on three different clustering modes applied on the same latent outputs over 1000 epochs. Noise mode is the 8D uniform distribution.

Figures 5.8, 5.9 and 5.10 show up to 10 cluster representations per row. From a human perspective, high level clustering works good for all algorithms.

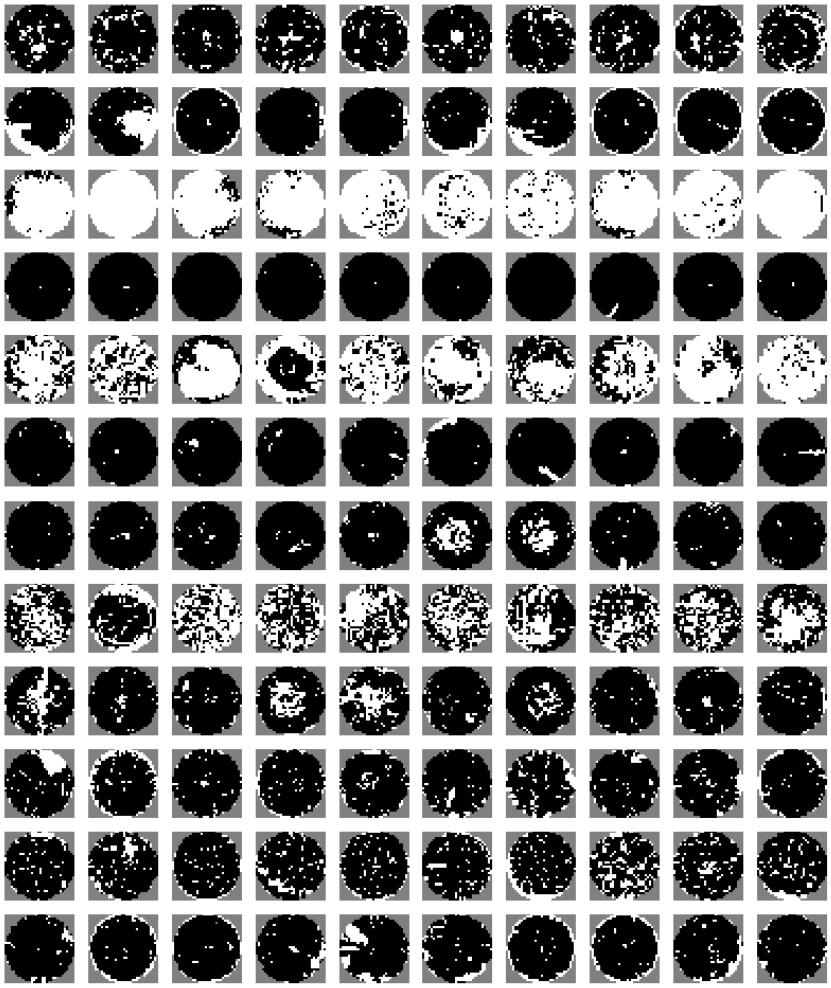


Figure 5.8: Wafer clusters using K-means clustering algorithm at epoch 170. Each row represents a sample of max 10 wafer map images of a cluster.

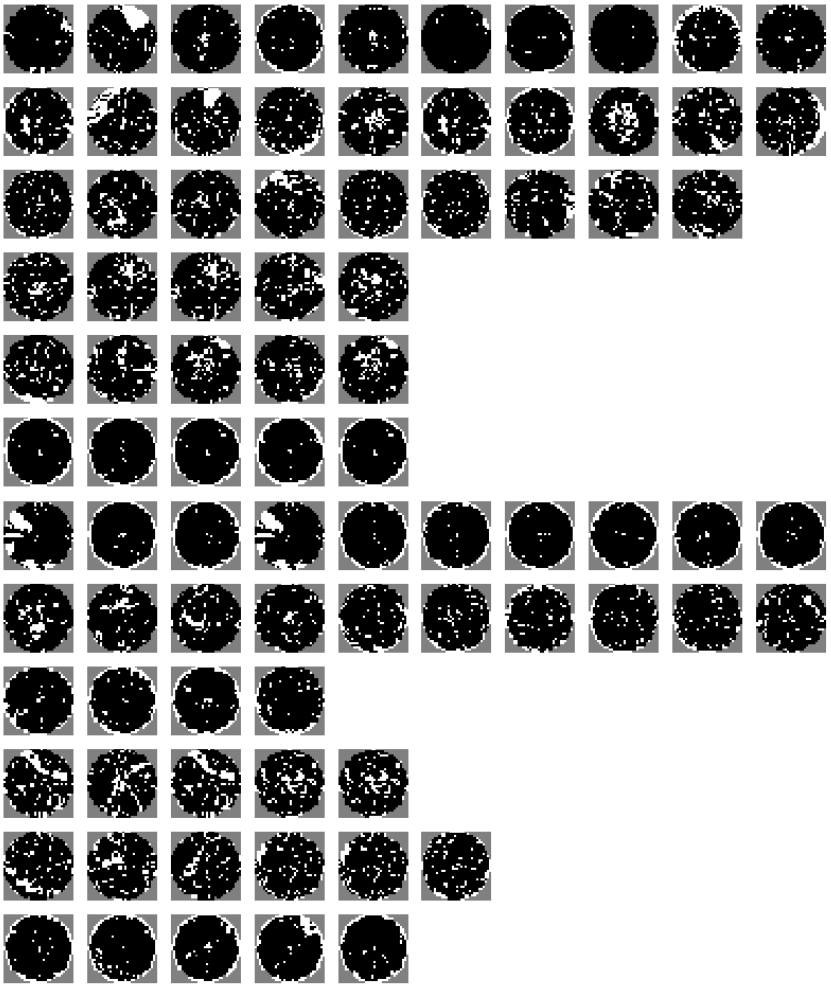


Figure 5.9: Wafer clusters using DBSCAN clustering algorithm at epoch 10. Each row represents a sample of max 10 wafer map images of a cluster.

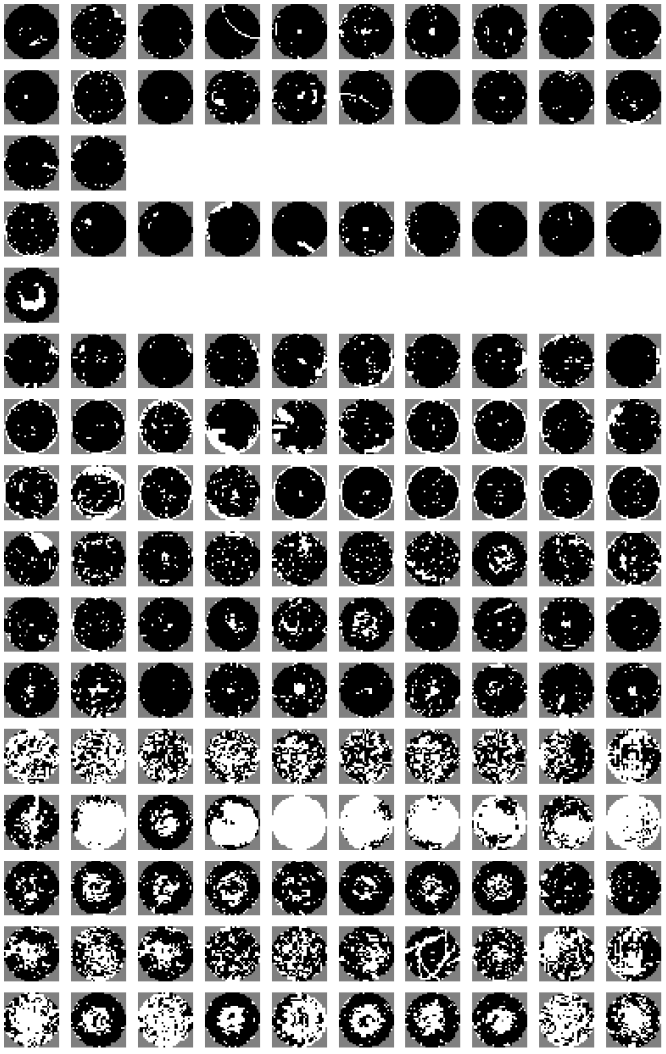


Figure 5.10: Wafer clusters using HAC clustering algorithm at epoch 740. Each row represents a sample of max 10 wafer map images of a cluster.

One-hot noise mode

Figure 5.11 shows the cluster performance over 1000 epochs for three different clustering algorithms using 12D one-hot distributed input vectors. HAC performs better than DBSCAN and K-means. The clustering result for HAC is comparable to the result of data set B.

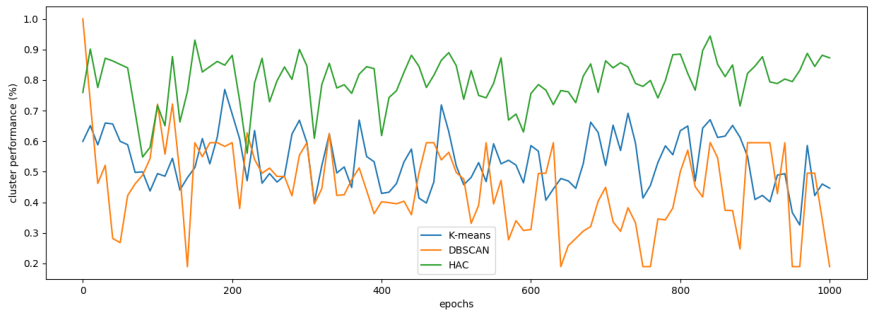


Figure 5.11: Cluster performance on three different clustering modes applied on the same latent outputs over 1000 epochs. Noise mode is 12D one-hot.

Figure 5.12 shows the generator output for all possible one-hot vectors. For this data set, the representations cannot be sufficiently used.

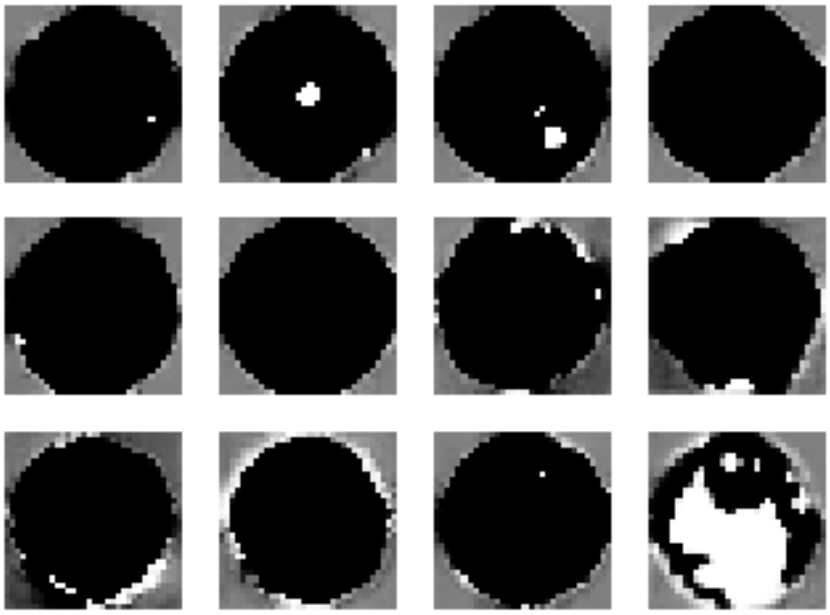


Figure 5.12: Intrinsic class representations of all possible one-hot vectors at epoch 30.

Figures 5.13, 5.14 and 5.15 show up to 10 cluster representations per row. From a human perspective, high level clustering works good for all algorithms.

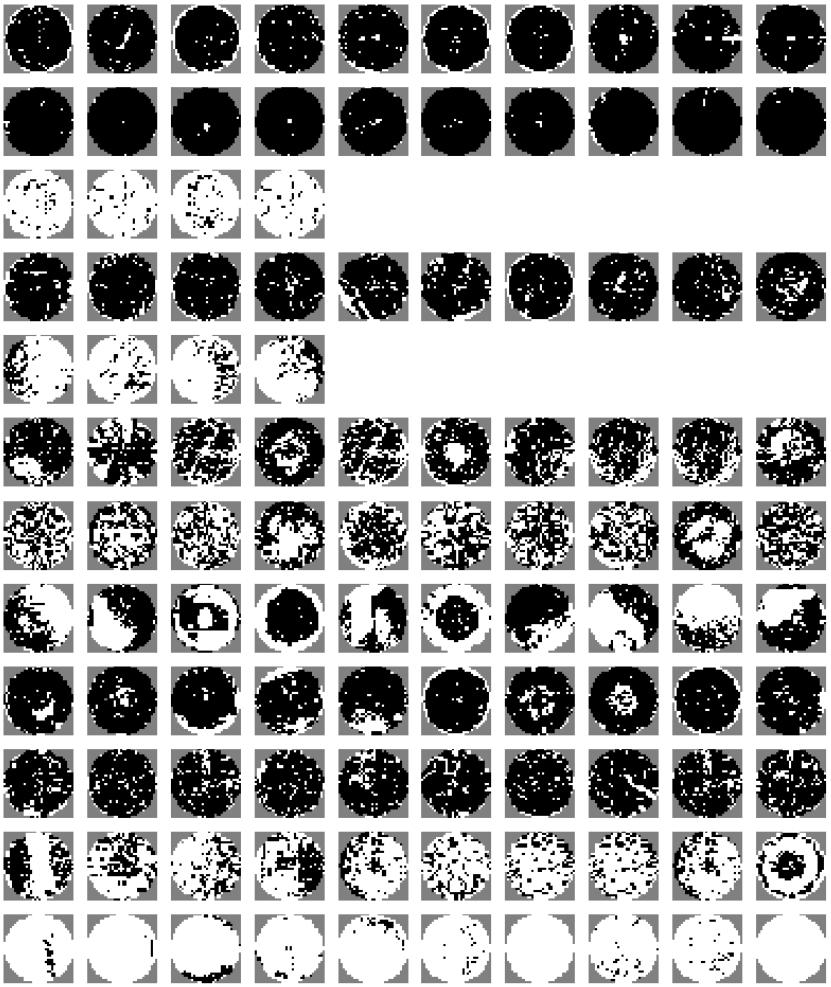


Figure 5.13: Wafer clusters using K-means clustering algorithm using 12D one-hot mode at epoch 0. Each row represents a sample of up to 10 wafer map images of a cluster.

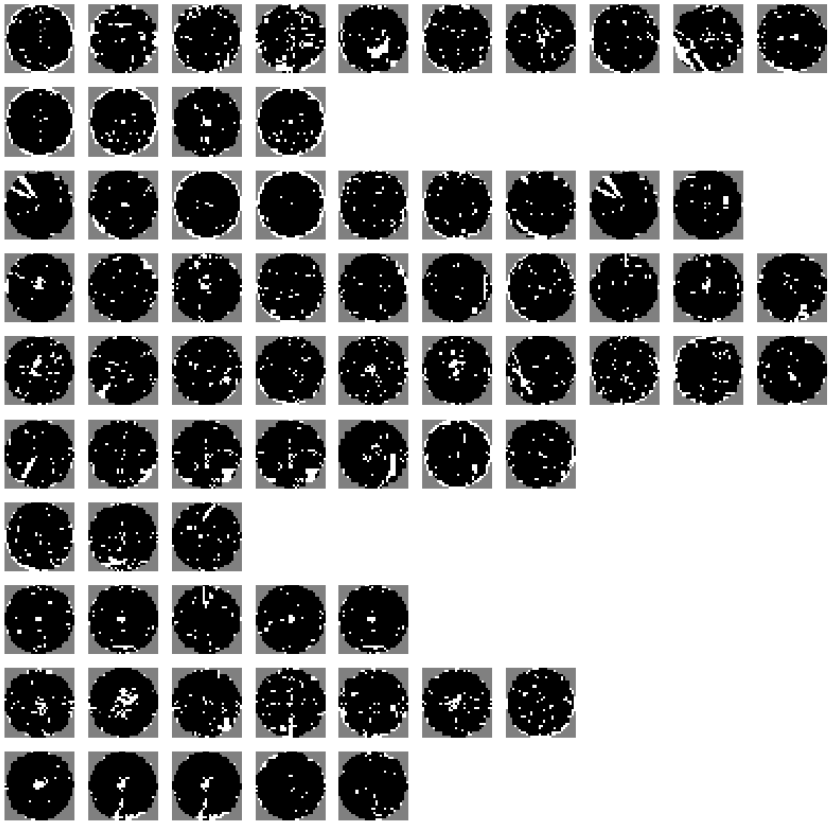


Figure 5.14: Wafer clusters using DBSCAN clustering algorithm using 12D one-hot mode at epoch 80. Each row represents a sample of up to 10 wafer map images of a cluster.

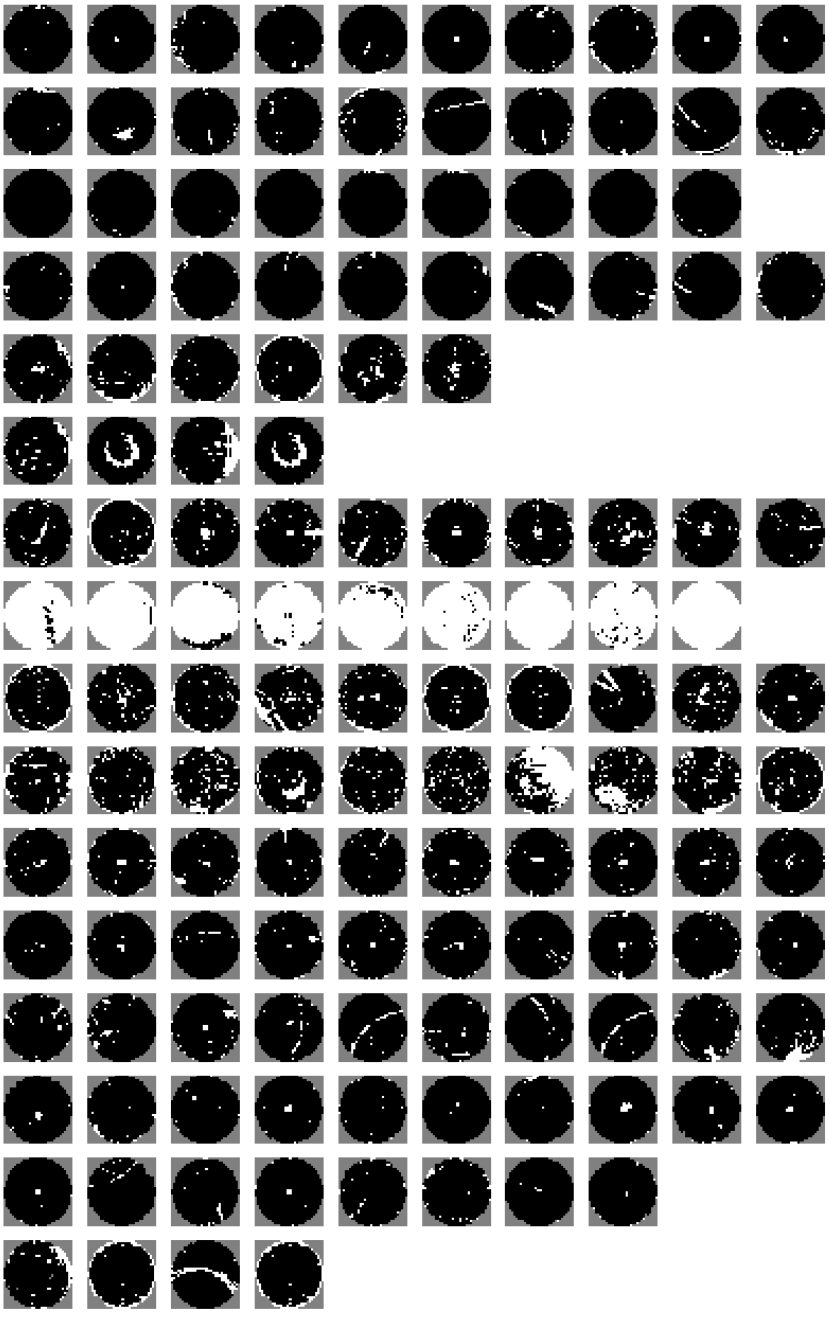


Figure 5.15: Wafer clusters using HAC clustering algorithm using 12D one-hot mode at epoch 80. Each row represents a sample of up to 10 wafer map images of a cluster.

5.2.2 Data set (B)

The evaluations of the cluster performance is done by using a test data set of 1024 wafers. No wafer of this set has been used during training before.

2D $\mathbb{U}(0, 1)$ noise mode

Figure 5.16 shows, that if the clustering methods are performed on the same latent space at uniform noise mode, HAC performs best, followed by DBSCAN and K-means.

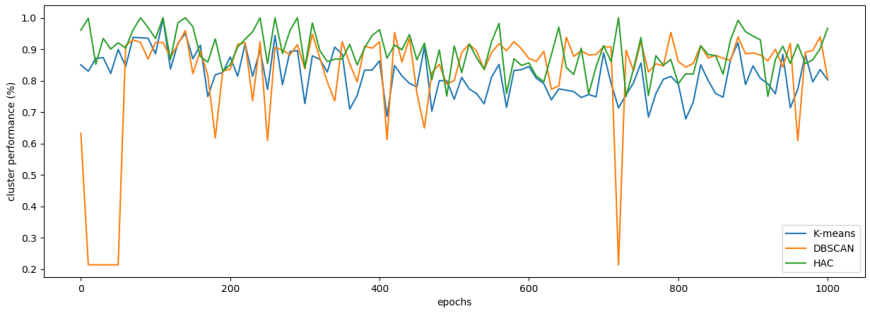


Figure 5.16: Cluster performance on three different clustering modes applied on the same latent outputs over 1000 epochs. Noise mode is the 2D uniform distribution.

The reconstruction loss is defined as the MSE between original image $\mathbf{x}$ and the generated image from the encoded latent vector $\mathcal{G}(\mathcal{E}(\mathbf{x}))$. Figure 5.17 shows that the reconstruction loss decreases steadily.

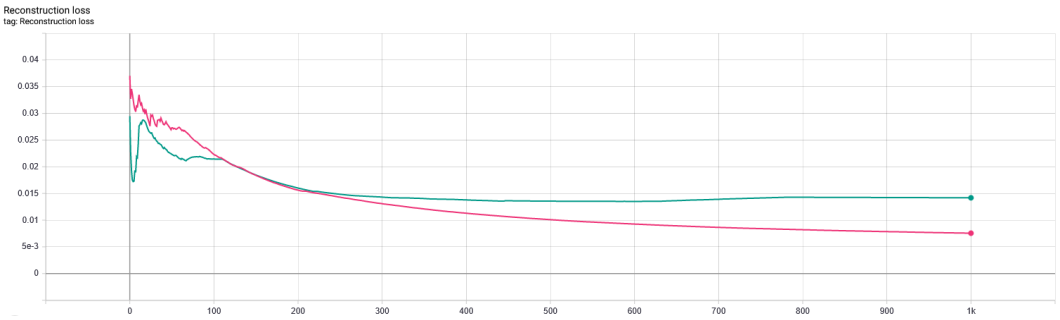


Figure 5.17: Reconstruction loss. The pink curve uses the 2D uniform noise mode, the green curve uses the 12D one-hot noise mode. The uniform noise mode steadily decreases, the one-hot mode starts to slightly increase after about 600 epochs.

The detailed clusterings in Figures 5.18, 5.19 and 5.20 show cluster representations per row. Most rows are of unique patterns with some exceptions.

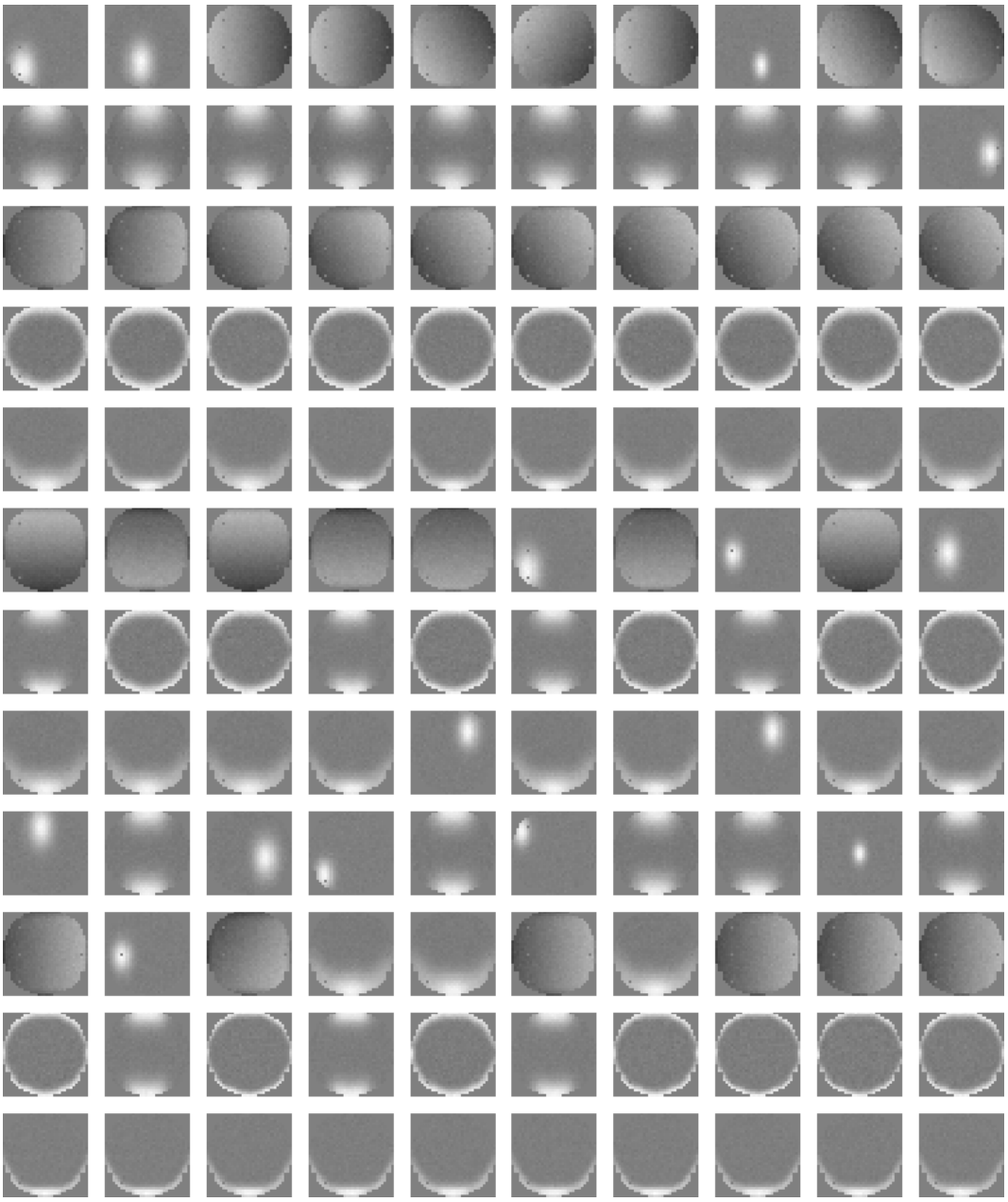


Figure 5.18: Wafer clusters using K-means clustering algorithm using 2D uniform noise mode. Each row represents a sample of max 10 wafer map images of a cluster.

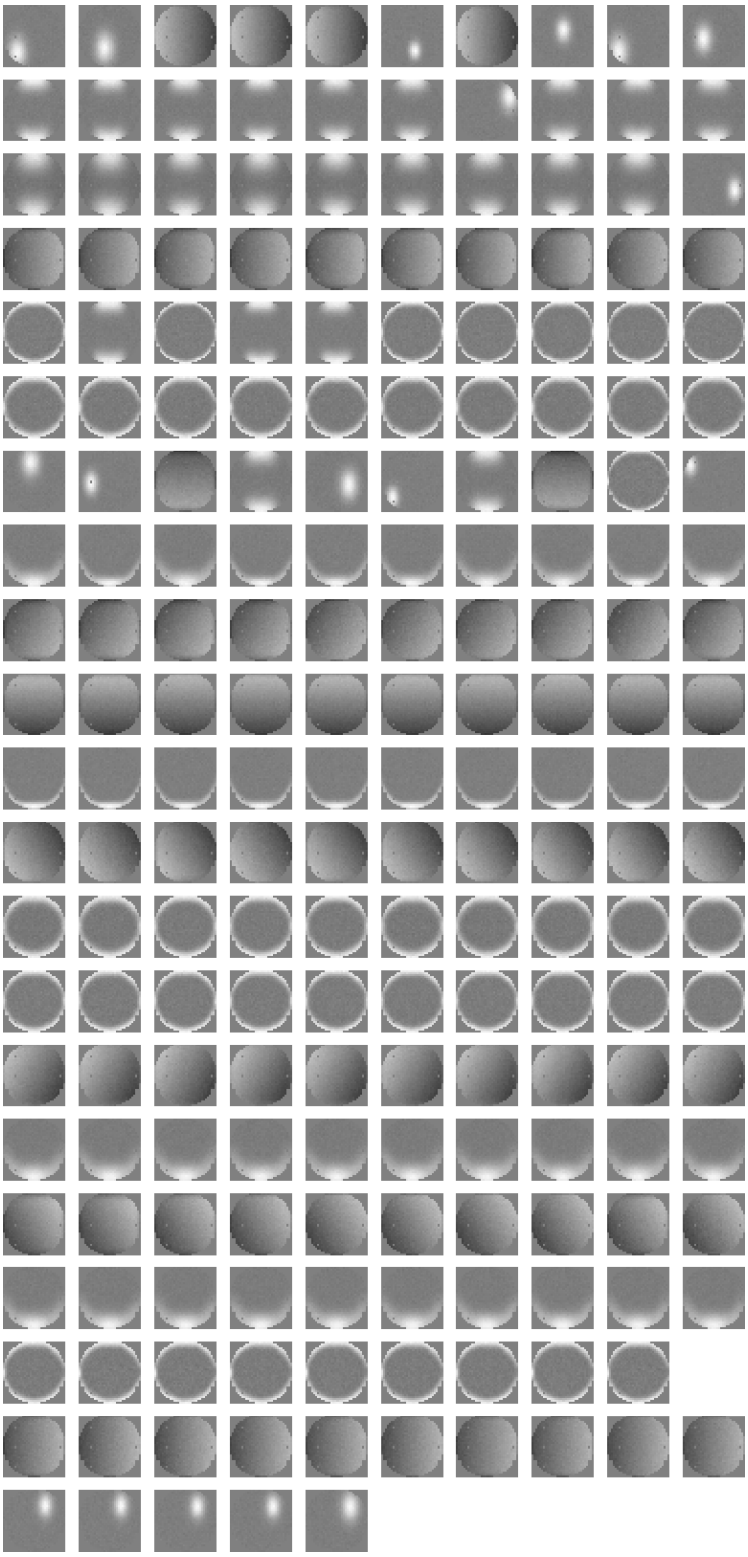


Figure 5.19: Wafer clusters using DBSCAN clustering algorithm using 2D uniform noise mode. Each row represents a sample of max 10 wafer map images of a cluster.

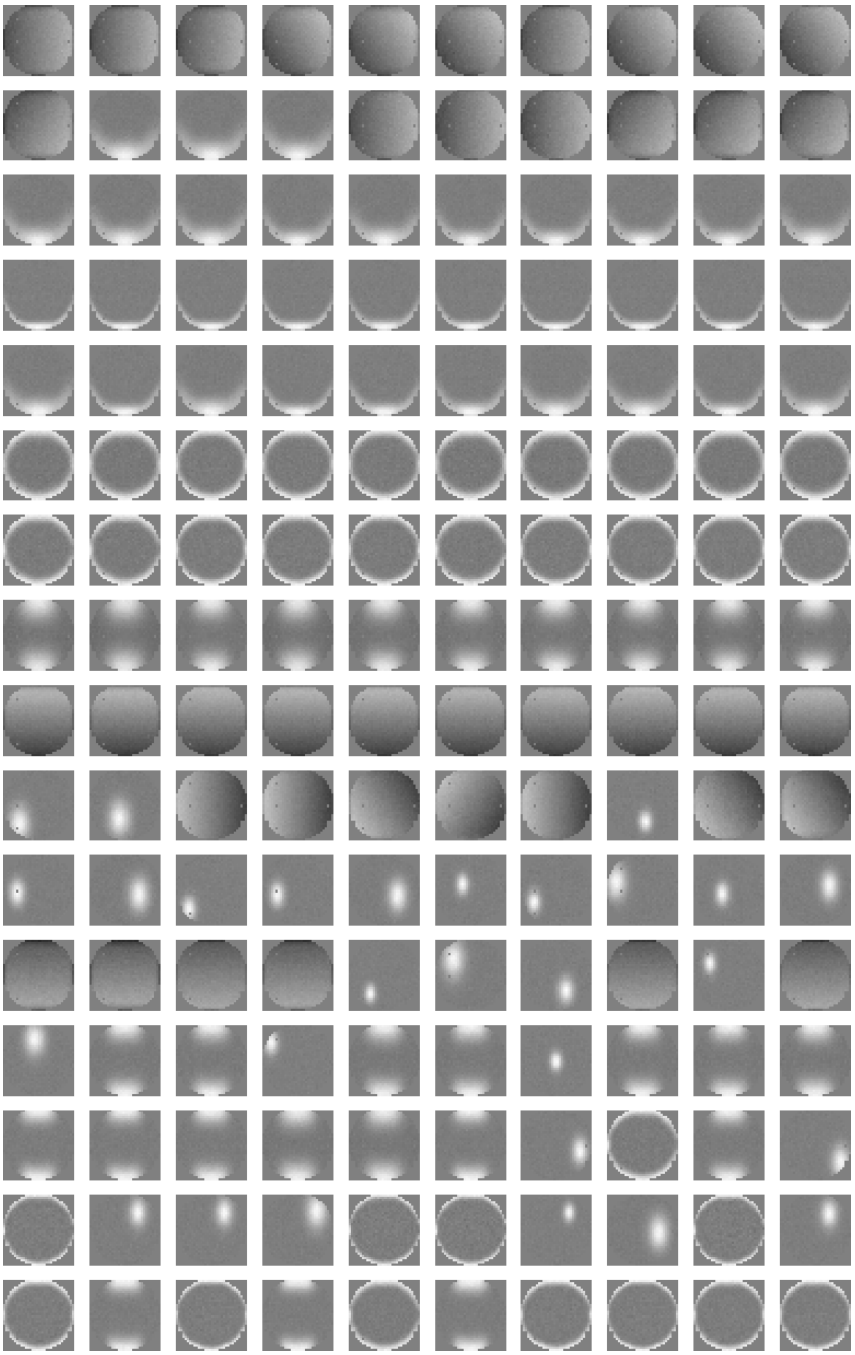


Figure 5.20: Wafer clusters using HAC clustering algorithm using 2D uniform noise mode. Each row represents a sample of max 10 wafer map images of a cluster.

Figure 5.21 shows the output latent space of the 2D encoder results with the corresponding clustering labels indicated by color.

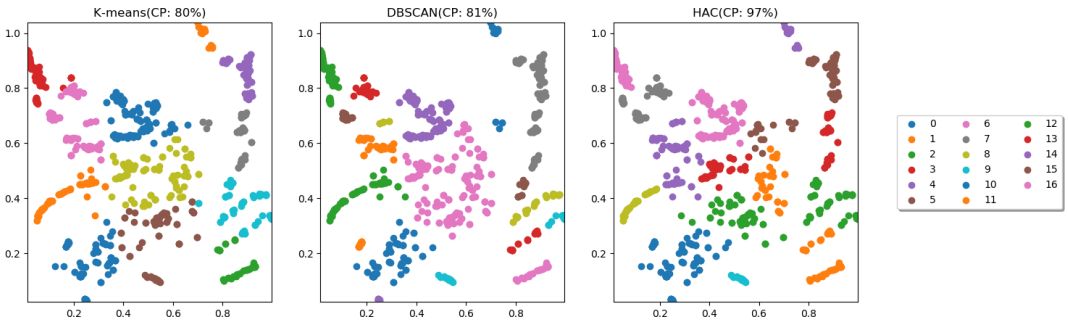


Figure 5.21: Latent space at epoch 1000 using the 2D uniform noise mode.

One-hot noise mode

For the one-hot noise mode, it can be shown that the resulting latent space is hard to cluster for DBSCAN. K-means and HAC perform good as shown in Figure 5.22.

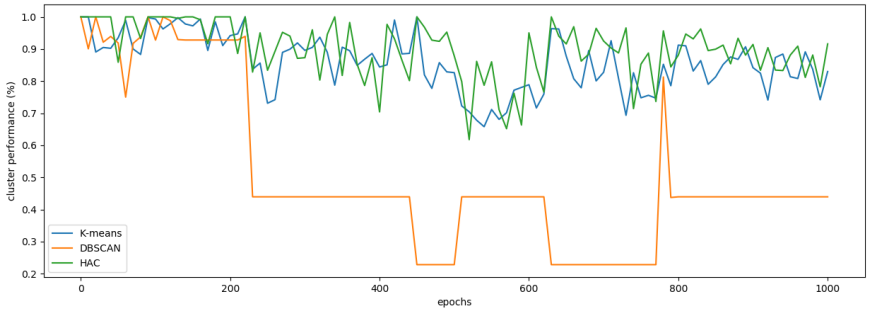


Figure 5.22: Cluster performance on three different clustering modes applied on the same latent outputs over 1000 epochs. Noise mode is 12D one-hot.

Figure 5.23 shows, that even without a clustering method, the intrinsic representation of all variants of the one-hot vectors can be used for extracting patterns.

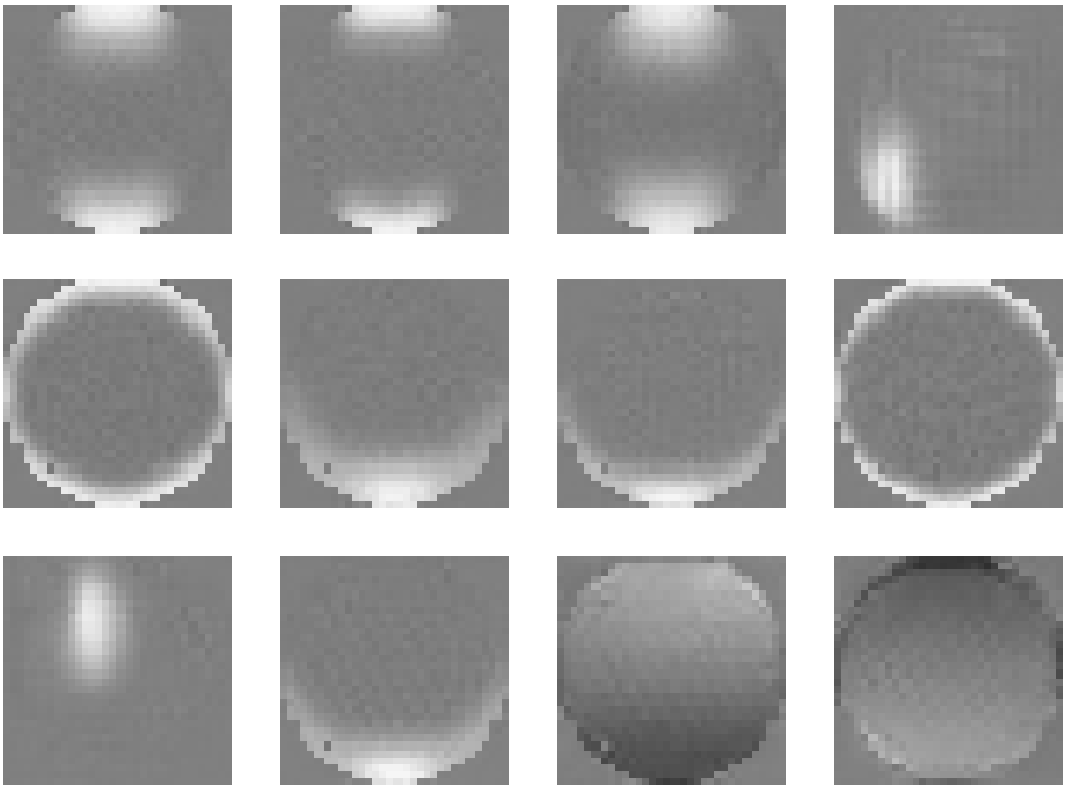


Figure 5.23: Intrinsic class representations of all possible one-hot vectors.

Compared to the uniform noise mode, DBSCAN produces only two clusters at epoch 1000 as shown in Figure 5.26. Both K-means and HAC in Figures 5.25 and 5.24 show good results from human perspective in clustering.

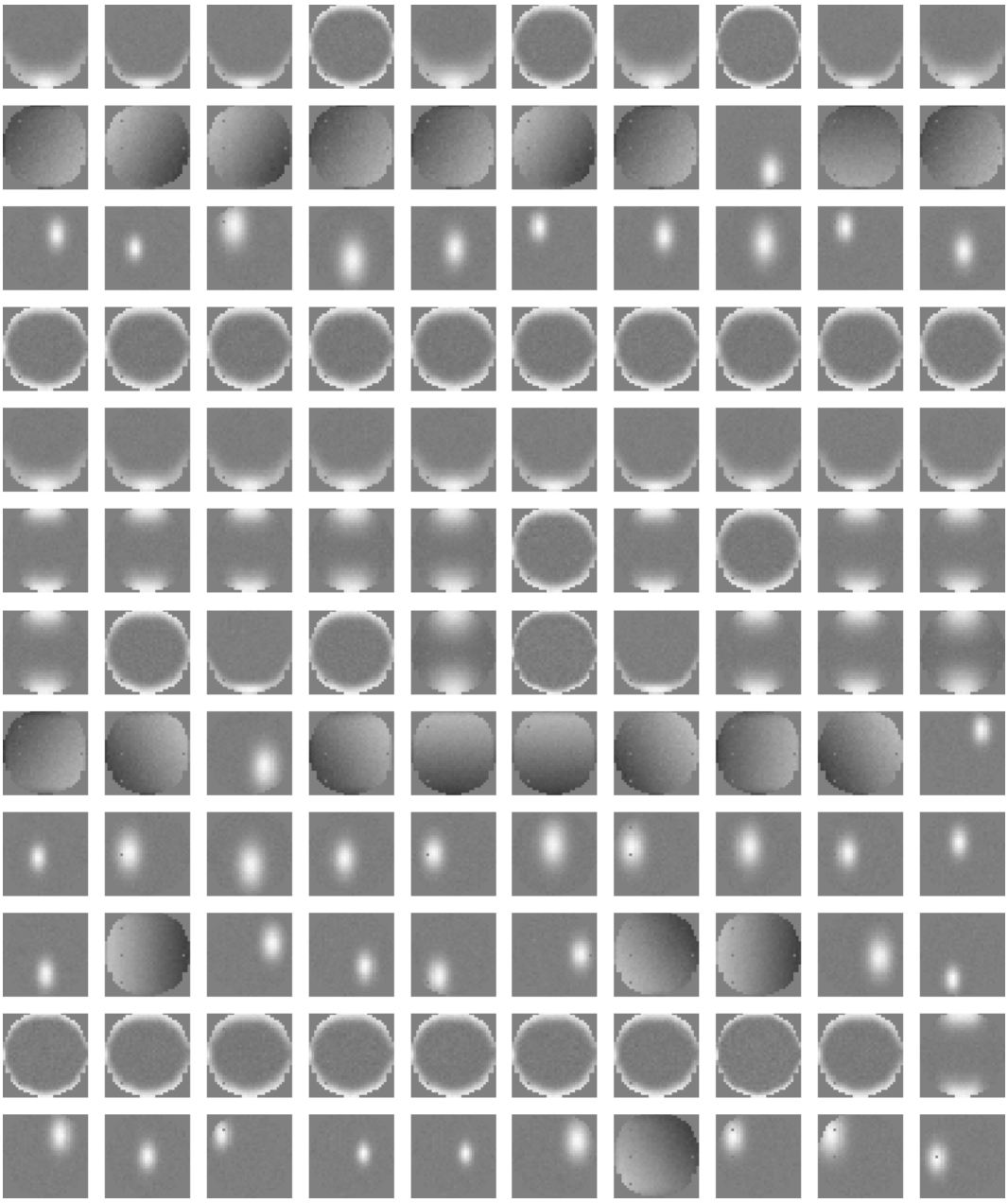


Figure 5.24: Wafer clusters using K-means clustering algorithm using 12D one-hot mode. Each row represents a sample of up to 10 wafer map images of a cluster.

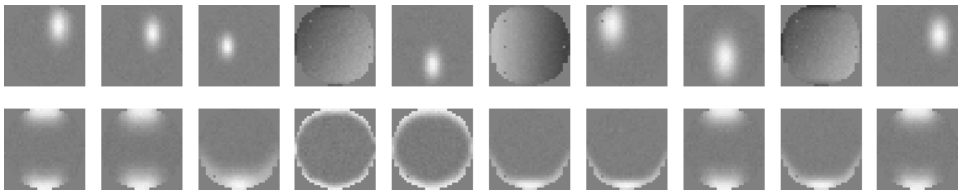


Figure 5.25: Wafer clusters using DBSCAN clustering algorithm using 12D one-hot mode. Each row represents a sample of up to 10 wafer map images of a cluster.

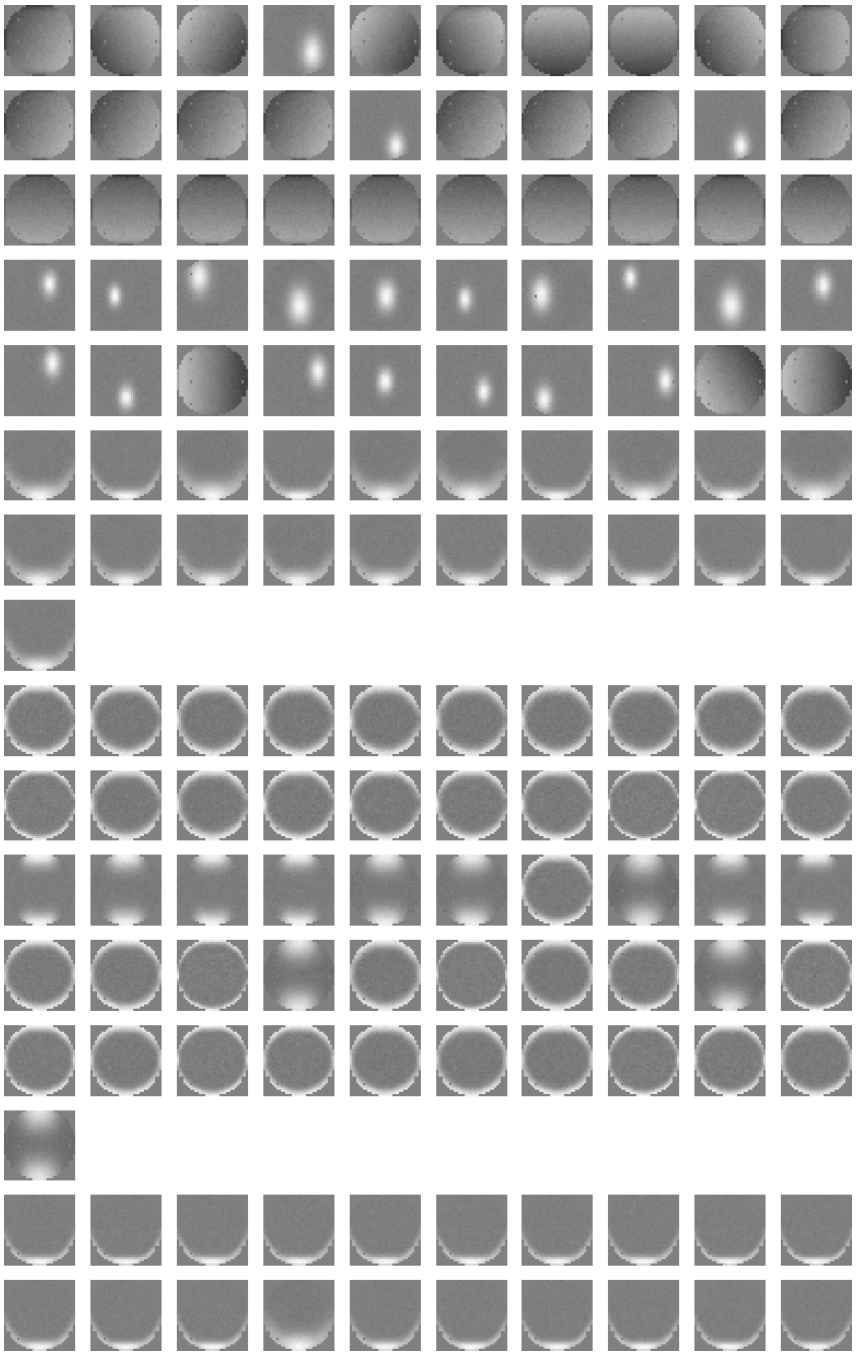


Figure 5.26: Wafer clusters using HAC clustering algorithm using 12D one-hot mode. Each row represents a sample of up to 10 wafer map images of a cluster.

5.3 Discussion

The experiments show, that the presented approach of using the encoder outputs of a BigBiGAN to extract patterns of a data set is possible. As for any type of NN and especially for generative ones, parametrization and data pre-processing is the crucial part of the work. The experiments presented in the previous section are the result of a long trial and error phase with different sizes of the latent space, different learning rates and optimizations within the given code framework. The result is reasonable, although it is highly dependent on the data set and the clustering algorithm used. Due to its simplicity and clearly defined patterns, data set B worked very good with all clustering methods provided. Although data set A consists of only binary data, the patterns provided are not as clearly visible as with data set B. This can also be seen at the latent output at Figure 5.21, which shows some more dense parts which appear to be the correct clusters, but also a variety of random spots which cannot be clustered accordingly. This is also the reason why all clustering methods worked worse at data set A. Experiments also show that if data is not properly pre-processed or if the learning process is done for too many epochs, also BigBiGAN tend to produce equal outputs for any kind of input. This phenomenon is called *mode loss* [26]. During experiments with data set A, also a different dimensionality of the latent space is shown. It can be shown, that increasing the complexity by increasing the dimensionality can improve the results depending on the data set and the clustering algorithm. The improvement is not significant and not steadily increasing.
Comparing the clustering algorithms, it can be seen that the algorithms where no fixed number of clusters are given tend to produce more clusters than in the actual reference data set. Although, this is mostly depending on the parameters of the algorithm, data also has an impact. As for example shown in Figure 5.19, in row number 9 there are some gradient shaped wafer maps with a vertical gradient direction. In the original data set labels, all gradient typed wafer maps are labelled

with the same label. As CNN based classifiers are not rotation invariant, wafers with a different gradient direction get clustered in a different cluster. This phenomenon, that different cluster results with the same original label happens throughout the experiments. As long as most clusters contain unique labels, the result is good. As a last step of real world pattern extraction, clusters need to be manually merged together for creating a new labelled data set.
In the VAE approach by Santos and Kern [37] also data set B was used. They found, that a traditional approach by using a PCA [48] followed by unsupervised clustering cannot distinguish between the ring pattern and the two-spot pattern. Similar to the results of the VAE approach, the BigBiGAN approach also fits well for all different kinds of patterns. So the capabilities of BigBiGAN is comparable to the VAE approach. The similarity between the results of the VAE and the BigBiGAN approach is obvious, as BigBiGANs discriminator is designed to make the network behave like an autoencoder in a global optimum state [27]. The main problem of BigBiGAN is, that the parametrization is really hard. If the training takes too many epochs, mode loss is a big problem and therefore also the separation quality is affected. If the training takes too few epochs, the encoder may not be able to separate the clusters accordingly. As BigBiGANs training is very dynamic, approaches like early stopping [49] are hard to implement. Within the experiments, all runs are made for 1000 epochs. Depending on the size of the latent/random vectors, the runtime with a NVIDIA GeForce RTX 2070 graphics card is between 2-3 hours for both data sets. The evaluation of single wafer maps are made within seconds.

5.4 Future work

As the experiments with data set B did not work as expected, some modifications can be used. The first part would be to evaluate only the training data, to make sure if the method works properly there. Also pre-processing could be improved

by filtering, as for example edge enhancement or better noise filtering. The second improvement could be made by creating a multi-level approach. The first part is the same as done in this experiments. The second part is, to use the clustered results as new reference data sets and run the same approach again on each of the new data sets. As the first part is good at high level clustering, the second part tries to cluster the details.
Also training BigBiGANs itself is a hard topic. As the learning curves are very dynamic, some non-dynamic metric is needed in order to use it with the early stopping approach. Early stopping stops training, if the metric does not change for a well-defined time range. One approach would be, to use the reconstruction loss as this non-dynamic metric.
The main usage for the proposed method is to create labelled data sets for further classification tasks. As the latent space itself provides a low level representation, this method could also be extended for direct classification at the latent space. The 2D latent space results could also be directly provided to the product engineers in order to make their own assumptions on the latent vectors position.

6 Conclusion

In this master thesis an approach has been proposed where the encoder outputs of the BigBiGAN network are used to cluster wafer map images into a set of labels. The experiments showed, that using BigBiGANs encoder to generate a low dimensional representative of some input data, clustering on the resulting latent space is an opportunity for extracting patterns for classification. Choosing the proper clustering method and its parametrizations is a crucial part for getting good results. Success is also very highly dependent on the data set used. The experiments also showed, that if patterns are clearly visible and easy to distinguish, the pattern extraction approach works better than with more complex data. This also showed, that pre-processing data in a proper way is the most important step for extracting patterns using this approach. It can be also shown, that using a one-hot driven random latent vector, classification results can be intrinsically achieved from the BigBiGAN network. In this experiments, the results were not fully satisfying but this can be part of a future work.

Bibliography

[1] Jin Zhou, Bulent Ayhan, Chiman Kwan, and Trac Tran. ATR performance improvement using images with corrupted or missing pixels. In Mohammad S. Alam, editor, *Pattern Recognition and Tracking XXIX*, volume 10649, pages 112 – 121. International Society for Optics and Photonics, SPIE, 2018.

[2] Christopher K. I. Williams, Charlie Nash, and Alfredo Nazábal. Autoencoders and probabilistic inference with missing data: An exact solution for the factor analysis case, 2018.

[3] Richard Szeliski. *Computer Vision: Algorithms and Applications - Histogram Equalization*, pages 106–110. Springer London, 2010.

[4] Alex C. Michalos, editor. *Encyclopedia of Quality of Life and Well-Being Research - Normal Distribution*, pages 4374–4374. Springer Netherlands, Dordrecht, 2014.

[5] Yadolah Dodge. *The Concise Encyclopedia of Statistics - Interquartile Range*, pages 266–267. Springer New York, New York, NY, 2008.

[6] Richard Szeliski. *Computer Vision: Algorithms and Applications - Interpolation*, pages 127–130. Springer London, 2010.

[7] Earl J. Kirkland. *Bilinear Interpolation*, pages 261–263. Springer US, Boston, MA, 2010.

[8] Olivier Rukundo and Hanqiang Cao. Nearest neighbor value interpolation. *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, 3(4):6, 2012.

[9] Richard Szeliski. *Computer Vision: Algorithms and Applications - Linear filtering*, pages 98–108. Springer London, 2010.

[10] Richard Szeliski. *Computer Vision: Algorithms and Applications - Feature detection and matching*, pages 181–234. Springer London, 2010.

[11] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 430–443, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[12] Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

[13] Alan Bundy and Lincoln Wallen. *Difference of Gaussians*, pages 30–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 1984.

[14] D.G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2, 1999.

[15] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, pages 778–792, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[16] Stan Z. Li and Anil Jain, editors. *L2 norm*, pages 883–883. Springer US, Boston, MA, 2009.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[18] Norbert M. Seel, editor. *Linear Regression*, pages 2052–2052. Springer US, Boston, MA, 2012.

[19] Marc D. Binder, Nobutaka Hirokawa, and Uwe Windhorst, editors. *Gradient Descent*, pages 1765–1766. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[20] A. Kızrak. Comparison of activation functions for deep neural networks. https://tinyurl.com/ftekar63, 2019.

[21] Philippe de Wilde. *Backpropagation*, pages 35–51. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[22] Alexander Semenov, Vladimir Boginski, and Eduardo L. Pasiliao. Neural networks with multidimensional cross-entropy loss functions. In Andrea Tagarelli and Hanghang Tong, editors, *Computational Data and Social Networks*, pages 57–62, Cham, 2019. Springer International Publishing.

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[24] M. Mursaleen. *Toeplitz Matrices*, pages 1–11. Springer International Publishing, Cham, 2014.

[25] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[26] Charlie Frogner, Chiyuan Zhang, Hossein Mobahi, Mauricio Araya-Polo, and Tomaso Poggio. Learning with a wasserstein loss, 2015.

[27] Jeff Donahue and Karen Simonyan. Large scale adversarial representation learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 10541–10551. Curran Associates, Inc., 2019.

[28] Thomas M. Sutter, Imant Daunhawer, and Julia E. Vogt. Multimodal generative learning utilizing jensen-shannon-divergence, 2020.

[29] Wei Li, Melvin Gauci, and Roderich Gross. A coevolutionary approach to learn animal behavior through controlled interaction. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 223–230, New York, NY, USA, 2013. ACM.

[30] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[31] Wei Li, Melvin Gauci, and Roderich Groß. Turing learning: a metric-free approach to inferring behavior and its application to swarms. *Swarm Intelligence*, 10(3):211–243, Sep 2016.

[32] Dongkuan Xu and Yingjie Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2, 08 2015.

[33] Clustering algorithms. `https://developers.google.com/machine-learning/clustering/clustering-algorithms`, 2020.

[34] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley*

Symposium on Mathematical Statistics and Probability, volume 1, pages 281–297. University of California Press, 1967.

[35] R. Wang and N. Chen. Wafer map defect pattern recognition using rotation-invariant features. *IEEE Transactions on Semiconductor Manufacturing*, 32(4):596–604, 2019.

[36] S. Schrunner, O. Bluder, A. Zernig, A. Kaestner, and R. Kern. Markov random fields for pattern extraction in analog wafer test data. In *2017 Seventh International Conference on Image Processing Theory, Tools and Applications (IPTA)*, pages 1–6, Nov 2017.

[37] Tiago Teixeira dos Santos and Roman Kern. Understanding wafer patterns in semiconductor production with variational auto-encoders. In *Proc. 26th European Symp. on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, 2018.

[38] H. S. Shon, E. Batbaatar, W. S. Cho, and S. G. Choi. Unsupervised pre-training of imbalanced data for identification of wafer map defect patterns. *IEEE Access*, 9:52352–52363, 2021.

[39] Wm-811k wafer map. `https://www.kaggle.com/qingyi/wm811k-wafer-map`. Accessed: 2019-09-27.

[40] Martin Pleschberger, Michael Scheiber, and Stefan Schrunner. Simulated Analog Wafer Test Data for Pattern Recognition, January 2019.

[41] Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. The reversible residual network: Backpropagation without storing activations, 2017.

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[43] Nitish Srivastava, Geoffrey E. Hinton, A. Krizhevsky, Ilya Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15:1929–1958, 2014.

[44] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks, 2018.

[45] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation, 2015.

[46] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[47] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[48] Stan Z. Li and Anil Jain, editors. *Principal Component Analysis*, pages 1091–1091. Springer US, Boston, MA, 2009.

[49] Federico Girosi, Michael Jones, and Tomaso Poggio. Regularization theory and neural networks architectures. *Neural Computation*, 7:219–269, 1995.

A Appendix

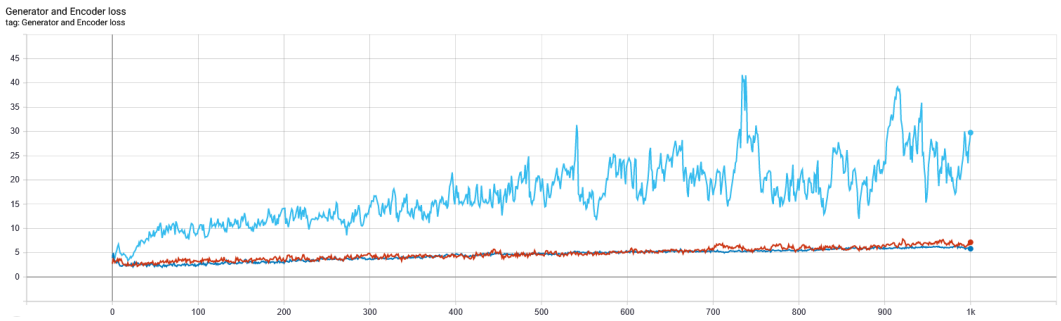


Figure A.1: Generator and encoder losses using data set A. (red - uniform 2D noise mode, blue - uniform 8D noise mode, light blue - 12D one-hot noise mode).

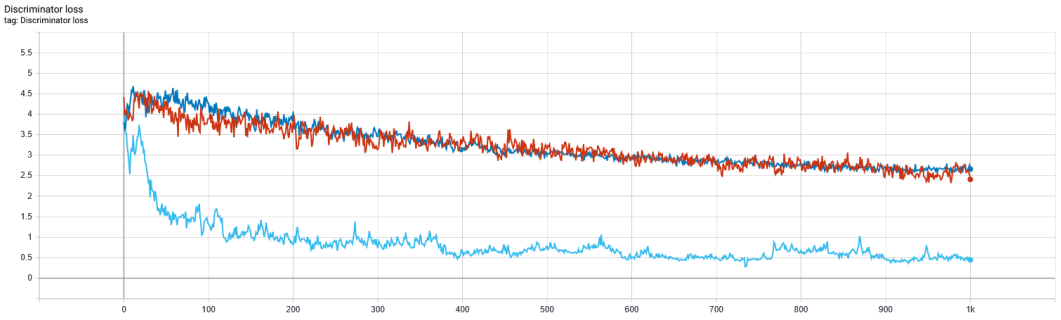


Figure A.2: Discriminator losses using data set A. (red - uniform 2D noise mode, blue - uniform 8D noise mode, light blue - 12D one-hot noise mode).

A Appendix

Figure A.3: Generator and encoder losses using data set B. (pink - uniform 2D noise mode, green - 12D one-hot noise mode).

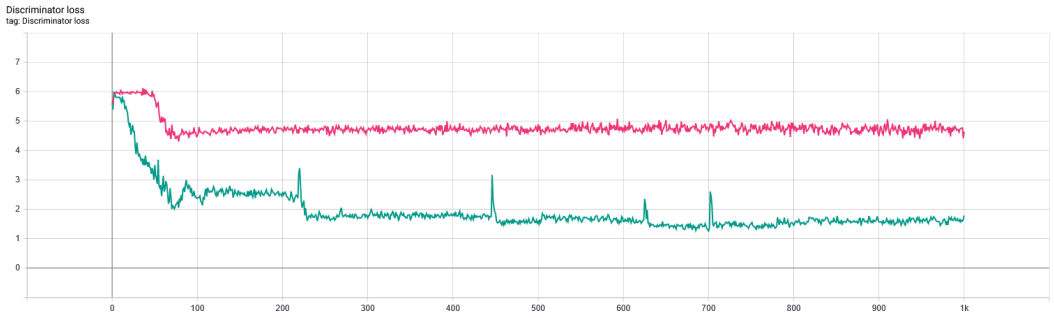


Figure A.4: Discriminator losses using data set B. (pink - uniform 2D noise mode, green - 12D one-hot noise mode).

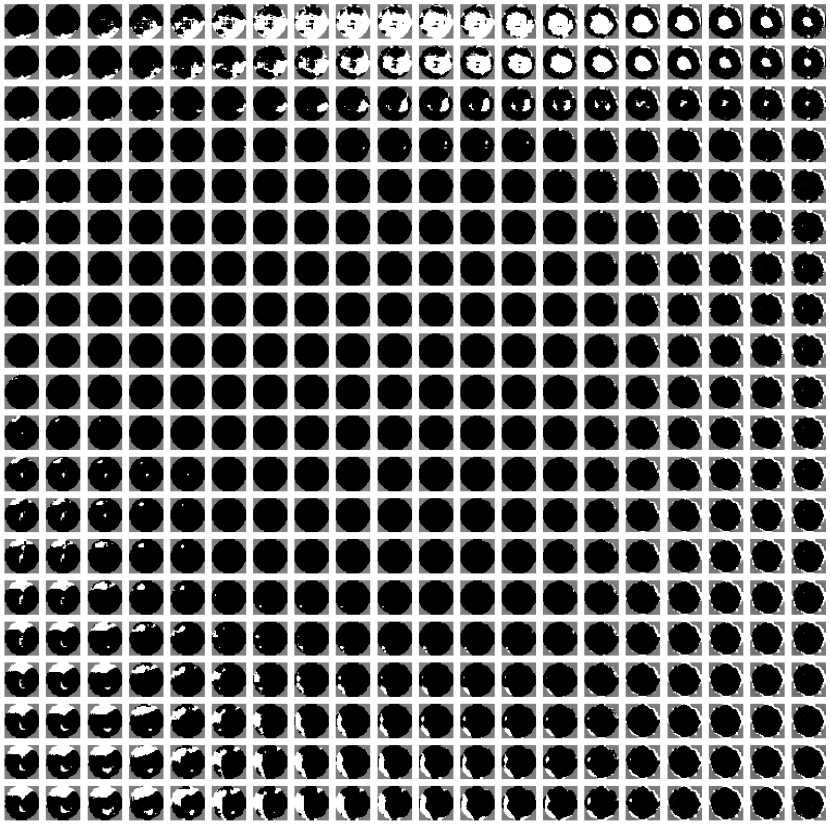


Figure A.5: Generated wafer maps of data set A using 2D uniform noise mode after 1000 epochs.

A Appendix

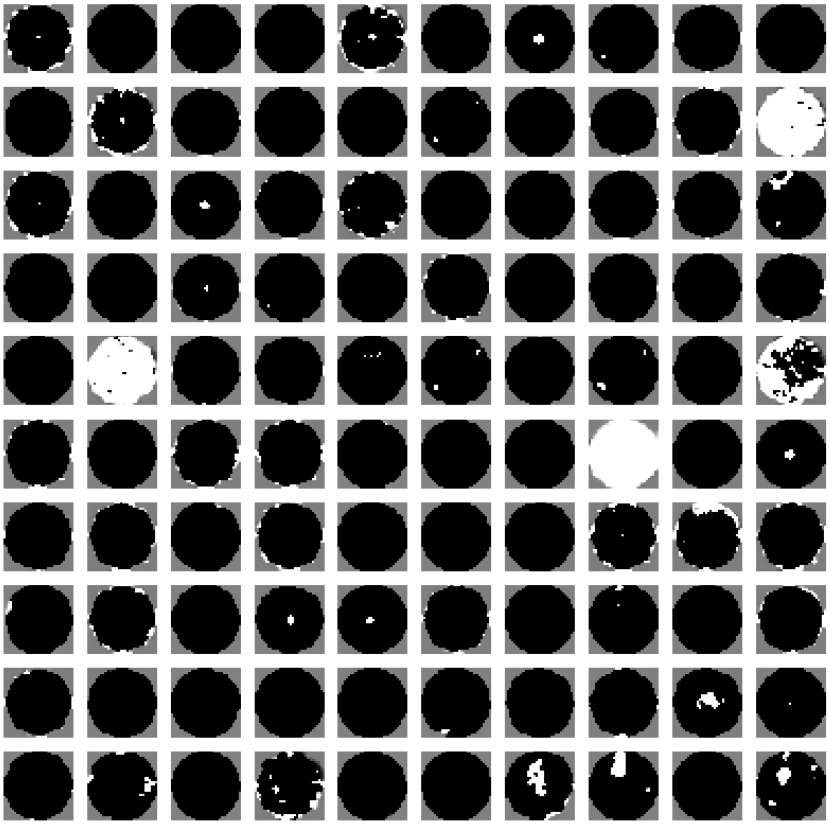


Figure A.6: Generated wafer maps of data set A using 8D uniform noise mode after 1000 epochs.

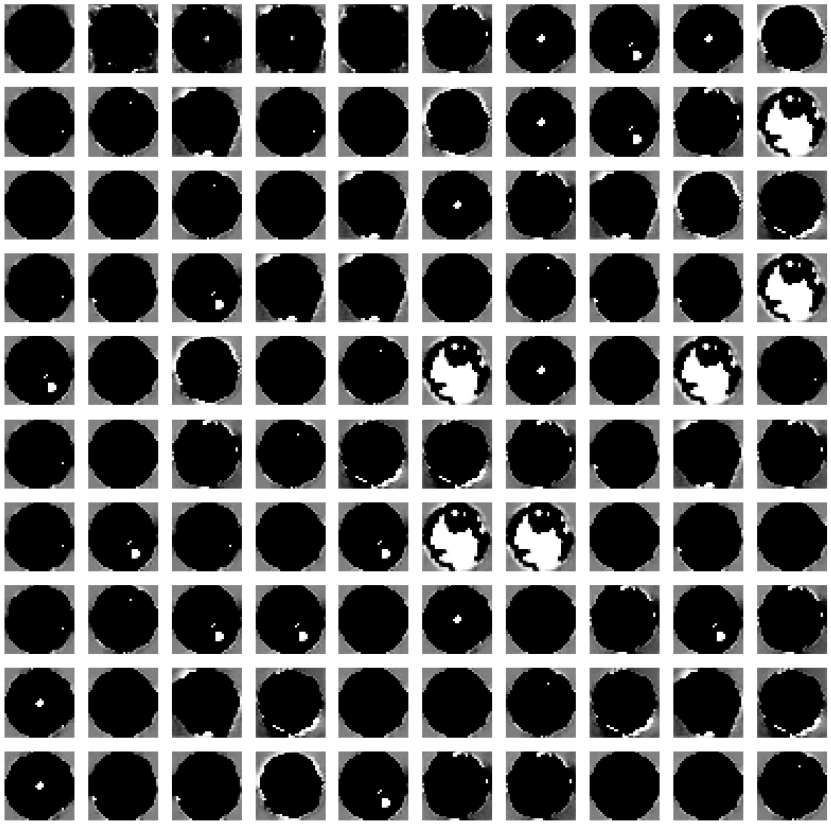


Figure A.7: Generated wafer maps of data set A using 12D one-hot noise mode after 1000 epochs.

A Appendix

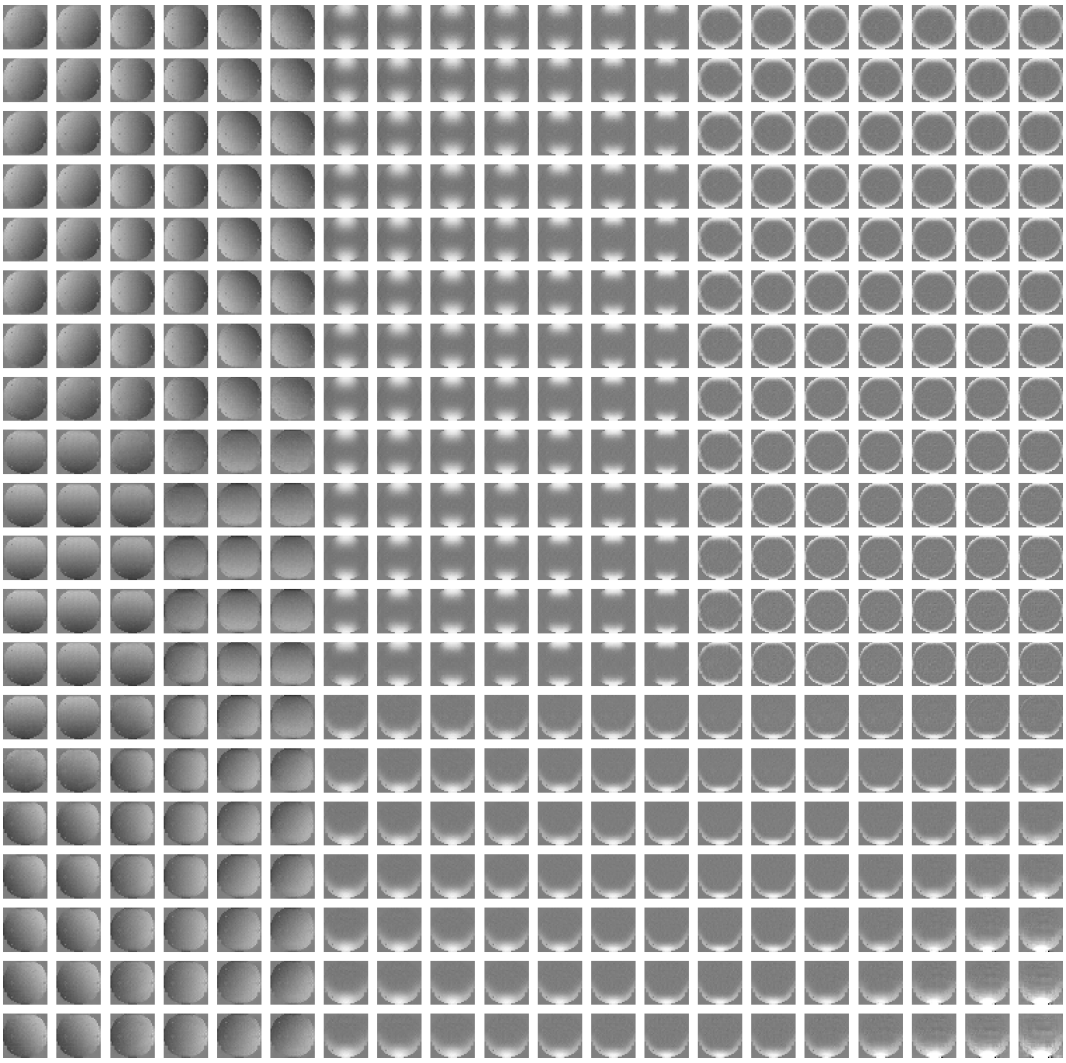


Figure A.8: Generated wafer maps of data set B using 2D uniform noise mode after 1000 epochs.

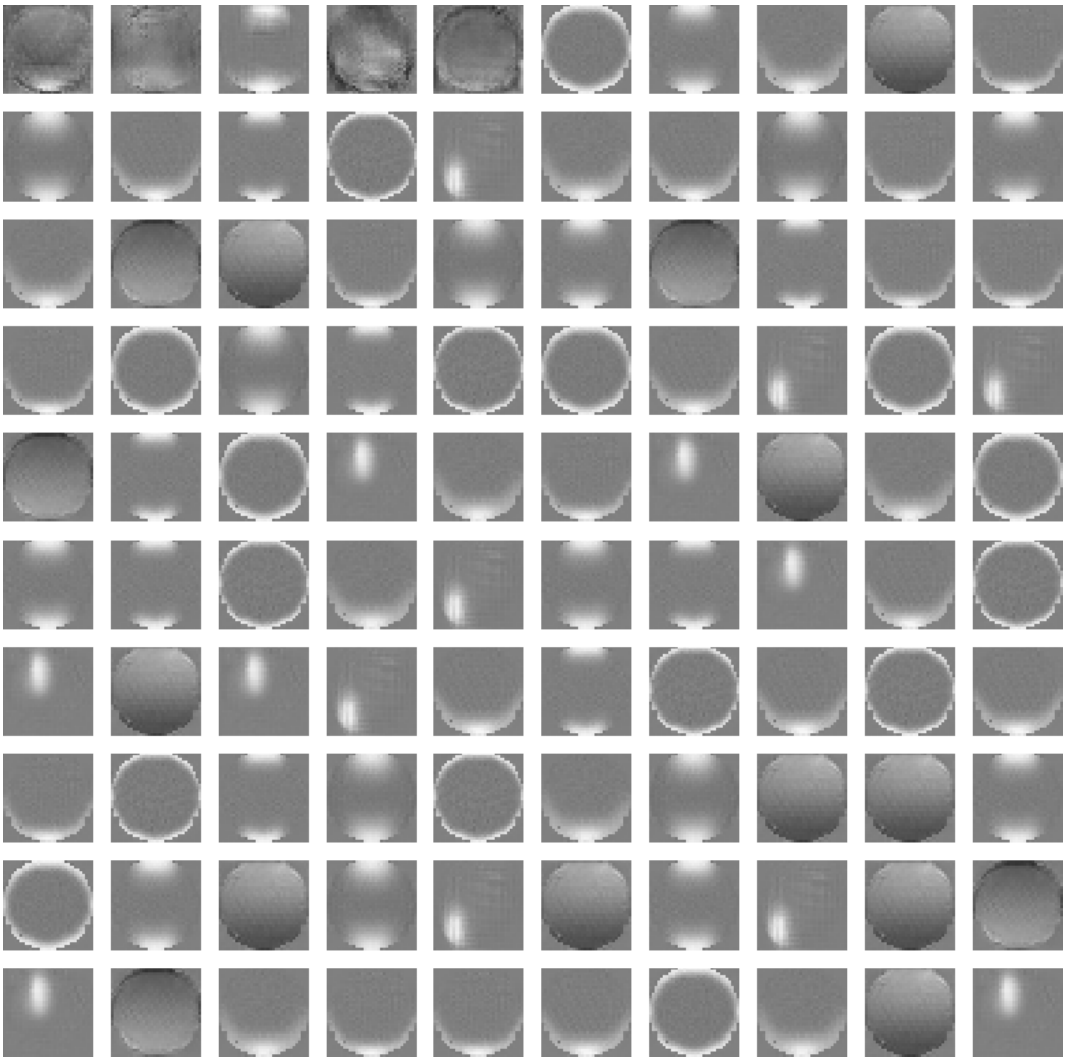


Figure A.9: Generated wafer maps of data set B using 12D one.hot noise mode after 1000 epochs.